CMSC 313 Lecture 16

- Announcement: no office hours today.

- Good-bye Assembly Language Programming
- Overview of second half on Digital Logic
- DigSim Demo

Good-bye Assembly Language

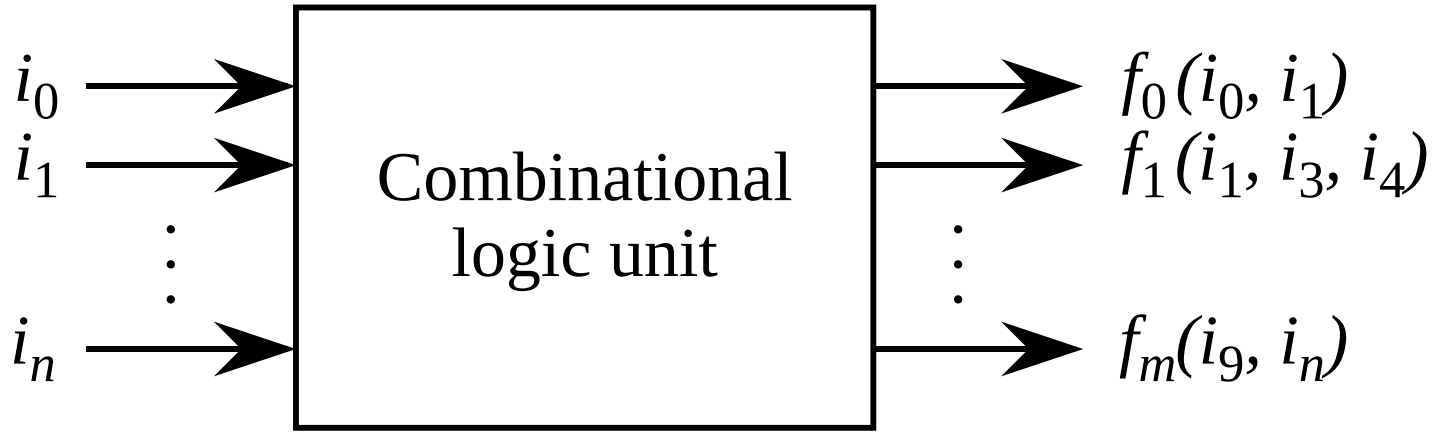
- What a pain!
- Understand pointers better
- Execution environment of Unix processes
  - the stack
  - virtual memory
- Linking & loading

Some Definitions

- ***Combinational logic:*** **a digital logic circuit in which logical decisions are made based only on combinations of the inputs. *e.g.* an adder.**
- ***Sequential logic:*** **a circuit in which decisions are made based on combinations of the current inputs as well as the past history of inputs. *e.g.* a memory unit.**
- ***Finite state machine:*** **a circuit which has an internal state, and whose outputs are functions of both current inputs and its internal state. *e.g.* a vending machine controller.**

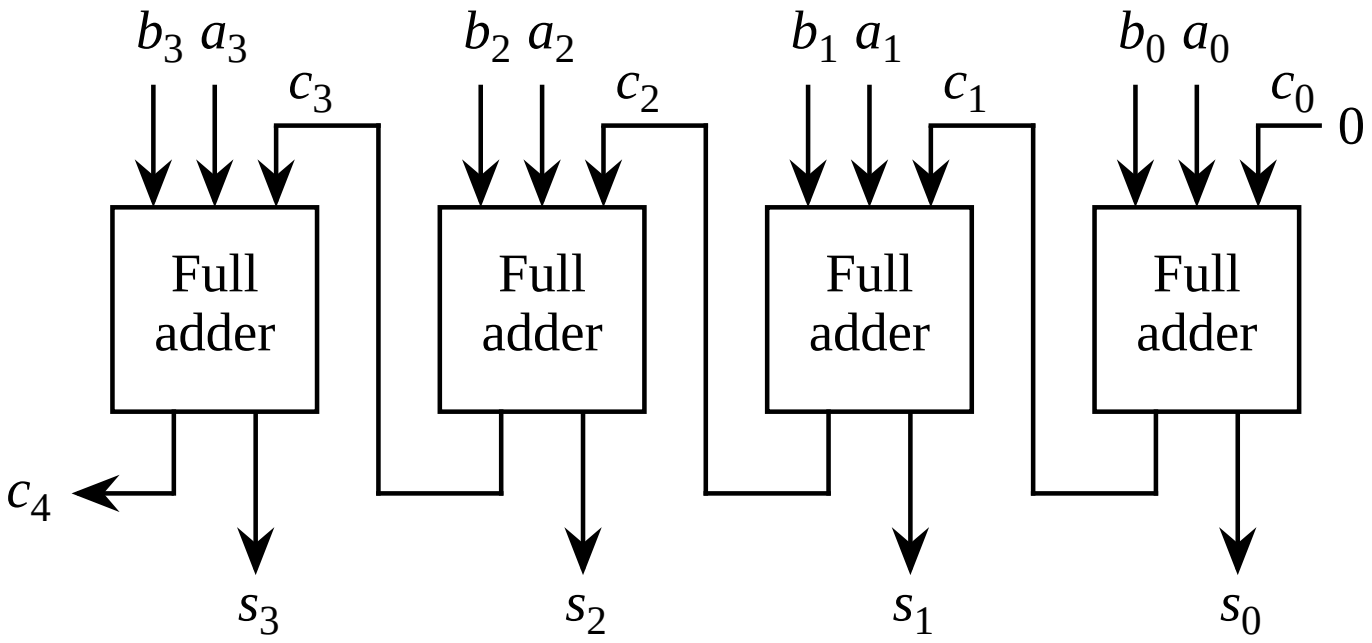
The Combinational Logic Unit

- **Translates a set of inputs into a set of outputs according to one or more mapping functions.**
- **Inputs and outputs for a CLU normally have two distinct (binary) values: high and low, 1 and 0, 0 and 1, or 5 V and 0 V for example.**
- **The outputs of a CLU are strictly functions of the inputs, and the outputs are updated immediately after the inputs change. A set of inputs $i_0 – i_n$ are presented to the CLU, which produces a set of outputs according to mapping functions $f_0 – f_m$.**



