國立交通大學

資訊科學與工程研究所

博 士 論 文

低功率分支目標緩衝器

Low-power Branch Target Buffer

研 究 生：喬偉豪

指導教授：鍾崇斌　教授

中 華 民 國 九 十 七 年 七 月

低功率分支目標緩衝器
Low-power Branch Target Buffer

研　究　生：喬偉豪　　　Student：Wei-Hau Chiao

指導教授：鍾崇斌　　　Advisor：Chung-Ping Chung

國　立　交　通　大　學
資　訊　科　學　與　工　程　研　究　所
博　士　論　文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in

Computer Science and Engineering

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

低功率分支目標緩衝器

學生：喬偉豪　　　　　　　　　　　　　　　　　　　指導教授：鍾崇斌 博士

國立交通大學 資訊科學與工程研究所

摘　　　　要

本論文探討低功率分支目的緩衝器之省電設計。我們首先紀錄在程式軌跡中兩相鄰分支指令間的非分支指令數量以減少不必要的分支目的緩衝器查詢動作。再透過 block address based indexing 以及 entry buffering，節省了分支目的緩衝器的存取功率。此外，為了節省分支目的緩衝器的靜態耗電，我們採用 decay-based 電源模式管理器並且提出一個 entry pre-activation 技術使得該電源模式管理器更有效率。

另一方面，我們也探討減少分支目的緩衝器儲存空間的技術。將指令快取記憶體之標籤記憶體與分支目的緩衝器共享可減短分支目的緩衝器的 entry 長度。再進一步透過提早產生分支目的位址的技術，可減少分支目的緩衝器之 entry 數量。此兩技術不但可以節省分支目的緩衝器所需的儲存空間，還可同時節省分支目的緩衝器的動態以及靜態功率。

最後，我們整合以上技術，使分支目的緩衝器在可容忍的效能下降前提下更省電。

Low-power Branch Target Buffer

- -

student：Wei-Hau Chiao　　　　　　　　Advisors：Dr. Chung-Ping Chung

Institute of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

This thesis addresses on low-power branch target buffer design. Through recording the number of non-branch instructions between a branch instruction and its subsequent instruction on execution path. The unnecessary BTB lookups are reduced. Through block address based indexing and entry buffering, the BTB access energy is also reduced. In order to reduce BTB leakage power, a decay-based power manager is applied and an entry pre-activation technique that makes the decay-based power manager being more efficient is developed.

On the other hand, we also address on the storage cost reduction techniques for BTB. Through sharing the tag memory of instruction cache to BTB, the BTB entry length is shortened. Moreover, through generating the branch target address early, the number of BTB entries can be reduced. These two techniques not only reduce the BTB storage, but also reduce both BTB dynamic and leakage power significantly.

Finally, we integrate the above techniques to further reduce BTB power consumption with tolerable performance degradation.

Contents

摘　　要 I

ABSTRACT II

CONTENTS III

LIST OF FIGURES VI

LIST OF TABLES VIII

CHAP 1 INTRODUCTION 1

CHAP 2 BACKGROUND 4

2.1 BRANCH PREDICTION 4

2.2 BTB FUNDAMENTALS 6

2.3 SIMULATION ENVIRONMENT 7

2.3.1 Simulator and Benchmarks 7

2.3.2 Power and Energy Models 8

CHAP 3 UNNECESSARY BTB LOOKUP FILTERING 10

3.1 RELATED WORK 10

3.2 DESIGN FOR ELIMINATING UNNECESSARY BTB LOOKUPS 12

3.2.1 System Overview 12

3.2.2 NBD collection and NBDT management 13

3.2.3 Filtering the unnecessary BTB and direction predictor lookups 16

3.2.4 Handling of incorrect BTB and direction predictor accesses 18

3.3 EXPERIMENTS 18

3.3.1 Energy Model 19

3.3.2 Results 20

3.3.3 Comparisons of the existing methods *24*

3.4 SUMMARY 29

CHAP 4 REDUCING BTB ACCESS POWER THROUGH BLOCK-ADDRESS BASED BTB INDEXING AND ENTRY BUFFERING 30

4.1 OBSERVATIONS OF BTB INDEXING METHODS 30

4.2 DESIGN FOR BTB ACCESS POWER REDUCTION 31

4.3 EXPERIMENTS 33

4.3.1 Goals *33*

4.3.2 Energy Model *34*

4.3.3 Results and analysis *34*

4.4 SUMMARY 39

CHAP 5 REDUCING BTB LEAKAGE POWER THROUGH POWER MANAGEMENT 40

5.1 RELATED WORKS 40

5.1.1 BTB power modes *40*

5.1.2 Power management for BTB *41*

5.2 DESIGN FOR BTB POWER MANAGEMENT 41

5.2.1 Decay-based power manager *41*

5.2.2 Entry pre-activation *42*

5.2.3 Implementation *44*

5.3 EXPERIMENTS 46

5.4 SUMMARY 52

CHAP 6 REDUCING BTB ENTRY SIZE THROUGH I-CACHE BASED BTB 53

6.1 RELATED WORK 53

6.2 DESIGN FOR I-CACHE BASED BTB 54

6.2.1 Mapping methods for I-cache lines and BTB entries....54

6.2.2 ICBTB management....55

6.3 EXPERIMENTS....60

6.3.1 Results and analysis....60

6.4 SUMMARY....63

CHAP 7 REDUCING THE NUMBER OF BTB ENTRIES THROUGH EARLY TARGET ADDRESS GENERATION....65

7.1 DESIGN FOR EARLY TARGET ADDRESS GENERATION....65

7.1.1 Overview....65

7.1.2 PC-relative branch handling unit (PCBHU)....67

7.1.3 Timing analyses and pipelining PCBHU....69

7.2 EXPERIMENTS....72

7.2.1 Energy model....72

7.2.2 Results and analysis....73

7.3 SUMMARY....75

CHAP 8 INTEGRATIONS....76

8.1 INTEGRATION OF POWER REDUCTION ONLY METHOD....76

8.2 INTEGRATION OF STORAGE COST REDUCTION METHODS....78

CHAP 9 CONCLUSION....80

REFERENCE....81

List of Figures

FIGURE 2-1: DYNAMIC BRANCH PREDICTOR 5
FIGURE 2-2: BTB 5
FIGURE 2-3: BTB MANAGEMENT 6
FIGURE 3-1: PPD METHOD 11
FIGURE 3-2: THE FUNCTION BLOCKS OF THIS DESIGN INTEGRATED IN A TYPICAL PIPELINE FRONT-END ... 13
FIGURE 3-3: NBD COLLECTION AND NBDT MANAGEMENT 14
FIGURE 3-4: THE CIRCUITS OF THE NBD CALCULATION AND WRITE CONTROLS OF NBDT 15
FIGURE 3-4: NEXT LOOKUP FILTER AND ITS CORRESPONDING LOOKUP FILTERING CONTROLS 17
FIGURE 3-5: TOTAL BRANCH DISTANCE RATIO FOR EACH BASIC BLOCK SIZE INTERVAL 20
FIGURE 3-6: BRANCH PREDICTOR LOOKUP RATIOS 22
FIGURE 3-7: ENERGY RESULTS FOR THE PROPOSED DESIGN (A) EBTB&LP_OUR WITH DIFFERENT VALUES OF N. (B) EBTB&LP_OUR DISTRIBUTIONS USING N=5 22
FIGURE 3-8: ENERGY COMPARISONS BETWEEN THE PROPOSED DESIGN AND THE RELATED WORKS 27
FIGURE 3-9: BTB LOOKUP RATIO COMPARISONS BETWEEN THE PROPOSED DESIGN AND THE RELATED WORKS 28
FIGURE 3-10: PERFORMANCE LOSS RATIO FOR THE RELATED WORKS 28
FIGURE 4-1: BTB INDEXING METHOD 30
FIGURE 4-2: LOW-POWER BTB LOOKUP 33
FIGURE 4-3: EVALUATIONS FOR THE OBSERVED BTBS 37
FIGURE 4-4: ENERGY RESULT FOR BABTB WITH EB 39
FIGURE 5-1: DECAY-BASED POWER MANAGERS 41
FIGURE 5-2: THE PROPOSED POWER MANAGEMENT. 45
FIGURE 5-3: COMPARISONS FOR ALL POWER-MANAGED BTBS. 52

Figure 6-1: ICBTB and its integration with a typical pipeline front-end 55

Figure 6-2: 2-grouping ICBTB attached to a 2-way I-cache..57

Figure 6-4: NPC accuracy comparisons 61

Figure 6-5: IPC comparisons..61

Figure 6-6: BTB dynamic energy comparisons...63

Figure 6-7: BTB leakage energy comparisons..63

Figure 6-8: $E_{BTB\&LP}$ comparisons 63

Figure 7-1: Ratio of PC-relative branches over total branches 65

Figure 7-2: Typical pipeline front-end with PC-BHU and RBTB 67

Figure 7-3: PCBHU...69

Figure 7-4: Pipelined PCBHU 70

Figure 7-5: Normalized IPC for the baseline BTB (256-entry 4-way), and the PCBHU design with various RBTBs..74

Figure 7-6: $E_{BTB\&LP}$ of a single stage PCBHU with 64-entry 2-way RBTB....................................74

Figure 7-7: $E_{BTB\&LP}$ of 2-stage PCBHU with 128-entry 2-way RBTB 75

Figure 8-1: Integration of NBD method and BABTB with EB...77

Figure 8-2: Integration of ICBTB and PCBHU...78

List of Tables

TABLE1-1: GOALS AND RESEARCH DIRECTIONS IN THIS THESIS 2

TABLE 2-1: SIMULATION PARAMETERS 7

TABLE 3-1: NBD COLLECTION PROCESS 16

TABLE 3-2: LOOKUP FILTERING PROCESS 17

TABLE 3-3: THE REASONS FOR NBD MISPREDICTIONS 21

TABLE 3-4: RESULTS OF DELAY ANALYSIS 23

TABLE 3-5: COMPARISONS OF THE LOW-POWER BRANCH PREDICTION TECHNIQUES 25

TABLE 4-1: ENERGY PARAMETERS 34

TABLE 4-2: EXPERIMENTS 35

TABLE 4-3: IPC, BTB DYNAMIC ENERGY, AND BTB LEAKAGE ENERGY OF BABTB AND CBTB........ 38

TABLE 5-1: BTB POWER MODES 40

TABLE 5-2: ENERGY PARAMETERS 47

TABLE 5-3: EXPERIMENTS 48

TABLE 7-1: ENERGY PARAMETERS 73

Chap 1 Introduction

Computer scientists have always tried to improve the performance of computers. But although today’s computers are much faster and far more versatile than their predecessors, they also consume a lot of power; so much power, in fact, that their power densities and concomitant heat generation are rapidly approaching levels comparable to nuclear reactors. These high power densities impair chip reliability and life expectancy, increase cooling costs, and, for large data centers, even raise environmental concerns.

At the other end of the performance spectrum, power issues also pose problems for smaller mobile devices with limited battery capacities. Although one could give these devices faster processors and larger memories, this would diminish their battery life even further.

Without cost effective solutions to the power problem, improvements in micro-processor technology will eventually reach a standstill. Low-power design is a multidisciplinary field that involves many aspects (i.e., energy, temperature, reliability), each of which is complex enough to merit a survey of its own. The focus of this thesis will be on techniques that reduce the total power consumed by the branch target buffer (BTB) in a typical single issue pipelined processor.

Almost all modern processors are highly pipelined today. To reduce the number of stall cycles due to branches, most processor cores perform dynamic branch prediction at the first pipeline stage. However, since the fetched instruction cannot be identified as a branch at this stage, the dynamic branch predictor, which includes a branch direction predictor and a BTB, is always exercised. Moreover, the BTB is a large array structure. Many works have shown that the BTB dissipates a non-trivial amount of power – averaging to 5-10% of the total processor power [1-4]. The power-hungry nature of the above discourages the portable devices from using the dynamic branch prediction. Nevertheless, dynamic branch prediction is still very attractive to processors for power-miser applications owing to its success in performance improvement. Therefore, low-power issue for BTB becomes a significant research topic.

$$E_{BTB} = E_{BTB\_acs} * Counts_{BTB\_acs} + P_{BTB\_leak} * Time_{EXE} \qquad \text{(EQ1)}$$

EQ1 shows the BTB energy equation, where the total BTB energy (E_{BTB}) is partitioned into total BTB dynamic energy and total BTB leakage energy. While total BTB dynamic energy is equal to the products of the average BTB access energy per access (E_{BTB_acs}) and total BTB access counts ($Counts_{BTB\_acs}$), total BTB leakage energy the product of the BTB leakage power (P_{BTB_leak}) and the total execution time ($Time_{EXE}$). Previous works show that an accurate BTB needs large storage and high associativity. However, it has negative effects on E_{BTB_acs} and P_{BTB_leak}. Therefore, reducing the power related terms ($Counts_{BTB\_acs}$, E_{BTB_acs}, and P_{BTB_leak}) with tolerable accuracy degradation is a challenge in this thesis.

Table1-1: Goals and research directions in this thesis

Goals	Research directions
Reducing $Counts_{BTB\_acs}$	a. Reducing the number of unnecessary BTB accesses
Reducing E_{BTB_acs}	b. Buffering the frequent used BTB entries
Reducing P_{BTB_leak}	c. Power management
Reducing P_{BTB_leak} and E_{BTB_acs}	d. Reducing BTB entry size e. Reducing the number of BTB entries

Table 1-1 lists the goals and the corresponding research directions in this thesis. First, reducing the number of unnecessary BTB accesses is the way to reduce $Counts_{BTB\_acs}$. Next, buffering the frequent used BTB entries can reduce E_{BTB_acs}, since most BTB accesses can be performed through accessing the buffering storage instead of accessing BTB. Next, power management is a well known technique to reduce the leakage power in cache [5-8]. We will introduce how to design a power manager for BTB in this thesis. Finally, reducing BTB entry size and the number of BTB entries can reduce both P_{BTB_leak} and E_{BTB_acs} simultaneously.

This thesis is organized as follows. Chapter 2 introduces the background. Chapter 3 to Chapter 7 draws the 5 research topics in this paper. Chap 8 describes the integration of the proposed methods for each BTB power sources. The last chapter gives the conclusions.

Chap 2 Background

2.1 Branch Prediction

The control dependency has great impact on processor performance due to pipelined instruction execution. Branch prediction and target address caching can be a great help here. While static branch prediction uses the branch offset to predict branch direction, dynamic branch prediction uses the runtime behavior such as branch history to do so. Since dynamic branch prediction performs much better than the static counterpart, we limit our following discussion on dynamic branch prediction only.

Figure 2-1 shows a typical dynamic branch predictor containing a direction predictor and a BTB. Various implementations of the direction predictor use different ways to record the branch status and use it to predict the branch direction [9-11]. Furthermore, hybrid implementations integrate several sub-predictors to improve prediction accuracy and are widely used in general-purpose processors for desktops or workstations [12]. BTB, which is used to record target addresses, is a cache in nature. If BTB hits and the branch direction is predicted taken, the branch target address is used as the next PC. Otherwise, the next sequential instruction address (PC+4) is used. A dynamic branch predictor in an embedded processor usually integrates the direction predictor and the BTB. Two branch history bits in each entry of BTB represent possible prediction states (see [13] for more details).

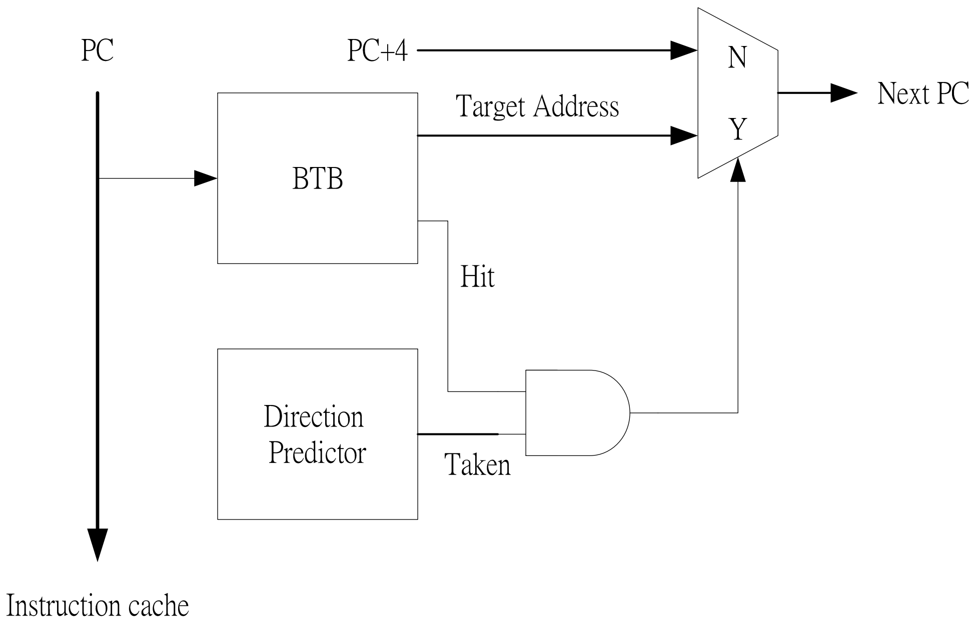


Figure 2-1: Dynamic Branch Predictor

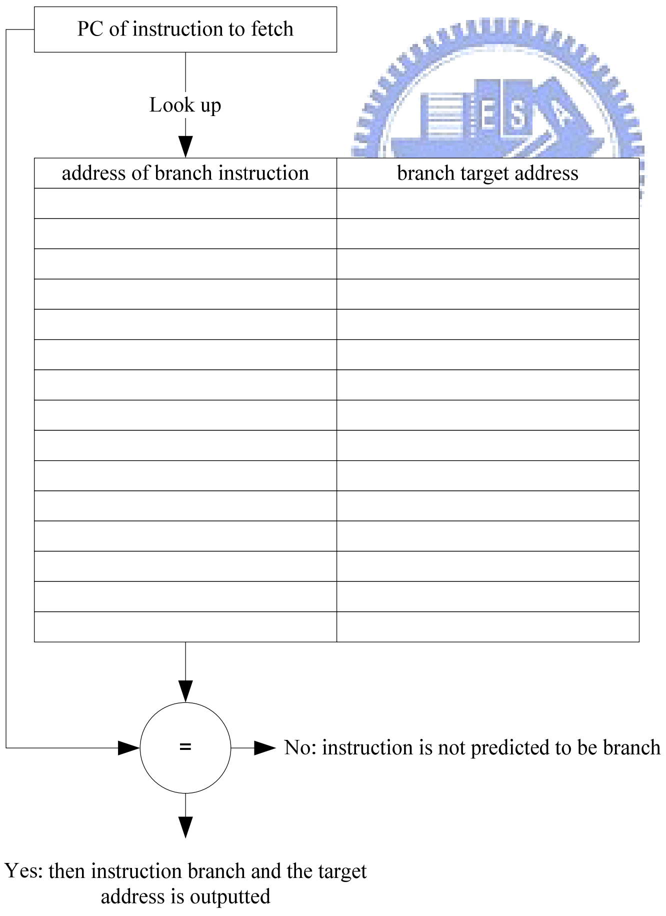

PC of instruction to fetch
Look up
address of branch instruction
branch target address
=
No: instruction is not predicted to be branch
Yes: then instruction branch and the target address is outputted

Figure 2-2: BTB

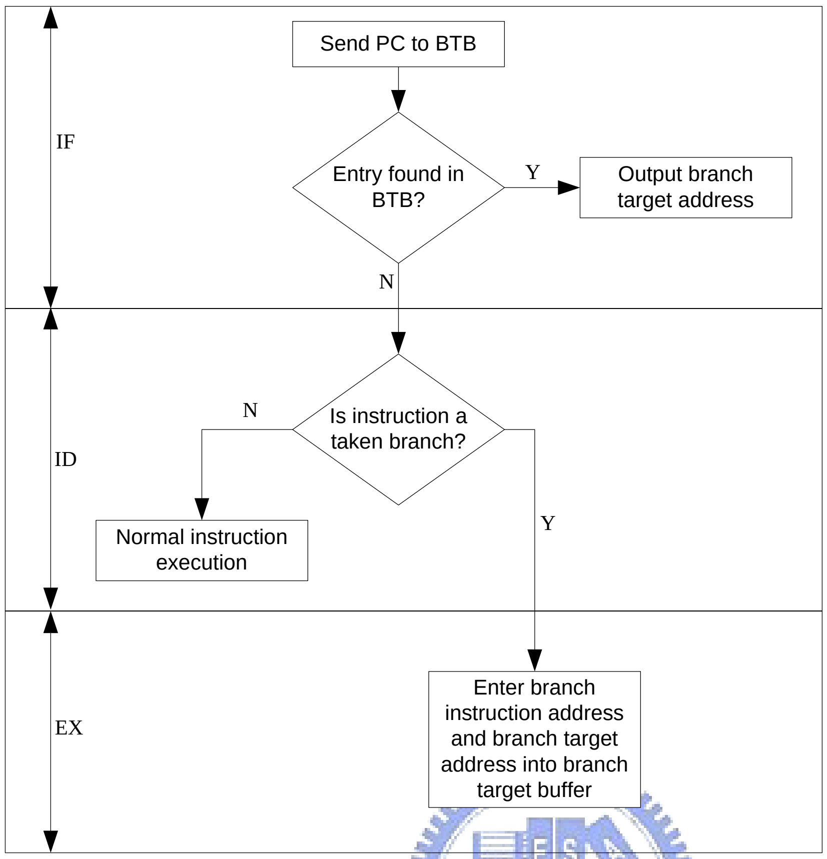

Send PC to BTB
IF
Entry found in BTB?
Y
Output branch target address
N
N
Is instruction a taken branch?
ID
Y
Normal instruction execution
EX
Enter branch instruction address and branch target address into branch target buffer

Figure 2-3: BTB management

2.2 BTB Fundamentals

Figure 2-2 shows a BTB. Because it provides the branch target address of the current fetched branch instruction, we must know whether the fetched instruction is a branch. If the PC of the current fetched instruction matches a PC in BTB, then the corresponding branch target address is outputted. The hardware for BTB is essentially identical to the hardware for a cache. Therefore, BTB can be direct-mapped or set-associative form for timing or hardware cost considerations. Figure 2-3 shows the BTB management for a typical pipeline front-end. From this we can see that BTB only stores the branch target addresses of taken branches.

Table 2-1: Simulation parameters

Processor Core	
Instruction window Issue width Functional units	64 RUU, 32 LSQ 1 instructions/cycle 1 IntALU, 1 IntMult, 1 FpAlu, 1 FpMult, and 1 Memory port
Memory Hierarchy	
L1 I-cache L1 D-cache L2 cache Memory	16KB, 4-way, LRU, 32B blocks, 1cycle, wb 16KB, 4-way, LRU, 32B blocks, 1cycle, wb 512KB, 4-way LRU, 32B blocks, 12-cycle, wb 100 cycles
Branch Prediction	
Direction Predictor Misprediction Penalty BTB	gshare 16K 6 cycles 256-entry, 4-way, LRU

2.3 Simulation Environment

This section introduces the simulation environment used in this thesis. Section 2.3.1 describes the simulator and benchmarks. Section 2.3.2 gives the power models for the baseline processor and BTB.

