Workshop 02 - Adders

Workshop 02 - More Combinatorial Hardware

The purpose of this workshop is to show you how an ALU could be built in hardware. This is based on **project 02** **(https://www.nand2tetris.org/project02)** from the Nand2Tetris course. In this case we will use builtin (working) versions of the chips you have already seen.

In this workshop you will be producing HDL code. For the HDL you may find t**he HDL Survival Guide** **(http://www.nand2tetris.org/software/HDL%20Survival%20Guide.html)** useful.

We have created a **Nand2Tetris Project 02 (https://myuni.adelaide.edu.au/courses/54311/pages/nand2tetris-project-02)** assignment in the **Web Submission System** **(https://cs.adelaide.edu.au/services/websubmission)** so that you can attempt this project independently from this workshop and be awarded up to 5 additional participation marks.

Participation Marks

Up to 5 participation marks are available for every workshop, 2 for preparation, 1 for attendance and 2 for completing an activity, subject to the conditions described on the **Participation Marks (https://myuni.adelaide.edu.au/courses/54311/pages/participation-marks)** page.

Attendance

One participation mark will be awarded if your workshop attendance is recorded and you make a submission to the Workshop 02 assignment in the Web Submission System by **Friday 11.55pm** of week 2. **Do not** leave a workshop until you have checked your attendance mark.

If you are using a CAT suite computer running Linux in a timetabled workshop, your presence should be automatically recorded when you visit our **practical marker** **(https://cs.adelaide.edu.au/services/pracmarker/)**. If your attendance mark is not displayed when you visit our **practical marker** **(https://cs.adelaide.edu.au/services/pracmarker/)**, your presence will need to be manually recorded. **Do not**

click on the "flag me for marking" button until a supervisor is standing next to you and is ready to record your presence. The flag is only to speed up the data entry and is only visible for 30 seconds.

Workshop 02 Background Reading

Review workshop 01 and read the second chapter of the textbook. This textbook chapter includes a description of how binary arithmetic can be performed and gives details of the ALU that you will be constructing in this workshop. We have also provided an additional description of **Why the ALU Works (https://myuni.adelaide.edu.au/courses/54311/pages/workshop-02-why-the-alu-works)** , that describes the controls and how to interpret their effects on arithmetic and logical operations. You must complete this background reading before attending your workshop.

You will need to write HDL code in the workshop so you should also read (and practice) sections I, II, and III of the tutorial on the hardware simulator and the Hardware Description Language (HDL). These can currently be found at the end of the Software page of the nand2Tetris website:

https://www.nand2tetris.org/software (https://www.nand2tetris.org/software)

Note that page 26 contains some important notes about how the simulator searches for hardware definitions.

Pencil and Paper

You will find it much easier to understand what is happening if you bring some paper and something to write with. Drawing pictures of how gates fit together and truth tables of what inputs produce what outputs can be very helpful.

Workshop 02 Preparation Task

Two participation marks will be awarded for completion of this preparation activity if it is completed at least 10 minutes before the first timetabled workshop of the week, see **Participation Marks (https://myuni.adelaide.edu.au/courses/54311/pages/participation-marks)** for details.

Note: in example commands % is the shell's prompt, it is not part of the command.

Note: this workshop assumes that you have already created directories for every assignment, workshop project and exam in your svn repository, as described on the **Startup Files for Workshops and Assignments**

(https://myuni.adelaide.edu.au/courses/54311/pages/startup-files-for-workshops-and-assignments) page.

Note: the web submission system will record 0 marks for completing this activity, the 2 marks are awarded later if and only if your attendance is recorded.

1. If required, checkout a working copy of the workshop02 directory from your svn repository.
2. Change directory to the working copy of the workshop02 directory.
3. Copy the newest zip file attached below into the updates sub-directory and add it to svn.
4. Run the following command to place the workshop's startup files in the correct locations:

```
% make install
```

5. Add the .hdl files to your svn repository:

```
% svn add --depth=empty ALU
% svn add *.hdl ALU/*.hdl
% svn commit -m "Workshop 02 Startup Files"
```

6. Goto the Web Submission System and make a submission to the Workshop 02 assignment. A successful submission that passes the preparation tests will complete the preparation activity.

