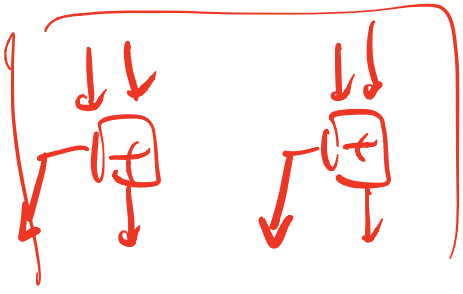
0x07 - Arithmetic Logic Unit and Memory

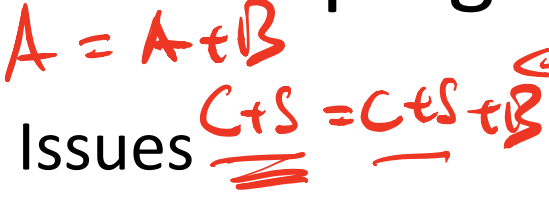
ENGR 3410: Computer Architecture

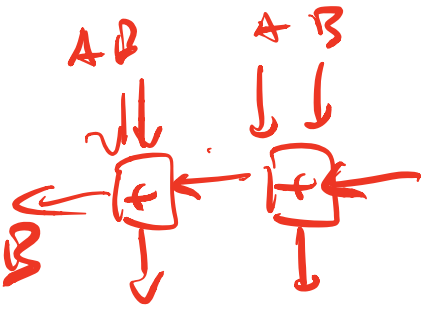
Jon Tse

Fall 2020



Housekeeping



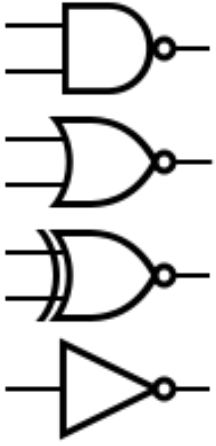


- ~~Lab 2 Makefile Issues~~
- Let's talk about Carry Save Adders some more

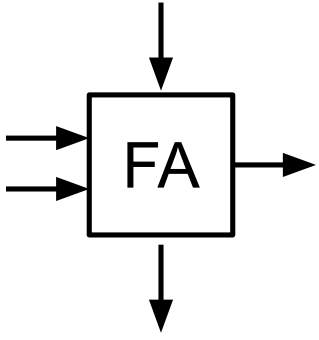
```
Press ENTER or type command to continue[comparch@docker-desktop lib]$ make test_ha.bin
iverilog -Wall -g2012 -o test_ha.bin test/test_ha.v dff.v csa.v ripple.v ha.v fa.v
dff.v:2: warning: Some modules have no timescale. This may cause
dff.v:2:        : confusing timing results.  Affected modules are:
dff.v:2:        :    -- module test_HA declared here: test/test_ha.v:2
ha.v:3: warning: timescale for half_adder inherited from another file.
ripple.v:2: ...: The inherited timescale is here.
csa.v:24: warning: Port 1 (A) of ripple_carry_adder expects 4 bits, got 5.
csa.v:24:        : Pruning 1 high bits of the expression.
csa.v:24: warning: Port 2 (B) of ripple_carry_adder expects 4 bits, got 3.
csa.v:24:        : Padding 1 high bits of the port.
csa.v:24: warning: Port 3 (Ci) of ripple_carry_adder expects 1 bits, got 32.
csa.v:24:        : Pruning (signed) 31 high bits of the expression.
csa.v:24: error: expression not valid in assign l-value: 'sd0
csa.v:24: error: Output port expression must support continuous assignment.
csa.v:24:      : Port 5 (Co) of ripple_carry_adder is connected to 'sd0
warning: Found both default and explicit timescale based delays. Use
       : -Wtimescale to find the design element(s) with no explicit
       : timescale.
1 error(s) during elaboration.
Makefile:18: recipe for target 'test_ha.bin' failed
make: *** [test_ha.bin] Error 1
[comparch@docker-desktop lib]$
```