2.3.1 Simulator and Benchmarks

Simplescalar v3.0d [14], an execution-driven, cycle-accurate simulator for modern processor cores, is used as the processor and BTB simulator. Cacti 4.2 [15] is utilized for power and energy estimation of BTB, since Cacti provides complete power libraries for BTB modules. Table 2-1 summarizes the detailed configurations of our simulations.

The benchmark programs, comprising 12 programs from SPECINT2000 suite, are evaluated. All these programs are compiled using the Compaq Alpha compiler with the SPEC peak settings. Most of them could be executed within Simplescalar. The failed test programs, applu, are removed from the simulation. The provided reference inputs are used. All benchmarks are fast-forwarded past the first half-billion instructions, and a full simulation is then performed for another half-billion instructions.

2.3.2 Power and Energy Models

The power models are constructed by the following equations:

- EQ2 models total chip energy (E_{cpu}). The BTB, the hardware of low-power design, and all other components in the chip are modeled separately, denoted as E_{BTB}, E_{LP}, and $E_{cpu\text{-}BTB}$, respectively.
- EQ3 models $E_{cpu\text{-}BTB}$. The dynamic part ($E_{cpu\text{-}BTB\_dyn}$) and leakage part ($E_{cpu\text{-}BTB\_lk}$) are modeled in EQ4 and EQ5, respectively. While $E_{cpu\text{-}BTB\_dyn}$ is equal to the products of the number of the total executed instructions ($Inst_{exe}$) and the average dynamic energy consumed by these components per instruction ($E_{cpu\text{-}BTB\_dyn/inst}$), $E_{cpu\text{-}BTB\_lk}$ is equal to the products of average leakage energy in them per cycle ($E_{cpu\text{-}BTB\_lk/cyl}$) and the total number of execution cycles ($Cycles_{exe}$).
- EQ6 models the energy impacts of the applied low-power designs; including E_{BTB}, E_{LP} and the energy of chip except for BTB (E_{stall}) consumed during the extra stall cycles ($Cyls_{extra}$) induced by these designs.
- EQ7 models E_{stall}. To simplify this modeling, we assume all processor components are exercised during the extra stall cycles.

$$E_{cpu} = E_{cpu\text{-}BTB} + E_{BTB} + E_{LP} \qquad \text{(EQ2)}$$

$$E_{\text{cpu-BTB}} = E_{\text{cpu-BTB\_dyn}} + E_{\text{cpu-BTB\_lk}} \qquad \text{(EQ3)}$$

$$E_{\text{cpu-BTB\_dyn}} = E_{\text{cpu-BTB\_dyn/inst}} * \text{Inst}_{\text{exe}} \qquad \text{(EQ4)}$$

$$E_{\text{cpu-BTB\_lk}} = E_{\text{cpu-BTB\_lk/cyl}} * \text{Cyls}_{\text{exe}} \qquad \text{(EQ5)}$$

$$E_{\text{BTB\&LP}} = E_{\text{BTB}} + E_{\text{LP}} + E_{\text{stall}} \qquad \text{(EQ6)}$$

$$E_{\text{stall}} = \text{Cyls}_{\text{extra}} * (E_{\text{cpu-BTB\_lk/cyl}} + E_{\text{cpu-BTB\_dyn/inst}}) \qquad \text{(EQ7)}$$

Chap 3 Unnecessary BTB lookup filtering

BTB provides target address only upon the current fetched instruction is a branch and the predicted branch direction is taken. Otherwise, BTB lookup is useless and wastes power. Therefore, filtering unnecessary BTB lookups can significantly reduce BTB access energy.

3.1 Related Work

A Compiler-hinted approach, EIB [16], tries to analyze the basic block length of a program. If the basic block length can be revealed to processor when the first instruction of the basic block is fetched, BTB lookups for the following non-branch instructions can be avoided. Moreover, the address of the branch instruction in the end of the basic block can be obtained. With this information, dynamic branch direction prediction can be performed early so as the BTB lookups for the branches that predicted as not taken can also be avoided. EIB use static analysis to obtain program hotspot and the basic block length in the hotspot. Then, load the basic block length information in program hotspots into a special hardware unit called branch identification unit (BIU). With BIU, BTB lookups are performed only for branches that predicted as taken in hot-spot portions of programs.

BIU has some limitations in nature. First, BIU could skip unnecessary BTB lookups in each hot spot of a program, but not the whole program. Secondly, it relies on the compiler to insert extended instructions, inevitably increasing the program size, and the complexity for the instruction decoder. Finally, which part of a program is hot spot is a runtime behavior which varies according to the program input data or user behavior. Changing the program input data or different user behavior may decrease the hot spot identification accuracy, resulting in lowering the power efficiency.

Architecture-level approaches examine dynamic behaviors of programs to avoid unnecessary predictor accesses. Figure 3-1 shows PPD method proposed in [2]. It employs a branch-free table (BFT) to determine whether the current accessed L1 I-cache set (the cache lines with the same indexes) is control-free, so that the lookup in the branch predictor can be avoided. If the full cycle has insufficient time to allow the BFT to be accessed in-series with the BTB (serial scenario), then the power saving to BTB is limited. It implies that the access to the BFT can not complete in time to avoid the full BTB access, so the accesses of the BFT and BTB must be in parallel (parallel scenario). This situation means that only partial BTB is terminated.

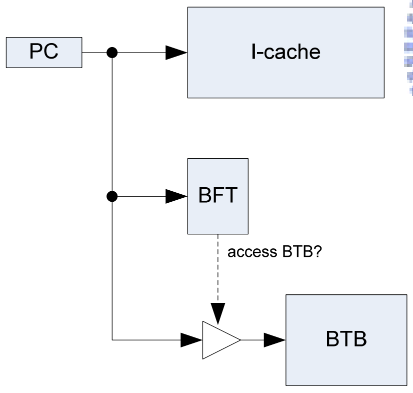


Figure 3-1: PPD method

Lazy BTB [17] dynamically profiles the taken traces to avoid unnecessary BTB accesses. The number of instructions between a taken branch and its subsequent taken branch on execution path are collected into an additional BTB field. This collected information is then used to eliminate the BTB lookups. The simulation results in [17] show that Lazy BTB reduces the BTB access energy consumption by about 77% on average, with 1.7% performance degradations. Unfortunately, the extra chip-wide energy due to the performance degradations (E_{stall}) may be serious, especially in deep pipeline processors. However, no discussions about the extra chip-wide energy are found in [17].

3.2 Design for eliminating unnecessary BTB lookups

This section introduces the design for eliminating unnecessary BTB lookups. We first present the system overview in Section 3.2.1. The function blocks of this design are then described in Section 3.2.2, 3.2.3 and 3.2.4.

3.2.1 System Overview

We intend to develop a method for the elimination of the unnecessary BTB lookups without performance degradation. The distance in terms of the number of non-branch instructions between a branch instruction and its subsequent branch instruction on execution path is defined as next branch distance (NBD). If NBD is revealed to the processor early, then the BTB lookups for these non-branch instructions can be avoided. Therefore, this work focuses on dynamically collecting NBDs and eliminating the unnecessary BTB lookups using these collected NBDs.

Figure 3-2 displays the function blocks of this design integrated in a typical pipeline front-end which is composed of instruction fetch (IF), instruction decode (ID), and execution (EX) stages. An extra storage, NBD table (NBDT), is used to record the NBDs. During EX stage, an NBD counter (NBDC) collects the NBDs of the executed branches into NBDT. During IF stage, BTB, direction predictor (Dir-Pred), and

NBDT are looked up simultaneously, where the BTB and Dir-Pred lookups are for dynamic branch prediction and the NBDT lookups are for NBD probe. If a NBD equal to m is found during NBDT lookup, the next lookup filter filters the following m lookups in BTB, Dir-Pred, and NBDT. The control bit EN is used for the lookup filtering for each fetched instruction. If the lookup to Dir-Pred and BTB is allowed, the predicted next PC is determined by the typical dynamic branch prediction. Otherwise, the static branch prediction (predicted not taken) is used.

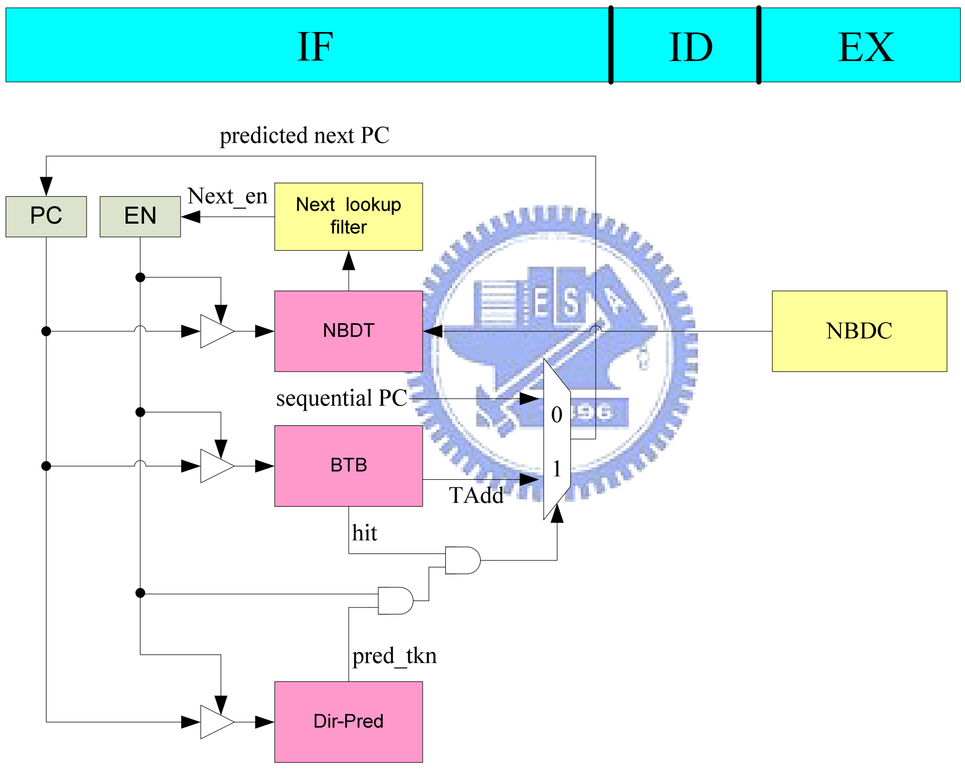


Figure 3-2: The function blocks of this design integrated in a typical pipeline front-end

3.2.2 NBD collection and NBDT management

NBDT records the NBDs of all branch instructions existing in BTB. Each BTB entry has its owned NBDT entry. Each NBDT entry contains two n-bit NBD fields and two single-bit valid fields, where tkn_NBD records the NBD of taken path, nt_NBD NBD of not taken path, tkn_v indicates if tkn_NBD is valid, and nt_v indicates if nt_NBD is valid. Initially, all valid fields are invalid.

Figure 3-3 shows the NBD collection algorithm for a branch instruction and its NBDT management, during EX stage. The detailed descriptions are:

- There is no operation if the branch is not taken and does not exist in BTB, since only the NBDs of the branch instructions existing in BTB are collected.
- If the NBD had been already collected, the repetitive collection operations are useless. Therefore, NBD collection operation is performed only while the corresponding NBD field in NBDT entry is invalid.
- The collected NBDs in NBDT should be consistent with the branches in BTB. In other words, if a BTB entry is allocated to a new branch or an indirect jump updates its target address, the corresponding NBD fields should be invalidated. This invalidation prevents the next lookup filter from receiving the NBD of the replaced BTB entry or the replaced jump target. After the invalidation, the new NBD starts to be collected.

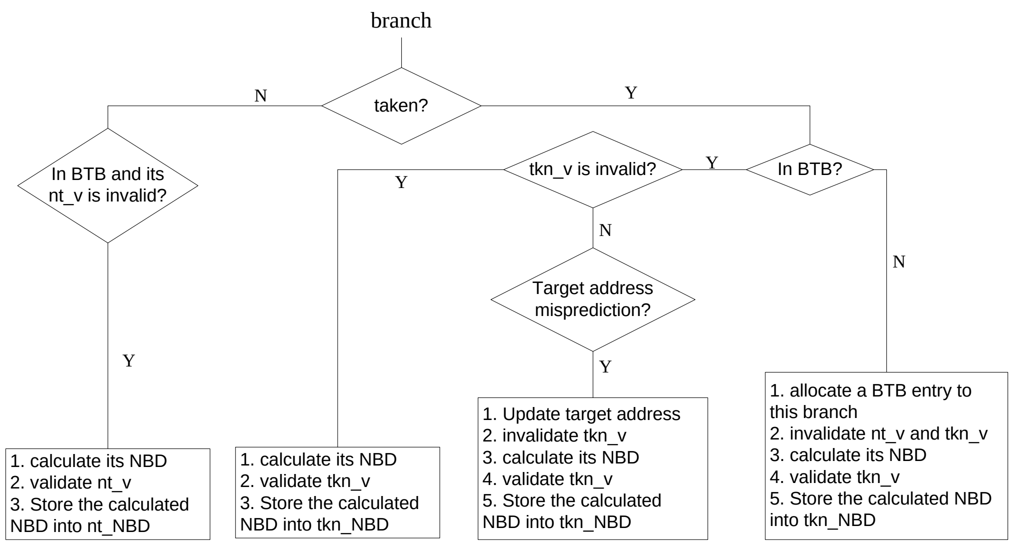

branch
taken?
N
Y
In BTB and its nt_v is invalid?
tkn_v is invalid?
In BTB?
Y
Y
N
N
Target address misprediction?
Y
Y
1. calculate its NBD 2. validate nt_v 3. Store the calculated NBD into nt_NBD
1. calculate its NBD 2. validate tkn_v 3. Store the calculated NBD into tkn_NBD
1. Update target address 2. invalidate tkn_v 3. calculate its NBD 4. validate tkn_v 5. Store the calculated NBD into tkn_NBD
1. allocate a BTB entry to this branch 2. invalidate nt_v and tkn_v 3. calculate its NBD 4. validate tkn_v 5. Store the calculated NBD into tkn_NBD

Figure 3-3: NBD collection and NBDT management

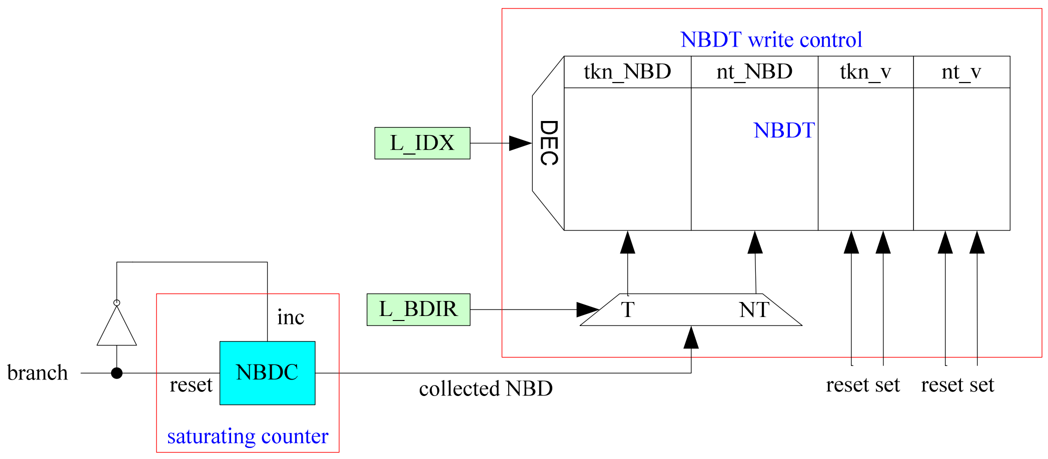


Figure 3-4: The circuits of the NBD calculation and write controls of NBDT

Figure 3-4 shows the circuits of the NBD calculation and write controls of NBDT. The detailed descriptions are:

- The NBD calculation is performed using an n-bit saturating counter (NBDC). NBDC resets itself after a branch instruction (B1) is executed, and then, starts to increment itself for each following non-branch instruction. While the subsequent branch (B2) is executed, the value of NBDC is equal to that of B1's NBD, and then, this value is stored into NBDT if the NBD collection of B1 is allowed. Therefore, a last BTB index register (L_IDX) and a last branch direction register (L_BDIR) are added to preserve the BTB index and branch direction of the branch instruction until the subsequent branch is executed.
- The reset and set signals connected to valid fields of NBDT are used for valid field invalidation and validation, respectively.

Table 3-1 shows an example of the NBD collection process using a trivial loop code. During the first iteration, BNE is placed in BTB. During the second iteration, the NBD of BNE on taken path (2) is calculated and stored into NBDT.

Table 3-1: NBD collection process

Executed Instructions	NBDC	tkn_NBD of BNE	tkn_v of BNE
L: ADD R0, R0, #1	1	X	0
CMP r0, #10	2	X	0
BNE L	0	2	1

3.2.3 Filtering the unnecessary BTB and direction predictor lookups

Figure 3-4 shows the next lookup filter and its lookup filtering controls. Its output signal (Next_en) determines whether the next lookup is filtered or not. In order to control the lookup filtering for each fetched instruction, Next_en is latched into EN every instruction cycle. In addition, an n-bit enable register (ER) is used to record the number of upcoming non-branch instructions before the subsequent branch instruction, on execution path. Therefore, with correct management of ER, Next_en can be simply generated by checking if ER is equal to zero.

The value of the ER is initialized with a default NBD value of zero. In order to find the NBD of the fetched branch instruction, NBDT and BTB are probed simultaneously. If the BTB lookup result is hit, then the fetched instruction is a branch. Both tkn_NBD and nt_NBD are read while NBDT is probed, yet only the one consistent with the predicted branch direction is selected. If the selected NBD is valid, then it is stored into ER; otherwise, ER is reset to the default NBD value of zero. In the non-branch instruction cycles, ER is decremented until its value reaches zero.

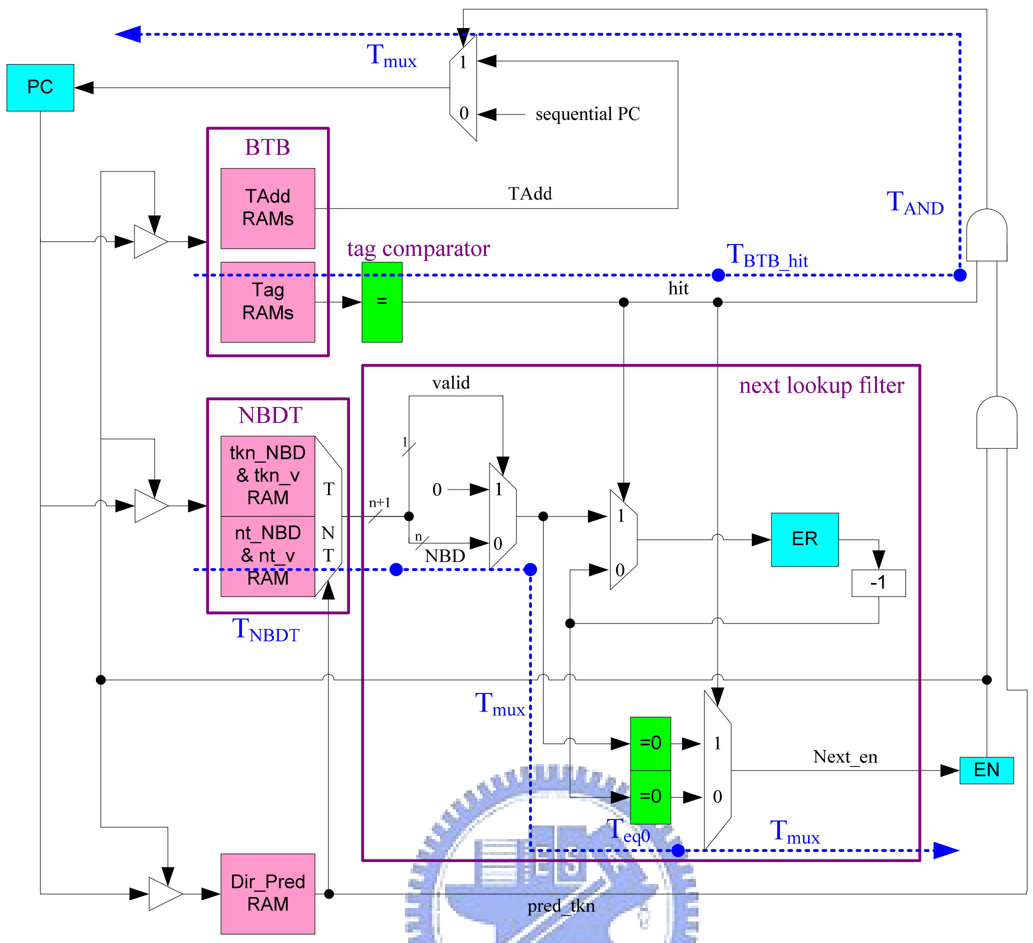


Figure 3-4: Next lookup filter and its corresponding lookup filtering controls

Table 3-2: Lookup filtering process

Fetched Instructions	ER	Next_en	EN
L: ADD R0, R0, #1	1	0	0
CMP r0, #10	0	1	0
BNE L	2	0	1

Table 3-2 shows the lookup filtering process using the same code example in Table 3-1. In this example, the NBD of BNE on taken path has already been collected in NBDT. While BNE is fetched, NBDT is probed. The NBD value of BNE on taken path (2) is selected (assume pred_tkn is 1), and latched into ER. Therefore, if BNE is actually taken, the BTB and DirPred lookups of ADD and CMP are successfully eliminated.

3.2.4 Handling of incorrect BTB and direction predictor accesses

There are two possible impacts on performance and energy if the value of ER is incorrect. An insufficient ER value actuates the BTB and direction predictor too early and wastes energy, whereas an over-estimated ER value paralyzes the BTB and direction predictor while the next branch instruction is being fetched, degrading both performance and energy efficiency. Since the over-estimated ER case has more serious impact on energy and performance, it should be avoided first.

According to the proposed NBD collection and NBDT management, the collected NBDs in NBDT are consistent with the branches in BTB. In other words, if the predicted branch direction and the target address prediction is correct, ER receives a correct NBD value, a default NBD value of zero (the corresponding NBDT field is invalid), or an insufficient NBD value (the width of the corresponding NBDT field is insufficient). If a branch direction or target address misprediction occurs, then the instruction pipeline starts to fetch the new instructions from the correct branch direction or target. In this situation, the value of ER is incorrect and should be reset to avoid the performance loss due to the over-estimated ER.

3.3 Experiments

The objective of these experiments is to evaluate the impacts of the proposed design on energy. The simulator and the benchmark have been described in Section 2.3.1. This section first defines the energy models used for this design. The simulation results are then provided. Finally, the proposed method and existing schemes are compared.

