

Εθνικό Μετσόβιο Πολυτεχνείο  
Σχολή ΗΜ&ΜΥ  
Προηγμένα Θέματα  
Αρχιτεκτονικής Υπολογιστών  
8<sup>ο</sup> εξάμηνο, Ροή Υ  
Ακαδημαϊκό Έτος: 2012



## 2<sup>η</sup> Σειρά Ασκήσεων

Γερακάρης Βασίλης  
<vgerak@gmail.com>  
Α.Μ.: 03108092

27 Μαΐου 2012

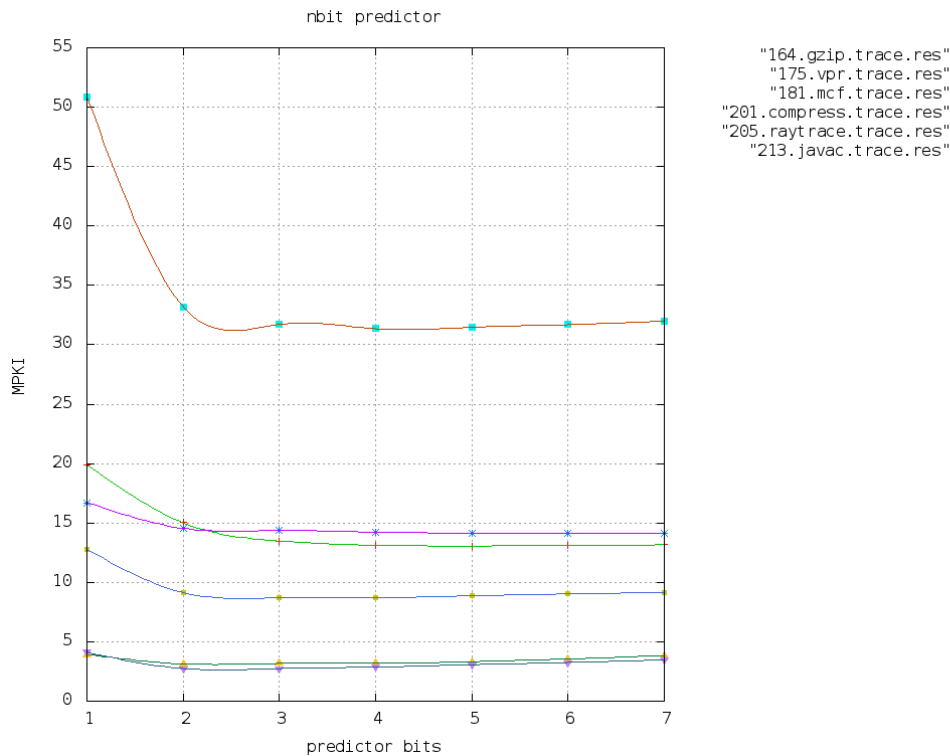
# Εισαγωγή

Στην άσκηση αυτή χρησιμοποιήσαμε ένα C++ Framework για την εξομοίωση προβλεπτών αλμάτων, όπως έχουν καταγραφεί από την εκτέλεση benchmarks της SPEC2000. Τα trace files που μας δώθηκαν περιέχουν τις εντολές άλματος που πραγματοποιήθηκαν κατά την εκτέλεση 100M εντολών.

## N-bit predictors

### A.1

Σε αυτό το τμήμα μελετήσαμε την απόδοση των N-bit predictors (1 έως 7), αξιολογώντας με βάση τα MPKI (Mispredictions Per Thousand Instructions). Στο τμήμα αυτό είχαμε σταθερά BHT entries, ίσα με 16K.



