Project_2 Report Redone

Overview-Simulation of Tomasulo with Rob Buffer

Tomasulo algorithm helps in dynamic scheduling and out-of-order execution which helps in efficient execution with the help of multiple processors.With the help of register renaming and reservation station it helps to prevent WAW and WAR hazard.
The Rob buffer helps in in-order exceptions and helps in preventing branch misprediction, handling and page faults.

Problems faced previously:

Previously ,I was not able to finish the project mainly because of 2 reasons

1.Dangling pointer issues and difficulty in debugging pointer implementation of reservation station rob and pending_instruction entries

2.Unable to implement a proper circular queue(Got totally confused with implementation)

Overcoming the challenges:

I fixed the issues with

1.Changing the original codes logic of using pointers for pending instructions rob and reservation stations entries to vector implementations.

2.Thanks to the thread in stack overflow(https://stackoverflow.com/questions/59097741/reorder-buffer-pointer-values-when-it-is-full)
through discussions and a lot of debugging I was able to get the Rob buffer working without explicit implementation of circular queues ,will be discussed in more details in the data Structures part of the report.

This implementation will help to port the simulation to other languages too that doesn't use a pointer like Java Python etc.

Current Status

Testcases	Status	Inference
Testcase1	Passed	Was having problems with address calculation during load and store took a long time debugging. The main caveat was when the reservation stations were full I was incrementing the PC,needed to put a another check for that in the issue stage. Also there were some bugs in the address calculation for load in execution phase but was squashed. Testcase1 was the hardest stage to implement
Testcase2	Passed	After Testcase1 passed I had to fix the flush protocol in a different function to fix a bug in the branch instruction for a second integer unit
Testcase3	Passed	No changes made
Testcase4	Failed	Seg Fault CC7: Debugging showed data memory was not being read properly. Must be a minor bug in address calculation still can't fix it. Getting mixed up in my own loops in excute stage during debugging. Sometimes it gives the whole result too and the test case matches
Testcase5	Passed	Surprising testcase4 failed but 5 passed.Maybe a memory leak as I am running the simulation on M1 chip,valgrind cannot run on this architecture yet
Tescase6	Almost Passed	There is one value in reservation station during execution complete that is not matching must be. Minor bug

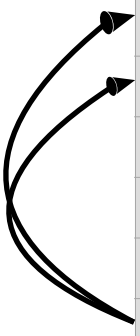
Testcases	Status	Inference
Testcase7	Failed	Execution not completing.Major bug in execution clockcycle or commit cc check.Cant be solved due to to time constraints
Testcase8	Failed	Same as 7
Testcase9	Failed	Same as 8
Testcase10	Failed	Same as 7

Data Structures Used

@Rob Buffer_entries(vectors)-It took me two days to figure the easiest implementation of Rob buffer.
After much discussions on git issues page of some git implementation and going through stack overflow.
Previously I was trying to use pointers and having a circular queue(not even properly implemented) for head and tail.
Now I use the concept of advancing the head and reiterate the rob every time.
Rob entries are vectors of 6 structs of rob having the rob properties of PC

Entry	Tag	Values	Retirate=True
1	Empty		
Tail:2	Empty		
Head:3	Load	1	
4	Store	2	
5	Add1	3	
6	Add2	4	

New Instruction ->suppose BNEZ



Entry	Tag	Values	Retirate=False
1	BNEZ	R1->Target	
Tail:2	Empty		
Head:3	Load	1	
4	Store	2	
5	Add1	3	
6	Add2	4	

So as we see here in commit stage I just need to advance the rob and on every issue I will reiterate the rob_buffer once so that the empty cells above Rob Heads(1 &2 where 2 is the so called “tail”) are also filled.
Advance the rob head when the instruction is committed

@reservation stations_entries(vectors) - All the reservations station entries are now vectors instead of pointer to arrays of structs for easier debugging

@pending instructions_entries(vectors)-These are also vectors and the concept of reiteration is also used here as in Rob in commit stage

@excution_unit_entries(arrays)-Already implemented from before

@instruction_window_t(structs)-has the following properties
unsigned src2; //second source register in the assembly instruction
unsigned dest; //destination register
unsigned immediate; //immediate field
string label;parsing
bool already_issued
bool already_executed
bool already_written//not used
bool already_commited .//not used

@executiion_units(structs):
exe_unit_t type; // execution unit type
unsigned latency; // execution unit latency
unsigned busy;
unsigned pc;
@instr_window_entry_t(structs);
unsigned pc
unsigned issue
unsigned exe
unsigned wr
unsigned commit
@rob_entry_t(struct)
bool ready=false
unsigned pc=UNDEFINED;
stage_t state;//Issue by default
unsigned destination
unsigned value

@res_station_type(struct):
res_station_t type; // reservation station type
unsigned name;
unsigned pc
unsigned value1
unsigned value2
unsigned tag1
unsigned tag2
unsigned destination
unsigned address

Functions Used:

@issue()->This function helps to issue the instruction in the clock cycle
@update_reservation_station_and_rob(res_stationEntry,rob entry,bool already issues):This functions fills the res_station and rob by using PC as map tool for instrcutions following the steps are followed:

- Find empty res_station-check if the instruction is not already issued and no other instruction is issued in same cc
- Find empty rob //see comments on how to fill it
  - Fill rob
  - If not filled from head reiterate just to be sure no empty spots above rob head

If instruction. Is not issued prevent PC from progressing

@execute->This function helps to excite the instruction by sending them through different execution station only if the instruction is not already excused
@start_excution_Operations(res_station_entry,execunuts,bool already_executed)

- Uses the PC as mapping variable to map the instruction from res-station To Rob entry
- Fills exec stations only if its not busy

@write_back-Writes the instruction to rob
Flush_protocol-Follows flush protocol of emptying the rob and res_station if branch is mispredicted.

@commit-commits from rob to register files
@run-runs the core of the simulator-it will complete all the instructions if cycle==0 using the runcycle concept as in project1

MAKE FILE CHANGES

OPT = -g -std=c++11 was added to both makefiles at line2 to make grendle compile instruction fro C+11 standard by default

Run the program

@grendel makefile
cd project2_code
cd c++
make
./bin/testcase1
./bin/testcase2
:
:
:
./bin/testcase10