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

Bill Siever
&
Michael Hall

This week

- Hw#4B due Wednesday
  - In-person demo required for full credit!
- Hw#5A posted tonight / due Sunday
  - A little shorter
  - In-person demo likely

RISC-V Edition of book!

RISC-V in the news…

Studio 4B Review & Ch 5

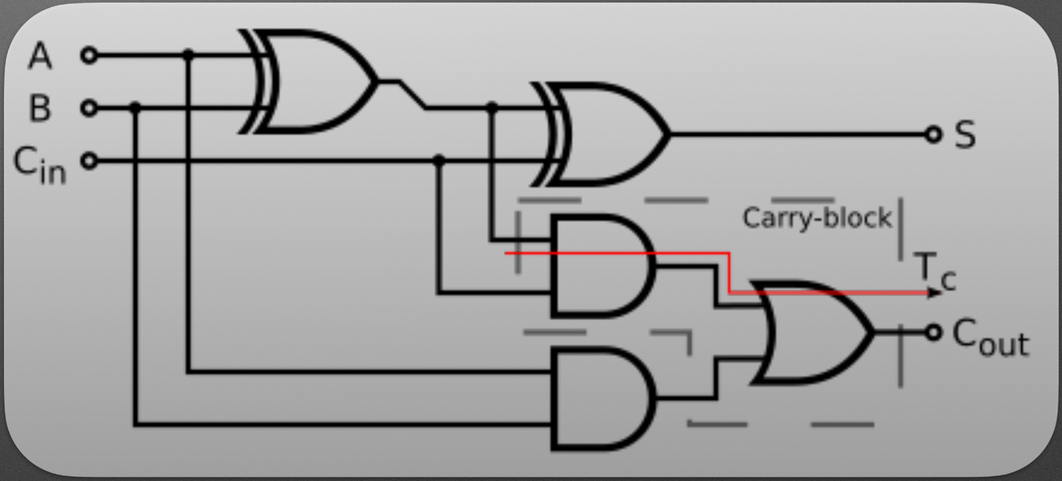
Studio 4B

- Full-Adder: Behavioral variations in simulation
  - iCE40 Mapping
- Full-Adder: Column of number

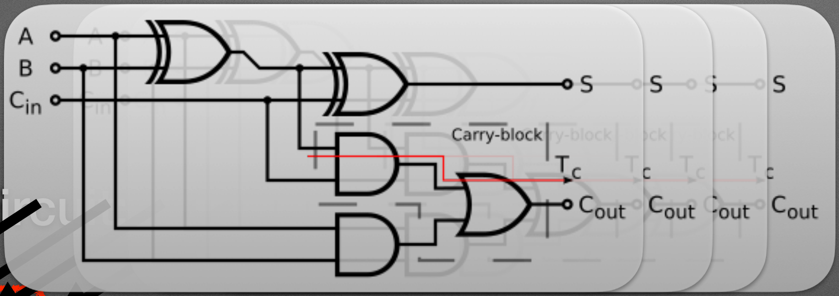
Ripple Adder

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

```
   1 1 1 1
 + 0 0 0 1
 _ _ _ _ _ _
```



Info in circuit



- As values in a ci

```
    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"
```

JLS Example: 1111+0001

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!

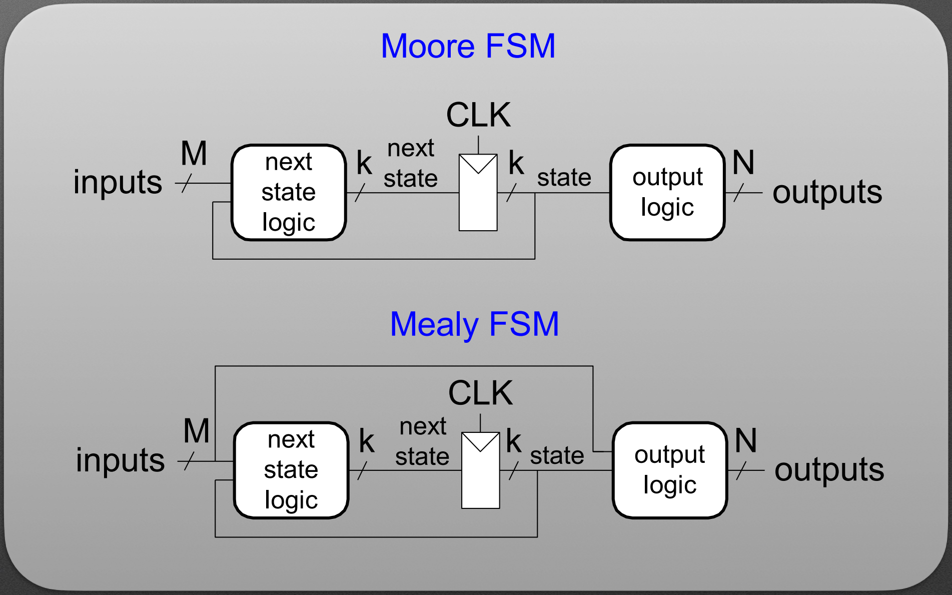
Wikipedia Animation

Studio 4B: Structural Ripple-Carry Adder (And Generate Statement) (And Hardware)

