MICROPROCESSORS

Project Report

Name: Mohamed Khaled	ID: 43- 16053	T : 17
Name: Aman Allah Rafaat	ID: 43- 17354	T : 27
Name: Salma Khaled	ID: 43- 2735	T : 23
Name: Ali Kabeel	ID: 43- 1172	T : 17
Name: Sarah Ahmed Mostafa	ID: 43- 17401	T : 17

January 12, 2021

1 Report

1.1 How to run

run the main method in Tomasulo.java you will be prompted to enter the number of instructions then enter the instructions one after the other each in separate line then after entering all the instructions to go to the next cycle enter the character 'n' and to terminate the execution press on any other character.

1.2 Assumptions

- The size of add reservation station is 3, the size of multiply reservation station is 2, the size of load buffer and store buffer is 3.
- the values in the integer registers and floating point registers are initialized such that the value at index i contains the value i.
- the memory size is 1024 and the initial value in the i-th address equals to i.
- if the tag (Qi,Qj or/and Qk) is not 0 the values in Vi,Vj or/and Vk are considered to be garbage values till the tag Qi,Qj or/and Qk become zero
- the number of cycles to execute add/subtract is 2 cycles, the number of cycles to execute multiply/divide is 10 cycles, the load/store executes in 2 cycles.
- The types of instructions are add, multiply, load, store with format as follows ADD.D, MUL.D, L.D, S.D respectively.
- we unset the busy flag in the reservation station at the end of execution since the tag-value pair is pushed to the queue of CDB (common data bus) at the end of execution so the reservation station / buffer is available for another instruction to occupy it (this is under the assumption that the write happens in the first half of the cycle and reading the value in the next half of the cycle).

1.3 Code Classes

We created classes for memory (class name: Memory) and register file (class name: RegisterFile) and instructions (Instruction)
We created classes for reservation stations and buffers of all types, namely:

1. Add reservation station , class name: AddReservationStation.
2. Multiply reservation station , class name: MulReservationStation.
3. Load buffers: class name: loadBuffers.
4. Store buffers: class name: storeBuffers.

1.4 Reservation station class

1.4.1 Reservation station class attributes

1. tag: Represents the tag which is the unique identifier of each instruction in the reservation station.
2. busy: Indicates whether the reservation station entry is empty or not. 1 represents a busy reservation station entry. 0 represents a free reservation station entry.
3. op: Represents the operation of the instruction in the reservation station.
4. vj: value of the first operand in the instruction.
5. vk: value of the second operand in the instruction.
6. qj: tag of the first operand in the instruction.
7. qk: tag of the second operand in the instruction.
8. time: time remaining till the end of execution.
9. instID: unique ID of the instruction.

1.5 Add/Multiply reservation station classes

1.5.1 Add/Multiply reservation station class attributes

1. addRS / mulRS: Array of reservation stations.
2. size: Size of the reservation station. (Number of entries).

1.5.2 Add/Multiply reservation station class functions

1. isFree() :The returns the index in the reservation station of the first free entry and returns -1 if there are no free entries.
2. nextInstruction(): Returns an array of indices in the reservation station of instructions that are ready to execute. Instructions are ready to execute if all their operands are available meaning that the qj and qk tags are not assigned (their values equal zero).

1.6 Load Buffer Class

1.6.1 Load Buffer Class Attributes

1. tag: Represents the tag which is the unique identifier of each instruction in the load buffer.
2. busy: Indicates whether the load buffer entry is empty or not. 1 represents a busy load buffer entry. 0 represents a free load buffer entry.
3. time: time remaining till the end of execution.
4. instID: unique ID of the instruction.
5. address: Address in memory where data is read from.

1.6.2 Load Buffers class functions

1. isFree() :The returns the index in the load buffer of the first free entry and returns -1 if there are no free entries.
2. nextInstruction(): Returns the index in the load buffer of the oldest issued load instruction that is ready to execute. Load instructions are ready to execute if the data to be loaded from the memory address is available(Qi tag of memory location equals zero).

1.7 Store Buffer Class

1.7.1 store Buffer Class Attributes

1. tag: Represents the tag which is the unique identifier of each instruction in the store buffer.
2. busy: Indicates whether the store buffer entry is empty or not. 1 represents a store buffer entry. 0 represents a free store buffer entry.
3. time: time remaining till the end of execution.
4. instID: unique ID of the instruction.
5. address: Address in memory where data is read from.
6. value: Value to be stored in the memory.

1.7.2 Store Buffers class functions

1. isFree() :The returns the index in the store buffer of the first free entry and returns -1 if there are no free entries.
2. nextInstruction(): Returns the index in the store buffer of the oldest issued store instruction that is ready to execute. Store instructions are ready to execute if the data to be stored to the memory address is available.

1.8 Tomasulo Class

1.8.1 Tomasulo Class Attributes

1. mem: Memory
2. addRS: add reservation station
3. mulRS: multiply reservation station
4. loadBuffers.
5. storeBuffers
6. intRegFile: Integer register file.
7. CDB: common data bus. It's a queue of tag-value pairs.

1.8.2 Tomasulo Main Logic

nextCycle() will handle the logic of executing a cycle. It calls:

1. handleAddUnit() / handleMulUnit():
    - Get the indices of the instructions in the reservation stations that are ready to execute using the nextInstruction() function in the reservation station.
    - For each instruction, it executes only one cycle and the remaining time is decremented.
    - If this cycle was the first cycle of execution in the instruction, the start of execution time of this instruction is set to be the current clock cycle.
    - If this cycle was the last cycle of execution, the busy flag of the reservation station entry is unset, the end of execution time of this instruction is set to be the current clock cycle and the result is pushed to the CDB (common data bus) queue (but not published yet).
2. handleMem():
    - Get the next memory instruction that is ready to execute using the nextInstruction() function.
    - It executes only one cycle and the remaining time is decremented.
    - If this cycle was the first cycle of execution in the instruction, the start of execution time of this instruction is set to be the current clock cycle.
    - If this cycle was the last cycle of execution, the busy flag of the buffer entry is unset, the end of execution time of this instruction is set to be the current clock cycle and if it was a load instruction the result is loaded from the memory location and pushed to the CDB (common data bus) queue (but not published yet) and if it was store the result is stored in the memory and it will also be pushed to the CDB queue (but not published)
3. issueInstruction():
    - it issues the next instruction and according to the type of the instruction it places it in the corresponding reservation station or buffer if there is available space in the required reservation station or buffer.
    - if the requested operands are available their values are assigned to the Vj and Vk from the register file.
    - if they are not available the operand tag are assigned to the Qj and Qk tag.
4. publish():
    - data is published in the next cycle after the end of execution.
    - if there is any data-tag pair in the CDB(common data bus) available it will be published and any component that needs this tag it will read the data from the bus.