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

2η Σειρά Ασκήσεων

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

27 Μαΐου 2012

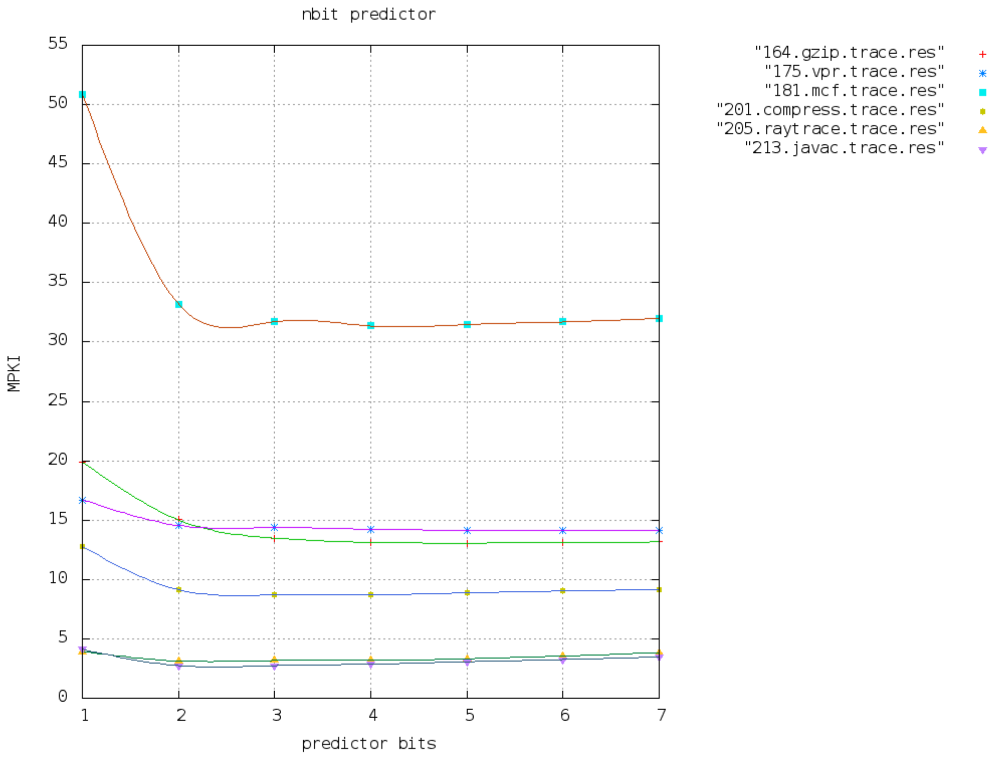
Εισαγωγή

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 εώς 7), αξιολογώντας με βάση τα MPKI (Mispredictions Per Thousand Instructions). Στο τμήμα αυτό είχαμε σταθερά BHT entries, ίσα με 16K.



