CMSC 313 Lecture 15

- 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

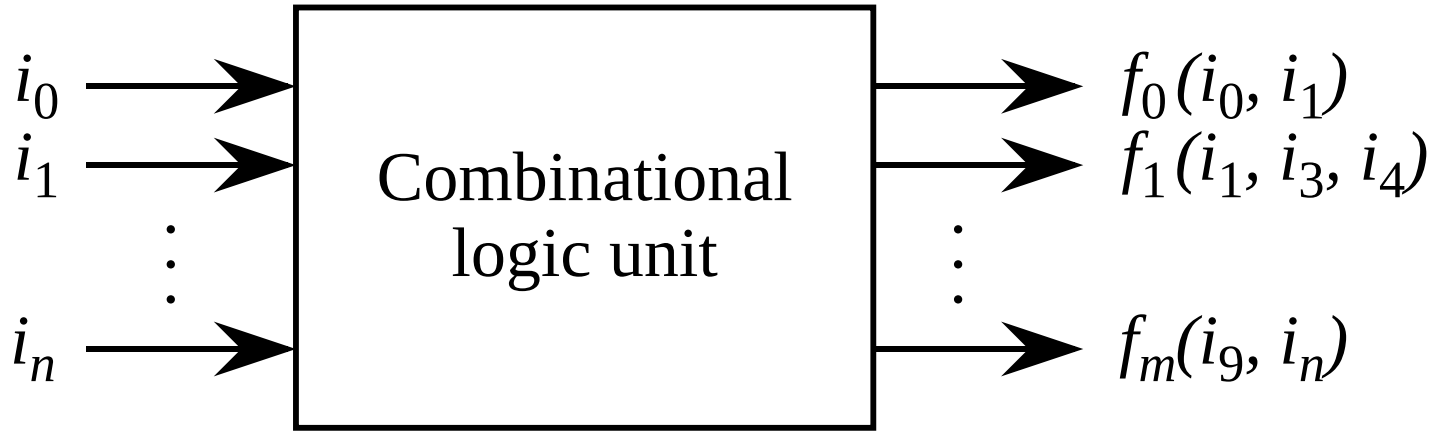
Some Definitions

- ***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.**

© 1999 M. Murdocca and V. Heuring

The Combinational Logic Unit

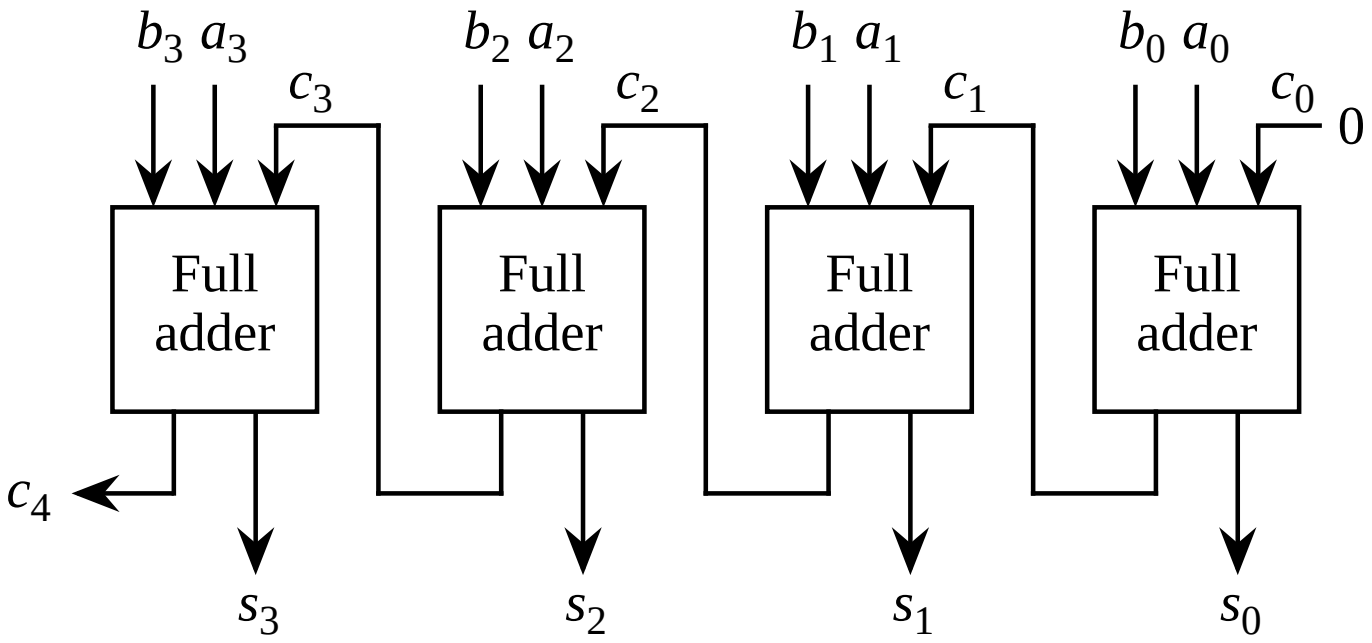
- **Translates a set of inputs into a set of outputs according to one or more mapping functions.**
- **Inputs and outputs for a CLU normally have two distinct (binary) values: high and low, 1 and 0, 0 and 1, or 5 V and 0 V for example.**
- **The outputs of a CLU are strictly functions of the inputs,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