Review

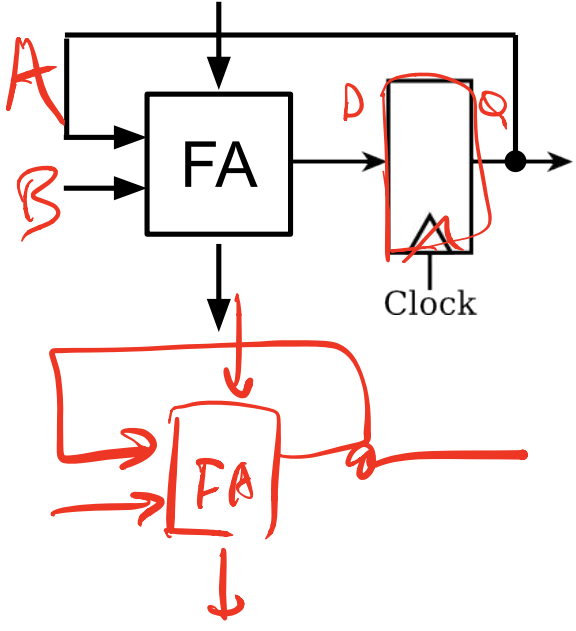
Gates



Combinational Gates



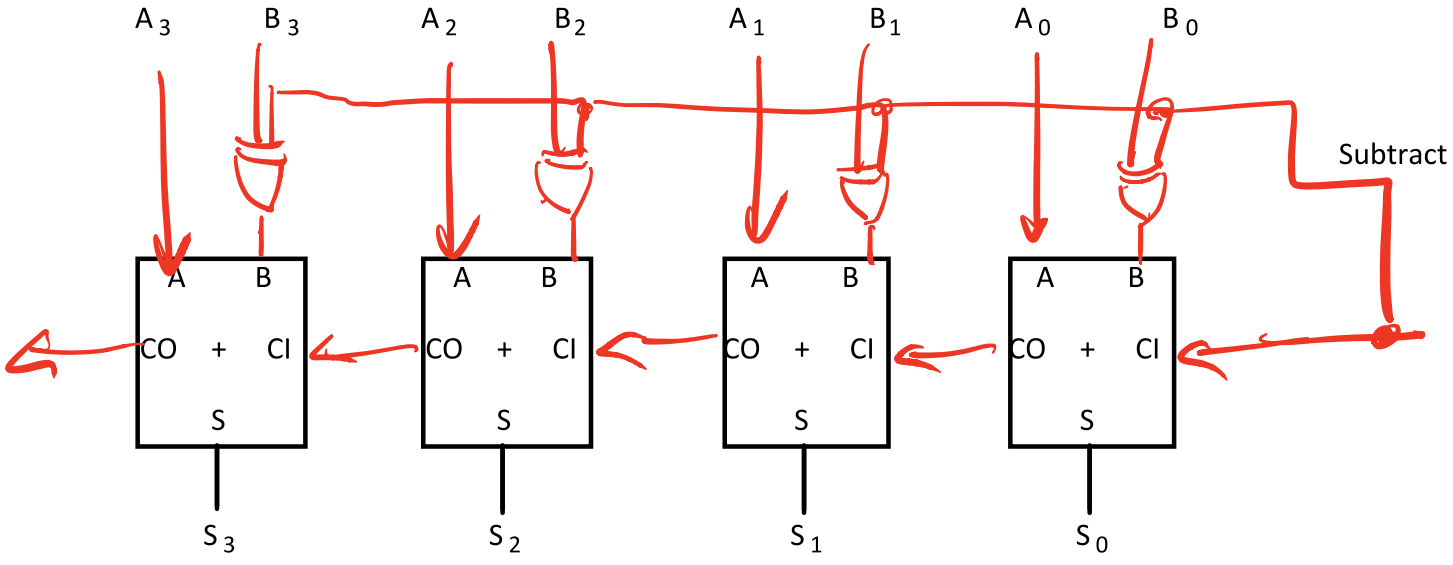
Sequential Gates



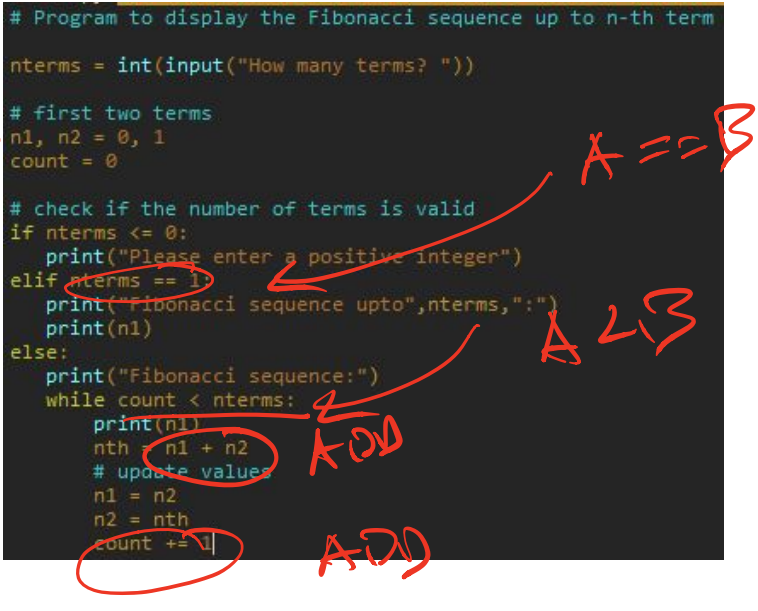
Review - Adder/Subtractor

- Add Control Line for Subtraction

$$A - B = A + (-B) = A + \overline{B} + 1$$



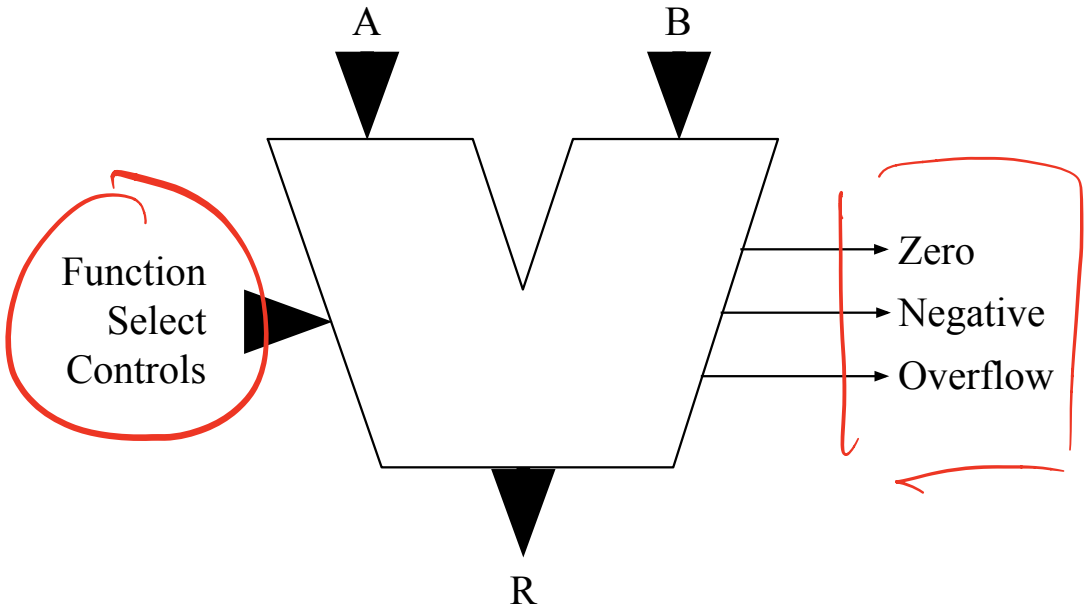
Computation




```
# Program to display the Fibonacci sequence up to n-th term

nterms = int(input("How many terms? "))

# first two terms
n1, n2 = 0, 1
count = 0

# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
else:
   print("Fibonacci sequence:")
   while count < nterms:
       print(n1)
       nth = n1 + n2
       # update values
       n1 = n2
       n2 = nth
       count += 1
```

ALU: Arithmetic Logic Unit

- Computes arithmetic & logic functions based on controls
  - Add, subtract
  - AND, NAND, OR, NOR, XOR, ==, <, overflow, …