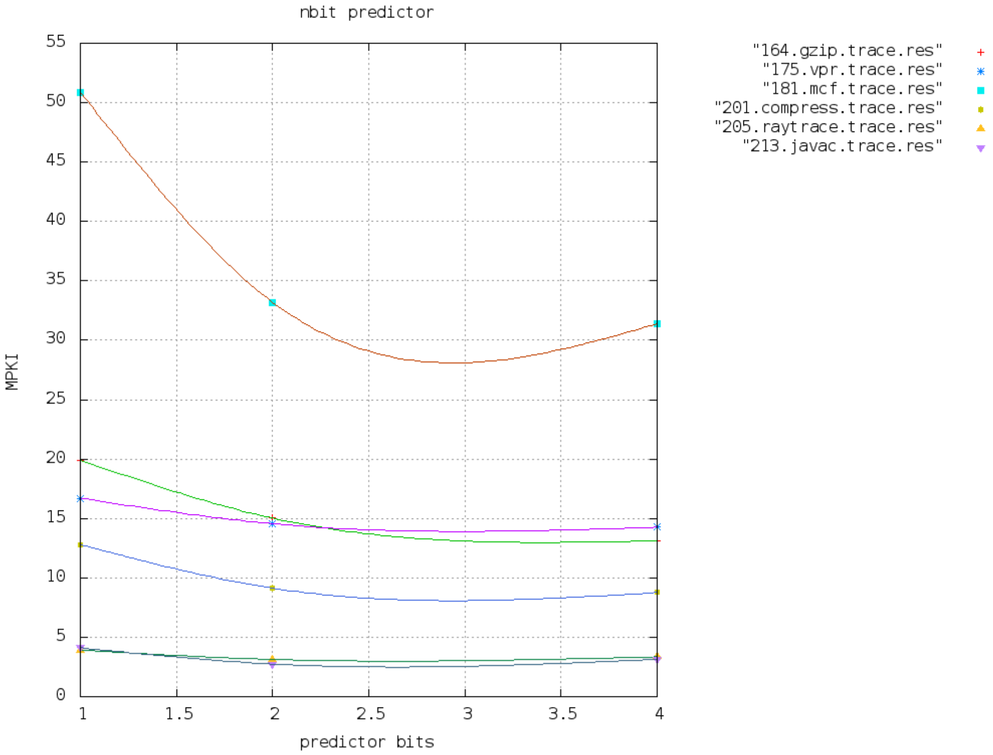
Σχήμα 1: 1 to 7 - bit predictors

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, αξιολογόντας 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

