

國立交通大學

資訊科學與工程研究所

碩士論文

針對低功率 drowsy BTB 設計之預先開啟機制

Next Entries Pre-activation for Low Power Drowsy BTB



研究生：潘漢倫

指導教授：單智君 教授

中華民國九十五年九月

針對低功率 drowsy BTB 設計之預先開啟機制
Next Entries Pre-activation for Low Power Drowsy BTB

研究生：潘漢倫

Student：Han-Lun Pan

指導教授：單智君

Advisor：Jean Jyn-Jiun Shann

國立交通大學

資訊科學與工程研究所



Submitted to Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science and Information Engineering

September 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年九月

國立交通大學

博碩士論文全文電子檔著作權授權書

(提供授權人裝訂於紙本論文書名頁之次頁用)

本授權書所授權之學位論文，為本人於國立交通大學 資訊工程 系所 系統

組，94 學年度第 二 學期取得碩士學位之論文。

論文題目：針對低功率 drowsy BTB 設計之預先開啟機制

指導教授：單智君

■ 同意

本人茲將本著作，以非專屬、無償授權國立交通大學與台灣聯合大學系統圖書館；基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立交通大學及台灣聯合大學系統圖書館得不限地域、時間與次數，以紙本、光碟或數位化等各種方法收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行線上檢索、閱覽、下載或列印。

論文全文上載網路公開之範圍及時間

本校及台灣聯合大學系統區域網路	<input checked="" type="checkbox"/> 立即公開
校外網際網路	<input checked="" type="checkbox"/> 立即公開

■ 全文電子檔送交國家圖書館

授權人：潘漢倫

親筆簽名：潘漢倫

中華民國 95 年 9 月 6 日

國立交通大學

博碩士論文紙本著作權授權書

(提供授權人裝訂於全文電子檔授權書之次頁用)

本授權書所授權之學位論文，為本人於國立交通大學 資訊工程 系所 系統 組，94 學年度第 二 學期取得碩士學位之論文。

論文題目：針對低功率 drowsy BTB 設計之預先開啟機制

指導教授：單智君

■ 同意

本人茲將本著作，以非專屬、無償授權國立交通大學，基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立交通大學圖書館得以紙本收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行閱覽或列印。

本論文為本人向經濟部智慧局申請專利(未申請者本條款請不予理會)的附件之一，申請文號為：_____，請將論文延至____年____月____日再公開。

授權人：潘漢倫

親筆簽名：潘漢倫

中華民國 95 年 9 月 6 日

國家圖書館

博碩士論文電子檔案上網授權書

ID:GT009317579

本授權書所授權之學位論文，為授權人於國立交通大學 資訊工程 系所 系統 組，94 學年度第 二 學期取得碩士學位之論文。

論文題目：針對低功率 drowsy BTB 設計之預先開啟機制

指導教授：單智君

茲同意將授權人擁有著作權之上列論文全文（含摘要），非專屬、無償授權國家圖書館，不限地域、時間與次數，以微縮、光碟或其他各種數位化方式將上列論文重製，並得將數位化之上列論文及論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

※ 讀者基於非營利性質之線上檢索、閱覽、下載或列印上列論文，應依著作權法相關規定辦理。

授權人：潘漢倫

親筆簽名：潘漢倫

中華民國 95 年 9 月 6 日

國立交通大學

研究所碩士班

論文口試委員會審定書

本校 資訊科學與工程 研究所 潘漢倫 君

所提論文：

針對低功率 drowsy BTB 設計之預先開啟機制

Next entries Pre-activation for Low-power Drowsy BTB

合於碩士資格水準、業經本委員會評審認可。

口試委員：

鍾崇斌

謝萬雲

單智君

指導教授：

單智君

所

長：

常文忠

中華民國九十五年七月二十一日

針對低功率 drowsy BTB 設計之預先開啟機制

學生：潘漢倫

指導教授：單智君 博士

國立交通大學資訊工程學系（研究所）碩士班



漏電流所造成之電耗已經成爲低功率處理器設計上的一項重要考量，特別是針對更先進的製程。BTB 是一種常見的動態分支預測器。除了快取記憶體之外，BTB 是 chip 上最大的元件。如果不會被用到的 BTB entry 能被調整至 leakage saving 的模式，漏電流所造成之電耗可以被節省。然而，由 leakage saving 模式調整至正常模式，需要一段的時間延遲。因此，這個研究設計出一套預先開啓的機制，能將下一個即將被存取的 BTB entry 預先調整至正常模式，以隱藏模式轉換造成之時間延遲，使得我們能藉由更積極的調整 BTB 至低電耗模式來節省更多在 BTB 上，因爲漏電流所造成之電耗。

Next Entries Pre-activation for Low Power Drowsy BTB

Student : Han-Lun Pan

Advisor : Jean Jyn-Jiun Shann

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung



Leakage energy consumption is becoming an important design consideration with the scaling of technology. Branch target buffer (BTB) is a popular kind of branch predictor. Besides cache, branch predictors are the largest on-chip array structures. If we put non-accessed branch target buffer entries into leakage saving mode, we gain branch target buffer leakage energy reduction. However, there is a period of time from leakage saving mode from normal mode and it introduces system leakage energy increment. This research proposes an accurate pre-active policy to switch the next accessed branch target buffer entry from leakage saving mode to normal mode in order to hide this latency. And we can put BTB entries into leakage saving mode more aggressively to reduce BTB leakage energy.

誌 謝

首先感謝我的指導老師 單智君教授，在他的諄諄教誨、辛勤指導與勉勵下，得以順利完成此篇論文。此外，我想感謝實驗室的另一位指導老師 鍾崇斌教授。鍾老師除了擔任我的口試委員外，在我的碩士生活中，也給我了許多寶貴的指導。同時感謝我的口試委員謝萬雲教授，在口試的過程中給予我的建議，使此篇論文更加完整。

感謝 Low Power 研究群的博士班學長 喬偉豪學長，以及其他實驗室的同學，熱心的與我討論，給我意見和鼓勵。

最後我要感謝我的家人長久以來毫無保留的給予我支持與鼓勵，讓我能你們的關心之中順利的完成學業

謝謝您們！

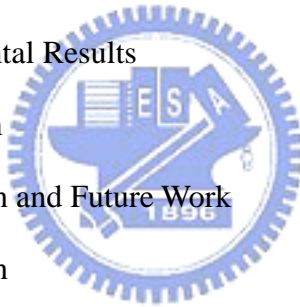


潘漢倫
2006年9月

Contents

摘要	i
Abstract	ii
誌謝	iii
Contents	iv
List of Figures	vi
List of Tables	viii
Chapter 1	Introduction1
1.1	Importance of Low Power Design1
1.2	Power Components of CMOS Circuit1
1.3	Importance of BTB Leakage Energy Reduction2
1.4	Motivation and Objective3
1.5	Thesis Organization4
Chapter 2	Background and Related Work5
2.1	Brief Introduction of Branch Target Buffer5
2.2	Circuit Techniques for Reducing SRAM Leakage power7
2.3	Related Works of BTB Active Power Reduction9
2.4	Related Works of BTB Leakage Power Reduction10
2.5	Summary of Related Work13
Chapter 3	Proposed Design15
3.1	Overview of Pre-activation Policy15
3.2	Two-direction Pre-active Policy16
3.2.1	Two-direction Based NBET16
3.2.2	Operations of Two-direction Based NBET18
3.3	One-direction Based NBET23

3.3.1	Why One-direction Pre-activation23
3.3.2	Design of One-direction Pre-activation24
3.3.3	Circuit Modification of One-direction Based NBET25
3.4	BTB Entry Deactivation27
3.5	Discussion28
Chapter 4	Evaluation32
4.1	Method32
4.2	Evaluation Metrics32
4.2.1	BTB Leakage Energy Consumption32
4.2.2	Performance Loss34
4.3	Environment34
4.4	Experimental Results36
4.5	Discussion40
Chapter 5:	Conclusion and Future Work42
5.1	Conclusion42
5.2	Future Work42
Reference	44



List of Figures

Figure 1-1: The ratio of dynamic power to static power source2
Figure 2-1 : Branch target buffer overview6
Figure 2-2: the subthreshold leakage power reduction of the SRAM cell with DVS	8
Figure 2-3: supply voltage of drowsy SRAM cell8
Figure 2-4: how to control the supply voltage of a SRAM cell9
Figure 2-5: state diagram for the leakage control circuit in [11]11
Figure 2-6: a code fragment with nested loops12
Figure 2-7: compilers inserts OFF and ON instructions for (a) the innermost loop and (b) the outermost loop12
Figure 3-1: architecture overview of our design15
Figure 3-2: NBET architecture of two-direction pre-active policy17
Figure 3-3: an example18
Figure 3-4: while updating branch instruction 1 into BTB, we use a register to record the necessary information18
Figure 3-5: while updating branch instruction into BTB, we can write the location information of branch instruction 2 into the NBET entry corresponding branch instruction 1.19
Figure 3-6: Next BTB location collection circuit20
Figure 3-7: how to record NBET location in LR20
Figure 3-8: how to find corresponding NBET entry and write21
Figure 3-9: Lookup-circuit of two direction based NBET21
Figure 3-10: Pre-activation circuit22
Figure 3-11: NBET architecture of one-direction pre-active policy24

Figure 3-12: NBET recording with predictor state25
Figure 3-13: Next BTB location collection circuit of on-direction based NBET25
Figure 3-14: write information into location register in one-direction pre-active policy26
Figure 3-15: writing information into NBET entry in one-direction pre-active policy27
Figure 3-16: Gating the deactivation signals for most recently accessed BTB and NBET entry28
Figure 3-17: code segment of a loop29
Figure 3-18: indirect jump destroys the branch instruction execution sequence30
Figure 4-1: BTB leakage energy components with two-direction pre-activation of Mibench37
Figure 4-2: BTB leakage energy components with one-direction pre-activation of Mibench37
Figure 4.3: comparison of two-direction and one-direction policy in Mibench38
Figure 4-4: BTB leakage energy components with two-direction pre-activation of SPEC200038
Figure 4-5: BTB leakage energy components with one-direction pre-activation of SPEC200039
Figure 4.6: comparison of two-direction and one-direction policy in SPEC200039



List of Tables

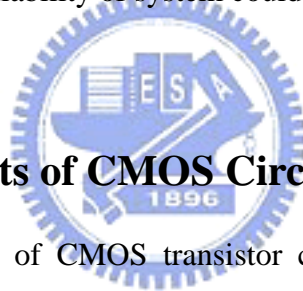
Table 4-1: Simulation parameters35
Table 4-2: leakage .energy parameters35
Table 4-3: best situation in my design40
Table 4-4: performance loss of best strategies41



Chapter 1 Introduction

1.1 Importance of Low Power Design

Low power design has become more and more important in microprocessor design. In embedded system, most portable devices are powered by battery, such as MP3 player, mobile phone, personal digital assistant (PDA), and etc... The life time of battery is an important consideration. Low power design helps to increase the life time of battery. On the other hands, for high-end machines, hardware complexity and high working voltage produces enormous heat, and high temperature results in some problems. Low power design reduces power consumption and the heat for high performance device. So the reliability of system could be promoted.



1.2 Power Components of CMOS Circuit

The power consumption of CMOS transistor consists of two parts: dynamic power and static power. The dynamic power is also called active power. There are two parts in dynamic power: switching power is dissipated by charging and discharging the gate output capacitance and short-circuit power is dissipated by the short-circuit current that V_{DD} and V_{SS} may be inter mittently shorted during logic gate operation. The static power is dissipated by leakage current that leaks through transistors, so is called leakage power. As processor techniques moves below 0.1um, leakage power consumptions begin dominate the total power consumption of circuit [1]. Figure 1-1 shows the ratio of active power and leakage power in different techniques. How to reduce leakage becomes an important research issue.

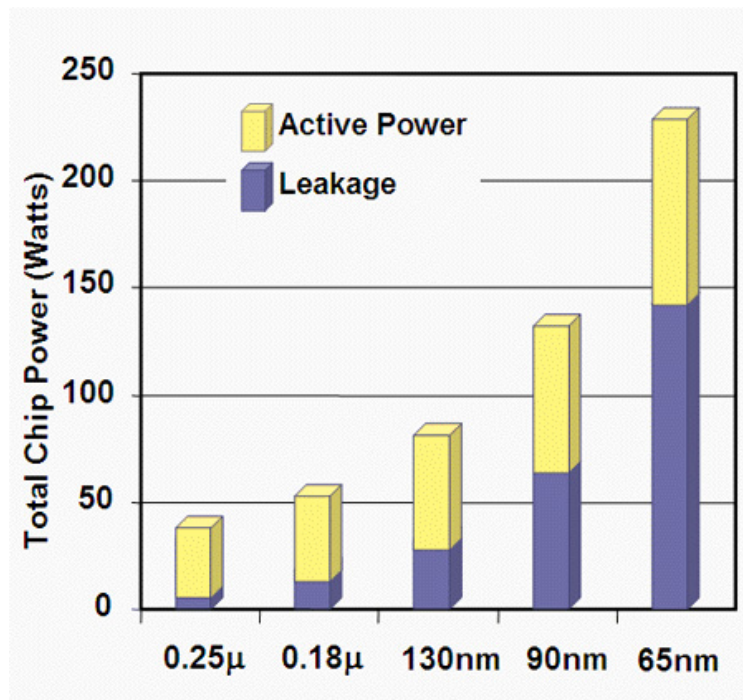


Figure 1-1: The ratio of dynamic power to static power source: EE-Times, 08/06/2004

1.3 Importance of BTB Leakage Energy Reduction

Almost all processors are deeply pipelined today. In order to reduce pipeline stall cycles due to branch instructions, most processors adopt dynamic branch prediction techniques. BTB, which is a large on-chip memory with the tag and branch target address, is used to support dynamic branch prediction. It dissipates 10 % of the processor's total dynamic power [2]. Besides caches, BTB is the largest on-chip memory among a typical processor. Current typical branch target buffer has 512-2048 entries (2-8K bytes) and equals the size of a small cache. In high performance microprocessors, in order to improve prediction accuracy, BTB has become larger and larger. For example, Pentium 4 has 4K entries BTB. Moreover, BTB is a thermal hotspot [3]. The leakage energy of BTB may be more than its size would suggest since the leakage energy increases exponentially with temperatures increment.

