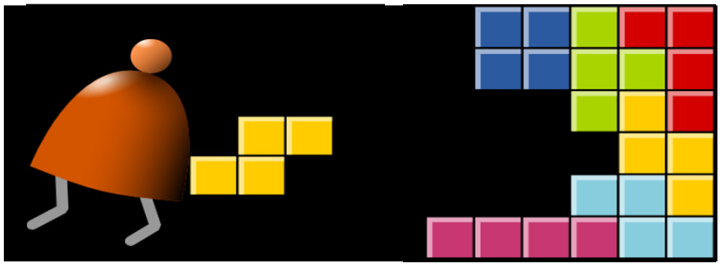
Boolean Arithmetic



Building a Modern Computer From First Principles

www.nand2tetris.org

Usage and Copyright Notice:

This presentation accompanies the textbook "The Elements of Computing Systems"
by Noam Nisan & Shimon Schocken, MIT Press, 2005.

We provide 13 such presentations.

Each presentation is designed to support about 3 hours of classroom or self-study instruction.

You are welcome to use or edit this presentation as you see fit for instructional and non-commercial purposes.

If you use our book and course materials, please include a reference to www.nand2tetris.org

If you have any questions or comments, please write us at nand2tetris@gmail.com

Counting systems

quantity	decimal	binary	3-bit register
	0	0	000
*	1	1	001
**	2	10	010
***	3	11	011
****	4	100	100
*****	5	101	101
******	6	110	110
*******	7	111	111
********	8	1000	overflow
*********	9	1001	overflow
**********	10	1010	overflow

Rationale

$$(9038)_{ten} = 9 \cdot 10^3 + 0 \cdot 10^2 + 3 \cdot 10^1 + 8 \cdot 10^0 = 9038$$

$$(10011)_{two} = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 19$$

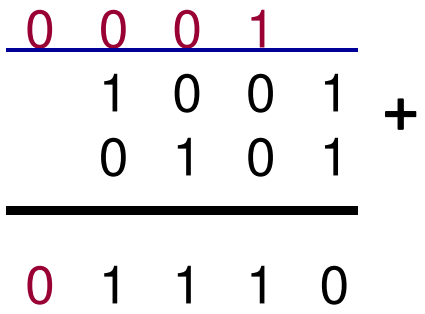
$$(x_n x_{n-1} \ldots x_0)_b = \sum_{i=0}^{n} x_i \cdot b^i$$

Hexadecimal and Binary

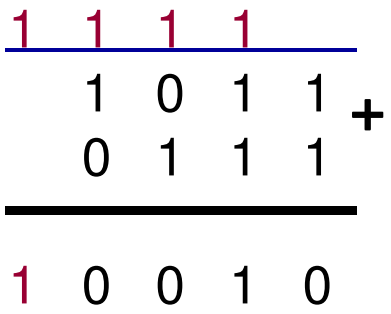
decimal	hexadecimal	binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Binary addition

Assuming a 4-bit system:



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 negate a number:
  flip (invert) all bits, then add 1

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

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

Signed Arithmetic (4-bit system)

Example 2 6 - 5 = 6 + (-5) = 0 1 1 0

+ 1 0 1 1

1 dropped; overflow??? 1 0 0 0 1 = 1

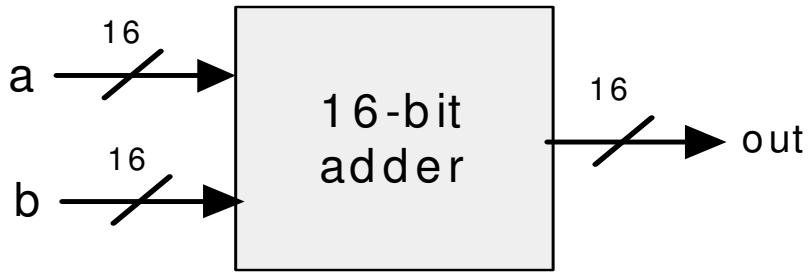
Signed Arithmetic (4-bit system)

Example 3

$$7 + 1 = \quad 0\,1\,1\,1$$
$$+\,0\,0\,0\,1$$

Now what??? $\quad 1\,0\,0\,0 \quad = \quad -8$

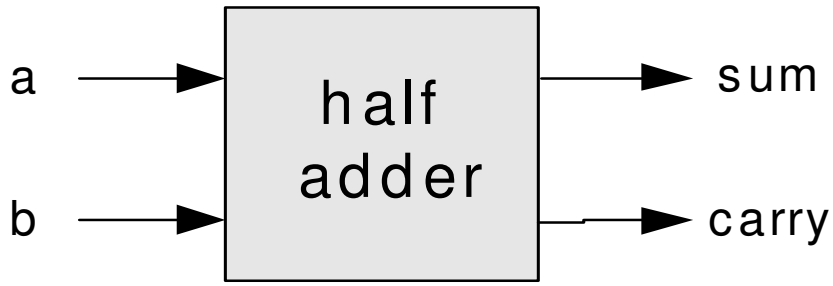
Building an Adder chip



- Adder: a chip designed to add two integers
- Proposed implementation:
  - Half adder: designed to add 2 bits
  - Full adder: designed to add 3 bits
  - Adder: designed to add two *n*-bit numbers.

