Contents

Main function

This function essentially tests the functionality of the Tomasulo algorithm by loading instructions and configuration, simulating the execution of instructions, and logging trace information for analysis and debugging purposes.

1. **Headers:**
   (`InputOutput.h`) (`Tomasulo.h`) are the programs used for this simulation.
   InputOutput(.c,.h) is exclusively used by the Tomasulo Program.
   However, we have access to Tomasulo functions which utilize them.
2. **External Variables Declaration:**
   External declarations for `cycles` and `bHalt` are made, as they are defined in Tomasulo(.c,.h) but are used in the main function as well.
3. **Initialize Tomasulo Algorithm:**
   Main calls `InitTomasulo("cfg.txt", "memin.txt")` to initialize the Tomasulo algorithm simulation. This function uses the file paths to load and parse using the InputOutput(.c,.h) program.
   Using this data, it sets up the initial values for the reservation tables (ADD,MUL,DIV), registers, and instruction queue and the queue for stations (more on the (6)).
4. **Simulation Loop:**
   A while loop is initiated, and it continues until the `bHalt` flag is activated which occurs when previous instructions to Halt have finished.
   `Fetch()`: Simulates the fetching of instructions.
   `Issue()`: Simulates the issuing of instructions to reservation stations.
   `Execute()`: Simulates the execution of instructions.
   `Write()`: Simulates the writeback phase.
   `cycles` is incremented after each iteration to track the current cycle count.
5. **Cleanup:**
   After the simulation loop exits, the `DestroyTomasulo()` function is called to clean up heap memory allocated from the Tomasulo Program.
   Instruction queue, station queue, and all stations are cleared from memory.
6. **Logging Trace Information:**
   Finally, the function calls `LogTomasulo("traceinst.txt", "tracecdb.txt")` to log trace information about executed instructions and data written to the Common Data Bus (CDB).

Note: `("cfg.txt", "memin.txt"),("traceinst.txt", "tracecdb.txt")` are mentioned as strings for simplicity. In reality, they are passing (`char* argv[]`) elements, which are parsed from the user via the command prompt.

InputOutput Program

This InputOutput(.h,.c) program is meant for parsing instructions and configurations from files as Hex characters to be shared with the Tomasulo(.h,.c) program. And writes the results it receives to output files.

Global Variables:

1. **Instruction instructions[LEN_INSTRUCTIONS]:** An array to store parsed instructions from `memin.txt`.
   `LEN_INSTRUCTIONS` is the maximum number of instructions that can be stored.
2. **Configuration config:** Stores configuration data parsed from `cfg.txt`.

Note: these variables are defined in the Tomasulo program via the header.

Structures:

1. **Instruction:** Represents a single instruction with fields for operation (`op`), destination (`dest`), source 0 (`src0`), and source 1 (`src1`) – each is 1 byte and the instruction is 4 bytes in total.
2. **Configuration:** Represents the configuration settings for the program. For each type of station (ADD, MUL, DIV) there are 3 configuration parameters: `(Units)` – number of function units; `(Reserves)` – number of reservation stations (henceforth simply called "station"); `(Delay)` – number of cycles needed to finish executing the functional unit.
3. **TraceInstr:** Represents logged details about each instruction from Issue to Write. It logs the following: the instruction itself (`dest`), cycle at which it was issued (`cycleIssue`), execute start cycle (`cycleExStart`) and execute end cycle (`cycleExEnd`), cycle at which it writes to the CDB (`cycleWriteCDB`), program counter (`pc`), and tag (`tag`) for identifying the station number and the corresponding CDB name.
4. **TraceCDB:** Represents logged details about all data written to the CDB. For each CDB (ADD,MUL,DIV) it logs the following: data itself (`data`) in decimal form, cycle at which it was written (`cycleWriteCDB`), program counter (`pc`), and tag (`tag`) for identifying the functional unit number and the corresponding CDB name.

Functions:

1. *createInstructions(const char* meminPath):* Parses instructions from a file specified by (`meminPath`), converts them from hexadecimal to numerical format, and stores them in the global (`Instruction instructions`) array.
2. *initConfig(const char* cfgPath):* Parses configuration settings from a file specified by (`cfgPath`) and initializes the global (`Configuration config`) variable with the parsed values.
3. *writeTraceRegout(const char* regoutPath, float* F):* Prints the final registers' values onto a file, its name specified by *regoutPath*.
4. *writeTraceInstr(const char* traceInstrPath, TraceInstr** traceLogInstr): Writes details of executed instructions to a file specified by the `traceInstrPath` string argument. It extracts the details extracted from the logging array `TraceInstr traceLogInstr[LEN_INSTRUCTIONS]` received from the Tomasulo(.h,.c) program. Then it is formatted to our desired form.
5. *writeTraceCDB(const char* traceCDBPath, TraceCDB** traceLogCDB): Writes details of data written to the 3 Common Data Buses (ADD,MUL,DIV) to a file specified by the `traceCDBPath` string argument.
   It extracts the details extracted from the logging array `traceLogCDB i.e.TraceCDB traceLogCDB[LEN_INSTRUCTIONS]` received from the Tomasulo(.h,.c) program. Then it is formatted to our desired form.

