Professor Steve Luetta
Professor Volodymyr Kindratenko
TA's:
Nathan Bush
Sergio Hidalgo
Yubo Liu
Shengyuan Zhang
Xiangyu Chen
He Huang
Pranay Vissa

Discussion section starts tomorrow!!!

5pm today
ECE ignition
here for all freshman and new transfer students

My office hours this week
Th 1-3 in 2022 ECEB

type: ECE198JL into google
https://wiki.cites.illinois.edu/wiki/display/ece120

Alan Turing 1936
universal computation device

Computable
android phone
iPod
Blue waters
Tianlte

Undecidable

Digital convergence
-almost all of you will use digital systems every day

-most solutions are digital
-critical set of skills for every one of you

Why bottom up?
-solid understanding of design and operation
-easier to make effective use and improvements
-our students have been successful based on this model

work load
14 (weekly) homework assignments
(first due next wed)
-->15% of grade
15 (weekly) laboratory assignments (software and hardware) --> 15% of grade

15 (weekly) discussion sections (group work and integration)-->5% of grade

3 midterms

Tuesday 16 September 7-9 pm
Tuesday 14 October 7-9 pm
Tuesday 11 November 7-9 pm

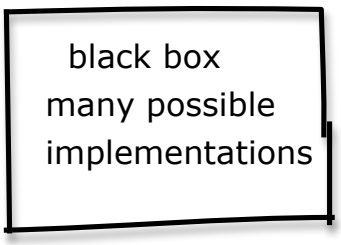
conflict exam 5-7 pm

final exam 25% of grade
Monday 15 December 8-11 am

abstraction in a digital system

an abstraction layer

functions for higher interfaces


black box
many possible
implementations

layers in a computer system
representations
binary digits (bits)
integer representations
Friday (probably some of it anyway)

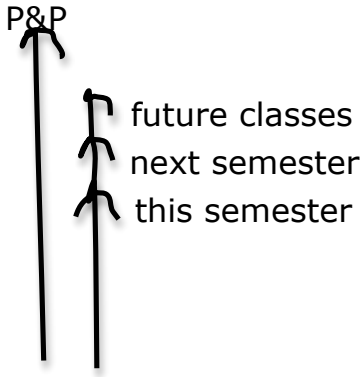
¬ ⌐>□ΛEΛ⌈V□⌊⌐ π⌐ □
give yourself a hand

Professor Kindratenko's office hours
M: 10am-11am in 2022
W: 10am-11am in 2022
F: 10am-11am in 4034

Lumetta's hours Th 1-3
upstairs at miaZa's (green street)

lab 1 is due 8/28 (tomorrow)
see lab document for lab rooms
exams designed as 1 hour (1.5 final); we give 2x time

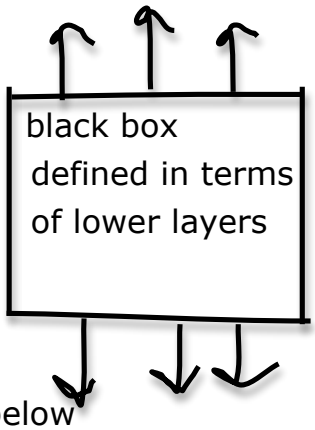
problems/tasks
algorithms
language
machine/instruction set architecture (ISA)
microarchitecture
circuits
devices



from P&P chapter 1

provide interfaces/functions to higher layers

an abstraction layer



use interfaces from below

Problems/tasks/applications
-stated in natural (human) language
-for example: what is the sum of numbers between 1 and 3
6,∞,2,42

problem: ambiguity in question

time flies like an arrow

a problems can be solved with many algorithms

algorithms is a step by step process to solve a problem
1. definiteness: no ambiguity
2. effective computability: each step simple enough for a computer to execute
3. finiteness: always terminates

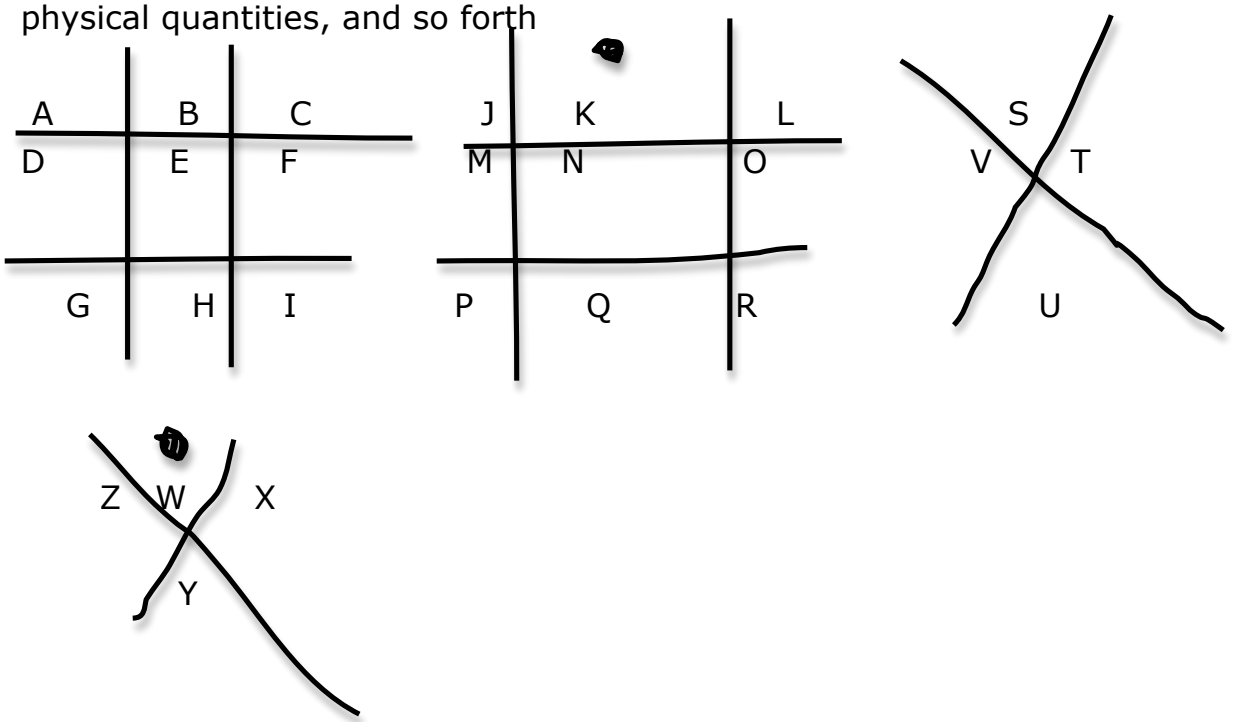
an algorithms can be implemented using many languages
-C, C++, Java, Python, and so forth
-1000s of choices
-we use C to start
  -simple mapping to ISA
  -subset of other language

A program can be executed on many instruction set architectures.

Machine/Instruction set architecture
-interface between software (s/w) and hardware (h/w)

x86-phenom, i5, i7
ARM-cortex, kinetis
PowerPC-many

representations
often useful to represent one type of information with other patterns, physical quantities, and so forth



what properties does a representation need to be useful?

0 1 2 3 4 5 6 7 8 9
A B C D E ... J
K L M N O ... T
U V W X Y Z

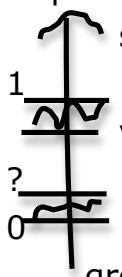
"143"
BED
BOX
VYN

no ambiguity: each pattern represents at most one value

also need the representation to be well defined
all users know representation in advance

BInary DigiTs (bits)
0 and 1

computers based on electrons



some high voltage (Vdd)

voltage at some place in a computer

ground (0V by definition)

-->one bit

positional/place value
decimal 42
10^2 10^1 10^0
4 2

binary 101010
2^1 2^0
1 0

000000
000001
...
111111
2^6

represent whole #s in range...
0 to 31-->5 bits gives 2^5 patterns
0 to 100? -->7 bits gives 2^7=128 patterns

Integer representations:
what #s should we represent?
5 bits-->represent which numbers
what makes a representation good?

unsigned representation
"good" representations
2's complement

recall: we can use bits to represent anything
useful examples: integers, real numbers, human language characters (alphabet, digits, punctuation)

important: computer does not "know" meaning. bits are bits.

