**ECE486/586**

**Computer Architecture**

**Final Project Report for Branch and Branch Target Prediction**

**18th March 2013**

**Student Name Student ID**

**Ameya Deswandikar 970606576**

**Nikhil Patil 926474005**

**Portland State University**

*Electrical and Computer Engineering*

*Maseeh College of Engineering and Computer Science*

**CONTENT**

1. Project specification ………………………………………………………………………………………………… 3
2. Assumptions……………………………………………………………………………………………………………...3
3. ALPHA Predictor Implementation………………………………………………………………………………4
   1. Block diagram………………………………………………………………………………………………….4
   2. Memory management……………………………………………………………………………………..5
   3. Flow chart………………………………………………………………………………………………………..6
   4. Testing strategy……………………………………………………………………………………………….8
   5. Test results……………………………………………………………………………………………………...8
   6. Benchmark result…………………………………………………………………………………………….9
4. Enhanced Alpha Predictor with BTB design……………………………………………………………...10
   1. Trace file analysis…………………………………………………………………………………………...10
   2. Design Space (Graphs variations Vs performance)………………………………………….12
   3. Design of Branch target buffer……………………………………………………………………….13
   4. Performance cases………………………………………………………………………………………….14
   5. Flow charts……………………………………………………………………………………………………..15
5. Development environment……………………………………………………………………………………….17
6. Repository…………………………………………………………………………………………………………………17
7. Appendix…………………………………………………………………………………………………………………..17
   1. Source code
   2. Trace file analysis result

**1. Project Specifications**

**Branch and Branch Target Prediction**

1. Branch Predictor - Implementation of Alpha 21264 branch predictor
2. Brach Target Predictor – Design and simulation of branch target predictor

**Design Constraints:**

* ***Memory Limit:*** Maximum 8K bytes of storage (total, including the Alpha predictor)
* ***Table sizes:*** All tables must be sized as powers of two.
* ***Associative tables***: ≤ 8 way
* Random replacement must be reproducible.
* All multiplying or dividing numbers must be in powers of two.

**2. Assumptions**

* Program counter is at most 32 bits wide.
* Size of an instruction is not fixed
* Instructions may not be aligned to any fixed size.

**3. ALPHA Predictor Implementation**

The Alpha 21264 branch predictor uses both local correlation and global correlation. To implement this, ALPHA predictor allocates dedicated memory to maintain local branch history and global branch history. To predict the direction of the current branch, the 21264 implements a tournament branch prediction counter to select between the local predictor and global predictor. Together, local and global correlation techniques improve prediction rate.