© 1999 M. Murdocca and V. Heuring

Ripple Carry Adder

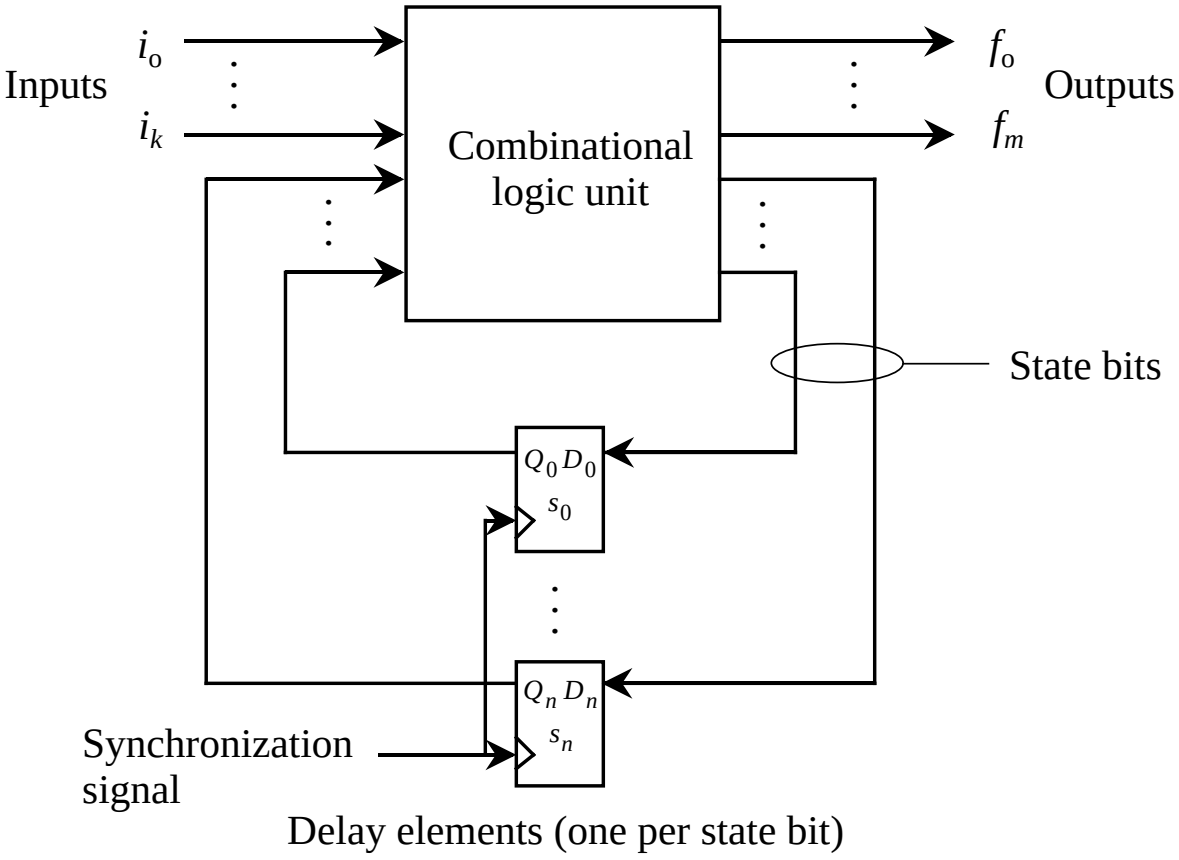
- **Two binary numbers *A* and *B* are added from right to left, creating a sum and a carry at the outputs of each full adder for each bit position.**



