Numbers & Arithmetic

Hakim Weatherspoon
CS 3410, Spring 2012
Computer Science
Cornell University

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

Goals for today

Today

- Review Logic Minimization

- Build a circuit (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

Logic Minimization

- How to implement a desired function?

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

Minimization with Karnaugh maps (1)

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

◆ Sum of minterms yields
- $\overline{a}\,\overline{b}c + \overline{a}bc + a\overline{b}\,\overline{c} + a\overline{b}c$

Minimization with Karnaugh maps (2)

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

- ◆ Sum of minterms yields
  - $\overline{a}\overline{b}c + \overline{a}bc + a\overline{b}\overline{c} + a\overline{b}c$

- ◆ Karnaugh maps identify which inputs are (ir)relevant to the output

c\ab

	00	01	11	10
0	0	0	0	1
1	1	1	0	1

Minimization with Karnaugh maps (2)

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

- ◆ Sum of minterms yields
  - $\overline{a}\overline{b}c + \overline{a}bc + a\overline{b}\overline{c} + a\overline{b}c$

- ◆ Karnaugh map minimization
  - Cover all 1's
  - Group adjacent blocks of 2^n 1's that yield a rectangular shape
  - Encode the common features of the rectangle
    - ◆ out = $a\overline{b} + \overline{a}c$

c \ ab

	00	01	11	10
0	0	0	0	1
1	1	1	0	1

Karnaugh Minimization Tricks (1)

c \ ab

	00	01	11	10
0	0	1	1	1
1	0	0	1	0

c \ ab

	00	01	11	10
0	1	1	1	1
1	0	0	1	0

Karnaugh Minimization Tricks (1)

◆ Minterms can overlap
- out = $b\bar{c}$ + $a\bar{c}$ + ab

◆ Minterms can span 2, 4, 8 or more cells
- out = $\bar{c}$ + ab

Karnaugh Minimization Tricks (2)

ab
cd

	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

ab
cd

	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

Karnaugh Minimization Tricks (2)

ab
cd   00 01 11 10

	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

- The map wraps around
  - out = $\overline{b}d$

ab
cd   00 01 11 10

	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

  - out = $\overline{b}\,\overline{d}$

Karnaugh Minimization Tricks (3)

ab
cd

	00	01	11	10
00	0	0	0	0
01	1	x	x	x
11	1	x	x	1
10	0	0	0	0

ab
cd

	00	01	11	10
00	1	0	0	x
01	0	x	x	0
11	0	x	x	0
10	1	0	0	1

Karnaugh Minimization Tricks (3)

ab
cd   00 01 11 10

	00	01	11	10
00	0	0	0	0
01	1	x	x	x
11	1	x	x	1
10	0	0	0	0

- "Don't care" values can be interpreted individually in whatever way is convenient
  - assume all x's = 1
  - out = d

ab
cd   00 01 11 10

	00	01	11	10
00	1	0	0	x
01	0	x	x	0
11	0	x	x	0
10	1	0	0	1

  - assume middle x's = 0
  - assume 4th column x = 1
  - out = $\overline{bd}$

Multiplexer
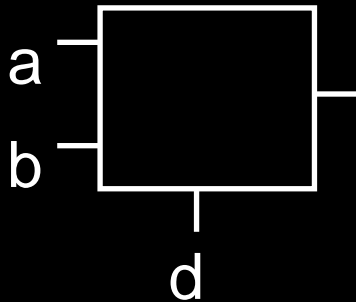
- A multiplexer selects between multiple inputs
  - out = a, if d = 0
  - out = b, if d = 1

- Build truth table
- Minimize diagram
- Derive logic diagram

Multiplexer Implementation

a

b

d
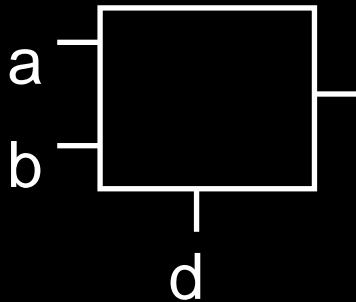
- Build a truth table

$$= abd + ab\overline{d} + \overline{a}bd + a\overline{b}\,\overline{d}$$

a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Multiplexer Implementation

a

b

d
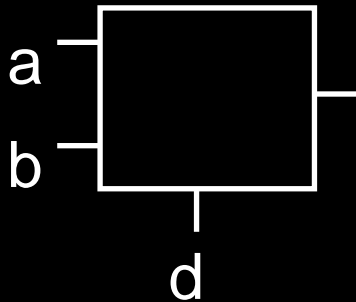
- Build the Karnaugh map

a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

d \ ab

	00	01	11	10
0	0	0	1	1
1	0	1	1	0

Multiplexer Implementation
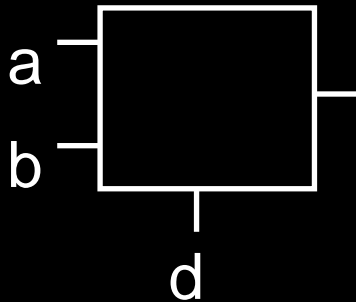
a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Derive Minimal Logic Equation

d ab

	00	01	11	10
0	0	0	1	1
1	0	1	1	0

- out = $a\bar{d} + bd$

Multiplexer Implementation
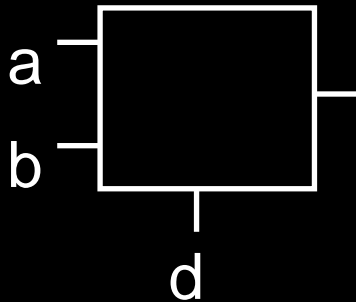
a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Derive Minimal Logic Equation

d \ ab	00	01	11	10
0	0	0	1	1
1	0	1	1	0

- out = $a\bar{d}$ + bd
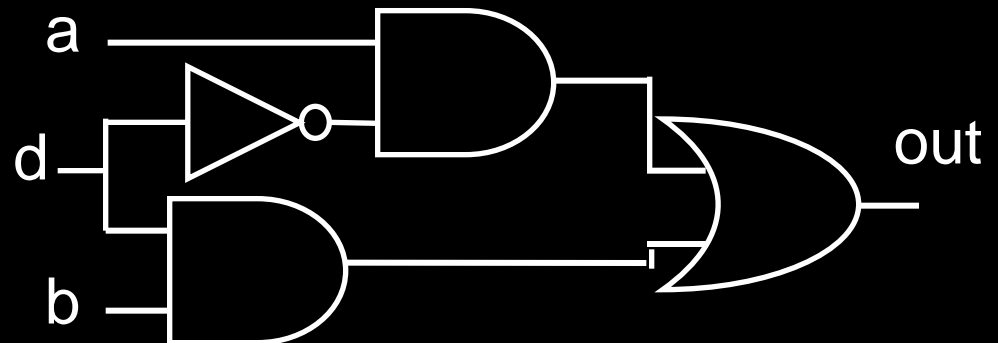
Logic Gates
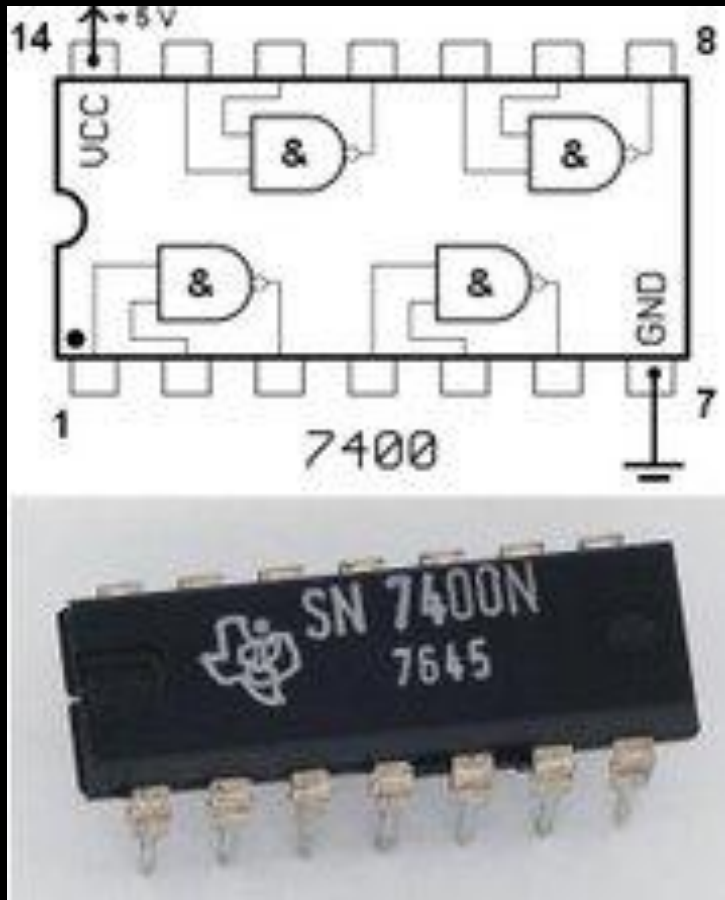
- One can buy gates separately
  - ex. 74xxx series of integrated circuits
  - cost ~$1 per chip, mostly for packaging and testing