1. **Block Diagram**

**GLOBAL\_PREDICTOR\_COUNTER**

**( 4096 x 2)**

**LOCAL\_PREDICTOR\_COUNTER**

**(1024 x 3)**

**LOCAL\_HISTORY\_TABLE**

**( 1024 x 10)**

**Program Counter**

**MUX**

**CHOICE\_PREDICTOR\_COUNTER**

**( 4096 x 2)**

**Branch Prediction**

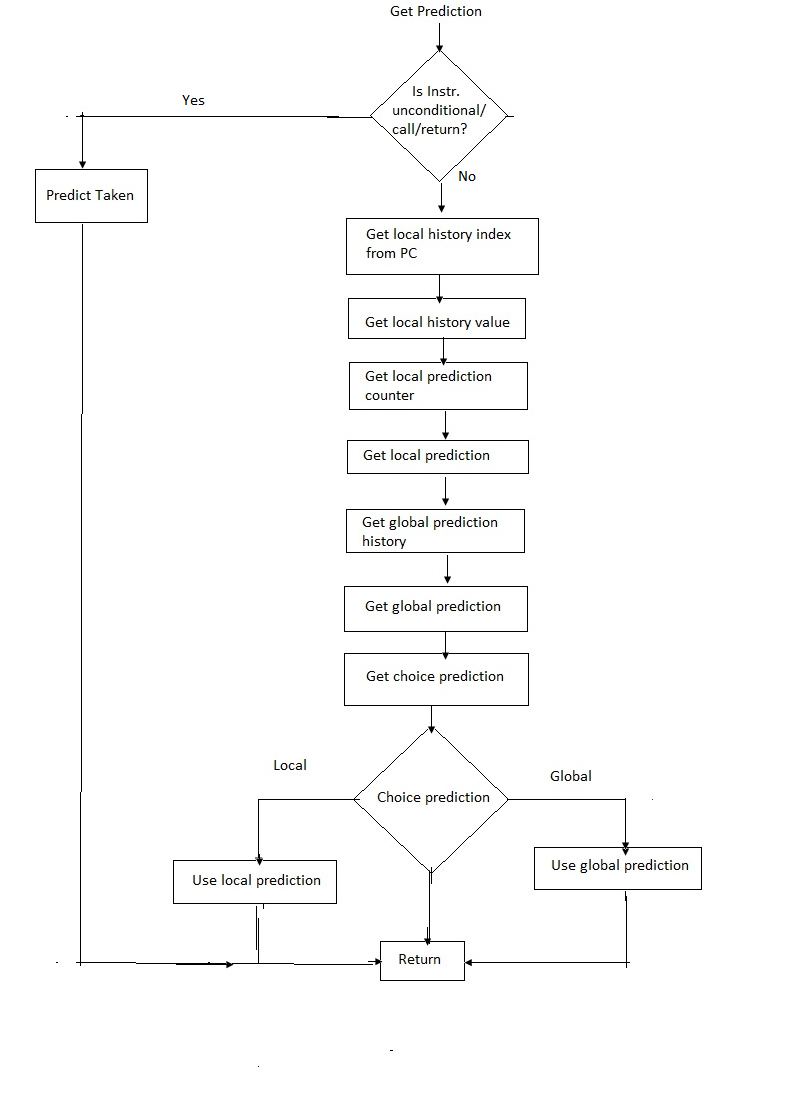
**PATH\_HISTORY\_TABLE ( 1 x 12)**

**Tournament Branch Predictor:**

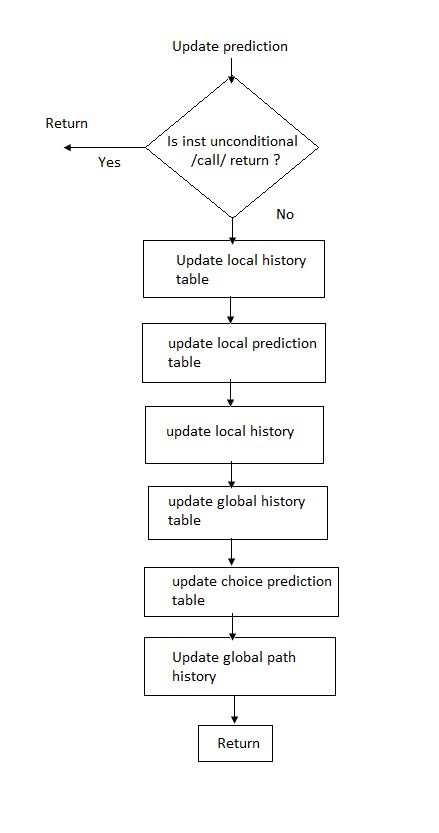
1. Local Branch Predictor:
   1. A local history table contains the past 10 outcome of the particular branch.
   2. A pattern history table contains 1024 entries.
   3. A 3-bit saturating counter for each entry in the pattern history table.
2. Global Branch Predictor:
   1. A global history register contains the outcome of the past 12 branches.
   2. A pattern history table contains 4096 entries.
   3. A 2-bit saturating counter for each entry in the pattern history table.
3. Choice Predictor Counter:
   1. A 2-bit saturating counter chooses which of the two branch predictors to use for each branch.
4. **Memory Management:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Memory Blocks** | **Indexed By** | **Size (bits)** |
| 1 | LOCAL\_HISTORY\_TABLE | **MEM**[ PROGRAM COUNTER] | 1024 x 10 |
| 2 | LOCAL\_PREDICTOR\_COUNTER | **MEM**[ LOCAL\_HISTORY\_TABLE] | 1024 x 03 |
| 3 | GLOBAL\_PATH HISTRORY\_TABLE |  | 1 x 12 |
| 4 | GLOBAL\_PREDICTOR\_COUNTER | **MEM**[ GLOBAL\_PATH\_HISTORY\_TABLE] | 4096 x 02 |
| 5 | CHOICE\_PREDICTOR\_COUNTER | **MEM**[ GLOBAL\_PATH\_HISTORY\_TABLE] | 4096 x 02 |
|  |  | **TOTAL MEMORY USED** | **29708** |