3.3.1 Energy Model

This section gives the energy models. EQ8, EQ9, EQ10, and EQ11 model the BTB energy with all possible impacts of the proposed design, PPD [2], Lazy BTB [17], and EIB [16], respectively. These equations are constructed from remodeling EQ6 into a more detailed version. The detailed descriptions are:

- Since this design focuses on BTB dynamic energy reduction, the leakage part of BTB energy is removed.
- E_{LP}, the energy consumed in the hardware of the proposed design, includes NBDT energy (E_{NBDT}), extra logics and counters energy (E_{others}). E_{LP} for the three related works are model in a similar way.
- Since Lazy BTB use an array structure similar to NBDT, this array structure is denoted as NBDT_lazy here.
- Branch identification unit (BIU), which records the branch distance information of each hot spot, is also an array structure proposed in EIB.

EQ12, EQ13, EQ14, EQ15, and EQ16 model E_{NBDT}, E_{PPD}, E_{NBDT_lazy}, E_{BIU} and E_{others}, respectively. Since they are modeled in a similar way to E_{BTB}, we ignore their descriptions here.

$$E_{BTB\&LP\_our} = E_{BTB\_acs} * Counts_{BTB\_acs} + E_{NBDT} + E_{others} + E_{stalls} \qquad \text{(EQ8)}$$

$$E_{BTB\&LP\_PPD} = E_{BTB\_acs} * Counts_{BTB\_acs} + E_{PPD} + E_{stalls} \qquad \text{(EQ 9)}$$

$$E_{BTB\&LP\_lazy} = E_{BTB\_acs} * Counts_{BTB\_acs} + E_{NBDT\_lazy} + E_{others} + E_{stalls} \qquad \text{(EQ 10)}$$

$$E_{BTB\&LP\_EIB} = E_{BTB\_acs} * Counts_{BTB\_acs} + E_{BIU} + E_{others} + E_{stalls} \qquad \text{(EQ 11)}$$

$$E_{NBDT} = Count_{NBDT\_acs} * E_{NBDT\_acs} + E_{NBDT\_leak} \qquad \text{(EQ 12)}$$

$E_{PPD} = Count_{PPD\_acs} * E_{PPD\_acs} + E_{PPD\_leak}$ (EQ 13)

$E_{NBDT\_lazy} = Count_{NBDT\_lazy\_acs} * E_{NBDT\_lazy\_acs} + E_{NBDT\_lazy\_leak}$ (EQ 14)

$E_{BIU} = Count_{BIU\_acs} * E_{BIU\_acs} + E_{BIU\_leak}$ (EQ 15)

$E_{others} = Count_{others\_acs} * E_{others\_acs} + E_{others\_leak}$ (EQ 16)

3.3.2 Results

3.3.2.1 The width of NBDT

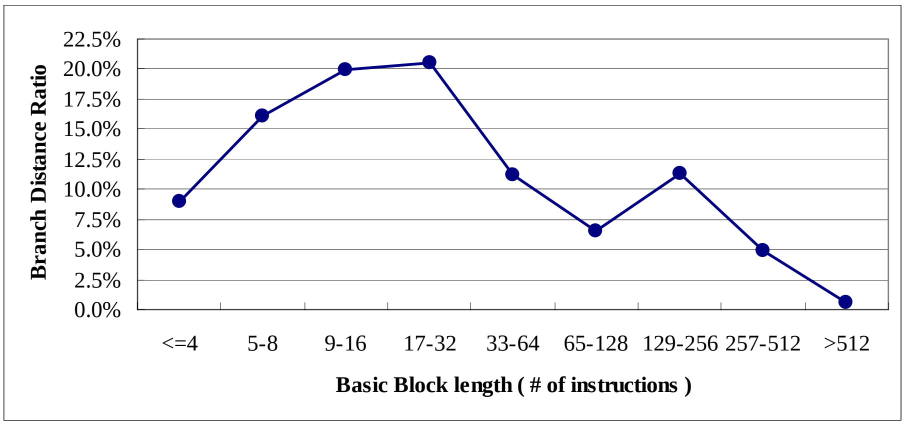


Figure 3-5: Total branch distance ratio for each basic block size interval

The width of tkn_NBD and nt_NBD (*n*) depends on the dynamic behavior of the benchmark programs. Figure 3-5 shows the average total branch distance ratio (R_Total_BD) for each basic block length interval for SPECcpu2000. Typically, most basic blocks have the lengths shorter than 32. For the best tradeoff between the BTB energy reduction and NBDT energy, the value of n is determined to be fixed 5-bit. The detailed simulation results are shown in Section 3.3.2.3.

3.3.2.2 Analysis for the NBD predictions and remained predictor lookups

Table 3-3 lists all reasons for NBD mispredictions and the NBD misprediction ratio for each reason. The S1 mispredictions are unavoidable in any conservative design. The S2 and S3 mispredictions are dependent on the size, structure and replacement policy of BTB. In other words, the S1, S2, and S3 mispredictions are regarded as independent of our design effort. The energy trade-off between NBDT and the reduction for S4 mispredictions is shown in Section 3.3.2.3.

Some unnecessary branch predictor lookups are still remained due to the NBD mispredictions. Figure 3-6 reports the predictor lookup ratios and their breakdown. The average lookup ratios are 11.68% and 26.26% for the branch and non-branch instructions, respectively.

Table 3-3: The reasons for NBD mispredictions

Reasons	Descriptions
S1	Upon first encountering of a branch instruction or a branch has not entered into BTB, there has been no NBD history in NBDT.
S2	After a branch instruction is replaced in BTB, it loses its NBD information.
S3	Upon branch misprediction, the enable register is reset. It forces the branch predictor lookups for the following non-branch instructions.
S4	The width of NBDT is not wide enough to record the full NBD.

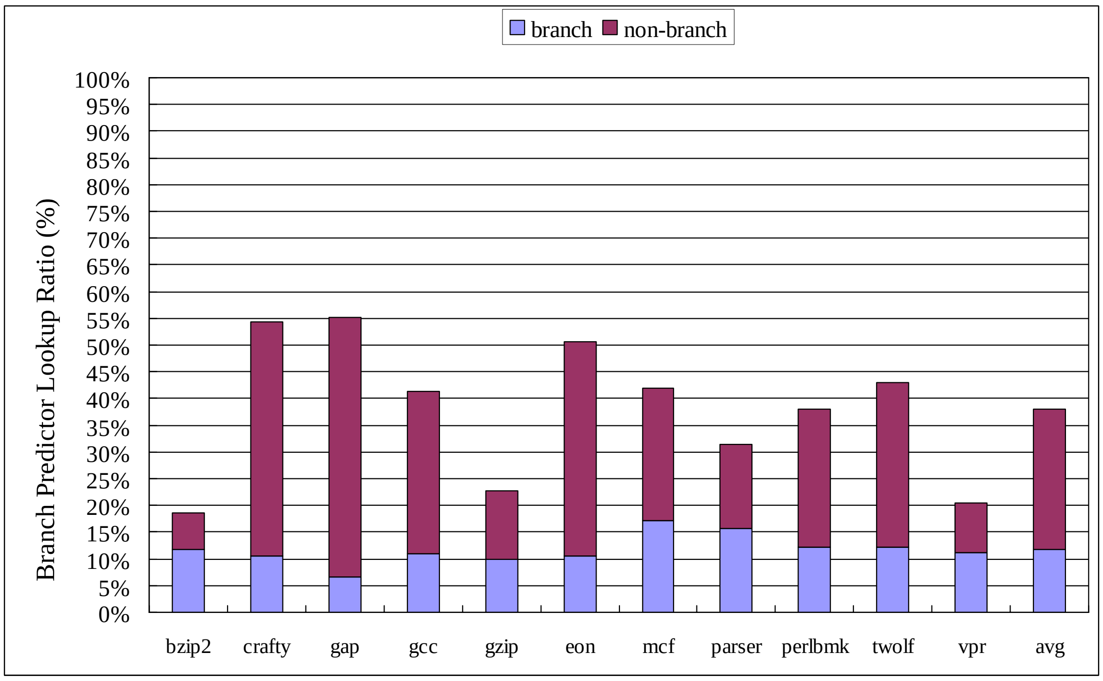
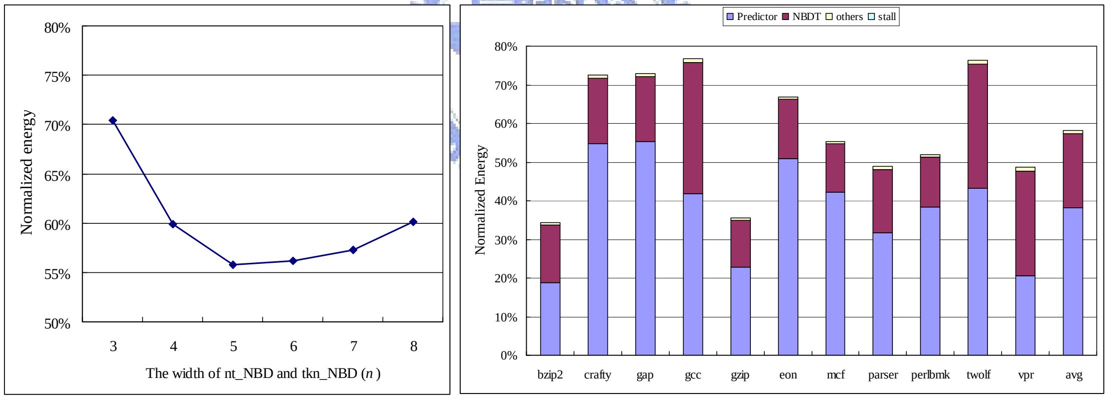


Figure 3-6: Branch predictor lookup ratios



(a) (b)

Figure 3-7: Energy results for the proposed design (a) EBTB&LP_our with different values of n. (b) EBTB&LP_our distributions using n=5

3.3.2.3 Energy Analysis

Figure 3-7(a) illustrates $E_{BTB\&LP\_our}$ with different values of n. All the energy numbers are normalized to the dynamic energy of the baseline BTB without any low-power schemes. According to the evaluation result from CACTI, using one extra bit of tkn_NBD and nt_NBD consumes 2.46% more energy. This extra energy can be well compensated for by the saved energy in the basic blocks with branch distance shorter than 32. Therefore, analytical results show that n=5 yields the largest energy reductions in branch predictor, on average 41.81%. The further partitioning of the $E_{BTB\&LP\_our}$ with n=5 is shown in Figure 3-7(b). The results are:

- $E_{BTB\&LP\_our}$ is 58.19% on average where the BTB dynamic energy occupies 38.23%.
- E_{NBDT} expends 19.23% on average. This is the main energy overhead of the proposed design, since the width of a NBDT entry is 20-bit.
- E_{others} occupies 0.73% on average due to the frequently set/reset the control registers and counters.
- E_{stall} is zero since the proposed design has no performance degradation.

Table 3-4: Results of delay analysis

Critical path of dynamic branch prediction				
T_{BTB_hit}	T_{AND}	T_{mux}		Total
1.20ns	0.06 ns	0.12 ns		1.38 ns
Critical path of lookup filtering circuit				
T_{NBDT}	T_{mux}	T_{eq0}	T_{mux}	Total
0.80 ns	0.12 ns	0.31 ns	0.12 ns	1.35 ns
Delay Evaluations				
T_{BTB_hit}, T_{NBDT} and T_{eq0} are obtained using Cacti 4.2 $T_{mux} = T_{eq0} * 2/5$ $T_{AND} = T_{mux} * 1/2$				

3.3.2.4 Timing Analysis

This section analyzes the timing effect of this design. As shown in Figure 3-4, we suppose that the most time critical paths in the proposed filtering circuit and dynamic branch prediction are [NBDT -> EN] and [BTB tag RAM -> PC], respectively. Table 3-4 lists the delay of each component on the two paths, where the delay time is obtained from Cacti 4.2 and basic gate level analysis. The results show that the proposed filtering circuit fits into the processor pipeline without lengthening the critical paths of branch prediction.

3.3.3 Comparisons of the existing methods

3.3.3.1 High-level comparisons

Compared to EIB, the proposed method is a hardware implementation without any software support. The proposed method can easily be adopted in processor cores without the need to modify program codes, system software, or ISA. The high-level comparisons between the proposed method and EIB are:

EIB filters the BTB lookups only in hot spots, not the entire program, where the hot spots are identified by static profiling. However, which part of a program is hot spot is a runtime behavior which varies according to the program input data or user behavior. Changing the program input data or different user behavior may decrease the hot spot identification accuracy in EIB, resulting in lowering the power efficiency. This drawback does not exist in the proposed method due to its dynamic nature.

The proposed method uses a runtime NBD collection, whereas EIB uses a static one. The static method has the negative effect on program compatibility, compiler complexity, program size, instruction decoder complexity, and the overhead of an extra storage to record the NBDs, whereas the proposed design on the power overhead of NBDT and NBDC only.

The NBD prediction coverage and accuracy of the methods in EIB are mainly depending on the hot spot identification accuracy, whereas that of the proposed method on the branch direction predictor accuracy and BTB hit rate. A higher accuracy branch direction predictor and BTB benefit the proposed method achieving a better power reduction result.

Table 3-5: Comparisons of the low-power branch prediction techniques

Tech.	Low-power schemes	Design targets	Lookup filtering
PPD	Avoid unnecessary BTB lookups	Non-branches	Partial
Lazy BTB	Avoid unnecessary BTB lookups	Non-branches and not-taken branches	Full
Proposed	Avoid unnecessary BTB lookups	Non-branches	Full

Table 3-5 lists the comparisons of the proposed method and other hardware methods. The proposed method has the following features:

- For the low-power schemes, the proposed design filters the lookups to the entire BTB, not only to its partial component.
- The BTB lookup filtering operation (controlled by EN) is independent of the filtering signal generation. If the BTB lookup is not allowed, the full BTB access is stopped, not only the partial one.

3.3.3.2 Simulation Setups

PPD, Lazy BTB, and EIB are included in our simulation, since their goals are closer to those of the proposed method. The simulator includes a PPD table with the number of entries exactly identical to the number of I-cache entries and a NBDT-like table with the number of entries exactly identical to the number of BTB entries. In order to make a fair comparison between the proposed design and Lazy BTB, the width of NBDT_lazy is equal to an optimum value of 3 of the in Lazy BTB simulation. Note that the overestimated NBD problem of Lazy BTB discourages it from using larger NBDT_lazy, since larger NBDT_lazy causes more extra branch mispredictions. The simulation setups for EIB are:

- The most *x*% frequently executed basic blocks of each benchmark program are hot spots. According to the fact that 90% of the execution cycles are spent on 10% of the code, we use *x*=90 as the best case of EIB (EIB_X90). The cases of *x*=80, 70, and 60 (EIB_X80, EIB_X70, and EIB_X60) are also evaluated.
- In each hot spot, the BTB lookups are performed only for the predicted taken branches, whereas the direction predictor lookups are performed after the next branch address is calculated by BIU. In the other part of program, the BTB and direction predictor lookups are performed for each fetched instruction.
- We suppose branch distance is D, the number of instructions fetched per cycle is i, the latency to access the BIU and calculate the next branch address is t cycles, and the latency to access direction predictor is s cycles. According to the EIB design rule, if the time interval D/i - t is shorter than s, a BTB lookup is conditionally performed based on the static branch prediction. According to the processor parameters listed in Table 2-1, the value of i is equal to 1. We assume t=2 and s=1. Therefore, in each hot spot, static branch prediction is used for the branches in the basic blocks with D<3. A profile-based static branch predictor [18] is included for EIB simulation.
- The configuration of BIU is determined according to [16]. We assume this storage capacity of BIU is sufficient to store all the required information of the basic blocks in each hot spot. Moreover, we only consider the energy overhead of BIU lookup and the extra chip-wide energy due to the static branch prediction. The other energy overheads are ignored.

3.3.3.3 Energy comparisons

Figure 3-8 displays $E_{BTB\&LP\_PPD}$, $E_{BTB\&LP\_lazy}$, $E_{BTB\&LP\_EIB}$ and $E_{BTB\&LP\_our}$. The corresponding branch predictor lookup ratios for these schemes are listed in Figure 3-9. The results show that the proposed method outperforms PPD both on branch predictor energy and lookup ratio, since the proposed method targets on all non-branch instructions, whereas PPD on the non-branch instructions in branch-free cache lines only.

Lazy BTB not only skips the lookups that the proposed design does but also the lookups of not-taken branches. However, this aggressive design also skips some necessary lookups, resulting in an extra payment on chip-wide energy. Therefore, the results show that $E_{BTB\&LP\_lazy}$ is less than $E_{BTB\&LP\_our}$.

$E_{BTB\&LP\_EIB}$ and the corresponding branch predictor lookup ratio are mainly dependent on the ratio of the most frequently executed basic blocks (x) being hot spots. Compared to the cases of x<=80, the proposed design has a better energy efficiency for all B_MISP. Even compared to EIB_X90, the proposed design is slightly better than EIB.

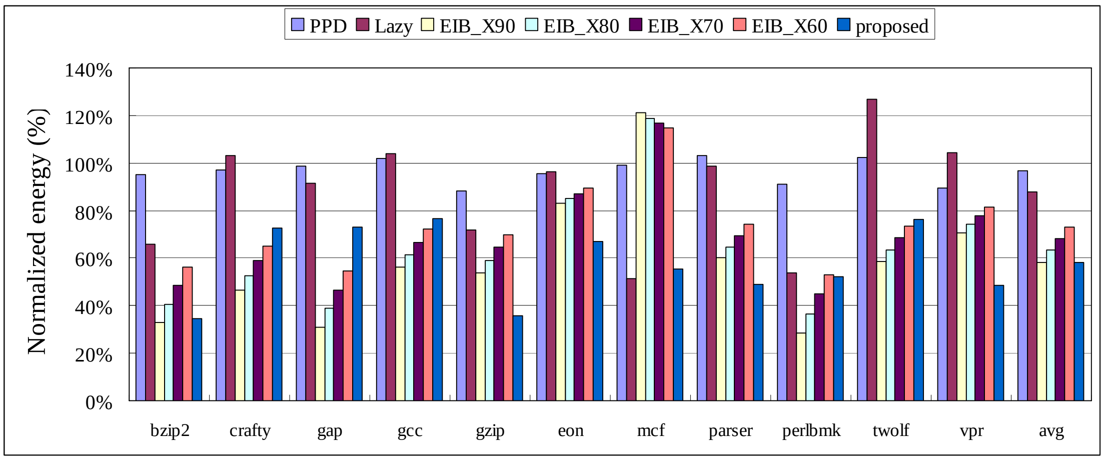


Figure 3-8: Energy comparisons between the proposed design and the related works

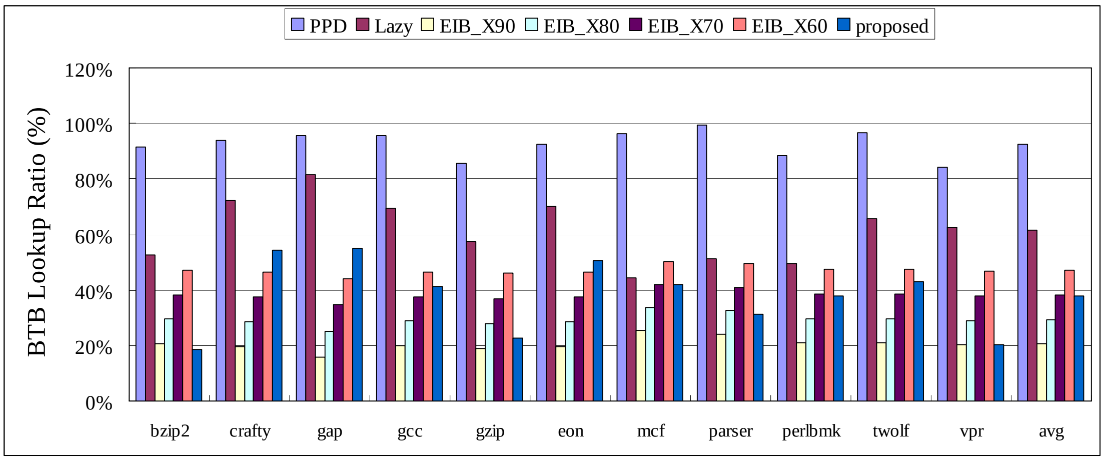


Figure 3-9: BTB lookup ratio comparisons between the proposed design and the related works

3.3.3.4 Performance comparisons

Since both PPD and our method have no performance degradations, we show the relative performance loss ratio only for Lazy BTB and EIB in Figure 3-10. Averagely, the performance loss ratio for Lazy BTB, EIB_X90, EIB_X80, EIB_X70, and EIB_X60 are 0.22%, 0.23%, 0.21%, 0.18% and 0.16%, respectively.

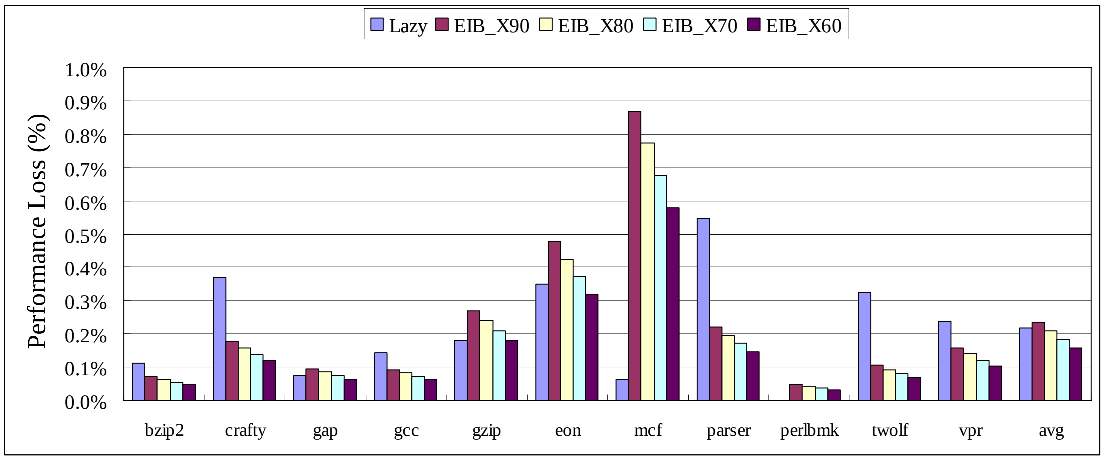


Figure 3-10: Performance loss ratio for the related works

3.4 Summary

This topic addresses the issue of filtering of unnecessary BTB lookups. A dynamic next branch distance generation and collection method is proposed to filter the useless branch predictor lookups. Simulation results show that the proposed design reduces the energy consumption in the BTB by an average of 41.81% without performance degradation. Moreover, the proposed method saves more energy than PPD, Lazy BTB, and EIB does.

Chap 4 Reducing BTB access power through block-address based BTB indexing and entry buffering

This chapter introduces the way to reduce BTB access power. Since the related works are the same as that listed in Chap3, we ignore the related work section here.