Workshop 02 Activity

Two participation marks will be awarded for completing the following activity. This need not be completed during the workshop but no participation marks will be awarded if the activity is not completed by **Friday 11.55pm** of week 2.

Adders

The first steps focus on building half and full adder chips. These steps are optional, you do not need to complete them before you move onto the ALU.

1. Change directory to the working copy of the workshop02 directory.
2. Write HDL code for a **HalfAdder** chip and test it:

```
% HardwareSimulator.sh HalfAdder.tst
```

3. Write HDL code for a **FullAdder** chip and test it:

```
% HardwareSimulator.sh FullAdder.tst
```

4. Commit the changes to your svn repository:

```
% svn commit -m workshop02-activity
```

5. Goto the Web Submission System and make a submission to the Workshop 02 assignment. A successful submission that passes the HalfAdder and FullAdder tests or any of the ALU tests will complete the workshop activity.

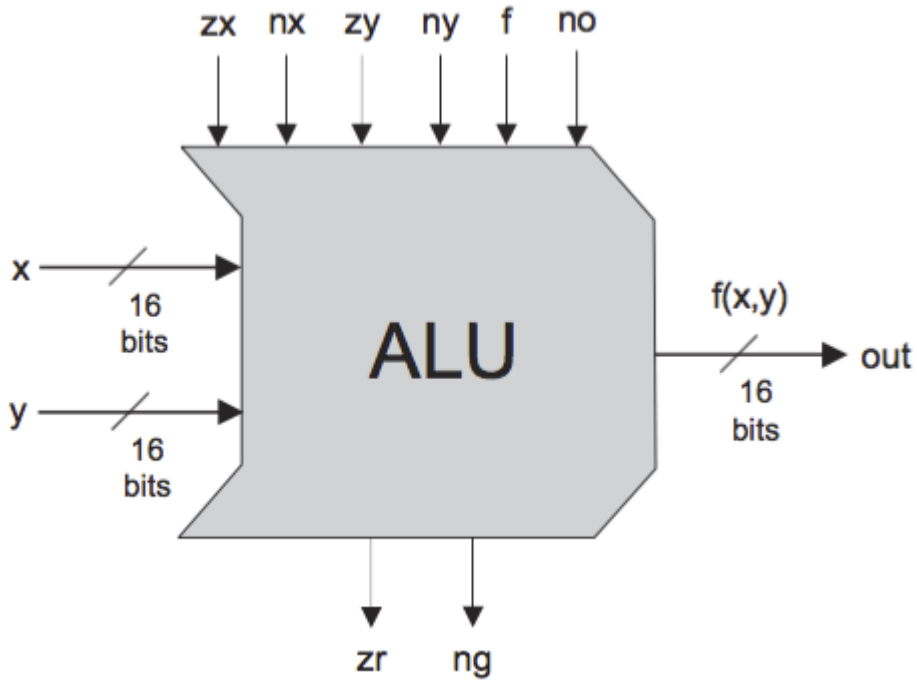
ALU

The next steps just focus on constructing the ALU using builtin versions of all of the components required. We placed copies of the files we want to work with in the sub-directory ALU. These will be the only chips in the new directory so the Hardware Simulator will automatically use working builtin versions of any other chips we need.

The remaining steps require you to edit the ***ALU.hdl*** file and write HDL code to generate the correct 16-bit output of an **ALU** chip.

You should use the And16, Add16, Not16 and Mux16 chips.