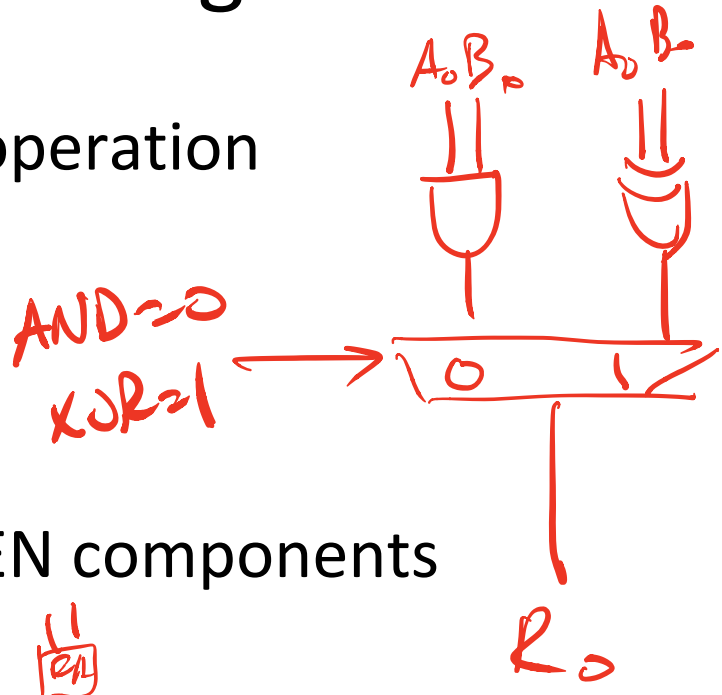
bitwise AND

$B_1 A_1$ $B_0 A_0$

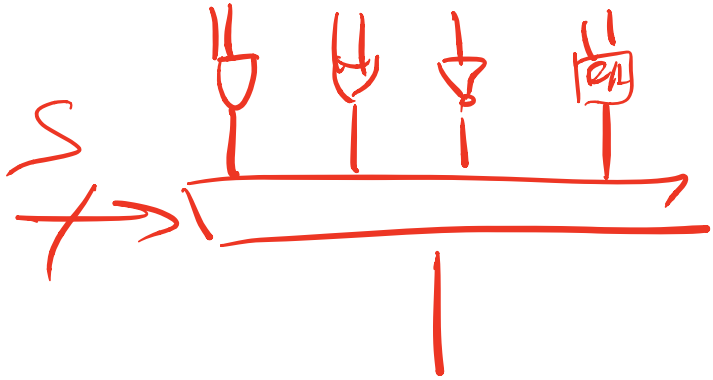
R_1 R_0

ALU: Initial Design

- Separate units for each operation
- Mux at the end

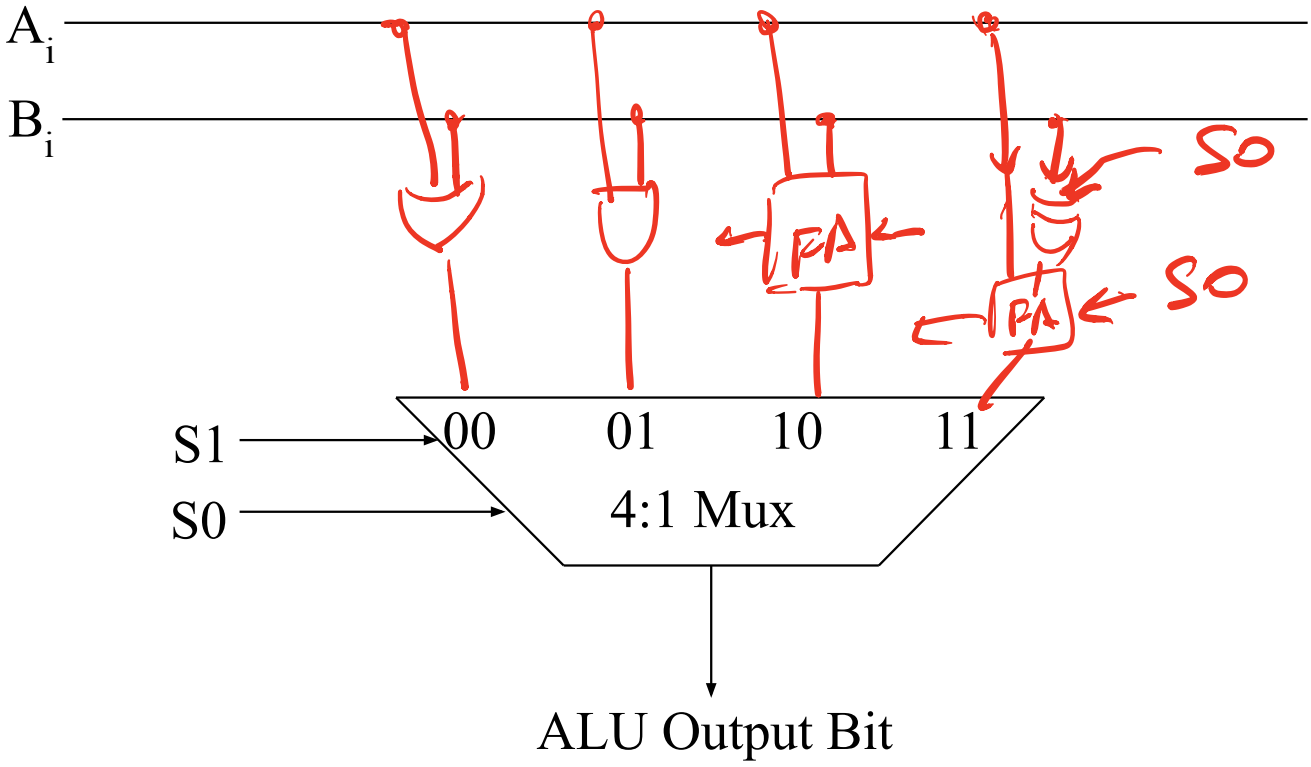


- Best version of each unit