-->Lab 1 grades are out
run 'svn update' in your lab1 folder and check grade.txt file
-->Lab 2 is out. it is due next Th.
-->HW 1 is due next W. In lecture
-due at start of lecture at back of room
-legible, stapled, and so forth (see instructions)
-put in your section's envelope
-will disappear ~9:30 (no credit afterwards)
-solutions up in 4034

no lecture Monday. Lab is open...
if access control allows you to enter...

Integer Representations:
What numbers should we represent?
We want to use numbers to do arithmetic.
What if we use 5 bits to represent 100...131?
124+131
100+101

-->represent numbers close to 0

Let's start with a human representation

base 2 from math
17_10-->10001_2

42_10-->101010_2

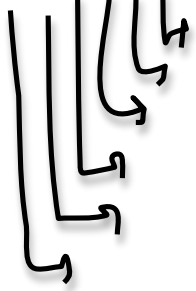
But there is no "blank" bit.
-->use a fixed number of bits
-->add leading 0s.
-->unsigned representation

What range of numbers is representable with the N-bit unsigned representation?

[0, 2^N-1]

How do we convert unsigned to decimal?

0 1 1 0 0 1



1=2^0
2=2^1
4=2^2
8=2^3
16=2^4
32=2^5

1+8+16=25

a5a4a3a2a1a0
represents: a5*2^5+a4*2^4+a3*2^3+...+a0*2^0

What about decimal-->unsigned conversion?

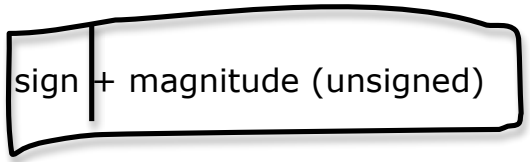
Start with even number-->a0=0
odd number-->a0=1

Subtract out a0, divide by two
Repeat until I get 0

137 a0=1 (137-1)/2=68
68   a1=0 (68-0)/2=34

34 a2=0 (34-0)/2=17
17 a3=1 (17-1)/2=8
8 a4=0 (8-0)/2=4
4 a5=0
2 a6=0
1 a7=1

What about negative numbers?
Use a minus sign?
-11000_2=-24_10
+ --> 0
- --> 1



signed-magnitude representation

What does 1000 represent in 4 bit signed magnitude? 0
0000-->0

Having multiple zero patterns makes hardware (like adders) more complicated?

What makes a representation good?
-efficient, not unary
5 0000011111
10 1111111111
-effective for use
-use same hardware for more than one representation

unsigned addition

A	B	Sum
0	0	00
0	1	01
1	0	01
1	1	10

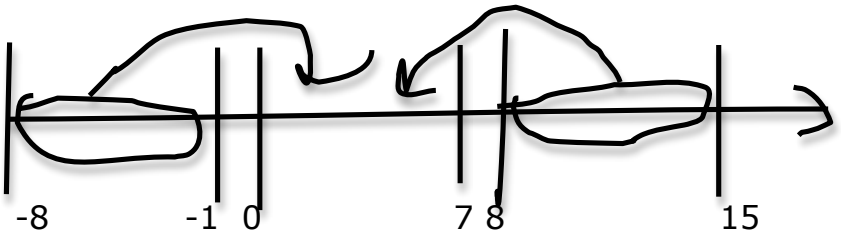
  01110 (14_10)
+00100 (4_10)
  10010 (18_10)

  10111 (23_10)
+01010 (10_10)
 100001 (33_10)
x00001 (1_10)
fixed width representation

overflow (answer; not representable)
2^N

Answer (sum) is always correct mod 2^N?

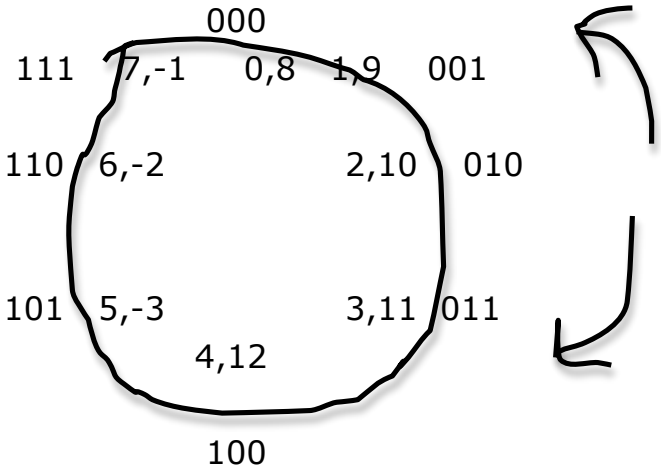
Similar to remainder.
numbers A and B are equivalent mod k if and only if A=B + pk for some integer p



Remember: open lab on Mondays 2022ECEB
Other labs: 440 DCL, 520 DCL, 057 Grainger
turn in labs by computer
next due 7pm Thursday
Today 2-5 pm between Everitt and Engineering Hall, International Engineering Fair.

2's complement
overflow
logic operations
challenge: prove that the expression:
(carry out)XOR(Carry into MSB) is equivalent to the 2's complement
Overflow definition given in lecture (today).
Recall: unsigned addition uses modular arithmetic
adder(A,B)={A+B if A+B<2^N}
{A+B-2^N if A+B≥2^N}
so
(adder_ N (A, B) = A+B) mod 2^N

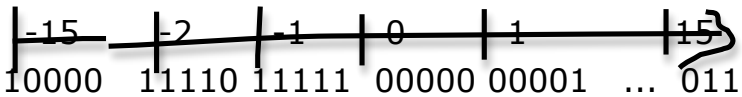
Another way to think about modular arithmetic: draw a circle



subtract to count counter clockwise

add to count clockwise

signed integer representations



10000 11110 11111 00000 00001 ... 01111

adder_ N (A,B)=(A+B) mod 2^N
given m≥0, 2^(N-1)>k>0, can we find p>0 such that
(-k + m=p + m) mod 2^N

if so, we can use p's representation to represent -k
subtract m from both sides
(-k=p) mod 2^N
(2^N=0) mod 2^N
(2^N-k=p) mod 2^N
let p=2^N-k, equation is true.
2^(N-1)<p<2^N (all unused patterns!)
-2^4+1=-15

How do you calculate 2^N-k?
100000 (N=5)
- k
_____

Instead, notice
(2^N - 1) + 1=2^N

11111 (2^N - 1 for N=5)
-k
_____
k complemented (0s become 1s, 1s become 0s)

42
00101010
11010101+1=11010110
2^N - 1 - k

11010001
k

-2^8+2^7+2^6+2^4+2^0
-2^7+rest of bits
-2^8+2^7=-2^7

Better to define
100000...0
N bits
as + 2^(N-1)
-->OR -2^(N-1)

```
  10111 (-9)
+10011 (-13)
_______
1 01010 (10)
```

```
01000
+1
01001
```

```
  01000 (8)
+01010  (10)
____________
10010
```

```
01101
+1
_____
01110
```

```
  A
+B
___
C
```

we have overflow if
1. addends are negative, sum is non-negative

OR 2. addends are non-negative, sum is negative

overflow=
(A AND B AND (NOT C)) OR ((NOT A) AND (NOT B) AND C)

Operators are Boolean logic functions
(true=1, false=0)

AND-the all function
returns value 1 iff all inputs operands are 1

OR-the any function.
returns 1 iff any input operand is 1

NOT-logical complement
(NOT 1) is 0
(NOT 0) is 1

XOR-the odd function
returns 1 iff an odd number of input operands are 1.

Truth table maps input values to output values.

A*B, AB A+B

A B	A and B	A OR B
0 0	0	0
0 1	0	1
1 0	0	1
1 1	1	1

A' or A_

A	NOT
0	1
1	0

A B A⊕B, AXORB
0 0 0

0 1 1
1 0 1
1 1 0

Logical completeness
fixed and floating point
representation taxonomy
next week C

ASCII
parity
hamming codes
book and notes

Recall:
A
+B
__
C

2s complement

overflow iff ABC_+A_B_C

AND=all

NOT=complement
OR=any
XOR=odd

representation taxonomy

bits

unsigned integers	signed integers	real integers	text	vegetables
			ASCI UNICODE	
unsigned binary	signed magnitude	2s complement	real numbers	
16 bit unsigned		16 b 32	floating point	
data type			IEEE single prec	
			IEEE double prec	

A B C A⊕B⊕C

0 0 0 0
0 0 1 1
0 1 0 1
0 1 1 0
1 0 0 1
1 0 1 0
1 1 0 0
1 1 1 1

$A \oplus B = B \oplus A$
Commutative

$A \oplus (B \oplus C) = (A \oplus B) \oplus C$
Associative

We can generalize also to sets of bits
A=a3a2a1a0
B=b3b2b1b0
C=AB
c3=a3b3
c2=a2b2
...

