CS 2461
Lab- Week 2

1

Today….

- Quick review of data representation and operations on bits
- Review Transistor circuits (gates)

2

2

Binary Representation Summary

- Every storage locations stores a finite sequence of bits
  - 8-bit, 16-bit, 32-bit etc.
- The same bit string can mean different things depending on how the program wants to look at it….based on data representation used

Address	7	6	5	4	3	2	1	0
35	1	0	0	0	0	0	0	1
36	1	0	0	0	0	1	1	1
37	1	1	1	0	0	0	0	1
38	0	1	1	0	1	1	0	1

Unsigned: +129

2C: -127

2C: 109

ASCII: 'm'

If string at address 38 is defined as type `int` in C then value = 109
If defined as `char` then value=m

3

3

Arithmetic and Logic Operations

- Arithmetic:
  - Addition
  - Subtract (negative number and add to second number)
  - Shift – left shift by one position is multiplying by 2; right shift is division by 2
    - Shift left twice = multiply by $2^2 = 4$
  - Multiplication….
- Logic operators
  - AND, OR, NOT,…… Define using truth tables
  - Bitwise operations – apply logical operator at each bit position

4

4

Shifting Bit Fields

	7	6	5	4	3	2	1	0
Original Pattern x	0	1	1	0	1	0	1	1
X << 1 – Left Shift by 1	1	1	0	1	0	1	1	0
X << 2 – Left Shift by 2	1	0	1	0	1	1	0	0
Original Pattern x	1	1	1	0	1	0	1	1
X >> 1 –Shift Right (logical) by 1	0	1	1	1	0	1	0	1
X >>> 1 – Shift Right (arithmetic) by 1	1	1	1	1	0	1	0	1

- Shift Left:
  - Move all #'s to the left, fill in empty spots with a 0
- Shift Right (2 kinds):
  - shift right logical (SRL) >>
    - shift 0's in from the left
  - shift right arithmetic (SRA) >>>
    - replicate the sign bit, (very useful for sign extension!)

5

5

Dose of reality: Finite Width and Overflow

- Integers have infinite width
  - There are an infinite number of them
- Hardware integers have finite (architecture defined) width
  - Limited by hardware circuits themselves
  - 64- bit these days (2^{64} integers):
  - LC3 integers are 16-bit (2^{16} or ~64,000)
- Overflow: when operation result is outside type's range
  - Example: 15 + 1 with 4-bit integers (16 needs 5 bits, 10000)

```
                      1111  (15)
overflow          +   0001  (1)
(carry-out)  ->      10000  (16)
```

Problem: using 4-bit representation the sum is 0!!

6

6

Arithmetic Overflow - Summary

- For unsigned numbers
  - Any addition that produces an 'extra bit' is a problem
- For 2C signed numbers
  - Sometimes addition or subtraction produce an extra bit – this is not necessarily a problem.
  - *Overflow if Signs of both operands are the same AND the sign of the sum is different*
  - Arithmetic overflow can occur when you are adding 2 positive or 2 negative numbers – in this case if the sign of the result is different from the sign of the addends you have an arithmetic overflow
    - (this is the key to determining overflow condition in 2C)
  - CPU architectures today, use 2C representation

7

7

Sign Extension

- Suppose we have a number which is stored in a four bit register and we want to add this number to a number stored in a eight bit register
- We have a device (an 8-bit adder) which will do the addition and it is designed to add two 8 bit numbers
  - SW Analogy: Calling a function with the correct (type, number) arguments
- Therefore extend 4-bit number to 8-bit.... How ?

- Suppose we just pad 0's to the left:
  - 4 bit 0100 (decimal 4) becomes 0000 0100 which is decimal 4
  - 4 bit 1100 (decimal -4) becomes 0000 1100 which is decimal 12…wrong!

- Solution: replicate Most Significant bit (pad MSB to the left)
  - 4 bit 0100 (decimal 4) becomes 0000 0100 which is still decimal 4
  - 4 bit 1100 (decimal -4) becomes 1111 1100 which is still decimal -4

8

8

Bitwise Logical Operations

- View n-bit field as a collection of n logical values
  - *Apply operation to each bit independently*
- Bitwise AND: useful for clearing bits
  - AND with zero = 0
  - AND with one = no change
- Bitwise OR: useful for setting bits
  - OR with zero = no change
  - OR with one = 1
- Computers don't support individual bits as a data type
  - Just use least significant bit of n-bit integer
  - Integers are generally more useful

```
    11000101
AND 00001111
    00000101
```

```
   11000101
OR 00001111
   11001111
```

9

9

Bitwise Operations – Group Exercises…..

