COMP 122

Fall 2023 Rev7-13-23

ASSEMBLY Programming

Logic Design

Dr Jeff Drobman

website → drjeffsoftware.com/classroom.html

email → jeffrey.drobman@csun.edu

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Index

Section

Logic

Logic

Relational Operators

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal

any type → boolean

Logical Operators

&&	short circuit AND
||	short circuit OR
!	NOT
^	exclusive OR

boolean → boolean

if (x <= y+3) && x > 2 || _FLAG == true

_FLAG == true ⇔ _FLAG
_FLAG == false ⇔ ! _FLAG

- AND has 2 uses:
  1) Mask (1 lets in)
  2) Filter (0 keeps out)
- XOR has 2 uses:
  1) Bit complement/toggle
  2) Bit equal

Logic Book

allaboutcircuits.com

ALL ABOUT CIRCUITS

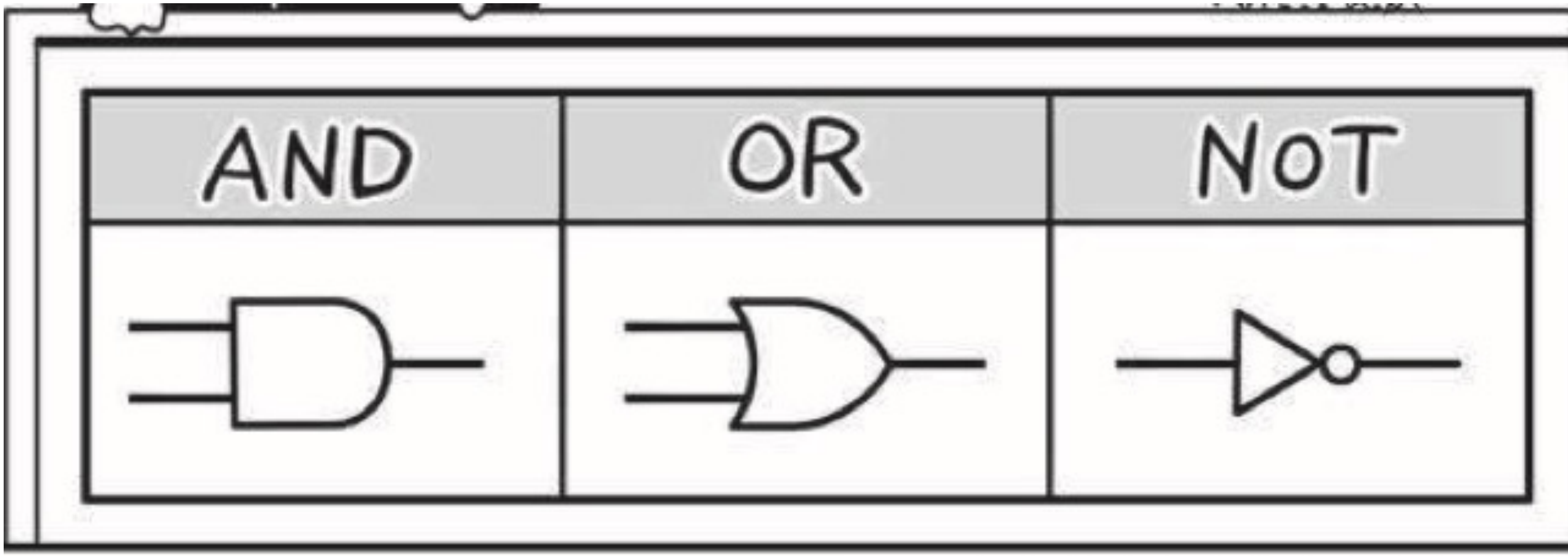
Chapters in this Volume

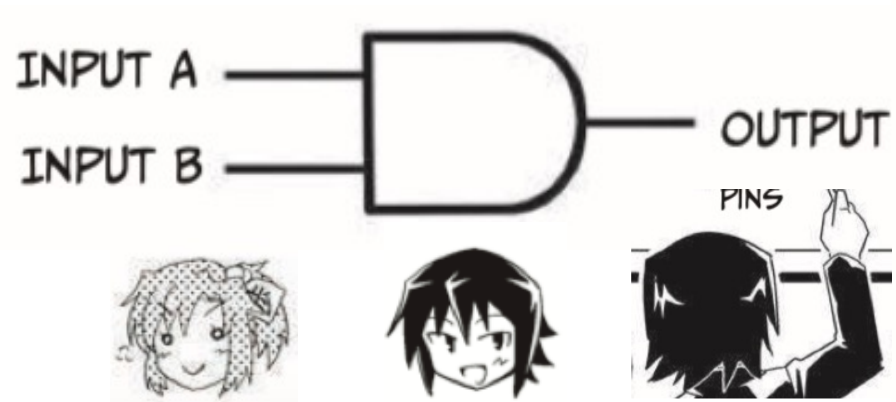
- *K-maps*

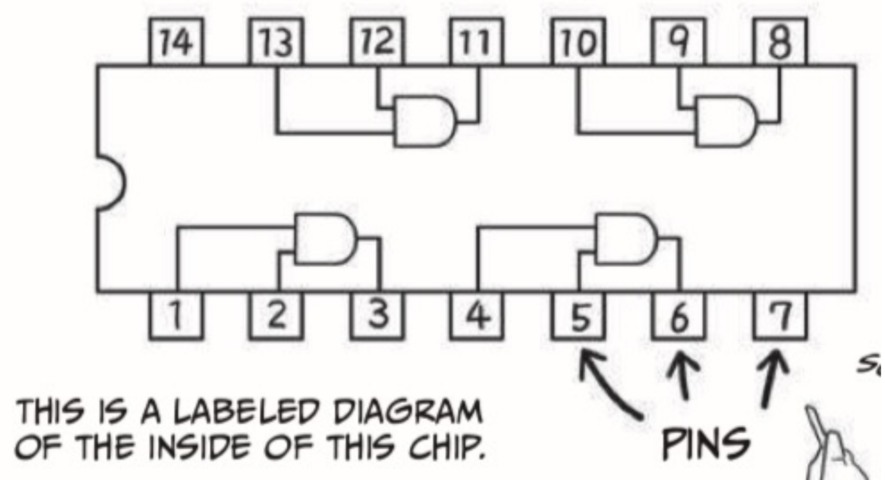
Logic Comic Book

Manga Guide

https://nostarch.com/download/MangaGuidetoMicroprocessors_sample_Chapter2.pdf







Truth Tables

OR

X	Y	X || Y
0	0	0
0	1	1
1	0	1
1	1	1

INclusive

AND

X	Y	X && Y
0	0	0
0	1	0
1	0	0
1	1	1

XOR

	X	Y	X ^ Y
pass	0	0	0
	0	1	1
flip	1	0	1
	1	1	0
	control	data	

EXclusive

- ❖ AND has 2 uses:
  1) Mask (1 lets in)
  2) Filter (0 keeps out)
- ❖ XOR has 2 uses:
  1) Bit complement/toggle
  2) Bit equal

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Bit-wise Operations

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Appendix G   p. 751   *NEW*

Operator	Name	Example	Result
&	Bitwise AND	11101 & 00111	00101
|	Bitwise OR	00010 | 11000	11010
^	Bitwise XOR	00111 ^ 11111	11000
~	1's complement	00111100	11000011
<<	Left shift (*2^n)	10101010 << 2	10101000
>>	Right shift, arith SE	10101011 >> 2	11101010
>>>	Right shift, logical	10101011 >> 2	00101010

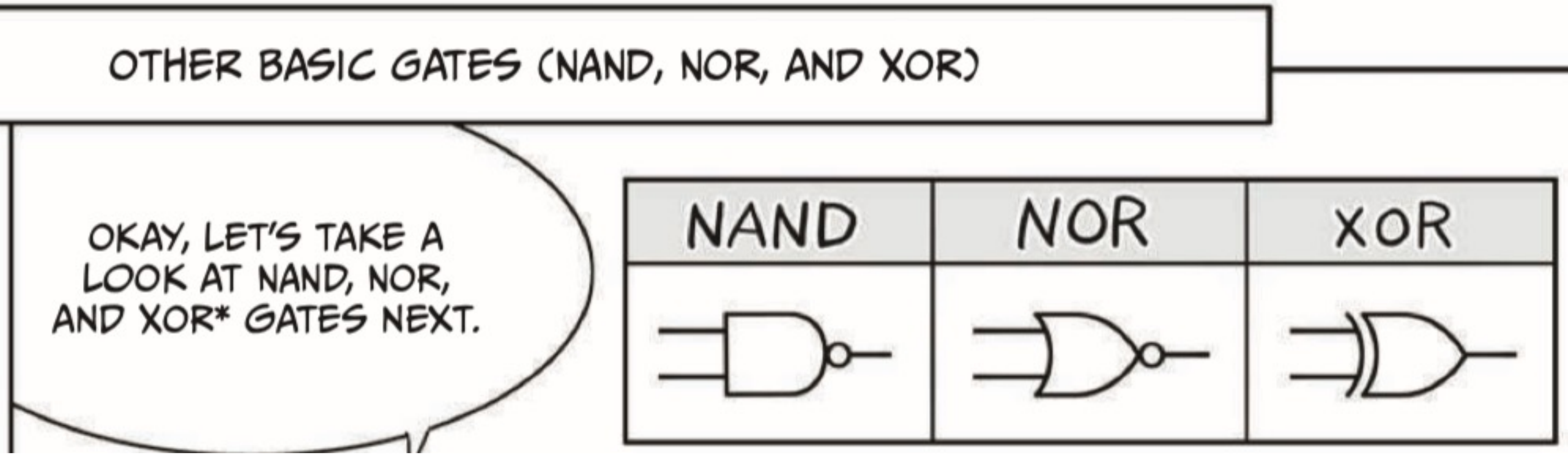
bit flip

*2^n

Integer types only

Compound Logic Gates

Manga Guide

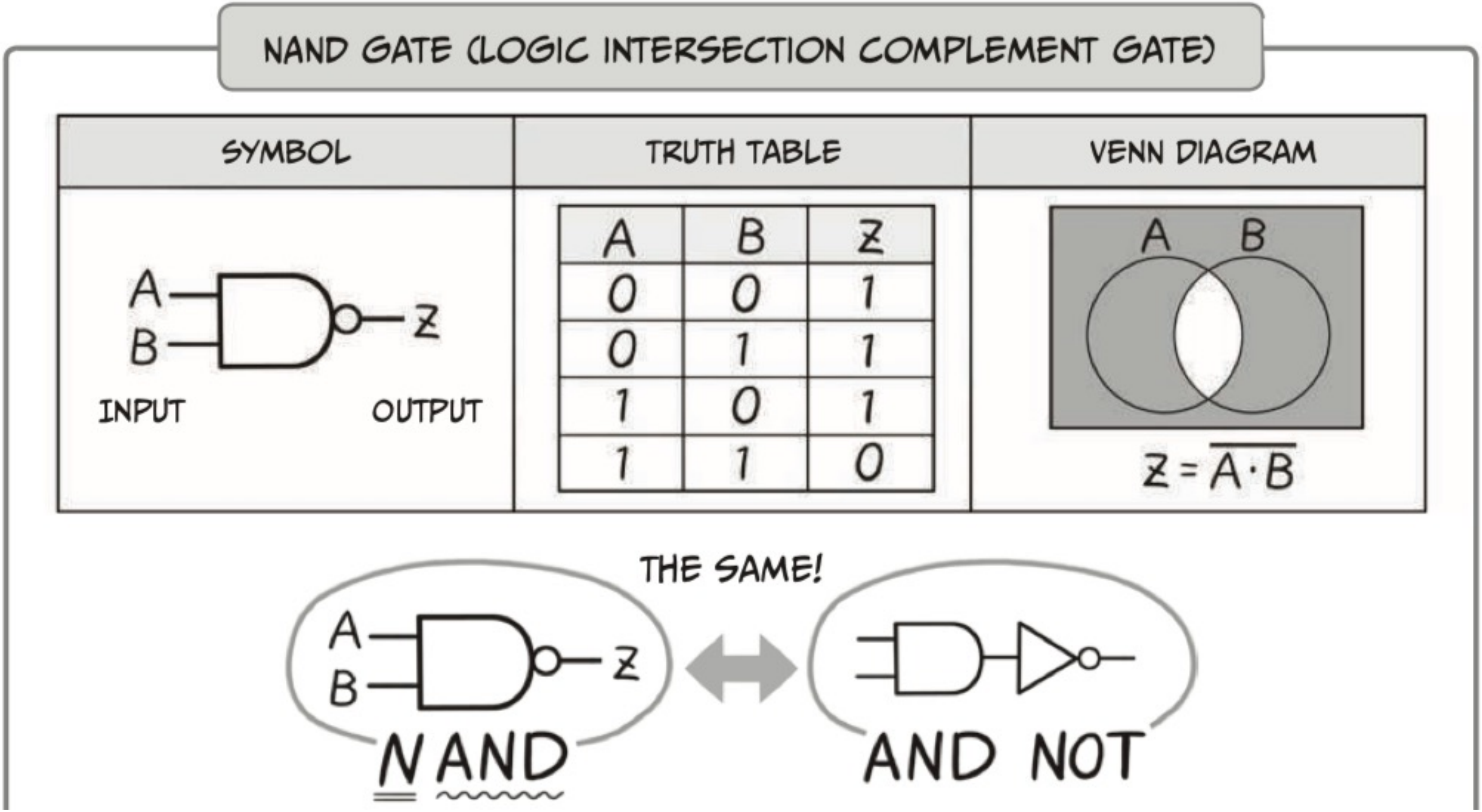


CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Compound Logic: NAND

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Manga Guide



A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

$$Z = \overline{A \cdot B}$$

Logic Gates: Polarity

DE MORGAN'S THEOREM

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

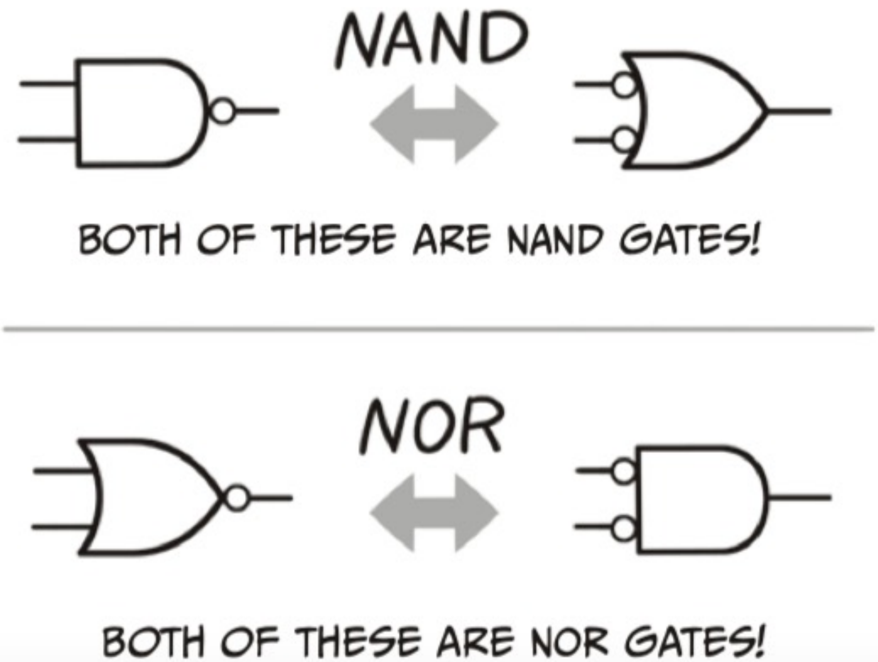
$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

Logic Gates: Polarity

DeMorgan's Law

Manga Guide

That's it! It also means that we can use De Morgan's laws to show our circuits in different ways. Using this technique, it's easy to simplify schematics when necessary.



Section

Transistors Make Gates

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Transistors to Computers

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Quora

If computers are really just many (billions) of on/off switches, how do they perform operations?

Jeff Drobman, Lecturer at California State University, Northridge (2016-present)

Answered just now

via a multi-level hierarchy of digital logic. transistors are combined to form logic “gates” of simple logic functions (AND, OR, NOT). the gates are combined to form more complex functions such as decoders, ALUs, and multiplexers. these functional blocks are then combined further into ever more complex logic blocks such as EU’s and then CPU cores. also, random logic implements the ICU as an FSM which includes pipelining. besides logic, computers have “storage” in the form of registers and memory (at up to 4 levels) via DRAM and SRAM cells formed from transistors (and a capacitor).

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

P→N→C MOS

DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman 2016-2023

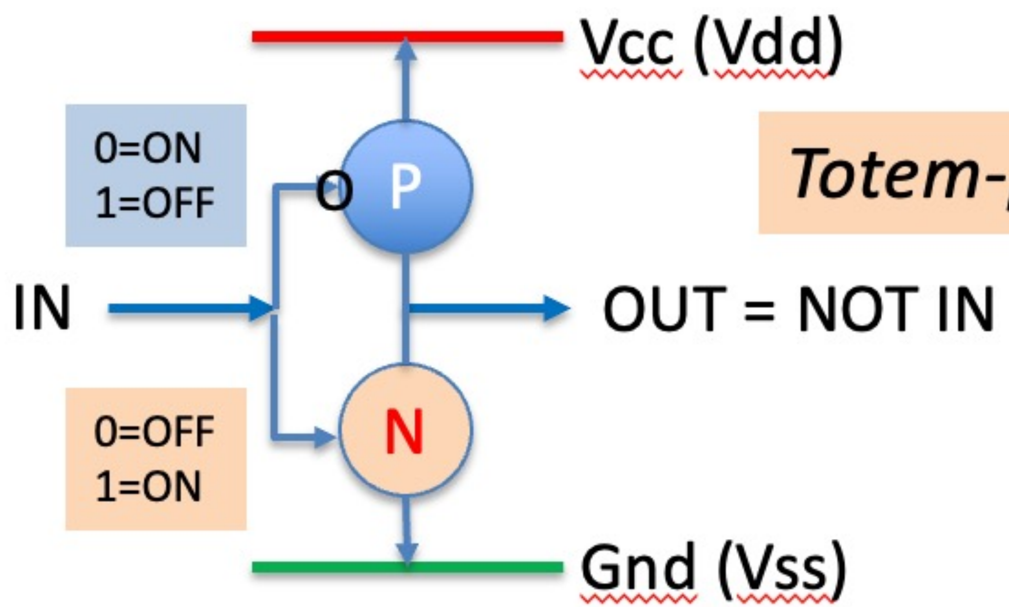
Device/Xtor
Physical
Level

Inverter/Gates

Complementary

CMOS

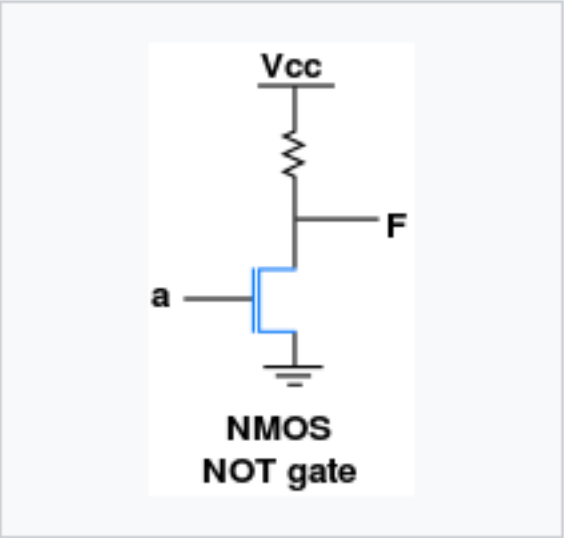
INVERTER



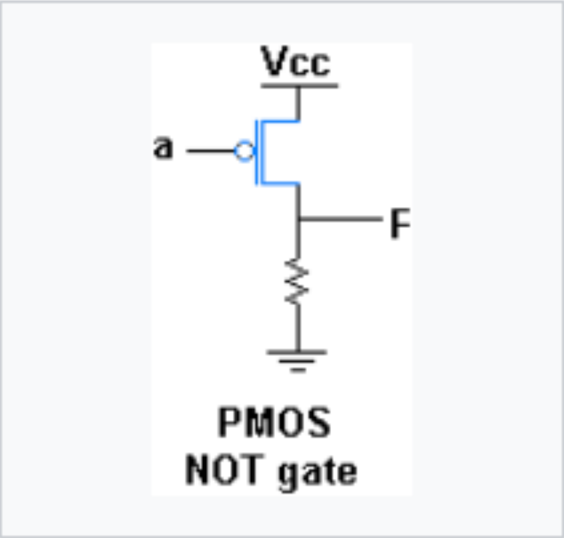
Totem-pole

MOS Gates

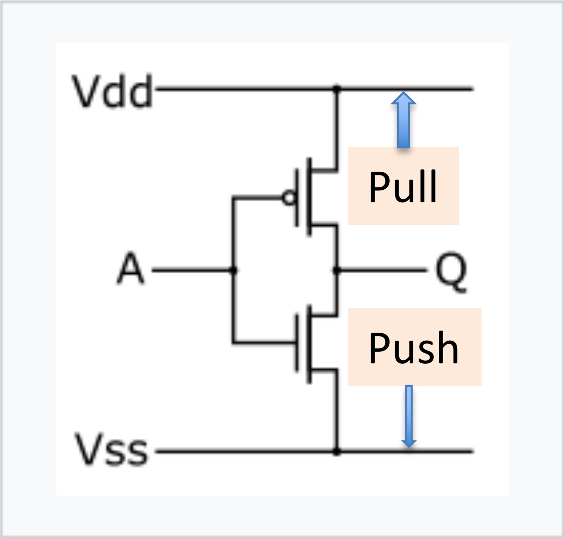
MOSFET



NMOS inverter



PMOS inverter

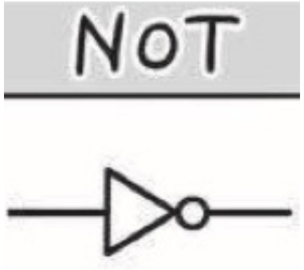


Static CMOS inverter

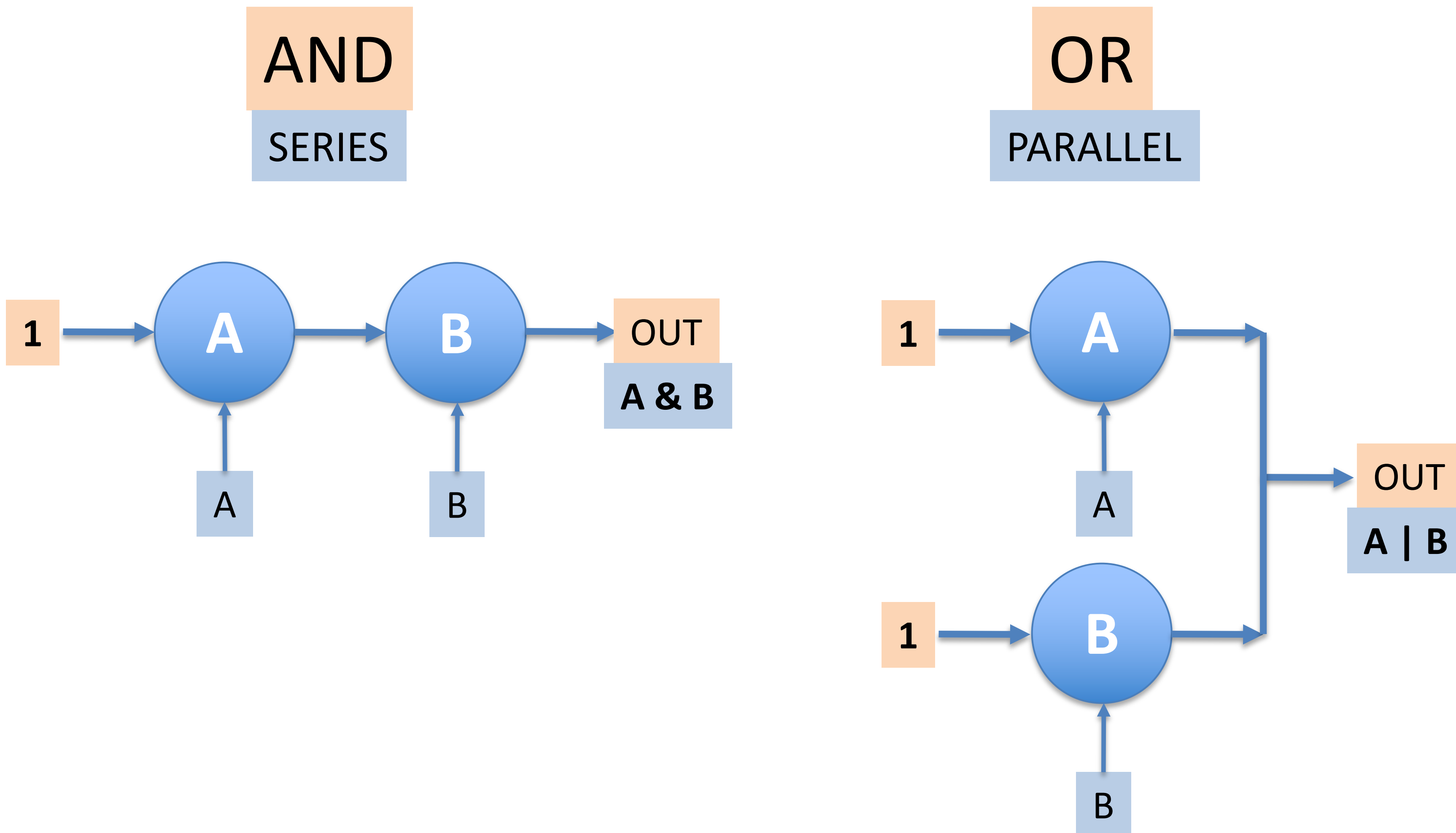
CMOS

P/N Totem pole

Push-Pull

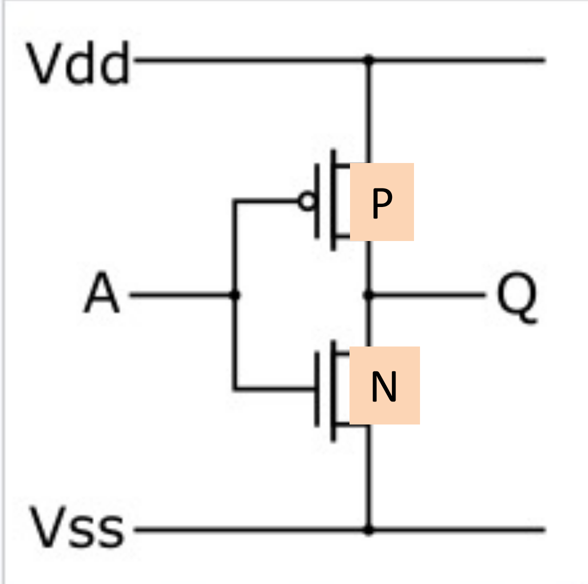


Logic Gates: AND, OR

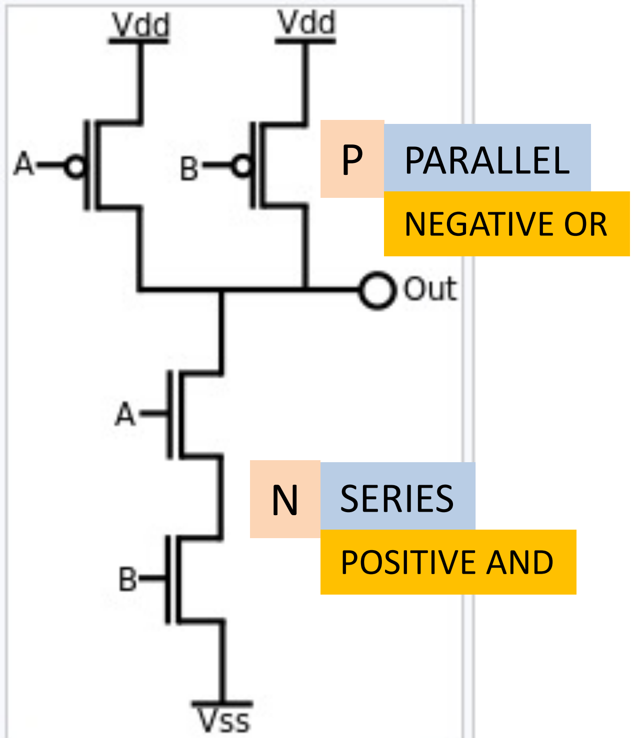


CMOS Gates

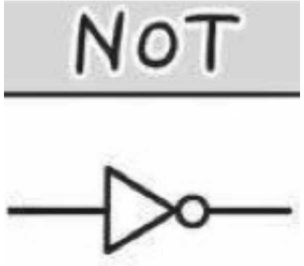
MOSFET

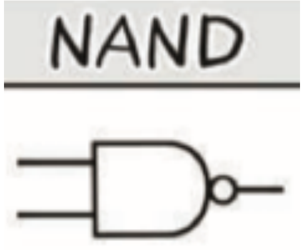


Static CMOS inverter. V_{dd} and V_{ss} are standing for drain and source respectively.



