CS 520 - Fall 2024 - Ghose

Programming Project 2: Out-of-Order APEX Processor

To be done by a team of 2 to 3 students

PART 1 (Detailed design documents) DUE: 11/11 (Mon)

PART 2 DUE (All code and other documents): 12/6 (Fri)

All demos to be completed by end of finals week

****EARLY SUBMISSION IS ENCOURAGED****

This project requires you to implement a cycle-by-cycle simulator for an out-of-order processor. The processor implements the same ISA as the one for Project 1, with 32 general-purpose architectural registers, R0 through R31 and an architectural register CC for the flags. The **two** exceptions to this is that that the JAL instruction is replaced by the JALP instruction and a special RET instruction is introduced to return from a function call. The format of JALP and its semantics are as follows:

JALP <dest> #<signed literal>

This instruction saves the return address in the specified destination register, <dest> and transfers control to the address obtained by adding the signed literal to the address of the JALP instruction.

The format of RET and its semantics are as follows:

RET <src>

This instruction takes the return address saved in the specified source register, <src> and transfers control to the address saved in the <src> register.

To facilitate control flow prediction for the JALP and RET (see later), additional actions are needed in the processor that you are simulating – these are described later.

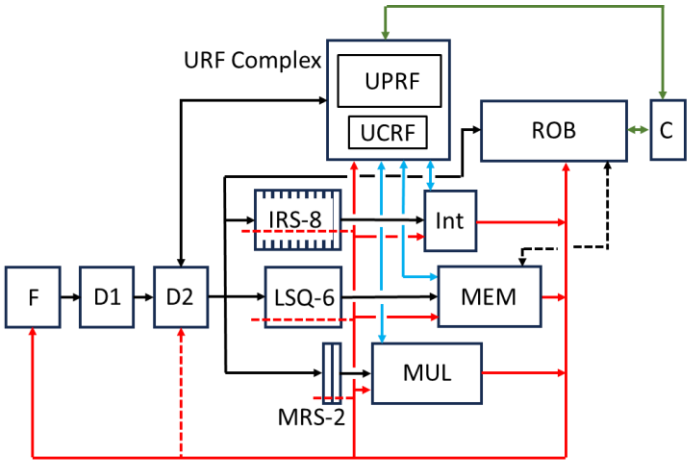
PROCESSOR DATAPATH AND KEY FEATURES

The datapath of the processor to be simulated is shown in the figure above. This processor uses:

- Three function units as shown, and as follows:
  1. A function unit for integer and bitwise logical operations, IntFU (shown as Int in the diagram), **with a single stage**. Branch, JALP, JUMP and NOP instructions also use this function unit. The RET instruction's control flow path is also verified by the IntFU (see later). Branch directions are also verified by the IntFU against what was predicted.
  2. A function unit, MUL, for the MUL instruction, pipelined into 4 stages. Operations complete for the MUL function unit in the last stage.
  3. A dedicated function unit, MEM, pipelined into 3 stages, that calculates the memory address and performs memory accesses for loads and stores. To calculate the memory address for a load or store instruction (including LOAD, STORE, LTR, STR), the LSQ entries have fields that contain values that are used to calculate memory addresses. Values of register operands used to calculate a memory address for a load or store are forwarded to the respective entries. The actual memory address is calculated when the LSQ entry for a load or store and its corresponding entry in the ROB

are at the head of these structures. The address calculation takes place within the first stage of 3-stage MEM function unit and the memory accesses are performed in the next two stages. **Bypassing or forwarding from a store to a load is NOT supported within the LSQ**. Operations complete for the MEM function unit in the last stage.

- Each function unit has its own forwarding bus, so arbitration to any of these buses is unnecessary. Forwarding tags are broadcast one cycle before the actual information that is forwarded. An instruction entering a function unit can pick up the forwarded information as an input operand as it is entering the function unit. Any instruction using a forwarding bus is removed from the last stage of the corresponding function unit **when it uses the forwarding bus for the actual information (register value or CC flag values) as it is broadcasted**. (During tag broadcast, instructions continue in the function units).