© 1999 M. Murdocca and V. Heuring

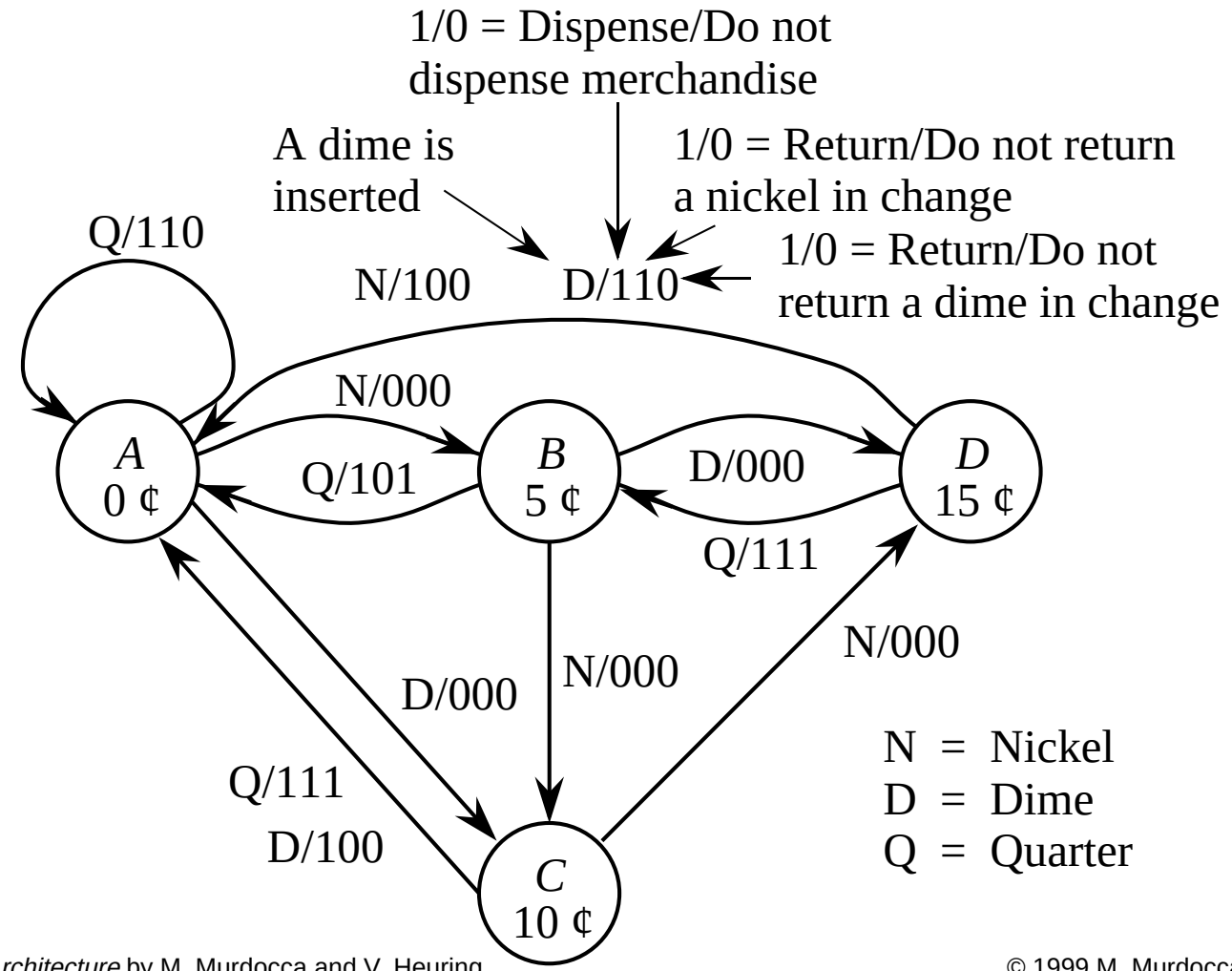
Classical Model of a Finite State Machine