- No optimization BETWEEN components



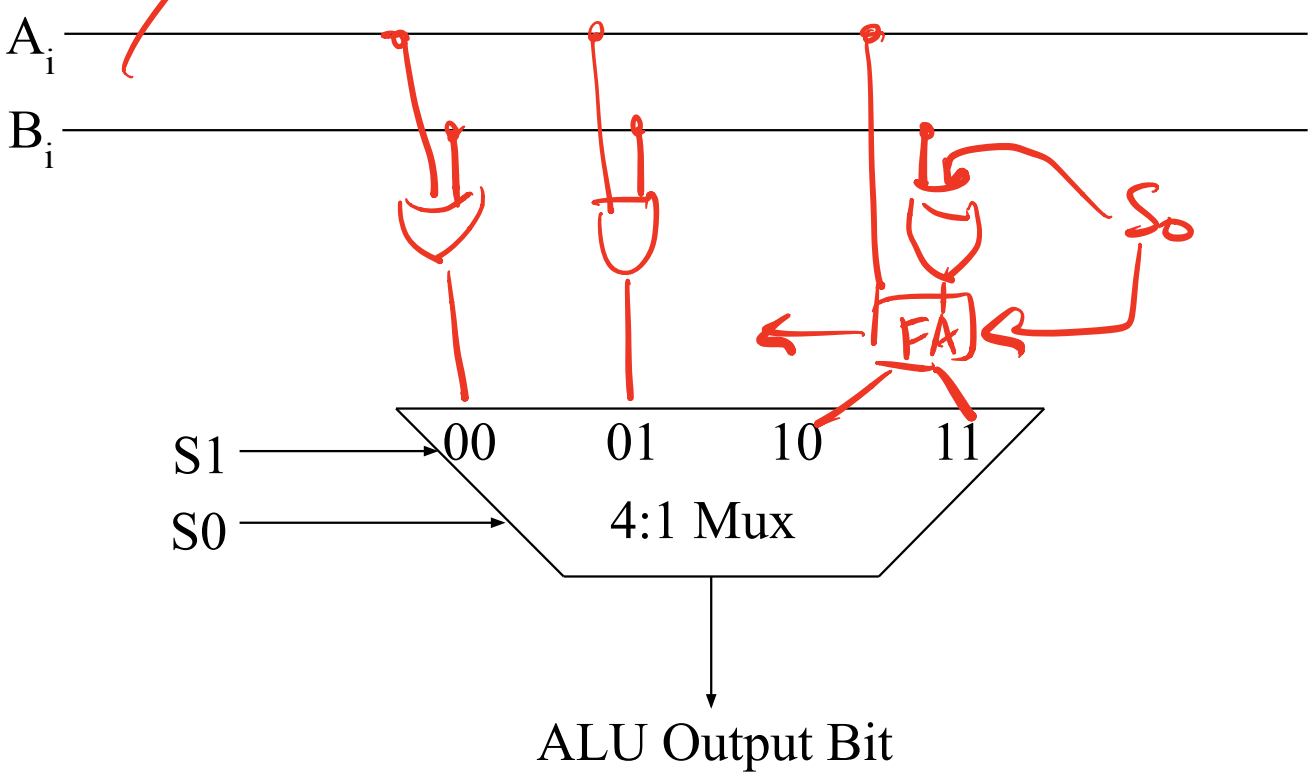
Bit Slice ALU Design

- 00: OR
- 01: AND
- 10: Add
- 11: Subtract



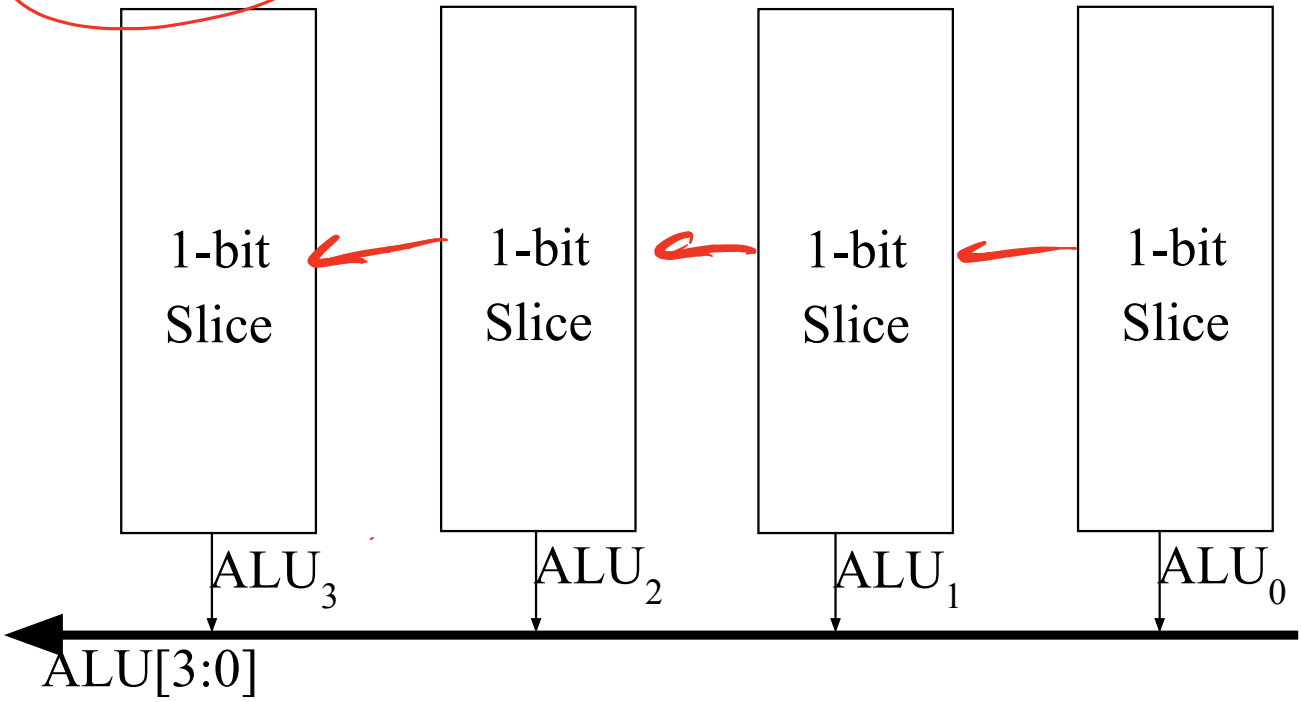
Bit Slice ALU Design

- 00: OR
- 01: AND
- 10: Add
- 11: Subtract



Bit Slice ALU Design (cont.)

