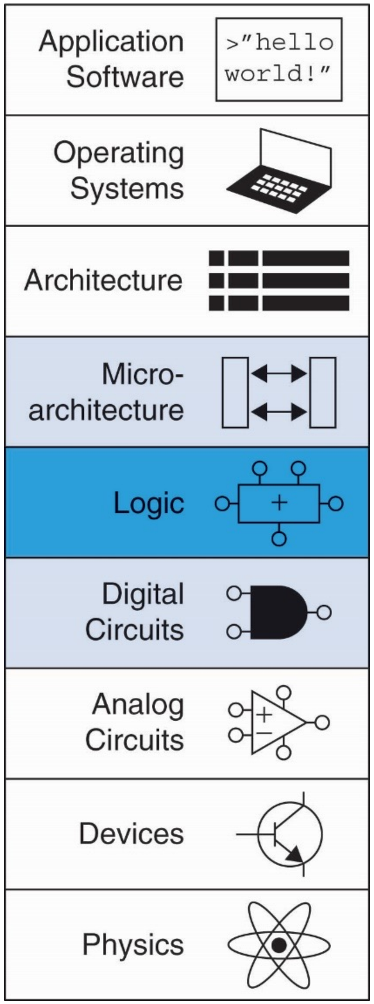
Chapter 5 :: Digital Building Blocks

Digital Design and Computer Architecture

David Money Harris and Sarah L. Harris

Chapter 5 :: Topics

- 5.1 Introduction
- 5.2 Arithmetic Circuits
- 5.3 Number Systems
- 5.4 Sequential Building Blocks
- 5.5 Memory Arrays
- 5.6 Logic Arrays

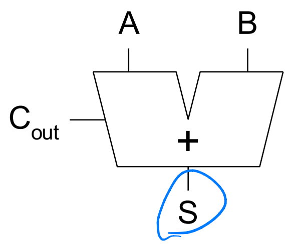


5.1 Introduction

- 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 extended to different sizes
- Will use many of these building blocks to build a microprocessor in Chapter 7

5.2 Arithmetic Circuit
1-Bit Adders

Half Adder

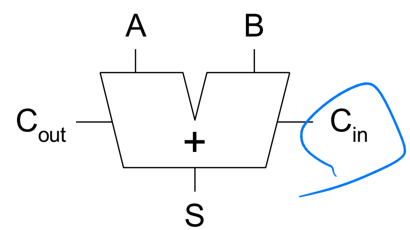


A 1
$+$ B 1
C_{out} S

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

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

S =
C_{out} =

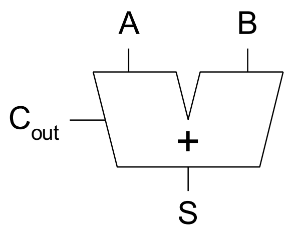
1-Bit Adders

$+ = \text{OR}$

$\cdot = \text{AND}$

$\oplus = \text{XOR}$

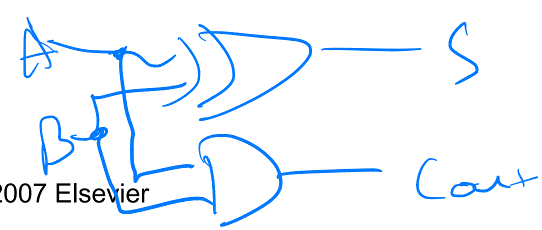
Half Adder



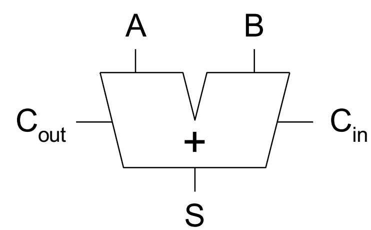
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$

$C_{out} = A \cdot B$



Full Adder



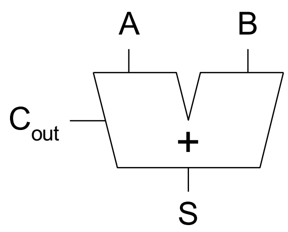
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S = A \oplus B \oplus C_{in}$

