Finally, our last instruction!

Branches conditionally transfer control to a specific target instruction.

But we'll also need the ability to compute the address of the target instruction - that ability is provided by the JMP instruction which simply sets the program counter to value from register "ra".

Like branches, JMP will write the PC+4 value into to the specified destination register.

This capability is very useful for implementing procedures in Beta code.

Suppose we have a procedure "sqrt" that computes the square root of its argument, which is passed in, say, R0.

We don't show the code for sqrt on the right, except for the last instruction, which is a JMP.

On the left we see that the programmer wants to call the sqrt procedure from two different places in his program.

Let's watch what happens… The first call to the sqrt procedure is implemented by the unconditional branch at location 0x100 in main memory.

The branch target is the first instruction of the sqrt procedure, so execution continues there.

The BEQ also writes the address of the following instruction (0x104) into its destination register, R28.

When we reach the end of first procedure call, the JMP instruction loads the value in R28, which is 0x104, into the PC, so execution continues with the instruction following the first BEQ.

So we've managed to return from the procedure and continue execution where we left off in the main program.

When we get to the second call to the sqrt procedure, the sequence of events is the same as before except that this time R28 contains 0x67C, the address of the instruction following the second BEQ.

So the second time we reach the end of the sqrt procedure, the JMP sets the PC to 0x67C and execution resumes with the instruction following the second procedure call.

Neat!

The BEQs and JMP have worked together to implement procedure call and return.

We'll discuss the implementation of procedures in detail in an upcoming lecture.

That wraps up the design of the Beta instruction set architecture.

In summary, the Beta has 32 registers to hold values that can be used as operands for the ALU.

All other values, along with the binary representation of the program itself, are stored in main memory.

The Beta supports 32-bit memory addresses and can access values in 2^32 = 4 gigabytes of main memory.

All Beta memory access refer to 32-bit words, so all addresses will be a multiple of 4 since there are 4 bytes/word.

The are two instruction formats.

The first specifies an opcode, two source registers and a destination register.

The second replaces the second source register with a 32-bit constant, derived by sign-extending a 16-bit constant stored in the instruction itself.

There are three classes of instructions: ALU operations, LD and ST for accessing main memory, and branches and JMPs that change the order of execution.

And that's it!

As we'll see in the next lecture, we'll be able parlay this relatively simple repertoire of operations into a system that can execute any computation we can specify.