This is well within the constraint of 8K bytes (65536 bits) **c. Flow Chart:**

**1. Get Branch Prediction**

**2. Update Branch Prediction**



**d. Testing Strategy:**

To test the branch predictor code we have considered two cases.

**Case I:** Single branch with execution pattern: **T N N N N N N N N N -----**To Test Local Correlations

**Case II:** 3 correlative conditional branches A, B & C --------------------------To Test Global Correlation

If ( A==0)

{

//Code for A

}

If ( B==0)

{

//Code for B

}

If( A==B)

{

//Code for C

}

If A -> Taken & B-> taken then C-> must be Taken.

The test cases were generated using a custom framework for test generation.

1. **Test Results:**

**--** Appendix b.

1. **Benchmark Results:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr No** | **Benchmark** | **Tracefile** | **Total Wrong cc Predicts / 1000 insts** | **Total Wrong target Predicts / 1000 insts \*** |
| 1 | Floating Point | DIST-FP-1.bz2 | 3.364 | 23.519 |
| 2 | Floating Point | DIST-FP-2.bz2 | 1.334 | 12.913 |
| 3 | Floating Point | DIST-FP-3.bz2 | 0.516 | 7.373 |
| 4 | Floating Point | DIST-FP-4.bz2 | 0.264 | 4.644 |
| 5 | Floating Point | DIST-FP-5.bz2 | 2.245 | 36.337 |
| 6 | Integer | DIST-INT-1.bz2 | 8.537 | 64.702 |
| 7 | Integer | DIST-INT-2.bz2 | 11.842 | 56.589 |
| 8 | Integer | DIST-INT-3.bz2 | 12.619 | 54.217 |
| 9 | Integer | DIST-INT-4.bz2 | 2.898 | 28.422 |
| 10 | Integer | DIST-INT-5.bz2 | 0.484 | 4.999 |
| 11 | Multimedia | DIST-MM-1.bz2 | 8.858 | 41.29 |
| 12 | Multimedia | DIST-MM-2.bz2 | 10.816 | 91.962 |
| 13 | Multimedia | DIST-MM-3.bz2 | 1.598 | 116.523 |
| 14 | Multimedia | DIST-MM-4.bz2 | 2.141 | 146.874 |
| 15 | Multimedia | DIST-MM-5.bz2 | 6.398 | 67.252 |
| 16 | Server | DIST-SERV-1.bz2 | 9.793 | 96.818 |
| 17 | Server | DIST-SERV-2.bz2 | 10.171 | 94.361 |
| 18 | Server | DIST-SERV-3.bz2 | 8.129 | 93.976 |
| 19 | Server | DIST-SERV-4.bz2 | 10.938 | 113.355 |
| 20 | Server | DIST-SERV-5.bz2 | 11.549 | 115.392 |
|  |  | **Geometric mean** | **3.855** | **43.244** |

\*Please note that the Branch predictor has no effect on the target predictions.