Drowsy cache [4] is a popular circuit technique for SRAM leakage power reduction. It provides two power modes for each row of SRAM arrays. While the normal mode is fully functional, the drowsy mode is data preserving only. A drowsy row needs to be waked up with about one cycle latency if an access requirement is comes for this row. Since only a small part of BTB entries would be accessed in a period of time, if the other unused entries can be managed into drowsy mode, the BTB leakage energy would be reduced

1.4 Motivation and Objective

Branch target buffet is a cache which stores information about branch instruction. It can gain performance benefit when branch target buffer look-up hit. Since branch instructions constitute parts of total executed instructions, on the other side, only a small part of branch target buffer entries will be accessed in a period of execution time, if those entries not be accessed recently can be put into drowsy mode. Leakage power consumption could be reduced.

In this research, we proposed a next entry pre-activation method cooperated with the decay policy [5] to manage power modes of each BTB entries. For the decay policy, a BTB entry which has not been accessed for a period of time is putted into drowsy mode. For next entry pre-activation, the possible BTB next entries are cached for each branch instruction existing in BTB and used to pre-activate the BTB entries that will be used in the near future. Unlike the decay management, our method helps to shorten the decay interval and hide wake-up latency, and both of these benefit leakage reduction.

For the branch instruction execution, there are at most two possible directions: taken and non-taken. Moreover, most branch instructions have fixed target addresses. The behavior of accessing branch target buffer can be tracked during execution time. For this property, our pre-active policy is to pre-activate the next possible accessed branch target buffer in both path of the path indicated by branch predictor.

1.5 Thesis Organization

The rest of this paper is organized as follows. Section two presents the related work of active and leakage reduction techniques. Section three describes our power mode management policies for BTB. Section four gives the simulation results, and the last section is our conclusion and possible future work.



Chapter 2 Background and Related Work

2.1 Brief Introduction of Branch Target Buffer(BTB)

Almost all processors are highly pipelined today. For a processor with deep pipeline, how to reduce control hazard becomes a major topic to improve system performance. A good dynamic branch prediction is a proper solution.

Branch target buffer (BTB) is a very commonly used dynamic branch predictor. The major function of BTB is to predict the direction of a branch direction and returns the target address of the branch. BTB performs branch prediction at the first pipeline stage. If the branch instruction direction is correctly predicted, the branch penalty could be reduced to zero. In other words, if all branch instructions can be correctly predicted, the pipeline will be kept full.

BTB is a cache which stores the information about branch instructions. It consists of three parts: tag array, data array, and branch direction predictor. The tag array saves source PC of branch instructions, the data array saves the target address of branch instructions, and branch direction predictor is used to predict whether the branch instruction will be taken or non-taken. While a branch instruction is executed and taken, it will be placed into BTB. While execution of the branch instruction next, we can get the direction information from BTB until this instruction being placed.

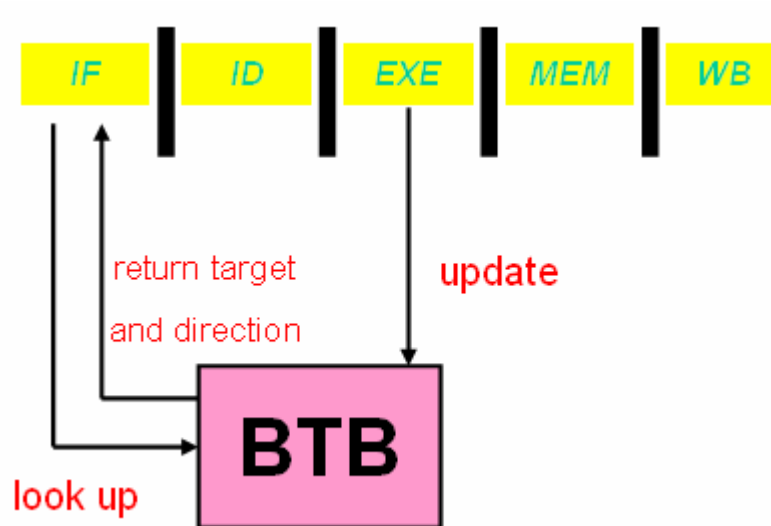


Figure 2-1 : Branch target buffer overview

Figure 2-1 is the system overview of branch target buffer in a 5-stage pipeline processor. When instruction fetcher fetches an instruction in the first pipeline stage, it sends the address of the instruction (current value of program counter) to look up branch target buffer. If look-up is hit and predicted taken, it means that current instruction is a branch instruction and it may be taken. So BTB returns the target address of this branch instruction. The instruction fetcher fetches the instruction from the address returned by BTB next cycle. If look-up is not hit or predicted non-taken, it means that current instruction is not a branch instruction or current branch instruction may be non-taken. So the instruction fetcher fetches from the address equals to current plus the size of an instruction. If the address is correctly predicted, no branch penalty is introduced for a branch instruction.

All information about branch instructions is gathered during run time. A branch instruction could be placed into branch target buffer only after it has been executed and taken. And may be swept out when a conflict happened. We assume that a branch instruction finishes execution in EXE stage of pipeline. If this branch is not in the

branch target buffer and it is taken, then its target address and predicted direction would be placed into BTB. If it is in the branch target buffer already, it only modifies the state of corresponding branch predictor. The branch predictor can predict the possible direction of the branch instruction execution next time.

2.2 Circuit Techniques for Reducing SRAM Leakage Power

The circuit techniques to reduce SRAM leakage power are classified into state-destroying and state-preserving techniques. State-destroying techniques such as gated-Vdd [6] that gates the supply voltage of the SRAM cells in its leakage saving mode. Therefore, the data stored in SRAM will be loss in this mode and the data need to refetch if we want to use again.

Unlike the state-destroying methods, state-preserving techniques preserve the data stored in SRAM cells in its leakage saving mode. One popular method of state-preserving techniques is scaling the supply voltage of SRAM cells dynamically [7]. Since higher supply voltage introduces higher leakage power consumption (see figure 2-2). If we adjust the supply voltage, the leakage power consumption would be reduced. It provides two power modes with different supply voltages for each row of SRAM arrays. In active mode, the data stored in SRAM cells can be accessed arbitrarily. In drowsy mode, the data in SRAM cells is preserved only. If an access requirement is comes for the drowsy row of the SRAM, it needs to be waked up with about one cycle latency before accessed.

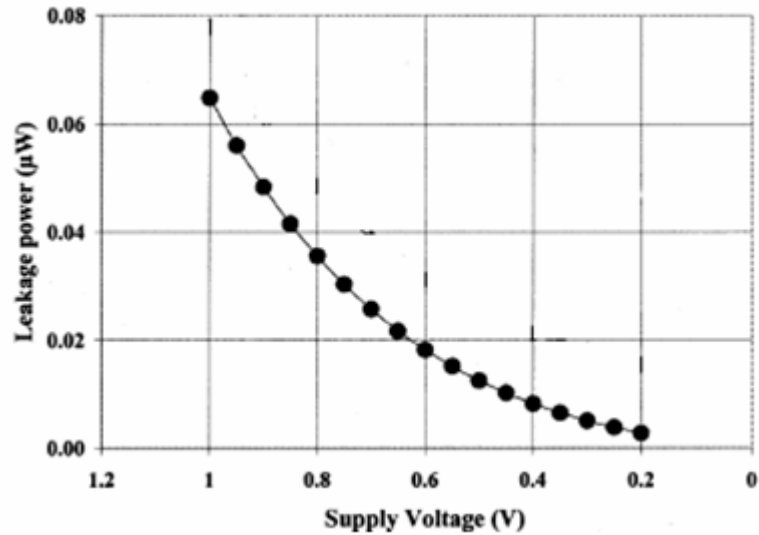


Figure 2-2: the subthreshold leakage power reduction of the SRAM cell with DVS

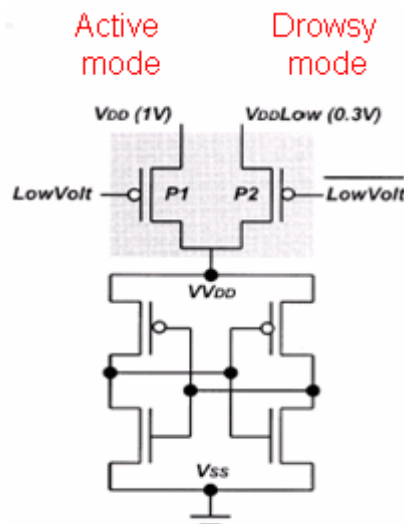


Figure 2-3: supply voltage of drowsy SRAM cell

Figure 2-3 is a drowsy SRAM cell with different supply voltage. The two pMOS control transistors, P1 and P2, control the supply voltage of the SRAM cell. When the SRAM cell is in the active mode, P1 supplies a standard supply voltage (1V), the value of the SRAM cell can be accessed normally. When the SRAM cell is in the drowsy mode, P2 supplies a lower voltage (300mV), however, the SRAM call access is not allowed. P1 and P2 are controlled by complementary supply voltage control

signals.

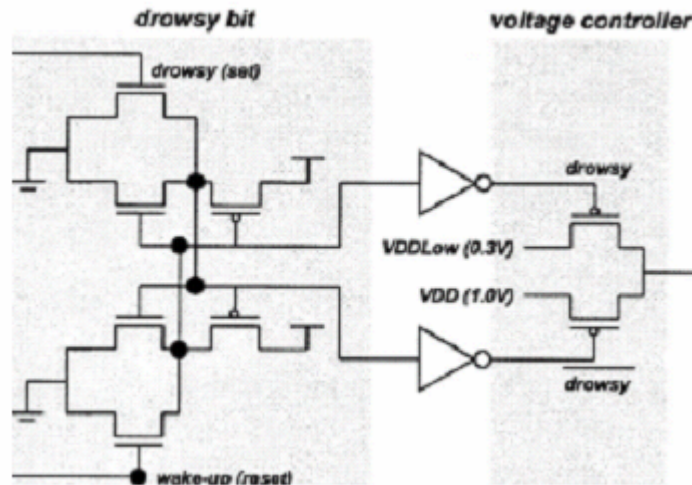


Figure 2-4: how to control the supply voltage of a SRAM cell

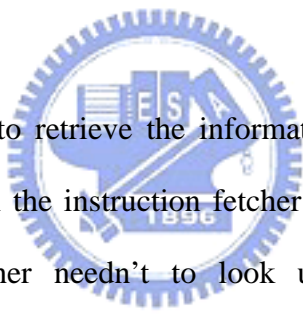
Figure 2-4 shows that how to control the supply voltage of a SRAM cell. The supply voltage of a SRAM cell depends on the state of the drowsy bit. If the drowsy bit is set, the voltage controller supplies low voltage and the SRAM cell is in drowsy mode. If the drowsy bit is cleared, the voltage controller supplies standard voltage and the SRAM cell is in active mode. When the SRAM cells are in drowsy mode, it takes a finite amount of time (wake-up latency) to restore the voltage level from drowsy mode to active mode, about one or two cycle. If we want to hide the wake-up latency, pre-active policy is a proper method.

2.3 Related Works of BTB Active Power Reduction

For branch target buffer dynamic energy reduction, since looking up branch target buffer is only valid when current instruction is a branch instruction and branch instruction is a small part of total executed instructions, how to skip useless branch

target buffer look-up will help to reduce dynamic energy of branch target buffer.

In [8], instructions are pre-decoded to filter out the non-branch instructions. When instructions are loaded to instruction cache from memory, they are decoded to identify if there exists any branch instruction in a certain cache line. The information is stored in a hardware called Prediction Probe Detector (PPD). When an instruction fetches an instruction from instruction cache, it looks up PPD. If PPD returns that current cache line contains branch instructions, the instruction fetcher looks up branch target buffer for all instructions in the cache line. If PPD returns that the cache line contains no branch instructions, the fetcher need not to look up branch target buffer for all instructions in current cache line.



In [9], it uses compiler to retrieve the information of branch instructions and inserts hint instructions to tell the instruction fetcher when to look up branch target buffer. The instruction fetcher needn't to look up branch target buffer until encountering a hint instruction. When the hint instruction is executed, it tells the instruction fetcher the distance of next branch instruction.

