# Basics of Logic Design
# Arithmetic Logic Unit (ALU)

**CPS 104**

**Lecture 9**

## Today's Lecture

- **Homework #3 Assigned Due March 3**
- **Project Groups assigned & posted to blackboard.**
- **Project Specification is on Web Due April 19**
- **Building the building blocks…**

**Outline**

- **Review**
- **Digital building blocks**
- **An Arithmetic Logic Unit (ALU)**

**Reading**

**Appendix B, Chapter 3**

## Review: Digital Design

- Logic Design, Switching Circuits, Digital Logic

Recall: Everything is built from transistors

- A transistor is a switch
- It is either on or off
- On or off can represent True or False

Given a bunch of bits (0 or 1)…

- Is this instruction a lw or a beq?
- What register do I read?
- How do I add two numbers?
- Need a method to reason about complex expressions



## Review: Boolean Functions

- Boolean functions have arguments that take two values ({T,F} or {0,1}) and they return a single or a set of ({T,F} or {0,1}) value(s).
- Boolean functions can always be represented by a table called a "Truth Table"
- Example: F: {0,1}3 -> {0,1}2

| a | b | c | $f_1$ | $f_2$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

## Review: Boolean Functions and Expressions

$$F(A, B, C) = (A * B) + (\sim A * C)$$

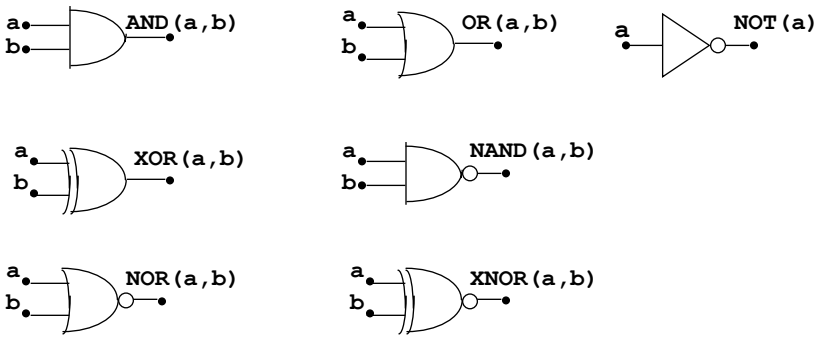
| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



## Review: Boolean Gates

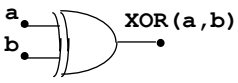
- **Gates** are electronics devices that implement simple Boolean functions

**Examples**



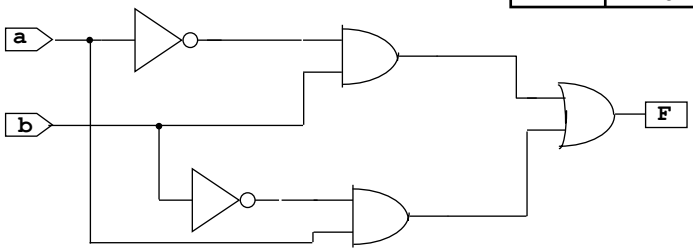
## Boolean Functions, Gates and Circuits

- **Circuits are made from a network of gates. (function compositions).**




**F = ~a*b + ~b*a**

| a | b | XOR(a,b) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |






## Digital Design Examples

**Input: 2 bits representing an unsigned number (n)**
**Output: $n^2$ as unsigned binary number**

**Input: 2 bits representing an unsigned number (n)**
**Output: 3-n as unsigned binary number**

## More Design Examples

- X is a 3-bit quantity

1. Write a logic function that is true if and only if X contains at least two 1s.

2. Implement the logic function from problem 1. using only AND, OR and NOT gates. (Note there are no constraints on the number of gate inputs.) By implement, I mean draw the circuit diagram.

3. Write a logic function that is true if and only if X, when interpreted as an unsigned binary number, is greater than the number 5.

4. Implement the logic function from problem 3. using only AND, OR and NOT gates. (Note there are no constraints on the number of gate inputs.)