Logical completeness
How many functions can we define on N bits of input?
C=f(A,B)
How many choices of f?
A B f(A,B)
0 0 c0
0 1 c1
1 0 c2
1 1 c3

2^4=2^2^2 functions

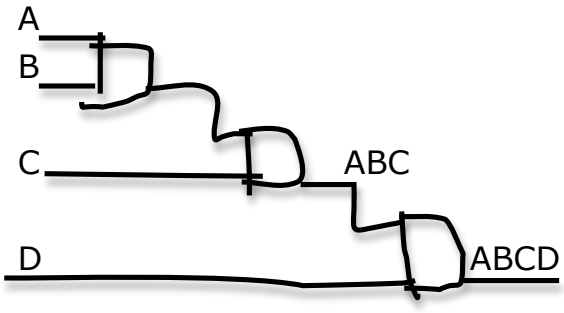
with 3 bits of input, how many functions?

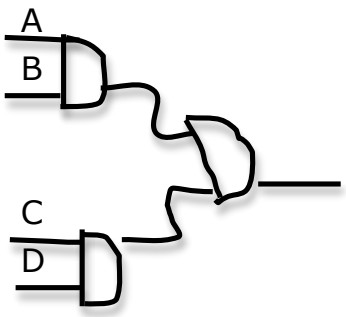
256=2^8=2^2^3

With N bits?
2^2^N (big!)

Why do we only talk about AND, OR, NOT?
claim: with enough 2-input ANDs, 2-input ORs, and NOTs, I can compose any logic function on any number of variables.

Proof: by construction
in other words, given a function, I'll show you how to build it.





we can build AND gates with any number of inputs

one 1
A_BC_
one AND gate and up to (# inputs) NOTS
A B C f
0 0 0 0
0 0 1 0
0 1 0 1

0 1 1 0
1 0 0 0
1 0 1 0
1 1 0 0
1 1 1 0

A B f
0 0 0
0 1 1 A_B
1 0 1 AB_
1 1 0

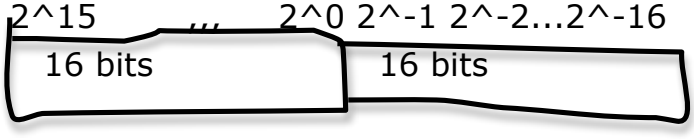
A_, A', (NOT A)

fixed and floating point
3.1415
  10^-1, 10^-2, 10^-3
decimal point

11.0010010001
2^-1, 2^-2
binary point

fixed point

2^15 ... 2^0 2^-1 2^-2...2^-16



Avogadro's number
6.02252*10^23
10^3≈2^10
so ~ 80 bits will work

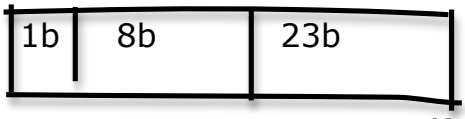
Planck's constant
6. 626*10^-27

so ~ 90 bits

170 bits fixed point!

Avogadro's number

Planck's constant
+6.626*10^-27
sign + leading digit + fraction + exponent



sign exponent mantissa (fraction)

-1^(sign)*1.(mantissa)*2^(exponent-127)

EX C Program

```
#include <stdio.h>

#define PI=3.1416 f

int main
{
/* declare variables */
float pi=PI;

/* print inc pi */
printf("pi=%f\n",pi);

return 0; /*exit*/
}
```

```
#include <stdio.h>
#define PI 3.1416f

int main()
{
  /* declare variables */
  float pi = PI;

  /* print message */
  printf("pi=%f\n", pi);

  /* exit */
  return 0;
}
```

a^2+b+c

<type> <name>;

{int}
{real}
{text}

2's complement

{int}
{short int 16 bit}
{long int 32 bit}
{long long 64 bit}
-->add unsigned

unsigned int-

float -32 IEEE
double -64 IEEE

constant

char -8 bits

```
int a=10;
int b;
int c;

b=20;

c=a+b;

c+b

int x, a;
float y, z;
z=x+y;
-->z=(float) x+y;
```

operators

=

a+b=c+d-->X

-->a=c+d

* / + - %

~-not

&-and

|-or

^-xor

<<-left shift

>>-right shift

```
unsigned int h,f,g
h=f&g
```

0-->false
~0=true

!-bit
&&-and
||-or

>-less
>=-less or equal
<-greater
<=-greater or equal
==-equal
!=-not equal

(a>b) &&(a<c)

a b a c

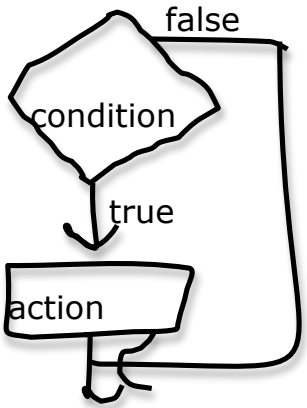
a=a+b;
a+=b;

a=a-b;
a-=b;

a^=b
a=a^b

a++
b--

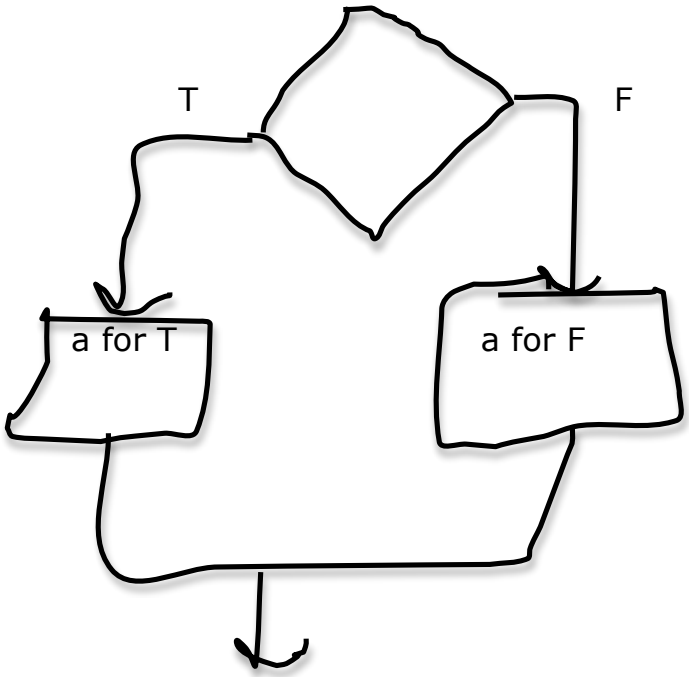
if(condition)
{
action
}



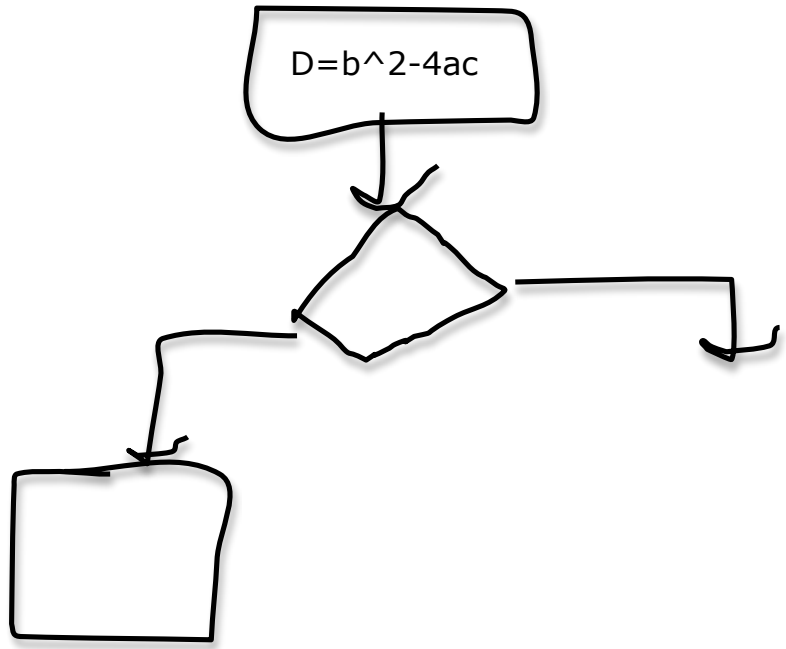
x=|x|;
x={x if x≥0}
{-x if x<0}

if ((x>5)&&(x<25))

