Arithmetic Logic Unit (ALU)

Introduction to Computer

Yung-Yu Chuang

with slides by Sedgewick & Wayne (introcs.cs.princeton.edu), Nisan & Schocken (www.nand2tetris.org) and Harris & Harris (DDCA)

Let's Make an Adder Circuit

Goal. x + y = z for 4-bit integers.

- We build 4-bit adder: 9 inputs, 4 outputs.
- Same idea scales to 128-bit adder.
- Key computer component.

```
     1  1  1  0
     2  4  8  7
+    3  5  7  9
     ----------
     6  0  6  6
```

Binary addition

Assuming a 4-bit system:

```
  0  0  0  1              1  1  1  1
     1  0  0  1              1  0  1  1
     0  1  0  1  +           0  1  1  1  +
  ------------            ------------
  0  1  1  1  0           1  0  0  1  0

   no overflow               overflow
```

- Algorithm: exactly the same as in decimal addition
- Overflow (MSB carry) has to be dealt with.

Representing negative numbers (4-bit system)

0	0000		
1	0001	1111	-1
2	0010	1110	-2
3	0011	1101	-3
4	0100	1100	-4
5	0101	1011	-5
6	0110	1010	-6
7	0111	1001	-7
		1000	-8

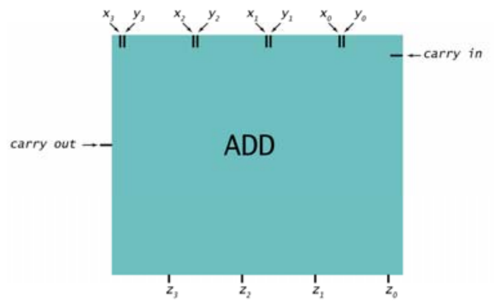
- The codes of all positive numbers begin with a "0"
- The codes of all negative numbers begin with a "1"
- To convert a number: leave all trailing 0's and first 1 intact, and flip all the remaining bits

Example: 2 - 5 = 2 + (-5) =

```
   0 0 1 0
 + 1 0 1 1
 ---------
   1 1 0 1    =  -3
```

Let's Make an Adder Circuit

Step 1. Represent input and output in binary.



	1	1	0	0
	0	0	1	0
+	0	1	1	1
	1	0	0	1

	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
	z_3	z_2	z_1	z_0

Let's Make an Adder Circuit

Goal. x + y = z for 4-bit integers.

c_{out}				c_{in}
	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
	z_3	z_2	z_1	z_0

Step 2. [first attempt]

- Build truth table.

4-Bit Adder Truth Table

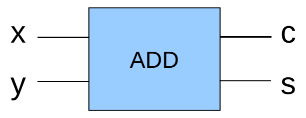
c_0	x_3	x_2	x_1	x_0	y_3	y_2	y_1	y_0	z_3	z_2	z_1	z_0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	1	1	0	0	1	1
0	0	0	0	0	0	1	0	0	0	1	0	0
.	.	.	.	.	.	.	.	.	.	.	.	.
1	1	1	1	1	1	1	1	1	1	1	1	1

2^{8+1} = 512 rows!

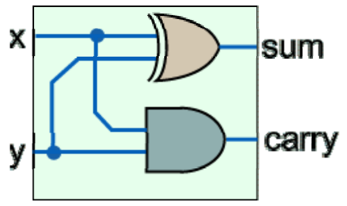
Q. Why is this a bad idea?
A. 128-bit adder: 2^{256+1} rows >> # electrons in universe!

1-bit half adder

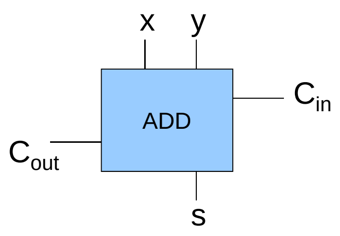
We add numbers one bit at a time.