$C_{out} = AB + BC_{in} + AC_{in}$

1-Bit Adders

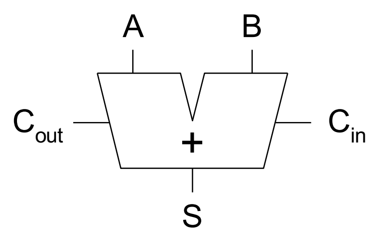
Half Adder



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



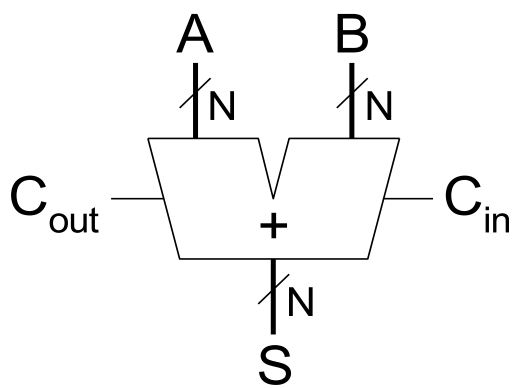
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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

Multibit Adder, also called CPA

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

Symbol



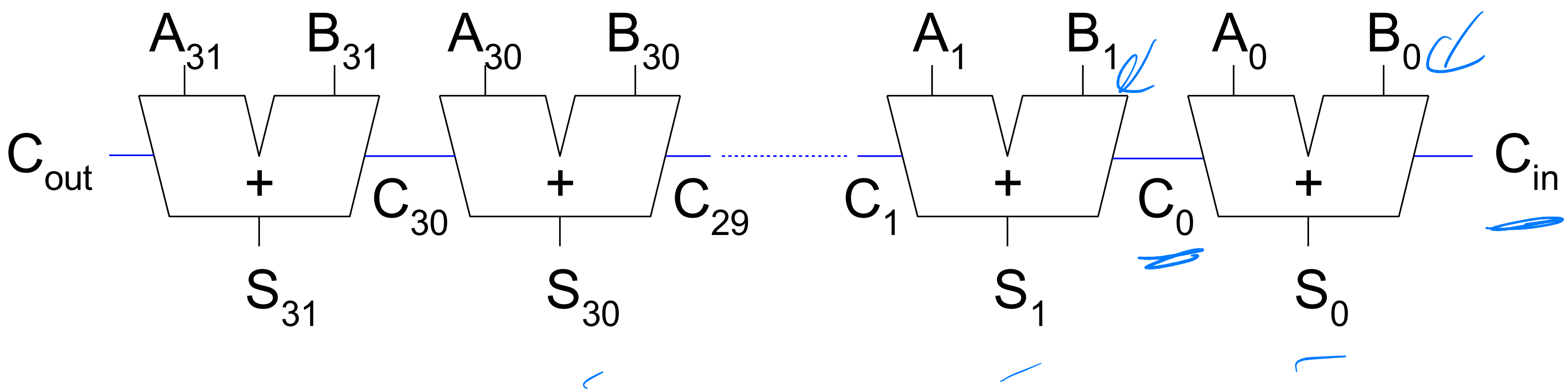
N= any
: integer
number

Ripple-Carry Adder

full

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

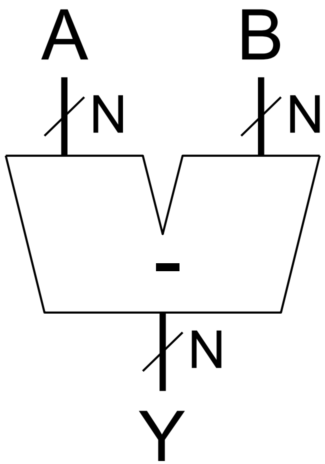
$$
\begin{array}{rcccc}
 & C_2 & C_1 & C_0 & C_{in} \\
 & A_3 & A_2 & A_1 & A_0 \\
+ & B_3 & B_2 & B_1 & B_0 \\
\hline
C_{out} & S_3 & S_2 & S_1 & S_0
\end{array}
$$