NAND gate in CMOS logic





Section

Arithmetic

ALU Ops

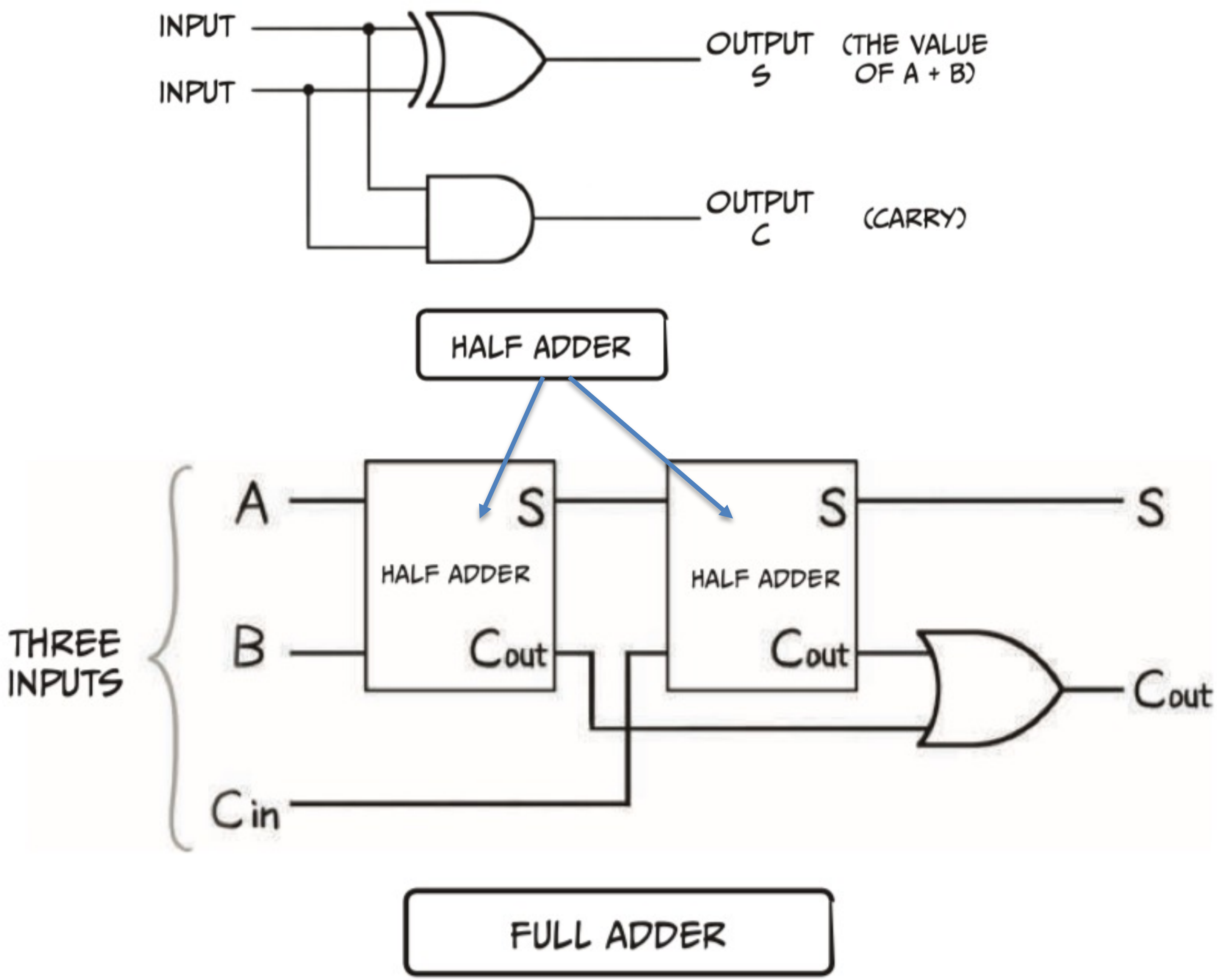
MARS

add $t1,$t2,$t3	Addition with overflow : set $t1 to ($t2 plus $t3)
add.d $f2,$f4,$f6	Floating point addition double precision : Set $f2 to double-precision floating p
add.s $f0,$f1,$f3	Floating point addition single precision : Set $f0 to single-precision floating p
addi $t1,$t2,-100	Addition immediate with overflow : set $t1 to ($t2 plus signed 16-bit immediate)
addiu $t1,$t2,-100	Addition immediate unsigned without overflow : set $t1 to ($t2 plus signed 16-bit
addu $t1,$t2,$t3	Addition unsigned without overflow : set $t1 to ($t2 plus $t3), no overflow
and $t1,$t2,$t3	Bitwise AND : Set $t1 to bitwise AND of $t2 and $t3
andi $t1,$t2,100	Bitwise AND immediate : Set $t1 to bitwise AND of $t2 and zero-extended 16-bit im

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Adders

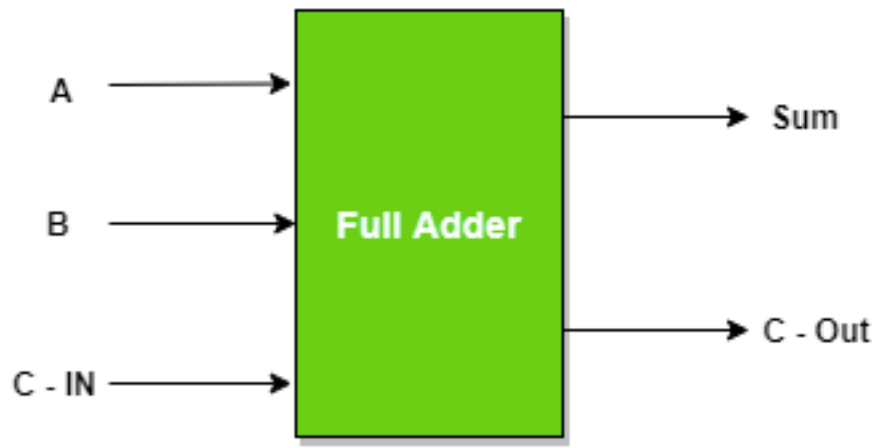
DR JEFF SOFTWARE
INDIE APP DEVELOPER



Adders

Full Adder in Digital Logic

GeeksforGeeks



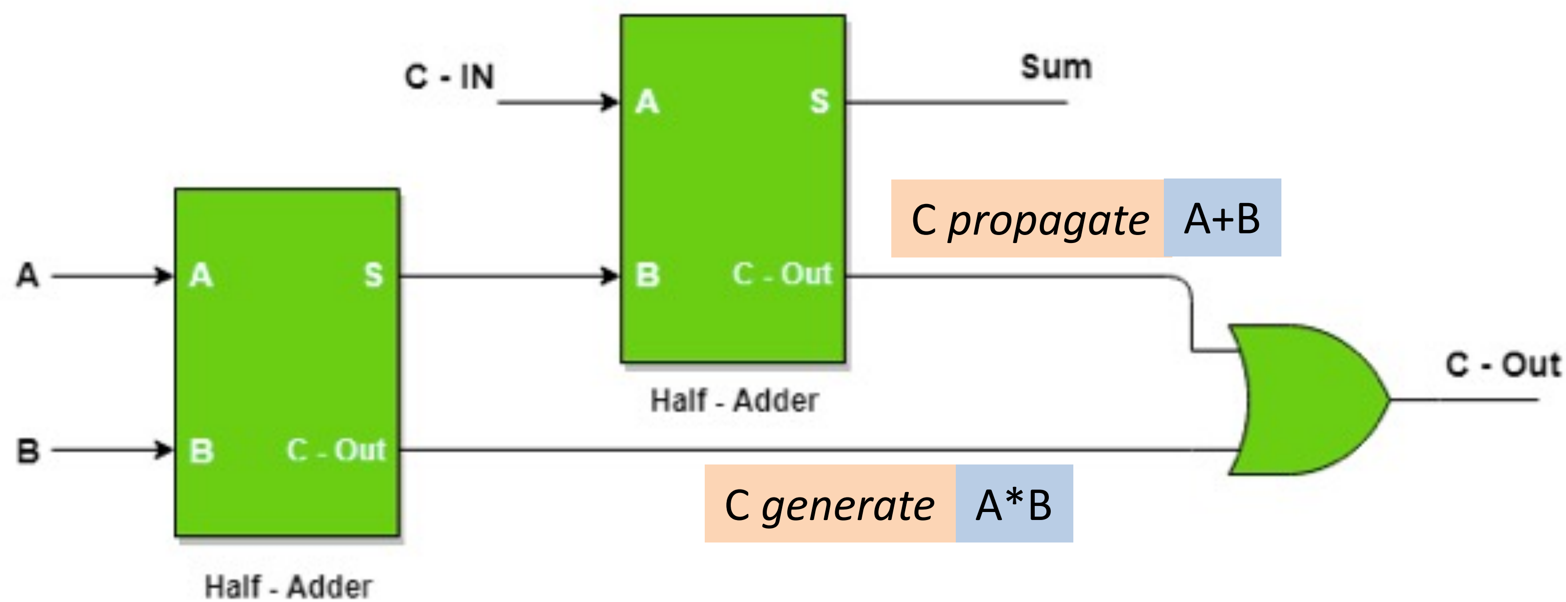
Inputs			Outputs	
A	B	C – IN	Sum	C - Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Adders

GeeksforGeeks

Implementation of Full Adder using Half Adders

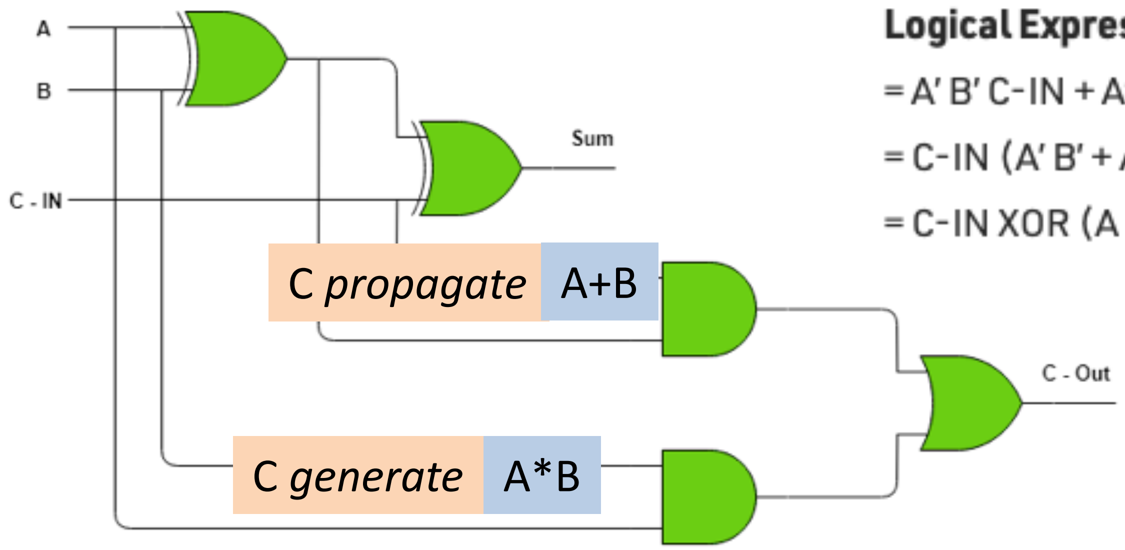
2 Half Adders and a OR gate is required to implement a Full Adder.



Adders

GeeksforGeeks

Therefore COUT = AB + C-IN (A EX − OR B)



Full Adder logic

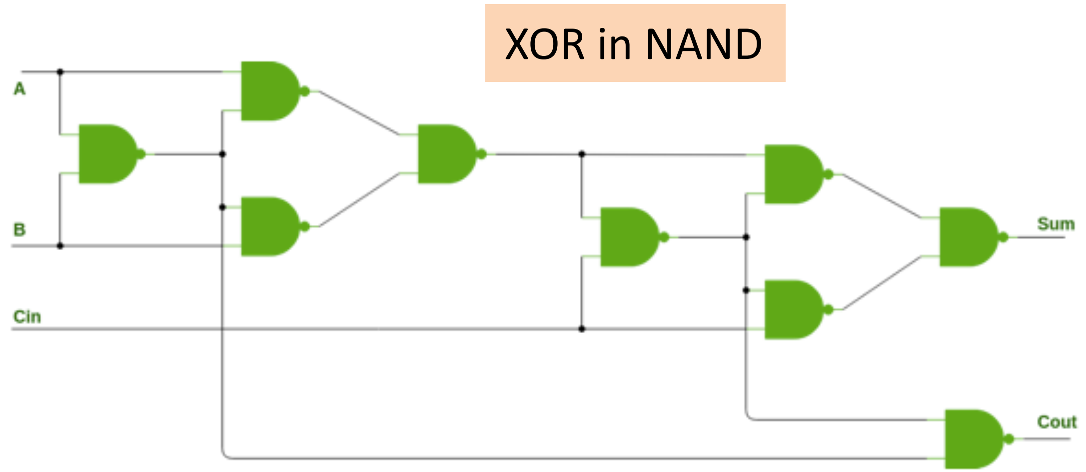
Logical Expression for SUM:

= A′ B′ C-IN + A′ B C-IN′ + A B′ C-IN′ + A B C-IN

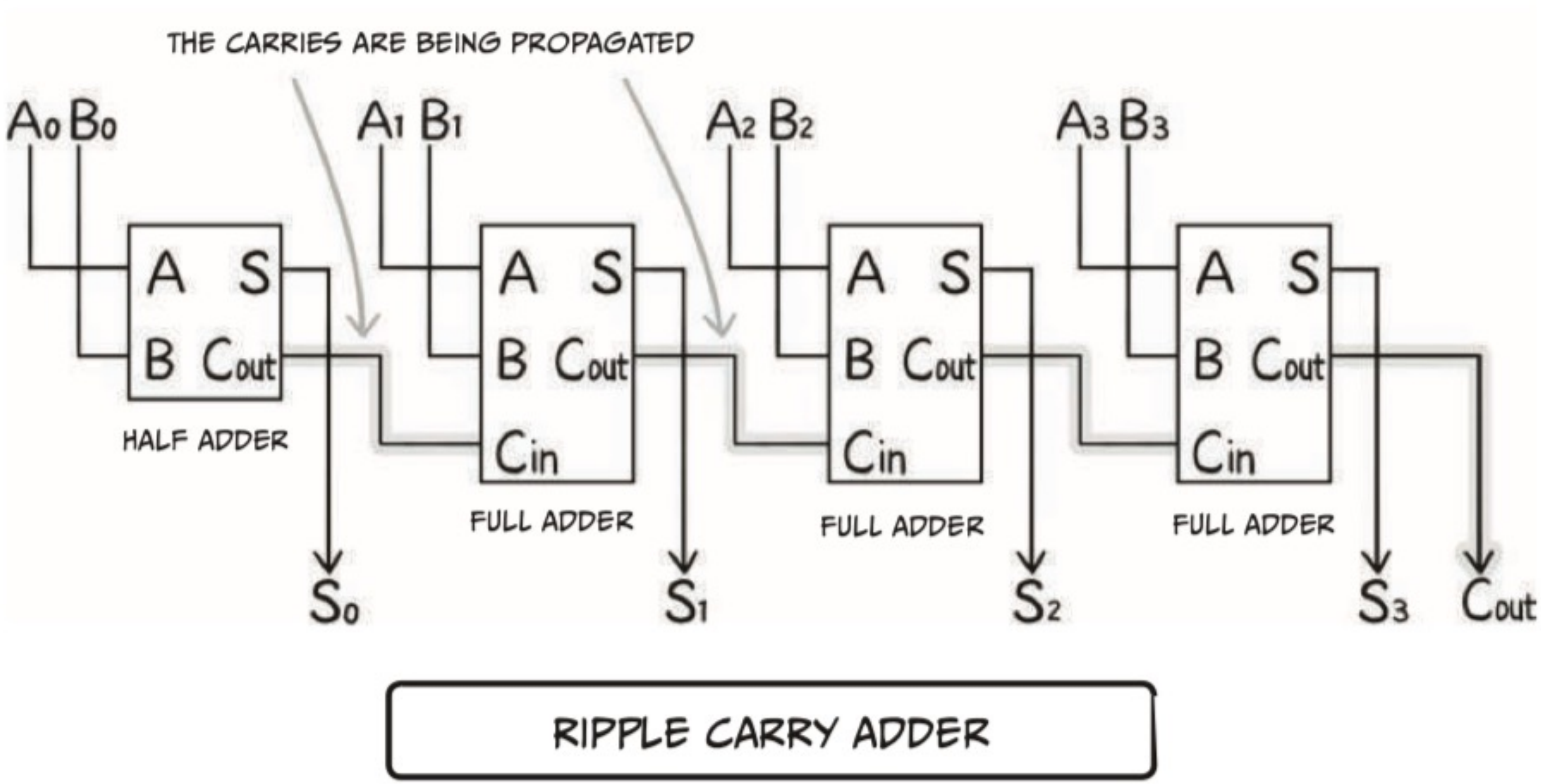
= C-IN (A′ B′ + A B) + C-IN′ (A′ B + A B′)

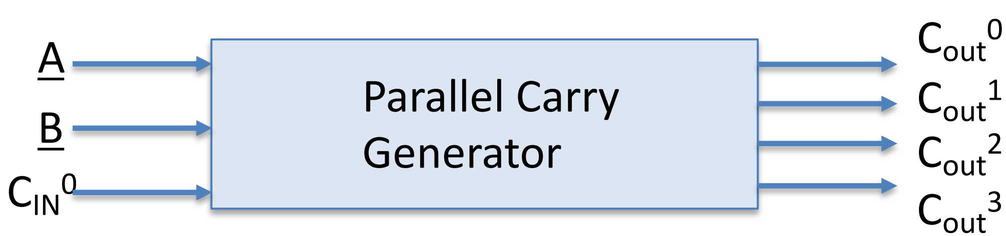
= C-IN XOR (A XOR B)

Implementation of Full Adder using NAND gates:



Adders: Ripple Carry



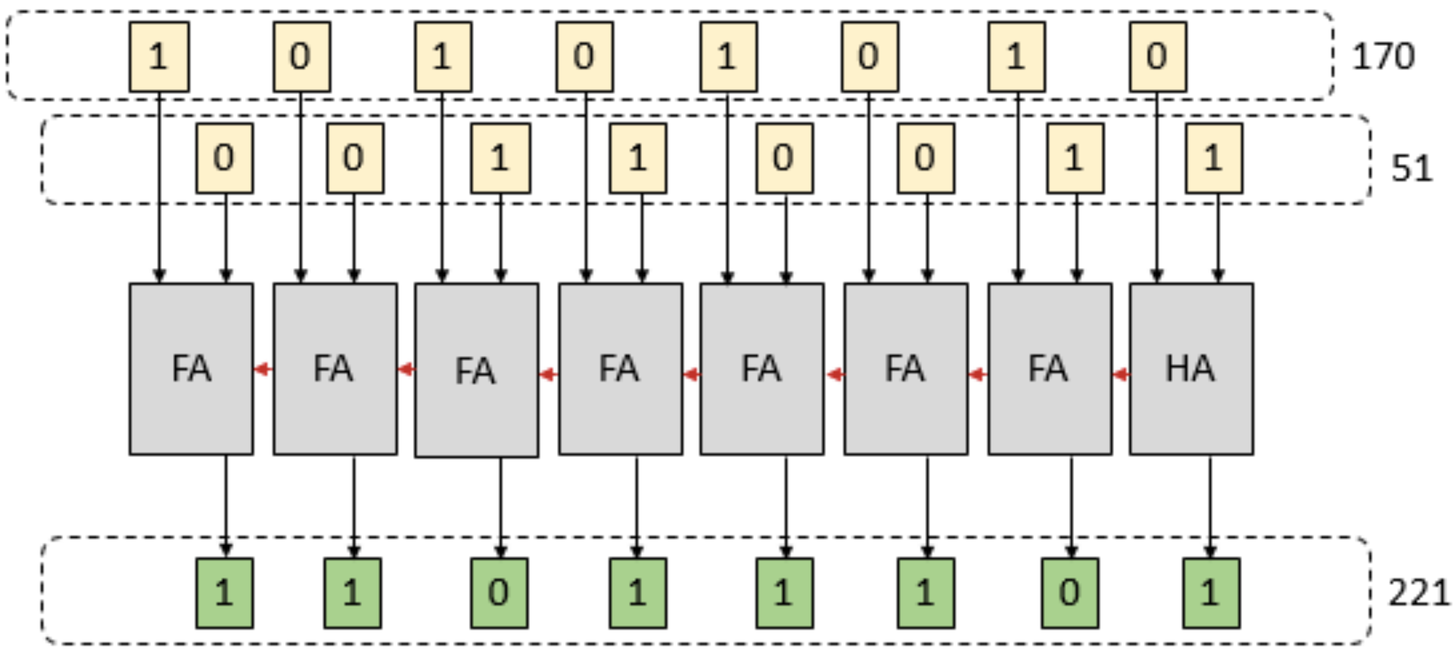


Adder

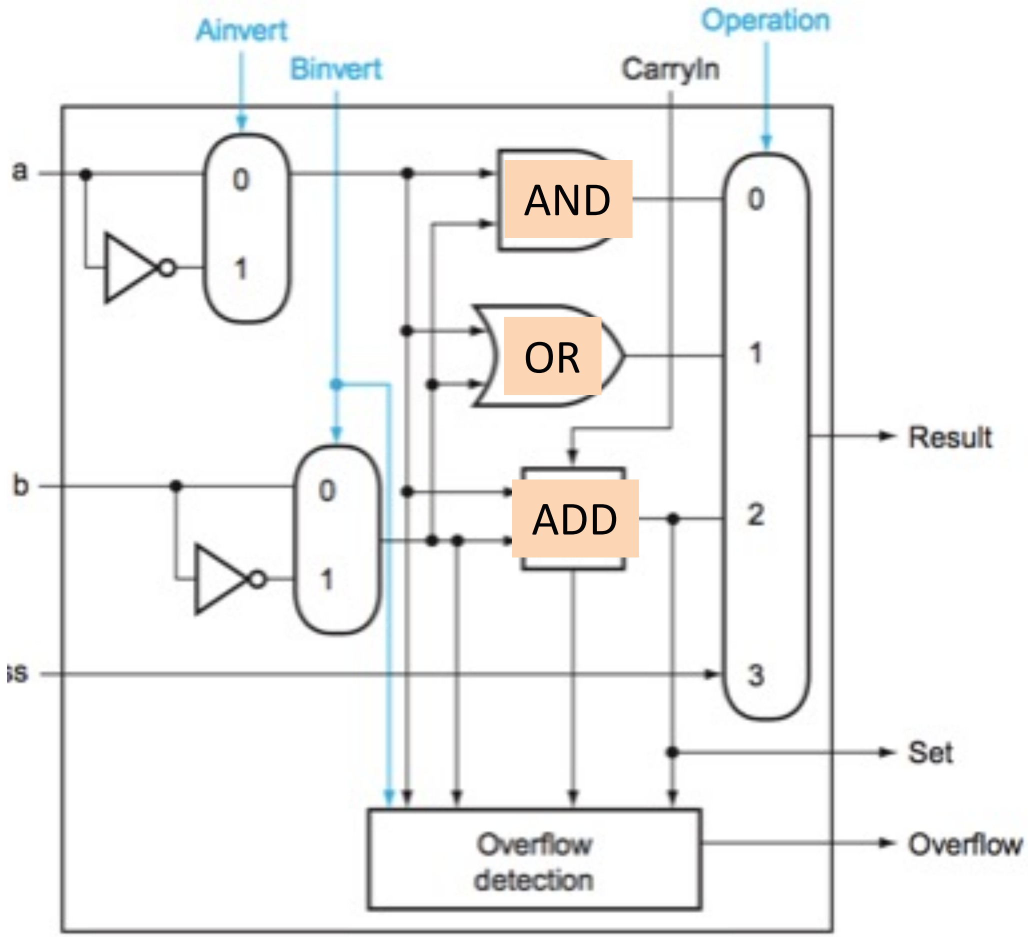
WOLFRAM

The 8-Bit Adder Principle

The 8-bit adder adds the numbers digit by digit, as can be seen in the schematic diagram below. In this example, the integers 170 and 51 represent input a and b, respectively, and the resulting output is the sum 221. The first adder does not have any carry-in, and so it is represented by a half adder (HA) instead of a full adder (FA).



MIPS/MARS ALU



MIPS ALU in Verilog

HDL

Figure 8.5.15: A Verilog behavioral definition of a MIPS ALU (COD Figure B.5.15).

```
module MIPSALU (ALUctl, A, B, ALUOut, Zero);
   input [3:0] ALUctl;
   input [31:0] A,B;
   output reg [31:0] ALUOut;
   output Zero;
   assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0
   always @(ALUctl, A, B) begin //reevaluate if these change
      case (ALUctl)
         0: ALUOut <= A & B;
         1: ALUOut <= A | B;
         2: ALUOut <= A + B;
         6: ALUOut <= A - B;
         7: ALUOut <= A < B ? 1 : 0;
         12: ALUOut <= ~(A | B); // result is nor
         default: ALUOut <= 0;
      endcase
    end
endmodule
```

AMD Products (1971)

- Analog
  - Op Amps
  - Voltage Regulators
- Packages
  - DIP
  - Others

Electronic Device Cos.

1972

Nationwide Distributor for the world's finest lines of quality electronic components!

HAMILTON

HAMILTON
ELECTRO SALES

(213) 870-7171

ORANGE COUNTY
(714) 522-8200
ZENITH 2-1474

10912 W. Washington Blvd. • Culver City, California 90230

ADVANCED MICRO DEVICES ...... Integrated Circuits

BOURNS .... Potentiometers

BURROUGHS .... Nixie Tubes, Digital Displays

FAIRCHILD ...... Semiconductors, Integrated Circuits, Microwave & Opto Devices

GENERAL ELECTRIC .... Semiconductors, Capacitors, Meters, Opto Devices, Relays, Volt Pacs

INTEL .... Integrated Circuits

KEMET .... Solid Tantalum, Ceramic Capacitors

LITRONIX .... Opto Devices

MEPCO/ELECTRA .... Potentiometers, Resistors

MONSANTO .... Opto Devices

MOTOROLA ...... Semiconductors, Integrated Circuits, Opto Devices

NATIONAL SEMICONDUCTOR .... Semiconductors, Integrated Circuits, Opto Devices

POTTER & BRUMFIELD — WOOD ... Relays, Switches, Circuit Breakers

RCA .... Semiconductors, Integrated Circuits, Opto Devices

SIGNETICS .... Integrated Circuits

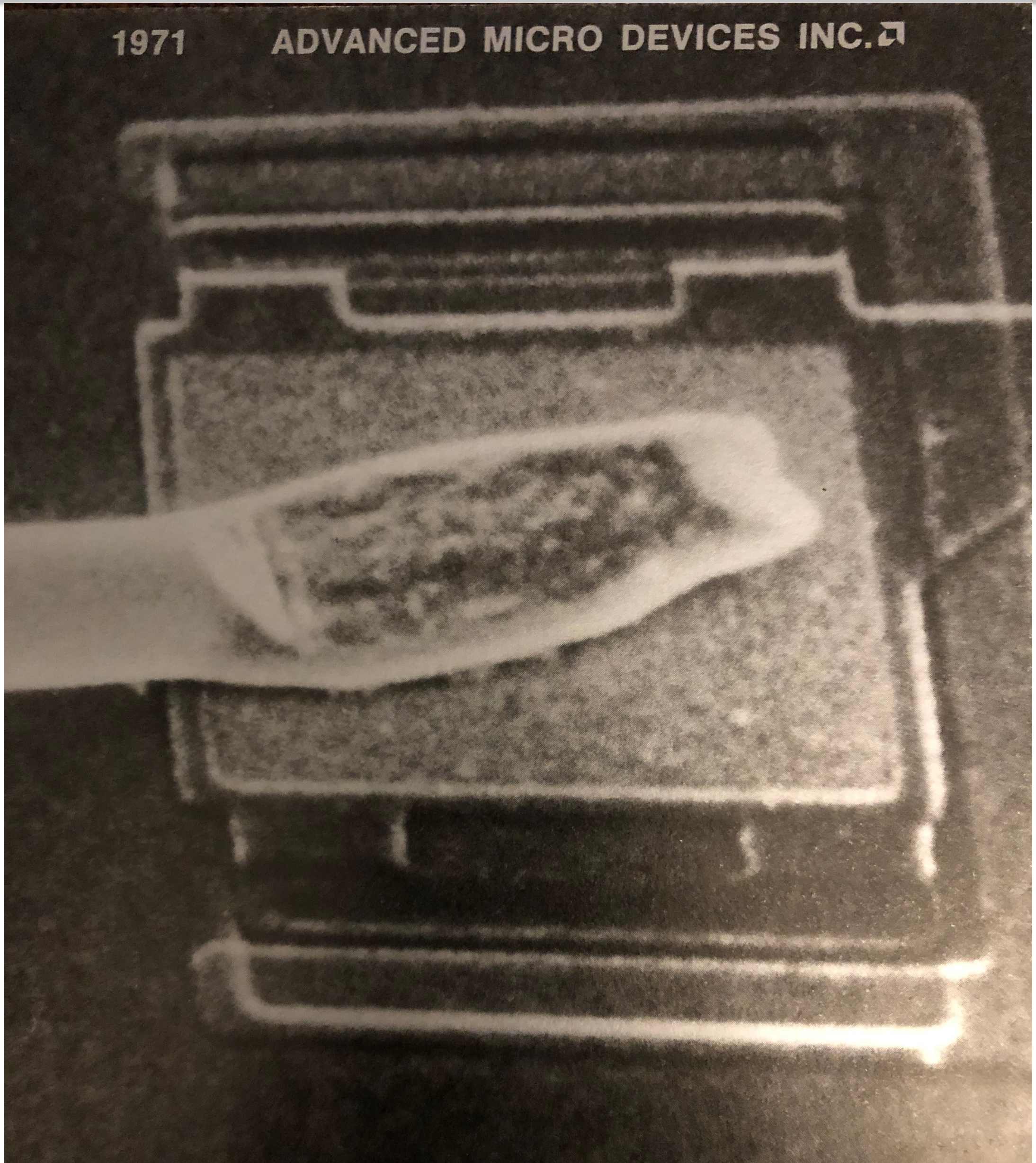
SILICONIX .... Semiconductors, Integrated Circuits

WESTINGHOUSE .... Semiconductors

STOCKING INVENTORY

IN EVERY LOCATION

AMD Catalog 1971



AMD Analog (Linear)

LINEAR CIRCUITS

Operational Amplifiers

General-Purpose Frequency Compensated
Dual
747/747C
(dual 741/741C)
Single
741/741C
107/207/307

General Purpose Uncompensated
Dual
Am1501
(dual 101A)
Single
748/748C
101A/201A/301A
108/208/308
101/201/301

High Speed
(10V/μsec)
715/715C
102/202/302
110/210/310

Low Power
108/208/308
108A/208A/308A

General-Purpose Frequency Compensated Operational Amplifiers

This class of circuit provides the trade-off of electrical parameters required for general-purpose linear and non-linear computing applications. These circuits should be input/output protected, latch-up free, Vos adjustable and pin-for-pin compatible with the 709.