Half adder (designed to add 2 bits)

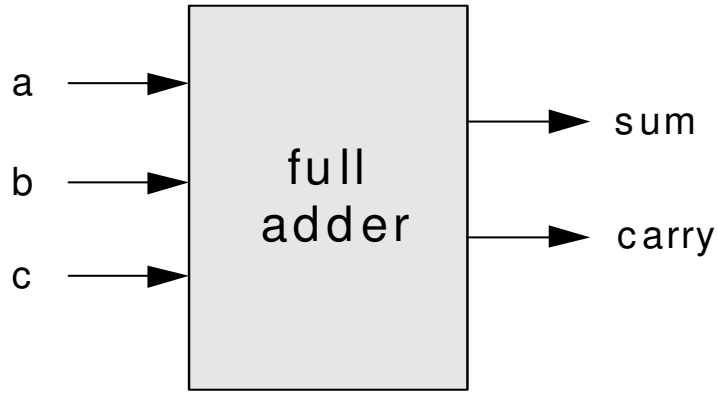
a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Implementation: based on two gates that you've seen before.

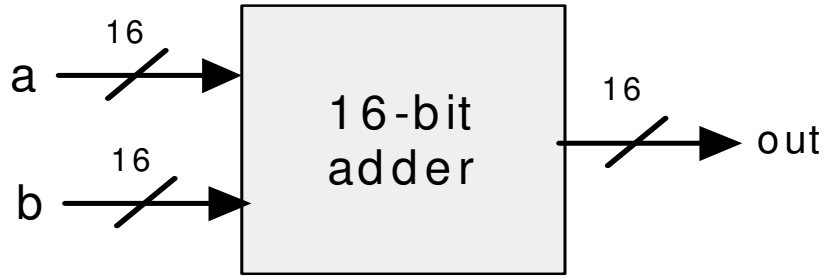
Full adder (designed to add 3 bits)

a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Implementation: can be based on half-adder gates.

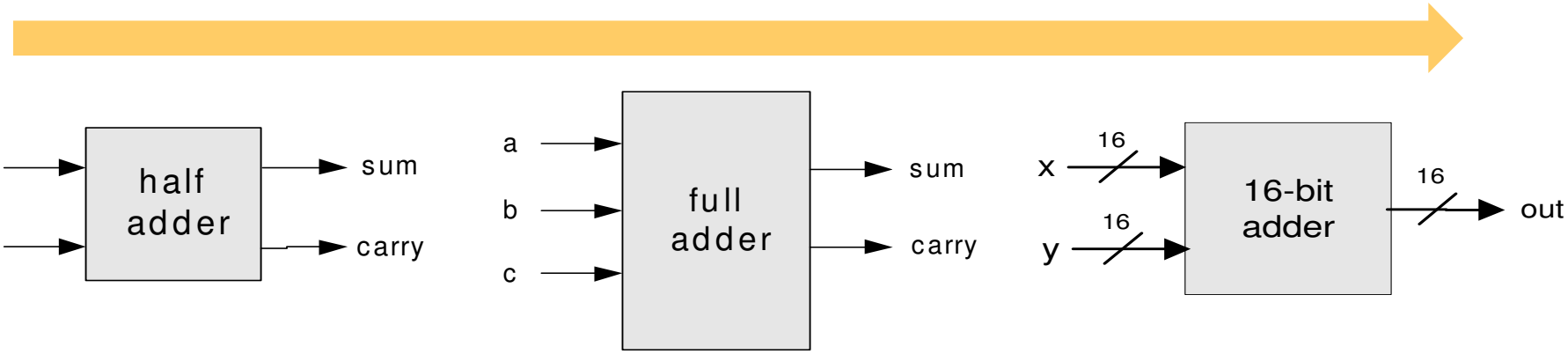
n-bit Adder (designed to add two 16-bit numbers)

