CSUN | CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

COMP 122

DR JEFF SOFTWARE
INDIE APP DEVELOPER

© Jeff Drobman
2016-2023

Spring 2023 | Rev 1-17-23

ASSEMBLY Programming

Logic & Numbers

Dr Jeff Drobman

website → drjeffsoftware.com/classroom.html

email → jeffrey.drobman@csun.edu

Index

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

- Intro
  - Numbers → slide 3
  - Radix, Codes → slide 16
  - Logic → slide 20
  - Arithmetic → slide 35
- Original MSI Logic → slide 45
  - AMD Catalog of Analog (Linear)
  - Digtial MSI → slide 52
  - Memory (RAM) → slide 72
  - Sequential Logic → slide 77
  - Logic: Multi/Div → slide 88
  - Logic Timing (AC) → slide 107
  - State Machines (FSM) → slide 113
  - Computer Logic Boards → slide 115

Section

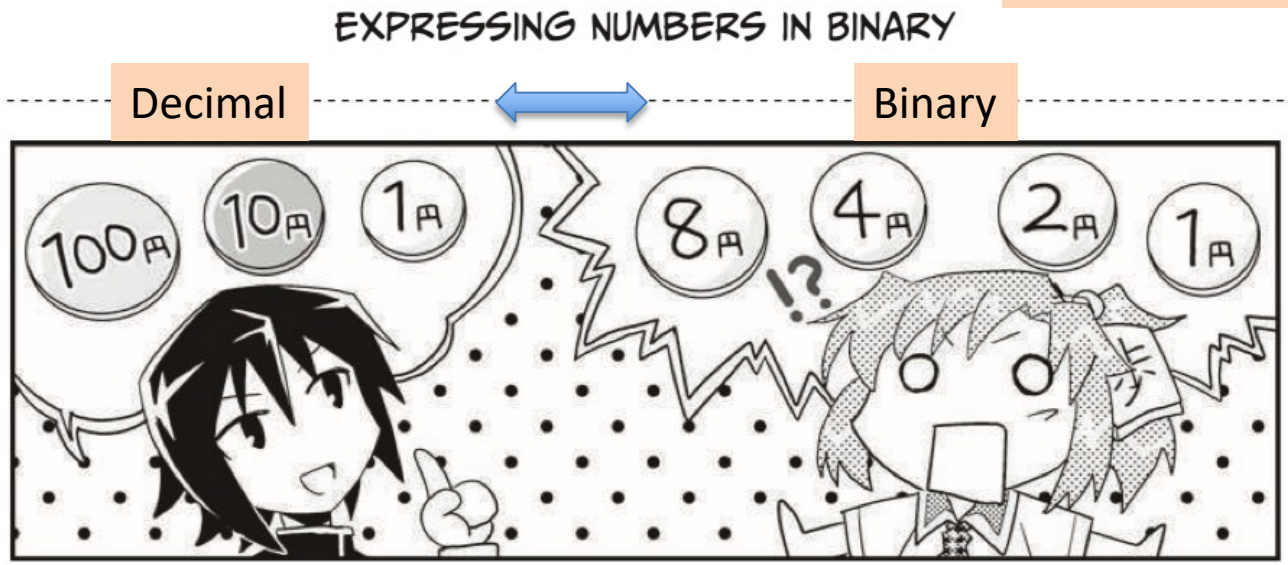
DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023

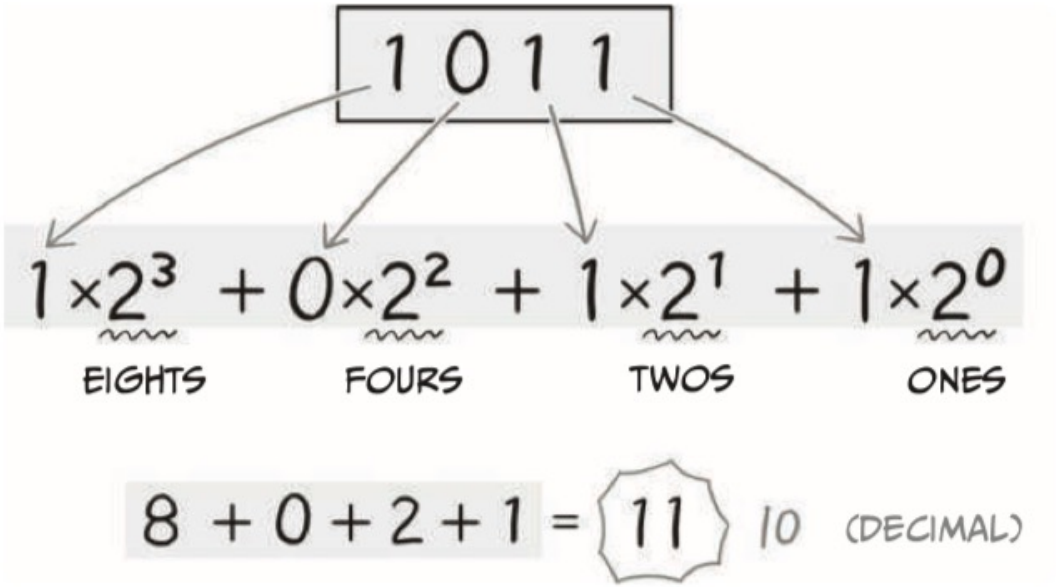
Numbers
(Numeric Data Types)

Radix: Binary to Decimal

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

Manga Guide





Number Systems

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

POSITIONAL REPRESENTATION

$$N = SUM_i\ (n_i * R^i)$$

Notes:
1. R = Radix
2. i = -infinity to +infinity
3. i >= 0 for integers

binary

radix = **2**: <-- $2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ with n = {0, 1}

128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Byte = {0..255} $2^8 = 256$

decimal

radix = **10**: <-- $10^5\ 10^4\ 10^3\ 10^2\ 10^1\ 10^0$ with n = {0 .. 9}

hexadecimal

radix = **16**: <-- $16^5\ 16^4\ 16^3\ 16^2\ 16^1\ 16^0$ with n = {0 .. 9, A, B, C, D, E, F}

binary					*hexadecimal*
	0)	0		8) 1000	
	1)	1		9) 1001	
	2)	10		10) 1010	A
	3)	11	+8	11) 1011	B
	4)	100	-->	12) 1100	C
	5)	101		13) 1101	D
	6)	110		14) 1110	E
	7)	111		15) 1111	F

Data Types– Assembly Level

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

- Hex: 0x5EC46A or 5EC46AH or just 5EC46A
- Binary: B '001100101001'
- Decimal: D '123456' or just 123456
- BCD: packed = [D,D] un-packed = [0,D] [0,D]
- ASCII: A 'TeXt234'
- Signed numbers (16-bit example)
  - Unsigned (addresses) → 0 to 65,535 MMMMM
  - Sign & Magnitude → -32,767 to +32,767 [+0, -0] S MMMM
  - Two's Complement → -32,768 to +32,767 [0 only] S [2^{16}-] MMMM
  - Sign extension SSSSSSSSS [2^{N}-] MMMM

    Preserving arithmetic value

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Numbers

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

- Integers: 123456789

int x = 123.567
assigns 123 to x

- Java ***truncates*** Integers
  - to *Round* add 0.5

- Fixed-point*: 12345.12345

Pi = 3.14159

*NOT a supported ***type*** in any HLL; must use Floating-point

- Floating-point: 1.234512345**e+4**

Pi = 3.14159**e0**

- Java ***rounds*** floats

Accuracy vs. *Precision*

no. digits correct no. digits stored/displayed

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

2's Complement

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

DEC

10's Comp = digits sum to 10

Dec#	10's C
0	10
1	9
2	8
3	7
4	6
5	5
6	4
7	3
8	2
9	1

Ex: 10's C of 98 = 2
89 = 11

HEX

Two's complement 4 bit integer values

Two's complement	Decimal
0111	7.
0110	6.
0101	5.
0100	4.
0011	3.
0010	2.
0001	1.
0000	0.
1111	–1.
1110	–2.
1101	–3.
1100	–4.
1011	–5.
1010	–6.
1001	–7.
1000	–8.

= --0 in SM

BYTE

Eight-bit signed integers

Decimal value	Two's-complement Representation
0	0000 0000
1	0000 0001
2	0000 0010
126	0111 1110
127	0111 1111
–128	1000 0000
–127	1000 0001
–126	1000 0010
–2	1111 1110
–1	1111 1111

Fixed-Pt & Floating-Pt

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

Manga Guide

FIXED-POINT	FLOATING-POINT
123. ← DECIMAL POINT	1.23×10^{2}
1230000.	1.23×10^{6}
0.00000123	1.23×10^{-6}

Floating Point

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

ALL *REAL* NUMBERS

Java

Integers:
Div by 0 → ArithmeticException

1.123456×10^{nnn}

mantissa — characteristic (exponent)

SPECIAL VALUES

- ❑ POSITIVE_INFINITY (=X/0)
- ❑ NEGATIVE_INFINITY (=-X/0)
- ❑ **NaN** (=0/0 and many more)

RANGES	Large (>>1)	Small (<<1)
SINGLE	3.403 E**+38** (2^{128})	1.4 E**-45**
DOUBLE	1.798 E**+308**	4.9 E**-324**

Positive & Negative

IEEE 754

Name	Common name	Base	Digits	Decimal digits	Exponent bits	Decimal E max	Exponent bias[6]	E min	E max
binary16	Half precision	2	11	3.31	5	4.51	$2^4-1 = 15$	−14	+15
binary32	Single precision	2	24	7.22	8	38.23	$2^7-1 = 127$	−126	+127
binary64	Double precision	2	53	15.95	11	307.95	$2^{10}-1 = 1023$	−1022	+1023

Operation Precedence

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

Liang

Expression Evaluation

- Does Not Apply To Assembly

TABLE 3.10 Operator Precedence Chart

Operator
var++ and var– (Postfix)
+, - (Unary plus and minus), ++var and --var (Prefix)
(type) (Casting)
! (Not)
*, /, % (Multiplication, division, and remainder)
+, - (Binary addition and subtraction)
<, <=, >, >= (Comparison)
==, != (Equality)
^ (Exclusive OR)
&& (AND)
|| (OR)
=, +=, -=, *=, /=, %= (Assignment operator)

arithmetic

comparison

logic

order of evaluation. All binary operators except assignment operators are *left associative*. example, since + and – are of the same precedence and are left associative, the expression

```
a - b + c - d  ═══ equivalent ═══  ((a - b) + c) - d
```

Assignment operators are *right associative*. Therefore, the expression

```
a = b += c = 5  ═══ equivalent ═══  a = (b += (c = 5))
```

Ordinals

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

Powers of 2 <> 10: 10:3

```
Technical ordinals
----------
10^(-24)  yacto
10^(-21)  zepto
10^(-18)  atto
10^(-15)  femto
10^(-12)  pico
10^(-9)  nano
10^(-6)  micro
10^(-3)  milli
10^(-2)  centi
10^(-1)  deci
10^(+1)  deka
10^(+2)  hecto
10^(+3)/2^(10)  kilo
10^(+6)/2^(20)  mega
10^(+9)/2^(30)  giga
10^(+12)/2^(40)  tera
10^(+15)/2^(50)  peta
10^(+18)/2^(60)  exa
10^(+21)/2^(70)  zetta
10^(+24)/2^(80)  yotta
```

Gazillions

10^(+6) million
10^(+9) billion
10^(+12) trillion
10^(+15) quadrillion
10^(+18) quintillion
10^(+21) sexillion
10^(+24) septillion
10^(+27) octillion
10^(+30) nonillion
10^(+33) decillion
10^(+36) undecillion
10^(+39) duodecillion
10^(+42) tredecillion
10^(+45) quattuordecillion
10^(+48) quindecillion
10^(+51) sexdecillion
10^(+54) septendecillion
10^(+57) octodecillion
10^(+60) novemdecillion
10^(+63) vigintillion
10^(+100) googol
10^(+303) centillion
10^(10^(+100))
googolplex

Ordinal	Power of 2	Power of 10	Actual
1K	2^{10}	10^{3}	1024
1M	2^{20}	10^{6}	1,048,576
1G	2^{30}	10^{9}	1.074×10^{9}
1T	2^{40}	10^{12}	1.0995×10^{12}

Name	2^n	M/G	Actual
byte	2^{8}	--	256
short	2^{16}	64K	65,536
integer	2^{32}	4B	4.3×10^{9}
long	2^{64}	16 Q	1.84×10^{19}
IPv6	2^{128}	340 uD	3.4×10^{38}

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

GiB/TiB ($2^{30}/2^{40}$)

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

Decimal	Abbreviation	Value	Binary term	Abbreviation	Value	% Larger
kilobyte	KB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	

Actual
1024
1,048,576
1.074×10^9
1.0995×10^{12}

Ordinal	Power of 2	Power of 10	Actual
1K	2^{10}	10^3	1024
1M	2^{20}	10^6	1,048,576
1G	2^{30}	10^9	1.074×10^9
1T	2^{40}	10^{12}	1.0995×10^{12}

Signed Numbers

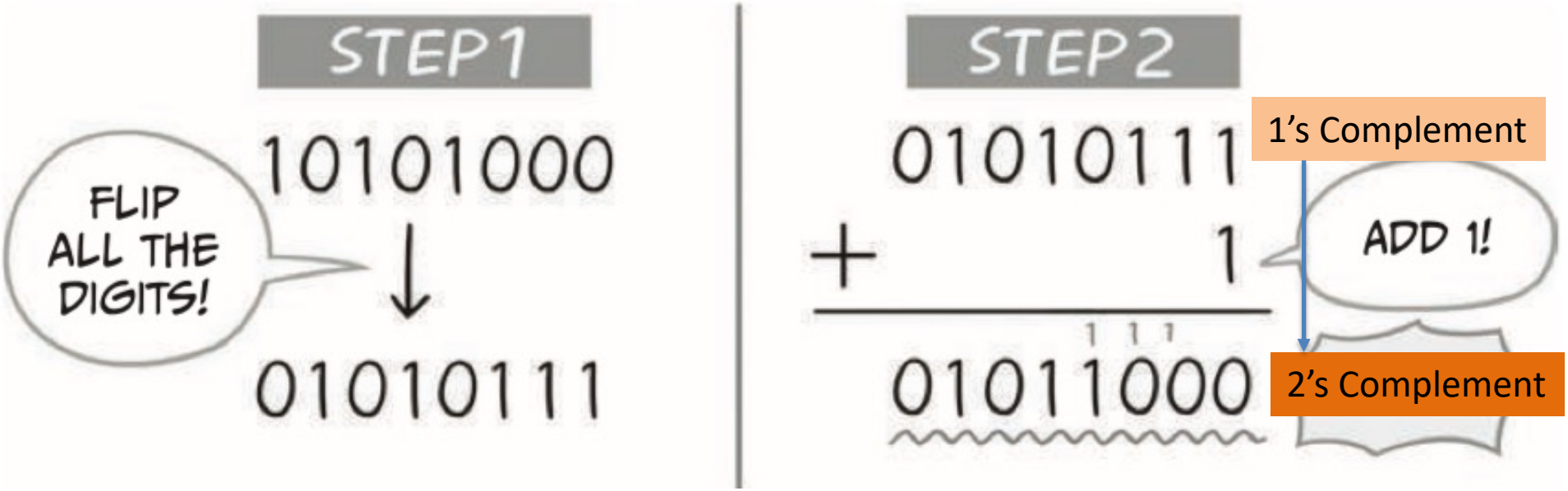
DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023

Binary	Unsigned	Sign Magnitude	Excess-127	Two's Complement
00000000	0	0	-127	0
00000001	1	1	-126	1
⋮	⋮	⋮	⋮	⋮
01111110	126	126	-1	126
01111111	127	127	0	127
10000000	128	-0	1	-128
10000001	129	-1	2	-127
⋮	⋮	⋮	⋮	⋮
11111110	254	-126	127	-2
11111111	255	-127	128	-1

ALU: Negative Numbers

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

2's Complement



- ❖ ADDed directly (no conversion needed)
- ❖ SUBtraction by complement and ADD

1111111111 = -0 in 1'sC
-1 in 2's C

Section

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

- Radix/Base
- Codes

Radix/Base

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

Base									
16	10	9	8	7	6	5	4	3	2
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	10
3	3	3	3	3	3	3	3	10	11
4	4	4	4	4	4	4	10	11	100
5	5	5	5	5	5	10	11	12	101
6	6	6	6	6	10	11	12	20	110
7	7	7	7	10	11	12	13	21	111
8	8	8	10	11	12	13	20	22	1000
9	9	10	11	12	13	14	21	100	1001
A	10	11	12	13	14	20	22	101	1010
B	11	12	13	14	15	21	23	102	1011
C	12	13	14	15	20	22	30	110	1100
D	13	14	15	16	21	23	31	111	1101
E	14	15	16	20	22	24	32	112	1110
F	15	16	17	21	23	30	33	120	1111

ASCII Codes

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

Binary, hexadecimal, and decimal equivalents for each character in "Hello World"

Character	Binary	Hexadecimal	Decimal
H	01001000	48	72
e	01100101	65	101
l	01101100	6C	108
l	01101100	6C	108
o	01101111	6F	111
	00100000	20	32
W	01010111	57	87
o	01101111	6F	111
r	01110010	62	98
l	01101100	6C	108
d	01100100	64	100
NUL	00000000	00	0

UTF-8 Variable Codes

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

UTF-8 encoding of the ISO/IEC 10646 code points

UCS Bits	First Code Point	Last Code Point	Bytes	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+10FFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

1. If the most significant bit of a byte is zero, then it is a single-byte character, and is completely ASCII-compatible.

2. If the two most significant bits in a byte are set to one, then the byte is the beginning of a multi-byte character.

3. If the most significant bit is set to one, and the second most significant bit is set to zero, then the byte is part of a multi-byte character, but is not the first byte in that sequence.

Section

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

Logic

Logic

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

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

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Logic Book

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

allaboutcircuits.com

ALL ABOUT CIRCUITS

Chapters in this Volume

Ch. 1 - Numeration Systems

Ch. 2 - Binary Arithmetic

Ch. 3 - Logic Gates

Ch. 4 - Switches

Ch. 5 - Electromechanical Relays

Ch. 6 - Ladder Logic

Ch. 7 - Boolean Algebra

Ch. 8 - Karnaugh Mapping ➢ *K-maps*

Ch. 9 - Combinational Logic Functions

Ch. 10 - Multivibrators

Ch. 11 - Sequential Circuits

Ch. 12 - Shift Registers

Ch. 13 - Digital-Analog Conversion

Ch. 14 - Digital Communication

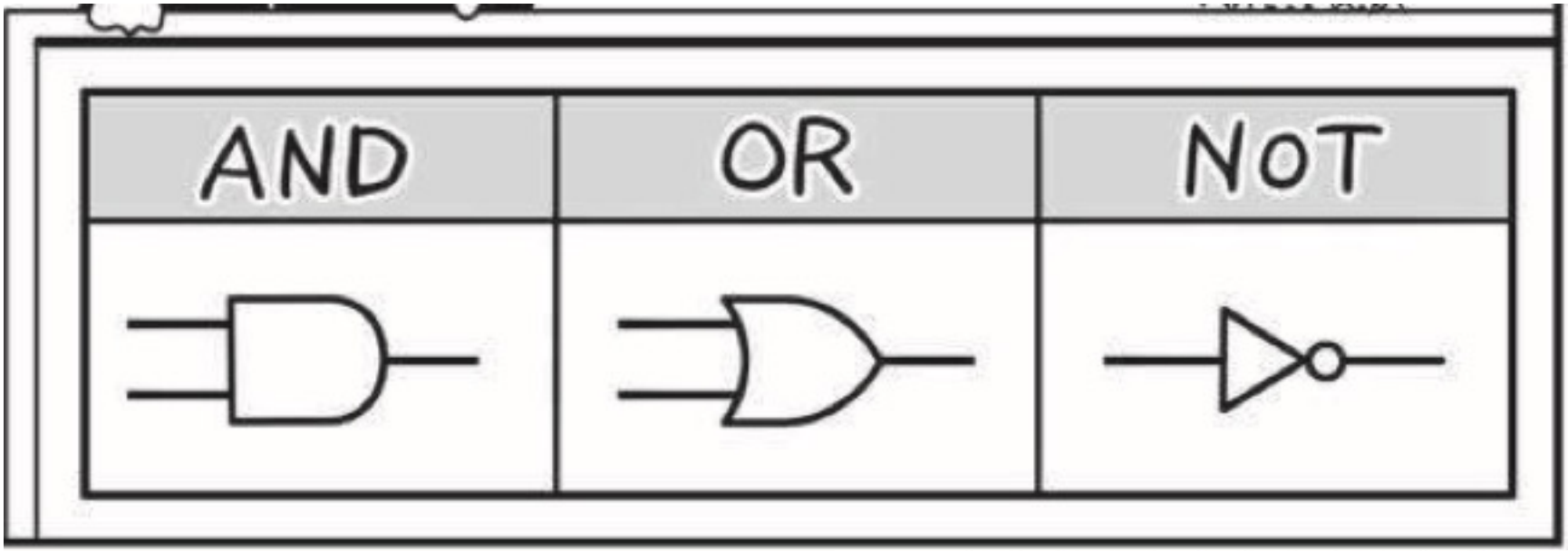
Ch. 15 - Digital Storage (Memory)

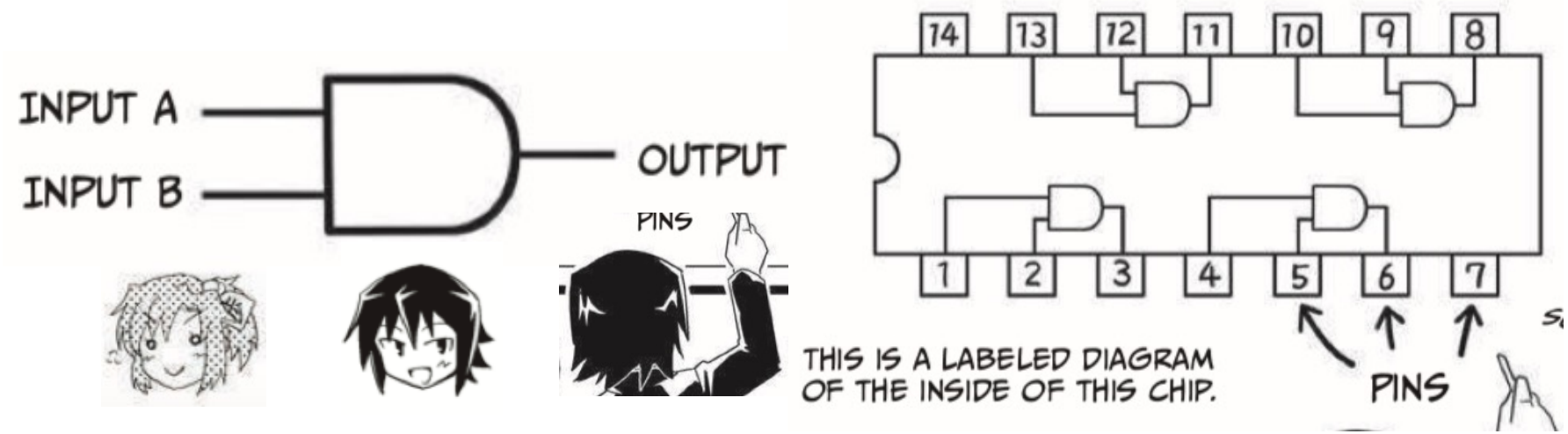
Logic Comic Book

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

Manga Guide

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





Truth Tables

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

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

Bit-wise Operations

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

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

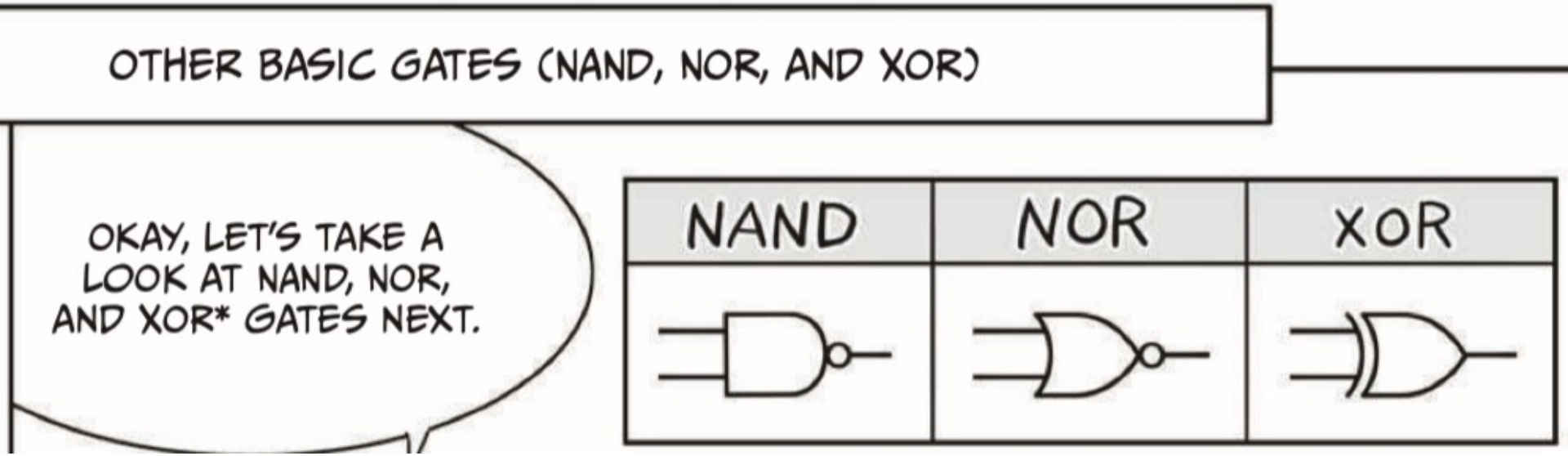
*2^n

Integer types only

Compound Logic Gates

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

Manga Guide



Compound Logic: NAND

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

Manga Guide

NAND GATE (LOGIC INTERSECTION COMPLEMENT GATE)

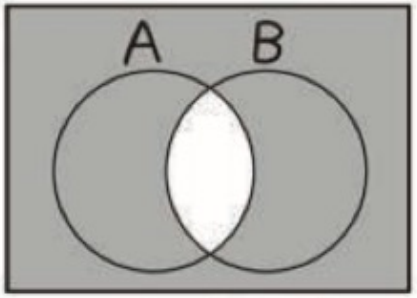
SYMBOL

INPUT OUTPUT

TRUTH TABLE

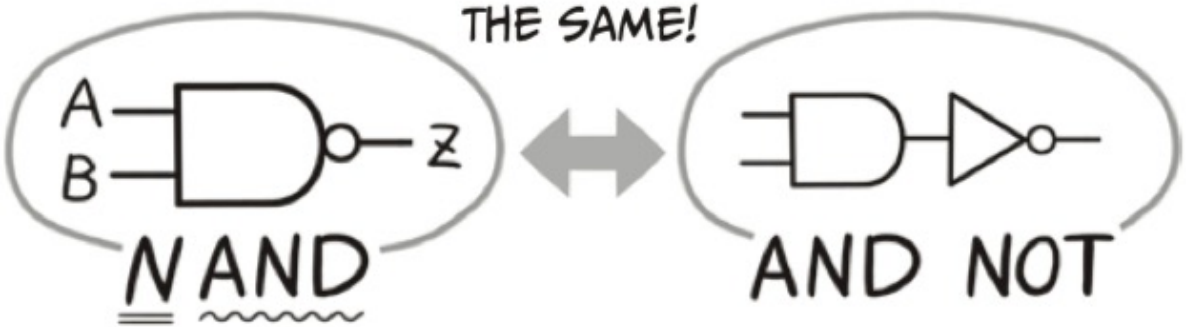
A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

VENN DIAGRAM



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

THE SAME!