General-Purpose Uncompensated Operational Amplifiers

Applications where amplifier phase margins will allow tailored-bandwidth operation in non-unity gain applications.

High-Speed Operational Amplifiers

The primary applications for these circuits are where internally

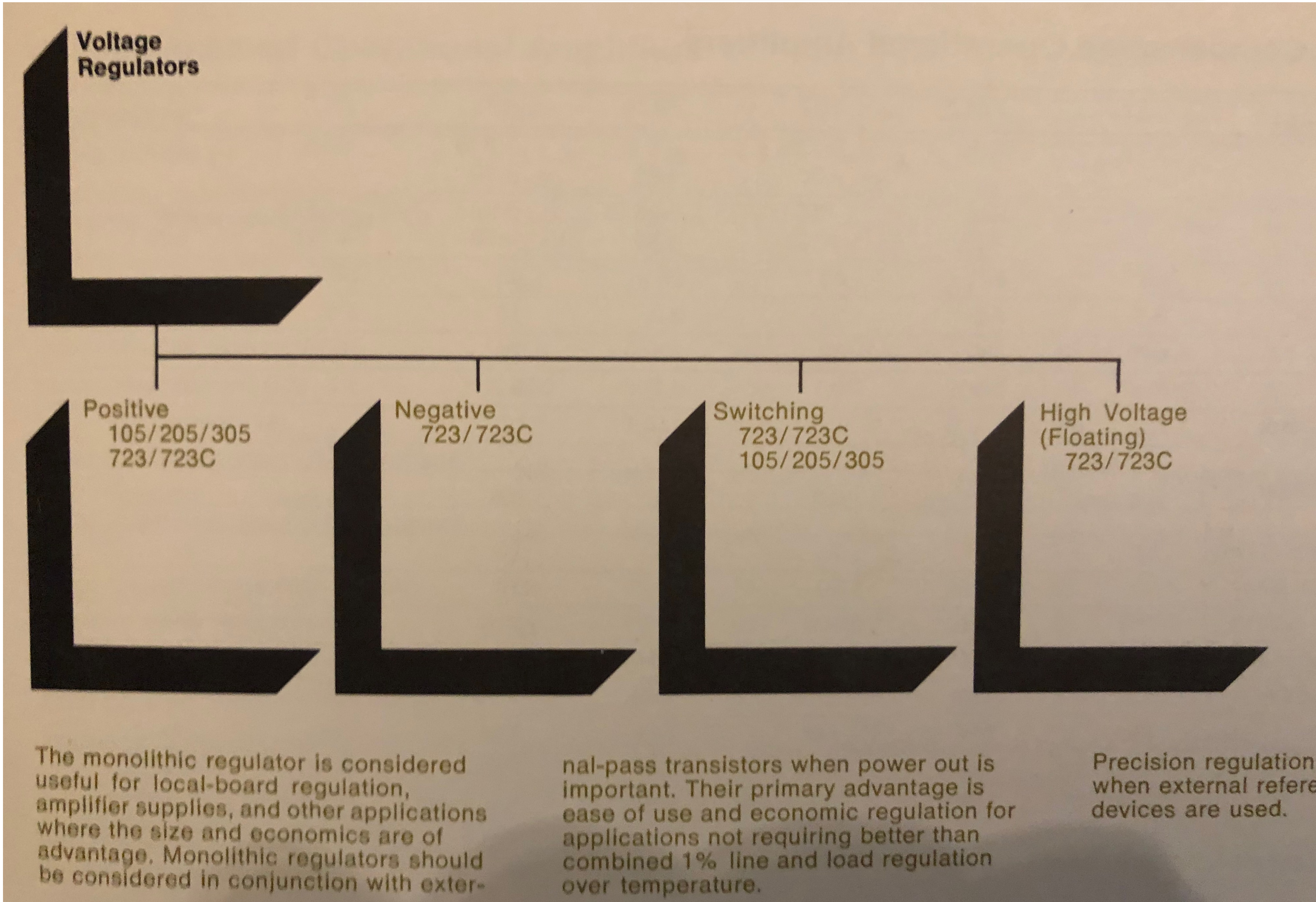
Because of the high-performance n of the application it may be of adva for the amplifier to have bandwidth tailoring capability (not internally compensated). These applications do require competent understanding of gain phase and compensation the

Low-Power Operational Amplifiers

AMD Analog (Linear)

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

DR JEFF SOFTWARE
INDIE APP DEVELOPER



Voltage Regulators

Positive
105/205/305
723/723C

Negative
723/723C

Switching
723/723C
105/205/305

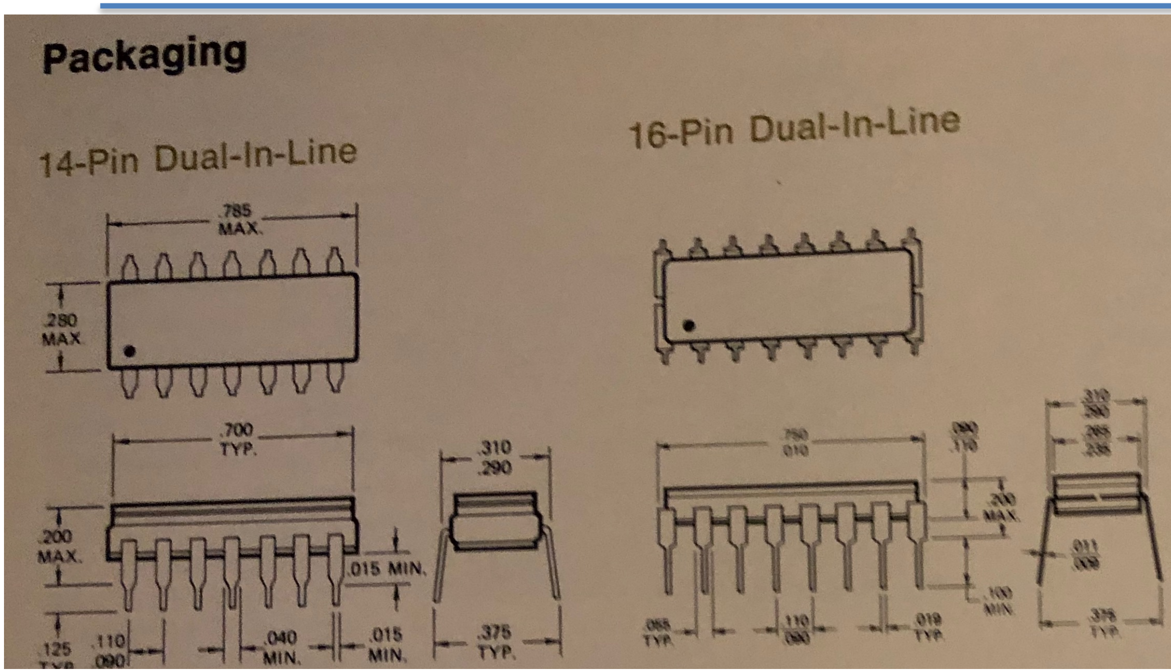
High Voltage
(Floating)
723/723C

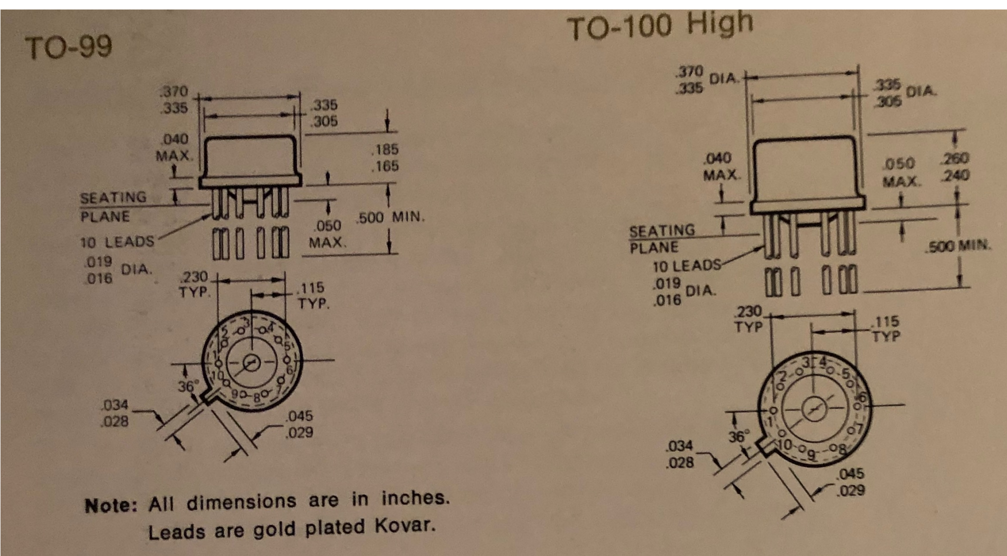
The monolithic regulator is considered useful for local-board regulation, amplifier supplies, and other applications where the size and economics are of advantage. Monolithic regulators should be considered in conjunction with exter-nal-pass transistors when power out is important. Their primary advantage is ease of use and economic regulation for applications not requiring better than combined 1% line and load regulation over temperature.

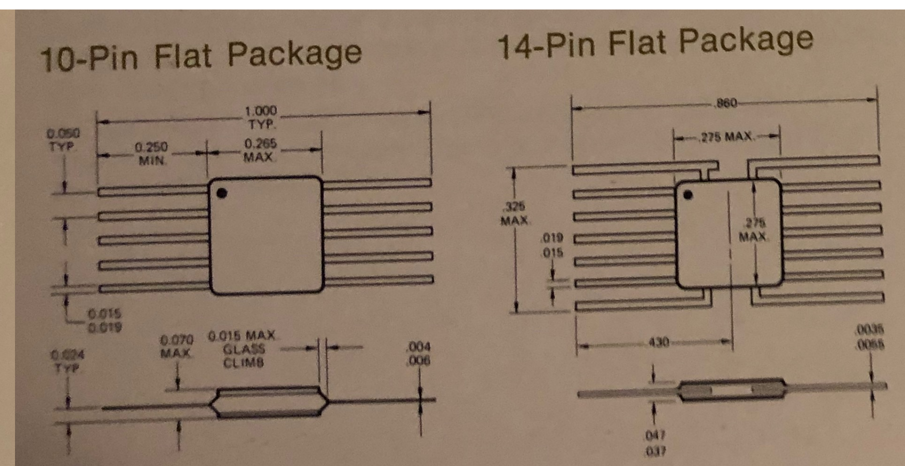
Precision regulation when external refer devices are used.

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

AMD Packages







Section

Logic

- Logic Product Lines
  - 54/7400
  - F9300/Am9300
- ALU slice (4-bit)
  - 54/74181
  - Am9340

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

AMD Digital MSI

DR JEFF SOFTWARE
INDIE APP DEVELOPER

1971

Am MSI

Registers

Am9300	Four-Bit Shift Register
Am9328	Dual Eight-Bit Shift Register

Counters

Am2501	Binary Hexadecimal Up/Down Counter
Am8284	Binary Hexademical Up/Down counters
Am8285	BCD Decade Synchronous Up/Down Counter
Am9306	BCD Decade Synchronous Up/Down Counter
Am9310	BCD Decade Counter
Am9316	Four-Bit Binary Counter

Encoders

Am9318	Eight-Input Priority Encoder

Multiplexers

Am9309	Dual Four-Input Multiplexer
Am9312	Eight-Input Multiplexer
Am9322	Quad Two-Input Multiplexer

Latches

Am9308	Dual Four-Bit Latch
Am9314	Four-Bit Latch

Operators

Am54/74181	Four-Bit Arithmetic Logic Unit
Am54/74182	Look-Ahead Carry Generator
Am9304	Dual Full Adder
Am9324	Five-Bit Comparator
Am9340	Four-Bit Arithmetic Logic Unit
Am9341	Four-Bit Arithmetic Logic Unit
Am9342	Carry Look Ahead

Decoder and Demultiplexer

Am9301	Demultiplexer/One of Ten Decoder
Am9311	Demultiplexer/One of Sixteen Decoder

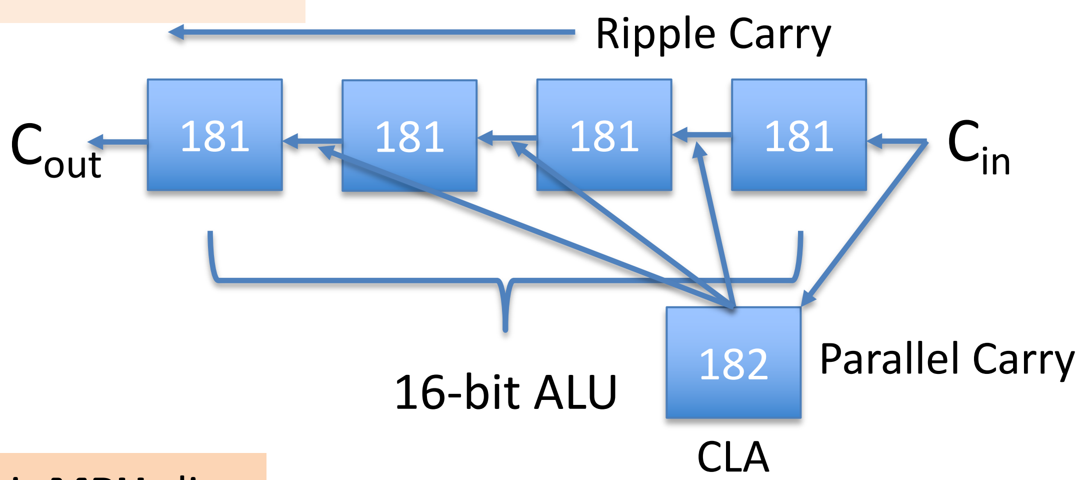
Logic IC's: ALU *Slices*

Bit-slice 1965-75

- Fairchild 9300 series
- Signetics
- National
- Motorola
- Texas Instruments

Texas Instruments

- **54/7400 Series**
  - 54 → **Mil** temp
  - 74 → **Com'l** temp
- 54/74SN**181**
  - 4-bit **ALU** slice



Am2505 2x4-bit **Multiplier** slice

- 4-bit **MPU** slice

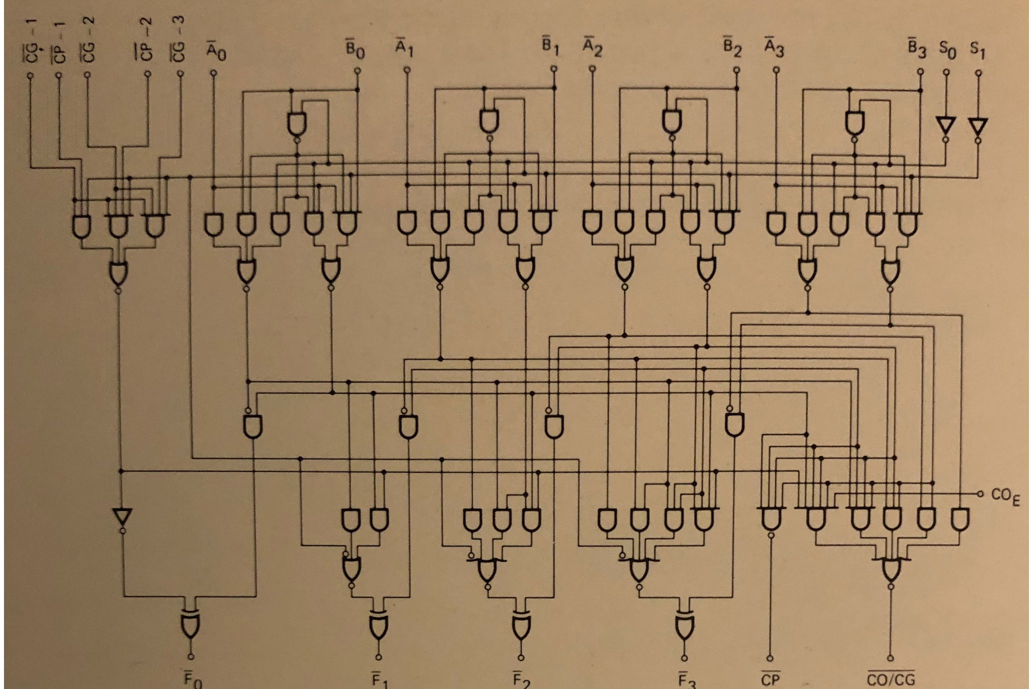
Replaced by **Am2900** *family* → Am29**01** — ALU + Register file — *microprogrammed*
Am29**10** — *Microprogram sequencer*
Am29**02** — CLA

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

AMD ALU – Am9340

DR JEFF SOFTWARE
INDIE APP DEVELOPER

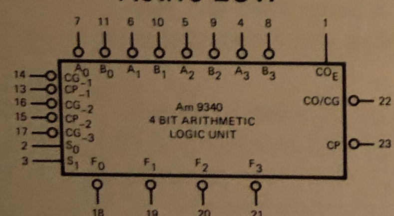
Logic Diagram/Symbols



Characteristics

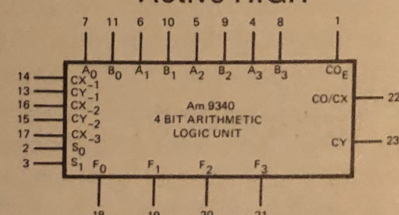
Typical Delays	Addition over 4 Bits	19 ns
	Addition over 16 Bits	33 ns
	Addition over 28 Bits	47 ns
Typical Power Dissipation		400 mW

Active LOW



V_{CC} = Pin 24
GND = Pin 12

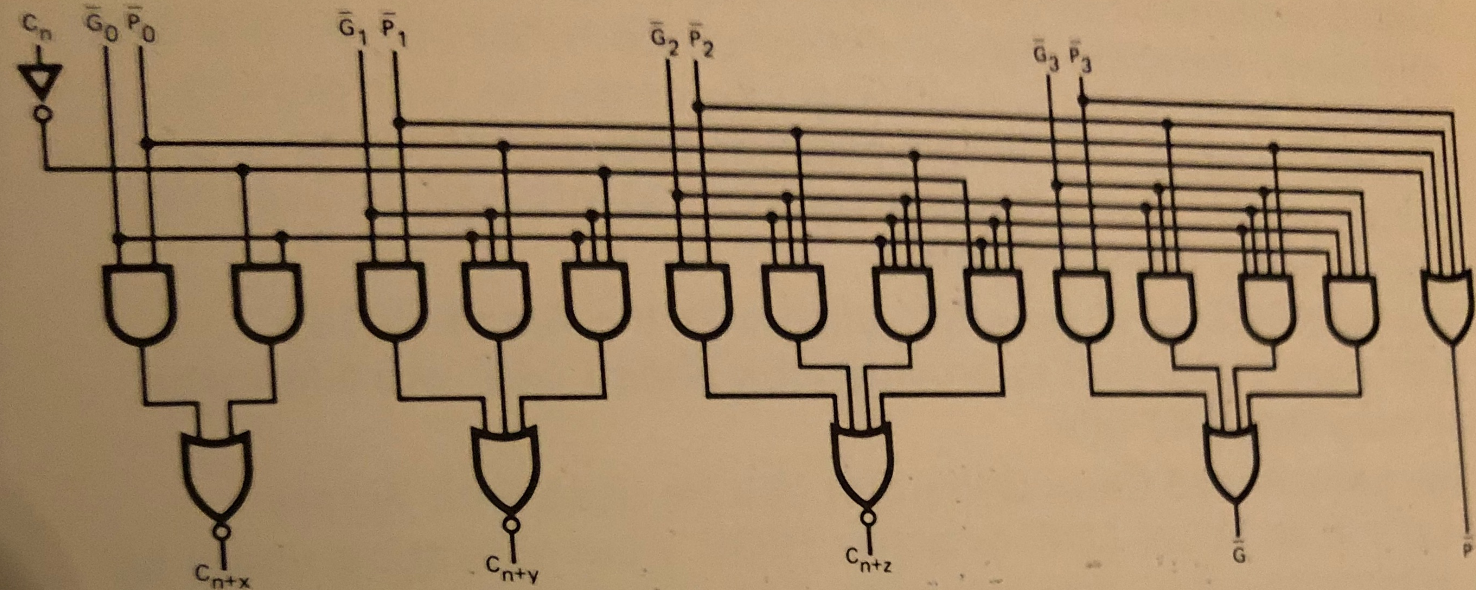
Active HIGH

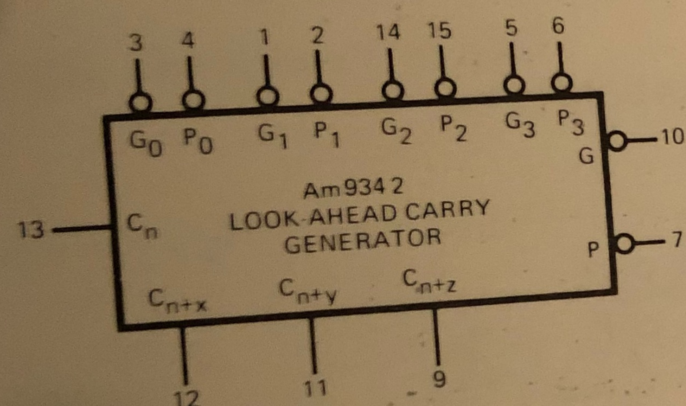


CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

AMD CLA – Am9342

Logic Diagram/Symbol





Characteristics

Typical Delay	$\overline{C}_{PO}$ to $\overline{C}_P$	7 ns
	C_{IN} to $\overline{C}_P$	13 ns
Typical Power Dissipation		135 mW

V_{CC} = Pin 16
GND = Pin 8

Section

Logic

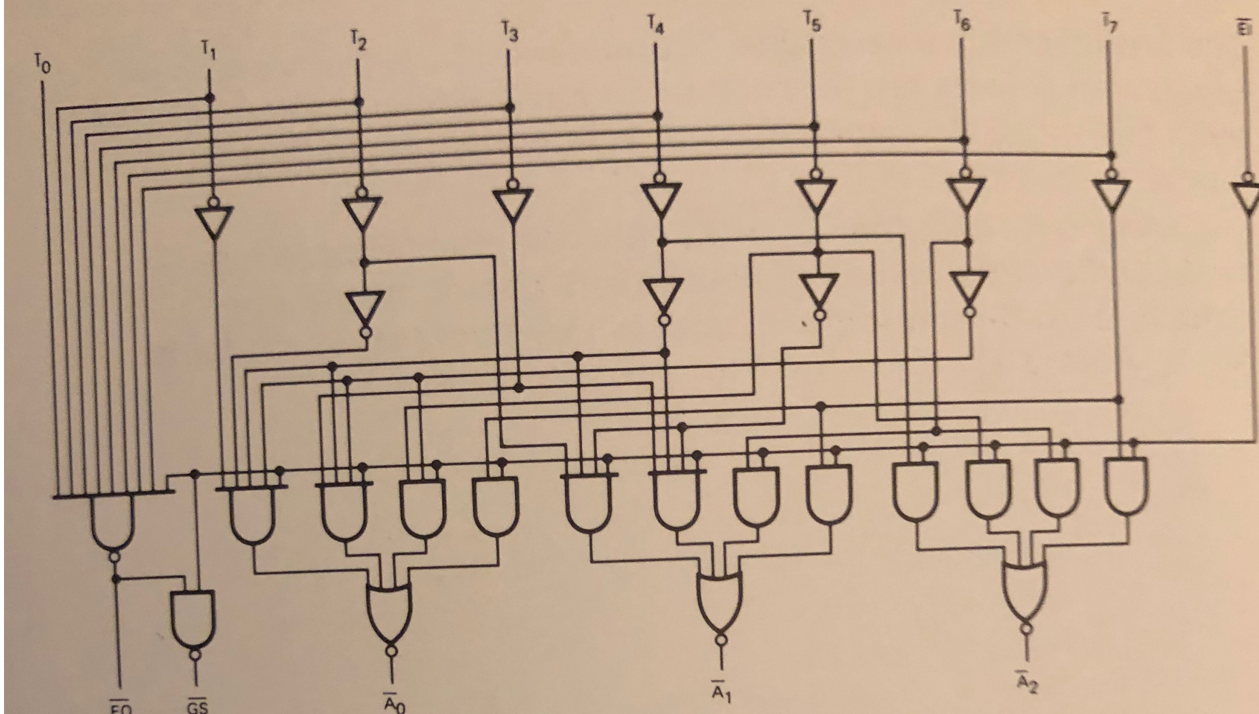
- PIC
- Muxes
- Decoders
- Logic functions

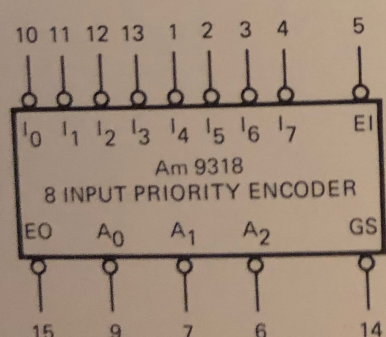
CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

PIC: Priority Interrupt Enc

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Logic Diagram/Symbol



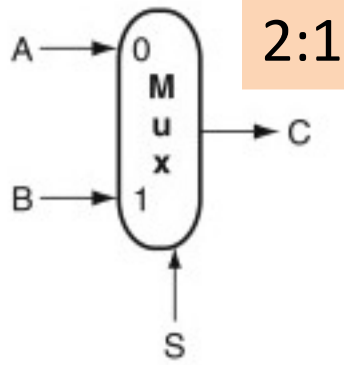


Characteristics

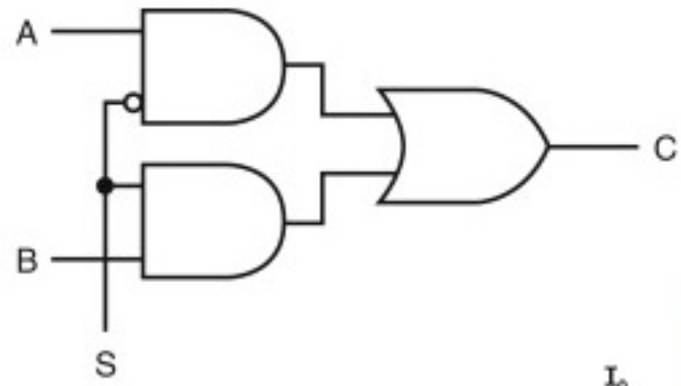
Typical Delay $\overline{I}$ to $\overline{A}$ 16 ns
Typical Power Dissipation 250 mW

V_{CC} = Pin 16
GND = Pin 8

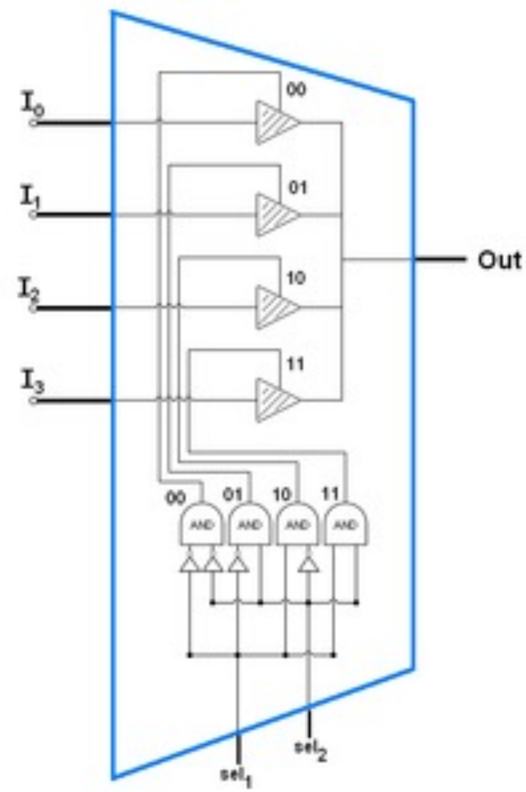
MUX

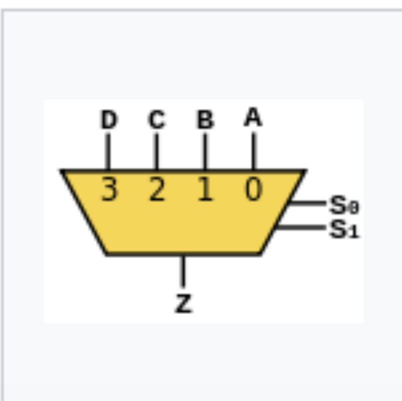


2:1

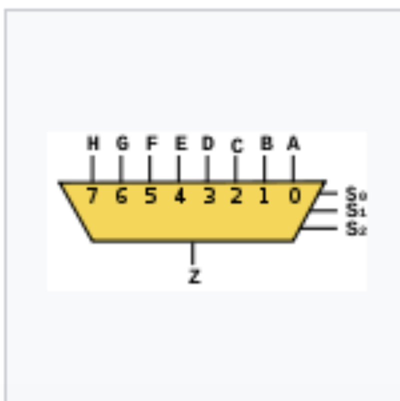


The following 4-to-1 multiplexer is constructed from 3-state buffers and AND gates