4.1 Observations of BTB indexing methods

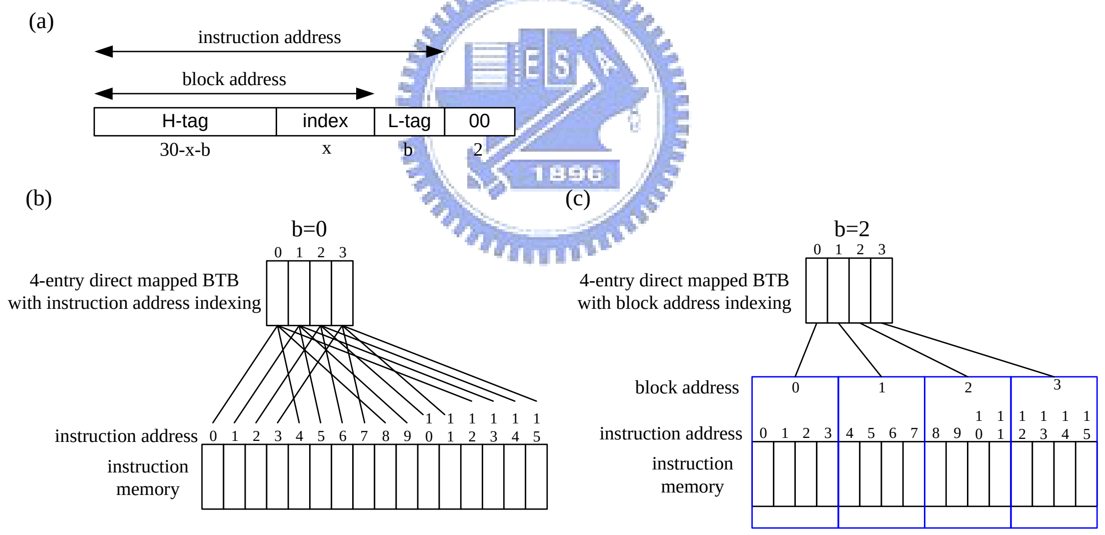


Figure 4-1: BTB indexing method (a) Address partition for a BTB access. (b) Instruction address based indexing. (c) Block address based indexing

Figure 4-1(a) shows the address partition for a BTB access where the address is 32-bit and all instruction words have a length of 4-byte. Each 2^b instructions in the entire address space form a code block. Assuming that there are 2^x BTB entries in the BTB, the general form of the BTB index equation is:

BTB index = (Instruction address >> b) MOD 2^x

In the cases of $b>0$, since the index bits are located in the middle part of the instruction address, the tag bits are split up into the high-order and low-order parts, denoted as H-tag and L-tag, respectively.

Conventional BTB (CBTB), in which case $b=0$, uses instruction address indexing. Figure 4-1(b) shows an example of a 4-entry direct mapped CBTB. It can be observed that at runtime, most incoming instruction addresses are successive; yet the mapped instruction addresses to each BTB entry are broken. Therefore, the accessed BTB entries of two successive BTB lookups are generally different; implying that the probability of a BTB access hitting the most recently accessed entry is low.

If block address indexing, in which case $b>0$, is used, the consecutive BTB accesses are likely to hit the same BTB entry. Figure 4-1(c) shows an example of the case $b=2$, where all other settings are identical to Figure 4-1(b). Since all instructions in a code block are mapped to the same BTB entry, there is a high probability that the accessed BTB entries of two consecutive BTB lookups are the same. There are only two exceptions: One is that the first lookup PC (program counter) is pointed to a block-end and the next PC is sequential; the other is that the next PC is non-sequential and crosses its block boundary. Note that only the index field is changed in block address indexing. The total length of tag is unchanged. Therefore, the branch identification in block address indexing BTB still works unambiguously.

4.2 Design for BTB access power reduction

Locality property has made cache and BTB designs being successful in performance improvement. Unfortunately, BTB and caches are very power inefficient since the fraction of unique BTB entries and cache lines accessed during an interval of thousands of instructions is small. Various research works have added a small cache that stores frequently accessed cache lines to save access power in cache, since a smaller cache has lower access power.

Similar idea can be applied to reduce BTB access power, yet the difficulty lies in the fact that temporal locality in BTB entries is weak due to conventional instruction address indexing. We use block address indexing such that consecutive BTB accesses are likely to hit the same BTB entry. An entry buffering approach that buffers the most recently accessed BTB entry is proposed here to filter the BTB lookups that hit the most recently accessed BTB entry. Figure 4-2(a) shows the low-power BTB lookup algorithm. Note that we show the process between BTB index decoding and BTB entry output only. The processes of tag comparison, ORing all the comparison results, and selecting the hit BTB entry according to the comparison results are not included in Figure 4-2(a). An entry buffer (EB) is added to record the most recently accessed BTB entry in such a manner that the access that hits this entry can be output from EB directly. If a new BTB entry is filled in BTB and the overwritten BTB entry is the most recently accessed one, a problem of inconsistency arises. In this situation, both BTB and EB are filled to avoid inconsistency.

Figure 4-2(b) displays the circuit implementation. Before BTB access, a comparison between the current and last BTB index is performed to generate the index the same signal (ITS). To avoid the comparator delay, ITS can be also generated using content-change-aware register proposed in [19]. If the current BTB index is not changed (ITS=1), the BTB access is gated, and the read enable signal of EB (R_EN) is enabled to directly output the BTB entry from EB. Otherwise, BTB is accessed normally, and the output BTB entry is stored into EB.

We continue to extend the low-power idea to a set-associative BTB. In a set-associative BTB, each BTB way is a direct-mapped BTB. Therefore, each BTB way has an EB to record the most recently accessed BTB entry.

(a) BTB access algorithm

```
BTB_ENTRY BTB[NUM_BTB_ENTRIES];
BTB_ENTRY EB;

BTB_ENTRY BTB_access(BLK_ADDR B)
{
  index = B % NUM_BTB_ENTRIES;
  if (index != last_index)
    {
              EB=BTB[index];
              return BTB[index];
    }
  else
              return EB;
  last_index = index;
}
```

(b) BTB access circuit

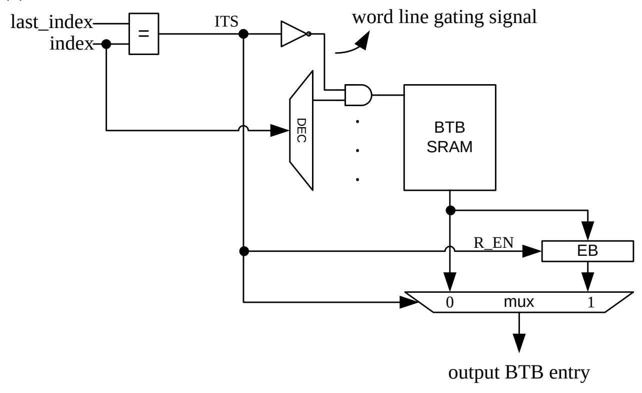


Figure 4-2: Low-power BTB lookup (a) Algorithm. (b) Circuit implementation.

4.3 Experiments

The objective of these experiments is to evaluate the impact of the proposed design on energy and performance. Section 4.3.1 describes the goals of these experiments; Section 4.3.2 introduces the power models; and Section 4.3.3 gives the results and analysis.

4.3.1 Goals

We reduce BTB dynamic energy by proposing entry buffering for BABTB. The goals which we intend to achieve through these experiments are:

- Determining the optimum configurations of CBTB and the BABTB.
- Evaluate the energy and performance of [BABTB with EB].

4.3.2 Energy Model

The power models are constructed by EQ17. It models the energy impacts of the applied low-power designs; including BTB dynamic energy, entry buffer energy (E_{EB}) and the energy of chip except for BTB (E_{stall}) consumed during the extra stall cycles ($Cyls_{extra}$) induced by these designs. Since this design is for dynamic energy reduction, the common part, BTB leakage energy, is not included in $E_{BTB\&LP\_EB}$. Table 4-1 lists the energy parameters estimated using CACTI 4.2.

$$E_{BTB\&LP\_EB} = E_{BTB\_acs} * Counts_{BTB\_acs} + E_{EB} + E_{stall} \quad \text{(EQ17)}$$

Table 4-1: Energy Parameters

Energy parameters	
BTB dyn energy/access	4.32pJ
EB dyn energy/access	0.28pJ
EB leak energy/cycle	0.016pJ

4.3.3 Results and analysis

In Table 4-2, all experiments are listed; BTB configurations are described in the first column and the evaluation metrics in the second. The first experiment observes the effects of code block size and BTB assosiativity on NPC accuracy (the number of correctly predicted next PC for branches by dynamic branch prediction over the number of total executed branches), IPC, and total chip energy. According to the results of total chip energy, the [256-entry, 4-way, 1-instruction block] and [256-entry, 4-way, 8-instruction block] are determined as the configurations for CBTB and the proposed BTB, respectively. The second experiments report the IPC and $E_{BTB\&LP\_EB}$.

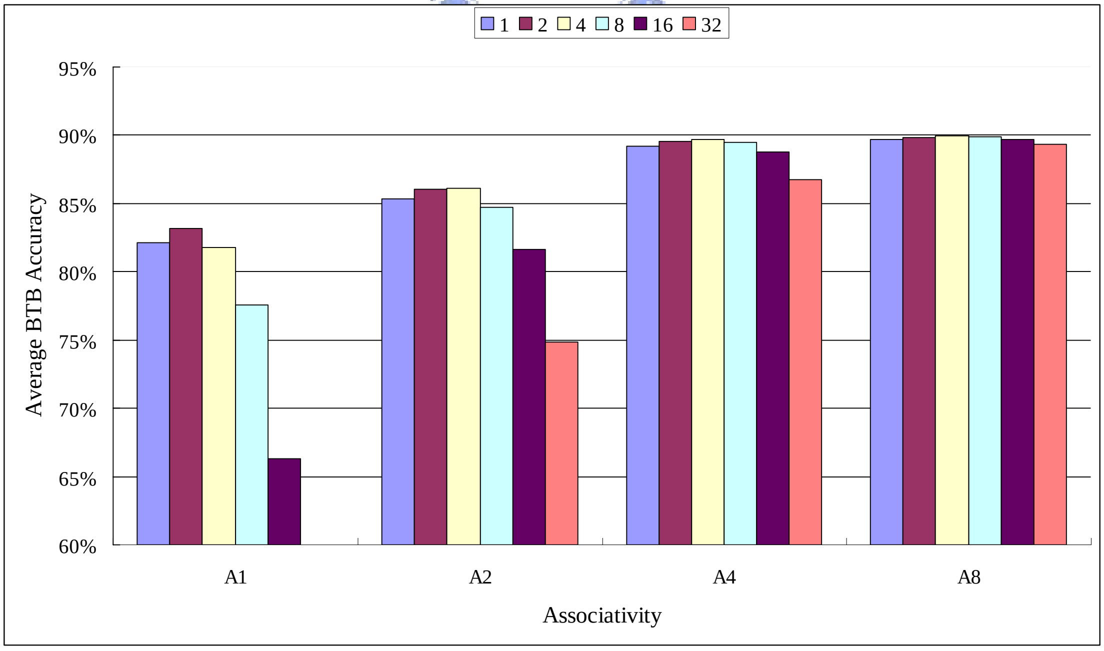
Table 4-2: Experiments

256-entry BTB configuration	Evaluation metrics
CBTB:1~8way without EB Proposed BTB:1~8way, 2~32-inst. blk with EB	NPC accuracy, IPC, and total chip energy
4-way, 8-inst. blk + EB (BABTB_EB)	IPC and $E_{BTB\&LP}$ without E_{BTB_lk} + E_{PM}

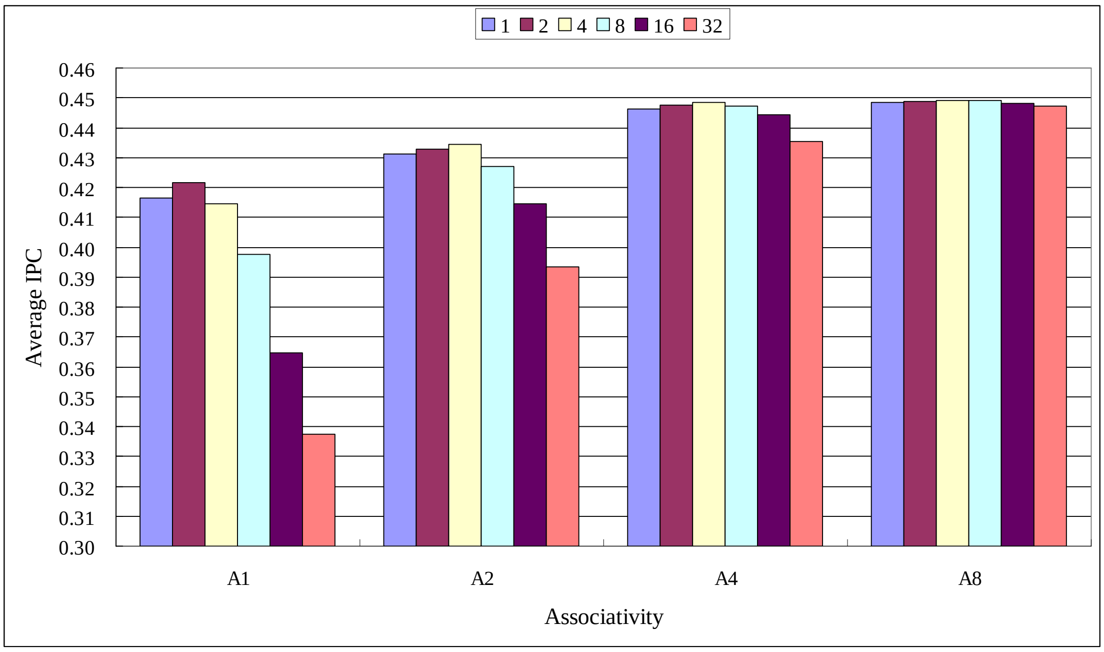
4.3.3.1 Block size and associativity

Figures 4-3(a) and 4-3 (b) show the average NPC accuracy and IPC of the observed BTBs, respectively. The results for each associativity parameter are arranged in the following order: 1-instruction, 2-instruction, 4-instruction, 8-instruction, 16-instruction, and 32-instruction block (denoted as 1, 2, 4, 8, 16, and 32, respectively). The 1-instruction BTBs (CBTB) are configured without EB, whereas the other BTBs (BABTB) are configured with EB. The first observation we made is that in the direct-mapped BTB with block sizes larger than 2, the NPC accuracy and IPC decrease while the block size increases. The cause of this is the fact that branches are not uniform distributed in programs. A contention problem for a single BTB entry may exist if the code blocks mapped to this entry contain more than one branch instruction. The second observation we made is that with higher associativity, the evil effects of this problem on NPC accuracy and IPC are not obvious. In some cases, the set associative BABTB even achieves higher NPC accuracy and higher IPC than the corresponding set associative CBTB.

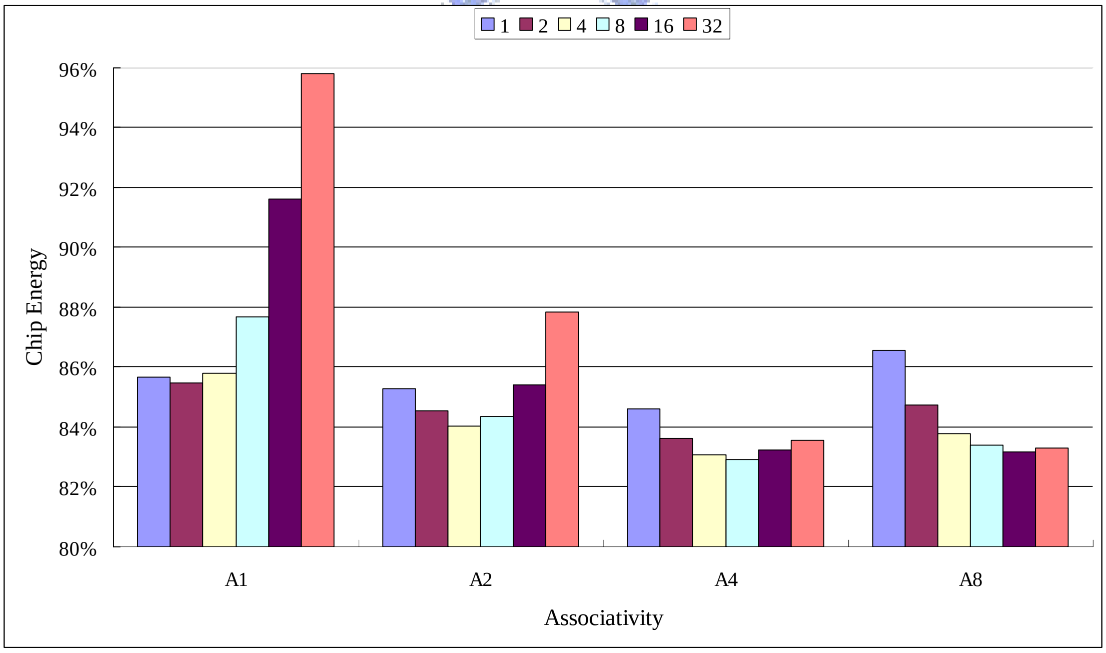
Figure 4-3(c) shows the total chip energy corresponding to the simulated BTBs in Figures 4-3(a) and (b). The energy number is normalized with respect to the total energy consumed by the processor without BTB. It can be observed that a highly accurate BTB reduces the chip energy during the branch stall cycles. The results show that increasing associativity benefits chip energy only when associativity is less than or equal to 4, because increasing associativity also hurts BTB access energy. In addition, although increasing block size always benefits BTB access energy, it benefits chip energy only when IPC is improved. The lowest chip-wide energy configurations among the observed CBTBs and the proposed BTBs are [4-way 1-instruction block] and [4-way 8-instruction block with EB], respectively. On average, the accuracy and IPC of the proposed BTB with the optimum configuration are 0.08% and 0.006% higher than that of CBTB, whereas the chip-wide energy is 1.68% lower. These configurations are used for the simulations in Section 4.3.3.2.



(a)



(b)



(c)

Figure 4-3: Evaluations for the observed BTBs (a) Average NPC accuracy. (b) Average IPC. (c) Total chip energy.

Table 4-3 lists IPC, total BTB dynamic energy, and total BTB leakage energy of the [256-entry 4-way 1-instruction block] and [256-entry 4-way 8-instruction block] BTB for each benchmark program. In the simulations in Section 4.3.3.2, all the energy and performance results are normalized to the corresponding item in Table 4-3.

Table 4-3: IPC, BTB dynamic energy, and BTB leakage energy of BABTB and CBTB

	BABTB			CBTB		
Benchmarks	IPC	BTB dyn (mJ)	BTB leak (mJ)	IPC	BTB dyn (mJ)	BTB leak (mJ)
bzip2	0.458	2.321	2.253	0.458	2.321	2.253
crafty	0.476	2.419	2.170	0.465	2.458	2.224
gap	0.502	2.313	2.057	0.500	2.314	2.065
gcc	0.191	2.453	5.408	0.190	2.474	5.430
gzip	0.599	2.327	1.723	0.599	2.327	1.723
eon	0.510	2.521	2.027	0.518	2.500	1.995
mcf	0.640	2.417	1.615	0.642	2.416	1.610
parser	0.431	2.378	2.399	0.430	2.381	2.402
perlbmk	0.581	2.514	1.777	0.581	2.514	1.777
twolf	0.200	2.479	5.171	0.199	2.482	5.181
vpr	0.233	2.394	4.432	0.233	2.397	4.435

4.3.3.2 Evaluations of BABTB with entry buffering

This section evaluates the energy and IPC of BABTB with EB. BABTB is setup as [256-entry 4-way 8-instruction block], which is determined in Section 4.3.3.1. All energy and performance values shown in this section are normalized to those in the BABTB without any low-power design.

Figure 4-4 shows the energy results for each benchmark program, respectively. We do not show performance result, since this design does not affect performance. [BABTB with EB] reduces the dynamic energy consumption in BABTB by an average of 73.17%, because [BABTB with EB] has the low BTB access ratio, low hardware overhead, and no performance degradation.

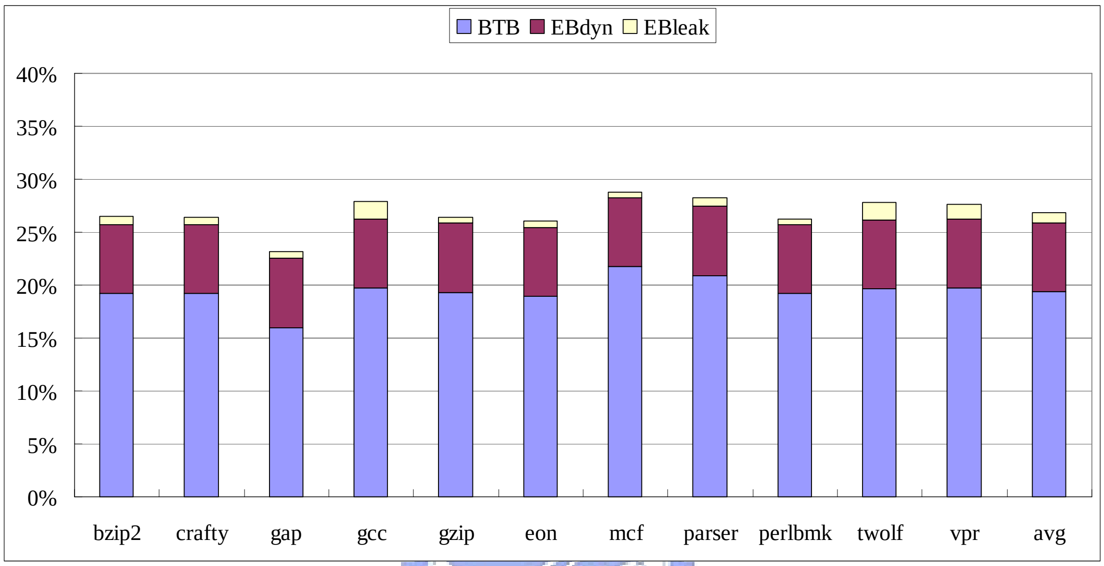


Figure 4-4: Energy result for BABTB with EB

4.4 Summary

This topic addresses the issue of BTB access power reductions. Through block address indexing and entry buffering, the dynamic power consumption of BTB access is reduced by an average of 73.17% without performance degradation. It outperforms the method proposed in Chap 3. These two methods target on different power sources, therefore, with the integration of these two methods, the dynamic power consumption in BTB can be further reduced. The integration issue will be discussed in Chap 8.

Chap 5 Reducing BTB leakage power through power management

Reducing leakage power becomes a popular research issue, since leakage power may dominate the total power consumption as processor technology moves below 0.1μm. This chapter introduces the way to reduce BTB leakage power.

