1 What are you expected to deliver for this assignment?

This assignment is applicable to students enrolled in CSE 5302 or ECE 5402. All CSE 4302 students should ignore this assignment as it will not count towards their grade.

First, perform a `git pull` on your cse4302 cloned GitHub repository. A new directory, named pa4 will be added. It has the starting files to complete this programming assignment. You will be modifying simulator.c to dynamically schedule instructions using Scoreboard Algorithm in a multi-cycle 4-stage pipelined implementation for the riscy-uconn ISA. The details on the ISA and what modifications you need to make to the simulator.c are outlined in the next sections of this document. We have provided a sample test-case, fibonacci.asm, but you are responsible for writing your own testcases to ensure the implementation is correct both in terms of functionality and performance. For this assignment you may run the provided testcase using the simulator command: `./simulator fibonacci.out`.

Once you are ready to submit your simulator.c file for this programming assignment, you are required to email it to the TA and the instructor at mohsin.shan, khan@uconn.edu before the deadline. You will receive an acknowledgment email back from the TA if your submission is accepted. It is your responsibility to ensure timely submission of your programming assignment. In the subject line, you must state PA4 submission :< your fullname >. You may need to schedule a WebEx or Teams meeting with the TA to complete a demo of your assignment to get credit. This process must complete by December 20, 2020.

2 Scoreboard Algorithm to Exploit ILP

This programming assignment implements the Scoreboard Algorithm to dynamically exploit ILP in a 4-stage pipeline. The stages in this pipeline are: Issue, ReadRegs, Execute, and Writeback. This is a multi-cycle pipeline and the instructions will spend different number of cycles in the execute stage according to their type. MEMORY_LATENCY, BRANCH_LATENCY, and OTHER_LATENCY specify the number of cycles required by memory, branch, and other instructions.

The process instructions() function executes the pipeline backward by executing Writeback, Execute, ReadRegs, and Issue functions in this order. After going through these four stages, the cycle counter is incremented by one cycle, and the process instructions() function is executed again to evaluate the next cycle. A debug pipeline tracing function show_scoreboard() is added to display the contents of SCOREBOARD after each cycle.

In this assignment, you will modify the following functions in simulator.c: issue(), read_reg(), execute(), and write_back(). Your submission is required not to modify any of the other functions in simulator.c. Moreover, you cannot modify the contents of simulator.h, mips_instructionMap.h, and the Makefile.

2.1 SCOREBOARD Data Structure

SCOREBOARD is defined in simulator.h, and it is an array of scb_entry which represents a single entry in the Scoreboard. Each scb_entry has its own execution unit. Therefore, the number of execution units is equal to the number of scoreboard entries. Each scb_entry holds a single instruction and stores information regarding the stage, hazard, and type of that instruction. Since each entry in the Scoreboard has its own execution unit; therefore, scb_entry also stores the execution unit’s state. The following fields are present in scb_entry.

- stage: Indicates the pipeline stage of the entry.
- hazard: Stores the hazard (if any).
- busy: Indicates whether the entry is being used or not.
- operation and sub_operation: The operation and sub_operation(for RTYPEOP) to be performed by this entry.
- imm, shamt: Immediate and Shamt values in the instruction.
- dest_reg, src_reg1, src_reg2: Destination and source register identifiers.
• src_reg_1.data, src_reg_2.data: Data of the source registers.
• scb_1 and scb_2: Scoreboard entry that produces data for a source register.
• scb_1.ready, scb_2.ready: Indicates whether the data is available in the source register.
• cycles, cycles_to_wait: The number of cycles spent and remaining cycles in the execution stage.
• instruction_number: Indicates the instruction number of current instruction. It’s a unique id for every instruction assigned at fetch.
• updated_pc: Stores the branch address to use when the branch is resolved as taken in WB stage.

PIPELINE_STAGE enum represents the different stages of the pipeline, and HAZARDS_STAGE enum represents the different types of hazards.

2.2 REGISTERS_RESULT Data Structure
The size of this array is equal to the total number of registers in our ISA, i.e., 32. It contains the number of scoreboard entry which is going to write this register. If no entry is going to write this register, its value is set to -1.

2.3 instruction_number Global Variable
This is a up counter which is used to assign a unique instruction_number to an entry in the scoreboard.

2.4 committed_inst Global Variable
This is the number of committed instructions. It is incremented by 1 after an instruction completes the write_back() stage.

2.5 inst_num_of_taken_branch Global Variable
This is the instruction number of the branch that changed the control flow of the program. This is used to identify kill the mis-predicted instructions in the pipeline.

3 Issue (no return value)
This stage fetches and issues instructions. In this pipeline, each stage flushes its corresponding instruction(s) being processed when br_taken is asserted. The write_back stage asserts br_taken when a conditional branch is taken, or an unconditional branch (J, JAL, JR) changes the program control flow. The issue() stage implements a static not-taken branch predictor that assumes the program counter increments sequentially until a taken branch modifies the control flow. On a branch mis-prediction, the write_back stage asserts br_taken to initiate pipeline flush. If br_taken is asserted, the issue stage returns without any further processing, so it can re-execute the same PC in the next cycle. Moreover, before return, it de-asserts br_taken flag since issue() is the last stage being handled by the process_instructions() function. If br_taken is not asserted, the Issue() stage performs the instruction memory lookup using the current pc (i.e., pc/4 as address).

