CSE 260M / ESE 260
Intro. To Digital Logic & Computer Design

Bill Siever
&
Jim Feher

This week

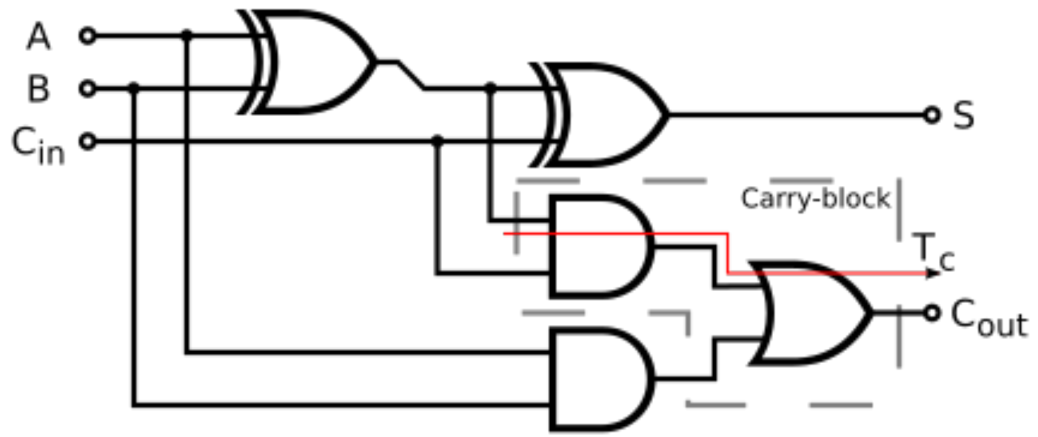
- Thursday:
  Studio — Here / Seigle 301
- Hw#5 posted was supposed to be posted by late Friday
  - Was *actually* Monday. Due Sunday at 11:59pm
  - Verilog!

Chapter 5 & 6

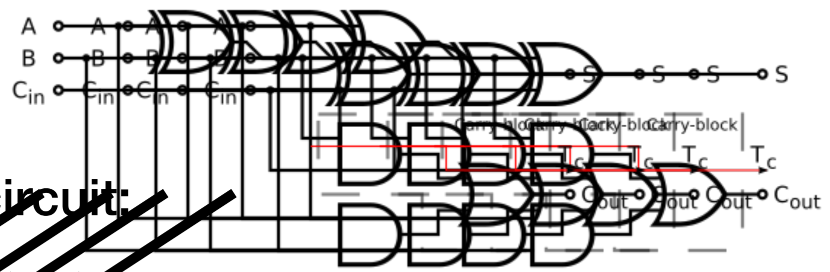
Ripple Adder

- Example: 1111 + 0001
    - As a traditional math problem:

```
   1 1 1 1
 + 0 0 0 1
 ― ― ― ― ― ―
```



Info in circuit



- As values in a circuit:

```
   0  0  0  0     <= Carry i-1 connected to carry in of i
   1  1  1  1     <= "A"
+  0  0  0  1     <= "B"
 ─ ─ ─ ─ ─ ─ ─
   0  0  0  0     <= "Sum"
```

Info in circuit: Initial

- As values in a circuit:

```
    0  0  0  0      <= Carry i-1 connected to carry in of i
    1  1  1  1      <= “A”
 +  0  0  0  1      <= “B”
 — — — — — —
    0  0  0  0    <= “Sum"
```

Info in circuit: After 1st “Sum” update

- As values in a circuit:

```
     0  0  1  0     <= Carry i-1 connected to carry in of i
     1  1  1  1     <= “A”
  +  0  0  0  1     <= “B”
  — — — — — —
     1  1  1  0     <= “Sum"
```

Info in circuit: After 2nd “Sum” update

- As values in a circuit:

```
   0 1 1 0    <= Carry i-1 connected to carry in of i
   1 1 1 1    <= “A”
 + 0 0 0 1    <= “B”
 ———————————
   1 1 0 0    <= “Sum"
```

