The Design and Development Of A Branch Target Buffer Based On A 2-bit Prediction Scheme For A 32-bit RISC32 Pipeline Processor

BY

Ho Ming Cheng

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONS)

COMPUTER ENGINEERING

Faculty of Information and Communication Technology

Department of Information Technology and Engineering

May 2013

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: _____

_____

_____

Academic Session: _____

I _____

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

_____ _____

(Author's signature) (Supervisor's signature)

Address:

_____

_____ _____

_____ Supervisor's name

Date: _____ Date: _____

DECLARATION OF ORIGINALITY

I declare that this report entitled "**The Design and Development Of A Branch Target Buffer Based On A 2-bit Prediction Scheme For A 32-bit RISC32 Pipeline Processor**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : _____

Name : _____

Date : _____

ACKNOWLEDGEMENTS

I would like to say thanks a lot to my final year project supervisor, Mr. Mok Kai Ming who guided me through out the project with the right direction to complete my project. Without his guidance, the project might not be easily completed, and really thanks for spending so much of precious time guiding me through.

Next, I wish to thanks my friends. They have been so supportive when I am facing problem. They have been helping in releasing my pressure and tension while doing the project.

Last but not the least, I would like to say a big thank you to my family for their support and encouragement to continue to work on and complete the course.

Ho Ming Cheng

ABSTRACT

This project is the enhance of the Branch Prediction development of the RISC32 processor based on RISC Architecture that previously processor is developed in Universiti Tunku Abdul Rahman under Faculty of Information and Communication Technology. After reviewing the previous projects, there is a part where the branch prediction is not complete. The purpose of this project is remodeling the branch prediction block of previous design and applying the cache technology as the buffer of branch prediction. All the modeling will be using HDL (Hardware Description Language) which is Verilog and verification will be done to test the compatibility.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BPB	Branch Prediction Block
BTB	Branch Target Buffer
ID	Instruction Decode
IF	Instruction Fetch
I/O	Input Output
LRU	Least Recently Used
MIPS	Microprocessor without Interlocked Pipelined Stages
PC	Program Counter
RISC	Reduced Instructions Set Computer
RISC32	RISC32 Micro-processor
RTL	Register Transfer Level

Chapter 1 - Background

1.1.MIPS - 32-bit RISC Processor

MIPS is the short form for Microprocessor without Interlock Pipeline Stages. It is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Technologies.

Its primary implementations are embedded system, eg. Windows CE devices, video game consoles and so on. It also has several optional extensions that are available, such as MIP-3D which added new instructions for improving the 3D graphics applications' performance, MDMX that for accelerating multimedia applications and so on.

1.2 Hazard - Control Hazard

Hazards are commonly occur in the pipeline microprocessor. Basically there are three categories, which is structural hazard, data hazard and control hazard. Throughout this report, it only will be focusing on the control hazard.

Control hazard, is also known as the branching hazard, it is an attempt to make decision on the program flow before the condition has been evaluated and the new arrival of PC (Program counter) target address.

Three events that cause control hazard to be occurred:

- Conditional branch: beq, bne
- Unconditional branch: j, jal, jr
- Exceptions

The program flow will not be affected if the branch is untaken and it will follow the pre-designed flow. If the branch is taken, the program flow will be incorrect. There are multiple ways designed to solve this problem, such as stalling the next instruction until the branch is completed, flushing all the other stages in the pipeline that causing huge

delay into the system and never the less, a hardware implementation.

1.3 Motivation and Problem Statement

Based on the ongoing project that has been developing in the Faculty of Information and Communication Technology of Universiti Tunku Abdul Rahman, it is consists of RISC32 RISC processor. Following reasons make the initiation of motivations of this project:

The current problem found:

- The branch prediction of the previous design is not well-implemented
- The design of the BTB is not fully applied
- The branch prediction has large latency
- Design to overcome control hazard not well-developed

1.4 Project Scope

The main purpose of this project is to build a 2-bit prediction scheme BTB for the RISC32 RISC processor. BTB is the main component in the branch prediction. Building the BTB could improve the performance of the CPU during the branch instructions. In order to build the BTB, cache memory design is needed. Following precautions should be taken place:

- Performance of BTB
- Read and write of BTB
- Hit rate (Branch taken) of BTB
- Verification methodology testing the overall function of the BTB
- Improve the previous design of BTB

1.5 Project Objective

The main objective of the project is to design and develop a BTB based on a 2-bit prediction scheme. In respect to this objective, there are some sub-objectives to be archived as follow:

- Analysis - Cache memory design and its performance, mapping and the operation of cache memory.
- Specification Design and Development - Micro-architecture specification of the BTB will be defined according to the instruction of the program
- RTL Modeling - BTB will be modeled in ModelSim, using HDL, hardware language, which Verilog is used in this project. The model will be formed by using RTL (Register Transfer Level)
- Verification - Multiple test cases will be developed to run testing and verify the functionality of BTB

1.6 Contribution and Innovation

As a short summary of above problem statement, there is an incomplete 32-bits RISC microprocessor core-based development environment being develop. The development environment refers to the following availability:

- A well-developed design document, includes chip specification and micro-architecture specification
- A fully functional well-developed BTB in the form of RTL written in Verilog
- A well-developed verification environment for BTB. The verification specification including suitable verification methodology, verification techniques, test plans, test program and so on.

With the new BTB is being developed, the control hazards in RISC32 processor can be solved. With the design and development in RTL model, researcher can quickly verify model obtaining results, without worrying the development of verification environment and modeling environment. Research work could be speed up significantly.

Chapter 2 - Literature Review

2.1 Branch Target Buffer (BTB)

Due to the incompetency of the BHT design in the MIPS32 RISC processor, a better buffer design is introduced to solve the branching problems. The main reason why BHT is incompetent because it just give information whether the branch is taken or untaken, as what the information needed for the CPU is much more than just results of previous occurrence, so its always fine for the whole system if branches are never taken, problem comes when a branch is taken.

As to improve the previous design, BTB has a column which stores the branch target address or "the next address" as well as combining the design of the BHT. BTB can reduce the penalty of branches in the pipeline processor as it predicted the target of branch and stored the information needed in it. For a 5-stages pipeline processor, it become a must for the pipeline to know the next address, either the following instruction address(branch untaken condition) or branch target address(branch taken condition). In BTB, it has prediction bit which is used to predict whether the branch is taken or untaken.

If the entry is exist in the BTB, which means it is a branch instruction as well as it occurs at least once. Unless it occur for the first time, or it is a non-branch instruction. As if the branch is predicted taken, it outputs the branch target address or else it loads the next sequence instruction address. If the branch is mis-predict taken, ID stage is being flushed and the next sequence address is loaded into the IF stage and if it is mis-predict untaken, BTB is updated and ID stage is being flushed as the next instruction address is to be cleared and the branch target address is to be loaded into the pipeline.

In order to implement a BTB, a fast but small memory is needed in this place, which means cache memory is made as the primary choice in this case. Other than its faster in retrieving the data inside the cache memory, it consume less power than other type of memory design.

2.2 Cache Memory Design

Cache memory is just like RAM, basically it is a small memory and the operation of retrieving and updating is much faster compare to the main memory. The accessing time taken is shorter than main memory. Cache memory is very commonly used in the computer system to speed up the processors' speed as well as its overall performance. Currently, there is a few different level of cache memory which is L1, L2, and L3. It has a smaller capacity but the access time is few cycles lesser than the main memory which give it a big advantage. Frequently used data is stored in the cache memory and the least using one will be flushed or removed to replace with new frequently used data.

2.3 Associativity of Cache

The replacement policy decides the entry of the cache for cache memory. As it has been brought out continuously that cache memory is a small and fast memory, so the data in the cache should be brought in and out of the cache memory continuously. The performance of cache memory mapping function is the key to speed. There are three types of mapping techniques:

- Direct Mapping - Each location has a specific place that held the data
- Set Associative Mapping - Specifies a set of cache lines for each memory block
- Fully Associative Mapping - Entire cache is being searched for an address. All of the entries can choose which block to be entered randomly within the cache.

Direct mapping, it is the easiest implementation. It specifies a single cache line for each memory block. In this mapping technique, part of the memory block address is to be used as index for cache to identify where the data is being held. It does not need any replacement algorithm because the new feeding data will be replacing all the old entries in the cache memory. It is more unpredictable but it gives a better performance.

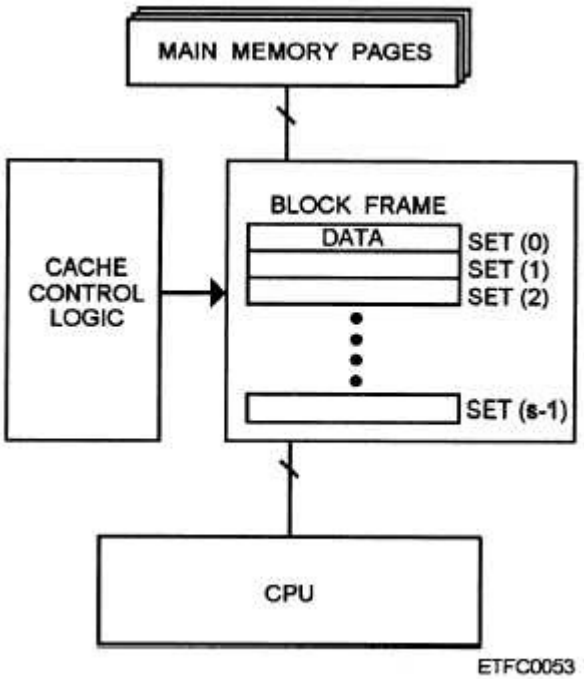


Figure 2.1 Direct Mapping in Cache Memory

Fully associative mapping is more complex, as it is much flexible than the others. It needs a replacement algorithm to selectively choose on the old entry to be removed. The replacement policy is free to choose any entry in the cache to hold a copy. All tag fields is being searched when the processor needs an address, to determine whether the data entry is in the cache memory or not. Each tag line is required to compare withe the desired address with tag field. All the tag fields are being checked parallelly.

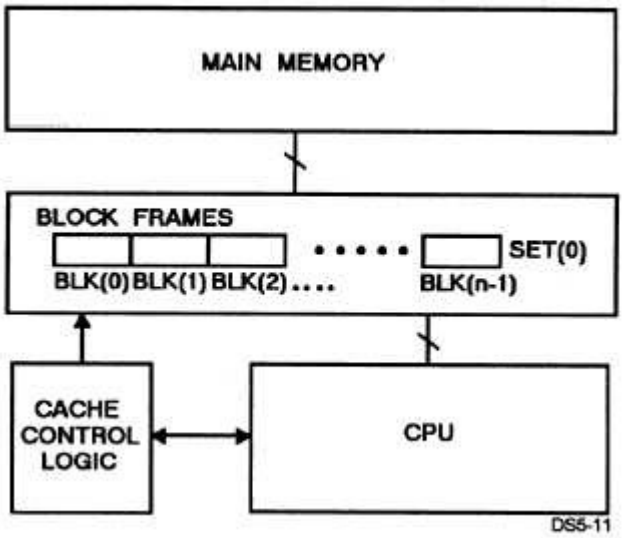


Figure 2.2 Fully Associative Mapping in Cache Memory

Set associative mapping is the combination of both direct mapping technique and fully associative mapping technique. The part that uses the direct mapping technique is that the blocks of data will be mapping into cache as specific set, but they can be in any N-cache block frames within each set, which is the associative mapping technique occur.

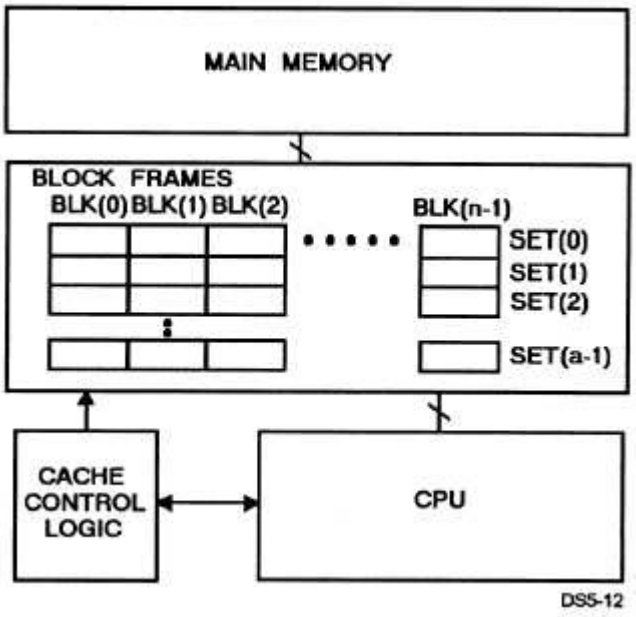


Figure 2.3 Set Associative Mapping in Cache Memory

2.4 Operation of Cache

The main operations of the cache memory are the read operation and write operation. For both of the operation, there are two scenarios which can be happening, a "hit" and a "miss". A "hit" means the data acquire is in the cache memory while a "miss" means the data acquire is not in the cache memory. When a data is needed, and as if the data is in the cache memory, it would be a "READ HIT", while as if the data does not appear in the cache memory, it would be a "READ MISS".

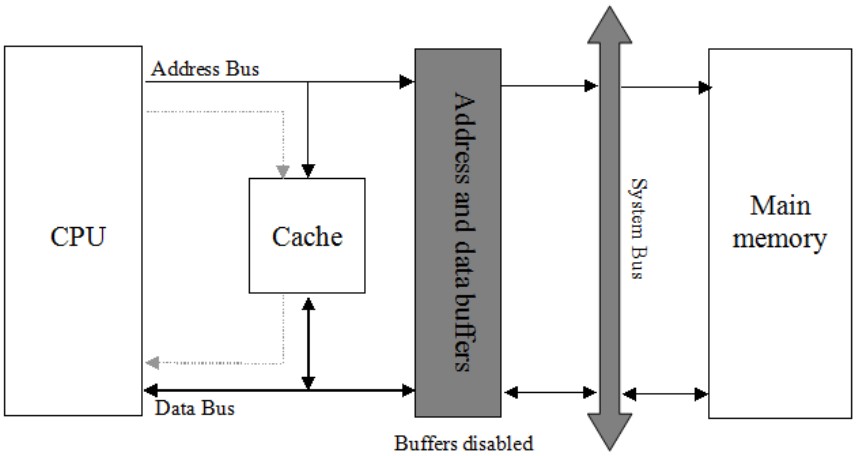


Figure 2.4 Read Hit Operation

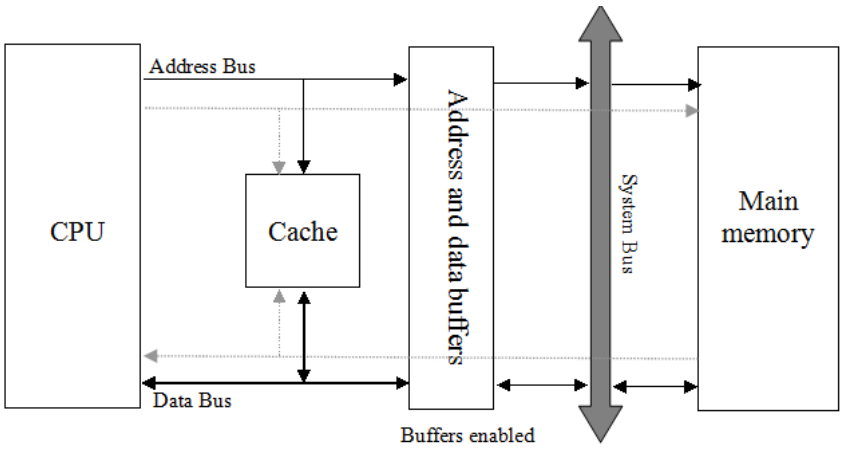


Figure 2.5 Read Miss Operation

On the other hand, the write operation is slightly different from the read operation in both

of the occurring scenarios. "Hit" in the write operation means "WRITE BACK" or "WRITE THROUGH" and "miss" means "WRITE ALLOCATE" or "NO WRITE ALLOCATE". For "WRITE ALLOCATE", the data is first loaded into the cache from main memory and followed by a WRITE HIT action. For "NO WRITE ALLOCATION", the data is not loaded into the cache memory and it directly modified in the main memory only. Similarly, "WRITE THROUGH" operation occurs on both cache memory and main memory where modified data is being written to both of them. "WRITE BACK" is an operation where it writes the modified data into the cache memory only. The status bit of the cache memory (dirty bit) is used to indicate whether it is modified (dirty) or unmodified (clean), so if it is a "clean", it means the data is not written on a "miss".