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



© 1999 M. Murdocca and V. Heuring

Vending Machine State Transition Diagram



© 1999 M. Murdocca and V. Heuring

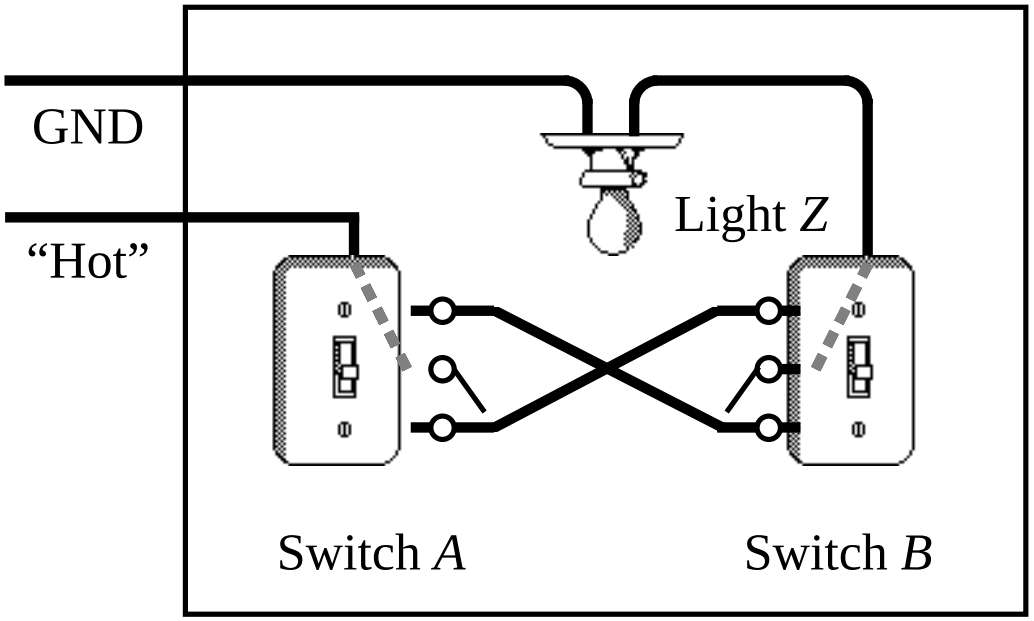
Course Syllabus

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 08/28	Introduction & Overview	1.1-1.8	1.1-1.6		
Tu 09/02	Data Representation I	2.1-2.2, 3.1-3.3	2.4-2.7, 3.6-3.8	hw1	
Th 09/04	Data Representation II				
Tu 09/09	i386 Assembly Language I		3.10-3.13, 4.1-4.8	hw2, proj1	hw1
Th 09/11	i386 Assembly Language II		6.1-6.5		
Tu 09/16	i386 Assembly Language III			proj2	hw2, proj1
Th 09/18	i386 Assembly Language IV				
Tu 09/23	Examples			proj3	proj2
Th 09/25	Machine Language		5.1-5.7		
Tu 09/30	Compiling, Assembling & Linking	5.1-5.3			
Th 10/02	Subroutines		7.1-7.4		
Tu 10/07	The Stack & C Functions			proj4	proj3
Th 10/09	Linux Memory Model	7.7	8.1-8.8		
Tu 10/14	Interrupts & System Calls		9.1-9.8		proj4
Th 10/16	**Midterm Exam**				
Tu 10/21	Introduction to Digital Logic	A.1-A.2	3.1-3.3		
Th 10/23	Transistors & Logic Gates	A.3-A.4		hw3	
Tu 10/28	In-class Lab I				
Th 10/30	Boolean Functions & Truth Tables	A.5-A.9		hw4	hw3
Tu 11/04	Circuits for Addition	3.5			
Th 11/06	Circuit Simplification I	B.1-B.2		hw5	hw4
Tu 11/11	Combinational Logic Components	A.10			
Th 11/13	Flip Flops	A.11		digsim1	hw5
Tu 11/18	In-class Lab II				
Th 11/20	Finite State Machines	A.12-A.15			
Tu 11/25	Circuit Simplification II	B.3			digsim1
Th 11/27	*Thanksgiving break*				
Tu 12/02	Finite State Machine Design			digsim2	
Th 12/04	More Finite State Machine Design				
Tu 12/09	I/O & Memory	7.1-7.6, 8.1-8.3			digsim2
Tu 12/16	**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

© 1999 M. Murdocca and V. Heuring

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

© 1999 M. Murdocca and V. Heuring

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

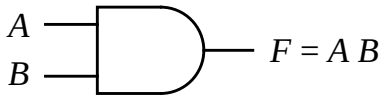
Inputs		Outputs							
A	B	*NOR*	*XNOR*	$\overline{B}$	$A + \overline{B}$	$\overline{A}$	$\overline{A} + B$	*NAND*	*True*
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

© 1999 M. Murdocca and V. Heuring

Logic Gates and Their Symbols

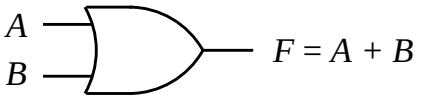
- **Logic symbols shown for AND, OR, buffer, and NOT Boolean functions.**
- **Note the use of the "inversion bubble."**
- **(Be careful about the "nose" of the gate when drawing AND vs. OR.)**

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



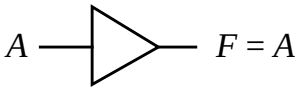
AND

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



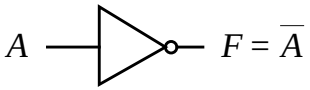
OR

A	F
0	0
1	1



Buffer

A	F
0	1
1	0

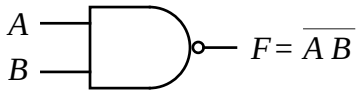


NOT (Inverter)

© 1999 M. Murdocca and V. Heuring

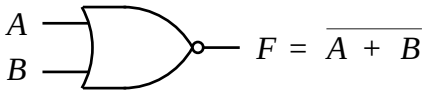
Logic Gates and their Symbols (cont')

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



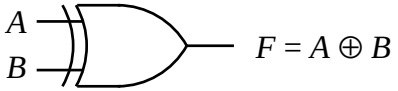
NAND

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



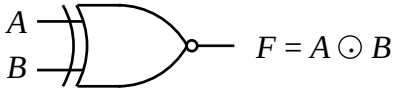
NOR

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



Exclusive-OR (XOR)