Info in circuit: After 3rd “Sum” update

- As values in a circuit:

```
     1  1  1  0     <= Carry i-1 connected to carry in of i
     1  1  1  1     <= “A”
  +  0  0  0  1     <= “B”
  ─ ─ ─ ─ ─ ─
     1  0  0  0     <= “Sum"
```

Info in circuit: After 3rd “Sum” update

- As values in a circuit:

```
     1  1  1  0     <= Carry i-1 connected to carry in of i
     1  1  1  1     <= “A”
  +  0  0  0  1     <= “B”
  — — — — — —
     0  0  0  0     <= “Sum"
```

Ripple Adder: Total Time

- N bits: Worst case scenario is ripple through all N
  - If T_c is the propagation delay through the Carry
    = $N \cdot T_c$
  - Dictates things like maximum clock cycle for any paths/loops that use addition
  - Lots of things rely on addition!

JLS

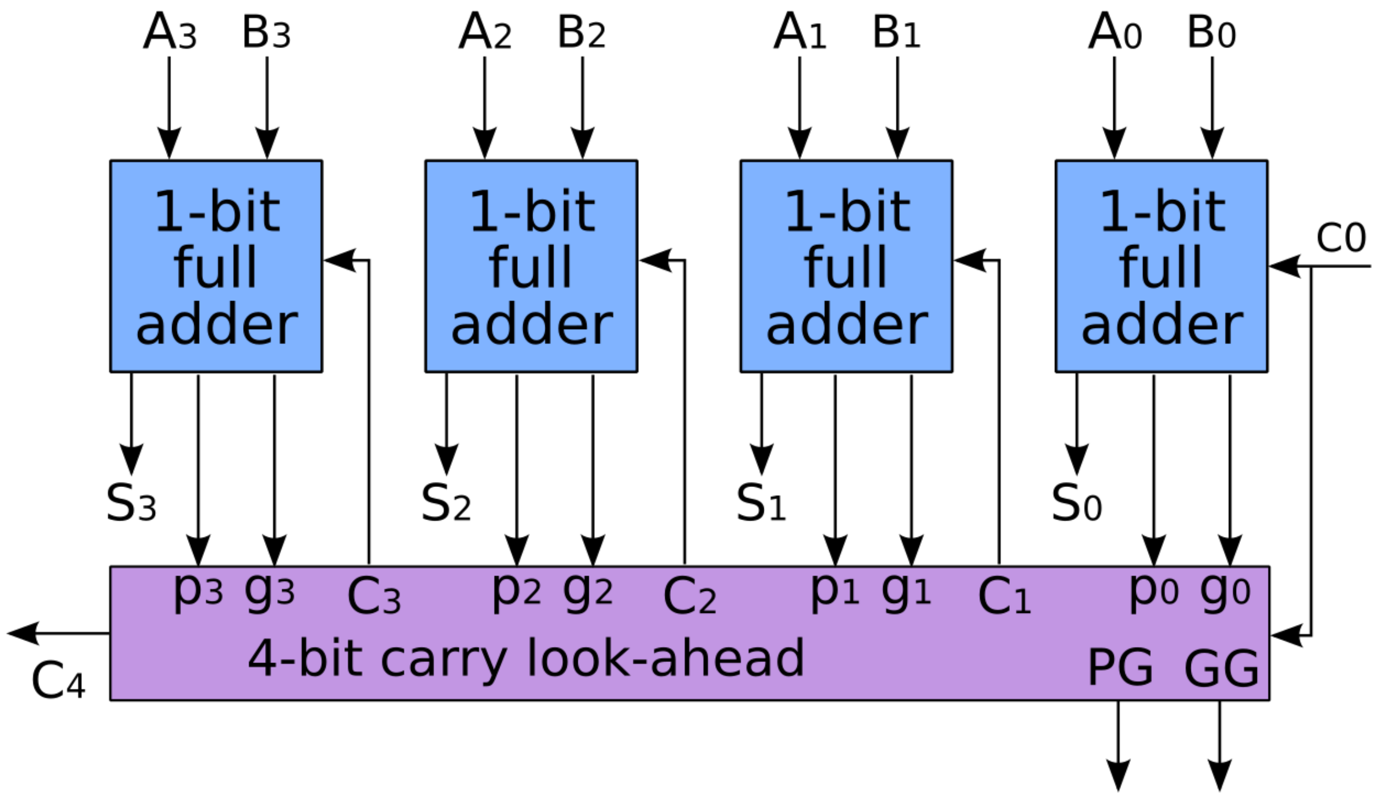
Wikipedia Animation

Carry Look-Ahead

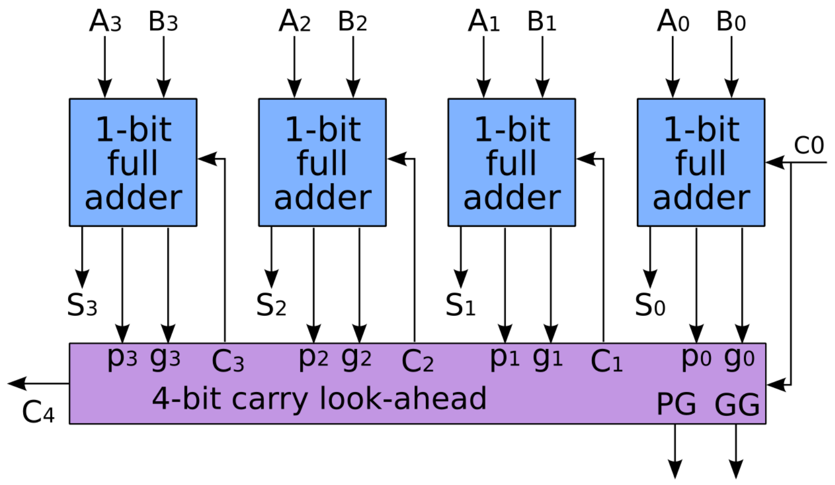
- Divide large addition into n-bit blocks
  - Within each block, determine if what each column would with a carry-in to the column
    - Would it "Generate" a carry? (g_x)
    - Would it merely "Propagate" the carry? (p_x)
    - Can the carry-out be represented in terms of a_x, b_x, cin_x, g_x and p_x?

?	<= Carry *in*
a	<= "A"
+ b	<= "B"
— —	
s	<= "Sum"

Building a Block (of 4)