- Connections for checking/setting various status bits and connections to/from UPRF and UCRF are not shown for simplicity
- Structures like the front-end and back-end rename tables, the control flow predictor for JALP, RET and branch instructions are also not shown

- The instructions are fetched in one cycle by the Fetch stage (F). Decoding, renaming and dispatching take place over two stages, D1 and D2. The Fetch stage incorporates a control flow predictor (see below).
- Instead of a centralized issue queue, distributed reservation stations are used:
    a. A reservation station for the integer function unit, IRS-8 with 8-entries.
    b. A LSQ with 6 entries, LSQ-6, which serves as the reservation station for MEM.
    c. A 2-entry reservation station, MRS-2, for the MUL function unit.

  Ties in IRS-8 and MRS-2 for issuing instructions are broken to favor the entry that was created earliest in time. Although this is impractical for real designs, the simulation implements this with a timestamp for entries in these two queues that has the clock cycle in which the entry was established as a timestamp.
- A unified register file for values that target general purpose architectural registers, shown as UPRF, with 60 entries.

- A unified register file for the condition code register, CC, called UCRF with 10 entries which holds values of the flags Z(ero), N(egative) and P(ositive).
- A "renamer deallocates" policy is used to free up entries in the UPRF or UCRF. Each of these register files have their own allocation list, arranged as a FIFO queue. Before any execution starts, all physical registers in these two register files are free and these free lists are organized as FIFO in increasing order of the register addresses. Because unified physical registers are used in this design, you need to implement a rename table for use during a dispatch and a back-end rename table used for commitment.
- A reorder buffer, ROB, with 80 entries.
- A **control flow predictor** is used by the Fetch stage to predict the direction of control flow for a (conditional) branch instruction (BZ, BNZ, BP, BN) or by the JALP and RET instructions. This has a total of 8 entries set up in FIFO order. When the 8 entries are full, the establishment of a new entry replaces the earliest entry in the predictor's table. Entries are looked up using the address of the instruction to be fetched. Each entry has the following fields:
    - Instruction type of matching entry. There are **3 types**: **branch instruction** (which include BZ, BNZ, BP, BN), **JALP** and **RET**.
    - A field for the target address of the branch instruction or JALP if the entry is for a branch instruction or JALP.
    - A pointer to a separate 4-deep return address stack, used by the RET instruction if the entry is for a RET instruction.
    - Prediction information, if the entry is for a branch instruction.

  The default prediction used is that any (conditional)branch with a negative offset is always taken, while a conditional branch instruction with a positive offset always behaves the same as its last execution. A JALP or RET is always considered as taken.

TIMINGS

The timings of specific operations are as follows:

- Function unit latencies are as specified earlier.
- The forwarding of a tag or the required information (register value or flags) take one cycle each. When the required information is forwarded, the corresponding instruction does not sit in the last stage of the function unit any more.
- The F, D1 and D2 stages take one cycle each. In one cycle, the F(etch) stage also looks up the control flow direction from the control flow predictor. Assume that updates made to the control flow predictor are made as soon as the control flow direction is resolved along with the calculation of a target address. Assume that on a miss in the control flow table, an instruction is decoded in the D1 stage and if it is found to be a conditional branch instruction or to be a JALP, an entry is created in the predictor's table from the D1 stage.
- The UPRF and UCRF register files are read out at the time of issue. Any forwarded operand can be picked as an instruction moves into the required function unit. Note that because of the use of early tag broadcast, physical register reading time and forwarding delays are effectively absorbed. Use this fact to simplify your simulator code!

- Unless forwarded, the contents of a register can be read after the cycle in which it is written to.
- The register reading for source operands
- Instruction commitment takes one cycle, using the stage C with a delay of one cycle. When an entry at the head of the ROB is waiting only for the value of a register and/or flags for commitment, commitment takes place at the end the same cycle as the value is forwarded, using C.

CONTROL FLOW DETAILS AND THE PREDICTOR

As mentioned earlier, the control flow instructions that use the predictor (which are the branches, JALP and RET) **have entries that are specific to each type** mentioned earlier.