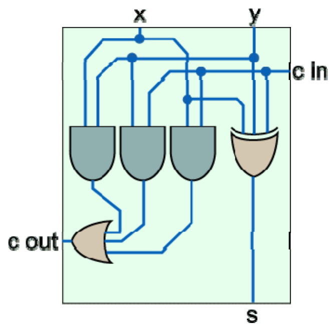


x	y	s	c



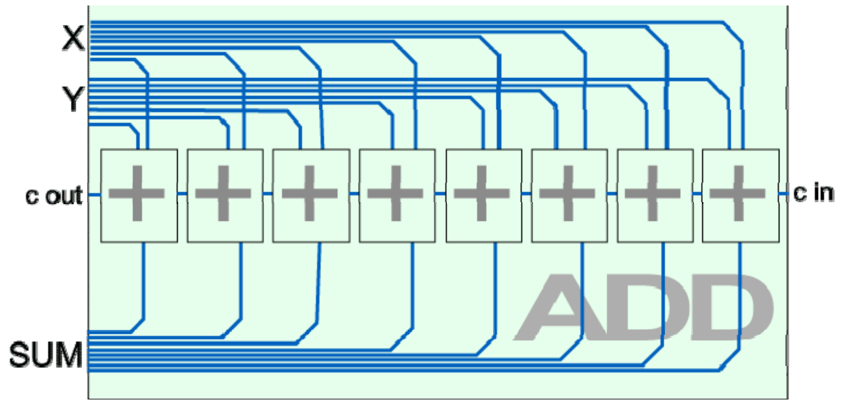
1-bit full adder





x	y	C_{in}	C_{out}	s

8-bit adder



Let's Make an Adder Circuit

Goal. x + y = z for 4-bit integers.

$$\begin{array}{cccccc} & c_{out} & c_3 & c_2 & c_1 & c_0 = 0 \\ & & x_3 & x_2 & x_1 & x_0 \\ + & & y_3 & y_2 & y_1 & y_0 \\ \hline & & z_3 & z_2 & z_1 & z_0 \end{array}$$

Step 2. [do one bit at a time]
- Build truth table for carry bit.
- Build truth table for summand bit.

Carry Bit

x_i	y_i	c_i	c_{i+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Summand Bit

x_i	y_i	c_i	z_i
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Let's Make an Adder Circuit

Goal. x + y = z for 4-bit integers.

Step 3.
- Derive (simplified) Boolean expression.