Logic Gates: Polarity

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

DeMorgan's Law

Manga Guide

DE MORGAN'S THEOREM

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

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

Logic Gates: Polarity

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

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.

BOTH OF THESE ARE NAND GATES!

BOTH OF THESE ARE NOR GATES!

Transistors to Computers

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

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).

P→N→C MOS

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

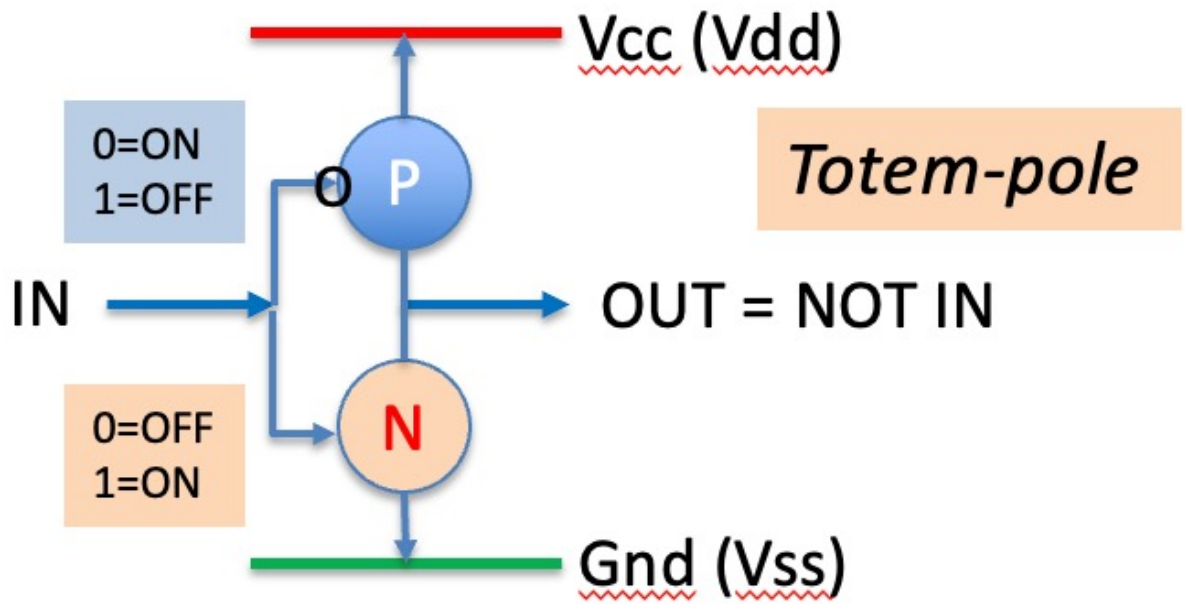
Device/Xtor
Physical
Level

Inverter/Gates

Complementary

CMOS

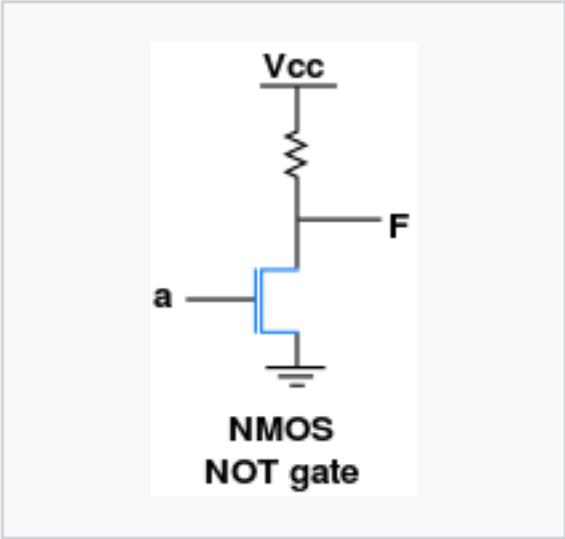
INVERTER



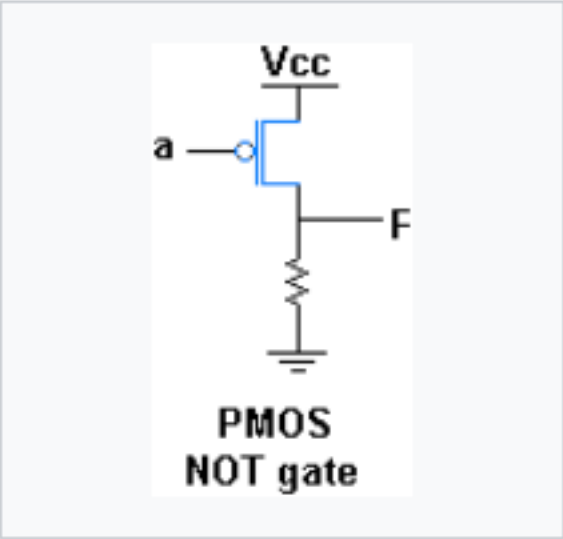
MOS Gates

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

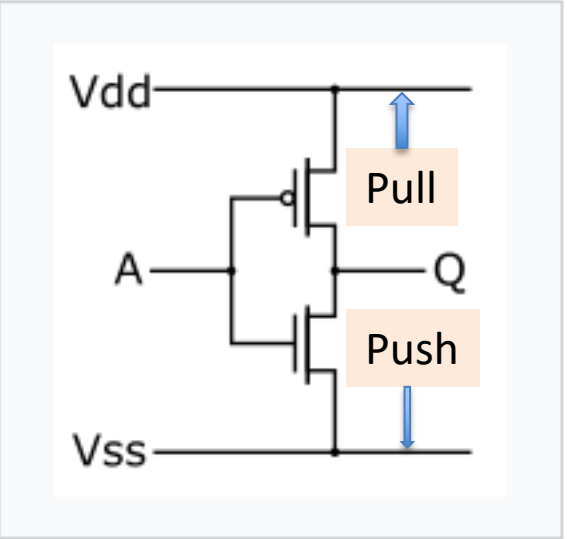
MOSFET



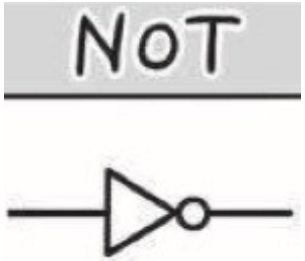
NMOS inverter



PMOS inverter



Static CMOS inverter



CMOS

P/N Totem pole

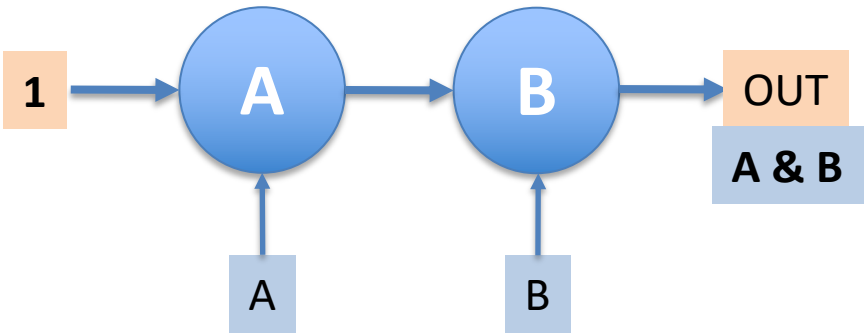
Push-Pull

Logic Gates: AND, OR

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

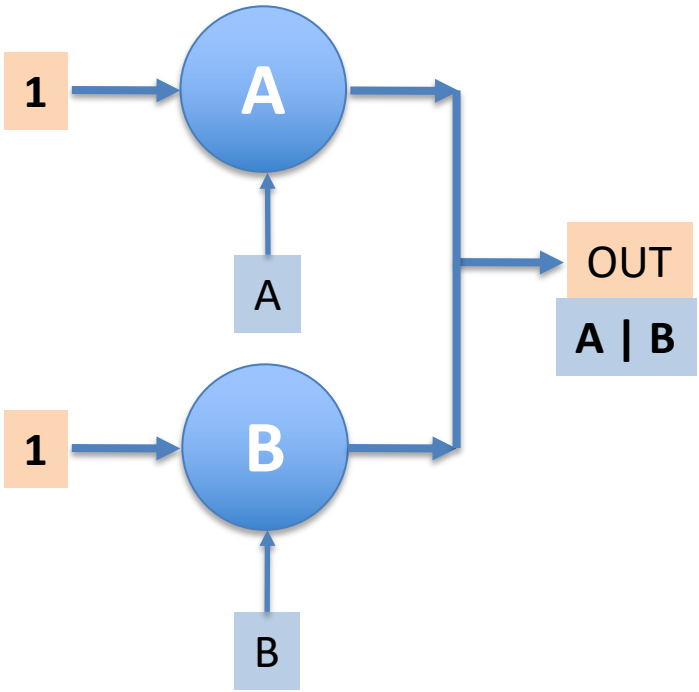
AND

SERIES



OR

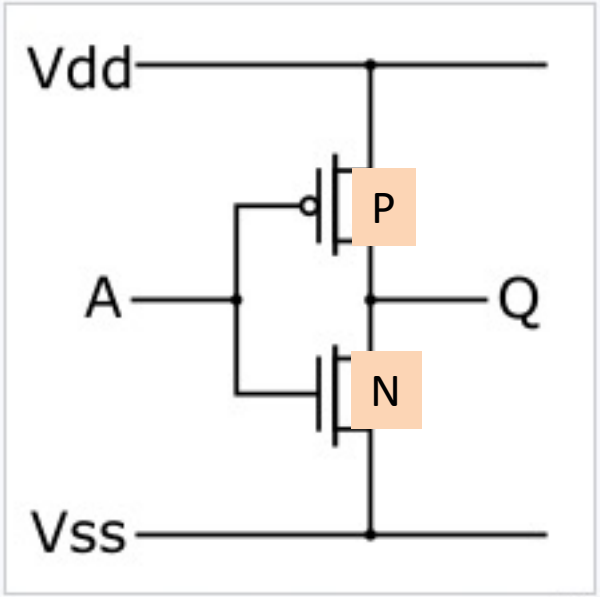
PARALLEL



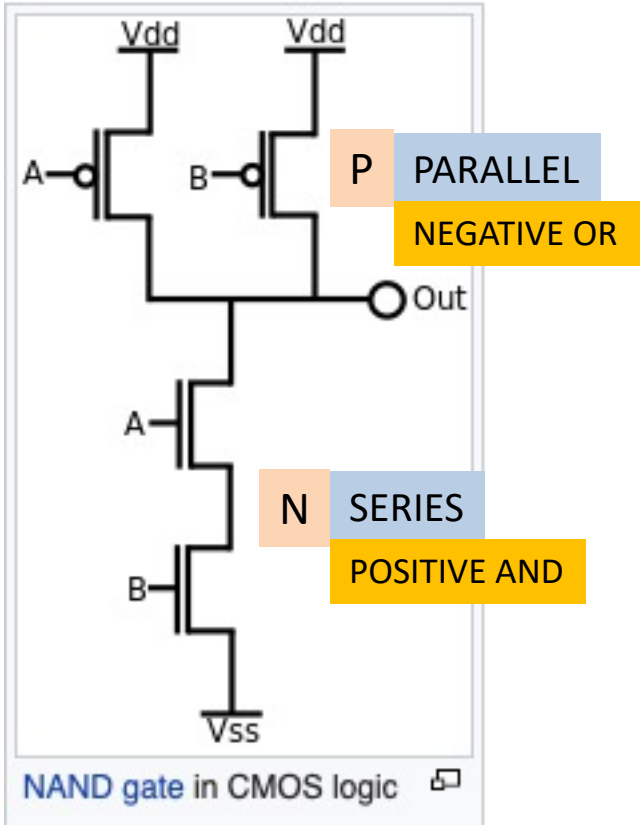
CMOS Gates

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

MOSFET

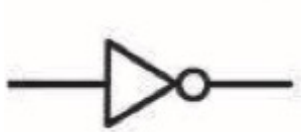


Static CMOS inverter. V_{dd} and V_{ss} are standing for drain and source respectively.



NAND gate in CMOS logic

NOT



NAND

Section

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

Arithmetic

ALU Ops

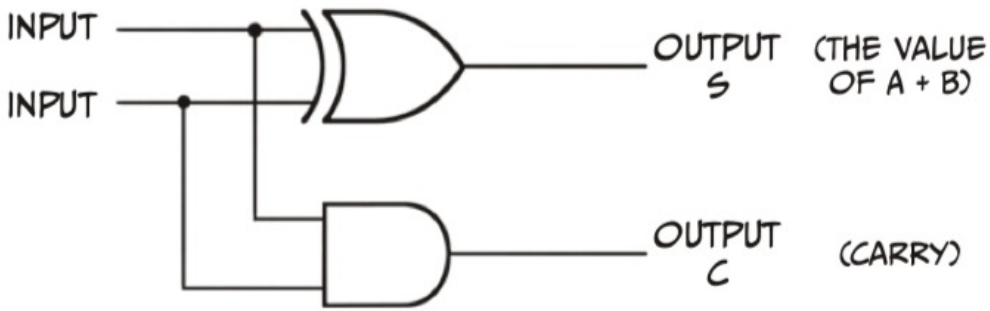
MARS

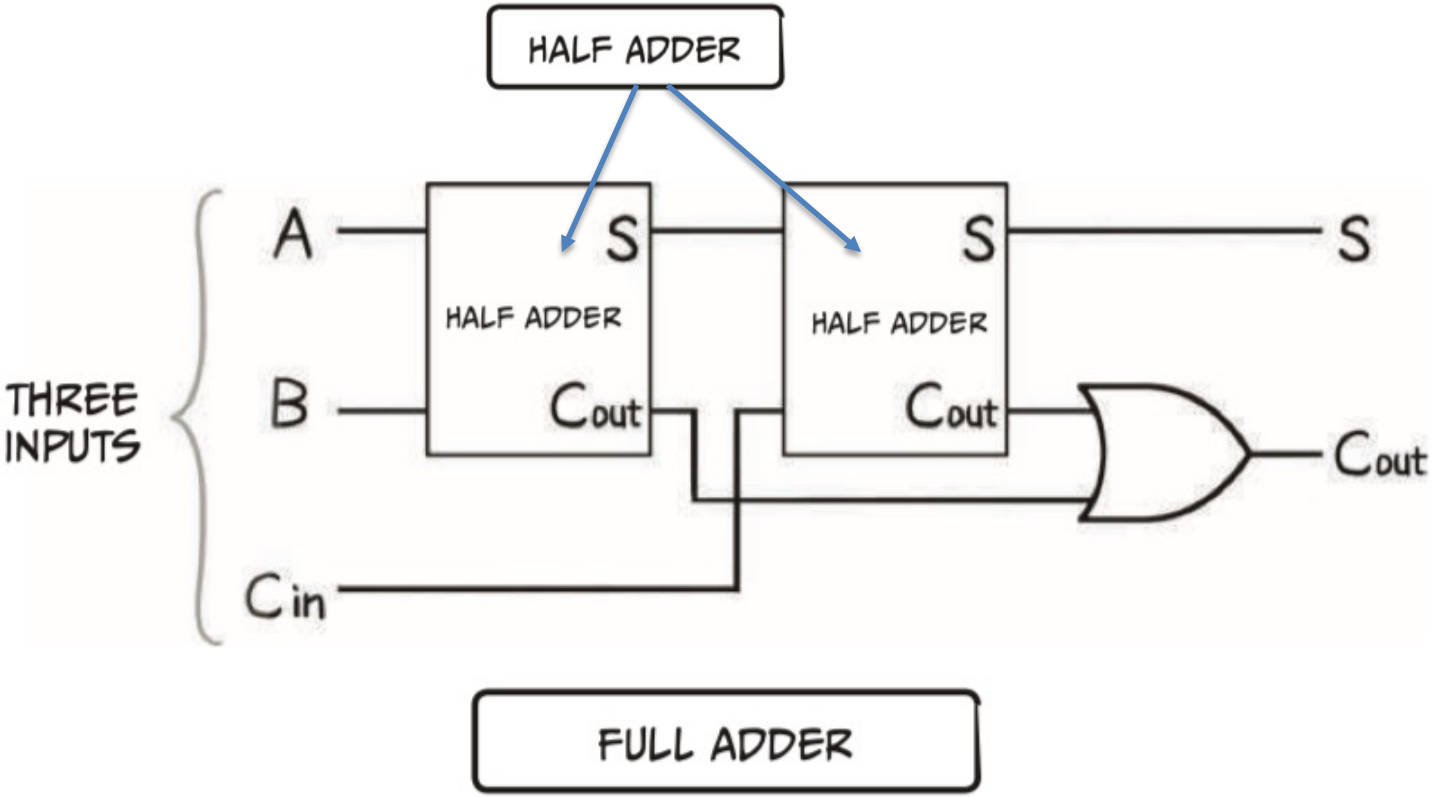
DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023

```
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
```

Adders

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



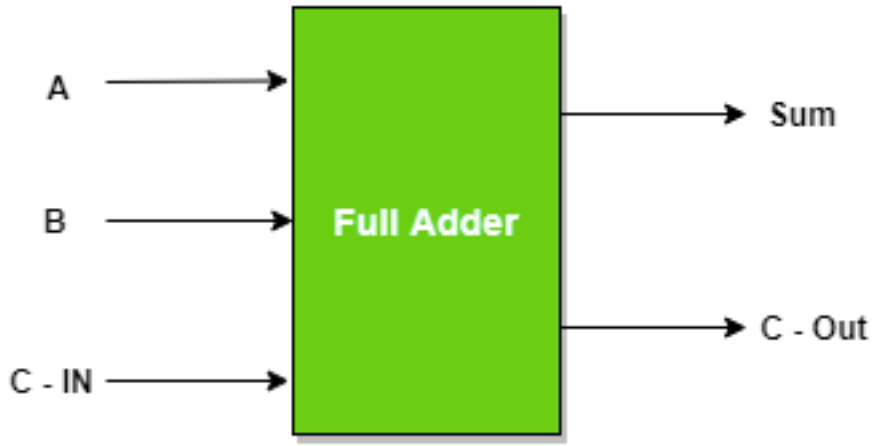


Adders

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

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

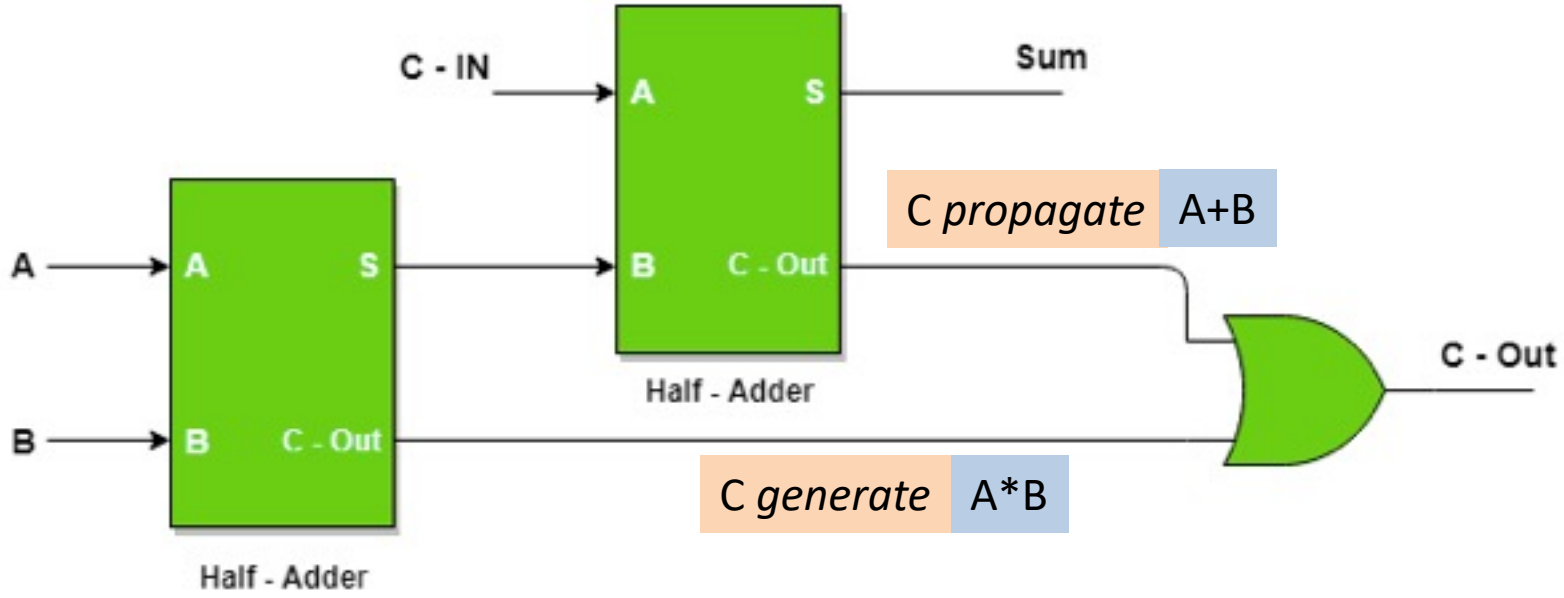
Adders

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

GeeksforGeeks

Implementation of Full Adder using Half Adders

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

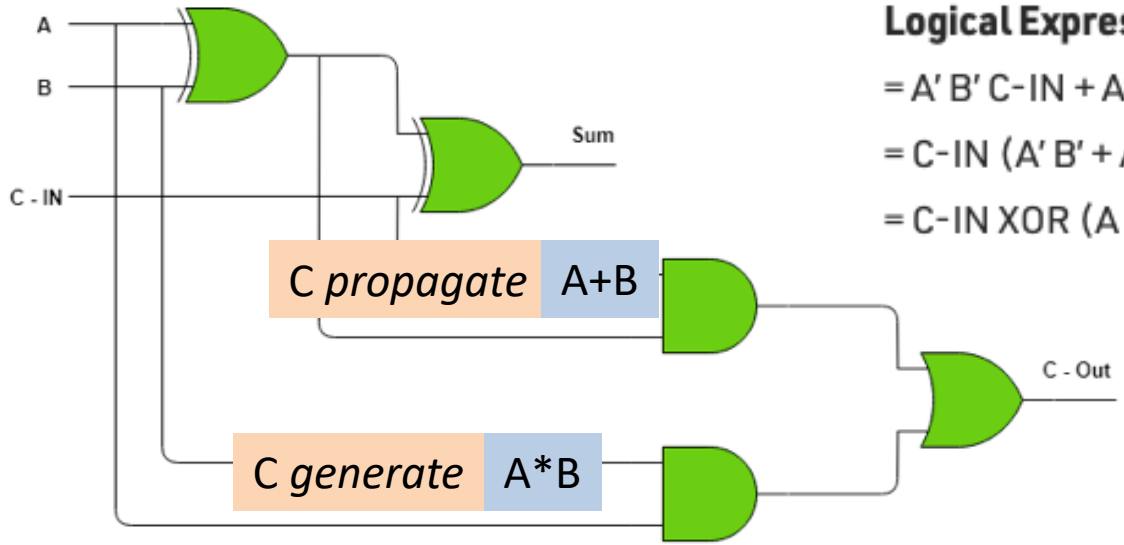


Adders

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

GeeksforGeeks

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



Logical Expression for SUM:

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

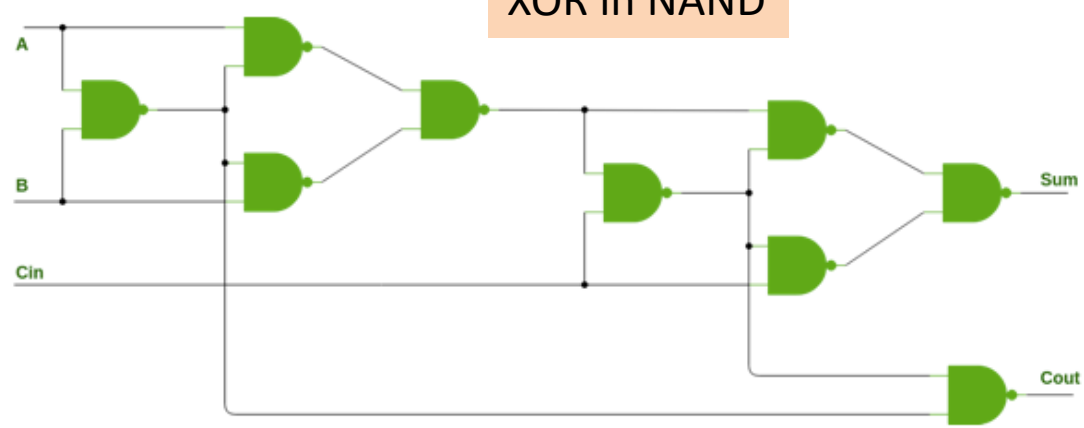
= C-IN (A′ B′ + A B) + C-IN′ (A′ B + A B′)

= C-IN XOR (A XOR B)