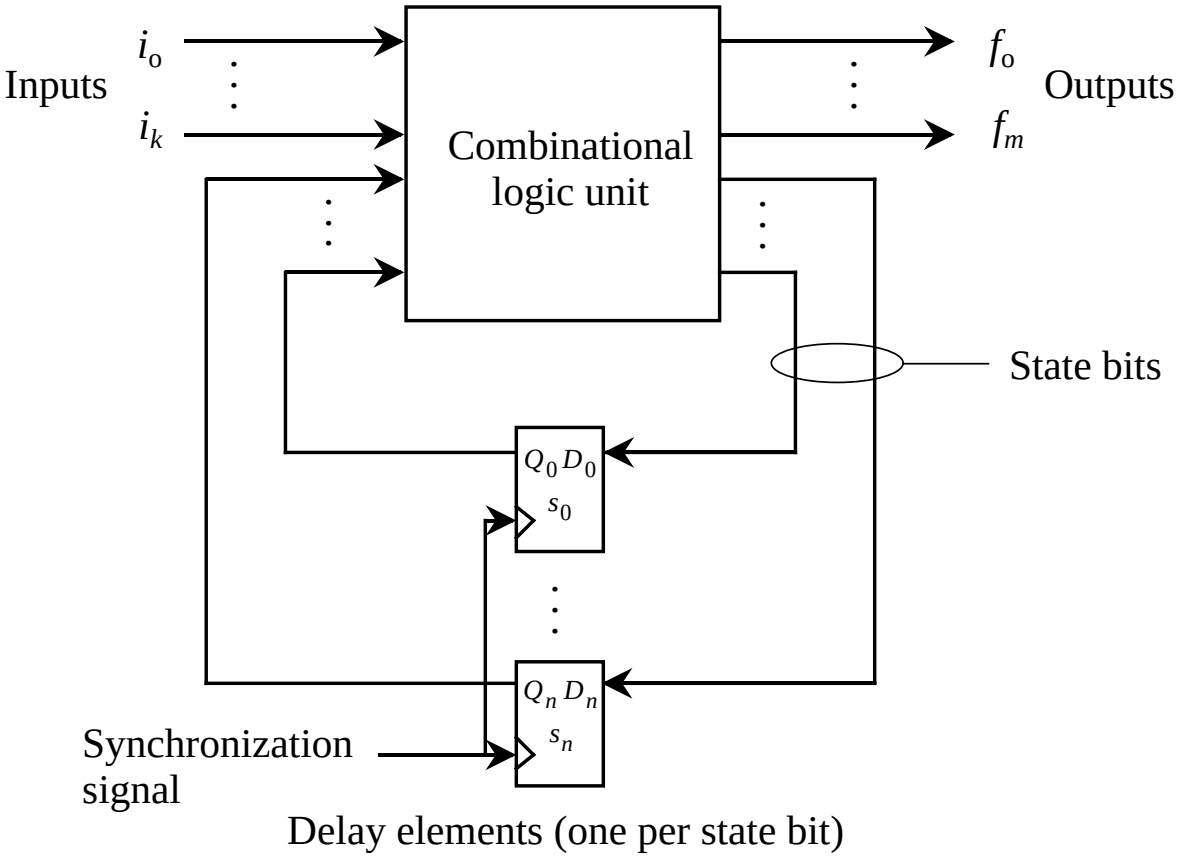
Ripple Carry Adder

- **Two binary numbers *A* and *B* are added from right to left, creating a sum and a carry at the outputs of each full adder for each bit position.**

