Project 2

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

Objective

Build the following chips:

HalfAdder

FullAdder

Add16

Inc16

ALU

The only building blocks that you can use are some of the chips listed in Project 1, and the chips that you will gradually build on top of them in this project.

Files (Same guidelines as in Project 1)

For each chip Xxx, (we use the terms *chip* and *gate* interchangeably) we provide a skeletal Xxx.hdl program, sometimes called a *stub file* with a missing implementation part. In addition, for each chip we provide an Xxx.tst script that tells the hardware simulator how to test it, along with an Xxx.cmp compare file that lists the correct output that the supplied test is expected to generate. All these files are available in your nand2tetris/projects/2 folder. Your job is to complete and test all the Xxx.hdl files in this folder. These files can be written and edited using any plain text editor.

Contract: When loaded into the hardware simulator, your chip design (modified .hdl program), tested on the supplied .tst file, should produce the outputs listed in the supplied .cmp file. If the actual outputs generated by the simulator disagree with the desired outputs, the simulator will report error messages.

Builtin chips

The chips that you will build in this project use, as chip-parts, some of the chips described in chapter 1. Even if you've built these lower-level chips successfully in HDL, we recommend using their builtin versions instead. As best-practice advice pertaining to all the hardware projects in Nand to Tetris, always prefer using builtin chip-parts instead of their HDL implementations. The builtin chips are guaranteed to operate to specification and are designed to speed up the operation of the hardware simulator.

There is a very simple way to follow this best-practice advice: Don't add to the current project folder, nand2tetris/projects/2, any .hdl file from project 1. Whenever the hardware simulator will encounter in your HDL code a reference to a chip-part from project 1, say And16, it will check

whether there is an And16.hdl file in the current folder. Failing to find it, the hardware simulator will resort by default to using the builtin version of this chip, which is exactly what we want.

Implementation plan

The Hack ALU computes a 16-bit value named out. In addition, it computes two 1-bit "status outputs" called zr and ng. We suggest building the ALU in two stages. First, implement a basic ALU that computes out, ignoring the zr and ng outputs. Then implement a final version that computes out as well as zr and ng. We provide two sets of files for supporting this staged development: ALU-basic.tst and ALU-basic.cmp are given for testing the basic version; ALU.tst and ALU.cmp are given for testing the final version.

References (same as in project 1)

HDL Guide

Chips Set API

Hardware Simulator: Intro

Building and Testing Chips

Script-Based Chip Simulation

Hardware Simulator Tutorial (click *slideshow*)

Consult each resource as needed: There is no need to go through the entire resource.

Implementation Tips (same as in project 1)

Each gate can be implemented in more than one way. The simpler the implementation, the better. As a general rule, strive to use as few chip-parts as possible.

Although each chip can be implemented directly from Nand gates only, use composite gates that were already implemented. See the previous tip.

There is no need to build "helper chips" of your own design. Your HDL programs should use only the chips listed above, and in project 1.

Implement the chips in the order in which they are listed above. If, for some reason, you don't complete the HDL implementation of some chip, you can still use it as a chip-part in other HDL programs. Simply rename the chip file, or remove it from the folder. This will force the simulator to use the chip's builtin implementation.