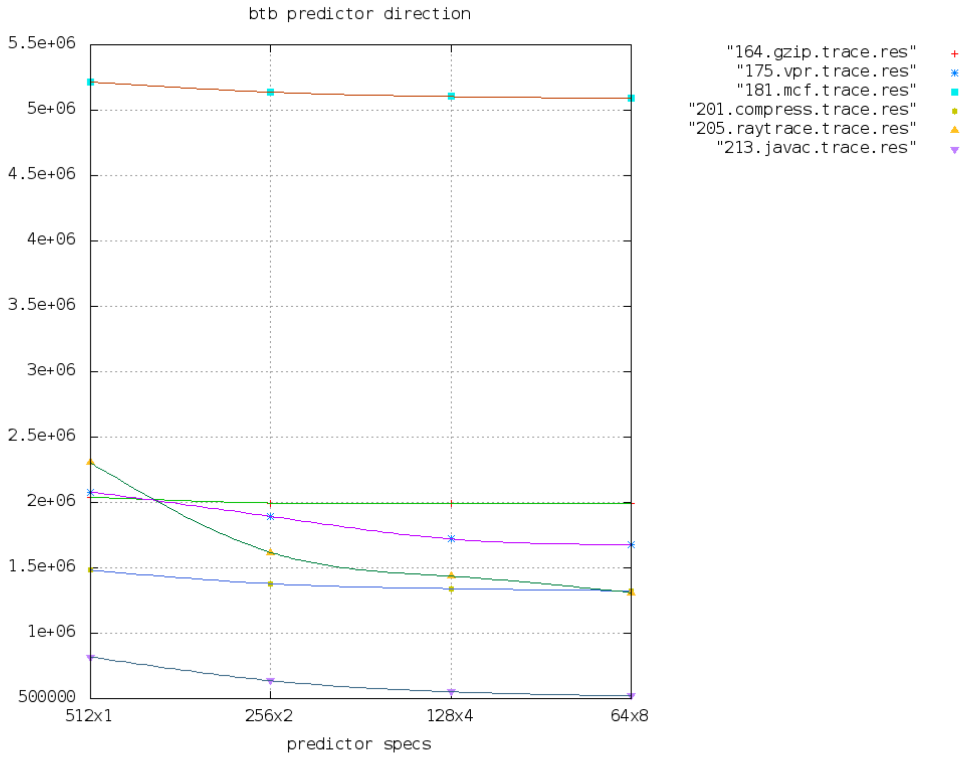


Σχήμα 2: 1,2,4 - bit predictors

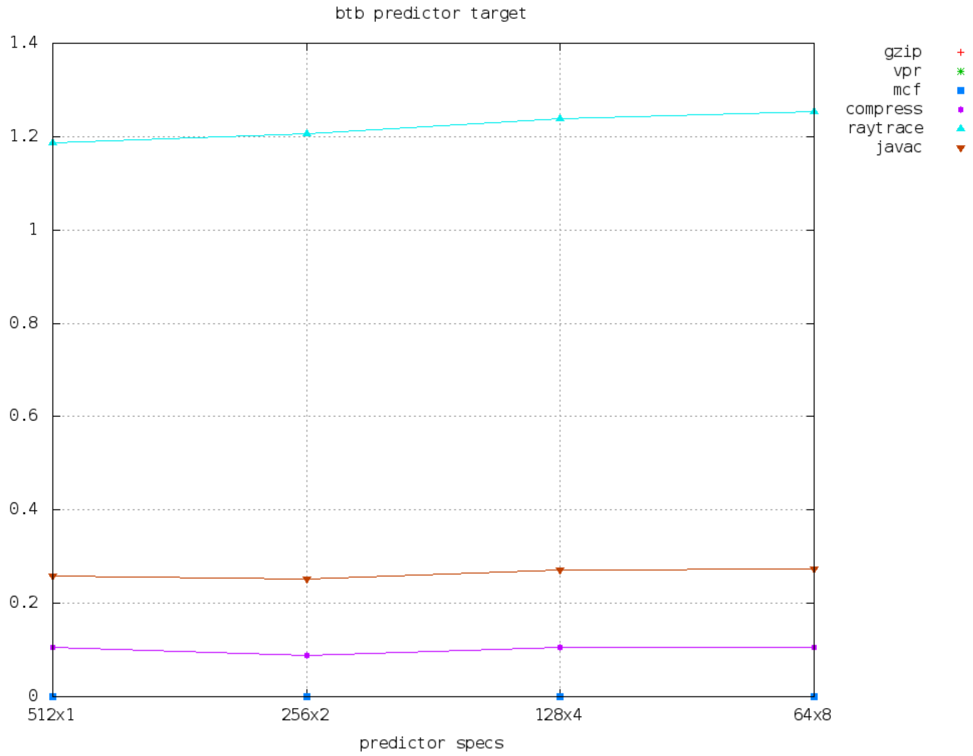
Παρατηρούμε πως ακόμα και στην περίπτωση που το hardware είναι σταθερό (32K), καλύτερη απόδοση εμφανίζει ο 4 bit predictor.

BTB predictor

btb_lines	btb_assoc
512K	1
256K	2
128K	4
64K	8



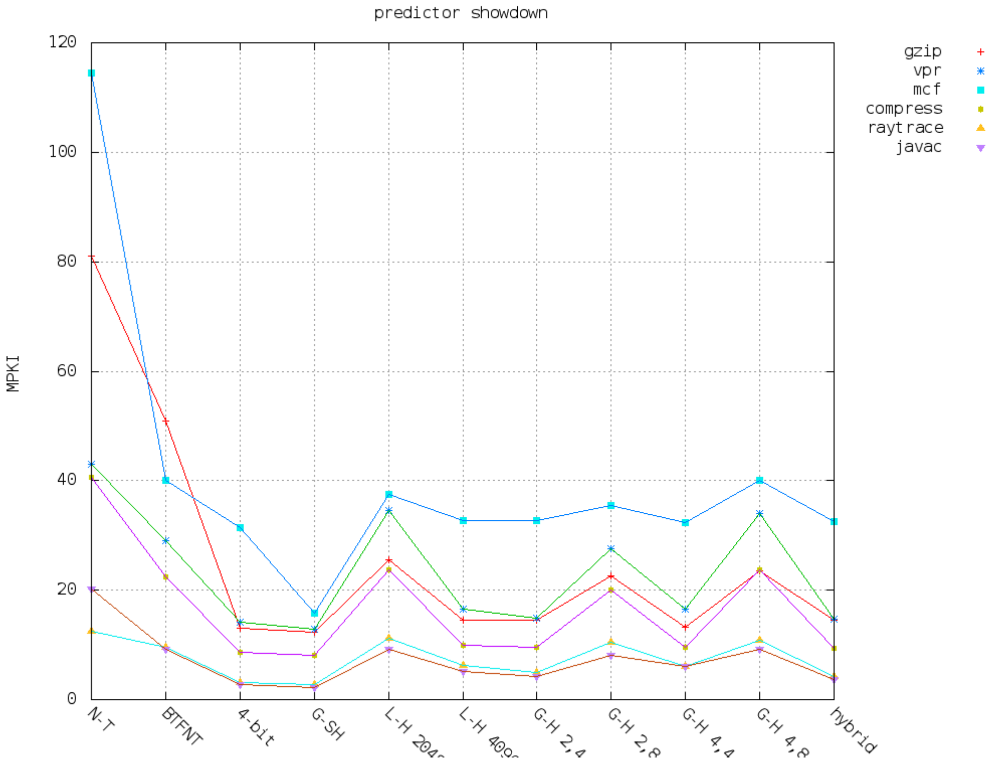
Σχήμα 3: Direction misspredictions (direction MPKI)