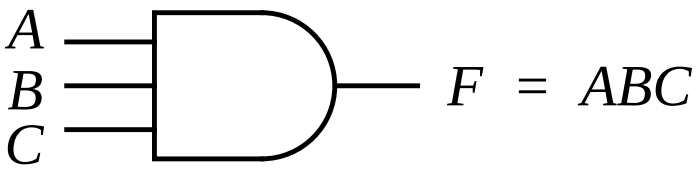
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1



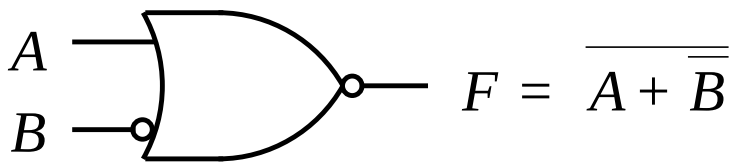
Exclusive-NOR (XNOR)

© 1999 M. Murdocca and V. Heuring

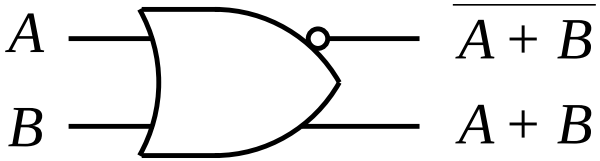
Variations of Logic Gate Symbols



(a)



(b)



(c)

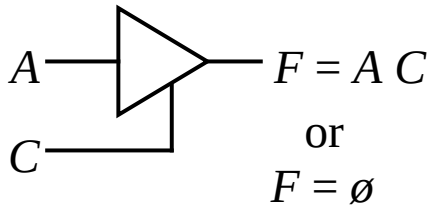
(a) 3 inputs **(b) A Negated input** **(c) Complementary outputs**

© 1999 M. Murdocca and V. Heuring

Tri-State Buffers

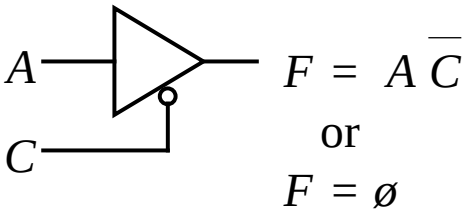
- **Outputs can be 0, 1, or “electrically disconnected.”**