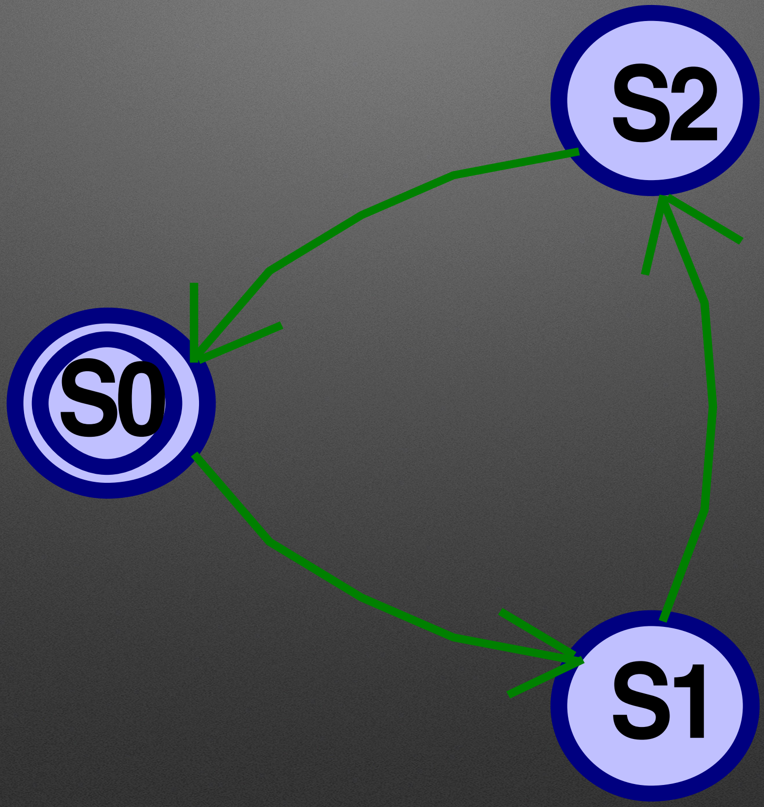
Studio 4B: State Machines

Verilog FSMs

- Three parts
  - Next state logic
    (arrows / next state table)
  - State register (active bubble)
  - Output logic (output equations)

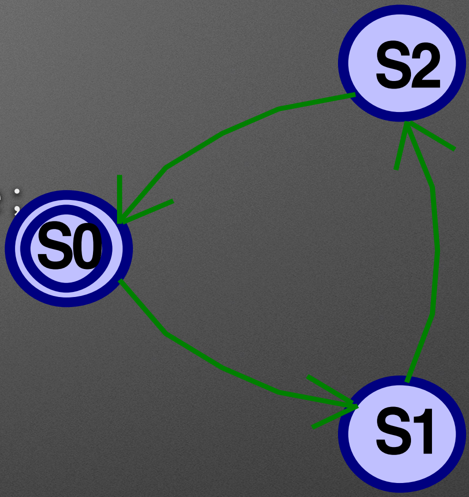


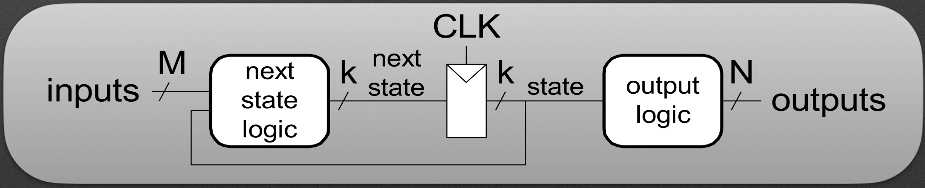
Divide by 3 Counter


$$\begin{tikzcd}
 & S2 \arrow[dl] \\
S0 \arrow[dr] & \\
 & S1 \arrow[uu]
\end{tikzcd}$$

Verilog

```
module divideby3FSM(input  logic clk,
                    input  logic reset,
                    output logic q);

   typedef enum logic [1:0] {S0, S1, S2} statetype;
   statetype state, nextstate;

   // state register
   always_ff @(posedge clk, posedge reset)
      if (reset) state <= S0;
      else       state <= nextstate;

   // next state logic
   always_comb
     case (state)
        S0:      nextstate = S1;
        S1:      nextstate = S2;
        S2:      nextstate = S0;
        default: nextstate = S0;
     endcase

   // output logic
   assign q = (state == S0);
endmodule
```





Chapter 5

Goal…and pattern…

$X = Y + Z$
(Using variables)

X = Y - Z
(Using variables)

X = Y < Z
(Using variables)

X = Y & Z
(Using variables)