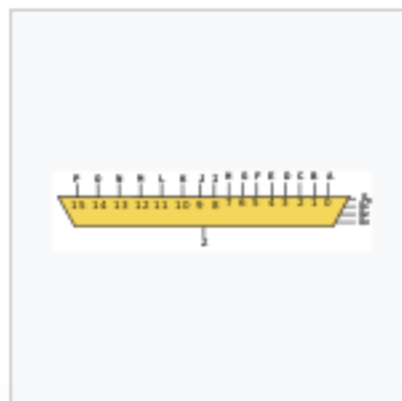




4-to-1 mux

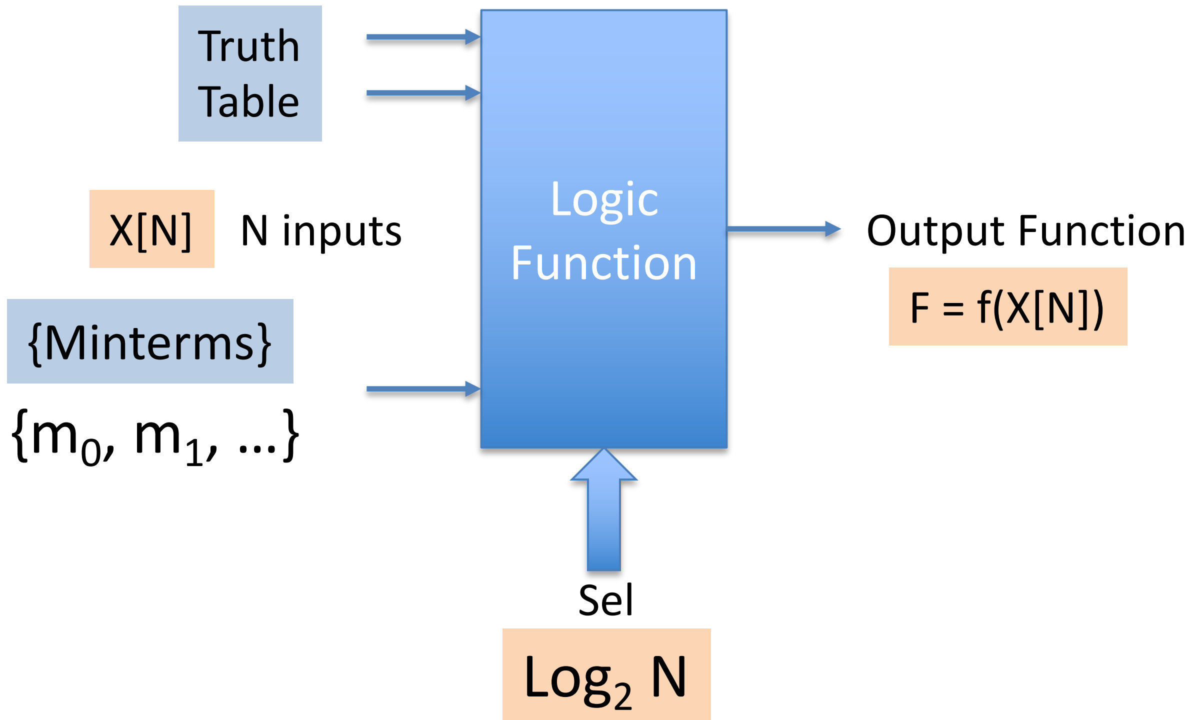


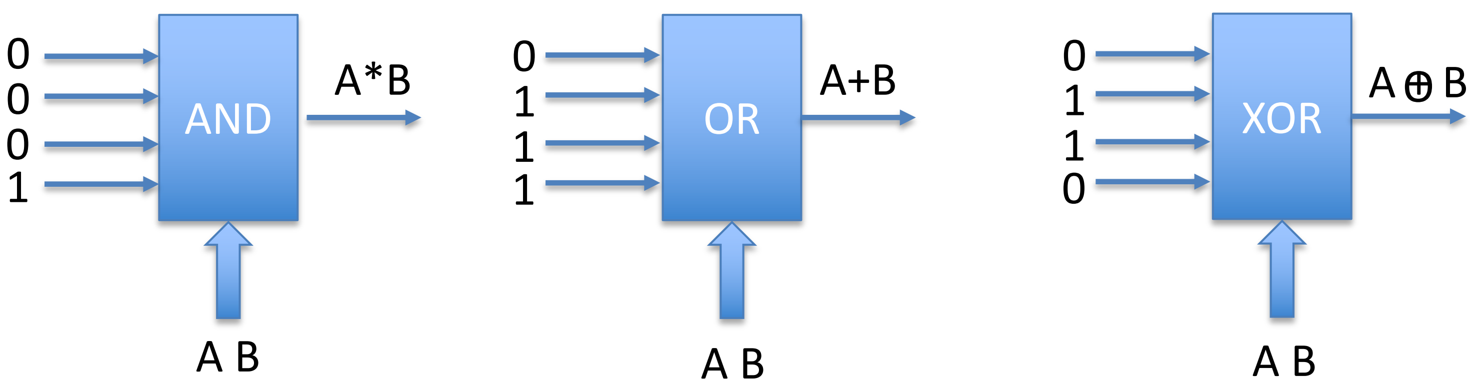
8-to-1 mux



16-to-1 mux

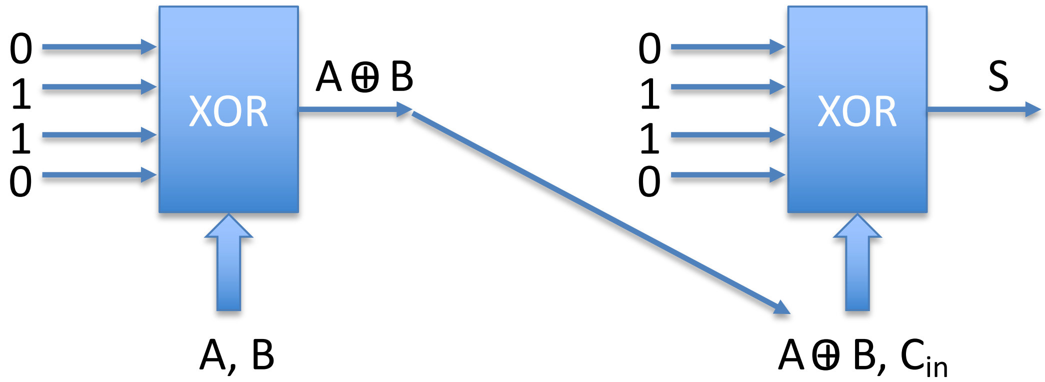
MUX as Function Generator





Full Adder via Mux

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

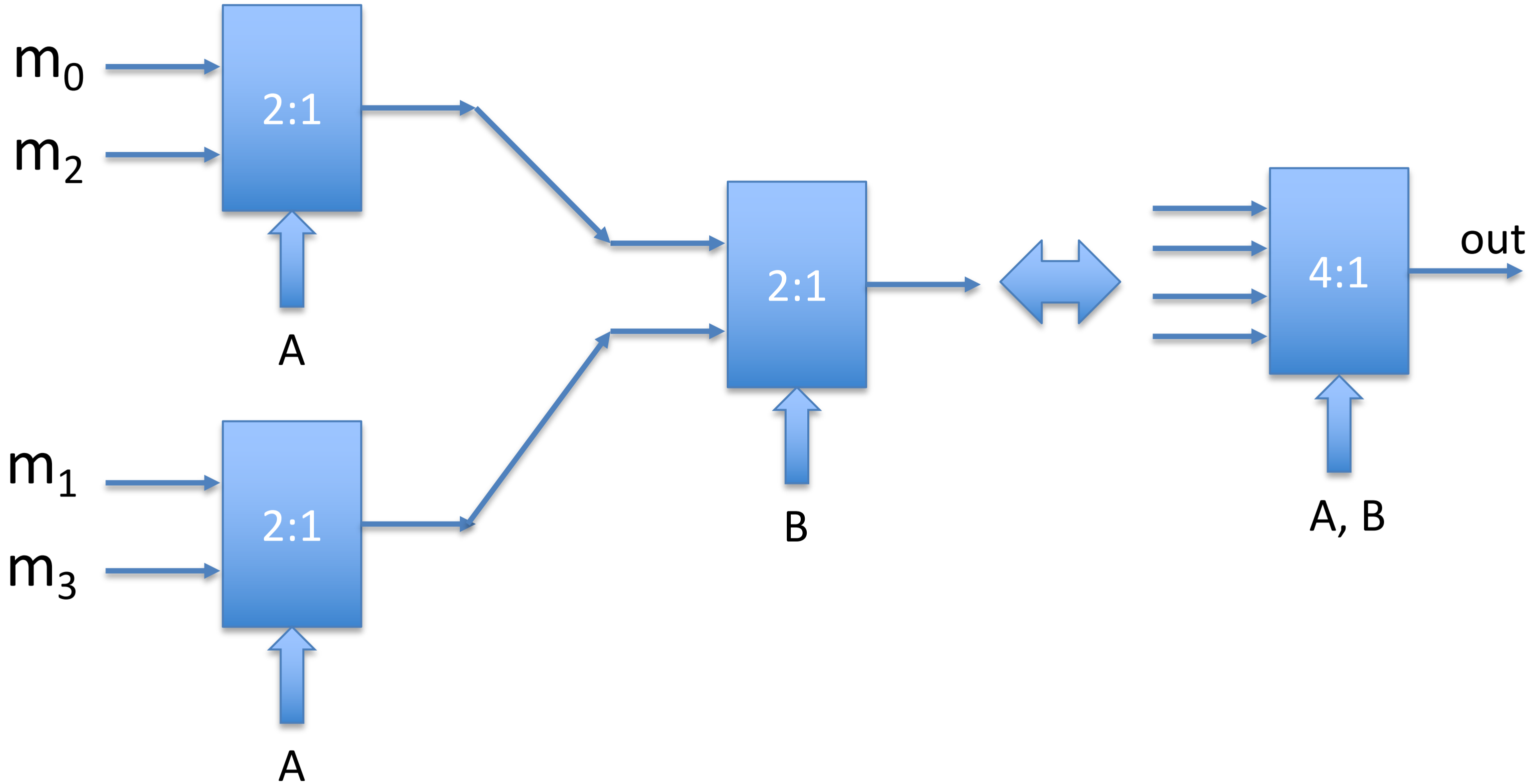


Mux Extension

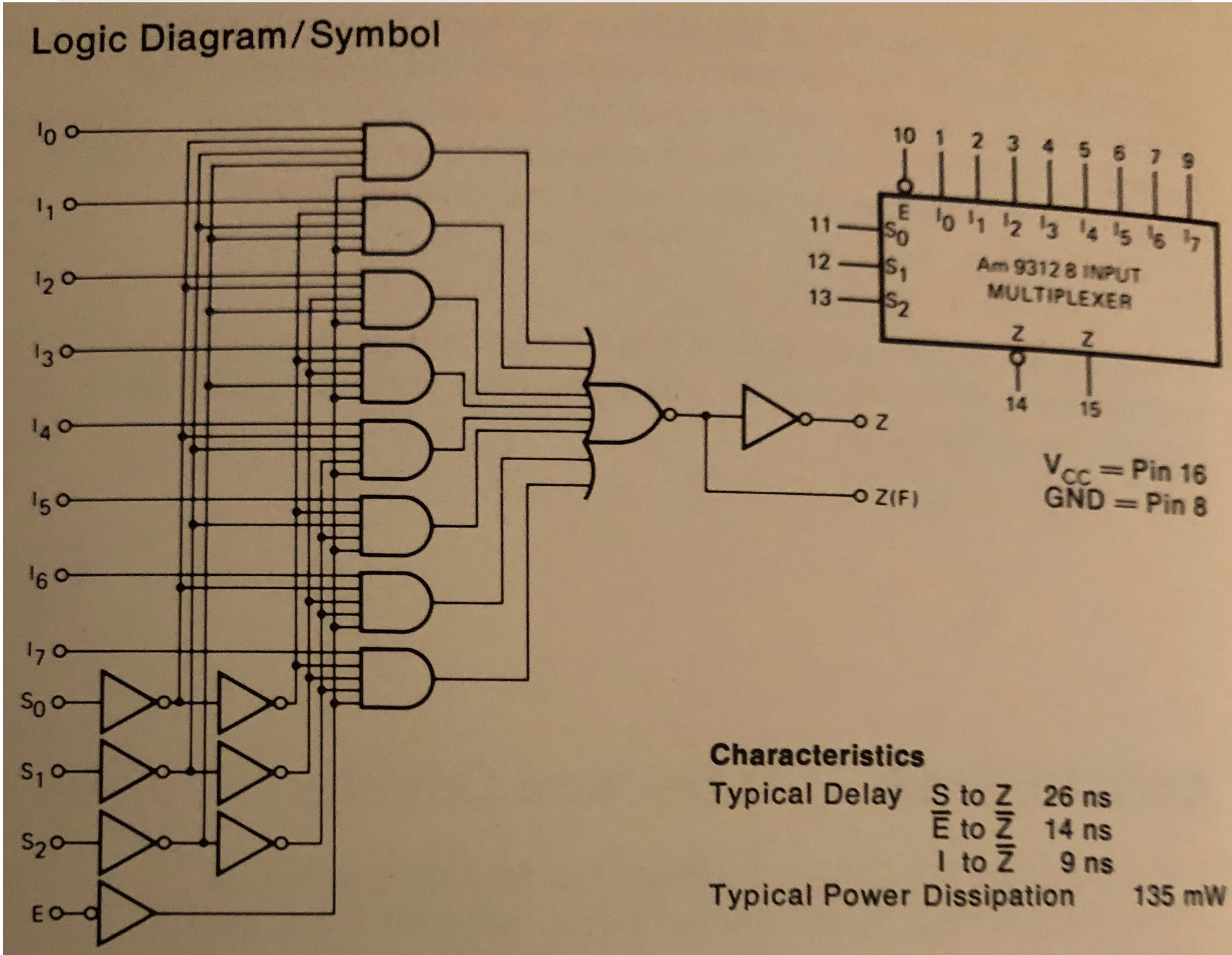
{Minterms}

2x 2:1 mux → 4:1 mux

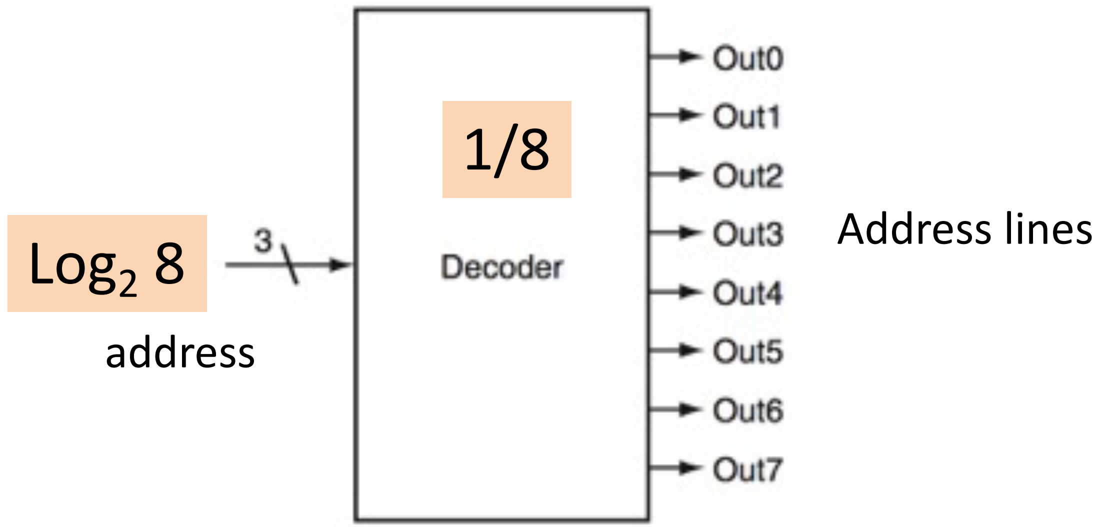
$\{m_0, m_1, \ldots\}$



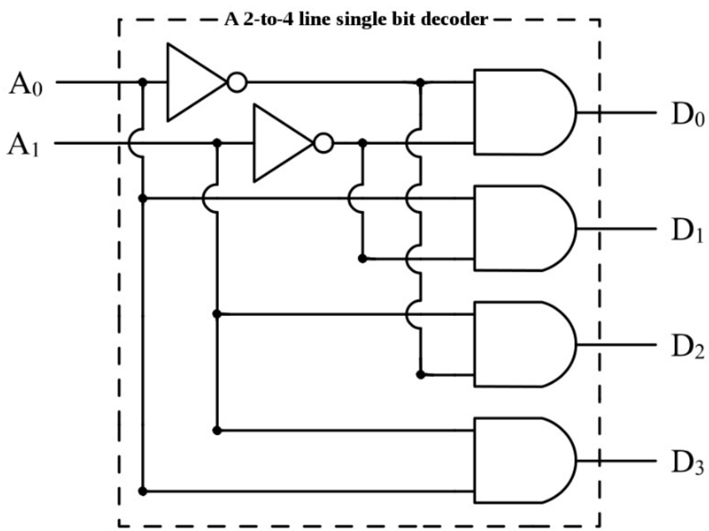
AMD 8:1 Mux


Logic Diagram/Symbol
I0
I1
I2
I3
I4
I5
I6
I7
S0
S1
S2
E
Z
Z(F)
10 1 2 3 4 5 6 7 9
11
12
13
E
S0
S1
S2
I0 I1 I2 I3 I4 I5 I6 I7
Am 9312 8 INPUT MULTIPLEXER
Z
Z
14
15
VCC = Pin 16
GND = Pin 8
Characteristics
Typical Delay S to Z 26 ns
E to Z 14 ns
I to Z 9 ns
Typical Power Dissipation 135 mW

Decoder



a. A 3-bit decoder



Truth Table

A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Minterm Equations

$$D_0 = \overline{A_1} \cdot \overline{A_0}$$

$$D_1 = \overline{A_1} \cdot A_0$$

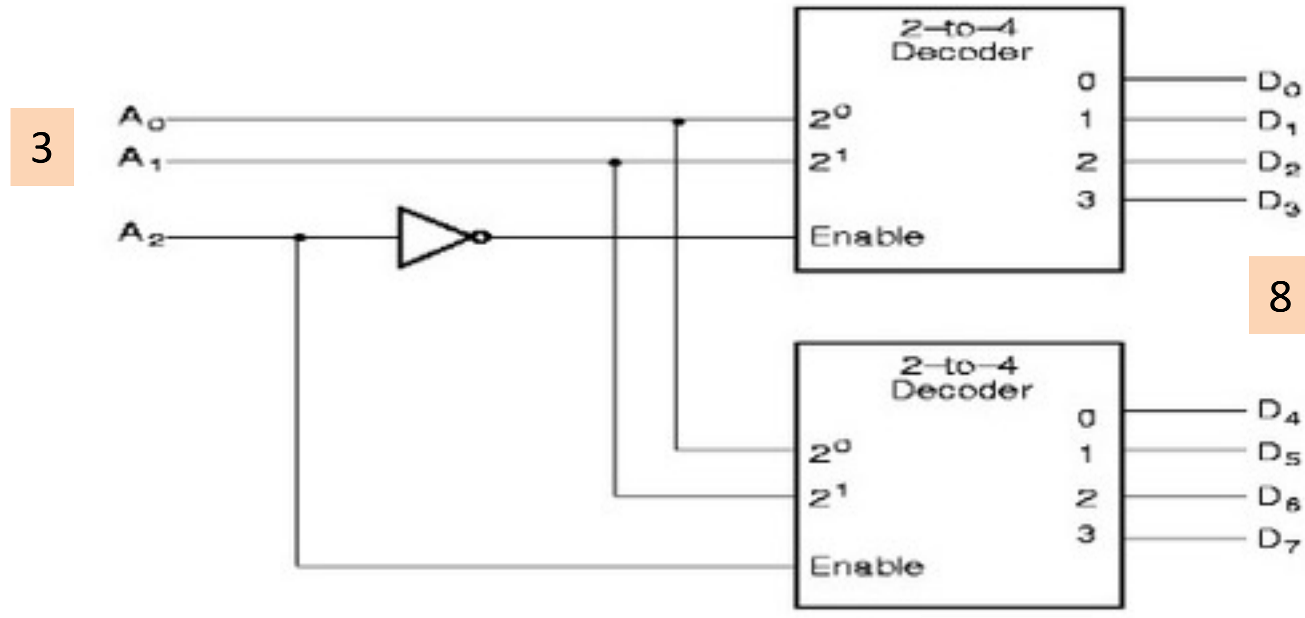
$$D_2 = A_1 \cdot \overline{A_0}$$

$$D_3 = A_1 \cdot A_0$$

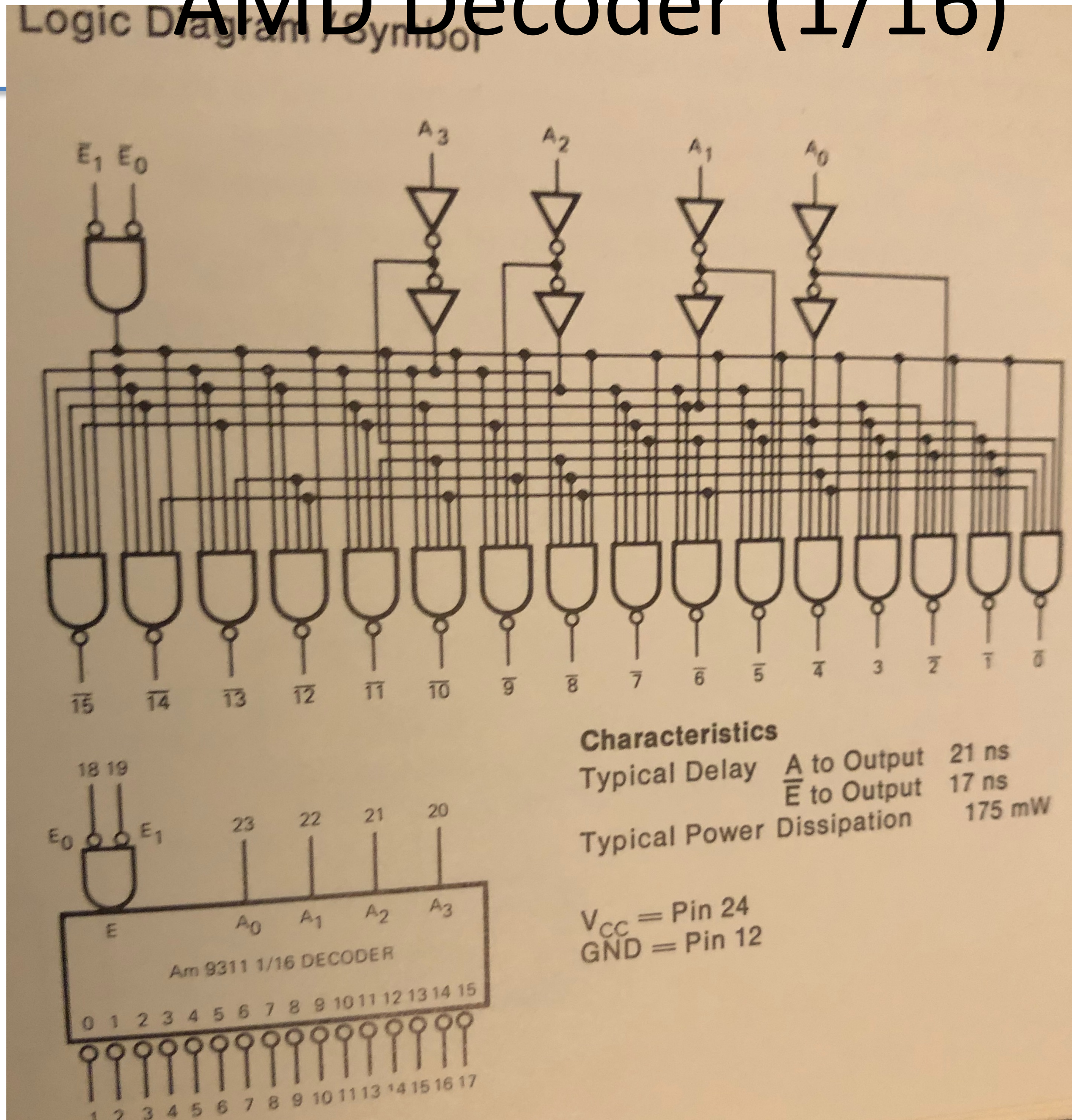
Decoder Expansion

DECODER EXPANSION

2x 2-to-4 → 3-to-8 (1 of 8)



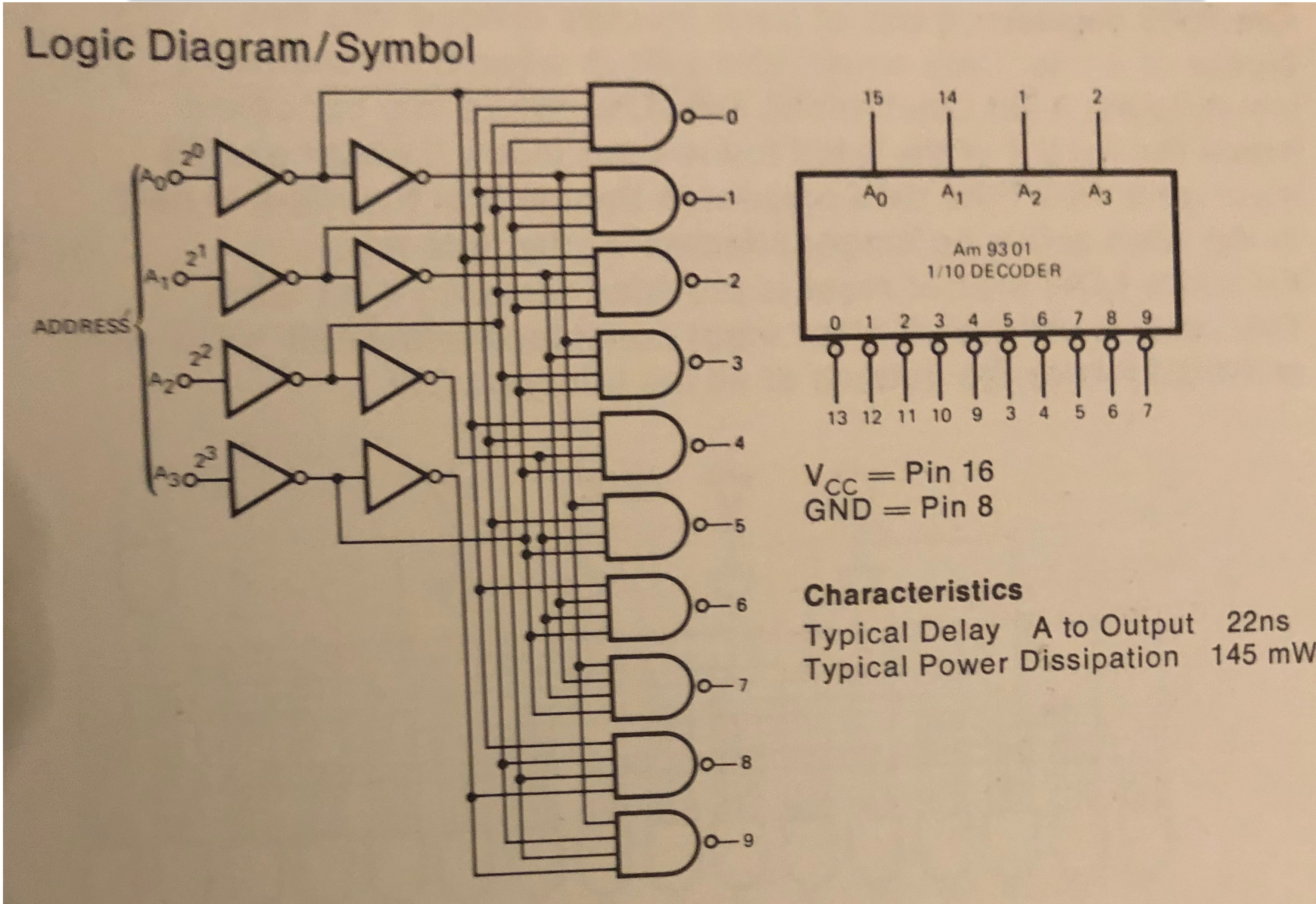
AMD Decoder (1/16)



CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

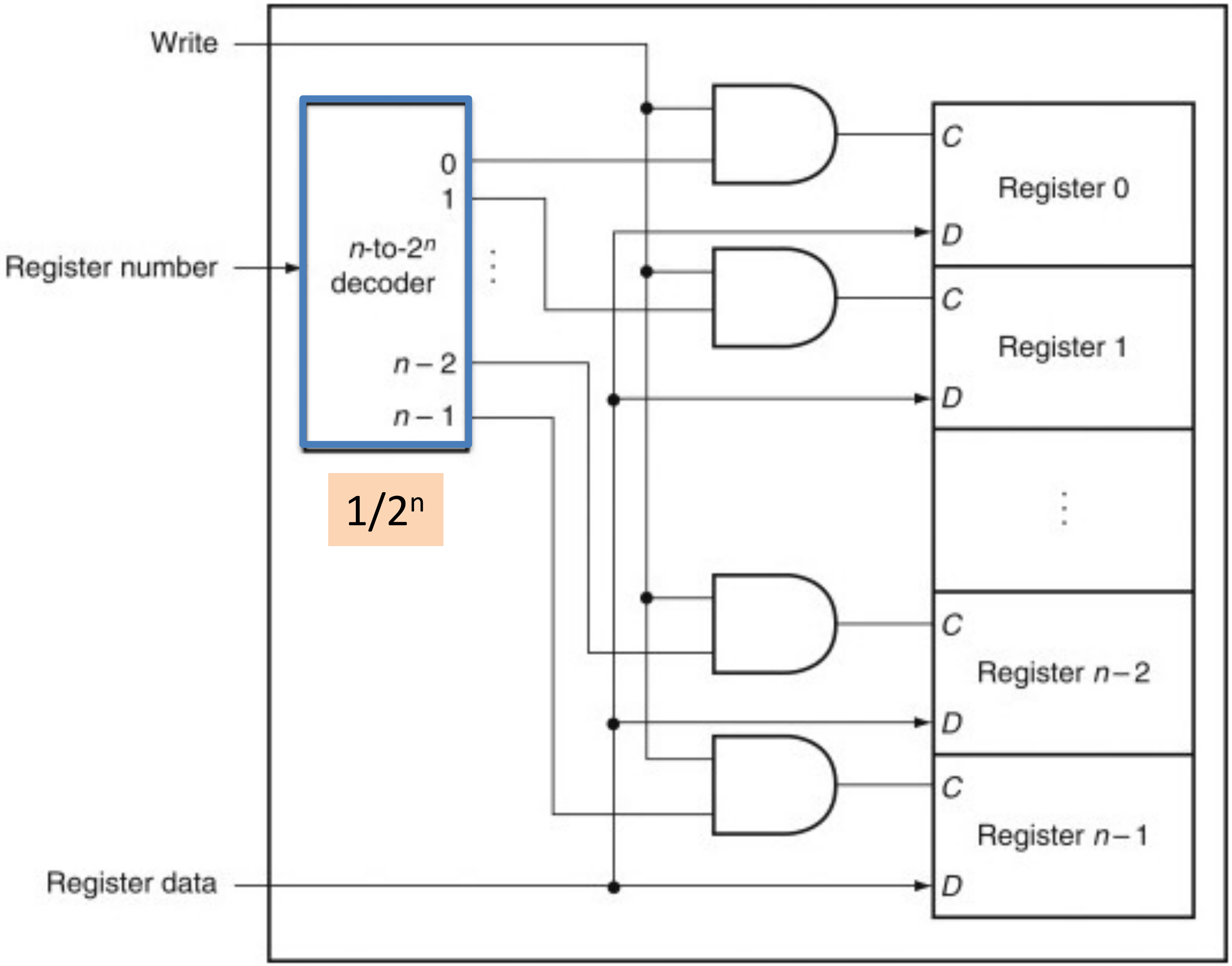
AMD Decoder (1/10)