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

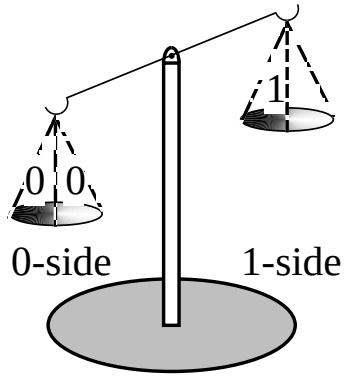
© 1999 M. Murdocca and V. Heuring

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

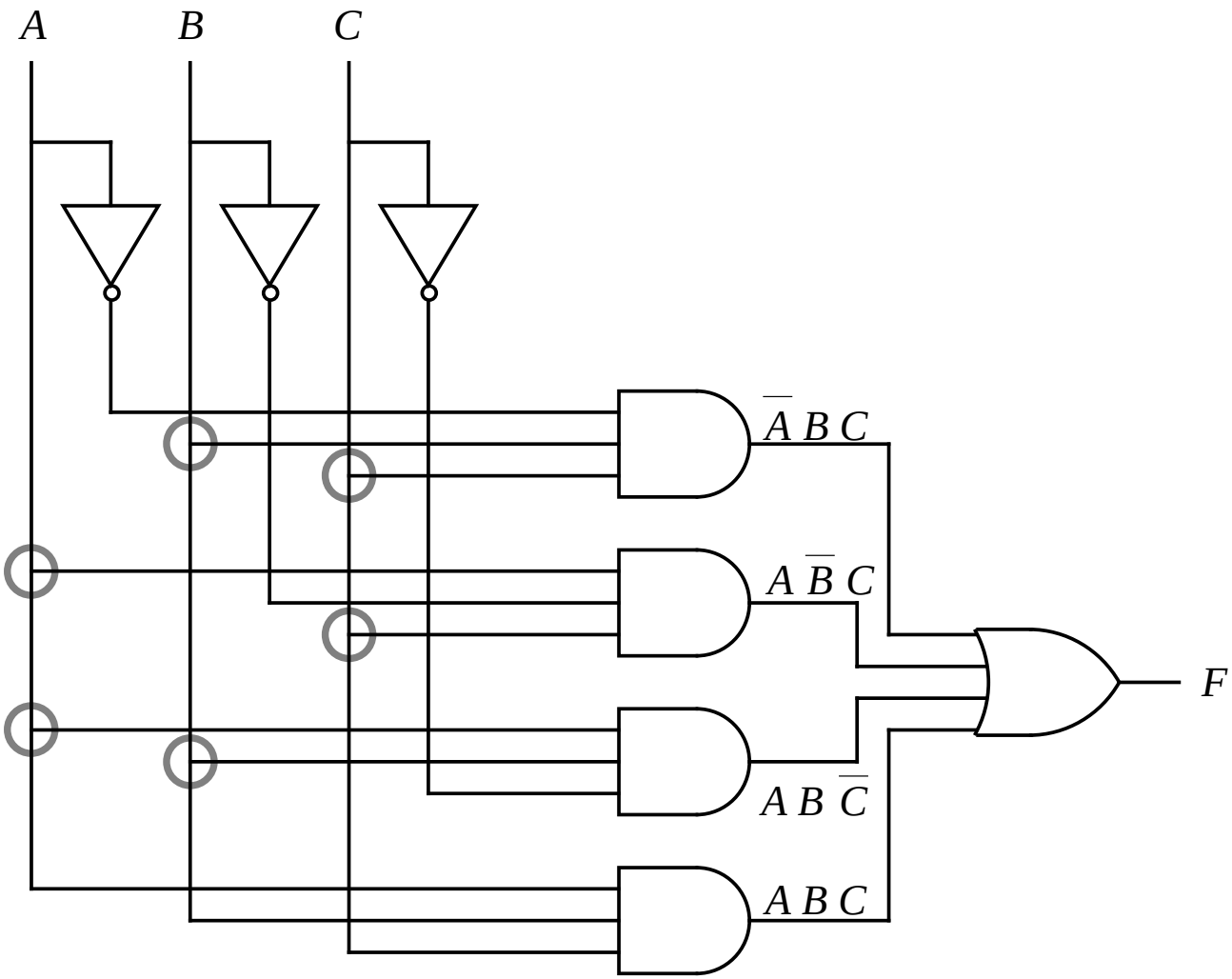


A balance tips to the left or right depending on whether there are more 0's or 1's.

© 1999 M. Murdocca and V. Heuring

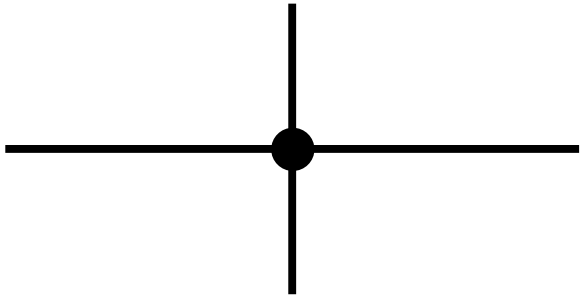
AND-OR Implementation of Majority

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

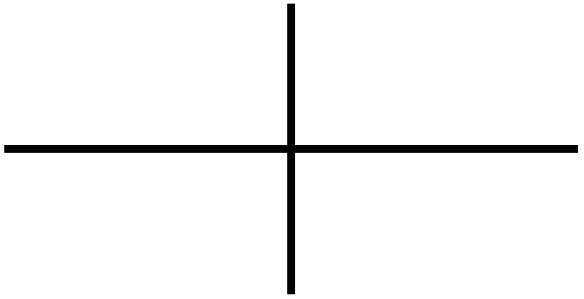


© 1999 M. Murdocca and V. Heuring

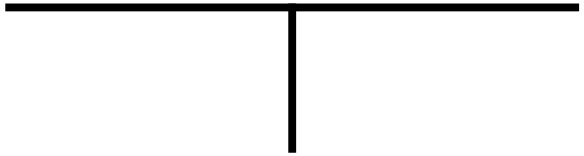
Notation Used at Circuit Intersections



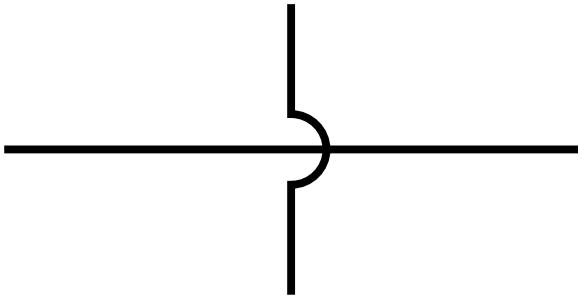
Connection



No connection



Connection



No connection

© 1999 M. Murdocca and V. Heuring

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$(x_1, x_2, x_3) = \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1x_2x_3 = \sum m(3, 5, 6, 7)$

