The American University in Cairo

Tomasulo Simulator

Project 2 – Computer Architecture

Milestone 1 report

Aya Shaker	900160580
Habiba Gamal	900151007
Ali El-Said	900150264
Ahmed Wael Fahmy	900160127

Introduction

In this milestone, we were required to have a framework ready that accepts user inputs. In our implementation, the user can input the instructions in any of three ways. The user can input from a text file, input instructions as a stream, or by choosing from a list. We decided to create a struct that contains instructions as separate fields **parsedInst**, and a function that handles parsing the instructions into the aforementioned struct **parser**. The reservation station, reorder buffer, and the instruction status tables will all be implemented as arrays of their respective entries. This was achieved by creating a struct for their entries called **ROBEntry**, **reservationStationEntry** and **instStatusEntry**. As required, the memory was implemented as an array to simulate how it is actually addressed.

Although we intend to implement some bonus features, no extra feature was yet done other than the multiple user inputs.

Input Description:

For the first input mode, the user simply inputs the filename. In case the file cannot be opened, the user is notified. The input file is formatted as follows: **.text** indicates the start of instructions. This is followed by @<starting instruction address>, Which is used to indicate the instructions' location is the memory. Similarly, data is preceded by **.data**, which in turn is followed by @<starting data address>. The program checks the data and instructions, making sure they don't collide.

The second input mode allows the user to sequentially input instructions through the command line. The user inputs the initial address, then they can write the instructions when they are done with the instructions starting at the specified address, the user inputs 0. The user can then either resume entering data starting at a new location or stop there if they are done.

The final Input mode allows the user to choose from a list. This is done through prompting the user incrementally about: The type of instruction, the instruction itself, then (depending on the instruction type) the user can input the register numbers or the immediate values. This is all prompted on the command line for ease of use.

Code Description:

As previously mentioned, all tables were implemented as arrays of structs. The ROB is implemented as an array, sized 6, of type (**ROBEntry**). The struct contains the instruction type and destination as strings, the value as an integer and a readiness Boolean.

We created a struct that describes a general reservation station entry. This contains a business Boolean, an operation string, **V_j** and **V_k** as inegers, **Q_j** and **Q_k** as strings, ad the destination and address as integers. We then declare arrays of the following sizes for each type of station we have: 2 LW, 2 SW, 3 JMP, 2BEQ, 3 for ADD, one NAND, and finally two MULT stations.

Similarly, we recreated the instruction status table as a vector of type **instStatusEntry**. This struct holds integers that hold the cycle of issuing, executing, writing, and committal of the instruction. This is implemented as a vector for easily adjusting the size depending on the instructions.

We created a **parser** function that parses the instruction (passed as a string) according to the specified formatting and returns the parsed instruction of type **parsedInst**. This struct is then used to create the instruction buffer, which is a vector of type **parsedInst**

readFile is a utility function that is used in input mode 2. This function accounts for data and instructions, and places them in the correct memory locations.