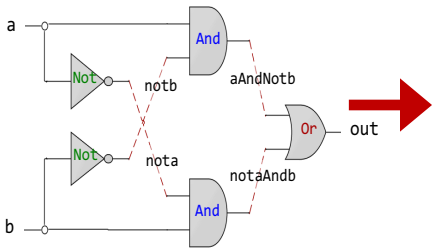
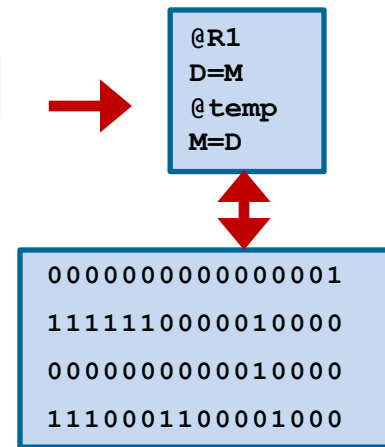
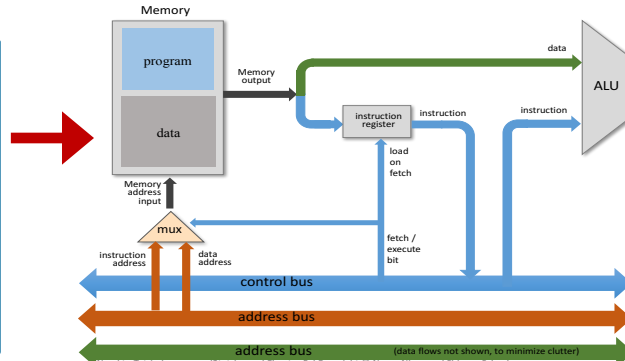




Digital Logic Design



```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
    Not(in=a, out=nota);
    Not(in=b, out=notb);
    And(a=nota, b=b, out=w1);
    And(a=a, b=notb, out=w2);
    Or(a=w1, b=w2, out=out);
}
```



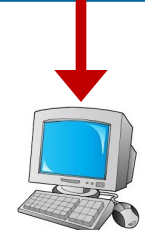
Lecture # 13-14

Design of ALU

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
  msg: db "Learning is fun with Arif", 0Ah, 0h
  len_msg: equ $ - msg
SECTION .text
main:
  mov rax,1
  mov rdi,1
  mov rsi,msg
  mov rdx,len_msg
  syscall
  mov rax,60
  mov rdi,0
  syscall
```

```
0: b8 01 00 00 00
5: bf 01 00 00 00
a: 48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```



Slides of first half of the course are adapted from:
<https://www.nand2tetris.org>
 Download s/w tools required for first half of the course from the following link:
<https://drive.google.com/file/d/0B9c0BdDz6XpZUh3X2dPR1o0MUE/view>

Instructor: Muhammad Arif Butt, Ph.D.



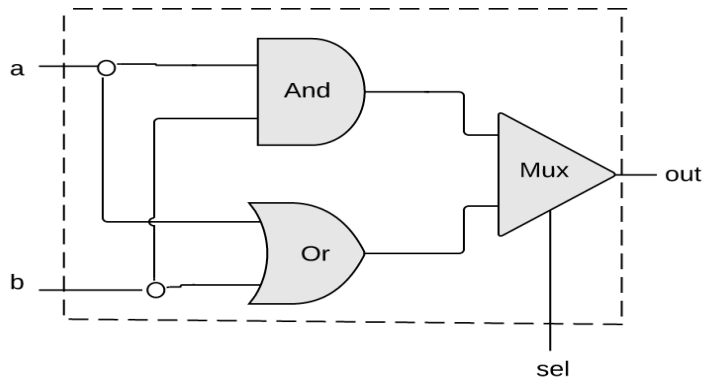
Today's Agenda

- Class Quiz
- Components of a Computer System
- Design of ALU
- The Hack ALU
- The Hack ALU Operations
- Design of Hack ALU
- HDL of Hack ALU
- Verifying the ALU chip on H/W Simulator





Arithmetic Logic Unit



a	b	sel	out
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

When sel==0
Operation = AND

When sel==1
Operation = OR

BitLU.hdl

```
CHIP BitLU {
  IN a, b, sel;
  OUT out;
  PARTS:
    And(a=a, b=b, out=andOut);
    Or(a=a, b=b, out=orOut);
    Mux (a=andOut, b=orOut, sel=sel, out=out);
}
```

