Cache Replacement Policies

Write a C++ program to simulate the various cache page replacement policies. There are six different replacement policies. They are:

1. Direct map
2. Least Recently Used (LRU)
3. First-In-First-Out (FIFO)
4. Last-In-First-Out (LIFO)
5. Optimal
6. Random

Start with the given template program. The direct map and random replacement policies have already been implemented in the template program, so you'll only need to do the remaining four.

You will use a fixed cache size of 4, and the sample pages to be fetched in are

8, 3, 2, 4, 6, 4, 2, 8, 2, 3, 6, 2, 4, 8, 3

Your program needs to put these pages, one at a time, in the order given into the cache. If there is a collision, i.e., the cache location is already occupied by another page, then you need to resolve it using the different replacement policy as discussed in the chapter (Chapter 5 slides 19 to 55).

Your program needs to keep track of the number of hits and misses, and print these two numbers out at the end.

Following the examples given in Chapter 5 slides 56 to 63, the results for each replacement policies are:

- Direct map: 4 hits and 11 misses
- LRU: 4 hits and 11 misses
- FIFO: 6 hits and 9 misses
- LIFO: 7 hits and 8 misses
- Optimal: 8 hits and 7 misses
- Random: ? hits and ? misses

Executable

There is an executable file for you to try out.

Sample run

```
Select a cache replacement strategy:
  1 - Direct map
  2 - Least Recently Used (LRU)
  3 - First-In-First-Out (FIFO)
  4 - Last-In-First-Out (LIFO)
  5 - Optimal
  6 - Random
```

```
  0 - End
  ? 1

Direct map: 4 hits and 11 misses
```

Some useful code

```
#include <bitset>  // needed to convert/print number in binary
using namespace std;

unsigned int d = 0xbe600000;              // 0x for hex constant
d = 0b10111110011000000000000000000000;  // 0b for bin constant
cout << sizeof(d) << endl;
cout << "dec=" << d << endl;
cout << "hex=" << hex << d << endl; // print d in hex
bitset<32> binary(d);               // convert 32-bit decimal d to bin
cout << "bin=" << binary << endl;   // print d in binary

binary[2] = 1;                      // set bit 2 to a 1
```