- Cumbersome, but possible to build devices using gates put together manually

Integrated Circuits
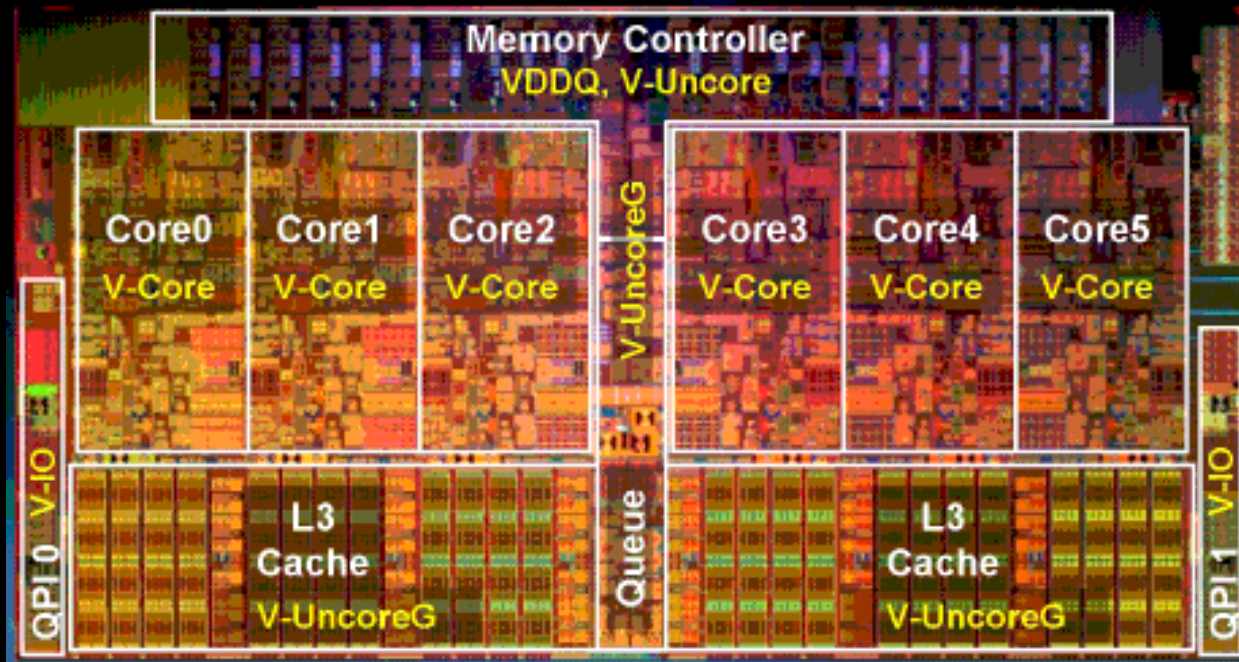
- Or one can manufacture a complete design using a custom mask
- Intel Westmere has approximately 1.17 billion transistors

Recap

- We can now implement any logic circuit
  - Can do it efficiently, using Karnaugh maps to find the minimal terms required
  - Can use either NAND or NOR gates to implement the logic circuit
  - Can use P- and N-transistors to implement NAND or NOR gates

Voting machine

- Lets build something interesting

- A voting machine

- Assume:
  - A vote is recorded on a piece of paper,
  - by punching out a hole,
  - there are at most 7 choices
  - we will not worry about "hanging chads" or "invalids"

Voting machine

- For now, let's just display the numerical identifier to the ballot supervisor
  - we won't do counting yet, just decoding
  - we can use four photo-sensitive transistors to find out which hole is punched out

- A photo-sensitive transistor detects the presence of light
- Photo-sensitive material triggers the gate

Ballot Reading

Ballots

The 3410 optical scan vote ~~counter~~ reader machine
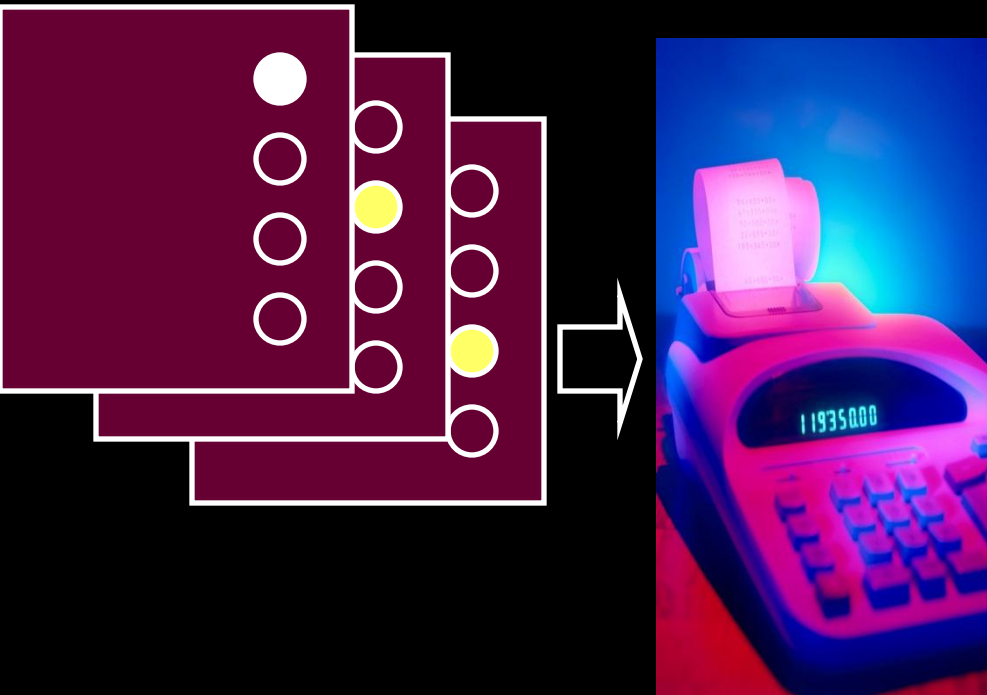
– **Input:** paper with a hole in it

– **Output:** number the ballot supervisor can record

Input

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

Vdd

i0
i1
i2
i3
i4
i5
i6
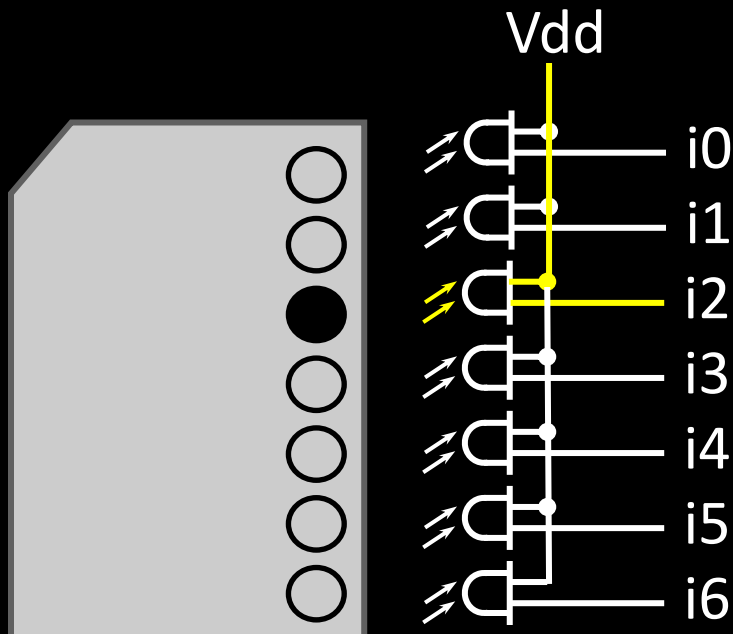
- ## Use array of N sensors for voting machine input