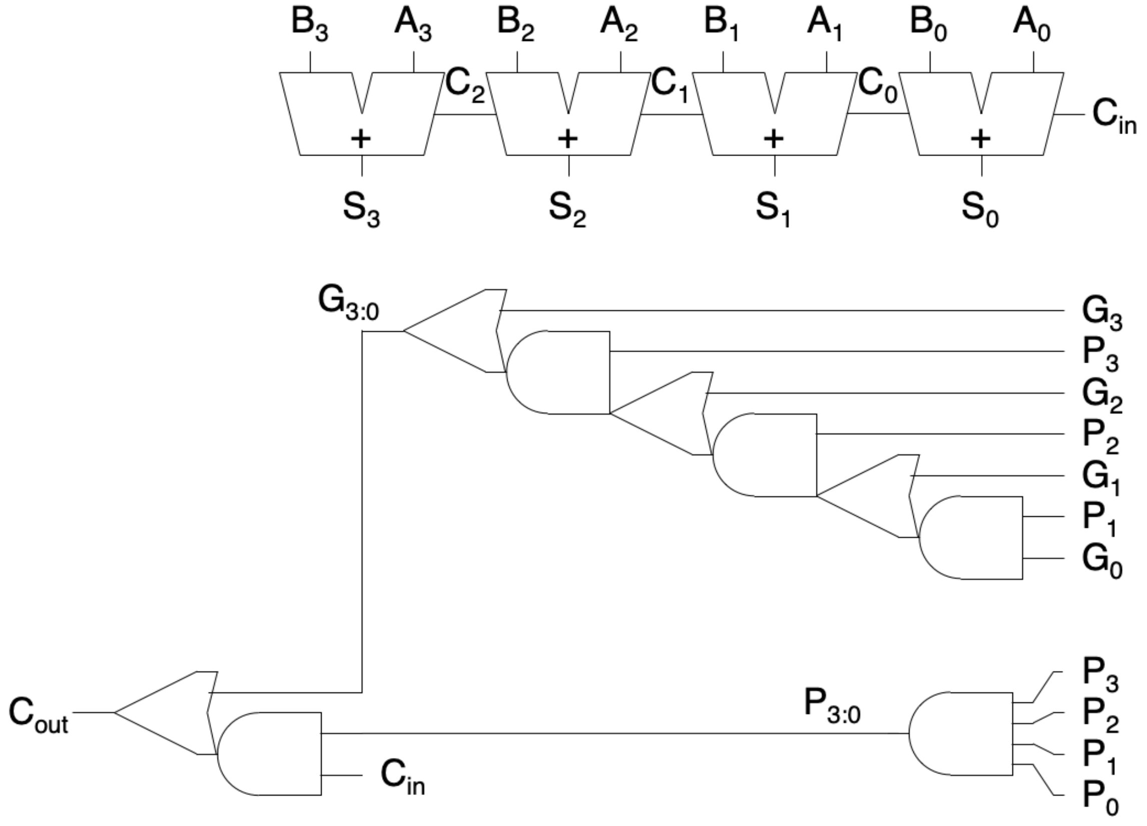
Extend “prediction” to block



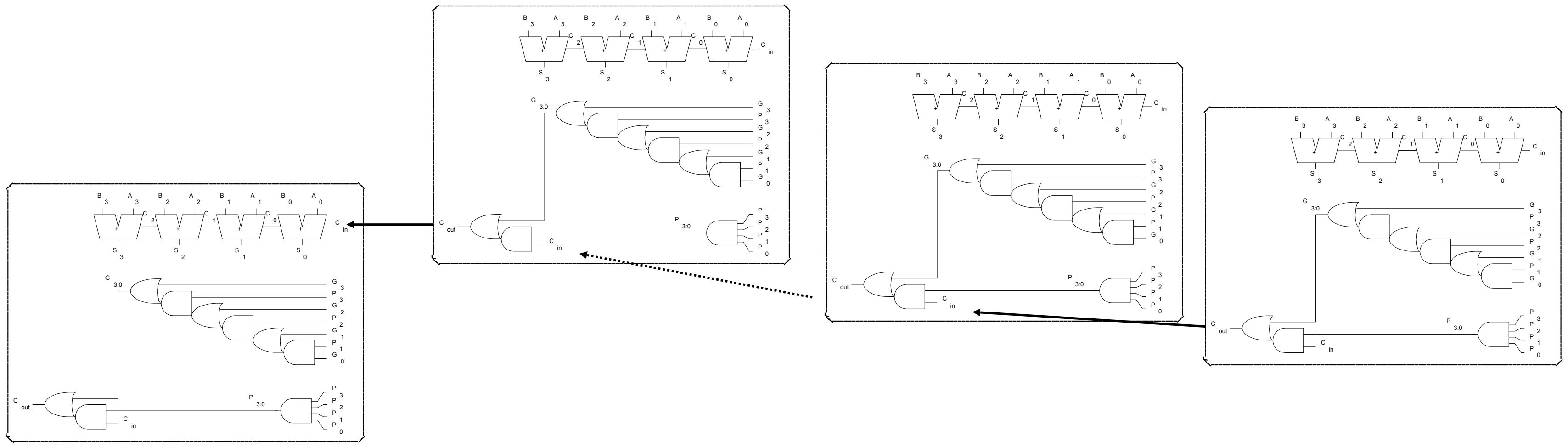
$$P_{block} = P_3 \cdot P_2 \cdot P_1 \cdot P_0$$

$$G_{block} = G_3 + P_3 \cdot (G_2 + (P_2 \cdot (P_1 \cdot G_0)))$$

Block “Prediction”



