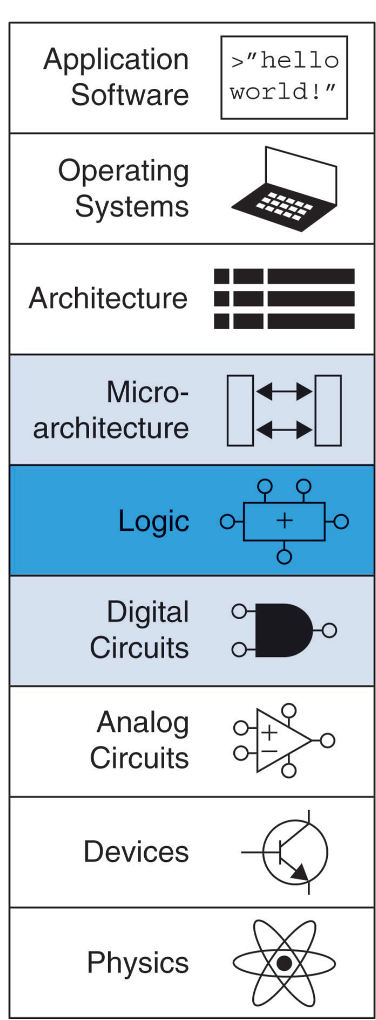
Chapter 5 – Datapath Circuits

***Digital Design and Computer Architecture*: ARM® Edition**

Sarah L. Harris and David Money Harris

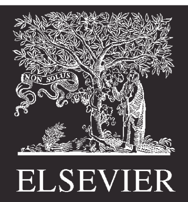
Chapter 5 :: Topics

- Introduction
- Datapath Circuits
- Number Systems
- Sequential Building Blocks
- Memory Arrays
- Logic Arrays



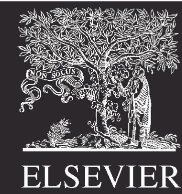
Introduction

- **Digital building blocks:**
  - Gates, multiplexers, decoders, registers, arithmetic circuits, counters, memory arrays, logic arrays
- **Building blocks demonstrate hierarchy, modularity, and regularity:**
  - Hierarchy of simpler components
  - Well-defined interfaces and functions
  - Regular structure easily extends to different sizes
- **Will use these building blocks in Chapter 7 to build microprocessor**



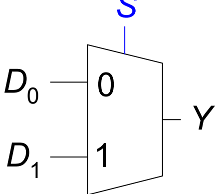
Combinational Building Blocks

- Multiplexers
- Decoders



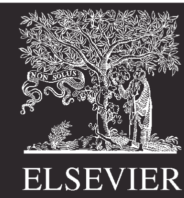
Multiplexer (Mux)

- Selects between one of N inputs to connect to output
- $\log_2 N$-bit select input – control input
- **Example:** **2:1 Mux**



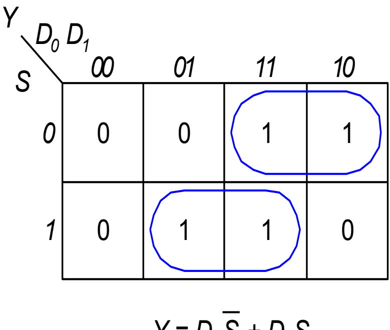
S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

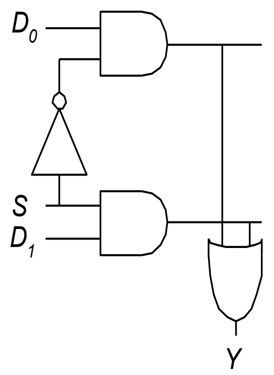
S	Y
0	D_0
1	D_1



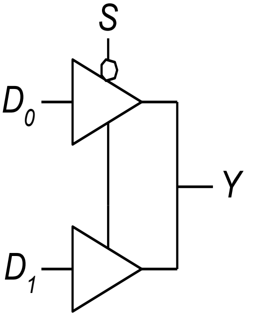
Multiplexer Implementations

- **Logic gates**
  - Sum-of-products form



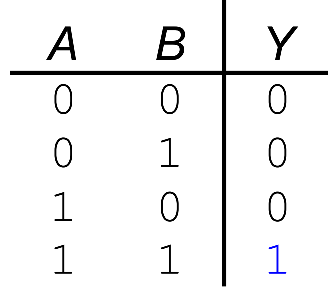


- **Tristates**
  - For an N-input mux, use N tristates
  - Turn on exactly one to select the appropriate input