Full Adder logic

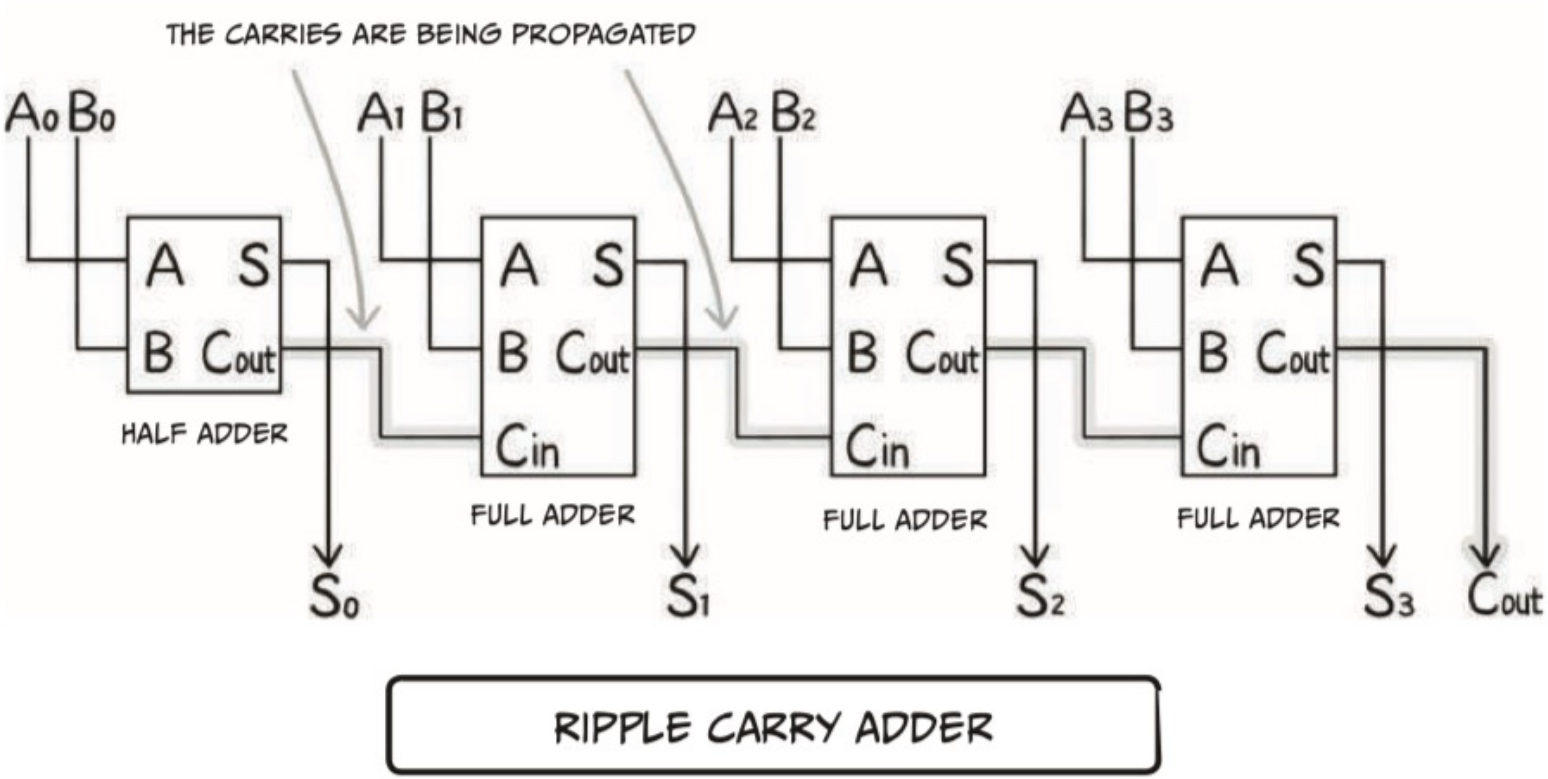
Implementation of Full Adder using NAND gates:

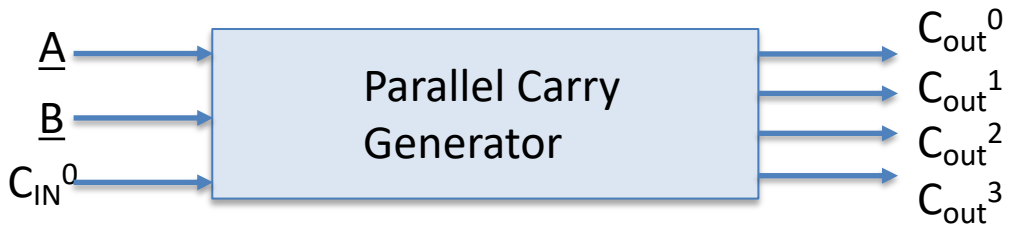
XOR in NAND



Adders: Ripple Carry

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





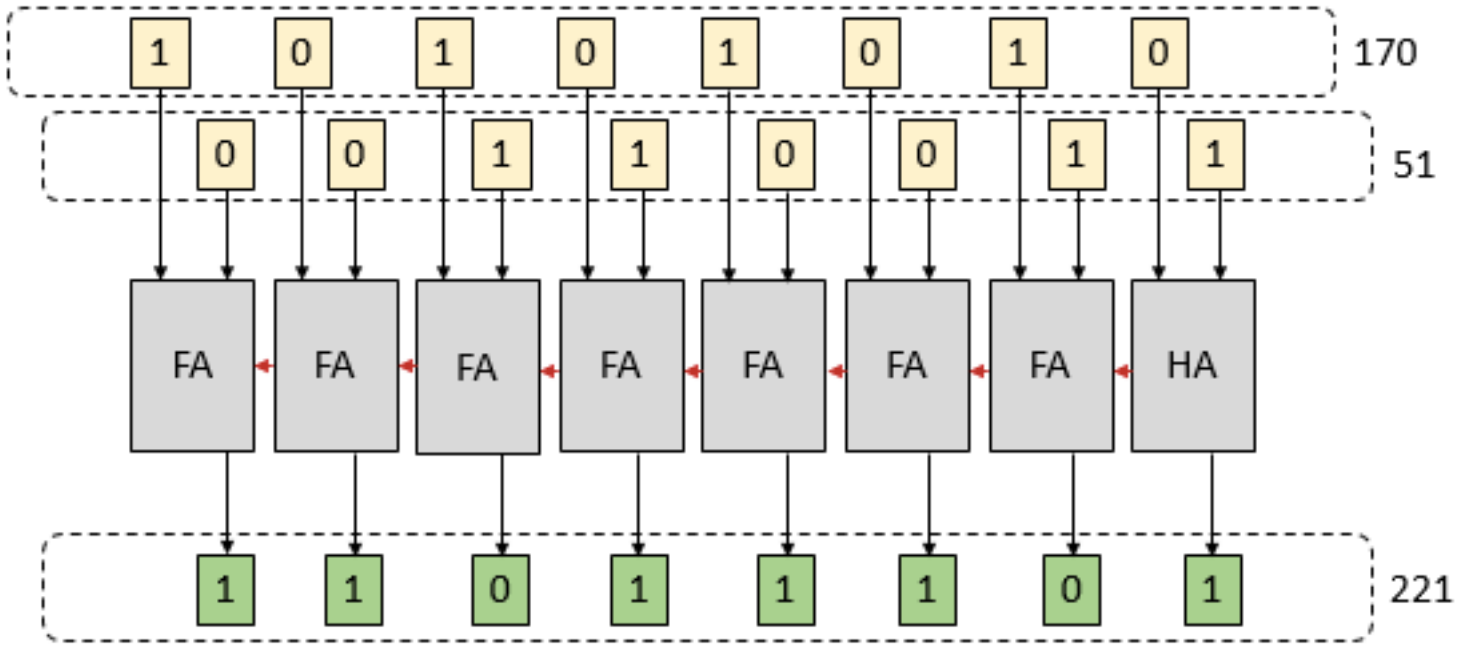
Adder

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

WOLFRAM

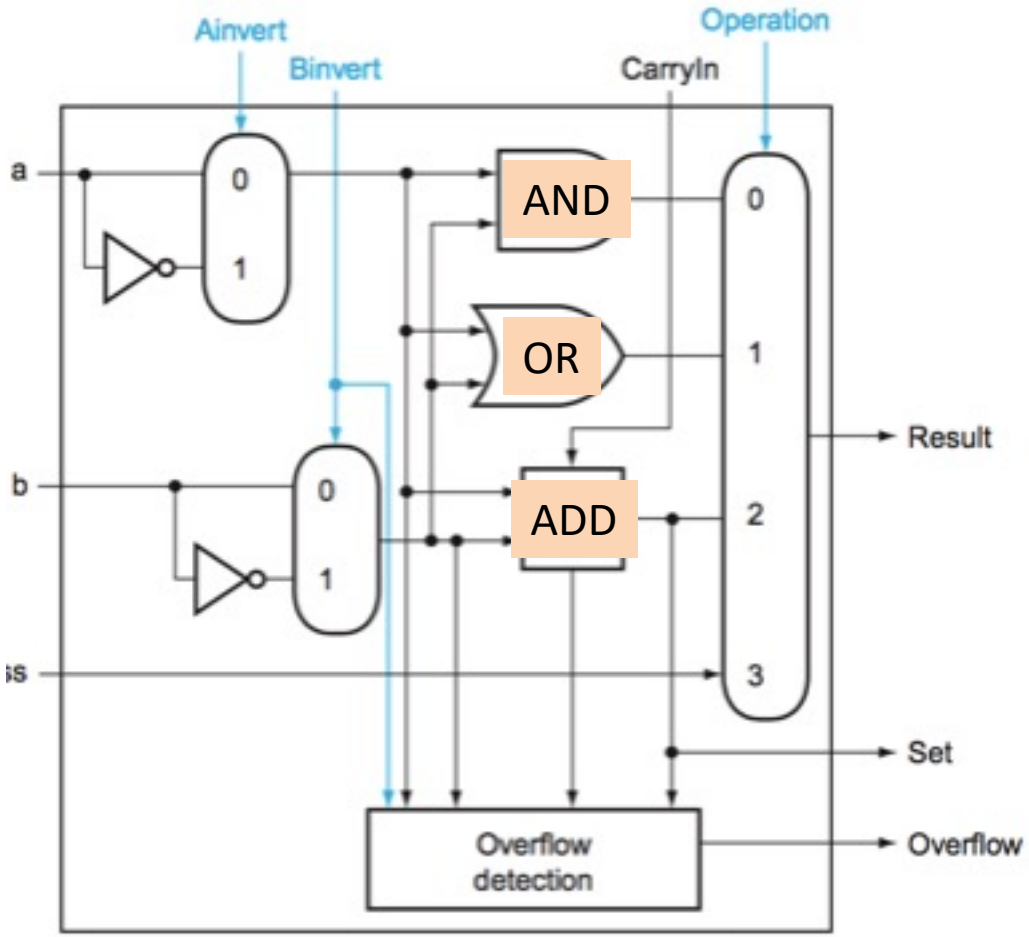
The 8-Bit Adder Principle

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

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



MIPS ALU in Verilog

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

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
```

Section

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

AMD Products (1971)

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

Electronic Device Cos.

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

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

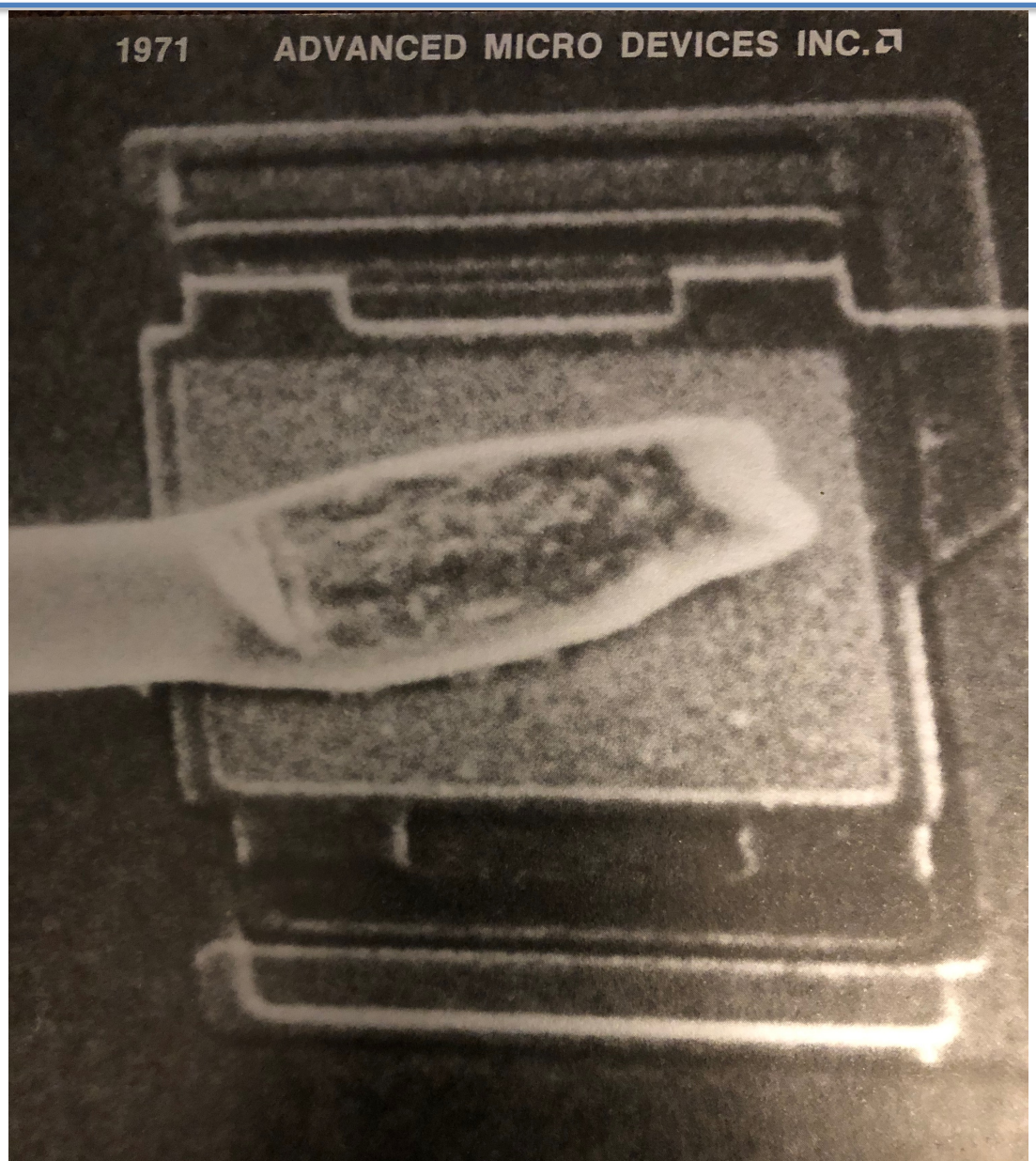
WESTINGHOUSE ..... Semiconductors

STOCKING INVENTORY

IN EVERY LOCATION

AMD Catalog 1971

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



AMD Analog (Linear)

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

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

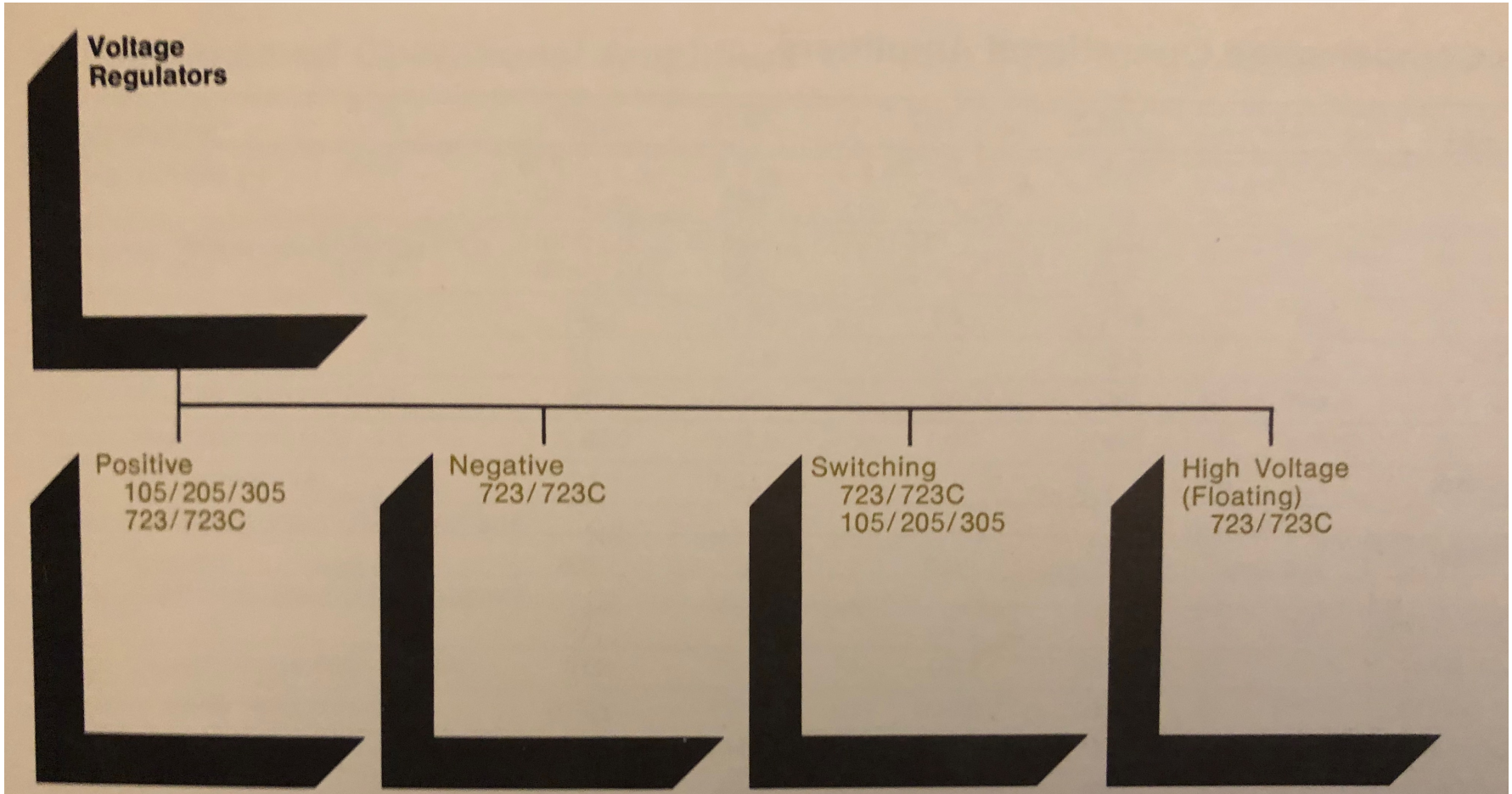
The primary applications for these circuits are where internally

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)

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

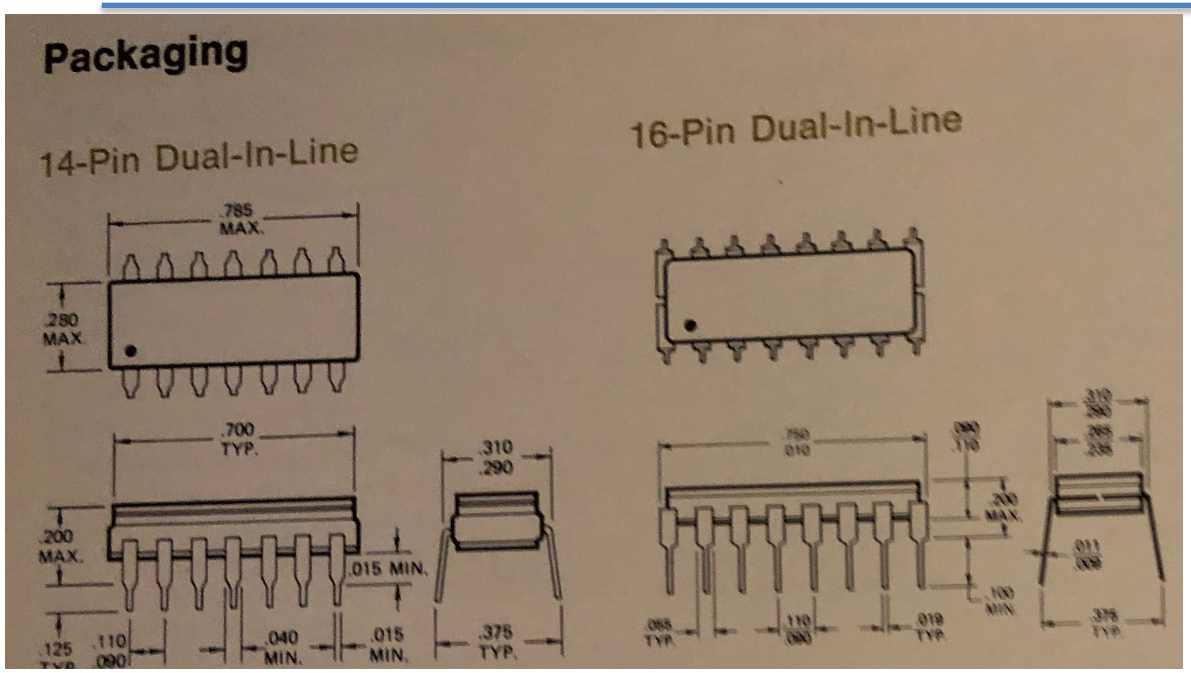


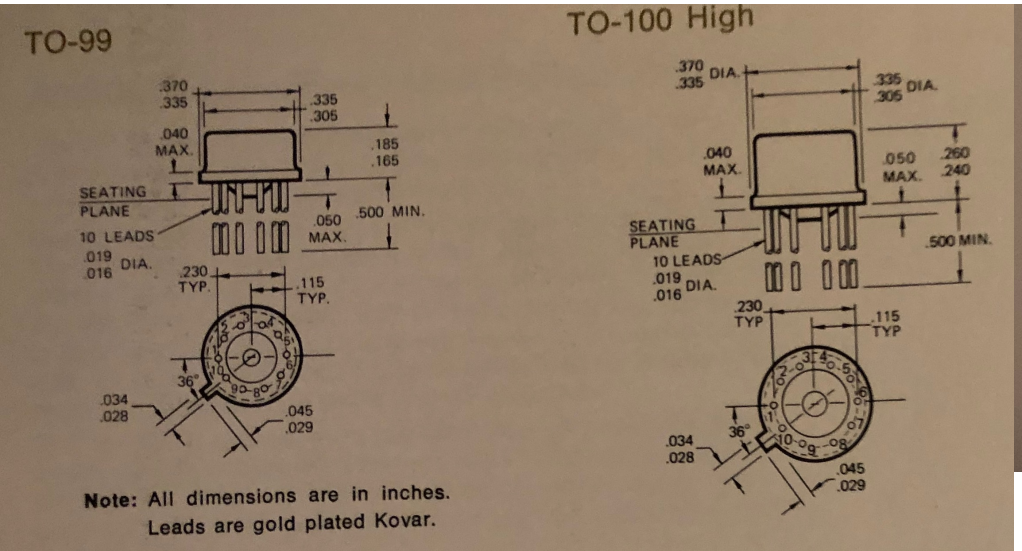
The monolithic regulator is considered useful for local-board regulation, amplifier supplies, and other applications where the size and economics are of advantage. Monolithic regulators should be considered in conjunction with external-pass transistors when power out is important. Their primary advantage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

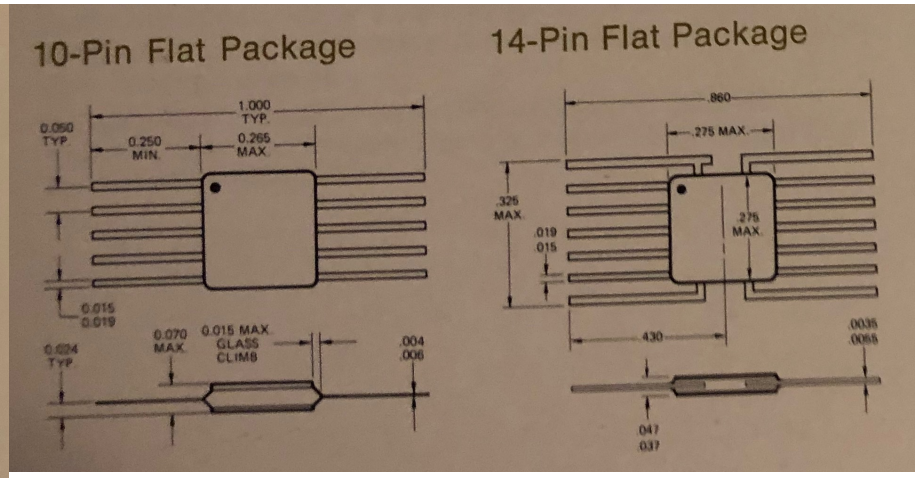
Precision regulation when external referer devices are used.

AMD Packages

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







Section

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

Logic

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

AMD Digital MSI

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

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*

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

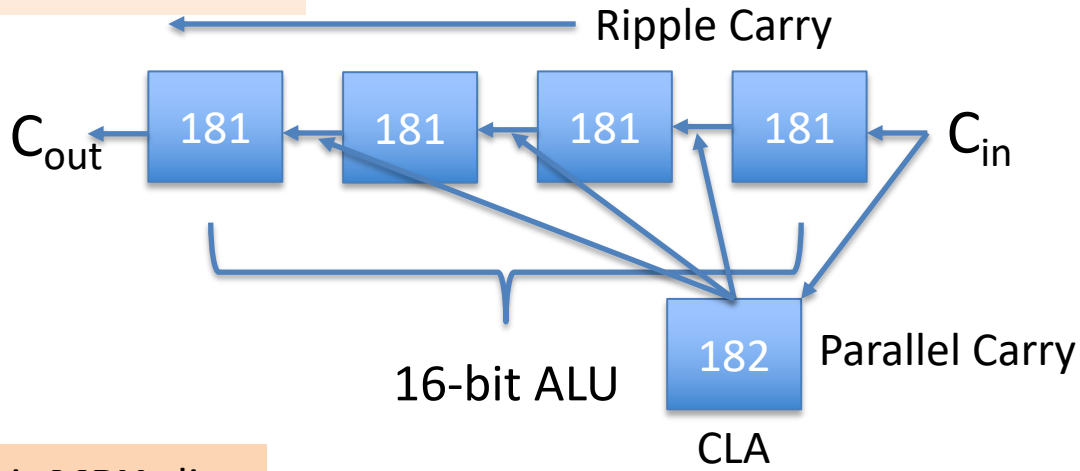
Bit-slice 1965-75

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

Texas Instruments

- 54/7400 Series
  - 54/74SN**181**
    - 4-bit **ALU** slice

- 54 → **Mil** temp
- 74 → **Com'l** temp



Am2505 2x4-bit **Multiplier** slice

- 4-bit **MPU** slice

Replaced by **Am2900** *family* → Am29**01** ALU + Register file *microprogrammed*

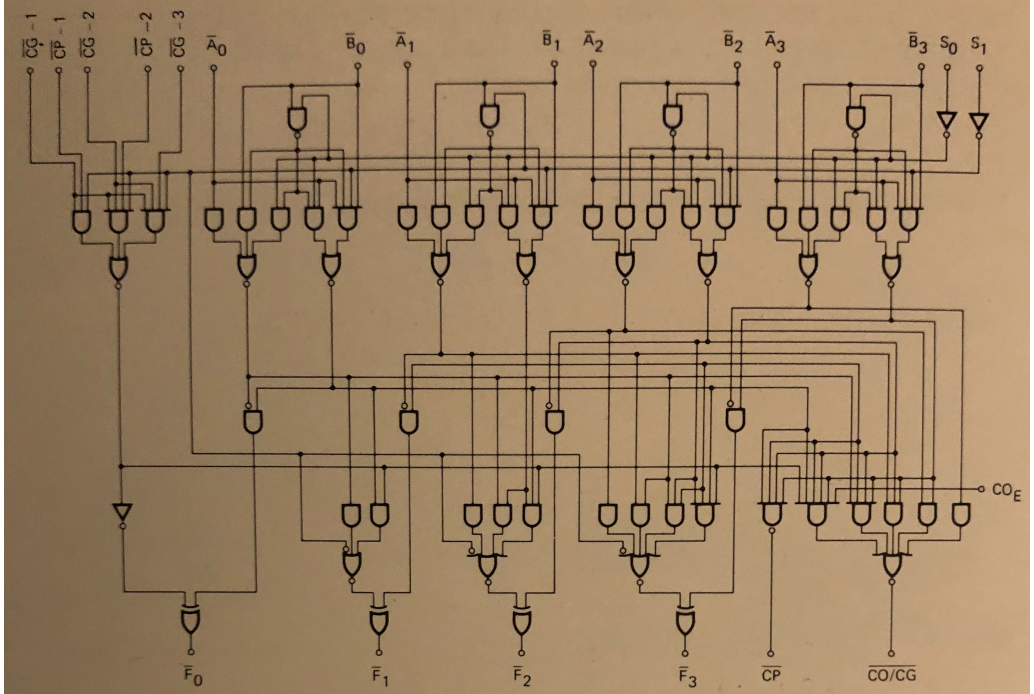
Am29**10** *Microprogram sequencer*

Am29**02** CLA

AMD ALU – Am9340

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

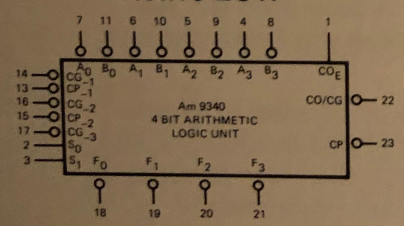
Logic Diagram/Symbols



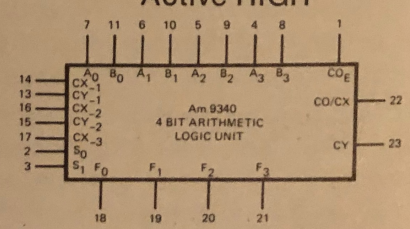
Characteristics

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



Active HIGH

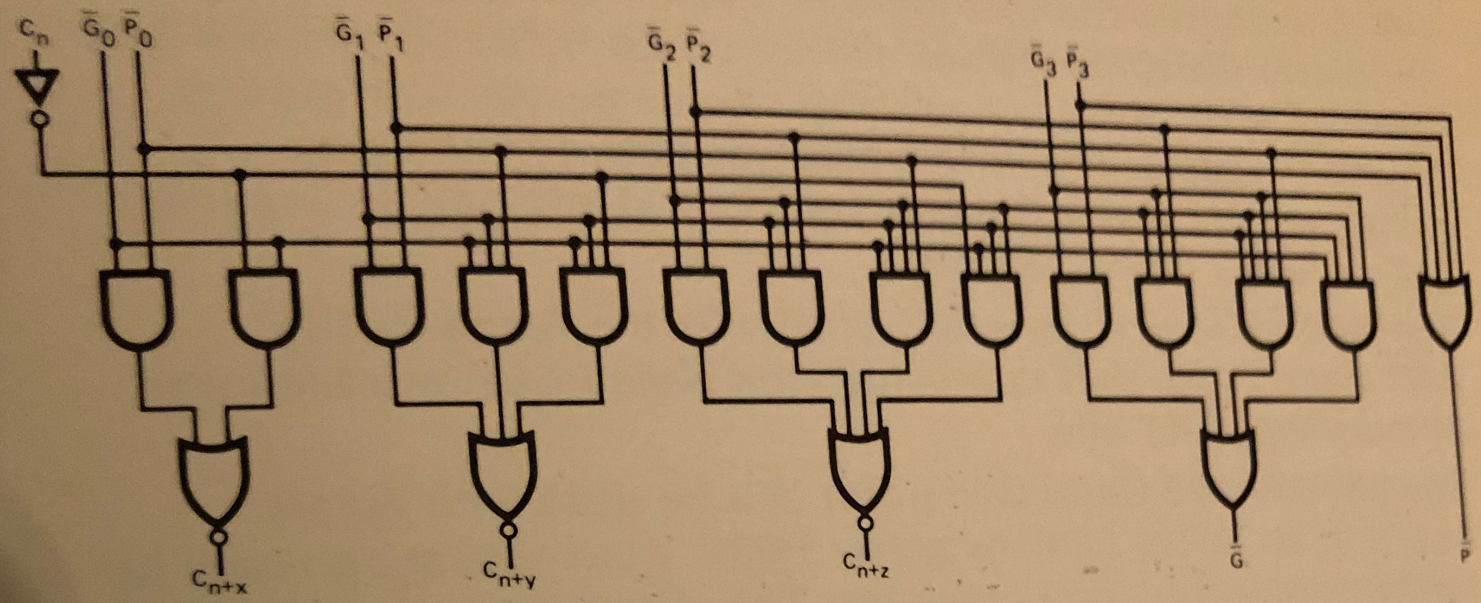


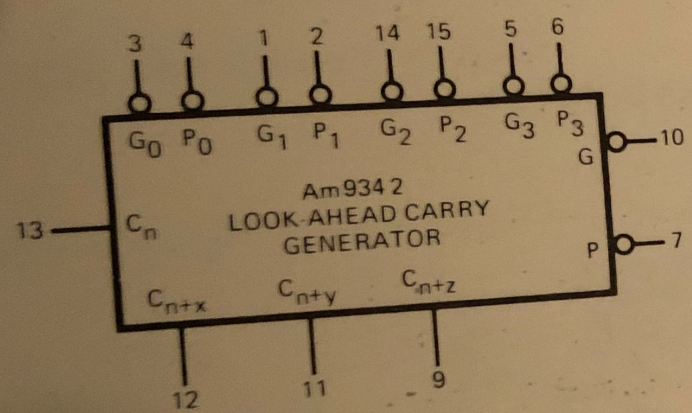
V_{CC} = Pin 24
GND = Pin 12

AMD CLA – Am9342

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