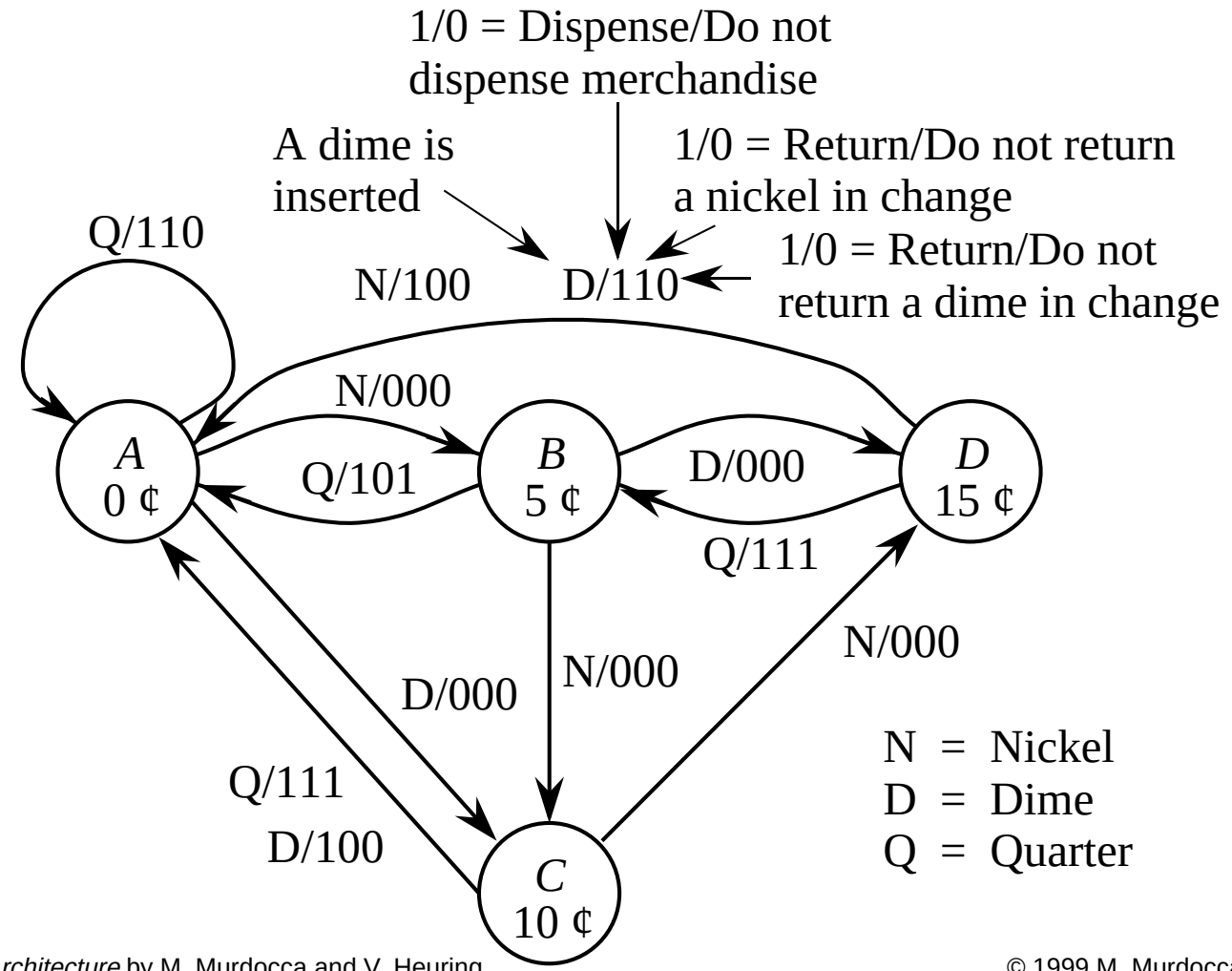


Classical Model of a Finite State Machine

- **An FSM is composed of a combinational logic unit and delay elements (called *flip-flops*) in a feedback path, which maintains state information.**



Vending Machine State Transition Diagram



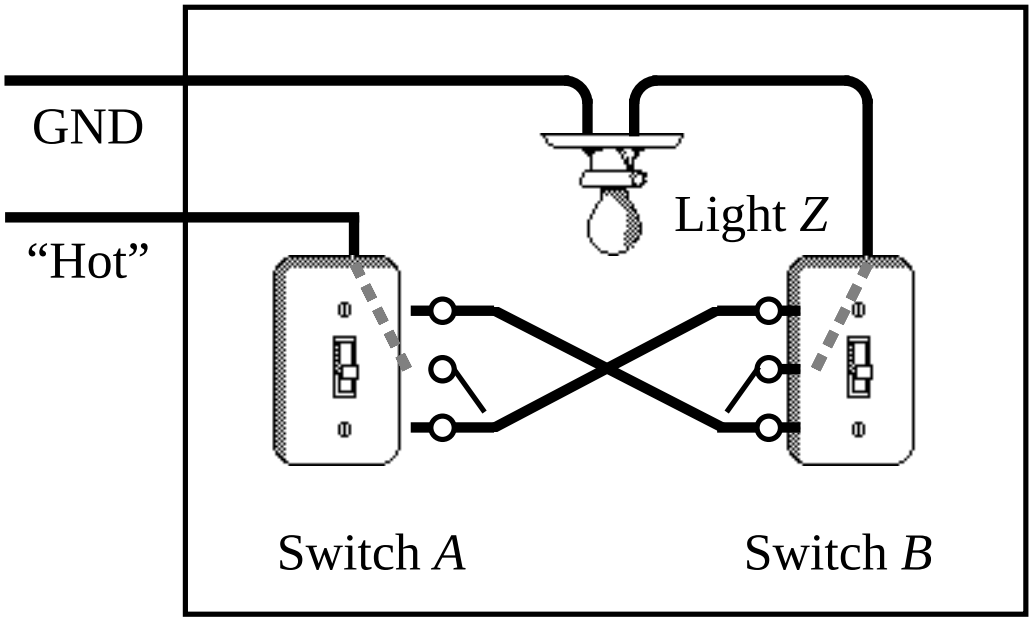
Course Syllabus

We will follow two textbooks: *Principles of Computer Architecture,* by Murdocca and Heuring, and *Linux Assembly Language Programming,* by Neveln. The following schedule outlines the material to be covered during the semester and specifies the corresponding sections in each textbook.