- Let A,B,C,D,F be *any* 8 bit 2's complement numbers
  - i.e., think of A as `type int` in a C program

1. What is C = `A AND 00000001`
   - How many different values can C have ? What are they and when do they occur ?
   - What property of A is determined by this "statement" ?
2. What is D `= (A >> 7) AND 00000001` *(right shift operator)*
   - How many different values can D have? When do they occur?
   - What property of A is determined by this "statement" ?
3. What is F = (A OR B) AND 00000001
   - How many different value can F have? When does each value occur ?
   - What property (of A,B) is determined by this "statement"

10

10

Review….Transistor Circuits and Logic gates

- Transistor acts as a voltage controlled switch
  - Send 0 or 1 to transistor Gate: switch closes or opens

- Two types of Transistors used in our circuits:
- p-type = switch closes if gate input =0, open if input =1
- n-type = switch closes if gate input =1, open if input =0

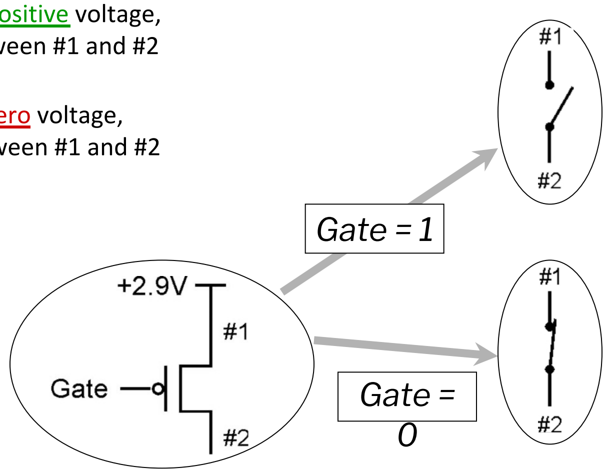
- Circuit Output = voltage measured at some location in the circuit
  - 1 if there is a positive voltage, and 0 if no voltage

11

11

Abstraction: Simplified view of p-type MOS Transistor

- p-type
  - when Gate has positive voltage, open circuit between #1 and #2 (switch open)
  - when Gate has zero voltage, short circuit between #1 and #2 (switch closed)



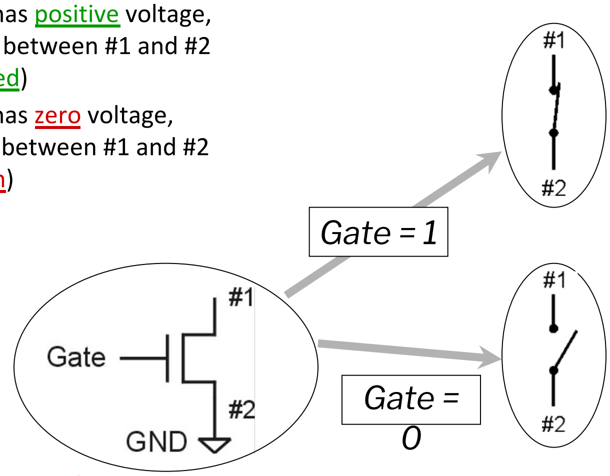
Important: For p-type, Terminal #1 must be connected to Voltage Source.

12

12

Abstraction: Simplified view of n-type MOS Transistor

- n-type *complementary* to p-type
  - when Gate has positive voltage, short circuit between #1 and #2 (switch closed)
  - when Gate has zero voltage, open circuit between #1 and #2 (switch open)

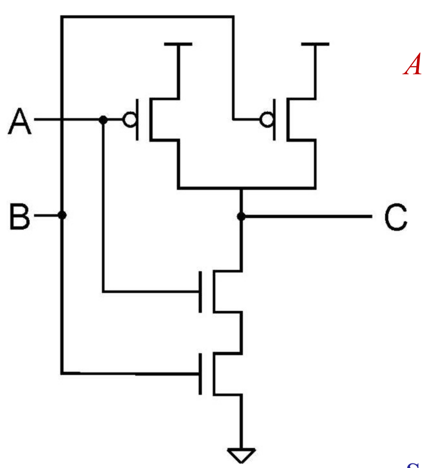


Important: For n-type, Terminal #2 must be connected to Ground (0V).

13

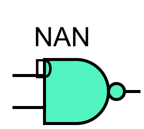
13

NAND Gate – NOT (AND): C = NOT (A AND B)



So how to build an AND gate ?
A AND B = NOT (NOT (A AND B))

Use NAND and NOT
Send output of NAND to
Input of NOT gate



Symbol for NAND gate

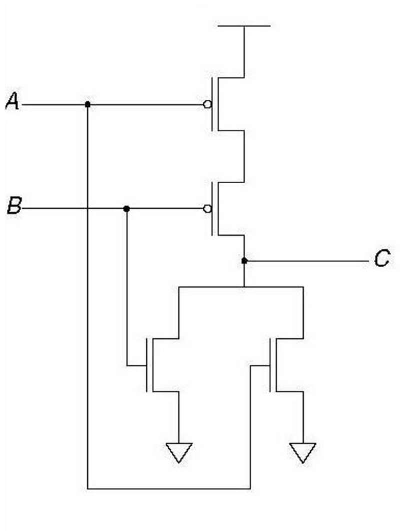
Truth Table

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Note: Parallel structure on top, serial on bottom.