Ripple in 4 block, 16-bit CLA



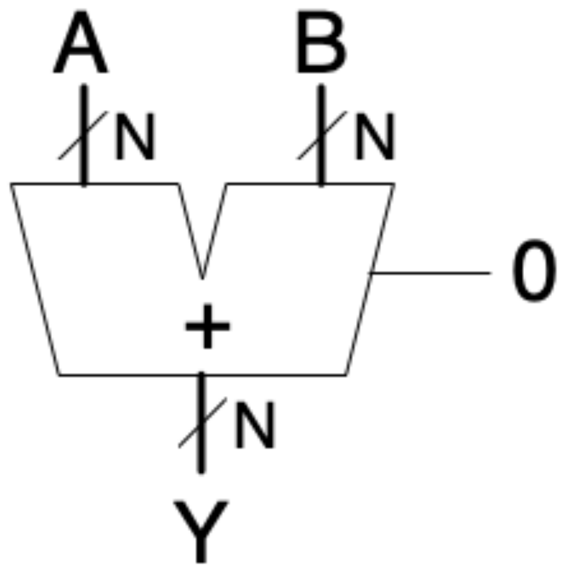
B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0
C_2 C_1 C_0 C_{in}
S_3 S_2 S_1 S_0
$G_{3:0}$
G_3 P_3 G_2 P_2 G_1 P_1 G_0
C_{out}
$P_{3:0}$
P_3 P_2 P_1 P_0
C_{in}

Trade off: Logic vs. Time

- CLA and other tricks (Prefix adder) add logic to reduce time
- Degenerate Case: A look-up table (full sum-of-products equation)
  - How many layers of logic? (Nots, ands, ors)?
  - To estimate complexity, how many rows and output columns are in a table to add two, 8-bit numbers?
    - Approximately how many AND gates?
      Approximately how many OR gates?
      Estimate the number of inputs that may be needed on OR gates