Date	Topic	M&H	Neveln	Assign	Due
Th 09/02	Introduction & Overview	1.1-1.8	1.1-1.6		
Tu 09/07	Data Representation I	2.1-2.2, 3.1-3.3	2.4-2.7, 3.6-3.8	HW1	
Th 09/09	Data Representation II				
Tu 09/14	i386 Assembly Language I		3.10-3.13, 4.1-4.8	HW2	HW1
Th 09/16	i386 Assembly Language II		6.1-6.5	Proj1	
Tu 09/21	i386 Assembly Language III				HW2
Th 09/23	i386 Assembly Language IV			Proj2	Proj1
Tu 09/28	Examples				
Th 09/30	Machine Language		5.1-5.7	Proj3	Proj2
Tu 10/05	Compiling, Assembling & Linking	5.1-5.3			
Th 10/07	Subroutines		7.1-7.4		
Tu 10/12	The Stack & C Functions				
Th 10/14	Linux Memory Model	7.7	8.1-8.8	Proj4	Proj3
Tu 10/19	Interrupts & System Calls		9.1-9.8		
Th 10/21	Cache Memory	7.6			Proj4
Tu 10/26	**Midterm Exam**				
Th 10/28	Introduction to Digital Logic	A.1-A.3	3.1-3.3	DigSim1	
Tu 11/02	Transistors & Logic Gates	A.4-A.7			
Th 11/04	Circuits for Addition	3.5		HW3	DigSim1
Tu 11/09	Combinational Logic Components	A.10			
Th 11/11	Circuit Simplification I	B.1-B.2		HW4	HW3
Tu 11/16	Flip Flops I	A.11			
Th 11/18	Flip Flops II			DigSim2	HW4
Tu 11/23	Finite State Machines	A.12-A.13			
Th 11/25	*Thanksgiving break*				
Tu 11/30	Circuit Simplification II	B.3		HW5	DigSim2
Th 12/02	Finite State Machine Design				
Tu 12/07	Registers & Memory	A.14-15, 7.1-7.5		DigSim3	HW5
Th 12/09	I/O	8.1-8.3			
Tu 12/14	TBA				DigSim3
Tu 12/21	**Final Exam 10:30am-12:30pm**				

A Truth Table

- **Developed in 1854 by George Boole.**
- **Further developed by Claude Shannon (Bell Labs).**
- **Outputs are computed for all possible input combinations (how many input combinations are there?)**
- **Consider a room with two light switches. How must they work?**



Inputs		Output
A	*B*	*Z*
0	0	0
0	1	1
1	0	1
1	1	0

Alternate Assignment of Outputs to Switch Settings

- **We can make the assignment of output values to input combinations any way that we want to achieve the desired input-output behavior.**

Inputs		Output
A	*B*	*Z*
0	0	1
0	1	0
1	0	0
1	1	1

Truth Tables Showing All Possible Functions of Two Binary Variables

- **The more frequently used functions have names: AND, XOR, OR, NOR, XOR, and NAND. (Always use upper case spelling.)**

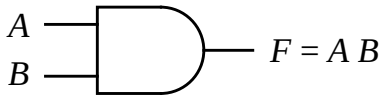
Inputs		Outputs							
A	B	*False*	*AND*	$A\overline{B}$	A	$\overline{A}B$	B	*XOR*	*OR*
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Inputs		Outputs							
A	B	*NOR*	*XNOR*	$\overline{B}$	$A + \overline{B}$	$\overline{A}$	$\overline{A} + B$	*NAND*	*True*
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Logic Gates and Their Symbols

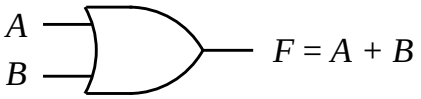
- **Logic symbols shown for AND, OR, buffer, and NOT Boolean functions.**
- **Note the use of the "inversion bubble."**
- **(Be careful about the "nose" of the gate when drawing AND vs. OR.)**

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



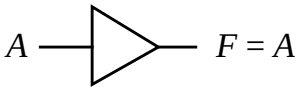
AND

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



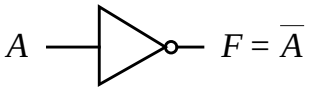
OR

A	F
0	0
1	1



Buffer

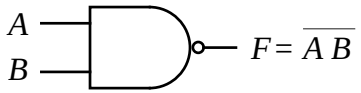
A	F
0	1
1	0



NOT (Inverter)

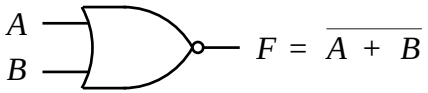
Logic Gates and their Symbols (cont')

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



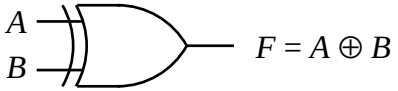
NAND

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



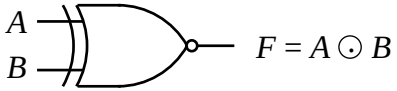
NOR

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



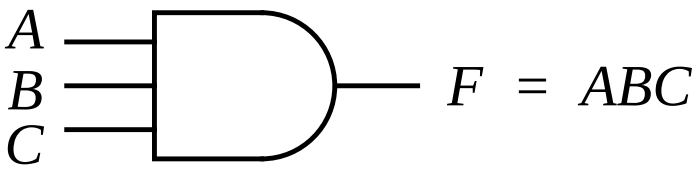
Exclusive-OR (XOR)