## Parity Example

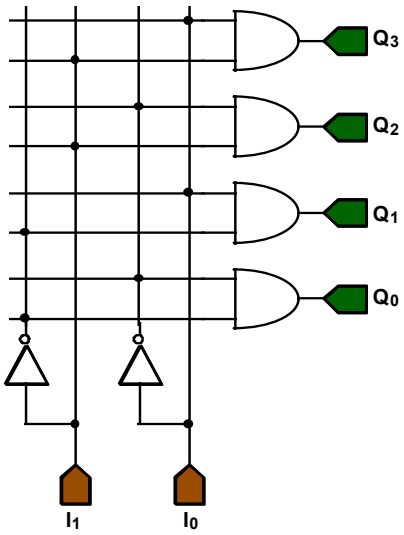
- The parity code of a binary word counts the number of ones in a word. If there are an even number of ones the parity code is 0, if there are an odd number of ones the parity code is 1. For example, the parity of 0101 is 0, and the parity of 1101 is 1.
- Construct the truth table for a function that computes the parity of a four-bit word. Implement this function using AND, OR and NOT gates. (Note there are no constraints on the number of gate inputs.)

## Design Example

- Consider machine with 4 registers
- Given 2-bit input (register specifier, $I_1$, $I_0$)
- Want one of 4 output bits ($O_3$-$O_0$) to be 1
  - E.g., allows a single register to be accessed
- What is the circuit for this?



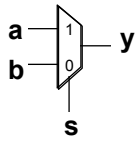

## Circuit Example: Decoder



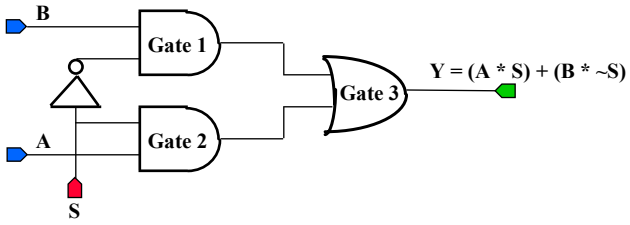

| $I_1$ | $I_0$ | $Q_0$ | $Q_1$ | $Q_2$ | $Q_3$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

# Circuit Example: 2x1 MUX

Multiplexor (MUX) selects from one of many inputs

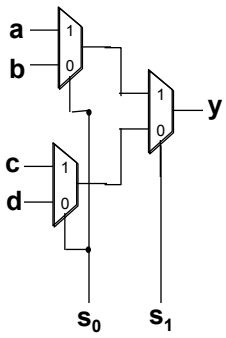


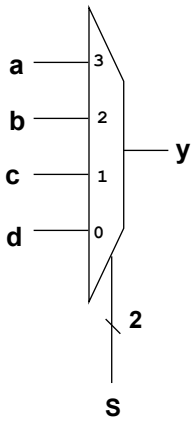

**MUX(A, B, S) = (A * S) + (B * ~S)**






# Example 4x1 MUX



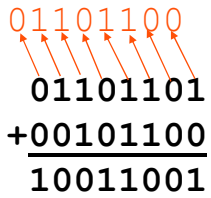



## Arithmetic and Logical Operations in ISA

- What operations are there?
- How do we implement them?
  - Consider a 1-bit Adder



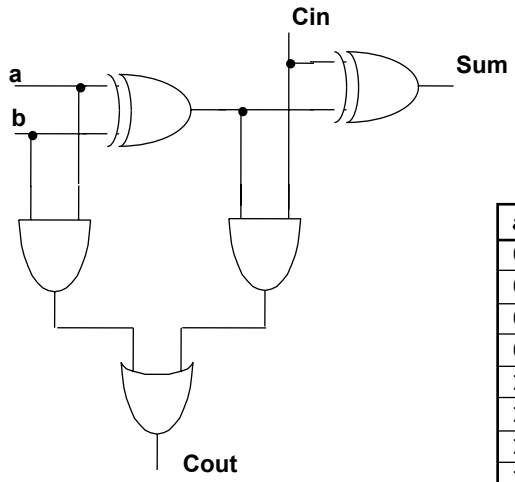

## Truth Table for 1-bit Addition




| a | b | $C_{in}$ | Sum | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

What is the circuit for Sum and for Cout?