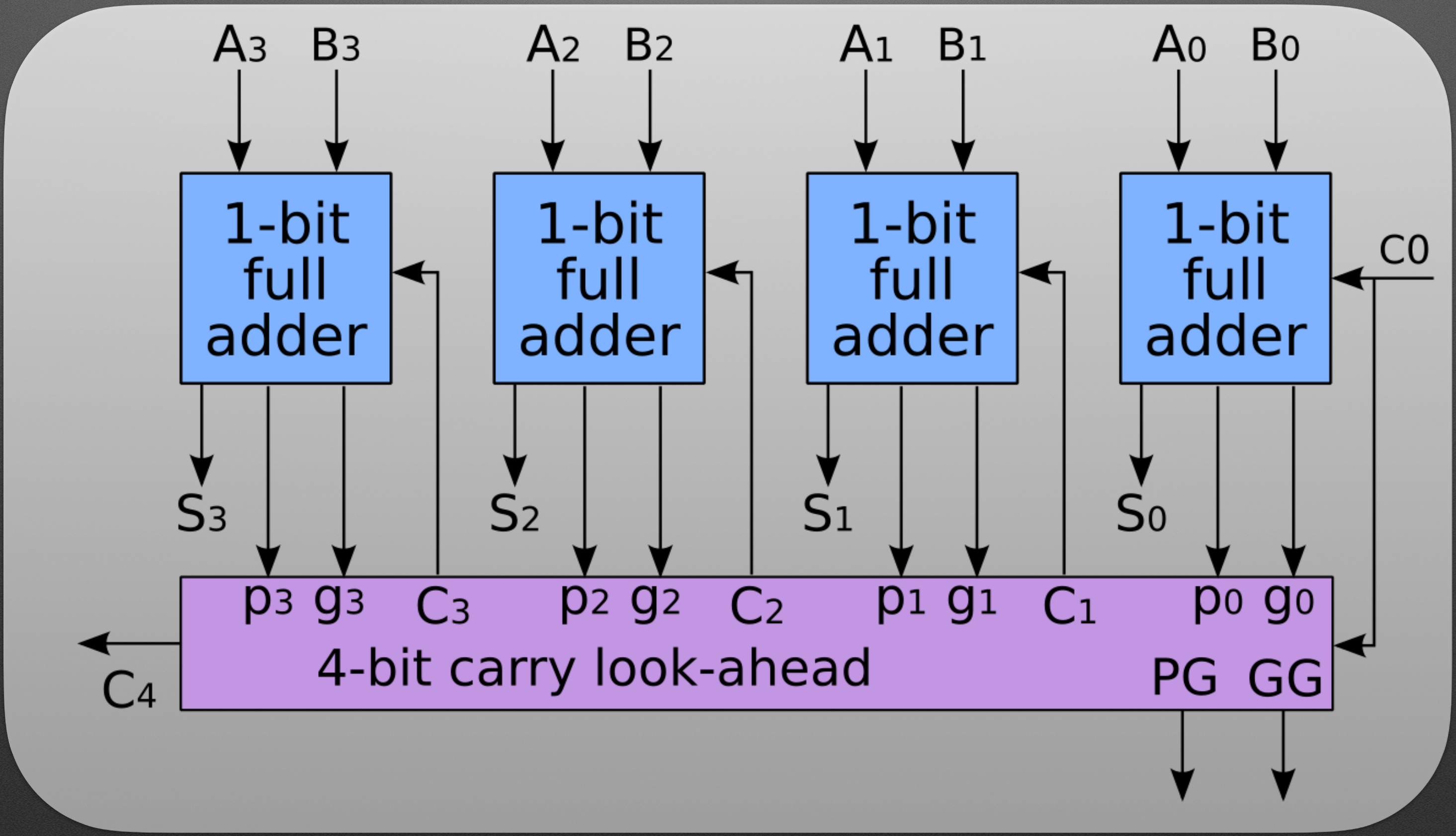
Better (faster) Addition

Carry Look-Ahead

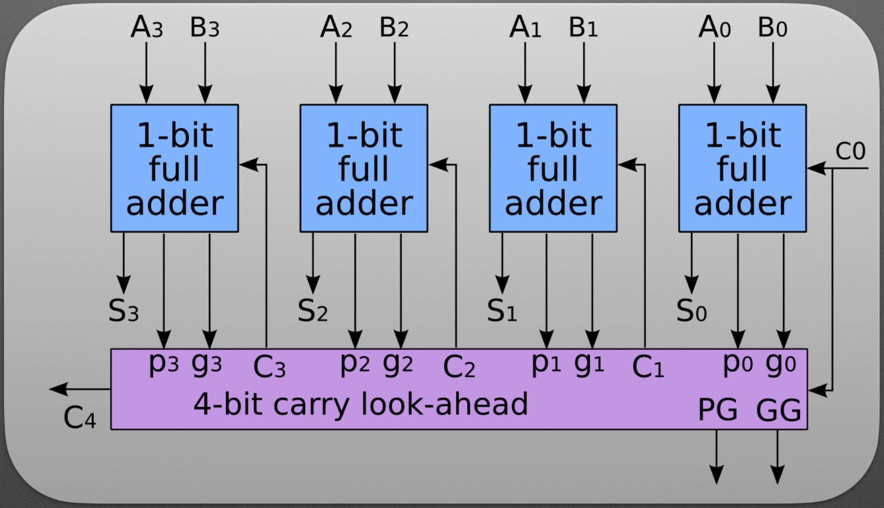
- Divide large addition into n-bit blocks
  - Within each block, determine if what each column would with a carry-in to the column
    - Would it "Generate" a carry? (g_x)
    - Would it merely "Propagate" the carry? (p_x)
    - Can the carry-out be represented in terms of a_x, b_x, cin_x, g_x and p_x?

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

Building a Block (of 4)



