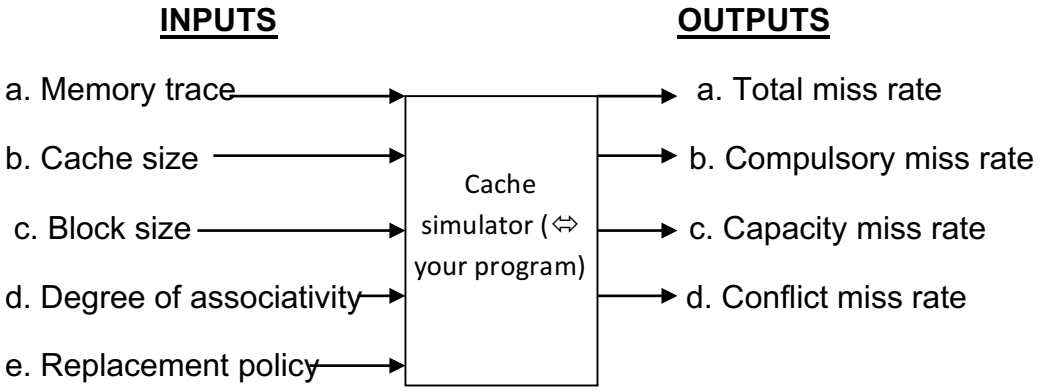
**Objectives:**

1. To write a cache simulator and to examine the performance of various cache organizations from a given trace file,
2. To be familiar with the programming requirements so that, under Lab Exam Conditions, you will be able to program a small subset of the simulator.

**Requirements**

You are required to program (in a high level language such as C, C++, Java) and implement a cache simulator which will have the following inputs and outputs:-




Memory trace ⇔ smalltex.din (during code development microtex.din – see later) and each address is the current address, in RAM, of a Byte required by the CPU.

Cache size ⇔ size of cache in bytes

Block size ⇔ size of blocks in bytes

Degree of associativity ⇔ direct (1-way), 2-way, 4-way, 8-way, and fully associative

Replacement policy ⇔ Random and LRU are to be implemented.

Total miss rate = compulsory + capacity + conflict miss rates.

**Requirements for your cache simulator**

Using your cache simulator and using smalltex.din as your memory trace determine the total miss rate, compulsory miss rate, capacity miss rate, and conflict miss rate for the following cache configurations by varying the inputs as suggested below. Examine your results/observations in each case and compare them to a run using dineroiv (a commercial cache simulator). This exercise is to validate the operation of your simulator compared to a commercial simulator and to familiarize yourself with the results that occur for each architecture. Comparing your results with dinero validates your cache simulator. Answering the questions below will help you gain a deeper understanding of the function/operation of caches with respect to various trace files and any patterns within trace files.

1. Keeping block size constant (say 64 bytes) compare the different replacement policy's with several cache sizes (say 16, 64, 256, 512KB) and associativity

(direct, 2-, 4-, 8-way, fully associative). Note the trends and confirm your observations from your cache simulator with dineroiv

2. Keeping replacement policy constant (for random and then LRU) and block size constant (say 64 bytes) collect total, compulsory, capacity, and conflict miss rates for each of the following cache organizations and compare with the results from dineroiv.:-
3. Cache sizes 4, 16, 64, 128, 256, 512 KB,
4. Degree of associativity 1-way, 2-way, 4-way, 8-way, fully.
5. To ensure that your cache simulator is fully operational you should be able to perform the following:-
   a. Accept any block sizes from 2 bytes to 2048 bytes,

   b. Accept any cache size from 1Kb to 8MB,

   c. Accept either LRU or random replacement policy for each run,

   d. Accept the following degree of associativity, Direct, 2-way, 4-way, 8-way, and fully associative.

6. Additionally, to help you anaylze the operation of caches with a trace file be prepared to perform the following:-
   (a) What is the largest address in smalltex.din? (Write a small C program to find this value is it will affect your declarations ⇔ int, long long, unsigned,…etc.)
   (b) Is there a pattern in the trace file – if so, estimate the occurrences of the pattern and how it will impact (validity, fair test, etc) on the performance of a cache simulator. (Hint. Again a small C program that can find a given address and outputs the line number for each occurrence)
   (c) Also, consider the following: if a trace file was to be constructed simply by generating random numbers between 0 and MAX RAM-mem (say 16MB), would this be a better/best way to test:-
      a. Cache simulator,
      b. A possible real program? {Remember principle of locality as iy applies to real data!!!}

**A Typical (subset) question that may be asked at the programming Exam**

*Given a trace file write a cache simulator program that compares the following two cache architectures:-*

- *128KB direct mapped cache, block size = 4 Bytes, and*
- *32KB ,4-way set associative cache, block size = 16 Bytes, using LRU replacement policy.*

*By outputting for each design the respective counts for:-*

- *Compulsory misses,*
- *Capacity misses,*
- *Conflict misses,*
- *Total miss count.*

**Assignment hints**

You will notice that smalltex.din has only two columns – you can ignore the first column, the second column is the byte address in **hexadecimal** of the byte address issued by the CPU. You can assume that: All the addresses are in RAM (⇔no page faults).

a. The addresses start at 0 and go through to some large number - (an important preliminary step is to check the trace file to find the biggest address present in a trace file)
b. Calculate which block_number the address is in first. From there on work in block numbers, and, as in your practice labs, if all numbering systems you use start from 0 (being the first element of, for example, cache_location, set_number, etc) then the calculations performed in our Prac labs will be correct.