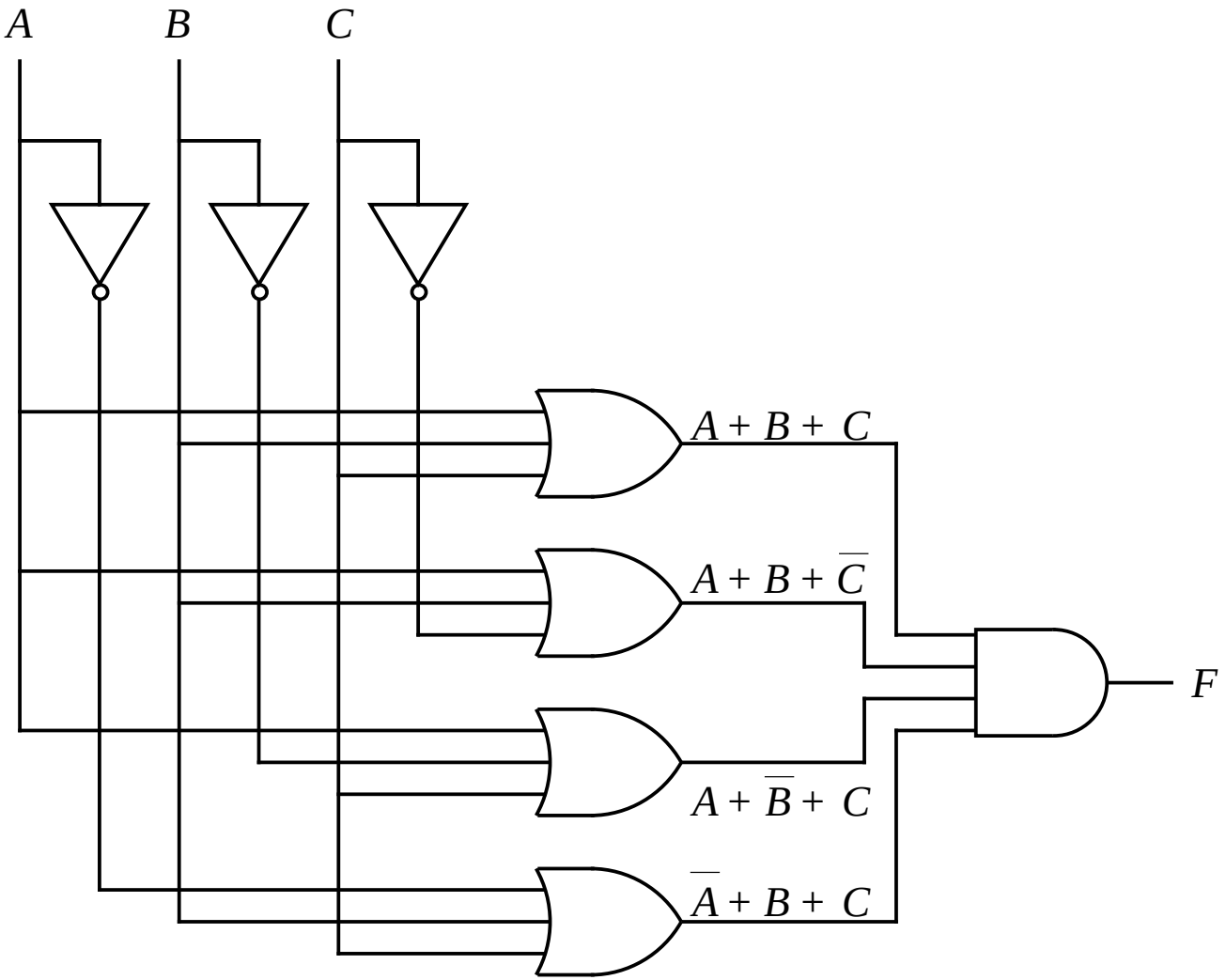
	x_1	x_2	x_3	MAJ3
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

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



© 1999 M. Murdocca and V. Heuring

Equivalences

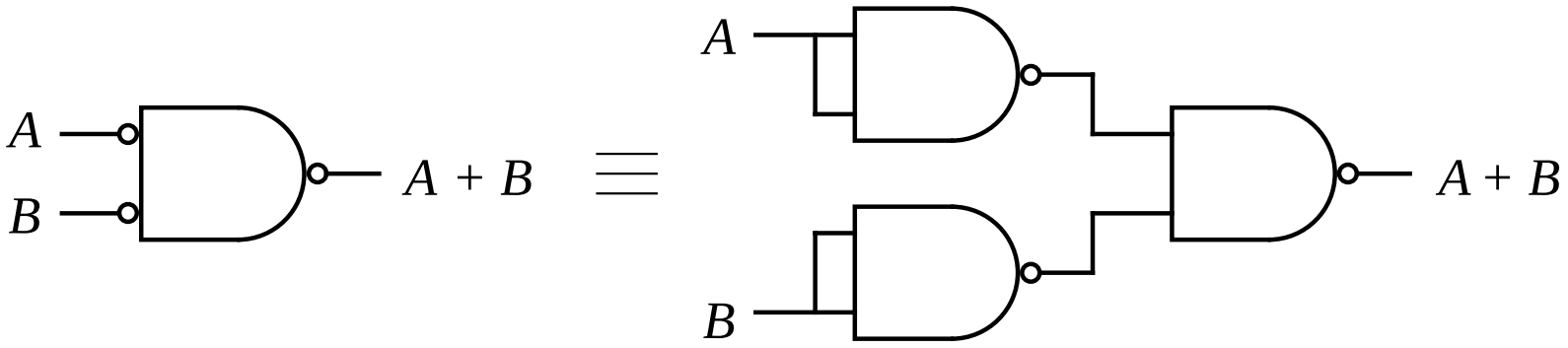
- Every Boolean function can be written as a truth table
- Every truth table can be written as a Boolean formula (SOP or POS)
- Every Boolean formula can be converted into a combinational circuit
- Every combinational circuit is a Boolean function
- Later you might learn other equivalencies:
  finite 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.**

