Project 2: Branch Predictor

Cory Brynds

cory.brynds@ucf.edu

EEL4768: Computer Architecture
Section 0002

Due: 26 November 2023
Submitted: 12 November 2023

1.0 Introduction

In this project, a global branch predictor was simulated using the gshare predictor type. In gshare, a prediction table of 2^M bits is created, and entries are indexed into the table using a hash index. The hash index is generated by a hash function, which XORs M bits of the branch address with the N-bit Global History Register (GHR), which is zero-extended to M bits. In the prediction table, prediction values are two bits, and are initialized to 2, which corresponds to “Weakly-Taken”.

2.0 Experimental Explanation

2.1 Source Code

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>

using namespace std;

int main(int argc, char* argv[]){
   ifstream streamFile;
   unsigned int N, M, numAddresses = 0, numEntries, numMispredictions = 0, prediction = 0, actualDecision, hashIndex;
   unsigned long long int currentAddr, curAddrMBits, GHR = 0, zeroExtendedGHR, bitMask = 0;
   string address, predictorType, streamFileLine;
   char decision;
   vector<int> bufferTable;

   if (argc != 5){
       cerr << "Error! Invalid number of arguments\n";
       exit(1);
   }

   predictorType = argv[1];
   M = atoi(argv[2]);
   N = atoi(argv[3]);
```

```
streamFile.open(argv[4], ios::in);

if (!streamFile.is_open()){
    cerr << "Error! File didn't open" << endl;
    exit(1);
}

// Set the number of entries to the buffer table and initialize the table
numEntries = pow(2, M);

for (int i = 0; i < numEntries; i++)
    bufferTable.push_back(2);

// Initilize bit mask to be M bits
for (int i = 0; i < M; i++){
    bitMask <<= 1;
    bitMask |= 1;
}

// Process all memory accesses
while(getline(streamFile, streamFileLine))
{
    numAddresses++;
    istringstream iss(streamFileLine);
    iss >> address >> decision;
    currentAddr = (stoull(address, nullptr, 16));

    // Extract M bits from the address
    curAddrMBits = (currentAddr >> 2) & bitMask;

    actualDecision = (decision == 't') ? 1 : 0;

    // Extend GHR to be M bits
    if (N > 0)
        zeroExtendedGHR = GHR << (M-N);
    else
        zeroExtendedGHR = GHR << (M-1);

    // Create hash value by XORing the M address bits and the GHR
    hashIndex = curAddrMBits ^ zeroExtendedGHR;

    // If entry at hash index is Weakly Taken or Strongly Taken, predict taken
```

```
        prediction = (bufferTable[hashIndex] > 1) ? 1 : 0;

        // If actually taken and entry isn't strongly taken, increment entry
        if (actualDecision == 1 && bufferTable[hashIndex] < 3)
            bufferTable[hashIndex]++;
        // Else if actually not taken and entry isn't strongly not taken, decrement
entry
        else if (actualDecision == 0 && bufferTable[hashIndex] > 0)
            bufferTable[hashIndex]--;

        // Update number of mispredictions if prediction is false
        if (prediction != actualDecision)
            numMispredictions++;

        // Set next GHR
        GHR = GHR >> 1;

        if (N > 0)
            GHR |= (actualDecision << (N-1));
        else
            GHR = actualDecision;
    }

    streamFile.close();
    cout << M << " " << N << " " << (float)numMispredictions/numAddresses*100 << "%" <<
endl;

