Lab6-Cache Simulator

Group 17　109550135 范恩宇 109550156 曾偉杰

Part I.　Detailed description of the implementation :

1. Direct-mapped cache :

First declare a function “ToBinary()”for turning a string into binary .

Since the block size and range of index must be binary for calculation , declare a function ‘‘BitNum’’ to turn entered decimal number into binary .

Then in ‘’direct_mapped()” , first assign 3 variables “byte_offset” (binary block size) , “index” (binary (cache_size / block_size)) , and “tag”(32 - byte_offset - index) . After that , create an unordered_map “cache” to store data of cache and read the file .

When reading file , first assign “bit_str”(binary “str”) , “tag_str” ((bit_str, 0, tag)) , “index_str”((bit_str, tag, index)) , then looking for “index_str” in “cache”. If founded , then increase “hit_num” by one or make “cache[index_str]” = “tag_str” , depends on whether “cache[index_str]” = “tag_str” . Otherwise , “cache[index_str]” = “tag_str” .

In the end of each time of file reading , increase “total_num” by 1 . In addition , break from the while loop when reaching the bottom of file . In the end , close the file and return “hit_num” / ”total_num” as the value.

```cpp
#include "direct_mapped_cache.h"
#include "string"
#include <fstream>
#include <bitset>
#include <unordered_map>

using namespace std;

string ToBinary(string& str) {
    string binary;
    int len = str.size();
    for (int i = 0; i < len; ++i) {
        if (str[i] >= 'a' && str[i] <= 'f') {
            binary += bitset<4>((str[i] - 'a' + 10)).to_string();
        } else {
            binary += bitset<4>((str[i] - '0')).to_string();
        }
    }
    if (binary.size() == 28) {
        binary = "0000" + binary;
    }
    return binary;
}

int BitNum(int num) {
    int n = 0;
    while (num >= 2) {
        num /= 2;
        n++;
    }
    return n;
}
```

```cpp
float direct_mapped(string filename, int block_size, int cache_size) {
    int total_num = 0;
    int hit_num = 0;

    /*write your code HERE*/
    ifstream file;
    string str;
    int byte_offset = BitNum(block_size);
    int index = BitNum((cache_size / block_size));
    int tag = 32 - byte_offset - index;
    unordered_map<string, string> cache;
    file.open(filename);
    while (file >> str) {
        if (file.eof()) {
            break;
        }
        string bit_str = ToBinary(str);
        string tag_str(bit_str, 0, tag);
        string index_str(bit_str, tag, index);
        if (cache.count(index_str)){
            if (cache[index_str] == tag_str){
                hit_num++;
            }
            else{
                cache[index_str] = tag_str;
            }
        }
        else{
            cache[index_str] = tag_str;
        }
        total_num++;
    }
    file.close();
    return (float)hit_num / total_num;
}
```

2. Set-associative cache :

In the beginning , declare a function ‘‘BitNum_sec’’ to turn entered decimal number into binary , just like we did in the direct-mapped cache , using different name is just for preventing multiple definition .

Then in ‘’set_associative()” , first assign 3 variables “byte_offset” (binary block size) , “index range” (cache_size / (block_size*way)) , “index” (binary index range) and 2 arrays “cache”(store tag of each address and each set has n blocks , n = way) , “used_block”(for storing number of used blocks in set) . After that , read the file .

When reading file , first assign hex string “str” to unsigned int “bit_str” , “idx” = (bit_str>> byte_offset) % idx_range , “tag” = bit_str >> (byte_offset + index) , then run a loop for used_block[idx] times . In the for loop , if “cache[idx][i]” = “tag” , then make “cache[idx][j]” = “cache[idx][j+1]” (for j = i , i+1 , … ,used_block[idx]-2) , “cache[idx][used_block[idx]-1]” = “tag” , increase hit number by 1 , and set current “hit” be true .

Also in the for loop , when doesn’t hit , if “used_block[idx]” = “way” , then “cache[idx][j]” = “cache[idx][j+1]”(for j = 0,1,…,way-2) and “cache[idx][way-1]” = “tag” . Otherwise , “cache[idx][used_block[idx]]” = “tag” and increase “used_block[idx]” by 1 . In the end of each time of file reading , increase “total_num” by 1 .