if (condition)
action for true
else
action for false
{
}



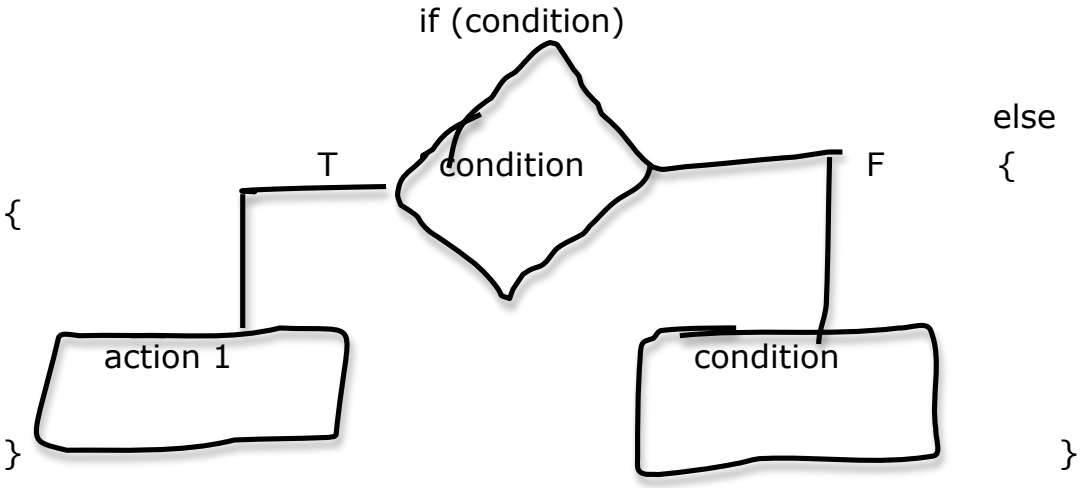
ax^2+bx+c=0
a=b^2-4ac
if D>0
x12=(-b±√D)/2a
if D=0
x=-b/2a



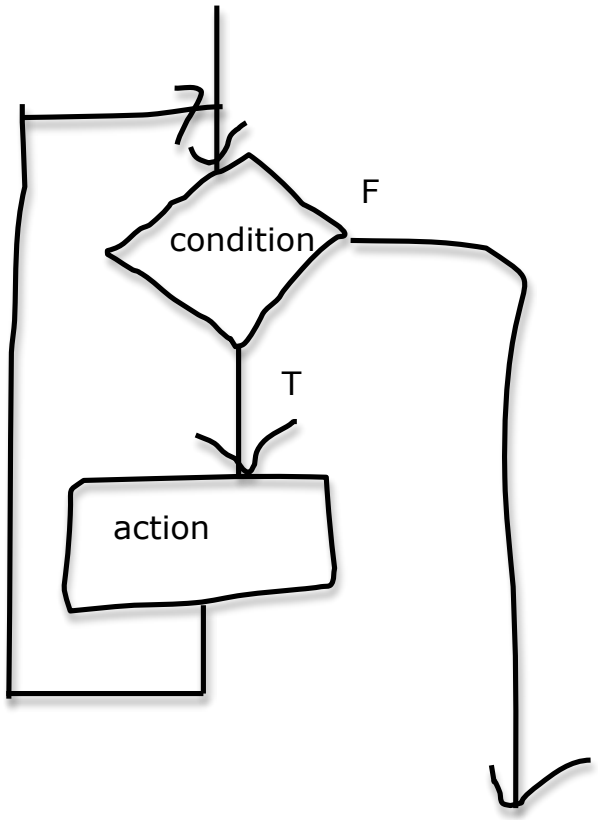

```
/* quadratic eq. */
#include <stdio.h>
#include <math.h>

int main()
{
float a,b,c;
float D;
float x1, x2;

printf("enter a b c: ");
scanf("%d %f %f", &a, &b, &c);

D=b*b-4*a*c;

if(d>0)
{
x1=(-b+sqrt(D))/(2*a);
x2=(-b-sqrt(D))/(2*a);
}
```

```
else if (D==0)
{
x1=-b/(2*a);
}
```

Sunday 12-2pm Review Session



while()
{
action;
}

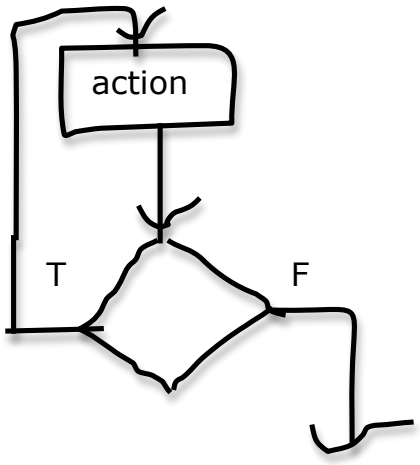


```
1. int a=0;
2. while(a<10)
{
3. print("a=%d\n",a);
4. a=a+1;
}
5.
```

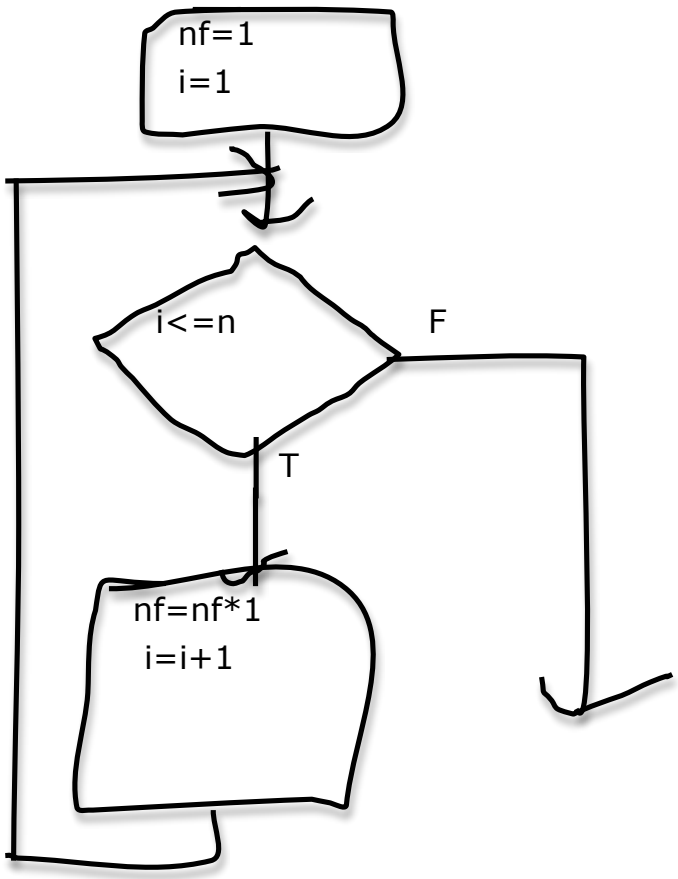
```
a++;
a+=1;
```

1. a=0
2. 0<10 1<10 a<10
3. print print
4. a=1 a=2 ... a=10
5.