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

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

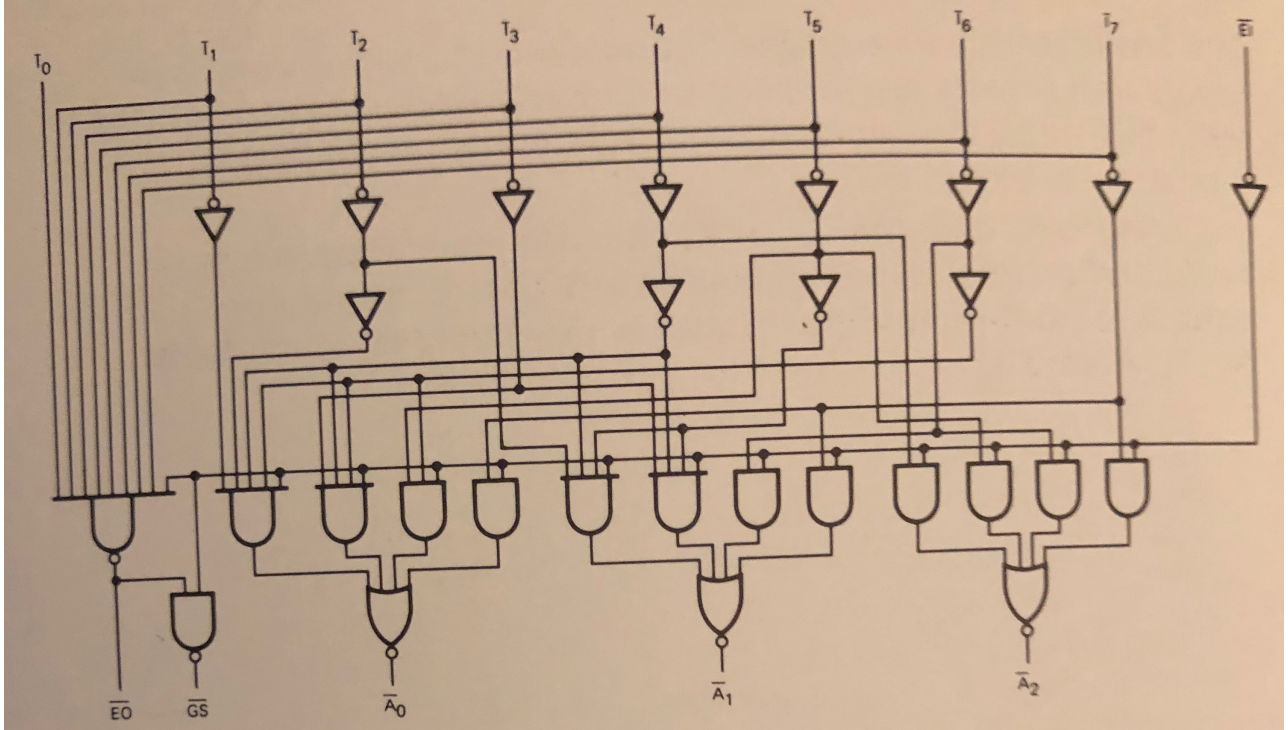
Logic

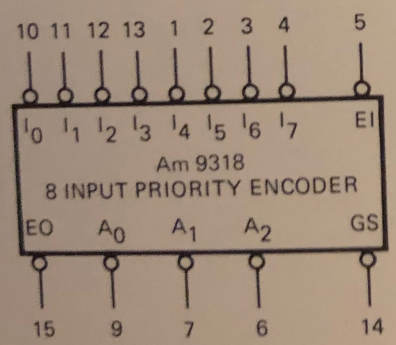
- PIC
- Muxes
- Decoders
- Logic functions

PIC: Priority Interrupt Enc

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

Logic Diagram/Symbol





Characteristics

Typical Delay $\overline{I}$ to $\overline{A}$ 16 ns

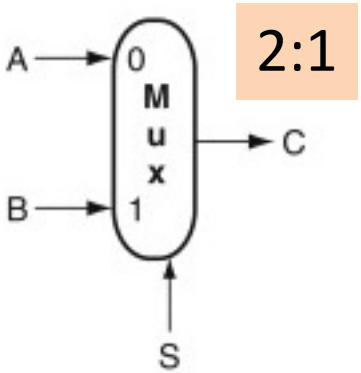
Typical Power Dissipation 250 mW

V_{CC} = Pin 16

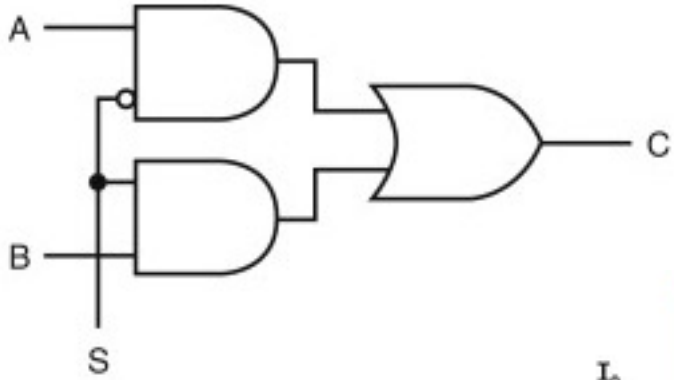
GND = Pin 8

MUX

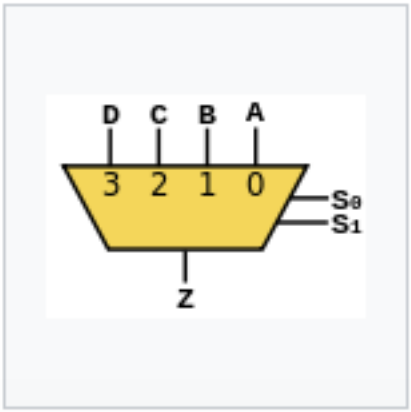
DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023



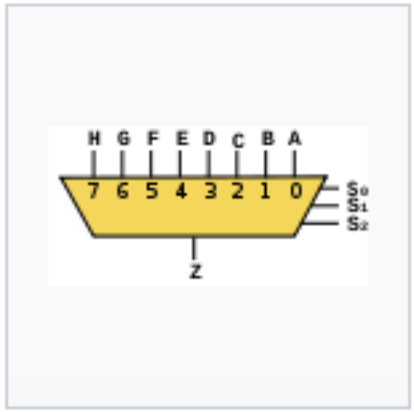
2:1



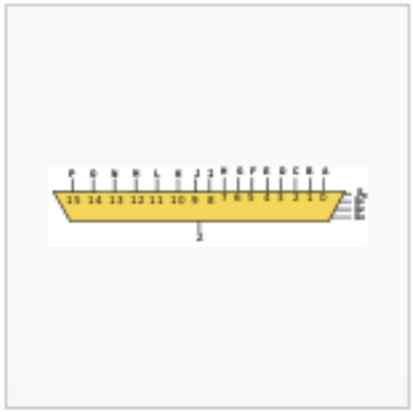
The following 4-to-1 multiplexer is constructed from 3-state buffers and AND gates



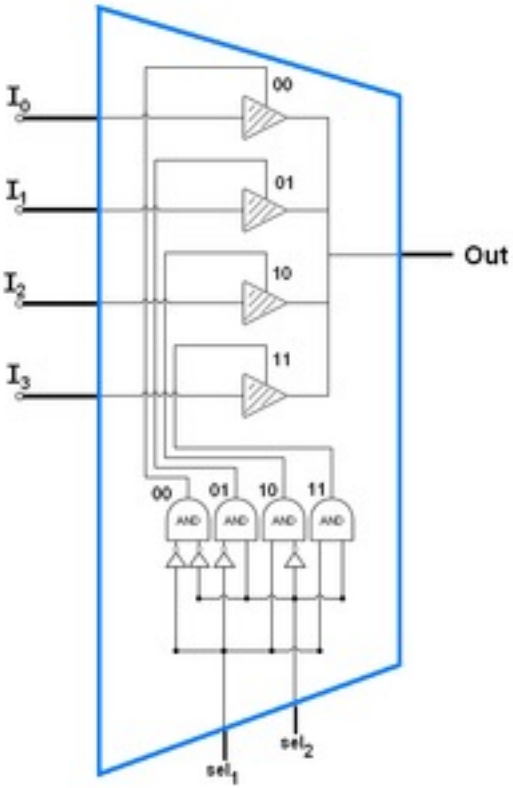
4-to-1 mux



8-to-1 mux

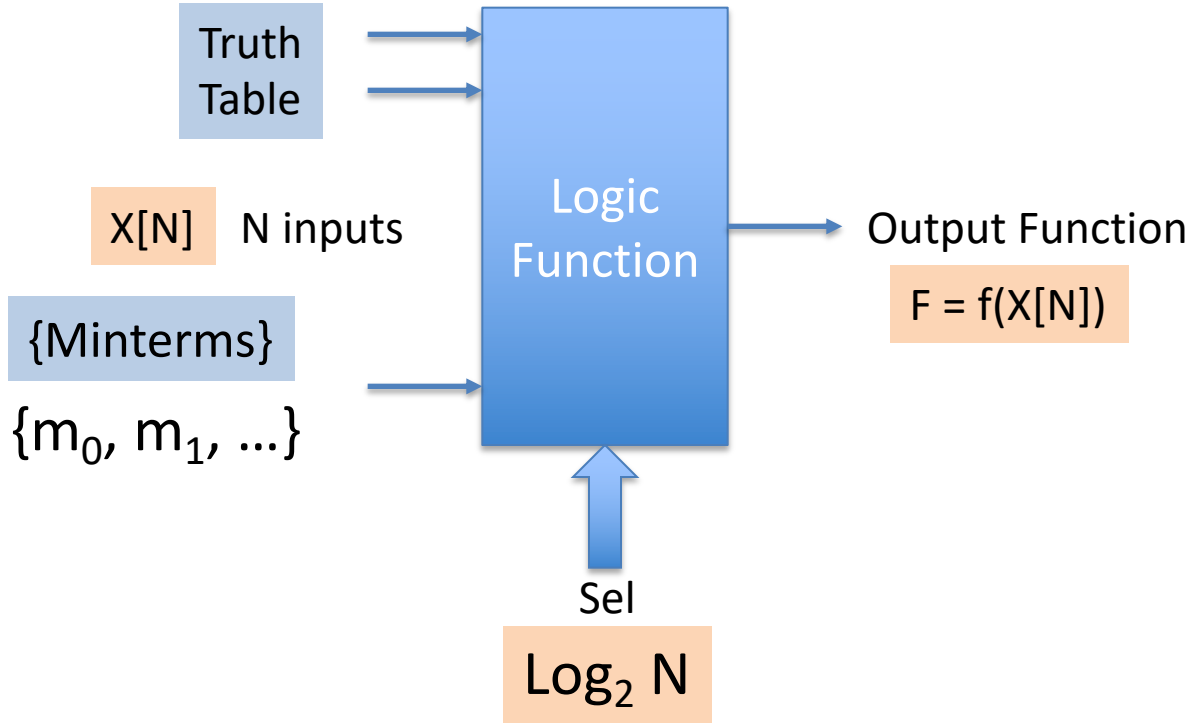


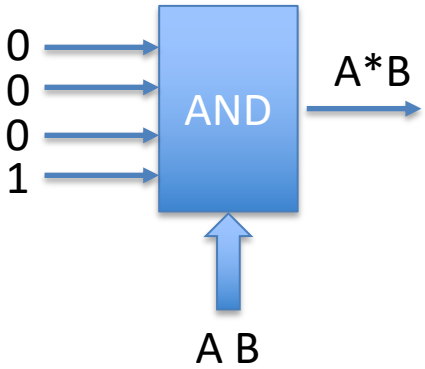
16-to-1 mux

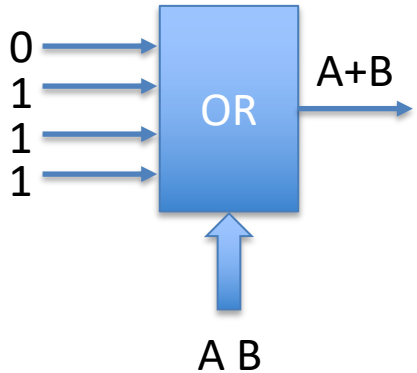


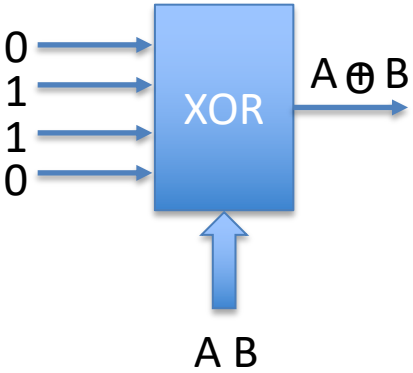
MUX as Function Generator

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





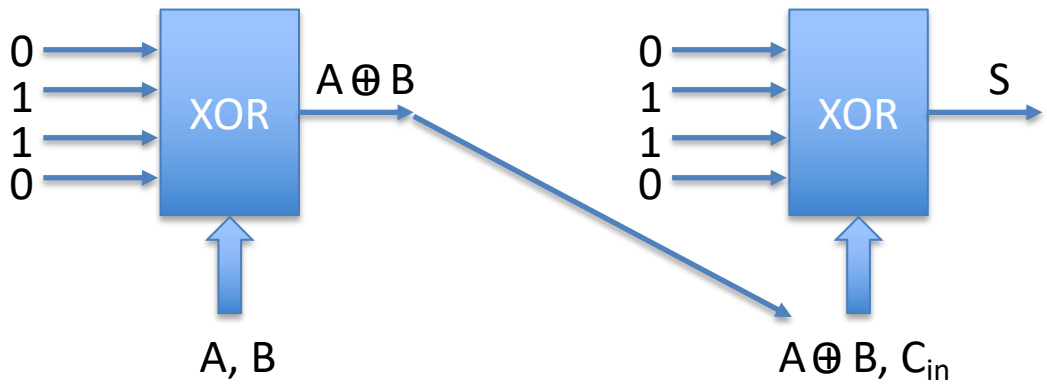




Full Adder via Mux

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

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



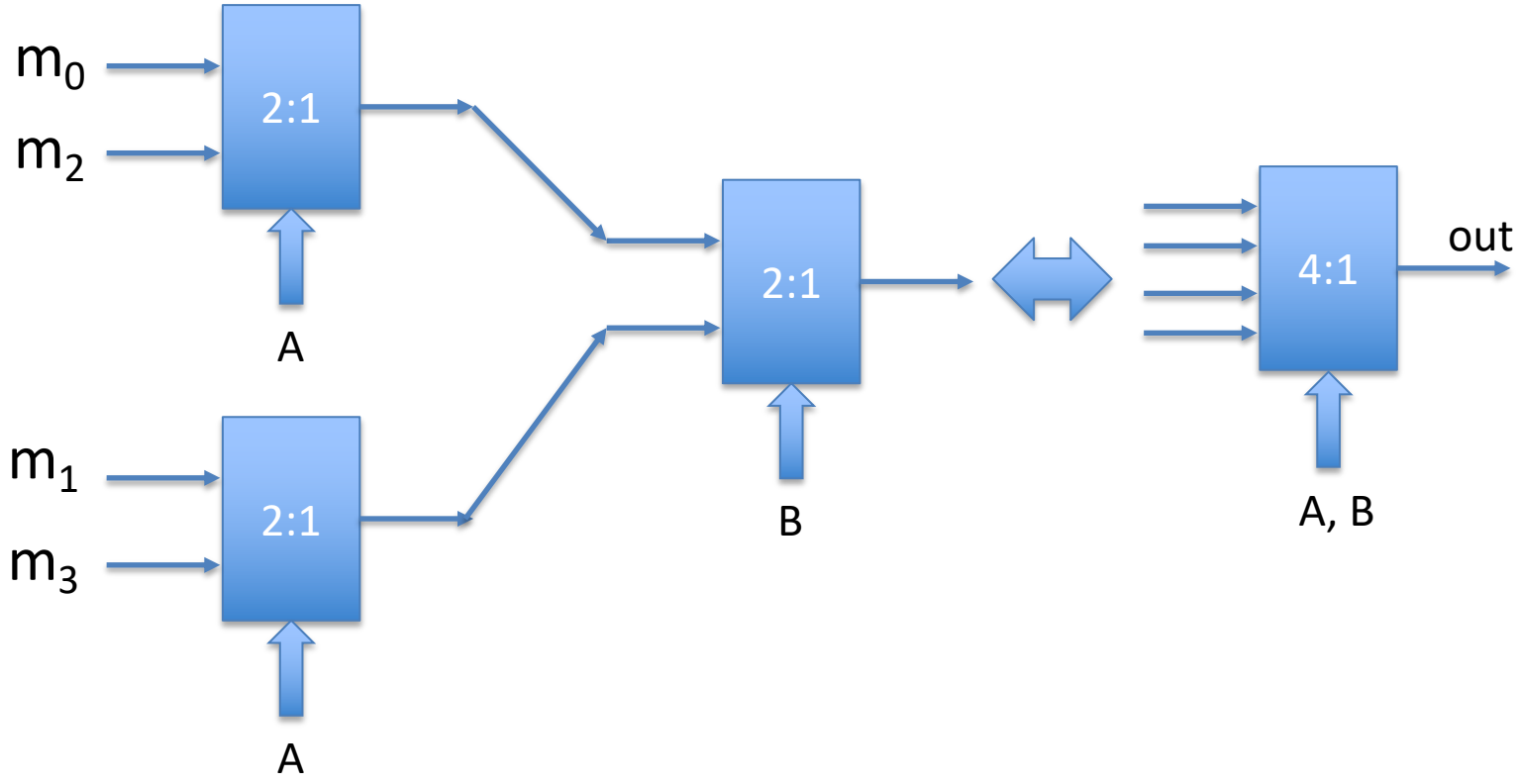
Mux Extension

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

{Minterms}

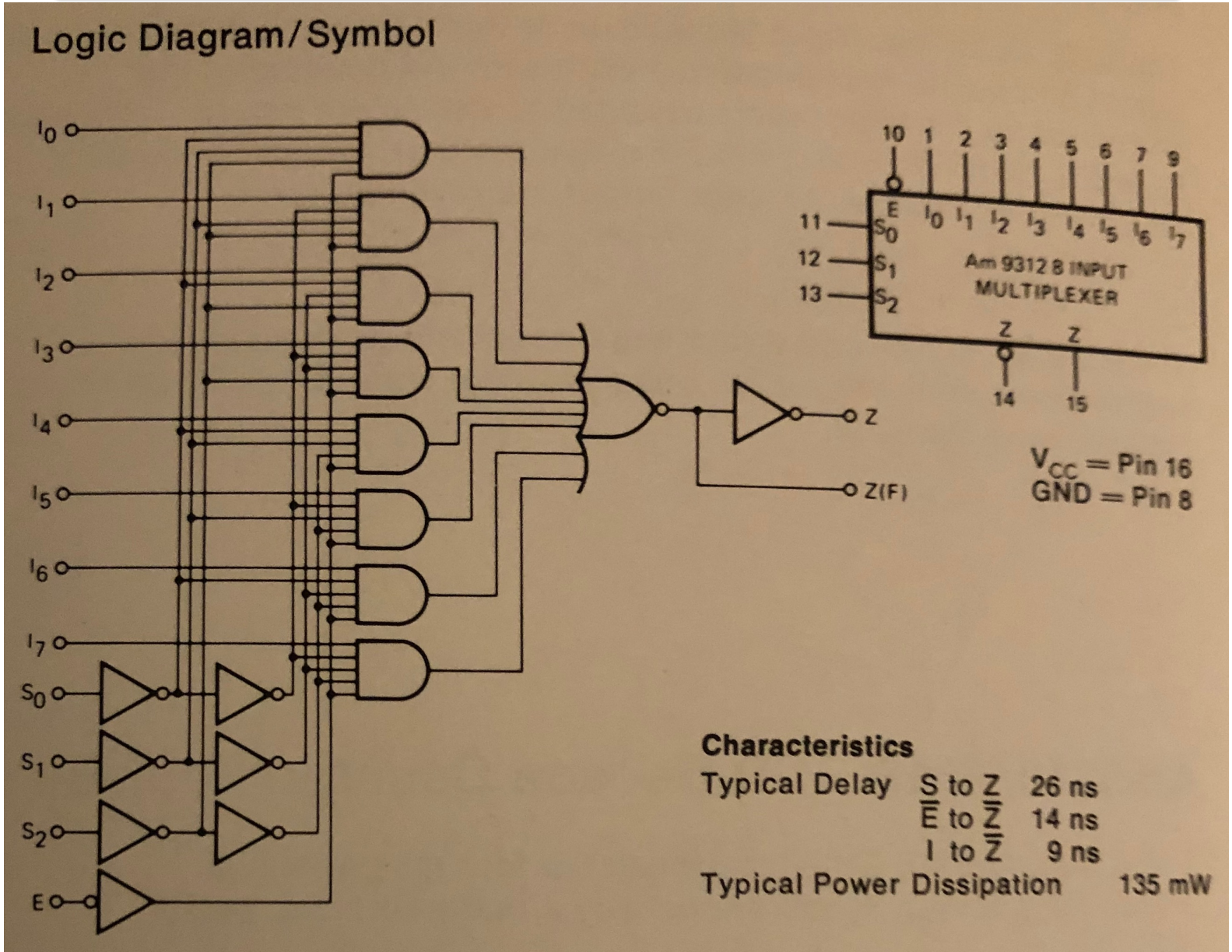
$\{m_0, m_1, \ldots\}$

2x 2:1 mux → 4:1 mux



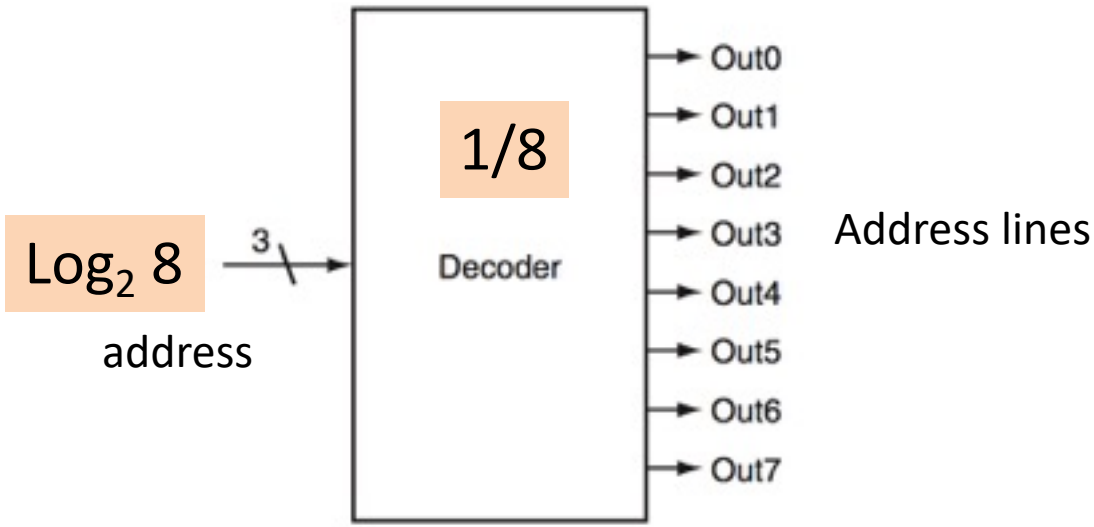
AMD 8:1 Mux

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

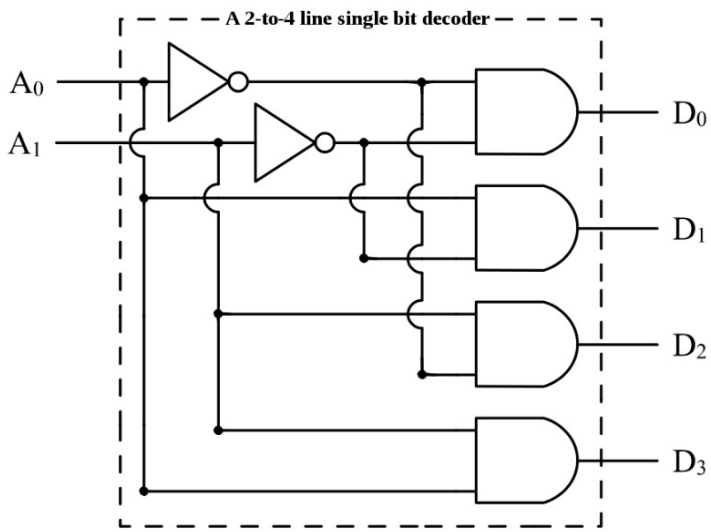


Decoder

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



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

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

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

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

$$D_3 = A_1 \cdot A_0$$

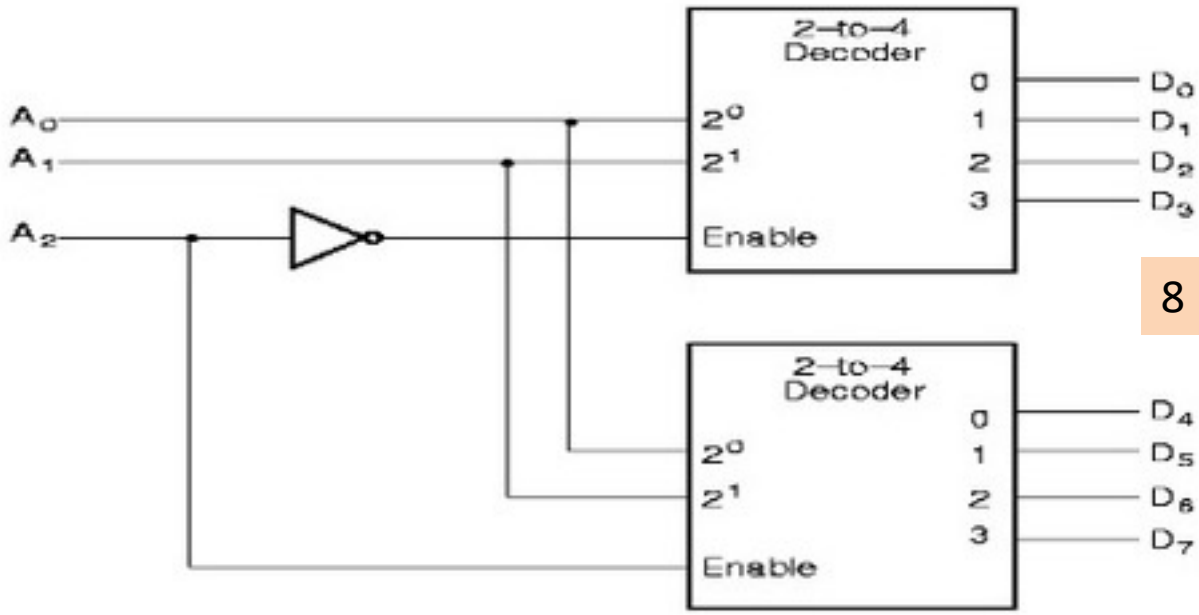
Decoder Expansion

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

DECODER EXPANSION

2x 2-to-4 → 3-to-8 (1 of 8)

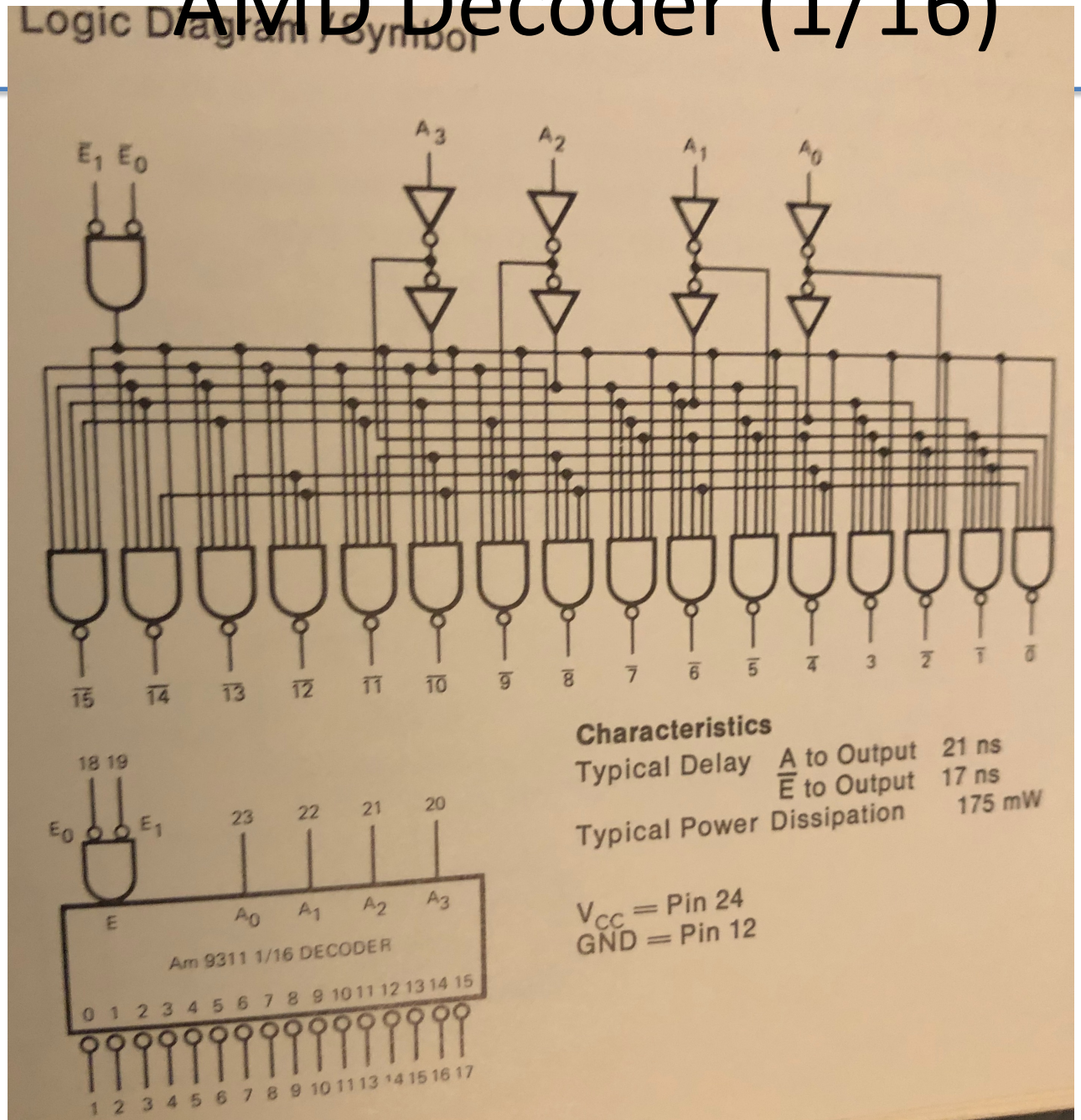
3



8

AMD Decoder (1/16)

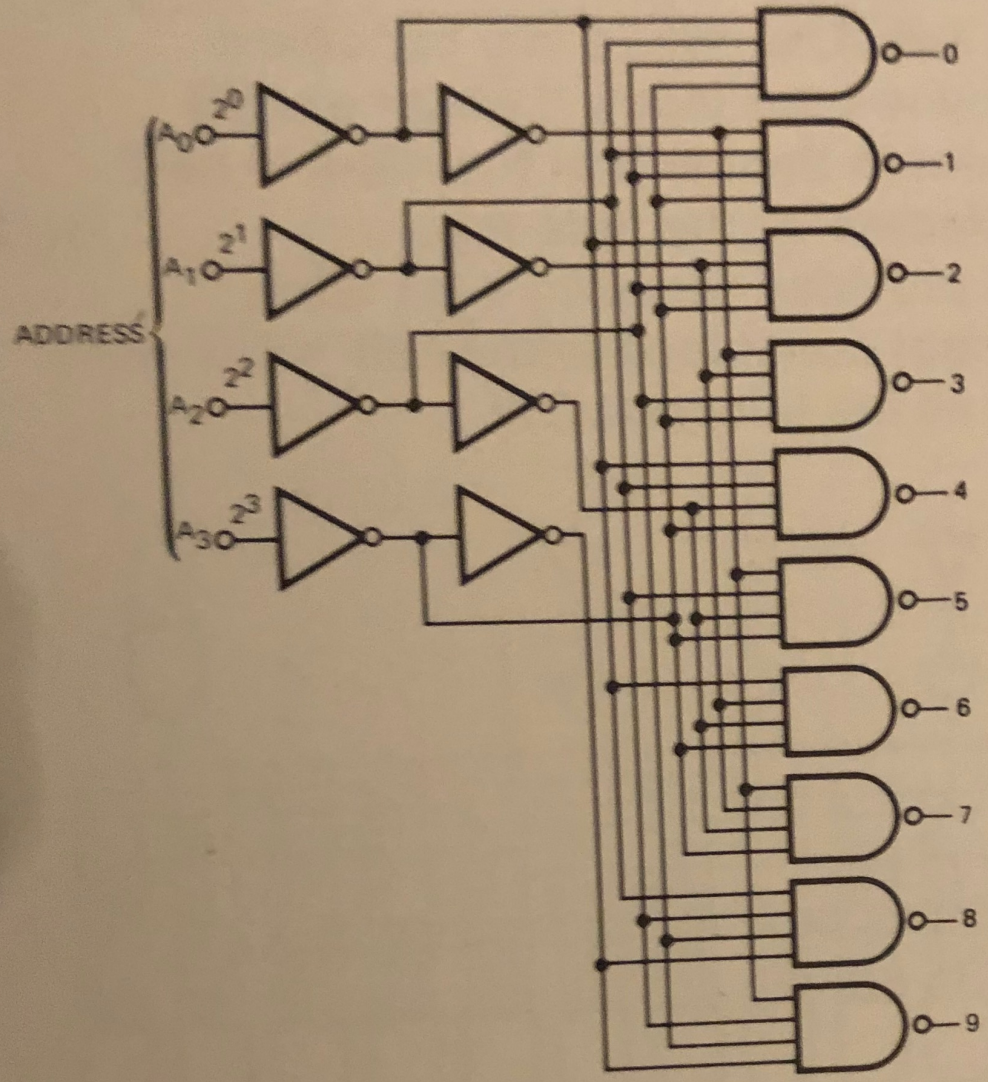
DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023

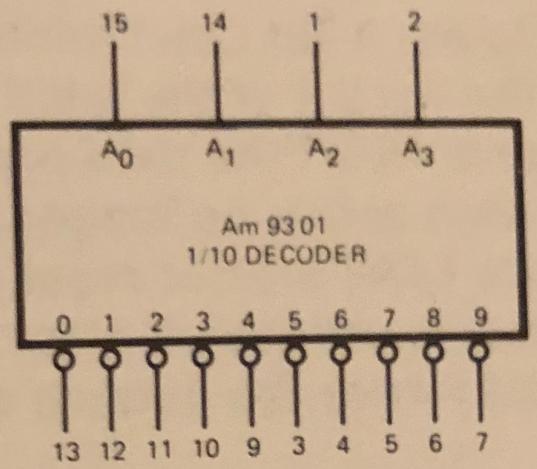


AMD Decoder (1/10)

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

Logic Diagram/Symbol





V_{CC} = Pin 16
GND = Pin 8

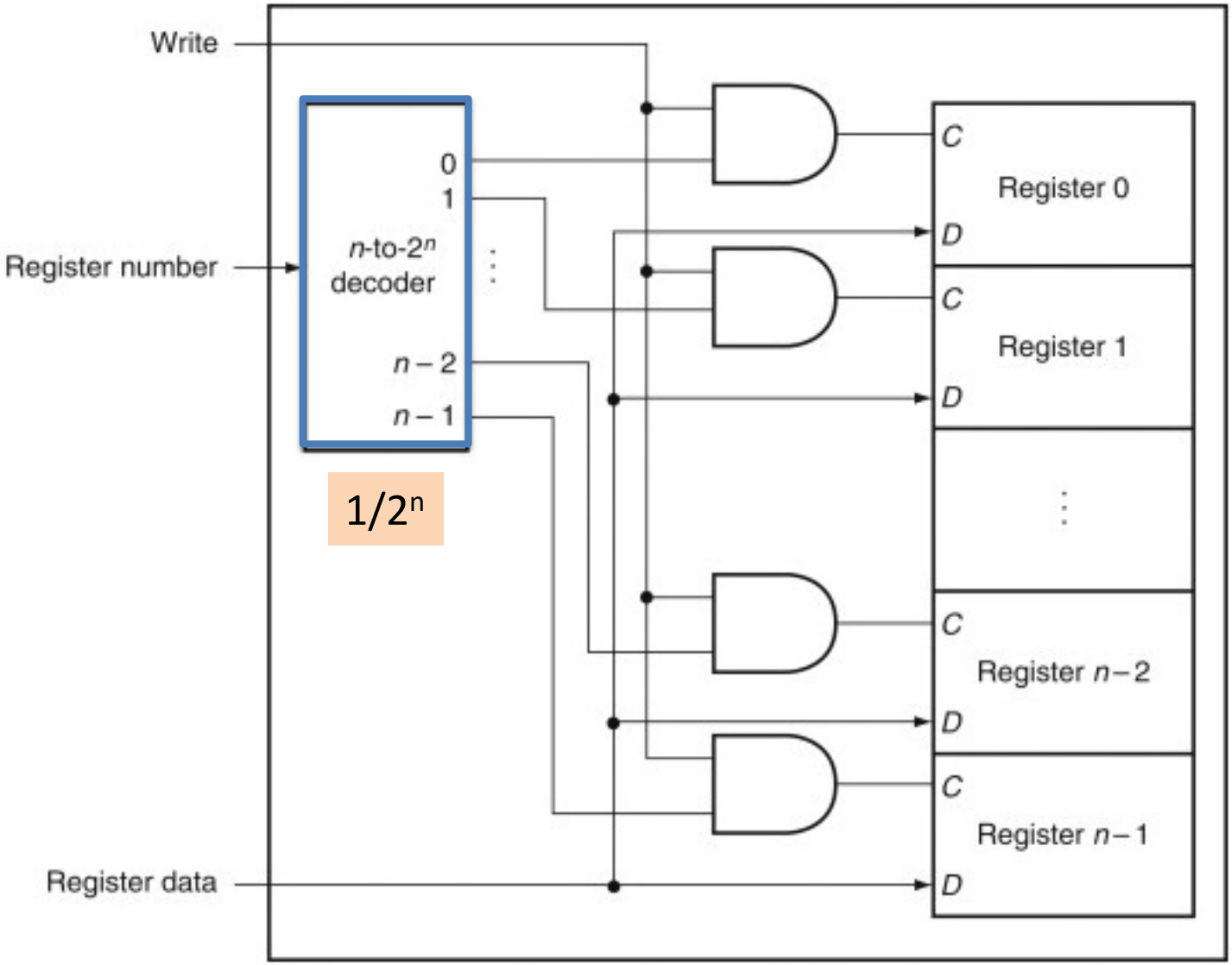
Characteristics

Typical Delay	A to Output	22ns
Typical Power Dissipation		145 mW

Register File Input Side

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

Decoders

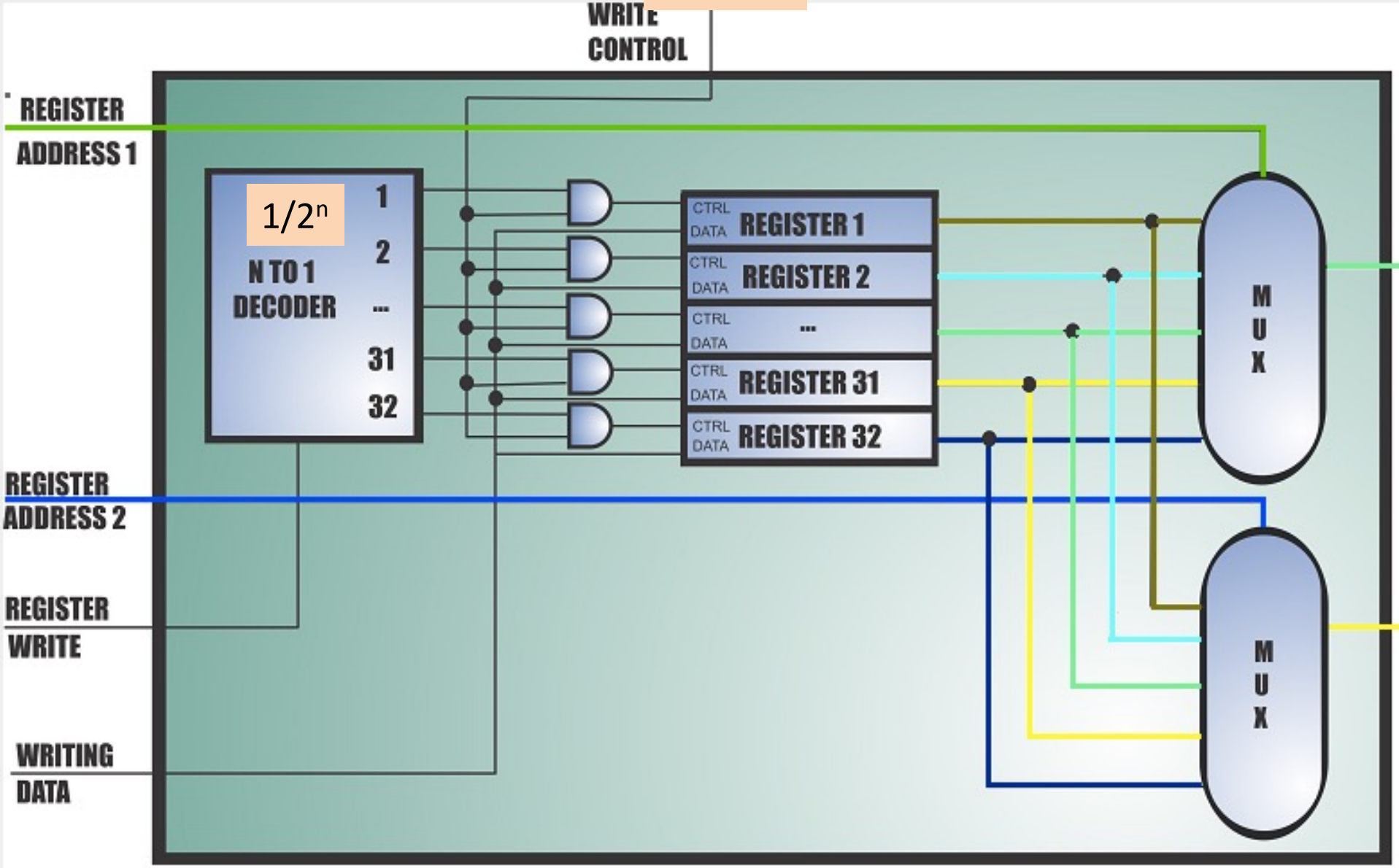


$1/2^n$

Register File on MARS

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

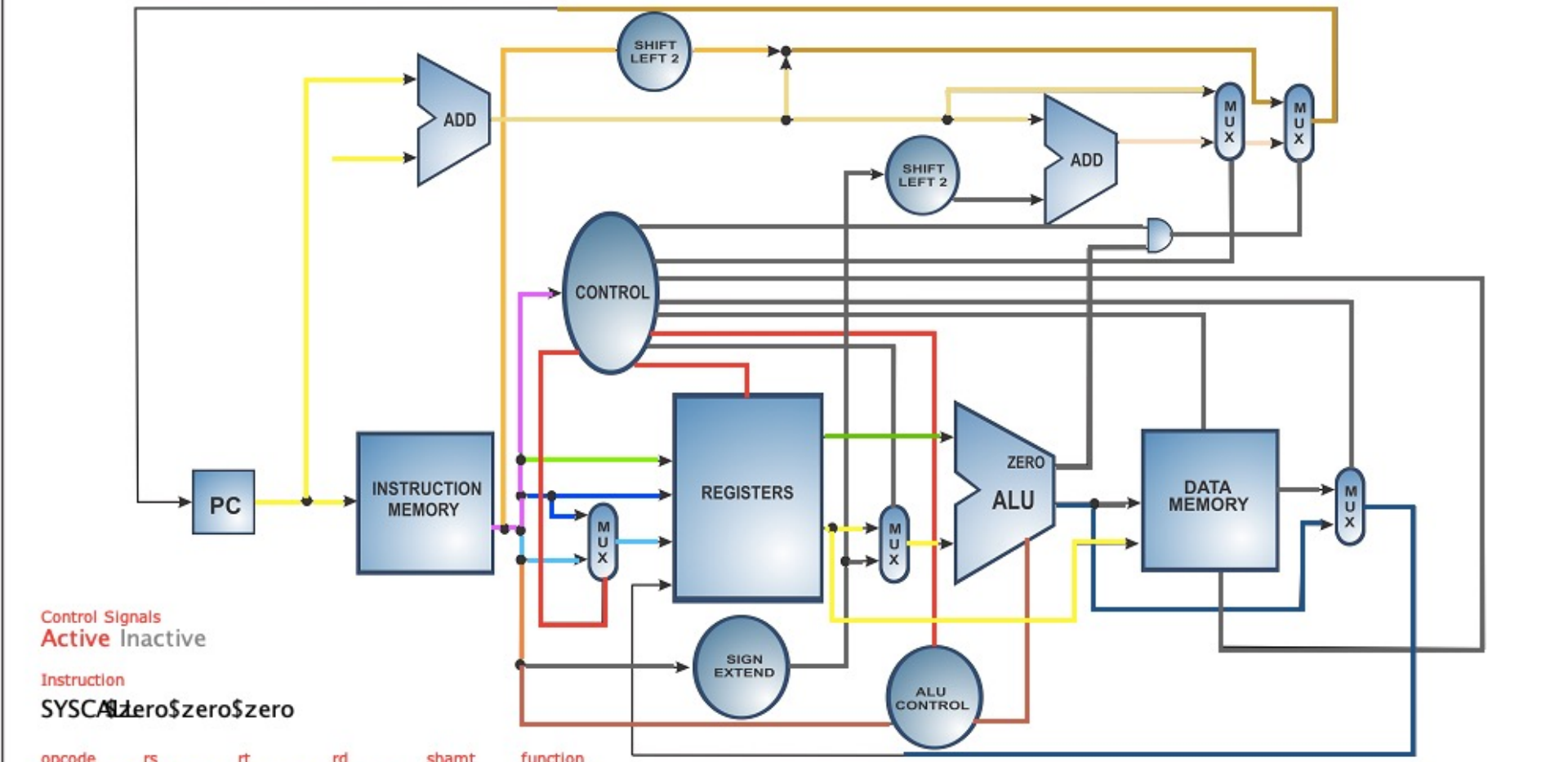
Decoders



MIPS on MARS

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

REGISTER TYPE INSTRUCTION



To see details of control units and register bank click inside the functional block

Logic Function Minimization

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

Combinational

- Quine-McKluskey
  - *Prime implicants*
  - *Essential* PI's