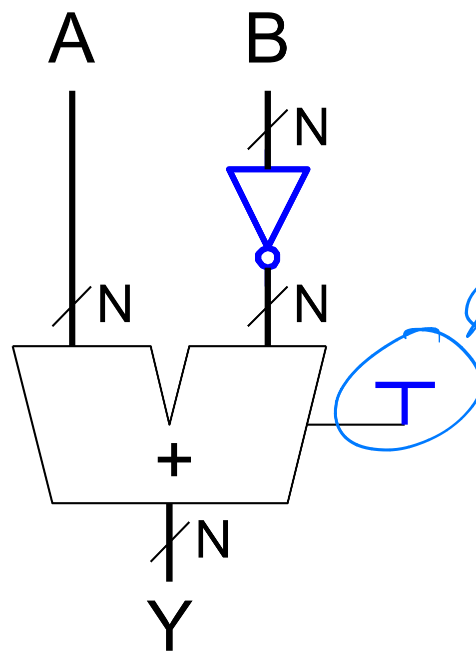
5.2.2Subtracter

$A - B = A + (-B)$

Symbol



Implementation

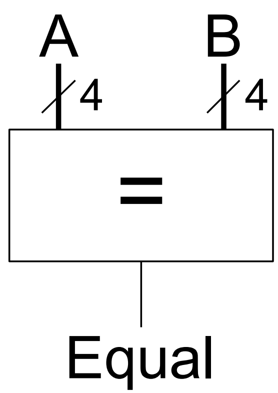


V_{cc} or power

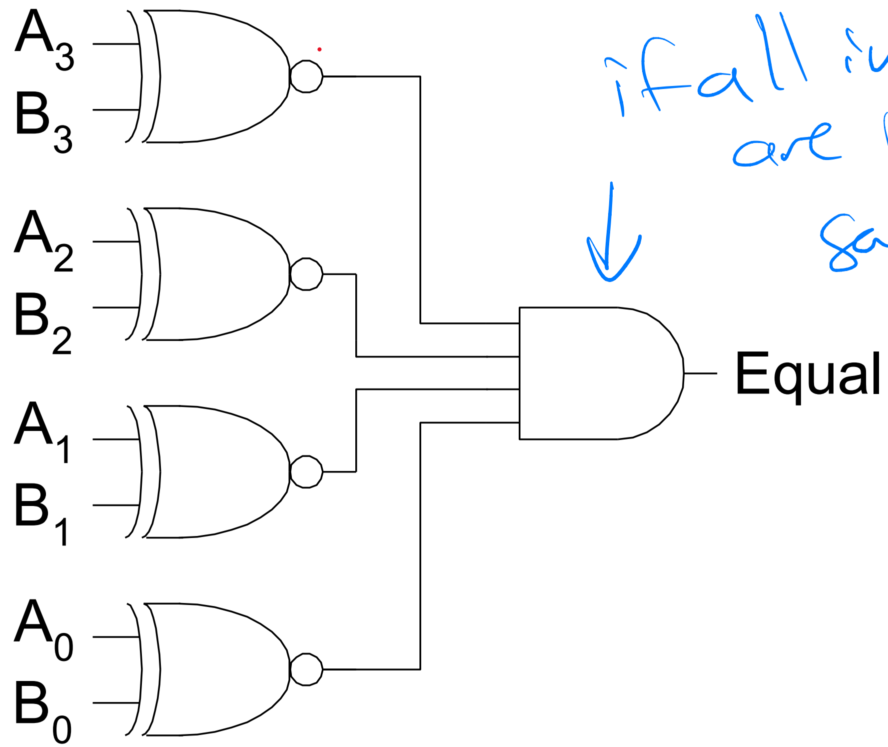
5.2.3 Comparator: Equality

true if inputs the same

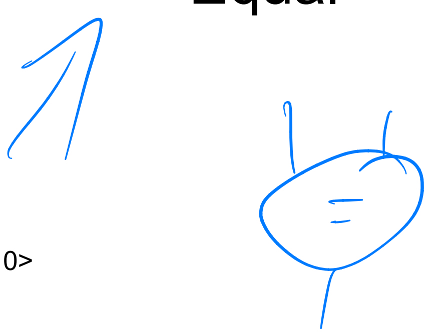
Symbol



Implementation



if all inputs are the same

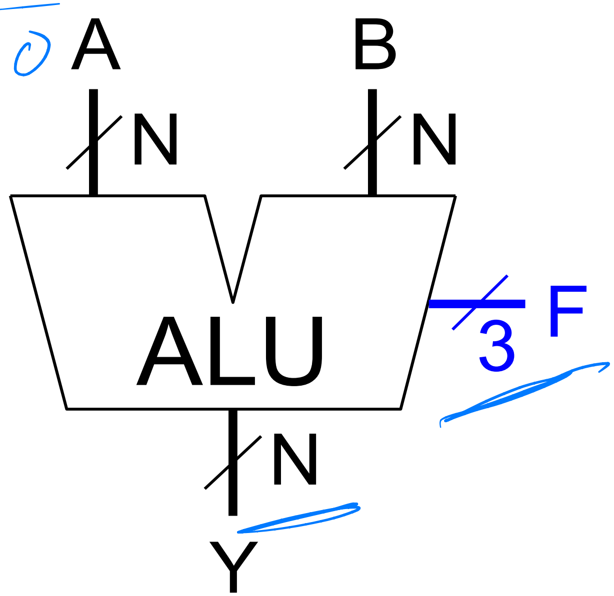


5.2.4 Arithmetic Logic Unit (ALU)

→ 1111111 00000000

define the operation ↓

& 0110
0110



bitwise operations

$F_{2:0}$	Function	
000	A & B	AND
001	A | B	OR
010	A + B	ADD
011	not used	
100	A & ~B	
101	A | ~B	
110	A - B	SUB
111	SLT	

← set if less than

Defined in Lab 2

true if A < B,
false otherwise

5.2.5 Shifters

- **Logical shifter:** shifts value to left or right and fills empty spaces with 0's
  - Ex: 11001 >> 2 = 00110
  - Ex: 11001 << 2 = 00100

bits represent a numeric value

- **Arithmetic shifter:** same as logical shifter, but on right shift, fills empty spaces with the old most significant bit (msb).
  - Ex: 11001 >>> 2 = 11110
  - Ex: 11001 <<< 2 = 00100
- **Rotator:** rotates bits in a circle, such that bits shifted off one end are shifted into the other end
  - Ex: 11001 ROR 2 = 01110
  - Ex: 11001 ROL 2 = 00111

Shifters

- **Logical shifter:** shifts value to left or right and fills empty spaces with 0's
  - Ex: 11001 >> 2 = 00110
  - Ex: 11001 << 2 = 00100

- **Arithmetic shifter:** same as logical shifter, but on right shift, fills empty spaces with the old most significant bit (msb).
