Tomasulo Coding Project Report

Done By:

- *Rana Emad , 43-2101*
- *Hoda Ragaie , 43-1182*
- *Fady Aziz , 43-1628*
- *Rafiq Mazen , 40-15059*

Link to video:
https://drive.google.com/file/d/1TokxhppZO_9klexj2TOexwYBYQG95Yyn/view?usp=sharing

Sample Input:

L.D F2 0 R1
ADD.D F4 F2 F0
MUL.D F6 F4 F2
S.D F6 0 R3

Note: Both floating point and integer registers are already initialized in the tomasulo class instructor. The FP register file registers currently initialized : F0, F2, F4, F6, F8, F10, where all of them are initially empty except for F0 has a value of 1.6. The integer register file registers initialized: R1 value 2, R2 value 1, R3 value 4.

Project Classes and Functionality:

A. Instruction Class:

Constructor: Takes the instruction entered, from the instruction buffer, and decodes it.

Fetch Fn: It is called in the Tomasulo class when an instruction is fetched to set the state of the instruction to IF (instruction fetch/issue) and sets the issue cycle to the current cycle.

Execute Cycle Fn: It is called in the Tomasulo class when an instruction is in the execute stage, to decrease the number of cycles left and checks if the instruction finished the cycles needed so, it would be added to the instructions that are going to be written back.

B. Instruction Buffer Class:

Constructor: It takes the number of instructions and the instructions one by one from the user and passes them to the instruction class.
The instructions entered are assumed to be correct, where the user should not read from a register that's initially empty or does not exist.

initializeIntRegister Fn: Takes the Register name, state and value and initializes a new integer register.

updateIntRegisterValue Fn: Takes the register name and value and updates the value of the integer register.

initializeRegister Fn: Takes the register name, state and value and initializes a new floating point register.

updateRegisterValue Fn: Takes the register name and value and updates the value of the integer register.

canReadFromReg Fn: Takes the register name to check if I can read from the register by checking the state to be "C" and returns a boolean, true for can read, false for can not read.

reserveReg Fn: Takes the destination register name and the tag of the station or the buffer and reserves the register.

C. Memory Class:

Constructor: Initializes a new data memory of size 8 and fills it with random numbers.

read Fn: Takes an address and returns the value of this address.

write Fn: Takes an address and a value to write in the memory.

D. RegisterEntry Class:

Constructor: Initializes the the state and value of the register.

E. Load Buffer Class:

Constructor: Initializes the load buffer: sets busy bit to false and address to 0.

reserve Fn: Takes an instruction and sets the busy bit to true and the address to the sum of the RS and RT (RS is already a number and RT register value is fetched from the integer register file).

clearStation: Sets the busy bit to false and the address to 0 to simulate emptying the buffer.

F. Load Buffer List Class:

Constructor: Takes a string tag title and a number, initializes a hash table with the size of the number entered and gives every entry a tag (tag = title + a num e.g: “L” + 1)

returnFirstEmptyLB Fn: returns the tag of the first empty buffer in the buffer list.

G. Store Buffer:

Constructor: Initializes the store buffer: sets busy bit to false and address to 0, V to 0 and the Q to null.

reserve Fn: Takes an instruction and sets the busy bit to true and the address to the sum of the RS and RT (RS is already a number and RT register value is fetched from the integer register file) and checks if the RD register entered value can be read and sets the V or the Q accordingly.

clearStation: Sets the busy bit to false, the address to 0, the V to 0 and the Q to null, to simulate emptying the buffer.

H. Store Buffer List:

Constructor: Takes a string tag title and a number, initializes a hash table with the size of the number entered and gives every entry a tag (tag = title + a num e.g: “S” + 1)

returnFirstEmptyLB Fn: returns the tag of the first empty buffer in the buffer list.

I. ResStation Class:

Constructor: Initializes the Reservation Station, sets the busy bit to false, the Op - Qj - Qk - A to null and the Vj - Vk to 0.

reserve Fn: Takes an instruction and sets the Op to the instruction operation, the busy bit to true and checks if the RS and RT can be read from and sets the Vj, Vk, Qj and Qk accordingly.

*clearStation Fn: S*ets the busy bit to false, the Op - Qj - Qk - A to null and the Vj - Vk to 0 to simulate emptying the station.

J. MulResStationList Class:

Constructor: Takes a string tag title and a number, initializes a hash table with the size of the number entered and gives every entry a tag (tag = title + a num e.g: “M” + 1)

returnFirstEmptyLB Fn: returns the tag of the first empty station in the buffer list.

K. AddResStationList Class:

Constructor: Takes a string tag title and a number, initializes a hash table with the size of the number entered and gives every entry a tag (tag = title + a num e.g: “M” + 1)

returnFirstEmptyLB Fn: returns the tag of the first empty station in the buffer list.

L. Tomasulo:

Constructor: Initializes the instances created for the cycle (to 0), the Instruction Buffer, the Multiplication reservation station, the Addition reservation station, the Load Buffer List, the Store Buffer List and the Data Memory. We initialize the integer and floating point register files as well.
A while loop is initialized and simulates the whole process by (1) incrementing the cycle by 1 - (2) calling the writeBack fn - (3) calling the execute fn - (4) calling the fetch fn

fetch Fn: checks if the unfetched Indices array is not empty then, fetches the first instruction in the array then checks the operation of the instruction to check if there’s an empty station/buffer before issuing the instruction. When the instruction is issued, it is removed from the unfetched array, added to the fetched array and then reserve the destination register as well as the station/ buffer. The appropriate readyToExec fn is then called.

readyToExecMul - readyToExecAdd - readyToExecSD Fn: Takes a tag, index and instruction and checks if the Q fields are set to null, sets the state to

EX, the start and end execution cycles, removes the instruction from the fetched array and lastly adds the instruction to the executing array.

execute Fn: loops over the executing array instructions and calls the executeCycle fn then checks if the instruction is done, sets the state to Done EX, the write cycle to the next cycle, removes the instruction from the executing array and lastly adds it to the write indices array.

writeBack Fn: Checks if the write Indices array is not empty, then gets the first instruction; removes the instruction from the write indices array, adds it to done Indices array then, checks the operation of the instruction- in case of an arithmetic op: to set the result value from the V fields accordingly, clear the station, update the destination register value and state then, writes the result in the bus. In case of Load op: sets the result value to the value of the calculated memory address, clear buffer, update destination register then write result in bus. In case of Store op: write the value of the register in the calculated address in memory then, clear buffer.

resultInBus Fn: Takes the tag of the station/buffer where the instruction written back was in and the value of the result then, loops over the fetched (not yet executed) array and check if this instruction needed the result of the written station in either the Qj or Qk then sets the Vj / Vk accordingly then lastly, calls the appropriate ready to execute fn.