- Karnaugh ("K") Maps

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

	y	
x	m_0	m_2
	m_1	m_3

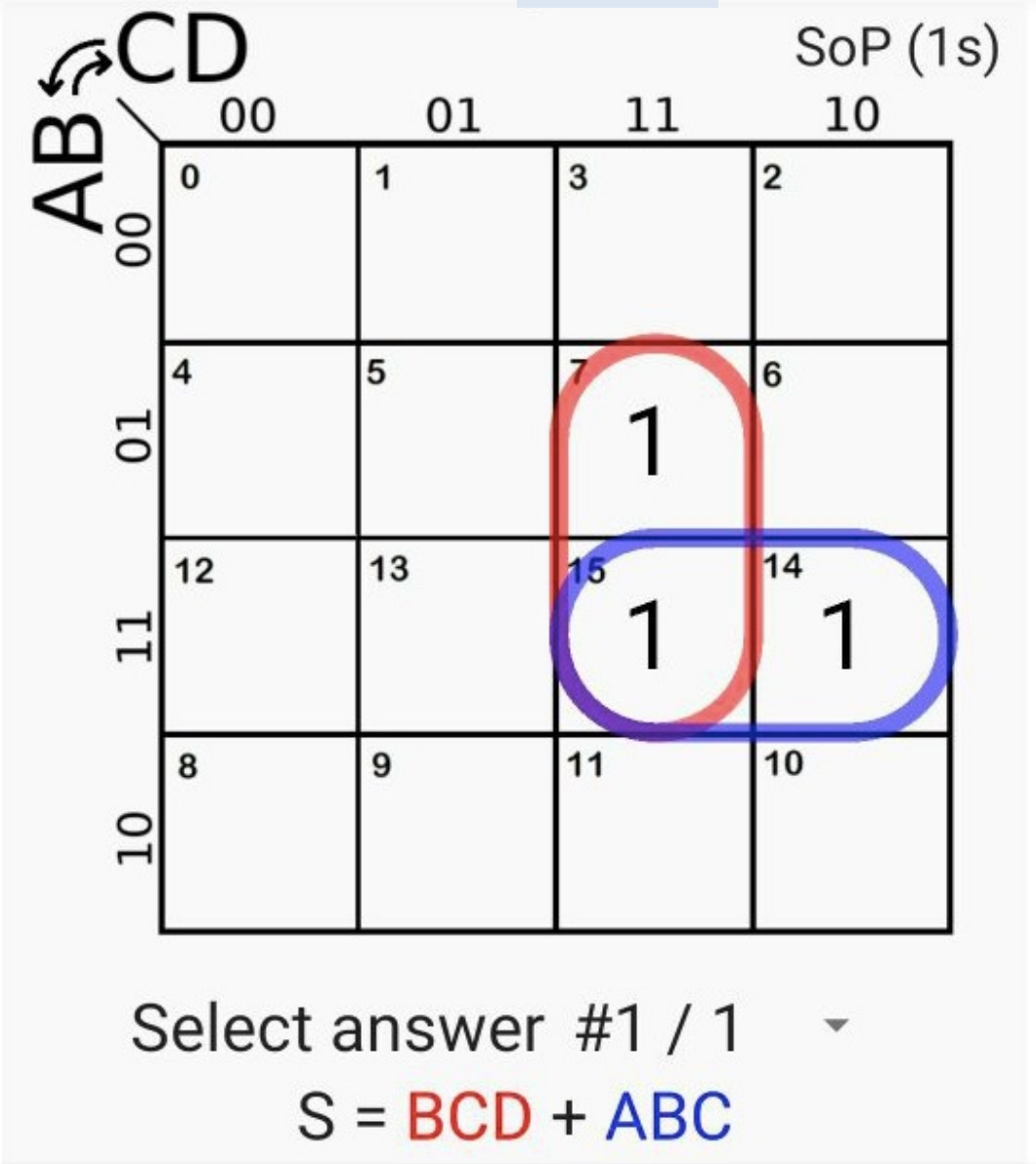
	y	
x	1	0
	1	0

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

K-Maps

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

Example



Gray codes

Logic distance=1

Section

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

Memory

- SRAM
- DRAM

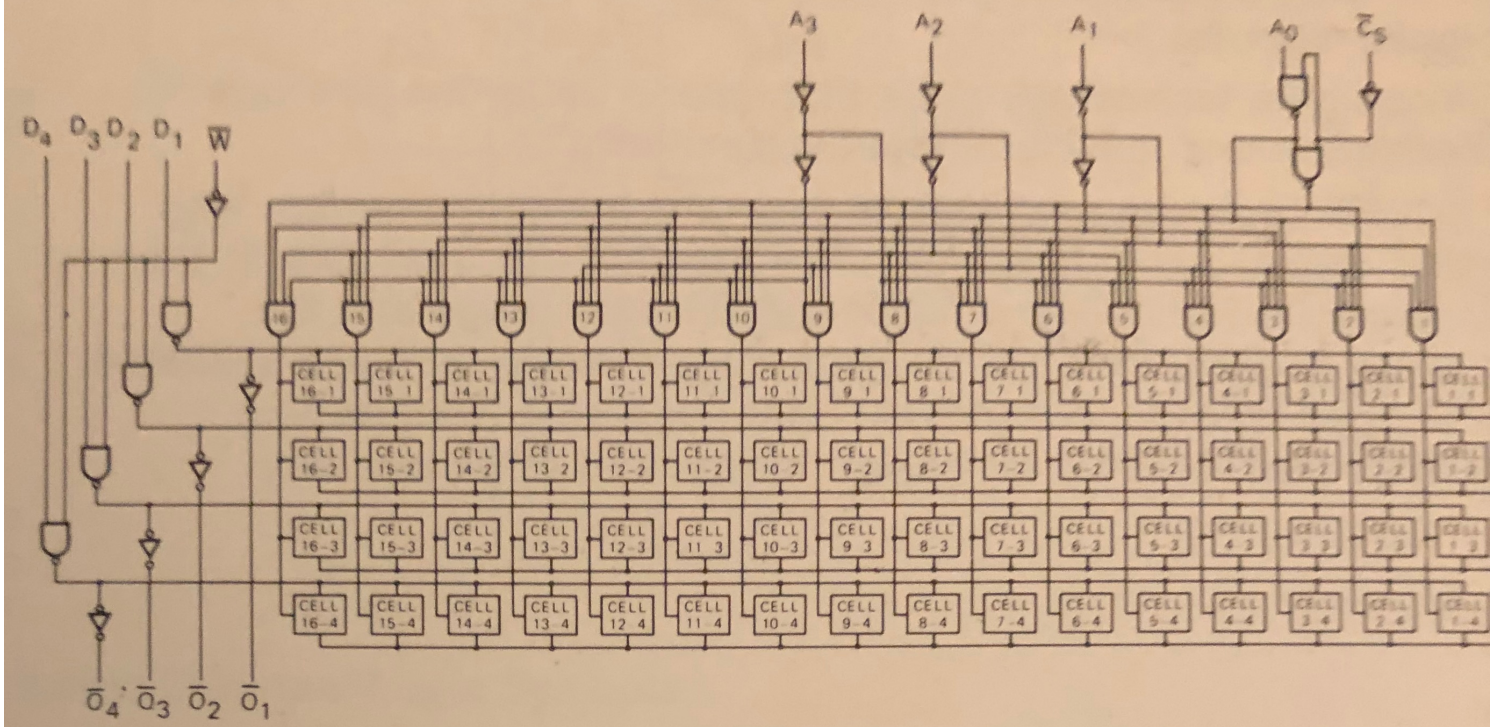
AMD 64-Bit Bipolar SRAM

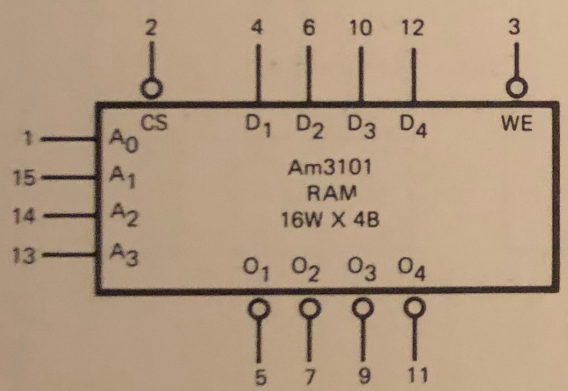
DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023

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

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

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:

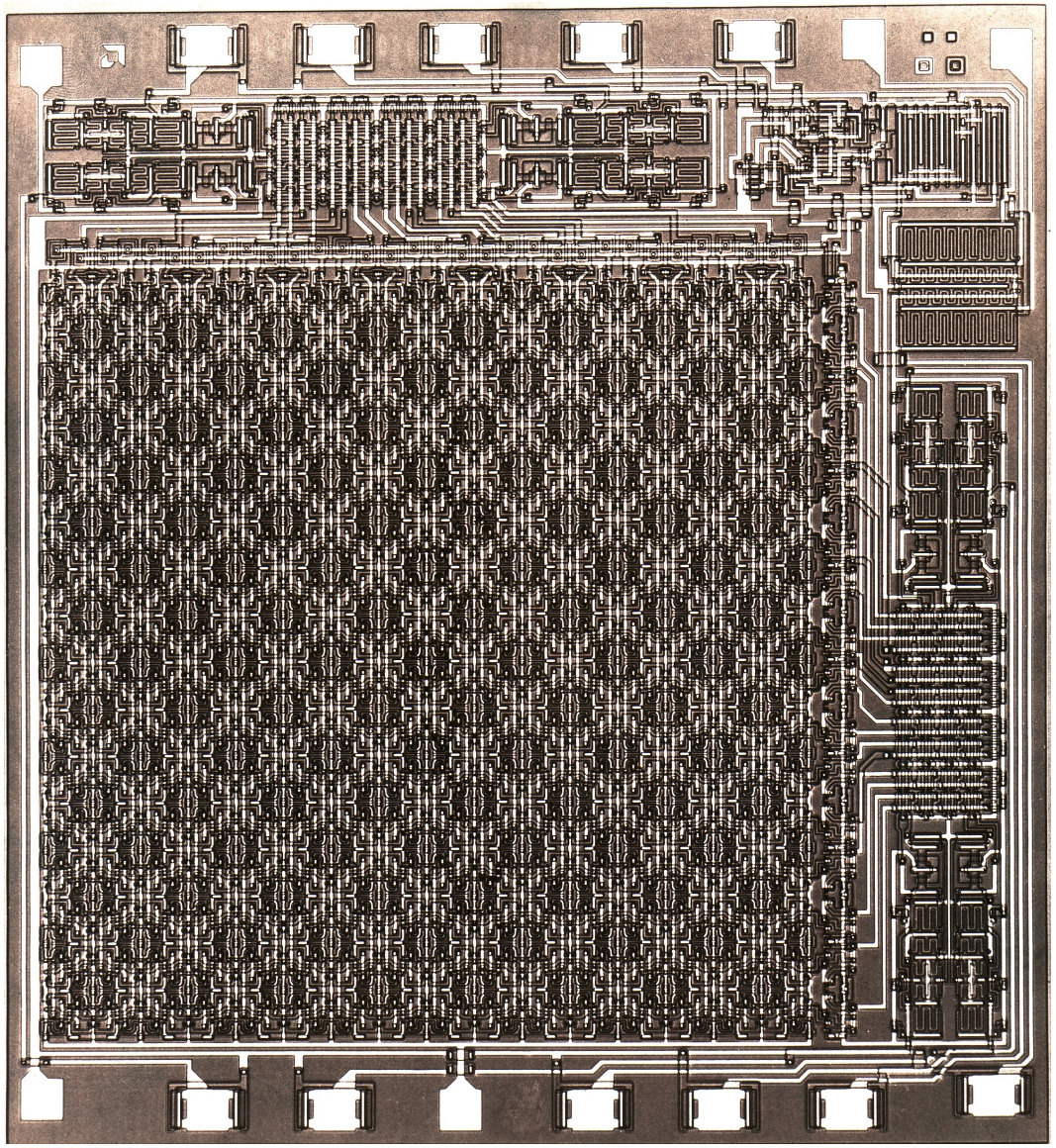
256-Bit Static Random Access Memory

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

1024-Bit Dynamic Random Access Memory

Am1101A 256x1 SRAM

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

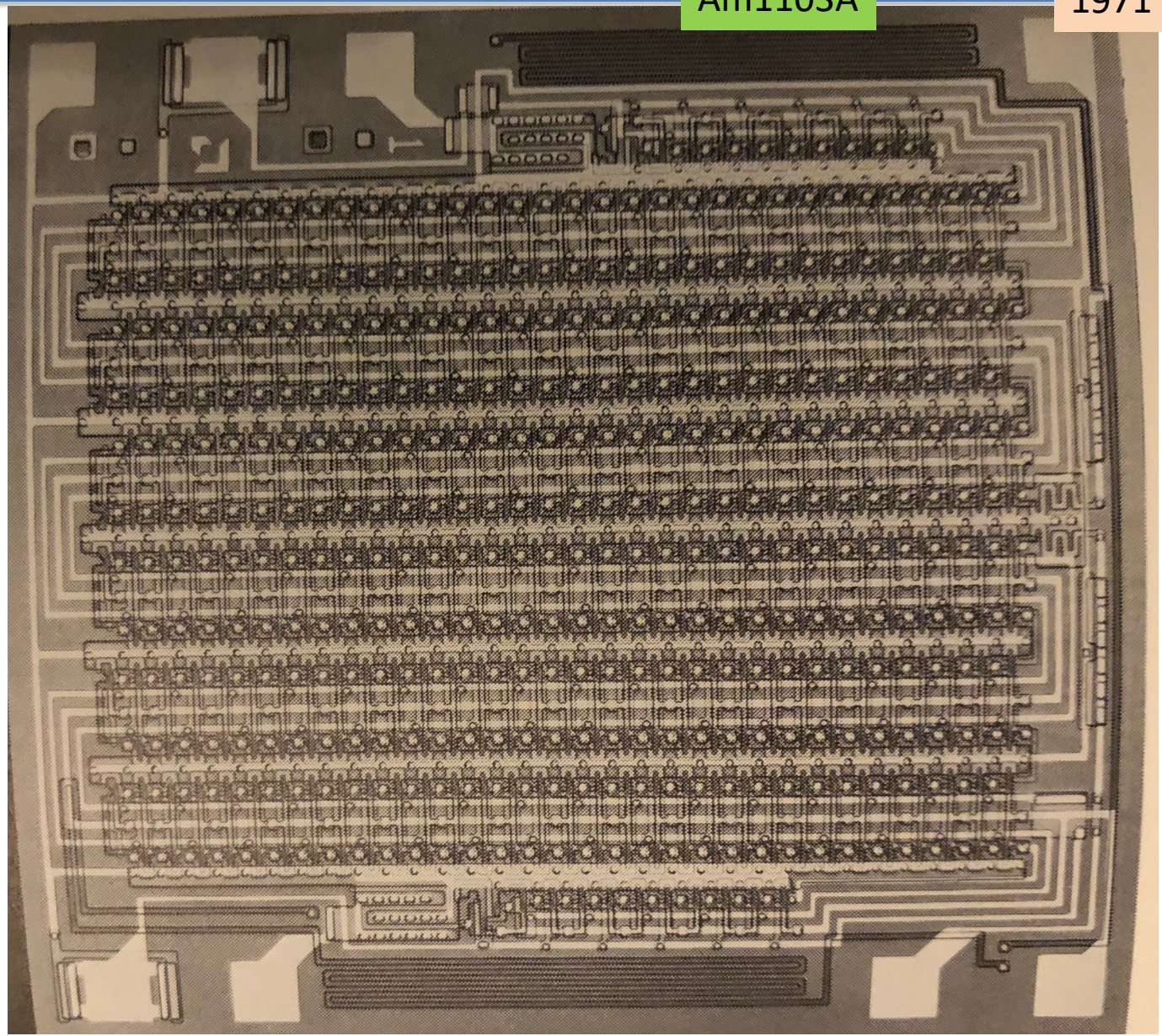


DRAM (AMD 1Kx1)

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

Am1103A

1971



Section

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

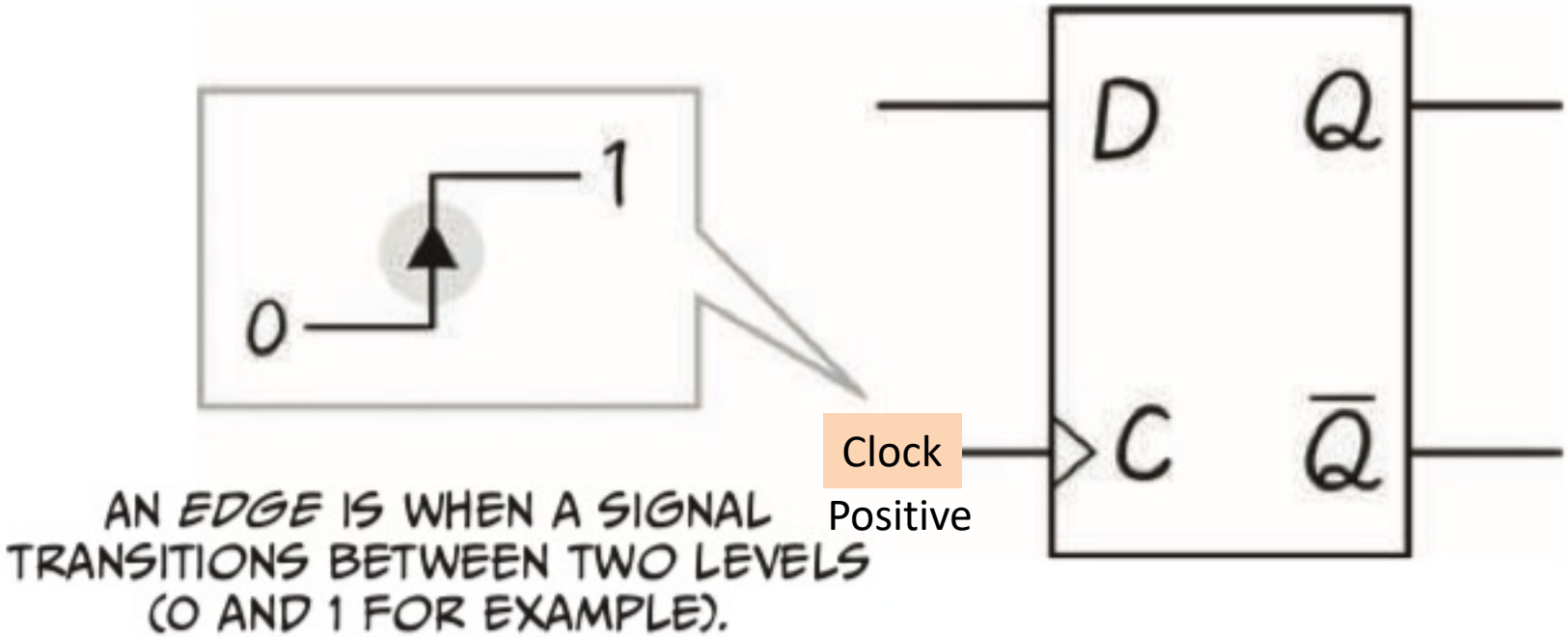
Logic

- Sequential
  - Flip-flops
  - Latches
  - Counters

Sequential Logic: Flip-Flops

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

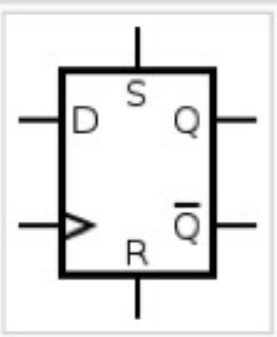
Clocked



Sequential Logic: Flip-Flops

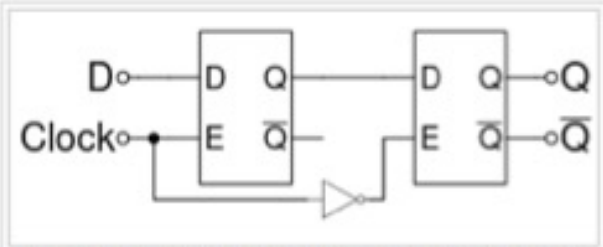
[from Wikipedia]

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

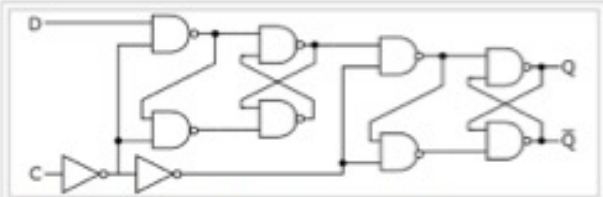


D flip-flop symbol

"D" FF

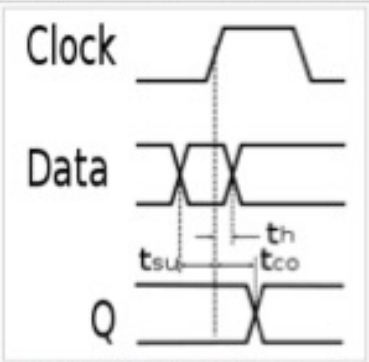


A master–slave D flip-flop. It responds on the falling edge of the *enable* input (usually a clock)



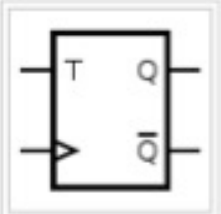
An implementation of a master–slave D flip-flop that is triggered on the rising edge of the clock

Timing: setup, hold, delay

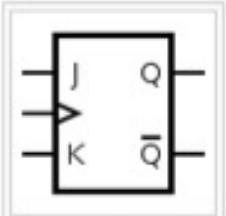


Flip-flop setup, hold and clock-to-output timing parameters

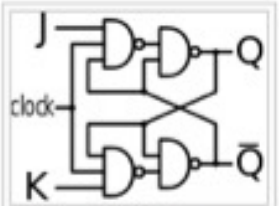
Other FFs



A circuit symbol for a T-type flip-flop



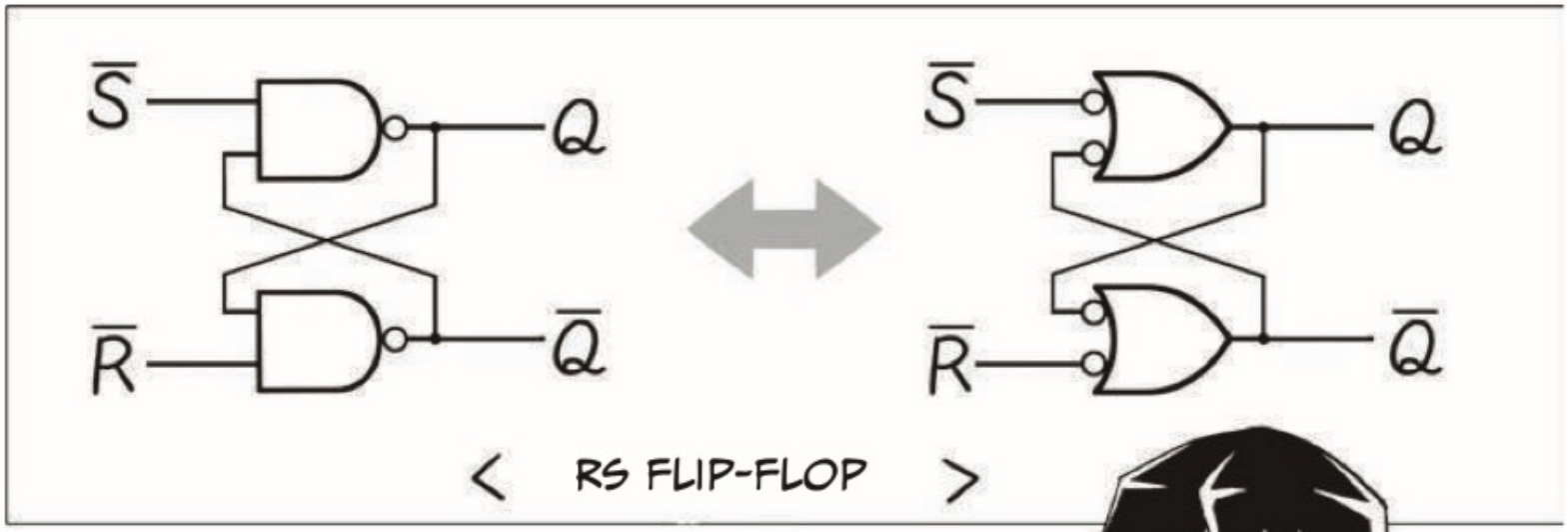
A circuit symbol for a positive-edge-triggered JK flip-flop



A JK flip-flop made of NAND gates

Sequential Logic: SR Latch

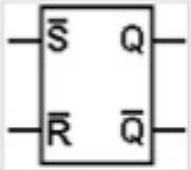
DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023



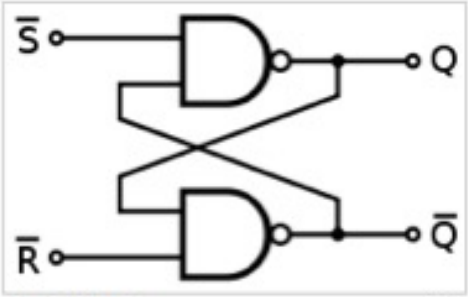
INPUTS		OUTPUTS		FUNCTION
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
1	1	DOES NOT CHANGE		RETAINS ITS CURRENT OUTPUT
0	1	1	0	SET
1	0	0	1	RESET
0	0	1	1	NOT ALLOWED

Sequential Logic: Latches

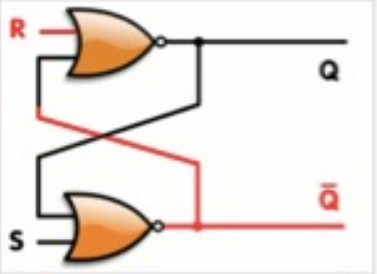
$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	not allowed
0	1	Q = 1
1	0	Q = 0
1	1	No Change



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



An $\overline{SR}$ latch

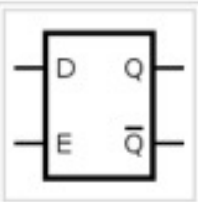


An SR latch, constructed from a pair of cross-coupled NOR gates.

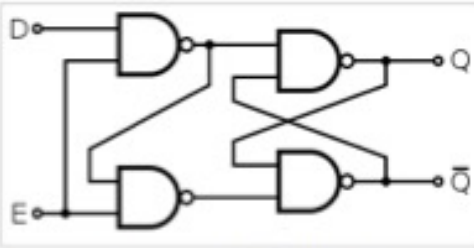
[from Wikipedia]

Gated D latch truth table

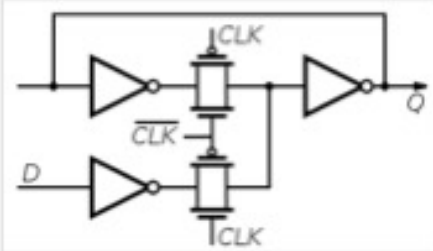
E/C	D	Q	$\overline{Q}$	Comment
0	X	Q_{prev}	$\overline{Q}_{prev}$	No change
1	0	0	1	Reset
1	1	1	0	Set



Symbol for a gated D latch