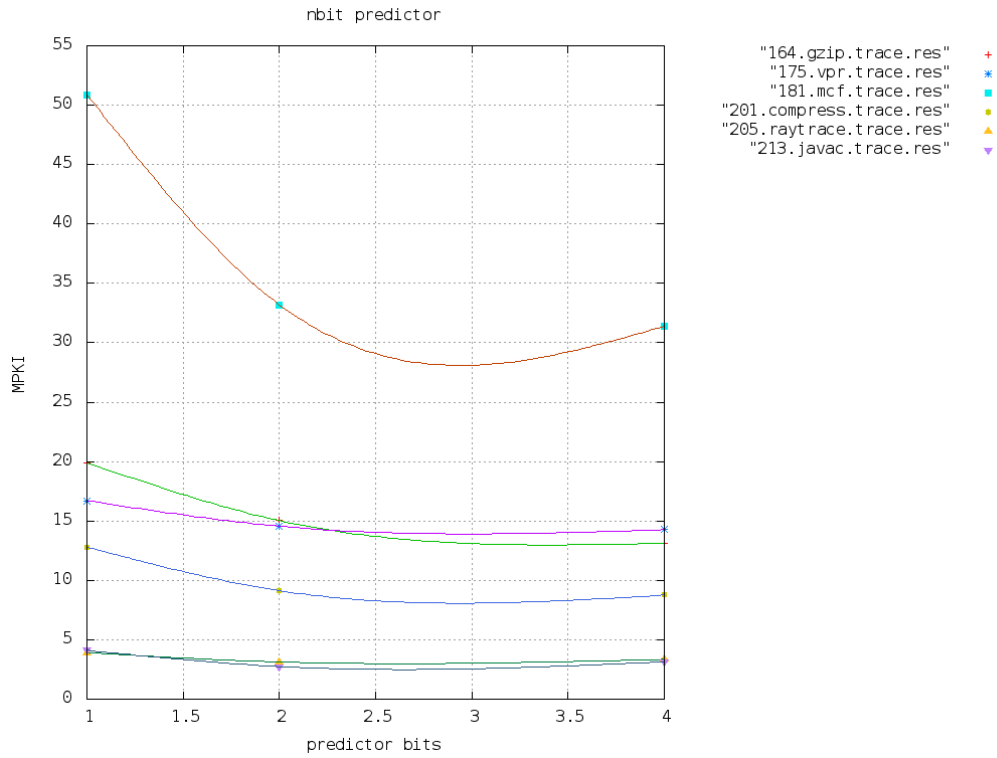
Σχήμα 1: 1 to 7 - bit predictors

Όπως παρατηρούμε από το παραπάνω διάγραμμα, στα περισσότερα benchmarks ο 4-bit predictor παρουσιάζει τη βέλτιστη επίδοση, καθώς εκείνος εμφανίζει τα λιγότερα misspredictions, έχοντας ταυτόχρονα λίγες απαιτήσεις από hardware.

## A2

Στο κομμάτι αυτό, μελετάμε τους {1,2,4}-bit predictors, αξιολογώντας πάλι με βάση τα MPKI. Αυτή τη φορά, έχουμε σταθερό hardware και ίσο με 32K, και μεταβάλλουμε το πλήθος των BHT entries.