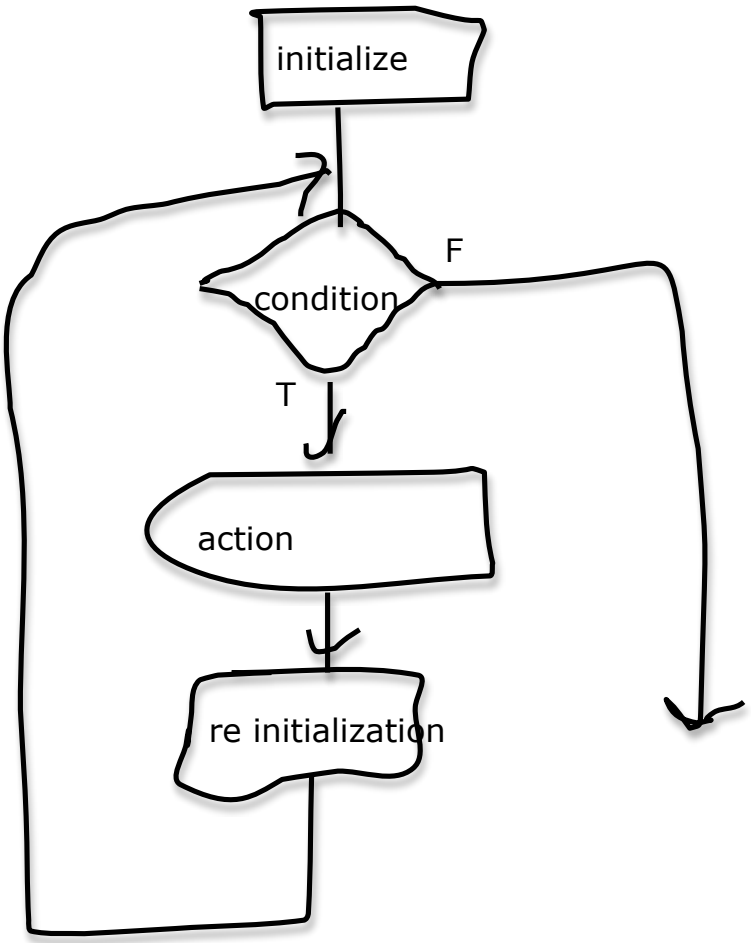
do while



$$n!=1*2*3*...*(n-1)*n=\prod_{i=1}^{n} i$$




```
int n=10;
int nf, i;
nf=i=1;

while(i<=n)
{
nf=nf*i;
i=i+1;
}
```




```
for(initialization; condition; re initialization)
{
action;
}
```

```
int n = 10;
int i, nf=1;

for(i=1;i<=n; i+1)
{
nf=nf*i;
}
```

n

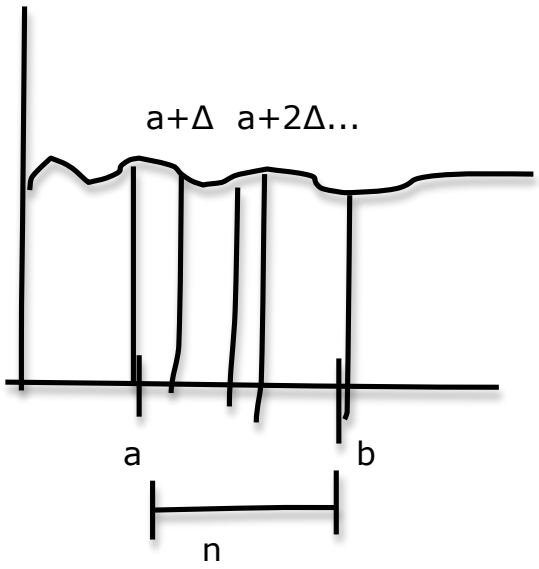
$x^n = x*x*...*x = \prod_{i=1}^{n} x$ (n times)

```
int i;
int x=5;
int xn=1;
for(i=0; i<n; i=i+1)
{
xn=xn*x;
}
```

$x^0 = 1$

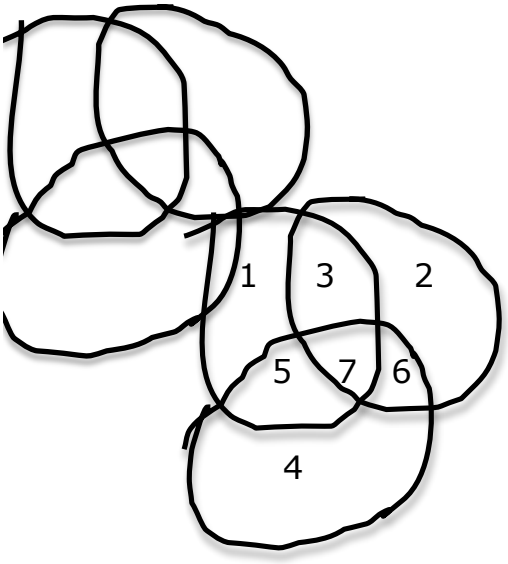
```
int i =1;
for ( ;condition; re initialization)

for([initialization]; [condition]; [re initialization])

if(condition)
  action;
else
  break;
-->don't do this


for(...)
{
continue;
...
...
}
-->don't do this either
```

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} (b-a)/n * f(a+(b-a)/n*i)$$



(a-b)/n

```
fint=0;
for(i=0;i<n;i++)
{
fint +=...;
}
```



signed
  1 1 x x
  1 1 x x
1 1 0 x x

0 1 1 1 1 1 1
d d d p d p p
c1 c2

1 1 1

1 1
1

c4
1 1 1 -->7
1 1 1 1 1 1 1

even 0
odd 1

abstraction
levels of transformation
bit representation
hex notation

codes, error detection and correction

binary-->decimal
arithmetic operators
fixed and floating point
Boolean operators
C programming:-variables
-operators
-conditional, iterative constructs
-problem solving with control structures

python-->electrons

1's complement
1111-->0000
1000-->0111

    1 1 0 1
    1 0 0 0
(1) 0 1 0 1

sign exponent mantissa

-1^x 2^(x-127)

IEEE 755
2^(x-63)

(x-63)

0001110011
0 0011100 11
-1^0=1
2^(28-63)=

11=1/2+1/4=3/4

754
2^29-127

1.75=1+3/4

AND
OR
NOT

XOR=
XOR-->NOR

XNOR:
110
011
010

OR
00 0
01 1
10 1
11 1

AND
00 0
01 0
10 0
11 1

1 AND 1=1

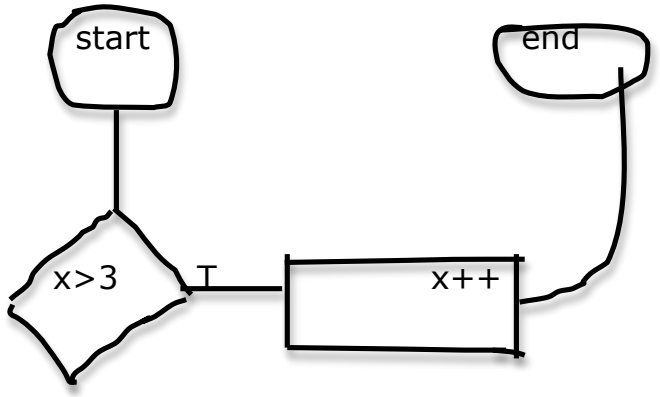
NOT
1 0
0 1

logical
!=
||
&&
!

bitwise
~

while()
{
fool();
}

do
{
fool();
}while();



if(x>3){
x++
}

Conflict exams in 4070
Others announced tonight
My office hours: Tu 2-4pm@miaZa's
(THIS WEEK ONLY)