# A 1-bit Full Adder

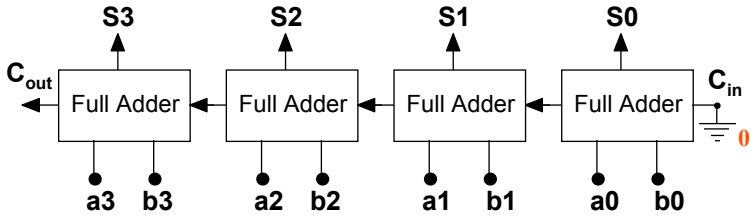



```
  01101100
  01101101
+ 00101100
  10011001
```

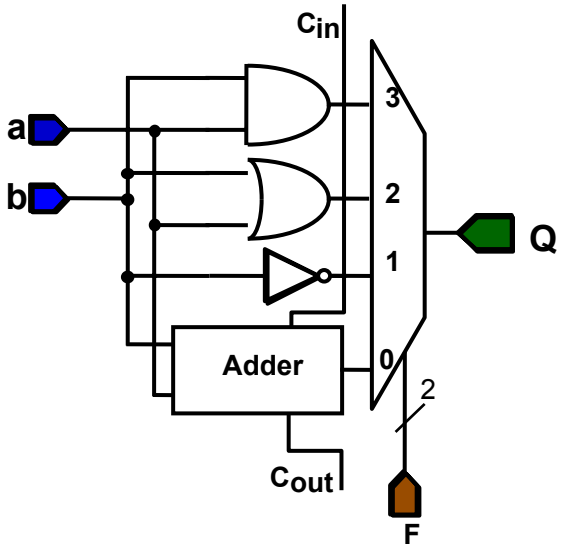
| a | b | $C_{in}$ | Sum | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |




# Example: 4-bit adder



## ALU Slice (Almost)



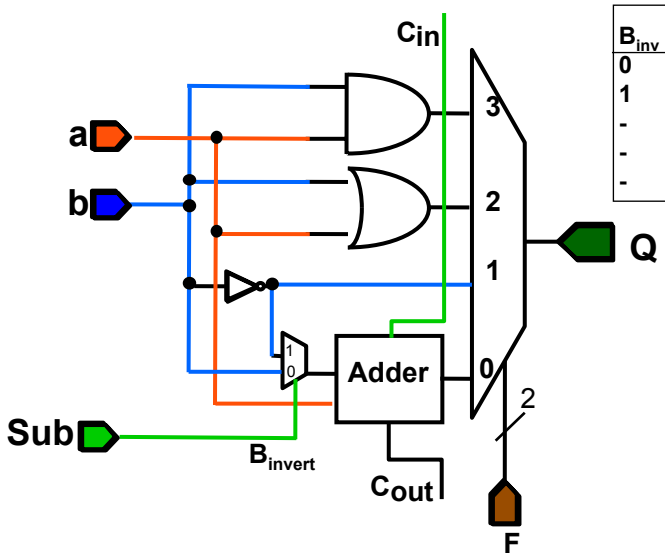

| F | Q |
|---|---|
| 0 | a + b |
| 1 | NOT b |
| 2 | a OR b |
| 3 | a AND b |



## Subtraction

- How do we perform integer subtraction?
- What is the HW?