2.4 Related Works of BTB Leakage Power Reduction

For branch target buffer leakage energy reduction, since branch target buffer is a cache-like data structure, the method for cache leakage energy reduction can be adapted for branch target buffer leakage energy reduction like decay strategy.

In [10], it applies the decay strategy to branch target buffer for leakage energy savings. At regular interval, if a BTB entry has not been accessed, they would be

placed into low power mode. The interval, called the decay interval, is measured in processor cycles and is a critical parameter for this scheme. This paper adapts state-destroy mode for leakage energy saving. In contrast to the decay strategy for caches, they find that the decay interval for branch predictor should be larger to avoid branch misprediction and the best decay interval is 64K cycles.

In [11], it uses a state-preserving leakage control mechanism and a compiler-directed approach based on loop. It uses a finite state machine with three states to implement the leakage control circuit (see figure 2-5).

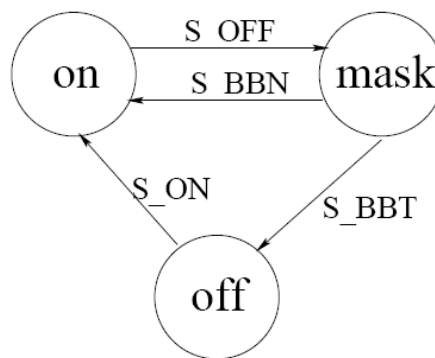


Figure 2-5: state diagram for the leakage control circuit in [11]

The S_OFF and S_ON signals are triggered by OFF and ON instruction inserted by the compiler. For both the innermost loop and the outermost loop strategies, The compiler inserts OFF instruction before the loop body and inserts ON instruction after the loop body (see figure 2-6, figure 2-7).

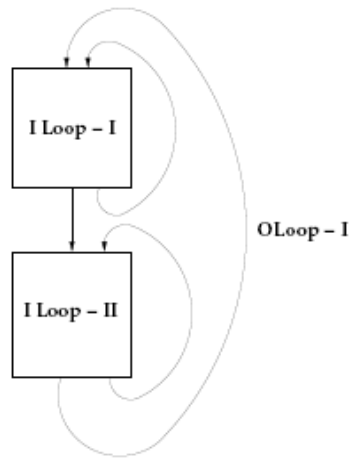


Figure 2-6: a code fragment with nested loops

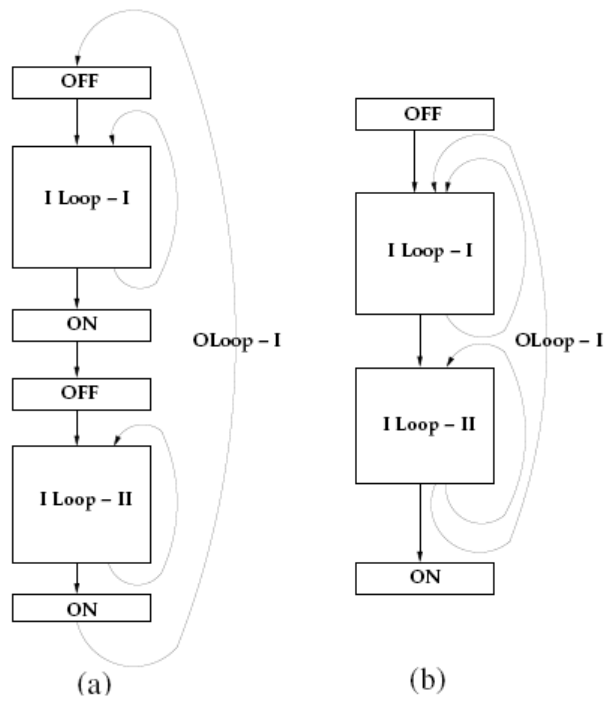


Figure 2-7: compilers inserts OFF and ON instructions for (a) the innermost loop and (b) the outermost loop

The S_BBT and S_BBN signals are generated by the first execution of the backward branch instruction in the end of the innermost or the outermost loop and the backward branch instruction can be identified by the compiler. The S_BBT signal is

triggered when the backward branch instruction is taken and the S_BBN signal is triggered when the backward branch instruction is non-taken.

Initially, the leakage control circuit is in ON state. When encountering an OFF instruction, the state moves to MASK state and the process will enter a loop. During first iteration of the loop, the accessed branch predictor entries would be marked. To implement this strategy, the paper associative a mask bit to each branch target buffer entry. If the entry is accessed during the first iteration, the bit is set. If the S_BBT signal is generated, the entries whose mask bit not set will be placed into drowsy mode. When leaving the loop, the ON instruction will be executed and all branch predictor entries will be put into active mode. If the S_BBN signal is generated, it means that this loop has only been executed for one iteration and the state will move to ON mode.



2.5 Summary of Related Work

In decay approach, so large decay is used to be sure that the decay entry is invalid, however, so large interval waste the opportunities to reduce BTB leakage energy. The compiler approach only handles the branch instructions in loop, not whole program. It may not be suitable to the programs with fewer loops. Second, it needs to insert extra instructions to support hardware operation so the code size will increase. The ISA needs to be modified and the system will becomes complex to implement. Finally, the accessed BTB entries will be keep in normal mode during all iteration of a loop. However, in a large loop, it is worth to switch the mode of accessed BTB entries.

If we adapt drowsy mode as leakage saving mode co-working with a pre-activation

policy, since the data is still preserved, we may gain more BTB leakage energy reduction by decreasing the value of decay interval and the performance will not degrade.



Chapter 3 Proposed Design

3.1 Overview of Pre-activation Policy

Generally speaking, for each branch instruction, there are at most two possible directions, taken or non-taken. Moreover, most branch instructions have fixed target address. Therefore, for most branch instructions, the possible next branch instructions on execution path are fixed.

For a drowsy-based BTB, if the next BTB index is revealed to processor early, the pre-activation operation becomes trivial. Therefore, our research focuses on how to record the next possible BTB entries.

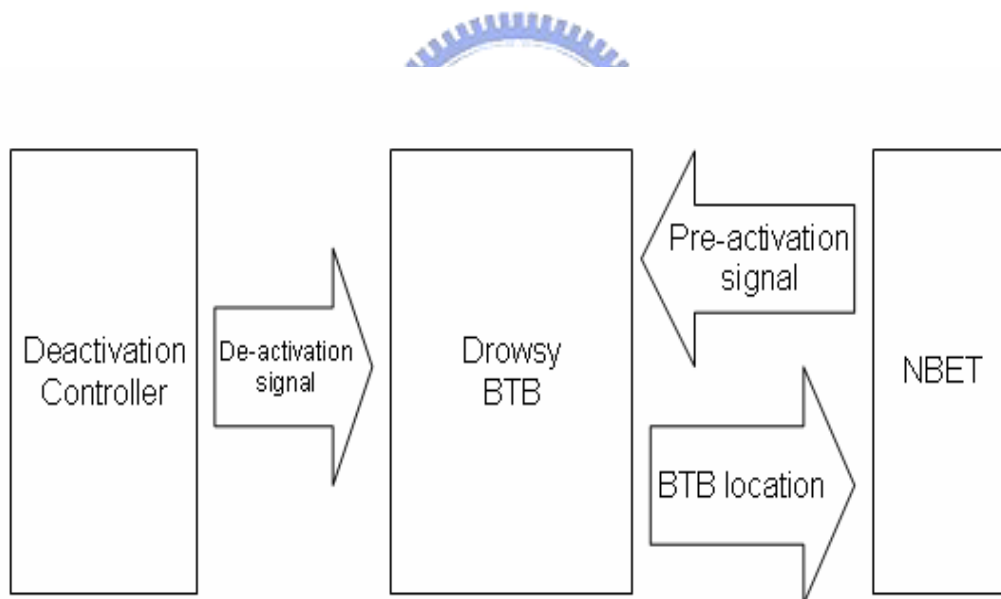


Figure 3-1: architecture overview of our design

Figure 3-1 shows the architecture overview of our design. The deactivation controller puts a BTB entry into drowsy mode if this entry has not been accessed for a period of time. Next BTB Entry Table (NBET) collects BTB location for next BTB entry pre-activation.

While executing the code sequence along the possible path, we will encounter the next branch instruction. If we can record where the next branch instruction is located in branch target buffer, we can use this information to pre-activate the next possibly accessed branch target buffer while executing current branch instruction next time.

3.2 Two-direction Pre-activation Policy

For pre-activating the next possible accessed entry, we need to record two things: 1) the executed code sequence of branch instructions and 2) where a branch instruction is located in branch target buffer. The information can be gathered during execution time.



Since there are at most two directions for each branch instruction, we must record the next possible accessed branch target buffer for a branch instruction along taken and non-taken path. In order to record the two possible next BTB locations (which set and which way in BTB) of all branch instructions existing in BTB, we configure NBET as two-direction based NBET which has the same number of entries as BTB. Each entry has two fields: one for next BTB entry of taken path and another one for not-taken path.

3.2.1 Two-direction based NBET

We need to record the necessary information of every branch instruction. Because a branch instruction may be placed into to one branch target buffer, so a branch target buffer entry has a corresponding NBET entry to record the next possible

accessed branch target buffer entries and the corresponding entry can be placed into drowsy mode with the branch target buffer to reduced the leakage energy from NBET.

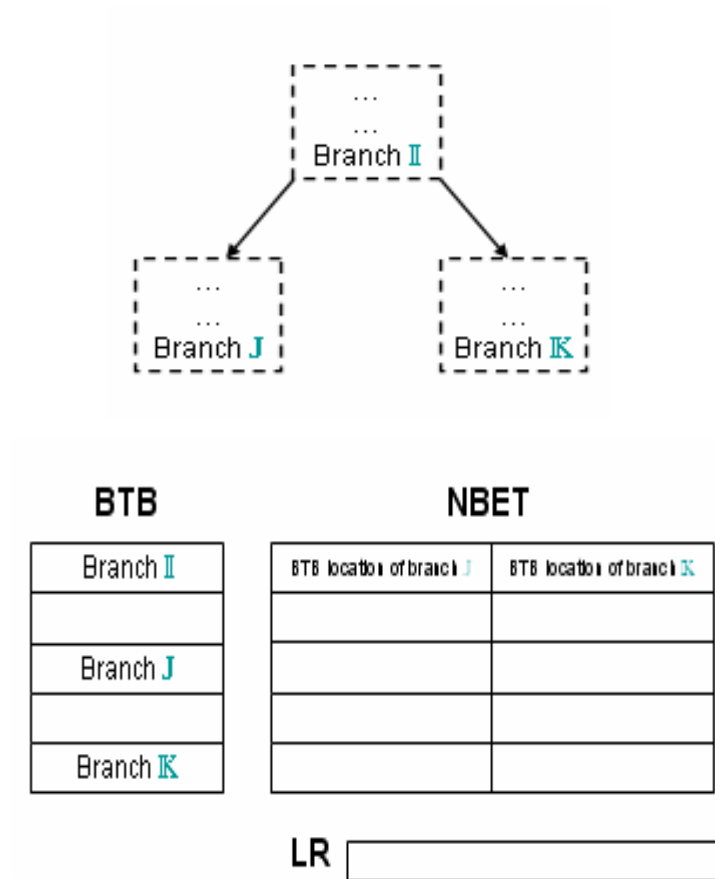


Figure 3-2: NBET architecture of two-direction pre-active policy

Figure 3-2 is the NBET architecture overview of two-direction pre-active policy. One NBET entry has two fields: one for taken path and the other for non-taken path. While executing branch instruction I, we will encounter branch instruction J or K, so we need to record the location in branch target buffer of branch instruction J and K for branch instruction I. We also need a register called “location register (LR)” to record some information temporarily.

3.2.2 Operations of Two-direction Based NBET

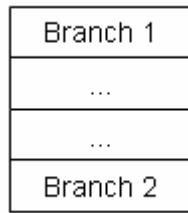


Figure 3-3: an example

Here we show that NBET how to work. Branch instruction 2 (see figure 3-3) will be executed after execution of branch instruction 1, for branch instruction 1, we need to record the location in branch target buffer. Assuming that execution of branch instruction 1 finishes in EXE stage of the processor pipeline, if branch instruction 1 is taken or it is in branch target buffer already, its information needs to be updated into branch target buffer. We need to record two things: 1) where the entry saving information of branch instruction 1 is located in branch target buffer and 2) whether branch instruction 1 is taken or not. 1) help to find the corresponding entry in BTB and 2) helps to record where next accessed entry is in taken or non-taken path. We use a register called “Location Register” (LR) to record the information (see figure 3-4).