DR JEFF SOFTWARE
INDIE APP DEVELOPER



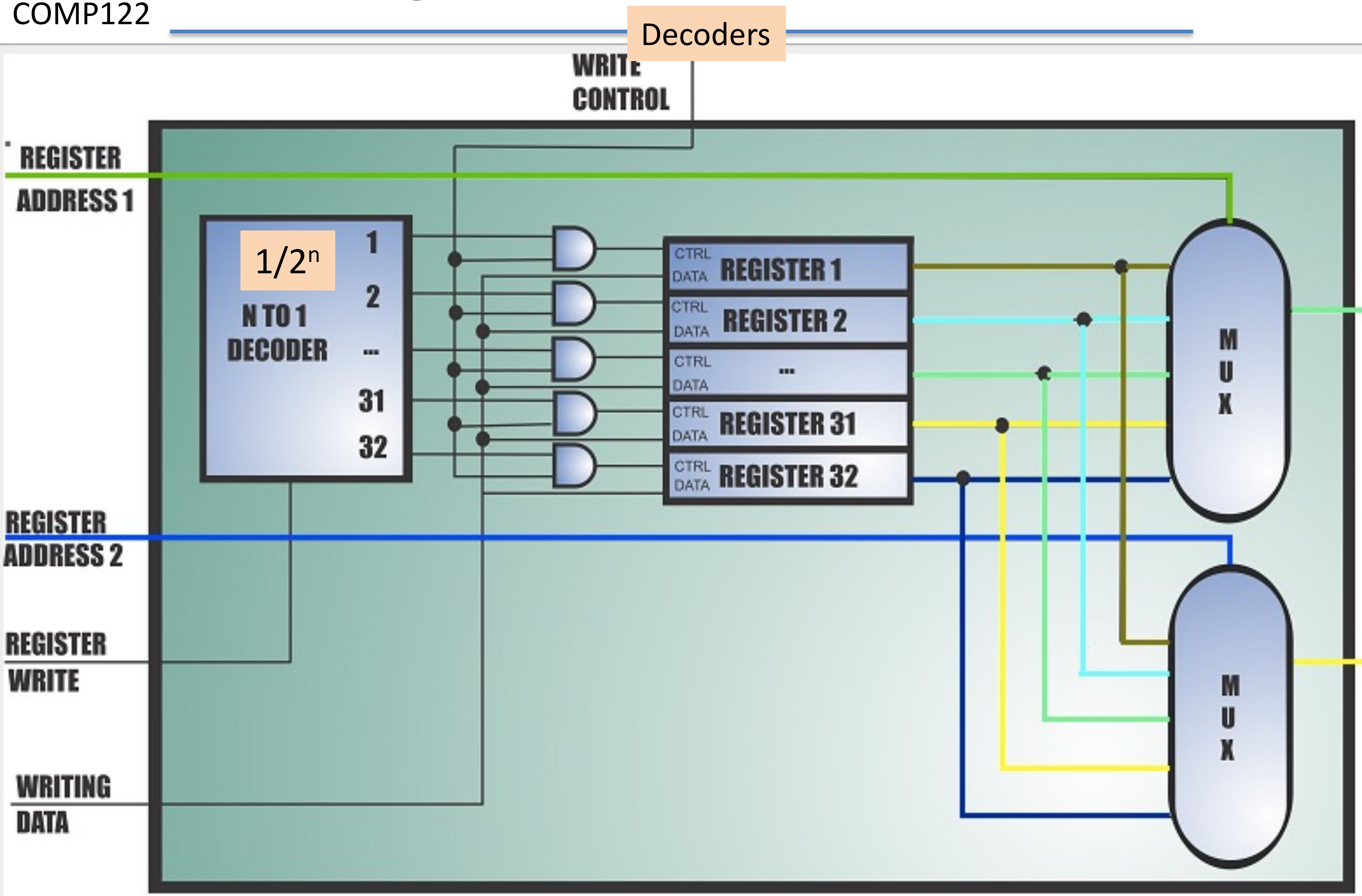
Register File Input Side

Decoders

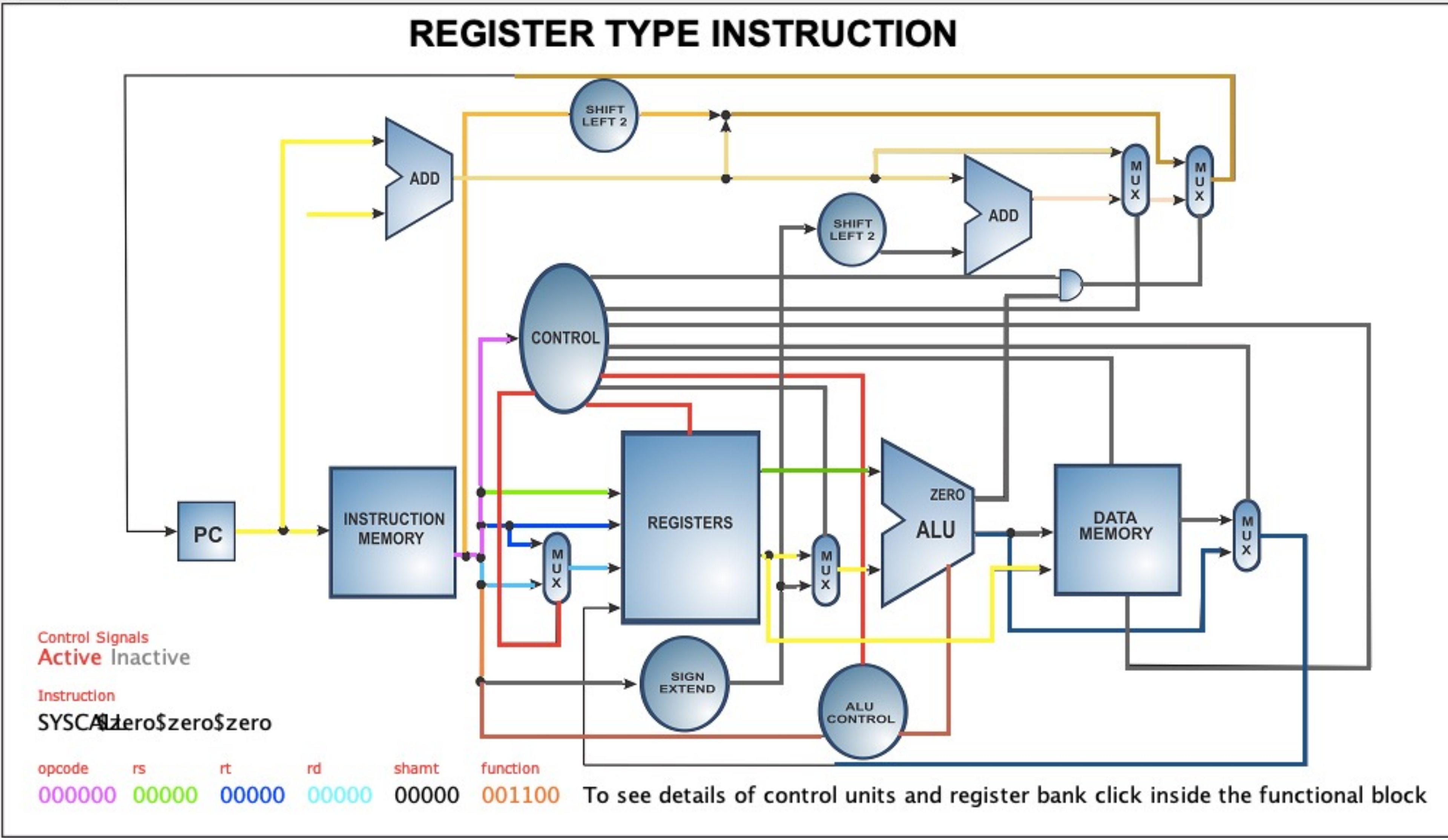


$1/2^n$

Register File on MARS



MIPS on MARS


REGISTER TYPE INSTRUCTION
SHIFT LEFT 2
ADD
MUX
SHIFT LEFT 2
ADD
MUX
CONTROL
PC
INSTRUCTION MEMORY
REGISTERS
MUX
MUX
ZERO
ALU
DATA MEMORY
MUX
SIGN EXTEND
ALU CONTROL
Control Signals
Active Inactive
Instruction
SYSCALL $zero$zero$zero
opcode rs rt rd shamt function
000000 00000 00000 00000 00000 001100
To see details of control units and register bank click inside the functional block

Section

Logic Minimization

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

Vijay Mankar · Follow
Head of Department (Electronics) at Government Polytechnic Nagpur

Quora

Encoder Example

Related **How do I make a converter (from excess 3 code to 8-4-2-1bcd code) using only 2-to -1 MUX and not gate?**

The truth table for excess-3 code to binary code converter is given below. Being 0000, 0001, 0010 and 1101, 1110, 1111 as invalid excess-3 code, output is made don't care.

TRUTH TABLE

X_3	X_2	X_1	X_0	B_3	B_2	B_1	B_0	
0	0	0	0	X	X	X	X	INVALID Excess-3 Code
0	0	0	1	X	X	X	X	
0	0	1	0	X	X	X	X	
0	0	1	1	0	0	0	0	
0	1	0	0	0	0	0	1	
0	1	0	1	0	0	1	0	
0	1	1	0	0	0	1	1	
0	1	1	1	0	1	0	0	
1	0	0	0	0	1	0	1	
1	0	0	1	0	1	1	0	
1	0	1	0	0	1	1	1	
1	0	1	1	1	0	0	0	
1	1	0	0	1	0	0	1	
1	1	0	1	X	X	X	X	INVALID Excess-3 code
1	1	1	0	X	X	X	X	
1	1	1	1	X	X	X	X	

$B_0 = \{m4,m6,m8,m10,m12\}$

$B_1 = \{m5,m6,m9,m10\}$

$B_2 = \{m7,m8,m9,m10\}$

$B_3 = \{m11,m12\}$

Logic Function Minimization

Combinational

- ❖ Quine-McKluskey
  - ❑ *Prime implicants*
  - ❑ *Essential* PI's
- ❖ Karnaugh ("K") Maps

Sum of Products: ***minterms***

General form $F(x, y) = \{m_0, m_1, m_2, m_3\}$

		y
x	m_0	m_2
	m_1	m_3

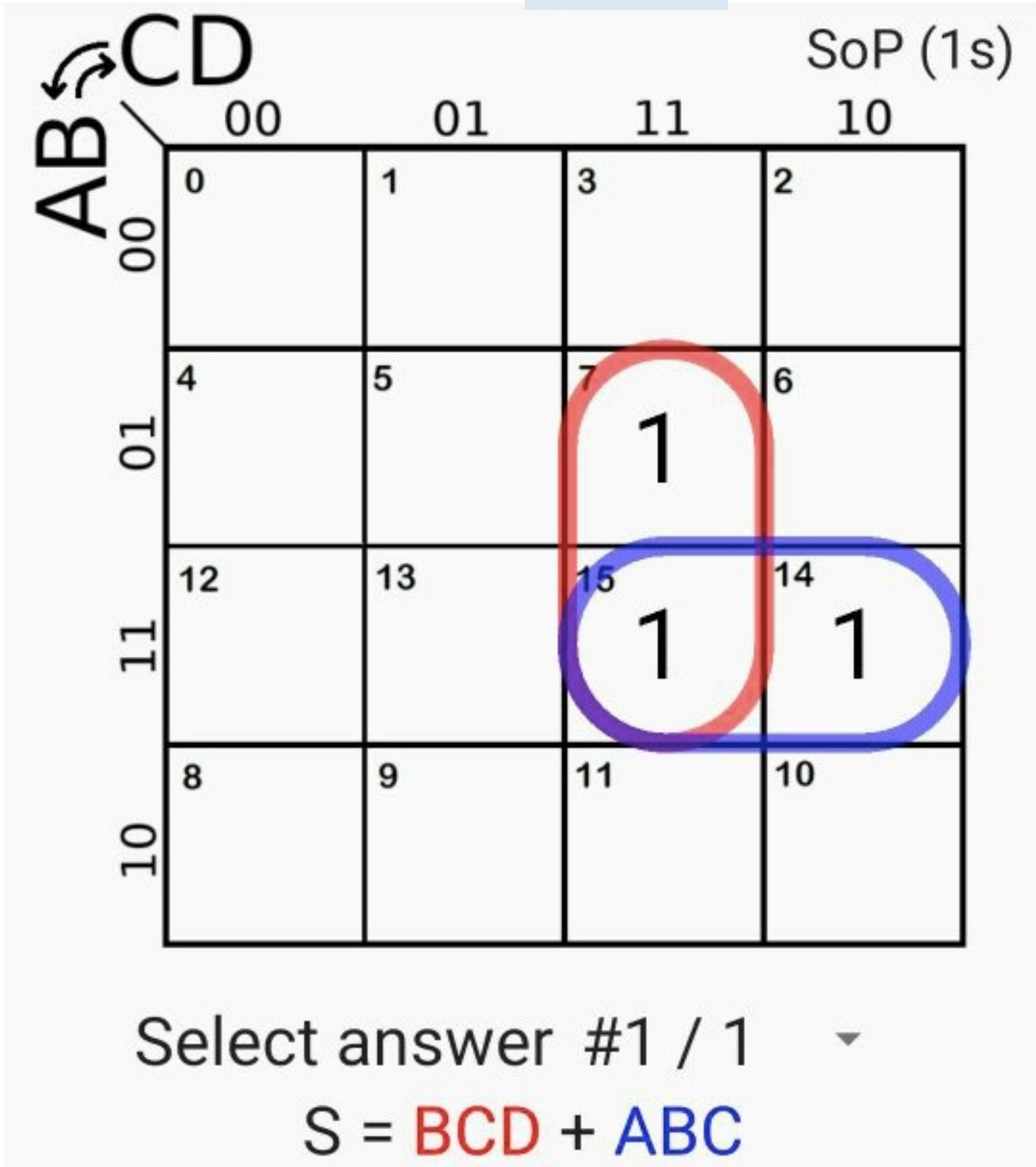
→

		y
x	1	0
	1	0

Example $F(x, y) = \{m_0, m_1\} = x'y' + xy' \rightarrow y'$

K-Maps

Example



Gray codes

Logic distance=1

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

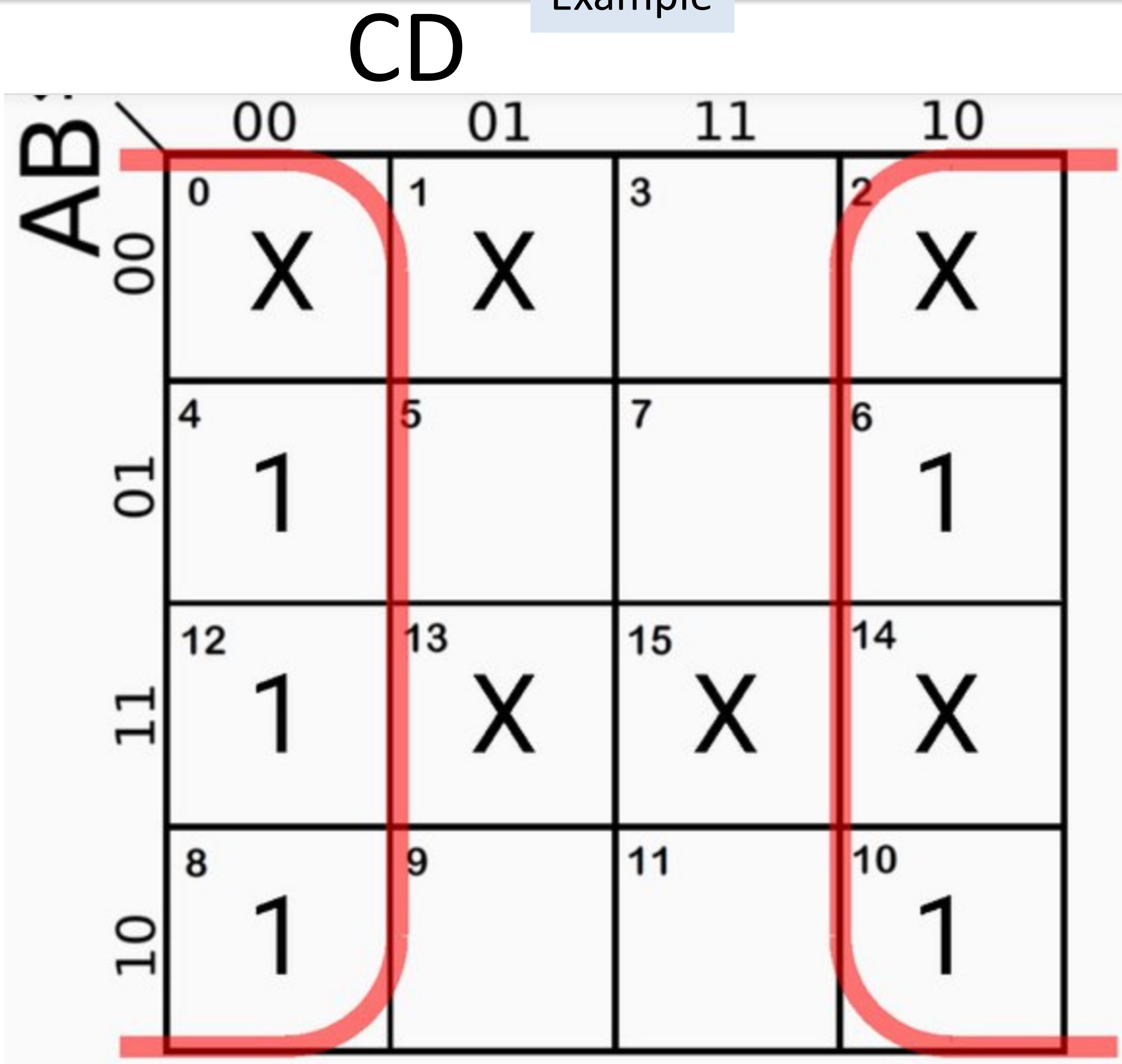
Vijay Mankar · Follow
Head of Department (Electronics) at Government Polytechnic Nagpur

Quora

K-Maps

Example

DR JEFF SOFTWARE
INDIE APP DEVELOPER



$$S = \bar{D}$$

Gray codes

Logic distance=1

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

Quora

Vijay Mankar · Follow

Head of Department (Electronics) at Government Polytechnic Nagpur

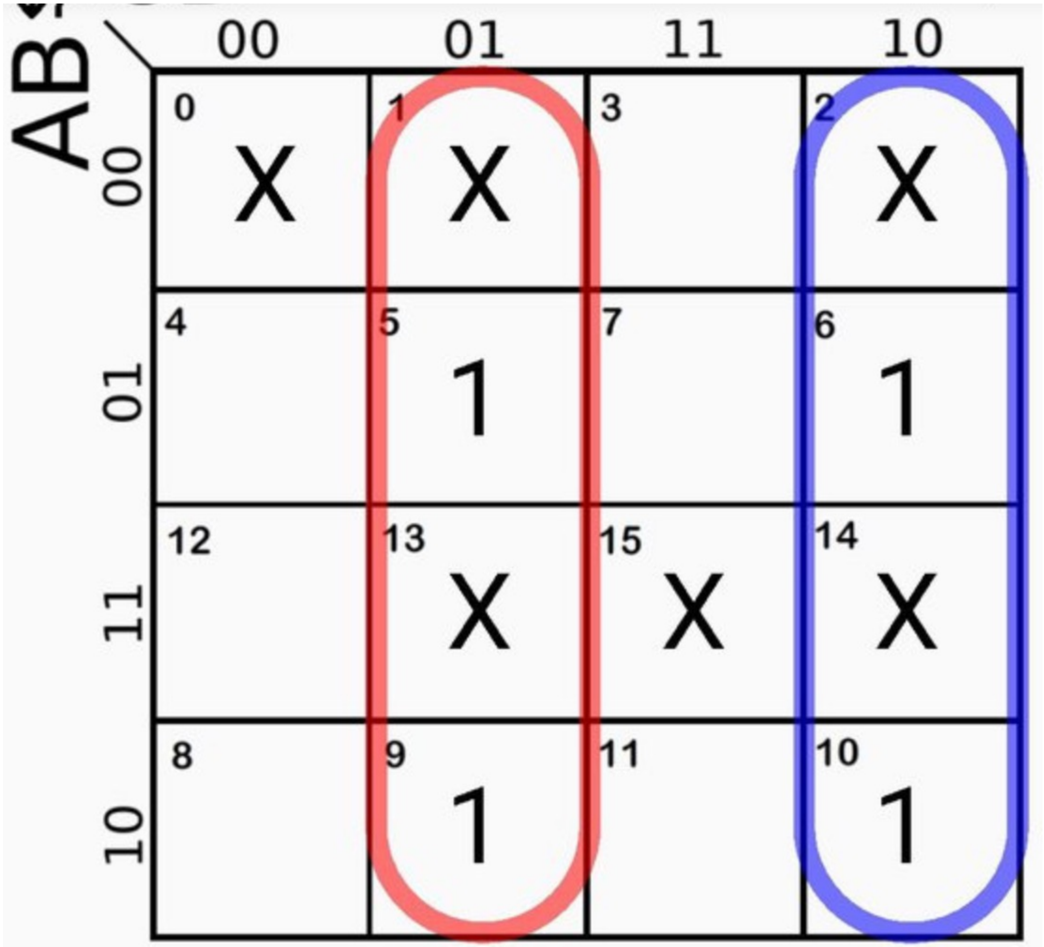
DR JEFF SOFTWARE

INDIE APP DEVELOPER

K-Maps

Example

CD



$$S = \bar{C}D + C\bar{D}$$

Gray codes

Logic distance=1

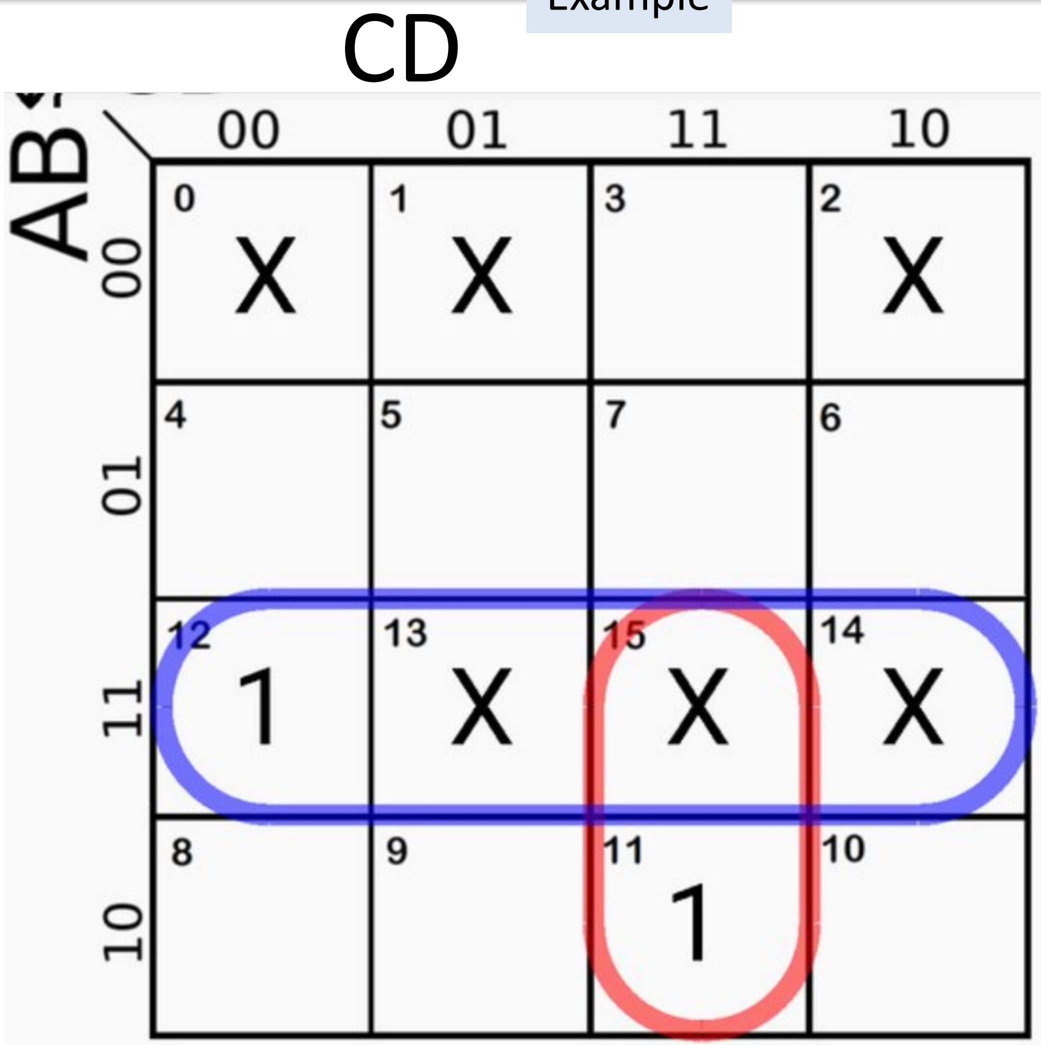
COMP122

Quora

Vijay Mankar · Follow
Head of Department (Electronics) at Government Polytechnic Nagpur

