Instruction Fetch and Decode

Instruction fetch and decode is the first unit in the processor that receives an incoming architecture-level instruction and handles it. The fetch and decode unit uses the opcode field in every instruction to identify it. This could be done for the purpose of this project by using the opcode field as an index into an array of control signal vectors that are stored in the decode unit. The vector identifies for the instruction a string of bits that represent the control signals that must be issued to carry out the necessary steps to implement the instruction. The vector length depends on the number of pipeline stages necessary to implement the instruction.

Example

Consider the instruction:

and r_d, r_s, r_t

This can plausibly be done as follows:

Step 1: Send control signals to the register file to read r_s and r_t

Step 2: Send routing instructions to route the output of the register file to the input ports of the logic unit.

Step 3: Send control signals to the logic unit to perform the “and” operation on the values in the input port.

Step 4: Send routing instructions to route the output of the logic unit to one of the input ports of the register file, and send the numeric representation of r_d to the other port, with control signals to store the value into register $r_{d.}$

Thus, this is an operation that can potentially be completed in four steps on the pipeline. The corresponding vector stored in the decode unit will contain four entries, each with all the control signals that must be asserted to carry out the instruction. Using the timer, these entries will be loaded into the control register which connects to all the functional units one step at a time. These control signals will be “ored” with the previous and subsequent instructions that are concurrently executing on the pipeline.

Checking for Hazards

There will be register, control and resource conflicts. You need to implement a mechanism to avoid these conflicts and schedule the steps of each instruction correctly. You can use a scoreboard or Tomasulo’s algorithm. However, your implementation needs to stick to realism as much as possible. Ideally, this part should have been implemented using VHDL, but this will be too slow for the purpose of this project. However, this is not a license to introduce “magic” where you put in your C++ code complex multi-statement functions. To the extent that is possible, stick to what a real hardware implementation does: 1) Check logic, 2) Simple value manipulation, and 3) indexing into arrays.

Multi-Issue and Out of Order

It is up to each team to decide the level of sophistication that they would like to implement. It is highly recommended that you start with a very simple, single-issue/in-order scheduler and ensure it runs before you try more sophisticated structures. Note that power and area need to be measured to avoid just “throwing hardware at the problem”.