**4. Enhanced Alpha Predictor with BTB design**

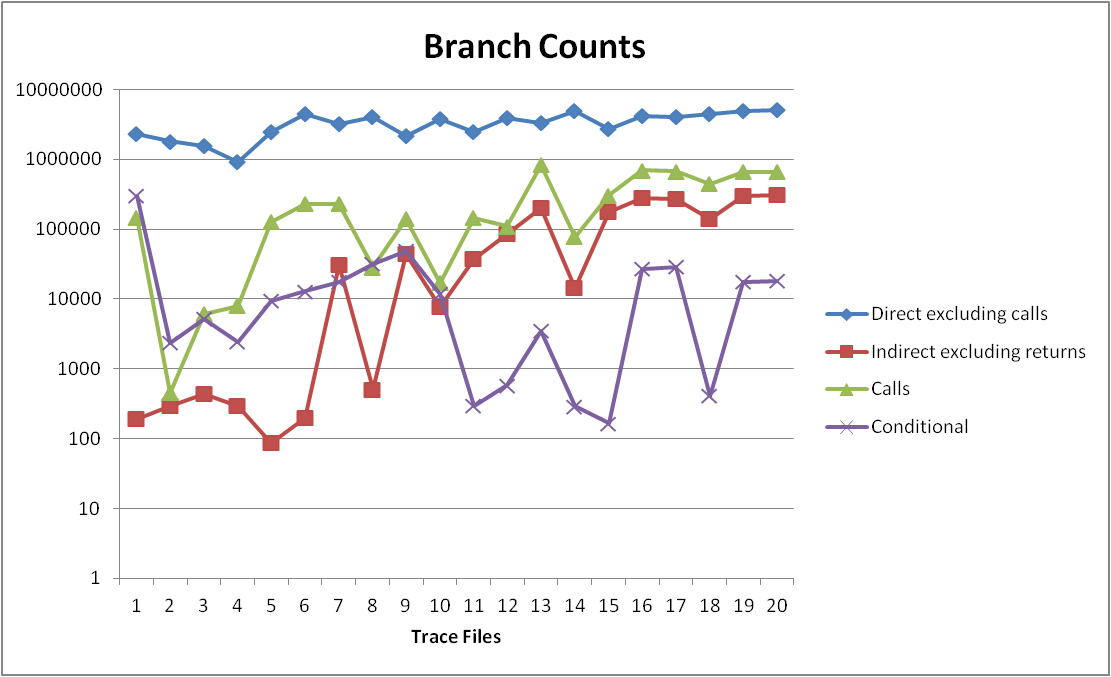
1. **Trace File Analysis:**

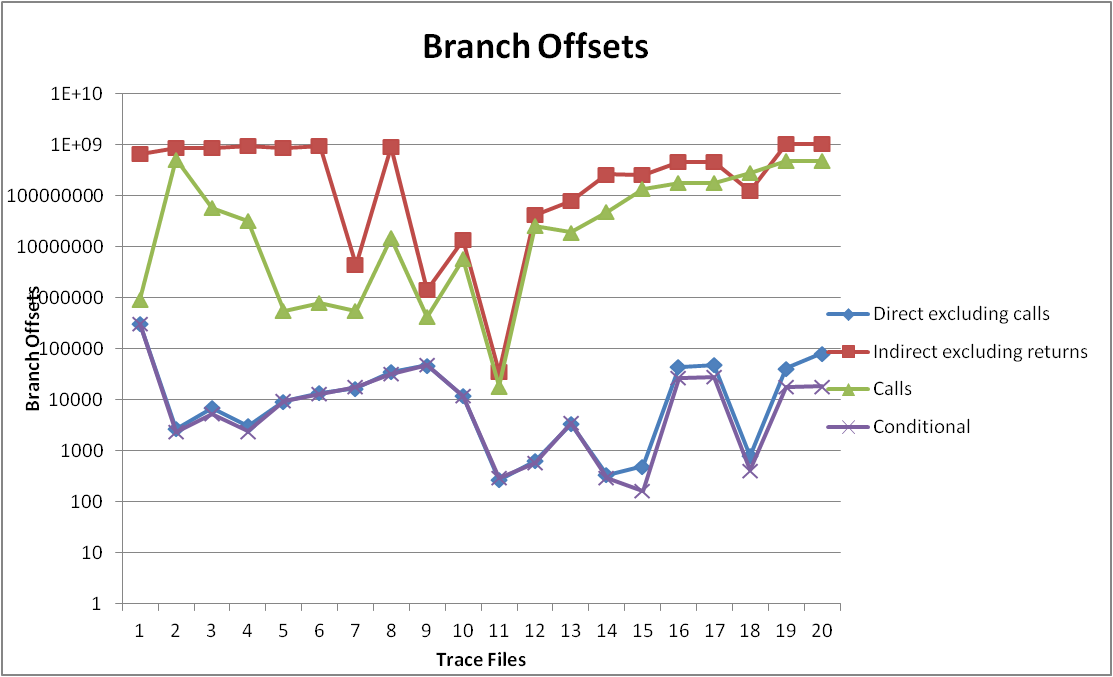
We have modified the given perl script and the framework to obtain the data related to all branches. We have used this strategy to exploit this information for designing the **Branch Target Buffer.**