K-Maps

Example



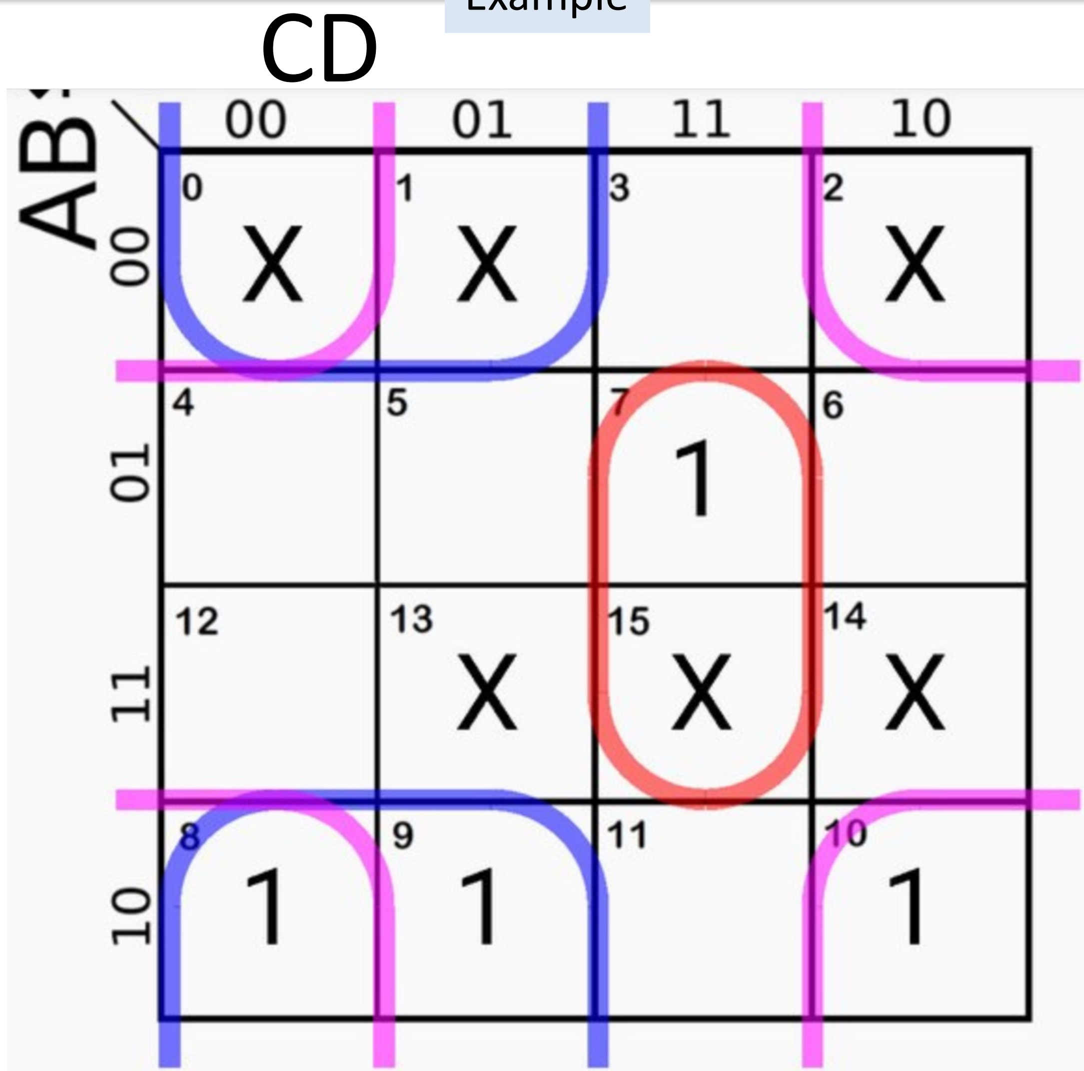
$S = ACD + AB$

Gray codes

Logic distance=1

K-Maps

Example



$$S = BCD + \bar{B}\bar{C} + \bar{B}\bar{D}$$

Gray codes

Logic distance=1

Section

Memory

- SRAM
- DRAM

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

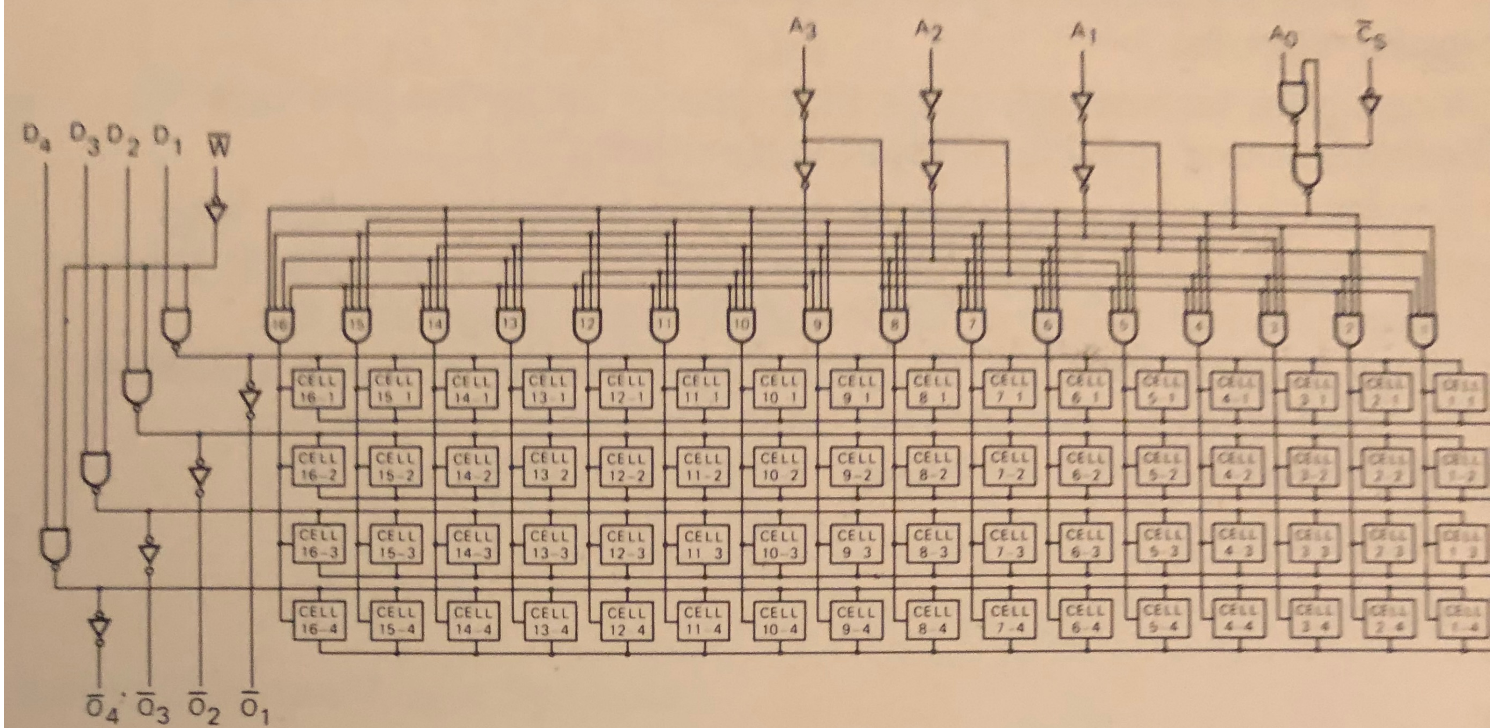
AMD 64-Bit Bipolar SRAM

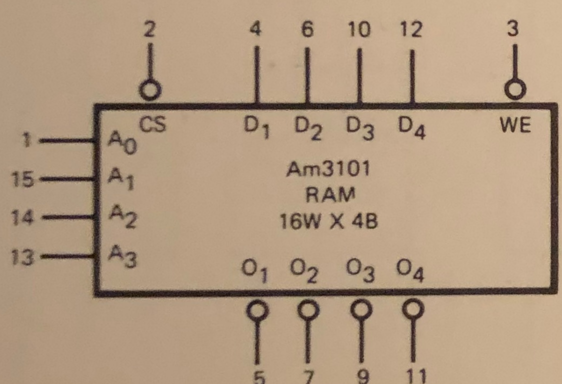
DR JEFF SOFTWARE
INDIE APP DEVELOPER

Am3101

1971

Logic Diagram/Symbol





Characteristics 3101

Typical Delay Access Time 35 ns

Typical Power Dissipation 400 mW

V_{CC} = Pin 16

GND = Pin 8

AMD MOS LSI

1971

MOS/LSI

MOS is a technology of today as well as tomorrow. Various MOS technologies have been developed, but we feel silicon gate to be the most promising. Silicon gate will be the primary technology used for memory – application MOS products from Advanced Micro Devices. The basis for this decision is:

1. It is directly TTL and DTL compatible
2. It has greater speed than conventional metal-gate MOS
3. It is more reliable
4. Its reproducibility is higher
5. It is lower cost on a per-function basis

Our commitment in MOS is to produce large-chip standard circuits. The circuits are to have a broad customer base, and be available in full military temperature range (−55°C to +125°C) as well as the commercial temperature range (0°C to +75°C). Future plans call for the following additions to our product line in 1971:

256-Bit Static Random Access Memory

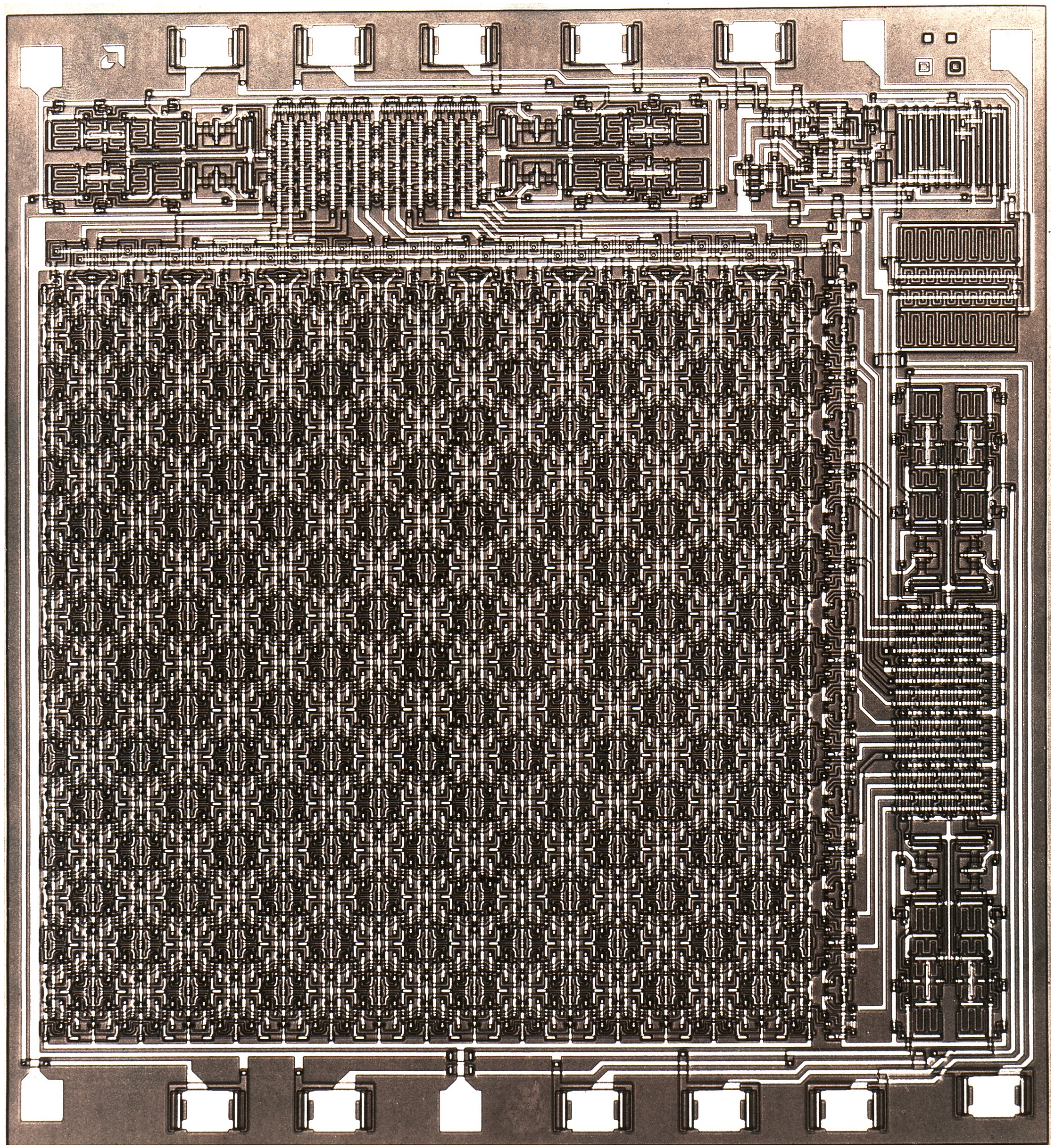
1024-Bit Dynamic Two-Phase Shift Register
(1024x1, 512x2, 256x4)

1024-Bit Dynamic Random Access Memory

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Am1101A 256x1 SRAM

DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023

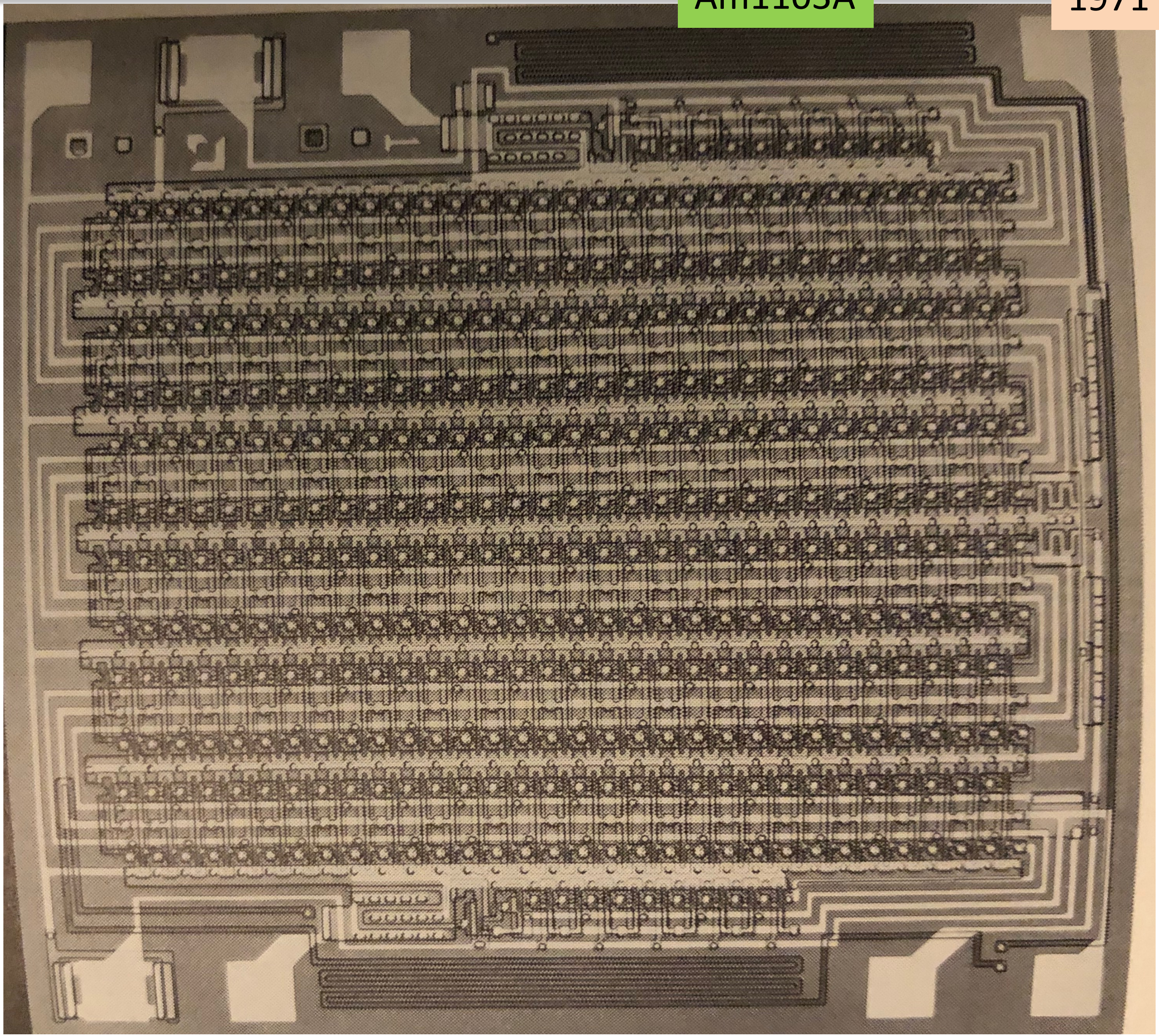


CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

DRAM (AMD 1Kx1)

Am1103A

1971



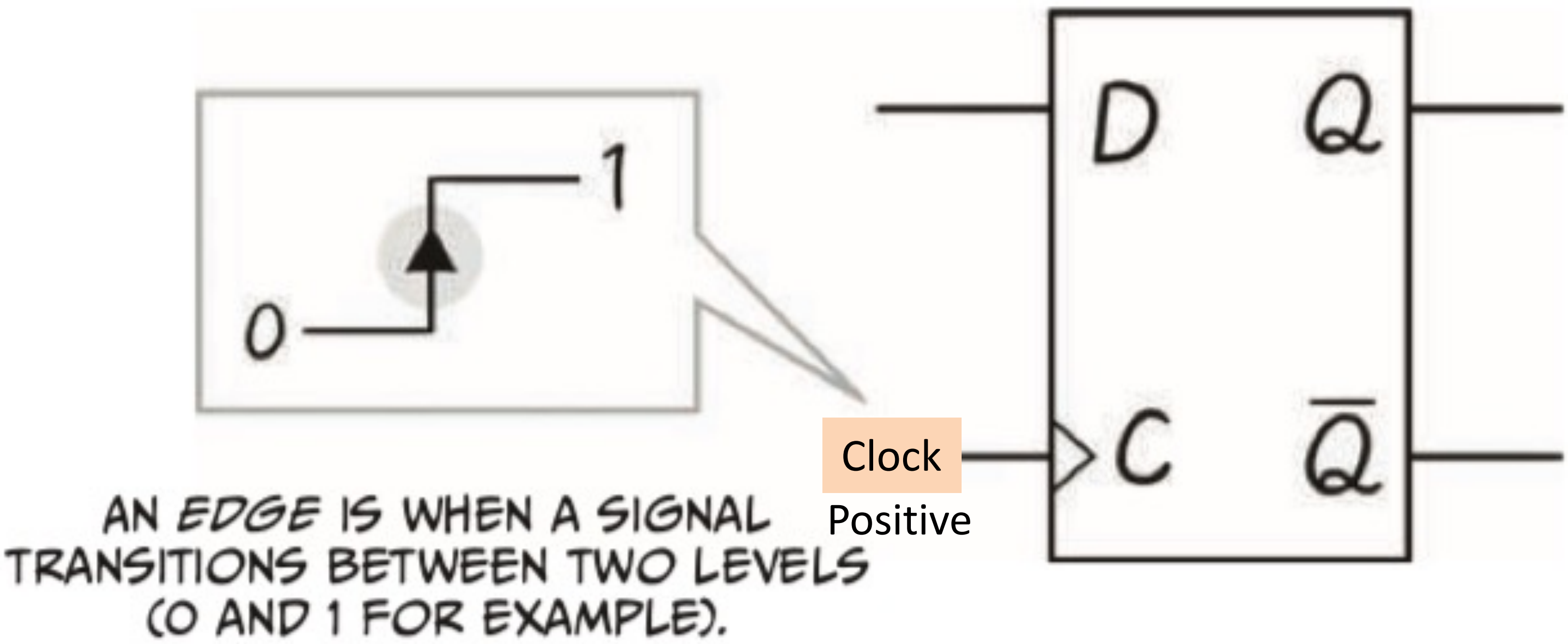
Section

Logic

- Sequential
  - Flip-flops
  - Latches
  - Counters

Sequential Logic: Flip-Flops

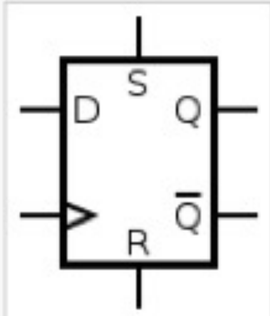
Clocked



Sequential Logic: Flip-Flops

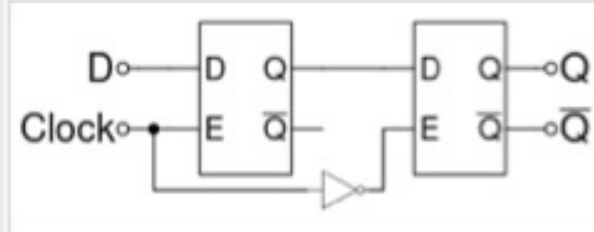
[from Wikipedia]

Clock	D	Q_{next}
Rising edge	0	0
Rising edge	1	1
Non-Rising	X	Q

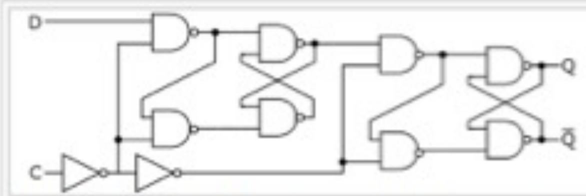


D flip-flop symbol

"D" FF

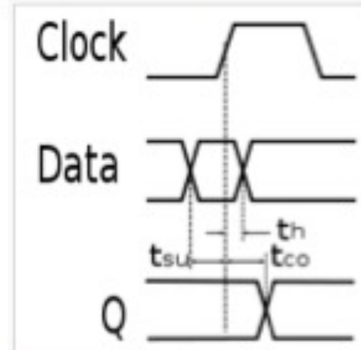


A master–slave D flip-flop. It responds on the falling edge of the *enable* input (usually a clock)



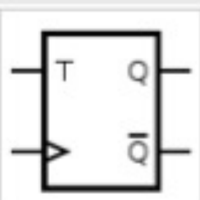
An implementation of a master–slave D flip-flop that is triggered on the rising edge of the clock

Timing: setup, hold, delay

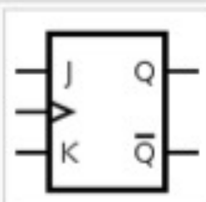


Flip-flop setup, hold and clock-to-output timing parameters

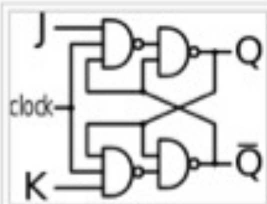
Other FFs



A circuit symbol for a T-type flip-flop

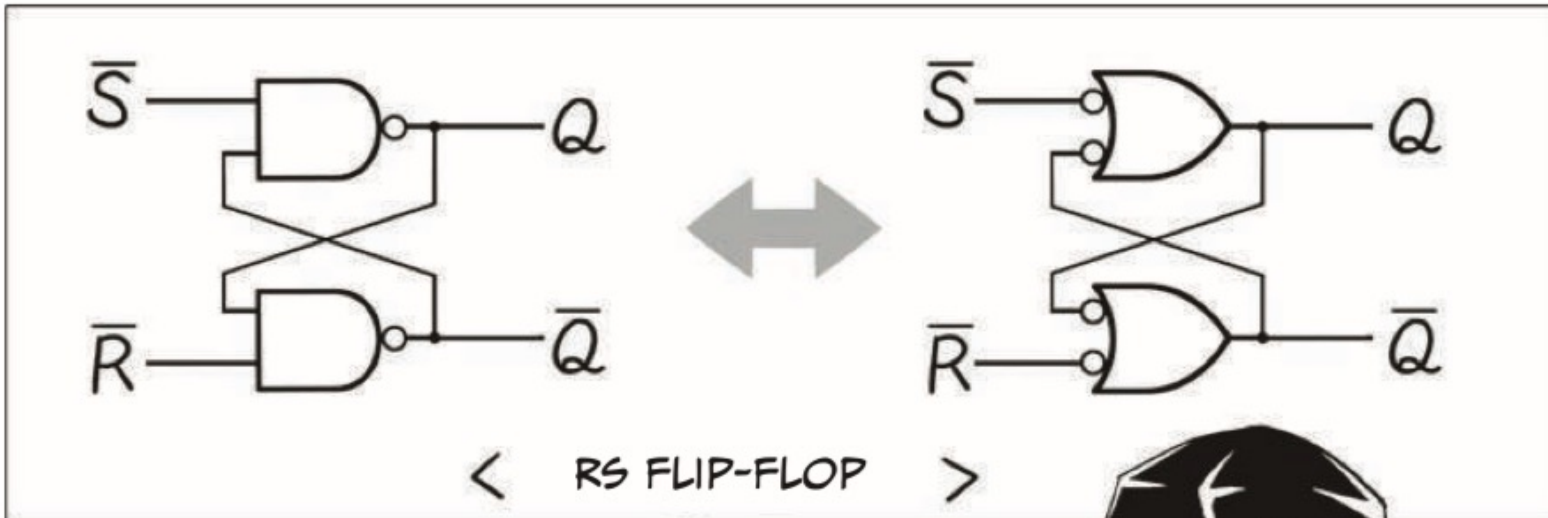


A circuit symbol for a positive-edge-triggered JK flip-flop

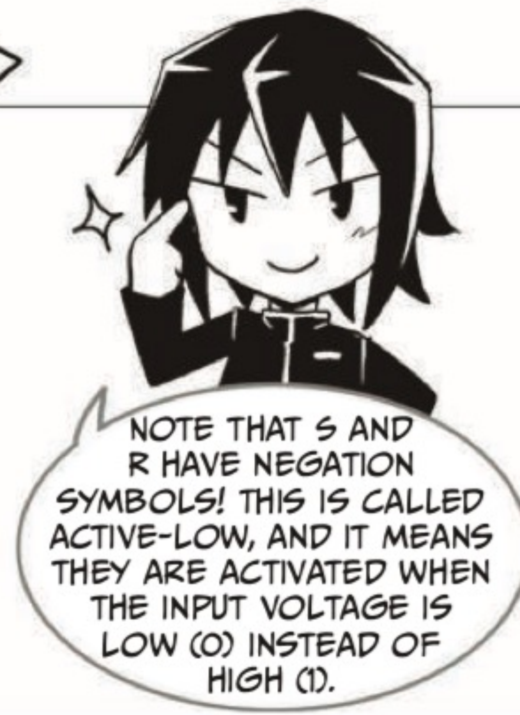


A JK flip-flop made of NAND gates

Sequential Logic: SR Latch



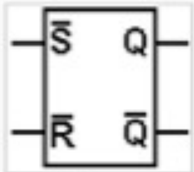
INPUTS		OUTPUTS		FUNCTION
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
1	1	DOES NOT CHANGE		RETAINS ITS CURRENT OUTPUT
0	1	1	0	SET
1	0	0	1	RESET
0	0	1	1	NOT ALLOWED



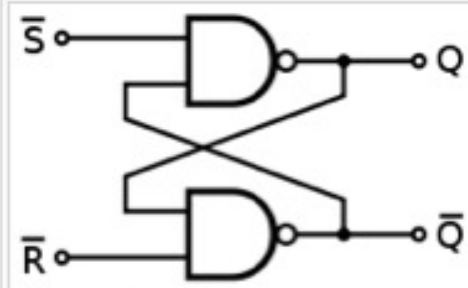
Sequential Logic: Latches

$\overline{SR}$ latch operation

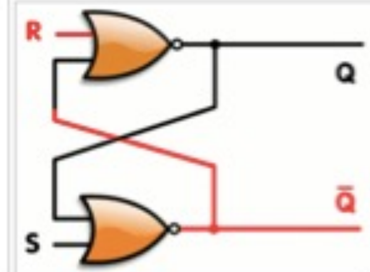
$\overline{S}$	$\overline{R}$	Action
0	0	not allowed
0	1	Q = 1
1	0	Q = 0
1	1	No Change



Symbol for an $\overline{SR}$ NAND latch



An $\overline{SR}$ latch

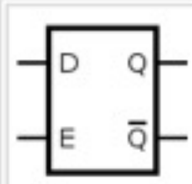


An SR latch, constructed from a pair of cross-coupled NOR gates.

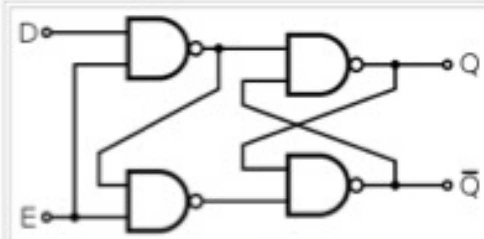
[from Wikipedia]

Gated D latch truth table

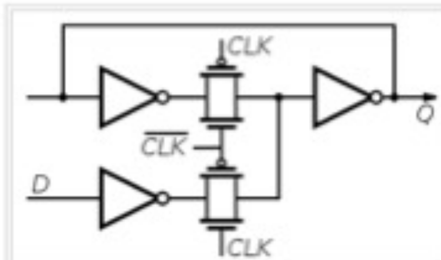
E/C	D	Q	$\overline{Q}$	Comment
0	X	Q_{prev}	$\overline{Q}_{prev}$	No change
1	0	0	1	Reset
1	1	1	0	Set



Symbol for a gated D latch



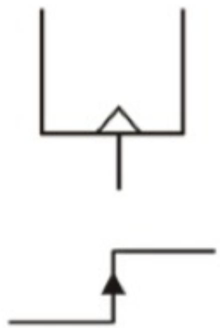
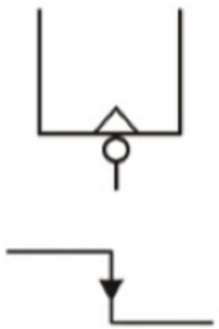
A gated D latch based on an $\overline{SR}$ NAND latch



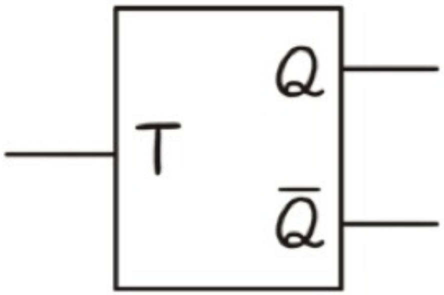
A gated D latch in pass transistor logic, similar to the ones in the CD4042 or the CD74HC75 integrated circuits.

Clocking

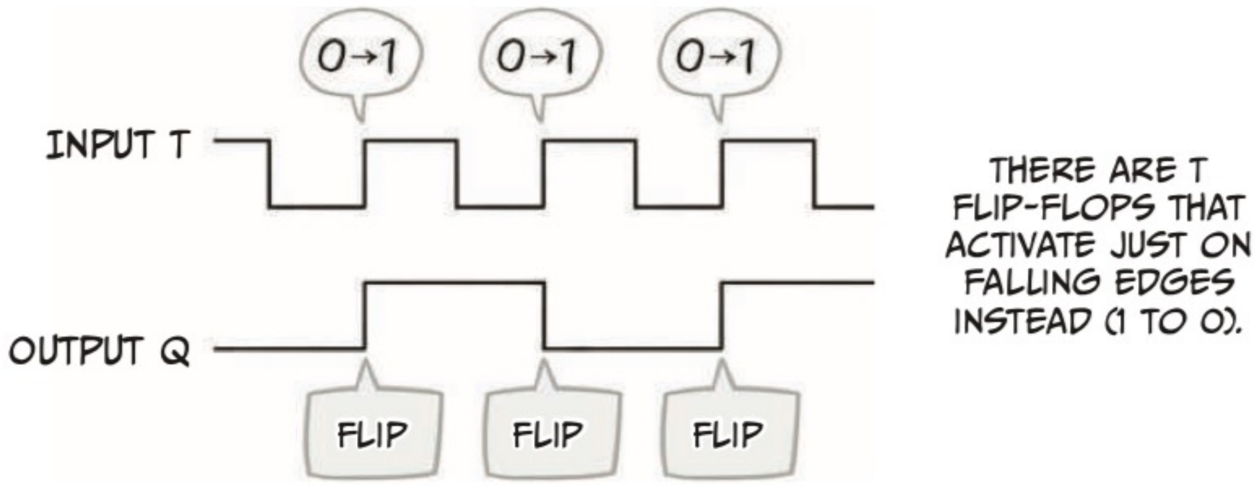
When the clock goes from low to high (0 to 1), we see a rising edge, and when it goes back from high to low (1 to 0), we see a falling edge.

RISING EDGE	FALLING EDGE
	
WHEN THE CLOCK GOES FROM LOW TO HIGH	WHEN THE CLOCK GOES FROM HIGH TO LOW

T: Toggle Flip-flop



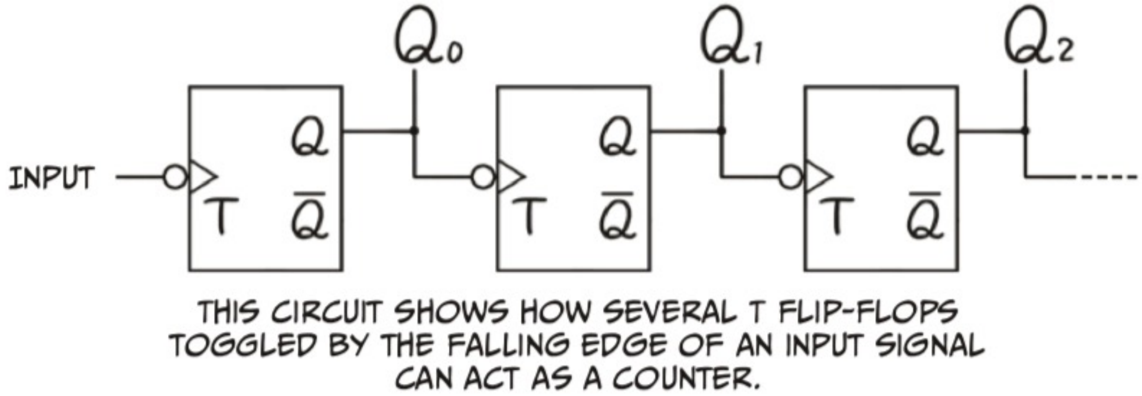
Fuhahaha! Like I would ever forget! The *T flip-flop* only has one input, as you can see, and is pretty simple. Whenever the input T changes from 0 to 1, or 1 to 0, the output stored in Q flips state. It looks something like this time chart.