Additional Notes:

- This InputOutput program is meant to be used by the Tomasulo(.h,.c) and not meant to be used by the user (Source.c) directly.

Private functions:

Functions that are directly used solely by the InputOutput program.

1. *readFile(char* dataPath, char fileStr[LEN_FILE]):* Reads the contents of a file specified by **`dataPath`** and stores them into the char array **`fileStr`**.
2. *hexToNum(char* input):* Converts hexadecimal characters in a string to their corresponding numerical values according to the ASCII table.

Definitions:

```
#define LEN_INSTRUCTIONS 4096
#define LEN_FILE 4096*9+1
#define LEN_CFG 512
#define LEN_REGISTERS 16
```

`LEN_INSTRUCTIONS` is set as specified in the file, 4096 maximum instructions.
`LEN_FILE` because 8 bytes are used to represent the instruction and there is a newline break after each one, 9 bytes in total for all 4096 instructions (including the last instruction which has a null terminator. 1 is added in case we end the last instruction with newline break.
`LEN_CFG` Because 23 bytes is the largest line we can have, and there are 9 of them – so it should be 23*9+1. However, to account for random spaces, newline breaks and whatnot, we make it 512 bytes.
`LEN_REGISTERS` **`is`** the length value of the arrays (tag and F) of the ***Registers*** struct.

Tomasulo Program

This program Tomasulo(.c,.h) implements a simplified version of the Tomasulo algorithm, a dynamic scheduling, parallel instruction level algorithm used for out-of-order execution of instructions with the aim of optimizing performance.

Global Variables:

1. **unsigned int cycles:** Keeps track of the current cycle in the simulation.
2. **Table addTable, mulTable, divTable:** Represent reservation tables for add, multiply, and divide operations respectively, each with its own set of stations.
3. **Registers regs:** Represents architectural registers, including floating-point values (`F`) and tags (`tag`).
4. **Unsigned int bHalt:** Flag indicating whether a HALT instruction has been encountered – activates after all previous instructions are finished writing.
5. **Queue* tail, head:** Pointers for the instruction queue, used mainly fetching and issuing instructions.
6. **unsigned int instrQSize:** Size of the instruction queue.
7. **unsigned int currentInstr:** Index of the current instruction to be fetched by the instruction queue (also indicates the PC).
8. **QueueStation* headStation, tailStation:** Pointers for the station queue, used for indicating busy stations that have not started execution (due to not having access to V_{jk} values).
9. **unsigned int stationQSize:** Size of the station queue.
10. **TraceInstr traceLogInstr[LEN_INSTRUCTIONS], TraceCDB traceLogCDB[]**

traceLogCDB[LEN_INSTRUCTIONS]: Arrays for logging executed instructions and data written to the Common Data Bus (CDB).
Note: here is a maximum number of instructions (**LEN_INSTRUCTIONS**) which the program will perform from Fetch to Write.

11. **traceIndexInstr, traceIndexCDB:** Index variables for trace logs.
    traceIndexInstr increments at *Issue*.
    traceIndexCDB increments at *Write*.

Structures:

1. **`Station`**: Represents a reservation station with fields to track status and progress.
   Added variables to the familiar signals (**`busy,opcode,Vjk,Qjk`**) are the following: (**`executeCount`**) indicates number of cycles since execution started. Used to permit a Write.
   (**`cycleIssue`**) used to handle Issue-Execute hazard (more information in Additional Notes: below). (**`traceIndexInstr`**) is used for logging the instruction during Execute and Write.
2. **`Registers`**: Represents registers with arrays for tags and values.
3. **`CDB`**: Represents the Common Data Bus with fields for data, tag, and busy status.
4. **`Table`**: Represents a reservation table with fields for stations, length (number of reservation stations), delay, free station count, freeUnitCount (number of units available for the stations to use), and CDB.
5. **`Queue`**: Represents a queue for instructions with fields, cycle fetch time, instruction, and pointers to the next node.
   Other than the familiar, we have the following fields: (**`cycleFetch`**) used to handle Fetch-Issue hazard (more information in Additional Notes: below).
6. **`QueueStation`**: Represents a queue for stations that do not have one of their (**`Vjk`**) signals. Fields for cycle fetch time, station pointer, and pointers to the previous and next nodes.
   Note: As evident by the use of (**`back`**) and (**`front`**) members, it does behave wholly like a queue. It does *enqueue*, but rather than *dequeuing* it uses *pop* which removes a specific station within the queue. Thereby behaving more as a linkedlist rather than a queue.