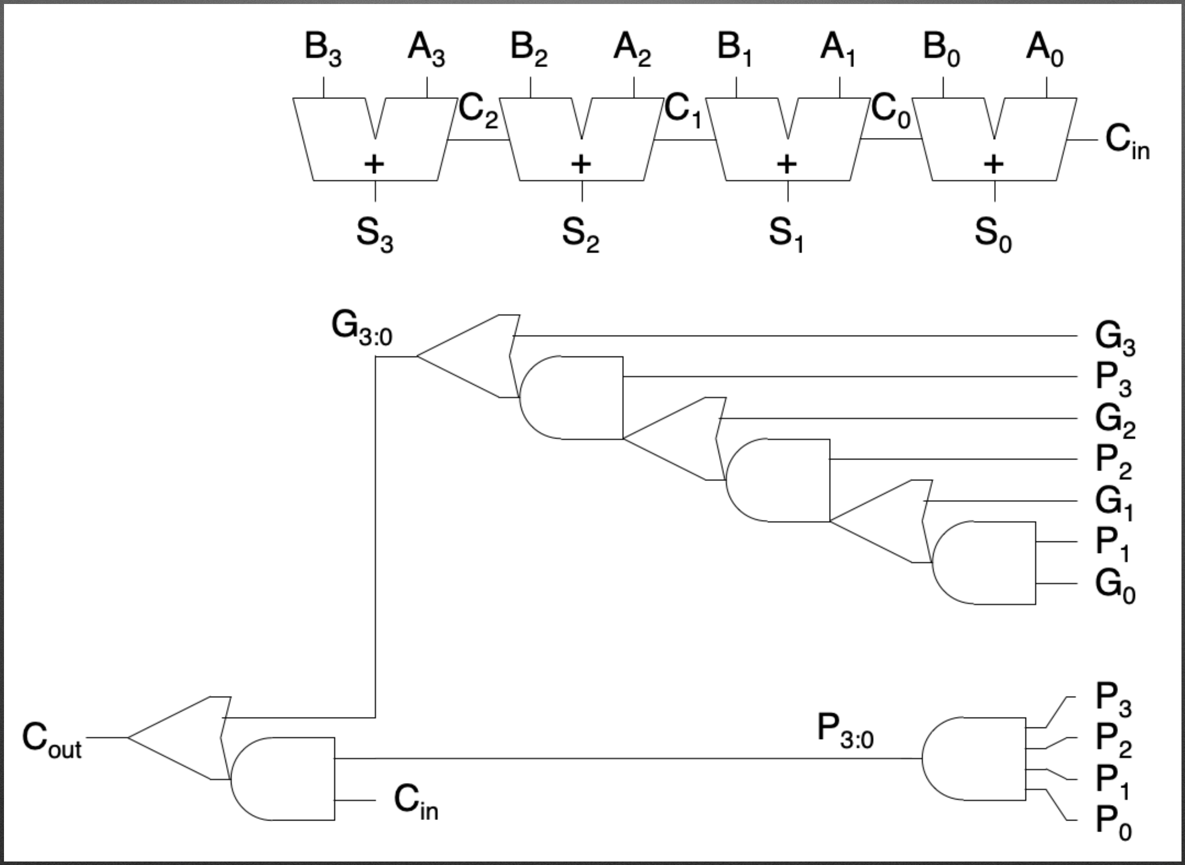
Extend "prediction" to block



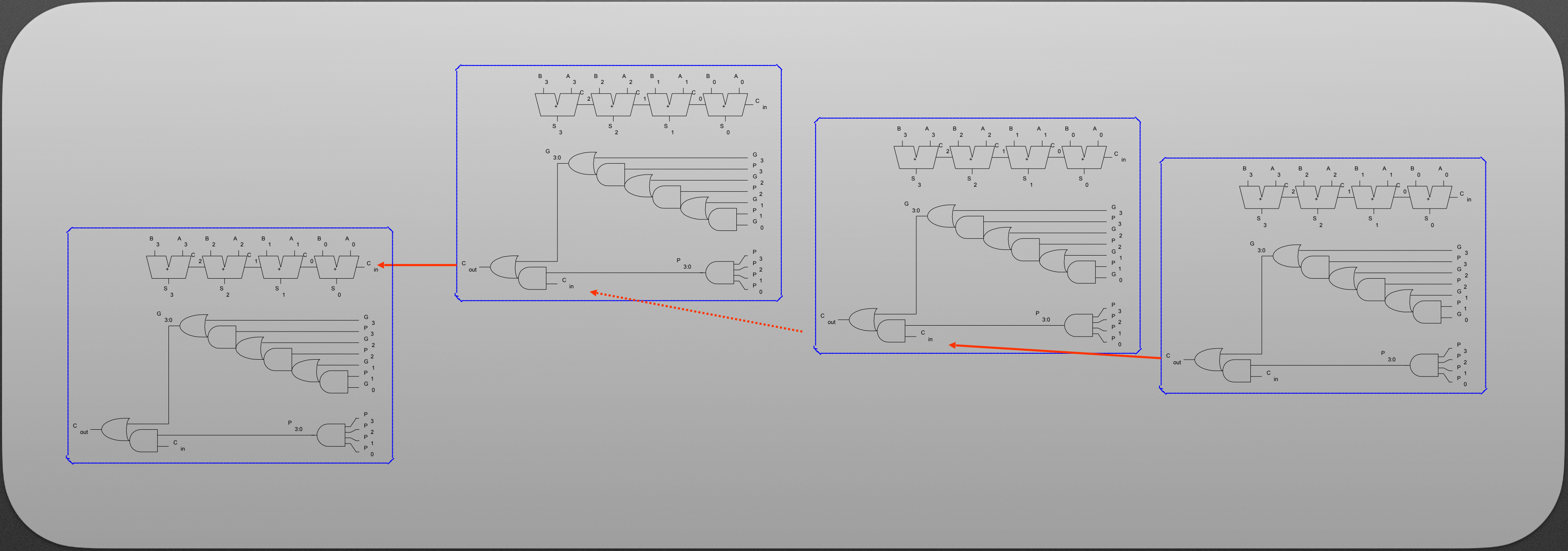
$$P_{block} = P_3 \cdot P_2 \cdot P_1 \cdot P_0$$

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

Block “Prediction”



Ripple in 4 block, 16-bit CLA



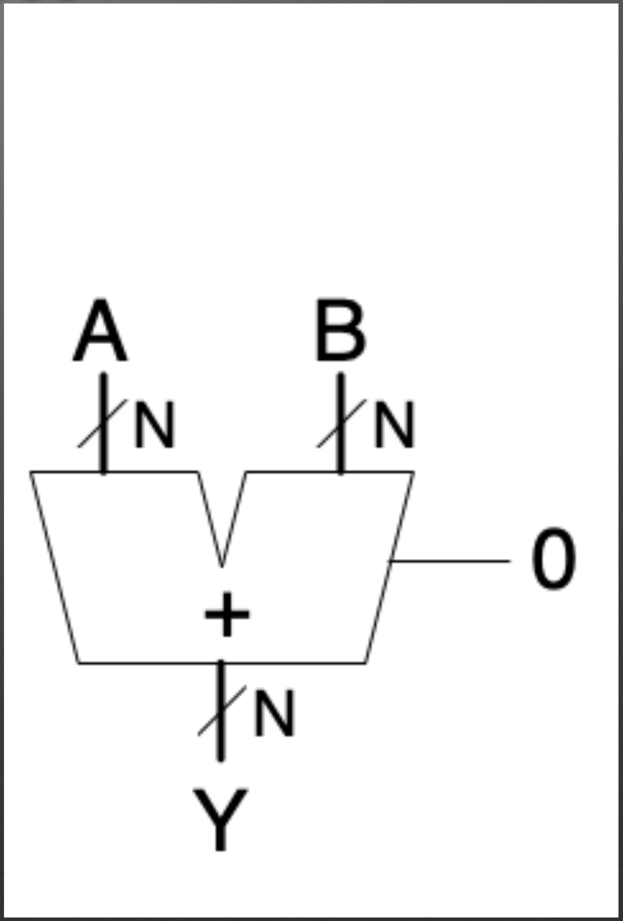
B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0
C_2 C_1 C_0 C_{in}
S_3 S_2 S_1 S_0
$G_{3:0}$
G_3 P_3 G_2 P_2 G_1 P_1 G_0
C_{out}
$P_{3:0}$
P_3 P_2 P_1 P_0