HW	bits	BHT entries
32K	1	32K
32K	2	16K
32K	4	8K

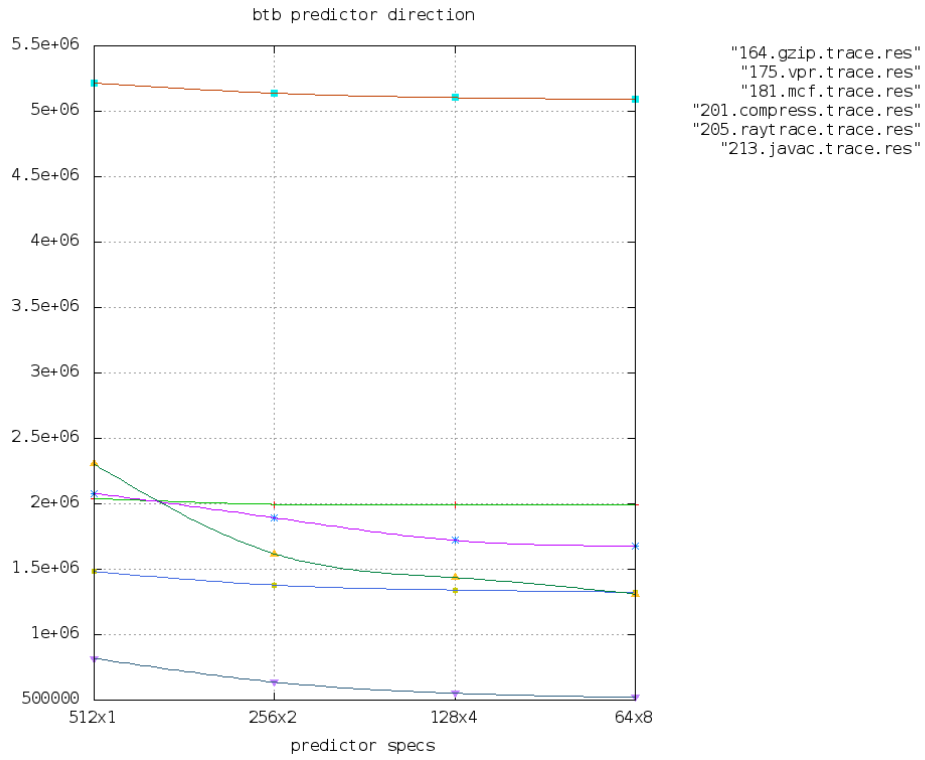


Σχήμα 2: 1,2,4 - bit predictors

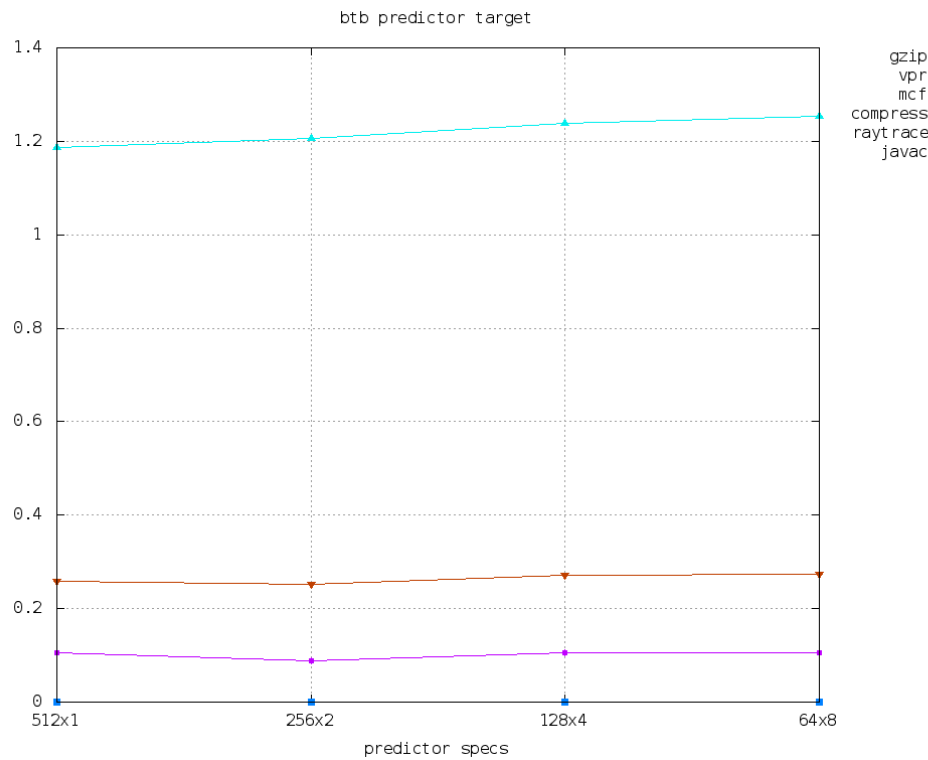
Παρατηρούμε πως ακόμα και στην περίπτωση που το hardware είναι σταθερό (32K), καλύτερη απόδοση εμφανίζει ο 4 bit predictor.