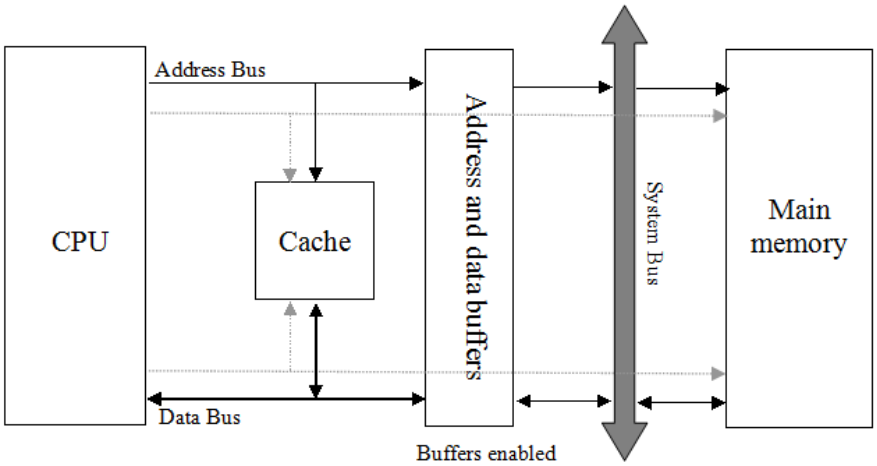


Figure 2.6 Write Through Operation

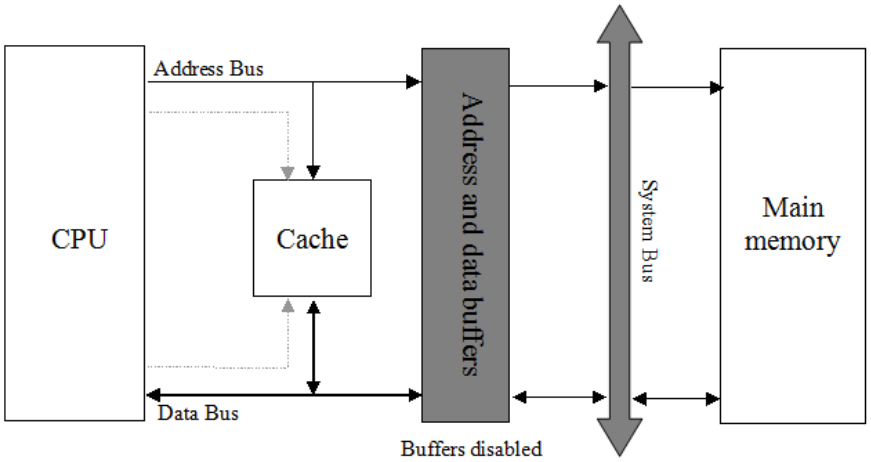


Figure 2.7 Write Back Operation

2.5 Performance of Cache

The performance of cache memory can be increased as the size of the cache increases. Larger the size of the cache memory, the miss rate of cache of different associativity and sizes decrease rapidly. The following graph can show us the result:

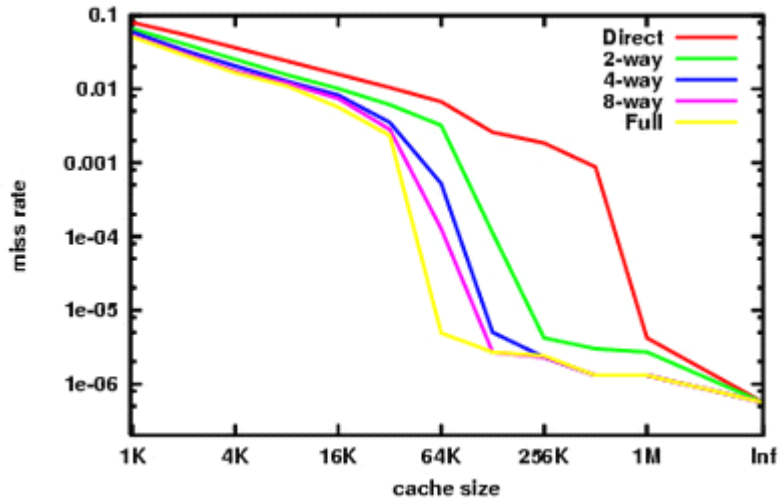


Figure 2.8 Miss Rate versus Cache Size with Different Mapping Techniques

Fully associative cache seems to have the best performance after all but yet it is the most complex one, while the direct mapped cache has the worst performance, yet the simplest between the three mapping techniques. Set associative cache's performance lies between both of the cache above. As the N value increases, the performance of the set associative

cache increases and it tends to be having the same result as the fully associative cache.

2.6 Cache Miss

Cache memory is created to improve the performance of the CPU, so in order to do so, cache memory should have a high "hit" rate. There are some misses that its unavoidable and it can be categories to three different misses (known as Three Cs):

- Compulsory Miss - It occurs when the very first access to a block that is unavailable in the cache and must be brought into the cache. It also can be called as the first reference miss or cold-start miss. The associativity and size of cache do not make any difference to the compulsory miss, but if the cache has the pre-fetching function, compulsory miss would be reduced.
- Capacity Miss - It occurs due to the finite size of the cache memory. This occurs because the cache memory size is small and it is unable to contain all the blocks needed during the execution of program as the blocks are being discarded and later retrieved.
- Conflict Miss - It occurs when there are multiple of memory accesses mapped into the same index set in the cache for the fully associative mapping technique. It also can be known as the interfere miss or collision miss.

Chapter 3 - Methodology and Technology Involved

3.1 Design Methodology

Design methodology is the method of development of system design. There are a few guideline to be fulfill:

- Correct functionality
- Satisfaction of performance and power goals
- Catching bugs early
- Good documentation

In this project, top-down design methodology will be used. This methodology keeps all level of the hierarchy with its own functionality. This methodology also provides advantages such as functionality, performance, power consumption and area of silicon.

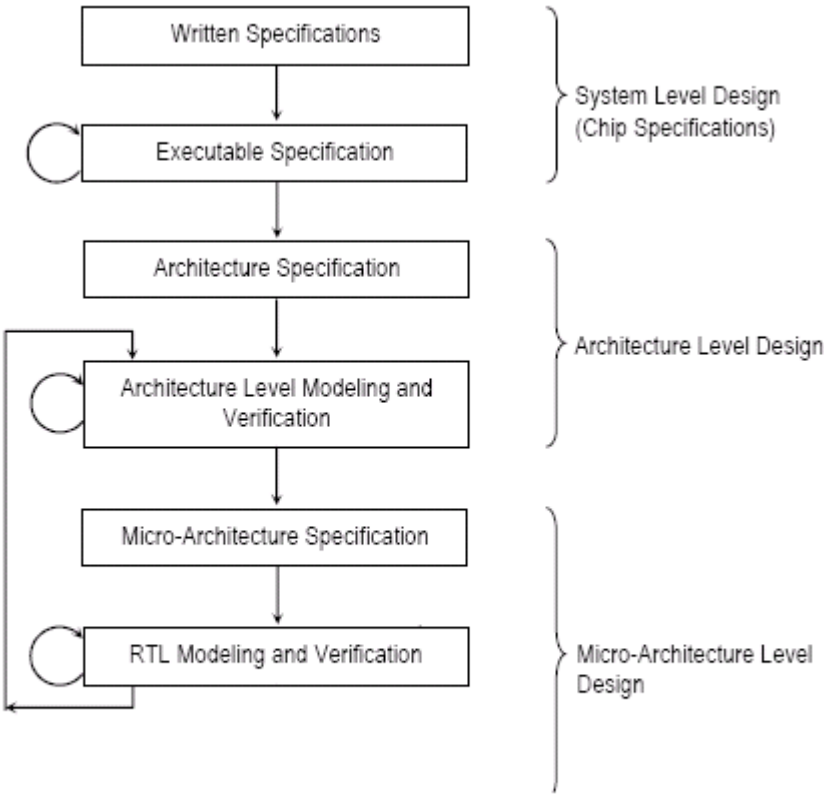


Figure 3.1 General Design Flow without Physical Design and Logic Synthesis

3.1.1 System Level Design

System level design is the design of chip specification. The system level design can be sorted as two categories:

- Written Specification: English written specification of function, performance and time, cost of design included as well. Furthermore, function specification, verification specification, development plan and packaging specification are included as well.
- Executable Specification: High level language is used to program according to the design features and functionalities. The language here refers to Verilog, VHDL and etc..

3.1.2 Micro-architecture Design

Micro-architecture design is the development of RTL design of the system. There are a few information is included, such as:

- Overview of functional description
- I/O pin description
- I/O timing requirements
- Function table
- Finite-state machine (FSM) and Algorithm-state machine (ASM)
- Test plan

RTL modeling with programming language can be done after the development of micro-architecture specification. The modeling of design can be done by software and verification can be done by setting test plan, timing and functionality verifications.

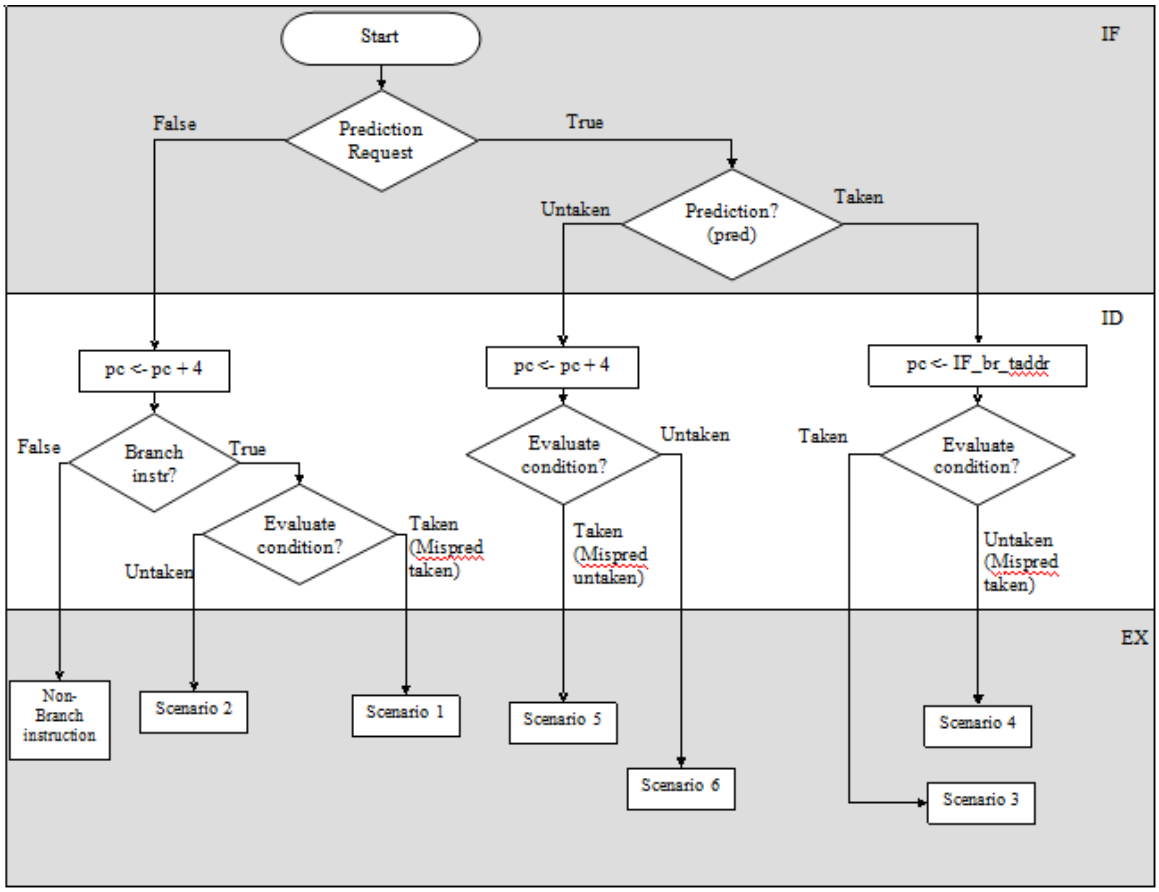


Figure 3.2 Data flow of the Branch Prediction Block

In the 2-bit scheme branch prediction, it need a "wrong" to strengthen the correct prediction. Either taken or untaken, it needs to be "11" or "00" as a "strong" confirmation of the condition.

From the data flow above, if the current pc is not appear in the buffer entry, it will be a false for the prediction request and also "Read miss" of the BTB. We then further inspect whether it is a branch instruction or a non-branch instruction. If it is a non-branch instruction, IF stage will just feed the next PC value and no entry is added into the BTB. On the other hand, if it is a branch instruction, the inspection must go further deep. If it is a branch equal instruction (beq), it suppose to be taken, but resulting as untaken in the BPB as it is not appear in the BTB, so scenario 1 is to be executed. For scenario 1, the ID stage will be flushed, pc will be given the branch target address. The BTB will be updated,

prediction bit will be a "weakly taken" (10) and the LRU state of all blocks with the same index will be updated. If it is not a equal, Scenario 2 is executed, which means branch is untaken, the branch is then correctly predicted, it will continue to execute the next sequential address in the IF stage.

If the prediction request is true, which means the current pc is an entry in the BTB, it will be a "Read Hit". If the prediction is predicted as taken, in ID stage, it detected a branch and it is evaluated as taken, the prediction is correct. As a result, Scenario 3 is to be executed. No new entry added as it exists in the BTB, the prediction bit is changed to "Strongly taken" and the LRU state of the same index is being updated. In this case, we need not to flush the ID stage.

If the prediction request is true, at ID stage, it detects a branch and evaluated as untaken, in this case, the prediction bit will be changed and the LRU state bit need to be update as well, but in this case, flushing of ID stage is needed as it is a misprediction case. It is Scenario 4.

For Scenario 5, it will occur when the prediction request is true, at ID stage, it detected a branch and the condition is evaluated as taken, which resulted as misprediction. The prediction bit of the block and the LRU state of the same index will be updated and due to misprediction, ID stage will be flushed.

Last but not the least, for scenario 6, it is executed when the prediction request is true, it detected a branch at ID stage and it is untaken, in this case, it has no misprediction. The prediction will be updated as "Strongly untaken" and the LRU state of the same index will be updated.

For reading of the BTB, it uses pc at the IF stage as tag and index. The tag here uses to find which cache it is stored and index uses as a guide to the block which has the data we needed. The tag bit here takes IF.pc[31:12] and index bit takes IF.pc[13:2].

For the update of BTB, it uses the pc at the ID stage as the tag and index of the entry. The tag bit, which is ID.pc[31:12] will be store in as part of the entry as well. The index, as we uses 4X1K cache, it consists of 10 bits, ID.pc[13:2] as a guide of which space that the entry should store the data to. In both cases, the last two bits of pc is ignored.

3.2 Design Tools

The main development tool used in this project is ModelSim SE 10.2b. It is a simulation and debugging tool to run this project. It support the HDLs like Verilog and VHDL as well as RTL simulation and gate-level design. It is equipped with graphical user interface (GUI) that shorten design time and keep debugging and simulation easy. With the aid of GUI, errors and warnings can be easily trace back during compilation and simulation. Tutorials and documentations are provided by ModelSim as well as technical support to users.

3.3 Design Language

The design language being used here is HDL (Hardware Description Language). The HDL used here is Verilog. Verilog is standardized as IEEE 1364, used to model electronic systems. It is commonly used to design and verify digital circuitries at the RTL.