I. The return address stack

A 4-deep return address stack is used by the processor and this is part of the control flow predictor. As part of a JALP instruction, the calculated return is pushed onto the stack (which is part the Fetch stage) at the same time as the destination register of the JALP is updated via the forwarding bus. On a lookup in the predictor which returns an entry whose "type" field indicates that it is for a RET instruction and also the fact that a RET instruction is being fetched, the return address stack is popped to get a predicted return address and the instruction fetch path is changed immediately to alter the control flow path. Predicted return addresses need to be verified (see below).

II. Issuing control flow instructions

All control flow instructions are dispatched to the 8-entry IRS queue. They are issued as follows:

- For branch instructions, the decision to take a branch or not is made in the IntFU itself. So, the branch instructions are ready to issue when the flag value to be tested is available.
- Similarly, a JUMP instruction is ready to issue only when the source register value used to calculate the target address. **Note that JUMP does NOT make any use of the predictor, as its target address is dependent on the value of the register**.
- A JALP needs to calculate the target address in the IntFU and is ready for issue after spending at least one cycle in the IRS reservation station.
- A RET is ready to from the IRS when its source register is available.

III. Verifying the predictions for control flow instructions that use the predictor

These verifications are carried out as follows within the time they spend in the IntFU:

- For branch instructions, where the target address does not change from one execution of a branch to another because of the use of PC-relative addressing, the only thing that needs to be verified is the branch condition actually evaluated against what was predicted by the predictor (or by default).
- For a JALP instruction, as PC-relative addressing is used and because the target address is specified relative to the PC, no verification is needed.
- **NO LONGER REQUIRED- ASSUME THAT STACK DOES NOT OVERFLOW OR UNDERFLOW** ~~For a RET instruction, the predicted target address popped of the 4-deep stack needs to be verified against what is stored in the specified source register, <src>.~~

To facilitate verification within the IntFU, whatever information is needed to be used for the verification needs to flow with these instructions into the IntFU from the Fetch stage (where the predictor lookup stakes place).

IV. Dealing with control flow mispredictions

The actual control flow path needs to be verified against the prediction and appropriate actions need to be taken if the predicted path was incorrect. As stated earlier, these verifications are carried out by the IntFU. When a misprediction is discovered for a branch or RET instruction fetched or processed along the incorrect path need to be flushed and the predictor needs to be updated appropriately. To simplify your implementation, assume that the processor does the following:

On decoding an instruction to be a (conditional) branch or RET, the following structures are checkpointed (that is saved) for restoration on a potential misprediction:

- The two rename tables – one at the front end and another at the back-end.
- The free lists for UPRF and UCRF, along with valid bits of registers.
- The return address stack.
- Any other structure relevant to restoration as used in your code.

On a misprediction, the entire pipeline is stalled and actions are taken to handle the misprediction (flush instructions along the mispredicted path, restore rename table and its back-end counterpart, free lists, register status, return address stack, any change to be made to the predictor etc. **To keep the simulator code simplified, assume that any time at most a single branch ~~or RET~~ instruction is in execution, while any other have to wait in IRS for issue; some or all of these will be flushed on a misprediction.**

Instructions along control flow path also needs to be tagged to facilitate flushing – you can use something like a BIS, as discussed in the lectures on speculative execution to do this.

You also need to make sure that when a predictor lookup is done for the RET, there is something in the 4-deep return address stack used by the predictor. If at the time of fetching a RET, the 4-deep return address stack is empty, there is not prediction to use and you will need to establish an entry and fill that entry in later when RET has exited the IntFU.

OTHER ITEMS

The memory map, simulator commands and instruction formats are similar to those of Project 1. For debugging and tracing, you also display other things: the prediction made, the contents of the return address stack etc.

WHAT TO SUBMIT

- **PART 1 (DUE Nov. 11th):** This will be a detailed design for the simulator code – all major data structures, what the main functions are, including details of predictor and prediction, decoding, dispatch, execution, forwarding and commitment. **It is understood that you may make changes to your design, including some radical changes!**
- **PART 2 (DUE Dec. 6th): *TWO* VERSIONS OF THE SIMULATOR**:
    - **One without any predictor, another with the predictor described – both for the datapath shown**

- **All test cases that you have used**
- **Updated design documents**

PLEASE START EARLY – NO EXTENSIONS ALLOWED THIS TIME WITHOUT SEVER PENALTY