Output

- **7-Segment LED**

  - photons emitted when electrons fall into holes

d7 d6 ⏚ d5 d4

d3 d2 ⏚ d1 d0

Block Diagram

detect

N

8

Encoders

encoder

0
1
2
3
4
5
6
7
...
N
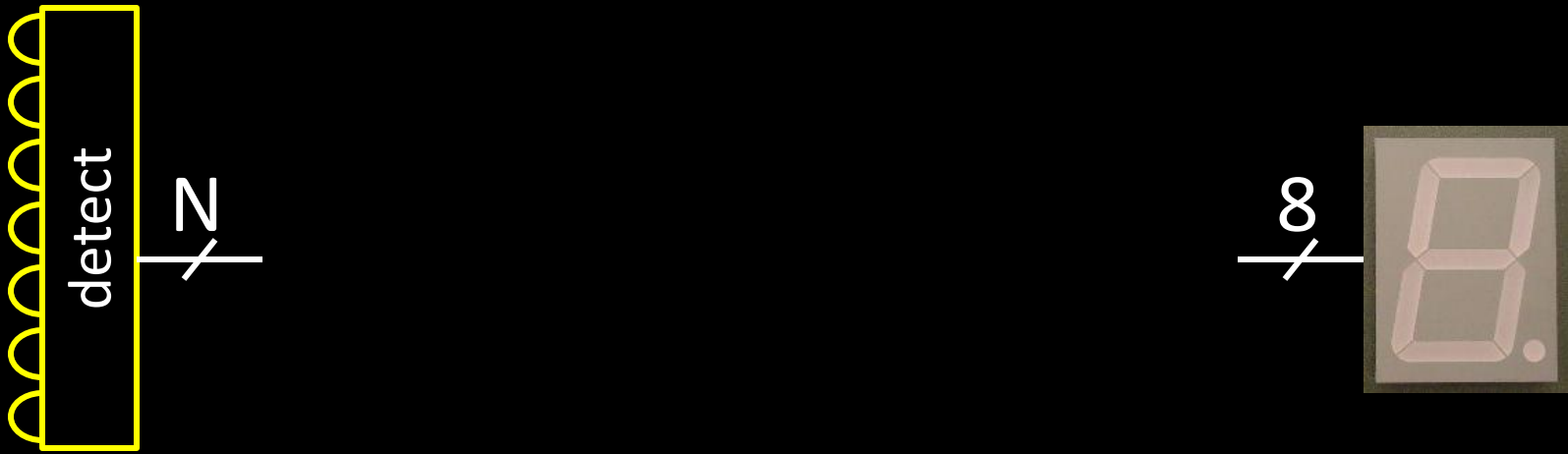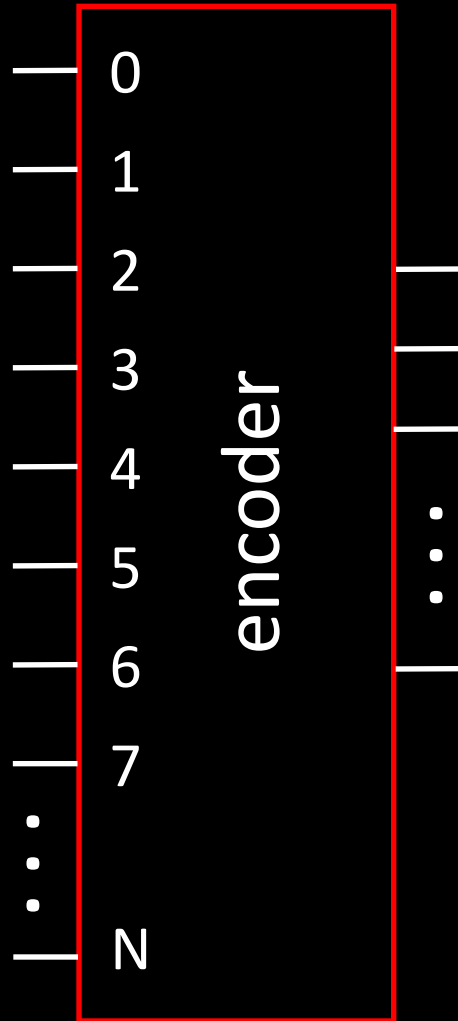
- N might be large
- Routing wires is expensive

- More efficient encoding?

Number Representations

- Base 10 - Decimal

$$6\ 3\ 7$$

$$10^2\ 10^1\ 10^0$$

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

Counting

- Counting

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

Base Conversion

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

Hexadecimal, Binary, Octal Conversions

Encoder Truth Table

a — 1

b — 2

c — 3

d — 4

o_0

o_1

o_2
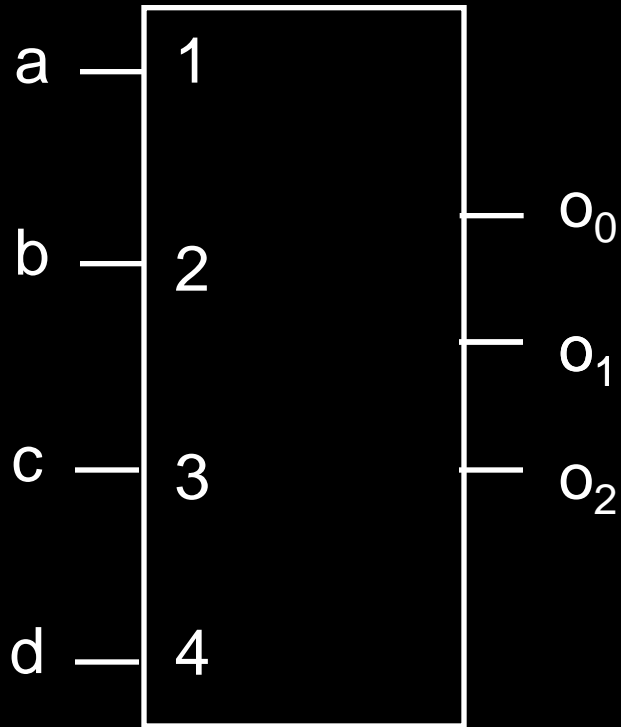
A 3-bit encoder

with 4 inputs for simplicity

Encoder Truth Table
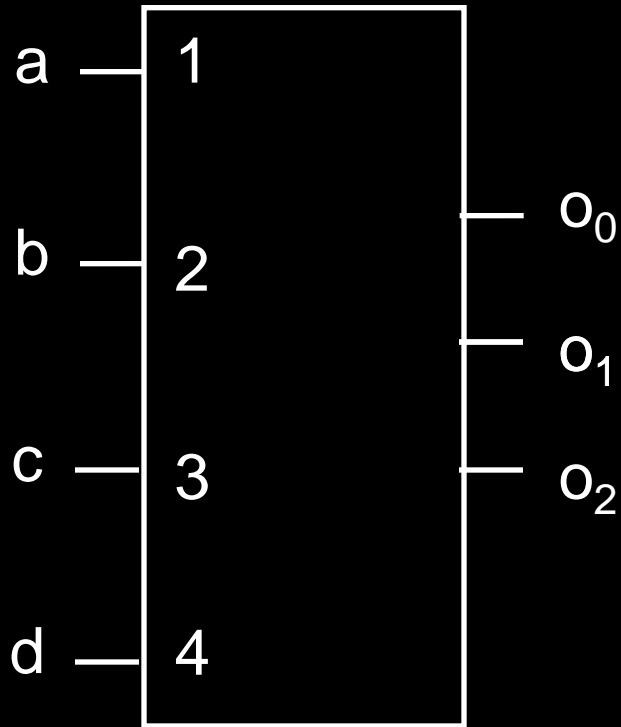
a	b	c	d		o2	o1	o0
0	0	0	0		0	0	0
1	0	0	0		0	0	1
0	1	0	0		0	1	0
0	0	1	0		0	1	1
0	0	0	1		1	0	0

a — 1

b — 2

c — 3

d — 4

o_0

o_1

o_2

A 3-bit encoder

with 4 inputs
for simplicity

- $o2 = \overline{abc}d$
- $o1 = \bar{a}b\bar{c}\bar{d} + \bar{a}\bar{b}c\bar{d}$
- $o0 = a\overline{bcd} + \bar{a}\bar{b}c\bar{d}$

