Project 2: Boolean Arithmetic

Background

The centerpiece of the computer's architecture is the CPU, or Central Processing Unit, and the centerpiece of the CPU is the ALU, or Arithmetic-Logic Unit. In this project you will gradually build a set of chips, culminating in the construction of the ALU chip of the Hack computer. All the chips built in this project are standard, except for the ALU itself, which differs from one computer architecture to another.

Objective

Build all the chips described in Chapter 2 (see list below), leading up to an Arithmetic Logic Unit - the Hack computer's ALU. The only building blocks that you can use are the chips described in chapter 1 and the chips that you will gradually build in this project.

Chips

Chip Name	Description	Test Scripts	Compare File
HalfAdder	Half Adder	HalfAdder.tst	HalfAdder.cmp
FullAdder	Full Adder	FullAdder.tst	FullAdder.cmp
Add16	16-bit Adder	Add16.tst	Add16.cmp
Inc16	16-bit incrementer	Inc16.tst	Inc16.cmp
ALU	Arithmetic Logic Unit (without handling of status outputs)	ALU-nostat.tst	ALU-nostat.cmp
ALU	Arithmetic Logic Unit (complete)	ALU.tst	ALU.cmp

Contract

When loaded into the supplied Hardware Simulator, your chip design (modified .hdl program), tested on the supplied .tst script, should produce the outputs listed in the supplied .cmp file. If that is not the case, the simulator will let you know.

Resources

See Chapter 2, the HDL Guide (except for A2.4), and the Hack Chip Set.

For each chip, we supply a skeletal .hdl file with a missing implementation part. In addition, for each chip we supply a .tst script that instructs the hardware simulator how to test it, and a .cmp ("compare file") containing the correct output that this test should generate. Your job is to complete and test the supplied skeletal .hdl files.

The tools that you need for this project are the supplied hardware simulator and the files listed above. If you've downloaded the Nand2Tstris Software Suite, these files are stored in your nand2tetris/projects/02 folder.

Tips

Use built-in chips: Your HDL programs will most likely include chip parts that you've built in project 1. As a rule, though, we recommend using the built-in versions of these chips instead. The use of built-in chips ensures correct, efficient, and predictable simulation. There is a simple way to accomplish this convention: make sure that your project directory includes only the .hdl files of the chips developed in the current project.

Implementation order: We recommend building the chips in the order in which they appear in Chapter 2. However, since the simulator features built-in versions of these chips, you can use chip-parts without first building them: the simulator will automatically use their built-in implementations.

The Hack ALU produces two kinds of outputs: a "main" 16-bit output resulting from operating on the two 16-bit inputs, and two 1-bit "status outputs" named 'zr' and 'ng'. We recommend building this functionality in two stages. In stage one, implement an ALU that computes and outputs the 16-bit output only, ignoring the 'zr' and 'ng' status outputs. Once you get this implementation right (that is, once your ALU.hdl code passes the ALU-nostat test), extend your code to handle the two status outputs as well. This way, any problems detected by ALU.tst can be attributed to the incremental code that you've added in stage two. We thank Mark Armbrust for proposing this staged implementation plan, and for supplying the test files to support it.