## BTB predictor

btb lines	btb_assoc
512K	1
256K	2
128K	4
64K	8



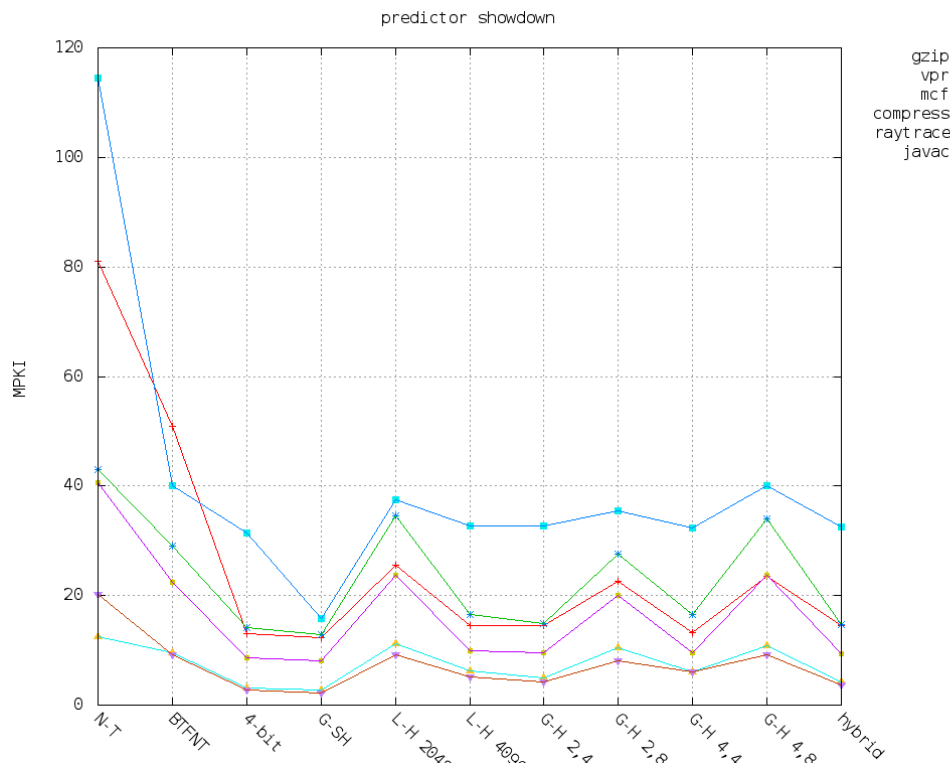
Σχήμα 3: Direction misspredictions (direction MPKI)



Σχήμα 4: Target Misspredictions (target MPKI)

Παρατηρούμε πως το target missprediction παραμένει σταθερό σχεδόν και είναι συγκριτικά αμελητέο, σε αντίθεση με το direction missprediction. Αυτό παρατηρούμε πως μειώνεται δραστικά στον 64x8 BTB predictor, οπότε επιλέγεται ως η επιθυμητή οργάνωση για τον BTB.

## C1. Σύγκριση διαφορετικών predictors



Σχήμα 5: Σύγκριση predictors

## Source Code

Ο πηγαίος κώδικας που χρησιμοποιήσαμε για τους predictors είναι ο ακόλουθος:

### Static Not-Taken

```

1  /* .....
2  * File Name : ntaken.h
3  * Purpose : 2nd Assignment in AdvCompArch
4  * Creation Date : 27-05-2012
5  * Last Modified : Sun 27 May 2012 18:52:27 EEST
6  * Created By : Vasilis Gerakaris <vgerak@gmail.com>
7  .....*/
8
9  #include "predictor.h"
10
11 class ntaken_update : public branch_update
12 {
13 public:
14     unsigned int index;
15 };
16
17 class ntaken_predictor: public branch_predictor
18 {
19     ntaken_update u;
20
21     branch_update *predict (branch_info & b)
22     {
23         if (b.br_flags & BR_CONDITIONAL)
24             u.direction_prediction (false);
25         else
26             u.direction_prediction (true);
27         u.target_prediction (0);
28         return &u;
29     }
30     void update (branch_update *u, bool taken, unsigned int target)
31     {
32     }
33 };