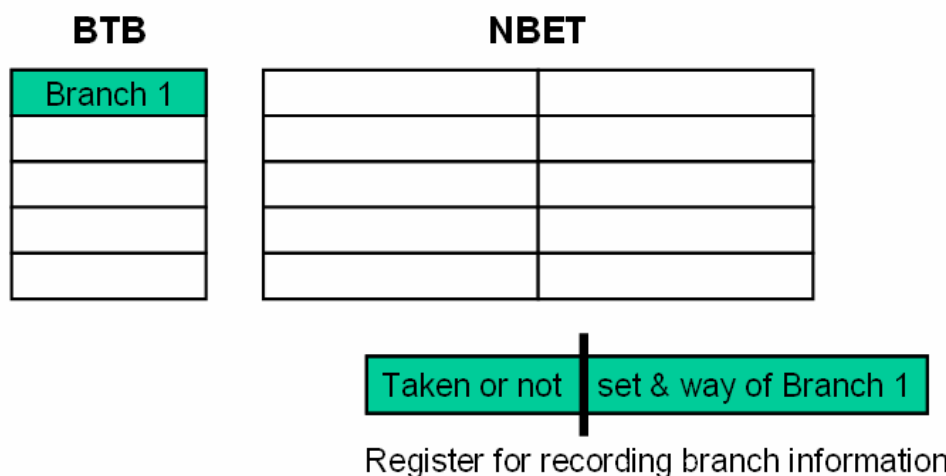


Figure 3-4: while updating branch instruction 1 into BTB, we use a register to record the necessary information

After executing branch instruction 2 in EXE stage of pipeline. if branch instruction 2 is taken or it is in branch target buffer already, its information needs to be updated into branch target buffer. Here we have known that where the branch instruction 2 is located in branch target buffer. Then we can use the information saved in LR to find the NBET entry corresponding branch instruction 1 to record the location in branch target buffer of branch instruction 2 (see figure 3-5).

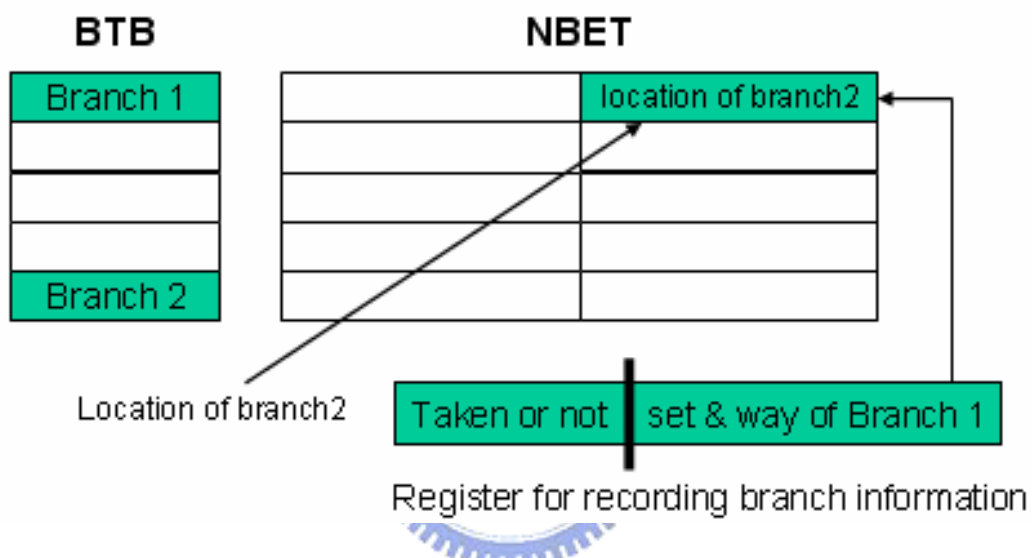


Figure 3-5: while updating branch instruction into BTB, we can write the location information of branch instruction 2 into the NBET entry corresponding branch instruction 1.

Figure 3-6 is the overview of BTB location collection circuit. Figure 3-7 shows that how to write information into location register. The SET of a branch target buffer entry can be get from the index part of PC. We need an encoder to identify in which way the entry is. We also record whether current is taken or not in the DIR field in location register. Figure 3-8 shows that we use the information of location register to find corresponding NBET entry. Then need a demultiplexer to decide which field of NBET entry to be written into.