Chapter 4 - System Specifications

4.1 Features

Chip level design: RISC32 processor

	RISC32 processor
Dummy Instruction Cache (KB)	16
Dummy Data Cache (KB)	16
Data width (bits)	32
Instruction width (bits)	32
General Purpose Register	32
Special Purpose Register	HILO, PC
Co-Processor Register	32
Pipelined Stage	5
Hazard Handling	Yes
Interlock Handling	Yes
Interrupt Handling	Yes
Data Dependency Forwarding	Yes
Branch Prediction	**Dynamic – 2bits scheme**
Branch Target Buffer (KB)	**4**
Multiplication (size of multiplier and multiplicand)	yes – 32 bits
Branch Delay Slot	Not supported
Instruction supported	41

Table 4.1 RISC32 Features

4.2 Naming Convention

Module - [lvl]_[mod. name]

Instantiation - [lvl]_[abbr. mod. name(3)]

Pin - [lvl]_[abbr. mod. name(3)]_[type]_[pin name]

- [lvl]_[abbr. mod. name(3)]_[type]_[stage]_[pin name]

Abbreviation:

	Description	Case	Available	Remark
lvl	Level	Upper	C: Chip U: Unit B: Block	
mod. name	Module name	Upper all	any	
abbr. mod. name	Abbreviated module name	Upper all	any	
(n)	Max n characters	N/A	N/A	
type	Pin type	Lower all	o: output i: input r: register w: wire f: function	
stage	Stage name	Upper all	IF, ID, EX, MEM, WB	
pin name	Pin name	Upper first	any	Several word separate by "_"

Table 4.2 Naming Convention

Chapter 5 - Micro-architecture Specification of Branch Prediction Block

5.1 Branch Prediction Block

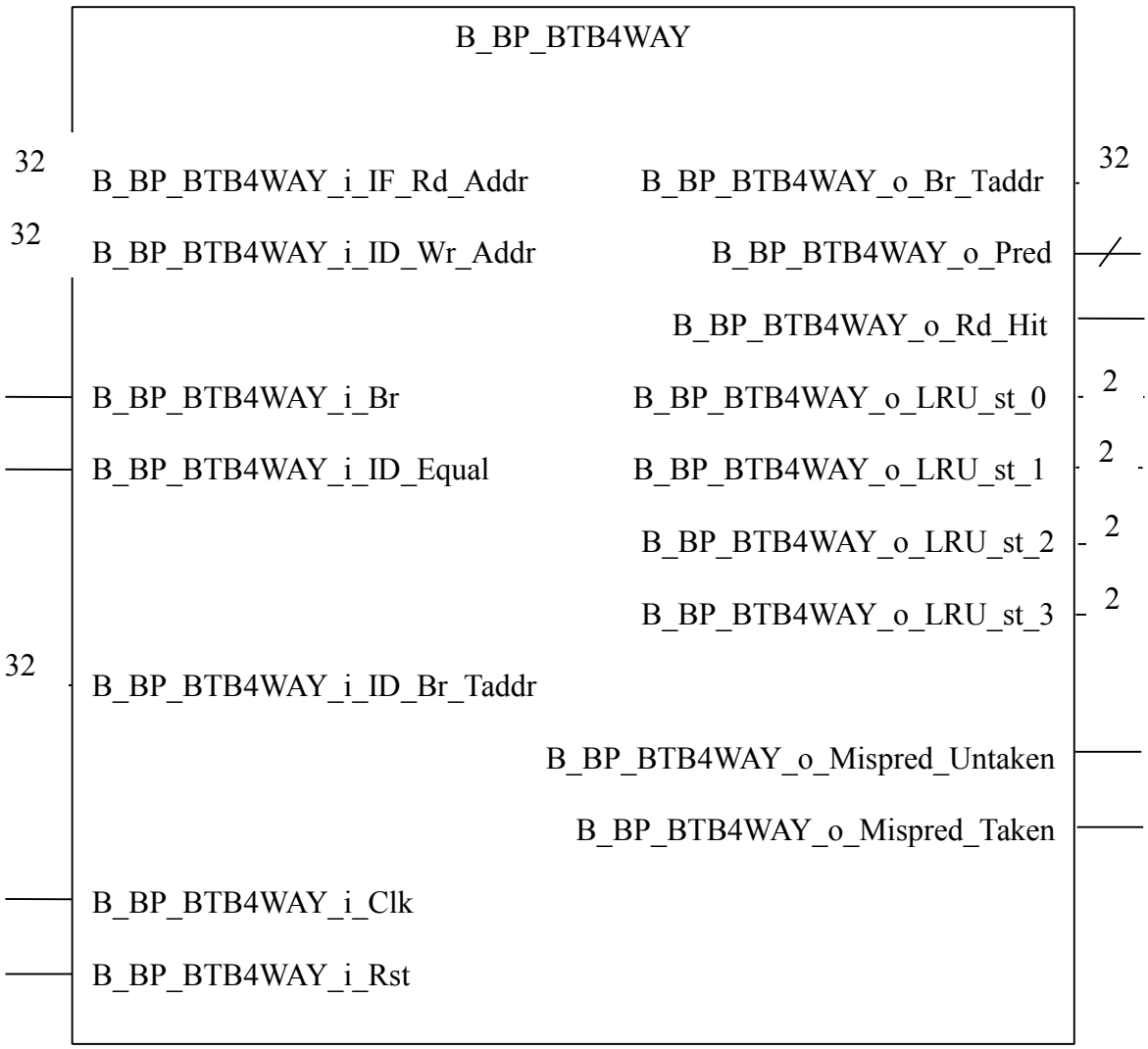


Figure 5.1 Branch Prediction Block Diagram

5.2 I/O Pin Descriptions

Block Input Pins Description

Pin Name: B_BP_BTB4WAY_i_IF_Rd_Addr [31:0]	Source -> Destination: Datapath Unit(IF) -> Datapath Unit(BPB)	Registered: No
Pin Function: Fetch the address as the index and tag used to find the block inside the buffer.		
Pin Name: B_BP_BTB4WAY_i_ID_Wr_Addr[31:0]	Source -> Destination: Datapath Unit(ID) -> Datapath Unit(BPB)	Registered: No
Pin Function: Fetch the address as the index and tag used to update the block inside the buffer.		
Pin Name: B_BP_BTB4WAY_i_Br	Source -> Destination: Main Control Unit -> Datapath Unit(BPB)	Registered: No
Pin Function: This is a flag to indicate the current instruction is a branch instruction.		
Pin Name: B_BP_BTB4WAY_i_ID_Equal	Source -> Destination: Datapath Unit(ID) -> Datapath Unit(BPB)	Registered: No
Pin Function: This is a flag to indicate the current instruction is a branch instruction.		
Pin Name: B_BP_BTB4WAY_i_ID_Br_Taddr[31:0]	Source -> Destination: Datapath Unit(ID) -> Datapath Unit(BPB)	Registered: No
Pin Function: Address to be stored in the buffer (Update)		
Pin Name: B_BP_BTB4WAY_i_Clk	Source -> Destination: Micro-processor -> Datapath Unit(BPB)	Registered: No

Pin Function: Synchronous system clock		
Pin Name: B_BP_BTB4WAY_i_Rst	Source -> Destination: Micro-processor -> Datapath Unit(BPB)	Registered: No
Pin Function: System Reset Control 0: Reset disabled 1: Rest asserted		

Table 5.1 BPB Input Pin Description

Block Output Pin Description

Pin Name: B_BP_BTB4WAY_o_Br_Taddr[31:0]	Source -> Destination: Datapath Unit(BPB) -> Datapath Unit(IF)	Registered: Yes
Pin Function: Output the Branch Target Address from the buffer.		
Pin Name: B_BP_BTB4WAY_o_Pred	Source -> Destination: Datapath Unit(BPB) -> Datapath Unit(IF)	Registered: Yes
Pin Function: Output the Prediction value from the buffer		
Pin Name: B_BP_BTB4WAY_o_Rd_Hit	Source -> Destination: Datapath Unit(BPB) -> Datapath Unit(IF)	Registered: Yes
Pin Function: Output the result of searching from the buffer. 0: Read Miss - The address is not a branch address / The address is not available in the buffer. 1: Read Hit - The address is found in the buffer		
Pin Name:	Source -> Destination:	Registered:

B_BP_BTB4WAY_o_LRU _st_0[1:0]	Datapath Unit(BPB) -> Datapath Unit(IF)	Yes
Pin Function: The LRU state of the buffer (Set 0). 0: Least recently used 1: Next least recently used 2: Further least recently used 3: Recently used		
Pin Name: B_BP_BTB4WAY_o_LRU _st_1[1:0]	Source -> Destination: Datapath Unit(BPB) -> Datapath Unit(IF)	Registered: Yes
Pin Function: The LRU state of the buffer (Set 1). 0: Least recently used 1: Next least recently used 2: Further least recently used 3: Recently used		
Pin Name: B_BP_BTB4WAY_o_LRU _st_2[1:0]	Source -> Destination: Datapath Unit(BPB) -> Datapath Unit(IF)	Registered: Yes
Pin Function: The LRU state of the buffer (Set 2). 0: Least recently used 1: Next least recently used 2: Further least recently used 3: Recently used		
Pin Name: B_BP_BTB4WAY_o_LRU _st_3[1:0]	Source -> Destination: Datapath Unit(BPB) -> Datapath Unit(IF)	Registered: Yes
Pin Function: The LRU state of the buffer (Set 3).		

<table>
<tr><td colspan="3">0: Least recently used
1: Next least recently used
2: Further least recently used
3: Recently used</td></tr>
<tr><td>Pin Name:
B_BP_BTB4WAY_o_Mispred_Untaken</td><td>Source -> Destination:
Datapath Unit(BPB) -> Datapath Unit(IF)</td><td>Registered:
Yes</td></tr>
<tr><td colspan="3">Pin Function:
To state whether the branch is a correct or wrong untaken prediction.
0: Not mispredict untaken
1: Mispredict untaken</td></tr>
<tr><td>Pin Name:
B_BP_BTB4WAY_o_Mispred_Taken</td><td>Source -> Destination:
Datapath Unit(BPB) -> Datapath Unit(IF)</td><td>Registered:
Yes</td></tr>
<tr><td colspan="3">Pin Function:
To state whether the branch is a correct or wrong taken prediction.
0: Not mispredict taken
1: Mispredict taken</td></tr>
</table>

Table 5.2 BPB Output Pin Description

5.3 Prediction Transition

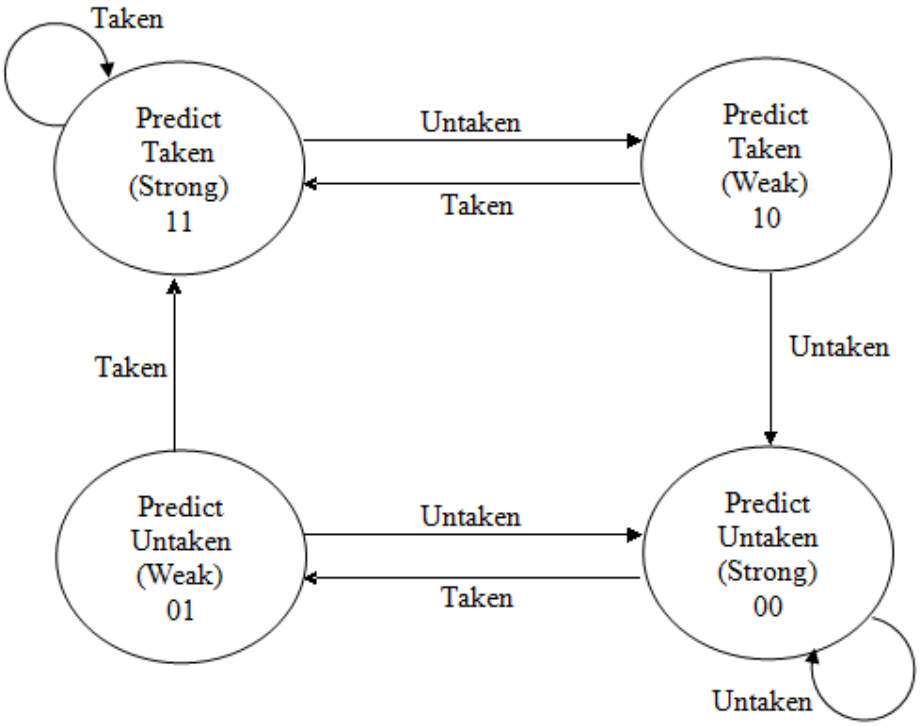


Figure 5.2 Prediction Transition

5.4 Contents of BTB

Buffer[56]	Buffer [55:36]	Buffer [35:4]	Buffer [3:2]	Buffer [1:0]
Valid content	Tag bit [19:0]	Branch Target Address [31:0]	Prediction bit [1:0]	LRU state bit [1:0]