Carry Bit

x_i	y_i	c_i	c_{i+1}	MAJ
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

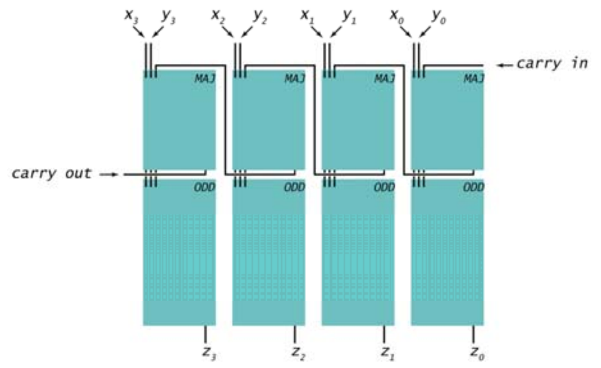
Summand Bit

x_i	y_i	c_i	z_i	ODD
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

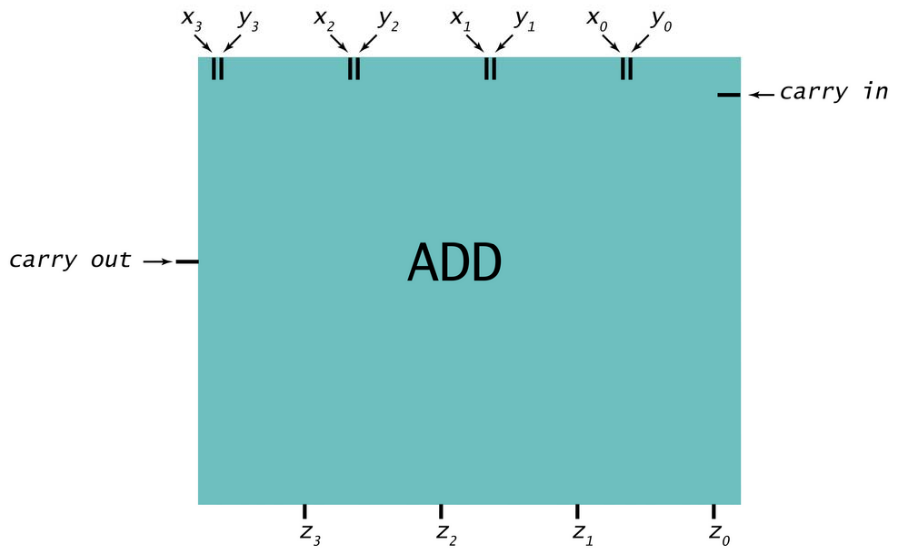
Let's Make an Adder Circuit

Goal. x + y = z for 4-bit integers.

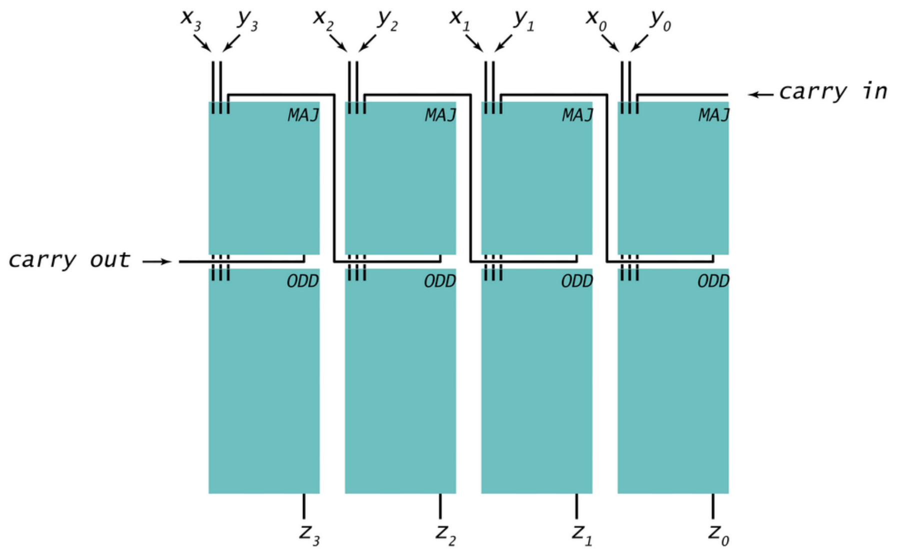
Step 4.
- Transform Boolean expression into circuit.
- Chain together 1-bit adders.



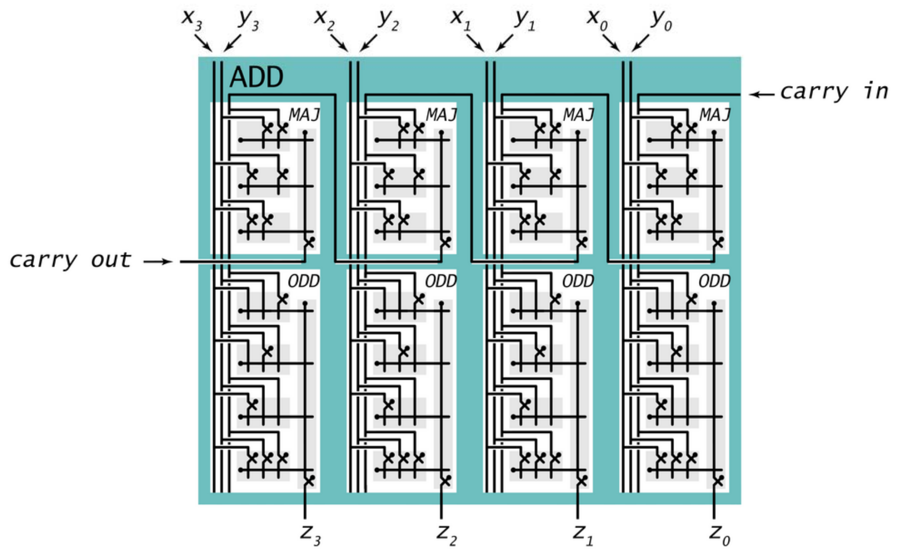
Adder: Interface



Adder: Component Level View



Adder: Switch Level View



Subtractor

Subtractor circuit: $z = x - y$.