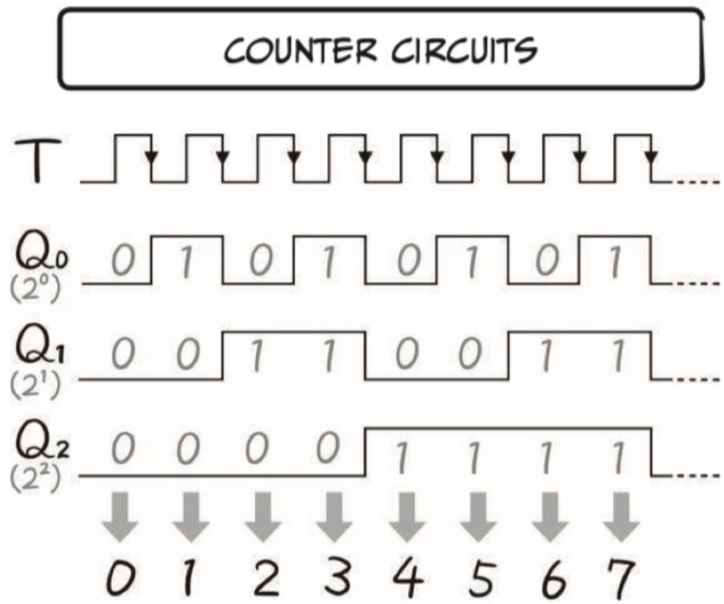


Counter

Toggle

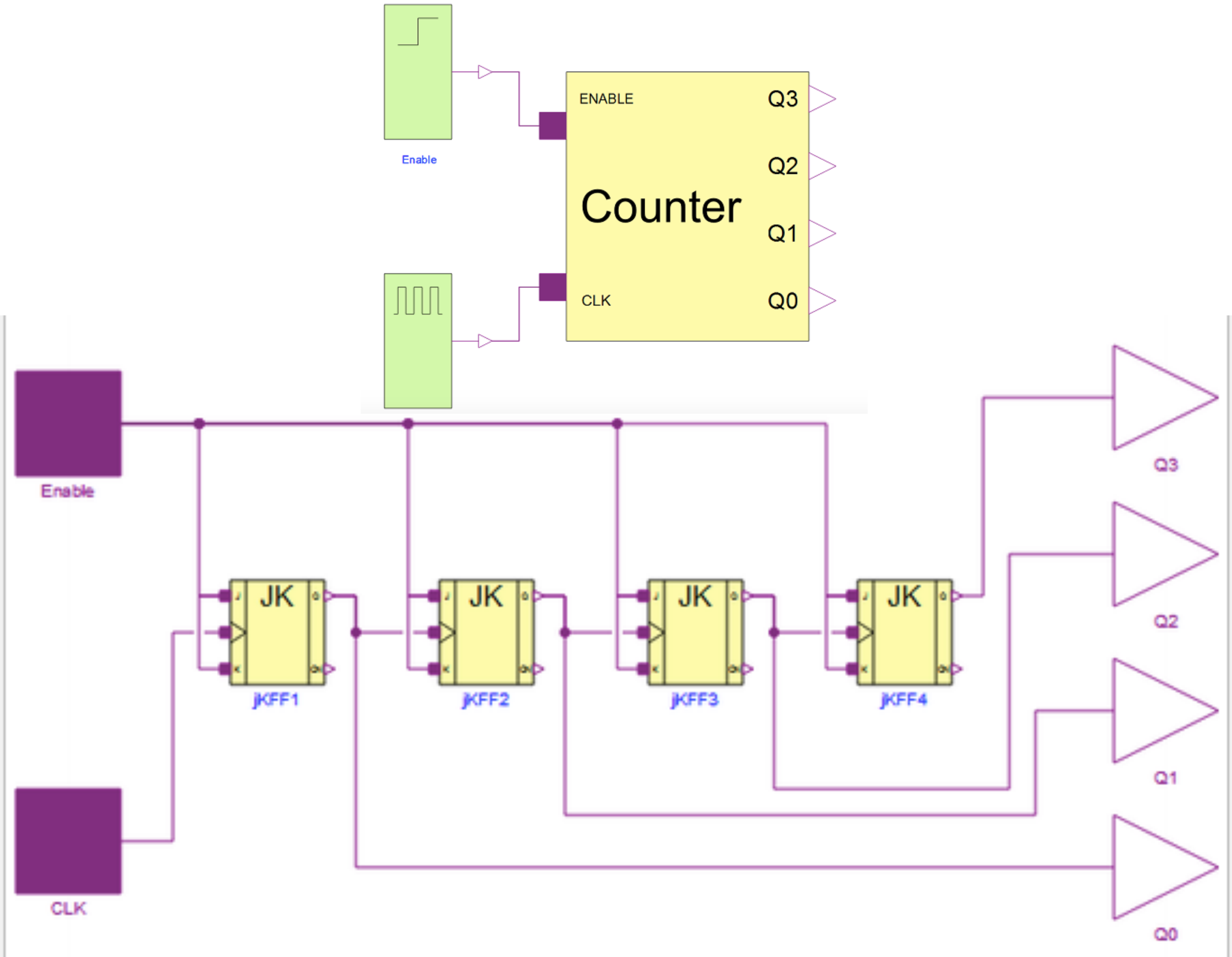
By the way, flipping between 1 and 0 is called *toggling*. The *T* in T flip-flop actually stands for *toggle*! Also, by connecting several T flip-flops together as in the schematic below, you can make a circuit that can count—a counter circuit.





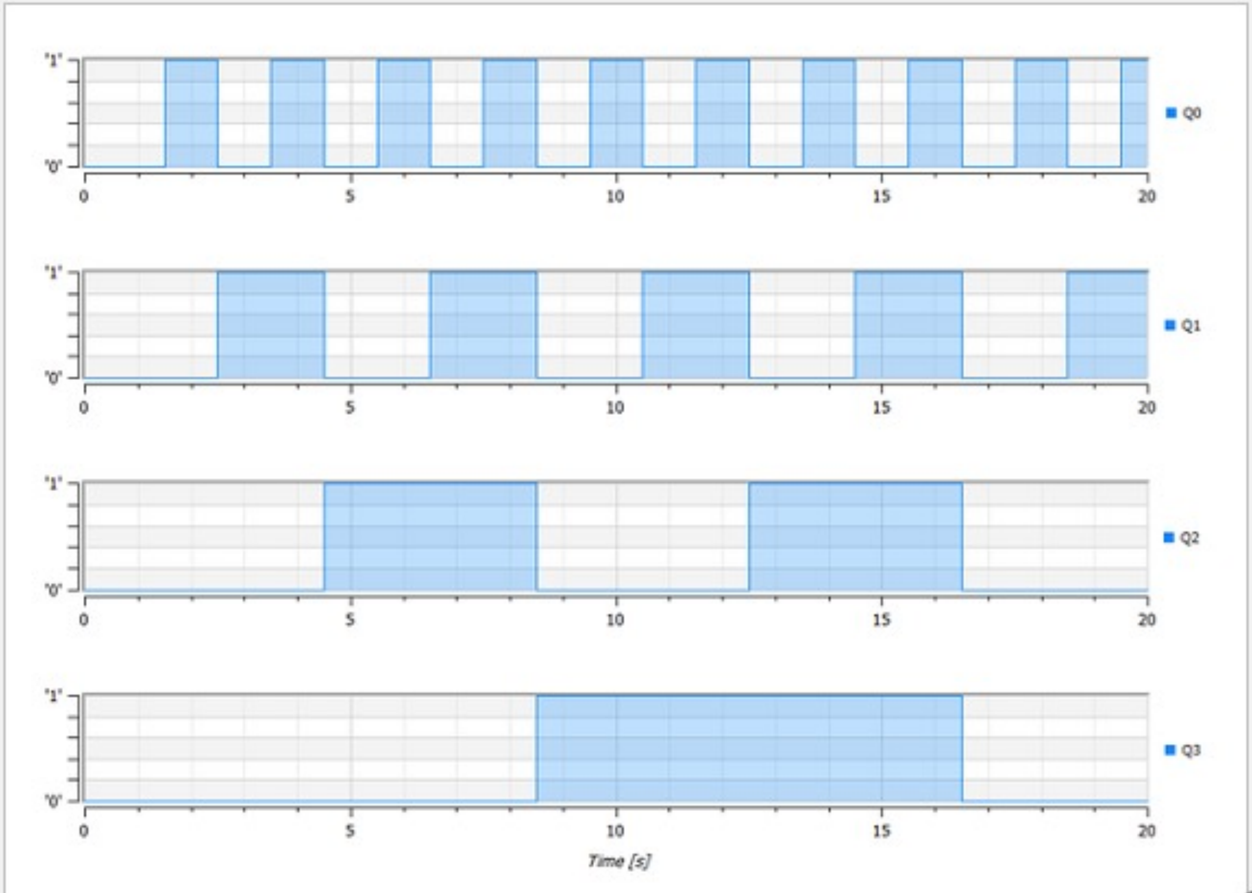
Falling edge triggered

Counter


Enable
ENABLE
Counter
CLK
Q3
Q2
Q1
Q0
Enable
JK
JK
JK
JK
jKFF1
jKFF2
jKFF3
jKFF4
CLK
Q3
Q2
Q1
Q0

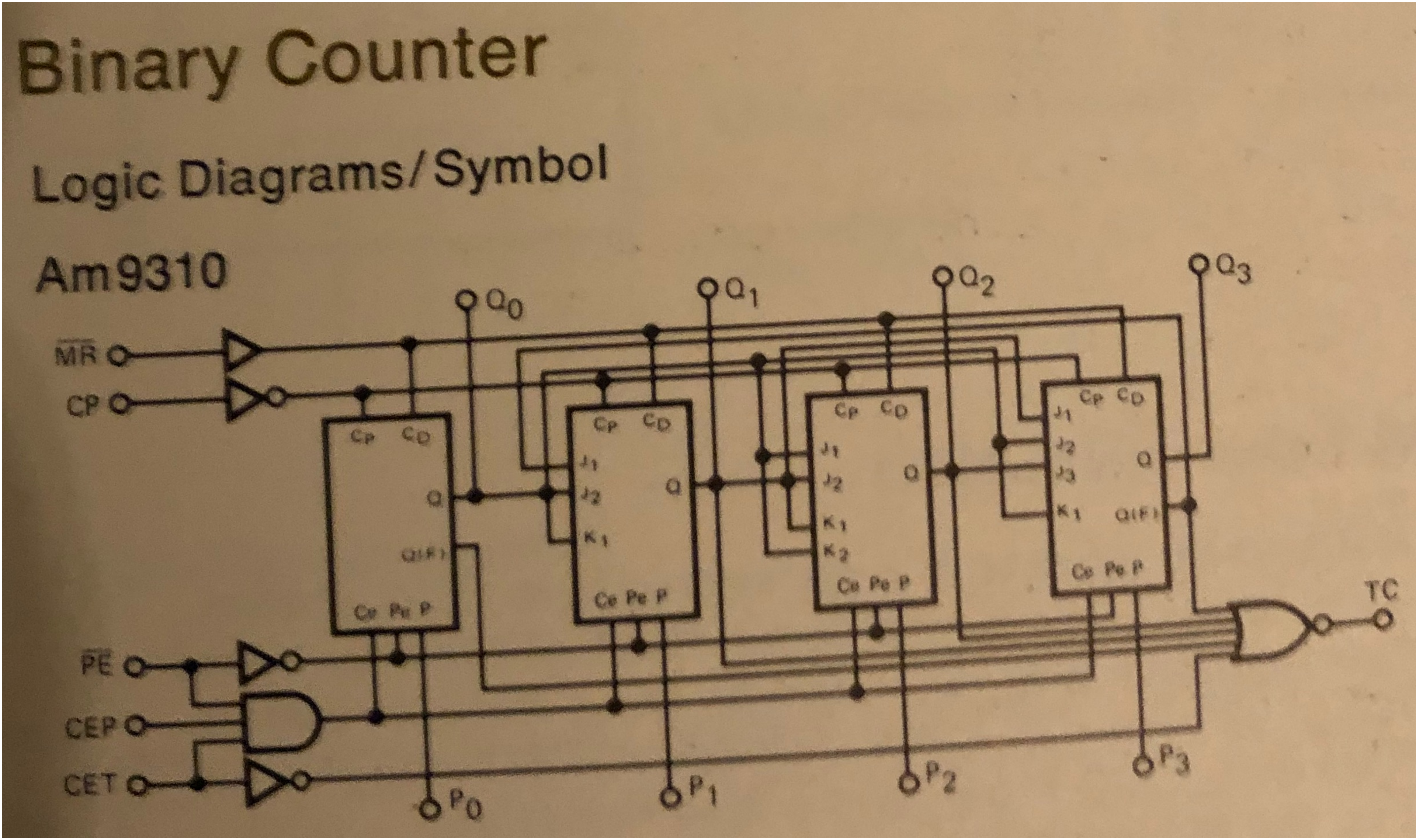
Counter

The counter in this example is a 4-bit asynchronous counter based on JK flip-flops. The flip-flops are connected with both their J and K terminals to the enable pin, putting them in "toggle mode". The flip-flop to the left, producing the Q0 signal, will change its output state for each falling edge of the clock signal, for example, a CPU clock. Since the output toggles for each falling edge of the clock, the clock toggles twice for each toggle of the output.



This diagram from a simulation shows how the logic levels of the four bits change over time. The enable signal goes from 0 to 1 after one second.

AMD Counter


Binary Counter
Logic Diagrams/Symbol
Am9310
MR
CP
PE
CEP
CET
Q0
Q1
Q2
Q3
CP
CD
J1
J2
J3
K1
K2
Q
Q(F)
Ce Pe P
TC
P0
P1
P2
P3

Section

Logic

- Multiplication
- Division

Binary Multiplication

Ancient Egyptian multiplication

From Wikipedia, the free encyclopedia
(Redirected from Russian peasant algorithm)

In mathematics, **ancient Egyptian multiplication** (also known as **Egyptian multiplication**, **Ethiopian multiplication**, **Russian multiplication**, or **peasant multiplication**), one of two multiplication methods used by scribes, was a systematic method for multiplying two numbers that does not require the multiplication table, only the ability to multiply and divide by 2, and to add. It decomposes one of the multiplicands (preferably the smaller) into a sum of powers of two and

Ancient ***Peasant*** Multiplication

Quora

Multiplication

Jeff Drobman · Just now

multiplication is usually done completely in hardware, via a 2D array of "XY(i) + C" multiplier modules, whereby each row generates a partial product of the next signed digit of the multiplier times the multiplicand. shifting occurs in the hardware placement of each row. this array can also be pipelined, so multiple operations can be performed in sequential concurrency.

(See the 1971 **Am2505** 2x4-bit multiplier slice, and my personal MS thesis.)

Binary Multiplication

Ancient ***Peasant*** Multiplication

How do I write an ARM (Assembly) program that determines the product of 2 numbers using Russian peasant multiplication?

Jeff Drobman · just now
Former Stock Trader and App Developer (2003–present)

first learn ARM assembly language. then, follow this algorithm:

1. determine the smaller operand (use a "compare" op) and make it the multiplier
2. create a table of 2x the larger operand (multiplicand)
3. sum the table entries where the binary bit position is 1, and skip the 0's.

Examples [edit]

This example uses peasant multiplication to multiply 11 by 3 to arrive at a result of 33.

3x11

Decimal:		Binary:	
11	3	1011	11
5	6	101	110
2	~~12~~	10	~~1100~~
1	24	1	11000
	33		100001

13x238

13	238	1101	(13)	11101110	(238)
~~6~~	~~476~~	~~110~~	~~(6)~~	~~111011100~~	~~(476)~~
3	952	11	(3)	1110111000	(952)
1	+1904	1	(1)	11101110000	(1904)
	3094				

2’s Complement Multiply

- Simple
  - Convert *Multiplier* to positive if negative
  - Invert *Multiplicand* (if needed)
- Booth’s algorithm
  - Use *Multiplier* as encoded by BA (groups of 2)
  - Leave *Multiplicand* as is

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. The algorithm was invented by Andrew Donald Booth in 1950 while doing research on crystallography at Birkbeck College in Bloomsbury, London. Booth's

Binary Multiplication

Signed 2'sC Multiplication

Booth's multiplication algorithm

Booth's Recoding

From Wikipedia, the free encyclopedia

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. The algorithm was invented by Andrew Donald Booth in 1950 while doing research on crystallography at Birkbeck College in Bloomsbury, London.[1] Booth's algorithm is of interest in the study of computer architecture.

The algorithm [edit]

Booth's algorithm examines adjacent pairs of bits of the 'N'-bit multiplier Y in signed two's complement representation, including an implicit bit below the least significant bit, $y_{-1} = 0$. For each bit y_i, for i running from 0 to $N - 1$, the bits y_i and y_{i-1} are considered. Where these two bits are equal, the product accumulator P is left unchanged. Where $y_i = 0$ and $y_{i-1} = 1$, the multiplicand times 2^i is added to P; and where $y_i = 1$ and $y_{i-1} = 0$, the multiplicand times 2^i is subtracted from P. The final value of P is the signed product.

The representations of the multiplicand and product are not specified; typically, these are both also in two's complement representation, like the multiplier, but any number system that supports addition and subtraction will work as well. As stated here, the order of the steps is not determined. Typically, it proceeds from LSB to MSB, starting at $i = 0$; the multiplication by 2^i is then typically replaced by incremental shifting of the P accumulator to the right between steps; low bits can be shifted out, and subsequent additions and subtractions can then be done just on the highest N bits of P.[2] There are many variations and optimizations on these details.

The algorithm is often described as converting strings of 1s in the multiplier to a high-order +1 and a low-order −1 at the ends of the string. When a string runs through the MSB, there is no high-order +1, and the net effect is interpretation as a negative of the appropriate value.

1-strings

0111..1 = 1000..0 -1

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Binary Multiplication

DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023

Signed 2'sC Multiplication

Drobman MS Thesis

Booth's Recoding

y_{i+1}	y_i	y_{i-1}	y^*_{i+1}	y^*_i	M_i	Operation
0	0	0	0	0	0	add 0
0	0	1	0	1	1	add X
0	1	0	0	1	1	add X
0	1	1	1	0	2	add 2X
1	0	0	$\bar{1}$	0	$\bar{2}$	subtract 2X
1	0	1	$\bar{1}$	1	$\bar{1}$	subtract X
1	1	0	0	$\bar{1}$	$\bar{1}$	subtract X
1	1	1	0	0	0	subtract 0

*bits recoded as a 1-string transformation

TABLE 2.1

Second-Order Recoding

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Binary Multiplication

DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023

Signed 2'sC Multiplication | Booth's Recoding

Drobman MS Thesis
1973

UNIVERSITY OF CALIFORNIA

Los Angeles

Theory and Design of a

High Speed, Two's Complement Arithmetic Unit

Using an Array of Digital Multiplier Modules

A thesis submitted in partial satisfaction of the

requirements for the degree Master of Science

in Computer Science

by

Jeffrey Howard Drobman

Binary Multiplication

Signed 2'sC Multiplication

Drobman MS Thesis

Booth's Recoding

2.3.2 Booth's Algorithm

The discussion, so far, has been restricted to multipliers in magnitude representation. The concepts developed can be extended, however, to multipliers in two's complement as well. In fact, a simple recoding scheme for two's complement multipliers was developed some time ago by A. D. Booth and K. H. V. Booth, as described in Chu [6]. Booth's Algorithm (as it is commonly called) involves a recoding of the type described in the previous section with k = 1, with the important exception that when the most significant bit is a member of a 1-string, the multiplier is not extended -- due to its being in two's complement. Ignoring this extension preserves implicit value for two's complement multipliers, quite unlike the case of its mag-

(2.2.19).)

Booth's Algorithm is merely the recoding scheme itself, and Chu's presentation [6] starts with the recoding scheme and then work back to the original expression. While this does prove Booth's Al-

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

Am2505 Multiplier

DR JEFF SOFTWARE INDIE APP DEVELOPER

Bit-slice | 1971-80

#18 SN 7437 N

6 of N 7400A Sig. DM 7400N Nat. N 7400P M U6A 7400 59 F

Am2505

Four-Bit by Two-Bit 2's Complement Multiplier

Advanced Micro Devices

Complex Digital Integrated Circuits

Distinctive Characteristics:

- Provides 2's complement multiplication at high speed without correction.
- Can be used in an iterative scheme or time sequenced mode.
- Multiplies two 12-bit signed numbers in typically 200ns.
- Multiplies in active HIGH (positive logic) or active LOW (negative logic) representations.
- Easy correction for unsigned, sign-magnitude or 1's complement multiplication.
- 100% reliability assurance testing in compliance with MIL STD 883.

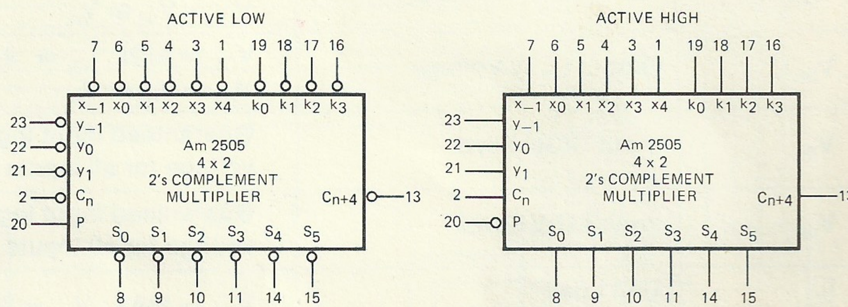
FUNCTIONAL DESCRIPTION:

The Am2505 is a high-speed digital multiplier that can multiply numbers represented in the 2's complement notation and produce a 2's complement product without correction. The device consists of a 4x2 multiplier that can be connected to form iterative arrays able to multiply numbers either directly, or in a time sequenced arrangement. The device assumes that the most significant digit in a word carries a negative weight, and can therefore be used in arrays where the multiplicand and multiplier have different word lengths. The multiplier uses the quaternary algorithm and performs the function S = X Y + K where K is the input field used to add partial products generated in the array. At the beginning of the array the K inputs are available to add a signed constant to the least significant part of the product. Multiplication of an m bit number by an n bit number in an array results in a product having m+n bits so that all possible combinations of product are accounted for. If a conventional 2's complement product is required the most significant bit can be ignored, and overflow conditions can be detected by comparing the last two product digits.

Figure 2 shows how multipliers are connected together in an array. A number of connection schemes are possible. Figure 4 shows diagramatically the connection scheme that results in the fastest multiply. If higher speed is required an array can be split into several parts, and the parts added with high-speed look-ahead carry adders such as the Am9340.

Provision is made in the design for multiplication in the active high (positive logic) or active low (negative logic) representations simply by reinterpreting the active level of the input operands, the product, and a polarity control P̄. For a more complete description and applications the user is referred to the Am2505 Application Note.

LOGIC SYMBOLS



V_{CC} = PIN 24
GND = PIN 12

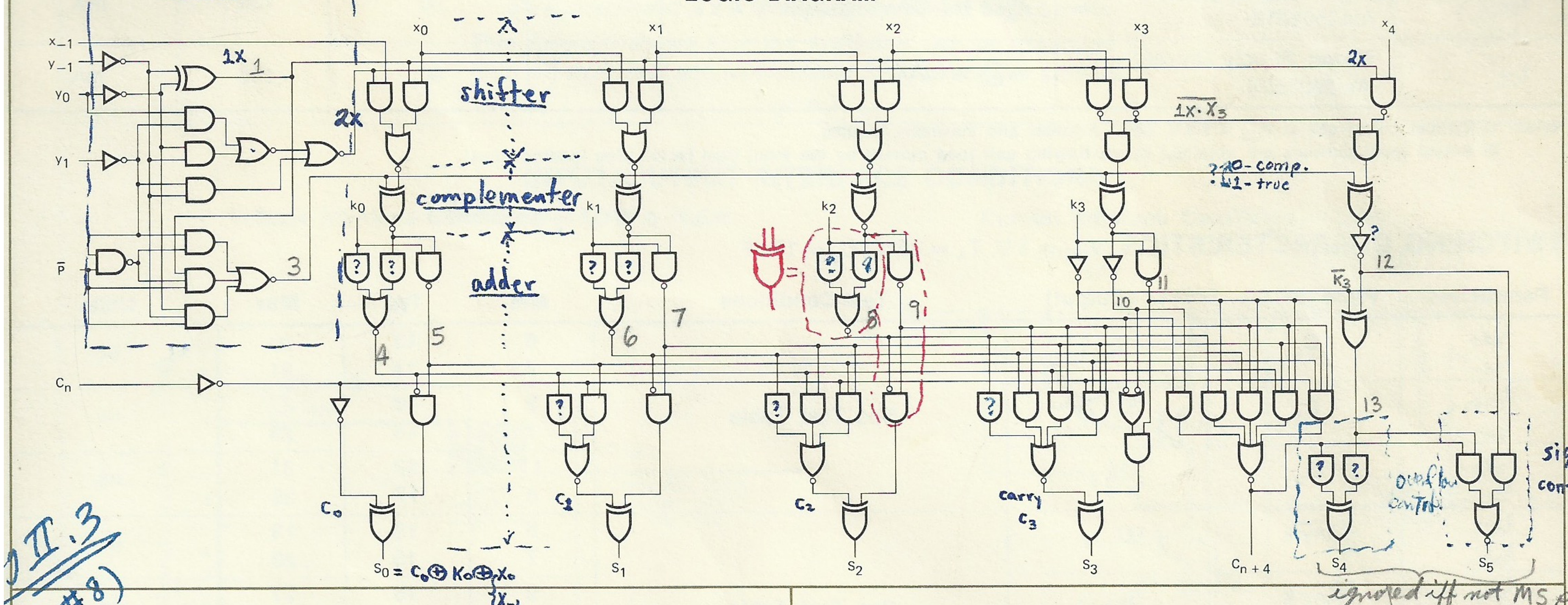
CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Am2505 Multiplier

Bit-slice | 1971-80

DR JEFF SOFTWARE
INDIE APP DEVELOPER

LOGIC DIAGRAM



$S_0 = C_0 \oplus K_0 \oplus X_0$

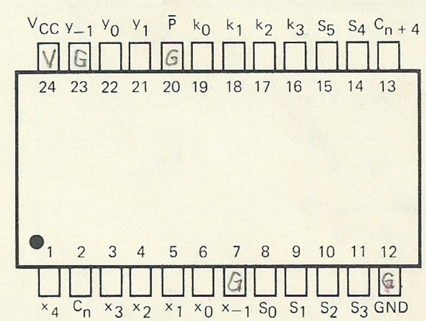
Am2505 ORDERING INFORMATION

Package Type	Temperature Range	Order Number
Silicone DIP	0°C to +75°C	AM250559C
Hermetic DIP	0°C to +75°C	AM250559F
Hermetic DIP	−55°C to +125°C	AM250551F
Hermetic Flat Pak	−55°C to +125°C	AM250551P
Dice	Note	AM2505XXD

Note: The dice supplied will contain units which meet both 0°C to +75°C and −55°C to +125°C temperature ranges.

Fig. II. 1.

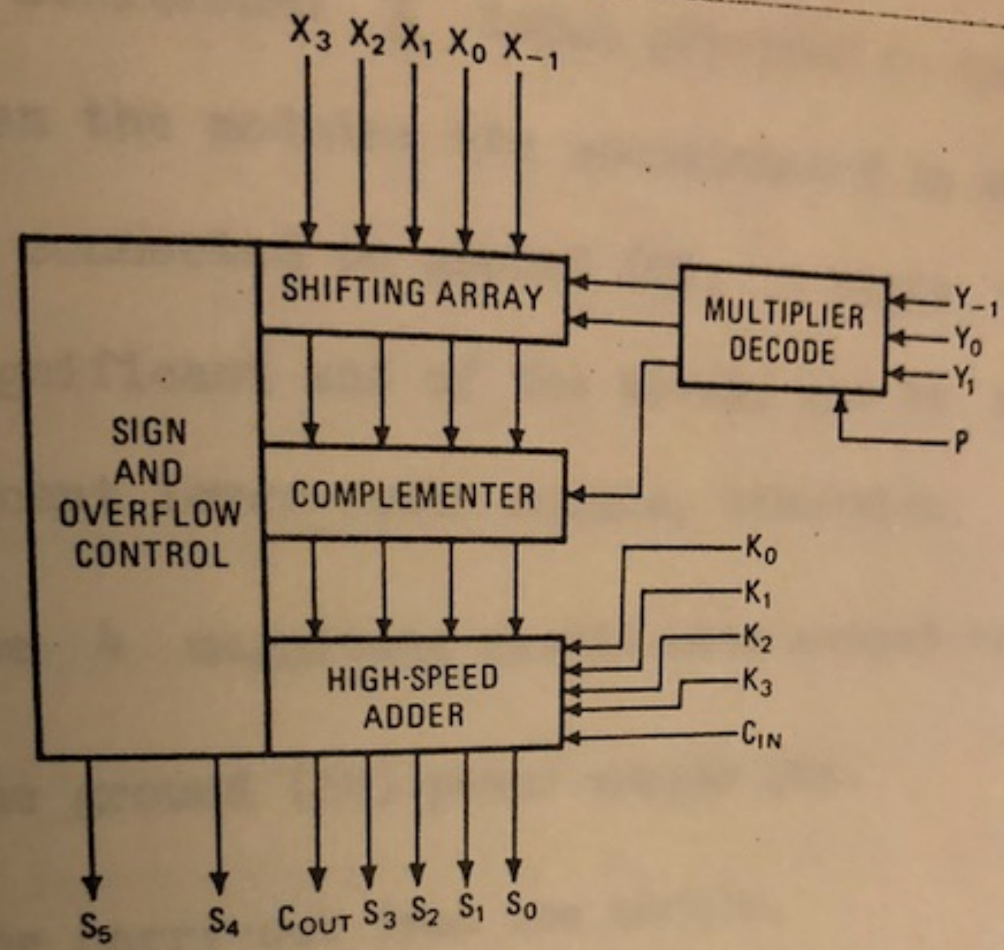
CONNECTION DIAGRAM
Top View



NOTE: PIN 1 is marked for orientation.

