CS520, Spring 2018: Homework 4

Due:	Friday, April 20
Late penalty after:	Sunday, April 22
Cutoff:	Wednesday, April 25

Steps and Objectives

- Make a new clone of the infrastructure repo: https://github.com/theosib/CpuSimulator
- Port your work from HW3 to this clone. There's probably not a whole lot to do for this step, although you might want to do some clean-up after seeing how I did HW3.
- **Implement register renaming. See below for steps.**

New tools and features

Properties and Globals

- On top of the datatypes handled by PropertiesContainer (interface IProperties), the GlobalData (interface IGlobals) adds support for a register file data type:
    - **public IRegFile getPropertyRegisterFile(String name) { … }**
- You may need this later to implement reorder buffer. Meanwhile, there is a method that provide backward compatibility:
    - **public IRegFile getRegisterFile() {**
    - **return getPropertyRegisterFile("reg_file");**
    - **}**
- (Note: At this point, the globals may be generic enough that we can move that class out of the implementation directory.)

Register file and register properties

- Previously, architectural registers had two properties, *invalid* and *is_float*. Now they have four properties (see utilitytypes/IRegFile.java):
    - FLAG_INVALID - Waiting on instruction to complete and fill the register.
    - FLAG_FLOAT - Although stored in an int, the value in this register was produced by a floating point instruction (directly or indirectly).

    - FLAG_USED - Register is allocated and is associated with a producer instruction and has a register alias table entry pointing at it.
    - FLAG_RENAMED - Only applied to "used" registers. When this flag is set, it indicates that the RAT entry that had referred to this physical register is now referring to a different physical register. Once you implement the issue queue (HW5), then there may still be instructions that want this register as a source, but it cannot be used as a source for any NEW instructions, since no RAT entry refers to this register.
- Important methods:
- **public boolean isUsed(int index);**
    - Returns true if FLAG_USED is set on the specified register.
- **public boolean isRenamed(int index);**
    - Returns true if FLAG_RENAMED is set.
- **public boolean isInvalid(int index);**
    - Returns true if FLAG_INVALID is set.
- **public boolean isFloat(int index);**
    - Returns true of FLAG_FLOAT is set
- **public int getFlags(int index);**
    - Returns the raw flag bit vector for the specified register.
- **public void markUsed(int index, boolean is_used);**
    - Sets or clears the USED state of the specified register based on the is_used argument. This change does not take effect until the next clock cycle.
- **public void markRenamed(int index, boolean is_renamed);**
    - Sets or clears the RENAMED state of the specified register based on the is_used argument. This change does not take effect until the next clock cycle (in other words, this setting is "clocked").
- **public void setInvalid(int index, boolean inv);**
- **default public void markValid(int index) { setInvalid(index, false); }**
- **default public void markInvalid(int index) { setInvalid(index, true); }**
    - Various ways to set or clear the INVALID flag. (This setting is clocked.)
- **public void markFloat(int index, boolean is_float);**
    - Sets or clears the FLOAT state depending on the argument.
- **public void changeFlags(int index, int flags_to_set, int flags_to_clear);**
    - Set and/or clear multiple flags at once. (See IRegFile.java for the SET_ and CLEAR_ constants that can be bitwise OR'd together for each argument.)
- **public void markPhysical();**
    - Register files default to being "architectural." Calling this method specifies that the register file is "physical."
- **public boolean isPhysical();**
    - Returns true if this is a physical register file.
- For other features of the register file, see the javadocs and other comments in IRegFile, RegisterFile, and manual/manual.html.

Operands and renaming

- Operands now have an **is_renamed** flag that keeps track of whether or not a register operand has been renamed.
- Important methods:
    - **public boolean isRenamed();**
        - Returns true if the operand is a register and it has been renamed.
    - **public int getRegisterNumber();**
        - Returns the architectural register number if the operand has not been renamed.
        - Returns the physical register number if the **has** been renamed.
    - When we need it, I will provide you a method to retrieve the original architectural register number for a renamed operand.
    - **public void rename(int phys_reg_num);**
        - Rename an architectural register to the given physical register.
        - Throws an exception if the register is already renamed.

Changes you have to make for register renaming

All renaming is controlled in the Decode stage.

Globals

- **Previously**, you had only an architectural register file with 32 entries.
- **Now**, you need to replace that with a physical register file with 256 entries. You make it a physical register file by calling **markPhysical()** on the register file once you have created an instance.
- You also need a 32-entry int array to hold your register alias table.
- Somewhere, you need to initialize the RAT to a sensible starting state, such as looping over all 32 architectural registers and setting **rat[i] = i**, so that arch reg **Ri** initially points to phys reg **Pi**. In the physical register file, you also have to mark the first 32 registers as "used." Personally, I added the code for this into MyCpuCore.initProperties.

How to rename source registers

- When to rename source registers in Decode:
    - **After** making a duplicate of the input Latch (so that you don't change the original)
    - **After** squashing any fall-through instruction for taking a branch

    - **After** checking to see if the in-coming instruction is null (and returning from the method if that is the case)
    - **Before** calling registerFileLookup.
- Look up the RAT from the Globals
- For each source operand that is a register (isRegister):
    - Get the architectural register number (getRegisterNumber).
    - Look up that number in the RAT.
    - Call the rename method on the Operand, passing the physical register number you just looked up.
    - Don't forget to also check opcode.oper0IsSource() for renaming oper0 when it is a source.

How to rename the destination register

- This only applies to instructions that actually have a destination register.
- After renaming the sources is the point where I recommend identifying which free physical register will become the renamed destination.
    - By some means (stack, brute force search, whatever), find a physical register whose USED flag is not set.
    - DO NOT at this time mark this register as USED. You don't yet know whether or not you're actually going to use it.
- For all instructions that have a destination register (opcode.needsWriteback() returns true, which includes CALL), you need to be sure that the next stage you will send the instruction to is able to accept that instruction (output.canAcceptWork() returns true) before you mark the destination register as USED. The correct order is:
    - Compute, based on opcode, which functional unit will receive the instruction being processed.
    - Look up the corresponding output pipeline register by name.
    - Bail out if that next stage is unable to accept new work:
        - **if (!output.canAcceptWork()) return;**
    - Only after this point is it appropriate to mark the chosen destination physical register as USED.
- Once all the necessary conditions are met, the procedure for renaming a destination register is as follows:
    - Get the architectural register number of oper0
    - Look up the **old** physical register for its entry in the RAT.
    - Use the markRenamed method on the register file to indicate that the RAT is no longer pointing at the **old** physical register.
    - On the **new** physical register (that you selected earlier), make the following flags changes (with regfile.changeFlags(), you can do them all in a single call):
        - Set USED

        - Set INVALID
        - Clear FLOAT
        - Clear RENAMED
    - Update the RAT entry for the architectural register number with the **new** physical register number.
    - Call the rename method on oper0 with the new physical register number.
- Besides the instruction (with renamed operands), do not forget to copy from the input latch to the output latch key properties, such as forward0, forward1, and forward2. (The copyAllPropertiesFrom() method will do this for you.)

Freeing physical registers

- Once a physical register is no longer needed, it is necessary to free it. A physical register may be freed when both of the following are true:
    - Its INVALID flag is **clear/false** (implying that all dependencies are satisfied)
    - Its RENAME flag is **set/true** (indicating that the RAT no longer points to it)
- Each clock, cycle use some means of identifying which phys regs can be freed (e.g. brute force search, a list of registers whose states were changed on the prior clock cycle, whatever).
- When such a register is identified, clear its USED flag.