Assignment 02 – ALU

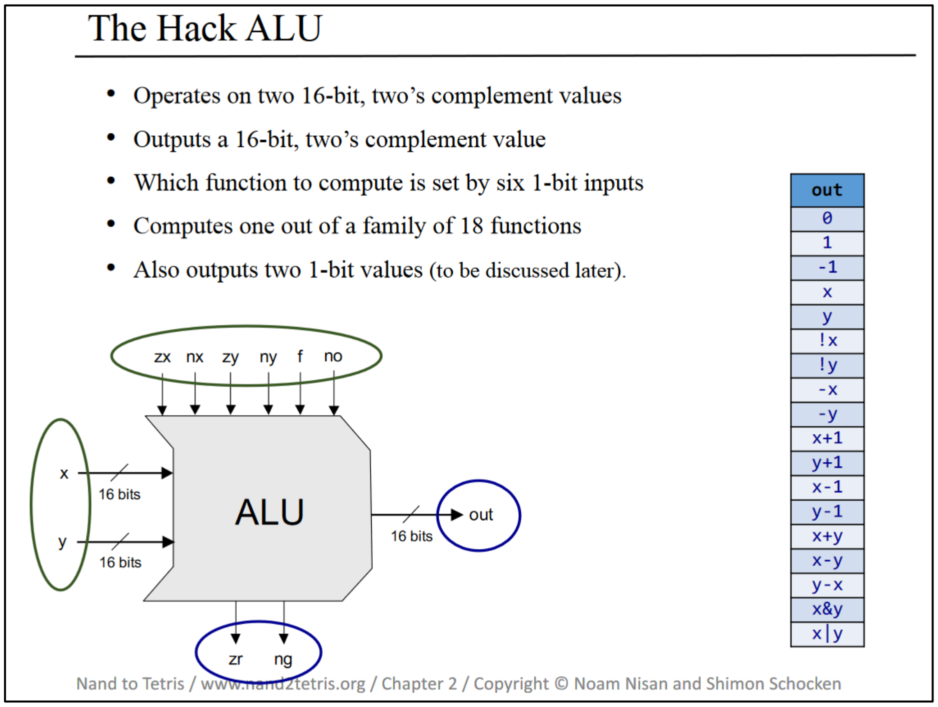
EDB HDL

ANM, v01

Introduction

The ALU is the arithmetic logic unit of the CPU. The IO is depicted in Figure 1:

- Two 16bit inputs **x** and **y**
- Six control bits: **zx**, **nx**, **zy**, **ny**, **f**, **no**
- 16bit output **out**
- Two output flags **zr** and **ng**



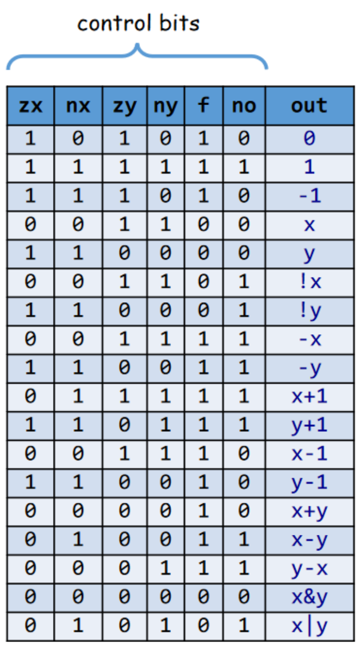
The Hack ALU

- Operates on two 16-bit, two's complement values
- Outputs a 16-bit, two's complement value
- Which function to compute is set by six 1-bit inputs
- Computes one out of a family of 18 functions
- Also outputs two 1-bit values (to be discussed later).

out
0
1
-1
x
y
!x
!y
-x
-y
x+1
y+1
x-1
y-1
x+y
x-y
y-x
x&y
x|y

Figure 1: ALU description. Source: https://www.nand2tetris.org/

The functions that shall be implemented in the ALU are listed in Figure 2.