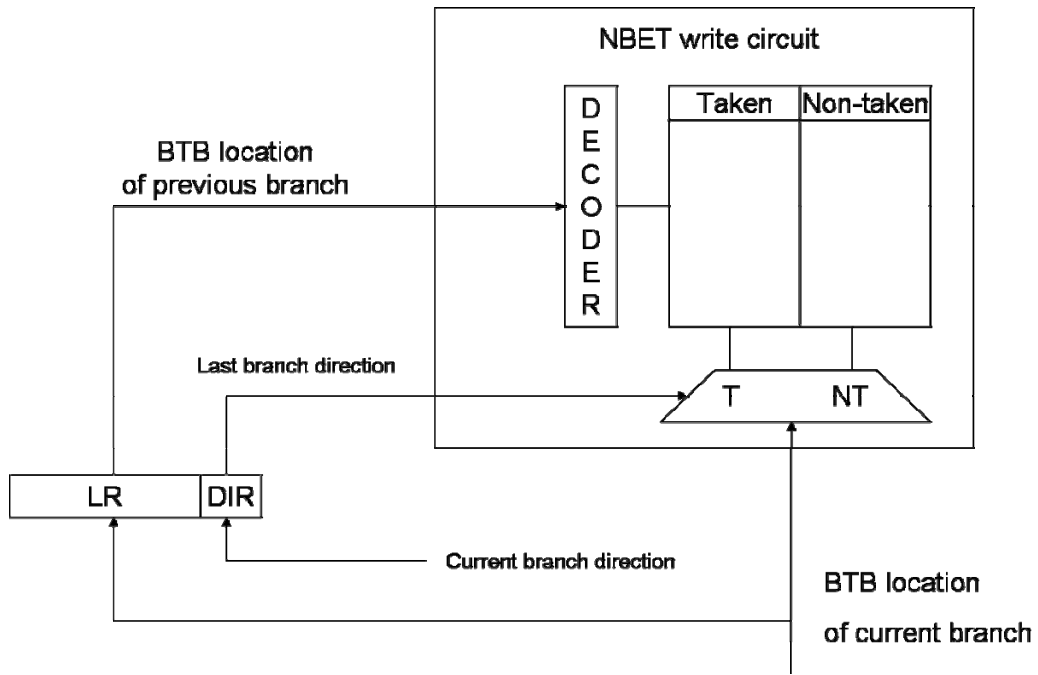


Figure 3-6: Next BTB location collection circuit

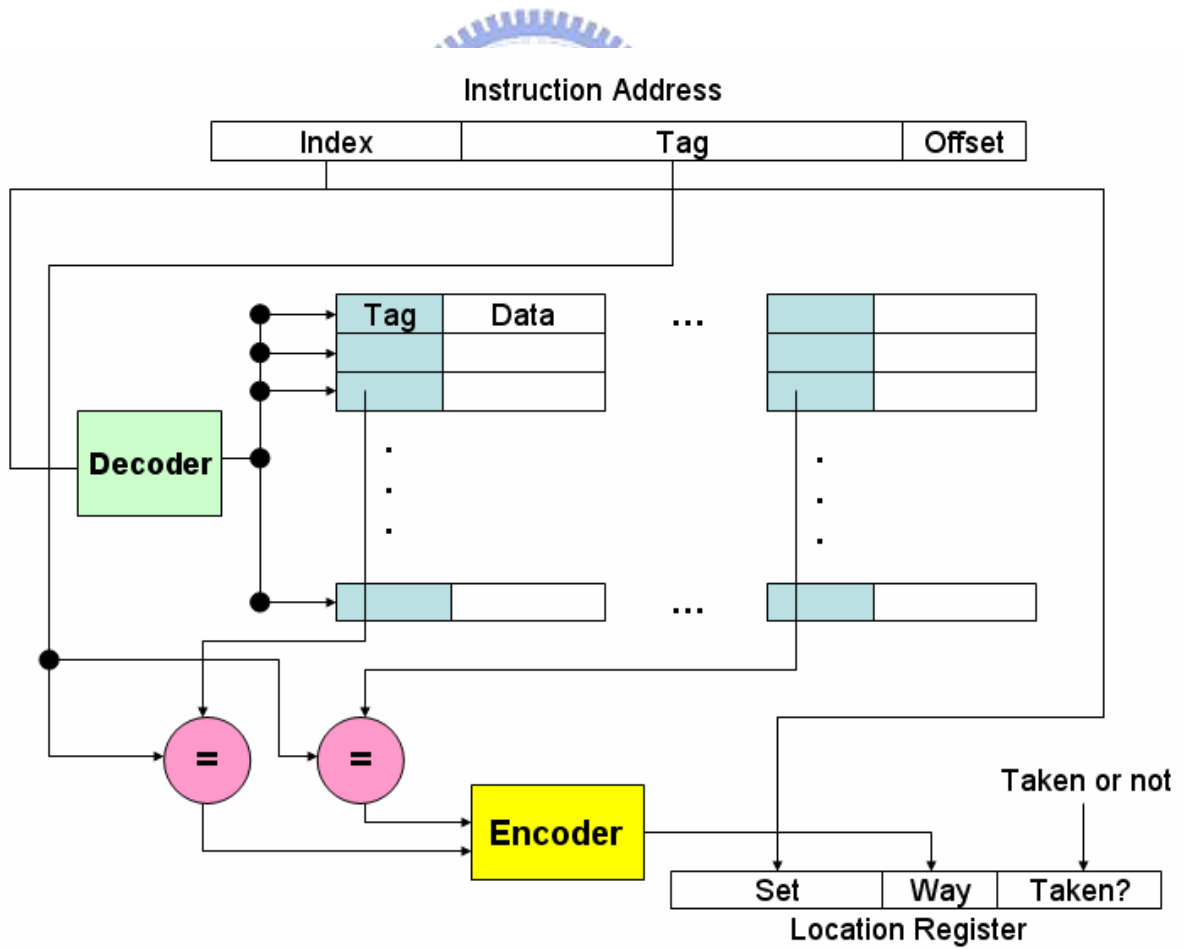


Figure 3-7: how to record NBET location in LR

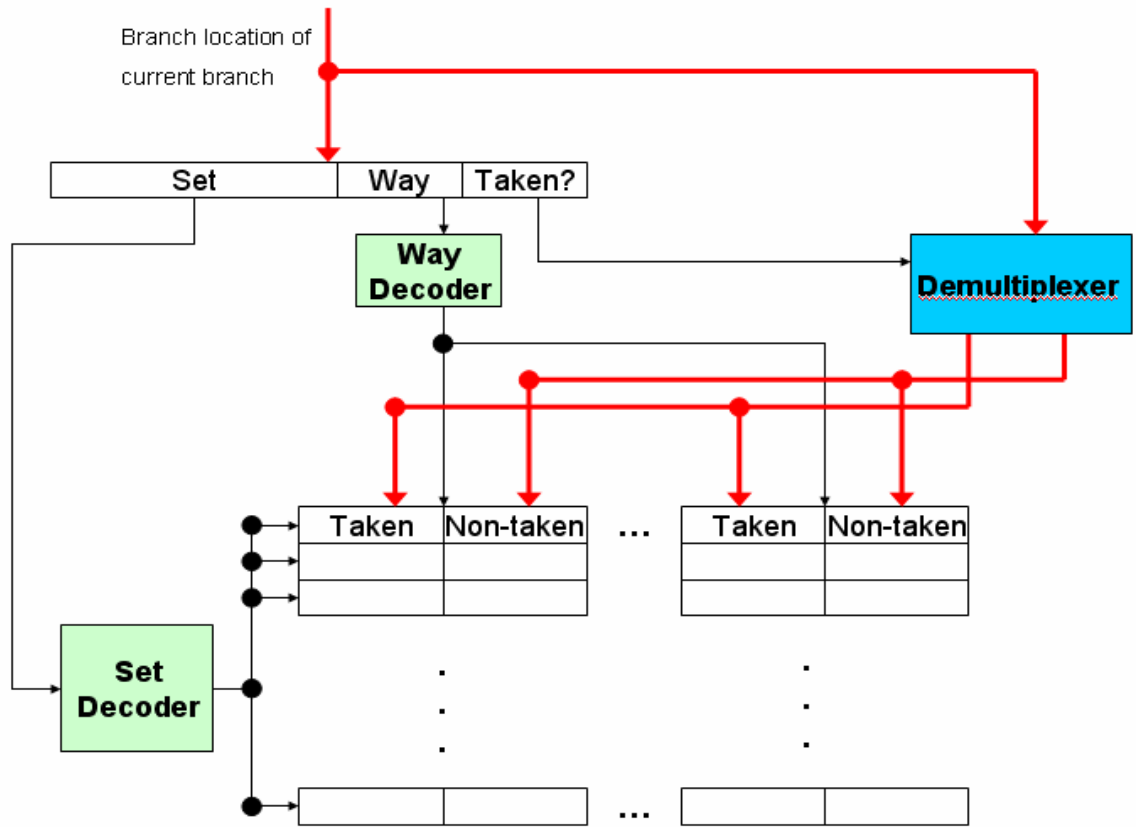


Figure 3-8: how to find corresponding NBET entry and write

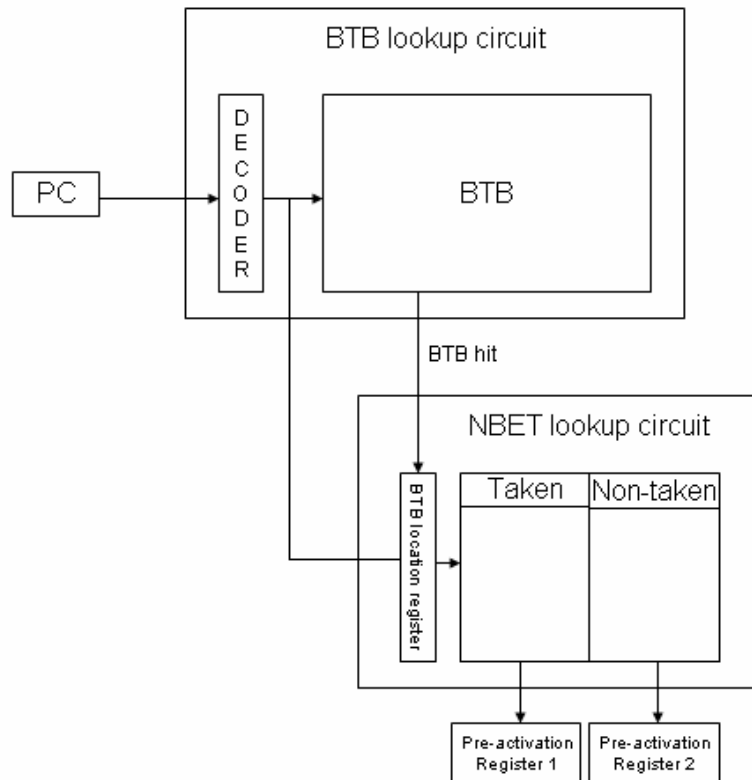


Figure 3-9: Lookup-circuit of two direction based NBET

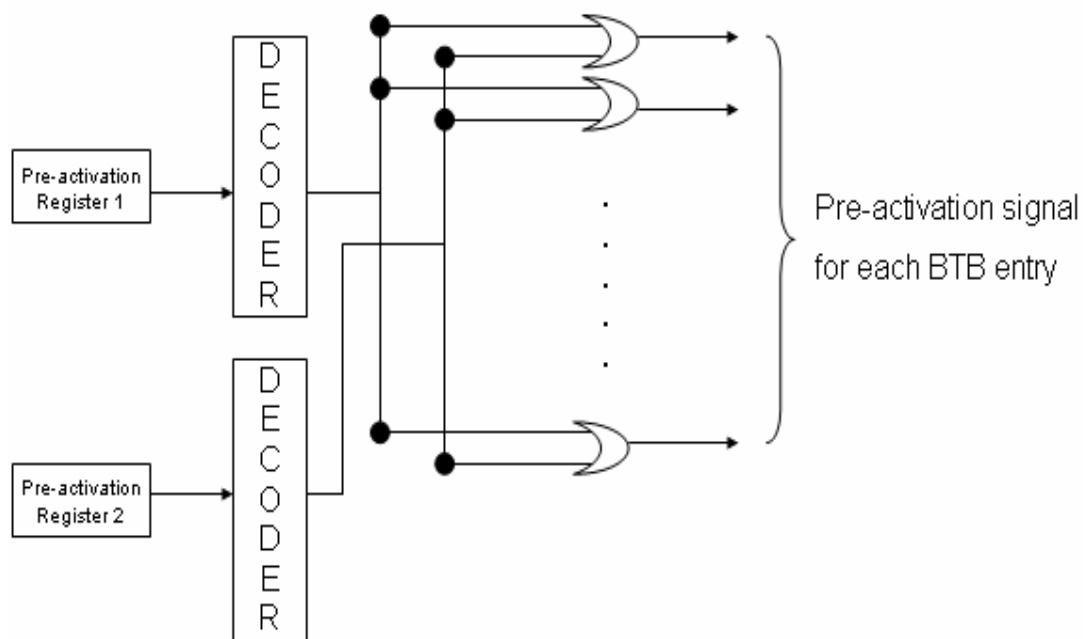


Figure 3-10: Pre-activation circuit

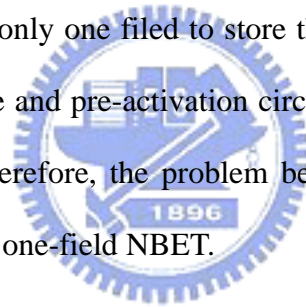
Figure 3-9 displays lookup-circuit of two-direction based NBET. BTB is looked up every cycle for target address prediction. If BTB hits, it means that current instruction is a branch instruction and the outputs of BTB row decoder are latched in BTB location register. Then, in the next cycle, the NBET lookup operation is performed to get the next BTB location. Note that this operation is performed only while the BTB lookup is hit in the previous cycle. After NBET lookup operation, the valid entries of the two corresponding NBET entries are latched in pre-activation registers. Figure 3-10 presents the pre-activation circuit. It decodes the contents of the two pre-activation registers to generate the pre-activation signals to power mode controller of drowsy BTB.

3.3 One-direction Pre-activation Policy

3.3.1: Why One-direction Pre-activation

There are two drawbacks for two-direction pre-activation. The first one is that there is a wasted field at sometimes. For example, unconditional branch instruction is always taken and highly biased branch instruction is frequently taken or not-taken. Another drawback is that if we always pre-activate the next possibly accessed BTB locations along taken and not-taken paths of a branch, at least one pre-activated BTB entry is unnecessary.

If each NBET entry has only one field to store the BTB location of next branch instruction, half of NBET size and pre-activation circuits can be saved and the above problems can be ignored. Therefore, the problem becomes how to record the most possible next BTB location in one-field NBET.



Here we introduce one-direction pre-active policy by branch predictor. Because the branch predictor can indicate the direction of a branch instruction, we would pre-activate the possible accessed along the path indicated by branch predictor. And the predicted direction can be gathered during runtime, we need not to record the next accessed entries along both direction, we need to record the next possible accessed entry along the path indicated by branch predictor.

3.3.2: Design of One-direction Pre-activation

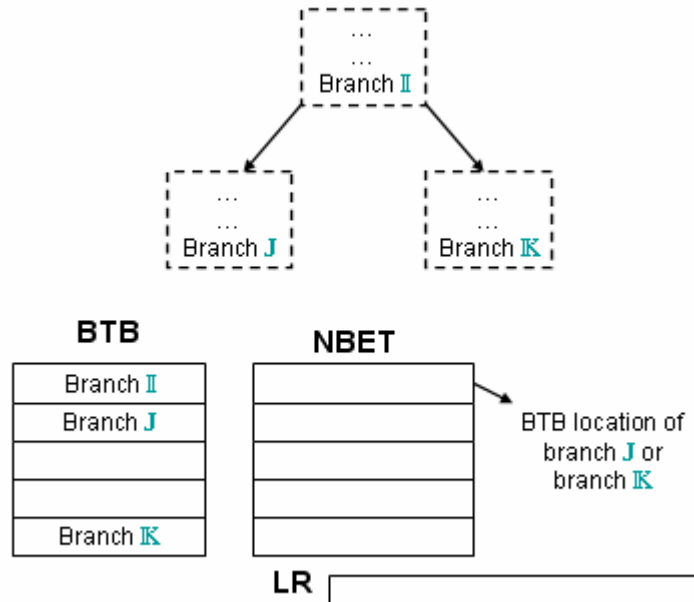


Figure 3-11: NBET architecture of one-direction pre-active policy

Figure 3-11 is the architecture overview of one-direction pre-active policy by branch predictor. After execution of branch instruction I, we will encounter branch instruction J or K. The NBET entry corresponding branch instruction I will selectively record where branch instruction J or K is located in branch target buffer. The selection is decided by branch predictor. Because the NBET record should consists with the predicted direction, so the NBET record changes only with predicted direction changing.

Figure 5 shows the state transition diagram of a typical 2-bit branch predictor. Initially, a taken branch is placed in BTB with weakly-taken (WT) state. The BTB location of next branch instruction along taken path is recorded in NBET. Then, in the following executions of the branch instruction, the NBET update is contents only when the next predicted direction is changed. In 2-bit branch predictor, only while the

predictor state changes from WT to SNT or WNT to ST, the NBET is needed to be update.

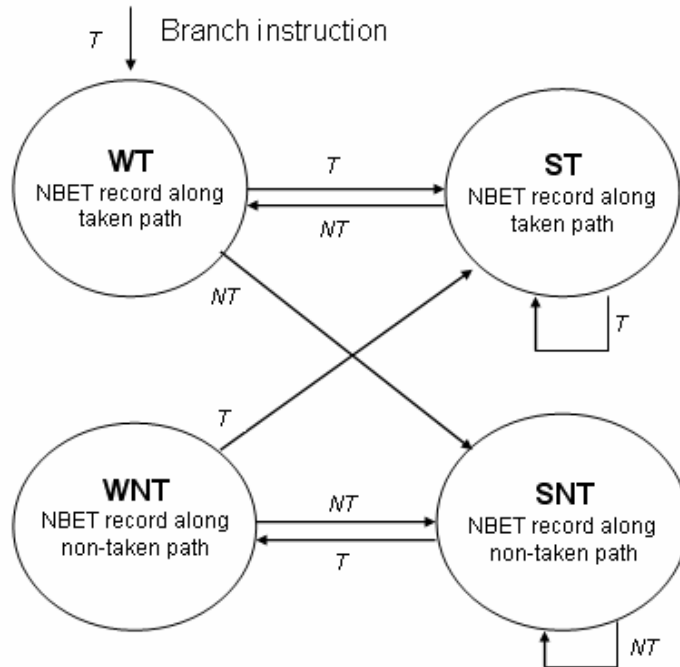


Figure 3-12: NBET recording with predictor state

3.3.3 Circuit Modification of One-direction based NBET

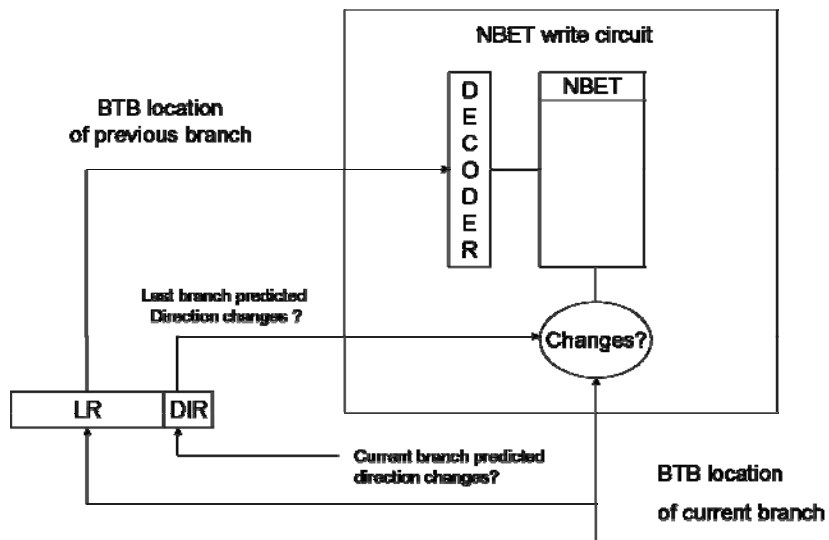


Figure 3-13: Next BTB location collection circuit of on-direction based NBET

In order to implement one-direction pre-active policy, we need to gather the kinds of information during run time: 1) whether the predicted direction changes or not and 2) where the entry saving information of a branch instruction is located in branch target buffer (see figure 3-14). While updating a branch instruction into branch target buffer, we can know the predicted direction next time by the state of branch predictor and whether current execution is taken or not. Because one NBET entry has only one field to record the next possible accessed branch target buffer, comparison with two-direction pre-active policy, half of NBET size can be reduced. While we look up branch target buffer and find current PC value is a branch instruction, we can use the value of corresponding NBET entry to pre-activate the next possible accessed branch target buffer entry.

