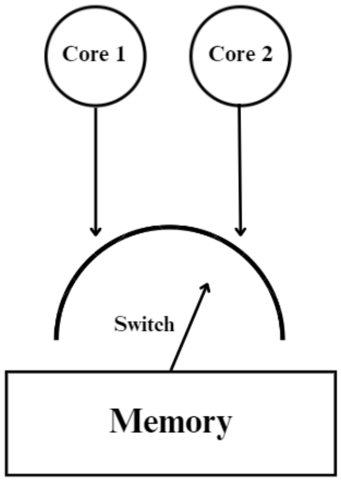
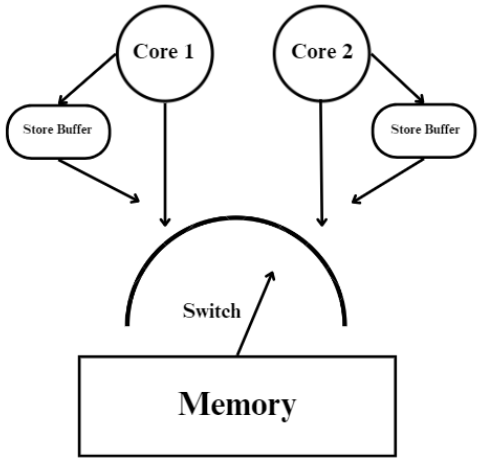
Project 3

Memory Ordering Demonstration in Multicore Shared Environments and Implementation of Sequential Consistency and Total Store Order



Sequential Consistency



Total Store Order

Description of the code is as follows:

Memory

Memory class creates a unified memory accessed by the cores for storing and loading purposes. A map **memoryData** of strings as **keys (representing memory addresses)** and integers as **values (representing the data stored at those addresses)**.

The **store()** function takes a string address and integer data as arguments and stores the data at the given address.
The **load()** function takes a string address as an argument and returns the data stored at that address.

Program

The Program class handles and reads the instructions from the program file.

The **getNextInstruction()** function returns the next instruction in the instructions vector and increments the **instructionPointer**. The instructionPointer is used to keep track of the current instruction.

The **hasNextInstruction()** function checks if any more instructions need to be read.

Executer

The executer class is a simulation of core for SC model containing 6 registers, initialized to 0.

The **executeInstruction()** function parses the instruction into operands and executes the instruction based on the first operand. If the first operand is:

- **Store:** stores the value of the specified register in the specified memory address. If the first operand is
- **Load:** loads the value from the specified memory address into the specified register.

If the **second operand is "="**, the value of the third operand to the specified register.

The **executeNextInstruction()** function retrieves the next instruction and executes it using the executeInstruction() function.

The **endOfProgram()** function checks if there are any more instructions to be executed.

TsoExecuter

The TsoExecuter class is a simulation of core for TSO model containing 6 registers, initialized to 0 and **registerDependency**, an unordered map that tracks dependencies between "Store" instructions and registers in the TsoExecuter class.

The **executeInstructionTSO()** function parses the instruction into operands and executes the instruction based on the first operand. If the first operand is:

- **Store:** stores the value of the specified register in the specified memory address. If the first operand is
- **Load:** loads the value from the specified memory address into the specified register.

If the **second operand is "="**, the value of the third operand to the specified register.

The **readInstruction()** function parses the instruction into operands and then checks the first operand. If the first operand is:

- **"Load" or the second operand is "="**
    - Check dependencies:
        - **No dependencies:-** Execute the instruction.
        - **If dependencies:-** Executes the store buffer before executing the instruction.
- **"Store"**
    - adds the instruction to the store buffer and sets the dependencies.

The **executeStoreBuffer()** function executes all instructions in the store buffer.

The **executeNextInstruction()** function retrieves the next instruction from the program and reads it. If there are no more instructions in the program, it executes the store buffer.

The **endOfProgram()** function checks if there are any more instructions in the program and if the store buffer is empty. If both conditions are met, the function returns true, indicating the end of the program.

Main

The main function executes two programs concurrently using the TSO and SC model. A shared memory space and cores are created to represent the cores of a multi-core processor.

As long as instructions are present in the cores, a random core is selected each time, and the next instruction present in that core is executed until no more instructions are present.
Once the instructions in a particular core are exhausted, it is removed from the list of cores.

Litmus tests

Refer code and mid-eval report.