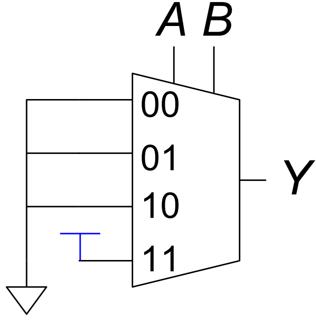


Logic using Multiplexers

Using mux as a lookup table

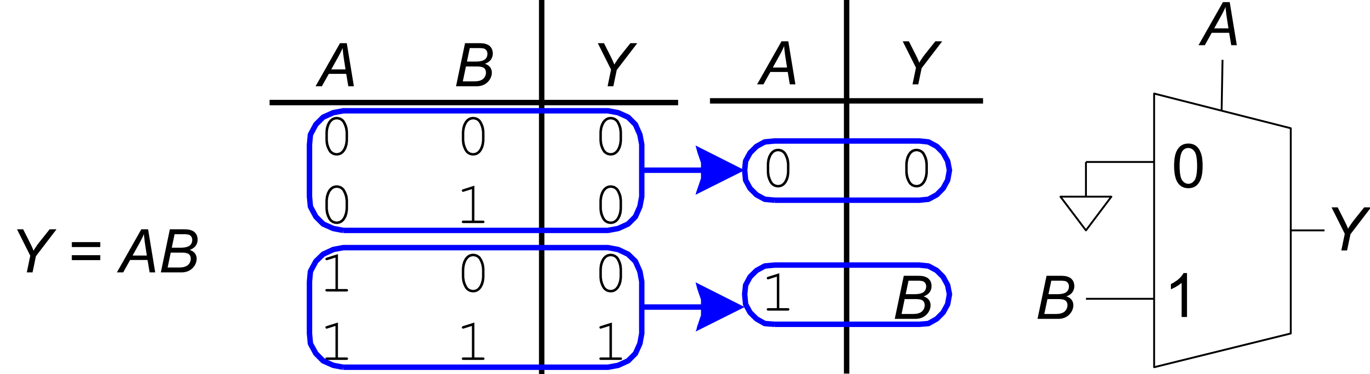


$Y = AB$



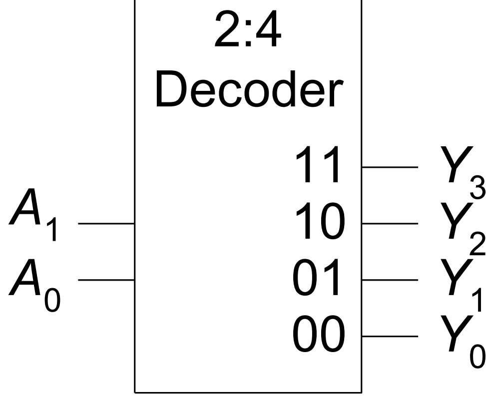
Logic using Multiplexers

Reducing the size of the mux



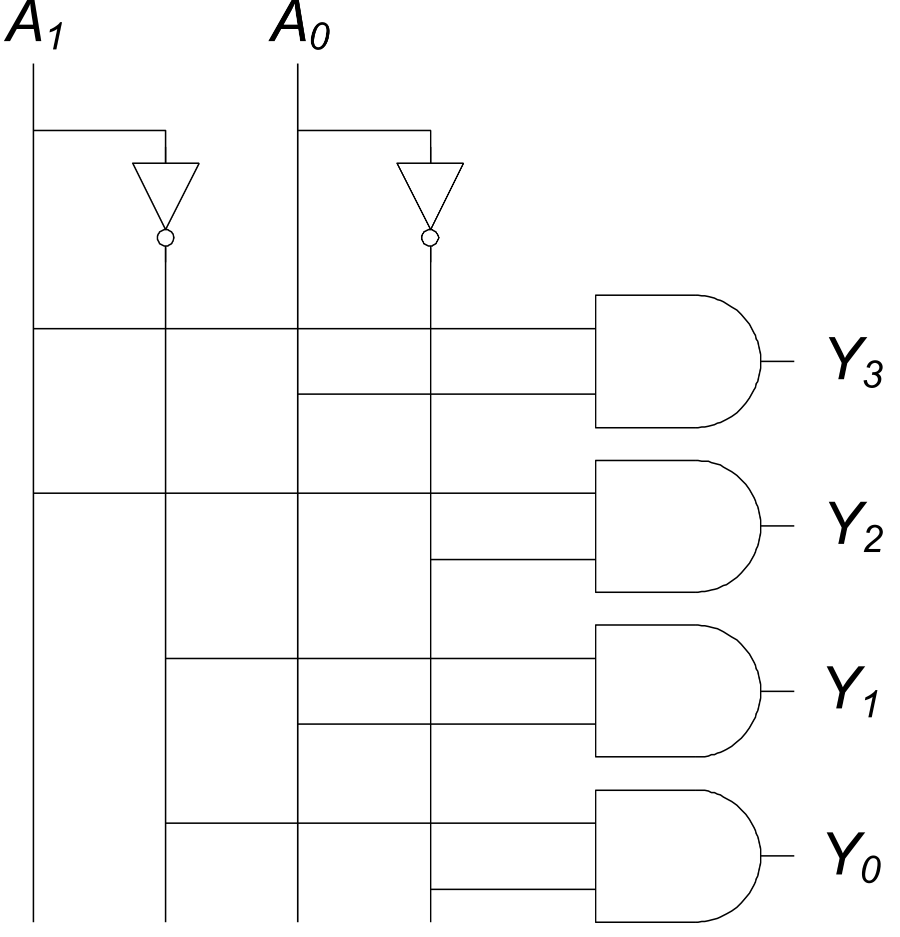
Decoders

- N inputs, 2^N outputs
- **One-hot** outputs: only one output **HIGH** at once



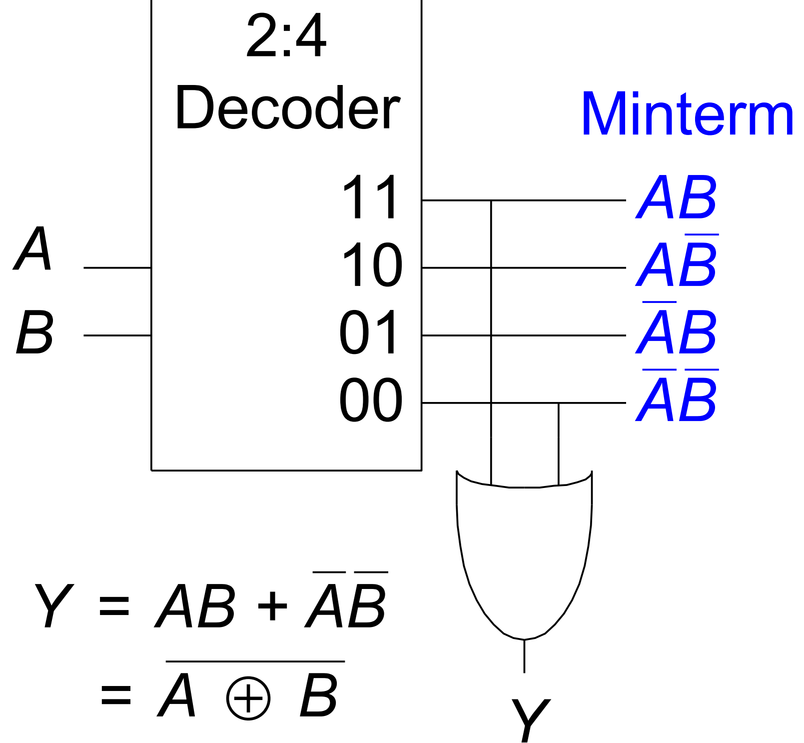
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decoder Implementation



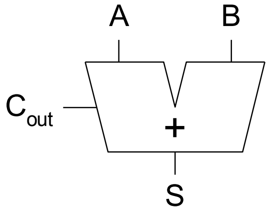
Logic Using Decoders

OR minterms



1-Bit Adders

Half Adder

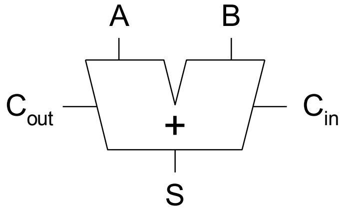


A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

S =

C_{out} =

Full Adder



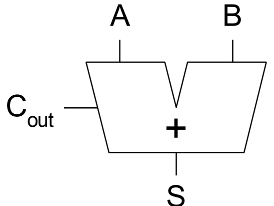
C_{in}	A	B	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

S =

C_{out} =

1-Bit Adders

Half Adder

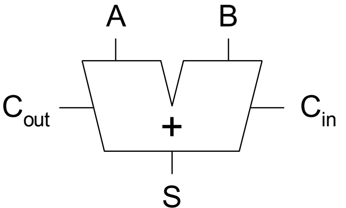


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

S =

C_{out} =

Full Adder



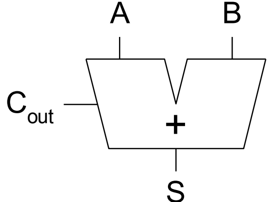
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S =