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

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : ntaken.h
 * Purpose : 2nd Assignment in AdvCompArch
 * Creation Date : 27-05-2012
 * Last Modified : Sun 27 May 2012 18:52:27 EEST
 * Created By : Vasilis Gerakaris <vgerak@gmail.com>
 _._._._._._._._._._._._._._._._._._._._._._.*/

#include "predictor.h"

class ntaken_update : public branch_update
{
public:
    unsigned int index;
};

class ntaken_predictor: public branch_predictor
{
    ntaken_update u;

    branch_update *predict (branch_info & b)
    {
        if (b.br_flags & BR_CONDITIONAL)
            u.direction_prediction (false);
        else
            u.direction_prediction (true);
        u.target_prediction (0);
        return &u;
    }
    void update (branch_update *u, bool taken, unsigned int target)
    {
    }
};
```

Static Backward Taken Forward Not Taken

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : btfnt.h
 * Purpose : 2nd Assignment in AdvCompArch
 * Creation Date : 27-05-2012
 * Last Modified : Sun 27 May 2012 22:58:13 EEST
 * Created By : Vasilis Gerakaris <vgerak@gmail.com>
 _._._._._._._._._._._._._._._._._._._._._.*/

#include <math.h>
#include "predictor.h"

class btfnt_predictor: public branch_predictor
{
    branch_update u;
    branch_info brInf;

    bool jump;

public:
        void set_target(bool t)
    {
        jump = t;
    }

    branch_update *predict (branch_info & b)
    {
        brInf = b;
        if (b.br_flags & BR_CONDITIONAL)
        {
            if (jump)
            {
                u.direction_prediction (false);
            }
            else
            {
                u.direction_prediction(true);
            }
        }
        else
            u.direction_prediction (true);
        u.target_prediction (0);
        return &u;
    }

    void update (branch_update *u, bool taken, unsigned int target)
    {
    }
};
```

4-bit predictor

```
// nbit_predictor.h
//
//

#include <math.h>

class nbit_update : public branch_update {
public:
        unsigned int index;
};

class nbit_predictor : public branch_predictor {
public:
#define NBP_TABLE_BITS        15  //number of entries = 2^15
        nbit_update u;
        branch_info bi;
        int counter_limit;
        int N_COUNTER_LENGTH;

        unsigned char tab[1<<NBP_TABLE_BITS];

        nbit_predictor (int length) :N_COUNTER_LENGTH(length) {
                memset (tab, 0, sizeof (tab));
                counter_limit = ((int) pow(2.0, N_COUNTER_LENGTH)) - 1;
        }

        branch_update *predict (branch_info & b) {
                bi = b;
                if (b.br_flags & BR_CONDITIONAL) {
                        u.index =  (b.address & ((1<<NBP_TABLE_BITS)-1));
                        u.direction_prediction (tab[u.index] >> (N_COUNTER_LENGTH-1));
                } else {
                        u.direction_prediction (true);
                }
                u.target_prediction (0);
                return &u;
        }

        void update (branch_update *u, bool taken, unsigned int target) {
                if (bi.br_flags & BR_CONDITIONAL) {
                        unsigned char *c = &tab[((nbit_update*)u)->index];
                        if (taken) {
                                if (*c < counter_limit) (*c)++;
                        } else {
                                if (*c > 0) (*c)--;
                        }
                }
        }
};
```