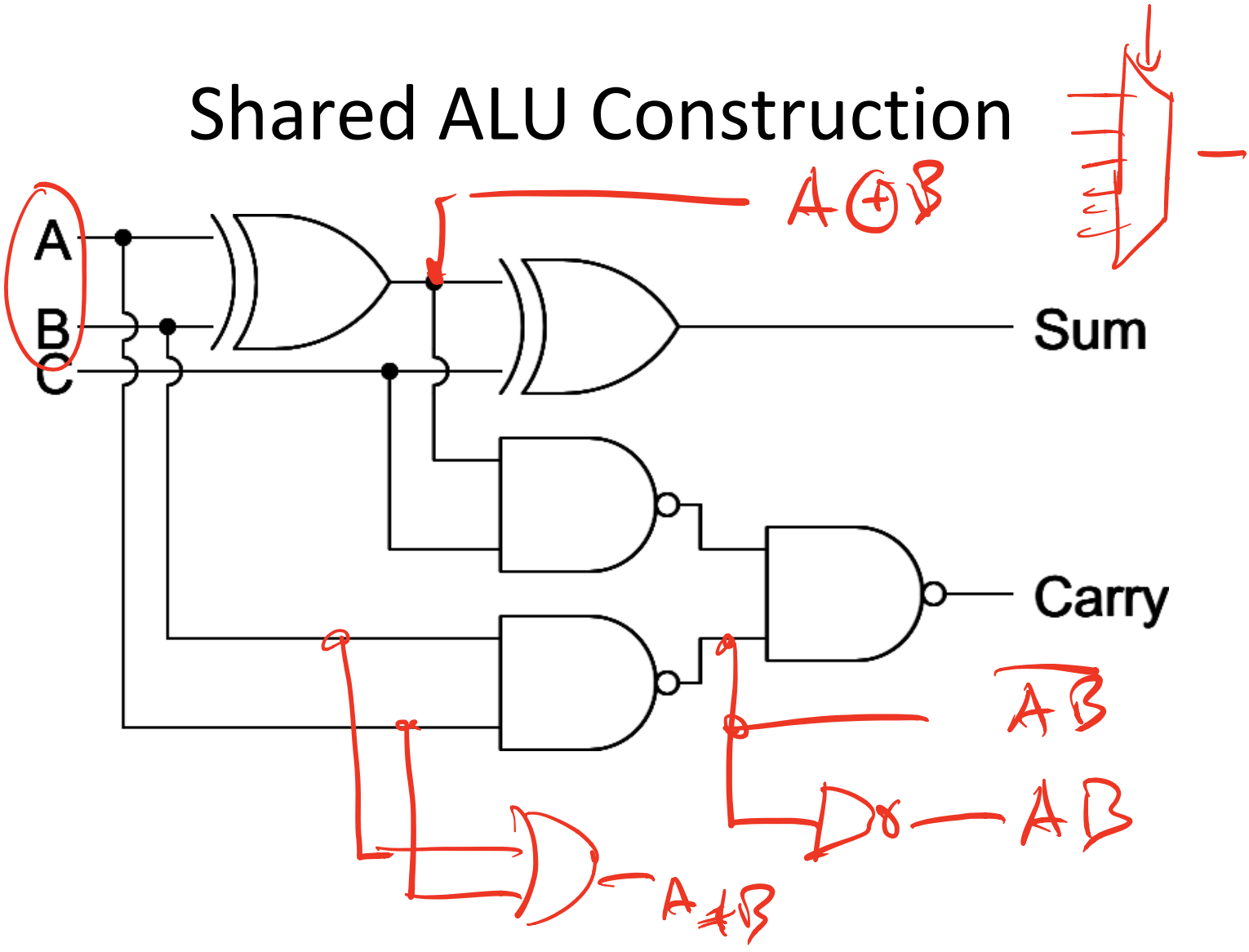
- Route Carries
- Overflow, zero, negative



ALU Construction Summary

- Brute force approach is full parallel with muxes as wide as the number of operations
- Re-use resources for space efficiency
  - Slower, Smaller, Narrower
- Make a SMALL ALU bit slice that does:
  - AND, OR, NAND, NOR, XOR, Add, Subtract

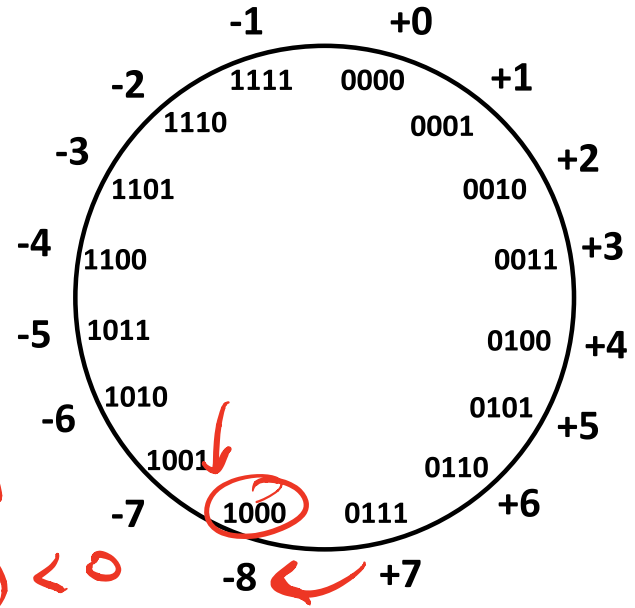
Shared ALU Construction



SLT - How to Check A < B?

- Set less than: if (A<B) then R = 1, else R = 0
  - How do we know if (A<B)
  - Interaction w/overflow?
  - Interaction w/carry out?

- Testing hints:
  - Try 2 random examples
  - Try 2 corner cases
  - Do they all work?