C_{out} =

1-Bit Adders

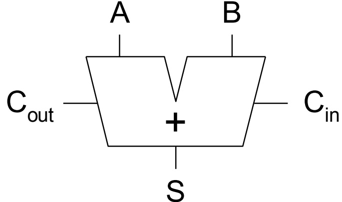
Half Adder



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C_{out} = AB$$

Full Adder

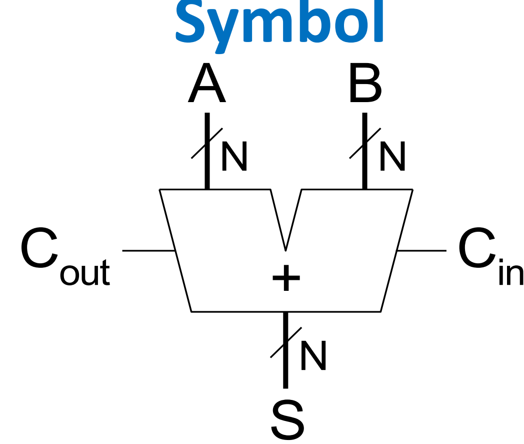


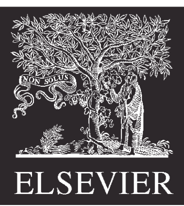
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Multibit Adders (CPAs)

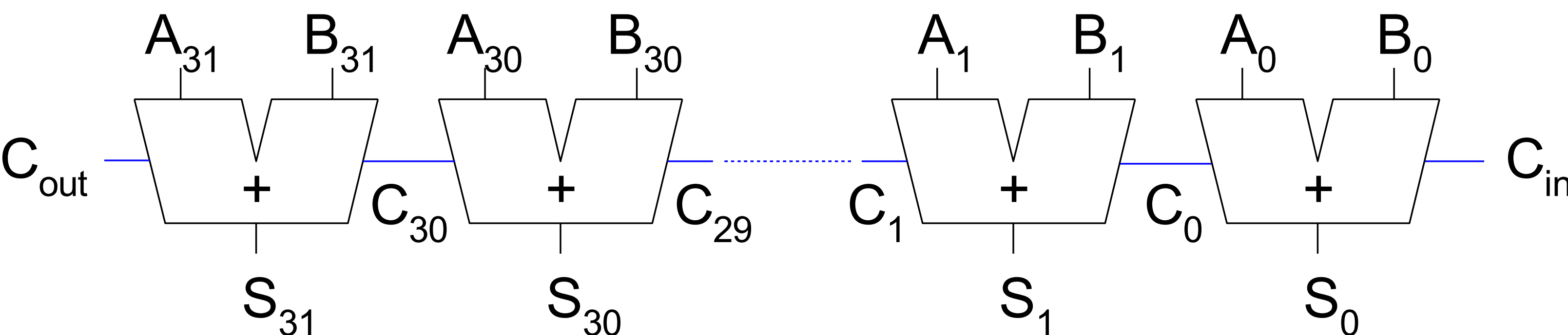
- Types of carry propagate adders (CPAs):
  - Ripple-carry (slow)
  - Carry-lookahead (fast)
  - Prefix (faster)
- Carry-lookahead and prefix adders faster for large adders but require more hardware





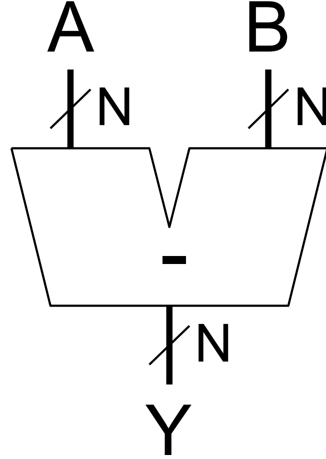
Ripple-Carry Adder

- Chain 1-bit adders together
- Carry ripples through entire chain
- Disadvantage: **slow**