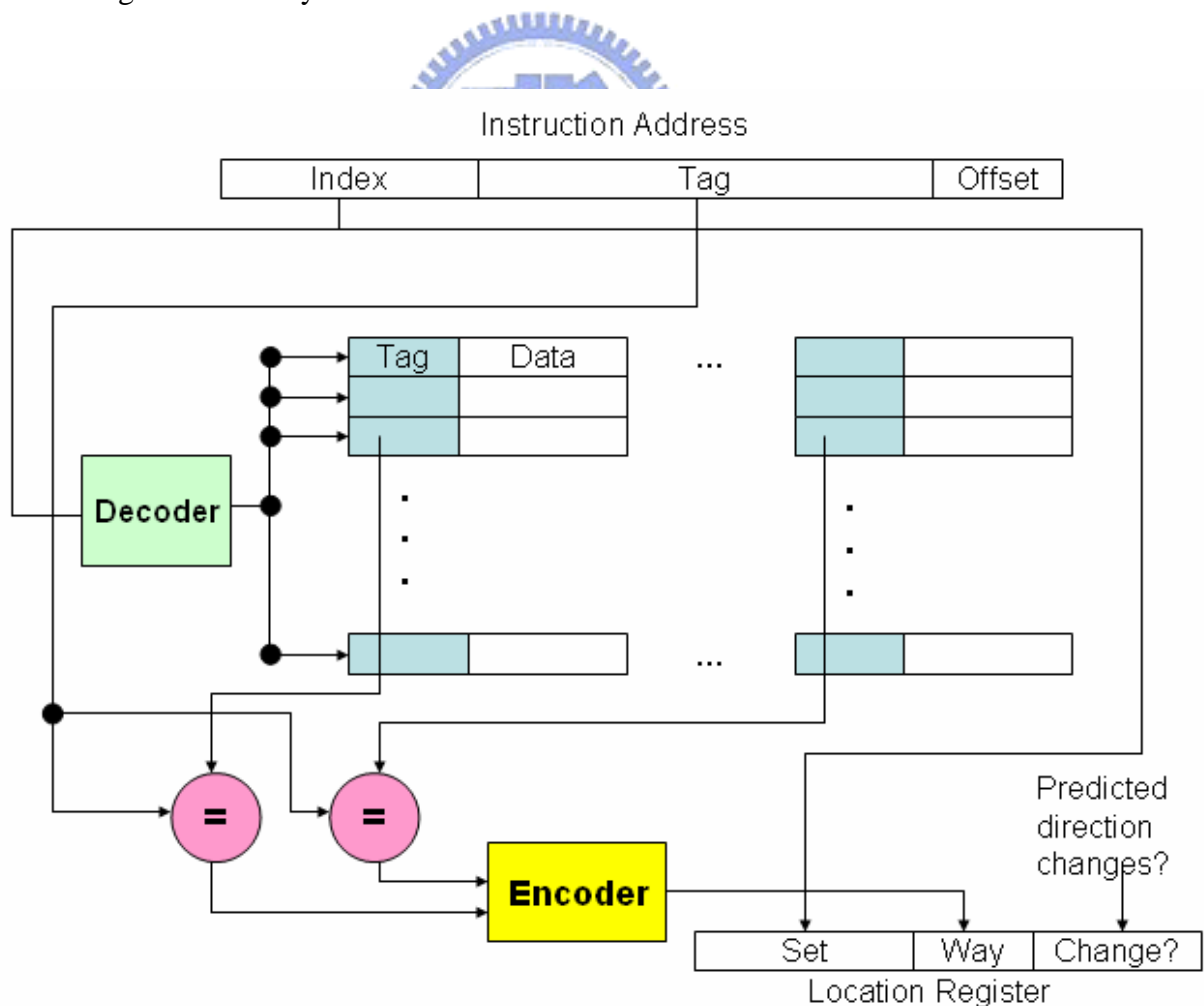


Figure 3-14: write information into location register in one-direction pre-active policy

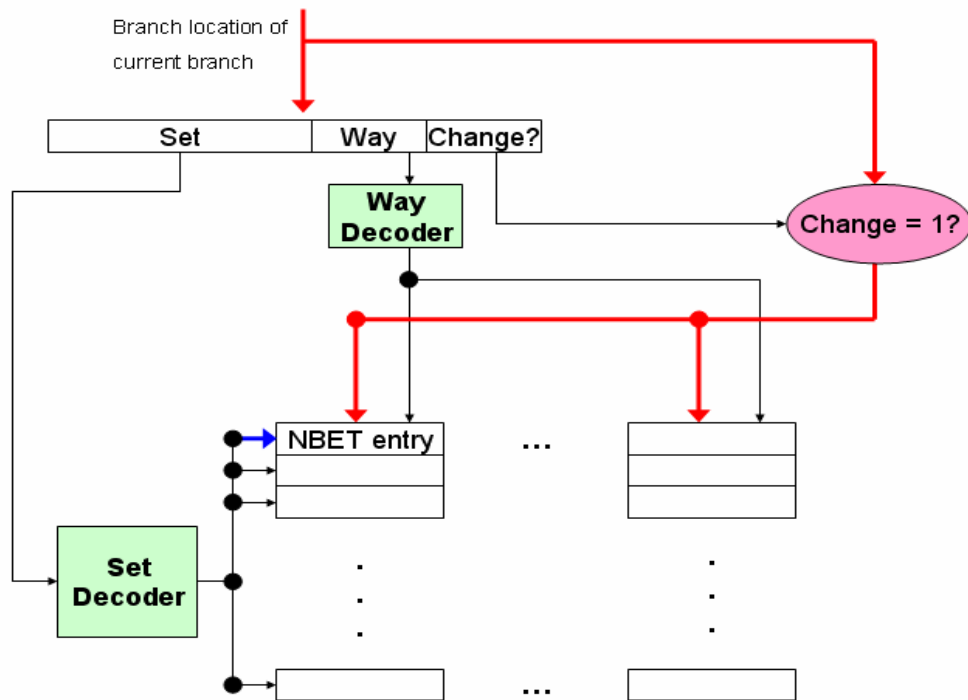
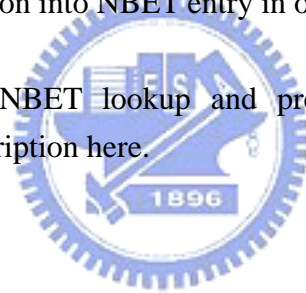


Figure 3-15: writing information into NBET entry in one-direction pre-active policy

The modifications for NBET lookup and pre-activation circuit are trivial. Therefore, we ignore the description here.



3.4 BTB Entry Deactivation

We adopt decay strategy proposed in [5] to deactivate BTB entries. In this method, a BTB entry is putted into drowsy mode if this entry has not been accessed for a period of time (decay interval). For the implementation of decay idea, a global counter and a set of local counters are required. The global counter reset itself after a period of time (global interval). The local counter adopted for each BTB entries resets itself while the corresponding BTB entry is accessed and increments itself at each time that the global interval is reached. If any local counter reaches its maximum value, the corresponding BTB and NBET entry is putted into drowsy mode. Note that the power modes of NBET are managed together to further save the NBET power overhead.

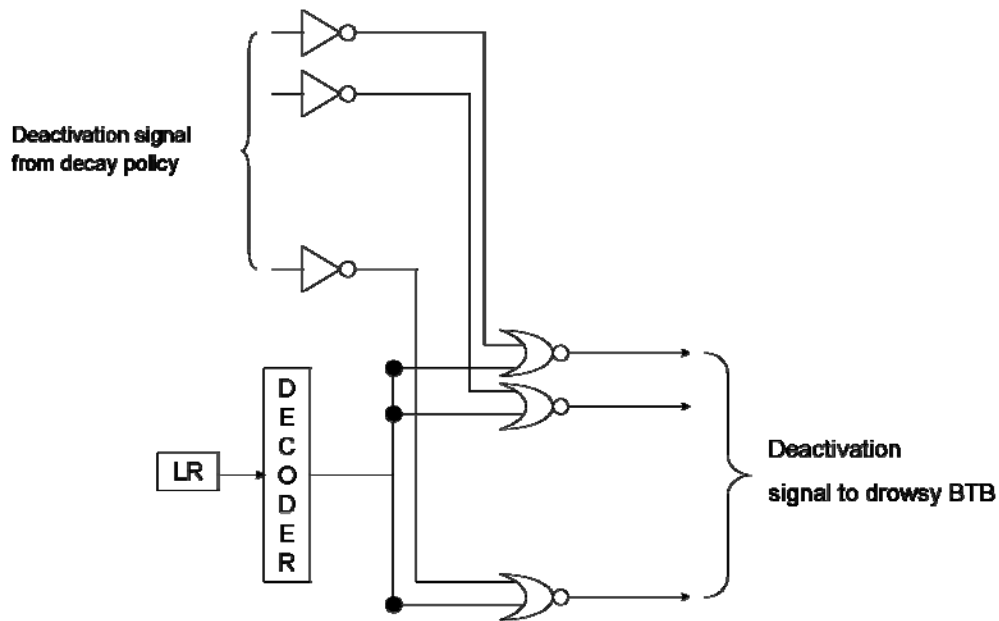


Figure 3-16: Gating the deactivation signals for most recently accessed BTB and NBET entry

With accurate pre-activation, the decay interval becomes only about hundreds of cycles, since the energy overhead due to power mode changes is very small. Unfortunately, while program execution flow enters into a large basic block, the previous accessed NBET entry may be deactivated before the next BTB location updating. Therefore, the previous accessed NBET entry should prevent to be deactivated. Figure 3-16 shows its implementation circuits. This circuit the gates deactivation signals for most recently accessed BTB and NBET entry.

3.5 Discussion

There are some situations our proposed pre-active policy can not work well. It is to say that when we access a branch target buffer entry but it is still in drowsy mode. We will discuss these situations.

First, we define perfect pre-activation. If next branch instruction is in branch target buffer already, it will be translated into to active mode before accessing. If next branch instruction is not in branch target buffer, no branch target buffer would be translated into active mode. If pre-activating a branch target buffer entry violates the roles, it is failed.

The NBET entry corresponding to a branch instruction has no information until next branch instruction finishes execution. Sometimes the next branch instruction is put into branch target buffer than current branch instruction, so the NBET has no information to pre-activate.

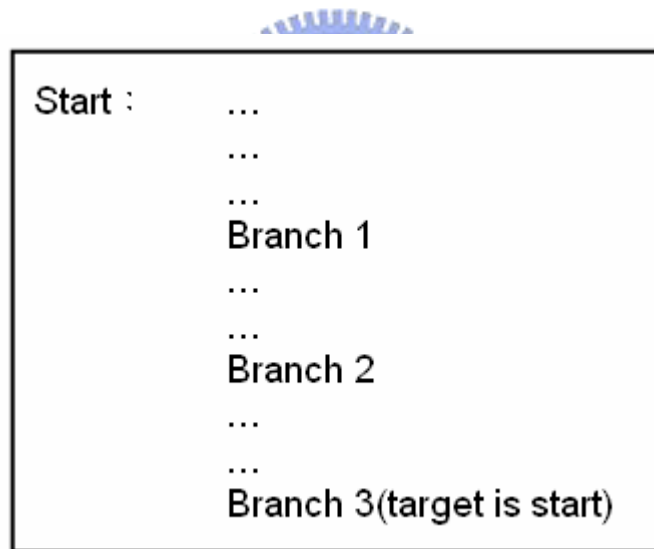


Figure 3-17: code segment of a loop

Figure 3-17 is a example of such a situation. When we enter the loop first time, we encounter branch instruction 1 first, if it is taken, it will be put into branch target buffer. In the end of this loop, we encounter branch instruction 3. if it is taken , we will jump to label “Start” and encounter branch instruction 1 next. Since we execute branch instruction 3 first time, we have no information to pre-activate the branch

target buffer entry saving branch instruction 1.

There is a period of time from drowsy mode to active mode and it is called wake-up latency. Assuming the wake-up latency is one cycle penalty. When we encounter continuous branch instruction, since we have accurate information, we still can not pre-activate the entry in time.

In our proposed design, we say that there are at most two possible directions of a branch. We will encounter next branch instructions along both paths. But indirect jump instruction will destroy the sequence. And it lets our pre-activation to be failed.

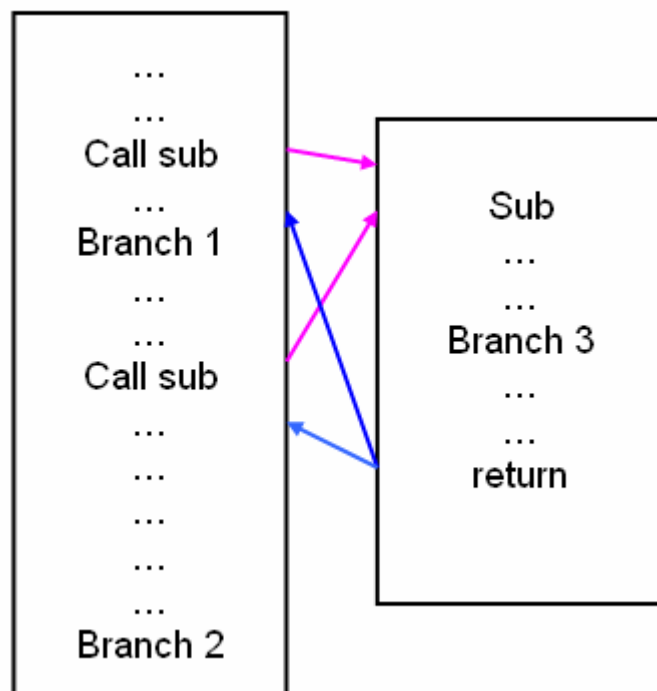


Figure 3-18: indirect jump destroys the branch instruction execution sequence

In figure 3-18, when we execute upper Call instruction, the processor will enter the subroutine and executes branch instruction 3 than 1. when we execute the second Call instruction, the processor will enter the subroutine and execute branch instruction

3 again, but it will pre-activate the branch target buffer entry saving information about branch instruction 1, not 2.

Since branch target buffer is a cache like data structure, conflict may happen sometimes. Sometimes some NBET entry will indicates the same branch target buffer entry. If a conflict happens and the value of the branch target buffer entry is replaced, Those NBET entries will save invalid information. But when we access these NBET entries, we still use the information to pre-activate.

Branch target buffer is a cache of branch instruction. The same as instruction cache or data cache, the conflict miss will happen on branch target buffer. Once miss happens, a branch instruction will be replaced and corresponding NBET value will be loss. It may make pre-activation fail.