5.1 Related works

5.1.1 BTB power modes

Table 5-1: BTB power modes

Power Mode	Voltage	State	Can be accessed?
Active	1V	preserved	Y
Drowsy[21]	0.3V	preserved	N
Sleep[20]	0V	destroyed	N

Two circuit techniques have been proposed to reduce the leakage power in SRAM. The first one is drowsy cache [21], which lowers the supply voltage of SRAM until the data is just preserved. Another one is gated V_{dd} [20], which gates the supply voltage of SRAM. Therefore, the data is destroyed. We use these two techniques to reduce BTB leakage power. Table 5-1 lists three BTB power modes. Active mode uses the normal supply voltage, drowsy mode uses the state-preserved supply voltage proposed in [21], and sleep mode gates the supply voltage.

5.1.2 Power management for BTB

Power management is a well known approach for reducing leakage power. A compiler-directed manager [22] manages BTB power modes in loops. During loop execution, this manager puts the BTB entries outside the loop into state preserving mode. All BTB entries are returned to normal mode after exiting the loop. Decay method [23] gates the supply voltage of the BTB entry predicted to be no longer used. However, in the time period from it entering in BTB to being used last time, it still consumes significant power.

5.2 Design for BTB power management

5.2.1 Decay-based power manager

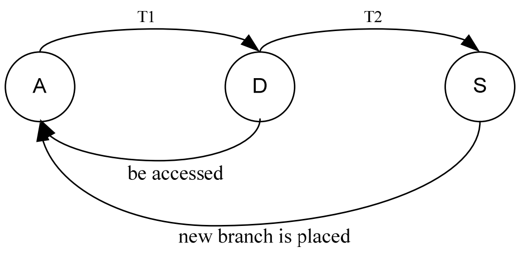


Figure 5-1: Decay-based power managers

Figure 5-1 shows the decay-based power manager for each BTB entry using a hybrid of state-destroying and state-preserving modes [6]. Each BTB entry (both tag and data fields are included) has three power modes which are implemented by combining the circuit techniques of gated-V_{dd} [20] and drowsy circuit [21]. In active mode (A), the entry is operated normally. In drowsy mode (D), the supply voltage of the BTB entry is lowered. The entry data in drowsy mode is preserved and can be reactivated with one cycle latency if necessary. In sleep mode (S), the supply voltage of the entry is gated, and the entry data is lost. While an active entry with an idle period of T1 is put into drowsy mode, a drowsy entry with an idle period of T2 is put into sleep mode. A sleep entry is returned to active mode if a new branch is placed in. The drowsy entry is returned to active mode with one cycle latency once an access requirement is coming.

The decay-based power manager is usable for both CBTB and [BABTB with EB]. The experiment data in Section 5.3.3 shows that [BABTB with EB] with the decay-based power manager has better power efficiency than CBTB with the decay-based power manager. Therefore, we intend to further reduce BTB leakage power of [BABTB with EB] with the decay-based power manager.

5.2.2 Entry pre-activation

We propose an entry pre-activation method, which wakes up a drowsy BTB entry if it is to be accessed during next cycle, for [BABTB with EB] with the decay-based power manager. In [BABTB with EB], it is accessed only if the current BTB index differs from the previous one. Therefore, if the BTB index of the next cycle ($i+1$) is to be changed, the entry pre-activation method should predict the index value and the predicted value should be available at the beginning of the current cycle (i).

Typically, there are five possible next PC sources: sequential PC, branch target from BTB, branch target from ALU, indirect jump address from register, and exception vector address. Since the first two cases account for the vast majority of PCs, the entry pre-activation method only handles these two cases:

- Sequential pre-activation: If the current PC is pointed to a block end and the next PC is sequential, the next BTB entry is the sequential entry. As a result, we pre-activate the sequential entry if the current PC is pointed to a block end. We do not wait for the dynamic branch prediction due to timing considerations. Note that the sequential pre-activation is different from the one proposed in [5], which is used for I-cache. In [5], when the current cache line is accessed, the sequential cache line is pre-activated.
- Target pre-activation: If the next PC comes from BTB and crosses the current block boundary, the next BTB entry is the target entry. In CBTB, it is difficult to predict the index of the target entry one cycle before this entry is to be accessed. Fortunately, this is not a problem in the proposed BTB because consecutive BTB accesses are likely to hit the same BTB entry. Therefore, we propose a target pre-activator for the target entry pre-activation. The target pre-activator always checks whether the tag of the sequential PC and the tag of the current accessed BTB entry are identical. If they are identical, the sequential instruction is a branch and its mapped BTB entry is the currently accessed one. In other words, in the current cycle (*i*), if the current PC is sequential and the checking result during the previous cycle (*i-1*) was identical, the current fetched instruction is a branch and its mapped BTB entry is the same as the one accessed during the previous cycle (*i-1*). In this situation, the target pre-activator pre-activates the BTB entry to which the checked BTB entry points. Note that it is not necessary to add an extra storage for preserving the checked BTB entry, since it is certain to be in EB in this situation.

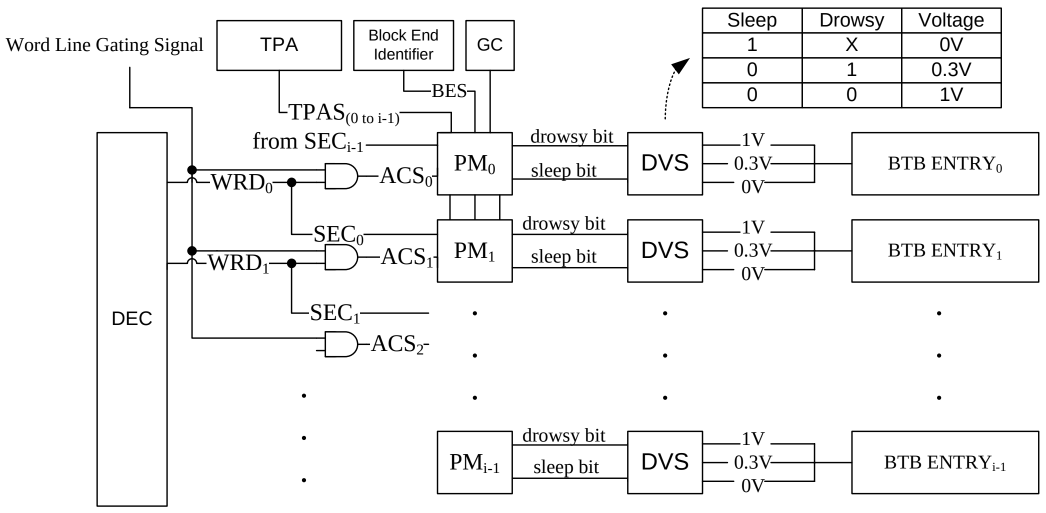
5.2.3 Implementation

Figure 5-2(a) shows a block diagram of FDPM with the entry pre-activation method for the proposed BTB. Each BTB entry is accessed (ACS=1) only when its word line is enabled (WRD=1) and is not gated, where the word line gating signal comes from the dynamic power reduction method shown in Figure 4-2(b). We use SEC to indicate if this entry is the sequential entry. Each BTB entry has two power mode bits denoted as drowsy bit and sleep bit. The dynamic voltage scaling (DVS) circuit supplies three types of voltages: 1V, 0.3V, and 0V; which correspond to active, drowsy, and sleep modes, respectively. The power mode control of each BTB entry determines its power mode bits according to ACS, SEC, the target pre-activator, a global counter, and a block end identifier; where the target pre-activator generates the target pre-activation signal (TPAS) to pre-activate the target entry and the block end identifier generates the block end signal (BES) which indicates if the current PC points to a block end.

Figure 5-2(b) presents the implementations of power mode control. We extend the hierarchical counter scheme in [8] to implement idle period identification in FDPM. A global counter and two 2-bit local counters are used, where the drowsy local counter (DLC) is used to identify the idle period T1 and the sleep local counter SLC T2. All other implementation details for idle period identification are identical to those in [8]. The entry pre-activation signal (PA) is enabled when the sequential pre-activation signal (SPA) or the target pre-activation signal (TPAS) are enabled, where the SPA is enabled if the PC points to a block end (BES=1) and the entry is sequential entry (SEC=1). If a drowsy BTB entry is pre-activated (PA=1) or accessed

(ACS=1), it is switched to active mode and its drowsy local counter is set to a maximum value of 3.

(a)

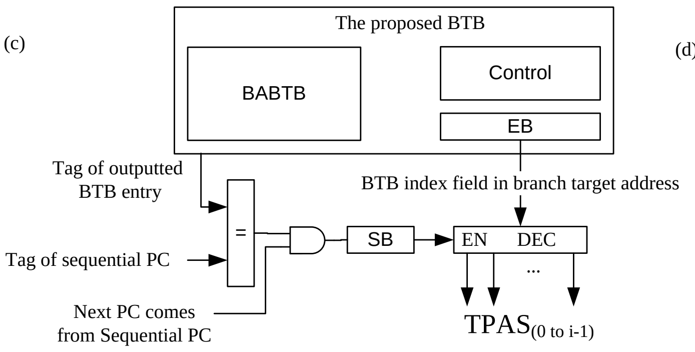


Sleep	Drowsy	Voltage
1	X	0V
0	1	0.3V
0	0	1V

(b)

```
Power_Mode_Control ()
{
  ST1=$2^i$;
  ST2=$2^j$;
  Switch(PowerMode)
  {
     Active:
        if (ACS) DLC=3;
        else
        {
           if ( (GC MOD ST1 == 0) && (DLC>0) )   DLC--;
           if (DLC == 0)  { PowerMode=Drowsy; SLC=3; }
        }
     Drowsy:
        SPA = SEC & BES;
        PA = SPA || TPAS;
        if (ACS || PA) { PowerMode=Active; DLC=3; }
        else
        {
           if ( (GC MOD ST2 == 0) && (SLC>0) )   SLC--;
           if (SLC == 0) PowerMode=Sleep;
        }
     Sleep:
        if (A new branch is placed in)  { PowerMode=Active; DLC=3; }
  }
}
```

(c)


The proposed BTB
BABTB
Control
EB
Tag of outputted BTB entry
BTB index field in branch target address
Tag of sequential PC
=
SB
EN DEC
...
Next PC comes from Sequential PC
TPAS(0 to i-1)

(d)

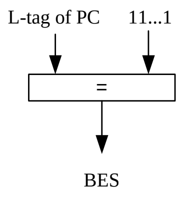


Figure 5-2: The proposed power management. (a) Block diagram. (b) PMC implementation. (c) TPA implementation. (d) Block end identifier implementation.

Figures 5-2(c) and 5-2(d) show the implementations of the target pre-activator and the block end identifier, respectively. The target pre-activator uses a comparator to compare the tags of the sequential PC and the outputted BTB entry. If the result of comparison is equal and the next PC is sequential, then the next fetched instruction is a branch and its target is recorded in the outputted BTB entry. We set a one-bit register SB to indicate this situation. The target pre-activation signal (TPAS) is generated by decoding the BTB index field of branch target address in EB if SB=1. Block end identifier uses a comparator to check if the bits of L-tag in PC are all equal to one.

5.3 Experiments

The objective of these experiments is to evaluate the impact of the proposed design on energy and performance. Section 5.3.1 describes the goals of these experiments; Section 5.3.2 introduces the energy model; and Section 5.3.3 gives the results and analysis.

5.3.1 Goals

We reduce the leakage energy in the [BABTB with EB] by the decay-based power manager and entry pre-activation. The goal which we intend to achieve through these experiments is that comparing energy and performance of [CBTB with the decay-based power manager], [BABTB with EB and the decay-based power manager], and [BABTB with EB, the decay-based power manager, and entry pre-activation].

5.3.2 Energy Model

The power models are constructed by EQ18. It models the energy impacts of the applied low-power designs; including BTB leakage energy, the energy of power manager (E_{PM}) and the energy of chip except for BTB (E_{stall}) consumed during the extra stall cycles ($Cyls_{extra}$) induced by these designs. Since this design is for leakage energy reduction, the common part, BTB dynamic energy, is not included in $E_{BTB\&LP\_PM}$.

$$E_{BTB\&LP\_PM} = P_{BTB\_leak} * Time_{EXE} + E_{PM} + E_{stall} \qquad \text{(EQ18)}$$

Table 5-2: Energy Parameters

Energy parameters	
BTB active mode leak energy/cycle	2.07pJ
BTB drowsy mode leak energy/cycle	0.33pJ
BTB sleep mode leak energy/cycle	0pJ
Power mode transition energy	0.432pJ
Counter dyn energy/access	0.06pJ
Counter leak energy/cycle	0.047pJ

Table 5-2 lists the energy parameters estimated using CACTI 4.2 web interface [15]. According to the power characteristics reported in [20] and [21], we assume that for each BTB entry, the leakage power consumed in the drowsy mode and in the sleep mode is 16% and 0% of that in the active mode, respectively. The mode transition energy spent in switching from low-power mode to active mode is 10% of the single-access energy in BTB (E_{BTB_acs}).

5.3.3 Results and analysis

Table 5-3: Experiments

256-entry 4-way BTB configuration	Evaluation metrics
1-inst. blk + decay based power manager (CBTB_PM)	IPC and $E_{BTB\&LP\_PM}$
8-inst. blk + EB + decay based power manager (BABTB_EB_PM)	
8-inst. blk + EB + decay based power manager + entry pre-activation (BABTB_EB_PM_EPA)	

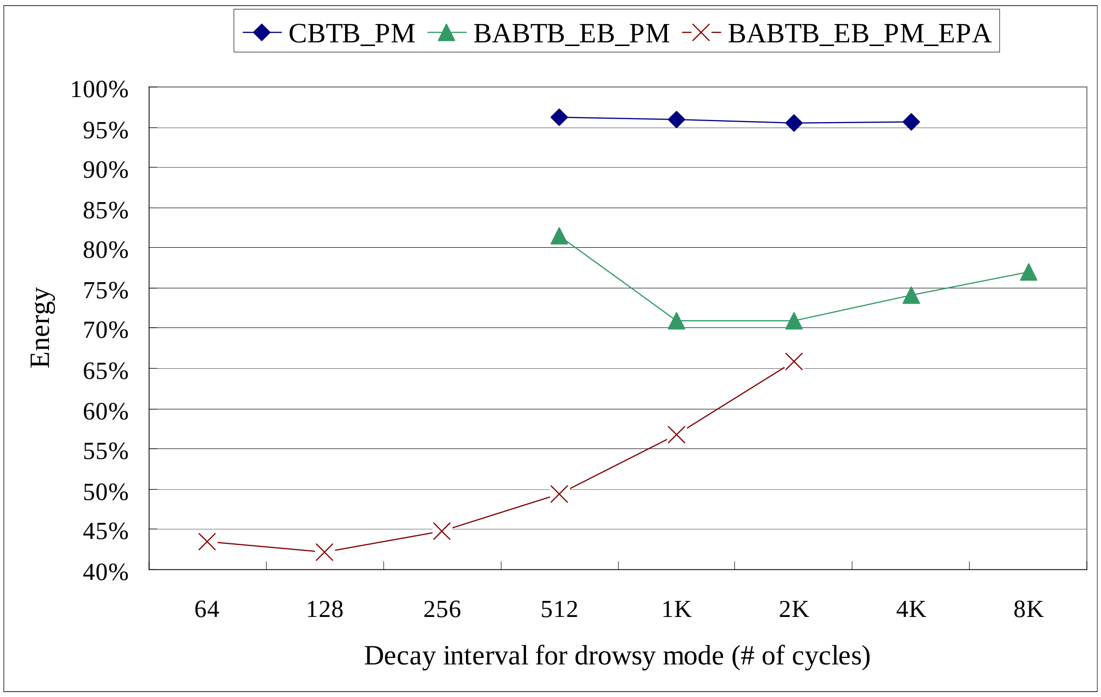
In Table 5-3, all experiments are listed; BTB configurations are described in the first column and the evaluation metrics in the second. The three experiments report the IPC and $E_{BTB\&LP\_PM}$ of CBTB and [BABTB with EB] each with the corresponding power managers.

Figure 5-3(a) presents the energy results. For each power-managed BTB, T2 is fixed to 16K cycles and the energy results are arranged into: 64-cycle, 128-cycle, 256-cycle, 512-cycle, 1K-cycle, 2K-cycle, 4K-cycle and 8K-cycle of T1. Each power-managed BTB has its own optimum T1. They are 2K cycles for CBTB_PM, 2K cycles for BABTB_EB_PM, and 128 cycles for BABTB_EB_PM. From greatest to least, the order in power efficiency of each combination with its optimum T1 is: BABTB_EB_PM, BABTB_EB_PM, and CBTB_PM.

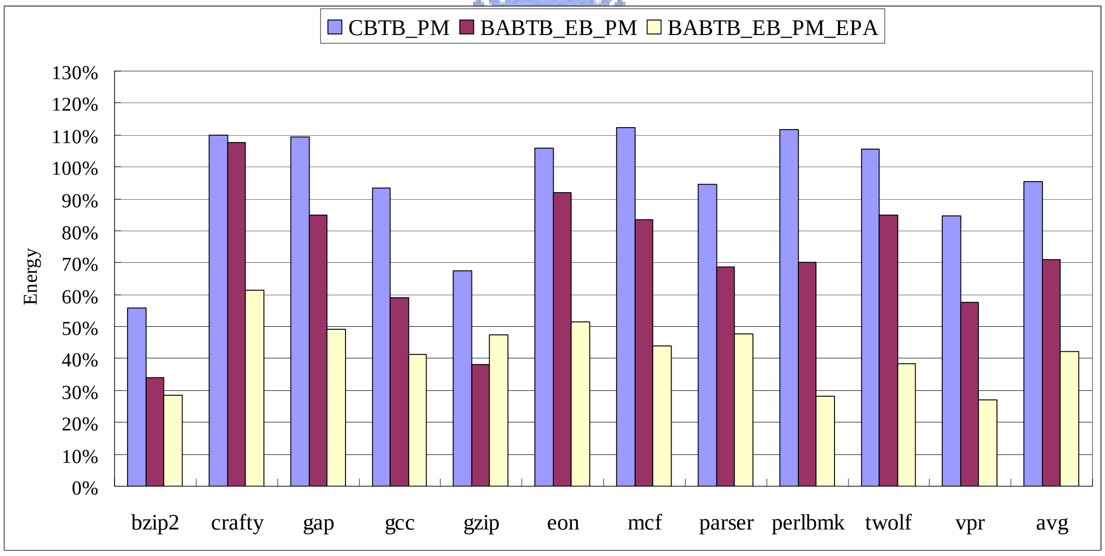
Figure 5-3(b) and (c) show the energy and performance results for each power-managed BTB with its optimum T1, respectively. Except for *gzip*, BABTB_EB_PM yields the lowest energy for all benchmark programs. On average, compared to the BABTB without any low-power design, the IPC degradation rate of BABTB_EB_PM is only 0.38%.

Figure 5-3(d) presents the average ratio of each power mode for each power-managed BTB with its optimum T1. Figure 5-3(e) further breaks down the average $E_{BTB\&LP\_PM}$ according to the power sources; including BTB active leakage energy (EActive), BTB drowsy leakage energy (EDrowsy), power manager energy (EPM), and the chip-wide energy during the extra stall cycles (EStall). The analyses of these results are:

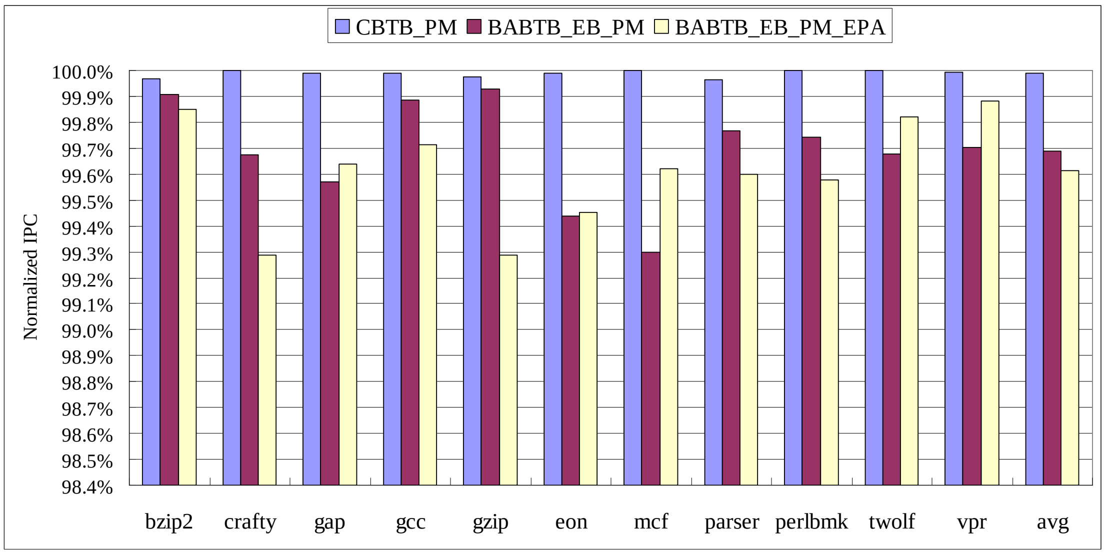
- CBTB_PM: In CBTB, each fetched instruction performs BTB lookup. Moreover, consecutive BTB lookups are likely to hit the different BTB entries. This phenomenon implies that BTB entries spend a lot of time in active mode due to the frequent accesses. This is the primary reason that CBTB_PM has the highest BTB active leakage energy.
- BABTB_EB_PM: Compared to CBTB_PM, power mode transition traffic in BABTB_EB_PM is decreased due to the fact that BTB is accessed only when the BTB index is changed. Therefore, it has better power efficiency than CBTB_PM.
- BABTB_EB_PM_EPA: Entry pre-activation helps to significantly shorten T1. Therefore, BABTB_EB_PM_EPA consumes the lowest leakage energy among all. On average, BABTB_EB_PM_EPA yields a BTB leakage energy reduction of 57.77%.