- One approach: design like adder circuit

Subtractor

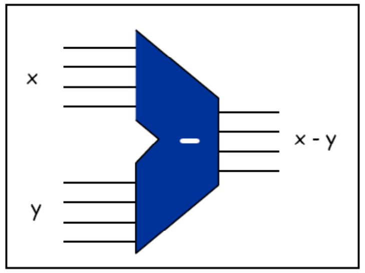
Subtractor circuit: $z = x - y$.

- One approach: design like adder circuit
- Better idea: reuse adder circuit
  - 2's complement: to negate an integer, flip bits, then add 1

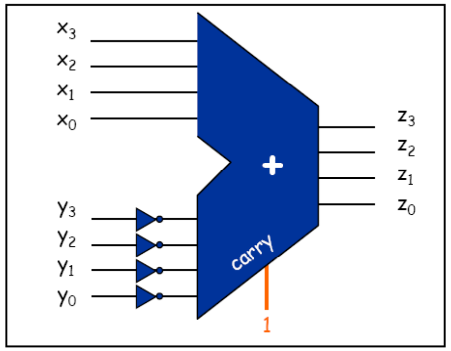
Subtractor

Subtractor circuit: $z = x - y$.

- One approach: design like adder circuit
- Better idea: reuse adder circuit
  - 2's complement: to negate an integer, flip bits, then add 1



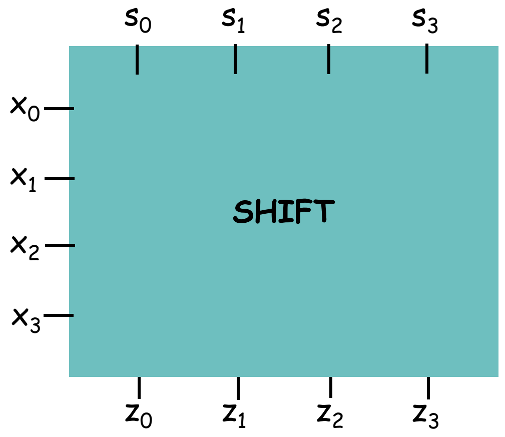
4-Bit Subtractor Interface



4-Bit Subtractor Implementation

Shifter

Only one of them will be on at a time.



4-bit Shifter

Shifter

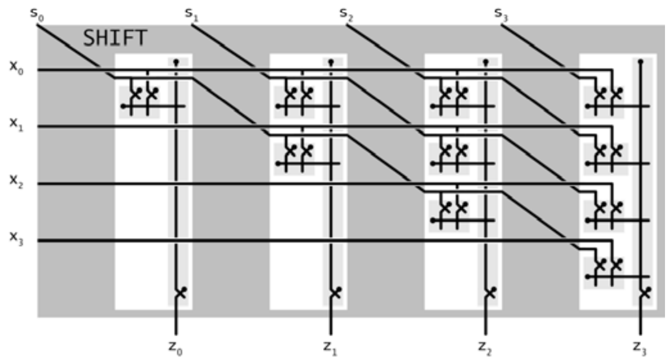
	z_0	z_1	z_2	z_3
s_0				
s_1				
s_2				
s_3				

Shifter

	z_0	z_1	z_2	z_3
s_0	x_0	x_1	x_2	x_3
s_1	0	x_0	x_1	x_2
s_2	0	0	x_0	x_1
s_3	0	0	0	x_0

$$z0 = s0 \cdot x0 + s1 \cdot 0 + s2 \cdot 0 + s3 \cdot 0$$
$$z1 = s0 \cdot x1 + s1 \cdot x0 + s2 \cdot 0 + s3 \cdot 0$$
$$z2 = s0 \cdot x2 + s1 \cdot x1 + s2 \cdot x0 + s3 \cdot 0$$
$$z3 = s0 \cdot x3 + s1 \cdot x2 + s2 \cdot x1 + s3 \cdot x0$$