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

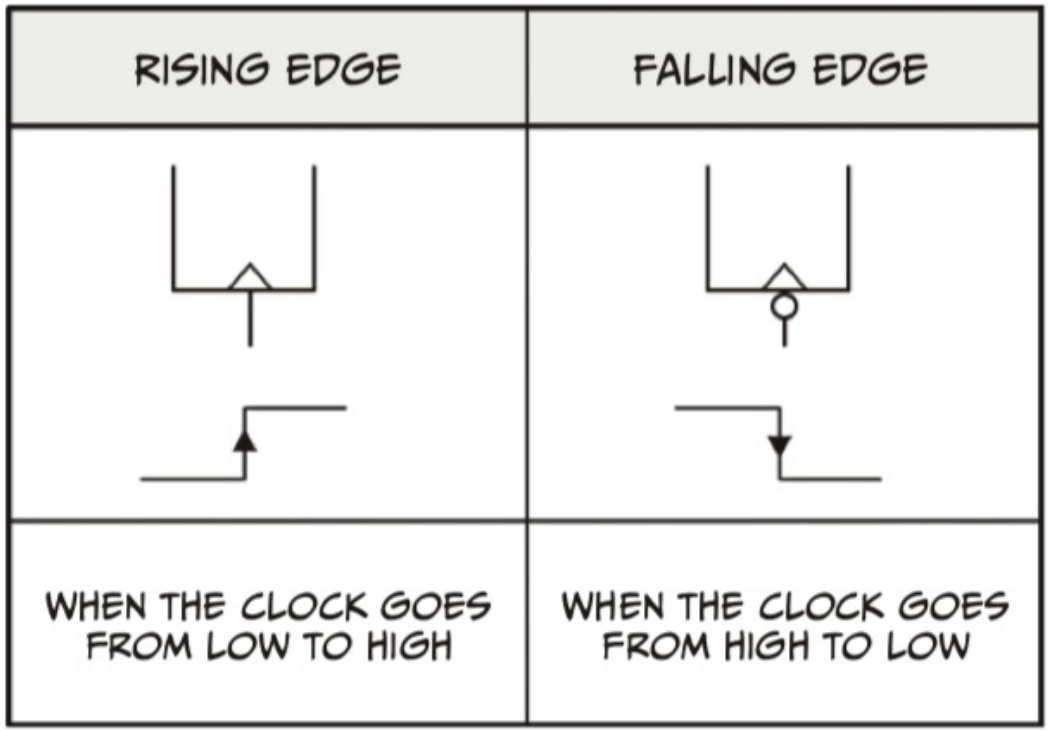


A gated D latch in pass transistor logic, similar to the ones in the CD4042 or the CD74HC75 integrated circuits.

Clocking

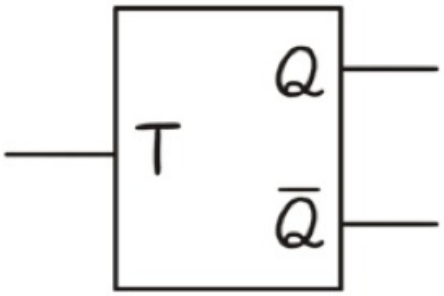
DR JEFF SOFTWARE
INDIE APP DEVELOPER
© Jeff Drobman
2016-2023

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.

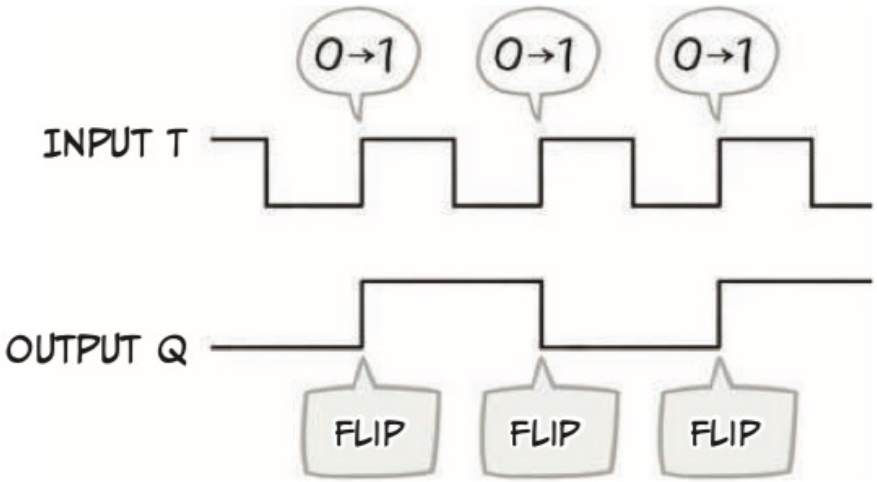


T: Toggle Flip-flop

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



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.



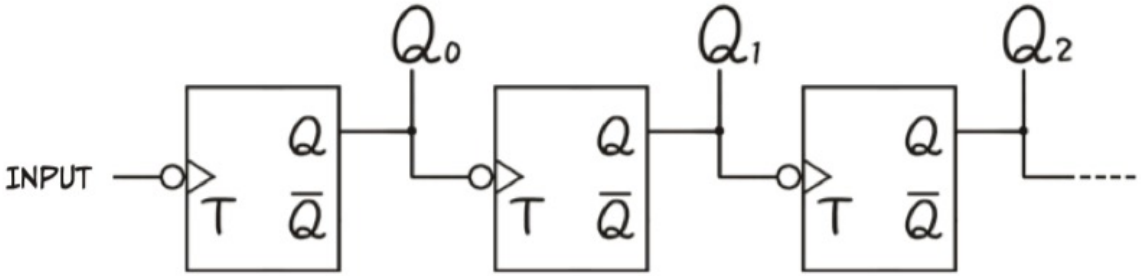
THERE ARE T FLIP-FLOPS THAT ACTIVATE JUST ON FALLING EDGES INSTEAD (1 TO 0).

Counter

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

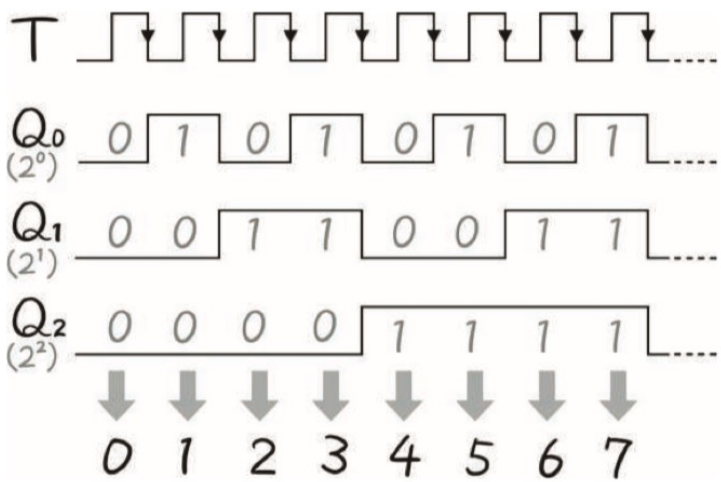
Toggle

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.



THIS CIRCUIT SHOWS HOW SEVERAL T FLIP-FLOPS TOGGLED BY THE FALLING EDGE OF AN INPUT SIGNAL CAN ACT AS A COUNTER.

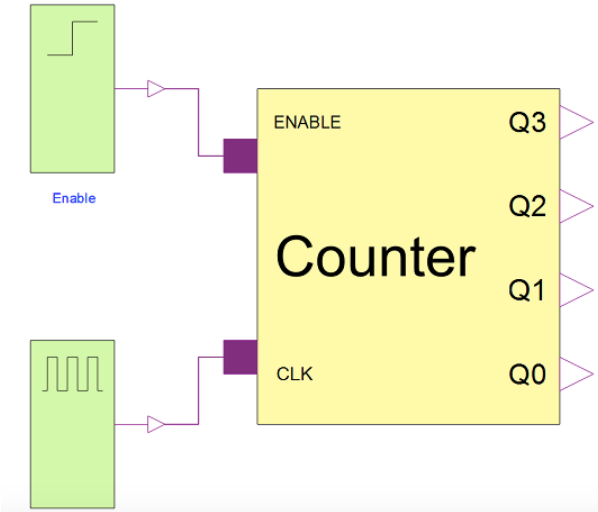
COUNTER CIRCUITS

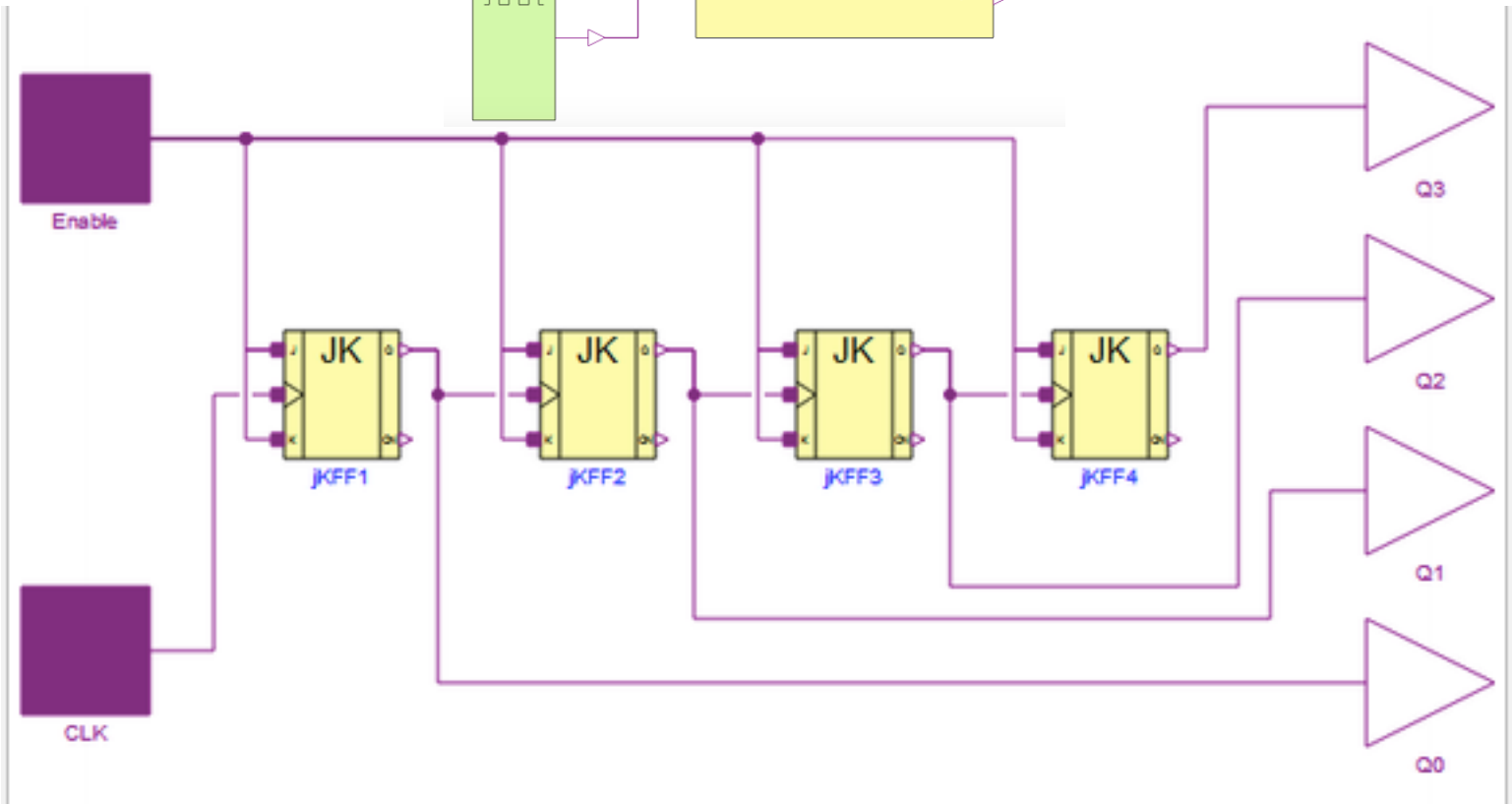


Falling edge triggered

Counter

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



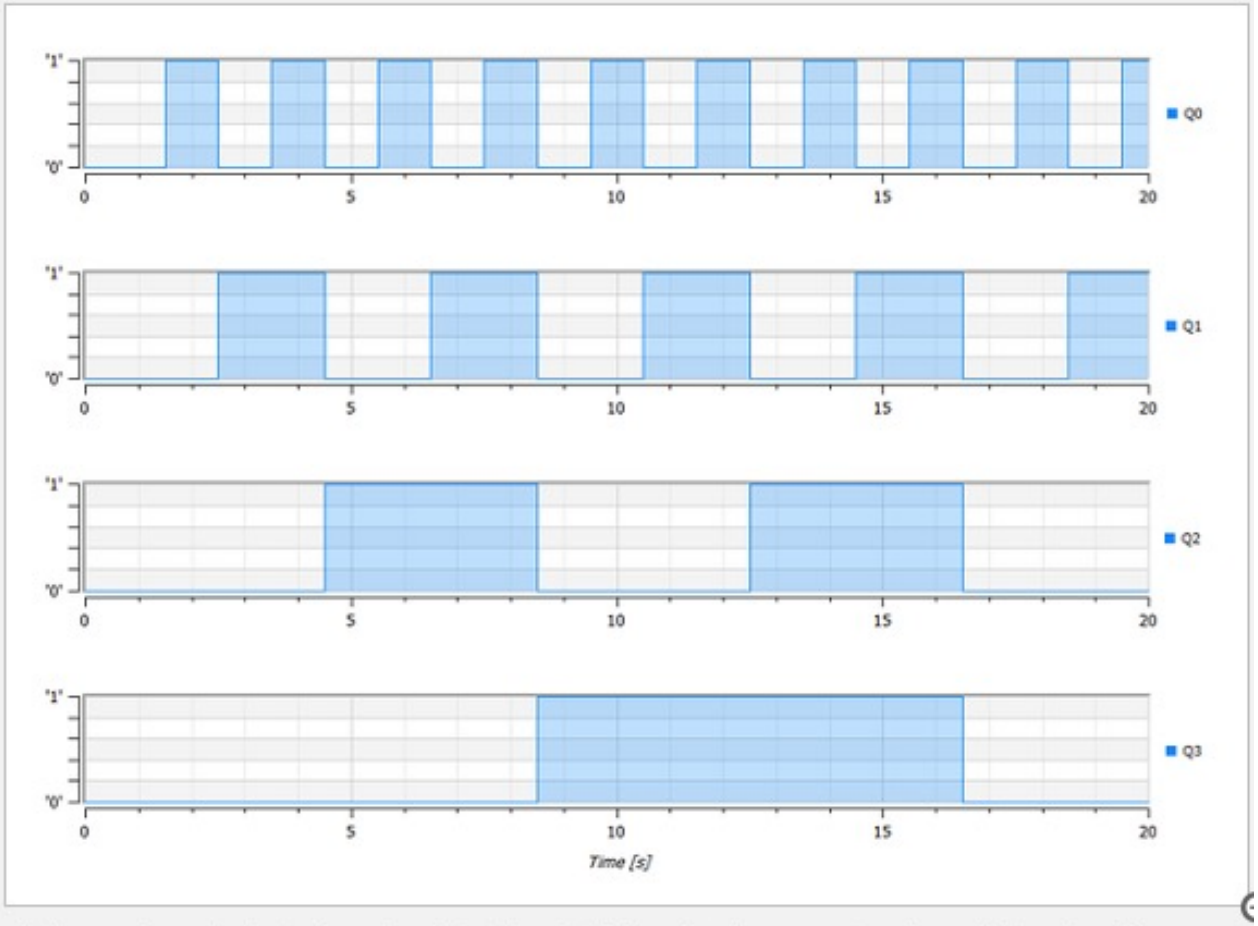


WOLFRAM

Counter

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

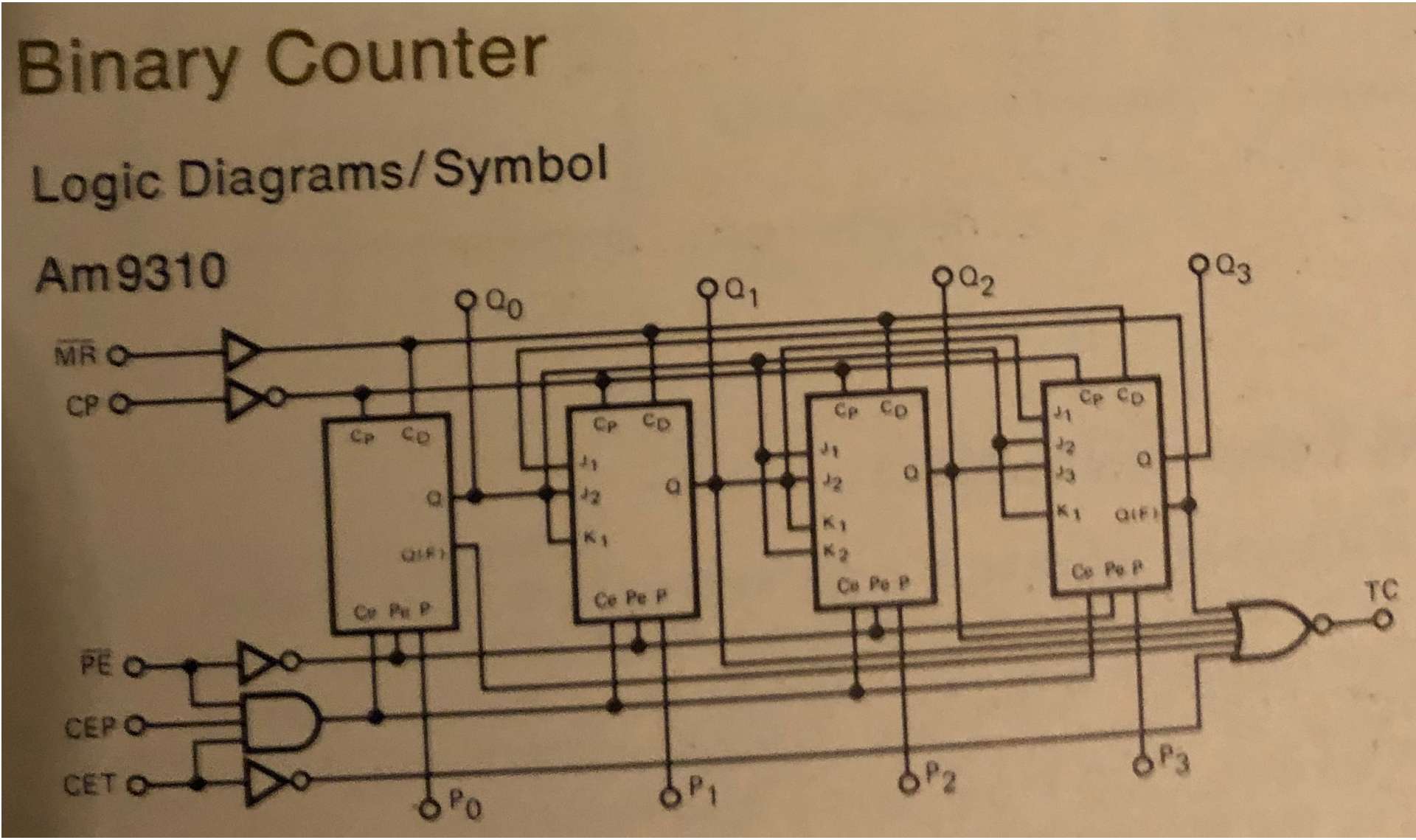
The counter in this example is a 4-bit asynchronous counter based on JK flip-flops. The flip-flops are connected with both their J and K terminals to the enable pin, putting them in "toggle mode". The flip-flop to the left, producing the Q0 signal, will change its output state for each falling edge of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

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



Section

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

Logic

- Multiplication
- Division

Binary Multiplication

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

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

Multiplication

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

Quora

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

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

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

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

- 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

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

Booth's multiplication algorithm

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

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

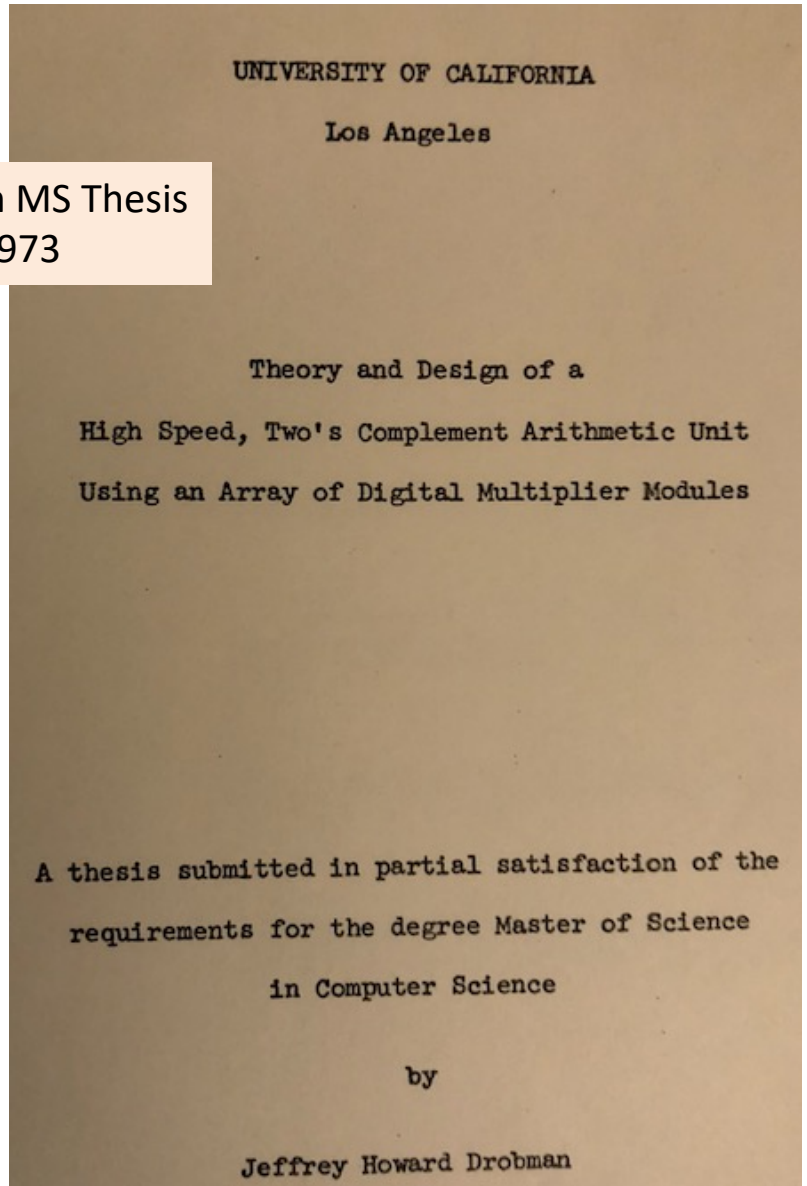
Binary Multiplication

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

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

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-

Am2505 Multiplier

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

Bit-slice 1971-80

#18 SN 7437 N

604
N 7400A Sig.
DM 7400N Nat.
NC 7400P M
UGA 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.

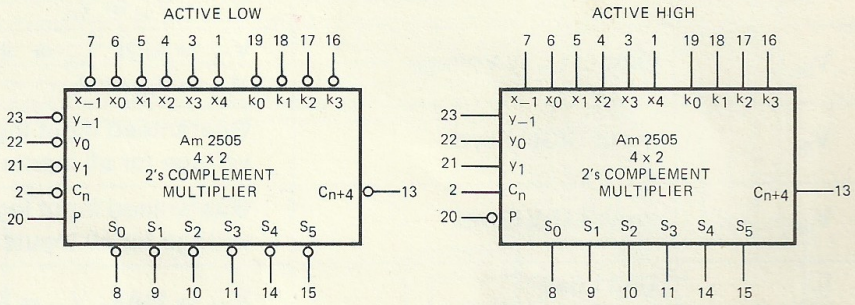
FUNCTIONAL DESCRIPTION:

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 $\overline{P}$. 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

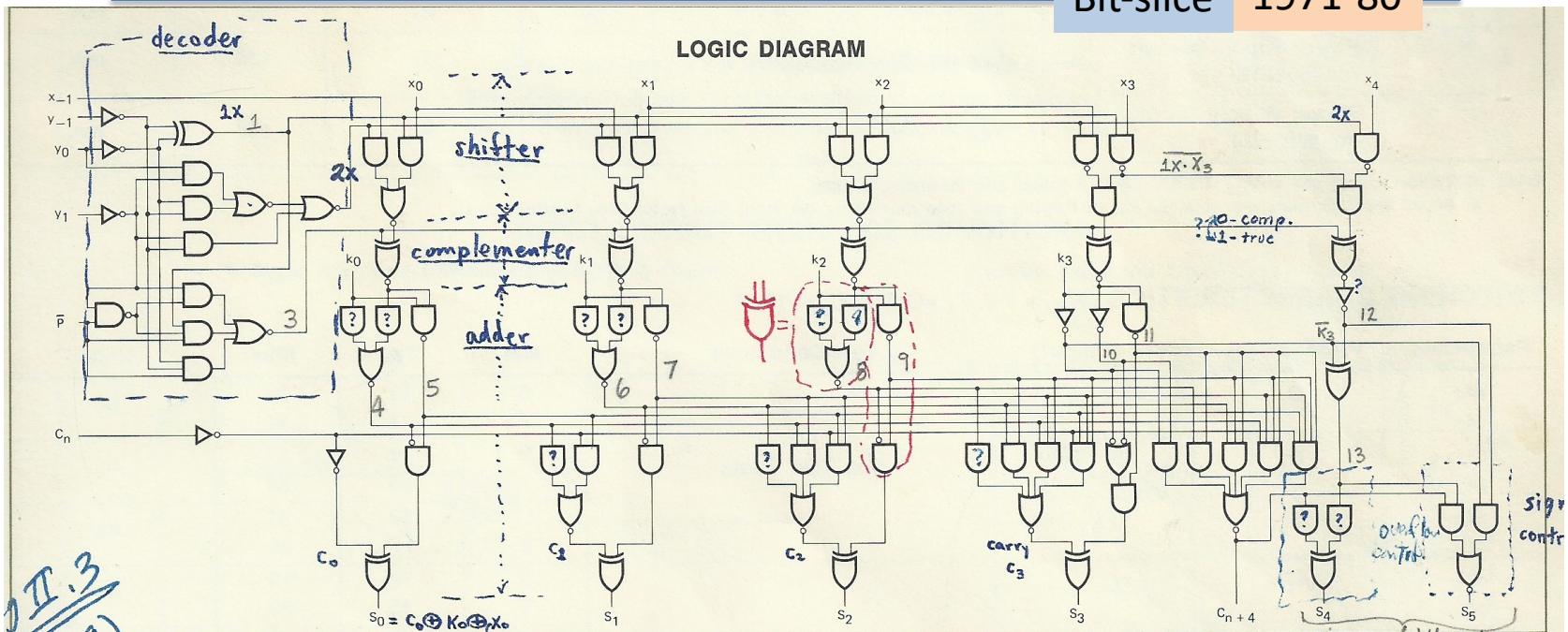
COMP122

Am2505 Multiplier

Bit-slice 1971-80

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