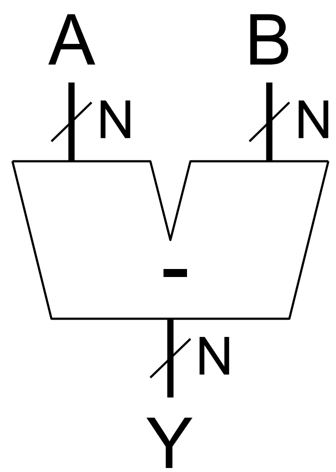
Subtracter

Symbol

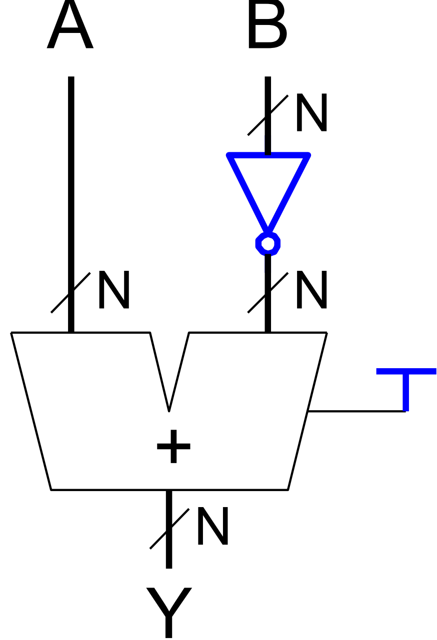


Subtracter

Symbol

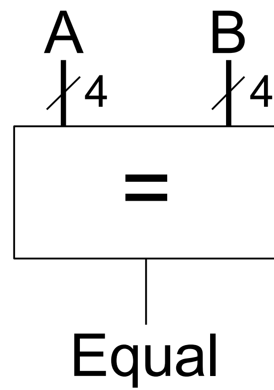


Implementation



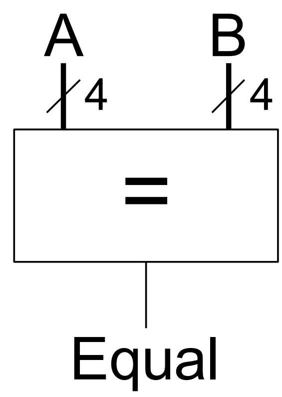
Comparator: Equality

Symbol

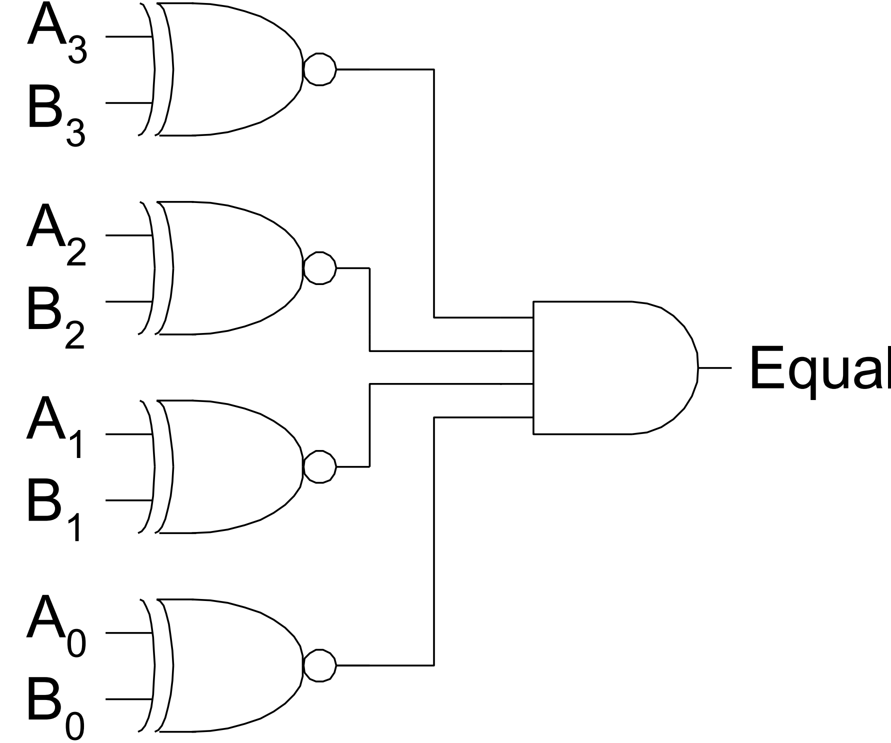


Comparator: Equality

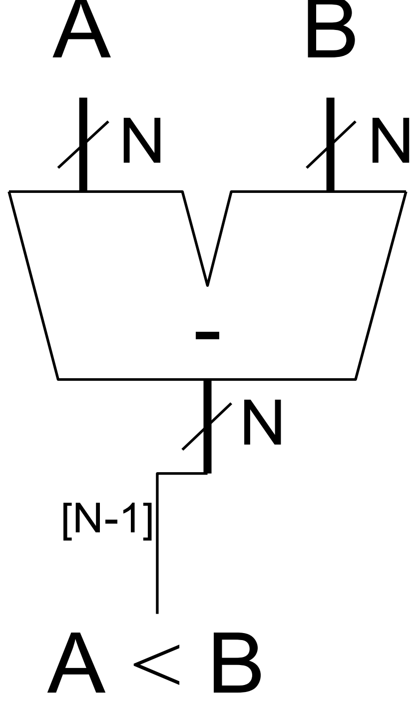
Symbol



Implementation



Comparator: Less Than



ALU: Arithmetic Logic Unit

ALU should perform:

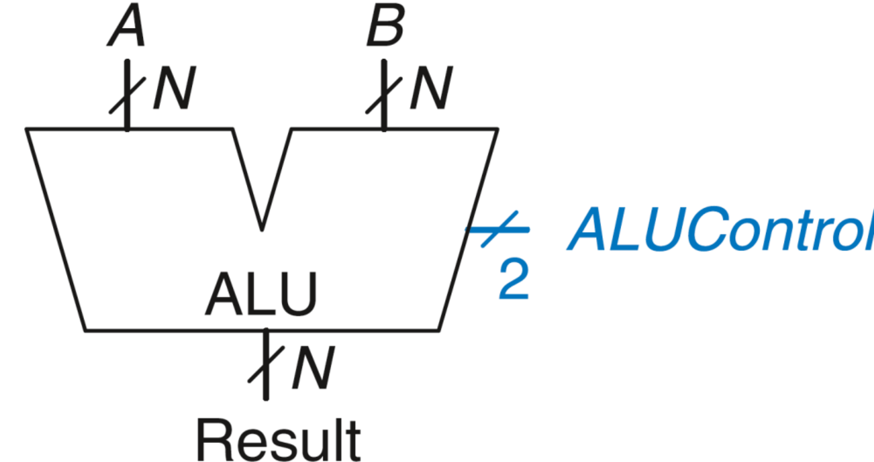
- Addition
- Subtraction
- AND
- OR

ALU: Arithmetic Logic Unit

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR

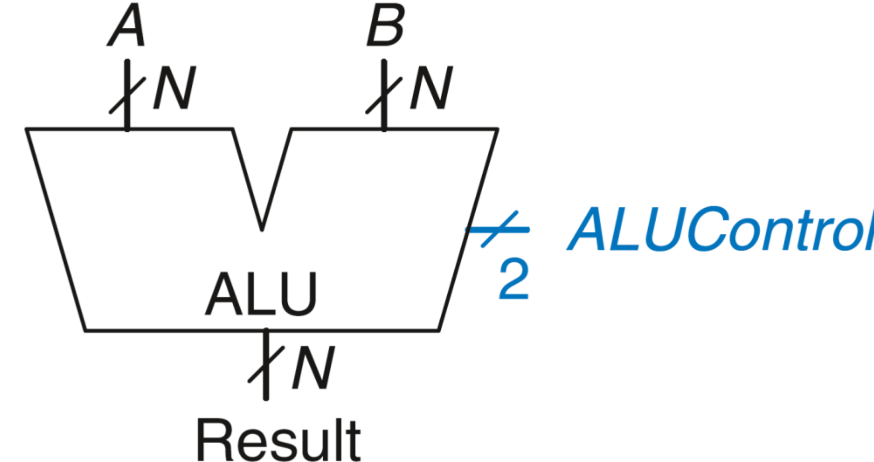
ALU: Arithmetic Logic Unit

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR



ALU: Arithmetic Logic Unit

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR



Example: Perform $A + B$

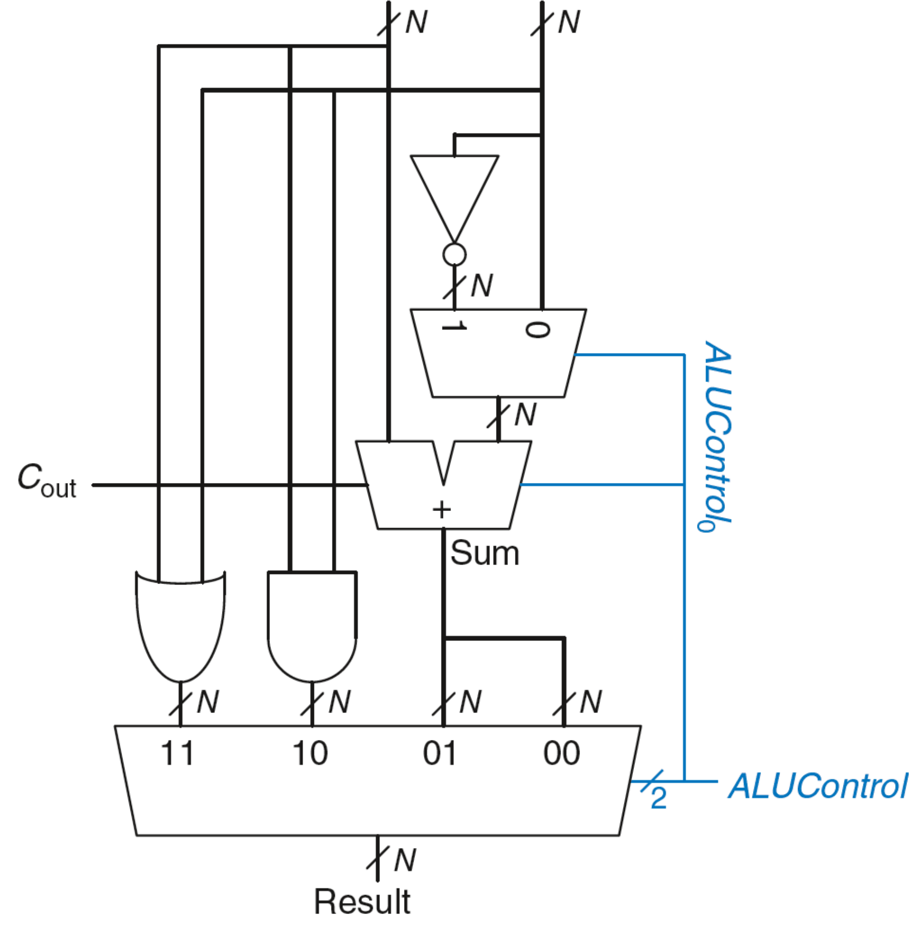
$ALUControl = 00$

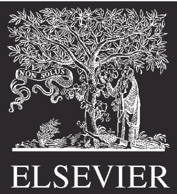
$Result = A + B$

ALU: Arithmetic Logic Unit

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR

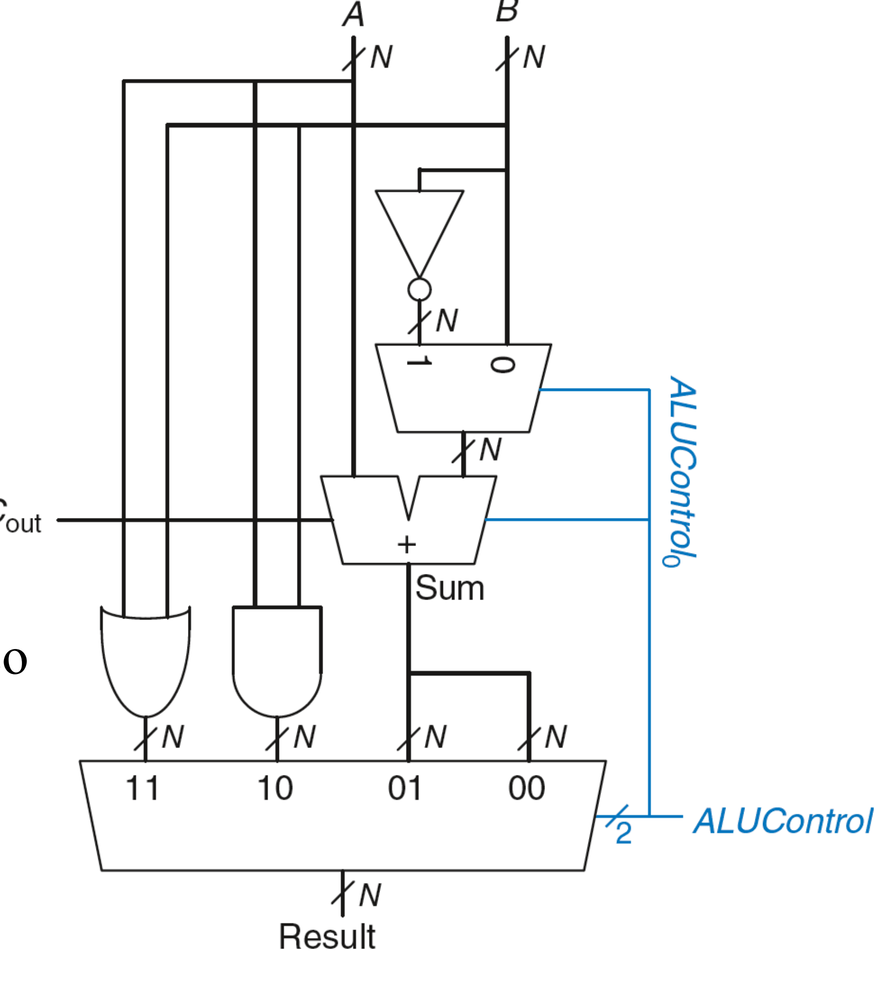
Example: Perform *A* OR *B*





ALU: Arithmetic Logic Unit

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR



Example: Perform *A* OR *B*

$ALUControl_{1:0} = 11$

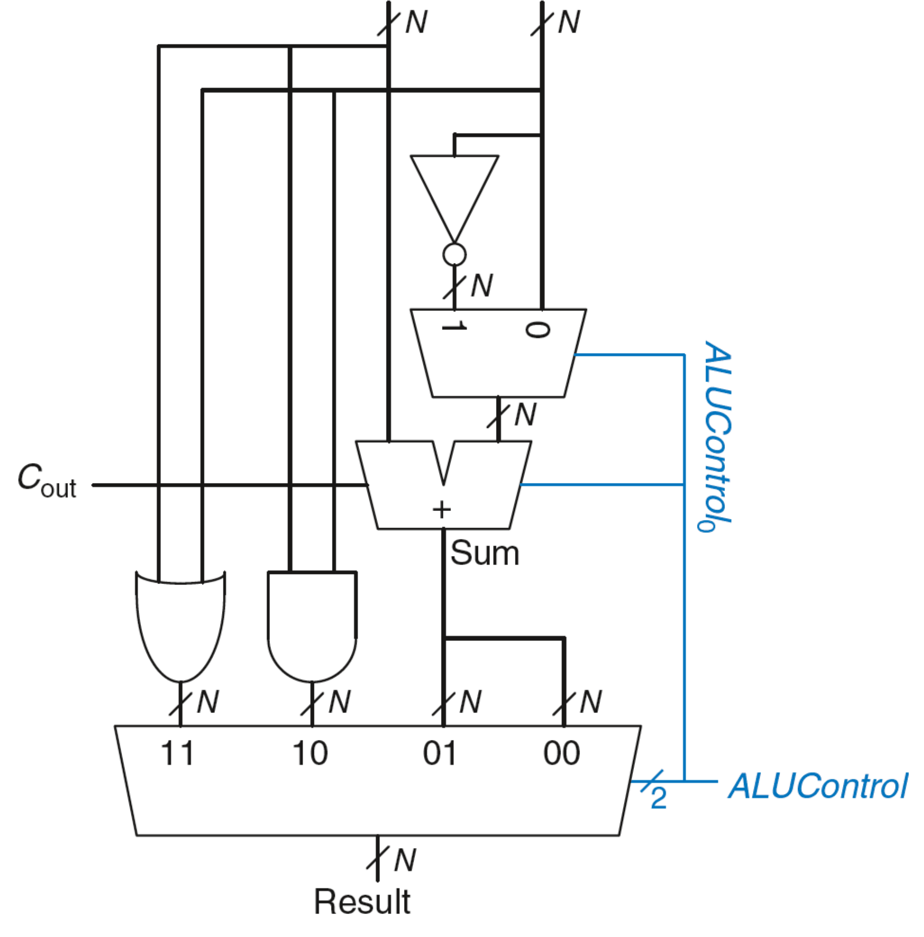
Mux selects output of OR gate as *Result,* so