Shifter

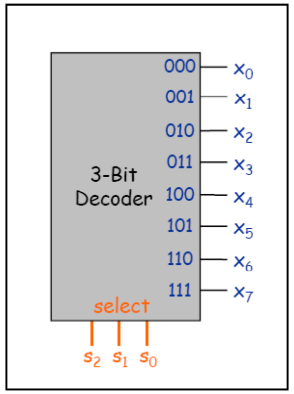


Right-shifter

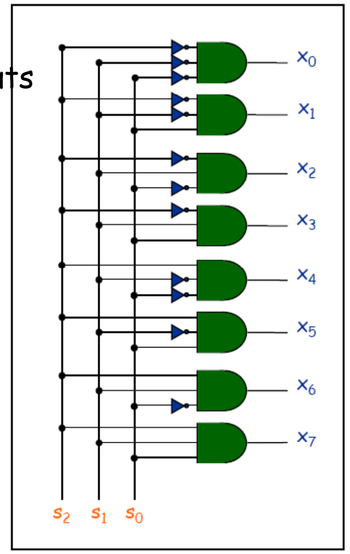
$$z0 = s0 \cdot x0 + s1 \cdot 0 + s2 \cdot 0 + s3 \cdot 0$$
$$z1 = s0 \cdot x1 + s1 \cdot x0 + s2 \cdot 0 + s3 \cdot 0$$
$$z2 = s0 \cdot x2 + s1 \cdot x1 + s2 \cdot x0 + s3 \cdot 0$$
$$z3 = s0 \cdot x3 + s1 \cdot x2 + s2 \cdot x1 + s3 \cdot x0$$

N-bit Decoder

N-bit decoder
- N address inputs, 2^N data outputs
- Addresses output bit is 1; all others are 0



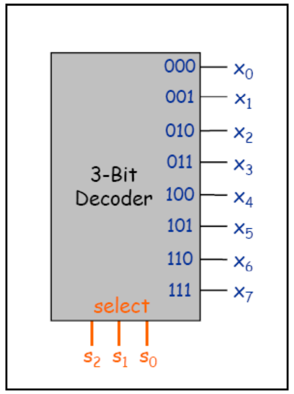
3-Bit Decoder Interface



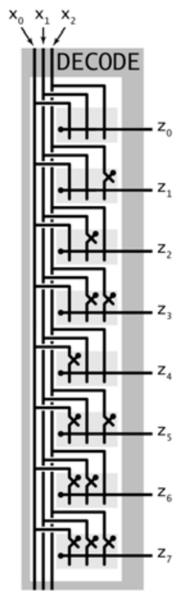
3-Bit Decoder Implementation

N-bit Decoder

N-bit decoder
- N address inputs, 2^N data outputs
- Addresses output bit is 1; all others are 0



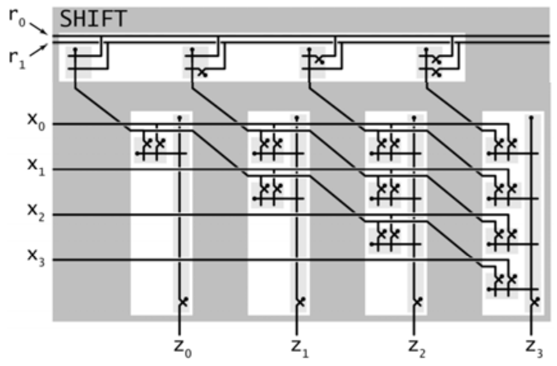
3-Bit Decoder Interface



Decoder

2-Bit Decoder Controlling 4-Bit Shifter

Ex. Put in a binary amount $r_0 r_1$ to shift.



Right-shifter with decoder

Arithmetic Logic Unit

Arithmetic logic unit (ALU). Computes all operations in parallel.

- Add and subtract.
- Xor.
- And.
- Shift left or right.

Q. How to select desired answer?

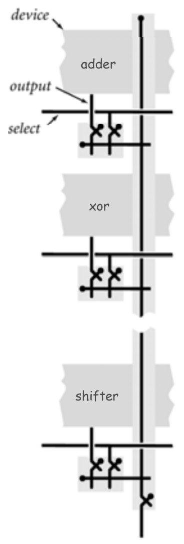
1 Hot OR

1 hot OR.

- All devices compute their answer; we pick one.
- Exactly one select line is on.
- Implies exactly one output line is relevant.

