31 October, 2014

ECE552 Lab 3Report
Dynamic Scheduling with Tomasulo

Mehrad Khalesi 998694202
Hooman Haji 999615000

Results

Following are the functional performance results of the simulated tomasulo design in Lab3.

EIO Trace	total number of cycles with tomasulo
gcc.eio	1,814,117
go.eio	1,852,942
compress.eio	1,979,818

Functions

Helper Functions

Below is a list of helper functions and a brief description of their task.

bool is_IFQ_empty()

- Check if IFQ is empty. If so, returns true, otherwise returns false.

bool is_RS_empty()

- Checks if all reservation stations are empty. If so, returns true, otherwise returns false.

bool is_FU_empty()

- Checks if all functional units are empty. If so, returns true, otherwise returns false.

bool pop_ifq()

- Removes the first element from IFQ and moves all the other elements forward. Last element is set to NULL.

Tomasulo Functions

Below is the list of functions that are executed in each cycle

static bool is_simulation_done (counter_t sim_insn)
In each cycle this function is invoked to check if there is no more instruction left to execute. All the following conditions have to be satisfied in order to stop execution/simulation.

1. Functional Units (FU) are empty
2. Reservation Stations (RS) are empty
3. IFQ is empty

In other word, simulation is done if all functions is_FU_empty(), is_RS_empty() and is_IFQ_emprty() return true.

static void execute_To_CDB(int current_cycle)

Scans all instructions in the functional units and saves all completed store instructions as well as the oldest completed instruction with an output(if any) in a separate buffer. Once it finds these instructions, the instruction that needs to broadcast its result to CDB does so. Then corresponding output locations in the maptable are cleared if the instructions match. Lastly, function de-allocates FUs and RSs corresponding to the completed instructions that it has found.

static void issue_To_execute (int current_cycle)

Finds all instructions with source operands ready and allocates an FU if there is one available.

The function checks all FUs and while a FU is available it finds the oldest matching instruction in RSs with source operands ready (no RAW hazards or all of its RAW dependencies has been resolved) and send the instruction into the FU. We repeat this until either there in no available FU or no more ready instruction.

At the end, we look for instructions in RSs that have dependencies on the result being broadcasted in the common data bus and we clear the dependencies.

static void dispatch_To_issue (int current_cycle)

The function pops the first instruction of the IFQ and dispatches it to a RS if there is any available. If not It does nothing. So we do the following in this function.

1. If the there is no instruction at the head of IFQ we just return
2. If the head instruction is conditional or unconditional branch we just pop the instruction and remove it from the queue and the return.

3. if the there is an available matching RS to the head instruction, remove the instruction from the queue, put it in the RS, update the map table for output register entries, read map table for source register entries to find dependencies

```
static void fetch_To_dispatch (int current_cycle)
```

If the IFQ is not full it fetches a new instruction from instruction trace and place it into the IFQ. Else it does nothing at all.

Testing

Simulation was ran on gcc.ieo and compress.eio trace for the first 10 and 15 instructions respectively:

```
sim-safe   -max:inst   10 /cad2/ece552f/benchmarks/gcc.eio
```

result: 33 cycles

```
sim-safe   -max:inst   15 /cad2/ece552f/benchmarks/compress.eio
```

result: 37 cycles

One of the members traced tomasulo for the first 15 instructions of gcc.eio and the other member separately traced tomasulo algorithm for the first 15 instructions of compress.eio. The results matched.

Bug Report

Generally, our team spends the bulk of time spent on the projects thinking about corner cases and the proper implementation of the project, before starting to write the code. As a result, not many bugs occurred after we wrote the code and we were able to get reasonable results even in the initial runs of our code. However, there were a couple of minor bugs that were resolved very quickly.

1. Updating the Reservation Stations and Functional Units:

In tomusulo's design, reservation stations and functional units are re-occupied in the same cycle as the previous instruction finished using these units. However, by calling our pipeline stages(in the simulation) in order (F to D, D to I, I to X and X to CDB) resources would become available after the search for available resource (RS or FU). This caused all waiting instructions to wait one extra cycle before occupying the resource they were waiting for. This bug was solved by reordering our function calls.

2. Reading from instr[0]:

Habitually, while looking for the first element in an array we called the *0-th* element. However, as the Lab handout mentioned the instruction array started from *1st* element and there was no instruction in instr[0] (just a random value). This bug was resolved after printing our instructions and noticing the incorrect format of instr[0].

Work Done by Each Member

Mehrad Khalesi: Writing code and testing

Hooman Haji: Writing code and testing