CBTB_PM
BABTB_EB_PM
BABTB_EB_PM_EPA
Energy
100%
95%
90%
85%
80%
75%
70%
65%
60%
55%
50%
45%
40%
64
128
256
512
1K
2K
4K
8K
Decay interval for drowsy mode (# of cycles)

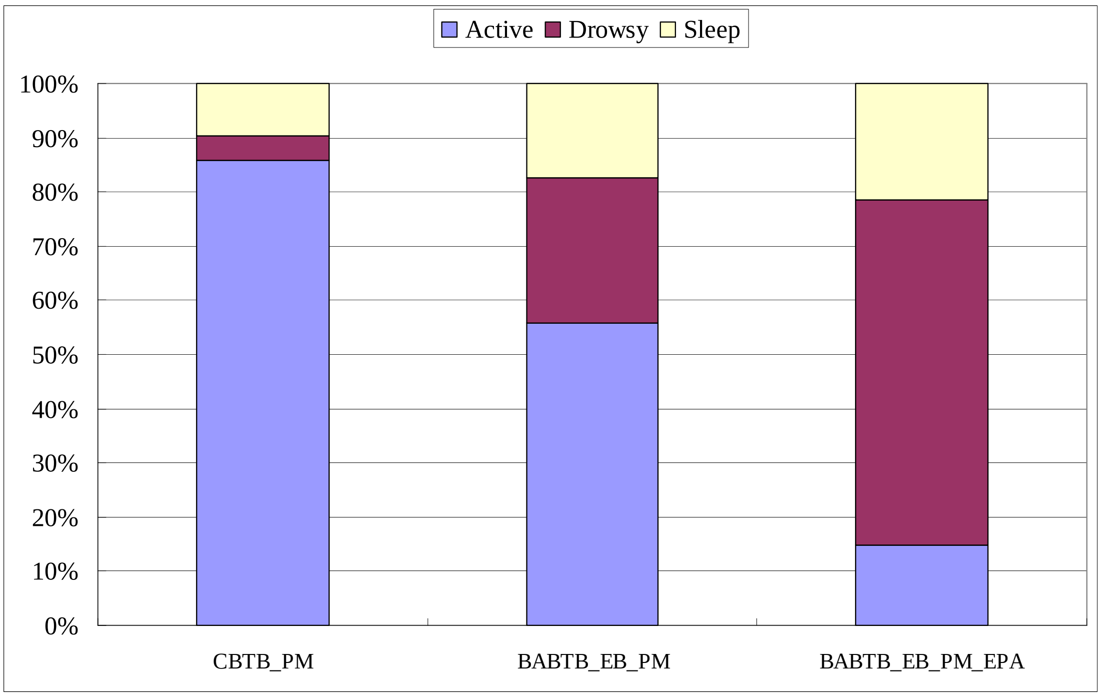
(a)


CBTB_PM
BABTB_EB_PM
BABTB_EB_PM_EPA
Energy
130%
120%
110%
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
bzip2
crafty
gap
gcc
gzip
eon
mcf
parser
perlbmk
twolf
vpr
avg

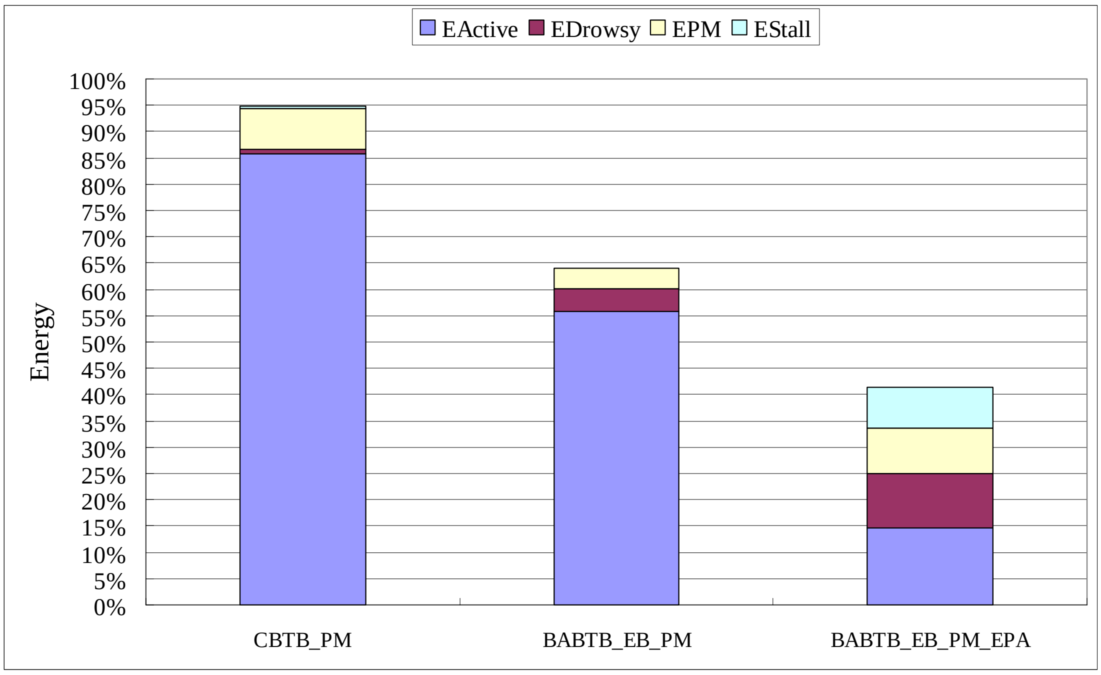
(b)


CBTB_PM
BABTB_EB_PM
BABTB_EB_PM_EPA
Normalized IPC
100.0%
99.9%
99.8%
99.7%
99.6%
99.5%
99.4%
99.3%
99.2%
99.1%
99.0%
98.9%
98.8%
98.7%
98.6%
98.5%
98.4%
bzip2
crafty
gap
gcc
gzip
eon
mcf
parser
perlbmk
twolf
vpr
avg

(c)


Active
Drowsy
Sleep
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
CBTB_PM
BABTB_EB_PM
BABTB_EB_PM_EPA

(d)



(e)

Figure 5-3: Comparisons for all power-managed BTBs. (a) Normalized BTB energy of all power-managed BTBs with various decay intervals of T1. (b) Normalized BTB energy for each power-managed BTB with its optimum T1. (c) Normalized IPC for each power-managed BTB with its optimum T1. (d) Ratio of each power mode for all power-managed BTBs with their optimum T1. (e) Energy breakdown for all power-managed BTBs with their optimum T1.

5.4 Summary

This topic addresses the issue of BTB leakage power reductions. Through decay based power manager and entry pre-activation, the leakage power consumption of BTB is reduced by an average of 57.77% with only 0.38% of performance degradation. It outperforms all the other power managed BTBs.

Chap 6 Reducing BTB entry size through I-cache based BTB

BTB and instruction cache (I-cache) have some similar fundamental characteristics. First, BTB lookup is performed in parallel with instruction fetch. Second, BTB is essentially an on-chip cache as well. Finally, the high order bits in the tag field of both BTB and I-cache are identical. If the tag memory of I-cache can be shared with BTB, both dynamic and static power of BTB can be significantly reduced. This chapter intends to reduce BTB dynamic and leakage energy simultaneously through the integration of instruction cache and BTB.

6.1 Related Work

Johnson et al. [24] proposed a BTB architecture which appends a BTB entry to each I-cache line. Flynn et al. [25] analyzed the performance between the typical BTB and Johnson's BTB. However, there was no discussion of power issues in Flynn's reports.

Compared with the typical BTB, the tag memory of Johnson's BTB is shared with that of I-cache. Instead of recording the full tag, each BTB entry only keeps $\log_2 x$ bits to indicate which instruction of the corresponding I-cache line has occupied this entry, where x is the number of instructions per I-cache line. Consequently, both the static and dynamic power consumptions of BTB are reduced due to the saving in storage. However, Flynn has shown that a typical 16-entry BTB performs more accurately than Johnson's BTB; even though there are more entries in Johnson's BTB.

This design may increase the chip-wide energy expenditure; hence, Johnson's BTB is not good choice for energy efficiency. If we want to take the advantage of tag-sharing with low-power, improving the accuracy of Johnson's BTB becomes a challenging design issue.

6.2 Design for I-cache based BTB

This chapter describes the proposed design for an instruction cache based BTB (ICBTB), including the mapping methods for I-cache lines and BTB entries (Section 6.2.1), the management for ICBTB (Section 6.2.2).

6.2.1 Mapping methods for I-cache lines and BTB entries

In order to share the I-cache tag memory with BTB while still maintaining high BTB accuracy, the contention problem in Johnson's BTB should be resolved. To obtain the maximal power reduction, the power consumptions of both the BTB and the entire chip are taken into consideration. Two methods achieving this goal are presented as follows.

The first method is to increase the capacity of Johnson's BTB. More specifically, we append k (k>1) BTB entries to each I-cache line. This straightforward idea only improves the BTB accuracy. Although the chip-wide energy is reduced due to the improvement of the BTB accuracy, the BTB energy may increase by a factor of k. Moreover, for the cache lines in which the number of branch instructions is less than k, some of the BTB entries are still useless and waste power. Therefore, this method is not a good choice.

The second method is to design an entry-sharing approach, through which the BTB entries can be used by multiple cache lines. Compared to Johnson's BTB, this method balances the unbalanced use of the BTB entries resulting from the non-uniformly distributed branches. The extra overhead of this alternative is that each BTB entry needs a few extra bits to indicate which cache line is using this entry. Therefore, the problem now becomes how to design this entry-sharing policy with minimal hardware cost.

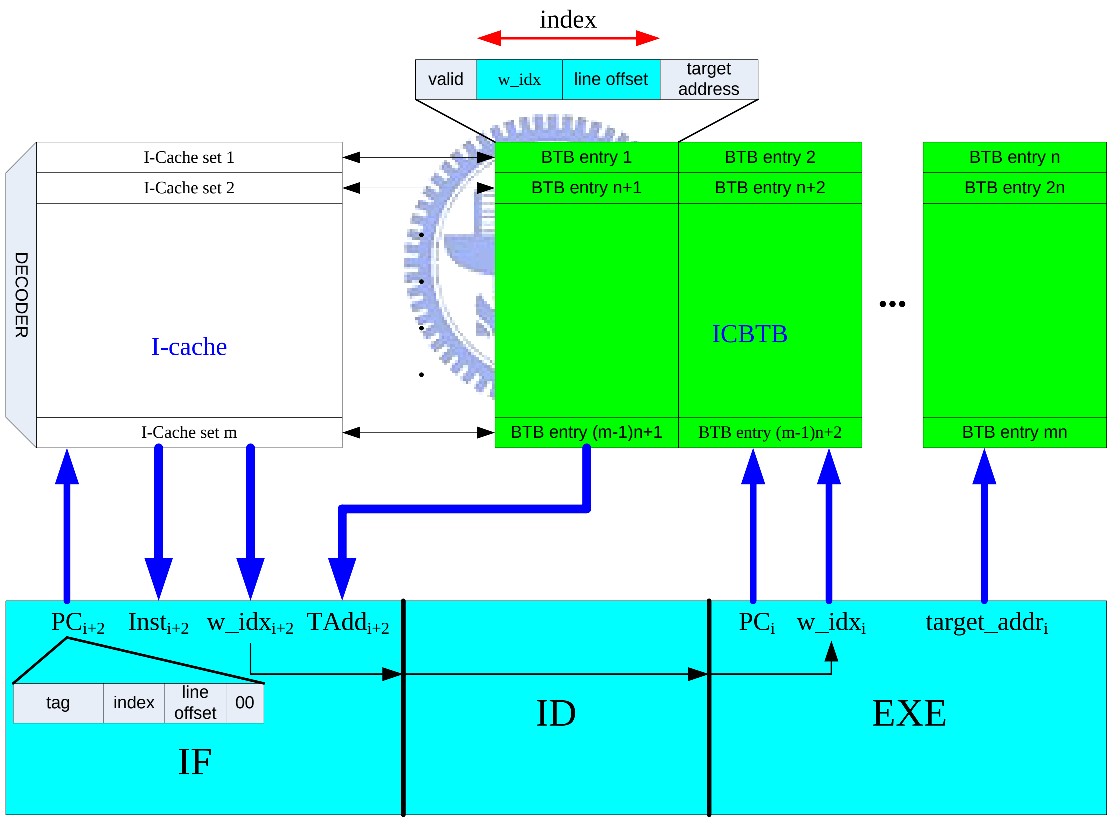


Figure 6-1: ICBTB and its integration with a typical pipeline front-end

6.2.2 ICBTB management

Figure 6-1 shows an ICBTB and its integration with a typical pipeline front-end, which is composed of instruction fetch (IF), instruction decode (ID), and execution (EX) stages. During IF stage, ICBTB lookup and instruction fetch are performed simultaneously. The dynamic branch predictor (not shown in Figure 6-1), which is either integrated in the BTB entry or implemented separately, performs the branch direction prediction at the same time. If ICBTB lookup result is a hit, then the fetched instruction is a branch. If the predicted direction is taken, the output of ICBTB, which is the target address of the branch, is used as the next PC. Otherwise, the sequential address is used.

As shown in Figure 6-1, *n* ICBTB entries are attached to each set of the I-cache (*n*-grouping ICBTB). In other words, the *n* ICBTB entries can be used by all branches of the corresponding I-cache set. This implies that the optimal number of *n* for low-power consumption is highly dependent on the I-cache configurations. We determine *n* through simulations. In addition, instead of recording the full tag, each ICBTB entry uses an index field to indicate which instruction in the corresponding I-cache set is using this entry. The index field is composed of a way index (w_idx) and a line offset, where the way index represents the cache line index of the set and the line offset indicates the offset within the line. For a typical *i*-way set-associative I-cache with *x* instructions per cache line, the width of the index field is $\log_2 x+\log_2 i$ bits.

During EX stage, both the target address and the direction of a branch are available. If the target address of the branch has not been recorded in ICBTB and the branch is taken, then the target address, way index and line offset will be recorded in ICBTB. While the line offset is directly extracted from the address of the branch, the way index is derived from the I-cache circuit. However, if the way index of an instruction is discarded after I-cache access, the way index information is lost when the instruction enters ICBTB (assume the instruction is a taken branch). Therefore, the way index of each instruction is preserved in the instruction pipeline until EX stage.

Figure 6-2 shows the block diagram of a 2-grouping ICBTB attached to a 2-way I-cache. In this case, the 2-grouping ICBTB is composed of two BTB arrays, and the number of BTB entries in each BTB array is equal to that of the I-cache sets. PC is divided into three parts to access the I-cache and the ICBTB simultaneously, where the index is used to determine the set, the tag is used to determine if there is a match in the I-cache, and the line offset is used to extract the current instruction from the accessed cache lines. Note that this block diagram is a functional description, not a physical implementation.

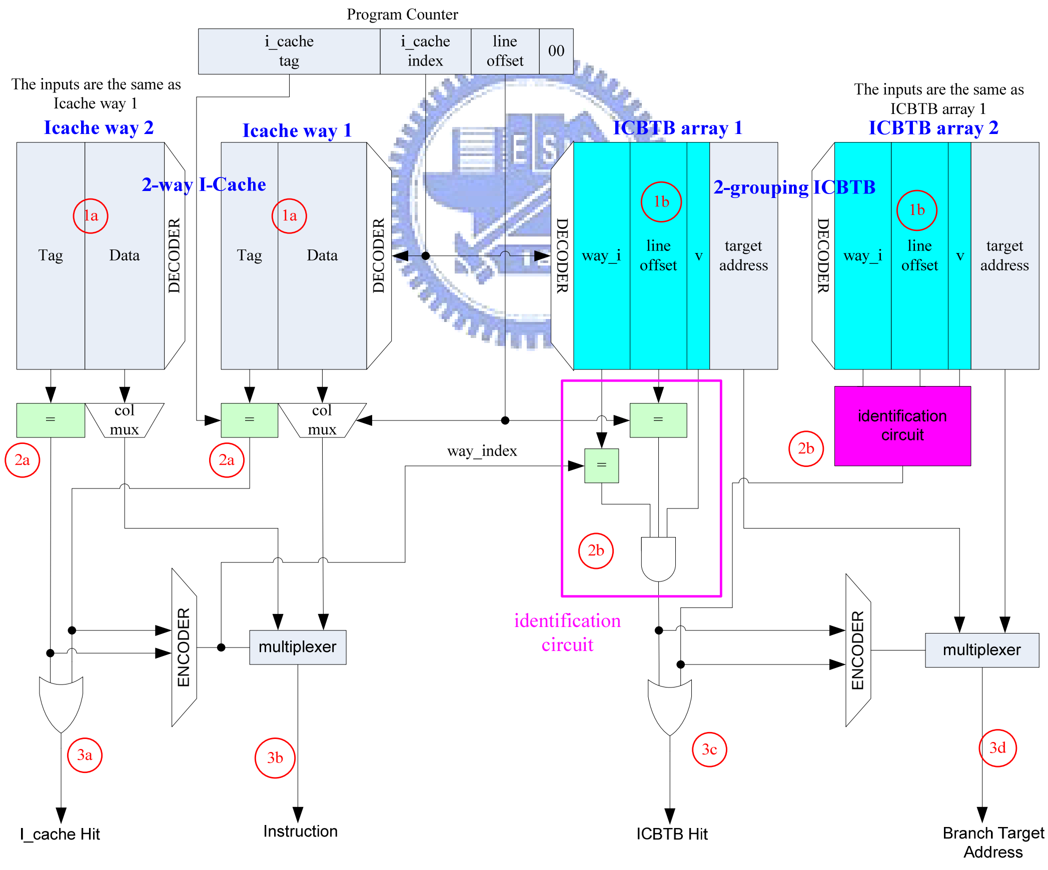


Figure 6-2: 2-grouping ICBTB attached to a 2-way I-cache

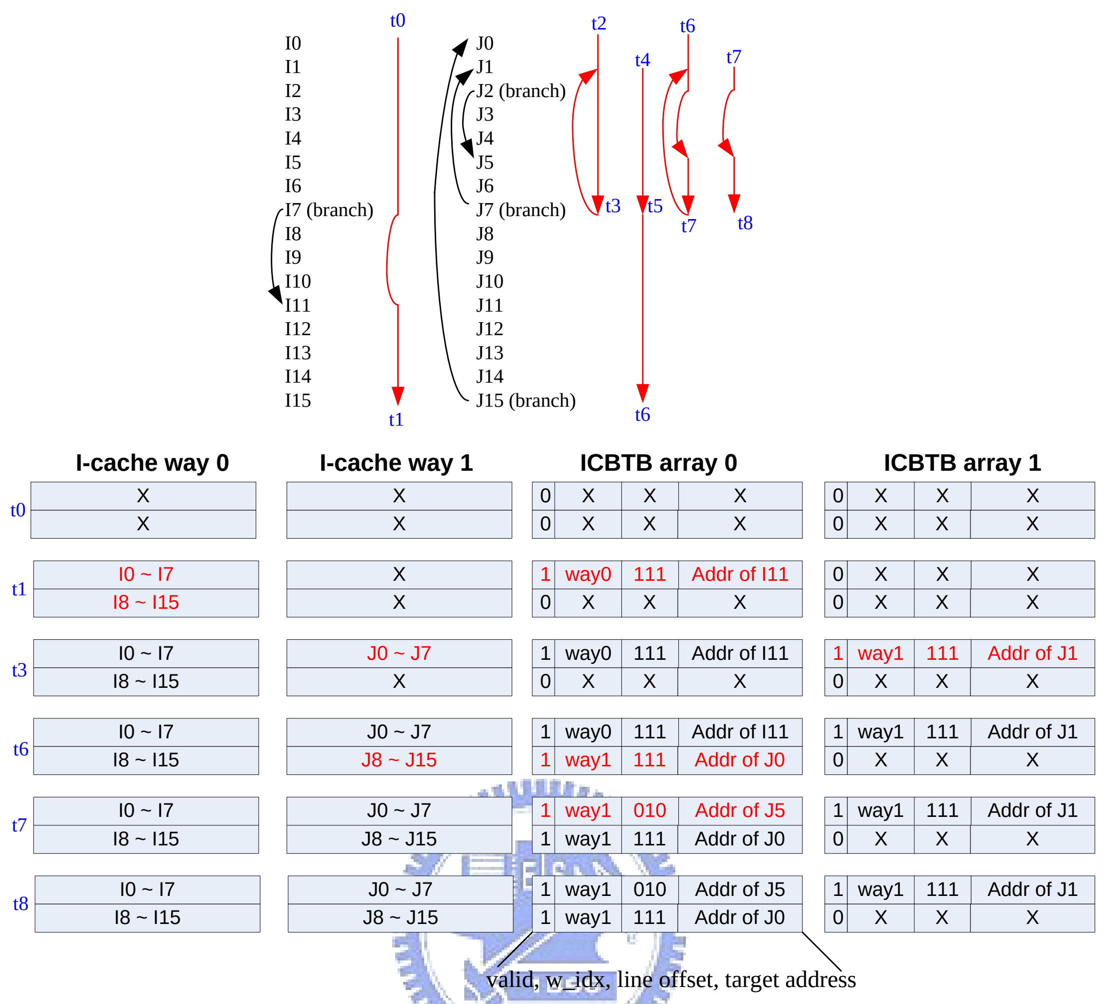
As shown in Figure 6-2, circles indicate the micro-operations of an ICBTB and I-cache access. These operations are described as follows:

- RAM accesses: The index part of PC is send to the I-cache RAMs and the ICBTB RAMs. Then, circles 1a and 1b indicate the RAM accesses of the I-cache and the ICBTB, respectively.
- Comparisons: The comparators perform the match operations. Circle 2a indicates the tag comparisons of each I-cache way. The way index is then generated by encoding the comparison results. Circle 2b indicates the way index comparisons and line offset comparisons of each ICBTB array. Then, whether the access to each ICBTB array is a hit is determined by ORing the comparison results and the valid bit.
- Data output: Based on the comparison results, the output instruction (circle 3b) and target address (circle 3d) are selected, and the I-cache hit signal (circle 3a) and ICBTB hit signal (circle 3c) are then produced.

When an I-cache line is replaced, the corresponding ICBTB entries are also invalidated. If a taken branch attempts to enter ICBTB and all the attached BTB entries are used, the LRU replacement operation is performed.

In order to illustrate ICBTB more clearly, we trace the operations in a 2-way I-cache and its attached 2-grouping ICBTB using a code example. Figure 6-3 shows the execution paths for two code segments and the corresponding operations in the I-cache and the ICBTB during the execution of these two code segments. The first code segment (I0~I15) takes place first, and then the second segment (J0~J15). There are four branch instructions: I7, J2, J7, and J15, and their targets are the addresses of I11, J5, J1, and J0, respectively. The stamps from t0 to t8 indicate the time order, and the arrows between the time stamps represent the execution paths: the arrow between t0 and t1 indicates the execution path on the first code segment, and all the remaining arrows are for the second code segment. The description over each time interval is listed as follows:

- t0->t1: Initially, both the I-cache and the ICBTB are empty. The first code segment is executed completely at t1, the corresponding code blocks are allocated in way 0 of the I-cache, and the first entry of array 0 in the ICBTB is assigned to I7.
- t2->t3: The program continues from J0 to J7. When J0 is fetched, the code block J0 to J7 is allocated in way 1 of the I-cache. After branch J7 is executed at t3, the first entry of array 1 in the ICBTB is assigned to J7.
- t4->t6: The program continues from J1 to J15. When J8 is fetched, the code block J8 to J15 is allocated in way 1 of the I-cache. After branch J15 is executed at t6, the second entry of array 0 in the ICBTB is assigned to J15.
- t6->t7: The program continues in the order of J0, J1, J2, J5, J6, and J7. Since branch J2 is taken and all the ICBTB entries are full, one of the entries should be replaced. In this example, the first entry of array 0 in the ICBTB is discarded.
- t7->t8: The program continues in the order of J1, J2, J5, J6, and J7. Both ICBTB lookups for branches J2 and J7 are hits at this interval.