```

## Static Backward Taken Forward Not Taken

```
1  /* -----  
2  * File Name : btfmt.h  
3  * Purpose : 2nd Assignment in AdvCompArch  
4  * Creation Date : 27-05-2012  
5  * Last Modified : Sun 27 May 2012 22:58:13 EEST  
6  * Created By : Vasilis Gerakaris <vgerak@gmail.com>  
7  -----*/  
8  
9  #include <math.h>  
10 #include "predictor.h"  
11  
12 class btfmt_predictor: public branch_predictor  
13 {  
14     branch_update u;  
15     branch_info brInf;  
16  
17     bool jump;  
18  
19 public:  
20     void set_target(bool t)  
21     {  
22         jump = t;  
23     }  
24  
25     branch_update *predict (branch_info & b)  
26     {  
27         brInf = b;  
28         if (b.br_flags & BR_CONDITIONAL)  
29         {  
30             if (jump)  
31             {  
32                 u.direction_prediction (false);  
33             }  
34             else  
35             {  
36                 u.direction_prediction(true);  
37             }  
38         }  
39         else  
40             u.direction_prediction (true);  
41         u.target_prediction (0);  
42         return &u;  
43     }  
44  
45     void update (branch_update *u, bool taken, unsigned int target)  
46     {  
47     }  
48 };
```

## 4-bit predictor

```
1  // nbit_predictor.h  
2  //  
3  //  
4  
5  #include <math.h>  
6  
7  class nbit_update : public branch_update {  
8  public:  
9      unsigned int index;  
10 };  
11  
12 class nbit_predictor : public branch_predictor {  
13 public:  
14     #define NBP_TABLE_BITS      15 //number of entries = 2^15  
15     nbit_update u;  
16     branch_info bi;  
17     int counter_limit;  
18     int N_COUNTER_LENGTH;  
19  
20     unsigned char tab[1<<NBP_TABLE_BITS];  
21  
22     nbit_predictor (int length) :N_COUNTER_LENGTH(length) {  
23         memset (tab, 0, sizeof (tab));  
24         counter_limit = ((int) pow(2.0, N_COUNTER_LENGTH)) - 1;  
25     }  
26  
27     branch_update *predict (branch_info & b) {  
28         bi = b;  
29         if (b.br_flags & BR_CONDITIONAL) {  
30             u.index = (b.address & ((1<<NBP_TABLE_BITS)-1));  
31             u.direction_prediction (tab[u.index] >> (N_COUNTER_LENGTH-1));  
32         } else {  
33             u.direction_prediction (true);  
34         }  
35         u.target_prediction (0);  
36     }
```

```

36         return &u;
37     }
38
39     void update (branch_update *u, bool taken, unsigned int target) {
40         if (bi.br_flags & BR_CONDITIONAL) {
41             unsigned char *c = &tab[((nbit_update*)u)->index];
42             if (taken) {
43                 if (*c < counter_limit) (*c)++;
44             } else {
45                 if (*c > 0) (*c)--;
46             }
47         }
48     }
49 };

```

## gshare predictor

```

1  // gshare_predictor.h
2  // This file contains a sample my_predictor class.
3  // It is a simple 32,768-entry gshare with a history length of 15.
4
5  class gshare_update : public branch_update {
6  public:
7      unsigned int index;
8  };
9
10 /*
11  * H klash gshare_predictor klhronomei thn klash
12  * branch_predictor kai kanei override tis me8odous
13  * predict kai update
14  */
15
16 class gshare_predictor : public branch_predictor {
17 public:
18     #define HISTORY_LENGTH      14
19     #define GSP_TABLE_BITS      15
20     gshare_update u;
21     branch_info bi;
22     unsigned int history;
23     unsigned char tab[1<<GSP_TABLE_BITS];
24
25     gshare_predictor (void) : history(0)
26     {
27         memset (tab, 0, sizeof (tab));
28     }
29
30     branch_update *predict (branch_info & b)
31     {
32         bi = b;
33
34         /*
35          * O gshare wrhsimopoietai mono gia conditional branches.
36          * Ta unconditional ginontai predicted panta TAKEN
37          */
38
39         if (b.br_flags & BR_CONDITIONAL)
40         {
41             u.index = (history << (GSP_TABLE_BITS - HISTORY_LENGTH)) ^ (b.address & ((1<<GSP_TABLE_BITS)-1));
42             u.direction_prediction (tab[u.index] >> 1);
43         }
44         else
45             u.direction_prediction (true);
46
47         // O gshare den kanei target prediction, gia auto to setoume sto 0.
48
49         u.target_prediction (0);
50         return &u;
51     }
52
53     void update (branch_update *u, bool taken, unsigned int target)
54     {
55         //O gshare wrhsimopoietai mono gia conditional branches
56         if (bi.br_flags & BR_CONDITIONAL) {
57             unsigned char *c = &tab[((gshare_update*)u)->index];
58             if (taken)
59             {
60                 if (*c < 3)
61                     (*c)++;
62             }
63             else
64             {
65                 if (*c > 0)
66                     (*c)--;
67             }
68             history <= 1;
69             history |= taken;
70             history &= (1<<HISTORY_LENGTH)-1;
71         }