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

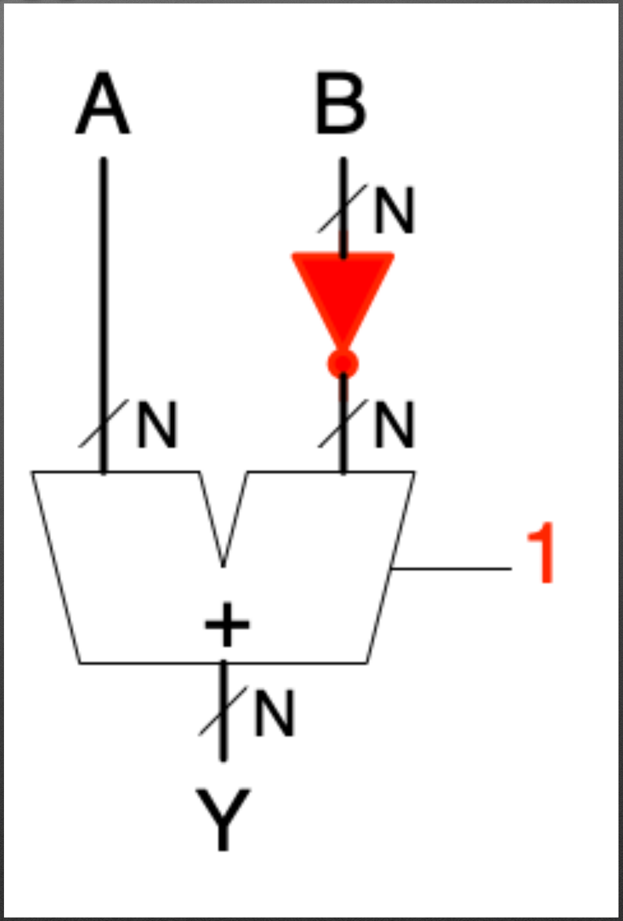
Subtraction

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



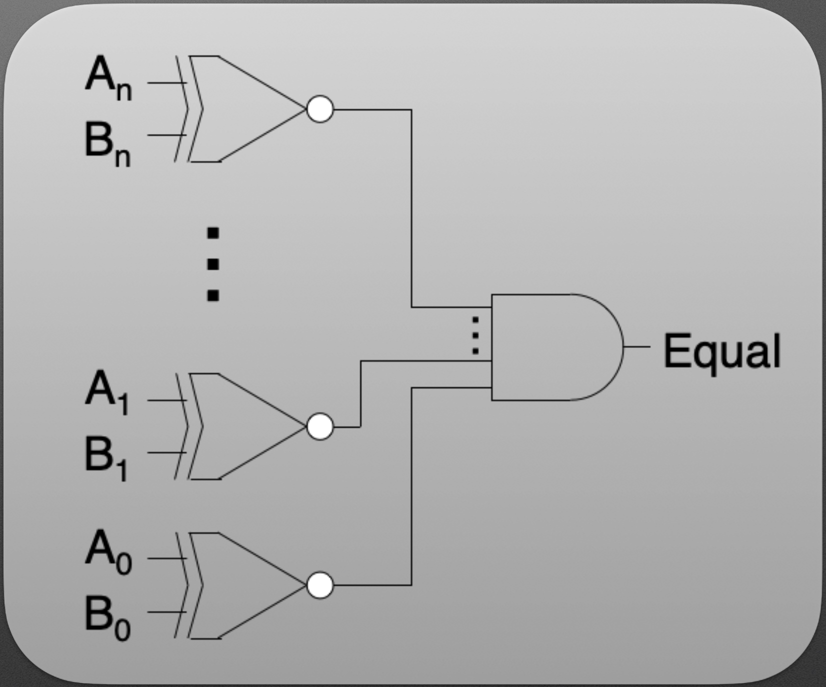
Subtraction

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



Comparisons

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

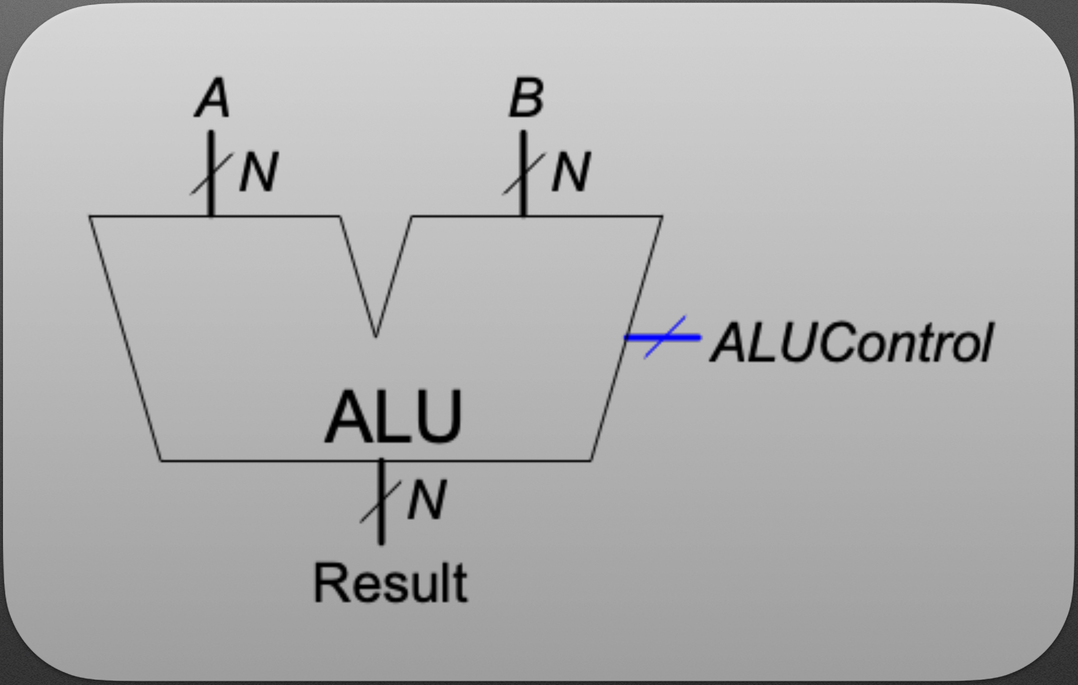


Comparisons

- 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

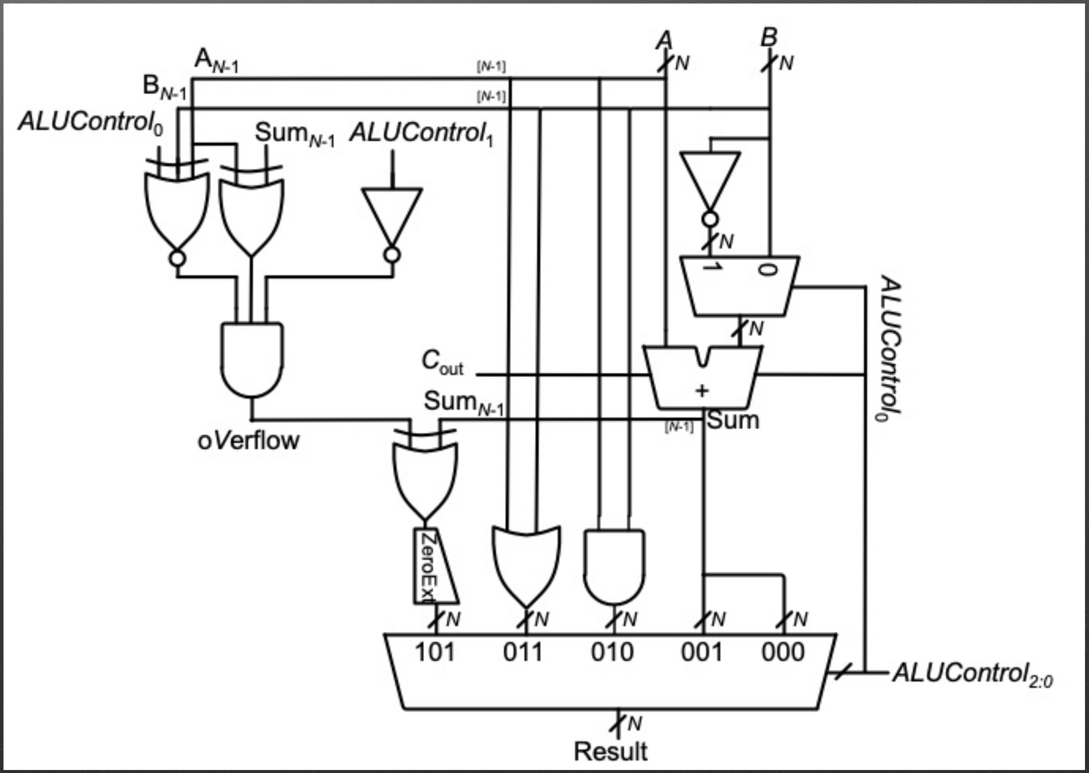
ALU: Arithmetic Logic Unit

- “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



Other Operations: Shift Left (one place)

- $2^5 + 2^4 + 2^2 + 2^1 = 54$

Value	0	0	1	1	0	1	1	0