Ballot Reading

detect

8

enc

3

8
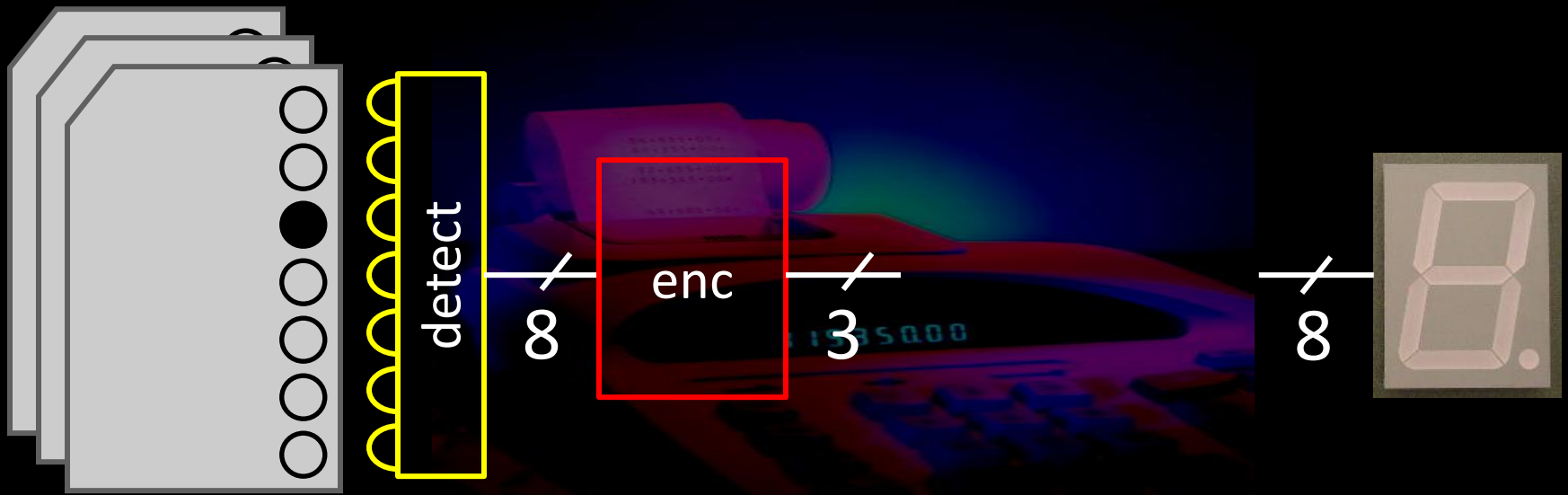
Ballot Reading
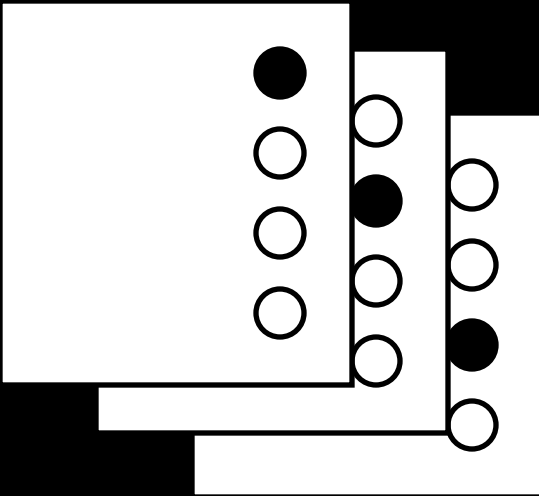
- Ok, we built first half of the machine

- Need to display the result

Ballots

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

7-Segment LED Decoder
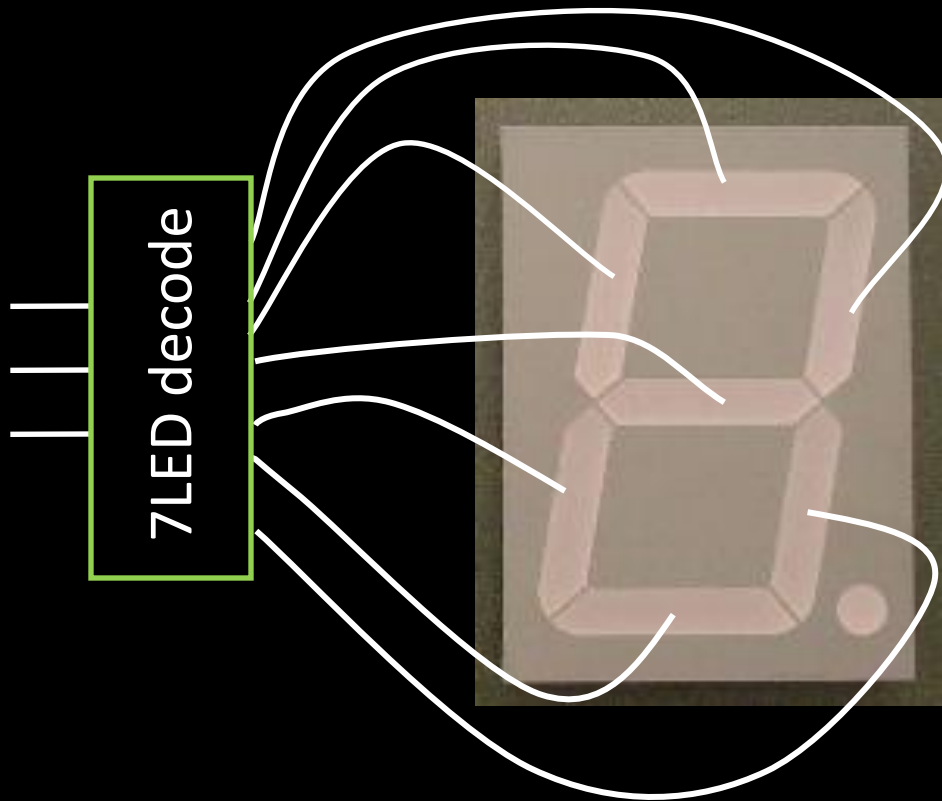
7LED decode

- 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	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
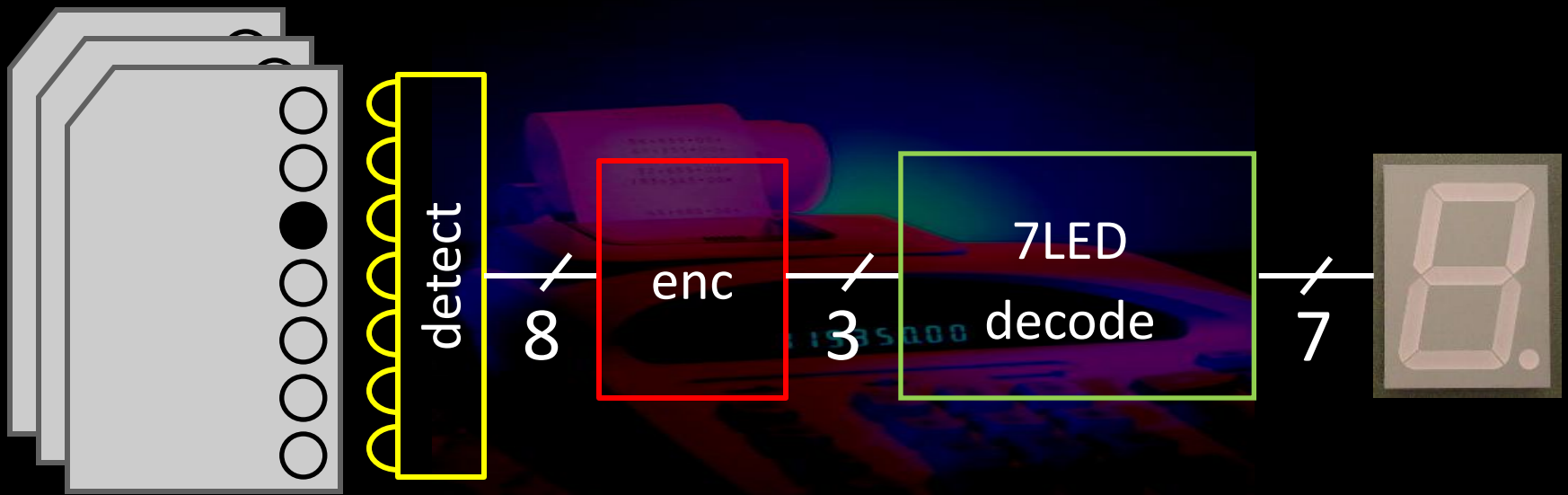
Ballots

detect

8

enc

3

7LED decode

7

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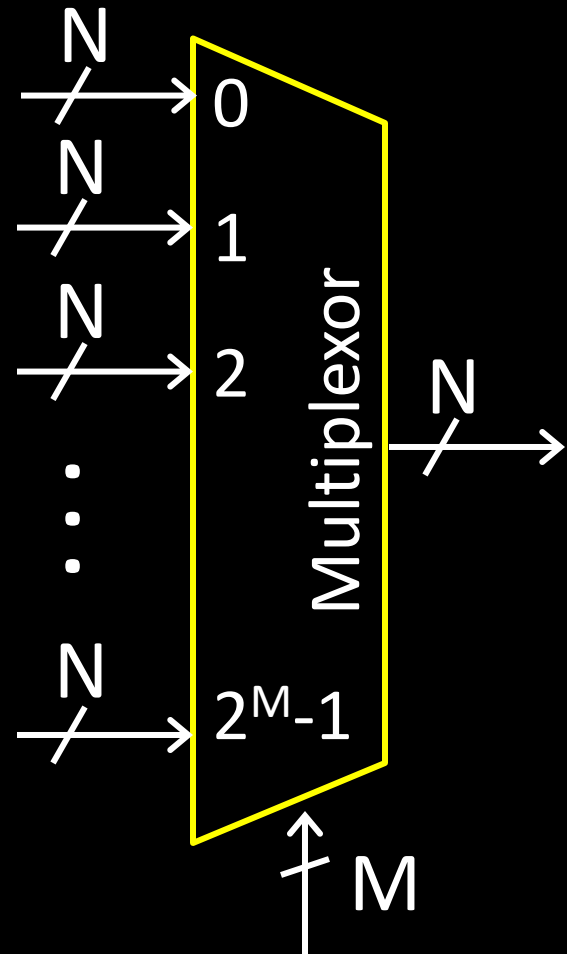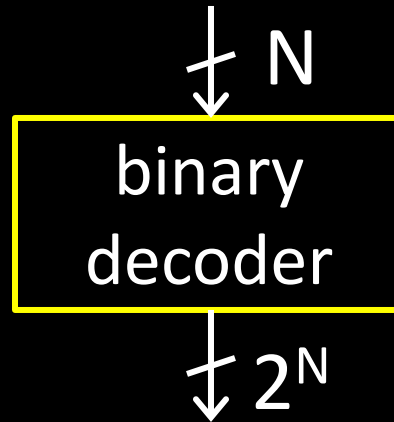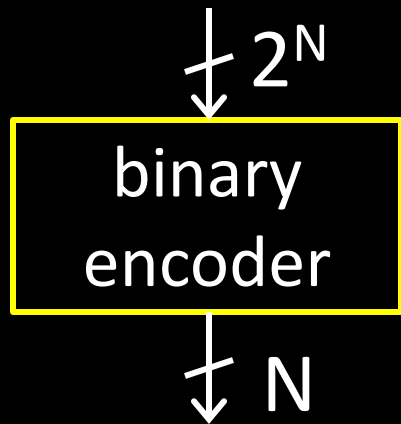
Administrivia

