Numbers & Arithmetic

Hakim Weatherspoon
CS 3410, Spring 2011
Computer Science
Cornell University

See: P&H Chapter 2.4 - 2.6, 3.2, C.5 – C.6

Announcements

Make sure you are

- Registered for class
- Can access CMS
- Have a Section you can go to
- Have a project partner

Sections are on this week

HW 1 out later today

- Due in one week, start early
- Work alone
- Use your resources
  - Class notes, book, Sections, office hours, newsgroup, CSUGLab

Announcements

Check online syllabus/schedule

- Slides and Reading for lectures
- Office Hours
- Homework and Programming Assignments
- Prelims: Thursday, March 11 and April 28th

- Schedule is subject to change

Goals for today

Review

- Circuit design (e.g. voting machine)
- Number representations
- Building blocks (encoders, decoders, multiplexors)

Binary Operations

- One-bit and four-bit adders
- Negative numbers and two's compliment
- Addition (two's compliment)
- Subtraction (two's compliment)
- Performance

Logic Minimization

- How to implement a desired function?

a	b	c	out
0	**0**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**0**	**0**
1	**0**	**1**	**1**
1	**1**	**0**	**0**
1	**1**	**1**	**0**

$\bar{a}\bar{b}\bar{c}$

$\bar{a}\bar{b}c$

$\bar{a}bc$

$a\bar{b}c$

Logic Minimization

- How to implement a desired function?

a	b	c	out	minterm
0	**0**	**0**	**0**	$\overline{a}\,\overline{b}\,\overline{c}$
0	**0**	**1**	**1**	$\overline{a}\,\overline{b}\,c$
0	**1**	**0**	**0**	$\overline{a}\,b\,\overline{c}$
0	**1**	**1**	**1**	$\overline{a}\,b\,c$
1	**0**	**0**	**0**	$a\,\overline{b}\,\overline{c}$
1	**0**	**1**	**1**	$a\,\overline{b}\,c$
1	**1**	**0**	**0**	$a\,b\,\overline{c}$
1	**1**	**1**	**0**	$a\,b\,c$

sum of products:

- OR of all minterms where out=1

corollary: *any* combinational circuit *can be* implemented in two levels of logic (ignoring inverters)

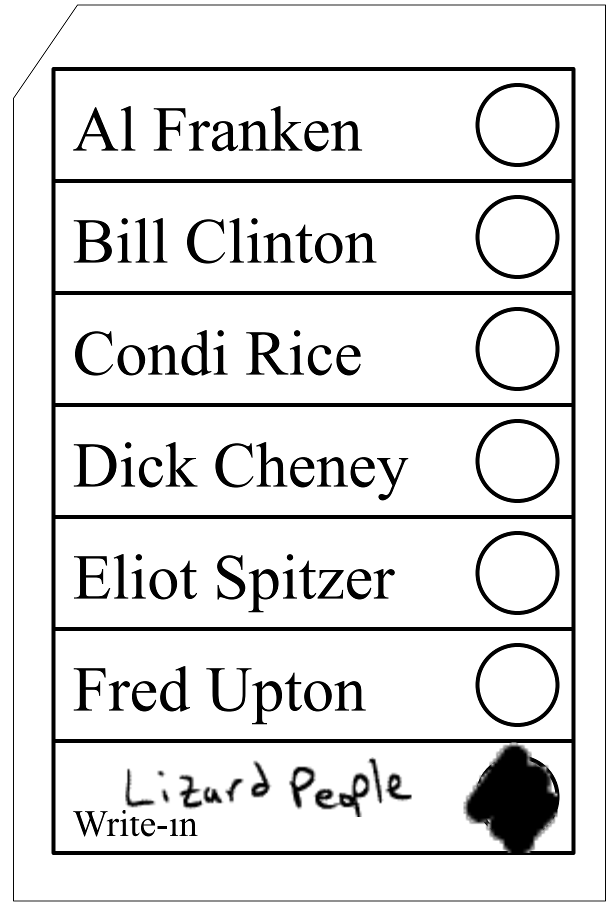
Karnaugh Maps

How does one find the most efficient equation?
- Manipulate algebraically until…?
- Use Karnaugh maps (optimize visually)
- Use a software optimizer

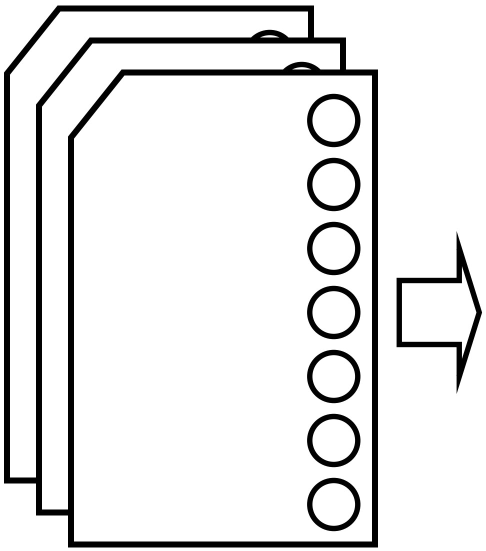
For large circuits
- Decomposition & reuse of building blocks

Voting machine

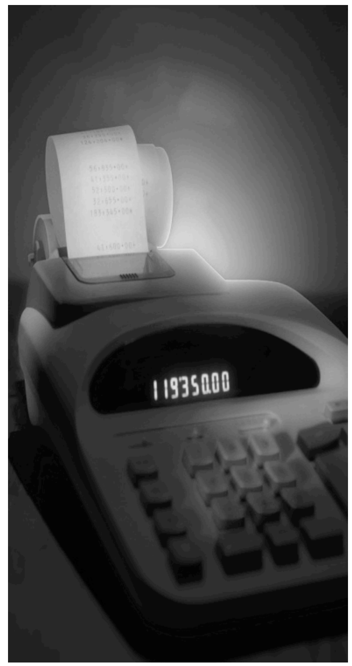
- Voting Machine!
  - optical scan (thanks FL)
- Assume:
  - vote is recorded on paper by filling a circle
  - fixed number of choices
  - don't worry about "invalids"



Voting Machine Components



Ballots



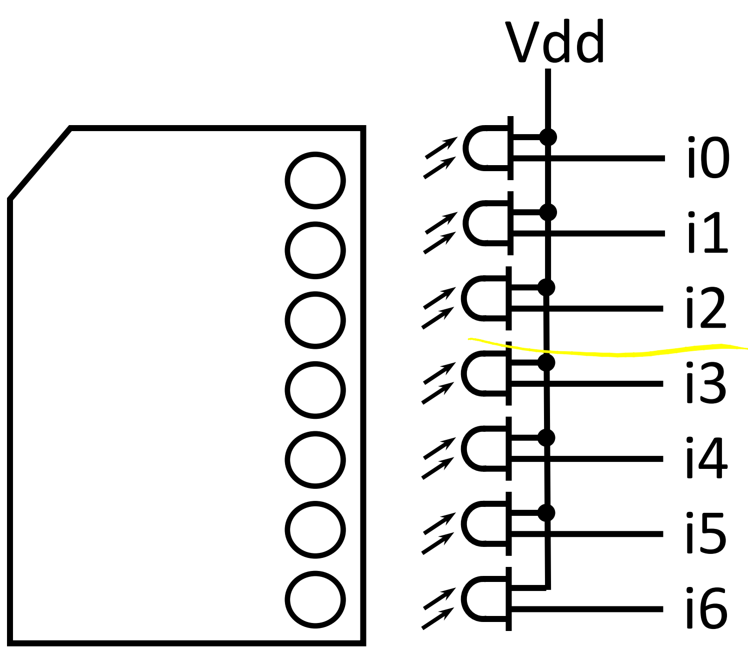
The 3410 optical scan vote ~~counter~~ reader machine

5 Essential Components?

- Input: paper with at exactly one mark
- Datapath: process current ballot
- Output: a number the supervisor can record
- Memory & control: none for now

Input

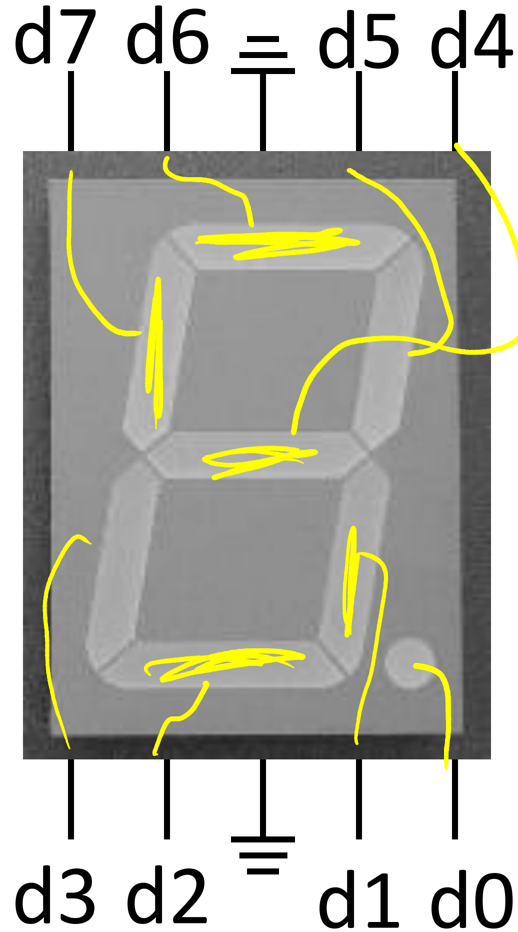
- Photo-sensitive transistor
  - photons replenish gate depletion region
  - can distinguish dark and light spots on paper