In one-direction pre-active policy, we pre-activate the next possible accessed by the result of branch predictor. If branch predictor predicts error, then our pre-activation will fail, either. In two-direction pre-activation, we don't care the result of the predicted result, so such situation will not happen, but when NBET has valid value about taken and non-taken path, it will pre-activate one unnecessary branch target buffer entry.

Chapter 4 Evaluation

4.1 Method

It is very difficult to time-consuming to re-design processor and implement my proposed design methods into it. Another approach is using a simulator to simulate the behavior of a processor. It is very commonly used approach in architecture design. Because it is economic than re-design a processor and it is still accurate. By modifying the simulator, we can observe the result of my proposed design.

I evaluate my design by a simulation-driven simulator. Like a real processor, the simulator simulates the behavior of the components in a real processor. I will gather the execution result of my proposed design through simulating the behavior of my design.



4.2 Evaluation Metrics

In this research, I use the following metrics to evaluate my proposed design:

- BTB leakage energy consumption
- Performance loss

This two metrics are meaningful for user.

4.2.1 BTB Leakage Energy Consumption

The purpose of my proposed design is to save “energy consumption”, and I focus on branch target buffer leakage energy. The BTB leakage energy consumptions may include the extra energy caused by additional hardware and performance loss.

Therefore, it composed of the following terms:

- BTB leakage energy
- NBET energy
- System leakage energy duo to performance loss
- Energy of extra control logics

We defines the terms as follows:

- BTB leakage energy
 - leakage energy consumption of BTB
- NBET energy
 - Leakage energy consumption of NBET
- System leakage energy duo to performance loss
 - There are extra cycles due to performance, the system leakage energy increases because of execution time increment.
- Energy of extra control logics
 - Some hardware are added to implement my design, the extra hardware will consume dynamic energy. It includes the reading and writing NBET, global and local counter to implement decay, and some logic diagram.

The leakage energy of BTB and NBET is calculated by the following equation:

$$\sum_{\text{every BTB entry}} (\text{active cycles} \times \text{active energy per cycle} + \text{drowsy cycles} \times \text{drowsy energy per cycle})$$


“Active cycles” is the number of cycles that a branch target buffer entry is in active mode and “Active energy per cycle” is the leakage consumption of a branch target buffer during a processor cycle. The sum of every branch target buffer’s leakage

energy consumption is the total leakage energy consumption of a branch target buffer.

4.2.2 Performance Loss

Performance loss means the increment of execution cycle. There is a finite period of time from drowsy mode to active mode and it so called “wake-up latency”. Here we assume wake-up latency is one cycle. If we activate a drowsy branch target buffer entry on demand, the overall system must wait until the entry translated to active mode. Because the execution time increases, the system leakage energy will increase, too. The purpose of my design is to hide wake-up latency. The performance loss is another important metric of my design.

4.3 Environment



The architectural simulator used in this research is the SimpleScalar/Alpha 3.0 and xtrem1.0. SimpleScalar/Alpha is a commonly used simulation-driven simulator in architecture design domain. It is a suite tools for the Alpha ISA. Xtrem1.0 is a simulator derived form the SimpleScalar, but ii is suite to ARM ISA. Table 4-1 are the main parameters of my simulation environment:

Most of the energy numbers are obtained from the power libraries in XTREM [12] tool set. The SRAM energy parameters of different power modes and the mode transition are listed in table 4-2 [13]. The number of execution cycles is obtained from SimpleScalar/Alpha 3.0.

Table 4-1: Simulation parameters

Parameter	Value
Inst. Window	16-RUU, 8-LSQ
Issue Width	4 instructions per cycle
Function Units	4 IntALUs 4 FPALUs
L1 I-cache	16KB, 2-way, 32B block
L1 D-cache	16KB, 2-way, 32B block
I-TLB	32 entries, fully assoc
D-TLB	32 entries, fully assoc
BTB	512-entries, 4-way
Direction	Bimodal predictor build
Predictor	in BTB

Table 4-2: leakage .energy parameters

Parameter	Value
Active leakage energy per BTB entry	0.33 pJ/cycle
Drowsy leakage energy per BTB entry	0.0495 pJ/cycle
Transition energy	11 pJ

The benchmark I selected in this research are Mibench and SPEC2000 benchmark.

Mibench is a free, commercially representative embedded benchmark suite and consist of six categories

- Automotive and Industrial Control
- Consumer Device
- Network
- Office
- Security
- Telecommunications

SPEC2000 is another commonly used benchmark for high end processor. I use these two kinds of benchmark to examine the effectiveness of my design in different domain of application.

4.4 Experimental Results

Figure 4-1 shows the BTB leakage energy consumption with two-direction pre-activation policy. The Y-axis in the figure is the ratio of branch target buffer leakage energy consumption of my design normalized to original branch target buffer leakage energy consumption. The X-axis are my proposed design with different decay interval.

The most left bar chart of figure is ideal case. From leakage energy parameters, we have the equation:

$$\frac{\text{transition energy}(11pJ)}{\text{active energy}(0.33pJ / cycle) - \text{drowsy energy}(0.0495pJ / cycle)} = 39.21...$$

For a branch target buffer entry, if the times between two successive accessing are more than 40 cycle. We will gain leakage reduction if we put the entry into drowsy mode. If the time is less than 40 cycles, the entry should be in active mode. The ideal case is to obey the above rule, always pre-activate accurately, and has best energy saving.

The most right chart bar is the simulation of related work [5]. I will compare with these two policies.

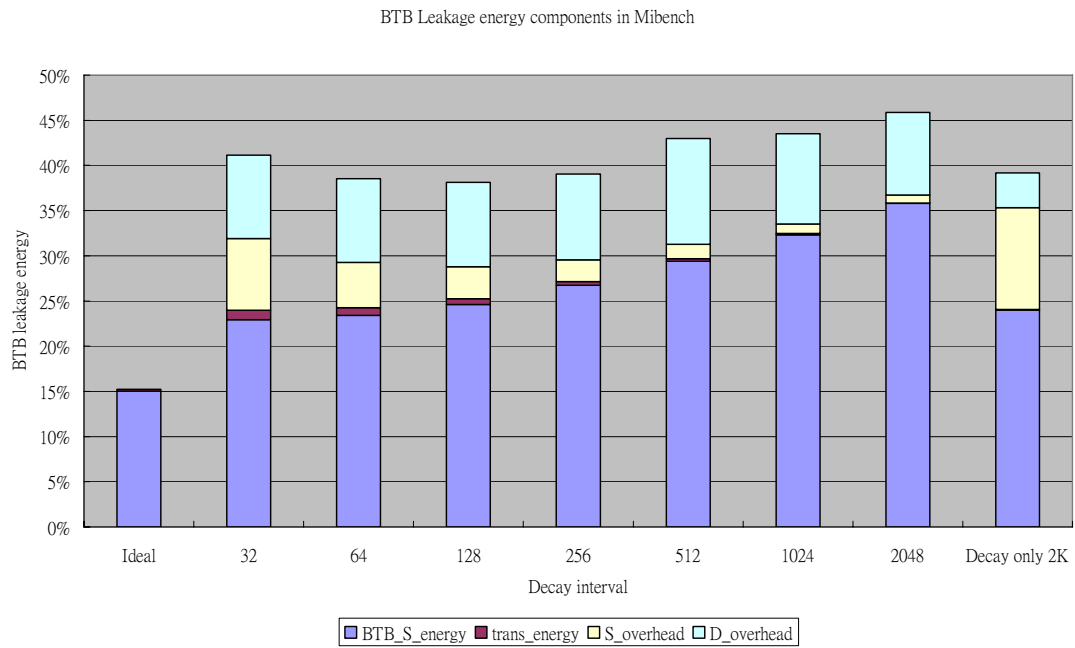


Figure 4-1: BTB leakage energy components with two-direction pre-activation of Mibench

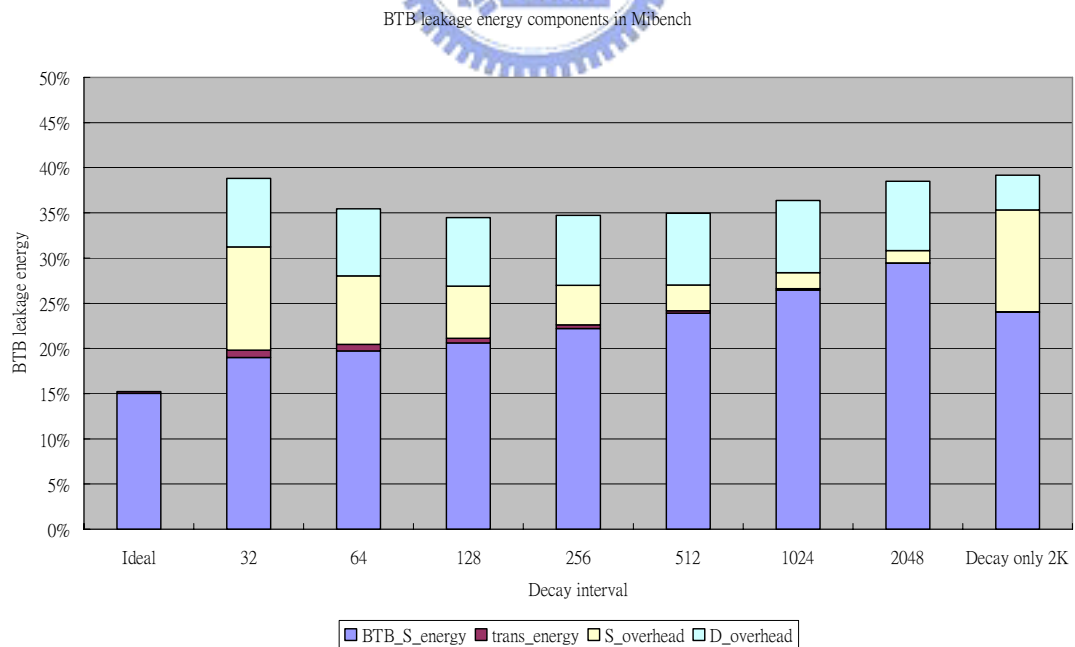
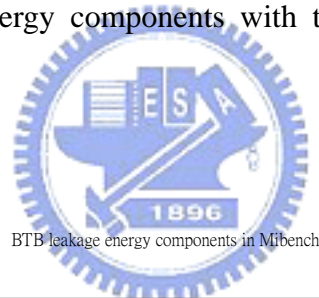


Figure 4-2: BTB leakage energy components with one-direction pre-activation of Mibench

