Okay, we're done!

Here's the final datapath for executing instructions and handling exceptions.

Please take a moment to remind yourself of what each datapath component does, i.e., why it was added to the datapath.

Similarly, you should understand how the control signals affect the operation of the datapath.

At least to my eye, this seems like a very modest amount of hardware to achieve all this functionality!

It's so modest in fact, that will ask you to actually complete the logic design for the Beta in an upcoming lab assignment :) How does our design compare to the processor you're using to view this course online?

Modern processors have many additional complexities to increase performance: pipelined execution, the ability to execute more than instruction per cycle, fancier memory systems to reduce average memory access time, etc.

We'll cover some of these enhancements in upcoming lectures.

The bottom line: the Beta hardware might occupy 1 or 2 sq mm on a modern integrated circuit, while a modern Intel processor occupies 300 to 600 sq mm.

Clearly all that extra circuitry is there for a reason!

If you're curious, I'd recommend taking a course on advanced processor architecture.

Here we've gathered up all the control signal settings for each class of instructions, including the settings needed for exceptions and during reset.

Wherever possible, we've specified "don't care" for control signals whose value does not affect the actions of the datapath needed for a particular instruction.

Note that the memory write enable signal always has a defined value, ensuring that we only write to the memory during ST instructions.

Similarly, the write enable for the register file is well-defined, except during RESET when presumably we're restarting the processor and don't care about preserving any register values.

As mentioned previously, a read-only memory (ROM) indexed by the 6-bit opcode field is the easiest way to generate the appropriate control signals for the current instruction.

The Z and IRQ inputs to the control logic will affect the control signals and this can be accomplished with a small amount of logic to process the ROM outputs.

One can always have fun with Karnuagh maps to generate a minimal implementation using ordinary logic gates.

The result will be much smaller, both in terms of size and propagation delay, but requires a lot more design work!

My recommendation: start with the ROM implementation and get everything else working.

Then come back later when you feel like hacking logic gates :) So that's what it takes to design the hardware for a simple 32-bit computer.

Of course, we made the job easy for ourselves by choosing a simple binary encoding for our instructions and limiting the hardware functionality to efficiently executing the most common operations.

Less common and more complex functionality can be left to software.

The exception mechanism gave us a powerful tool for transferring control to software when the hardware couldn't handle the task.

Have fun completing the hardware design of your Beta.

Thousands of MIT students have enjoyed that "Yes!" moment when their design works for the first time.

For their efforts we reward them with the "Beta Inside" sticker you see here, which you can see on laptops as you walk around the Institute.

Good luck!