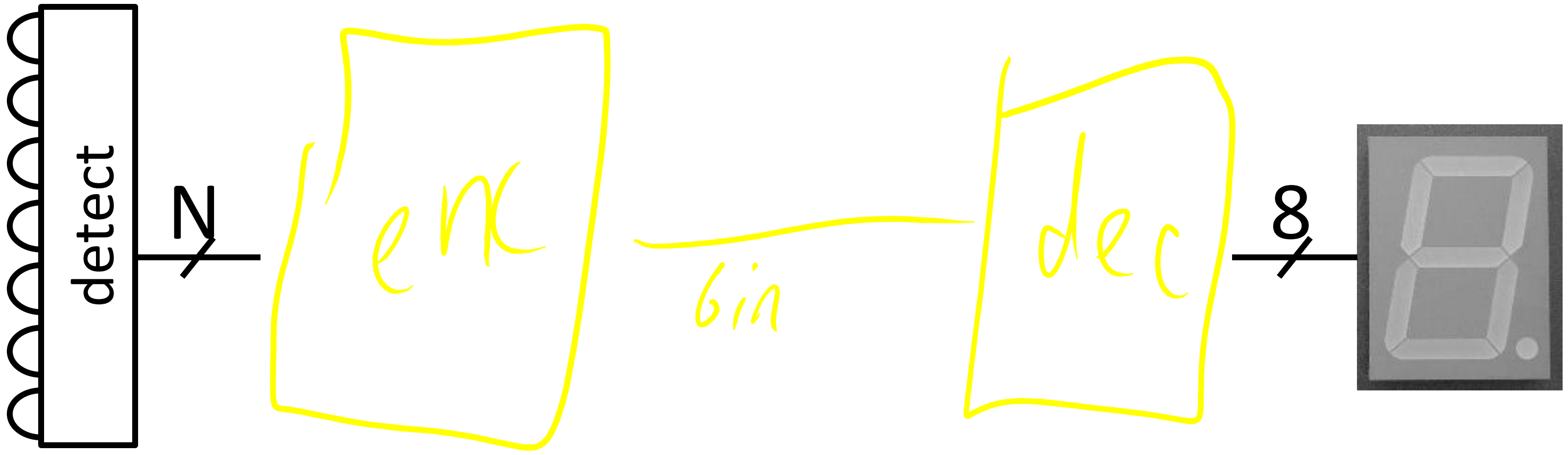
- Use array of N sensors for voting machine input

Output

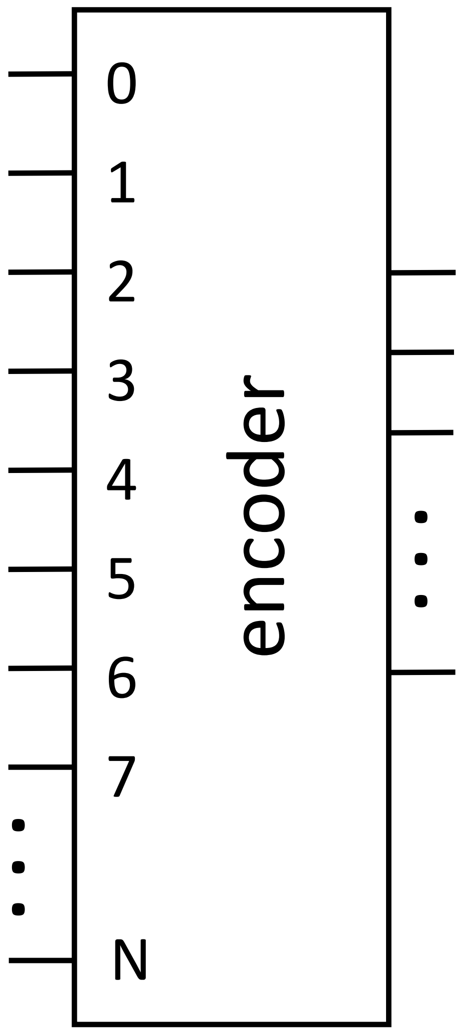
- 7-Segment LED
  - photons emitted when electrons fall into holes



Block Diagram



Encoders



- N might be large
- Routing wires is expensive
- More efficient encoding?

$\log_2(N)$

Number Representations

6 3 7
___ ___ ___

10^2 10^1 10^0

- Base 10 - Decimal

- Just as easily use other bases
  - Base 2 - Binary
  - Base 8 - Octal
  - Base 16 - Hexadecimal

Counting

- Counting

dec

0
1
2
3
4
5
6
7
8
9
10
:
100

oct

0
1
2
3
4
5
6
7
10
11
12
:
77
100

Base Conversion

- Base conversion via repetitive division
  - Divide by base, write remainder, move left with quotient

637 ÷ 10 = 63 rem 7 (lsb)

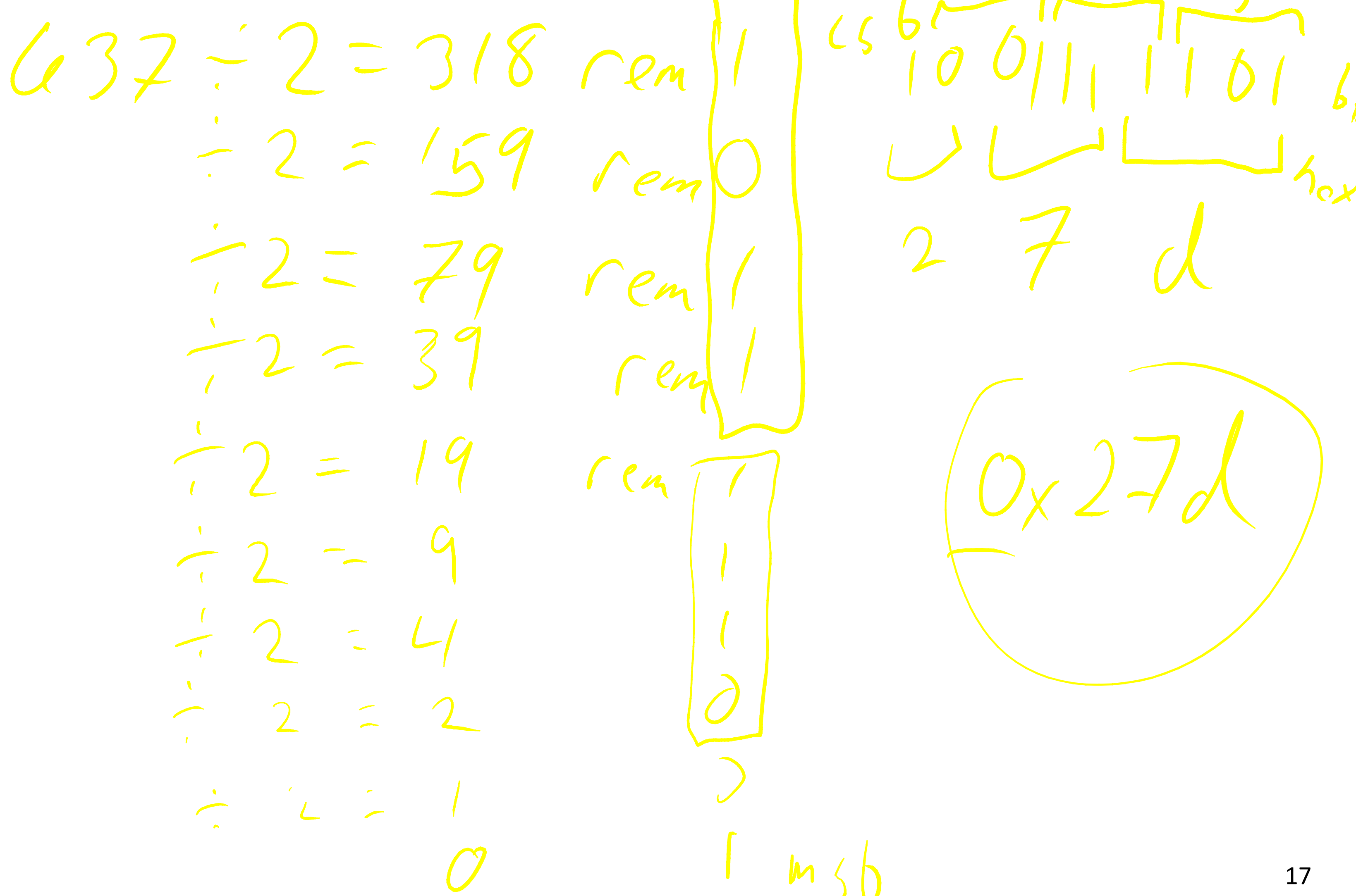
÷ 10 = 6 rem 3

÷ 10 = 0 rem 6 (msb)