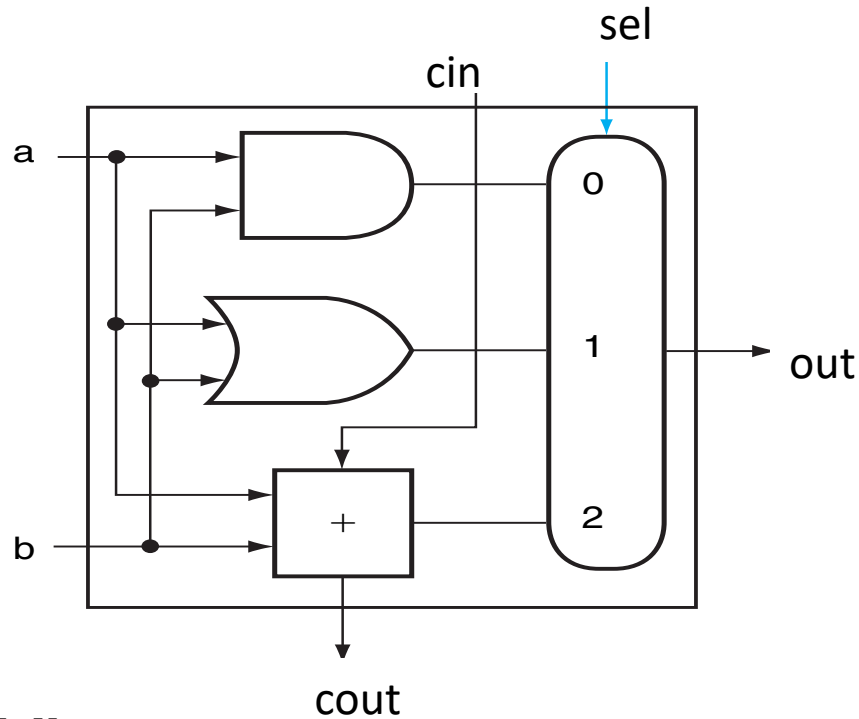


Single Bit Logic Unit Demo





Class Quiz: 1-Bit ALU



BitALU.hdl

```
CHIP BitALU {  
    IN a, b, cin, sel[2];  
    OUT out, cout;  
    PARTS:  
        //write your code here  
}
```