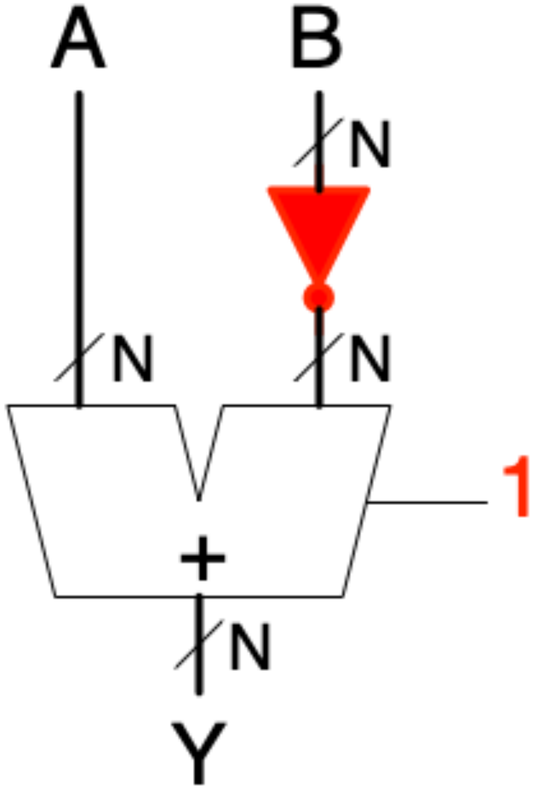
Subtraction

- The beauty of 2's complement
  - $A - B = A + \overline{B} + 1$



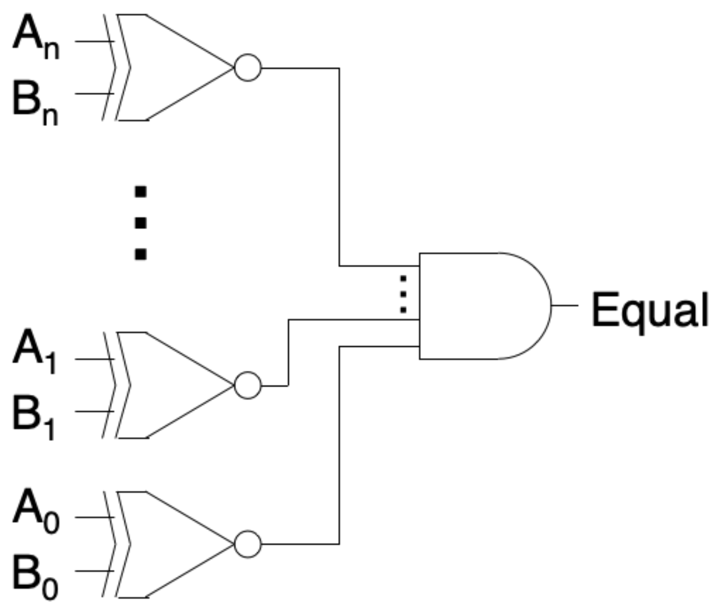
Subtraction

- The beauty of 2's complement
  - $A - B = A + \overline{B} + 1$



Comparisons

- Equality
  - Easy: Are any bits different?
- Equal to zero?
  - ?

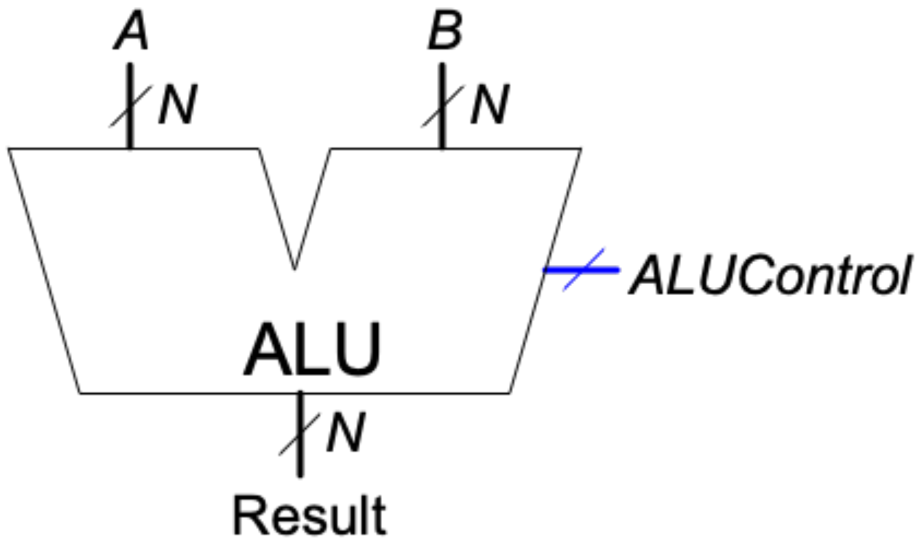


Comparisons

- Less than (signed): Is A<B?
- Leverage Subtraction: A<B is equivalent to A-B<0
  - Subtract and check result
    - General: Is A-B negative?
    - But…large numbers can “overflow”.
      Need to consider overflow and signs of A & B