637

Base Conversion

- Base conversion via repetitive division
  - Divide by base, write remainder, move left with quotient



Base Conversion

- Base conversion via repetitive division
  - Divide by base, write remainder, move left with quotient

637 ÷ 16 = 39 rem 13 (lsb)
÷ 16 = 2 rem 7
÷ 16 = 0 rem 2 (msb)

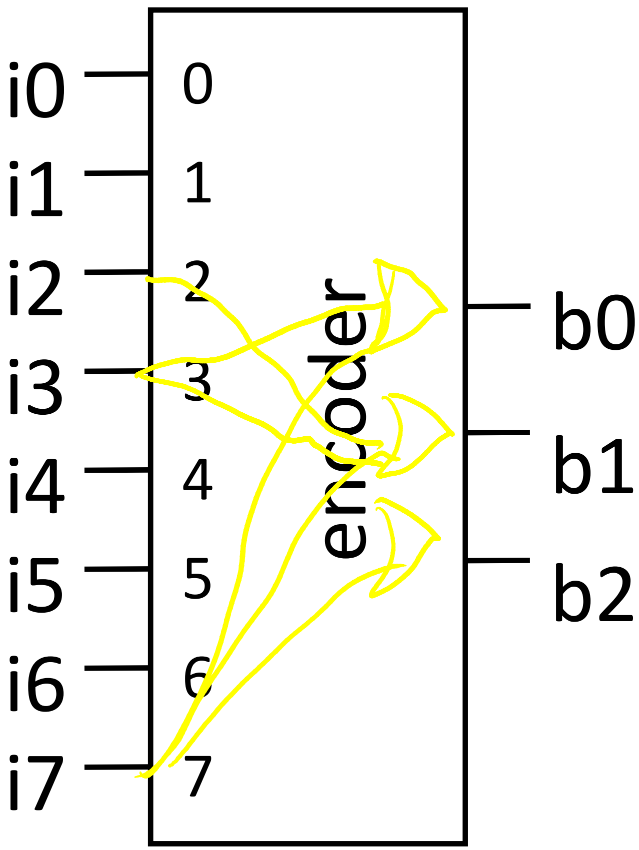
16^2 = 16, × 40 = 640

2 7 13
2 7 d
0x27d

Hexadecimal, Binary, Octal Conversions

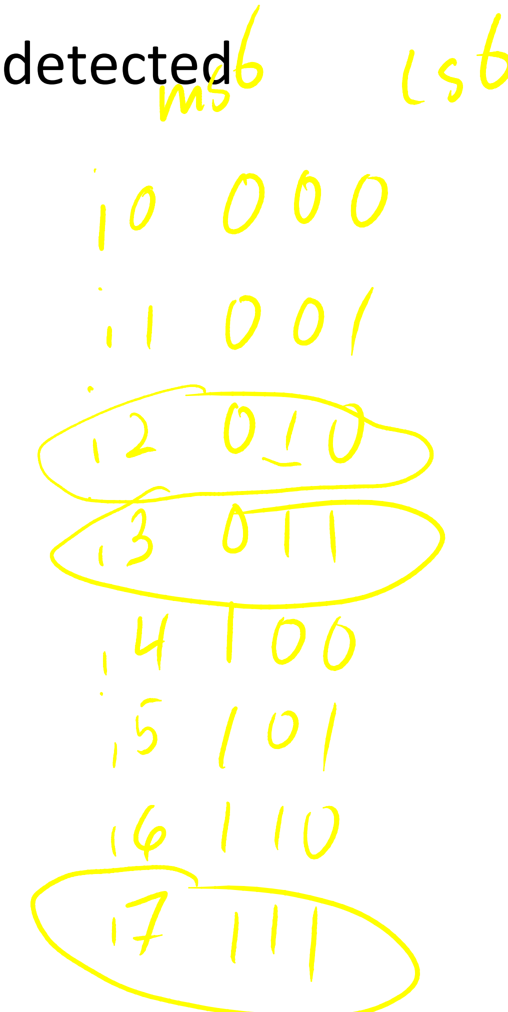
Encoder Implementation

- Implementation . . .
  - assume 8 choices, exactly one mark detected

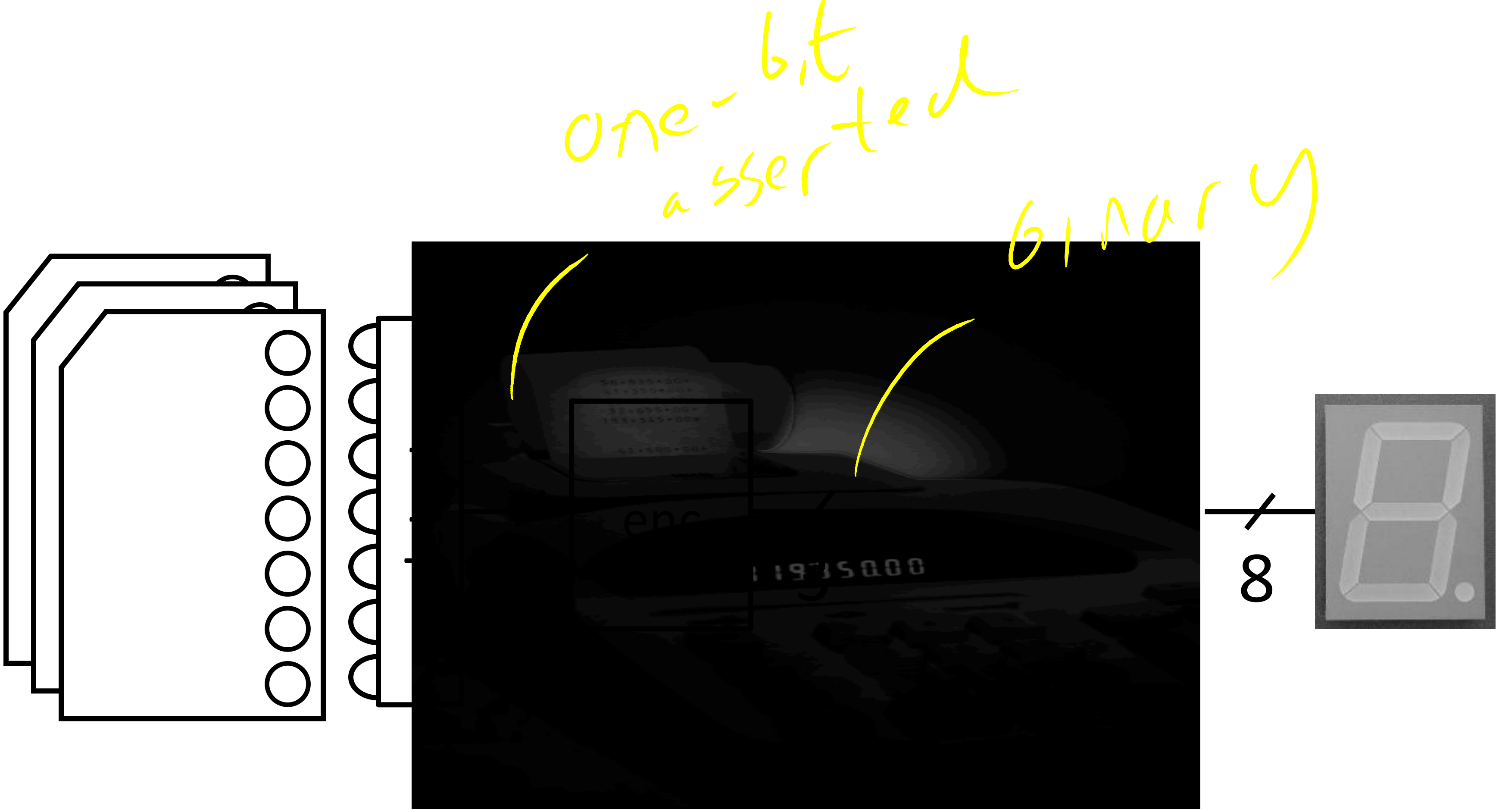


3-bit encoder

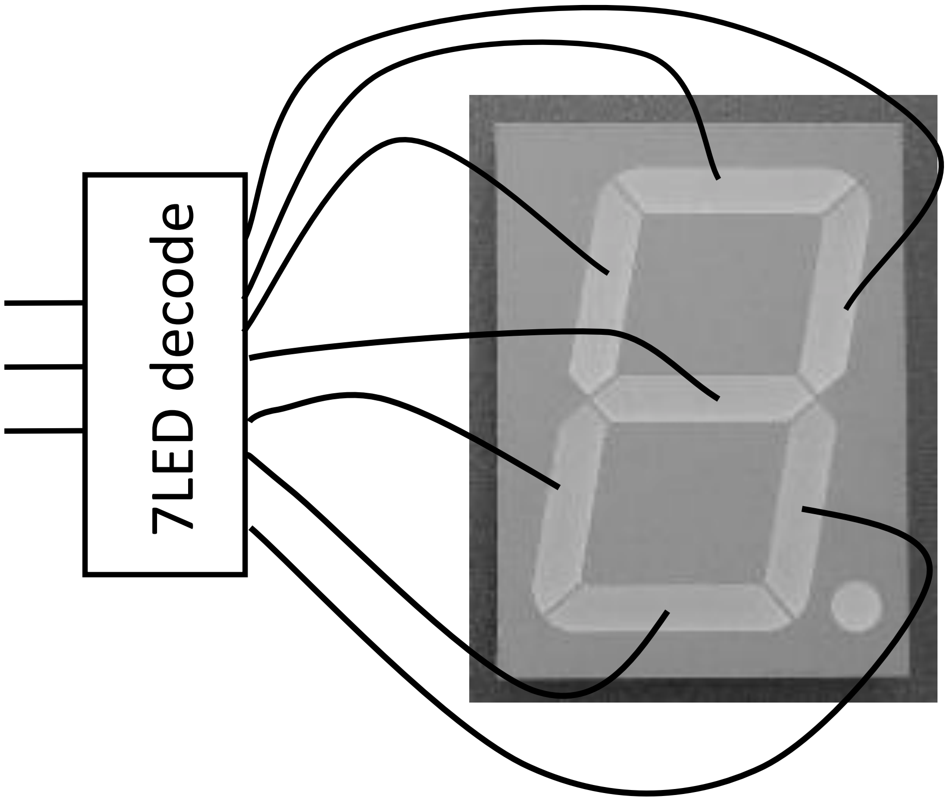
(8-to-3)



Ballot Reading



7-Segment LED Decoder

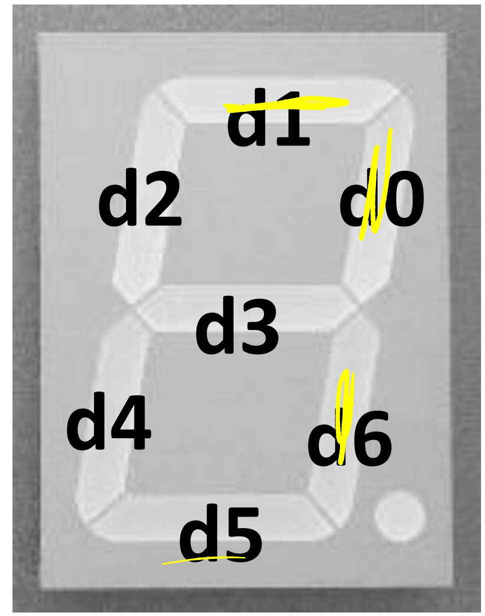


- 3 inputs
- encode 0 – 7 in binary

- 7 outputs
- one for each LED

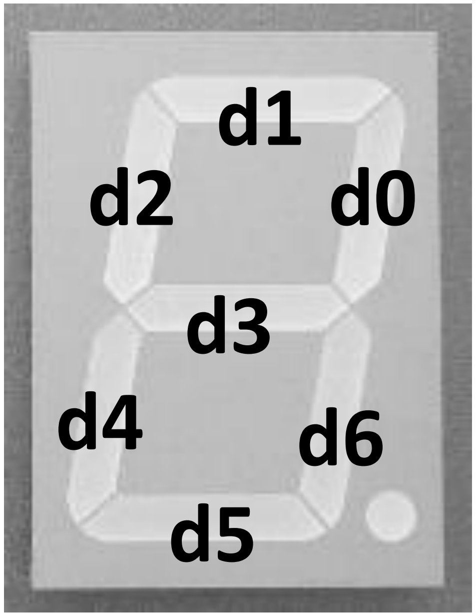
7 Segment LED Decoder Implementation

b2	b1	b0	d6	d5	d4	d3	d2	d1	d0
0	0	0							
0	0	1							
0	1	0							