```



$$\begin{array}{c|c} 72 & \} \\ 73 & \}; \end{array}$$

## Local-History two-level predictors

```

1  /* .....*/
2  * File Name : lhistory.h
3  * Purpose : 2nd Assignment in AdvCompArch
4  * Creation Date : 27-05-2012
5  * Last Modified : Sun 27 May 2012 22:55:19 EEST
6  * Created By : Vasilis Gerarakis <ugarak@gmail.com>
7  .....*/
8
9  #include <math.h>
10 #include <string.h>
11 #include "predictor.h"
12
13 class lhistory_update : public branch_update
14 {
15 public:
16     unsigned int pindex;
17     unsigned int bindex;
18 };
19
20 class lhistory_predictor: public branch_predictor
21 {
22     lhistory_update u;
23     branch_info brInf;
24
25     int p_limit;
26     int b_limit;
27
28     int p_entries;
29     int p_nbit;
30
31     int b_entries;
32     int b_length;
33
34     unsigned char *pht;
35     unsigned char *bht;
36
37     unsigned int pht_mask;
38
39 public:
40     lhistory_predictor (int x,int z)
41     {
42         p_entries=8192;
43         p_nbit=2;
44
45         pht = new unsigned char [p_entries];
46         memset (pht, 0, sizeof (pht));
47
48         bht = new unsigned char [x];
49         memset (bht, 0, sizeof (bht));
50
51         pht_mask = ((1<<(((int) log2(p_entries))-b_length))-1);
52
53         p_limit = (1<<p_nbit);
54         b_limit = (1<<b_length);
55     }
56
57     branch_update *predict (branch_info & b)
58     {
59         brInf = b;
60         if (b.br_flags & BR_CONDITIONAL)
61         {
62
63             u.bindex = (b.address & (b_entries-1));
64             u.pindex = ((b.address & pht_mask)<<b_length);
65             u.pindex |= bht[u.bindex];
66             u.direction_prediction(pht[u.pindex]>>(p_nbit-1));
67         }
68         else
69             u.direction_prediction(true);
70         u.target_prediction (0);
71         return &u;
72     }
73
74     void update (branch_update *u, bool taken, unsigned int target)
75     {
76         if (brInf.br_flags & BR_CONDITIONAL)
77         {
78             {
79                 unsigned char *c = &pht[((lhistory_update*)u)->pindex];
80                 unsigned char *d = &bht[((lhistory_update*)u)->bindex];
81
82                 if (taken)
83                 {

```

```

84         if (*c < p_limit)
85             (*c)++;
86     }
87     else
88         if (*c > 0)
89             (*c)--;
90     (*d) <= 1;
91     (*d) != taken;
92     (*d) &= b_limit;
93 }
94 }
95 };