Result* = *A* OR *B

ALU: Arithmetic Logic Unit

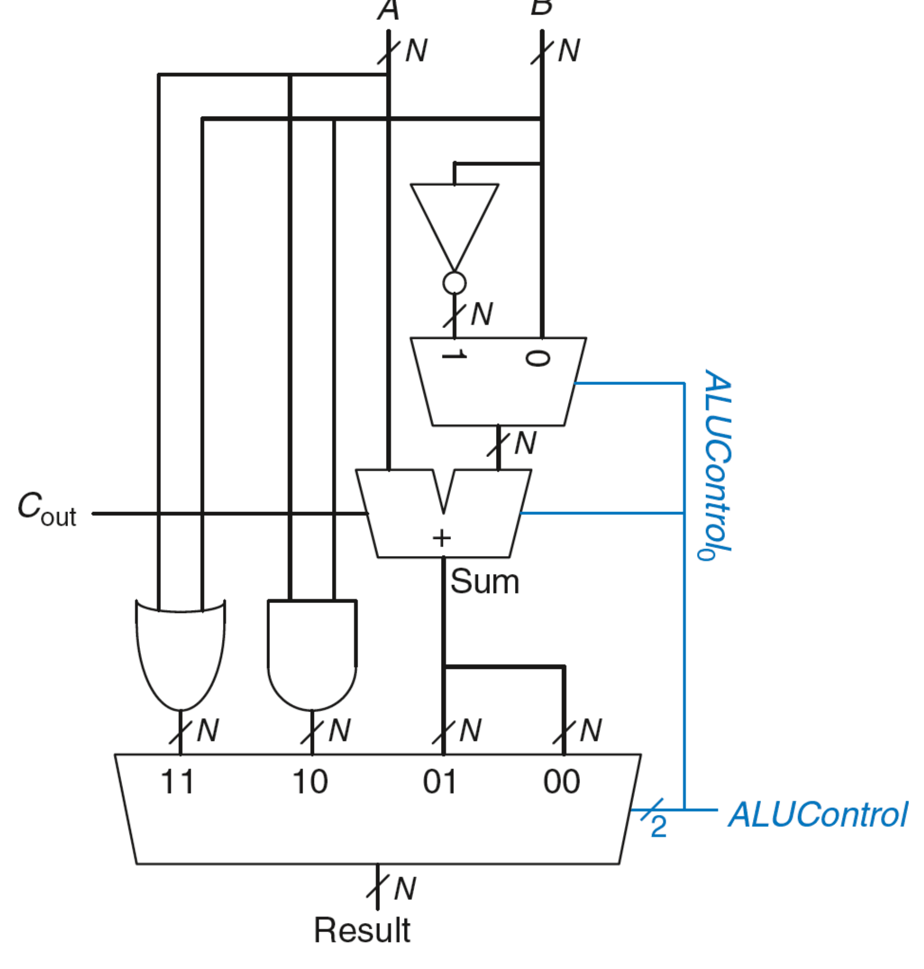
$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR

Example: Perform *A* + *B*



ALU: Arithmetic Logic Unit

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR



Example: Perform *A* + *B*

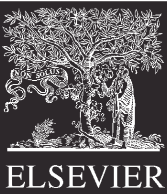
$ALUControl_{1:0}$ = 00

$ALUControl_0$ = 0, so:

- Cin to adder = 0
- 2nd input to adder is *B*

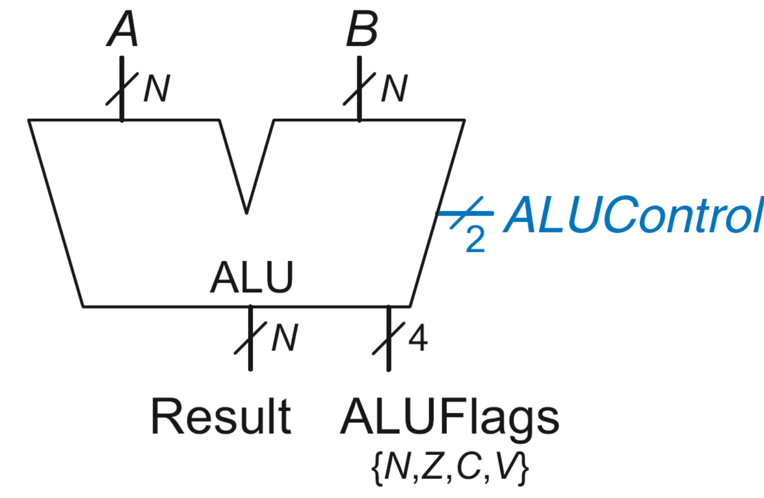
Mux selects *Sum* as *Result,* so

Result* = *A* + *B



ALU with Status Flags

Flag	Description
N	*Result* is **N**egative
Z	*Result* is **Z**ero
C	Adder produces **C**arry out
V	Adder o**V**erflowed



ALU with Status Flags

A
N
B
N
A_{31}
B_{31}
$ALUControl_0$
Sum_{31}
$ALUControl_1$
C_{out}
N
1
0
N
+
Sum
$ALUControl_0$
N N N N
11 10 01 00
2
ALUControl
$Result_{31}$
N
N Z C V
4
oVerflow
Carry
Negative
Zero
Result
ALUFlags

ALU with Status Flags: Negative

***N* = 1** if:

Result is **negative**

So, *N* is connected to most significant bit of *Result*

ALU with Status Flags: Zero

***Z* = 1** if:

all of the bits of *Result* are 0

ALU with Status Flags: Carry

C = 1 if:

C_{out} of Adder is 1

AND

ALU is adding or subtracting (ALUControl is 00 or 01)