0	1	1							
1	0	0							
1	0	1							
1	1	0	1	0	0	0	0	1	1
1	1	1							



7 Segment LED Decoder Implementation

b2	b1	b0	d6	d5	d4	d3	d2	d1	d0
0	0	0	1	1	1	0	1	1	1
0	0	1	1	0	0	0	0	0	1
0	1	0	0	1	1	1	0	1	1
0	1	1	1	1	0	1	0	1	1
1	0	0	1	0	0	1	1	0	1
1	0	1	1	1	0	1	1	1	0
1	1	0	1	1	1	1	1	1	0
1	1	1	1	0	0	0	0	1	1

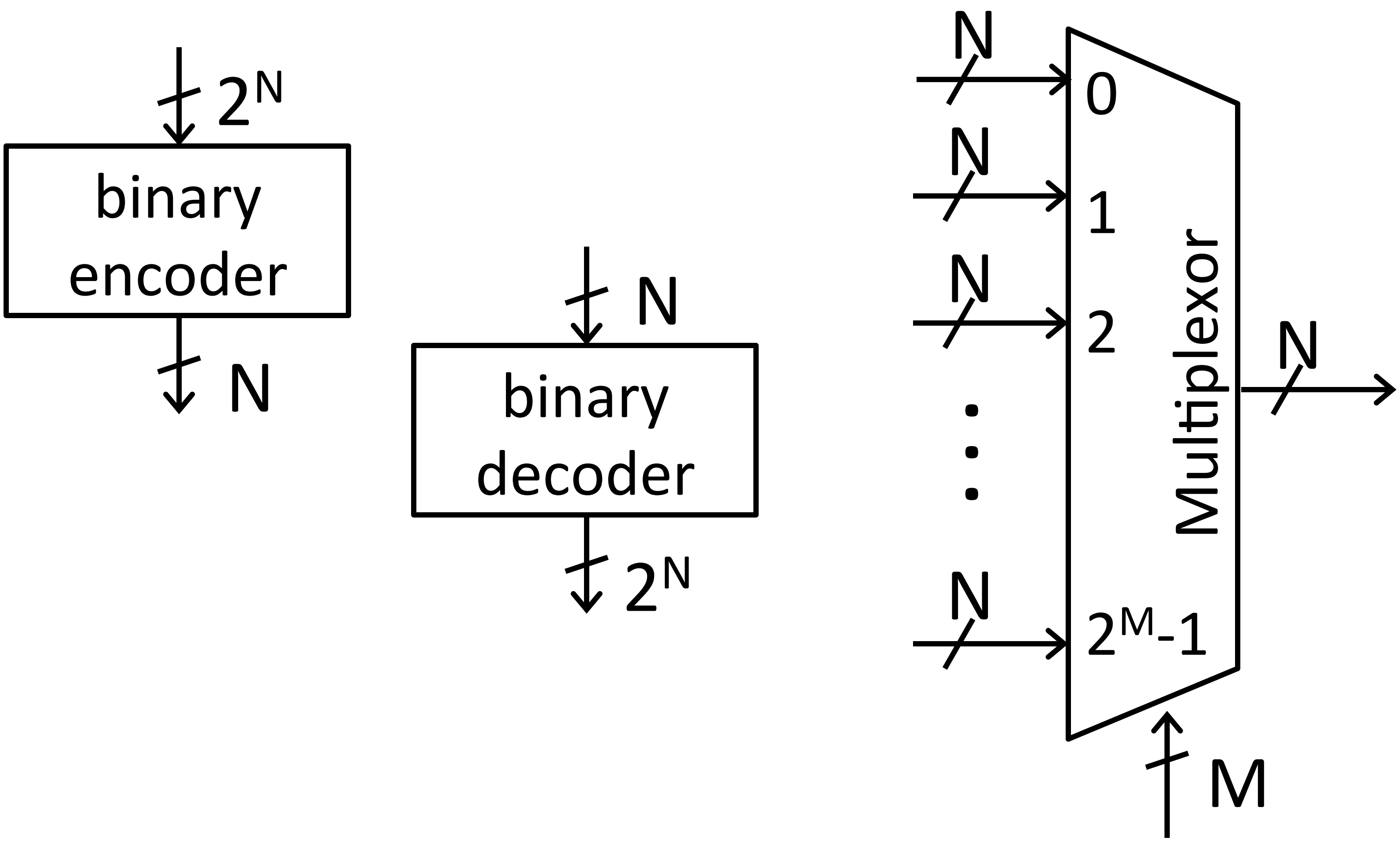


Ballot Reading and Display

Ballots

The 3410 optical scan
vote ~~counter~~ reader
machine

Building Blocks



Goals for today

Review

- Circuit design (e.g. voting machine)
- Number representations
- Building blocks (encoders, decoders, multiplexors)

Binary Operations

- One-bit and four-bit adders
- Negative numbers and two's compliment
- Addition (two's compliment)
- Subtraction (two's compliment)
- Performance

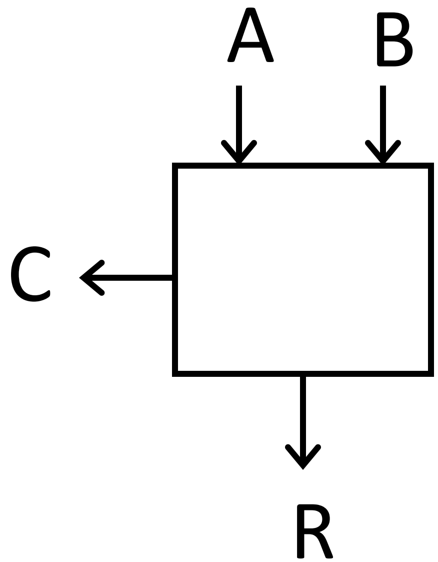
Binary Addition

```
  183
+ 254
```

- Addition works the same way regardless of base
  - Add the digits in each position
  - Propagate the carry

```
+ 011100
```