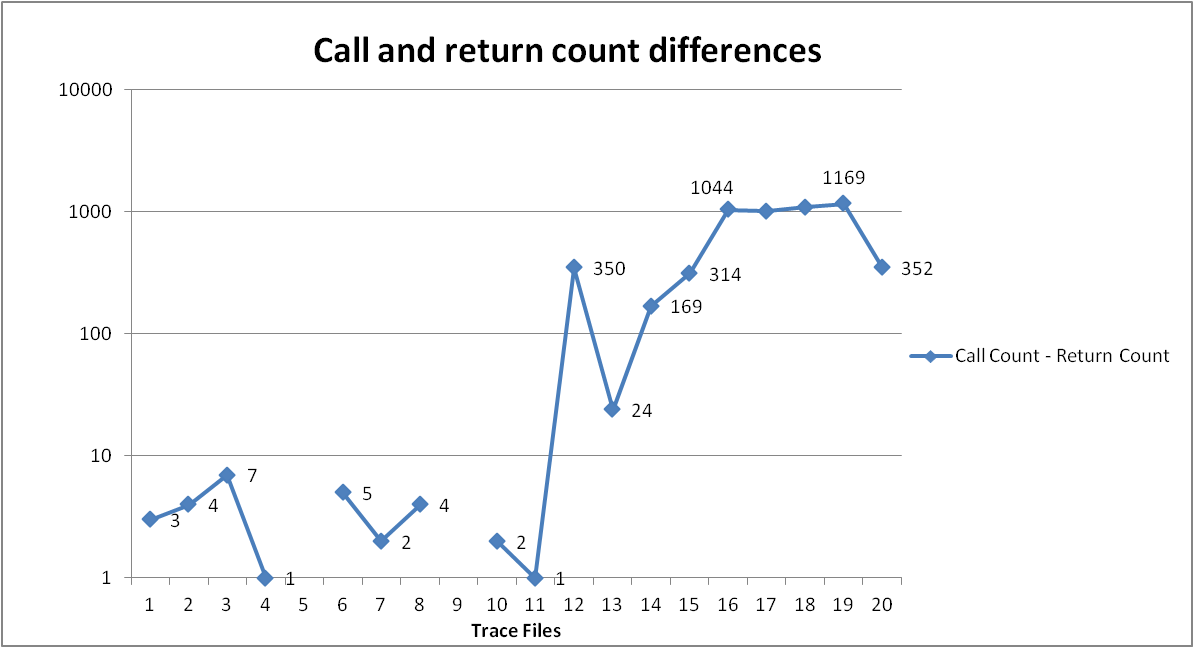
For branch offsets and call and return differences, each point is the average value of that parameter within that trace file.

We extracted the following results:

1. Branch counts for each type of branches
2. Average Branch offsets
3. Average Call and return count differences



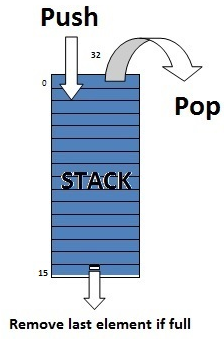




**Observations:**

1. The branch count for the direct branches is the highest. They are larger by almost two orders of magnitude as compared to other branch types.
2. The offset for direct branches offset is also minimum. The offset can fit into 17 bits.
3. Counts for the Call and return branches provide an estimate for the return stack size.

1. **Design (Design Variations Vs Prediction Performance)**

****

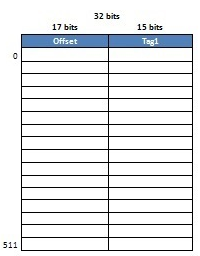
**Return Stack Design**

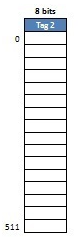
We designed a return stack such that if it is becomes full, the first pushed element is removed. We obtained performance characteristics for various stack sizes: 0 (no stack at all) to 16 deep. Using the perl script we obtained the geometric mean for target mispredicts in all 20 trace files, for various stack sizes, absence of any branch target buffer

Stack size of 16 provides optimum balance between performance and space required.