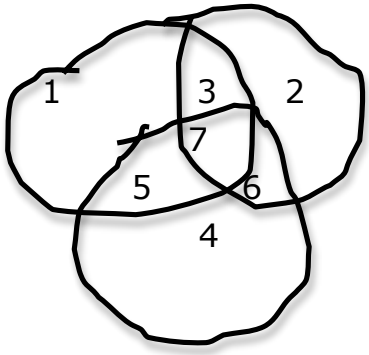
Hamming codes

IEEE Floating point

Halting problem

Parity Bits
# of 1's in a set of bit's

even parity-->even # of 1 bits

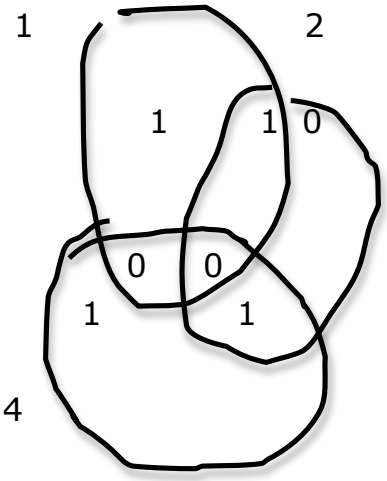


P4 P2 P1
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

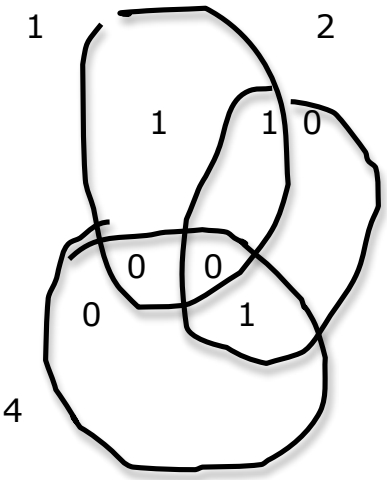
1, 2^n - 1

d3 d2 d1 p4 d0 p2 p1

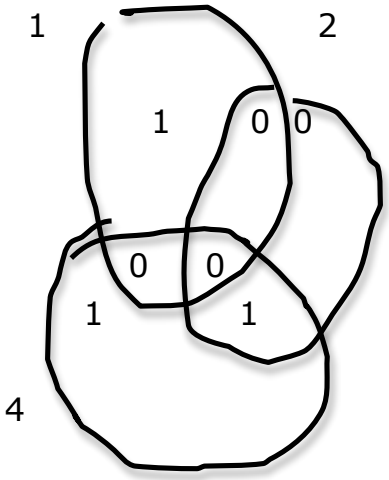
x7 x6 x5 x4 x3 x2 x1



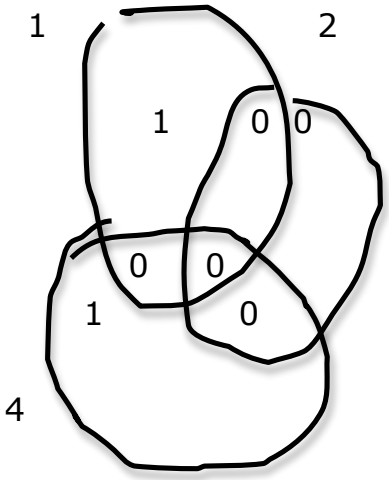
C4 C2 C1
0 0 0



C4 C2 C1
1 0 0



C4 C2 C1
0 1 1



C4 C2 C1
1 0 1

Halting problem
This sentence is false.

f(n)=-f(n-1)

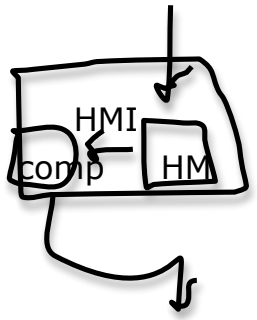
lim f(n)=?

n-->∞

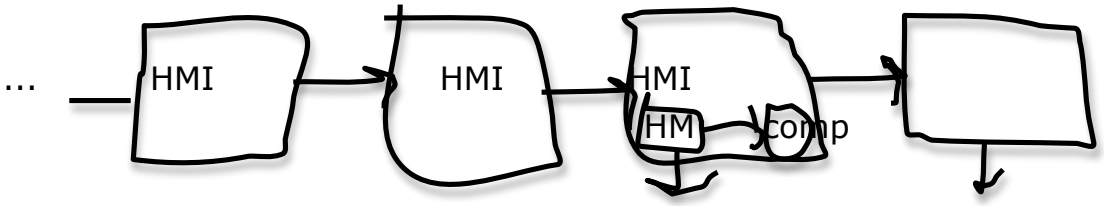
program
and input to program --> HM

-->halt or not halt

some program and its input



output complement of HM
run forever if answer of HM is yes



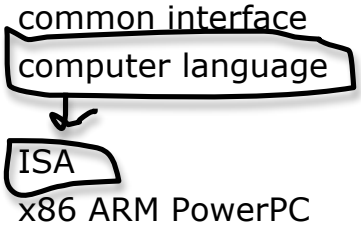
yes
no

yes
no
yes

Writing truth tables

for and while loops

layers of abstractions



x86 ARM PowerPC

-->different implementations

IEEE floating point

1101.11011
1.10111011X2^3

1=neg 8 bits
0=pos 23 bit
0 10000010 101110110...0
1+1*10^-9=1
-1+(1+1*10^-9)=0
(-1+1)+1*10^-9=1*10^-9

-3.125
0.125*2=0.250-->0
x2=0.500-->0
x2=1.000-->1
-1=0

1 11.001000...0
1.10010000*2^1-->exponent bits
10000000

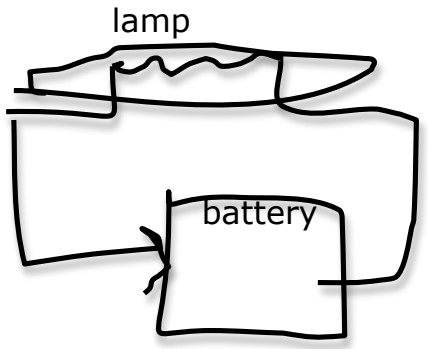
(-1)^(sign)*1.(mantissa)*2^(exponent-127)
2^0 2^-1 2^-2 ...

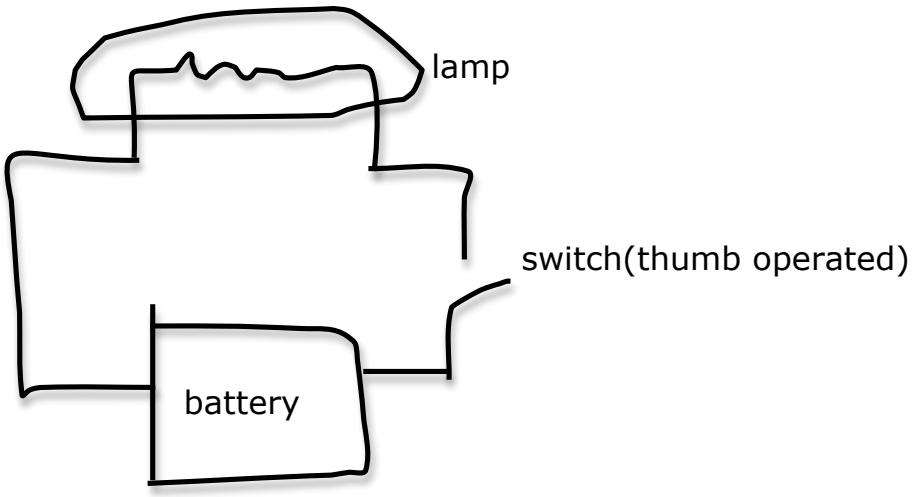
-HW 2 re grade requests are due next Monday
-pick up your exam after this lecture
-exam re grade requests are due next monday too

transistors
functions on bits (gates)
two voltage levels=1bit
circuits
devices
electrons
how can we build gates?

transistors
gates
optimizing Boolean expressions

	F14 (407 people)	F12 (27 people)	F12 MT2
mean	83	86	69
median	85.5	91	78
st dev	11	14	15
max	100	100	90



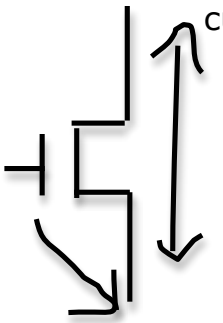


transistor=voltage controlled switch
-->really cool thing!

transistors can be used to control other transistors

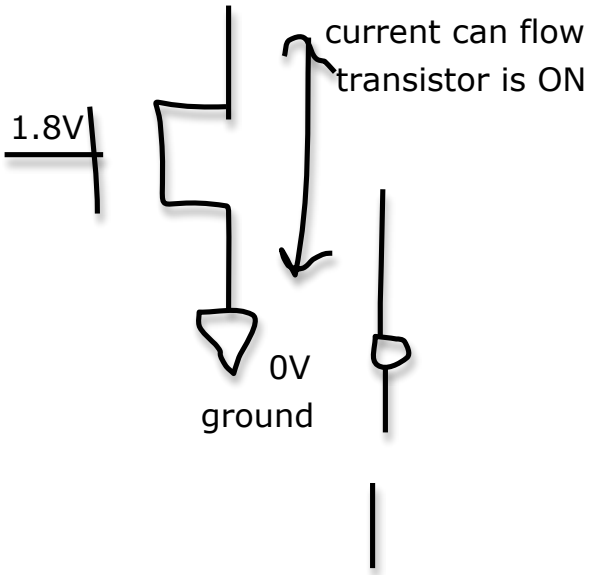
Bragging break

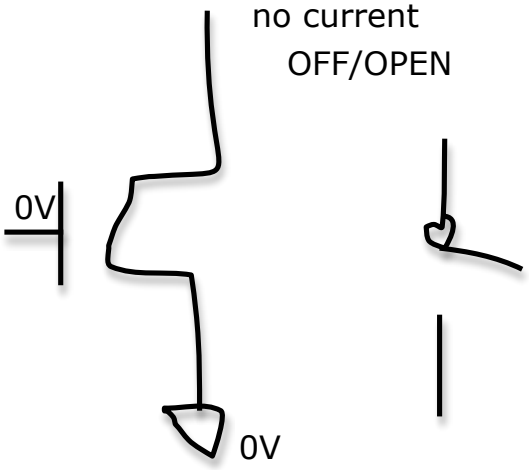
MOSFETS
FET-Field effect transistors
MOS-metal oxide semiconductors