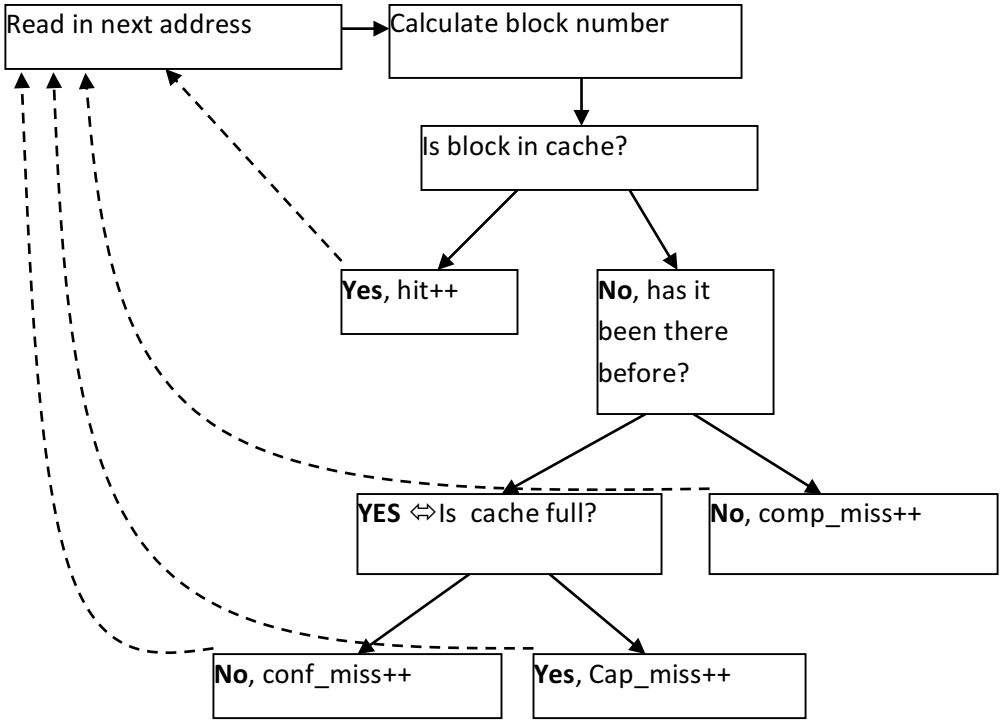



**Diagram: Logic to determining which type of miss occurred**

**NOTE:** When determining whether cache is full – it means over the whole cache regardless of the mapping (due to its associativity), thus any ***free space*** even if the block can't go there due to its mapping/associativity - means that the cache **is not yet full.**

When we refer to the least recently used (LRU) block in cache, this means the block that was least recently used/accessed ⇔ i.e. the *"stalest"* with respect to time. Whenever a block is used (either called into cache (a miss), or hit while in the cache), you will need to note/remember that this block then becomes the most recently used block in that set. How you implement this is entirely up to you, however

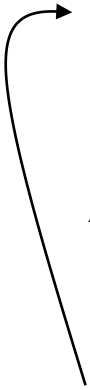
I would suggest that you keep it simple. This does not require any advanced algorithms or data structures.

Smalltex.din is a very large file. While developing your code, you can just make a small file (eg. Microtex.din) which only has, say, the first 100 entries (by deleting the rest of smalltex.din contents). If you do this it will allow room in your account for other data/programs (due to student account limits) and you wont experience the dreaded "account full – permission denied" when you wish to write back a program you may be developing. However, make sure that when you obtain your final results you use the **unchanged smalltex.din**. (smalltex.din will remain available throughout the semester for you to copy/access whenever you wish.)

As the results are dependent upon the trace file being used, some of the combined characteristics of a cache are only ***just*** observable, while other (combined) characteristics are readily observable (characteristics such as miss types, replacement policy, and block size). Thus, it is desirable to find under what conditions (input parameters) the characteristic that is being measured is observable and analyse these rather than do a lot of runs that are giving the same result/cache performance.

An important step is to be able to use Dinero – a commercial cache simulator this is also available to you – see Dinero.doc and associated ".docs" that give explanations and a quick tutorial on its operation). When using Dinero you can accept its output as bing OK as long as you use the correct command line invocation, and, can then compare your simulator's results to Dinero's.

Finally, a really useful way to construct your program when developing your "full-blown simulator" is to have a menu so that you don't have to start the whole program again each time you want to do a run. Typically it could be as simple as:-



*Wish to do another run Y/N*

*Cache-size……<Enter>*

*Block-size…….<Enter>*

*Associativity…..<Enter>*

*Replacement policy…….. <Enter>*

*<output results>*