Make sure you are

- Registered for class, can access CMS
- Have a Section you can go to
- Have project partner in same Lab Section

Lab1 and HW1 are out

- Both due in one week, next Monday, start early
- Work alone
- But, use your resources
  - Lab Section, Piazza.com, Office Hours, Homework Help Session,
  - Class notes, book, Sections, CSUGLab

Homework Help Session

- Wednesday and Friday from 3:30-5:30pm
- Location: 203 Thurston

Administrivia

Check online syllabus/schedule

- http://www.cs.cornell.edu/Courses/CS3410/2012sp/schedule.html

- Slides and Reading for lectures

- Office Hours

- Homework and Programming Assignments

- Prelims (in evenings):
  - Tuesday, February 28th
  - Thursday, March 29th
  - April 26th

Schedule is subject to change

Binary Addition

183

+ 254
___

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

001110

+ 011100
_____

1-bit Adder

A   B

C

R
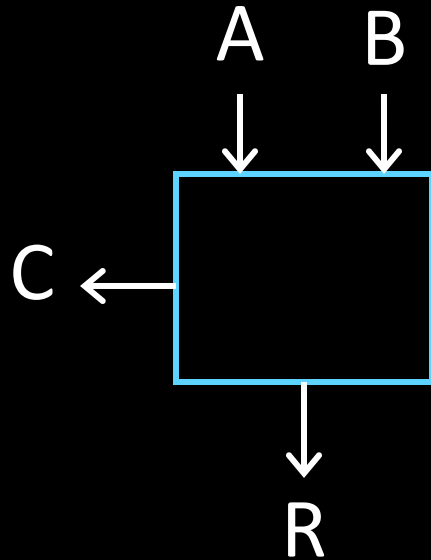
Half Adder

- Adds two 1-bit numbers

- Computes 1-bit result and 1-bit carry

1-bit Adder with Carry

A   B

Cout ← [ ] ← Cin

R
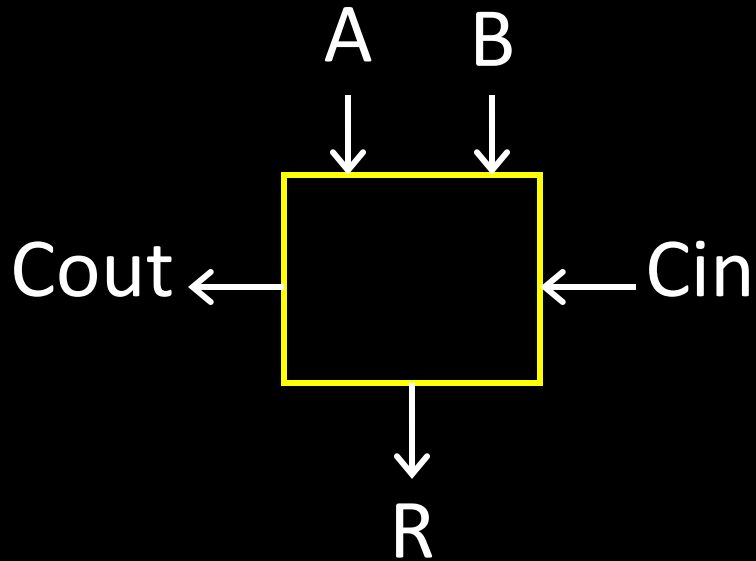
Full Adder

- Adds three 1-bit numbers

- Computes 1-bit result and 1-bit carry
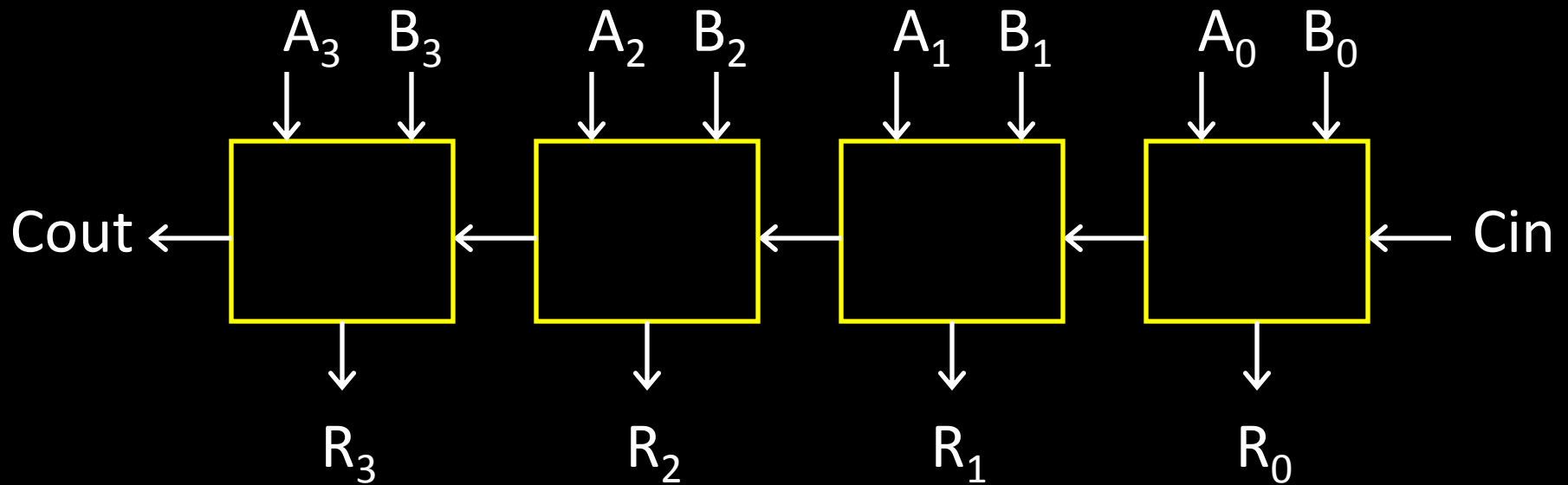
- Can be cascaded

4-bit Adder

A[4] B[4]

