Technology Fundamentals – Assignment One

Computer Architecture Simulation in Java

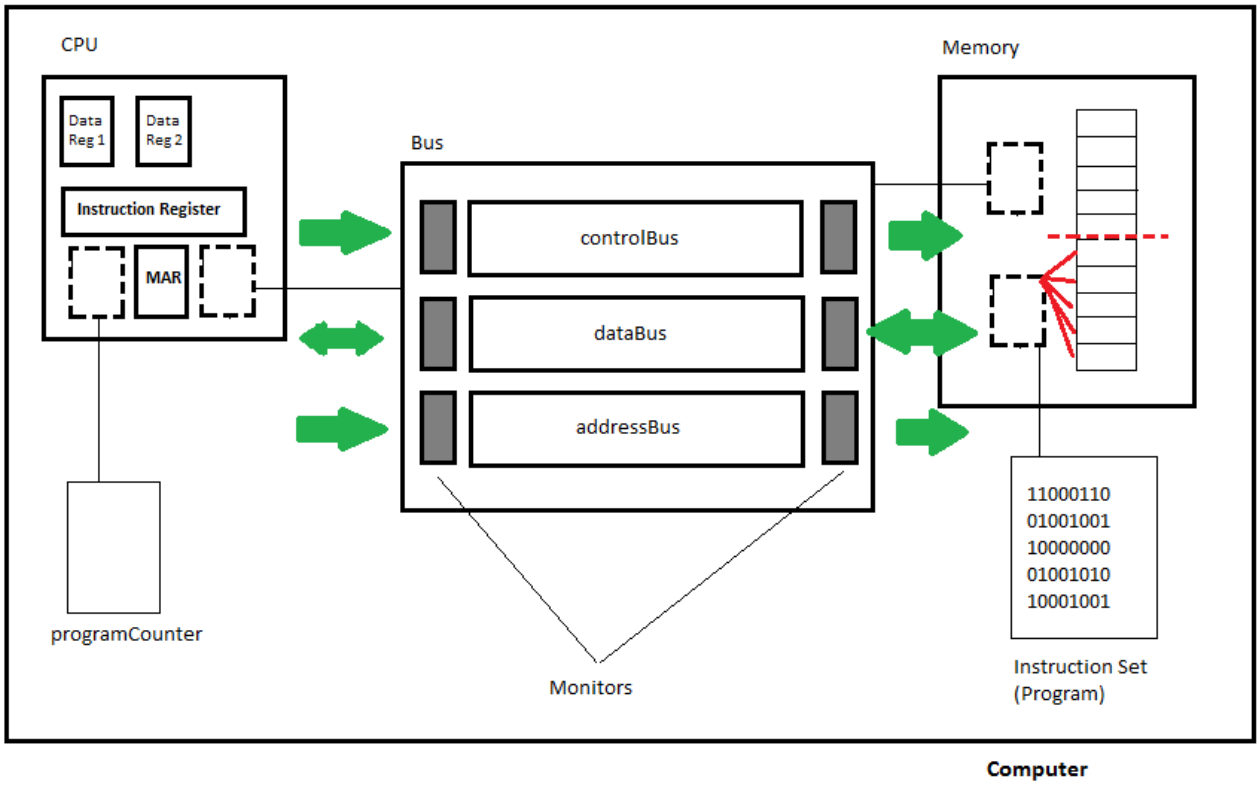
name: Peter O'Reilly

studentID: D12123601

email: peter.oreilly6@student.dit.ie

JRE version 1.7

Junit 4



Computer Architecture Model Java Implementation

Instructions to execute application

Assuming successful compilation of the code, the application will interact with the User via the console. The user will be prompted to either start or exit the program.

1. Press “y” and then press “enter” to start the Addition Programme.

The user will then be prompted to enter an unsigned byte (i.e between the range of 0-255 inclusive).

2. Enter the value and press “enter”.

The User will be requested to repeat the previous step until all 5 values have been entered. Once all values have been entered the program is then loaded into memory and the application will run. Once the programme has finished executing, the result will be displayed on the console and the application will exit.

The Executing Programme

Once the programme begins to execute, the console displays all the enqueueing and dequeueing of each of the busses, i.e. Data, Control and Address. It also identifies each Thread (eg. Thread-1, Thread-2) and displays when Memory is reading and writing.

In all of these displays, the data that is being read, written, and transported is displayed in binary form and the addresses that they are being sent to and read from.

Code Implementation

A two way producer – consumer model was used in the implementation of this assignment. Both the CPU & Memory class both act as either a Producer or Consumer at different stages within Fetch-Decode-Execute cycle.

Fetch Cycle

Once the programme is loaded into memory and all the valid parameters for the programme have too been written to memory, the CPU and Memory Thread are started, with the default state of the Memory Thread being disabled and in write enabled mode.

The CPU, which has the Programme counter set, transfers its value (i.e. Address of first line of assembly code) to the MAR which is then then transferred to the address bus, the Programme Counter is then incremented. The control bus is also set to enable memory and set to enable read. Whilst these operations would be overseen by the Control Unit, this element of the model was excluded. The CPU Thread then passes the lock to the Memory thread which dequeues both the control bus and address bus and executes a read. Once read, the data is then pushed onto the databus, the CPU thread is then notified and the lock passed.

Decode Cycle

Once an instruction is returned, it is transferred to the instruction register where it is decoded as per the project guidelines by the CPU.

Execute Cycle

Depending on the instruction retrieved from memory either one of the MOV() or the ADD() method will be executed.

MOV() - from Memory

If it is a move from memory, the CPU will enque the appropriate command to the control bus and will put the decoded memory address onto the address bus and pass the lock to the Memory thread. Once successfully read, the returned data will be placed on the data bus and the lock returned to the CPU thread. The CPU will then dequeue the data and place it in the appropriate register.

MOV() - from CPU Data Register

After the successful execution of an ADD() the summed data is stored in Data Register 2. An assumption of this Addition Programme is that if there is an ADD operation, the result will be written back to Memory.

In this instance the CPU will enqueue the control bus to enable memory and to enable a write cycle. Once the data is enqueued to the data bus it will pass the lock to the Memory and allow it to execute the write.

ADD() - within the CPU

The add() takes place when the CPU has the lock and the appropriate instruction in its instruction register. The result of the operation is stored in the second data register with the second operand being over-written.

Problems with the Java Model

There are a few design flaws with the current java model which I believe need to be qualified. Firstly the PC object exists external to the CPU and is passed by reference in the CPU constructor. Whilst the developer understood that the CPU has a program counter and that it does not exist external to the CPU this was something that was sacrificed in the design due to time constraints, with a similar excuse being applied to the control unit.

A further critique is that the program is directly loaded to memory by memory.
The developer would have aimed to have each instruction loaded into memory via the CPU and buses, but this again was not achievable within the given time frame. This would have facilitated the dynamic assignment of variables e.g. The Programme Counter, and a counter to stop the programme once it had exhausted all its instructions. As it stands, these have to be manually assigned at compile time to ensure successful execution.
Finally there is no implementation of cache in the model.

Conclusion

The decision was made to implement the model using Threads. This being the developer's first experience with using them turned out to be a costly exercise regarding time spent debugging each thread within the critical section of code namely the bus instance variable themselves. Despite this I feel that whilst there may have been some sacrifices in the Java implementation of the model, there was definitely a greater understanding achieved of the challenges posed by implementing Threaded applications.