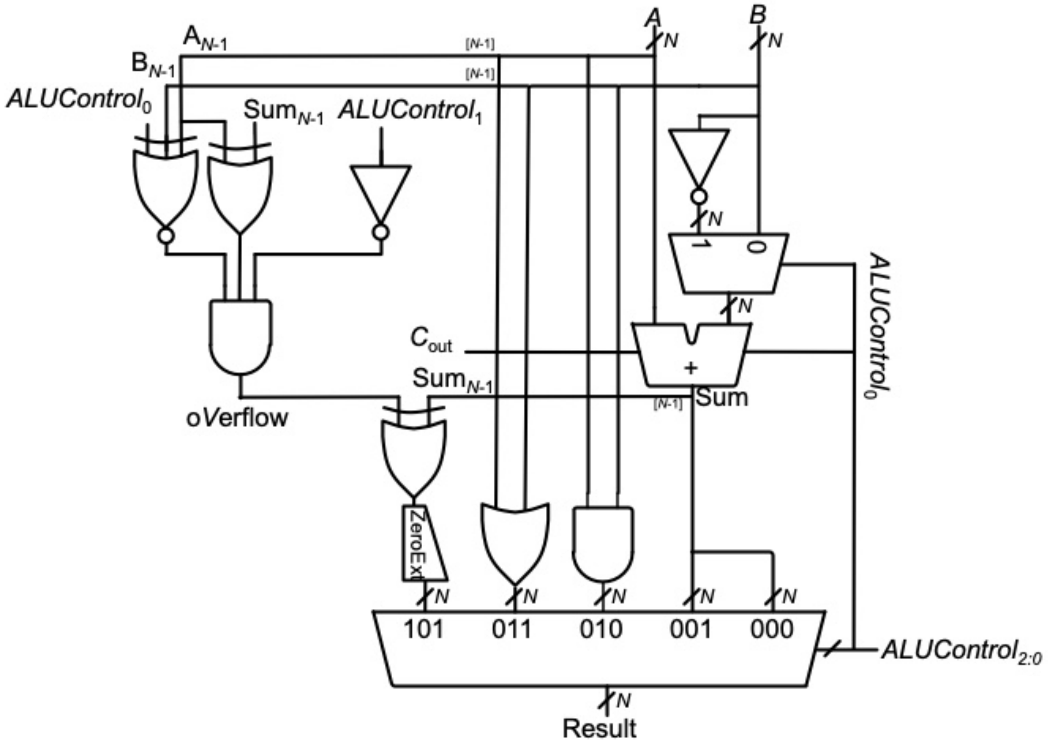
ALU: Arithmetic Logic Unit

- “Heart” of CPU: Does the computation stuff.
  - Basic operations
    - Addition
    - Subtraction
    - Bitwise AND
    - Bitwise OR
    - Comparison (<)



ALU: Arithmetic Logic Unit

- “Heart” of CPU: Does the computation stuff.
  - Basic operations
    - Addition: 000
    - Subtraction: 001
    - Bitwise AND: 010
    - Bitwise OR: 011
    - Comparison (<): 101



Memory / Storage

- Common types
  - Static Random Access Memory (SRAM)
  - Dynamic Random Access Memory
  - Read Only Memory (contents can’t be easily changed)

Memory / Storage

- General Approach
  - Store in a 2D grid of elements
  - Call each row a “word”
  - Each row has an index to access the content of the entire row
  - Concept: Computer programming
    - An Array (List) is a representation of memory

RAM: SRAM vs. DRAM

SRAM vs. DRAM

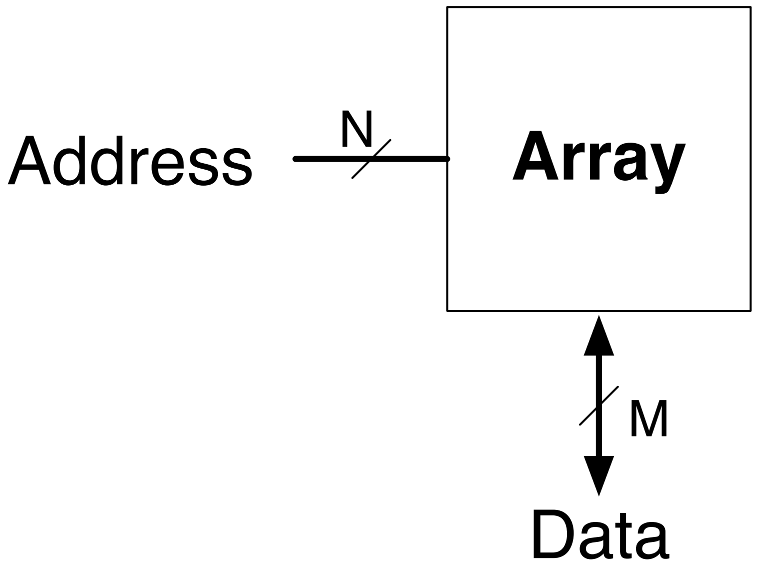
- S = “Static”/Unchanging (well, only changing when requested!)
  - Could be build from D Flip Flops (or similar “self-reinforcing” circuits)
- D = Dynamic: Values fade if not refreshed
- RAM: “Random Access”
  - About performance of reading/updating
  - Time take (*propagation delay*) does not depend on index requested