    return 0;
}
```

2.2 Test Script

To test the performance of the branch predictor at different bit values for M and N, the following bash script was written to run all of the test cases and output the results to a text file. Commands are given in the form

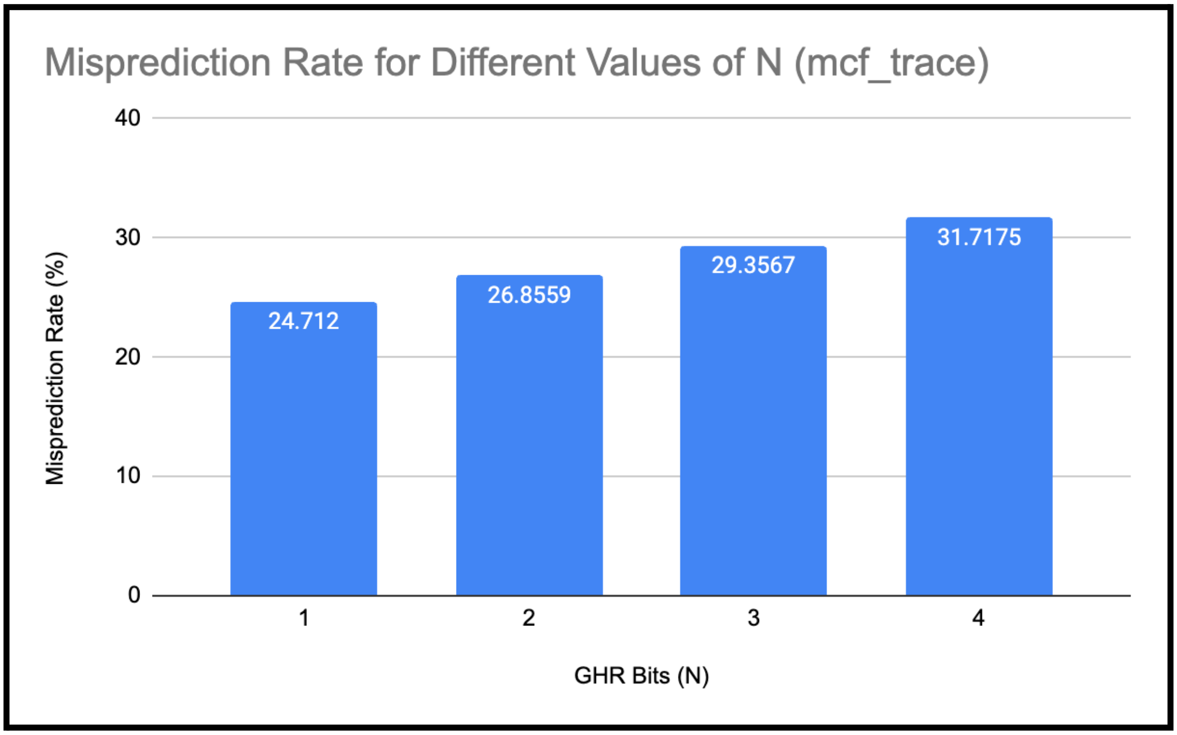
./sim <predictor type> <M> <N> <test case filepath>

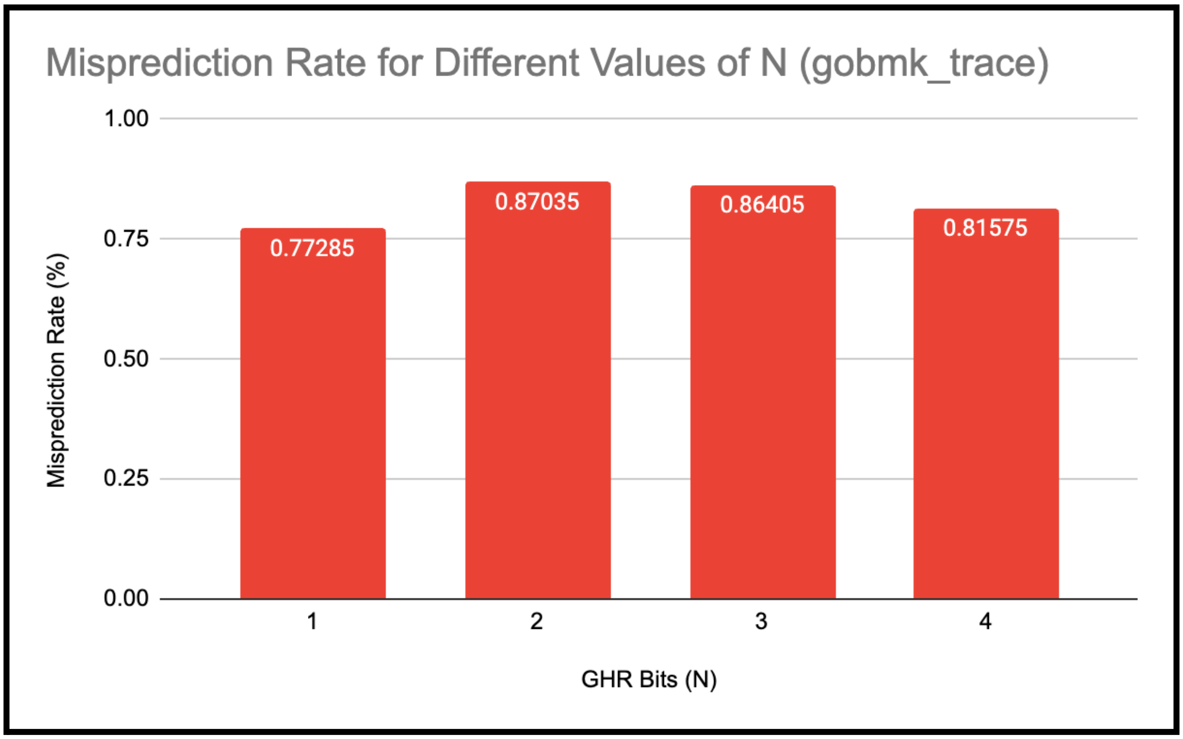
```
#!/bin/bash