Cout ←⎯⎯ ⎯⎯← Cin

R[4]
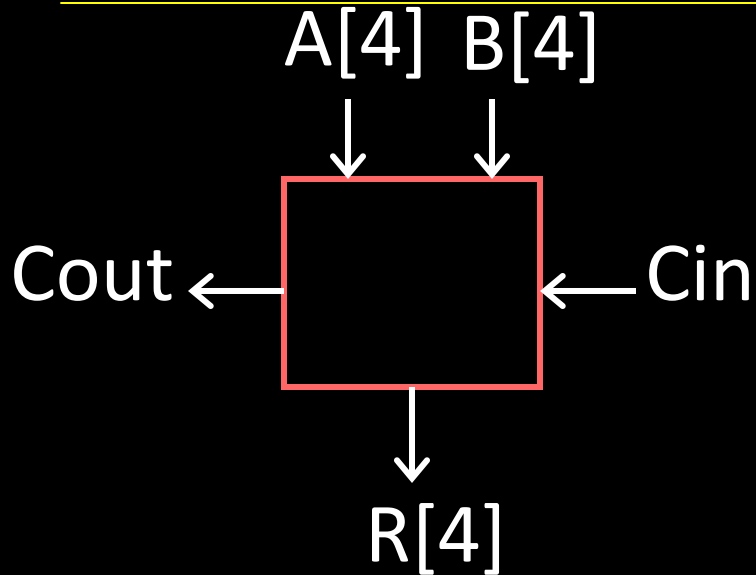
4-Bit Full Adder

- Adds two 4-bit numbers and carry in

- Computes 4-bit result and carry out

- Can be cascaded

4-bit Adder

A_3 B_3 A_2 B_2 A_1 B_1 A_0 B_0

Cout ←   ←   ←   ←   Cin

R_3 R_2 R_1 R_0
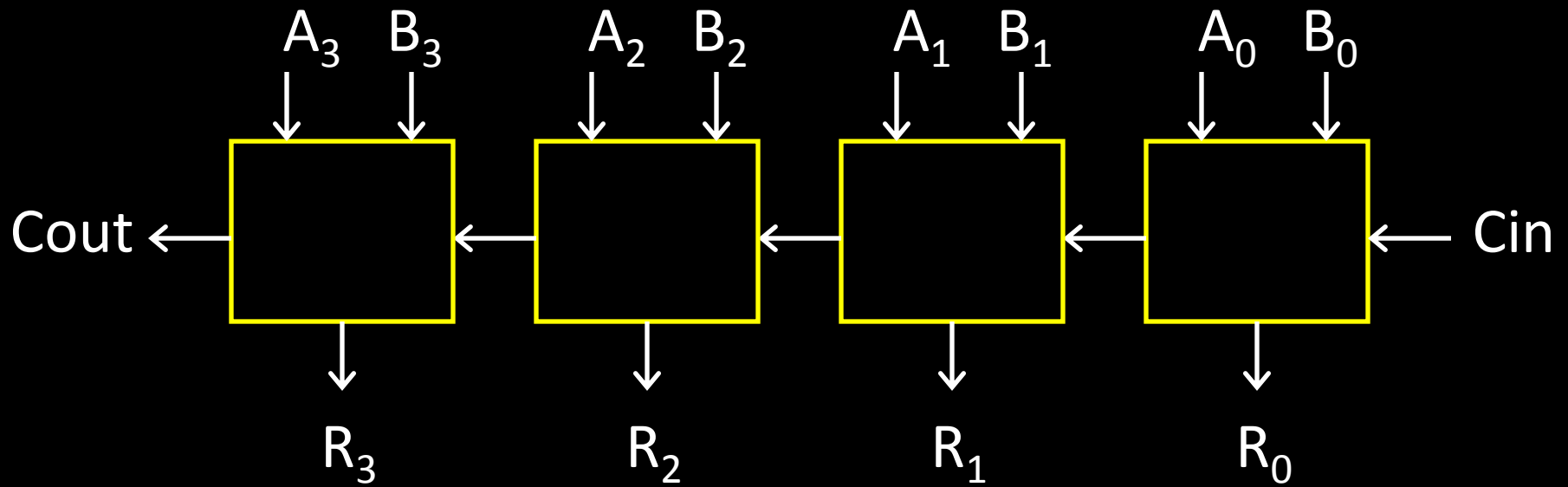
4-bit Adder

- 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

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

Two's Complement

- Non-negatives Negatives
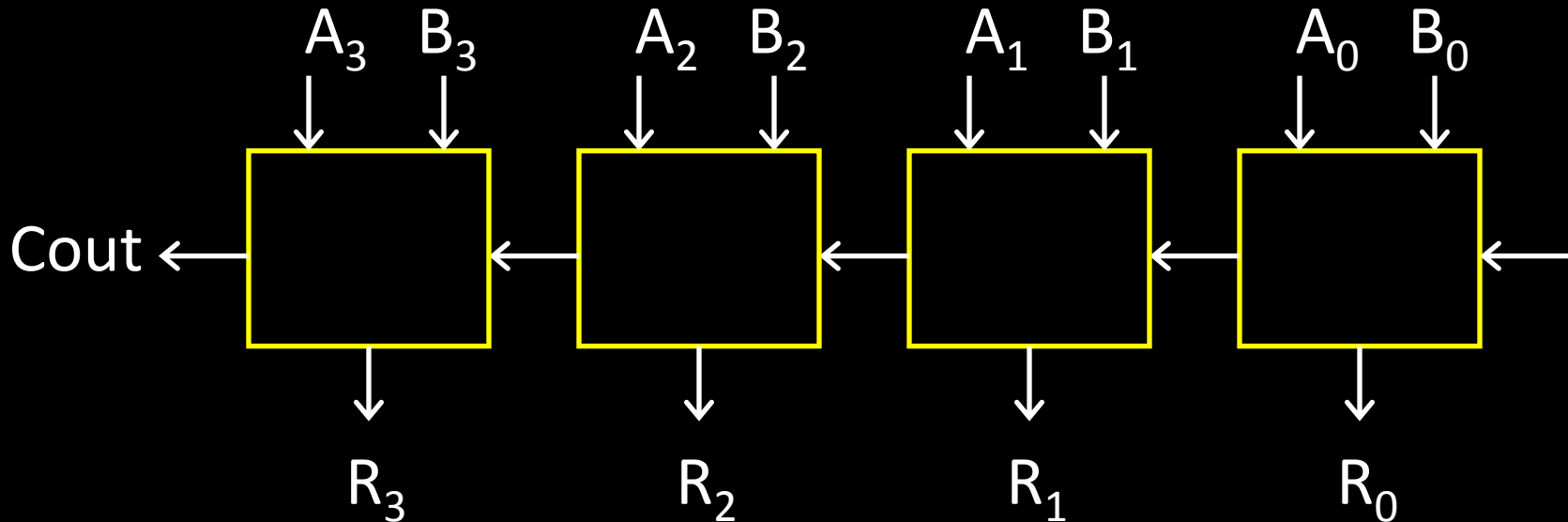- (as usual):   (two's complement: flip then add 1):

	Non-negatives	~	Negatives
	$+0 = 0000$	$\sim 0 = 1111$	$-0 = 0000$
	$+1 = 0001$	$\sim 1 = 1110$	$-1 = 1111$
	$+2 = 0010$	$\sim 2 = 1101$	$-2 = 1110$
	$+3 = 0011$	$\sim 3 = 1100$	$-3 = 1101$
	$+4 = 0100$	$\sim 4 = 1011$	$-4 = 1100$
	$+5 = 0101$	$\sim 5 = 1010$	$-5 = 1011$
	$+6 = 0110$	$\sim 3 = 1001$	$-6 = 1010$
	$+7 = 0111$	$\sim 7 = 1000$	$-7 = 1001$
	$+8 = 1000$	$\sim 8 = 0111$	$-8 = 1000$

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:
    - eg: 8 bits $\Rightarrow$
  - signed (two's complement):
    - ex: 8 bits $\Rightarrow$

Sign Extension & Truncation

- **Extending** to larger size

- **Truncate** to smaller size

Two's Complement Addition

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

$$A_3 \quad B_3 \qquad A_2 \quad B_2 \qquad A_1 \quad B_1 \qquad A_0 \quad B_0$$

Cout

$$R_3 \qquad\qquad R_2 \qquad\qquad R_1 \qquad\qquad R_0$$

- How does that work?

$$-154$$
$$\underline{+283}$$

Overflow

- **Overflow**
  - adding a negative and a positive?

  - adding two positives?

  - adding two negatives?

- **Rule of thumb:**
- Overflow happened iff carry into msb != carry out of msb

Two's Complement Adder

- Two's Complement Adder with overflow detection
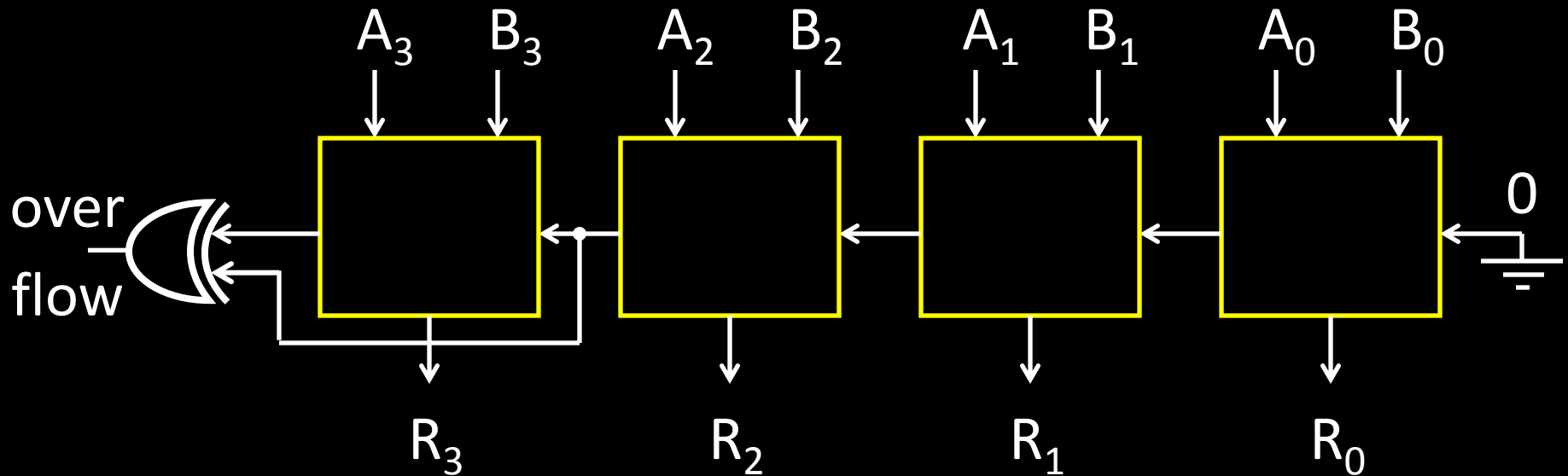
Binary Subtraction

