**Superscalar out of order architectural simulator (With Memory Hierarchy)**

A project for CSEN702 Microprocessors

Supervised by:

* Dr. Mohamed Taher
* Eng. Jolly Salah

Team Members:

* Ahmed Tarek 31-4177
* Mostafa Ibrahim
* Omar
* Nada Bahaa

How the work was divided:

* Memory Hierarchy: Ahmed Tarek
* Functional Units: Mostafa Ibrahim
* Tomasulo: Mostafa Ibrahim, Nada Bahaa, Omar Yousry
* Utilities: Ahmed Tarek
* Assembler: Ahmed Tarek

Memory Hierarchy Stage:

Block: The basic block unit.

* Attributes:
  + String [] data; // to hold data in bytes
  + private String tag; // tag bits
  + private int validBit; // determines if the content is valid
  + private int dirtyBit; // used in writeBack only

Set: is the building unit of a cache, it holds the blocks

* Attributes:
  + Block[] blocks; // Blocks in a Set

Cache:

Attributes:

Set [] sets; // array of sets containing blocks, 1 set if fully Associative, same as number of blocks if direct mapped

private int size; // size of cache

private int lineSize; // line size of cache

private int m; // associativity

private String writePolicyHit; // writeThrough or writeBack

private String writePolicyMiss; // writeThrough or WB

private int accessCycles; // access time (in cycles)

private int totalHits; // cache hits

private int totalMisses; // cache misses

// tracking cache state

private boolean isBeingAccessed; // Data is currently being accessed from a D-Cache

private int accessCyclesRemaining; // tracks the number of cycles remaining complete instruction fetch.

private boolean isBeingFetched; // An instruction is currently being fetched from an I-Cache

private int fetchCyclesRemaining; // tracks the number of cycles remaining complete instruction fetch

Functions:

// This one takes a string address and data, it writes the data to that address (byte addressable memory)

public void writeByte(String address, String data)

// This one takes a string address , it reads the data in that address location and returns it

public String read(String address)

// takes a string address and returns true if it is a cache hit and false otherwise

public boolean hit(String address)

// takes an address and returns the tag bits of that address in string form

public String getTagBits(String addr)

// takes an address and returns the tag bits of that address in string form

public String getIndexBits(String addr)

// takes an address and returns the offset bits of that address in string form

public String getOffsetBits(String addr)

// Calculated Attrs Getters

//This one returns the hit rate of the cache

public double getHitRate()

public double getMissRate()

// The miss rate is the number of totalMisses / TotalCacheAccesses

public double getTotalCacheAccesses() {

// The total number of cache accesses is the number of hits + the number of misses

Memory:

* Attributes:

String [] memory; // Main memory array, length should be 2^16

private int accessTime; // Input: Number of cycles to access

private int totalCycles; // Output: Total number of cycles in the current stimulation.

private boolean isBeingAccessed; // tracking memory state

private int fetchCyclesRemaining; //tracks cycles remaining to fetch instruction

private int dataAccessCyclesRemaining; //tracks cycles remaining to access data

* Functions:

// Takes an int address as an input and returns the data in the memory location associated with it.

public String read(int address)

// Takes an int address as an input and a string data, writes the data to the memory location of index address.

public void write(int address, String data)

Memory Hierarchy:

* Attributes:

Public Cache [] caches; holds the caches of the microprocessor

Public Memory memory ; holds the main memory

* Functions:

/\* This one takes 2 String address and data as inputs

\* It first checks if the data is cached in the top level cache if so, it writes and keeps on writing to the cache levels below until a cache with WB policy is encountered

\* If the data is not cached in the first level, the data is first read and cached in the lower level caches

\*/

public void write(String address, String data)

/\*

\* This one takes an address in string form, it returns the index of the cache level where the data resides

\* If the data resides in the memory 1+ number of cache levels is returned to indicate main memory

\* This function does not do caching in anyway, it is a simple check for Tomasulo calculations

\* If the address is invalid i.e: it is not in the memory -1 1 is returned.

\*/

public int getCorrespondingCacheLevel(String address)

/\* This one takes an address in string form

\* Returns A string represents the block where the byte resides in within cache level 1

\* The Data is returned only when the cycles required to access it are finished

\* other wise the method returns null since the data was not yet accessed

\*/

public String loadData(String address)

/\*

\* This one takes a string address as an input and a cache level.

\* it returns the number of cycles remaining to retrieve the data from the cache

\* or ZERO if the caching is done, Its also responsible for caching the item

\* If you want to fetch data from memory to cache level 3, you'd just wait for getCacheCyclesRemaining(4, address)

\* to return 0, then the data would successfuly be in level 3 cache

\*

\*/

public int getCacheCyclesRemaining(int cacheLevel, String address)

/\* This one takes an address in string form

\* It returns the Data associated with that address

\* The method works as follows, It loops through the caches in a non decreasing order till it gets a hit, or it accesses the memory

\* Once The above condition is satisfied, the data is cached to the lower cache in the caches list

\* Then the loop goes back to that cache, and cashes the data in the cache preceiding it. This goes on till the level one cache has the data cached in it

\* The data is then returned from the level one cache as planned

\*/

private String readAndCacheData(String address)