Issue stage processes one instruction in any given cycle to ensure in-order issue. First, the instruction is decoded and the contents are stored in a temporary reg_temp struct. After the decode is complete, the instruction is checked for WAW hazard with any active instruction in the scoreboard. This can be determined by monitoring the REGISTERS_RESULT array, which should be -1 for the destination register of current instruction. If there is a hazard then instruction must stall in the issue stage and re-execute next cycle. Then the SCOREBOARD is searched for an available entry, i.e any entry whose busy flag is set to 0. If no such entry is found, there is a structural hazard. If there is no structural hazard, the instruction is assigned to this scoreboard entry. Otherwise, issue() returns and re-executes next cycle.

To populate the scoreboard entry, the busy flag of the assigned entry is set to 1, and the stage is set to ISS. The cycles for the newly issued instruction is to 0 and the cycles_to_wait is set according to the type of instruction. The instruction_number of the current entry is set to the global instruction_number. Other scoreboard entry components are also setup based on the decoded information. Moreover, for instructions that update a register, the entry for destination register in the REGISTERS_RESULT is set to the id of
the assigned scoreboard entry. The stage member of the assigned entry is set to ISS, so that it be processed by the execute stage. Since we have a not taken predictor, pc is always incremented by 4 after a scoreboard entry has been assigned, and the issue() stage returns.

4 Read_reg(no return value)

This stage goes through all the entries in the scoreboard, and processes the entries that have completed the issue stage. This is done by checking the scoreboard busy and stage flags. This stage also kills the mis-predicted instruction if br_taken is asserted. In this case, all instructions with instruction number greater than inst_num_of_taken_branch must be killed. This is done by clearing the busy bit of the respective scoreboard entry, as well as the corresponding REGISTERS_RESULT entry.

If there is no mis-prediction, each active scoreboard entry checks if all its instruction operands are ready in this cycle. This can be done by checking the scb.1_ready and scb.2_ready members of the current scoreboard entry. These two members are only set during issue and write_back stage and are not re-evaluated in the read_reg stage. If the operands are ready, they are read from the registers and the scoreboard entry is marked done for read. In case the scoreboard entry is awaiting at least one operand, it is stalled in this stage due to RAW hazard. An instruction is allowed to read registers when RAW dependency is cleared. After an entry has read its registers, its stage is set to RD.

Memory Dependence Handling: Memory addresses for LW/SW are not computed until the execute stage. A younger load can have its address available while the older store is still waiting for the address. If the LW/SW were to be done at the same address then LW will have incorrect value. To cater this problem, this programming assignment takes a conservative approach: the LW instruction will stall in the read stage until all the previous stores have been completed.

5 Execute (no return value)

This stage goes through all the entries in the scoreboard and processes those entries which have completed the read stage. This stage also kills the mis-predicted instruction if br_taken is asserted. In this case, all instructions with instruction number greater than inst_num_of_taken_branch must be killed. This is done by clearing the busy bit of the respective scoreboard entry, as well as the corresponding REGISTERS_RESULT entry.

For this assignment, three configurable parameters MEMORY_LATENCY, BRANCH_LATENCY, and OTHER_LATENCY are statically defined based on their respective instruction type. First parameter is for LW, SW; second parameter is for J, JAL, JR, BNE, and BEQ; and the last parameter is for the remaining instructions in the ISA. By default, the first parameter is set to 3 cycles, second to 1 cycles, while third to 5 cycle execution latency. Each active scoreboard entry tracks its execution cycles for its assigned instruction in this stage. If an instruction requires multiple cycles to execute, it is stalled in this stage accordingly.

In the last cycle of execute for an instruction, the desired operation is performed to compute the result and the stage of the corresponding scoreboard entry is set to EX. This process is repeated for each active scoreboard entry in this stage.

6 Writeback (no return value)

Each active scoreboard entry that has completed its execution checks if it has a WAR dependency with an older instruction with an active scoreboard entry. If so, the scoreboard entry stalls in the writeback stage. Otherwise, the scoreboard entry’s instruction is allowed to writeback by either writing to the destination register or updating memory. Moreover, the scoreboard entry also updates scb.1_ready and/or scb.2_ready of any instruction that is waiting for the current instruction to complete. Note that multiple instructions may be ready to writeback in a given cycle, and the machine imposes no restrictions on the number of writebacks. Therefore, the writeback stage attempts to commit as many instruction as it can in a given cycle. After the instruction has completed, its scoreboard entry is de-allocated by de-asserting the busy flag of the entry and REGISTERS_RESULT is set to -1 (if required). Moreover, the committed_inst is also incremented by 1 to keep count of committed instructions.

Control Flow: If the scoreboard entry that is ready to writeback is a control flow instruction (J, JAL, JR, BNE, and BEQ) that updates the PC, then the writeback stage (1) asserts the br_taken signal, and (2) updates the inst_num_of_taken_branch which is the instruction number at which branch was taken.