- **Two's Complement Subtraction**

- Lazy approach ─

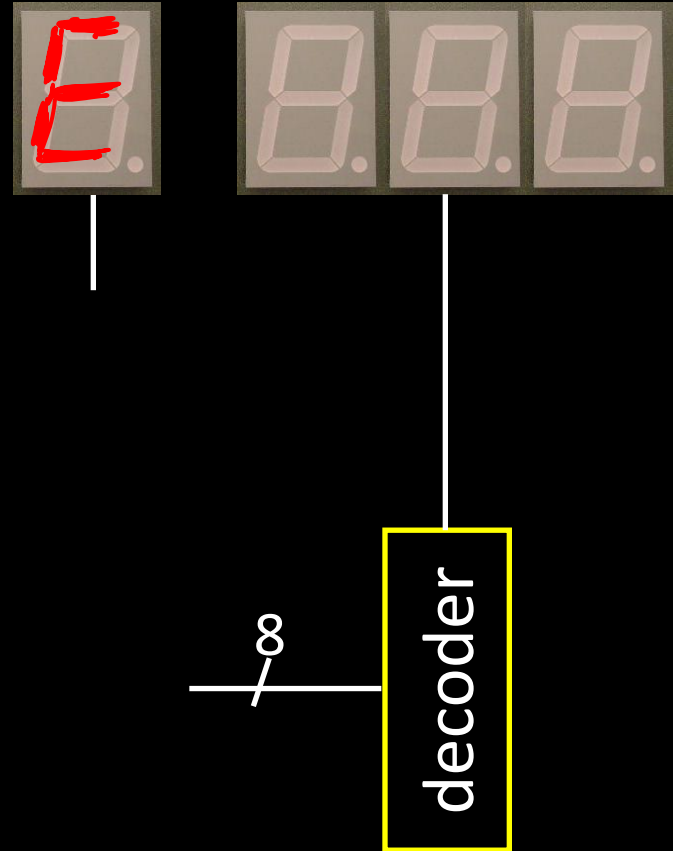- $A - B = A + (-B) = A + (B + 1)$
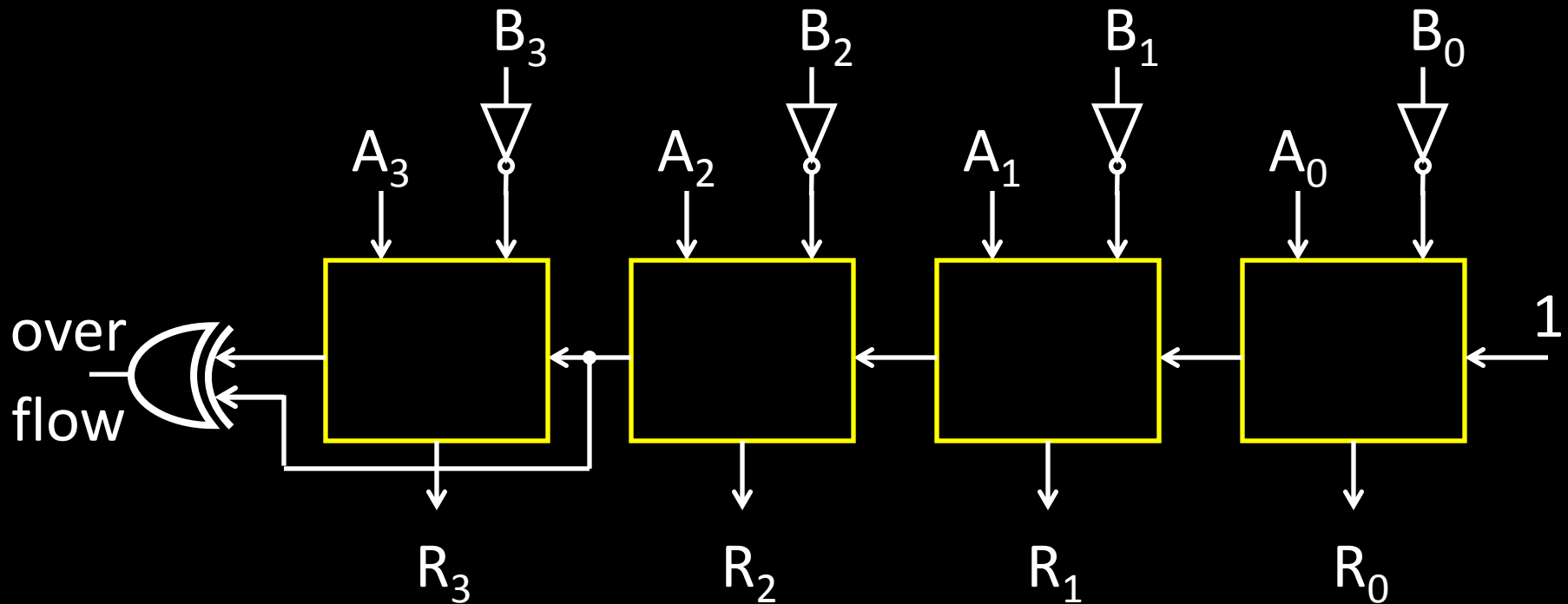
Q: What if (-B) overflows?

A Calculator

A —/— (8)

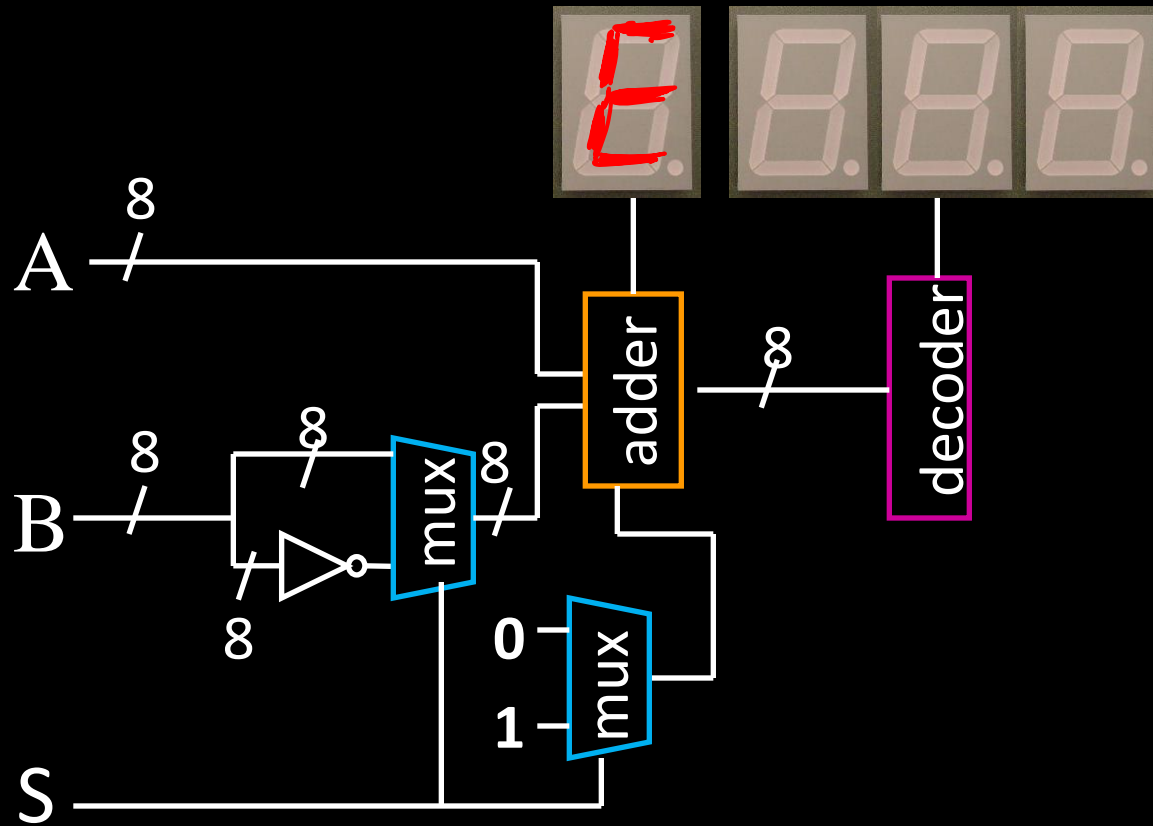B —/— (8)