1-bit Adder

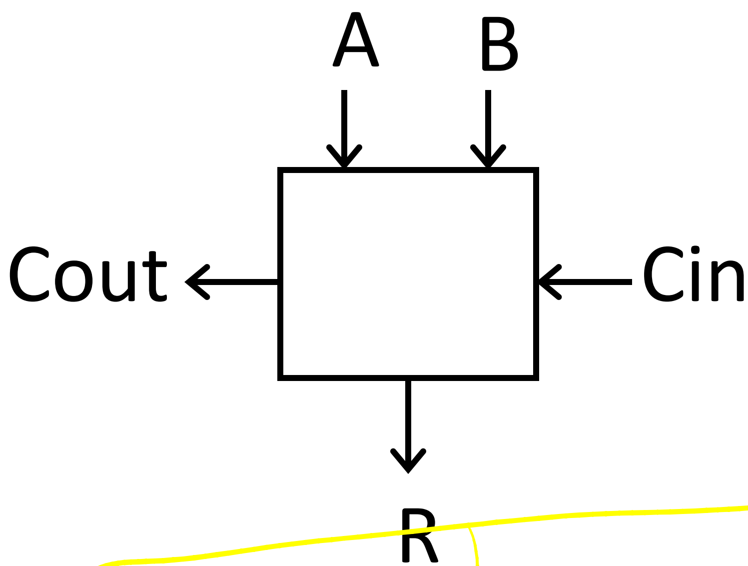


A	B	C	R
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half Adder

- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry

1-bit Adder with Carry

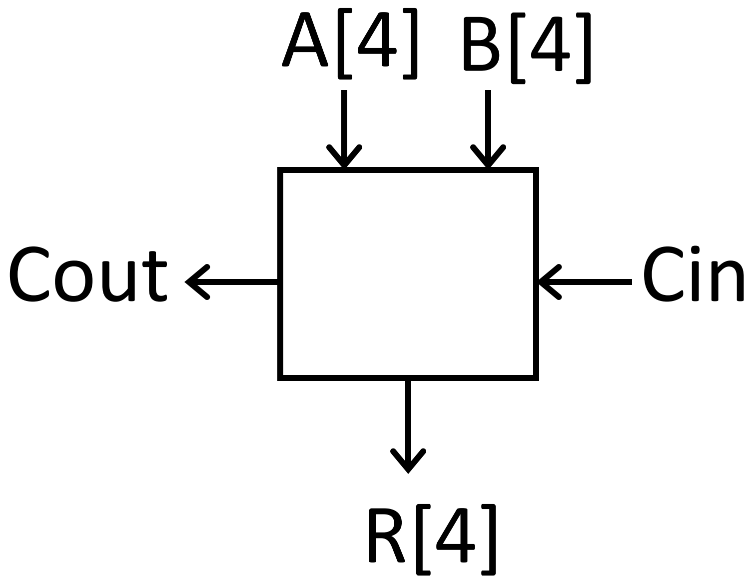


Full Adder

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

A	B	C_{in}	C_{out}	R
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

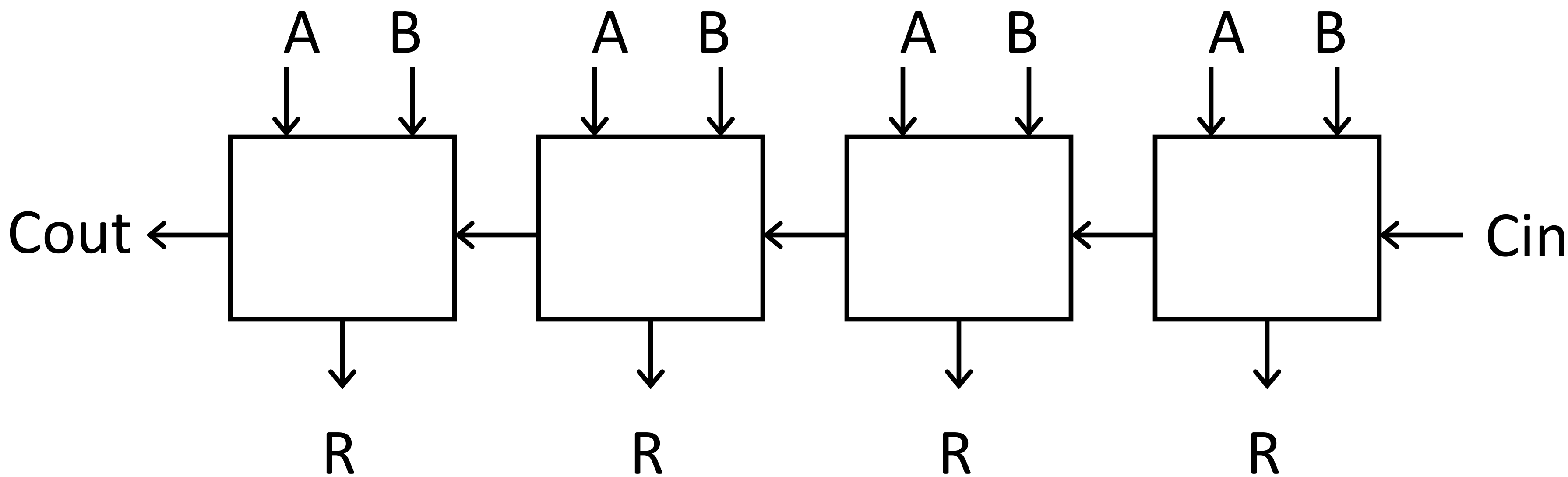
4-bit Adder



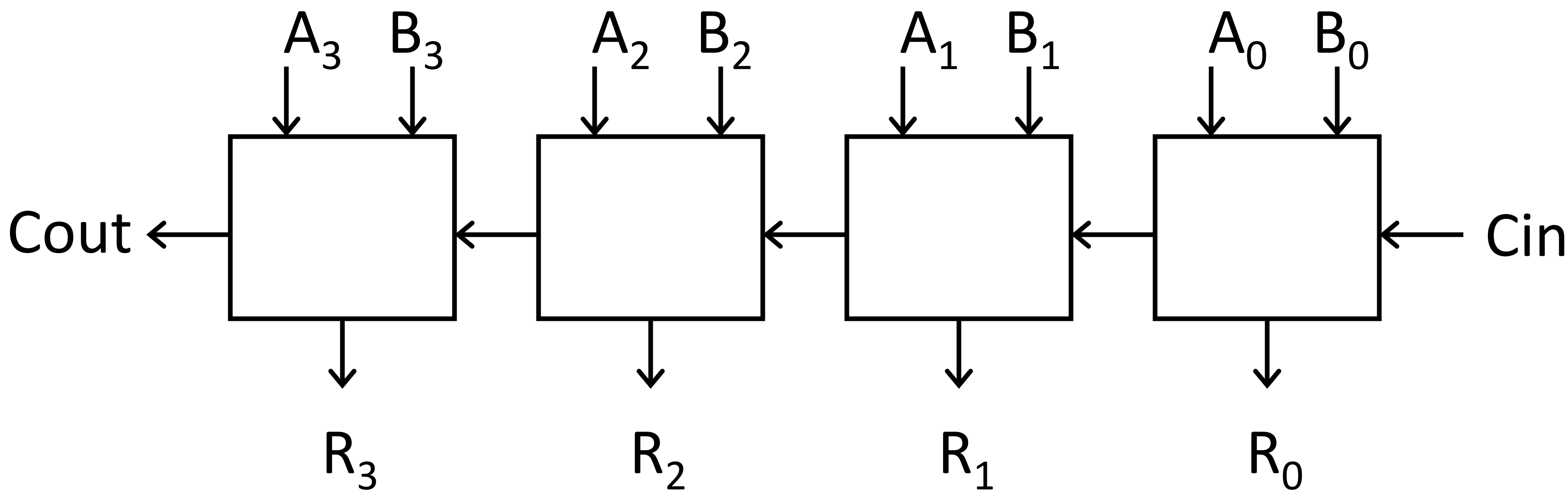
4-Bit Full Adder

- Adds two 4-bit numbers and carry in
- Computes 4-bit result and carry out
- Can be cascaded

4-bit Adder



4-bit Adder



- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out

Arithmetic with Negative Numbers

- Addition with negatives:
  - pos + pos   add magnitudes, result positive
  - neg + neg   add magnitudes, result negative
  - pos + neg   subtract smaller magnitude, keep sign of bigger magnitude

First Attempt: Sign/Magnitude Representation

- First Attempt: Sign/Magnitude Representation
- 1 bit for sign (0=positive, 1=negative)
- N-1 bits for magnitude