Table 5.3 Branch Target Buffer

5.5 Branch Target Table

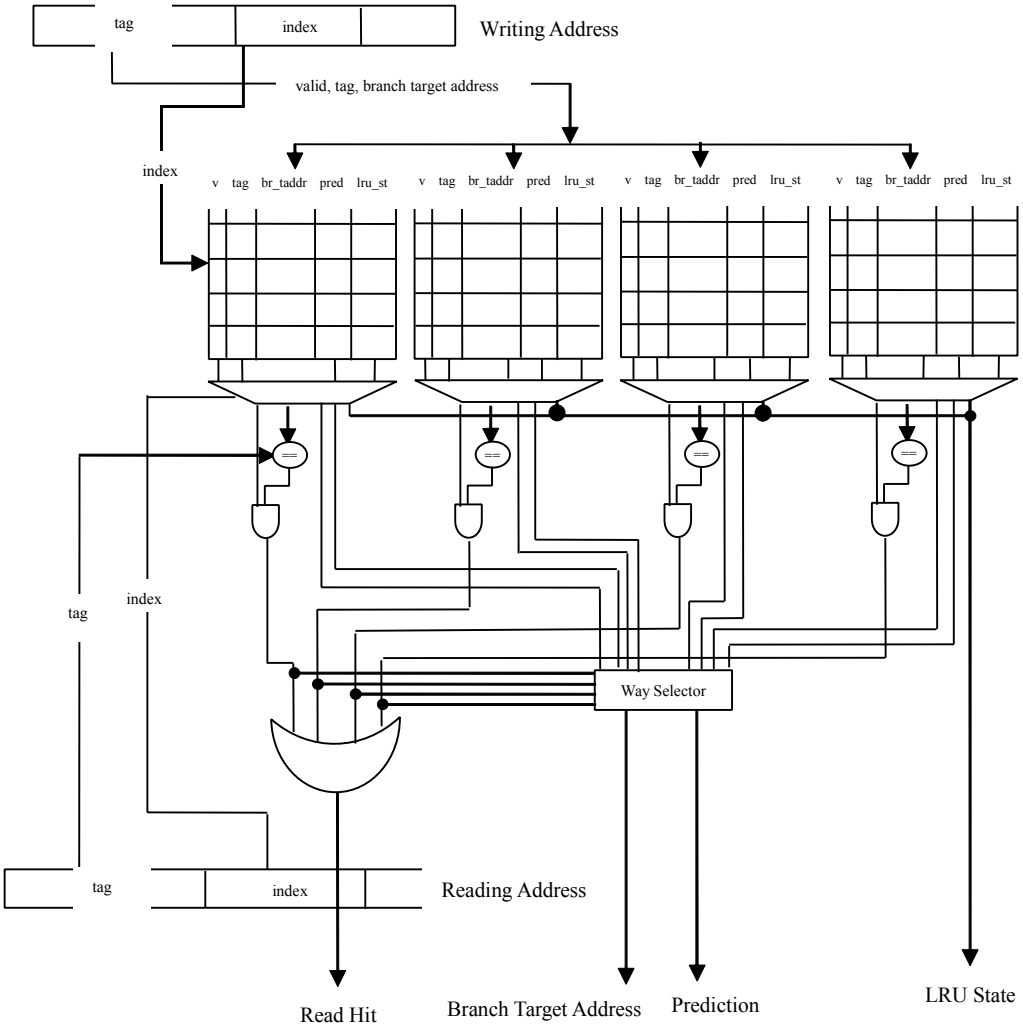


Figure 5.3 Implemented Branch Prediction Table

5.6 Branch Target Dataflow

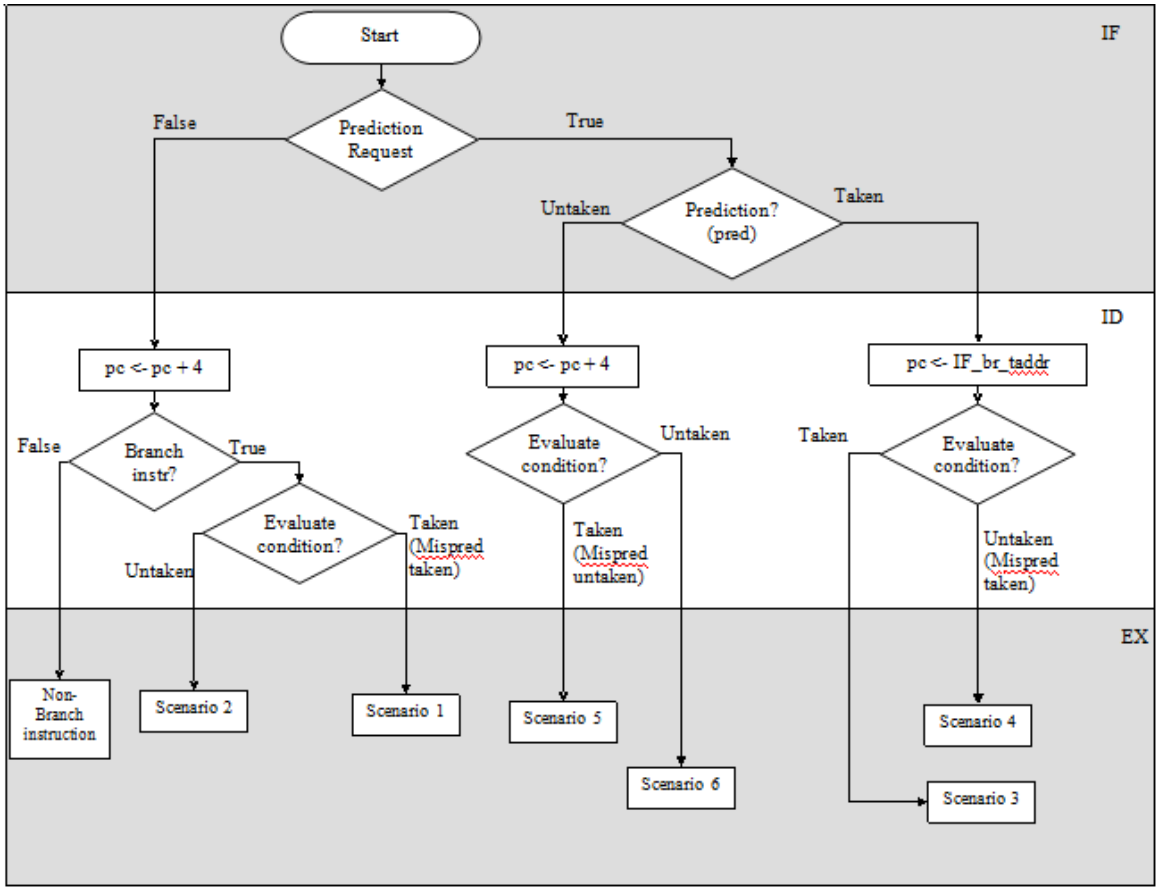


Figure 5.4 Dataflow of the BTB

5.7 Test Plan

	Descriptions	Expected Result
1	Reset Reset on the block is enabled.	Valid bit = 0, Prediction bit = 2'b10, LRU_state_0 = 2'b00, LRU_state_1 = 2'b01, LRU_state_2 = 2'b10, LRU_state_3 = 2'b11.
2	Scenario 1 Buffer Read Miss Update BTB due to read miss	Valid bit = 1, Prediction bit = 2'b10, *LRU_state = 2'b11
3	Scenario 2 Buffer Read Miss The instruction is not a branch instruction	No changes on BTB
4	Scenario 3 Buffer Read Hit - Predict taken, No mispredict Update prediction bit and LRU state	Prediction bit = 2'b11, *LRU_state = 2'b11
5	Scenario 4 Buffer Read Hit - Predict taken, Mispredict Update prediction bit and LRU state	Prediction bit decrease, *LRU_state = 2'b11
6	Scenario 5 Buffer Read Hit - Predict untaken, Mispredict Update prediction bit and LRU state	Prediction bit increase, *LRU_state = 2'b11
7	Scenario 6 Buffer Read Hit - Prediction untaken, No mispredict Update prediction bit and LRU state	rediction bit = 2'b00, *LRU_state = 2'b11

Table 5.4 Test Plan of BPB

*LRU_state stated above is only for the current entry in the BTB, others are updated too,

but the value will not be the same, so it will not be stated here.

Chapter 6 - Verification Specification

6.1 Test Program for BPB

In order to test out the functionality of the BPB, a test bench program is written to test out the hardware model. Test bench is a virtual environment used to verify the correctness of the design.

In test bench, the input is to be fed into the DUT to provide the output according to the model. As the verification is done inside the computer, it can avoid the mistake and wastage if all model need to fabricate before testing.

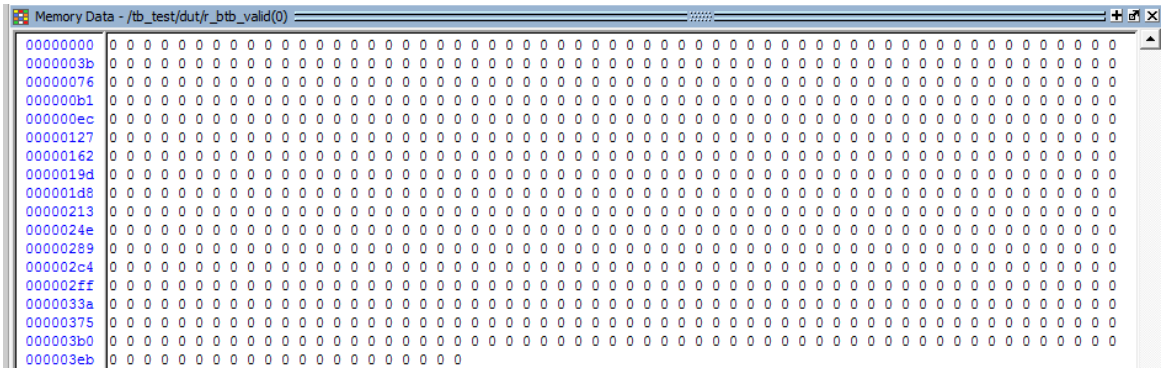
For the testing of BPB, ModelSim SE 10.2b is used as a tool to program the test bench and running the result simulation. The development language using here is also the HDLs, Verilog. The test bench is program according to the test plan on the given model.

6.2 Verification Result

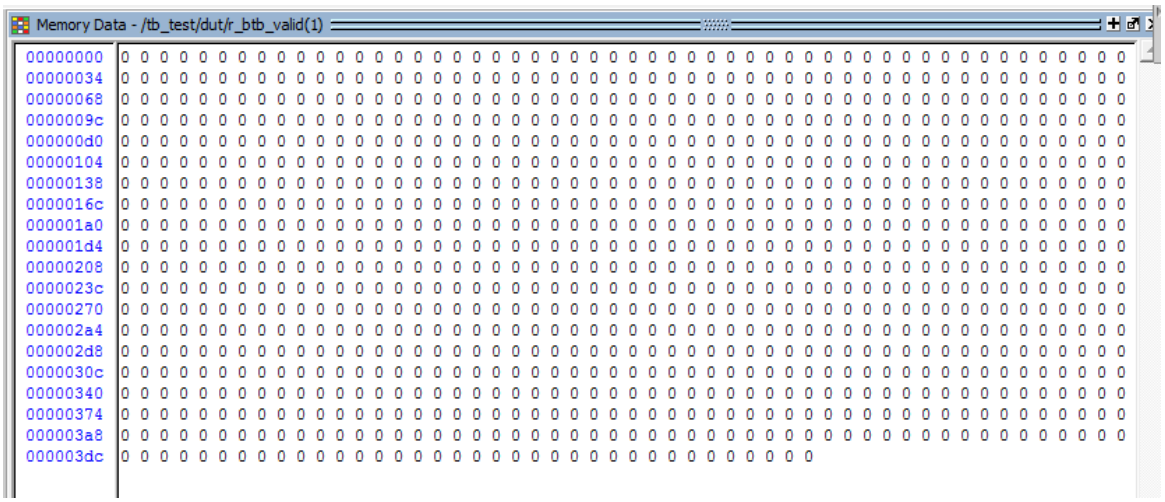
1. Reset of BPB

B_BP_BTB4WAY_i_Rst <= 1'b1;