  - Ex: 11001 >>> 2 = **11**110
  - Ex: 11001 <<< 2 = 00100

- **Rotator:** rotates bits in a circle, such that bits shifted off one end are shifted into the other end
  - Ex: 11001 ROR 2 = 01110
  - Ex: 11001 ROL 2 = 00111

Shifters as Multipliers and Dividers

- A left shift by N bits multiplies a number by 2^N
  - Ex: 00001 << 2 = 00100 ($1 \times 2^2 = 4$)
  - Ex: 11101 << 2 = 10100 ($-3 \times 2^2 = -12$)

$\underbrace{\quad}_{-3}$ $-16+4 = -12$

- The arithmetic right shift by N divides a number by 2^N
  - Ex: 01000 >>> 2 = 00010 ($8 \div 2^2 = 2$)
  - Ex: 10000 >>> 2 = 11100 ($-16 \div 2^2 = -4$)

-16 $-16+8+4 = -4$

5.3 Number Systems

- What kind of numbers do you know ?
- How to represent using binary representations?
  - Positive numbers
    - Unsigned binary
  - Negative numbers
    - Two's complement
    - Sign/magnitude numbers
- What about fractions?

$\pm 1.011 \times 2^{N}$

fixed point

$\frac{1}{2}$ $\frac{1}{4}$ $\frac{1}{8}$ $\frac{1}{16}$

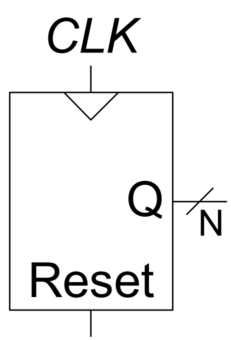
$2^3\ 2^2\ 2^1\ 2^0\ .\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$

5.4 Sequential Building Blocks

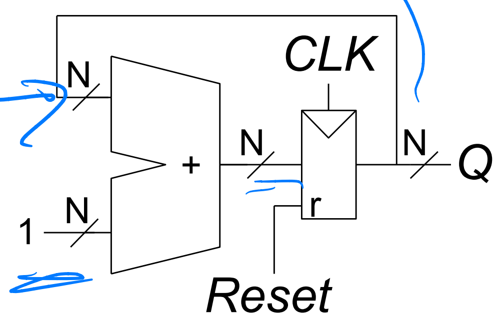
5.4.1 Counters

- Increments on each clock edge.
- Used to cycle through numbers. For example,
  - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001…
- Example uses:
  - Digital clock displays
  - Program counter: keeps track of current instruction executing

Symbol



Implementation



0001

register