control bits

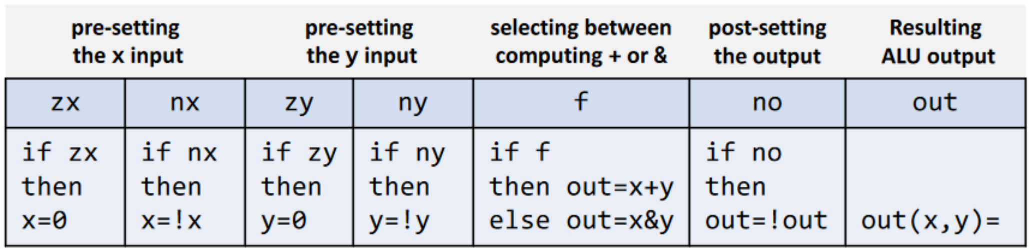
zx	nx	zy	ny	f	no	out
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

Figure 2: Arithmetic and logic operations to be implemented in the ALU. Source: https://www.nand2tetris.org/

Task 01 – Implementation of the ALU

To keep the design simple and easy to understand the processing of the inputs **x** and **y** the ALU shall be structured in the following way:

- Pre-setting of **x** and **y** as determined by **zx**, **nx**, **zy**, **ny**
- Operation as set by **f**
- Post-setting of the output as selected by **no**
- Assign the output to **out**.
- Set the output flags **zr** and **ng**
    - If (out==0) then zr = 1'b1 else zr = 1'b0
    - If (out < 0) then ng = 1'b1 else ng = 1'b0



pre-setting the x input		**pre-setting the y input**		**selecting between computing + or &**	**post-setting the output**	**Resulting ALU output**
zx	nx	zy	ny	f	no	out
if zx then x=0	if nx then x=!x	if zy then y=0	if ny then y=!y	if f then out=x+y else out=x&y	if no then out=!out	out(x,y)=

Figure 3: Operations as set by the control bits. Source: https://www.nand2tetris.org/

- Create the folder structure
    - ./alu/sim
    - ./alu/src
- [1cp] Create a SystemVerilog module "alu.sv" in the ./alu/src folder that implements the specified ALU.

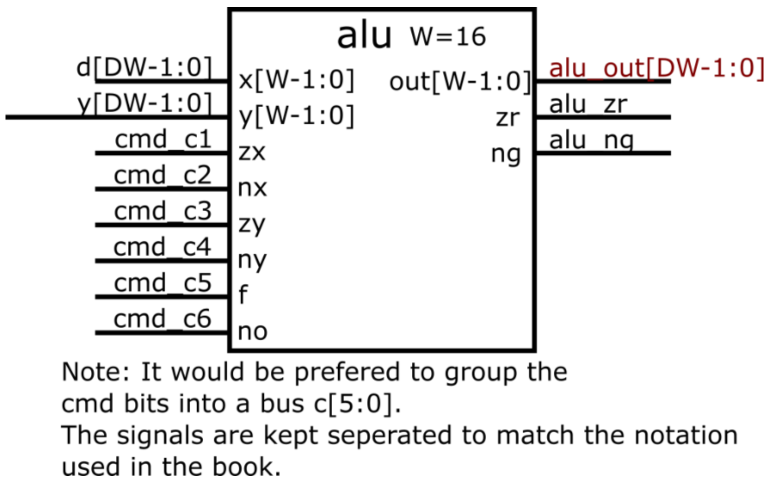


Figure 4: Block diagram of the ALU.

- Requirements
  - See description of the ALU operations above.
- [2cp] Create a SystemVerilog test bench “tb_alu.sv” in the ./alu/sim folder
  - Perform the following arithmetic or logic operation in the TB. For the control bit settings please refer to Figure 2):
    - out = 1 + 2
    - out = 100-50
    - out = 50-100
    - out = 0
    - out = 0xaa & 0x55
    - out = 0xaa | 0x55
    - out = -1
    - out = !255
    - out = -255
  - The test bench shall allow to check the correctness of the implemented logic in the wave window of the simulator.
  - All results (i.e. the outputs **out**, **zr** and **ng**) are self-checked by the testbench. The number of errors is recorded. At the end of the test bench a Pass/Fail message is shown.
  - Explain the control bits for the operation “out = 0xaa | 0x55”.
- [1cp] Create a TCL script file “sim_tb_alu.tcl” that controls the compile and simulation process. A wave window is opened and all relevant signals are displayed.
- [1cp] Create a short summary report “doc_alu.pdf” that shows the simulation result.
  - Show that your design fulfils the specification (verification).
  - Discuss why you chose your implementation method (advantages, disadvantages).

Appendix