	I-cache way 0	I-cache way 1	ICBTB array 0				ICBTB array 1			
t0	X	X	0	X	X	X	0	X	X	X
	X	X	0	X	X	X	0	X	X	X
t1	I0 ~ I7	X	1	way0	111	Addr of I11	0	X	X	X
	I8 ~ I15	X	0	X	X	X	0	X	X	X
t3	I0 ~ I7	J0 ~ J7	1	way0	111	Addr of I11	1	way1	111	Addr of J1
	I8 ~ I15	X	0	X	X	X	0	X	X	X
t6	I0 ~ I7	J0 ~ J7	1	way0	111	Addr of I11	1	way1	111	Addr of J1
	I8 ~ I15	J8 ~ J15	1	way1	111	Addr of J0	0	X	X	X
t7	I0 ~ I7	J0 ~ J7	1	way1	010	Addr of J5	1	way1	111	Addr of J1
	I8 ~ I15	J8 ~ J15	1	way1	111	Addr of J0	0	X	X	X
t8	I0 ~ I7	J0 ~ J7	1	way1	010	Addr of J5	1	way1	111	Addr of J1
	I8 ~ I15	J8 ~ J15	1	way1	111	Addr of J0	0	X	X	X

Figure 6-3: Code examples to illustrate ICBTB

6.3 Experiments

The objective of the experiments is to evaluate the impact of the proposed ICBTB on energy, performance, and area. The energy model used in this design is the same in that listed in Chap 2-3. Therefore, we report the results of the related metrics for various BTBs here.

6.3.1 Results and analysis

We now examine the interaction among BTB configuration, performance, energy, and storage characteristics for SPEC 2000. In the descriptions below, the term "average" means the arithmetic mean for the metrics across the simulated benchmark programs. Except for BTB accuracy, and IPC, all metrics reported in this section are normalized with respect to those of a 256-entry 4-way conventional BTB. The term "Base" means the conventional BTB, "Johnson" Johnson's BTB, and ICBTB_k the k grouping ICBTB.

6.3.1.1 BTB accuracy and IPC

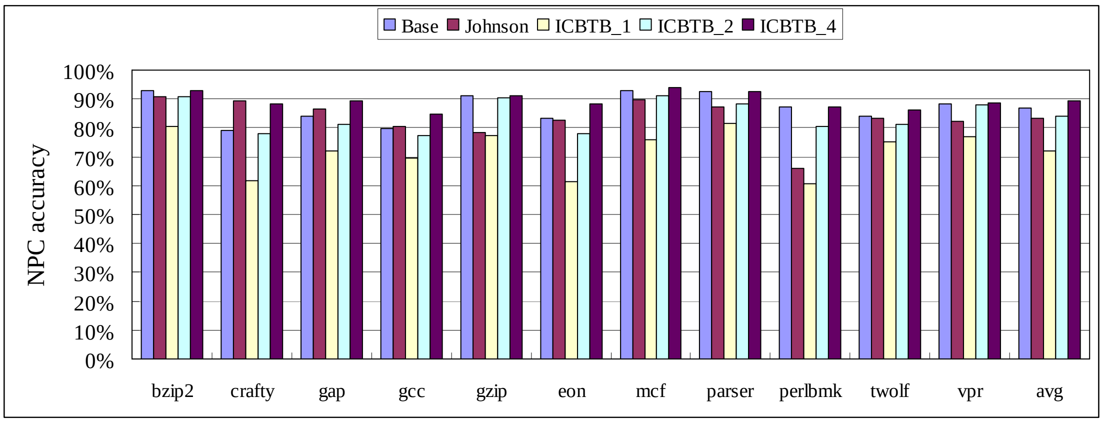


Figure 6-4: NPC accuracy comparisons

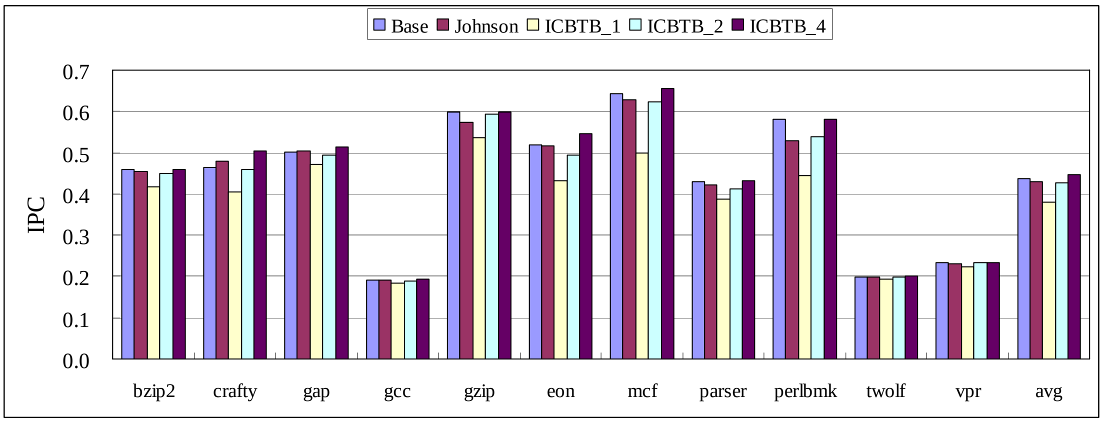


Figure 6-5: IPC comparisons

Figure 6-4 presents the average NPC accuracy, and Figure 6-5 presents the corresponding IPC. The important observations and analyses of the simulation results are listed as follows:

- Johnson's BTB has 512 entries, while the baseline BTB only has 256 entries. However, Johnson's BTB only achieves a relatively low BTB accuracy and low IPC due to the contention problem of the BTB entry for each I-cache line.
- ICBTB_2, which has the same entries with the baseline BTB, improves the NPC accuracy and IPC, significantly. ICBTB_2 archives similar NPC accuracy and IPC of the baseline BTB. Compared with the baseline BTB, ICBTB_2 only has 2.6% of performance degradation.

6.3.1.2 Energy results

Figure 6-6 shows the BTB dynamic energy, Figure 6-7 shows the BTB leakage energy, and Figure 6-8 shows $E_{BTB\&LP}$. The important observations and analyses of the simulation results are listed as follows:

- Compared with the baseline BTB, Johnson's BTB has relative high BTB dynamic and leakage energy due to it has more entries.
- ICBTB_2 saves 55.44% and 34.19% of BTB dynamic and leakage energy, respectively. With the overhead energy (E_{stall}) included, ICBTB_2 saves 16.54% BTB energy, with only 2.6% of performance degradation.

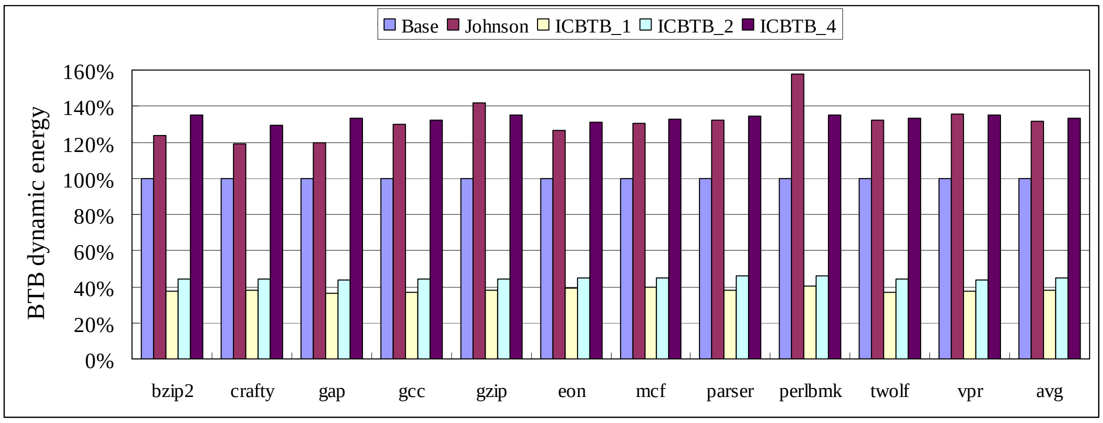


Figure 6-6: BTB dynamic energy comparisons

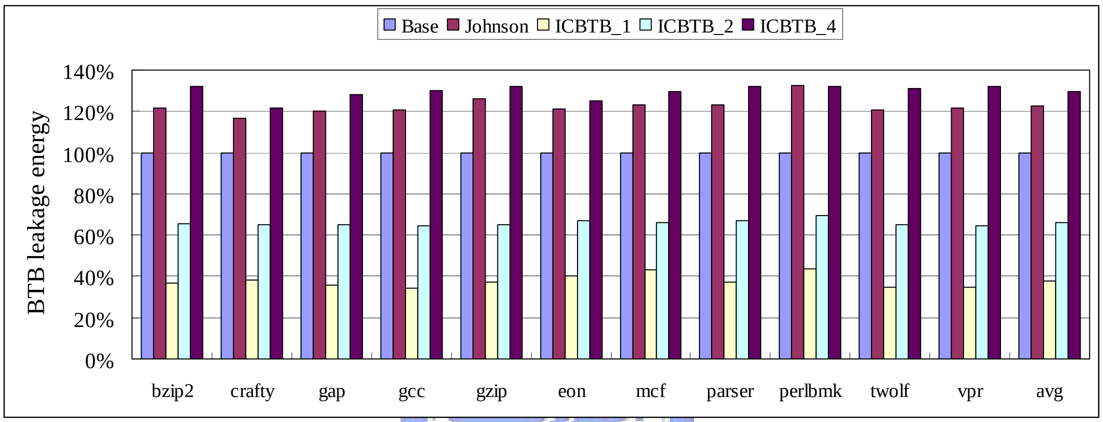


Figure 6-7: BTB leakage energy comparisons

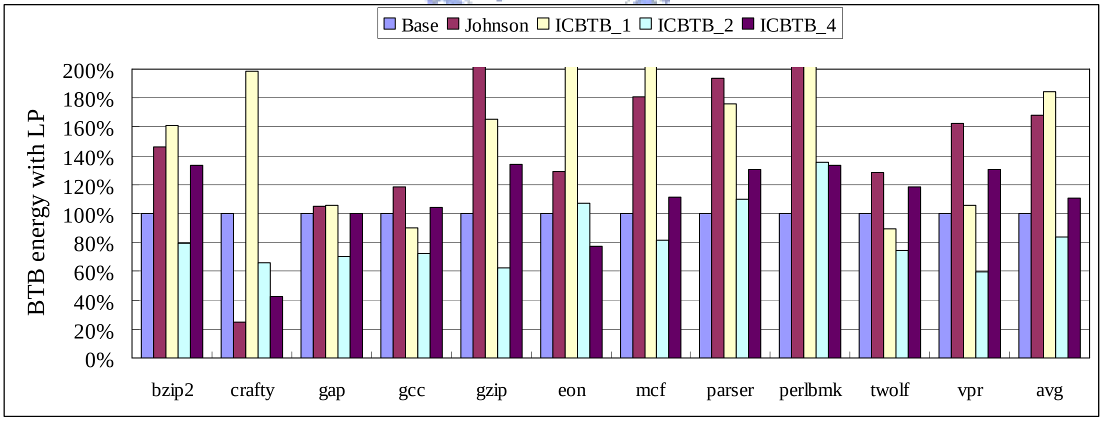


Figure 6-8: $E_{BTB\&LP}$ comparisons

6.4 Summary

This topic addresses the issue of BTB tag memory cost reductions. Through tag memory sharing with I-cache, both dynamic and leakage power in BTB are reduced. Since the BTB entries are shared to the branches in each I-cache set, the performance degradation is insignificant. This method outperforms Johnson's design.

Chap 7 Reducing the number of BTB entries through early target address generation

If the number of BTB entries can be reduced with tolerable performance degradation, both dynamic and leakage BTB power can be reduced significantly. We address this issue in this chapter. Since no related research addressed in this topic, we introduce the proposed method directly.

7.1 Design for early target address generation

7.1.1 Overview

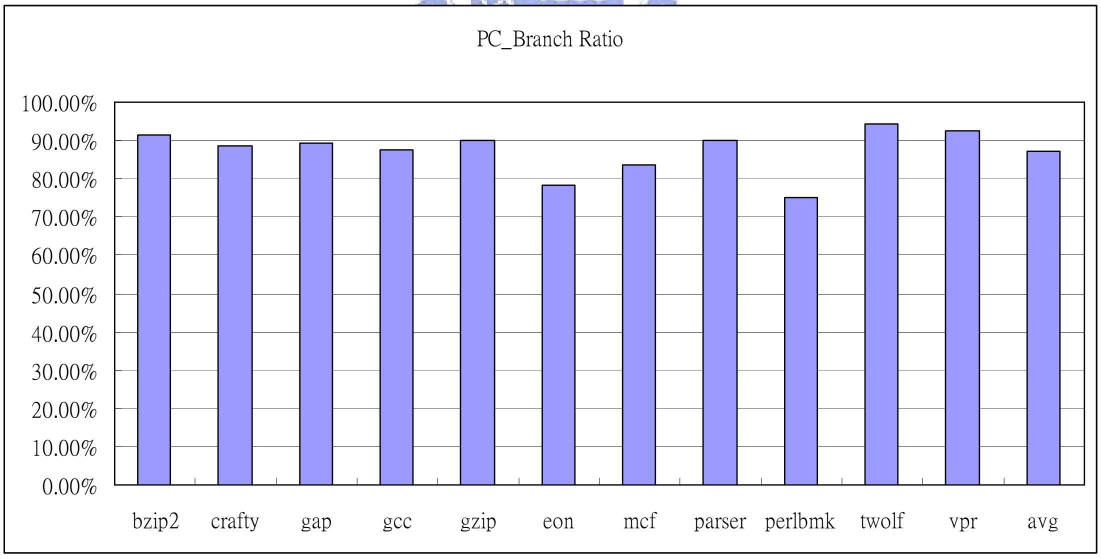


Figure 7-1: Ratio of PC-relative branches over total branches

To reduce the number of BTB entries without causing performance degradation, we must design a mechanism that is logically equivalent to BTB, that is, provide targets of branch instructions for the pipeline front-end. More specifically, the mechanism must fetch the instruction, identify its type (branch or not?), and furthermore provide the target address. Note all operations mentioned above should be finished in one cycle to meet the behavior of a conventional BTB. By having such a mechanism, branch instructions that can be processed need not to reside in the BTB anymore.

There're two major discoveries: the first result shows the proportion of different types of branches in SPEC2000, as in Figure 7-1. As we can see, the PC-relative branches were the absolute majority of all branches, averaging 89.31%. The second observation is shown in EQ19, which draws the target address generation rules for PC-relative branches. Generally, target address of PC-relative branches can be calculated when branch address and instruction (branch offset) is at hand.

$$\text{Target address} = (\text{branch address} + 1) + \text{branch offset} \qquad \text{(EQ19)}$$

Figure 7-2 shows a typical pipeline front-end with the proposed design. Based on the mentioned facts, a PC-relative branch handling unit (PCBHU) is established. PCBHU fetches the incoming instructions in advance, determines if the fetched instructions are PC-relative branches (pc_branch), and generates their corresponding target addresses (PCBHU_TA). To generate target address in time, identification and target address calculation mechanisms need to be established without incurring additional delay to the pipeline frond-end. If the branch target address of the current fetched instruction in the pipeline front-end has been already generated by PCBHU, this target address does not need to be stored in BTB. In other words, with PCBHU, the number of BTB entries can be reduced. We call the smaller BTB as reduced BTB (RBTB). To exploit maximum power reduction, the placement and indexing methods for RBTB need to be redesigned. A dynamic branch direction predictor (DBP) is used to predict the branch direction. If a branch instruction is predicted taken, its target address is provided by RBTB or PCBHU, and is used as the next PC. Otherwise, the sequential PC is used.

Section 7.1.2 introduces the PCBHU design. Section 7.1.3 analyzes the timing and describes the pipelining PCBHU for high clock rate processors. Section 7.1.4 draws the design and optimizations for RBTB.

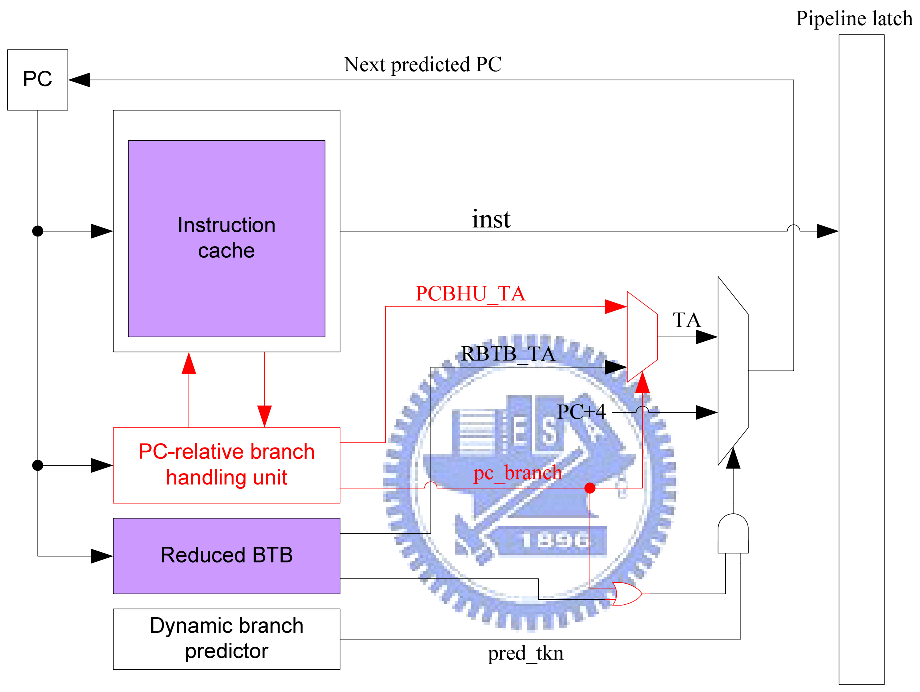


Figure 7-2: Typical pipeline front-end with PC-BHU and RBTB

7.1.2 PC-relative branch handling unit (PCBHU)

This section would be put into two parts. First, we introduce the fast instruction fetch and how it’s related to our PCBHU mechanism. Secondly, we take a closer look at the details of our PCBHU.

7.1.2.1 Fast instruction fetch

Fast instruction fetch is essential to the PCBHU must be understood. Since the PCBHU must be provided with the instruction for branch identification and target address generation, a fast instruction fetch mechanism ought to be established to supply the input for PCBHU. A cache line buffer [26], which buffers the presently accessed I-cache line, can meet the requirement of short access latency. The fast instruction fetch delivered by the cache line buffer provided the base for later on branch identification and target address generation. By accessing the cache line buffer, not only the PCBHU can fetch the instruction in minimum time, the overall power-hungry cache accessing procedure can also be aborted once the instruction is obtained from the instruction buffer. Note that PCBHU only fetch instructions from instruction buffer due to the timing consideration.

Here we would like to emphasize that the cache line buffer is not a part of our PCBHU design, but a proposed idea that has already been widely applied in modern processor implementations. Last but not least, the cache line buffer plays an important role in our work, and by correct adjustment, it could even be greatly beneficial to the design.

7.1.2.2 Branch Identification and Target address generation

Branch identification and target address generation are the two main logic functions of the PCBHU. In order to achieve shortest delay of PCBHU, the two logic units are designed to be exercised in parallel. That is, for a certain instruction, it should be decoded and be generated a target address simultaneously. If the instruction is a PC-relative branch, the generated target address can be used immediately.

Otherwise, the generated target address is meaningless. By referring to the result of the partial decoder, we can filter these meaningless target addresses from accidentally being used as the next PC.

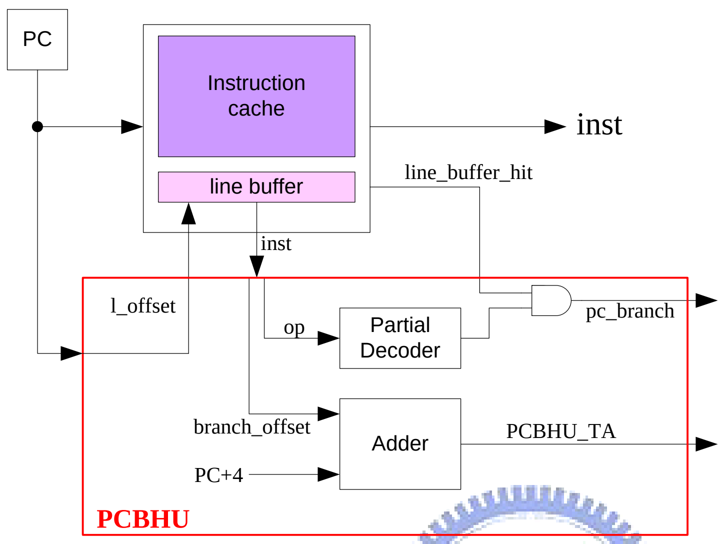


Figure 7-3: PCBHU

Figure 7-3 shows the design of PCBHU. We use a partial instruction decoder and a 32-bit adder to identify PC relative branch instructions and generate its targets, respectively. The partial instruction decoding can be easily done once the instruction is in hand. By examining the op code (or portion of op code) of the instruction, a simple logical decoder can identify the instruction in shortest amount of time and least overhead. If the currently fetched instruction is not a PC-relative branch or the current instruction cycle is cache line buffer miss, the generated target address is meaningless.

7.1.3 Timing analyses and pipelining PCBHU

If the target addresses of all PC-relative branches can be generated by PCBHU, RBTB do not need to store the target addresses of PC-relative branches anymore. Unfortunately, this is not possible in practical. The problem involves physical constrains of both the instruction buffer and the PCBHU. Strategies must be proposed for the timing issue caused by the physical constrains.

First, the cache line buffer cannot always be ready to provide the correct instruction for PCBHU. The cache line buffer should be refilled as program proceeds to a different cache line, under two circumstances: as execution sequentially stride into the next cache line or jump into a different cache line due to control instruction. During the refilling cycle, the PCBHU fails to obtain inputs (i.e. op code and branch offset) from cache line buffer.

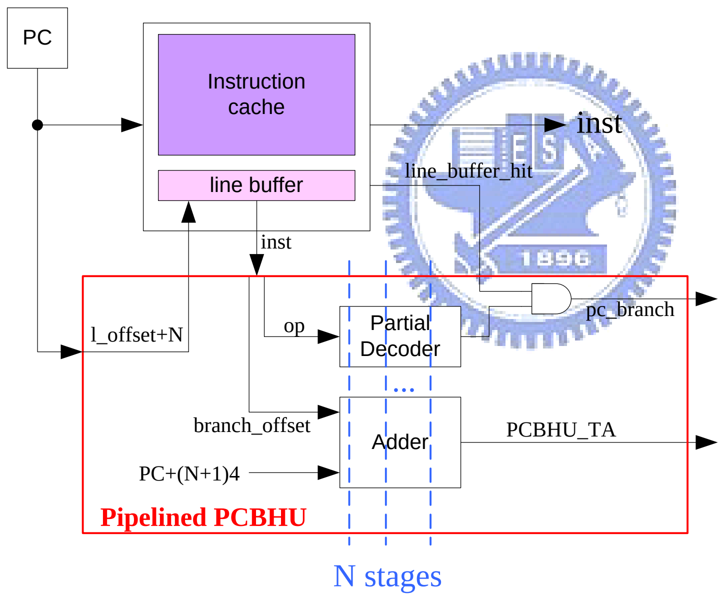


Figure 7-4: Pipelined PCBHU

Second, the circuits of partial decoding and target address generation have their time delays. Although one another can be hidden by exercising them in parallel, the process will introduce undesired delay to pipeline if either of the latencies exceeds the cycle time. Note that both partial decoding and target address generation can only be started after the instruction is successfully extract from the instruction buffer. These two physical constrains would lead to malfunction of PCBHU, which we would have a clearer view in the next section.