SLT – Test Case Selection

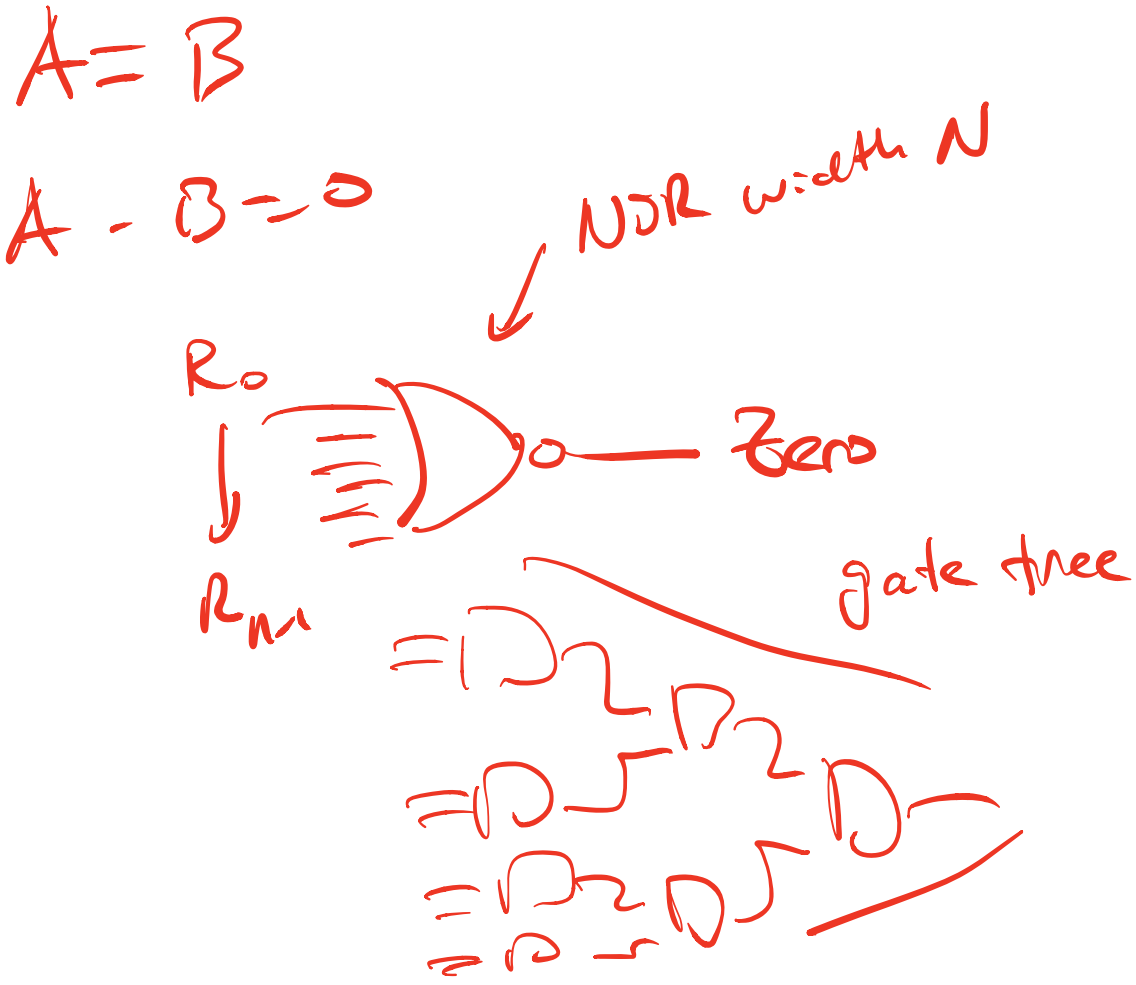
- Create categories of test cases:
  - Equal operands
  - Sign of A, Sign of B, A<B
    - Eight categories – e.g. ++T, -+F, etc
  - Result
    - Positive
    - Negative
    - Positive but overflowed
    - Negative but overflowed

GEQ - How to check $A \geq B$?

- $A \geq B$
- $A - B \geq 0$

$A < B$

What about A == B?



Do a Barrel Roll

- We can multiply or divide a number by 2 by moving it left or right one position

$$\sum_{i=0}^{n-1} 2^i d_i$$

- This is called a “shift”
  - b00000011 d3
  - b00000110 d6
  - b00001100 d12
  - b00011000 d24

01

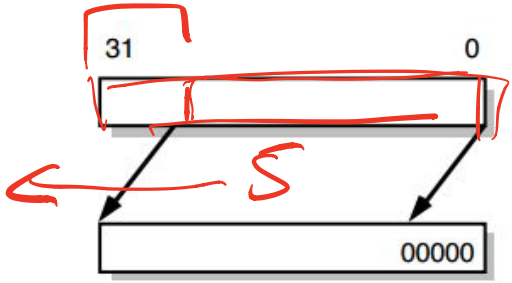
$2^0(1)$

10

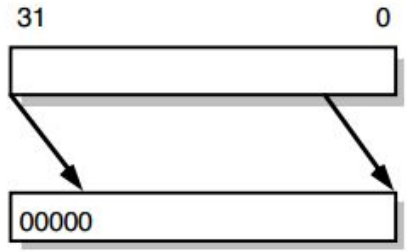
$2^1(1)$

Do a Barrel Roll

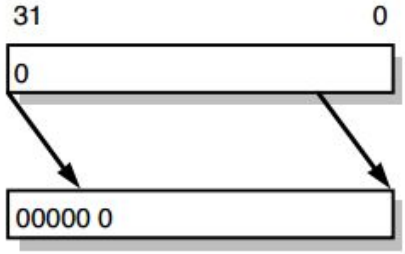
- “Arithmetic” shifts obey 2’s complement
  - Sign extension
- “Logical” shifts do not
  - Assume unsigned
  - Pad zeros
- Barrel Rotate “wraps” around