0 111 = 7

1 111 = -7

Two's Complement Representation

- Better: Two's Complement Representation
- Leading 1's for negative numbers
- To negate any number:
  - complement *all* the bits
  - then add 1

$$6 = 0110$$
$$\bar{6} = 1001$$
$$+1$$
$$-6 = 1010$$

$$20 = 0001\ 0100$$
$$\overline{20} = 1110\ 1011$$
$$+1$$
$$-20 = 1110\ 1100$$

Two's Complement

- Non-negatives Negatives
- (as usual): (two's complement: flip then add 1):
- +0 = 0000
- +1 = 0001 1110 -1 = 1111
- +2 = 0010 ~2 =
- +3 = 0011 ~3 = 1100 -3 = 1101
- +4 = 0100
- +5 = 0101
- +6 = 0110
- +7 = 0111 ~7 = 1000 -7 = 1001
- +8 = 1000 ~8 = 0111 -8 = 1000

Two's Complement Facts

- Signed two's complement
  - Negative numbers have leading 1's
  - zero is unique: +0 = - 0
  - wraps from largest positive to largest negative
- N bits can be used to represent
  - unsigned: $0 \ldots 2^N - 1$
    - – eg: 8 bits ⇒ $0 \ldots 2^8 - 1 = 255$
  - signed (two's complement):
    - – ex: 8 bits ⇒ $-128 \ldots 0 \ldots +127$

$-\left(\frac{2^N}{2}\right) \ldots \left(\frac{2^N}{2}\right) - 1$

Sign Extension & Truncation

- Extending to larger size

6 = 0110   0000 0110

-1 = 1111   1111 1111

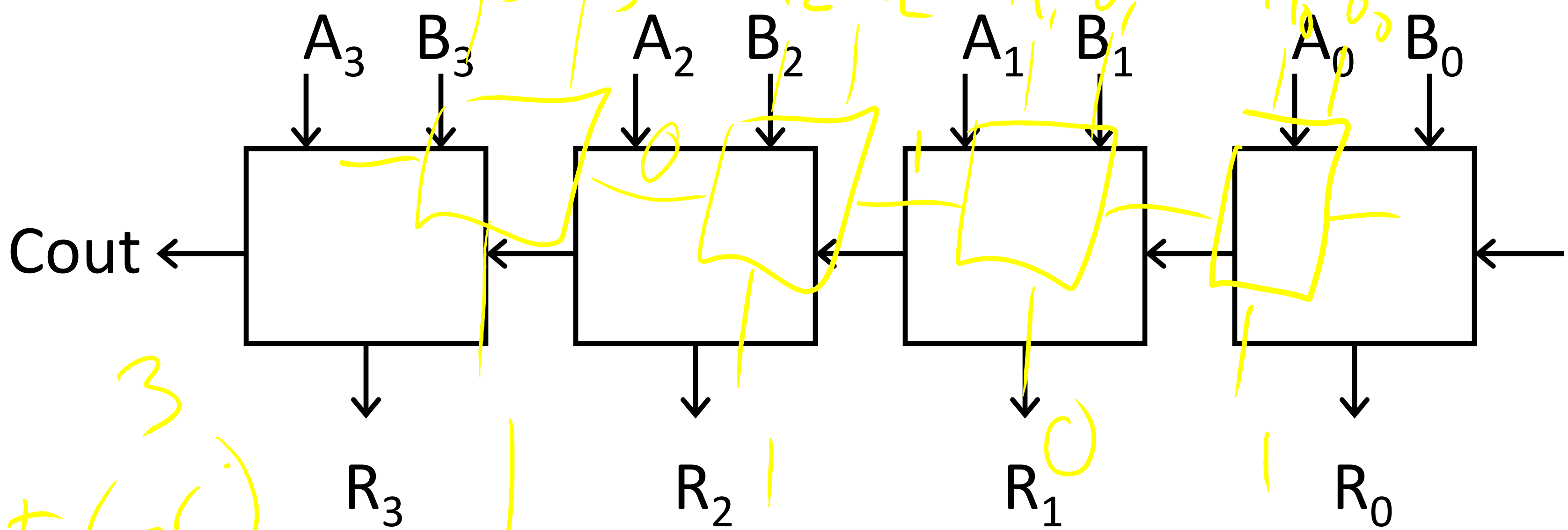
- Truncate to smaller size

0000 1111 = 15

0111 = 7

Two's Complement Addition

- Addition with two's complement signed numbers
- Perform addition as usual, regardless of sign (it just works)



Diversion: 10's Complement

- How does that work?

$$\begin{array}{r} -154 \\ \underline{+283} \end{array}$$

Overflow

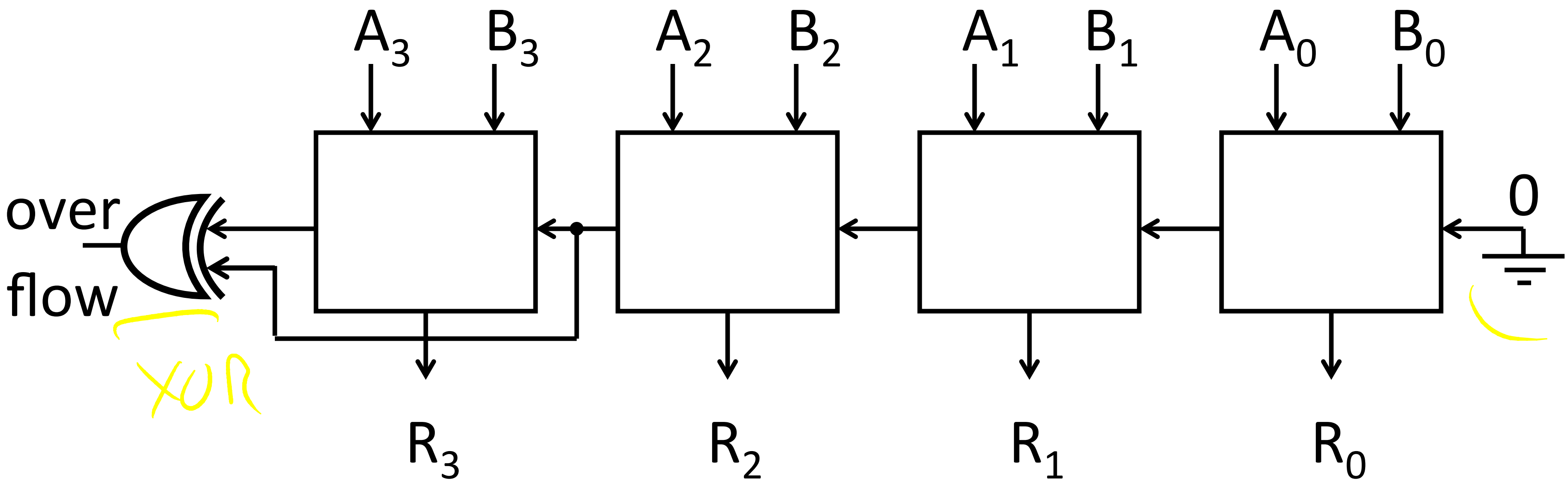
- Overflow
  - adding a negative and a positive?
  - adding two positives?
  - adding two negatives?
- Rule of thumb:

carry out msb != carry in msb

- Overflow happened iff
  carry into msb != carry out of msb

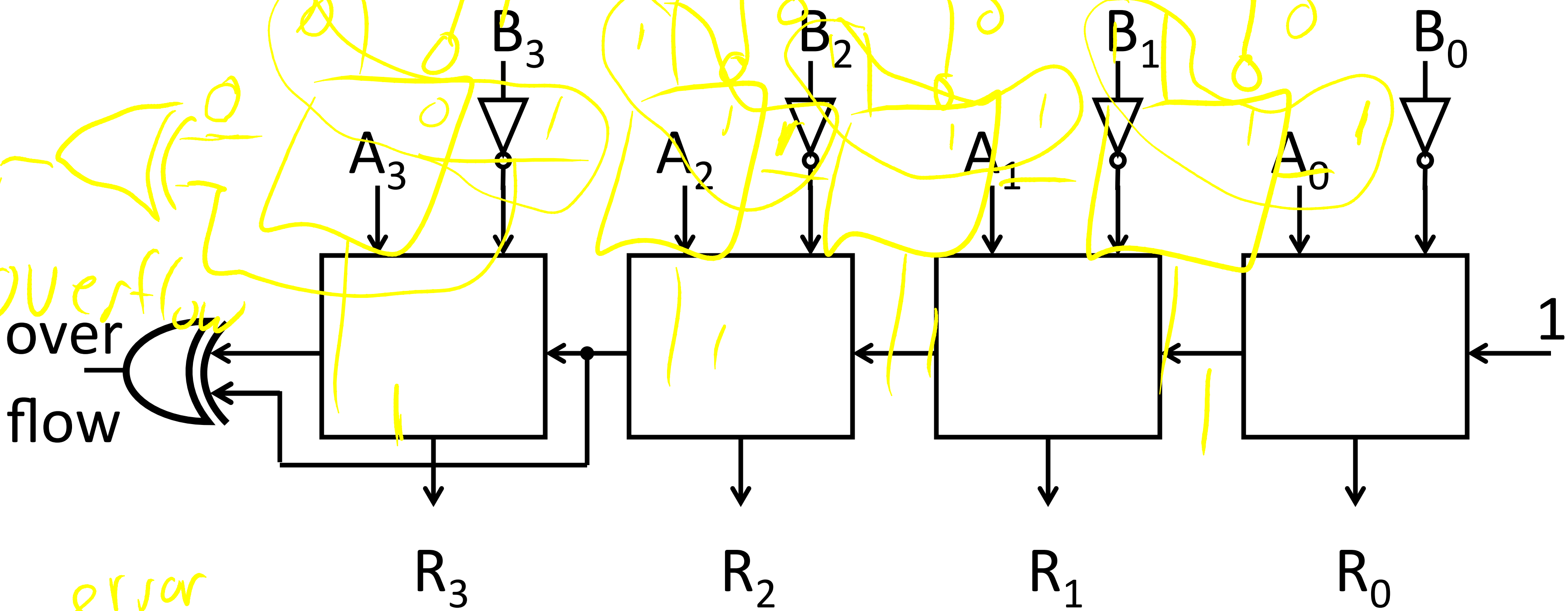
Two's Complement Adder

- Two's Complement Adder with overflow detection



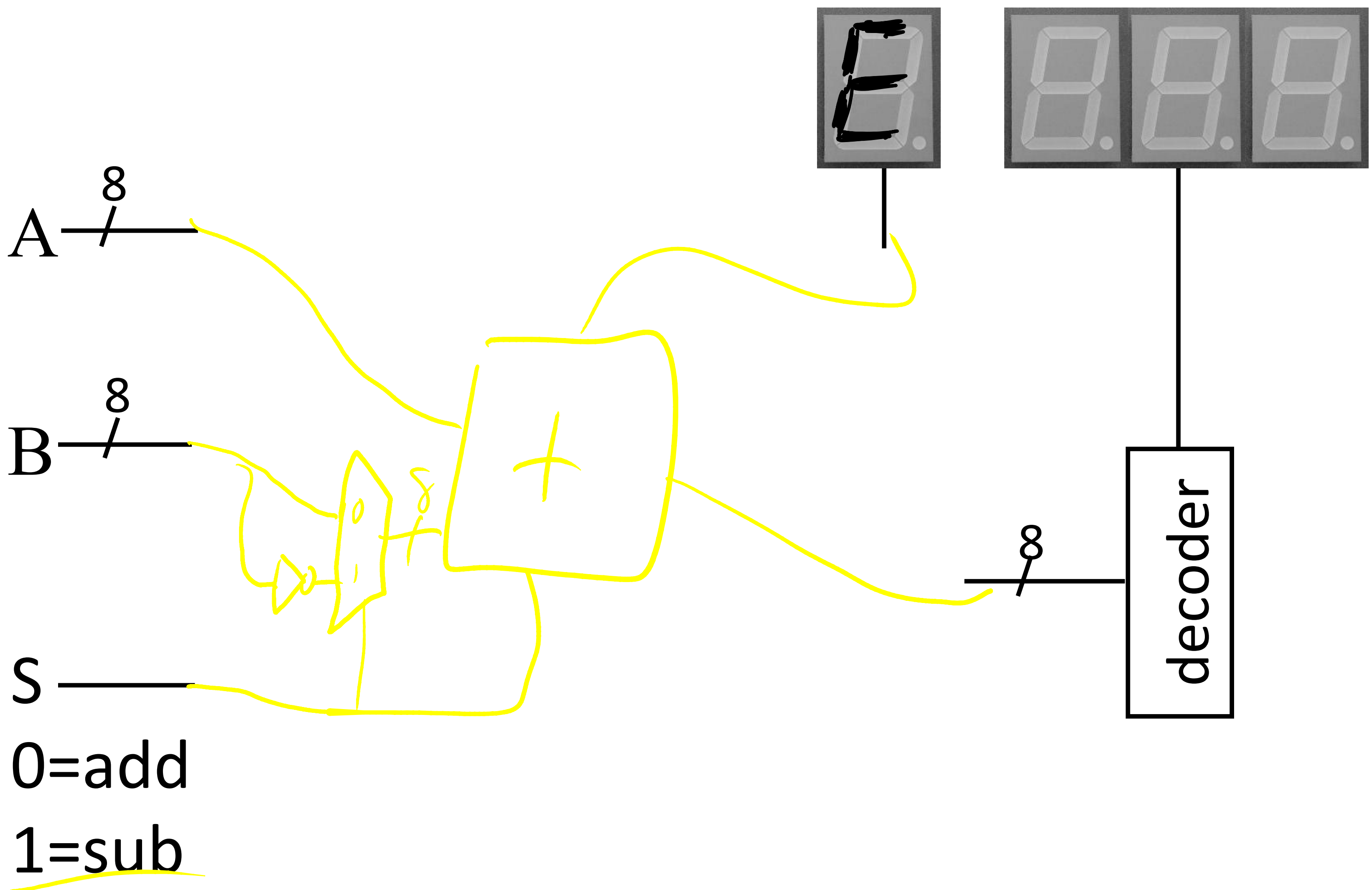
Binary Subtraction

- Two's Complement Subtraction
- Lazy approach
- $A - B = A + (-B) = A + (\bar{B} + 1)$

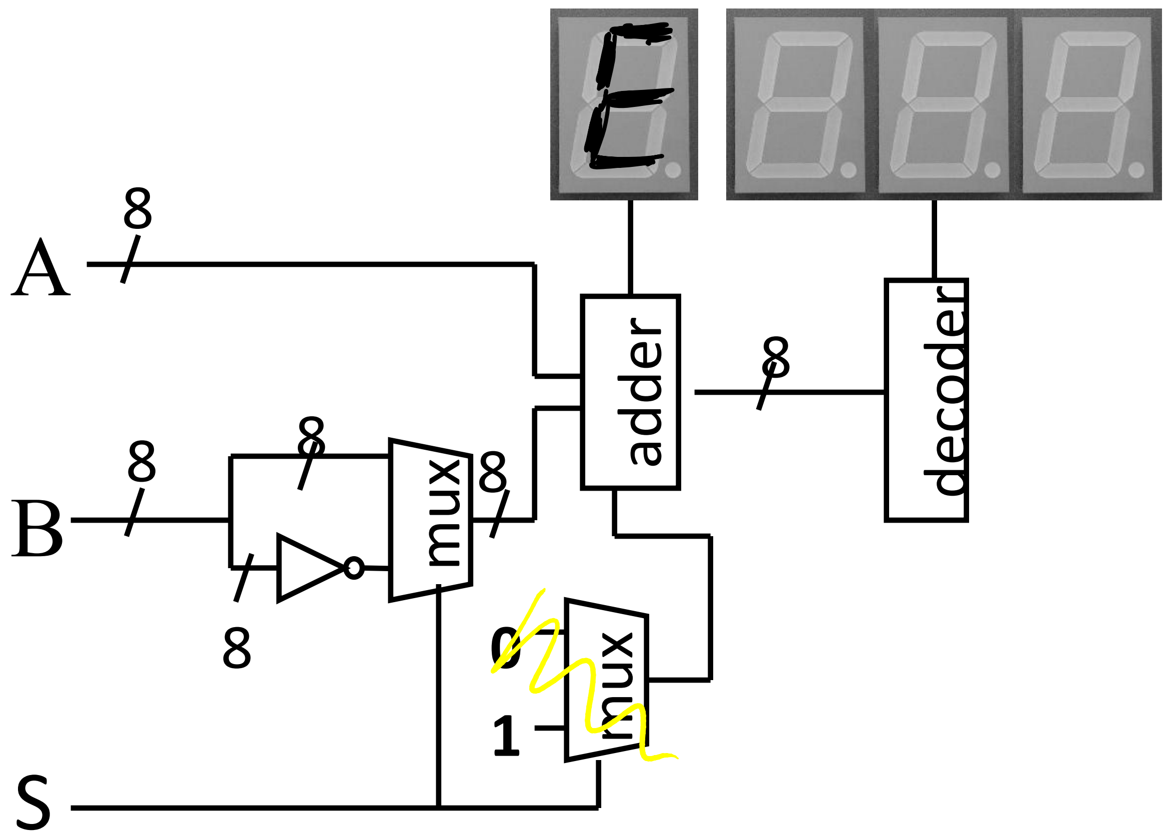


Q: What if (-B) overflows?