Suppose T_{PCBHU} is the delay time of PCBHU, TIB the delay time of cache line buffer, T_{PD} the delay time of the partial instruction decoder, T_{cycle} the clock cycle time and T_{add} the delay time of the 32-bit adder. Since the PCBHU has partial decoder and target address generation exercised in parallel, the overall latency of the PCBHU should be dominated by the critical path of the two logic units. Hence we have:

$T_{PCBHU} = \max (T_{PD}, T_{TAG})$

Assume N represents the number of cycles needed to successfully generate an target address. EQ20 can be used to find N.

$$(N-1) * T_{cycle} < T_{PCBHU} \leqq N * T_{cycle} \qquad \text{(EQ20)}$$

N is the main parameter that should be taken into consideration when it comes to integrations. With different values of N, come different approaches to implement PCBHU. The case N = 1 shown in Figure 7-3 is that the target address generation can be completed within one cycle.

While with $N \geqq 2$, modifications should be adapted for not incurring extra delay to the pipeline. The PCBHU mechanism should be pipelined, and a look-ahead technique ought to be applied. Take N = 2 as an example, an instruction must have its identification and target address generation process started **one cycle before it enters the pipeline front-end**. In other words, PC+8 must enter the pipelined PCBHU as PC hits the system pipeline front-end, and from then on, PCBHU is always one step ahead the system. In general, this can be parameterized according to the value of N and the modified PCBHU is shown in Figure 7-5.

Timing incompatibility is initiated by an instruction buffer refill. During the first cycle the instruction buffer is invalidated for the refill process to complete. Then, another (N-1) cycle is needed for a target address to be generated. The instructions that enter the pipeline within the 1 + (N-1) cycles are said to be **timing incompatible**. For target address cannot be correctly generated for these instructions by PCBHU. Intuitively, if there are PC-relative branches among these timing incompatible instructions, branch penalty would be incurred. In order to avoid performance degradation caused by these timing incompatible branches, we propose to put them into the Reduced BTB to keep the accuracy.

7.2 Experiments

The objective of the experiments is to evaluate the impact of the proposed idea on energy and performance. Section 7.2.1 shows the energy model. Section 7.2.2 presents the results and analysis.

7.2.1 Energy model

The power models are constructed by EQ21. It models the energy impacts of the applied low-power designs; including BTB leakage energy, BTB dynamic energy, and the overhead energy of PCBHU. Since we only use the no-performance-degradation configuration of this design, E_{stall} is not included in $E_{BTB\&LP\_PM}$. Table 7-1 lists the energy parameters estimated using CACTI 4.2 web interface [15].

$$E_{BTB\&LP\_PM} = E_{BTB\_acs} * Counts_{BTB\_acs} + P_{BTB\_leak} * Time_{EXE} + E_{PCBHU} \qquad \text{(EQ21)}$$

Table 7-1: Energy Parameters

Energy parameters	
Baseline BTB access energy per access	4.32pJ
Baseline BTB leak energy/cycle	2.07pJ
PCBHU access energy per access	0.22pJ
PCBHU leak energy/cycle	0.1pJ
RBTB (256-entry 2-way) access energy per access	3.12pJ
RBTB (256-entry 2-way) leak energy/cycle	2.01pJ
RBTB (128-entry 4-way) access energy per access	3.17pJ
RBTB (128-entry 4-way) leak energy/cycle	1.16pJ
RBTB (128-entry 2-way) access energy per access	2.16pJ
RBTB (128-entry 2-way) leak energy/cycle	1.07pJ
RBTB (64-entry 4-way) access energy per access	3.26pJ
RBTB (64-entry 4-way) leak energy/cycle	0.7pJ
RBTB (64-entry 2-way) access energy per access	1.72pJ
RBTB (64-entry 2-way) leak energy/cycle	0.6pJ

7.2.2 Results and analysis

7.2.2.1 Performance results

Figure 7-5 shows the normalized IPC for the baseline BTB (256-entry 4-way), and the PCBHU design with various RBTBs. The notation "256_4_2" means a 2-stage PCBHU with a 256-entry 4-way RBTB. The analyses are:

- A PCBHU with a smaller RBTB yields a better IPC than the baseline BTB. It implies that PCBHU successfully reduces the number of BTB entries.
- For single stage PCBHU and under the constraint of no performance degradation, the cheapest RBTB configuration is 64-entry 2-way.
- For 2-stage PCBHU and under the constraint of no performance degradation, the cheapest RBTB configuration is 128-entry 2-way.

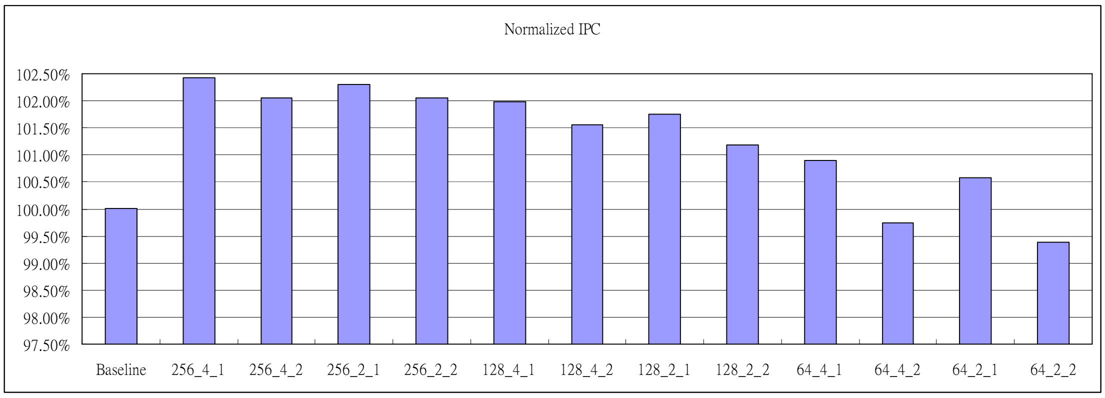


Figure 7-5: Normalized IPC for the baseline BTB (256-entry 4-way), and the PCBHU design with various RBTBs

7.2.2.2 Energy results

Figure 7-6 shows the $E_{BTB\&LP}$ of a single stage PCBHU with 64-entry 2-way RBTB. Figure 7-7 shows the $E_{BTB\&LP}$ of 2-stage PCBHU with 128-entry 2-way RBTB. The results show that with a single stage PCBHU with a 64-entry 2-way RBTB, the BTB energy is reduced by an average of 61.11%. A 2-stage PCBHU with a 64-entry 2-way RBTB reduces the BTB energy by an average of 44.58%.

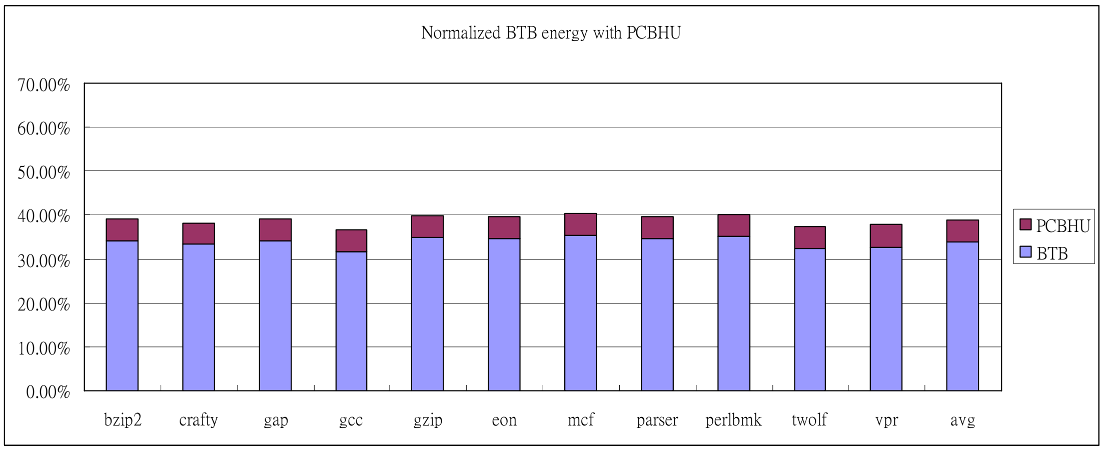


Figure 7-6: $E_{BTB\&LP}$ of a single stage PCBHU with 64-entry 2-way RBTB

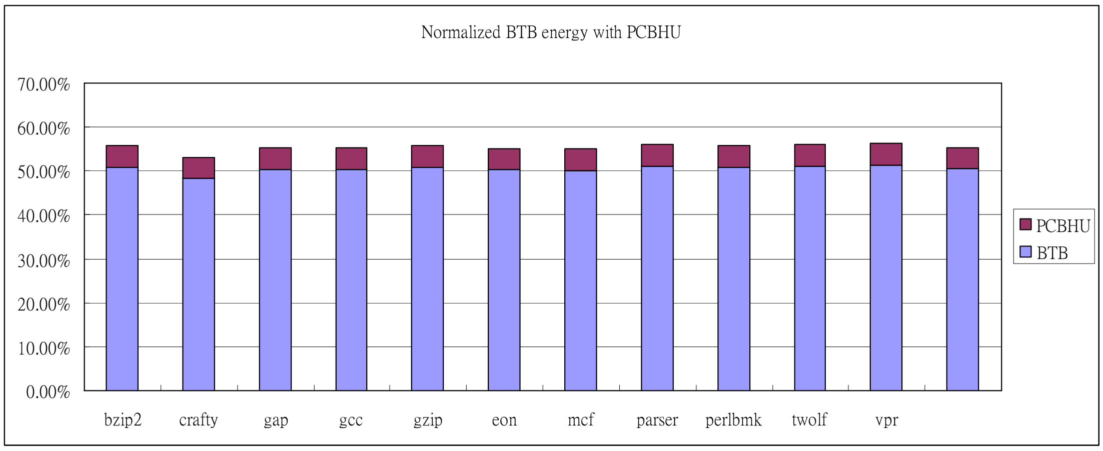


Figure 7-7: $E_{BTB\&LP}$ of 2-stage PCBHU with 128-entry 2-way RBTB

7.3 Summary

This topic addresses the issue of the reduction of the number of BTB entries. Through early generating the branch target address of PC-relative branches, most branch target addresses of PC-relative branches do not need to be recording in BTB. With this design, both dynamic and leakage power in BTB are reduced without performance degradation.

Chap 8 Integrations

This chapter introduces the integrations of the proposed methods in this thesis. Section 8.1 presents the integration of power reduction only method. Section 8.2 gives integration of storage cost reduction methods.

8.1 Integration of power reduction only method

In the design of BABTB with EB, BABTB lookups are performed only upon the current BABTB index is different than the last one. However, if the current accessed block is branch-free, the corresponding BABTB lookups are useless. If the subsequent code block is branch-free or not is known, the filtering of the useless BABTB lookups can be done easily. It can further reduce the BABTB access power.

In Section 3.2, we had proposed a NBD method for unnecessary typical BTB lookup filtering. If the NBD recording idea is applied for the recoding of the number of branch-free blocks before the subsequent branch on execution path, the number of BABTB accesses can be further reduced.

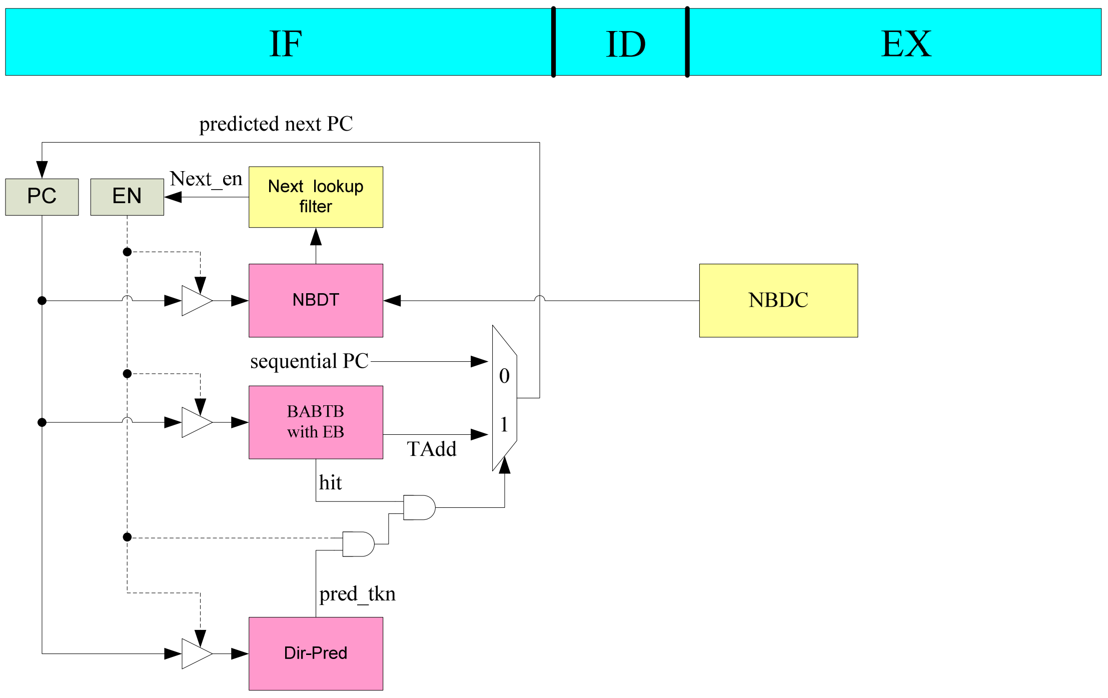


Figure 8-1: Integration of NBD method and BABTB with EB

Figure 8-1 shows the system architecture for the integration of the NBD method and BABTB with EB. We modify the NBD method to filter the useless BABTB lookups. The modification is listed as follows:

- NBDC is used to record the number of branch-free blocks from a branch instruction to its subsequent branch instruction on execution path.
- NBDT is used to record the NBDC collecting results.
- An n-bit enable register (ER) is used to record the number of upcoming branch-free blocks before the subsequent branch instruction, on execution path. Therefore, with correct management of ER, further lookup filtering of BABTB becomes trivial. The ER management is similar to the one proposed in Section 3.2.3.
- All the other implementation details are the same as the NBD method proposed in Section 3.2.

With the above design, BABTB lookups are performed only upon the current fetched block has branch instructions. If this BTB can be power managed, the leakage power can be also saved. The decay-based power manager with entry pre-activation can be applied to this BTB. However, if the entry pre-activation is applied without any modification, the next accessed BTB entry may be pre-activated too early. Therefore, the pre-activation is performed only when ER is equal to 0.

8.2 Integration of storage cost reduction methods

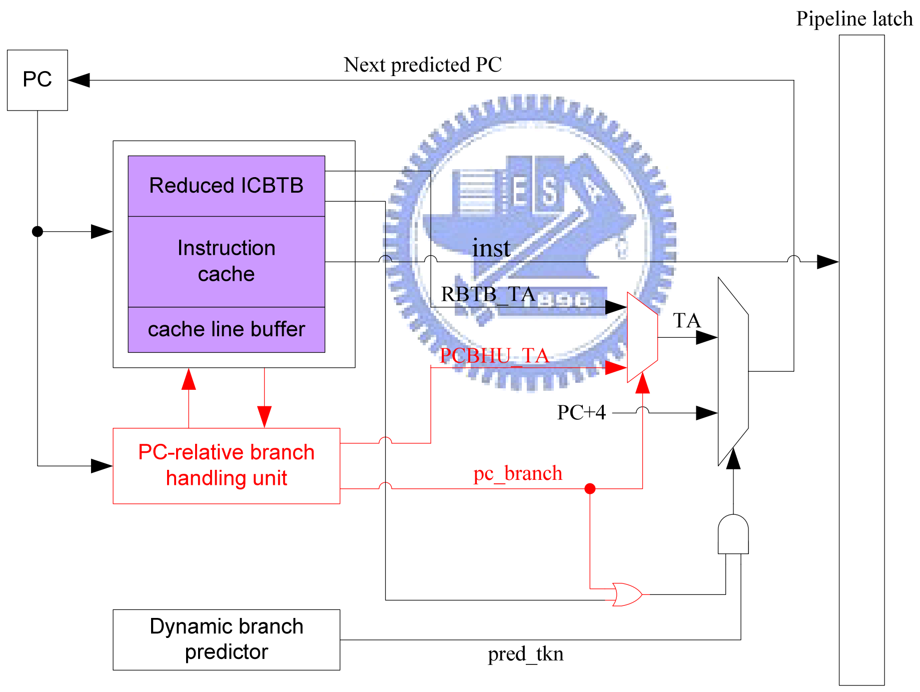


Figure 8-2: Integration of ICBTB and PCBHU

The proposed methods in Chap 6 and Chap 7 are for the purpose of BTB storage cost reduction. ICBTB can reduce the BTB entry size. PCBHU can reduce the number of BTB entries. If these two methods can be integrated, the BTB storage cost can be further reduced.

Figure 8-2 shows the integration. PCBHU is adapted to a processor with ICBTB. All the branch target addresses that can be handled by PCBHU are not stored in ICBTB. Therefore, the number of ICBTB entries can be reduced.

Chap 9 Conclusion

This thesis addresses the low-power issue for BTB. Through the proposed methods, the number of BTB accesses, the BTB access power, and the BTB leakage power are reduced with tolerant performance degradation. With the integration of the proposed methods, the BTB total BTB power consumption can be further reduced. The methods and the integrations can easily be adopted in processor cores without the need to modify program codes, system software, or ISA.

Reference

[1] S. Manne et al., “Pipeline Gating: Speculation Control for Energy Reduction,” in *Proceedings of International Symposium on Computer Architecture*, 1998, pp. 132-141

[2] D. Parikh et al., “Power-aware branch prediction: Characterization and design,” *IEEE Transactions on Computers,* Vol. 53, 2004, pp. 168-186.

[3] P. Petrov, and A. Orailoglu, “Low-power branch target buffer for application specific embedded processors,” *IEE Proceeding Computers & Digital Techniques*, Vol. 152, 2005, pp. 482-488.

[4] Y.-C. Hu et al., “Low-Power Branch Prediction,” in *Proceedings of the 2005 International Conference on Computer Design,* 2005, pp. 211-217.

[5] J. S. Hu, A. Nadgir, N. Vijaykrishnan, M. J. Irwin, M. Kandemir, “Exploiting Program Hotspots and Code Sequentiality for Instruction Cache Leakage Management,” appear in Proc. of the International Symposium on Low Power Electronics and Design (ISLPED'03), Seoul, Korea, August, 2003.

[6] W. Zhang, J. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. Irwin, “Compiler-directed instruction cache leakage optimization,” in Proc. IEEE/ACM Int. Symp. Microarchitecture, 2002, pp. 208–218.

[7] N. Kim, K. Flautner, D. Blaauw, and T. Mudge. “Circuit and microarchitectural techniques for reducing cache leakage power,” IEEE Trans. VLSI, 12(2):167–184, Feb. 2004.

[8] S. Kaxiras, Z. Hu, and M. Martonosi, “Cache decay: Exploiting generational behavior to reduce cache leakage power,” in Proc. IEEE/ACM Int. Symp. Computer Architecture (ISCA28), 2001, pp. 240–251.

[9] Smith J. E.1981.A study of branch prediction strategies. In Proceedings of the 8th Annual International Symposium on Computer Architecture, May. 1981, 135–148

[10] Yeh, T.-Y. AND Patt, Y. N. 1991. Two-level adaptive training branch prediction. In Proceedings of the 24th Annual International Symposium on Microarchitecture, Albuquerque, New Mexico, November 1991, 51-61

[11] Pan, S.-T., So, K., AND Rahmeh, J. T. 1992 Improving the accuracy of dynamic branch prediction using branch correlation. In Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, Massachusetts, Oct. 1992, 76–84

[12] Chang, P.-Y., Hao, E., AND Patt, Y. N. 1995 Alternative implementations of hybrid branch predictors. In Proceedings of the 28th Annual International Symposium on Microarchitecture, Dec. 1995, 252–257

[13] Intel Corporation. 2003. Intel XScale(R) Microarchitecture for the PXA255 Processor User Manual.

[14] D. Burger, and T. Austin, "The SimpleScalar tool set, version 2.0," Computer Architecture News, June 1997, pp. 13–25.

[15] David T. et al., "Cacti 4.2 web interface," in *http://quid.hpl.hp.com:9081/cacti/*

[16] C. Yang, and A. Orailoglu, "Power Efficient Branch Prediction through Early Identification of Branch Addresses," in *Proceedings of the international conference on Compilers, architecture and synthesis for embedded systems*, 2006, pp. 169-178.

[17] Y.-J. Chang, "Lazy BTB: reduce BTB energy consumption using dynamic profiling", in *Proceedings of the conference on Asia South Pacific design automation*, 2006, pp. 917-922

[18] J. L. Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach," 3rd Ed., Morgan Kaufmann Publishers, Inc., 2003, pp. 315.

[19] Chang, Y.-J., and Lan, M.: “Two New Techniques Integrated for Energy-Efficient TLB Design”, IEEE Transactions on Very Large Scale Integration Systems, 2007, 15, (1), pp. 13 – 23

[20] Powell, M., Yang, S.-H., Falsafi, B., Roy, K., and Vijaykumar, T. N.: “Gated Vdd: A Circuit Technique to Reduce Leakage in Cache Memories”, Proceedings of the 2000 International Symposium on Low Power Electronics and Design, Rapallo, Italy, Jul. 2000, pp. 90 – 95

[21] Flautner, K., Kim, N., Martin, S., Blaauw, D., and Mudge, T.: “Drowsy caches: Simple techniques for Reducing Leakage Power”, Proceeding of the 29th International Symposium on Computer Architecture, Anchorage, Alaska, May 2002, pp. 148 – 157

[22] Zhang, W., and Allu, B.: “Loop-based Leakage Control for Branch Predictors”, Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, Washington DC, USA, Sep. 2004, pp. 149 – 155

[23] Hu, Z., Juang, P., Skadron, K., Clark, D. and Nartonosi, M.: “Applying Decay Strategies to Branch Predictors for Leakage Energy Savings”, Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors, Freiburg, Germany, Sep. 2002, pp. 442 – 445

[24] W. Johnson, “Super-Scalar Processor Design”, Technical Report NO. CSL-TR-89-383, Stanford University, June 1989

[25] Brian K. Bray and M. J. Flynn, “Strategies for branch target buffer”, In Proceedings of the 24th annual international symposium on Microarchitecture, 1991, pp. 42-50.

[26] Ching-Long Su and Despain, A.M., “Cache designs for energy efficiency”, In Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences, 1995, pp. 306-315