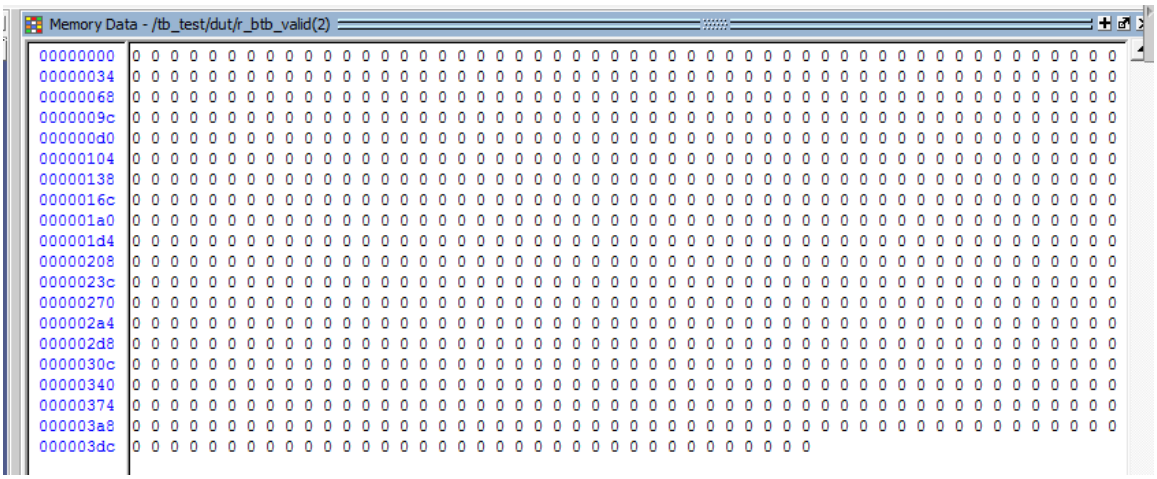
- Valid bit Way 0 in BTB



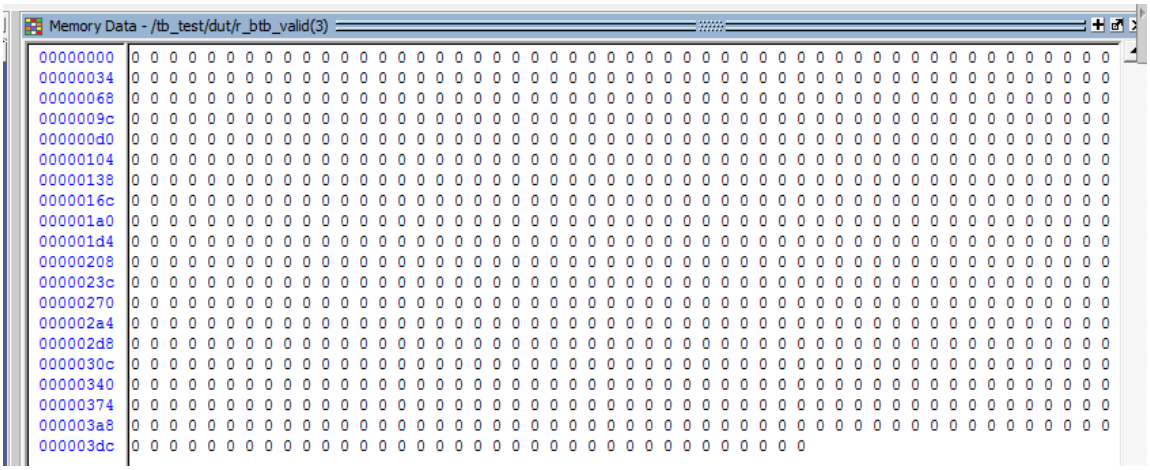
- Valid bit Way 1 in BTB



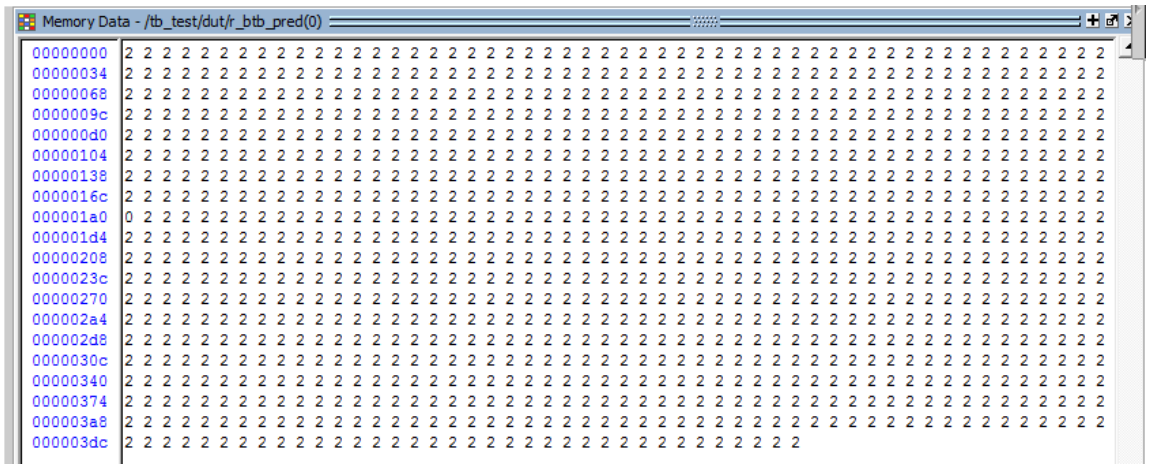
- Valid bit Way 2 in BTB



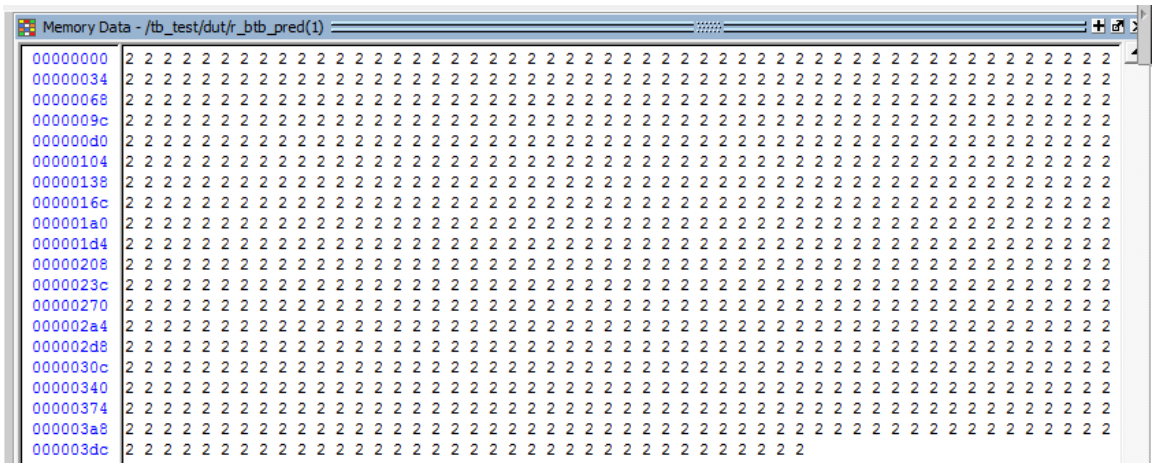
- Valid bit Way 3 in BTB



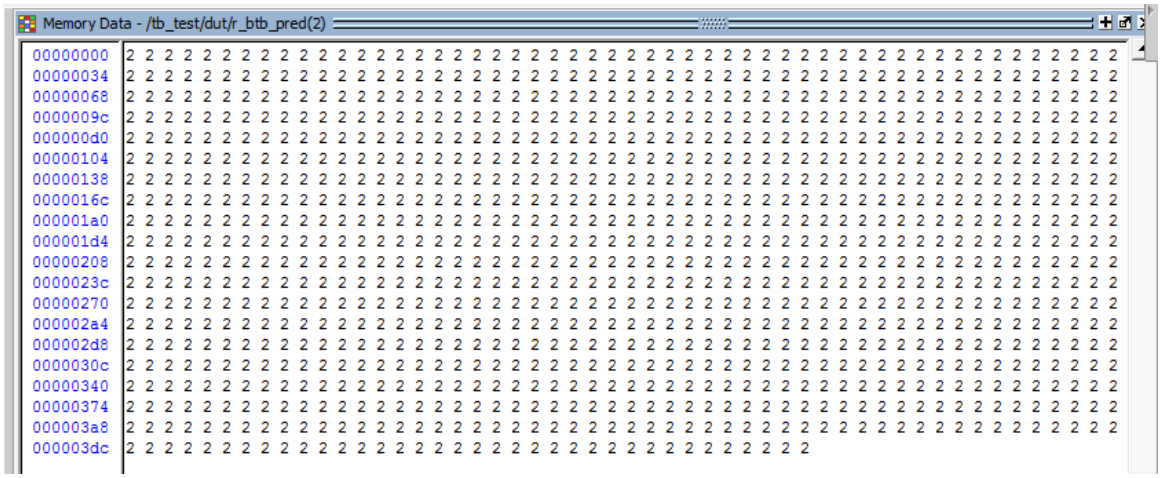
- Prediction bit Way 0 in BTB



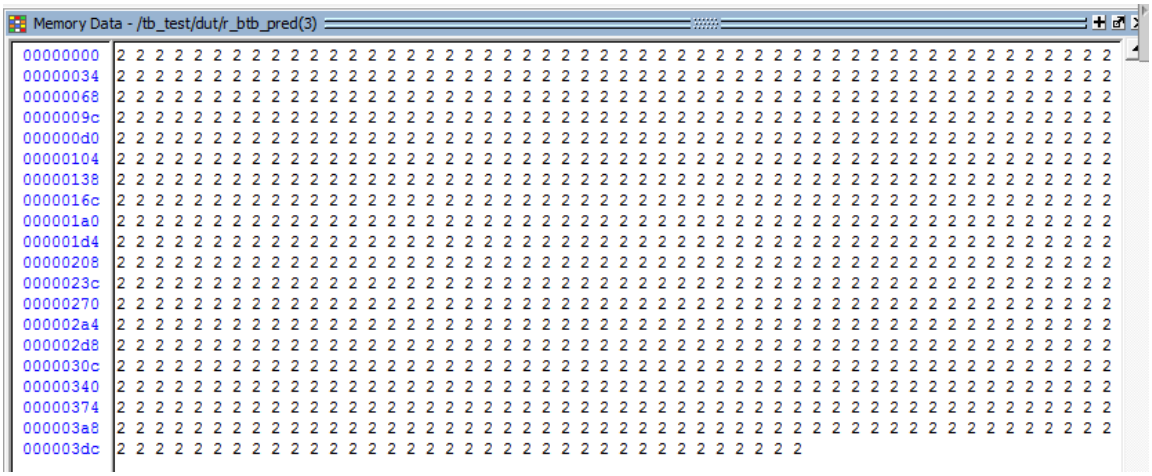
- Prediction bit Way 1 in BTB



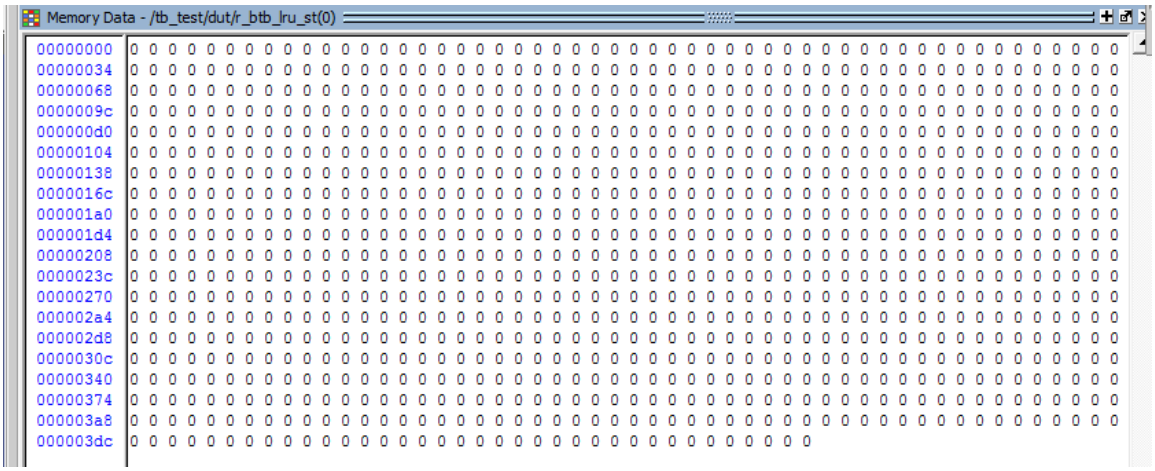
- Prediction bit Way 2 in BTB



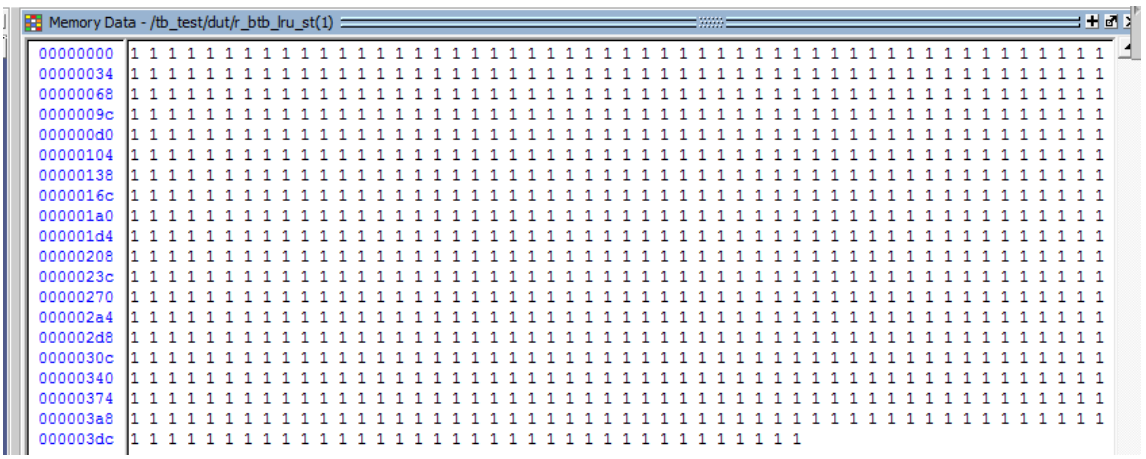
- Prediction bit Way 3 in BTB



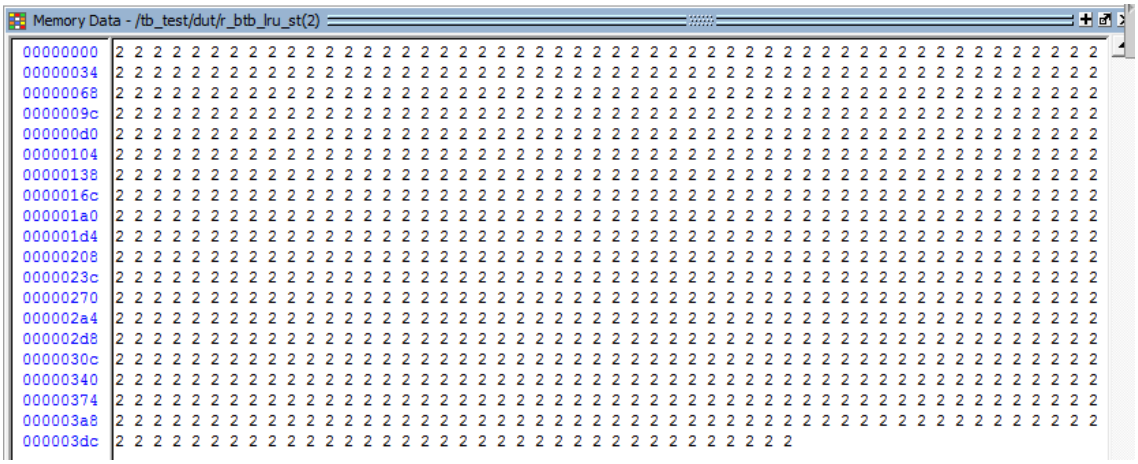
- LRU state bit Way 0 in BTB



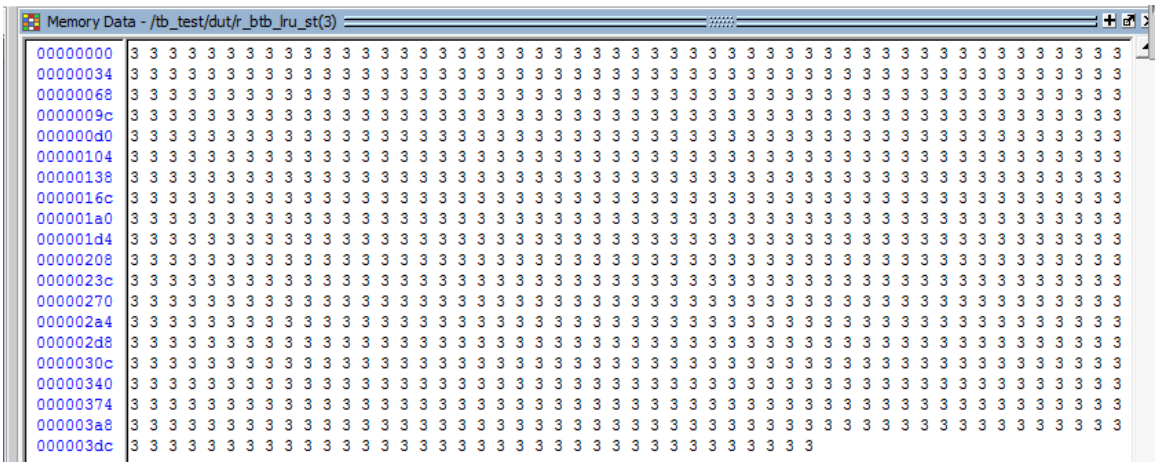
- LRU state bit Way 1 in BTB



- LRU state bit Way 2 in BTB



- LRU state Way 3in BTB



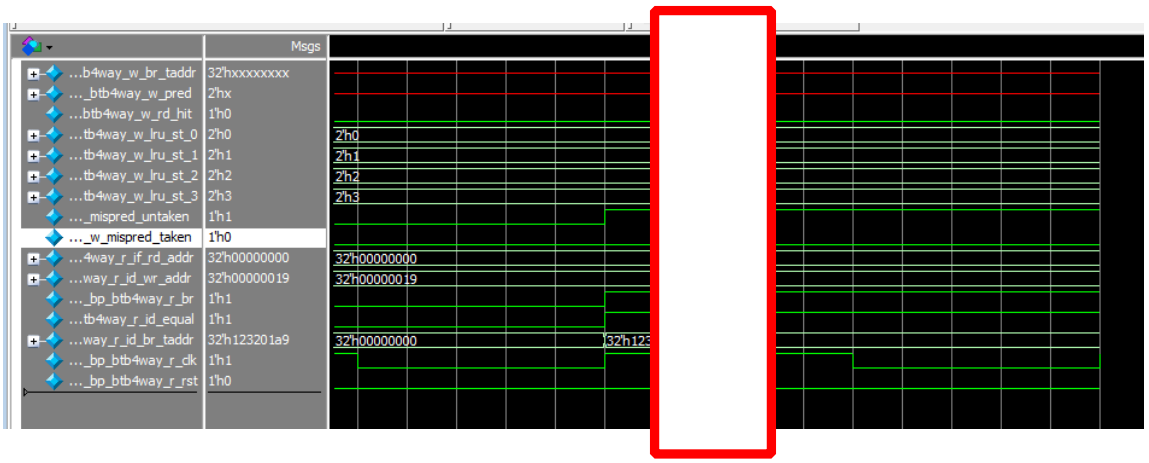
2. Scenario 1 - Read Miss

B_BP_BTB4WAY_i_Br <= 1'b1;

B_BP_BTB4WAY_i_ID_Equal <= 1'b1;

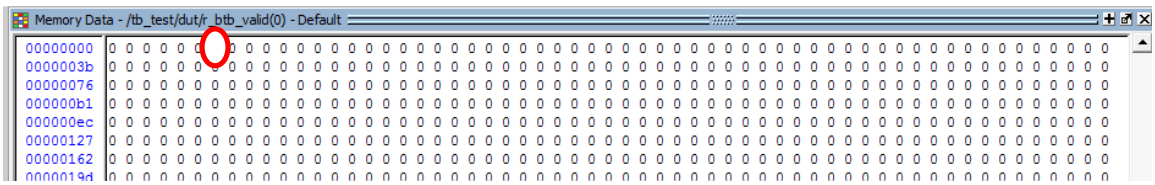
B_BP_BTB4WAY_i_ID_Wr_Addr <= 32'h0000_0018; -> Index = 006, Tag = 00000

B_BP_BTB4WAY_i_ID_Br_Addr <= 32'h1232_01a9;

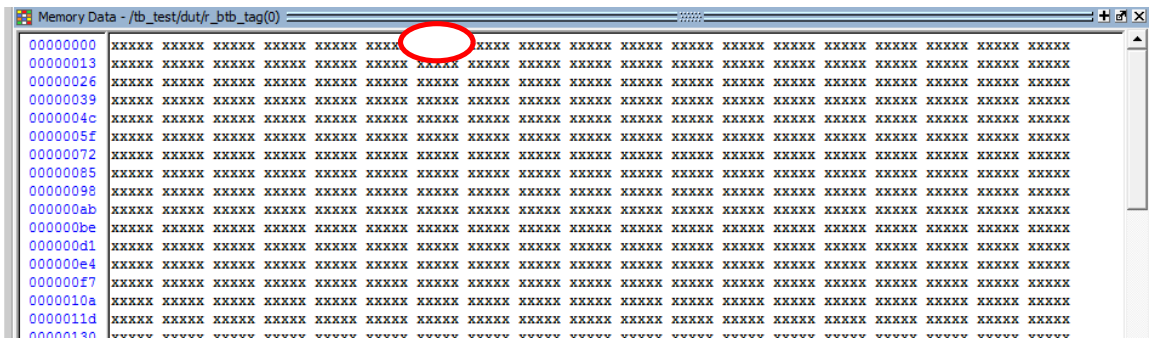


It resulted as a mispredict untaken, and the BTB is updated a clock cycle after the input is pass in. (Due to using value at ID stage not IF stage)