The ALU takes two 16 bit inputs, **x** and **y**, and produces a 16 bit output, **out**. In addition the **zr** output is 0 if the 16 bit output is 0 or 1 otherwise. The **ng** output is 1 if the 16 bit output is a negative number or 0 otherwise. There are six control inputs **zx**, **nx**, **zy**, **ny**, **f** and **no**, that control how **x** and **y** are transformed to produce **out**. The file **ALU.hdl** contains a description of what effect each control has on the output.



1. Change directory to the working copy of the workshop02 directory.
2. Start by running the ALU tests to see what happens. There are two test scripts, ***ALU/ALU-nostat.tst*** and ***ALU/ALU.tst***. The ***ALU/ALU-nostat.tst*** script runs the same tests as ***ALU/ALU.tst*** but does not check the **zr** and **ng** outputs:

```
% HardwareSimulator.sh ALU/ALU-nostat.tst
Comparison failure at line 3
% HardwareSimulator.sh ALU/ALU.tst
Comparison failure at line 2
```

3. Start by adding the **x** and **y** inputs together and connecting the result to **out**. To do this you must use the Add16 chip.
4. Now that you can add two numbers together, use the And16 chip to produce a bitwise and of **x** and **y** and then use a Mux16 chip to connect either the Add16 or And16 to **out**. The control input **f** should be used as the selector input to the Mux16 chip. If **f** is a 1 the result of the Add16 should connected to **out** otherwise the result of the And16 should be connected to **out**.
5. In the previous step we used the Mux16 to choose between two 16 bit inputs. In this step we want to implement a choice between the output of the previous step or the bitwise not of the previous step. To achieve this add a new Mux16 which has its output

connected to **out**. One of its inputs should be the output from the previous step. The other input is produced by connecting the output of the previous step to a Not16 chip which flips all the bits. If **no** is 1 the new Mux16 chooses the output from the Not16 gate.

6. We can repeat the previous step to implement the bitwise not controls **nx** and **ny**. In each case we feed the **x** or **y** to a new Mux16 chip and also to a Not16 chip then use the **nx** or **ny** control as a selector.
7. The final steps are to implement the **zx** and **zy** controls. These controls should be applied before the **nx** and **ny** controls. In these cases we need to produce a 0 or the original **x** or **y**. To obtain a 0 we use the keyword **false** as one the inputs to a new Mux16 chip.

   At this stage you should be able to pass all of the tests run by ***ALU/ALU-nostat.tst***.