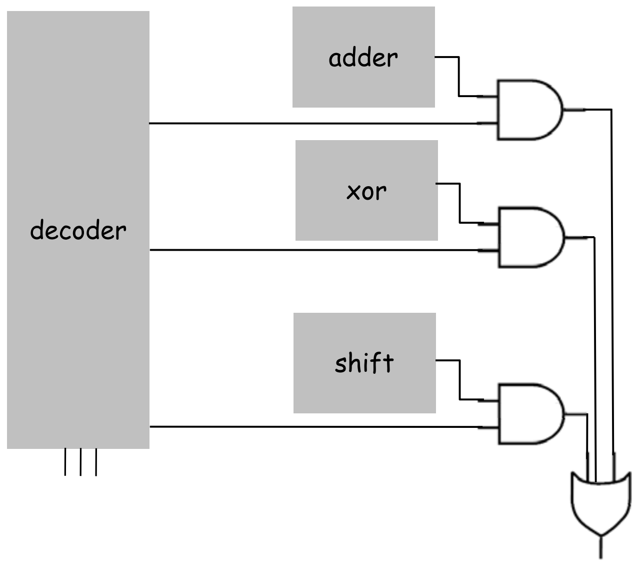
$x \cdot 1 = x$
$x \cdot 0 = 0$

$x + 0 = x$



Output select with one-hot OR

1 Hot OR

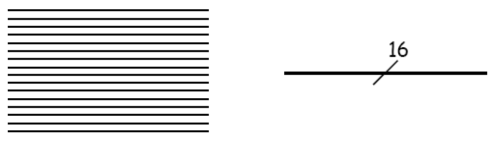


$x \cdot 1 = x$
$x \cdot 0 = 0$

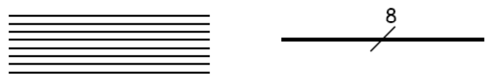
$x + 0 = x$

Bus

16-bit bus
- Bundle of 16 wires
- Memory transfer
  Register transfer



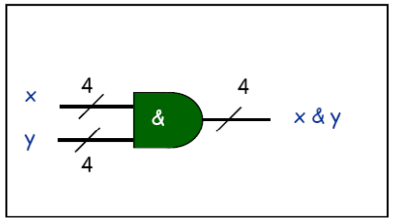
8-bit bus
- Bundle of 8 wires
- TOY memory address



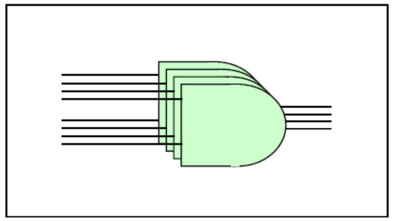
4-bit bus
- Bundle of 4 wires
- TOY register address

Bitwise AND, XOR, NOT

Bitwise logical operations
- Inputs x and y: n bits each
- Output z: n bits
- Apply logical operation to each corresponding pair of bits



Bitwise And Interface

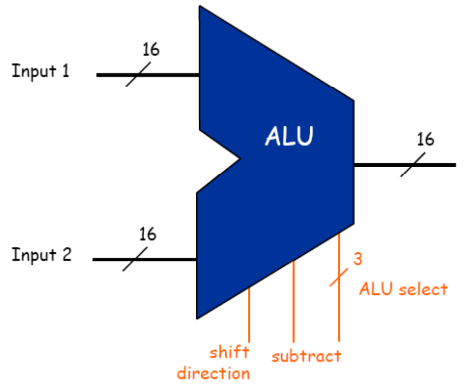


Bitwise And Implementation

TOY ALU

TOY ALU
- Big combinational logic
- 16-bit bus
- Add, subtract, and, xor, shift left, shift right,