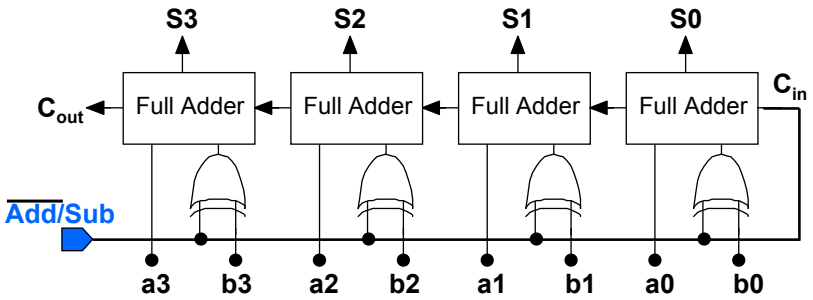
# ALU Slice




| $B_{inv}$ | F | Q |
|---|---|---|
| 0 | 0 | a + b |
| 1 | 0 | a - b |
| - | 1 | NOT b |
| - | 2 | a OR b |
| - | 3 | a AND b |



# Example: Adder/Subtracter




$\overline{Add}/Sub = 0$ => Addition
$\overline{Add}/Sub = 1$ => Subtraction

# Overflow

**Example1:**

01000000

$0110101_2$ $(= 53_{10})$
$+0101010_2$ $(= 42_{10})$
$1011111_2$ $(=-33_{10})$

**Example2:**

1000000

$1010101_2$ $(=-43_{10})$
$+1001010_2$ $(=-54_{10})$
$0011111_2$ $(= 31_{10})$

**Example3:**

1100000

$0110101_2$ $(= 53_{10})$
$+1101010_2$ $(=-22_{10})$
$0011111_2$ $(= 31_{10})$

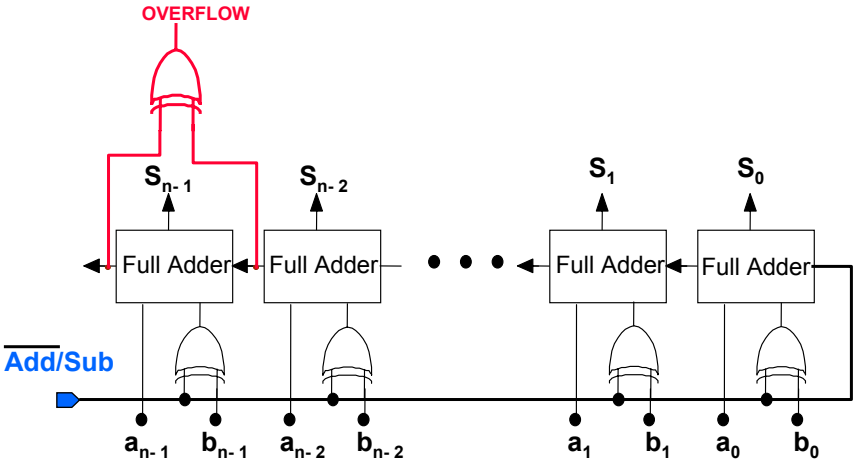
**Example4:**

0000000

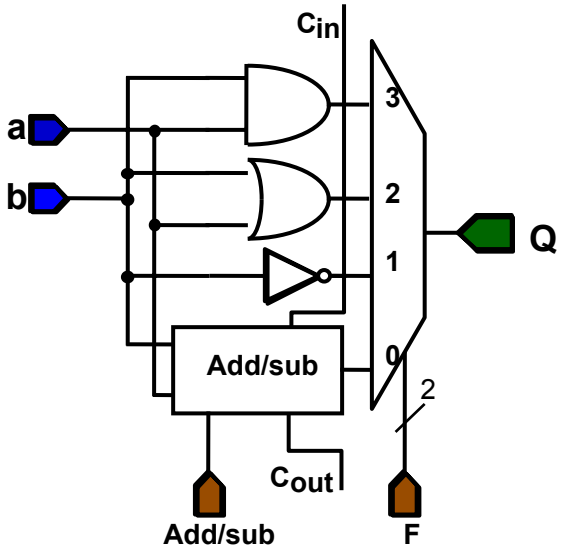
$0010101_2$ $(= 21_{10})$
$+0101010_2$ $(= 42_{10})$
$0111111_2$ $(= 63_{10})$