```

## Global-History two-level predictors

```

1  /* -.-.-.-.-.
2  * File Name : ghistory.h
3  * Purpose : 2nd Assignment in AdvCompArch
4  * Creation Date : 27-05-2012
5  * Last Modified : Sun 27 May 2012 22:57:52 EEST
6  * Created By : Vasilis Gerakaris <vgerak@gmail.com>
7  -.-.-.-.-.*/
8  #ifndef GHISTORY_H
9  #define GHISTORY_H
10
11 #include <math.h>
12 #include <string.h>
13 #include "predictor.h"
14
15 class ghistory_update : public branch_update
16 {
17 public:
18     unsigned int index;
19 };
20
21 class ghistory_predictor: public branch_predictor
22 {
23     ghistory_update u;
24     branch_info brInf;
25
26     int p_limit;
27     int b_limit;
28
29     int p_entries;
30     int p_nbit;
31
32     int bhr;
33     int b_length;
34
35     unsigned char **pht;
36
37     unsigned int pht_mask;
38
39 public:
40     ghistory_predictor (int x, int y, int z)
41     {
42         bhr=0;
43
44         pht = new unsigned char *[1<<b_length];
45
46         for (int i = 0; i < (1<<b_length); ++i)
47             pht[i] = new unsigned char [p_entries>>b_length];
48         for (int i = 0; i < (1<<b_length); ++i)
49             memset(pht[i], 0, sizeof (pht[i]));
50
51         p_limit = (1<<p_nbit);
52         b_limit = (1<<b_length);
53         pht_mask = ((p_entries>>b_length)-1);
54     }
55
56     branch_update *predict (branch_info & b)
57     {
58         brInf = b;
59         if (b.br_flags & BR_CONDITIONAL)
60         {
61             u.index = (b.address & pht_mask);
62             u.direction_prediction(pht[bhr][u.index]>>(p_nbit-1));
63         }
64         else
65             u.direction_prediction (true);
66         u.target_prediction (0);
67         return &u;
68     }
69
70
71
72
73     void update (branch_update *u, bool taken, unsigned int target)

```

```

74     {
75         if (brInf.br_flags & BR_CONDITIONAL) {
76
77             unsigned char *c = &pht[bhr][((ghistory_update*)u)->index];
78
79             if (taken) {
80                 if (*c < p_limit)
81                     (*c)++;
82             }
83             else {
84                 if (*c > 0)
85                     (*c)--;
86             }
87             bhr <= 1;
88             bhr |= taken;
89             bhr &= b_limit;
90
91         }
92     }
93
94 };
95 #endif

```

## Predict.cc

```

1  // predict.cc
2  // This file contains the main function. The program accepts a single
3  // parameter: the name of a trace file. It drives the branch predictor
4  // simulation by reading the trace file and feeding the traces one at a time
5  // to the branch predictor.
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include <assert.h>
11
12 #include "branch.h"
13 #include "btfnt.h"
14 #include "ghistory.h"
15 #include "gshare_predictor.h"
16 #include "hybrid.h"
17 #include "lhistory.h"
18 #include "nbit_predictor.h"           //the .h files of the branch predictors' implementations
19 #include "ntaken.h"
20 #include "predictor.h"
21 #include "trace.h"
22
23
24 int main (int argc, char *argv[]) {
25
26     // make sure there is one parameter
27
28     if (argc != 2) {
29         fprintf (stderr, "Usage: %s <filename>.gz\n", argv[0]);
30         exit (1);
31     }
32
33     // open the trace file for reading
34
35     init_trace (argv[1]);
36
37     // initialize competitor's branch prediction code
38
39     // you can use more than one predictor in an array of predictors!!!
40
41     branch_predictor **p = new branch_predictor*[11];
42
43     p[0] = new ntaken_predictor();
44     p[1] = new btfnt_predictor();
45     p[2] = new nbit_predictor(4);
46     p[3] = new gshare_predictor();
47
48     /* Local History */
49     /* X = 2048 */
50     p[4] = new lhistory_predictor(2048, 8);
51     /* X = 4096 */
52     p[5] = new lhistory_predictor(4096, 4);
53
54     /* Global History */
55     /* X=2 BHR=4 */
56     p[6] = new ghistory_predictor(16384, 2, 4);
57     /* X=2 BHR=8 */
58     p[7] = new ghistory_predictor(16384, 2, 8);
59     /* X=4 BHR=4 */
60     p[8] = new ghistory_predictor(8192, 4, 4);
61     /* X=4 BHR=8 */
62     p[9] = new ghistory_predictor(8192, 4, 8);
63