Place Value	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

- $2^6 + 2^5 + 2^3 + 2^2 = 108$

Value	0	1	1	0	1	1	0	0
Place Value	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Shifting (unsigned / ∞ width)

- Left n bits: Equivalent to multiplication by 2^n
  - Multiplication algorithm is a mix of addition and multiplication by b^k
    - Ex: 123×12
    - Ex: $1011_2 \times 11_2$
- Right n bits: Equivalent to division by 2^n and truncation

Memory

Memory / Storage

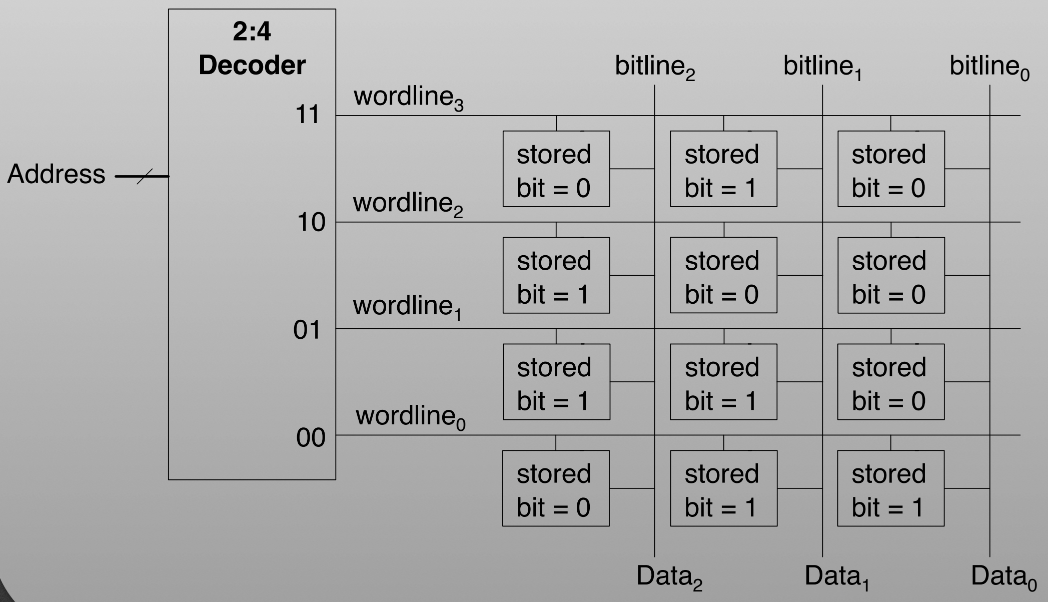
- Common types
  - Static Random Access Memory (SRAM)
  - Dynamic Random Access Memory
  - Read Only Memory (contents can’t be easily changed)