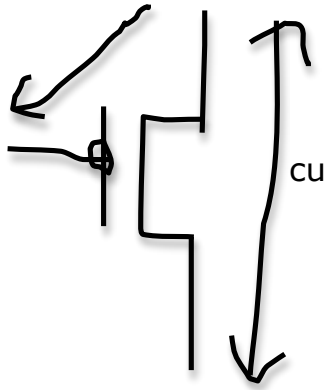
is voltage > threshold

n type MOSFET





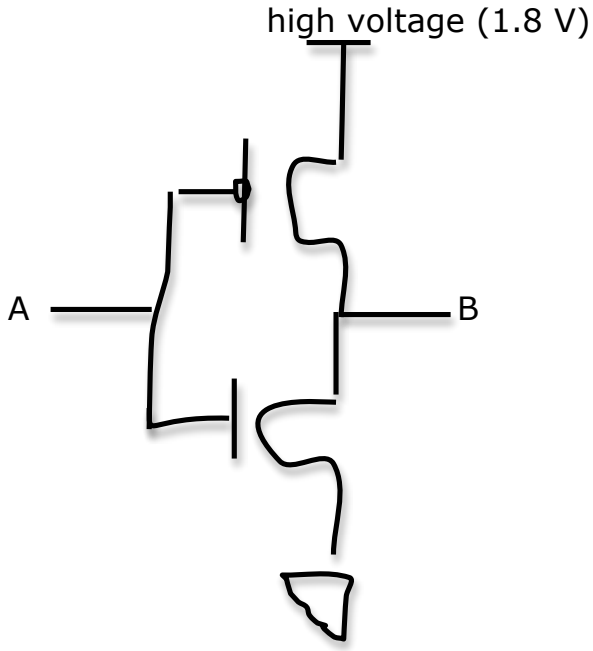
voltage>threshold



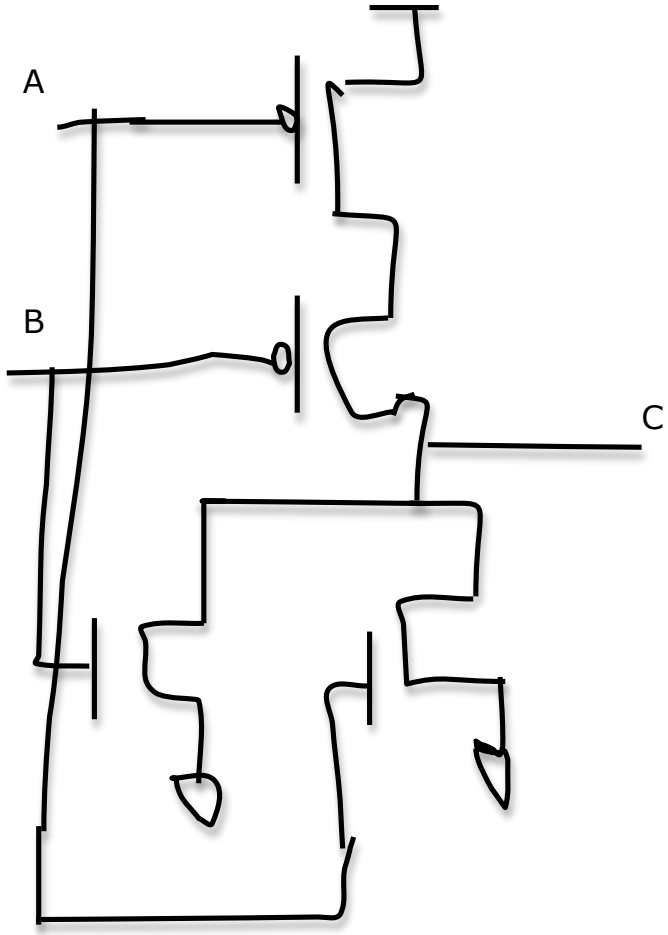
current flows if voltage exceeds threshold

p-type MOSFET

Gates

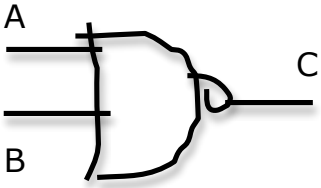


A	B
0 0V	1 1.8 V
1 1.8V	0 0V



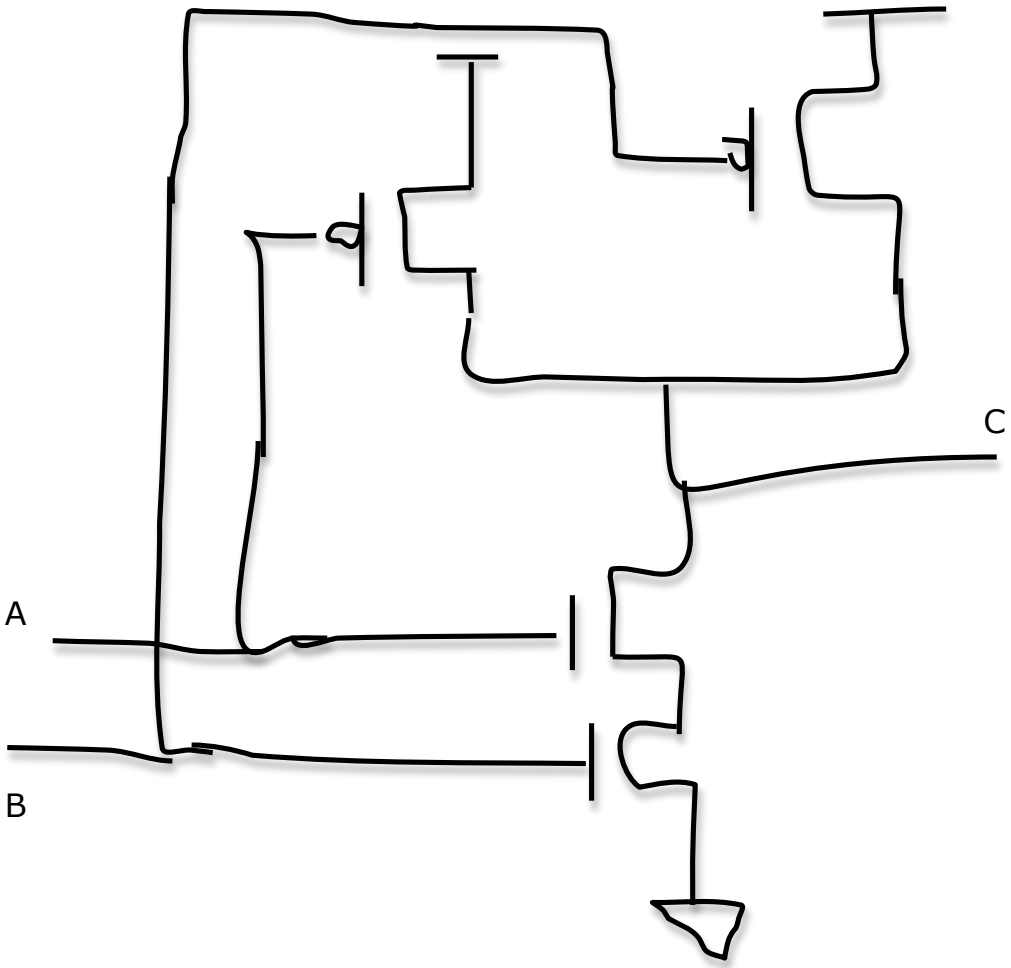
A B C
0 0 1
0 1 0
1 0 0
1 1 0

NOR



A B A+B (A+B)_ AB (AB)_

0 0 0	1	0	1
0 1 1	0	0	1
1 0 1	0	0	1
1 1 1	0	1	0



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

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

0 1 0 0
0 1 1 0
1 0 0 0
1 0 1 1 AB_C
1 1 0 1 ABC_
1 1 1 1 ABC

x F=AB_C+ABC_+ABC
y F=AB+AC
z F=A(B+C)

the answer (which is best?) depends on our metric?
area/size/cost
performance/speed
power/energy consumption

a heuristic for area

Count literals (A,A_,B,B_,C,C_)
add the number of operations (not including literal complements)
why? each input line-->2 transistors

operators-->2 transistors

form x-->9 literals, 4 operators=13
form y-->4+3=7
form z-->3+2=5

a heuristic for delay
find the maximum number of gates from any input to any output
why? each gate takes time to switch (gate delay)