gshare predictor

```
// gshare_predictor.h
// This file contains a sample my_predictor class.
// It is a simple 32,768-entry gshare with a history length of 15.

class gshare_update : public branch_update {
public:
        unsigned int index;
};

/*
 * H klash gshare_predictor klhronomei thn klash
 * branch_predictor kai kanei override tis me8odous
 * predict kai update
 */

class gshare_predictor : public branch_predictor {
public:
#define HISTORY_LENGTH        14
#define GSP_TABLE_BITS        15
        gshare_update u;
        branch_info bi;
        unsigned int history;
        unsigned char tab[1<<GSP_TABLE_BITS];

        gshare_predictor (void) : history(0)
    {
                memset (tab, 0, sizeof (tab));
        }

        branch_update *predict (branch_info & b)
    {
                bi = b;

                /*
                 * O gshare xrhsimopoieitai mono gia conditional branches.
                 * Ta uncoditional ginontai predicted panta TAKEN
                 */

                if (b.br_flags & BR_CONDITIONAL)
        {
                        u.index = (history << (GSP_TABLE_BITS - HISTORY_LENGTH)) ^ (b.address & ((1<<GSP_TABLE_BITS)-1));
                        u.direction_prediction (tab[u.index] >> 1);
                }
        else
                        u.direction_prediction (true);

                // O gshare den kanei target prediction, gia auto to 8etoume sto 0.

                u.target_prediction (0);
                return &u;
        }

        void update (branch_update *u, bool taken, unsigned int target)
    {
                //O gshare xrhsimopoieitai mono gia conditional branches
                if (bi.br_flags & BR_CONDITIONAL) {
                        unsigned char *c = &tab[((gshare_update*)u)->index];
                        if (taken)
            {
                                if (*c < 3)
                    (*c)++;
                        }
            else
            {
                                if (*c > 0)
                    (*c)--;
                        }
                        history <<= 1;
                        history |= taken;
                        history &= (1<<HISTORY_LENGTH)-1;
                }
        }
};
```

Local-History two-level predictors

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : lhistory.h
 * Purpose : 2nd Assignment in AdvCompArch
 * Creation Date : 27-05-2012
 * Last Modified : Sun 27 May 2012 22:55:19 EEST
 * Created By : Vasilis Gerakaris <vgerak@gmail.com>
 _._._._._._._._._._._._._._._._._._._._._._.*/

#include <math.h>
#include <string.h>
#include "predictor.h"

class lhistory_update : public branch_update
{
public:
    unsigned int pindex;
    unsigned int bindex;
};

class lhistory_predictor: public branch_predictor
{
    lhistory_update u;
    branch_info brInf;

        int p_limit;
        int b_limit;

        int p_entries;
        int p_nbit;

        int b_entries;
        int b_length;

    unsigned char *pht;
    unsigned char *bht;

        unsigned int pht_mask;

public:
    lhistory_predictor (int x,int z)
    {
        p_entries=8192;
        p_nbit=2;

        pht = new unsigned char [p_entries];
        memset (pht, 0, sizeof (pht));

        bht = new unsigned char [x];
        memset (bht, 0, sizeof (bht));

        pht_mask = ((1<<(((int) log2(p_entries))-b_length))-1);

        p_limit = (1<<p_nbit);
        b_limit = (1<<b_length);
    }

    branch_update *predict (branch_info & b)
    {
        brInf = b;
        if (b.br_flags & BR_CONDITIONAL)
        {

            u.bindex = (b.address & (b_entries-1));
            u.pindex = ((b.address & pht_mask)<<b_length);
            u.pindex |= bht[u.bindex];
            u.direction_prediction(pht[u.pindex]>>(p_nbit-1));
        }
        else
            u.direction_prediction(true);
        u.target_prediction (0);
        return &u;
    }


    void update (branch_update *u, bool taken, unsigned int target)
    {
        if (brInf.br_flags & BR_CONDITIONAL)
        {
            unsigned char *c = &pht[((lhistory_update*)u)->pindex];
            unsigned char *d = &bht[((lhistory_update*)u)->bindex];