Functions:

1. ***InitTomasulo****(const char* cfgPath, const char** meminPath): Initializes the Tomasulo algorithm simulation.
   Reads configuration parameters using *initConfig* from InputOutput(.c,.h).
   Loads instructions using *createInstructions* from from InputOutput(.c,.h).
   Setting up reservation tables and their stations by granting them initial values and config values.
   Initializing registers, queues, and resetting counters.
2. **DestroyTomasulo():** Cleans up memory allocated in heap.
   Frees tables' allocated stations, instruction queue and station queue.
3. **Fetch():** Simulates instruction fetching, loading 1/2 instructions into the instruction queue from memory.
4. **Issue():** Simulates instruction issue, assigning instructions from the instruction queue to reservation stations if they are available.
5. **Execute():** Simulates instruction execution, updating execution counts for each station in the station tables.
   Each table (ADD,MUL,DIV) is executed using the private function *executeTable()*.
6. **Write():** Simulates the writeback phase, writing results to the CDB and updating registers and reservation stations accordingly.
7. ***LogTomasulo****(const char* regoutPath, const char* traceInstrPath, const char** traceCDBPath): Writes the trace logs of executed instructions and data written to the CDB to the output files specified in the prototype's arguments using the data collected in the trace log global variables (**TraceInstr traceLogInstr[LEN_INSTRUCTIONS**) and (**TraceCDB traceLogCDB[LEN_INSTRUCTIONS]**).

Additional Notes:

- **HALT Handling:** The code includes handling for HALT instructions to ensure proper termination of the simulation when all instructions have been processed.
- **Fetch-Issue Handling:** Cannot Fetch and Issue the same instruction within the same cycle). (**pc**) is used for logging CDB and Instruction traces.
- **Issue-Execute Handling:** Cannot Issue and Execute the same instruction within the same cycle.
- **Tag build:** tag = Y[31:3]-X[2:0] – where Y[28:0] is the number of station and X[2:0] is the OP code ($ADD: 2, SUB: 3, MUL: 4, DIV: 5, HALT: 6$).
  This is done using bitwise manipulation of 32 bit unsigned int.

Private Functions:

Functions that are directly used solely by the Tomasulo program.

1. **`initQueue(), enqueue(), dequeue():`**

A. **`initQueue()`** initializes the instruction queue (**`Queue`**).
Sets the head and tail pointers to **`NULL`** and sets the queue size to zero.

B. **`enqueue()`** adds a new instruction to the end of the queue.
It dynamically allocates memory for a new queue node, copies the instruction data, and updates the queue pointers.

C. **`dequeue()`**: Removes the first instruction from the queue.
It deallocates the memory, and returns the instruction.

2. **`initQueueStation(), enqueueStation(), popStation(QueueStation* current):`**

A. **`initQueueStation()`** initializes the station queue (**`QueueStation`**) in a similar manner to **`initQueue()`**.

B. **`enqueueStation()`** adds a new station to the end of the station queue.
It allocates memory for a new station node and updates the queue pointers.

C. **`popStation()`** removes a specific station, `current`, from the station queue.
It updates the queue pointers by connecting the one behind it (**`back`**) and the one in front of it (**`front`**), and deallocates the memory.

3. **`writeTable(Table* table)`**:

This function writes the results of executed instructions to one of the CDBs using (**`table`**) , and updates the stations accordingly.

A. First, it finds a busy station that is finished executing and writes to its corresponding CDB with (**`data`**) and (**`tag`**) (if it is not busy!).
B. Second, it updates (**`Vjk`**) values for stations in the `QueueStation` with the matching tag.
C. Third, it updates the register with the matching tag.
D. Fourth, it clears the chosen station, the corresponding CDB, and releases the functional unit.

4. **`executeTable(Table* table):`**
It finds a busy and ready station ((**`Vjk`**) available), increases the (**`executeCount`**) of the station.

Note: we also handle the Issue-Execute hazard (more information in Additional Notes:).
Note: we also preemptively set (**`cyclesExEnd`**) to a value upon reaching the first execute (as well as setting (**`cyclesStartEnd`**), as we know it finishes execution in exactly (**`delay`**) cycles, as given by the config file.

Definitions

```
#define LEN_QUEUE 16
#define ADD 2
#define SUB 3
#define MUL 4
#define DIV 5
#define HALT 6
```

`LEN_QUEUE` is the length value of ***Queue*** struct.

`ADD,SUB,MUL,DIV,HALT` are all opcodes.