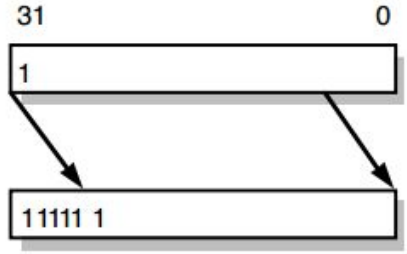
LSL #5



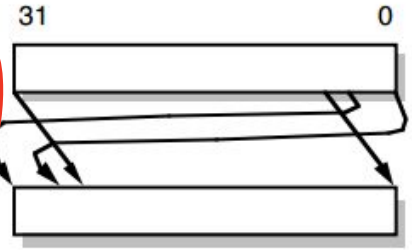
LSR #5



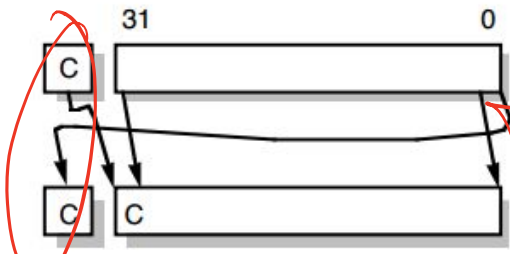
ASR #5, positive operand



ASR #5, negative operand



ROR #5



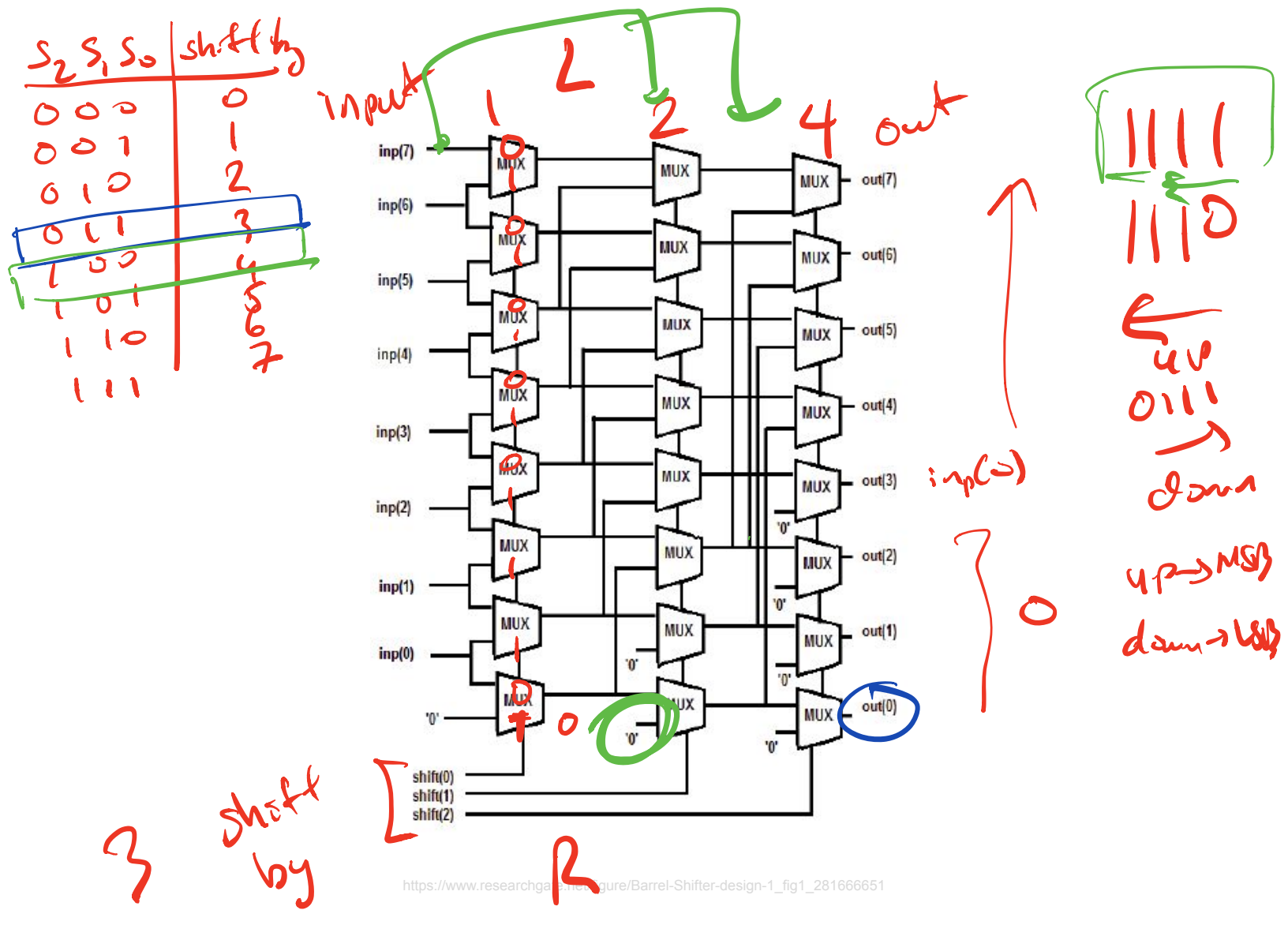
RRX

Create a Shifter

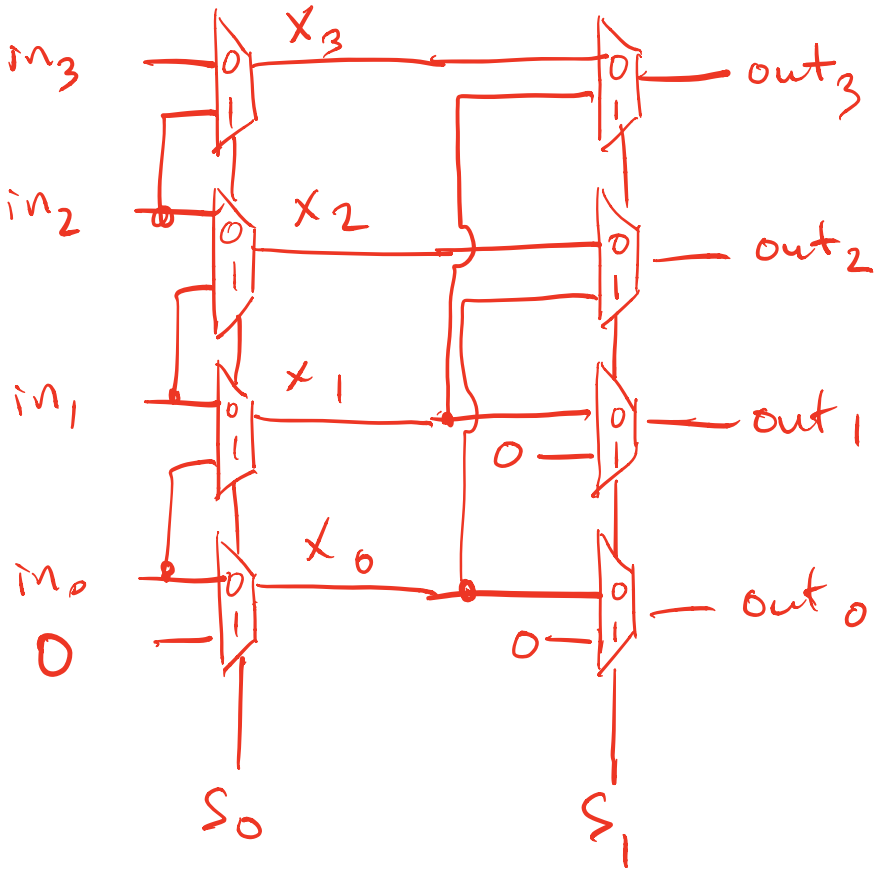
- Start with Logical Shift Left (LSL) only
  - Add the others maybe
- Construction Options:
  - Layers of Shift by R^N
  - Bit slices that shift a single bit an arbitrary amount
- Where in the ALU would you put it?