```

```

64  /* Tournament Hybrid */
65  p[10] = new hybrid_predictor(512);
66
67
68  // some statistics to keep, currently just for conditional branches
69
70  long long int
71      tmiss[11],          // number of target mispredictions
72      dmiss[11];          // number of direction mispredictions
73
74  for(int i = 0; i < 11; i++)
75      dmiss[i] = tmiss[i] = 0;
76
77
78  // keep looping until end of file
79
80  for (;;)
81  {
82
83      // get a trace
84
85      trace *t = read_trace ();
86
87      // NULL means end of file
88
89      if (!t) break;
90
91      // send this trace to the competitor's code for prediction
92
93      branch_update *u;
94
95      /* not taken */
96      u = p[0]->predict(t->bi);
97      p[0]->update(u, t->taken, t->target);
98      dmiss[0] += u->direction_prediction() != t->taken;
99      tmiss[0] += u->target_prediction() != t->target;
100
101      /* b-taken, f-not taken */
102      ((btfnt_predictor *)p[1])->set_target(t->target > t->bi.address);
103      u = p[1]->predict(t->bi);
104      p[1]->update(u, t->taken, t->target);
105      dmiss[1] += u->direction_prediction() != t->taken;
106      tmiss[1] += u->target_prediction() != t->target;
107
108      /* 4 bit */
109      u = p[2]->predict(t->bi);
110      p[2]->update(u, t->taken, t->target);
111      dmiss[2] += u->direction_prediction() != t->taken;
112      tmiss[2] += u->target_prediction() != t->target;
113
114      /* gshare */
115      u = p[3]->predict(t->bi);
116      p[3]->update(u, t->taken, t->target);
117      dmiss[3] += u->direction_prediction() != t->taken;
118      tmiss[3] += u->target_prediction() != t->target;
119
120      /* local history, X=2048 */
121      u = p[4]->predict(t->bi);
122      p[4]->update(u, t->taken, t->target);
123      dmiss[4] += u->direction_prediction() != t->taken;
124      tmiss[4] += u->target_prediction() != t->target;
125
126      /* local history, X=4096 */
127      u = p[5]->predict(t->bi);
128      p[5]->update(u, t->taken, t->target);
129      dmiss[5] += u->direction_prediction() != t->taken;
130      tmiss[5] += u->target_prediction() != t->target;
131
132      /* global history, X=2, BHR=4 */
133      u = p[6]->predict(t->bi);
134      p[6]->update(u, t->taken, t->target);
135      dmiss[6] += u->direction_prediction() != t->taken;
136      tmiss[6] += u->target_prediction() != t->target;
137
138      /* global history, X=2, BHR=8 */
139      u = p[7]->predict(t->bi);
140      p[7]->update(u, t->taken, t->target);
141      dmiss[7] += u->direction_prediction() != t->taken;
142      tmiss[7] += u->target_prediction() != t->target;
143
144      /* global history, X=4, BHR=4 */
145      u = p[8]->predict(t->bi);
146      p[8]->update(u, t->taken, t->target);
147      dmiss[8] += u->direction_prediction() != t->taken;
148      tmiss[8] += u->target_prediction() != t->target;
149
150      /* global history X=4, BHR=8 */
151      u = p[9]->predict(t->bi);
152      p[9]->update(u, t->taken, t->target);

```

```

153     dmiss[9] += u->direction_prediction() != t->taken;
154     tmiss[9] += u->target_prediction() != t->target;
155
156     /* Tournament hybrid! */
157     u = p[10]->predict(t->bi);
158     p[10]->update(u, t->taken, t->target);
159     dmiss[10] += u->direction_prediction() != t->taken;
160     tmiss[10] += u->target_prediction() != t->target;
161 }
162
163
164     // done reading traces
165
166     end_trace ();
167
168     // give final mispredictions per kilo-instruction and exit.
169     // each trace represents exactly 100 million instructions.
170
171     for(int i = 0; i < 11; ++i)
172     {
173         printf("%d\t%.3f\n", i + 1, 1000.0 * (dmiss[i] / 1e8));
174         delete p[i];
175     }
176
177     delete [] p;
178
179     exit (0);
180 }

```

## Script για εκτέλεση benchmarks

```

1  #!/bin/bash
2  #script for running part 1
3
4  for i in `ls ../../team03`
5  do
6      echo "Running test $i"
7      echo "$i" >> ../../Results/part1/result
8      predict ../../team03/$i >> ../../Results/part1/result
9      echo " " >> ../../Results/part1/result
10 done
11 echo ""
12 cat ../../Results/part1/result

```