TEST_CASES=(
    # Part A: Vary GHR Bits (N)
    "./sim gshare 4 1 testcases/mcf_trace.txt"
```

```
   "./sim gshare 4 2 testcases/mcf_trace.txt"
   "./sim gshare 4 3 testcases/mcf_trace.txt"
   "./sim gshare 4 4 testcases/mcf_trace.txt"

   "./sim gshare 4 1 testcases/gobmk_trace.txt"
   "./sim gshare 4 2 testcases/gobmk_trace.txt"
   "./sim gshare 4 3 testcases/gobmk_trace.txt"
   "./sim gshare 4 4 testcases/gobmk_trace.txt"

   # Part B: Vary Address Bits (M) with N = 4
   "./sim gshare 4 4 testcases/mcf_trace.txt"
   "./sim gshare 5 4 testcases/mcf_trace.txt"
   "./sim gshare 6 4 testcases/mcf_trace.txt"
   "./sim gshare 7 4 testcases/mcf_trace.txt"

   "./sim gshare 4 4 testcases/gobmk_trace.txt"
   "./sim gshare 5 4 testcases/gobmk_trace.txt"
   "./sim gshare 6 4 testcases/gobmk_trace.txt"
   "./sim gshare 7 4 testcases/gobmk_trace.txt"

   # Part C: Vary Address Bits (M) with N = 0
   "./sim gshare 4 0 testcases/mcf_trace.txt"
   "./sim gshare 5 0 testcases/mcf_trace.txt"
   "./sim gshare 6 0 testcases/mcf_trace.txt"
   "./sim gshare 7 0 testcases/mcf_trace.txt"

   "./sim gshare 4 0 testcases/gobmk_trace.txt"
   "./sim gshare 5 0 testcases/gobmk_trace.txt"
   "./sim gshare 6 0 testcases/gobmk_trace.txt"
   "./sim gshare 7 0 testcases/gobmk_trace.txt"
)

for ARGS in "${TEST_CASES[@]}"; do
   echo "Testing with arguments: $ARGS"
   $ARGS