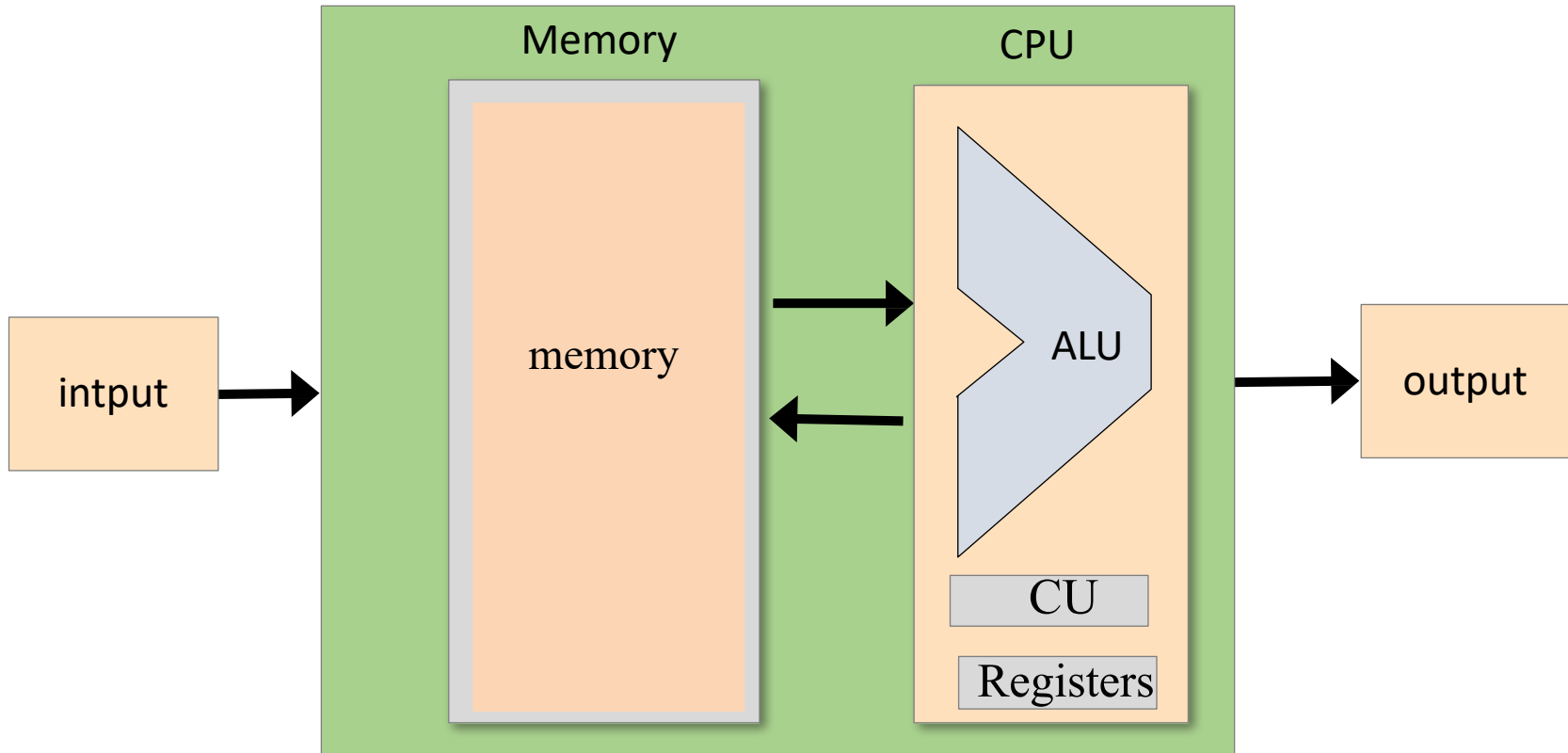
Single Bit ALU Demo





The Computer System

Computer System

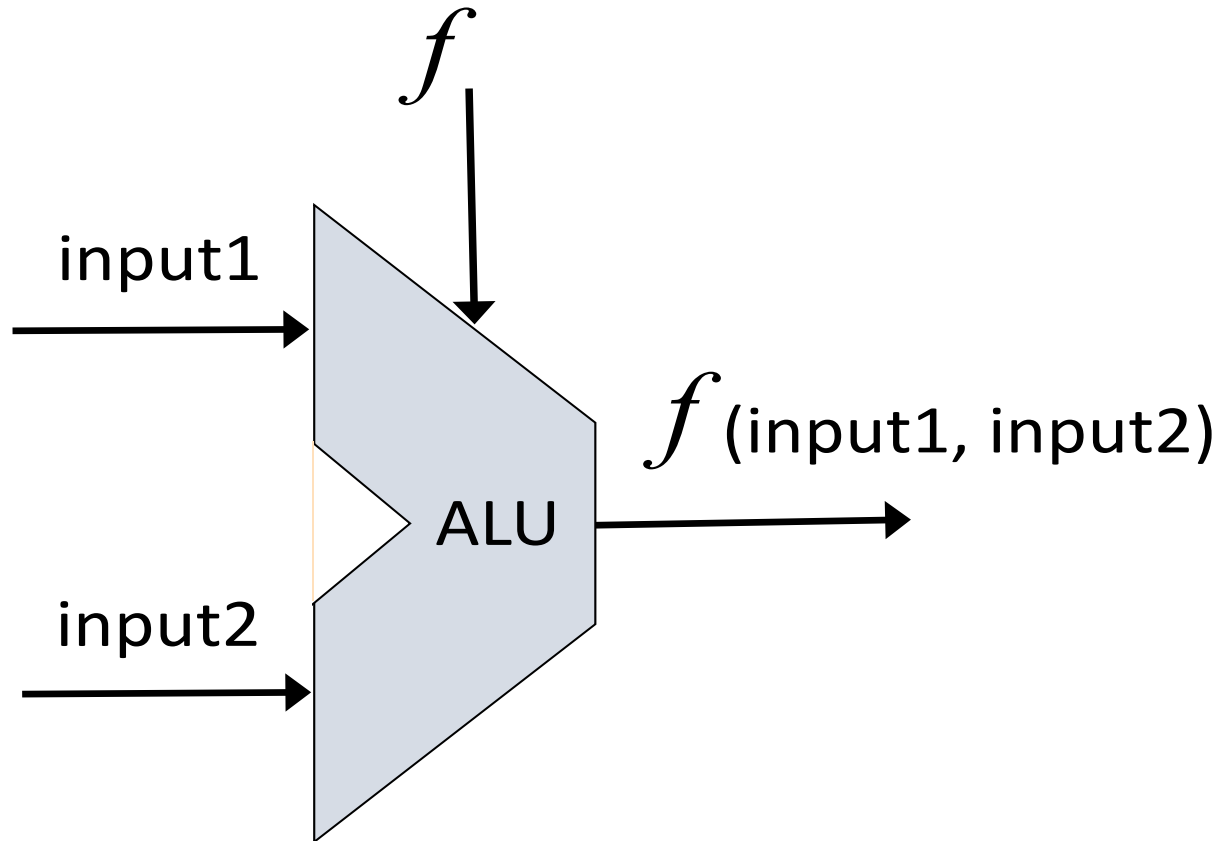




Design of Arithmetic Logic Unit

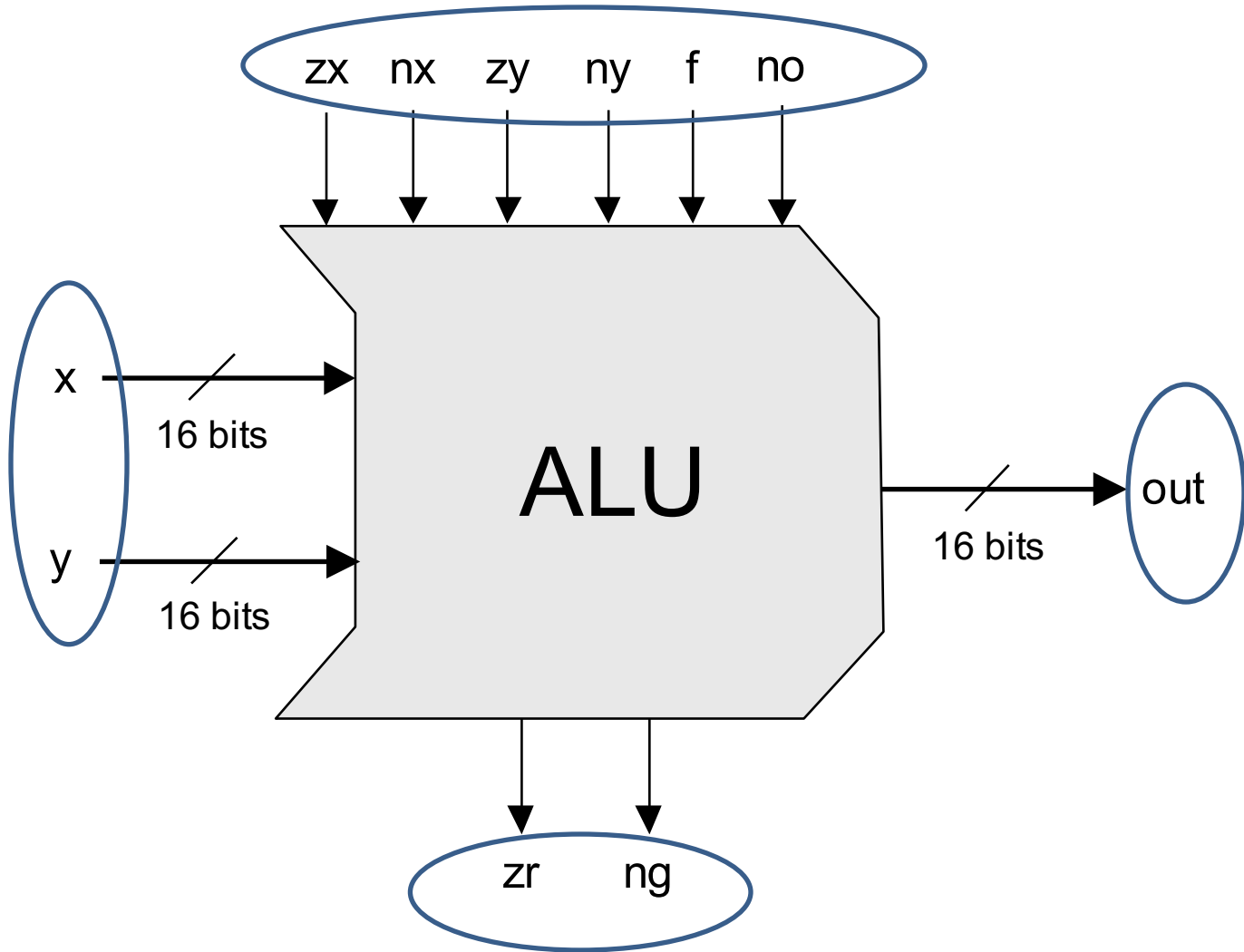


The Arithmetic Logical Unit





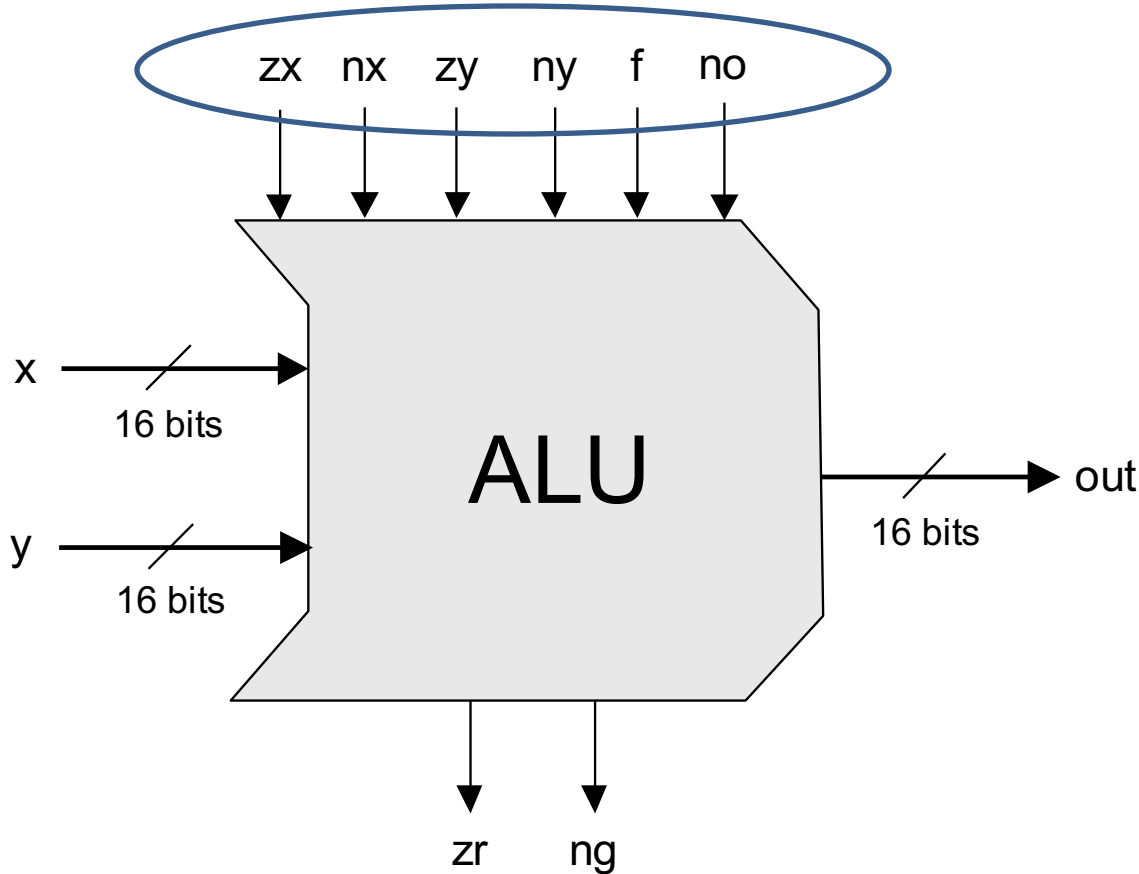
The Hack ALU



out
0
1
-1
x
y
$!x$
$!y$
$-x$
$-y$
$x+1$
$y+1$
$x-1$
$y-1$
$x+y$
$x-y$
$y-x$
$x\&y$
$x y$



The Hack ALU (cont...)

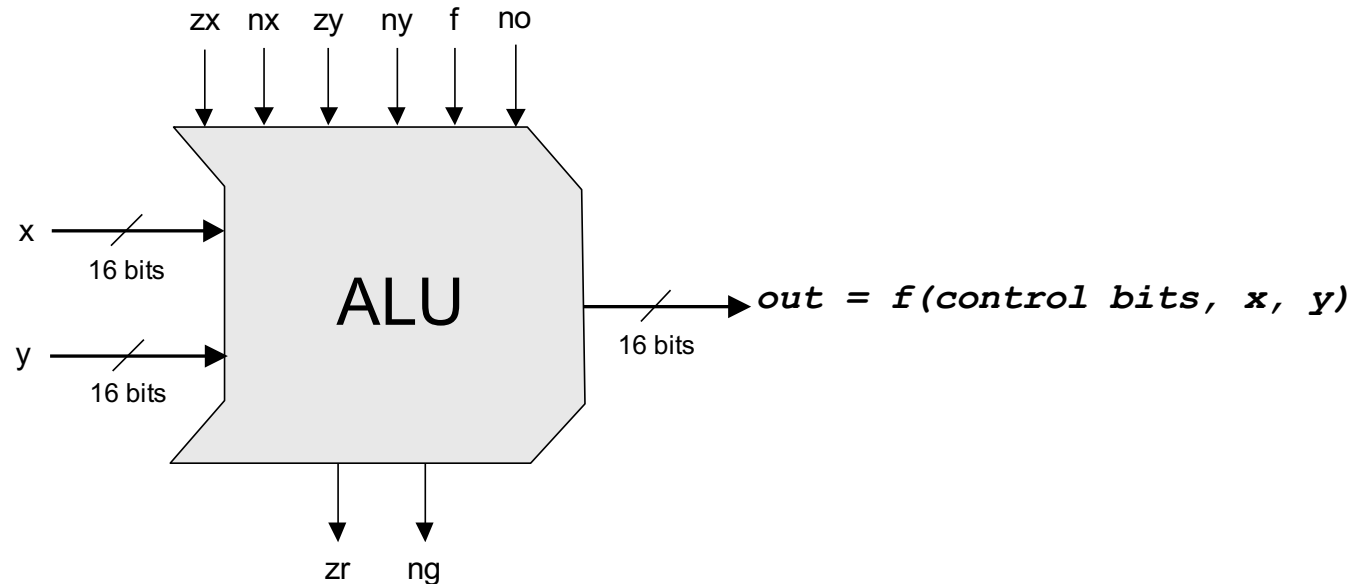


zx	nx	zy	ny	f	no	out
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	$!x$
1	1	0	0	0	1	$!y$
0	0	1	1	1	1	$-x$
1	1	0	0	1	1	$-y$
0	1	1	1	1	1	$x+1$
1	1	0	1	1	1	$y+1$
0	0	1	1	1	0	$x-1$
1	1	0	0	1	0	$y-1$
0	0	0	0	1	0	$x+y$
0	1	0	0	1	1	$x-y$
0	0	0	1	1	1	$y-x$
0	0	0	0	0	0	$x\&y$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



The Hack ALU Operation

Pre-setting the x input		Pre-setting the y input		Selecting op + or &	Post-setting o/p	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) = out





The Hack ALU Operation

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)=
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



How the ALU Perform

!X



The Hack ALU Operation: (!x)

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)=
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				!y
0	0	1				-x
1	1	0				-y
0	1	1				x+1
1	1	0				y+1
0	0	1				x-1
1	1	0				y-1
0	0	0				x+y
0	1	0				x-y
0	0	0				y-x
0	0	0				x&y
0	1	0				x y