S ——
0=add
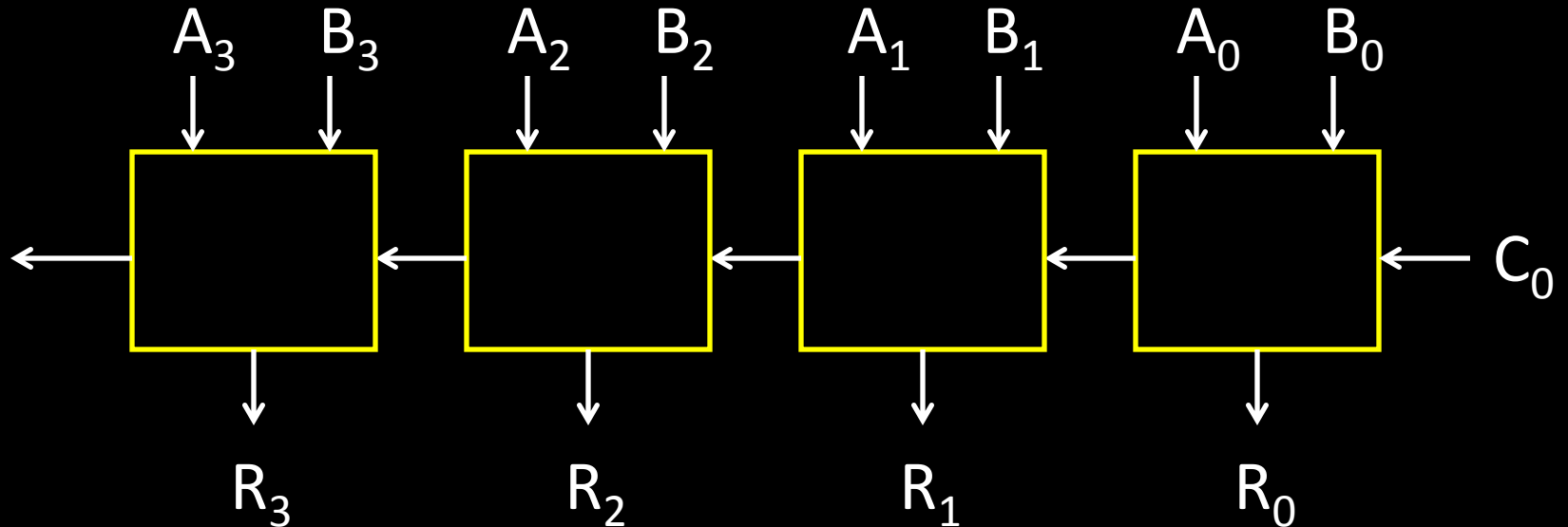1=sub

E (display: three additional digits)
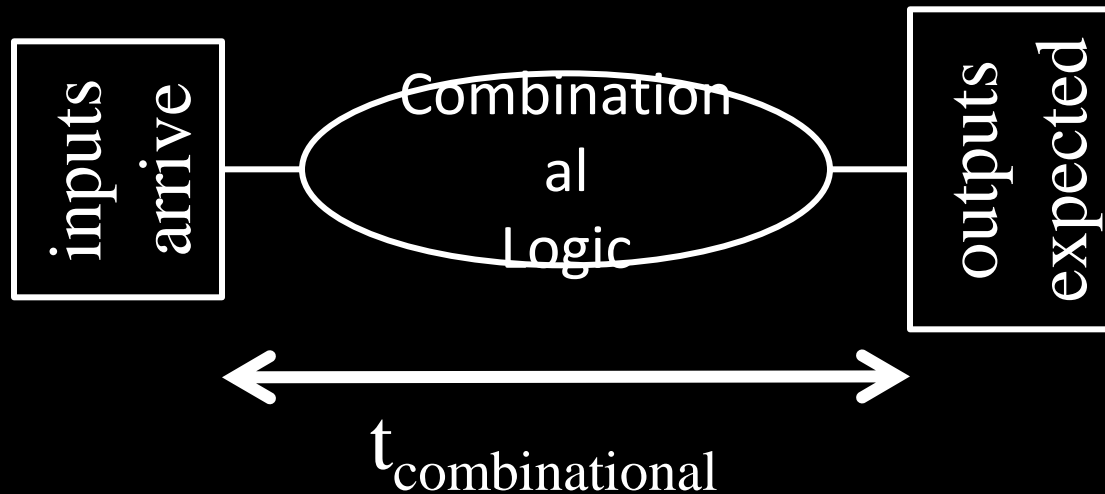
—/— (8) — decoder

Efficiency and Generality

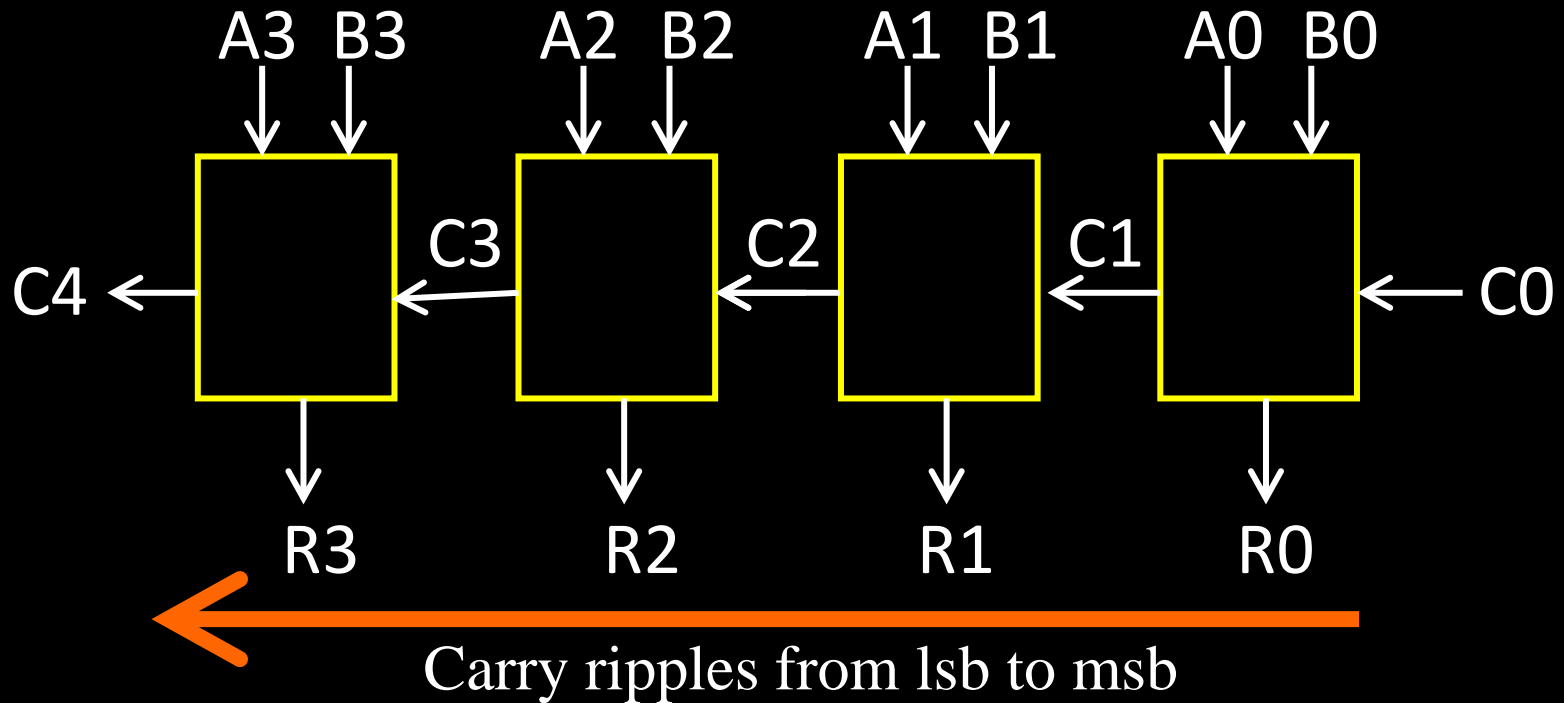- Is this design fast enough?
- Can we generalize to 32 bits? 64? more?

Performance

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

$$t_{combinational}$$

4-bit Ripple Carry Adder

A3 B3   A2 B2   A1 B1   A0 B0

C4 ← [ ] ← C3 ← [ ] ← C2 ← [ ] ← C1 ← [ ] ← C0

R3   R2   R1   R0

Carry ripples from lsb to msb

- 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…