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

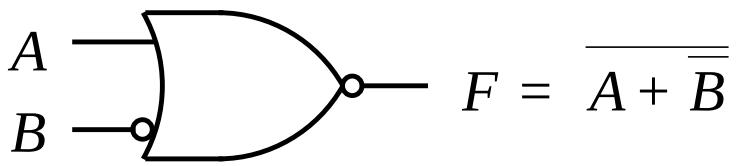


Exclusive-NOR (XNOR)

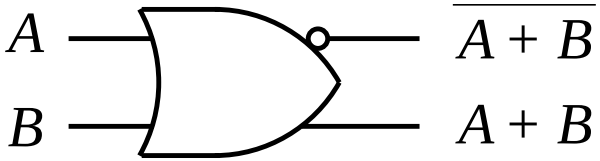
Variations of Logic Gate Symbols



(a)



(b)



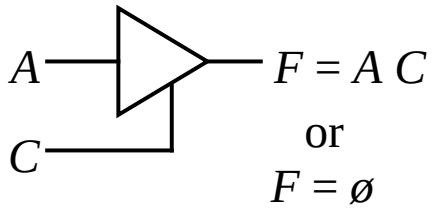
(c)

(a) 3 inputs **(b) A Negated input** **(c) Complementary outputs**

Tri-State Buffers

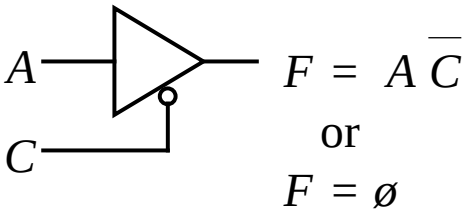
- **Outputs can be 0, 1, or “electrically disconnected.”**

C	A	F
0	0	ø
0	1	ø
1	0	0
1	1	1



Tri-state buffer

C	A	F
0	0	0
0	1	1
1	0	ø
1	1	ø



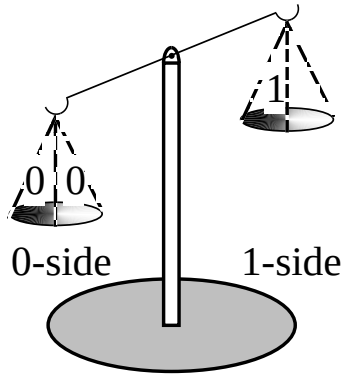
Tri-state buffer, inverted control

Sum-of-Products Form: The Majority Function

- **The SOP form for the 3-input majority function is:**

  $M = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC = m3 + m5 + m6 + m7 = \Sigma(3, 5, 6, 7)$.
- **Each of the 2^n terms are called *minterms*, ranging from 0 to $2^n - 1$.**
- **Note relationship between minterm number and boolean value.**

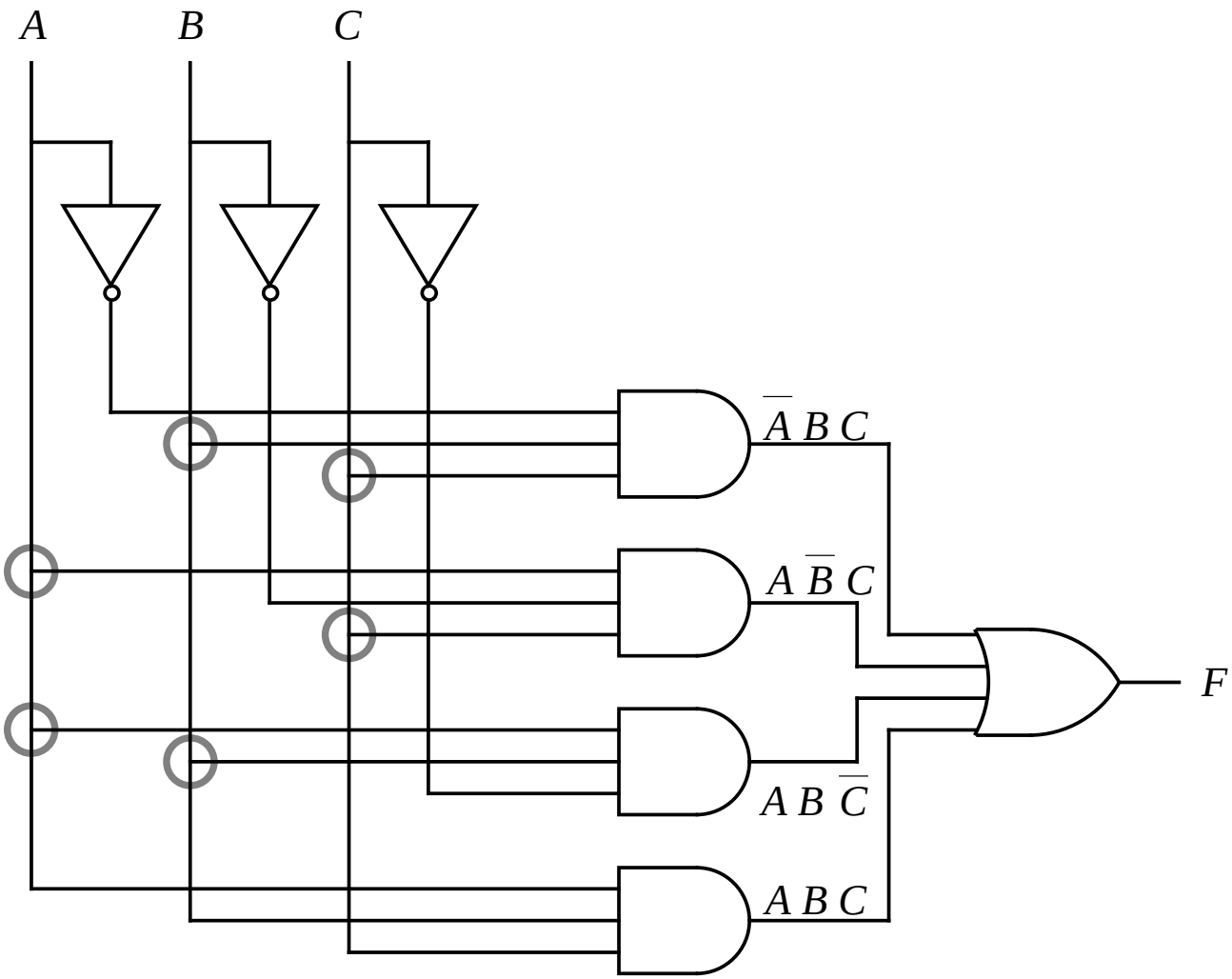
Minterm Index	*A*	*B*	*C*	*F*
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