In the end of each time of file reading , increase “total_num” by 1 . In addition , break from the while loop when reaching the bottom of file . In the end , close the file and return “hit_num” / ”total_num” as the value.

```cpp
#include "set_associative_cache.h"
#include "string"
#include <fstream>

using namespace std;

int BitNum_sec(int num) {
    int n = 0;
    while (num >= 2) {
        num /= 2;
        n++;
    }
    return n;
}

float set_associative(string filename, int way, int block_size, int cache_size){
    int total_num = 0;
    int hit_num = 0;

    /*write your code HERE*/
    ifstream file;
    string str;
    int byte_offset = BitNum_sec(block_size);
    int idx_range = cache_size / (block_size*way);
    int index = BitNum_sec(idx_range);
    int cache[idx_range][way];
    int used_block[idx_range]={};

    file.open(filename);

    while (file >> str) {
        if (file.eof()) {
            break;
        }
        bool hit = false;
        unsigned int bit_str = strtoul(str.c_str(), NULL, 16);
        unsigned int idx = (bit_str>> byte_offset) % idx_range;
        unsigned int tag = bit_str >> (byte_offset + index);

        for(int i = 0; i < used_block[idx]; i++){
            if(cache[idx][i] == tag){
                for(int j = i; j < used_block[idx]-1; j++){
                    cache[idx][j] = cache[idx][j+1];
                }
                cache[idx][used_block[idx]-1] = tag;
                hit_num++;
                hit = true;
                break;
            }
        }

        if(hit == false){
            if(used_block[idx] == way){
                for(int j = 0; j < way-1; j++){
                    cache[idx][j] = cache[idx][j+1];
                }
                cache[idx][way-1] = tag;
            }
            else{
                cache[idx][used_block[idx]] = tag;
                used_block[idx]++;
            }
        }
        total_num++;
    }

    file.close();
    return (float)hit_num/total_num;
}
```

Part II. Implementation results :

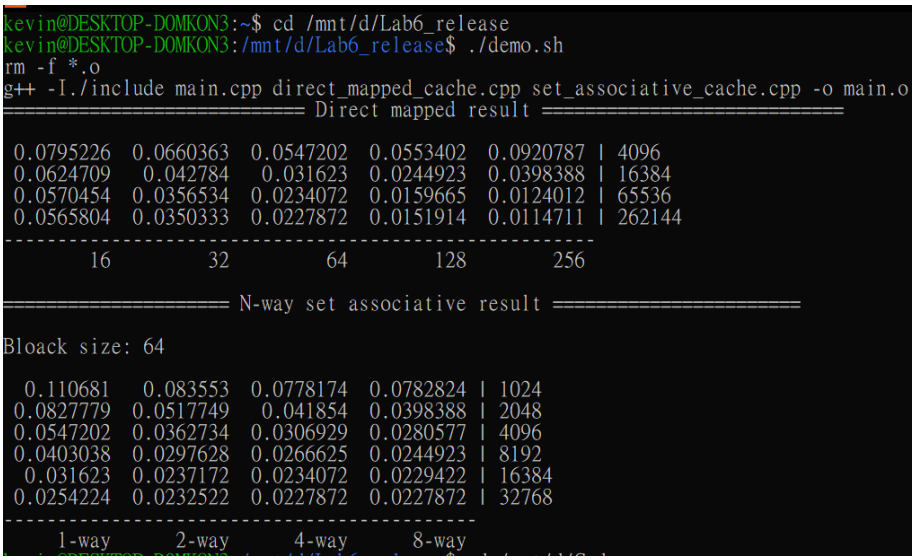
Direct-mapped cache :

Miss rate		Block size (Byte)				
		16	32	64	128	256
Cache size (Byte)	4k	0.0795226	0.0660363	0.0547202	0.0553402	0.0920787
	16k	0.0624709	0.042784	0.031623	0.0244923	0.0398388
	64k	0.0570454	0.0356534	0.0234072	0.0159665	0.0124012
	256k	0.0565804	0.0350333	0.0227872	0.0151914	0.0114711

Set-associative cache :

Miss rate		Associativity (Block size:64B)			
		1-way	1-way	1-way	1-way
Cache size (Byte)	1k	0.110681	0.083553	0.0778174	0.0782824
	2k	0.0827779	0.0517749	0.041854	0.0398388
	4k	0.0547202	0.0362734	0.0306929	0.0280577
	8k	0.0403038	0.0297628	0.0266625	0.0244923
	16k	0.031623	0.0237271	0.0234072	0.0229422
	32k	0.0254224	0.0232522	0.0227872	0.0227872

Full Result :



Part III. Problems encountered and solutions :

Since this lab is implemented by C++ and doesn't require connecting so many modules with each other , it's not as complicate as the previous ones . However , there's also something that we need to be cautious of when implementing this lab .

First , it took us some time to think about how to store the data of cache . Eventually , we tried unordered map and 2D array to achieve our goal .

Second , we set the "decimal to binary function" in two cache files with the same name in the beginning , and here comes the multiple definition error . We didn't know what went wrong at first , since there's only a function called "BitNum" in each file . In the end , we found out that when running demo.sh , it went through two cache files , thus the two functions with the same name conflicted . After giving the two functions different names , the code worked .

It's really nice to work on a lab implemented by C++ in the end of the semester , Verilog is sometimes really complicated .