Am2505 Multiplier

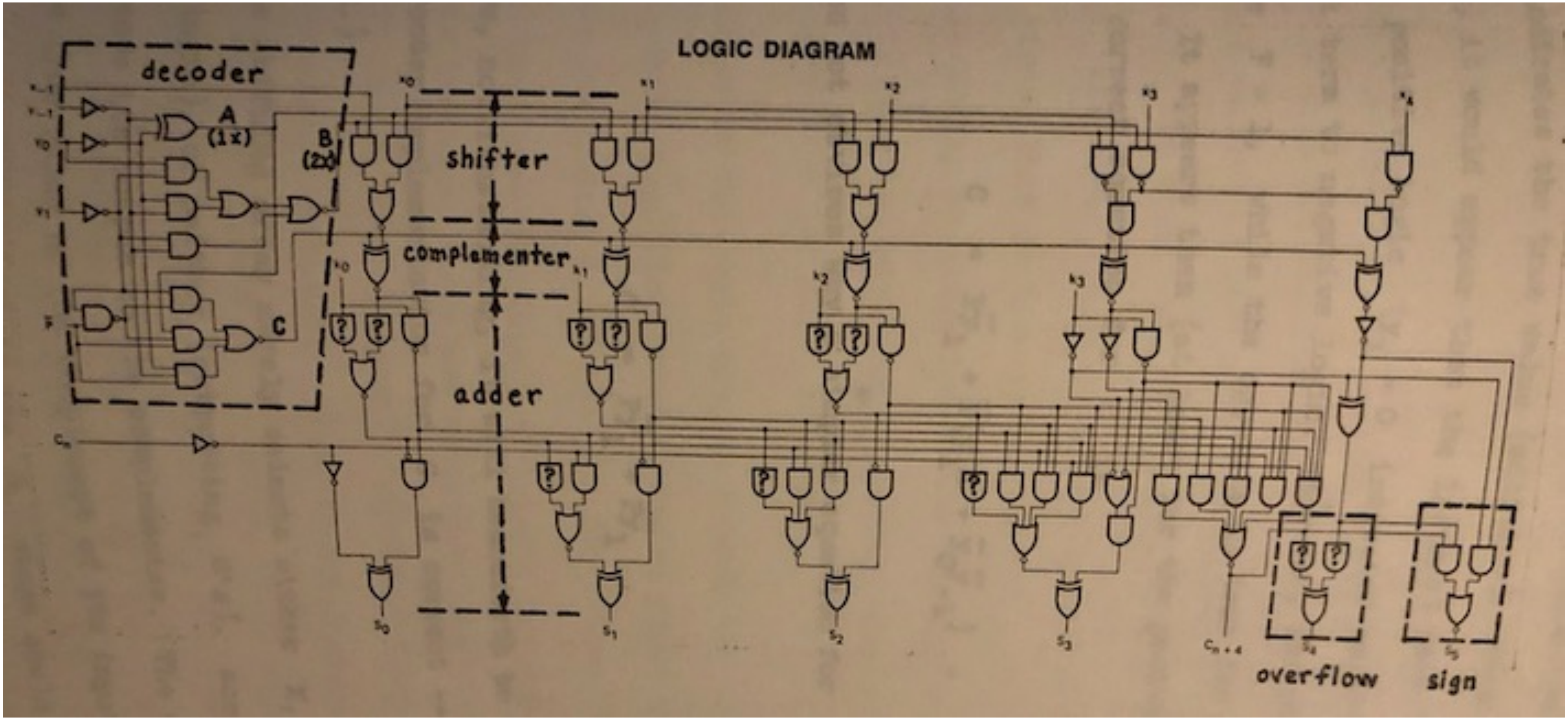
Drobman MS Thesis | Bit-slice | 1971-80



Am2505 Functional Block Diagram

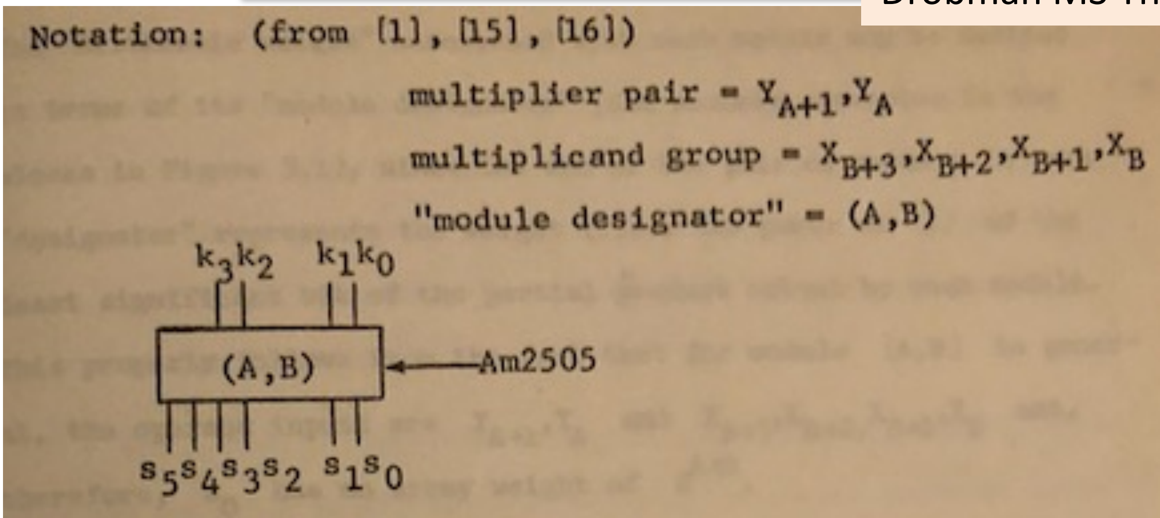
Am2505 Multiplier

Drobman MS Thesis | Bit-slice | 1971-80



Am2505 Multiplier

Drobman MS Thesis | Bit-slice | 1971-80



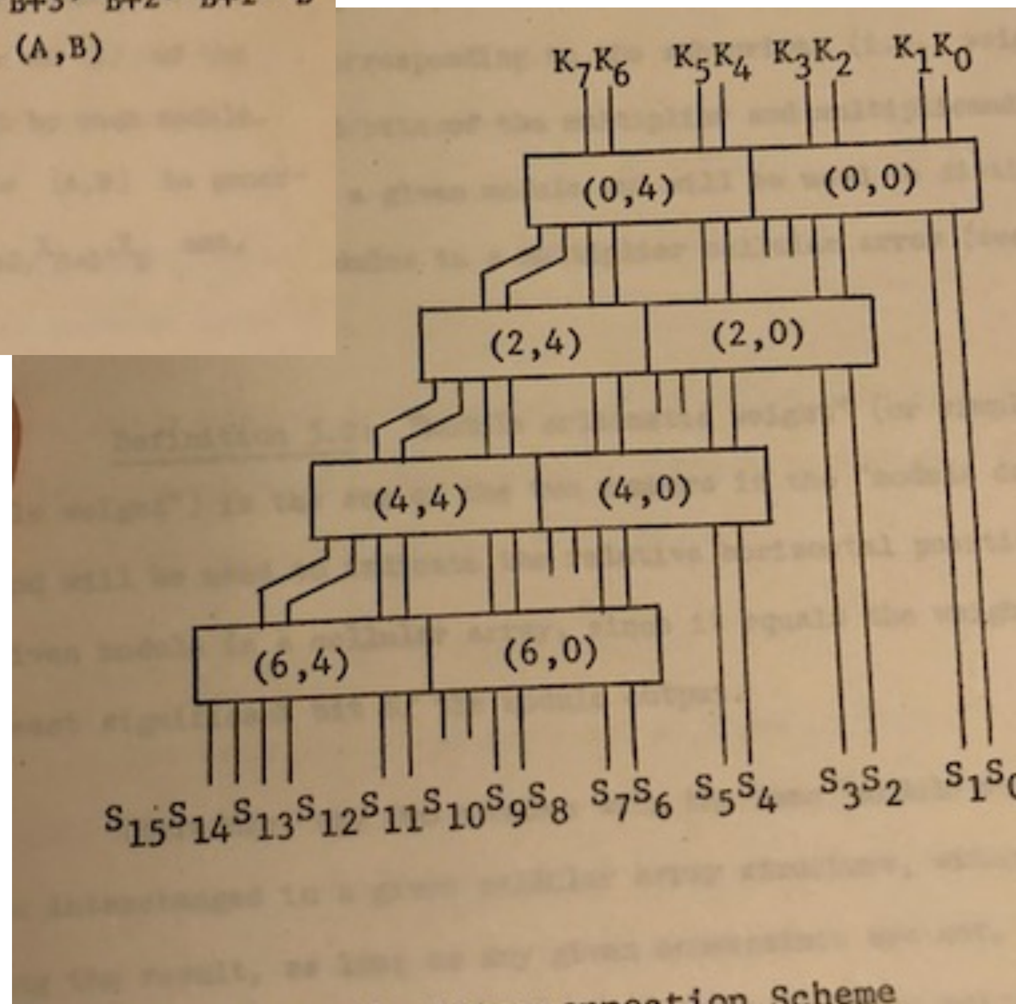


Figure 3.1. Basic Interconnection Scheme

2x4-bit slices

8-bit x 8-bit multiply

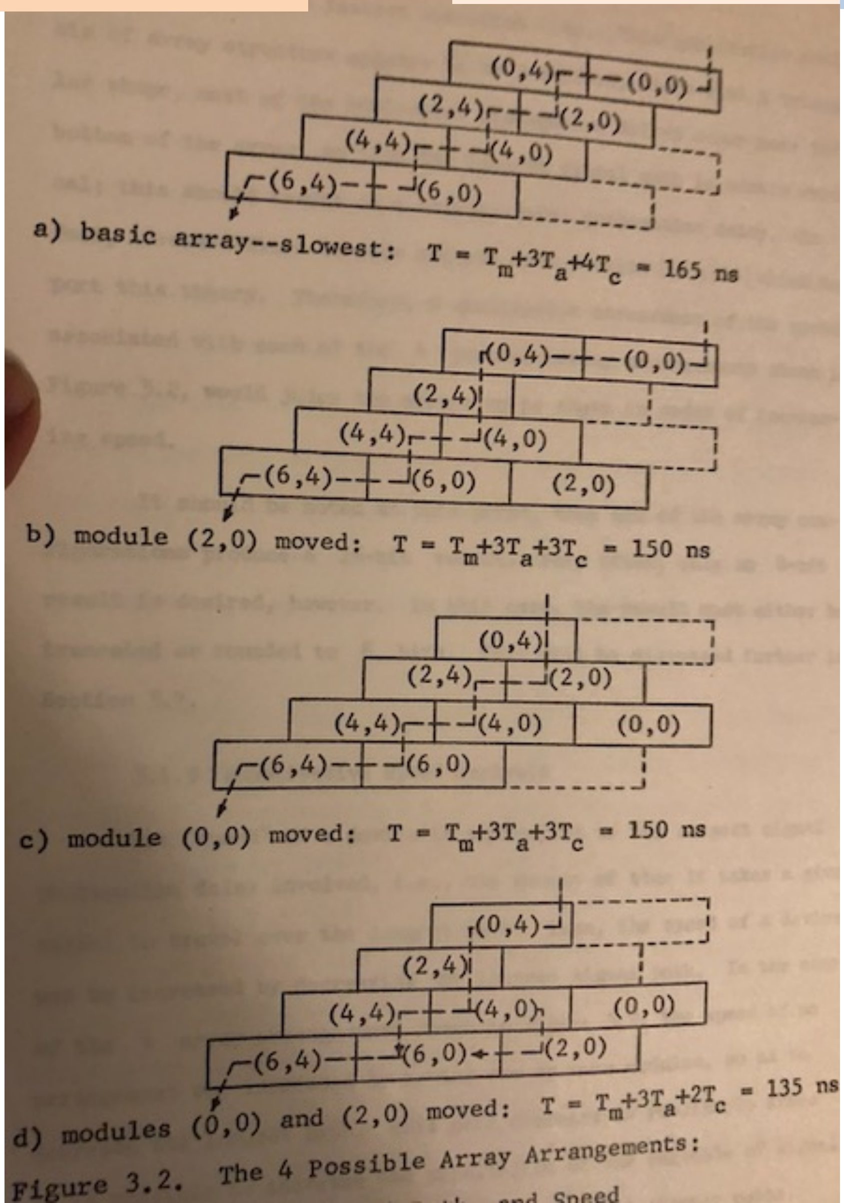
CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

Am2505 Multiplier

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Timing Analysis | Drobman MS Thesis | Bit-slice | 1971-80



Division

Non-Restoring Div

How do calculators calculate binary division?

Jeff Drobman, Lecturer at California State University, Northridge (2016-present)

Answered just now

the most common division algorithm used in the past was "non-restoring". but there are others, as listed in Wikipedia:

"Division algorithms fall into two main categories: slow division and fast division. Slow division algorithms produce one digit of the final quotient per iteration. Examples of slow division include restoring, non-performing restoring, non-restoring, and SRT division. Fast division methods start with a close approximation to the final quotient and produce twice as many digits of the final quotient on each iteration. Newton–Raphson and Goldschmidt algorithms fall into this category."

Quora

Division

Simple

Here's one way to implement the iterative approach: (There are others...)

1. Align the leading 1s of the *significands.* This is usually easy in floating point—they're already aligned by the format, unless one or both of the numbers is *subnormal (aka. denormal).*
2. Compare the dividend and the divisor.
   a. If the dividend is not smaller than the divisor, subtract the divisor from the dividend and write a 1.
   b. Otherwise, don't subtract, and write a 0.
3. Shift the dividend (or what remains of it) left by 1 bit.
4. Repeat steps 2 and 3 until you have sufficient quotient bits—namely, that you have a 1 in the "hidden 1" position of the quotient.

Quora

Division

The original Pentium implemented a faster iterative approach that produced *2* bits per iteration: *Radix-4 SRT division.* I won't go into the details of the algorithm. I will point out three salient features:

1. It recodes the numbers into a *redundant representation,* meaning that each bit of the inputs expand to multiple bits in the recoded representation. The redundant representation allows deferring carries and borrows.
2. It uses a large lookup table to decide what action to take at each step.
3. Rather than just producing 2 bits of quotient per iteration, it actually produces one of 5 values at each step: -2, -1, 0, +1, +2. Later steps can refine errors introduced in earlier steps.

The infamous Pentium FDIV bug arose from the lookup table mentioned above: There were 5 missing +2 entries in the lookup table on the buggy versions of the Pentium.

SRT division is a nice speedup, but it's only a linear speedup. That is, it doubles the speed. Double precision is still around twice as expensive as single precision.

Quora

Division

As transistors have gotten cheaper, modern hardware has turned to even faster approaches:

- *Newton-Raphson division* works by iterating Newton's root-finding method on $f(x) = \frac{1}{x} - d$ to find the reciprocal of the divisor d. Once you find that, you multiply the dividend by that reciprocal. The infamous Quake 3 Hack is based on this approach, although in that case it was inverse square root rather than an ordinary divide.

- *Goldschmidt division* works a little differently. Multiply both dividend and divisor by a common factor F. F is chosen to push the divisor toward 1.0. Repeat until the divisor is close enough to 1, and stop. If you choose the common factor properly, this converges quickly. AMD processors since Athlon use this approach.

What's neat about Newton-Raphson and Goldschmidt approaches is that both converge *quadratically* when implemented properly. That is, each iteration doubles the number of valid bits in the result estimate. That means single precision results come after just a few iterations, and double precision computations usually only require one additional iteration.

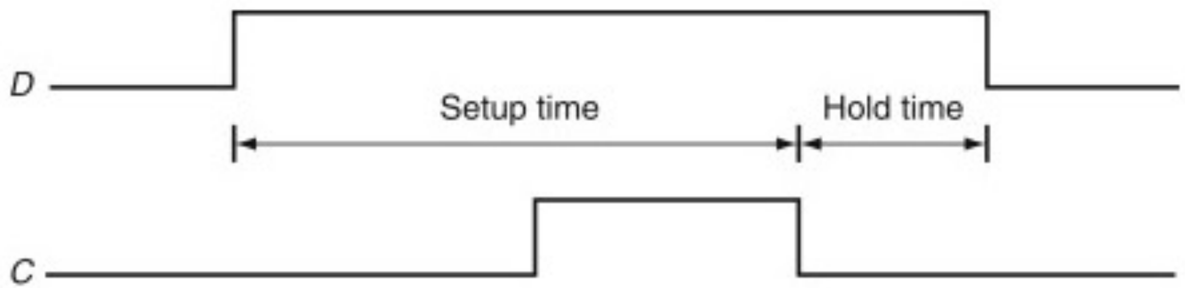
Section

Logic Timing

- Am2900

- Combinational
  - Prop delays
- Sequential
  - Setup times
  - Hold times

Sync Timing (AC)

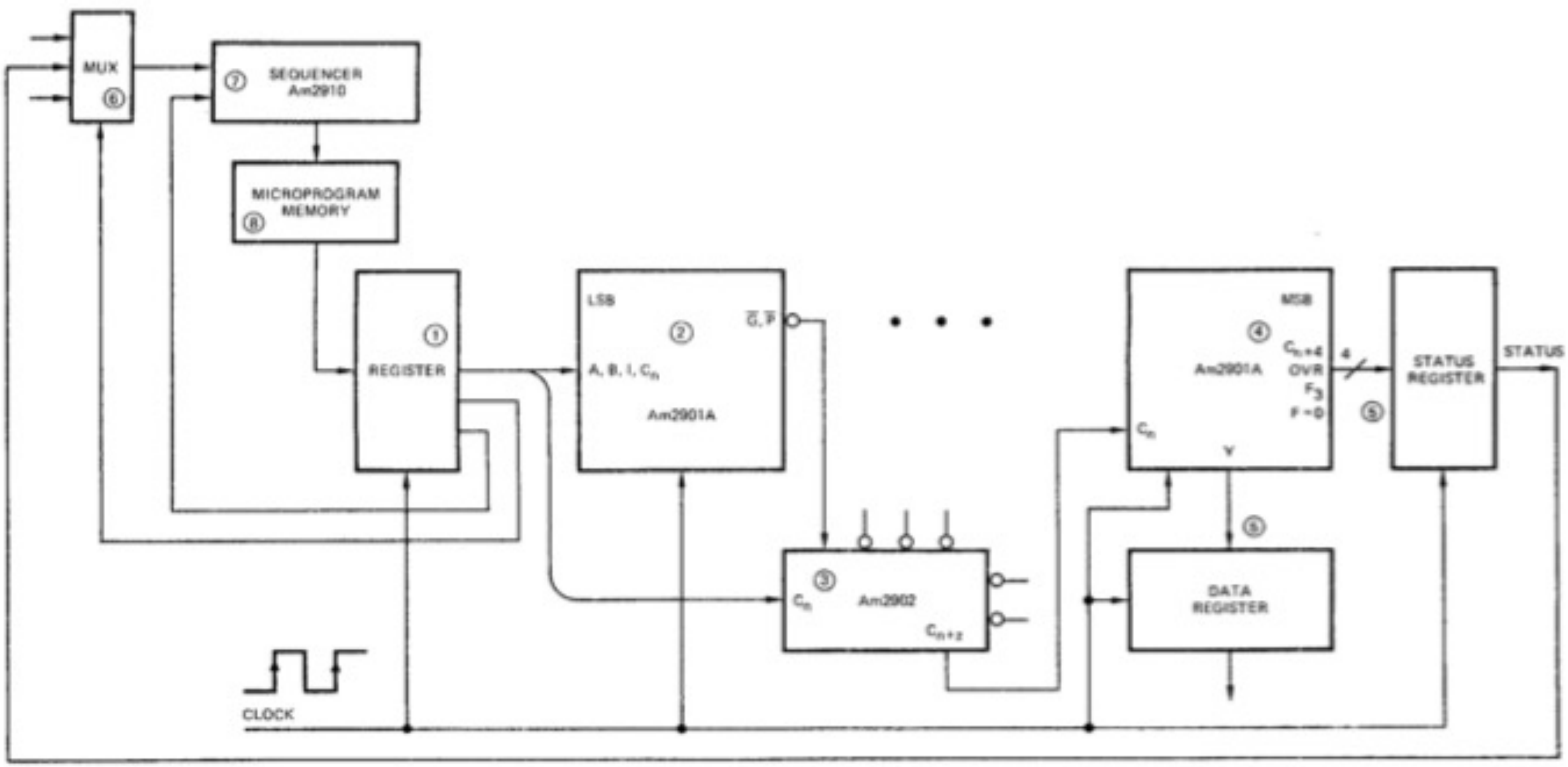


Timing – Cycle Times

Am2900

MINIMUM CYCLE TIME CALCULATIONS FOR 16-BIT SYSTEMS

Speeds used in calculations for parts other than Am2901B are representative for available MSI parts.



Timing

Am2900

B. Combinational Propagation Delays.
C_L = 50pF

From Input \ To Output	Y	F3	Cn+4	$\overline{G}$, $\overline{P}$	F=0	OVR	RAM0 RAM3	Q0 Q3
A, B Address	60	61	59	50	70	67	71	–
D	38	36	40	33	48	44	45	–
Cn	30	29	20	–	37	29	38	–
I012	50	47	45	45	56	53	57	–
I345	51	52	52	45	60	49	53	–
I678	28	–	–	–	–	–	27	27
A Bypass ALU (I = 2XX)	37	–	–	–	–	–	–	–
Clock ↑	49	48	47	37	58	55	59	29

C. Set-up and Hold Times Relative to Clock (CP) Input.

Input	CP: Set-up Time Before H → L	Hold Time After H → L	Set-up Time Before L → H	Hold Time After L → H
A, B Source Address	20	0 (Note 3)	69 (Note 4)	0
B Destination Address	15	Do Not Change		0
D	–	–	51	0
Cn	–	–	39	0
I012	–	–	56	0
I345	–	–	55	0
I678	11	Do Not Change		0
RAM0, 3, Q0, 3	–	–	16	0

Timing – Logic

Am2900

TABLE IV E-2
Guaranteed Combinational Delays
T_C = –55°C to +125°C, V_{CC} = 4.5V to 5.5V
Two's Complement Multiply Instruction
($I_{8765} = 2_H$, $I_{4321} = 0_H$, $I_0 = 0$)

From Input \ To Output	Slice Position	Y	C_{n+4}	$\overline{G}, \overline{P}$	Z (s)	N	OVR	DB	$\overline{WRITE}$	QIO_0 QIO_3	SIO_0	SIO_3	SIO_0 Parity
A, B Address (Arith. Mode)	MSS	113	93	–	–	102	118	52	–	–	97	–	–
	IS, LSS	101	93	84	–	–	–	52	–	–	97	–	–
DA, DB Inputs	MSS	78	62	–	–	66	94	–	–	–	64	–	–
	IS, LSS	64	62	51	–	–	–	–	–	–	64	–	–
$\overline{EA}$	MSS	85	56	–	–	60	87	–	–	–	58	–	–
	IS, LSS	60	56	43	–	–	–	–	–	–	58	–	–
C_n	MSS	58	30	–	–	40	59	–	–	–	38	–	–
	IS, LSS	40	30	–	–	–	–	–	–	–	38	–	–
I_0	MSS	105	97	–	–	89	102	–	–	•	71•	•	–
	IS	105	97	81	–	–	–	–	–	•	71•	•	–
	LSS	105	97	81	42	–	–	–	53	•	71•	•	–
I_{4321}	MSS	112	98	–	–	94	111	–	–	•	75•	•	–
	IS	112	98	85	–	–	–	–	–	•	75•	•	–
	LSS	112	98	85	43	–	–	–	53	•	75•	•	–
I_{8765}	MSS	99	86	–	–	78	100	–	–	•	74•	•	–
	IS	99	86	84	–	–	–	–	–	•	74•	•	–
	LSS	99	86	84	48	–	–	–	50	•	74•	•	–
Clock	MSS	107	90	–	–	89	116	39	–	42	91	–	–
	IS, LSS	89	90	74	57	–	–	39	–	42	91	–	–
Z	MSS	90	65	–	–	70	81	–	–	–	72	–	–
	IS	90	65	48	–	–	–	–	–	–	72	–	–
$\overline{IEN}$	Any	–	–	–	–	–	–	–	24	–	–	–	–
SIO_3, SIO_0	Any	26	–	–	–	–	–	–	–	–	–	–	–

$F = S + C_n$ if $Z = 0$
$S + R + C_n$ is $Z = 1$
$Y_3 = F_3 \oplus OVR$ (MSS)
$Z = Q_0$ (LSS)

Timing – Clocked

Am2900

TABLE IV B
Guaranteed Set-up and Hold Times
T_C = −55°C to +125°C, V_{CC} = 4.5V to 5.5V
All Functions
CAUTION: READ NOTES TO TABLE B. NA = Not Applicable; no timing constraint.

Input	With Respect to this Signal	HIGH-to-LOW Set-up	HIGH-to-LOW Hold	LOW-to-HIGH Set-up	LOW-to-HIGH Hold	Comment
Y	Clock	NA	NA	23	3	To store Y in RAM or Q
$\overline{WE}$ HIGH	Clock	25	Note 2	Note 2	0	To Prevent Writing
$\overline{WE}$ LOW	Clock	NA	NA	35	0	To Write into RAM
A, B as Sources	Clock	38	3	NA	NA	See Note 3
B as a Destination	Clock and WE both LOW	6	Note 4	Note 4	3	To Write Data only into the Correct B Address
QIO_0, QIO_3	Clock	NA	NA	23	3	To Shift Q
I_{8765}	Clock	24	Note 5	Note 5	0	
$\overline{IEN}$ HIGH	Clock	30	Note 2	Note 2	0	To Prevent Writing into Q
$\overline{IEN}$ LOW	Clock	NA	NA	30	0	To Write into Q
I_{43210}	Clock	24	-	74	0	See Note 6

Notes:

1. For set-up times from all inputs not specified in Table IV B, the set-up time is computed by calculating the delay to stable Y outputs and then allowing the Y set-up time. Even if the RAM is not being loaded, the Y set-up time is necessary to set-up the Q register. All unspecified hold times are less than or equal to zero relative to the clock LOW-to-HIGH edge.
2. $\overline{WE}$ controls writing into the RAM. $\overline{IEN}$ controls writing into Q and, indirectly, controls $\overline{WE}$ through the write output. To prevent writing, $\overline{IEN}$ and $\overline{WE}$ must be HIGH during the entire clock LOW time. They may go LOW after the clock has gone LOW to cause a write provided the $\overline{WE}$ LOW and $\overline{IEN}$ LOW set-up times are met. Having gone LOW, they should not be returned HIGH until after the clock has gone HIGH.
3. A and B addresses must be set-up prior to clock LOW transition to capture data in latches at RAM output.
4. Writing occurs when CP and $\overline{WE}$ are both LOW. The B address should be stable during this entire period.
5. Because I_{8765} control the writing or not writing of data into RAM and Q, they should be stable during the entire clock LOW time unless $\overline{IEN}$ is HIGH, preventing writing.
6. The set-up time prior to the clock LOW-to-HIGH transition occurs in parallel with the set-up time prior to the clock HIGH-to-LOW transition and the clock LOW time. The actual set-up time requirement on I_{43210}, relative to the clock LOW-to-HIGH transition, is the longer of (1) the set-up time prior to clock L → H, and (2) the sum of the set-up time prior to clock H → L and the clock LOW time.

Section

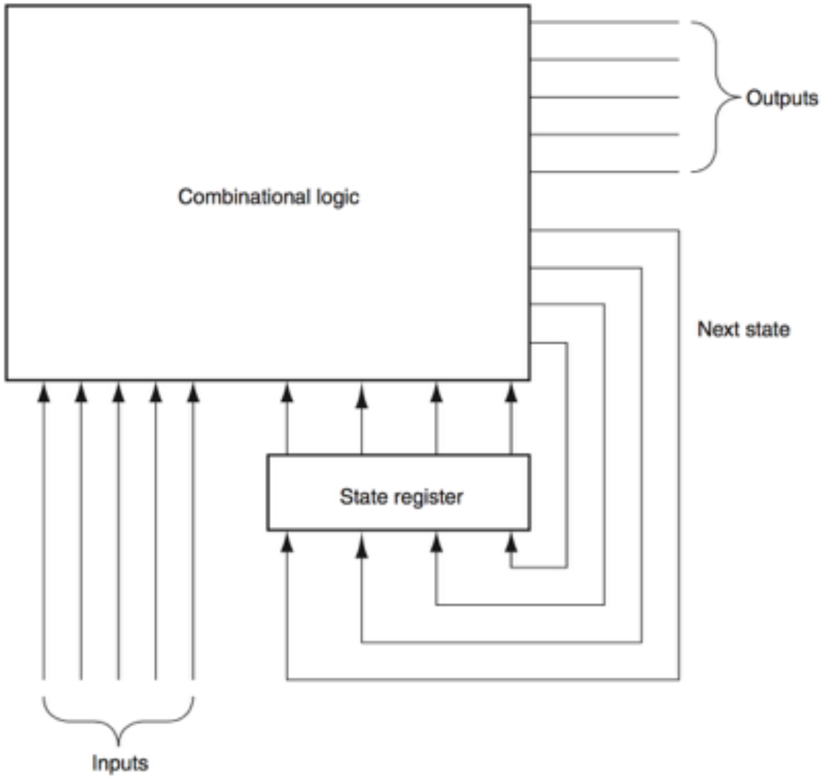
State Machines
FSM

State Machines

FSM

Figure 8.10.3: A finite-state machine is implemented with a state register that holds the current state and a combinational logic block to compute the next state and output functions (COD Figure B.10.3).

The latter two functions are often split apart and implemented with two separate blocks of logic, which may require fewer gates.



Section

Computer Logic Boards

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

DG Nova 16-bit Mini Circuit

DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023

4x TI ‘**181** 4-bit ALU slice