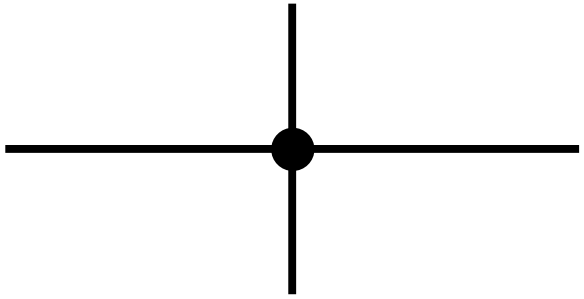
A balance tips to the left or right depending on whether there are more 0's or 1's.

AND-OR Implementation of Majority

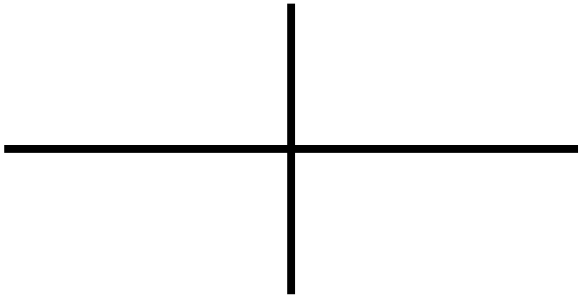
- **Gate count is 8, gate input count is 19.**



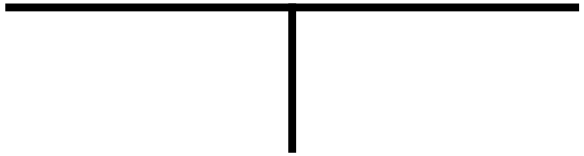
Notation Used at Circuit Intersections



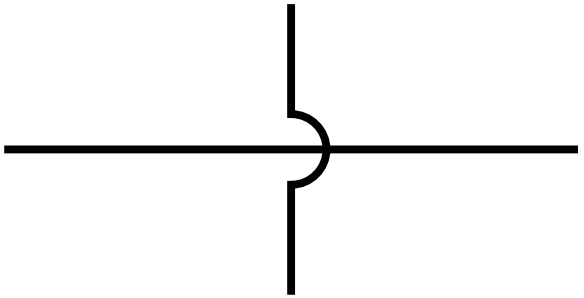
Connection



No connection



Connection



No connection

Sum of Products (a.k.a. disjunctive normal form)

- OR (i.e., sum) together rows with output 1
- AND (i.e., product) of variables represents each row
  e.g., in row 3 when $x_1 = 0$ AND $x_2 = 1$ AND $x_3 = 1$
  or when $\overline{x_1} \cdot x_2 \cdot x_3 = 1$
- MAJ3$(x_1, x_2, x_3) = \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1x_2x_3 = \sum m(3, 5, 6, 7)$