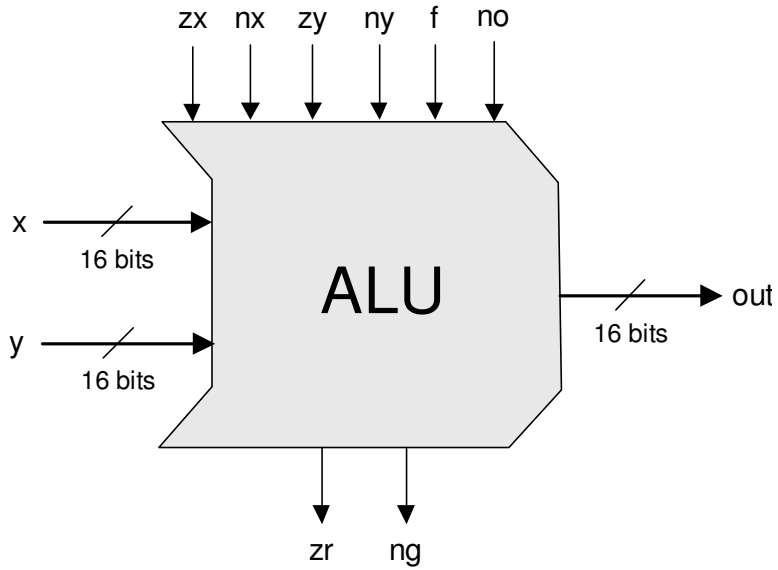


...	1	0	1	1	a
					+
...	0	0	1	0	b
...	1	1	0	1	out

Implementation: array of full-adder gates.

The ALU (of the Hack platform)





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,

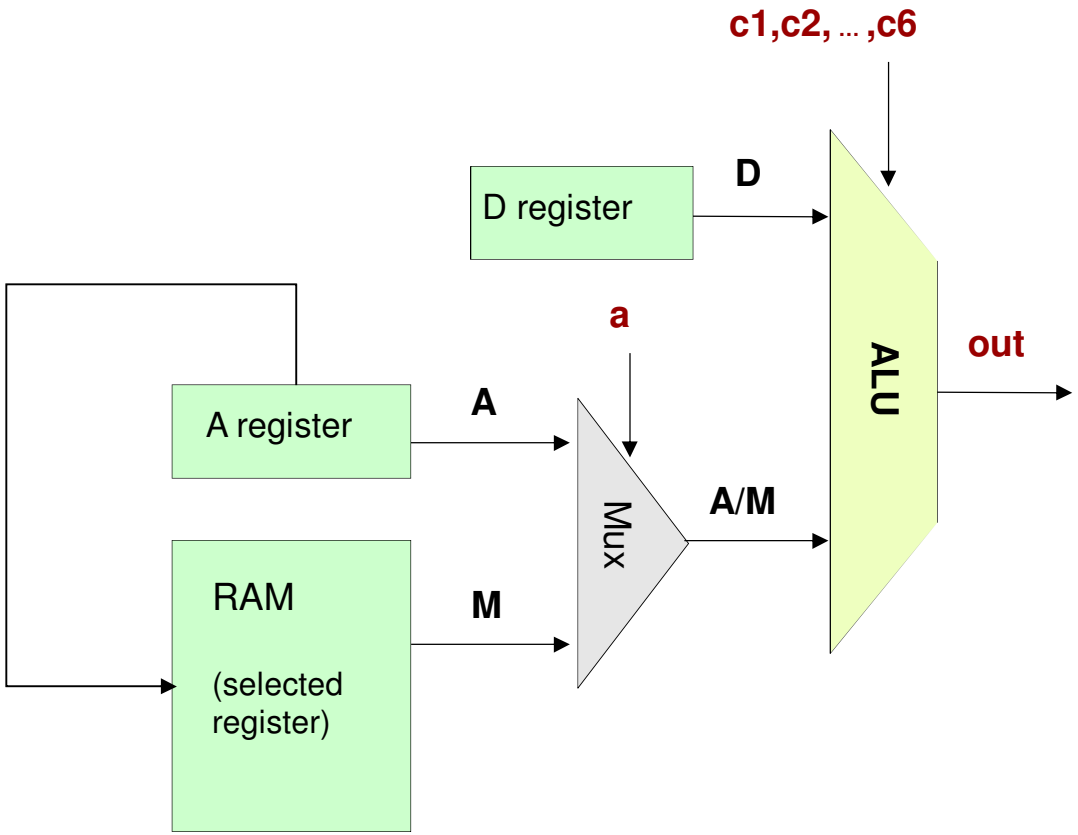
x&y, x|y

ALU logic (Hack platform)

These bits instruct how to pre-set the x input		These bits instruct how to pre-set the y input		This bit selects between + / And	This bit inst. how to post-set 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 And 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						y
0						!x
1						!y
0						-x
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

Implementation: build a logic gate architecture that "executes" the control bit "instructions": if zx==1 then set x to 0 (bit-wise), etc.

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.