            if (taken)
            {
                if (*c < p_limit)
                    (*c)++;
            }
            else
                if (*c > 0)
                    (*c)--;
            (*d) <<= 1;
            (*d) |= taken;
            (*d) &= b_limit;
        }
    }
};
```

Global-History two-level predictors

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : ghistory.h
 * Purpose : 2nd Assignment in AdvCompArch
 * Creation Date : 27-05-2012
 * Last Modified : Sun 27 May 2012 22:57:52 EEST
 * Created By : Vasilis Gerakaris <vgerak@gmail.com>
 _._._._._._._._._._._._._._._._._._._._._.*/
#ifndef GHISTORY_H
#define GHISTORY_H

#include <math.h>
#include <string.h>
#include "predictor.h"

class ghistory_update : public branch_update
{
public:
    unsigned int index;
};

class ghistory_predictor: public branch_predictor
{
    ghistory_update u;
    branch_info brInf;

        int p_limit;
        int b_limit;

        int p_entries;
        int p_nbit;

        int bhr;
        int b_length;

    unsigned char **pht;

        unsigned int pht_mask;

public:
    ghistory_predictor (int x, int y, int z)
    {
        bhr=0;

        pht = new unsigned char *[1<<b_length];

        for (int i = 0; i < (1<<b_length); ++i)
            pht[i] = new unsigned char [p_entries>>b_length];
        for (int i = 0; i < (1<<b_length); ++i)
            memset(pht[i], 0, sizeof (pht[i]));

        p_limit = (1<<p_nbit);
        b_limit = (1<<b_length);
        pht_mask = ((p_entries>>b_length)-1);

    }

    branch_update *predict (branch_info & b)
    {
        brInf = b;
        if (b.br_flags & BR_CONDITIONAL)
        {
            u.index = (b.address & pht_mask);
            u.direction_prediction(pht[bhr][u.index]>>(p_nbit-1));
        }
        else
            u.direction_prediction (true);
        u.target_prediction (0);
        return &u;
    }



    void update (branch_update *u, bool taken, unsigned int target)
    {
        if (brInf.br_flags & BR_CONDITIONAL) {

            unsigned char *c = &pht[bhr][((ghistory_update*)u)->index];

            if (taken) {
                if (*c < p_limit)
                    (*c)++;
            }
            else {
                if (*c > 0)
                    (*c)--;
            }
            bhr <<= 1;
            bhr |= taken;
            bhr &= b_limit;

        }
    }

};
#endif
```