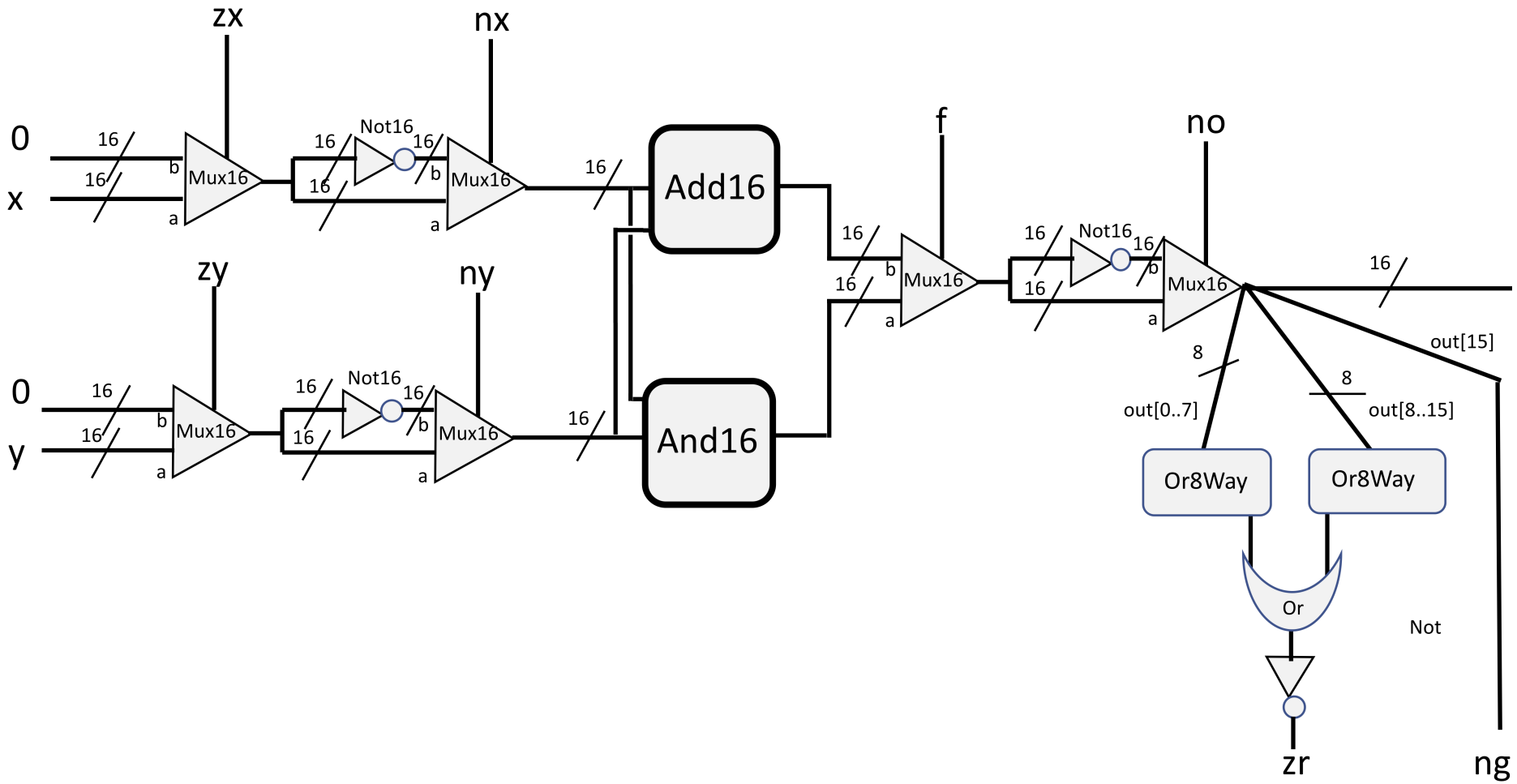
```
% HardwareSimulator.sh ALU/ALU-nostat.tst
End of script - Comparison ended successfully
```

8. Write HDL code to generate the correct status outputs, **zr** and **ng**, for an ALU.

   You will need to use the following additional chips to achieve this, Not, Or and Or8Way. The Or8Way chip allows you to do a bitwise or operation across 8 inputs at once. It produces a 0 if all of its inputs are 0.
   Note: you need to direct the output of the final Mux16 to 4 different wires of different widths. You can do this by replacing out=out with:

```
out=out, out[15]=ng, out[0..7]=or8a ,out[8..15]=or8b
```

The final ALU should look something like this:



At this stage you should be able to pass all of the tests run by ***ALU/ALU.tst***.

```
% HardwareSimulator.sh ALU/ALU.tst
End of script - Comparison ended successfully
```

9. Commit the changes to your svn repository, this can be done after any earlier step:

```
% svn commit -m workshop02-activity
```

10. Goto the Web Submission System and make a submission to the Workshop 02 assignment, this can be done after any earlier commit. A successful submission that passes the HalfAdder and FullAdder tests or any of the ALU tests will complete the workshop activity.

** **Important** ** a key thing to note about the ALU that you have constructed - it is always performing an **add** operation and an **and** operation even if the results of one or other operation is not required.

Startup Files

The newest of the following zip file(s) must be placed in the updates sub-directory and added to svn. When make is run, the newest zip file in the updates directory is used to update the startup files. Any files you are required to edit will not be updated but, a copy of the latest version of those files will be placed in the sub-directory originals.

- **workshop02-20200806-105038.zip (https://myuni.adelaide.edu.au/courses/54311/files/7198787/download?wrap=1)**
- workshop02-20200724-193244.zip