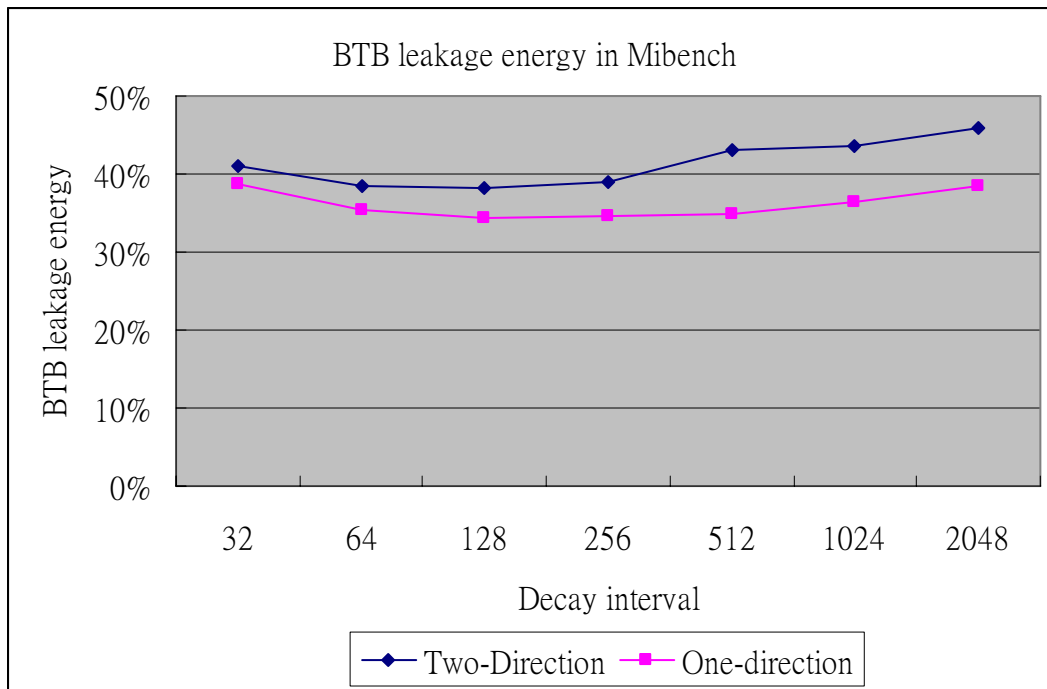


Figure 4.3: comparison of two-direction and one-direction policy in Mibench

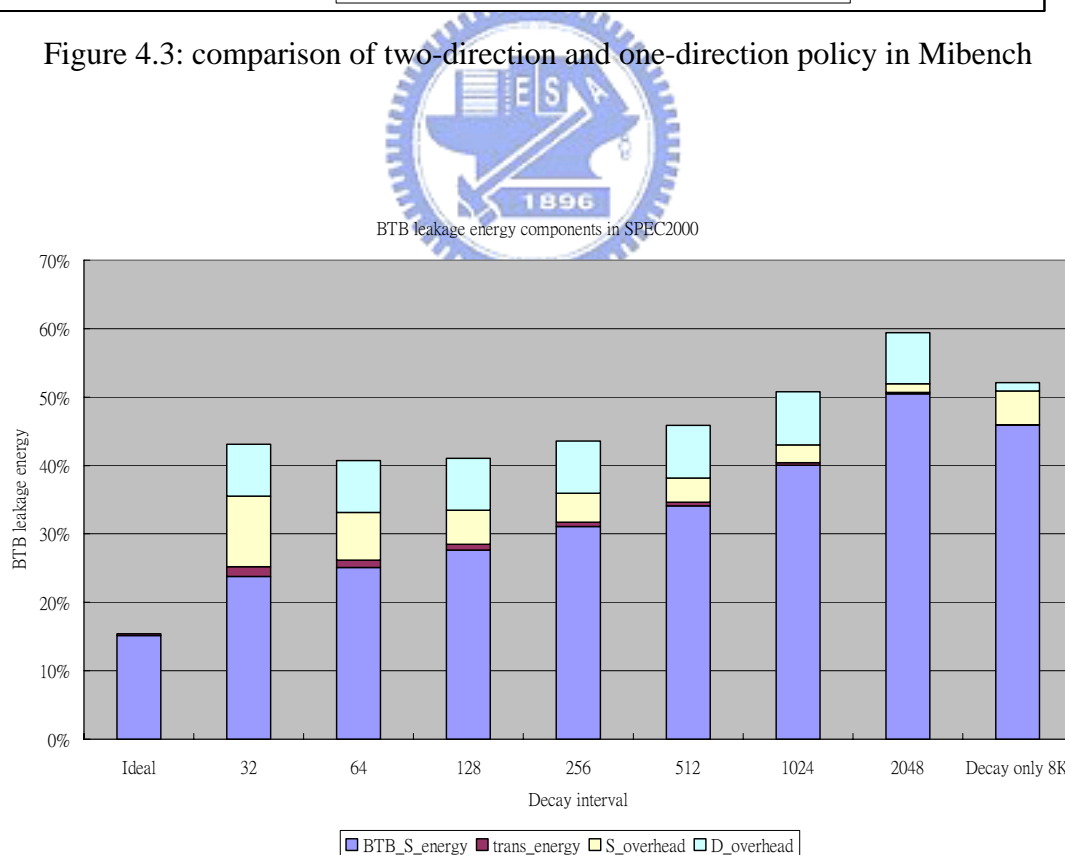


Figure 4-4: BTB leakage energy components with two-direction pre-activation of SPEC2000

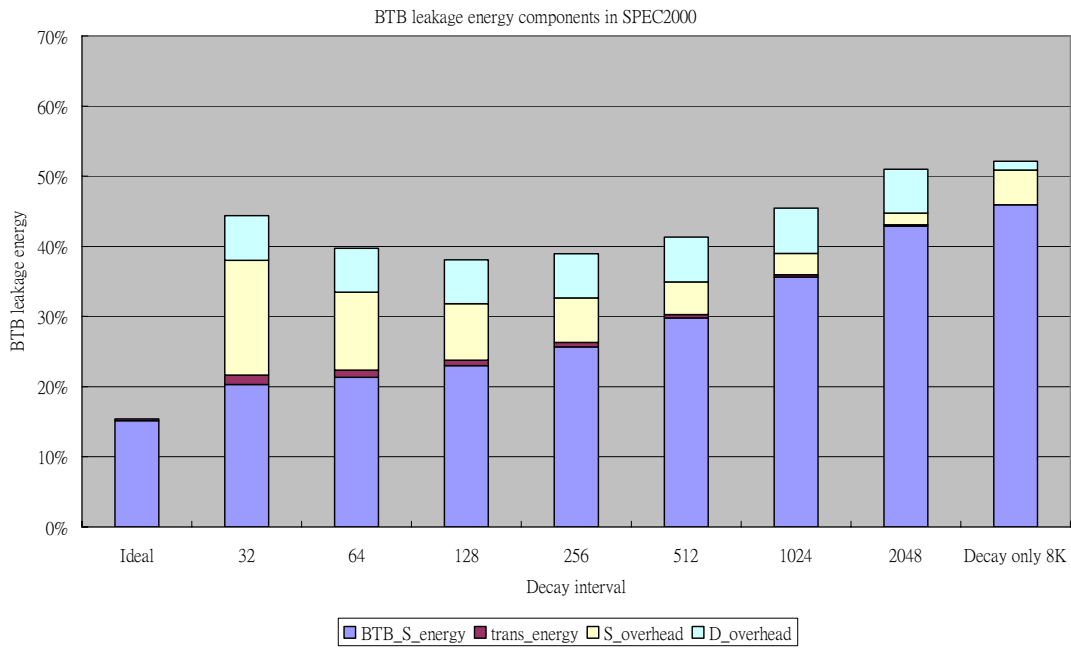


Figure 4-5: BTB leakage energy components with one-direction pre-activation of SPEC2000

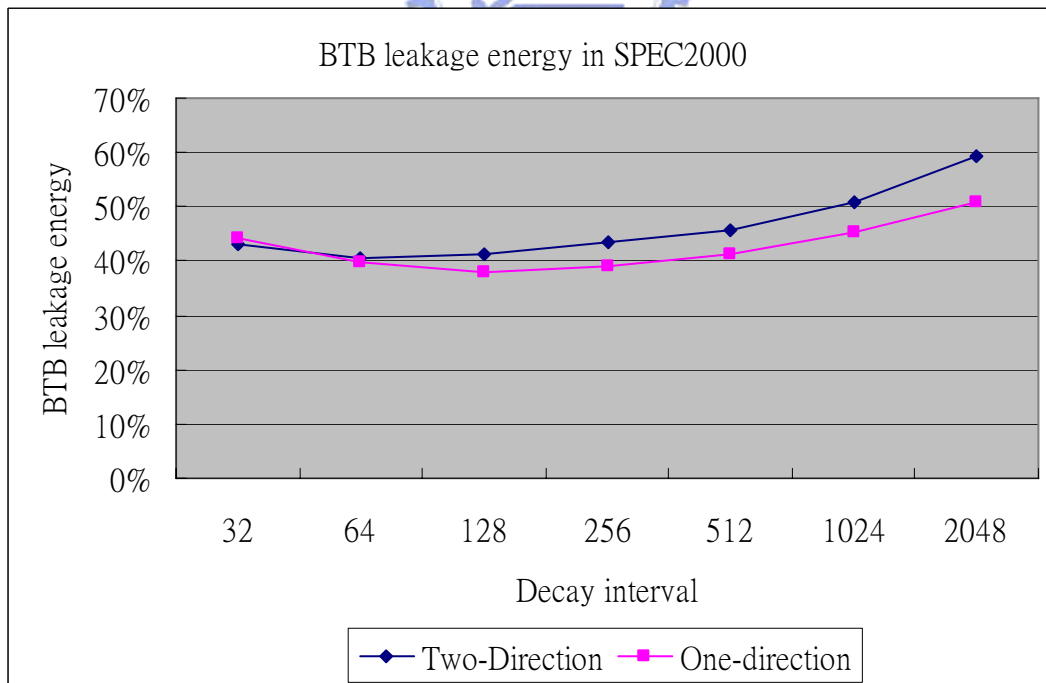


Figure 4.6: comparison of two-direction and one-direction policy in SPEC2000

4.5 Discussion

From figure 4-1 and figure 4-2, my design has about 5% better than related work in Mibench. From figure 4-4 and figure 4-5, my design has about 14% better than related work in SPEC2000. The characteristic of the benchmarks makes the result. Mibench has smaller loop than SPEC2000 and it has the ratio of branch instruction is smaller than SPEC2000, too. The decay-only strategy has good effect of leakage energy saving already in Mibench. Although we switch the mode of branch target buffer entries more aggressively, the improvement is not obvious. In SPEC2000, my proposed design has better effect. Table 4-3 is my best situation comparing to related work in these two benchmarks.

Table 4-3: best situation in my design

Benchmark	Strategy	Energy saving	Decay only
Mibench	One-direction with decay 128	0.047	0.391836
SPEC2000	One-direction with decay 128	0.141	0.521057

From figure 4-3 and figure 4-6, we also find that one-direction pre-active policy is better than two-direction pre-active policy. Although one-direction has poor performance because branch prediction error, it reduces half of NBET size.

Putting branch target buffer entries into drowsy mode more aggressively may have better branch target buffer leakage energy saving, but we will encounter more mode switching. We find that with decay interval decreasing, the leakage energy of branch target buffer decreases and the system leakage energy increases. So the decay interval is not better with smaller value. In my experiment, the best value of decay interval is 128 cycles.

Performance loss is an important metric of my design. Table 4-4 introduces the performance loss in my best situation. my proposed still keeps the performance well.

Table 4-4: performance loss of best strategies

benchmark	My strategy	Performance loss
Mibench	One-direction with decay 128	0.53%
SPEC2000	One-direction with decay 128	0.57%

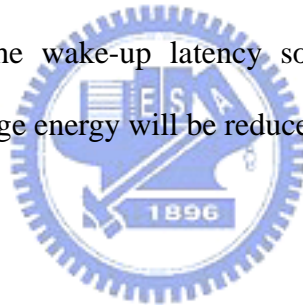


Chapter 5 Conclusion and Future Work

5.1 Conclusion

Since accessing branch target buffer pattern has the characteristic of locality. Only a small part of branch target buffer will be accessed in a period of time. We can put the non-accessed branch target buffer entries into drowsy mode to reduce leakage energy of branch target buffer.

If we put branch target buffer entries into drowsy mode more aggressively, we gain more leakage energy reducing. But it will introduce serious wake-up latency. My proposed effectively hides the wake-up latency so that the execution time will increase only a few. The leakage energy will be reduced effectively.



5.2 Future Work

In this research, we reduce the leakage energy of branch target buffer effectively. But there are other directions to future reduce energy consumption of branch target buffer.

In some processors, the branch target buffer separates from branch predictor. Only when branch predictor predicts taken, it needs to access branch target buffer. We can record the branch target buffer accessed pattern to pre-activate.

Moreover, several related research directions worth further studying. For

example, power mode managements for instruction caches, data caches, and L2 caches. The management policies are designed according to the different access patterns



Reference

- [1] NS Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, JS Hu, Mj Irwin, M. Kandemir, and V. Narayanan, "Leakage Current: Moore's Law Meets Static Power", IEEE Computer Society, Volume 36, Issue 12, pages 68-75, December 2003.
- [2] D. Parikh, K. Skadron, Y. Zhang, and M. Stan. "Power-aware Branch Prediction: Characterization and Design", IEEE Transactions on Computers, Volume 53, Issue 2, pages 168-186, February 2004.
- [3] K. Skadron, T. Abdelzaher, and M. R. Stan. "Control-theoretic Technique and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management", In Proceedings of the 8th International Symposium on High-Performance Computer Architecture, pages 17-28, February 2002.
- [4] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. "Drowsy caches: Simple techniques for Reducing Leakage Power", In Proceeding of the 29th International Symposium on Computer Architecture, pages 148-157, May 2002.
- [5] Z. Hu, P. Juang, K. Skadron, D. Clark and M. Narttonosi "Applying Decay Strategies to Branch Predictors for Leakage Energy Savings", In Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors, pages 442-445, September 2002.
- [6] M. D. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated Vdd: A Circuit Technique to Reduce Leakage in Cache Memories", In Proceedings of the [International Symposium on Low Power Electronics and Design](#), pages 90-95, July 2000.
- [7] N. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Circuit and Microarchitectural Techniques for Reducing Cache Leakage Power", IEEE Transactions on Very Large Scale Integration Systems, Volume 12, Issue 2, pages 167-184, February 2004.
- [8] D. Parikh, K. Skadron, Y. Zhang, and M. Stan, "Power-aware Branch Prediction: Characterization and Design", 2004.

[9]M. Monchiero, G.Palermo, M. Sami, C.Silvano, V. Zaccaria, and R. Zafalon, “Power-aware Branch Prediction Techniques: A Compiler-hints based Approach for VLIW Processors”, 2004

[10] Wei Zhang, Bramha Allu “Loop-based Leakage Control for Branch Predictors”, In Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, pages 149-155, September 2004.

[11] D.C. Burger and T. M. Austin. “The SimpleScalar Tool Set, Version 2.0”, ACM SIGARCH Computer Architecture News, Volume 25, Issue 3, pages 13-25, June 1997.

[12] Gilberto Contreras, Margaret Martonosi, Jinzhan Peng, Roy Ju, and Guei-Yuan Lueh “XTREM: A Power Simulator for the Intel XScale ® Core”, In Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, pages 115-125, June 2004.

[13] W. Zhang, M. Karakoy, M. Kandemir, G. Chen “Reducing Data Cache Leakage Energy Using a Compiler-based Approach”, ACM Transactions on Embedded Computing Systems, Volume 4, Issue 3, pages 652-678, August 2005.