Growing Shifters

- Chain optional power-of-two shifts
  - Sometimes called a logarithmic shifter
  - Each Layer shifts by either zero or 2^“Layer Number”
  - Each Layer shifts by R^“Layer Number”*[0..R)
- Requires $\log_R(N)$ layers
  - N=32, R=2 -> 5 layers
- How big / fast is a R=2 Logarithmic Shifter?



4-bit Logical Shift "Left"



$$X_3 = \begin{cases} x_3, & S_0 = 0 \\ x_2, & S_0 = 1 \end{cases} \qquad out_3 = \begin{cases} x_3, & S_1 = 0 \\ x_1, & S_1 = 1 \end{cases}$$

$$X_2 = \begin{cases} x_2, & S_0 = 0 \\ x_1, & S_0 = 1 \end{cases} \qquad out_2 = \begin{cases} x_2, & S_1 = 0 \\ x_0, & S_1 = 1 \end{cases}$$

$$X_1 = \begin{cases} x_1, & S_0 = 0 \\ x_0, & S_0 = 1 \end{cases} \qquad out_1 = \begin{cases} x_1, & S_1 = 0 \\ 0, & S_1 = 1 \end{cases}$$

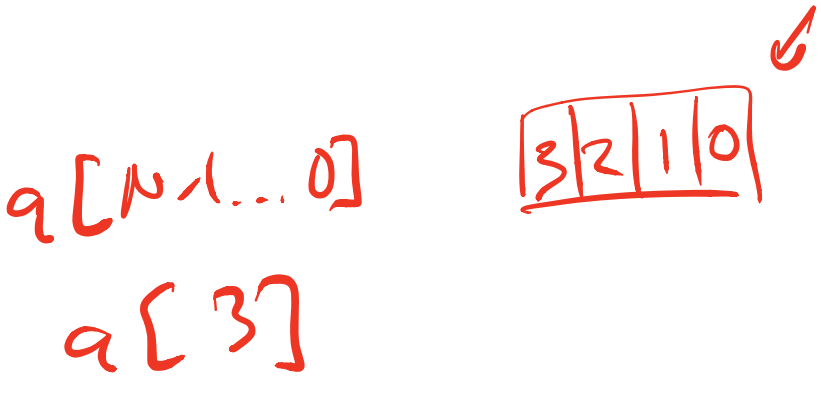
$$X_0 = \begin{cases} x_0, & S_0 = 0 \\ 0, & S_0 = 1 \end{cases} \qquad out_0 = \begin{cases} x_0, & S_1 = 0 \\ 0, & S_1 = 1 \end{cases}$$

Let’s talk Memory!

```
# Program to display the Fibonacci sequence up to n-th term

nterms = int(input("How many terms? "))

# first two terms
n1, n2 = 0, 1
count = 0

# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
else:
   print("Fibonacci sequence:")
   while count < nterms:
       print(n1)
       nth = n1 + n2
       # update values
       n1 = n2
       n2 = nth
       count += 1
```

Memory Attributes

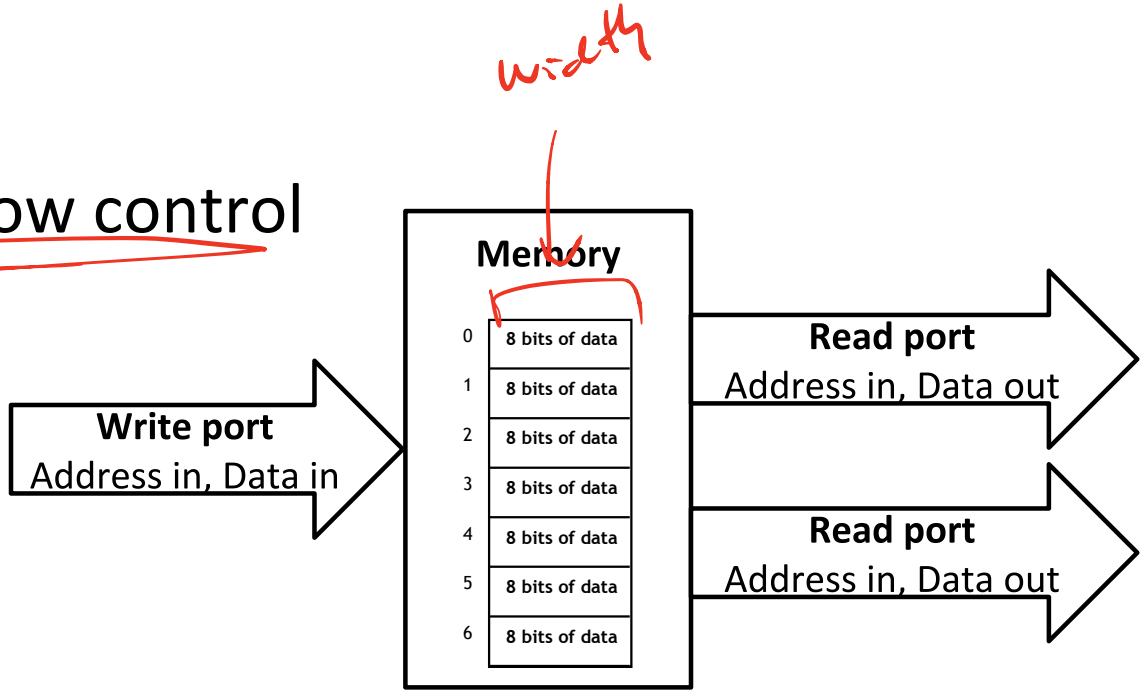
- Mutability:
  - Read/Write
  - Read Only
  - Write Once
  - Reprogrammable
- Volatility:
  - Non-Volatile
  - Volatile
  - Refreshed
- Access Type
  - Random
  - Sequential



Memory ports

Ports have:

- Direction (Read, Write, Read/Write)
- Width
- Latency
- Enable/flow control



Implementing Memory