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

Memory Structure

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

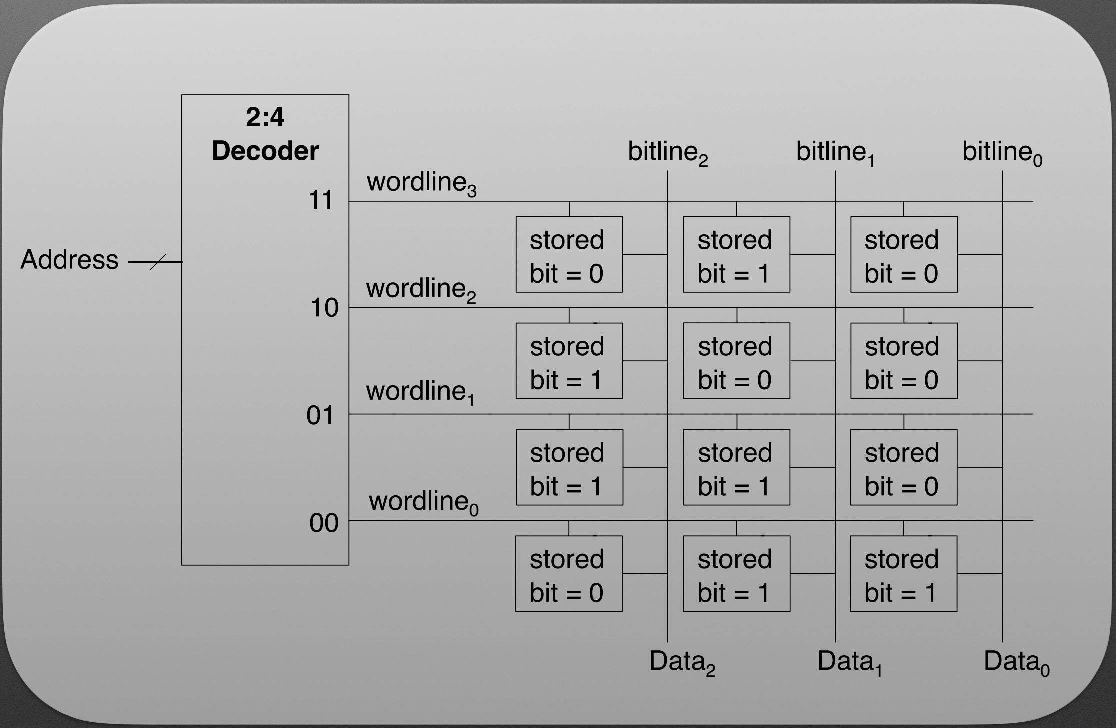


Memory / Storage

- Concept: Computer programming
    - An Array (List) is a representation of memory
- “Random” : Largely about time to access
    - The “random” means the location doesn’t have much impact on access time
    - Vs. “Sequential”

Memory Structure

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



A Sequential Example

RAM: SRAM vs. DRAM

SRAM vs. DRAM

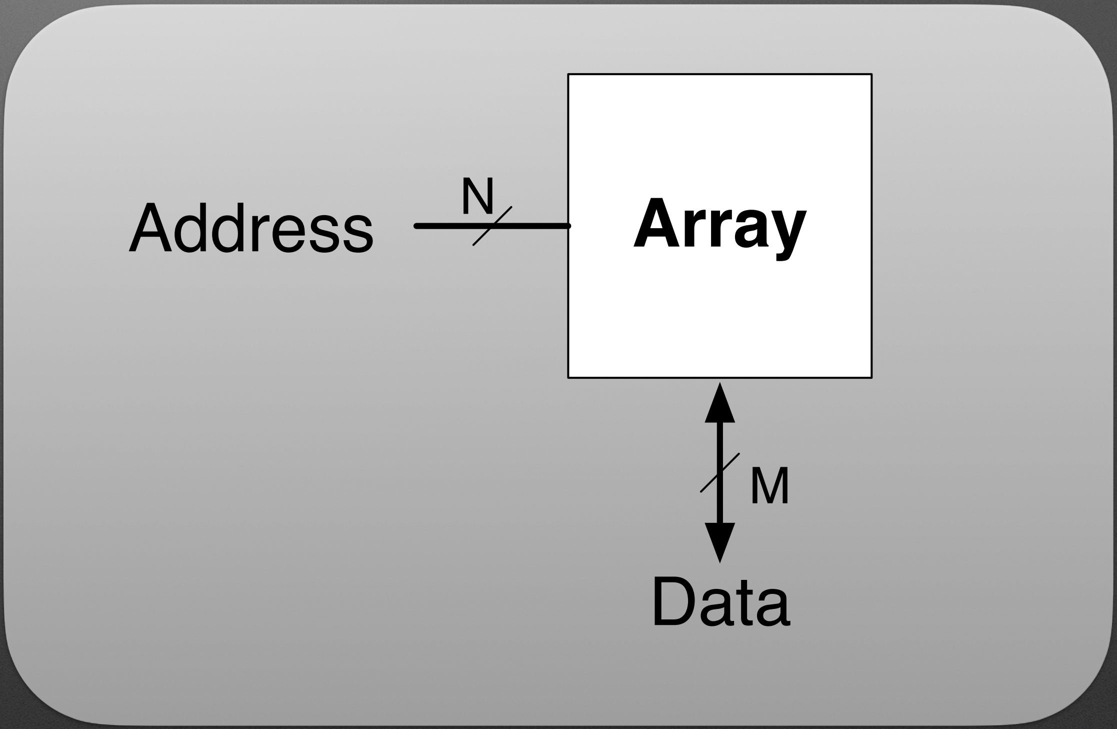
- S = “Static”/Unchanging (well, only changing when requested!)
    - Could be built from D Flip Flops (but similar “self-reinforcing” circuits more likely)
- D = Dynamic: Values fade if not refreshed
- RAM: “Random Access”
    - About performance of reading/updating
    - Time take (*propagation delay*)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