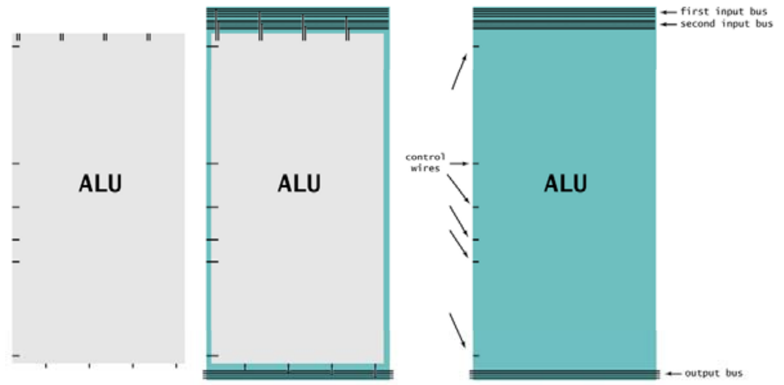
op	2	1	0
+, -	0	0	0
&	0	0	1
^	0	1	0
<<, >>	0	1	1
input 2	1	0	0



Device Interface Using Buses

16-bit words for TOY memory

Device. Processes a word at a time.
Input bus. Wires on top.
Output bus. Wires on bottom.
Control. Individual wires on side.

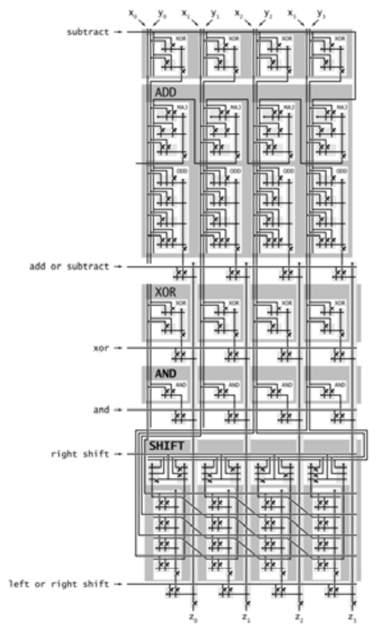


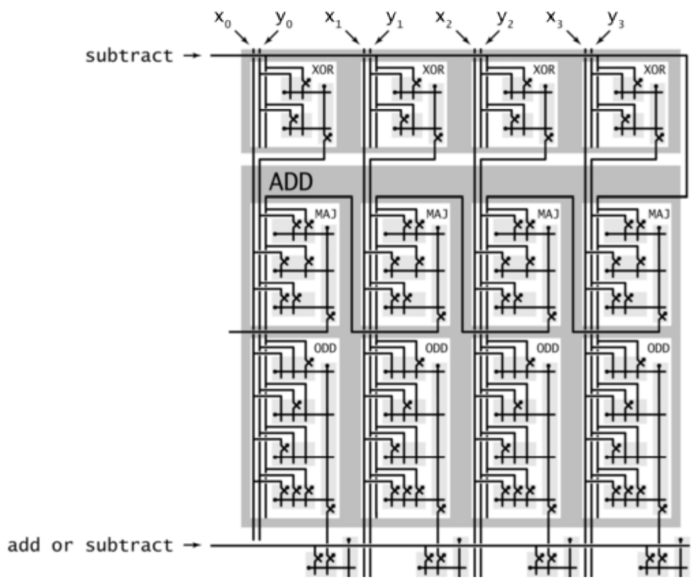
ALU

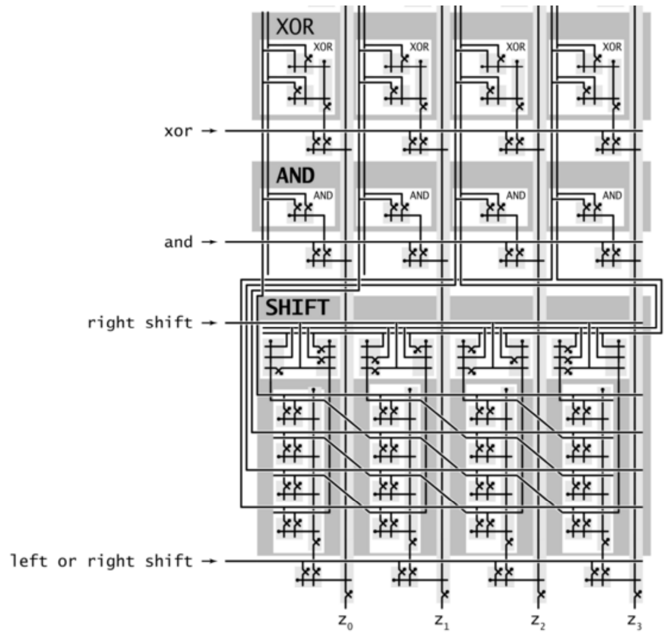
Arithmetic logic unit.
- Add and subtract.
- Xor.
- And.
- Shift left or right.