LOGIC DIAGRAM



II.3
use #8)
p.3

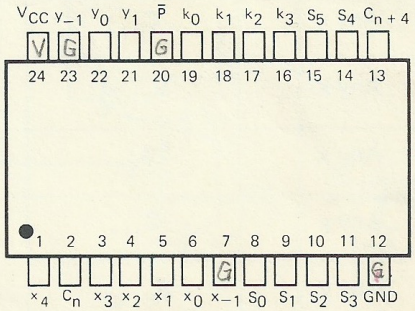
Am2505 ORDERING INFORMATION

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.

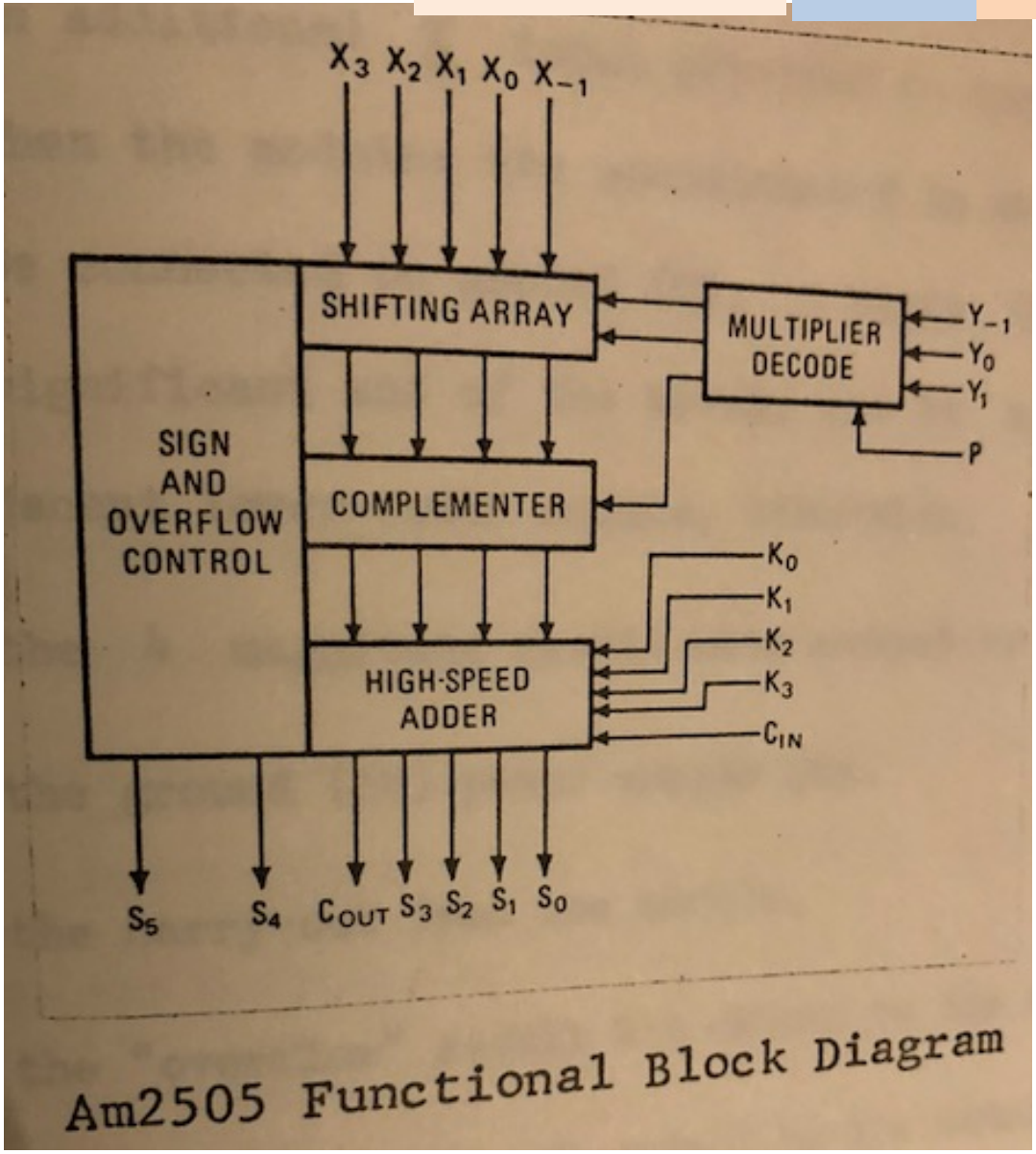
ignored diff not MS pa
logic
card:
top plane:
bottom plane:

$\bar{S}_5 = (k_2 \oplus x_2) C_{n+4} + \bar{x}_3 \bar{k}_2$

Am2505 Multiplier

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

Drobman MS Thesis | Bit-slice | 1971-80



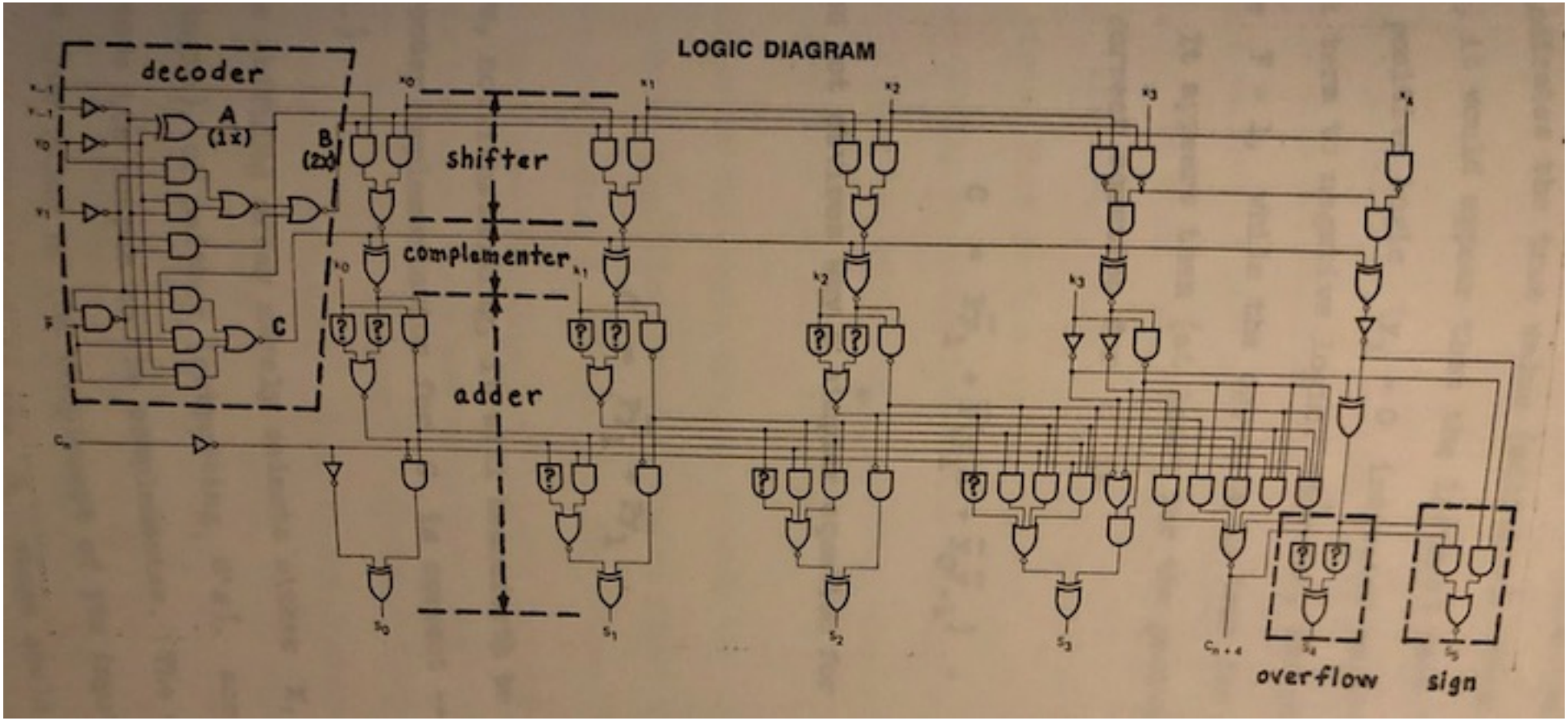
Am2505 Functional Block Diagram

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Am2505 Multiplier

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

Drobman MS Thesis | Bit-slice | 1971-80



Am2505 Multiplier

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

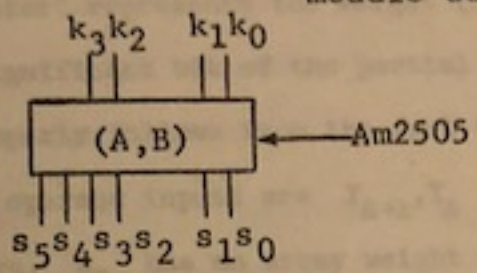
Drobman MS Thesis | Bit-slice | 1971-80

Notation: (from [1], [15], [16])

multiplier pair = Y_{A+1}, Y_A

multiplicand group = $X_{B+3}, X_{B+2}, X_{B+1}, X_B$

"module designator" = (A,B)



2x4-bit slices

8-bit x 8-bit multiply

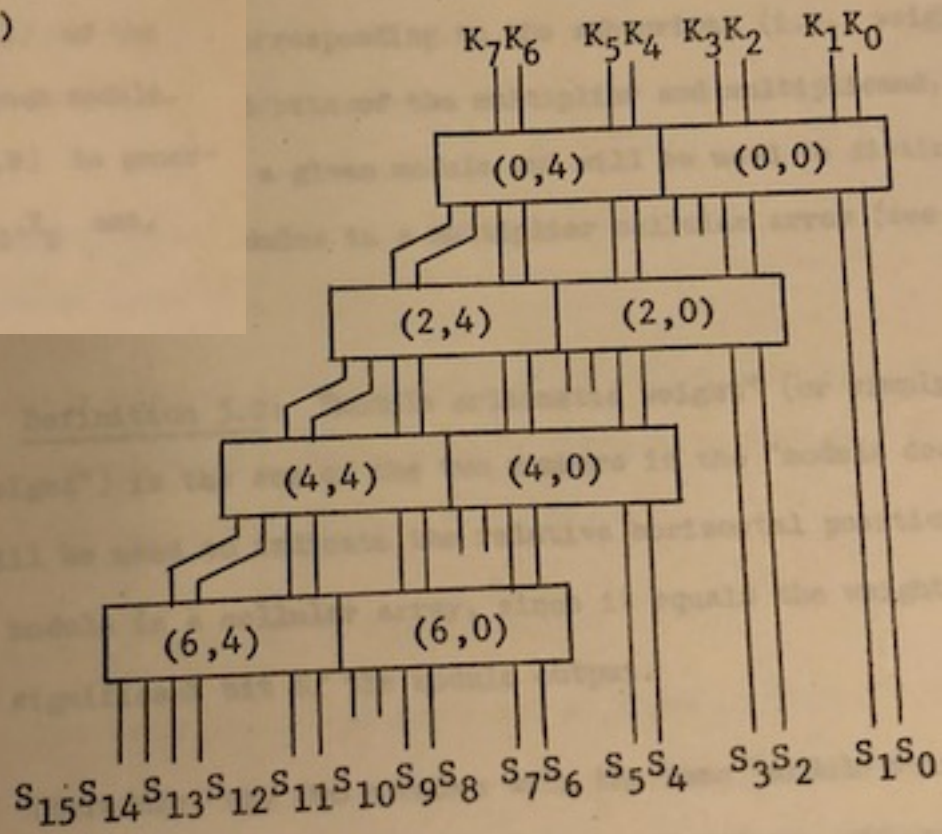
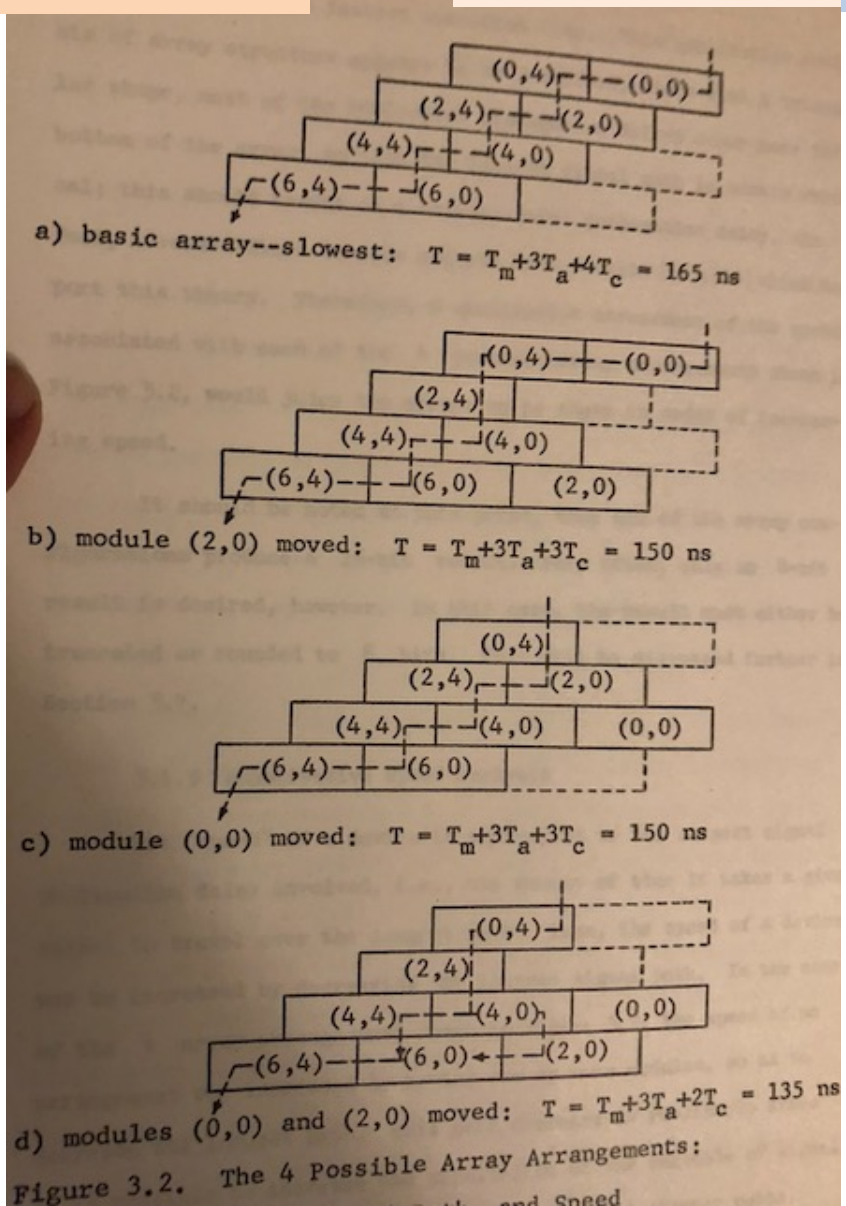


Figure 3.1. Basic Interconnection Scheme

Am2505 Multiplier

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

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



Division

Non-Restoring Div

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

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."

Division

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

Quora

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.

Division

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

Quora

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.

Division

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

Quora

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

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

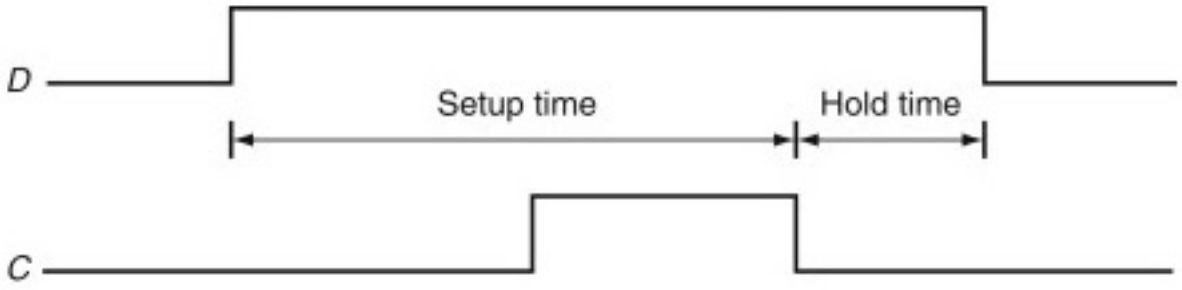
Logic Timing

- Am2900

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

Sync Timing (AC)

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



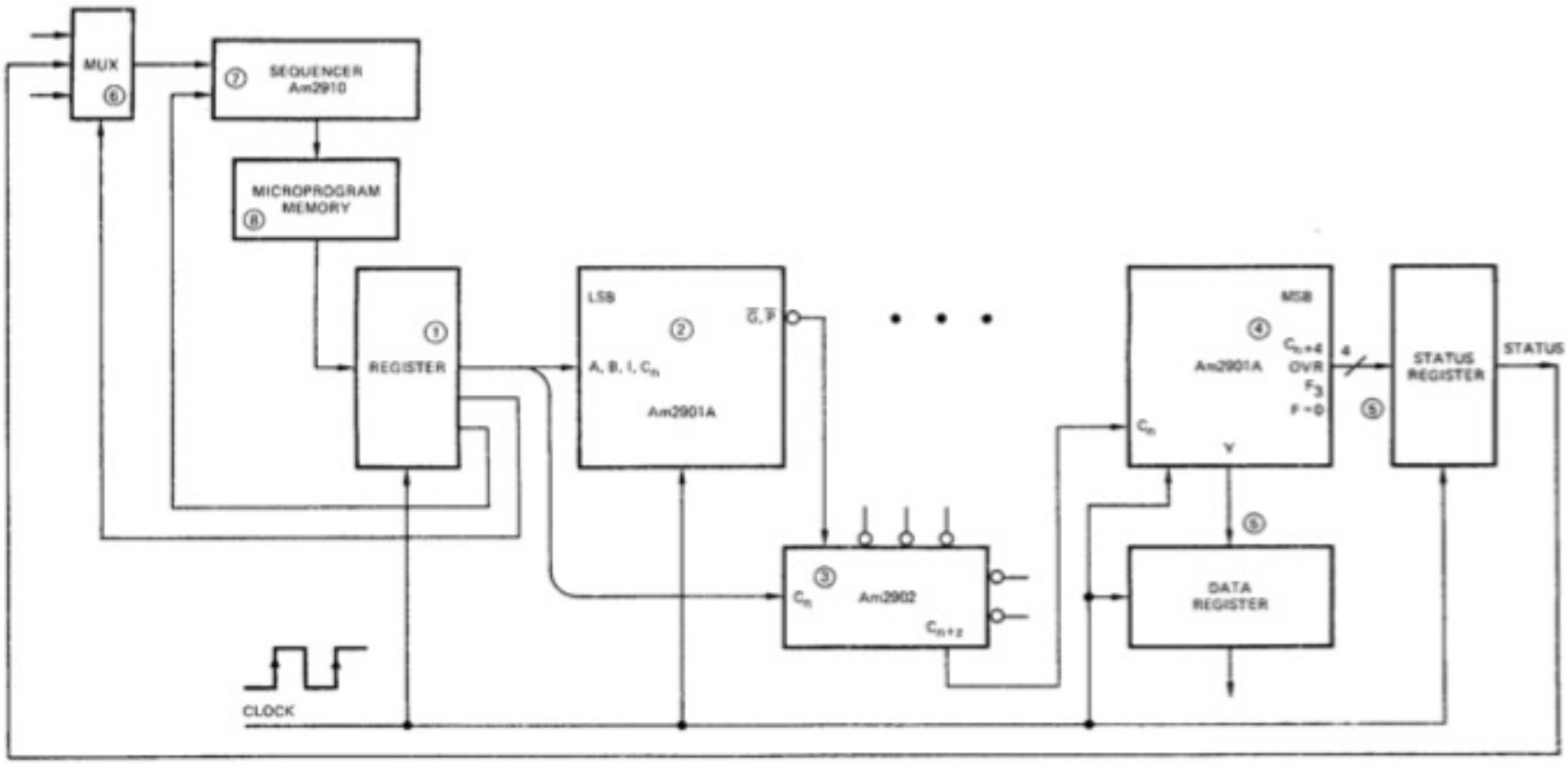
Timing – Cycle Times

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

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

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

Am2900

B. Combinational Propagation Delays.
C_L = 50pF

To Output / From Input	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

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

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)

To Output / From Input	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

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

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

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

State Machines
FSM

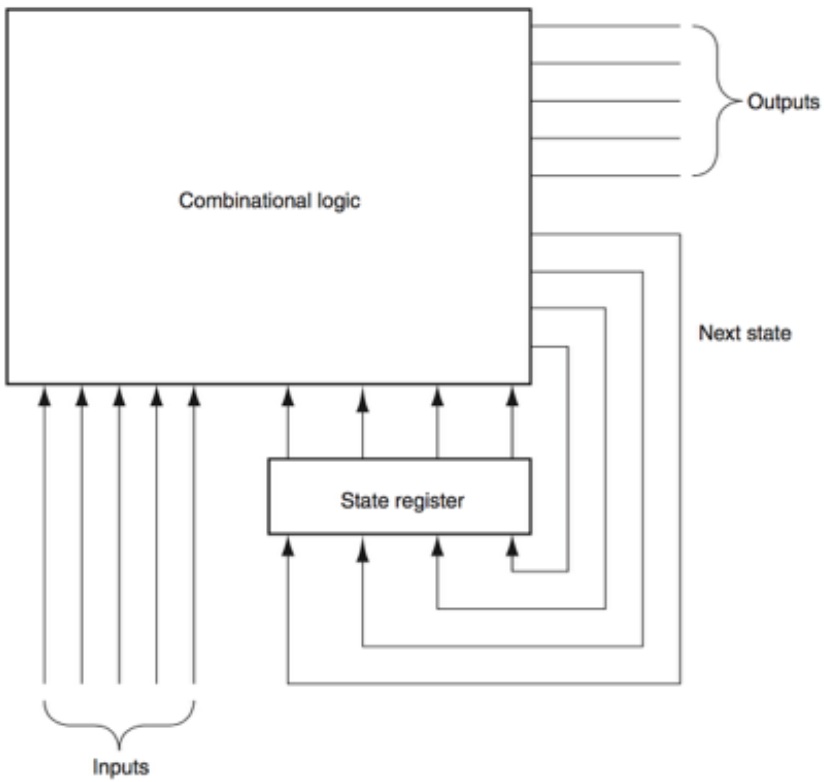
State Machines

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

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

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

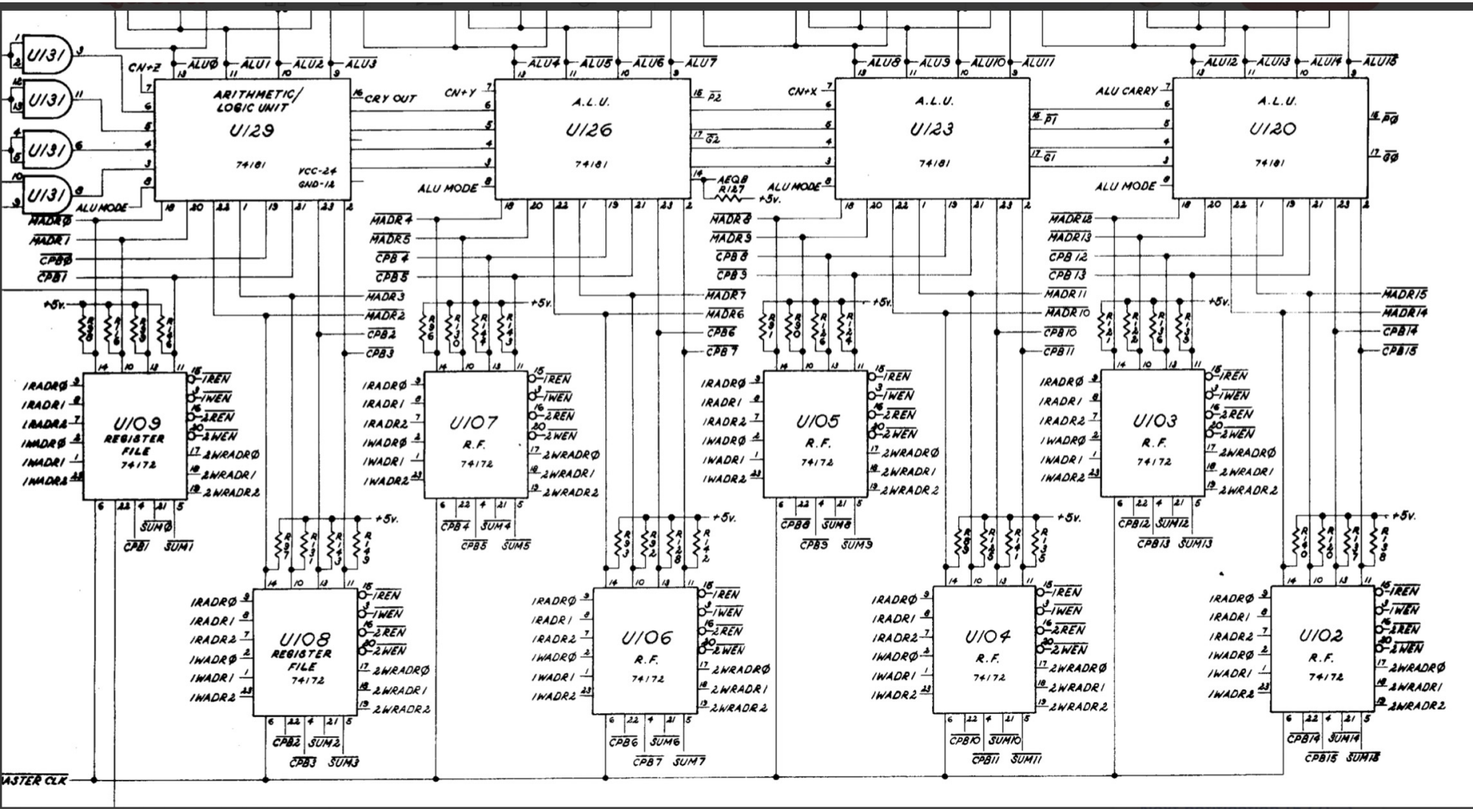
Computer Logic Boards

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

DG Nova 16-bit Mini Circuit

4x TI '**181** 4-bit ALU slice

DR JEFF SOFTWARE
INDIE APP DEVELOPER

© Jeff Drobman
2016-2023