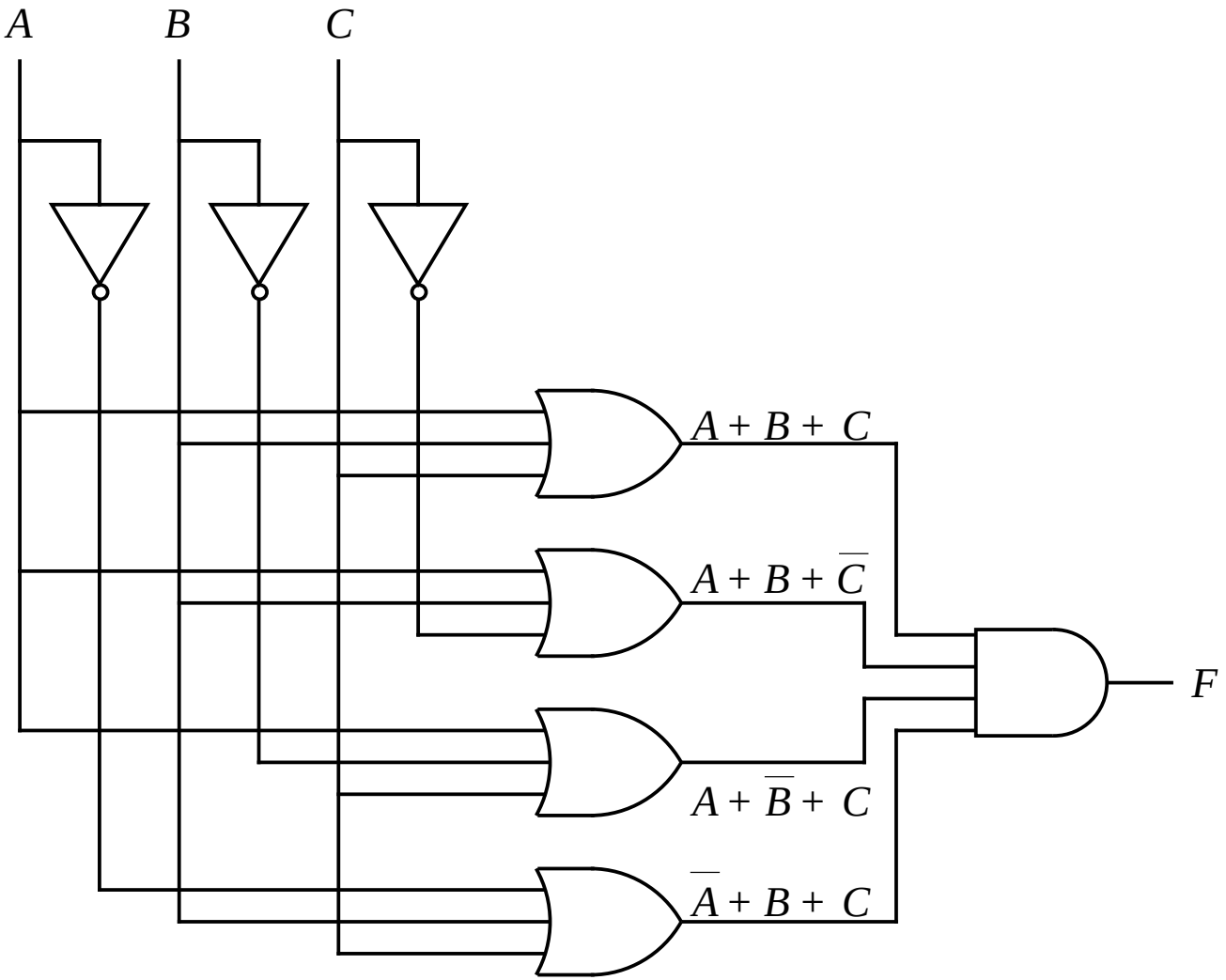
	x_1	x_2	x_3	MAJ3
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Product of Sums (a.k.a. conjunctive normal form)

- AND (i.e., product) of rows with output 0
- OR (i.e., sum) of variables represents negation of each row
  e.g., NOT in row 2 when $x_1 = 1$ OR $x_2 = 0$ OR $x_3 = 1$
  or when $x_1 + \overline{x_2} + x_3 = 1$
- MAJ3$(x_1, x_2, x_3) = (x_1+x_2+x_3)(x_1+x_2+\overline{x_3})(x_1+\overline{x_2}+x_3)(\overline{x_1}+x_2+x_3)$
  $= \prod M(0, 1, 2, 4)$

	x_1	x_2	x_3	MAJ3
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

OR-AND Implementation of Majority



Equivalences

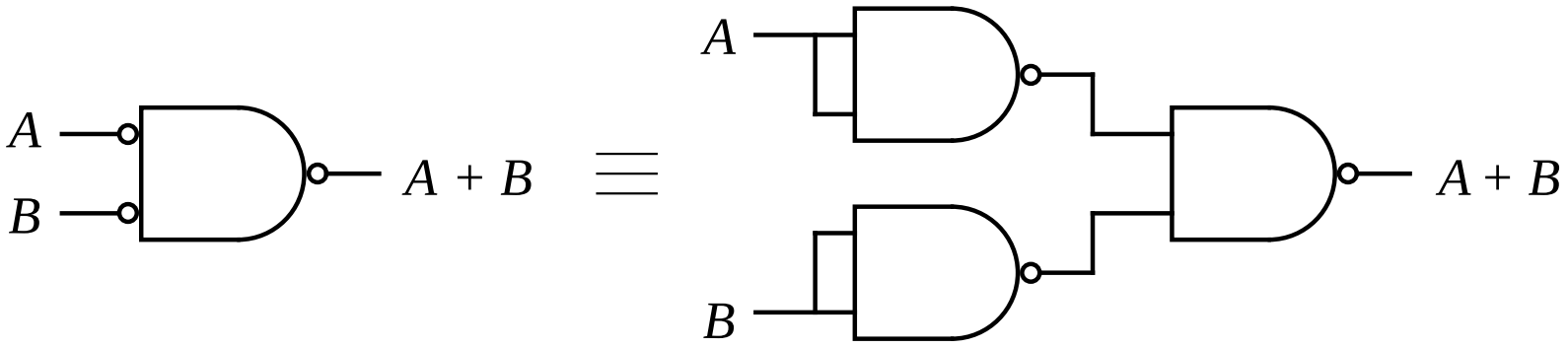
- Every Boolean function can be written as a truth table
- Every truth table can be written as a Boolean formula (SOP or POS)
- Every Boolean formula can be converted into a combinational circuit
- Every combinational circuit is a Boolean function
- Later you might learn other equivalencies:
  finite automata $\equiv$ regular expressions
  computable functions $\equiv$ programs

Universality

- Every Boolean function can be written as a Boolean formula using AND, OR and NOT operators.
- Every Boolean function can be implemented as a combinational circuit using AND, OR and NOT gates.
- Since AND, OR and NOT gates can be constructed from NAND gates, NAND gates are universal.

