CS 520 - Fall 2016 - Ghose
Project 2: Full Version

3 to 4 students to a project team
Due: Monday, December 12

This is a preliminary specification to get you started. Further details will follow.

For Project 2, you need to extend the simulator that you have developed for Project 1 with a renaming mechanism that uses a unified register file, a centralized IQ and a ROB. There is no LSQ.

You can implement the IQ in any way within the simulator as long as wakeups and selection are implemented correctly. **The wakeup signal for a function unit is generated one cycle before the function unit completes to support back-to-back execution**. Ties for selection of a specific function unit are broken using a FIFO policy that selects the instruction dispatched earlier. You can implement the FIFO by adding an IQ entry field that holds the cycle in which the instruction was dispatched. Speculative execution is not supported.

The following function units are used and all function units, excepting the branch FU, has a writeback stage with a one cycle latency:

- A two-stage pipelined integer ALU (two stages, one cycle per stage) implementing all arithmetic instructions excepting a multiply. This function unit also implements the MOVC instruction by adding an implicit zero value to the literal in MOVC and writing it to the destination.
- A non-pipelined multiplication unit with a latency of 4 cycles that implements the multiply operation.
- A single cycle branch FU that computes the target address and decides whether to branch or not. This function unit also implements the JUMP and BAL instruction.
- A two-stage pipelined LSFU (one cycle per stage) implementing the LOAD and STORE instructions. LSFU generates memory address (stage 1 of LSFU), performs TLB lookup (LSFU 2nd stage) and then accesses cache when LOAD or STORE is at the head of the ROB. Assume Cache access performed by LSFU retrieves data in one cycle. There is no bypassing of earlier STORES by a later LOAD. When a LOAD completes, the result is written to the destination via the associated WB stage.

The unified register file has enough write ports to allow results from 3 FUs to be written to the unified register file. As a result is written to the unified register file, it is also forwarded to waiting instructions. Up to 3 wakeup signals can be sent per cycle and 3 results can be simultaneously forwarded per cycle. Only one commit per cycle is allowed.

You will need to implement all artifacts associated with renaming for a unified register file, including the rename table, the back-end register alias table, register allocation and deallocation operations.

Further details are included below. All specifications above this line are the same as in the "Preliminary" version that was posted earlier.

Details:

1. Instructions, formats, data memory addressing, instruction accessing etc. are all as in Project 1. Forwarding has to be implemented. Be sure to consider all forwarding scenarios! Assume 16 architectural registers (R0 through R15) as before and a unified register file with 32 registers as the default. (The number of registers in the URF can be changed before simulation starts using the Set_URF_size command (see next page). The IQ size is 12 and ROB size is 40.

2. The dispatch of a branch or any control flow instruction (BZ, BNZ, BAL, JUMP) stalls till the previous branch or the previous control flow instruction has been **issued**.
3. In the cycle that a result is being written to a destination register, it can be forwarded to the instruction that needs it. An instruction that needs this result as an input can begin execution in the same cycle in which the result is being written back to the register. That is, forwarding and writeback takes place in the same cycle.
4. Issue of instructions to LSFU take place in program order - issued op stays there, keeping 2nd stage of LSFU busy till the matching in the ROB entry moves to head of ROB. Note that for other FUs, issues can take place out of program order and program order is used only to break any ties in case two awakened instructions need the same FU.
5. Allocate free registers in ascending order of their address. As an example, if P5, P8 and P14 are free, P5 is allocated first. At the end of a cycle, after registers are freed up, they are added to the free list and the free list of physical register is sorted. In the next cycle, the allocation step uses the newly-sorted free list.
6. Add the simulator commands listed below. All commands, except the Set_URF_size command, are invoked after the simulation stops after the number of cycles specified in the "Simulate" command.

- Set_URF_size <n>: used before simulation to set the number of registers in the unified register file to n.
- Print_map_tables: prints front rename table and back-end register alias table.
- Print_IQ: prints issue queue entries and their status, one entry per line.
- Print_ROB: prints current ROB contents, one entry per line.
- Print_URF: Prints contents of URF and their status (allocated, committed, free)
- Print_Memory <a1> <a2>: prints out contents of memory locations with addresses ranging from a1 to a2, both inclusive. The addresses a1 and a2 are at 4 Byte boundaries.
- Print_Stats: prints the IPC realized up to the point where this command is invoked, the number of cycles for which dispatched has stalled, the number of cycles for which no issues have taken place to any function unit, number of LOAD and STORE instructions committed (separately).

The submission requirements are exactly the same as the ones for Project 1.