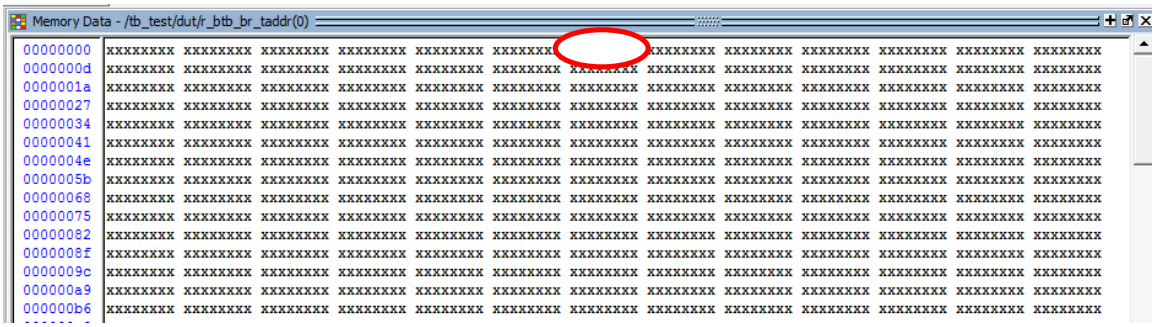
- Valid bit of BTB



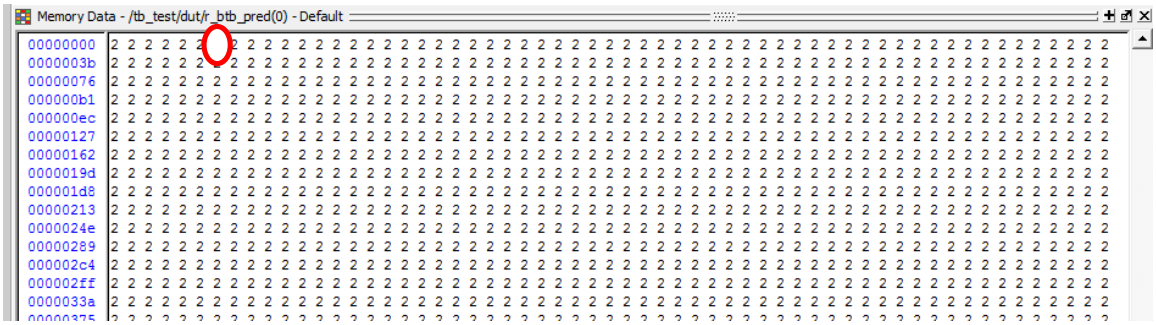
- Tag bit of BTB



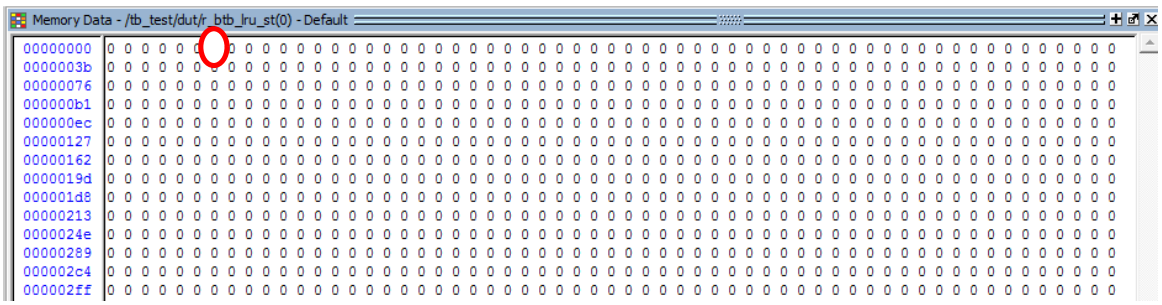
- Branch Target Address



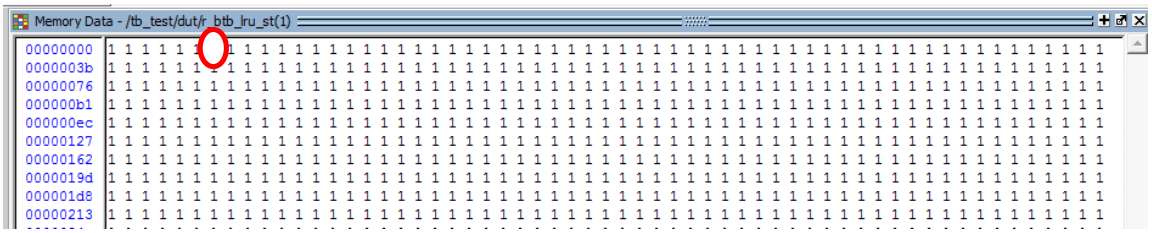
- Prediction bit in BTB



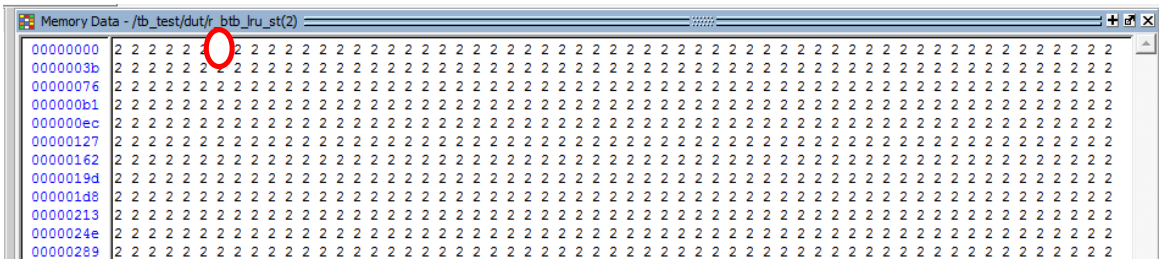
- LRU state at the way it store - Way 0



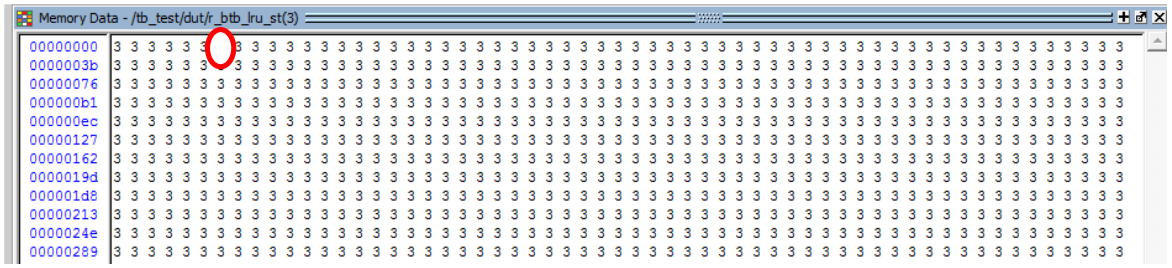
- LRU state at Way 1



- LRU state at Way 2



- LRU state at Way 3

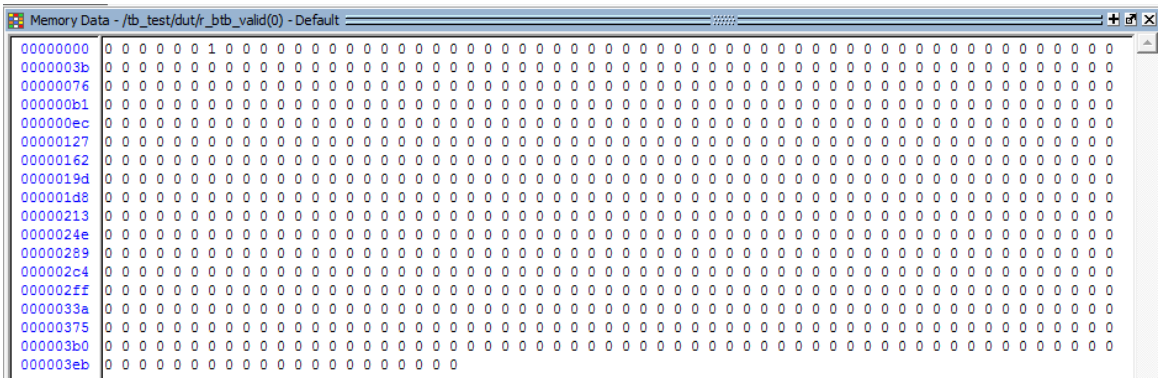


3. Scenario 2 - Read Miss

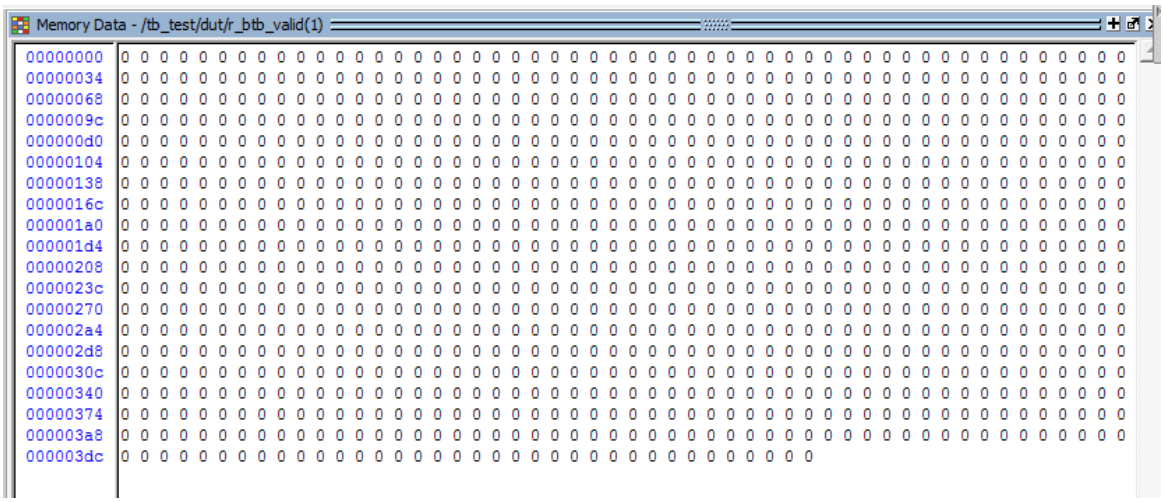
B_BP_BTB4WAY_i_Br <= 1'b1;

B_BP_BTB4WAY_i_ID_Equal <= 1'b0;