ROM

- Read Only
  - But still “Random Access” performance
- Fixed look-up table. Could be built with combinational logic!
  - Earlier example of “adder” could just be a ROM

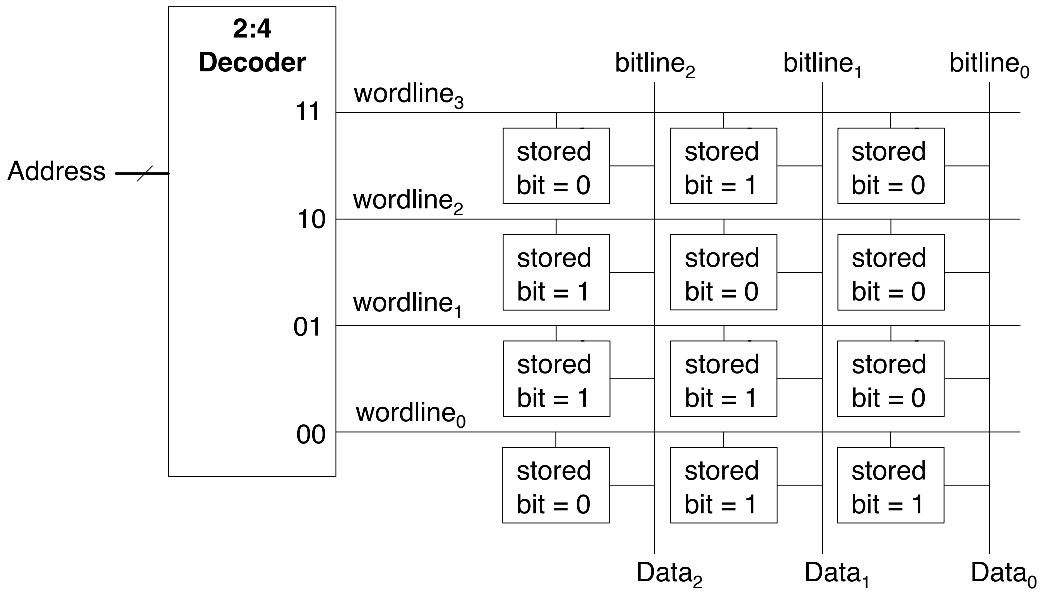
Reading Memory

- M = Word size
- N = “address size”
- How many total bits are stored?



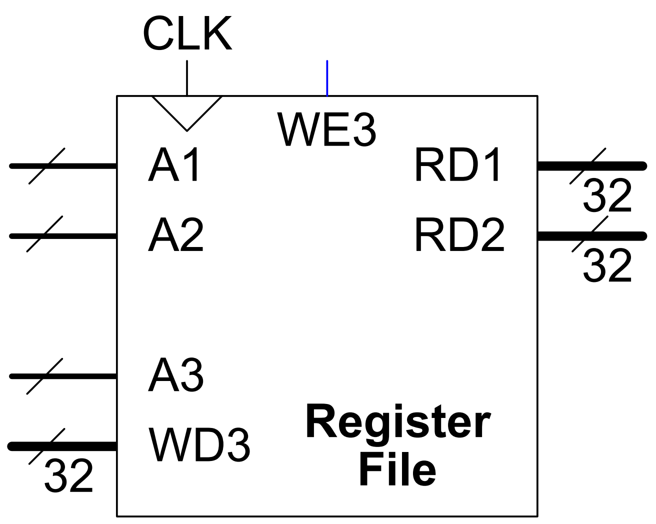
Memory Structure

- One approach
  - Bits are “enabled” to connect to shared output line



ALU Operations

- Need TWO inputs: need a memory structure that provides two values (I.e. dual output ports)
- The “Register File”
- Also supports writing (updating)



JLS Register File (W/ D Flip Flops)

FPGA

- Field Programmable
- Gate Array
  - Lattice iCE40 UP5k: Architecture Overview
    - RAMs, (Dual and Single Port)
    - Look Up Tables (LUTs): 4 inputs
    - D Flip Flops
    - Lots: ~5,000

Next Time

- Studio