1. **Design of Branch target buffer**

From the trace file analysis, we concluded that a majority of branch offsets can fit within a 17 bit range. Thus we chose a direct mapped table 32 bits wide with 512 entries. The 17 bits are used for branch offsets and 15 bits for the program counter tag1 bits. Since the table is 512 deep, we get index bits of 9 bits from the program counter. Thus, the remaining 8 bits (tag2) are stored in another table (direct mapped) that is 512 deep with each entry 8 bits wide.







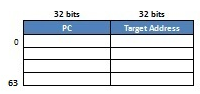


Figure 3: Fully associative table 1



Figure 1: Offset Table

In the space left, we used two fully associative tables that store complete program counter for any branch, without indexing.

Figure 4: Fully associative table 2

Replacement policy:

The last entry in this fully associative table is removed and other entries are shifted down. The new item is placed in the first slot.

In order to improve the branch predictor’s performance, we have increased the local prediction counter’s size from 3 bit to 4 bit by doing this we have also utilized the available memory effectively. This increases the branch predictor memory requirement by 1024 bits.

Memory requirement for target predictor:

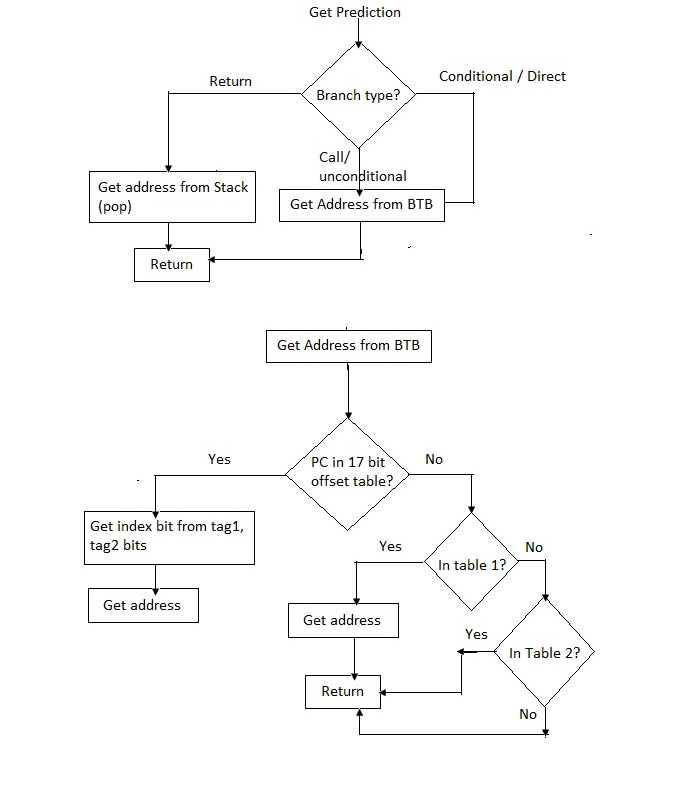
|  |  |  |
| --- | --- | --- |
| Table | Number of unique branches | Memory (bits) |
| Offset Table | 512 | 16384 |
| Tag bits table | 0 | 4096 |
| Fully associative table 1 | 128 | 8192 |
| Fully associative table 2 | 64 | 2048 |
| Local History table | - | 10240 |
| Local Prediction table | - | 4096 |
| Global Prediction table | - | 8192 |
| Choice prediction table | - | 8192 |
| Global path history | - | 12 |
| Total | **704** | **61452** |

We saved 65536 - 61452 = 4084 bits.

