Faculty of Media Engineering and Technology

Dept. of Computer Science and Engineering

Dr. Mohamed Taher Eng. Jailan Salah El−Din

# Report

1. *<u>Brief Description:</u>*

*Classes:*

- exceptions/CacheMissException: Thrown when a cache miss occurs.
- exceptions/StructuralException: Thrown when an index out of bounds occur in a cache.
- memoryHierarchy/CacheBlock: The CacheBlock object contains an array of Words , tag and a dirty bit.
- memoryHierarchy/CacheSet: The CacheSet object holds an array of CacheBlock and the line size.
- memoryHierarchy/Cache: The Cache object holds the cache size, line size, number of lines, mWays, the write policy, access time of the cache and an array of CacheSets.
- memoryHierarchy/Memory: The Memory has an array of words and the access time. It handles fetching and writing a word in the memory. *fetch*(int address, int offset, int lineSize) fetches the whole block, *write*(Word[] data, int address, int offset) writes a whole block in the memory. *writeByte*(Word data, int address, int offset , int lineSize) writes a word in a memory address and returns the whole block that it wrote in.
- memoryHierarchy/MemoryHierarchy: Combine the memory parts and build the whole hierarchy. The MemoryHierarchy contains a Memory, and array of Cache, totalAccessTime and latestAccessTime (the time the last instruction took in memory). The object also keeps track of the each cache accesses and misses. The *fetch* method takes the address and translates it into index, offset and a tag. It then searches the caches starting from the furthest one from memory till a match occurs, if no matches occurred then it accesses the main memory and fetch the address and writes the block in the upper levels. The *write* method takes a word and an address, it also traverses the

whole cache levels till it finds a match, if a match is found it writes in the block and if the write policy of the cache is write through then it writes in the lower level and updates all the upper levels, otherwise it just sets the dirty bit and updates the upper levels. Writing in lower levels and in upper levels is done by the two recursive methods *writeInLowerLevel* and *writeInUpperLevel*.

- instructionSetArchitecture/Instruction: It contains the fields of the instruction, the access time of the instruction and it handles the instruction's execution. Updating the pc occurs inside the Instruction class as it depends on the instruction type.
- instructionSetArchitecture/Register: Holds the name and value of the register.
- instructionSetArchitecture/ISA: It contains all the Registers including the PC. It is also responsible for writing and reading from registers.
- tomasulo/FunctionalUnit: The functional unit object has a name, number of instances, execution time, whether it's busy or not and all the necessary fields for the reservation station score board such as the vj, vk, qj, qk and the address. So the object acts as a line or couple of lines (in case number of instances is more than one) inside the reservation station score board.
- tomasulo/InstExtend: This object just adds another field to the instruction which is the expected cycle the instruction should have finished the fetch phase and should get inside the instruction buffer.
- tomasulo/ReservationStation: It combines all the FunctionalUnits and build the score board.
- tomasulo/ROBEntry: The ROBEntry has an instruction, a destination, value and a ready flag.
- tomasulo/ROB: The ROB has an array of entries of type ROBEntry and two pointers; a head and a tail.
- tomasulo/Tomasulo: This class is responsible for handling the algorithm. It has an instruction buffer queue, the number of ways that the user chose, instruction count to keep track of the number of instructions being issued and finally an integer array commit data. It contains the main five methods or we can call them phases; *fetch, issue, execute, write* and *commit*. First we fetch the instructions from the instruction

memory and add them in the wait buffer until they're completely fetched from the memory (the cycles spent during the memory access of the instructions have passed) then it can be inserted in the instruction buffer. Then the instructions are issued and inserted in the ROB. By the time the instruction has no dependencies it can then start the execution phase then wait one cycle for the write, and finally commit when it can.

- tomasulo/Simulator: This class is responsible for combining the whole program together. It consists of all the hardware parts and the clock. It is responsible for using the five methods of the tomasulo algorithm (the five phases). It is also responsible for calculating the output of the simulator.

Roles:

Mohamed Shokr :

Implemented the base structure of the memory hierarchy.

Linked the GUI with the engine.

Hazem ElAgaty:

Implemented the main structure of the memory hierarchy package.

Implemented the instruction set architecture.

Implemented the Tomasulo class.

Youssuf Sameh Radi:

Implemented the Simulator class and extra data structures for Tomasulo.

Implemented the Memory hierarchy main structure.

Omar Omar Soliman:

Implemented the GUI windows.

Created the test programs and the functionality to read the test from a file in the GUI.