A Calculator

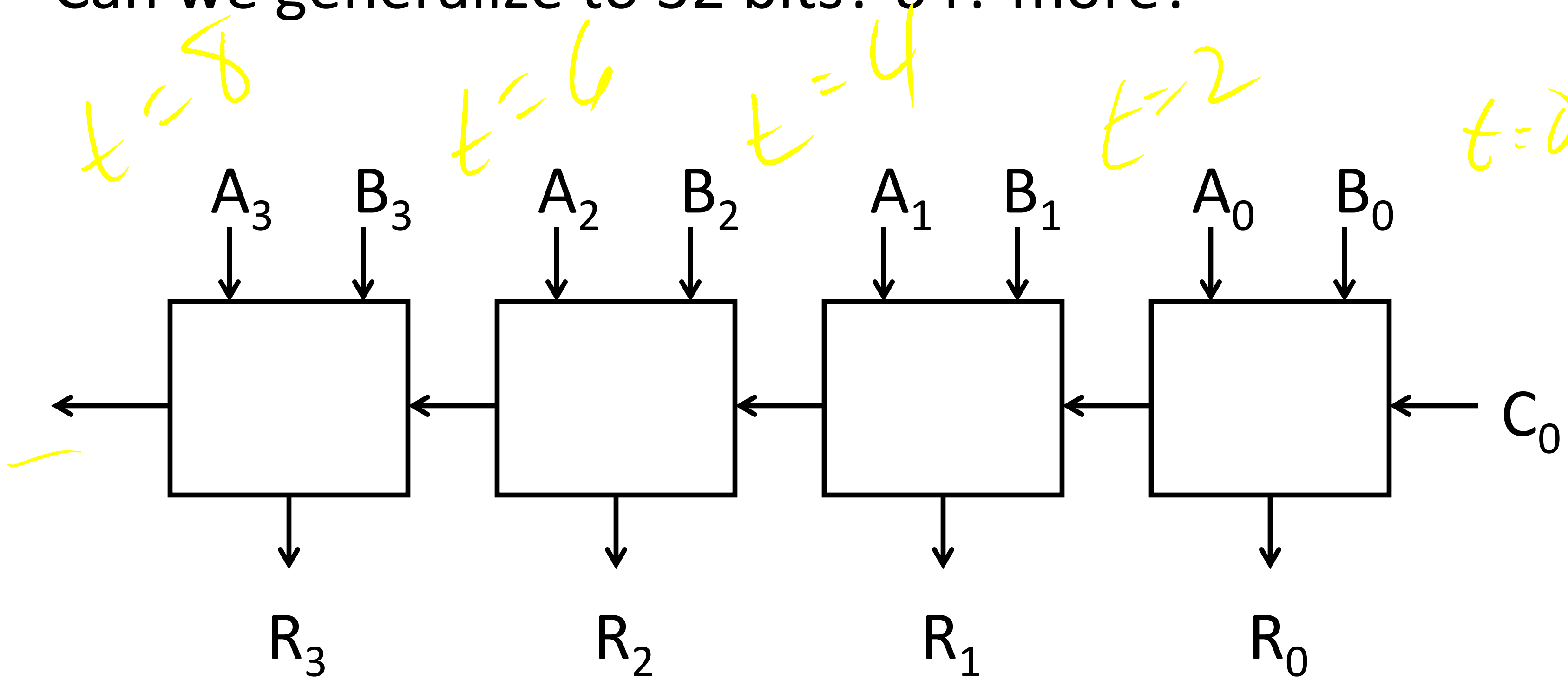


A Calculator



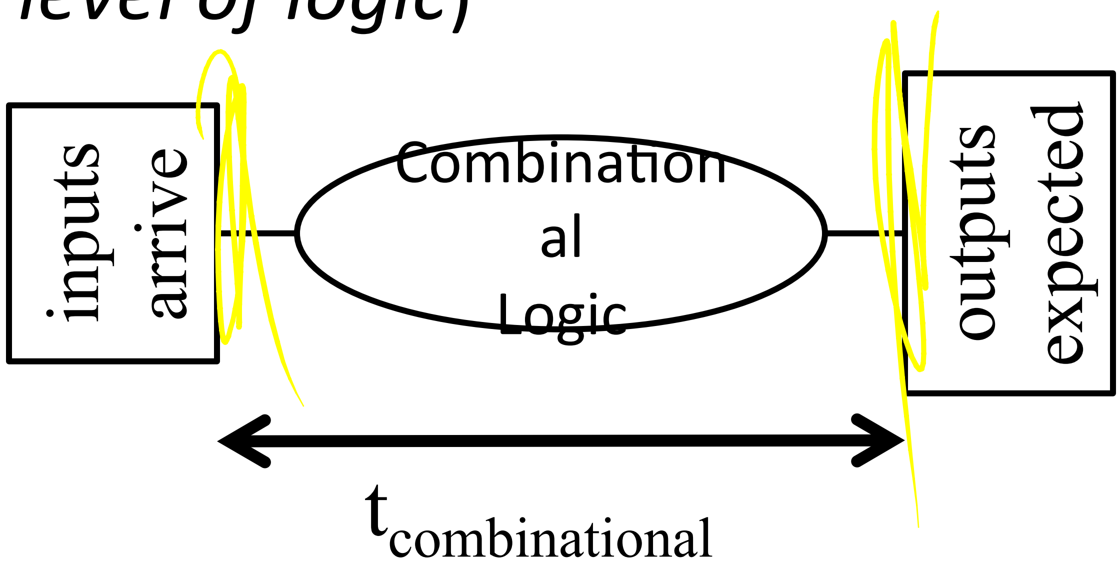
Efficiency and Generality

- Is this design fast enough?
- Can we generalize to 32 bits? 64? more?

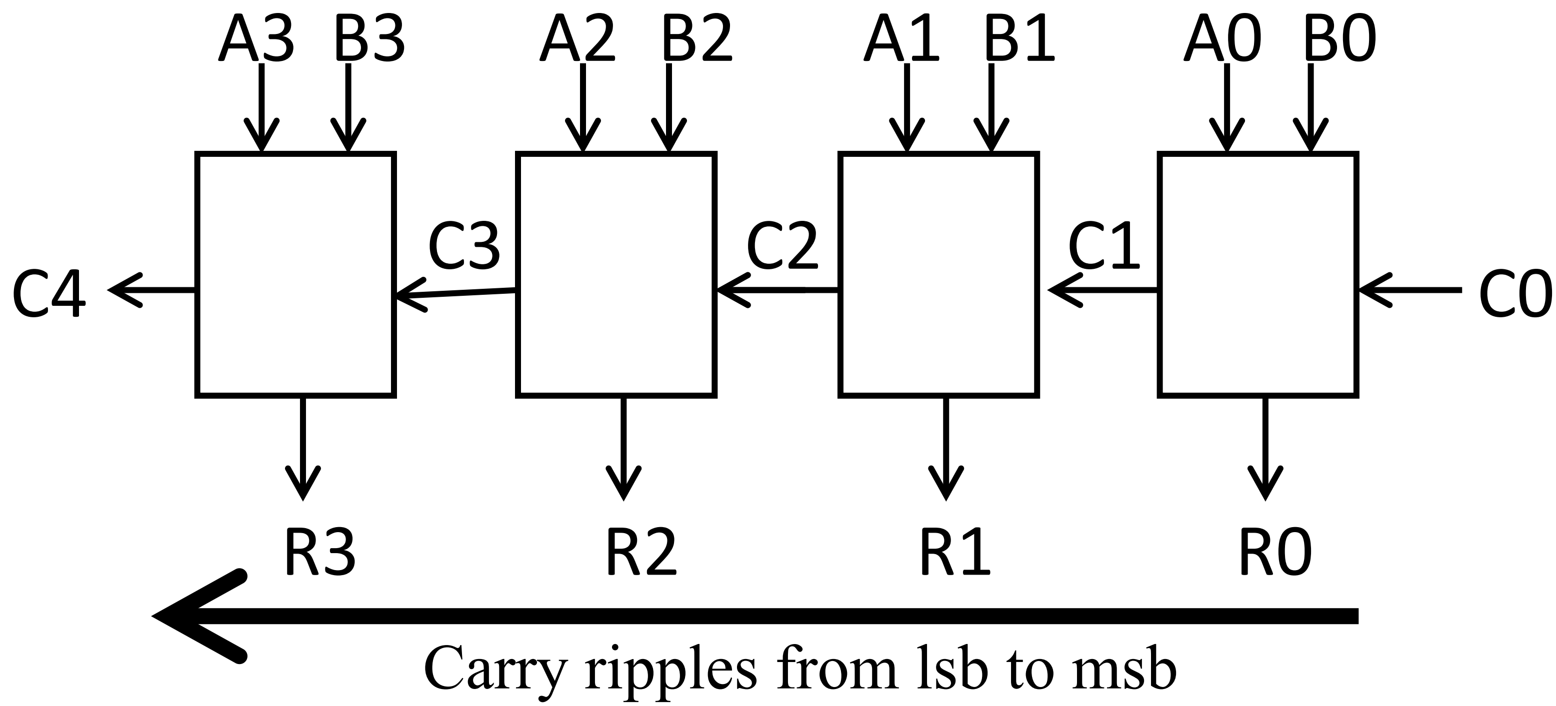


Performance

- Speed of a circuit is affected by the number of gates in series (on the *critical path* or the *deepest level of logic*)



4-bit Ripple Carry Adder



- First full adder, 2 gate delay
- Second full adder, 2 gate delay
- …

Summary

- We can now implement any combinational (combinatorial) logic circuit
  - Decompose large circuit into manageable blocks
    - Encoders, Decoders, Multiplexors, Adders, ...
  - Design each block
    - Binary encoded numbers for compactness
  - Can implement circuits using NAND or NOR gates
  - Can implement gates using use P- and N-transistors
  - And can add and subtract numbers (in two's compliment)!
  - Next time, state and finite state machines…