Example: compute !x

```

x:      1 1 0 0
y:      1 0 1 1
Following Pre Setting
x:      1 1 0 0
y:      1 1 1 1
Computation and post setting
x&y:    1 1 0 0
!(x&y): 0 0 1 1

```



How the ALU Perform

$$y - X$$



The Hack ALU Operation: (y-x)

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)=
1	0	1	0	1	0	0
				1	1	1
				1	0	-1
				0	0	x
				0	0	y
				0	1	!x
				0	1	!y
				1	1	-x
				1	1	-y
				1	1	x+1
				1	1	y+1
				1	0	x-1
				1	0	y-1
				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

Example: compute y - x

x: 0 0 1 0 (2)

y: 0 1 1 1 (7)

Following pre-setting:

x: 0 0 1 0

y: 1 0 0 0

Computation and post-setting:

(x+y): 1 0 1 0

!(x+y): 0 1 0 1 (5)



How the ALU Perform

$$x - y$$



The Hack ALU Operation: (x-y)

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)=
1	0	1	0	1	0	0
0	1	1	0	1	1	1
0	1	0	0	1	0	-1
1	0	0	0	0	0	x
0	0	0	0	0	0	y
0	0	0	1	0	1	!x
0	0	0	1	0	1	!y
1	0	1	0	1	1	-x
1	0	1	1	1	1	-y
0	1	0	0	1	1	x+1
0	1	0	1	1	1	y+1
0	0	1	0	1	0	x-1
0	0	1	1	1	0	y-1
0	1	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

Example: compute x - y

x: 0 1 0 1 (5)

y: 0 0 1 1 (3)

Following pre-setting:

x: 1 0 1 0

y: 0 0 1 1

Computation and post-setting:

(x+y): 1 1 0 1

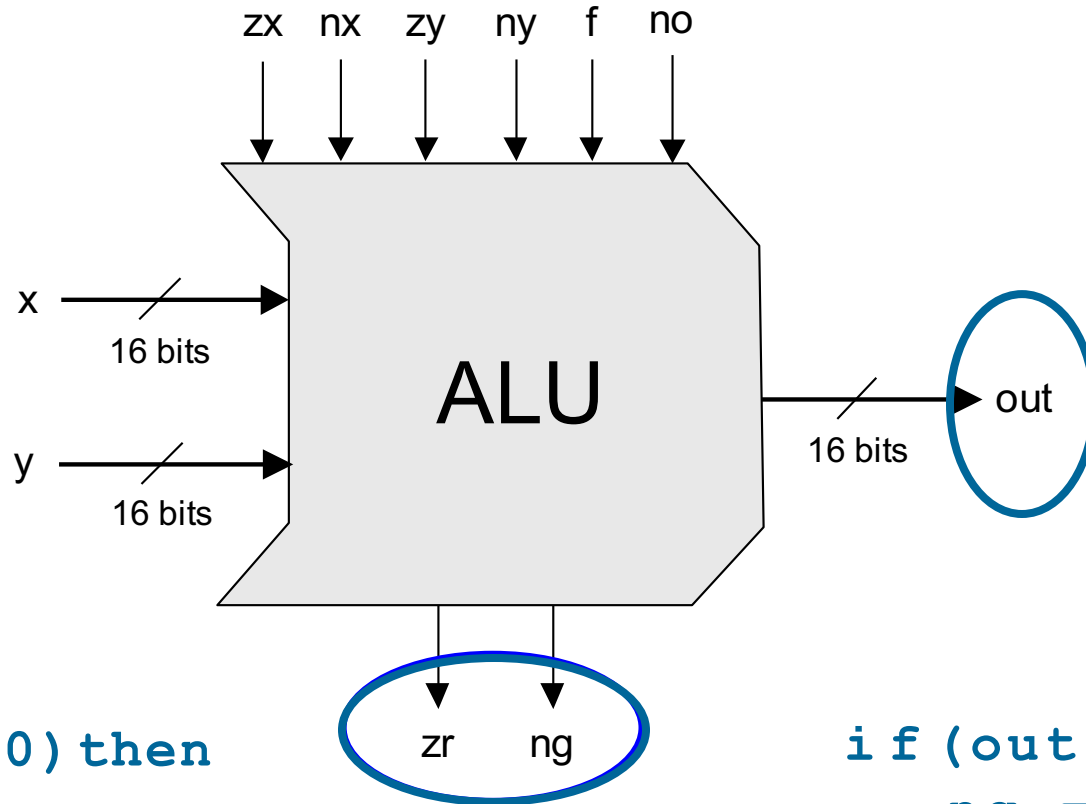
!(x+y): 0 0 1 0 (2)

0 1 0 0 1 1 1

x+y
x-y



The Hack ALU Output Control Bits



```
if (out == 0) then
    zr = 1
else
    zr = 0
```

```
if (out < 0) then
    ng = 1
else
    ng = 0
```

- These two control bits will come into play when we build the complete computer's architecture