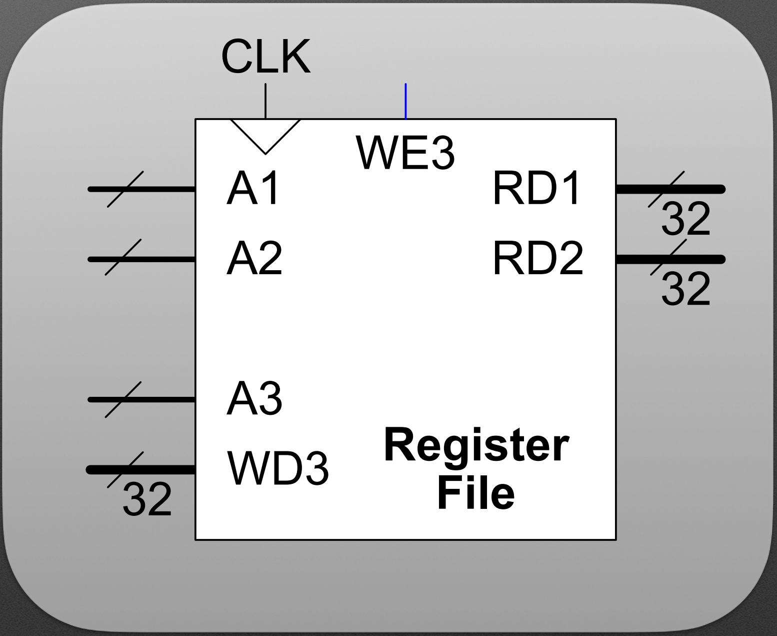
Reading Memory

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



ALU Operations

- Context: X=Y+Z
  - We need places to hold Y, Z, and X.
- Need TWO inputs:
  - need a memory structure that provides 2 values (I.e. dual output ports)
- The "Register File"
- Also supports writing (updating)

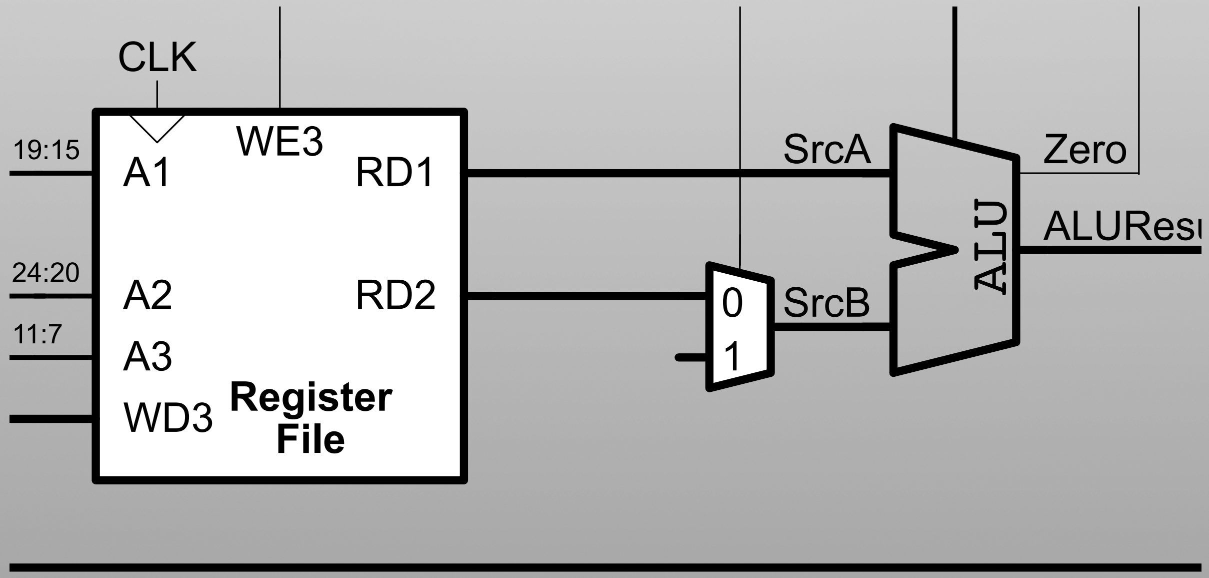


JLS Register File (W/ D Flip Flops)

Verilog: RISC-V Register File

```
// 32 x 32 register file with 2 read, 1 write port
module regfile(input  logic        clk,
               input  logic        we3,
               input  logic [4:0]  ra1, ra2, wa3,
               input  logic [31:0] wd3,
               output logic [31:0] rd1, rd2);

  logic [31:0] rf[31:0];

  always_ff @(posedge clk)
    if (we3) rf[wa3] <= wd3;

  assign rd1 = rf[ra1];
  assign rd2 = rf[ra2];
endmodule
```

Big Picture: add x, y, z



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

Questions

- Chapter fits well with 361S
  - Wait until next chapter…
- Why so many memory types / what are the differences?
  - Evolution over time
  - Different needs: Capacity vs. Need — the memory hierarchy

Questions

- PLA vs. FPGA
  - PLA: (largely) <u>2-level logic</u> / simple combinational logic
  - FPGA: <u>Array of many programmable blocks with programmable interconnects</u>
    - Can efficiently achieve more than 2-layer logic
    - Memory/storage is inherent (can do full state machine…see hw 4b)