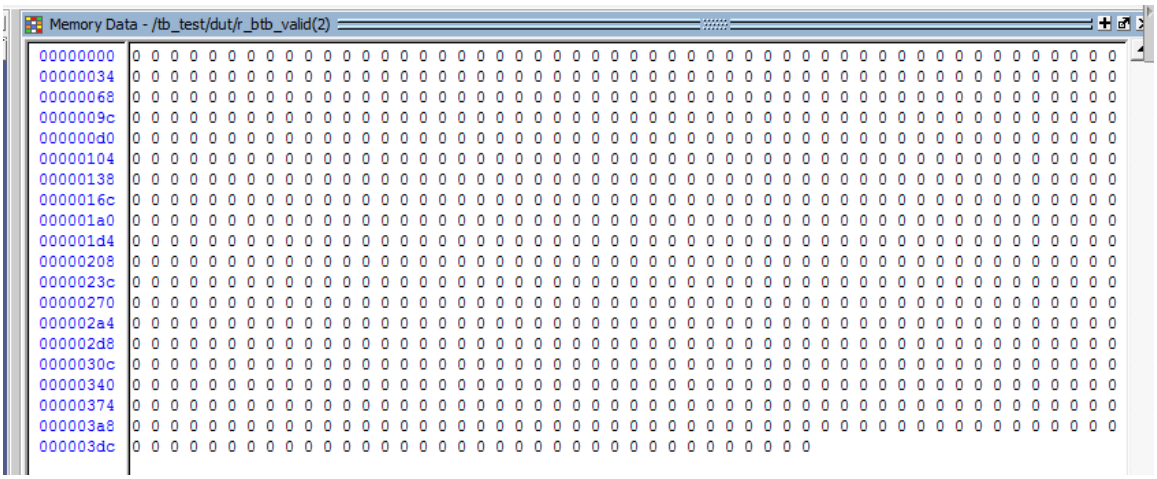
- Valid bit Way 0 of BTB



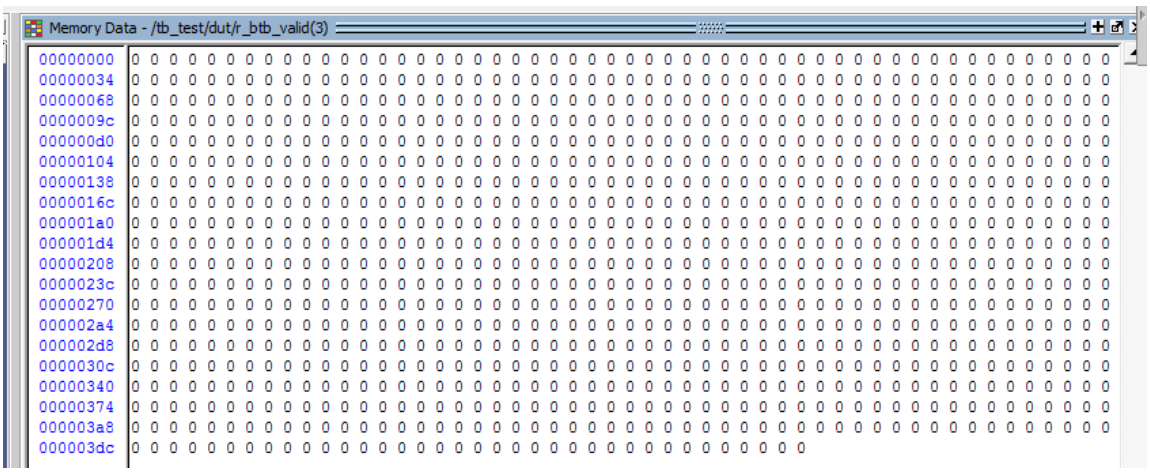
- Valid bit Way 1 in BTB



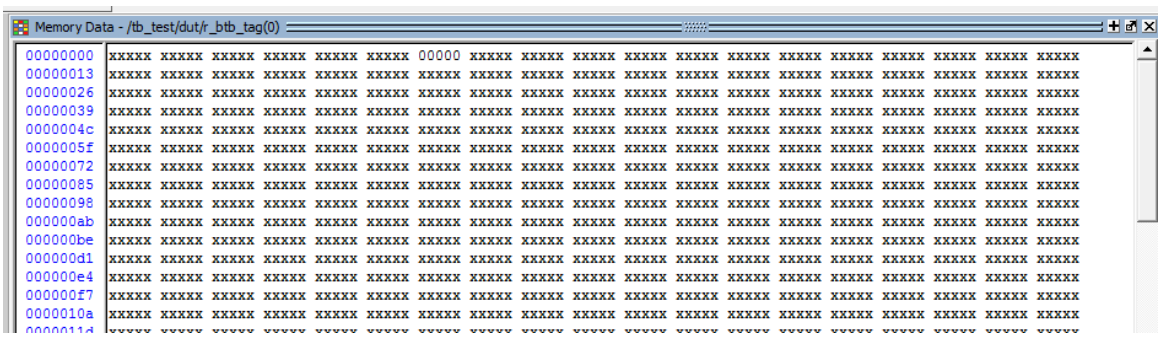
- Valid bit Way 2 in BTB



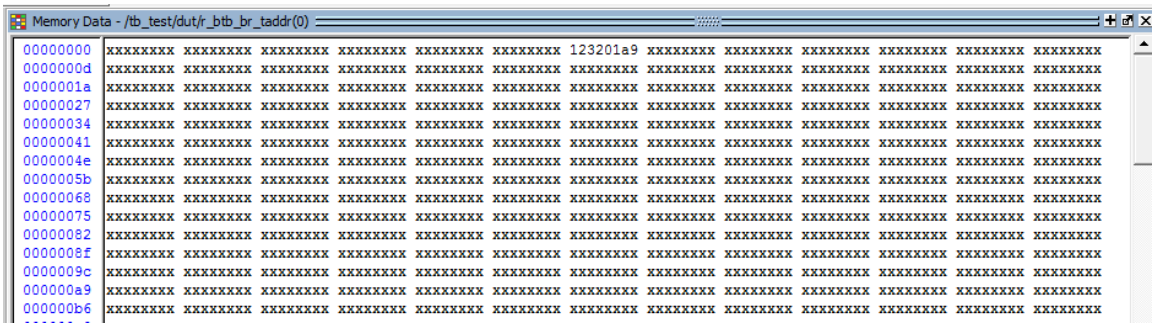
- Valid bit Way 3 in BTB



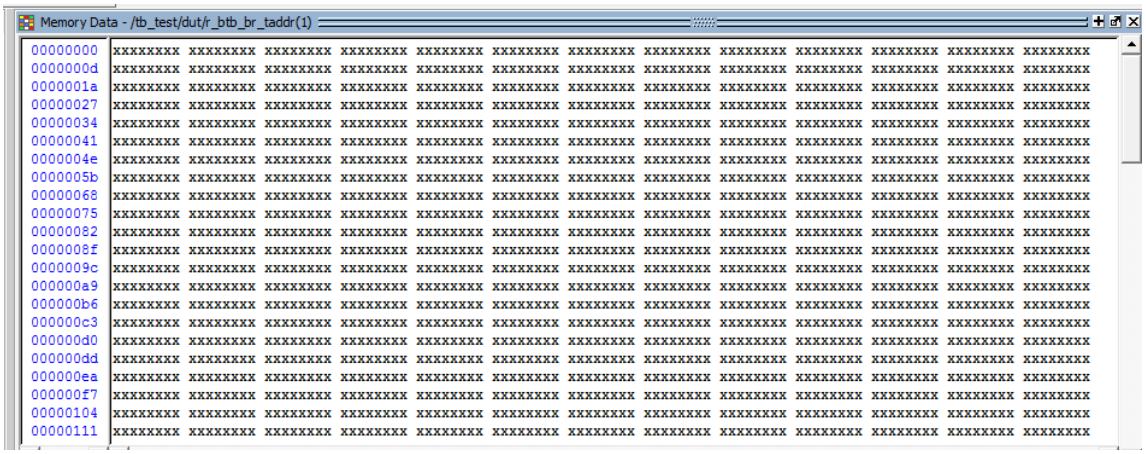
- Tag bit of BTB



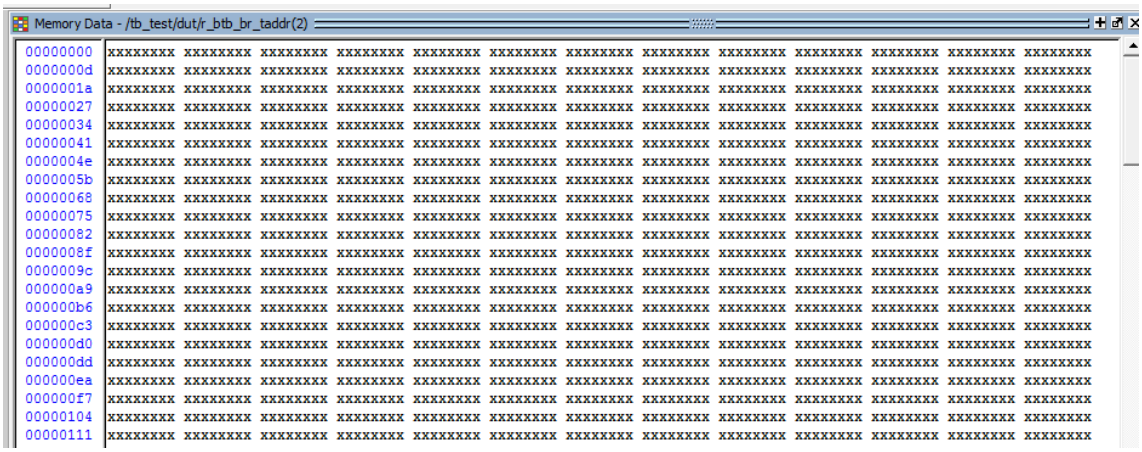
- Branch Target Address Way 0



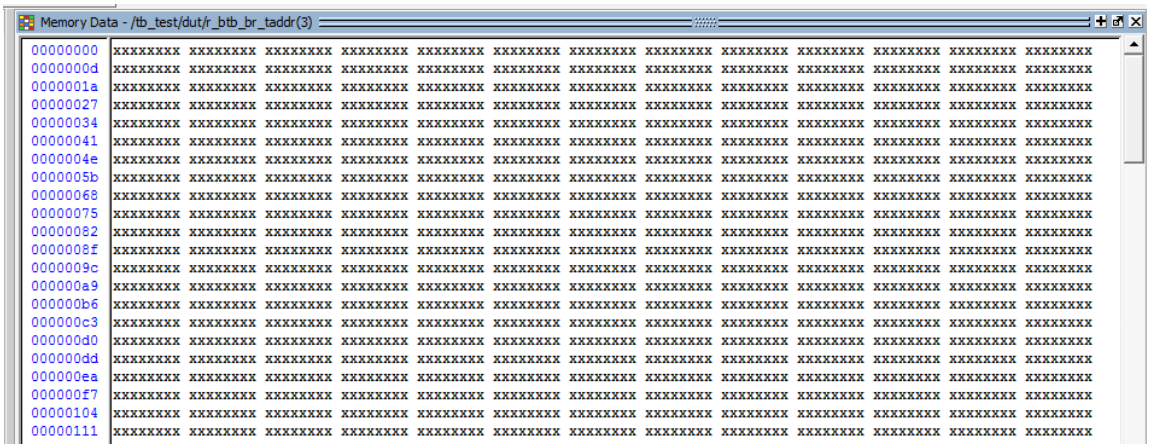
- Branch Target Address Way 1



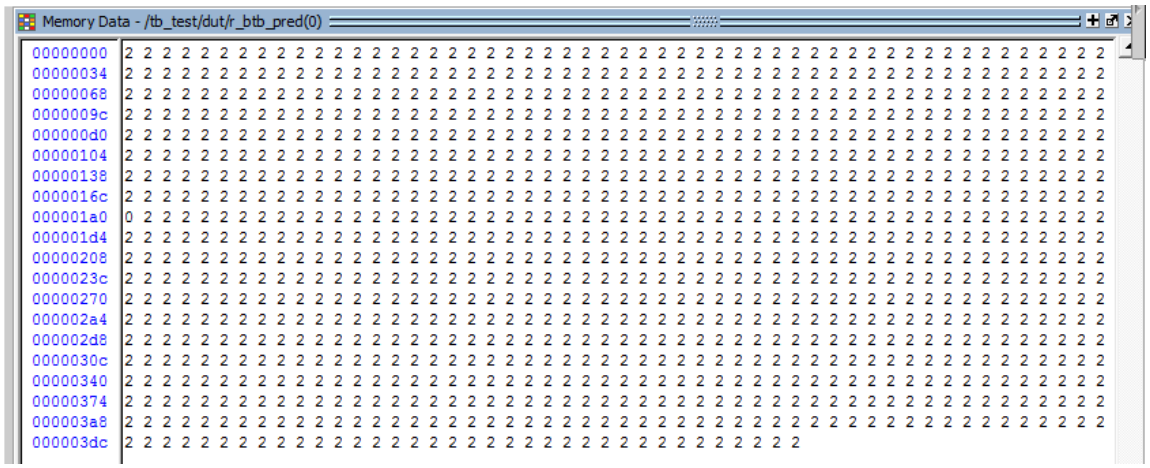
- Branch Target Address Way 2



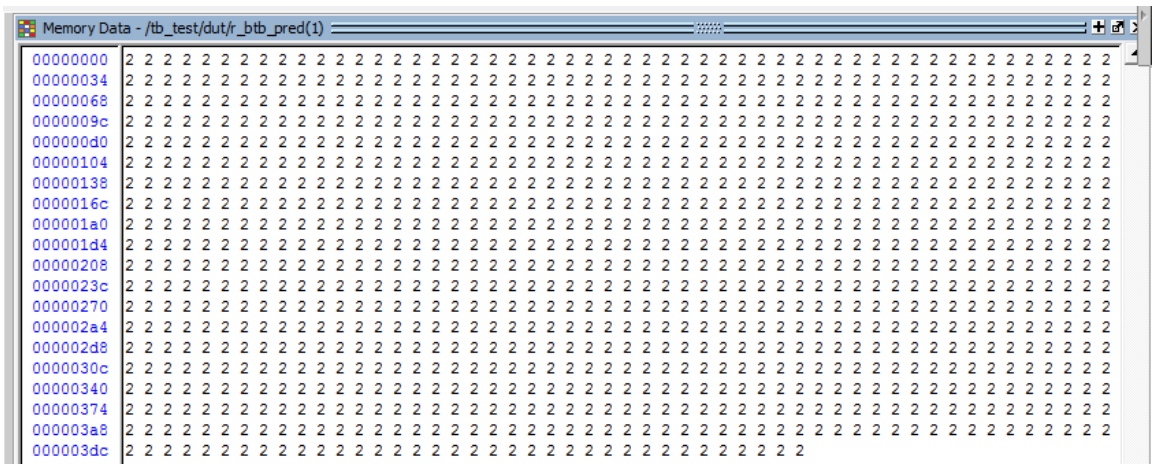
- Branch Target Address Way 3



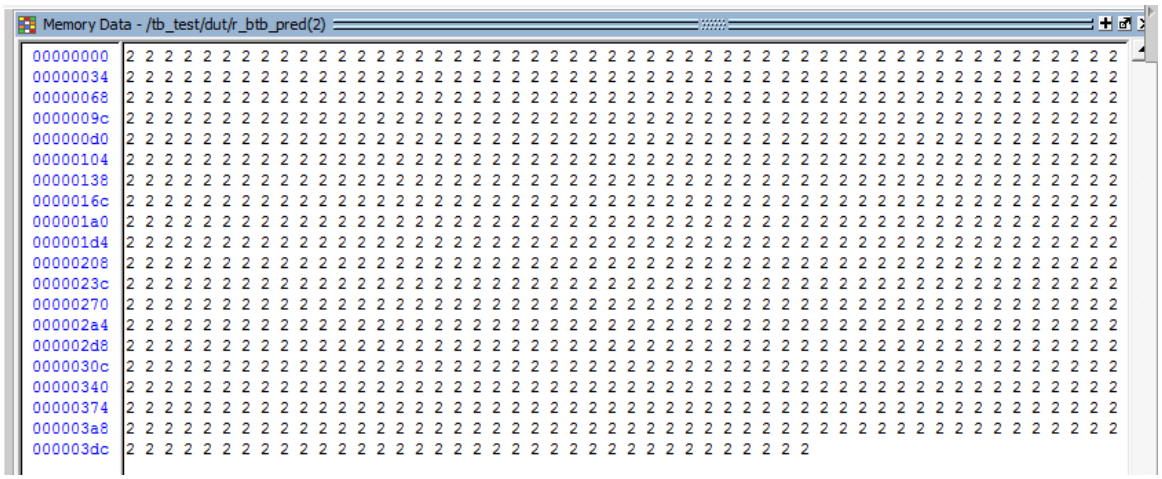
- Prediction bit Way 0 in BTB



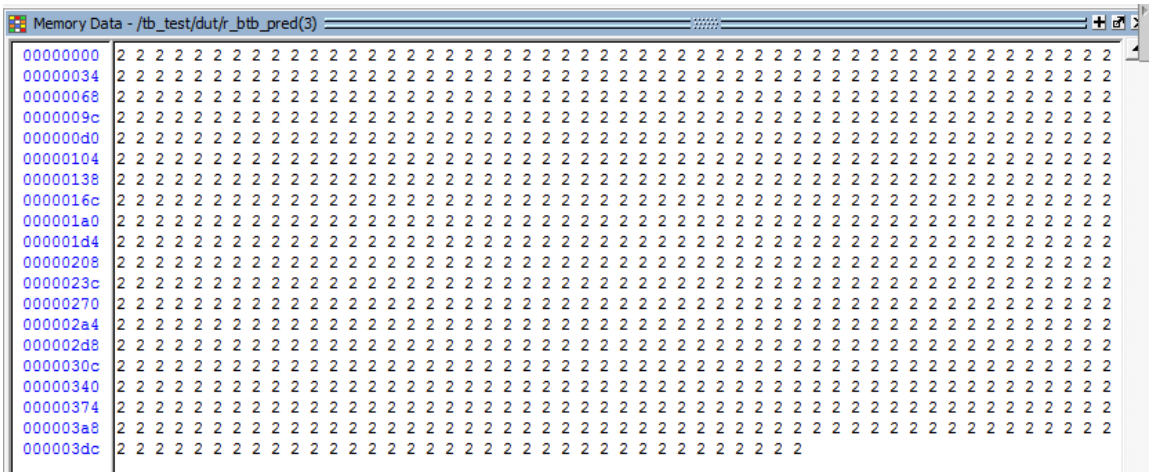
- Prediction bit Way 1 in BTB



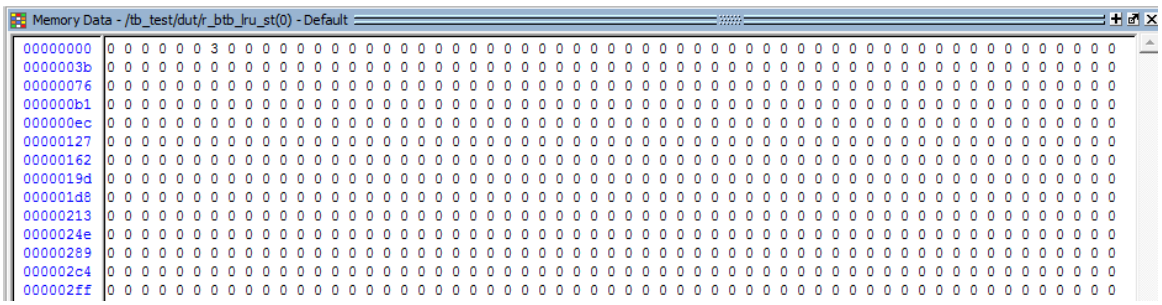
- Prediction bit Way 2 in BTB



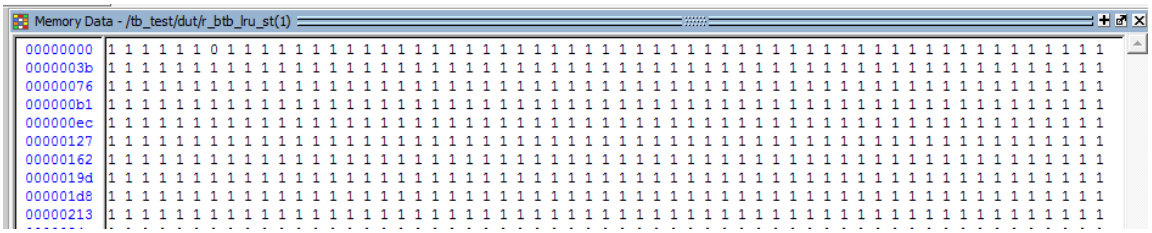
- Prediction bit Way 3 in BTB



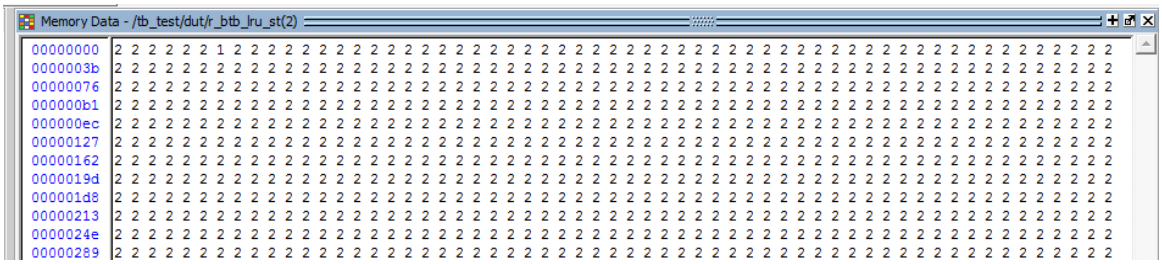
- LRU state at Way 0



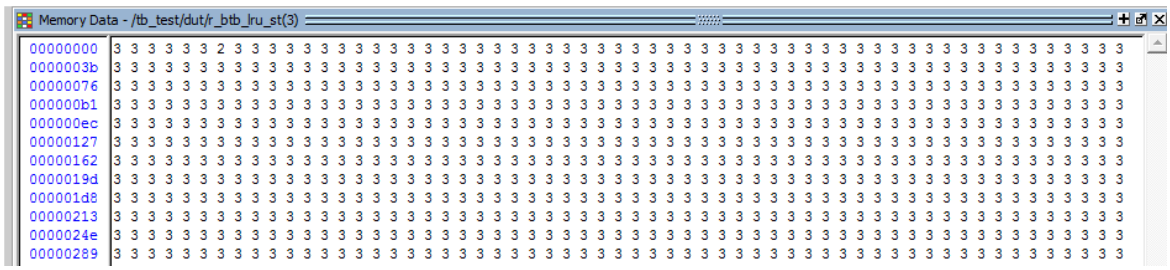
- LRU state at Way 1



- LRU state at Way 2



- LRU state at Way 3

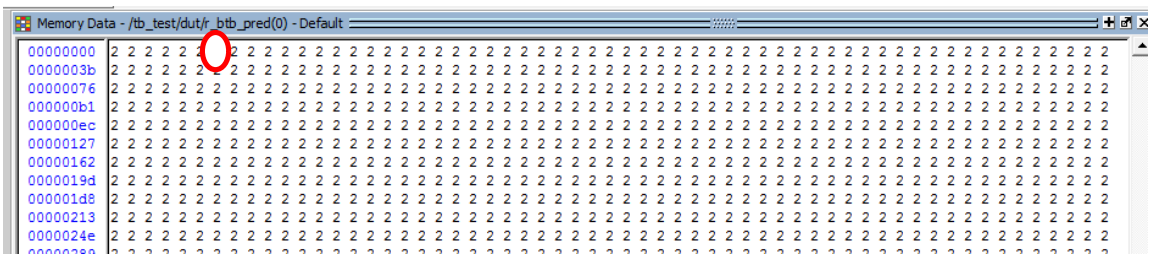


4. Scenario3 - Read Hit

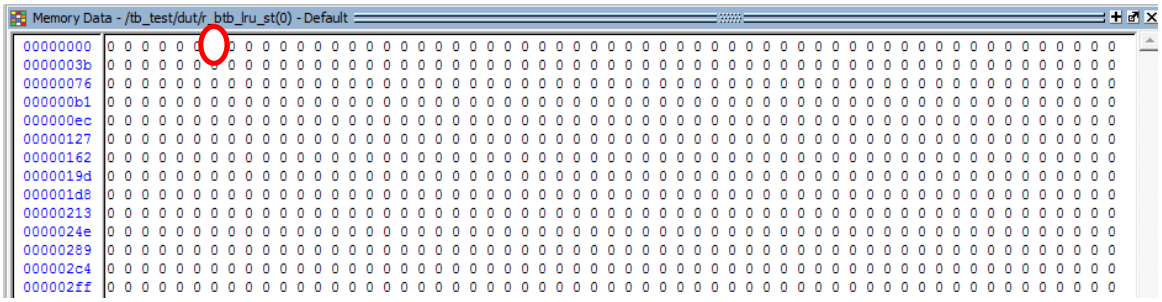
B_BP_BTB4WAY_i_Br <= 1'b1;

