Introduction

Before we discuss designing a datapath to handle the execution of MIPS machine instructions, we need to consider the major hardware elements that will be available.

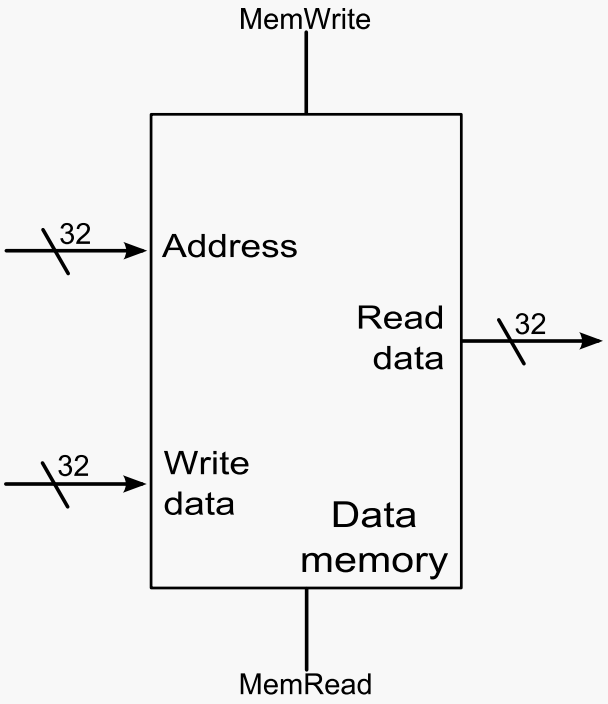
For the simple design we'll consider first, the following components will be needed:

- a memory unit to store instructions and data values
- registers to store instructions and data values within the processor
- an arithmetic-logic unit (ALU) to implement basic operations
- multiplexors to allow selection among different choices for operands and data values
- a control unit that decodes instructions and manages other elements during execution
- a sign-extension unit to "widen" values from 16 to 32 bits
- a shift unit to perform multiplication by a power of 2

Memory Unit

The memory unit will have the following interface:

- input: a 32-bit addresses, which specifies the location to be accessed
- input: a 32-bit data value to be written to that address
- output: a 32-bit data value that has been read from that address
- control: a single bit indicating whether the address should be written to
- control: a single bit indicating whether the address should be read from



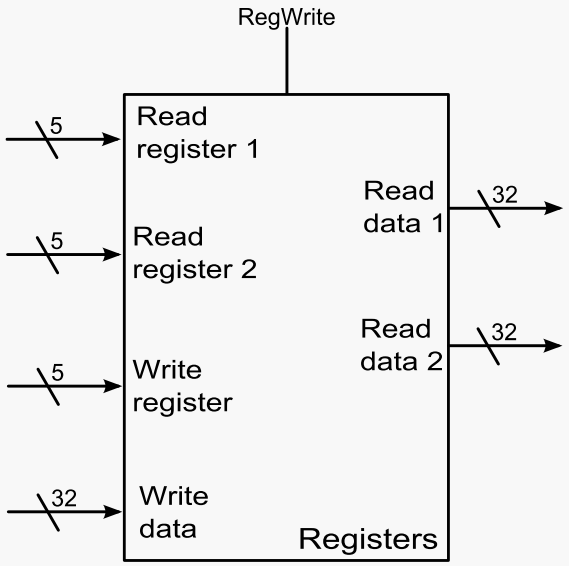
Registers

A register is capable of storing a 32-bit value persistently, and can be read from and written to.

There will be 32 general registers, and discussed before, specified by numbers 0-31.

These registers will be organized in a *register file*, which will have the following interface:

- input: two 5-bit register numbers, which specify two* registers to be read from
- input: one 5-bit register number, which specifies a registers to be written to
- input: a 32-bit data value to be written to the specified (write) register
- output: two 32-bit data values that have been read from the read registers
- control: a single bit indicating whether the write register should be written to

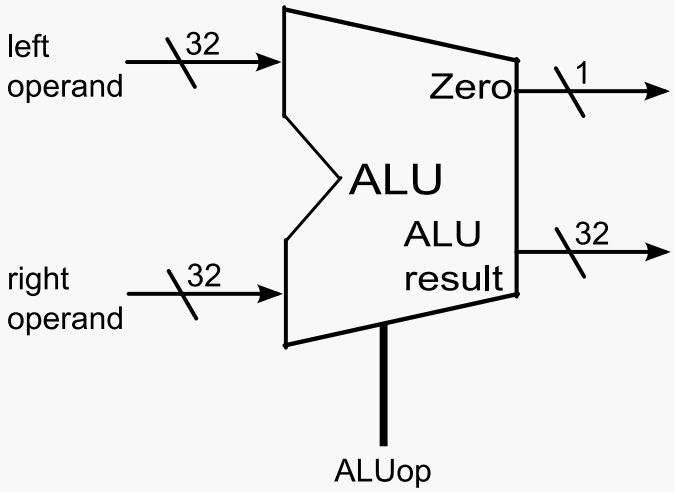


***QTP: why?**

ALU

The arithmetic-logic unit implements the fundamental computations that can be performed on data values.