# Add/Subtract With Overflow detection



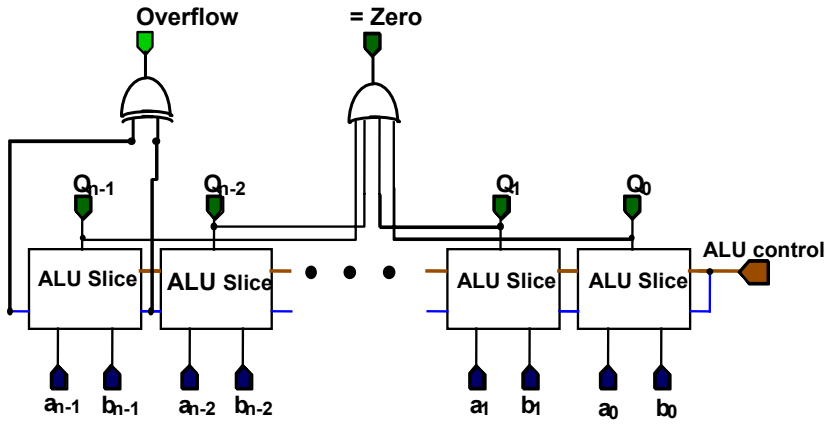
# The new ALU Slice




| A | F | Q |
|---|---|---|
| 0 | 0 | a + b |
| 1 | 0 | a - b |
| - | 1 | NOT b |
| - | 2 | a OR b |
| - | 3 | a AND b |

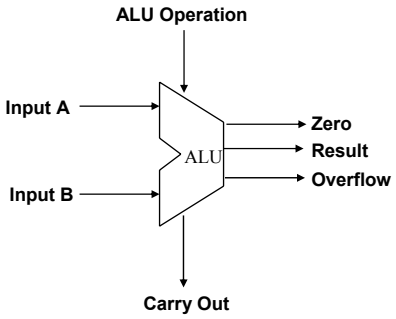


# The ALU



## Abstraction: The ALU

- General structure
- Two operand inputs
- Control inputs

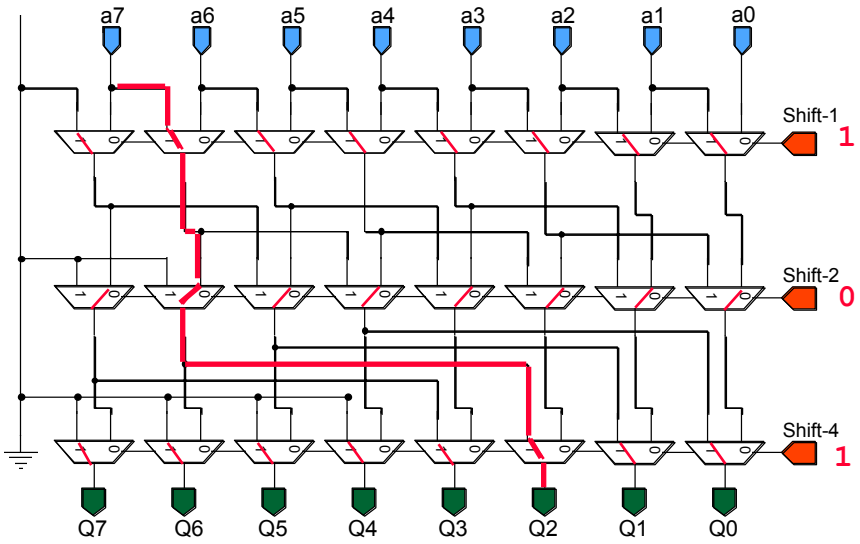





## The Shift Operation

- Consider an 8-bit machine
- How do I implement the shift operation?

## Shifter






## Summary thus far

- Given Boolean function, generate a circuit that "realizes" the function.
- Constructed circuits that can add and subtract.
- The ALU: a circuit that can add, subtract, detect overflow, compare, and do bit-wise operations (AND, OR, NOT)
- Shifter

Next up: Storage Elements: Registers, Latches, Buses