B_BP_BTB4WAY_i_ID_Equal <= 1'b1;

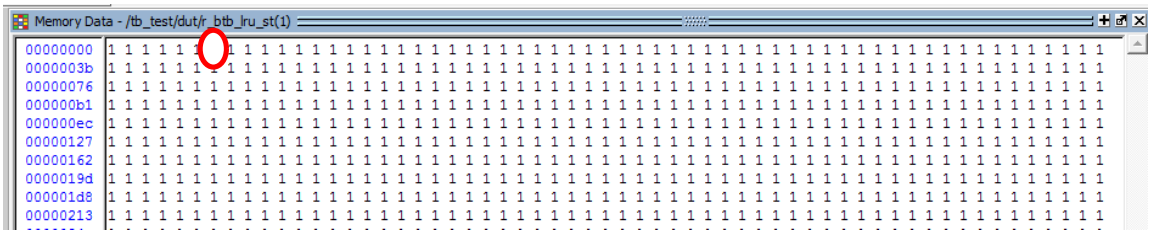
- Prediction bit



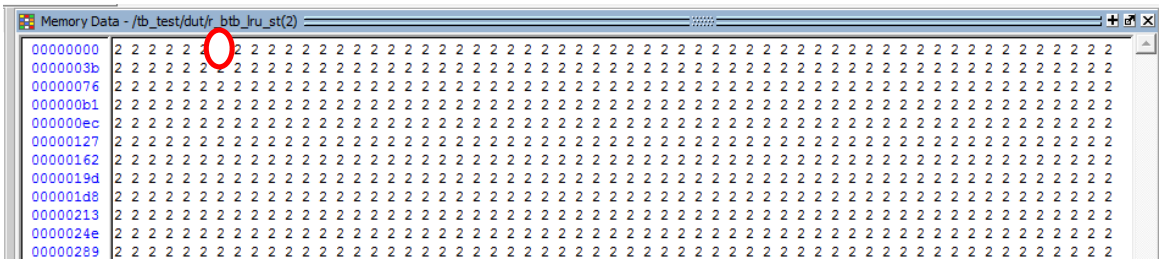
- LRU state at the way it store - Way 0



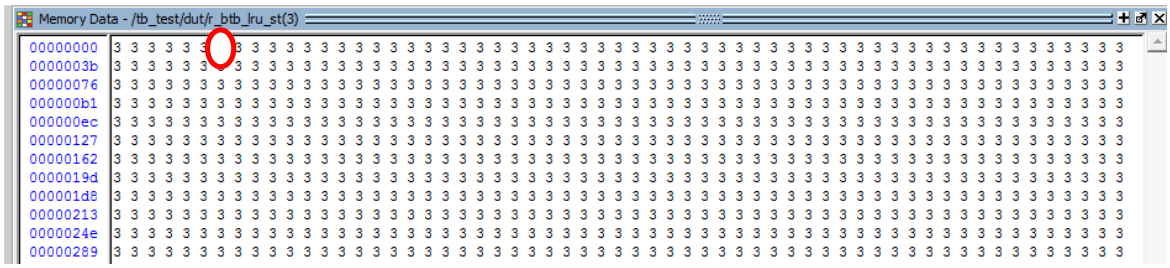
- LRU state at Way 1



- LRU state at Way 2



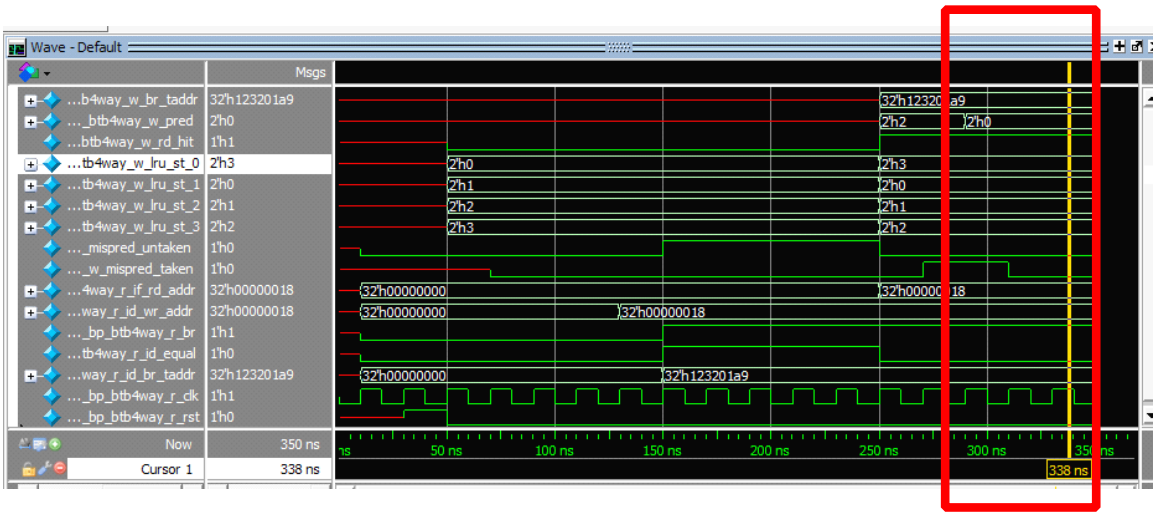
- LRU state at Way 3



5. Scenario 4 - Read Hit - Mispredict

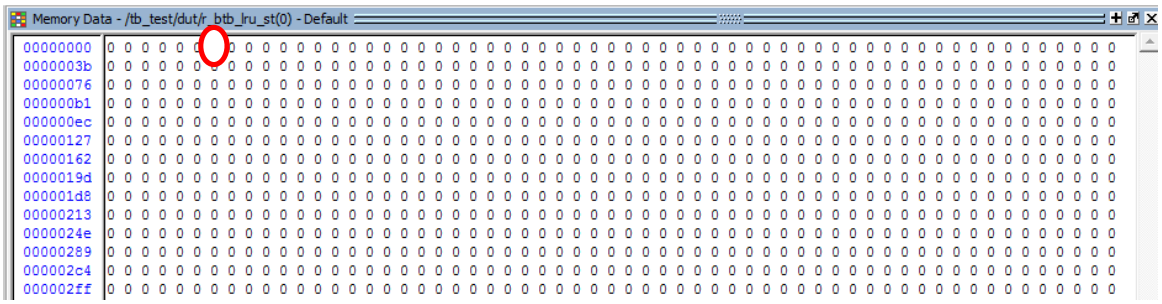
B_BP_BTB4WAY_i_Br <= 1'b1;

B_BP_BTB4WAY_i_ID_Equal <= 1'b0;

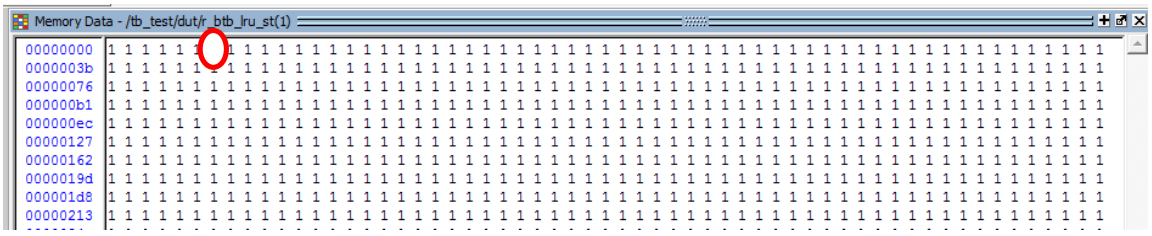


Prediction bit changes from "10" to "00". (Weakly taken -> Strongly untaken)

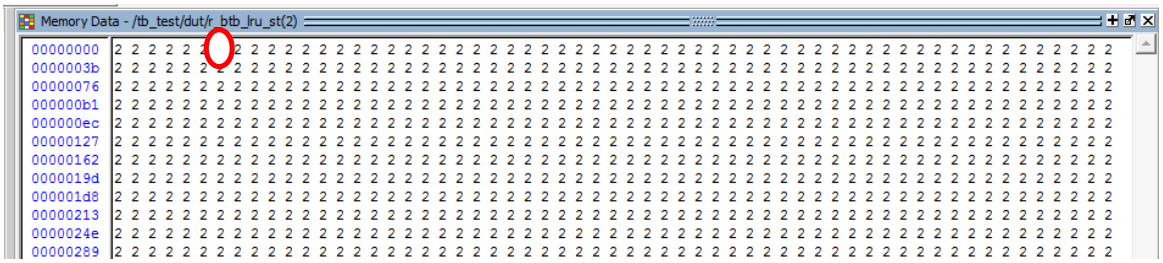
- LRU state at the way it store - Way 0



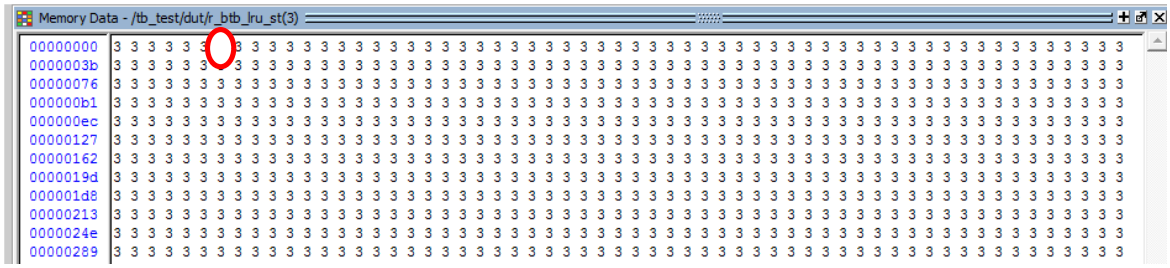
- LRU state at Way 1



- LRU state at Way 2



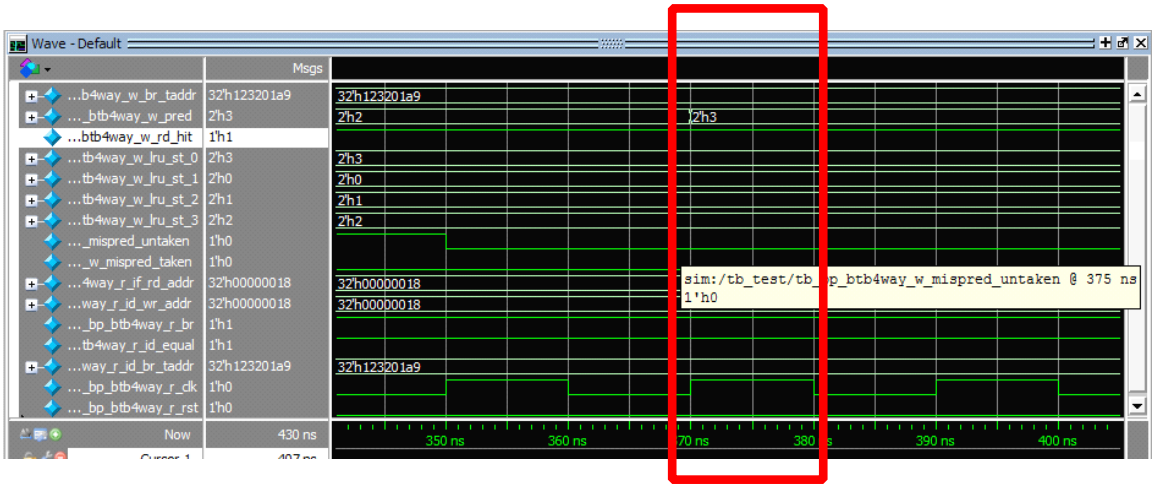
- LRU state at Way 3



6. Read Hit - Mispredict

B_BP_BTB4WAY_i_Br <= 1'b1;

B_BP_BTB4WAY_i_ID_Equal <= 1'b0;

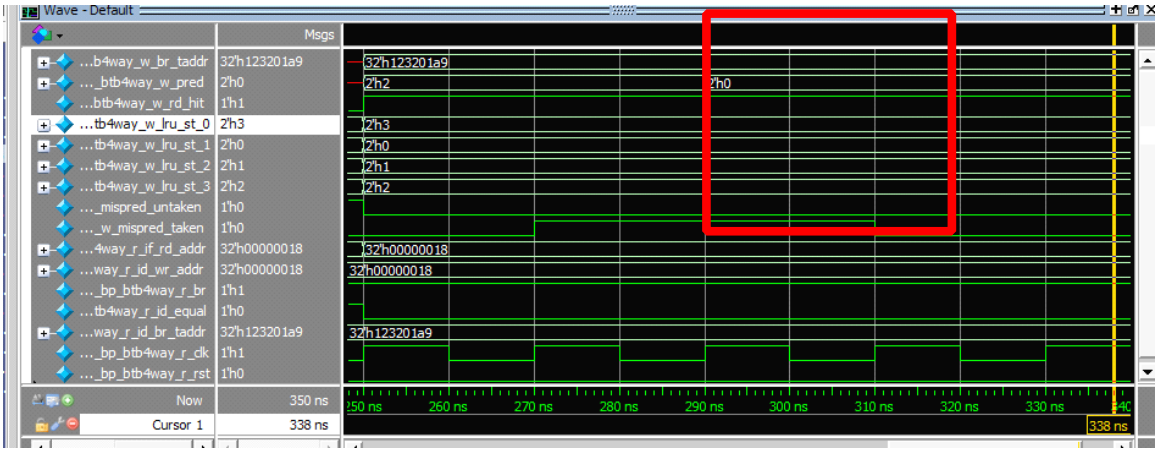


Misprediction, so the prediction bit changes. (Increase)

7. Read Hit

B_BP_BTB4WAY_i_Br <= 1'b1;

B_BP_BTB4WAY_i_ID_Equal <= 1'b1;



No misprediction, and prediction bit change to "Strongly untaken"

Chapter 7 - Discussion

7.1 Discussion

The branch predictor is essential as there is always branch instruction occurring in a microprocessor.

The branch predictor developed in this project is capable of solving the beq instruction. It able to send out the branch targeted address, read hit or miss of buffer, prediction bit value as well as the LRU. If there is any misprediction occur, it will notify the BPB to update the entry inside the BTB.

In this design, the BTB are capable of handling accessing and updating at the same moment as it uses the dual-port configuration and both the process will not have the same output source.

Cache memory technology is used as a guide of designing the buffer, so the BTB characteristics is almost similar with a normal cache but it has much more information store inside the BTB. Somehow, there is always a miss while first start of the microprocessor and this model only capable for beq function only.

Lastly, the objective of this project is achieved. The BPB is developed in RTL and work well. The functionality of BPB is tested as well. The BTB that work inside the BPB also functioning well.

7.2 Future Works

The branch predictor in this project is completed, yet, it still need to be implemented into the RISC32 processor. The future improvement should implement the model into the microprocessor and enhance it to make it capable for other branch instruction as well.

Bibliography

1. Mok, K.M. Computer Organization and Architecture Notes.. : University of Tunku Abdul Rahman, Faculty of Information and Communication Technology, 2009.

2. Patterson, David. Computer Architecture A Quantitative Approach. 2ed . s.l. : Morgan Kaufmann.

3. Hennessy, John L. and Patterson, David A. Computer Organization and Design : The Hardware/Software Interface. 4th. San Francisco : Morgan Kaufmann,

4. *Cache Memory.* (2009). Retrieved 2012, from http://codingfreak.blogspot.com/2009/03/cache-memory-part2.html

5. *Electronics Technician.* (n.d.). Retrieved 2012, from http://electronicstechnician.tpub.com/14091/css/14091_122.htm

6. Finley, T. (2000). *Cache.* Retrieved 2012, from http://www.tfinley.net/notes/cps104/cache.html#details_miss

7. Handy, J. (1998). *The Cache Memory Book.* San Diego: Academic Press.

8. Hennessy, J.L., Patterson, D.A. (2003). *Computer Architecture.* San Francisco: Morgan Kaufmann.

9. Mok, K. M. Digital System Design Lecture Notes. University Tunku Abdul Rahman. Kampar , Lecture Notes.

10. Yee Hen, Ong, Enhancement of CPU - RISC32. Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman.

11. Chun Jin, Teoh, RISC32Interrupt Handling For Enhanced RISC32 Architecture. Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman. August 2008