Remember that all these take two operands.



The ALU will have the following interface:

- input: two 32-bit operands to be acted on
- output: one 32-bit result computed from the operands
- output: one 1-bit signal indicating whether the last result was equal to zero*
- control: a multi-bit signal, selecting which of the supported operations the ALU will perform on the operands

***QTP: why?**

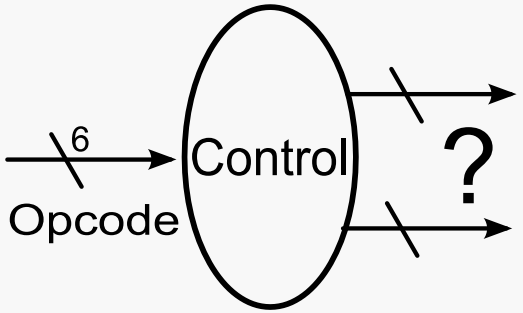
Control Unit

We've seen that the memory unit, register file and ALU all require control signals to manage their operations.

So, there must be an element in the datapath that identifies the current instruction and sets each of those control signals appropriately.

The interface will include:

- input: 6-bit opcode from current instruction
- output: currently unknown number of 1-bit signals to other elements



Multiplexors

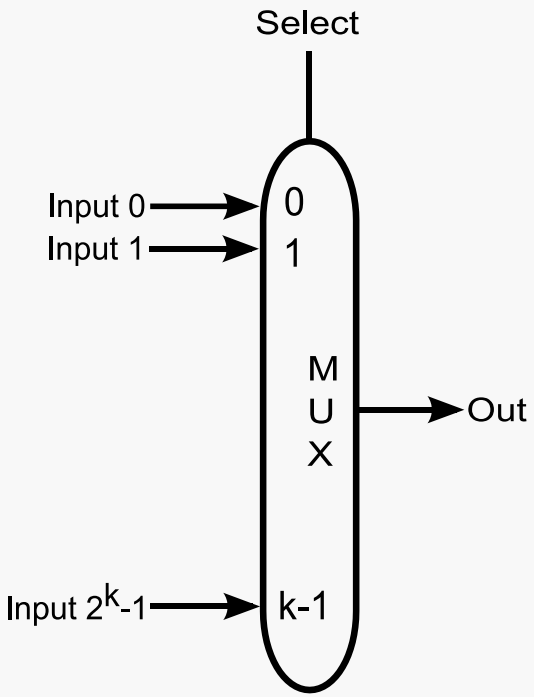
A multiplexor takes 2^k data inputs and k selector bits that determine which one of those 2^k data inputs will be passed through:

The k selector bits let us form the integers 0 through 2^k-1 in base-2,

which lets us specify any one of the 2^k inputs.

The data inputs can be of any width.

A multiplexor is analogous to a multi-way switch statement in C, and lets us implement a selection mechanism whenever one is needed.



Sign-Extension

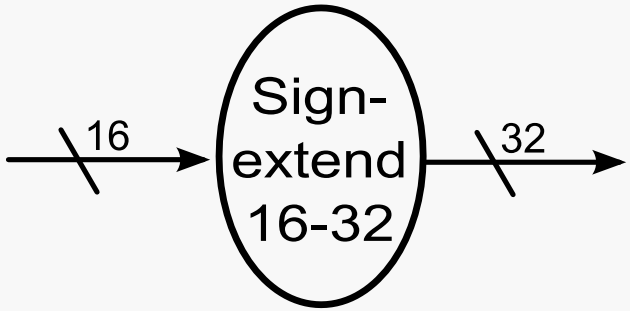
The I-type MIPS instructions contain a 16-bit immediate that's used in the execution of those instructions.

No matter which of these instructions is being executed, that 16-bit value must be added to a 32-bit value in order to compute an address.

Hardware adders require their operands to be the same width, so we need an element that can take a 16-bit value and widen it to 32 bits.

Moreover, this must preserve the sign of the 16-bit value, so what's called for is an element that performs *sign-extension*.

$d_{15}d_{14}d_{13}d_{12}\ldots d_3d_2d_1d_0$ → $d_{15}d_{15}\ldots d_{15}d_{15}d_{15}d_{15}d_{15}d_{14}d_{13}d_{12}\ldots d_3d_2d_1d_0$



Shifter

When executing the branch-on-equal instruction, the offset used to calculate the branch target address must be multiplied by 4.

Of course, that should be performed by applying a left-shift of 2 bits.

The ALU could do this, but we need the ALU to do the comparison of the register operands when a `beq` is being executed.

So, we need a datapath element that performs this left shift operation:

$d_{31}d_{30}d_{29}d_{28}\ldots d_3d_2d_1d_0$ → $d_{29}d_{28}d_{27}d_{26}\ldots d_3d_2d_1d_000$