Computer Architecture Final Project Report

106072237 黃采瑩

First of all, the flow of my code is the demonstration below.

Read the settings in the cache file ->
Output the settings ->
Read one requirement from the inference file ->
Check the cache to see if there is same index with same tag already exist ->
    If yes ->
    Output hit
    If no ->
    Output miss and decrease miss rate ->
    Check if there are free spaces remaining ->
        If yes ->
        Fill up the empty space
        If no ->
        Use NRU replacement policy ->
            (Set all flag bits initially 0, if referenced then 1)
            Check if all bits are referenced ->
                If yes ->
                Set all bits to zero
            Find the tag with flag = 0 nearest to the head, replace it
Load a new requirement ->
Till the requirements are all loaded, output miss count

Let's see the code with the flow, I read cache with file.open(), since only reading from the file so the setting is ios::in. Due to the format of the cache file, I set the reading breakpoint when the file is about to enter the setting value, which is at the point when the char is a blank space. To prevent the setting value more or equal than ten, I multiplex the previous value with ten and added to the new added one.

```
    file.open(argv[1], ios::in);
    for(int i = 0; i < 4; i++){
        setting[i] = 0;
        while(file.get(c))
            if(c == ' ') break;
        while(file.get(c)){
            if(c >= '0' && c <= '9')
```

```
                    setting[i] = c - '0' + setting[i]*10;
                else
                    break;
            }
        }
        file.close();
```

To output the setting, I open the file with another fstream which is named output, then I print them in the required order. The indexing bits are printed from the biggest to the smallest.

```
        output.open(argv[3], ios::out);
        output << "Address bits: " << setting[0] << endl;
        output << "Block size: " << setting[1] << endl;
        output << "Cache sets: " << setting[2] << endl;
        output << "Associativity: " << setting[3] << "\n\n";
        output << "Offset bit count: " << offset << endl;
        output << "Indexing bit count: " << index << endl;
        output << "Indexing bits:";
        for(int i = offset + index - 1; i >= offset; i--)
            output << " " << i;
        output << "\n\n" <<    ".benchmark testcase1\n";
```

To read the requirements, first open the file required to be read, then us getline to consume the first line, since we don't have to use it. Then enter while loop to do the real checking.

```
        file.open(argv[2], ios::in);
        getline(file, requirement);
        while(1){
            getline(file, requirement);
```

To save the tag and index, I transform them into decimal, so it is more convenient for me to do the continuing calculation. I use pow to get the power of two.

```
            // to tranform index input from binary to decimal
            count = 0;
            index_input = 0;
            for(int i = setting[0] - offset - 1; i >= setting[0] - offset - index; i--){
                index_input += (requirement[i] - '0') *
                pow(2.0, static_cast<double>(count));
```

```
            count++;
        }

        // to transform tag from binary to decimal
        count = 0;
        tag_input = 0;
        for(int i = setting[0] - offset - index - 1; i >= 0; i--){
            tag_input += (requirement[i] - '0') *
            pow(2.0, static_cast<double>(count));
            count++;
        }
```

To check if the instruction is hit or miss, first set a flag that will be set to 0 if cache is hit. Since the cache is initialize to zero, it is possible that the instruction with index and tag both equal to zero, never entered cache before, but being judged as hit, so I set another flag called flag_twozero to 1, which means that the instruction had never entered cache before, and will go to cache miss function directly.
If the cache is hit, also set NRU[index_input][i] to 1, meaning that the cache block has been referenced.

```
    // to see miss or hit
        flag_miss = 1;
        // hit
        for(int i = 0; i < setting[3]; i++){
            cout << "entered hit testing function\n";
            if(cache[index_input][i] == tag_input){
                // to detect the index = 0, i = 0 first enter issue
                if(index_input == 0 && tag_input == 0 && flag_twozero == 1){
                    flag_twozero = 0;
                    break;
                }
                // to detect normal issue
                output << requirement << " hit\n";
                flag_miss = 0;
                NRU[index_input][i] = 1;
            }
        }
```

If it is cache miss, check if there is still empty cache. I originally check the value in the

cache to decide if it is empty, since it is initialized to zero. However, I found out that under the condition of tag is zero, the cache will be chosen as valid cache accidently, so I use another variable called valid to check the cache is usable.

```
        // miss
        if(flag_miss == 1){
                output << requirement << " miss\n";
                miss_count++;

                // to see if there's any free space
                flag_free = 0;
                for(int i = 0; i < setting[3]; i++){
                        if(valid[index_input][i] == 0){
                                valid[index_input][i] = 1;
                                cache[index_input][i] = tag_input;
                                flag_free = 1;
                                break;
                        }
                }
```

If there is no empty cache, NRU replacement must be used. I set NRU[index_input][i] initially to zero, if it is referenced, it will be change to one. In the NRU replacement section, first check if all the bits are referenced (all the bits are set to one), in order to decide whether to set all the bits in NRU[index_input] to zero.
After checking the value in NRU[index_input], find the zero (not referenced cache) that is nearest to head and replace it.

```
                // if there's no free space
                if(flag_free == 0){
                        // replacement
                        flag_zero = 1;
                        // if all are referenced
                        for(int i = 0; i < setting[3]; i++){
                                if(NRU[index_input][i] != 1){
                                        flag_zero = 0;
                                        break;
                                }
                        }
                        // if all are refrenced, set all to zero
                        if(flag_zero == 1){
```

```
                    for(int i = 0; i < setting[3]; i++)
                        NRU[index_input][i] = 0;
                }
                // find the zero nearest to the head
                for(int i = 0; i < setting[3]; i++){
                    if(NRU[index_input][i] == 0){
                        cache[index_input][i] = tag_input;
                        break;
                    }
                }
            }
```

After reading and checking a requirement, loop to read new requirements.
By end of reading requirement, output the total cache miss count and end the files opened.

```
    output << ".end\n\n" << "Total cache miss count: " << miss_count << endl;
    file.close();
    output.close();
```