Implementation of the Microarchitecture

Overview

Each device consists of a functional unit that performs the desired function, one or more input ports, and one or more output latches. The device receives control signals if necessary and receives a periodic clock signal. When a clock signal is received (typically on the leading edge), the result of the functional unit is loaded into the latch. The operation of a device can be summarized as:

- At the start of a cycle, the values at the input ports start propagating through the logic of the functional unit.
- The device control signals may alter the function that is computed by the device.
- The result should be computed at some point during the cycle.
- The cycle ends when a clock signal is received, at which point the result of the functional unit are loaded into the output latch. A new cycle starts, and the values in the latch can now propagate to the input ports of the next device that is connected downstream.

Example

An adder consists of two input ports, a functional unit that implements the logic of the addition, and an output latch that receives the results of the addition at the next clock cycle. The input ports of the adder are connected to the source of data, typically the output latch of the register file. The adder, being a very simple device, doesn't receive control signals.

The Task

You will implement a microarchitecture by using the devices from the list below. If your design needs a device that is not listed, contact the instructor to agree on a reasonable alternative. To put some discipline into the coding, it is recommended that you follow this methodology:

- Implement the functional unit of a device.
- Implement the connection of the input ports to the sources of data (typically output latches of other devices).
- Implement the state transfer at the receipt of a clock.
- Congratulations, you have a device module.

It is recommended that you use object-oriented programming, where every device is represented by an object (you can also implement ports and latches as objects). The connections will be implemented as methods on the object at the beginning of the program. The connections, along with the control signals, encode the microarchitecture of the processor.

Importance of Unit Testing

Do not implement the microarchitecture at once and then start testing. Instead, write unit tests for each device you finish, make sure it works using your unit tests. As you add more devices, you will add more unit tests and you should have some form of a regression test that you conduct periodically (every night) as the tests accumulate. Once you are done with the devices individually, start implementing the connections and test the connection in the same incremental, iterative way. Then start testing each instruction on its own (one at a time). You also are adding now larger tests for each instruction. As you add things incrementally, the older unit tests will become more useful to you as they show you when you are breaking things.

Example

An adder will look like something like this:

```
class Adder : Device {
public:
        Adder()                 {}      // Initialize the input ports and the latch as necessary
        receive_clock()         { out.value = result; }
        do_function()           { result = in[0].connection.value + in[1].connection.value; }
        // receive_clock and do_function should be virtual functions inherited from Device and
implemented here
        connect (int port_id, Latch l)  { in[0].connection = l; }
private:
        Port in[2];
        Latch out;
        long long result;
};
```

Implementing the Instructions

For each instruction, you need to implement the flow of data through the microarchitecture to produce the required result. This should consist of the following step:

- When an instruction is fetched and decoded, you will need to construct the schedule of the instruction. This should be a schedule that is encoded into the control registers and enable the data to flow through the microarchitecture to enable the implementation .

Now you can test the flow of the operations through the microarchitecture and check the actual result.

Example

To implement the instruction **add r1, r2, r3**, you will implement the following operations (just like in class):

- Send a control signal to the register file to read r2.
- Route the results to the adder's first port, send a control signal to register file to read r3.
- Route the result to the adder's second port.
- The adder performs the addition.
- Route the result of the adder to the register input port, and send a signal to the register file to store the data in the input port in r1.
-

Implementing the Instruction Scheduler

Now that you have every instruction implemented and tested (and accordingly, more unit tests have been added), it is now time to implement the instruction scheduler.

- Before encoding the instruction steps on the control register, ensure that the dependencies among the instructions are respected. Use either a scoreboard or Tomasulo's algorithm. Both were covered in the earlier homework during the semester.
- Add to the existing schedule (stored in the control register) the necessary operations. At this point, if the first step did its job well when it comes to avoiding data hazards, then there will be no conflict.

Implement the Processor

With the microarchitecture running, you can now implement stage 1 of the project. Everything now should fall in place nicely. Just construct a program as follows:

- Create all devices.
- Connect the devices.
- Start the execution loop:
    - Fetch and decode an instruction (or more, if multi-issue).
    - Schedule the instruction.
    - For all devices, call do_function
    - For all devices, call receive_clock
    - Until you hit the halt instruction

Stage 1 is now complete.