done
```

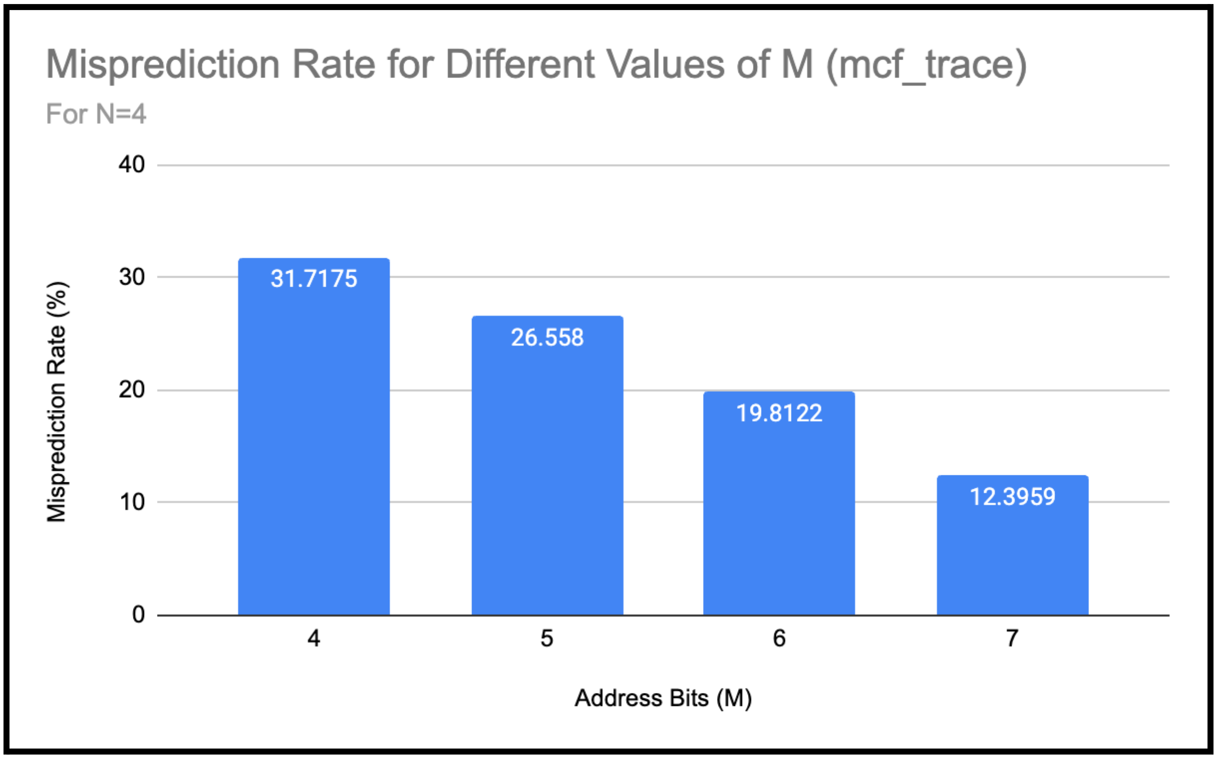
3.0 Performance Evaluation

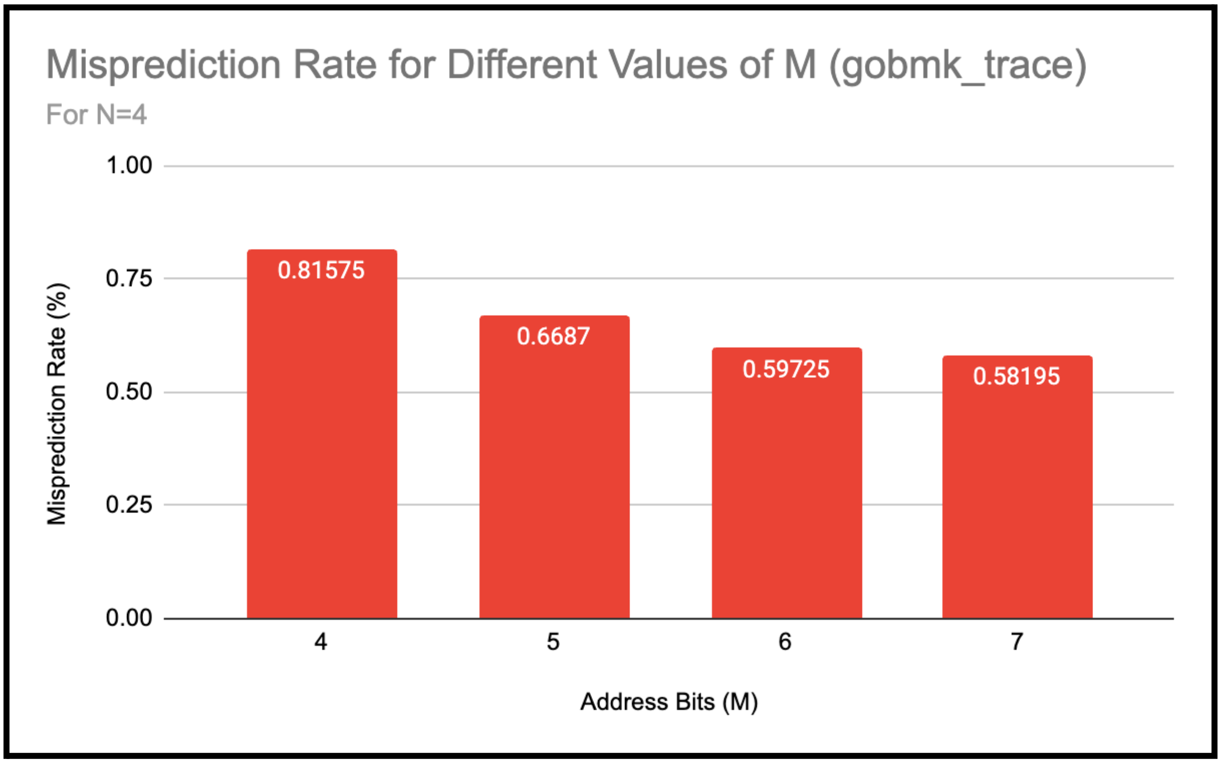
2.1 Misprediction Rate for Different Values of N



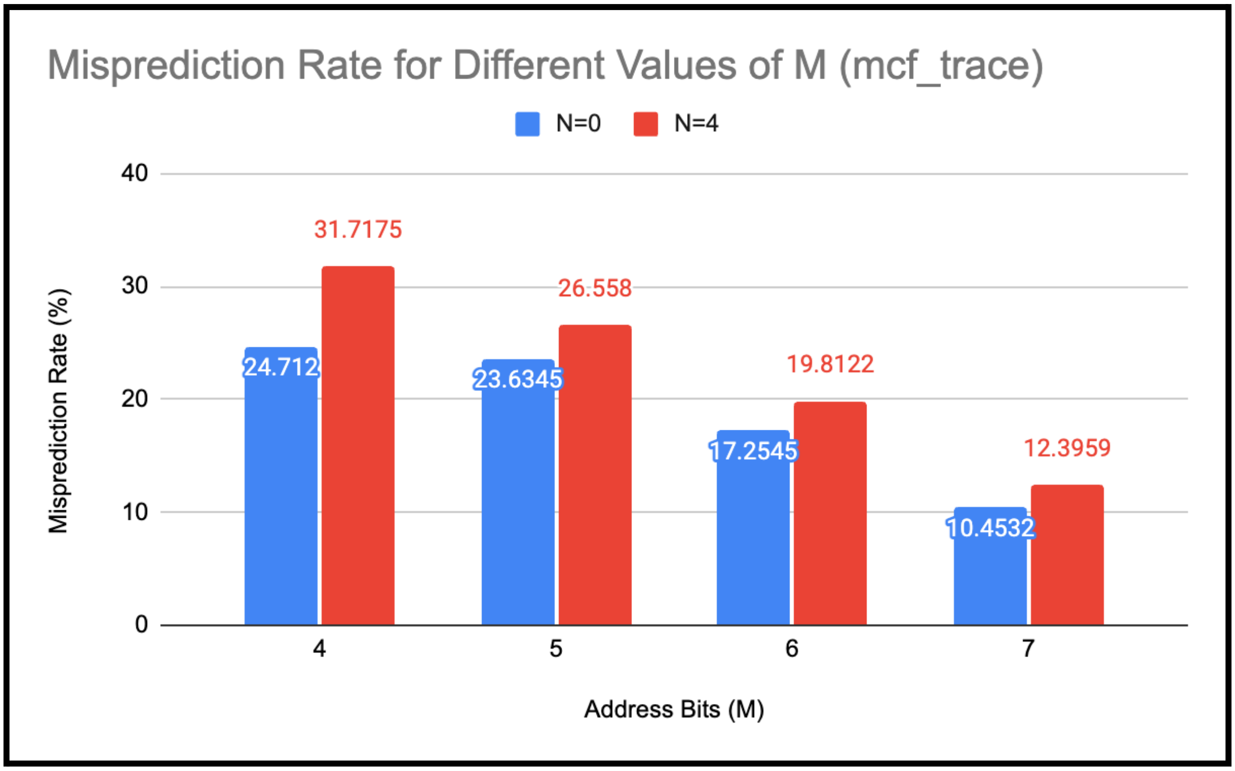


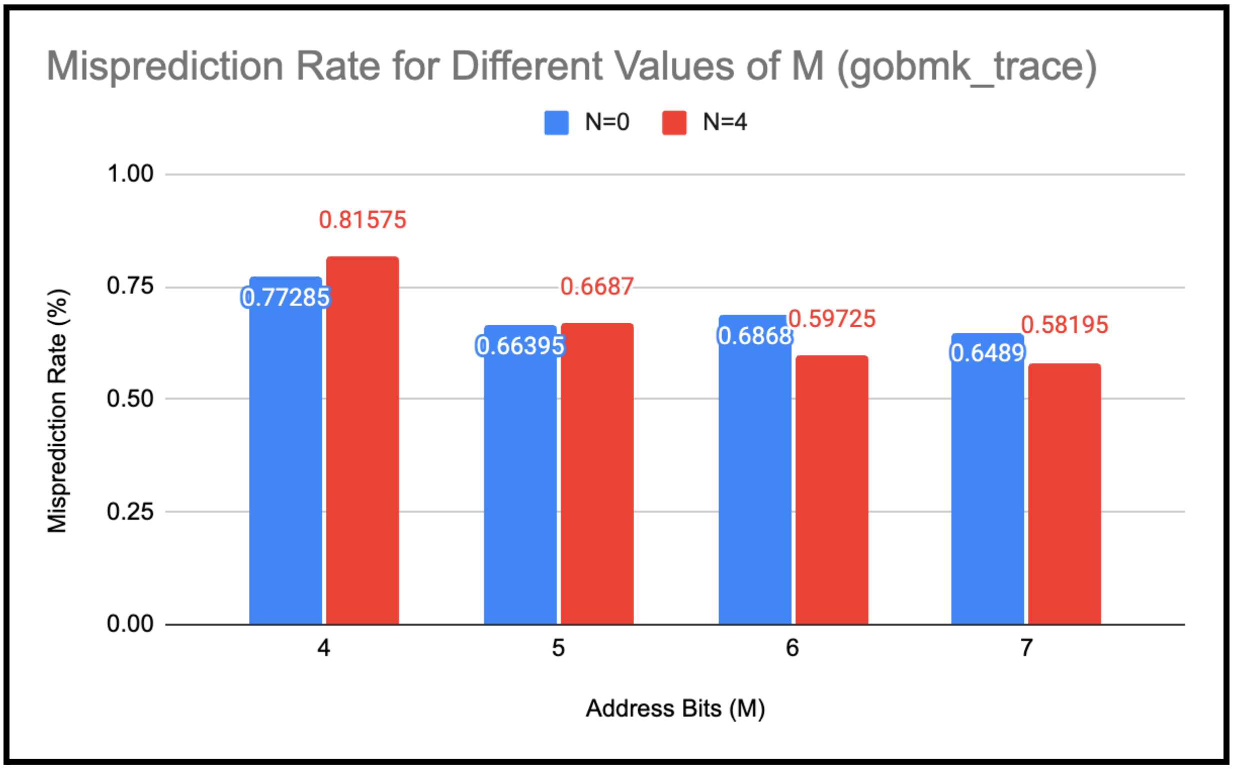
2.2 Misprediction Rate for Different Values of M for N = 4





2.3 Misprediction Rate for Different Values of M for N = 0





3.0 Results Analysis & Conclusion

As seen in the charts above, the most effective way to reduce the misprediction rate is to increase the number of bits M that are taken from the address to form the hash index. This was proven to decrease the misprediction rate in both the mcf and gobmk test cases. In increasing M, however, this will exponentially increase the size of the prediction table, which has 2^M entries. A performance vs memory usage tradeoff must be considered.

For varying the values of N, it appears to be more effective to use a smaller number of bits to represent the global history register. For the mcf trace file, as N was increased, the misprediction rate increased, and it was lowest when N=0. A similar trend was observed in the gobmk trace file; however, the misprediction rate decreased for N=4, compared to N=3. In part 3, it was seen that increasing N from 0 to 4 unanimously increased the misprediction rate for the mcf trace file. However, in the gobmk trace file, increasing N from 0 to 4 only increased the misprediction rate for M = 4 and M = 5; in the other two cases, the misprediction rate decreased from N = 0 to N = 4.

In conclusion, when designing a branch predictor with gshare, it is advantageous to use a larger amount of bits to index each address (trading off memory complexity) and a smaller amount of bits for the global history register.