Predict.cc

```
// predict.cc
// This file contains the main function.  The program accepts a single
// parameter: the name of a trace file.  It drives the branch predictor
// simulation by reading the trace file and feeding the traces one at a time
// to the branch predictor.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include "branch.h"
#include "btfnt.h"
#include "ghistory.h"
#include "gshare_predictor.h"
#include "hybrid.h"
#include "lhistory.h"
#include "nbit_predictor.h"                //the .h files of the branch predictors' implementations
#include "ntaken.h"
#include "predictor.h"
#include "trace.h"


int main (int argc, char *argv[]) {

        // make sure there is one parameter

        if (argc != 2) {
                fprintf (stderr, "Usage: %s <filename>.gz\n", argv[0]);
                exit (1);
        }

        // open the trace file for reading

        init_trace (argv[1]);

        // initialize competitor's branch prediction code

        // you can use more than one predictor in an array of predictors!!!

        branch_predictor **p = new branch_predictor*[11];

    p[0] = new ntaken_predictor();
    p[1] = new btfnt_predictor();
    p[2] = new nbit_predictor(4);
    p[3] = new gshare_predictor();

    /* Local History */
        /* X = 2048 */
    p[4] = new lhistory_predictor(2048, 8);
        /* X = 4096 */
    p[5] = new lhistory_predictor(4096, 4);

    /* Global History */
        /* X=2 BHR=4 */
    p[6] = new ghistory_predictor(16384, 2, 4);
        /* X=2 BHR=8 */
    p[7] = new ghistory_predictor(16384, 2, 8);
        /* X=4 BHR=4 */
    p[8] = new ghistory_predictor(8192, 4, 4);
        /* X=4 BHR=8 */
    p[9] = new ghistory_predictor(8192, 4, 8);

    /* Tournament Hybrid */
    p[10] = new hybrid_predictor(512);


        // some statistics to keep, currently just for conditional branches

        long long int
                tmiss[11],        // number of target mispredictions
                dmiss[11];         // number of direction mispredictions

        for(int i = 0; i < 11; i++)
                dmiss[i] = tmiss[i] = 0;


        // keep looping until end of file

        for (;;)
    {

                // get a trace

                trace *t = read_trace ();

                // NULL means end of file

                if (!t) break;

                // send this trace to the competitor's code for prediction

                branch_update *u;

        /* not taken */
        u = p[0]->predict(t->bi);
        p[0]->update(u, t->taken, t->target);
        dmiss[0] += u->direction_prediction() != t->taken;
        tmiss[0] += u->target_prediction() != t->target;

                /* b-taken, f-not taken */
        ((btfnt_predictor *)p[1])->set_target(t->target > t->bi.address);
        u = p[1]->predict(t->bi);
        p[1]->update(u, t->taken, t->target);
        dmiss[1] += u->direction_prediction() != t->taken;
        tmiss[1] += u->target_prediction() != t->target;

                /* 4 bit */
        u = p[2]->predict(t->bi);
        p[2]->update(u, t->taken, t->target);
        dmiss[2] += u->direction_prediction() != t->taken;
        tmiss[2] += u->target_prediction() != t->target;

                /* gshare */
        u = p[3]->predict(t->bi);
        p[3]->update(u, t->taken, t->target);
        dmiss[3] += u->direction_prediction() != t->taken;
        tmiss[3] += u->target_prediction() != t->target;

                /* local history, X=2048 */
        u = p[4]->predict(t->bi);
        p[4]->update(u, t->taken, t->target);
        dmiss[4] += u->direction_prediction() != t->taken;
        tmiss[4] += u->target_prediction() != t->target;

                /* local history, X=4096 */
        u = p[5]->predict(t->bi);
        p[5]->update(u, t->taken, t->target);
        dmiss[5] += u->direction_prediction() != t->taken;
        tmiss[5] += u->target_prediction() != t->target;

                /* global history, X=2, BHR=4 */
        u = p[6]->predict(t->bi);
        p[6]->update(u, t->taken, t->target);
        dmiss[6] += u->direction_prediction() != t->taken;
        tmiss[6] += u->target_prediction() != t->target;

                /* global history, X=2, BHR=8 */
        u = p[7]->predict(t->bi);
        p[7]->update(u, t->taken, t->target);
        dmiss[7] += u->direction_prediction() != t->taken;
        tmiss[7] += u->target_prediction() != t->target;

                /* global history, X=4, BHR=4 */
        u = p[8]->predict(t->bi);
        p[8]->update(u, t->taken, t->target);
        dmiss[8] += u->direction_prediction() != t->taken;
        tmiss[8] += u->target_prediction() != t->target;

                /* global history X=4, BHR=8 */
        u = p[9]->predict(t->bi);
        p[9]->update(u, t->taken, t->target);
        dmiss[9] += u->direction_prediction() != t->taken;
        tmiss[9] += u->target_prediction() != t->target;

        /* Tournament hybrid! */
        u = p[10]->predict(t->bi);
        p[10]->update(u, t->taken, t->target);
        dmiss[10] += u->direction_prediction() != t->taken;
        tmiss[10] += u->target_prediction() != t->target;
    }


        // done reading traces

        end_trace ();

        // give final mispredictions per kilo-instruction and exit.
        // each trace represents exactly 100 million instructions.

    for(int i = 0; i < 11; ++i)
    {
        printf("%d\t%0.3f\n", i + 1, 1000.0 * (dmiss[i] / 1e8));
        delete p[i];
    }

        delete [] p;

        exit (0);
}
```

Script για εκτέλεση benchmarks

```
#!/bin/bash
#script for running part 1

for i in `ls ../../team03`
do
    echo "Running test $i"
    echo "$i" >> ../../Results/part1/result
    predict ../../team03/$i >> ../../Results/part1/result
    echo " ">> ../../Results/part1/result
done
echo ""
cat ../../Results/part1/result
```