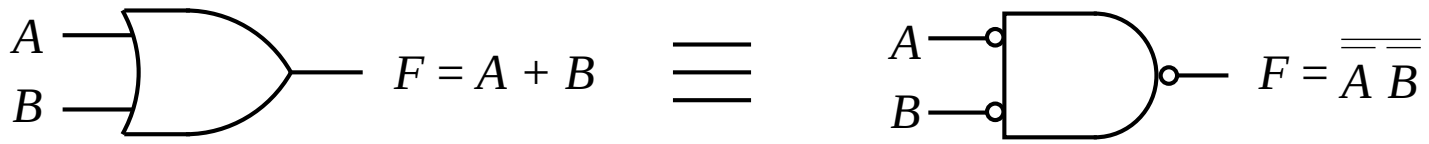


© 1999 M. Murdocca and V. Heuring

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: $A + B = \overline{\overline{A} + \overline{B}} = \overline{\overline{A}\,\overline{B}}$



© 1999 M. Murdocca and V. Heuring

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 3: J-K Flip-Flops

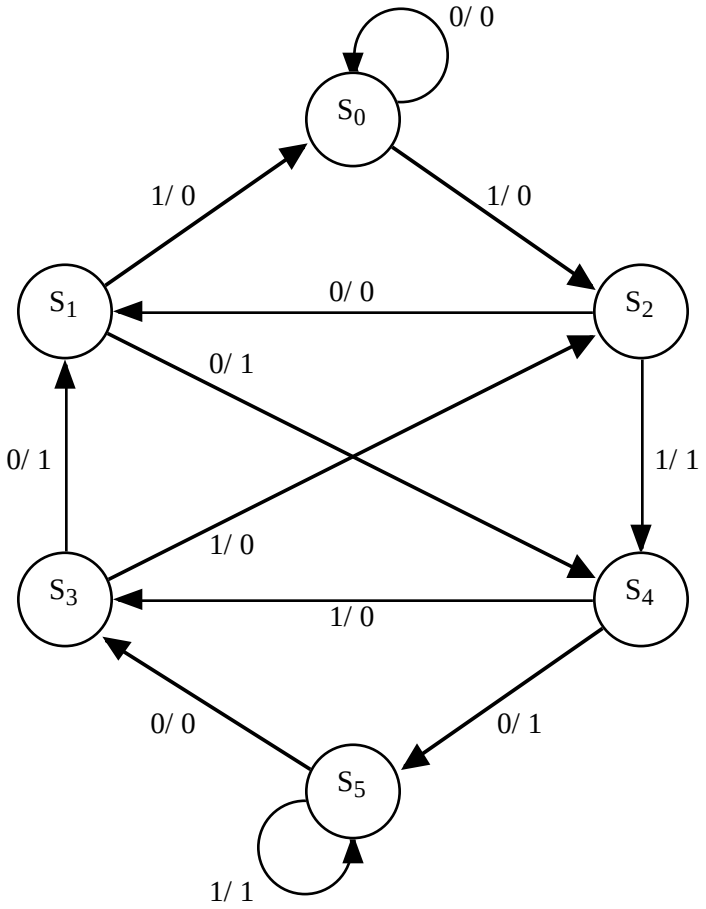
Due: Tuesday May 14, 2002

Objective

The objective of this assignment is to implement a finite state machine using J-K flip-flops.

Assignment

Consider the following transition diagram (from *Contemporary Logic Design*, Randy H. Katz, Benjamin-Cummings Publishing, 1994) for a finite state machine with 1 input bit and 1 output bit:



Your assignment is to implement this finite state machine using J-K flip flops. Assume that the state assignments are 000 for S_0, 001 for S_1, 010 S_2, 011 for S_3, 100 for S_4, and 101 for S_5.

1. In the truth table on the next page, let A, B and C be the current states and A', B' and C' be the next states stored in the J-K flip flops. (E.g., S_4 is assigned A=1, B=0 and C=0.) We also use D for the 1 bit input. Fill in the rest of the truth table using the excitation table for J-K flip flops. For example, in row 9, flip-flop A is currently storing 1 and must store 0 in the next state. To achieve this, we look at the 10 entry of the excitation table and note that the J input to flip-flop A (call it JA) can be set to anything, but the K input, KA, must be set to 1.
2. Use the Karnaugh maps provided to simplify the Boolean formulas for the J and K inputs to each J-K flip flop and for the output value z.
3. Implement the resulting circuit in DigSim.

Next time

- Transistors & Gates