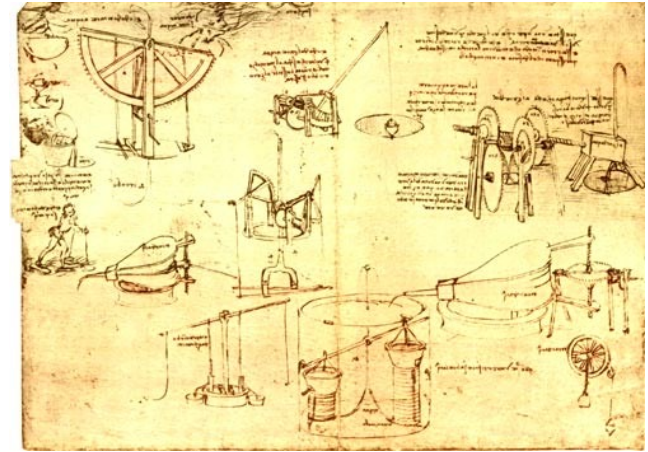


Perspective

The Hack ALU is:

- Simple
- Elegant
- To implement this ALU, you only need to know how to:
 - Set a 16-bit value to 0000000000000000
 - Set a 16-bit value to 1111111111111111
 - Negate a 16-bit value (bit-wise)
 - Compute plus or And on two 16-bit values

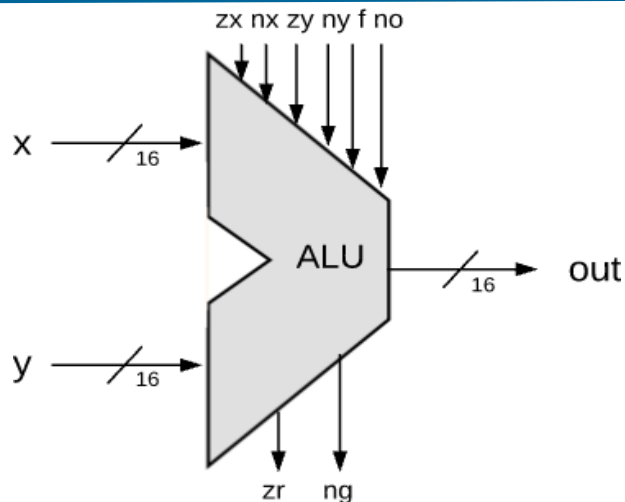
That's it!



“Simplicity is the ultimate sophistication.”
— Leonardo da Vinci



Writing HDL for Hack ALU - I

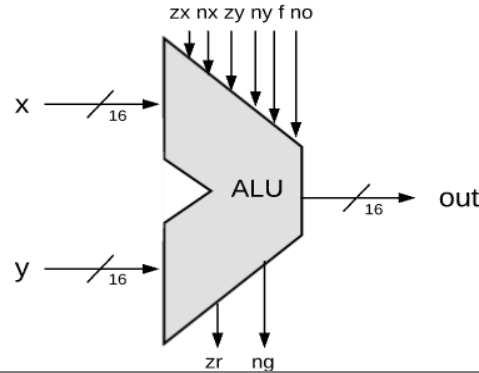


ALU.hdl

```
/**The ALU computes one of the following 18 functions:  
* x+y, x-y, y-x, 0, 1, -1, x, y, -x, -y, !x, !y,  
* x+1, y+1, x-1, y-1, x&y, x|y on two 16-bit inputs,  
* according to 6 input bits denoted zx,nx,zy,ny,f,no;  
* In addition, the ALU computes two 1-bit outputs:  
* if ALU output == 0, zr is set to 1; otherwise zr is set to 0;  
* if ALU output<0, ng is set to 1; otherwise ng is set to 0;  
*/
```



Writing HDL for Hack ALU - II



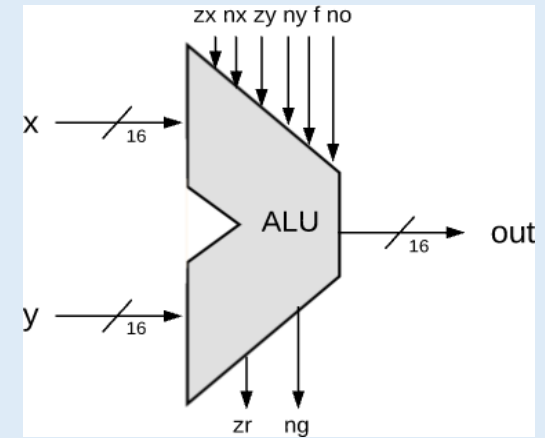
ALU.hdl

```
// Implementation:
// if (zx == 1) set x = 0           // zero the x input
// if (nx == 1) set x = !x         // negate the x input
// if (zy == 1) set y = 0           // zero the y input
// if (ny == 1) set y = !y         // negate the y input
// if (f == 1) set out = x + y     // 2's complement addition
// if (f == 0) set out = x & y     // bitwise and
// if (no == 1) set out = !out     // negate the out output
// if (out == 0) set zr = 1        // set zr output bit to 1
// if (out < 0) set ng = 1         // set ng output bit to 1
```