ALU with Status Flags: oVerflow

A_{31} B_{31} A B N N

$ALUControl_0$ Sum_{31} $ALUControl_1$ C_{out}

1 0 N + Sum $ALUControl_0$

N N N N 11 10 01 00 2 ALUControl

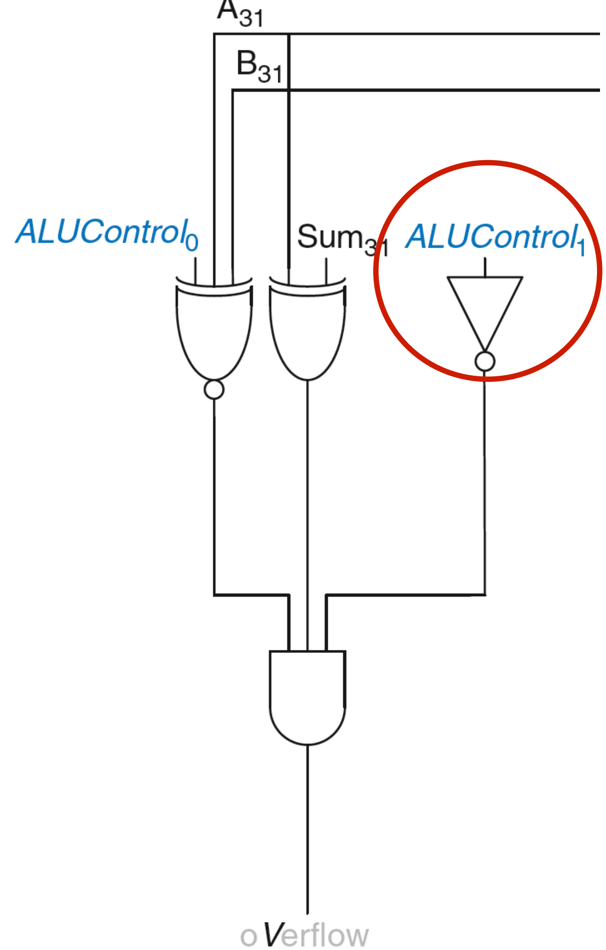
$Result_{31}$ N N Z C V 4

oVerflow Carry Negative Zero Result ALUFlags

***V* = 1** if:

The addition of 2 **same-signed numbers** produces a result with the **opposite sign**

ALU with Status Flags: oVerflow

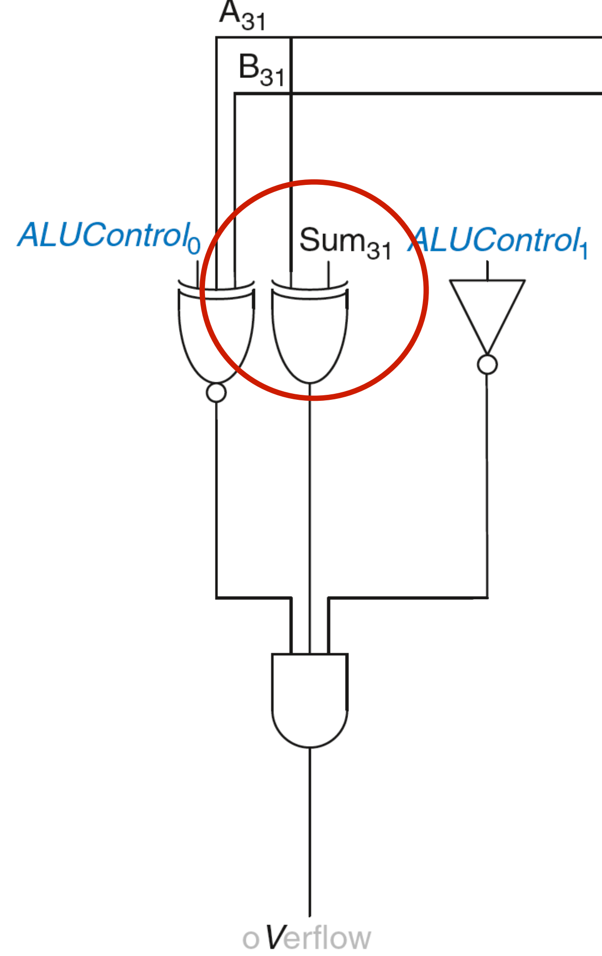


***V* = 1** if:

ALU is performing addition or subtraction

($ALUControl_1 = 0$)

ALU with Status Flags: oVerflow



$V = 1$ if:

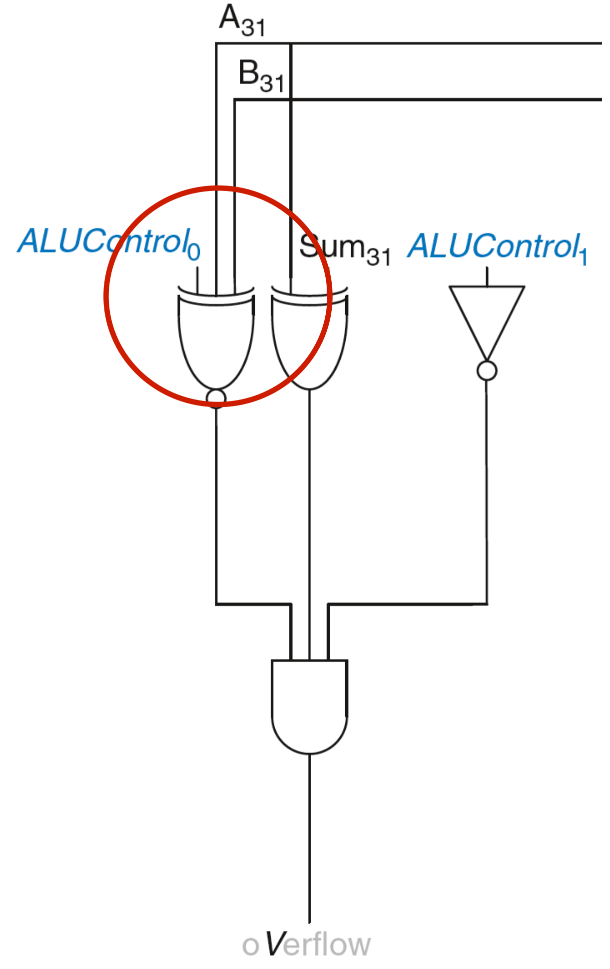
ALU is performing addition or subtraction

($ALUControl_1 = 0$)

AND

A and Sum have opposite signs

ALU with Status Flags: oVerflow



***V* = 1** if:

ALU is performing addition or subtraction

($ALUControl_1$ = 0)