Arithmetic logic unit.
- Computes all operations in parallel.
- Uses 1-hot OR to pick each bit answer.

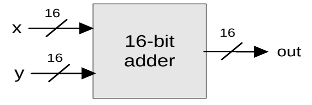
How to convert opcode to 1-hot OR signal?

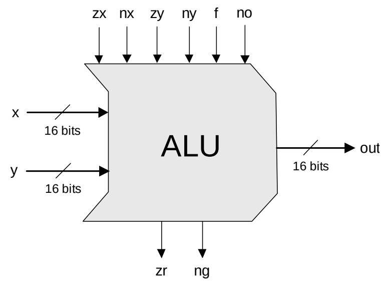






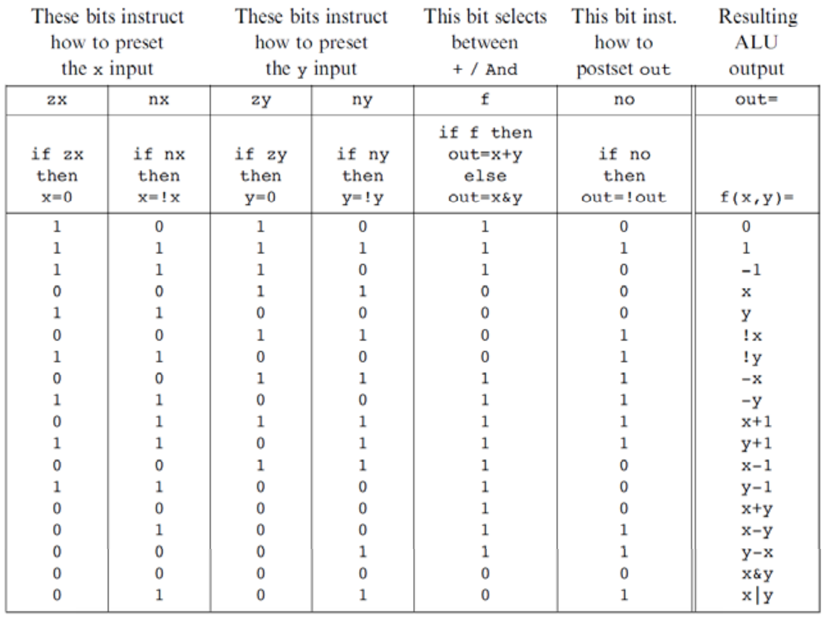
Hack ALU





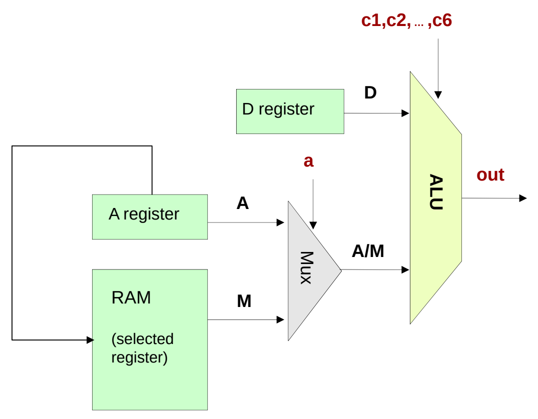
```
out(x, y, control bits) =
  x+y, x-y, y–x,
  0, 1, -1,
  x, y, -x, -y,
  x!, y!,
  x+1, y+1, x-1, y-1,
  x&y, x|y
```

Hack ALU



These bits instruct how to preset the x input		These bits instruct how to preset the y input		This bit selects between + / And	This bit inst. how to postset out	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	f(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

The ALU in the CPU context (a sneak preview of the Hack platform)



Perspective

- Combinational logic
- Our adder design is very basic: no parallelism
- It pays to optimize adders
- Our ALU is also very basic: no multiplication, no division
- Where is the seat of more advanced math operations?
  a typical hardware/software tradeoff.