All-NAND Implementation of OR

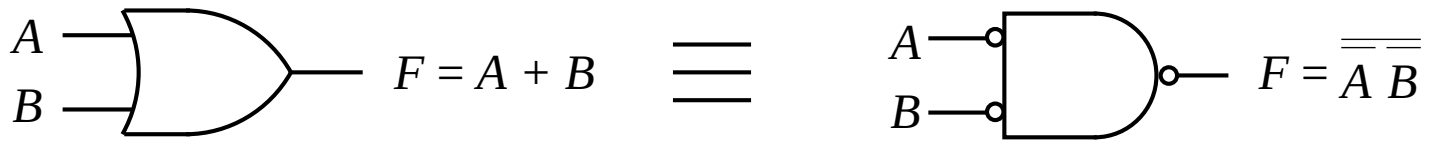
- **NAND alone implements all other Boolean logic gates.**



DeMorgan's Theorem

A B	$\overline{A\,B} = \overline{A} + \overline{B}$		$\overline{A + B} = \overline{A}\,\overline{B}$	
0 0	1	1	1	1
0 1	1	1	0	0
1 0	1	1	0	0
1 1	0	0	0	0

DeMorgan's theorem: $A + B = \overline{\overline{A} + \overline{B}} = \overline{\overline{A}\,\overline{B}}$



DigSim

- Java applet/application that simulates digital logic
- Not for industrial use, good enough for us
- Advantages: FREE, runs on most platforms
- Disadvantages: slow, timing issues, saving issues

DigSim Assignment 1: Getting Started

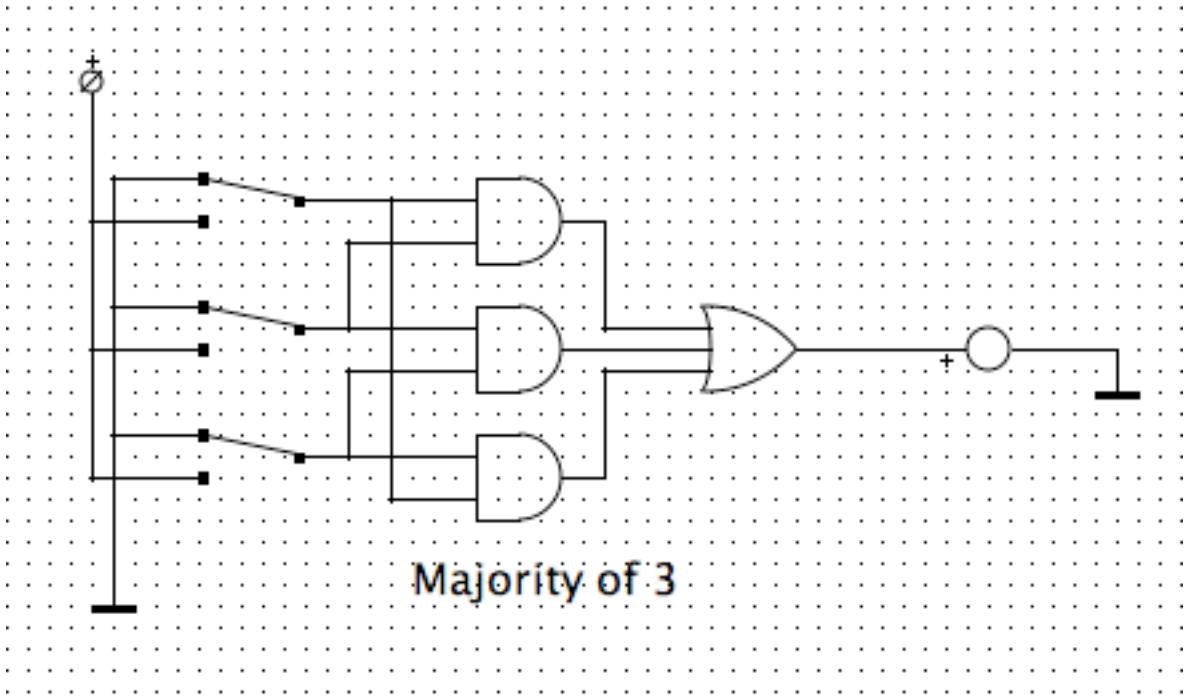
Due: Thursday November 4, 2004

Objective

The objective of this assignment is to make sure that everyone has access to DigSim and, most importantly, can save DigSim circuits for submission.

Assignment

Using DigSim, wire up the following circuit diagram, play with the switches, create a text box with your name, and save the circuit diagram.



Turning in your program

The file which has your circuit diagram should be a plain text file that starts with something like:

```
# Digsim file
version 1 0
describe component ThreeOrPort
 pos 31 15
```

Use a text editor to look at the file and make sure that the file is not empty and has some data similar to the above. Next, use DigSim to load the file and make sure that this still works. If all is well, submit the circuit file using the Unix submit command as in previous assignments. The submission name for this assignment is `:digsim1`.The UNIX command to do this should look something like:

```
submit cs313_0101 digsim1 majority3.sim
```

Next time

- Transistors & Gates