AND

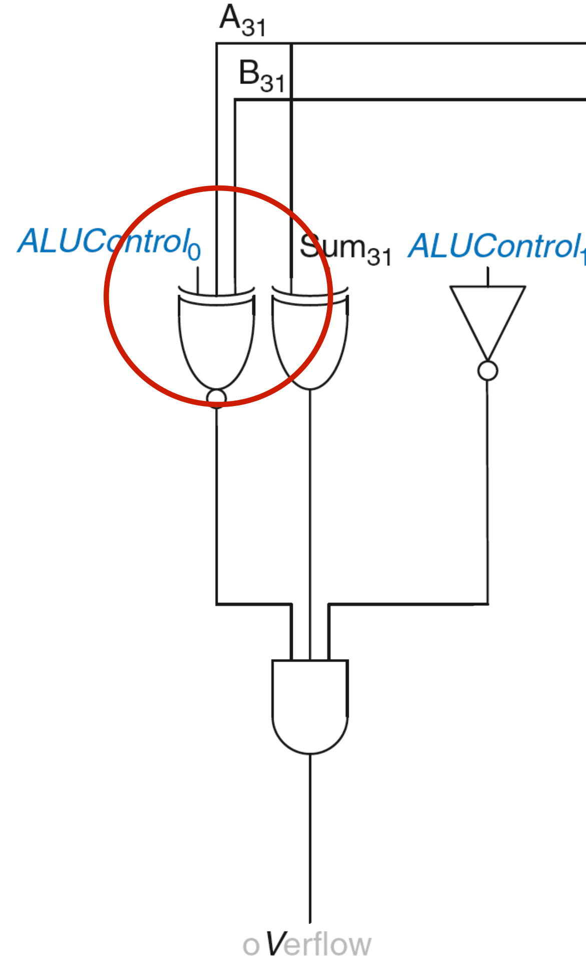
A and Sum have opposite signs

AND

A and B have same signs upon addition **OR**

A and B have different signs upon subtraction

ALU with Status Flags: oVerflow



$V = 1$ if:

ALU is performing addition or subtraction

($ALUControl_1 = 0$)

AND

A and Sum have opposite signs

AND

A and B have same signs upon addition

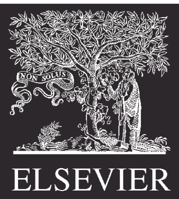
($ALUControl_0 = 0$) **OR**

A and B have different signs upon subtraction

($ALUControl_0 = 1$)

ALU with Status Flags

A_{31}
B_{31}
A
B
N
$ALUControl_0$
Sum_{31}
$ALUControl_1$
C_{out}
1
0
+
Sum
$ALUControl_0$
11
10
01
00
2
ALUControl
$Result_{31}$
N Z C V
4
oVerflow
Carry
Negative
Zero
Result
ALUFlags



Shifters

Logical shifter: shifts value to left or right and fills empty spaces with 0's

- Ex: 11001 >> 2 =
- Ex: 11001 << 2 =

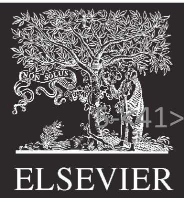
Arithmetic shifter: same as logical shifter, but on right shift, fills empty spaces with the old most significant bit (msb)

- Ex: 11001 >>> 2 =
- Ex: 11001 <<< 2 =

Rotator: rotates bits in a circle, such that bits shifted off one end are shifted into the other end

- Ex: 11001 ROR 2 =
- Ex: 11001 ROL 2 =

Copyright © 2007 Elsevier



Shifters

Logical shifter:

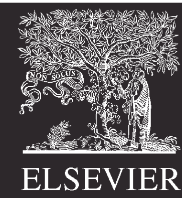
- Ex: 11001 >> 2 = 00110
- Ex: 11001 << 2 = 00100

Arithmetic shifter:

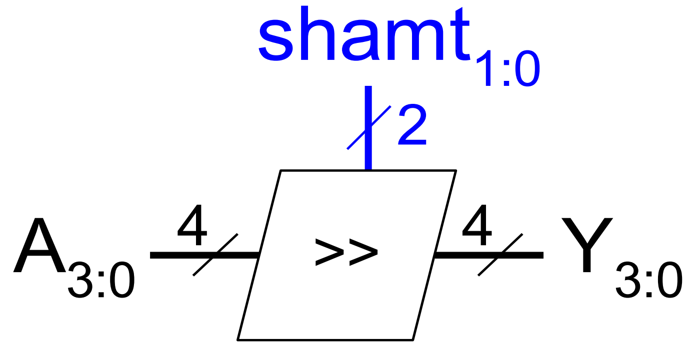
- Ex: 11001 >>> 2 = 11110
- Ex: 11001 <<< 2 = 00100

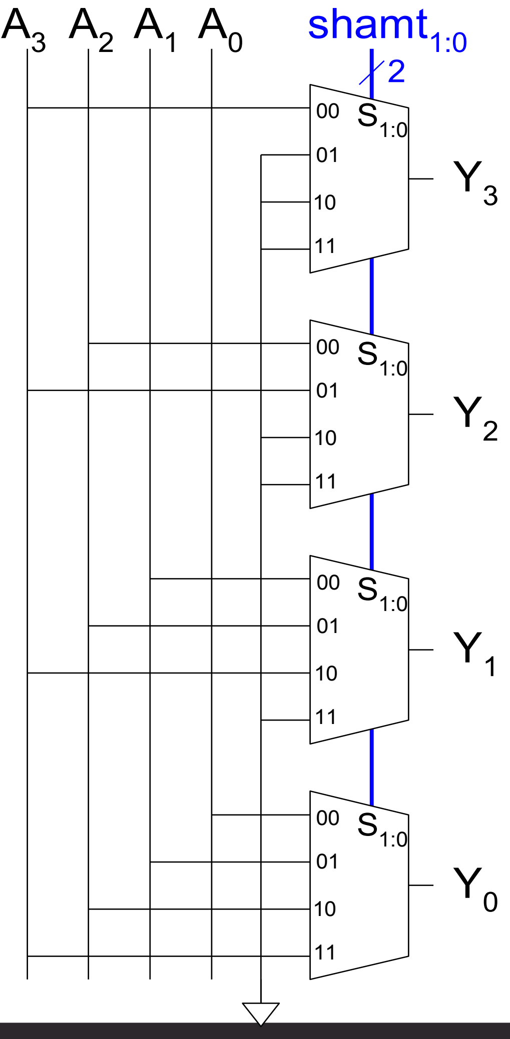
Rotator:

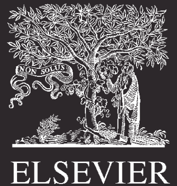
- Ex: 11001 ROR 2 = 01110
- Ex: 11001 ROL 2 = 00111



Shifter Design

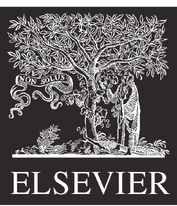






Shifters as Multipliers, Dividers

- $A << N = A \times 2^N$
  - **Example:** 00001 << 2 = 00100 ($1 \times 2^2 = 4$)
  - **Example:** 11101 << 2 = 10100 ($-3 \times 2^2 = -12$)
- $A >>> N = A \div 2^N$
  - **Example:** 01000 >>> 2 = 00010 ($8 \div 2^2 = 2$)
  - **Example:** 10000 >>> 2 = 11100 ($-16 \div 2^2 = -4$)



Multipliers

- **Partial products** formed by multiplying a single digit of the multiplier with multiplicand
- **Shifted** partial products **summed** to form result

Decimal		Binary
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial products	0101
+ 920		0101
		0101
		+ 0000
9660	result	0100011

230 x 42 = 9660

5 x 7 = 35

4 x 4 Multiplier