14

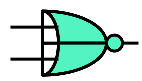
Solution to Question – Group Exercise



1. Derive truth table for this circuit,
Inputs= A,B
Output =C
2. What function is implemented ?

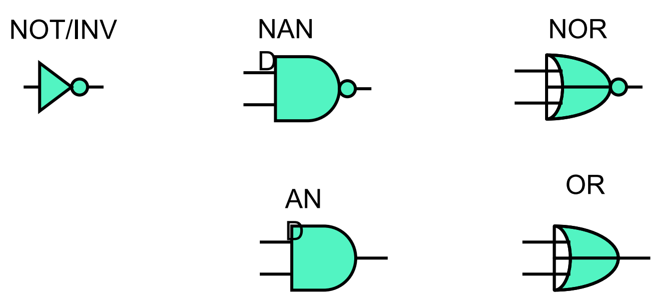
NOR gate = NOT (A OR B)

NOR



15

Basic Logic Gates - Symbols

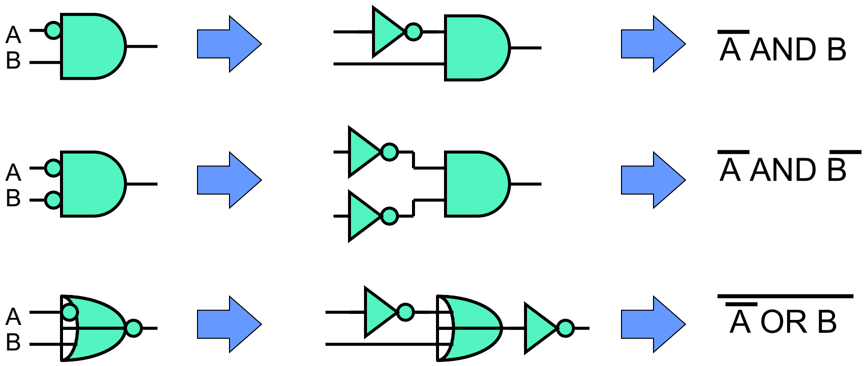


16

16

Shorthand for Inverting Signals

- Invert a signal by adding either
  - a ○ before/after a gate
  - a "bar" over letter

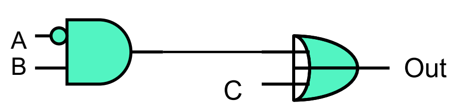


17

17

Example: Your first combinational circuit

- Combinational logic circuits ~ propositional logic statements
- Use gates to implement the logic operators ('functions')
  - *No necessity to show the circuit using transistors since each gate corresponds to an implementation using transistors*
- Output = (<u>(NOT A) AND B)</u> OR C
- Need one AND gate and one OR gate (and one NOT gate/invertor)

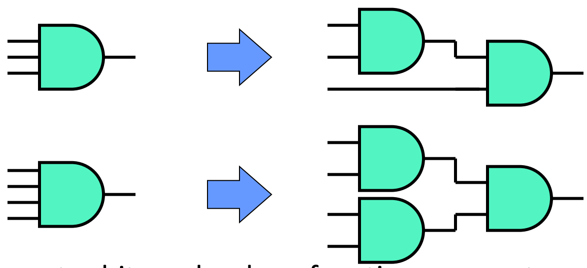


18

18

More than 2 Inputs? Arbitrary Functions?

- AND/OR can take any number of inputs
  - AND = 1 if all inputs are 1
  - OR = 1 if any input is 1 (0 if all inputs are 0)
- Implementation
  - Multiple two-input gates or single CMOS circuit

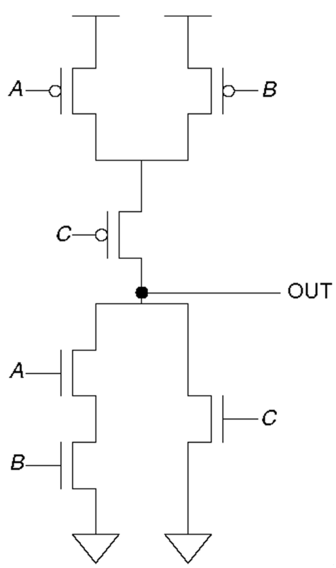


- Can implement arbitrary boolean functions as a gate
  - More complex n- and p- networks

19

19

Exercise….Truth table for circuit



A	B	C	OUT
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

20

20