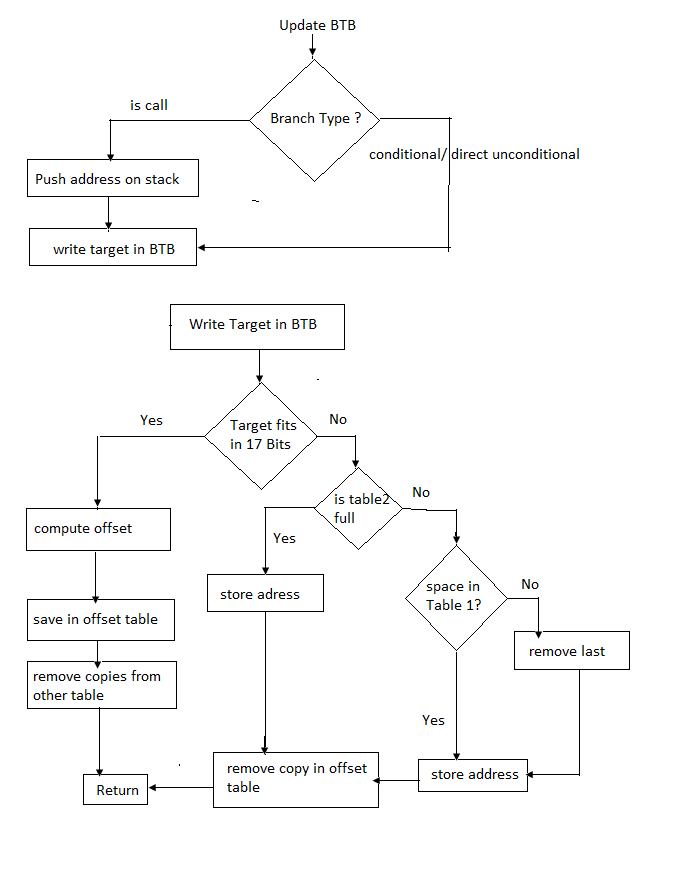
1. **Performance Cases:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Case** | **Performance Case Description** | | | | | | **Prediction Performance (Total Wrong Predicts / 1000 instructions )** | |
| **Changes In Alpha Branch Predictor** | **Changes in BTB** | | | | | **Branch Prediction** | **Branch Target Prediction** |
| **Offset BTB Size** | **Tag Bits table** | **Fully Associative Table 1 Size** | **Fully Associative Table 2 Size** | **Stack for Return branch** |
| Case 1 | No | Without BTB | | | | | 3.855502454 | 43.24417785 |
| Case2 | No | 512 x ( 32 + 32 ) bits | NA | NA | NA | 32 x 16 bits | 3.855502454 | 10.20245655 |
| Case 3 | No | 512 x (16 + 16 ) bits | 512 x 8 bits | NA | NA | 32 x 16 bits | 3.855502454 | 8.857686195 |
| Case 4 | No | 512 x ( 17 + 15 ) bits | 512 x 8 bits | 128 x ( 32 + 32 ) bits | 64 x ( 32 + 32 ) bits | 32 x 16 bits | 3.855502454 | 7.340589735 |
| Case 5 | Yes - 4 Bit Local Prediction Counter | 512 x ( 17 + 15 ) bits | 512 x 8 bits | 128 x ( 32 + 32 ) bits | 64 x ( 32 + 32 ) bits | 32 x 16 bits | 3.838446283 | 7.293957785 |
| Case 6 | 4 Bit Local prediction counter + 11 bit Global History Table | 512 x ( 17 + 15 ) bits | 512 x 8 bits | 256 x ( 32 + 32 ) bits | 64 x ( 32 + 32 ) bits | 32 x 16 bits | 4.043052555 | 7.307459419 |

1. **Flow chart for Get Prediction**



**Flow chart for update prediction**



**5. Development environment**

We used the Eclipse IDE designing and debugging our target predictor simulator. We used the make files provided in the framework, and not the Eclipse auto generated makefiles. The compiler used was Mingw port of the GNU g++ running on the Cygwin environment on the windows platform.

**6. Repository**

We used Subversion for source control. The repository is hosted on Google Code and provides up to 5 GB space for hosting projects. All previous revisions can be easily obtained via the repository. It also contains an issue tracking system for managing defects.

**7. Appendix**

1. **Source code:**
2. **Trace file analysis result**