Writing HDL for Hack ALU - III

```
CHIP ALU {
  IN  x[16], y[16], zx, nx, zy, ny, f, no;
  OUT out[16], zr, ng;
  PARTS:
  //if (zx == 1) set x = 0
    Mux16(a=x, b=false, sel=zx, out=x1);
  //if (zy == 1) set y = 0
    Mux16(a=y, b=false, sel=zy, out=y1);
  //if (nx == 1) set x = !x
    Not16(in=x1, out=notx1);
    Mux16(a=x1, b=notx1, sel=nx, out=x2);
  //if (ny == 1) set y = !y
    Not16(in=y1, out=noty1);
    Mux16(a=y1, b=noty1, sel=ny, out=y2);
  // if (f == 1) set out = x + y else set out = x & y
  Add16(a=x2, b=y2, out=x2Plusy2);
  And16(a=x2, b=y2, out=x2Andy2);
  Mux16(a=x2Andy2, b=x2Plusy2, sel=f, out=xFuncy);
```





Writing HDL for Hack ALU - IV

```
// if (no == 1) set out = !out
Not16(in=xFuncy, out=notxFuncy);
Mux16(a=xFuncy, b=notxFuncy, sel=no, out=output);

// if (out < 0) set ng = 1
And16(a=output, b=true, out[15]=ng);

// if (out == 0) set zr = 1
And16(a=true, b=output, out[0..7]=outlast8);
And16(a=true, b=output, out[8..15]=outfirst8);
Or8Way(in=outlast8, out=Or8Wayoutlast8);
Or8Way(in=outfirst8, out=Or8Wayoutfirst8);
Or(a=Or8Wayoutlast8, b=Or8Wayoutfirst8, out=outputIsNotZero);
Not(in=outputIsNotZero, out=zr);

// out == output
And16(a=true, b=output, out=out); // out = output
}
```



ALU Demo





The Hack ALU In Action: Compute $y-x$

The screenshot shows a logic simulator interface with several components and annotations:

- Toolbar:** Contains various navigation and simulation controls. A calculator icon is circled in blue.
- ALU Component:** A component named "ALU" is selected. Its inputs and control bits are set to test values. A callout box points to the control bits, stating: "1. Set the ALU's inputs and control bits to some test values (000111 codes 'output y-x')".
- Output Display:** The ALU's output is shown as a 3-bit value: -10 , 0 , 1 . This value is circled in blue, with a callout box stating: "3. Inspect the ALU outputs".
- HDL Code:** A window shows the Verilog code for the ALU implementation. A callout box points to the code, stating: "Built-in ALU implementation". The code includes comments and logic for setting control bits based on the ALU operation.
- GUI Side-effects:** A callout box points to the ALU component, stating: "The built-in ALU implementation has some GUI side-effects".
- ALU GUI Diagram:** A diagram shows the ALU component with two inputs: "D Input" (value 30) and "M/A Input" (value 20). The output is "ALU output" (value -10). The component is labeled "M-D".



The Hack ALU In Action: Compute x and y

The screenshot shows the Hack ALU simulator interface. The top menu includes File, View, Run, and Help. The toolbar contains various control icons and settings for animation (Program flow, Bi..., Scr...), format, and view. The main window displays the chip name 'ALU' and the time '0'. The input pins table shows x[16] and y[16] with binary values, and control bits zx, nx, zy, ny, f, and no. The output pins table shows out[16], zr, and ng. The HDL code is visible at the bottom left, and the logic diagram at the bottom right shows the ALU block with D and M/A inputs and an output.

Input pins		Output pins	
Name	Value	Name	Value
x[16]	1110101110000110	out[16]	0000100000000100
y[16]	0001100001101101	zr	0
zx	0	ng	0
nx	0		
zy	0		
ny	0		
f	0		
no	0		

Set to binary I/O format

Inspect the ALU outputs

Set the ALU's inputs and control bits to some test values (000000 codes "compute x&y")

```
HDL
// This file is part of the material for the book:
// "The Elements of Computing Systems" by John V.
// MIT Press. Book site: www.idc.se.liu.se/books/
// File name: tools/builtIn/ALU.

/**
 * The ALU. Computes a pre-defined operation on two
 * 16-bit integers x and y. The operation is determined
 * by a set of 6 control bits: zx, nx, zy, ny, f, and no.
 * The ALU operation can be described as follows:
 *   if zx=1 set x = 0
 *   if nx=1 set x = !x
 *   if zy=1 set y = 0
 *   if ny=1 set y = !y
 */
```

ALU
D Input : -5242
M/A Input : 6253
ALU output : 2052



Things To Do

- Do lot of practice of different ALU operations using paper and pencil
- Carry out verification of your paper working and ensure that the ALU chip that we have designed today is working correctly. Use the built-in ALU chip as well as download the HDL of ALU chip from the course bitbucket repository:



https://github.com/arifpucit/COAL_VLecs

- Whenever there is a confusion, please refer to HDL survival guide available on

<http://www.arifbutt.me>

Coming to office hours does NOT mean you are academically week!