Finish sample heuristics
terminology
k-maps
two level logic
[Pareto optimality]

recall: area, performance, power
example function

X: F=ABC_+AB_C+ABC
Y: F=AB+AC
Z: A(B+C)

area heuristic
-count literals
(variables of complements)
X-->9+4=13
Y-->4+3=7
Z-->3+2=5<--best

delay heuristic
-find maximum number of gates from any input to any output
X-->3 (NOT, AND, OR)
Y-->2 (AND, OR)
Z-->2(OR, AND)
Y, Z best

Terminology
Literal-variable or its complement
Sum-several terms OR'd together
Product-several terms AND's together
Minterm on N inputs-product in which each variable or its complement appears once
Maxterm [on N inputs]-sum in which each variable or its complement appears exactly once
Examples:

A+B_+C_
A+B+C_
not AB+C

A B C A+B_+C_
0 0 0 1
0 0 1 1
0 1 0 1
0 1 1 0
1 0 0 1
1 0 1 1
1 1 0 1
1 1 1 1

sum of products (SOP)-sum (OR) of products (AND) of literals
example: AB+BC
but not: A(B+C)+D

product of sums (POS)-product of sums of literals
Example: (A+B)(B+C)
but not: (A+BC)D
canonical SOP: sum of minterms

canonical: unique way to write expressions

canonical POS: product of maxterms
implicants
A function G is an implicant of a function F if G-->F.
in other words every row in G's truth table with output 1 also has output 1 in F's truth table

in digital design, implicants must be products of literals

A F G
0 0 1

1 1 1
F-->G

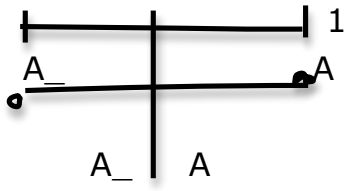
first step in simplification
given an implicant G of F
can we remove any of the literals from G and get another implicant

F=ABC_+AB_C+ABC
ABC_
implicant of F

A	B	C	F	BC_	AC_	AB
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	1	0	0
0	1	1	0	0	0	0
1	0	0	0	0	1	0
1	0	1	1	0	0	0
1	1	0	1	1	1	1
1	1	1	1	0	0	1

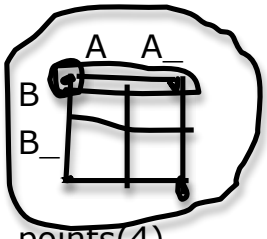
AB
AC_
implicant of F
if we cannot remove any literals from G, G is a prime implicant of F

Karnaugh maps (K-maps)

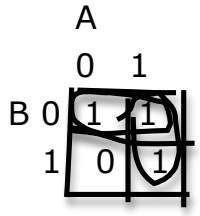


A 1D hypercube
How many implicants on one variable? A, A_, 1

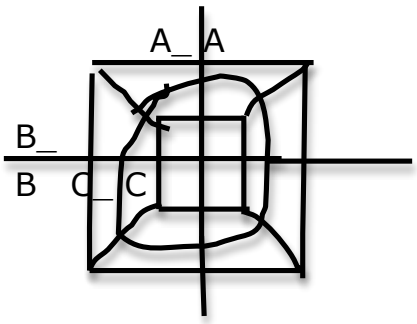
F 0 1



points(4)
(AB, AB_, A_B, A_B_)
edges(4)
A, A_, B, B_
face(1)
1

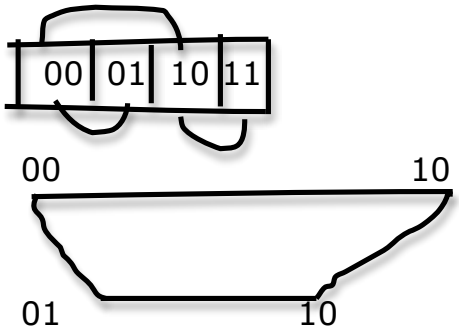


X=A+B_

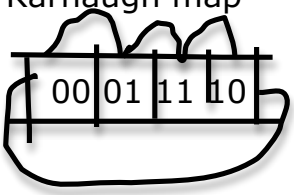


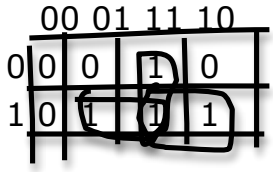
3^3=27
points(8)
edges(12)
faces(6)
cube(1)

Veitch

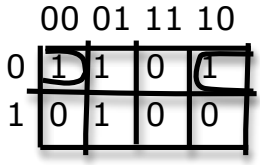


Karnaugh map





C_ in=A B + B C_ in + A C_ in

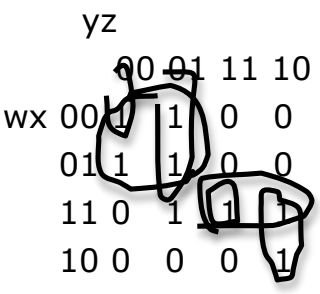


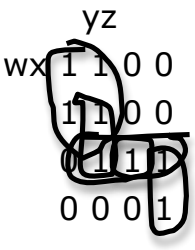
If some implicant is the only prime implicant covering some input combination, we say that the implicant is essential.

F=B_C_+A_B

Exam re grade requests are due NOW.

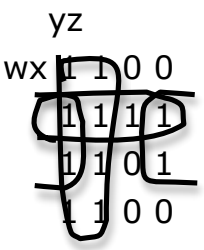
Problem statement-->truth table, k-map-->boolean expression(SOP, POS) -->circuit



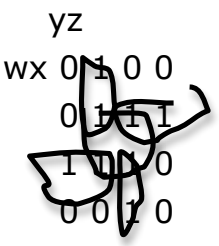


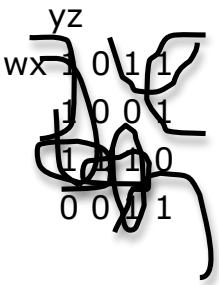
f(w, x, y, z)=w'y'+wyz'+wxz

implicant: all legal loops
prime implicant: all biggest loops
essential prime implicant: prime implicant must be there-biggest loops that do not include cells covered by another loop
min SOP



y'+w'x+xz'



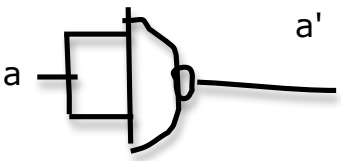


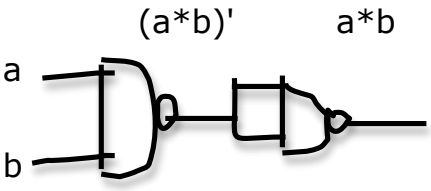
A + wxz + wzy'
A + xy'z' + wxz

2-level circuit

{AND, OR, NOT}
{NAND}

(a*a)'=a'



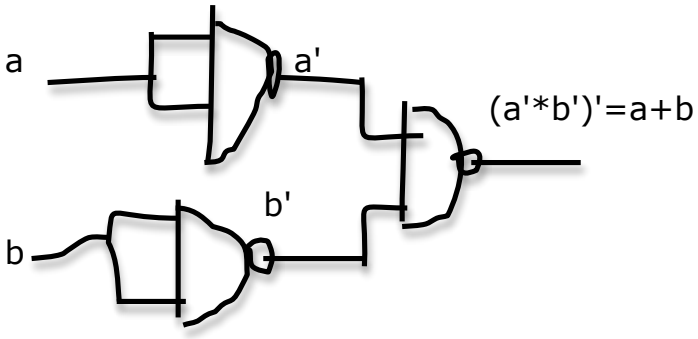


OR
(a*b)'=a'+b'

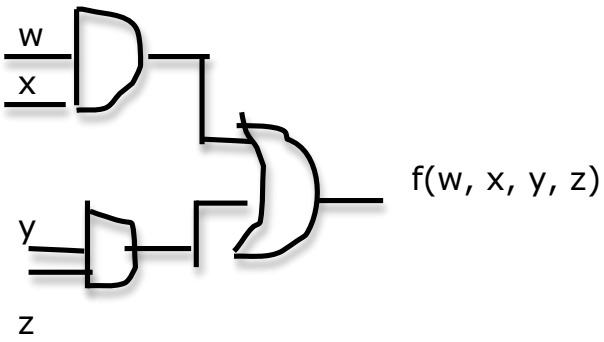
a b (a*b)' (a'+b')
0 0 1 1

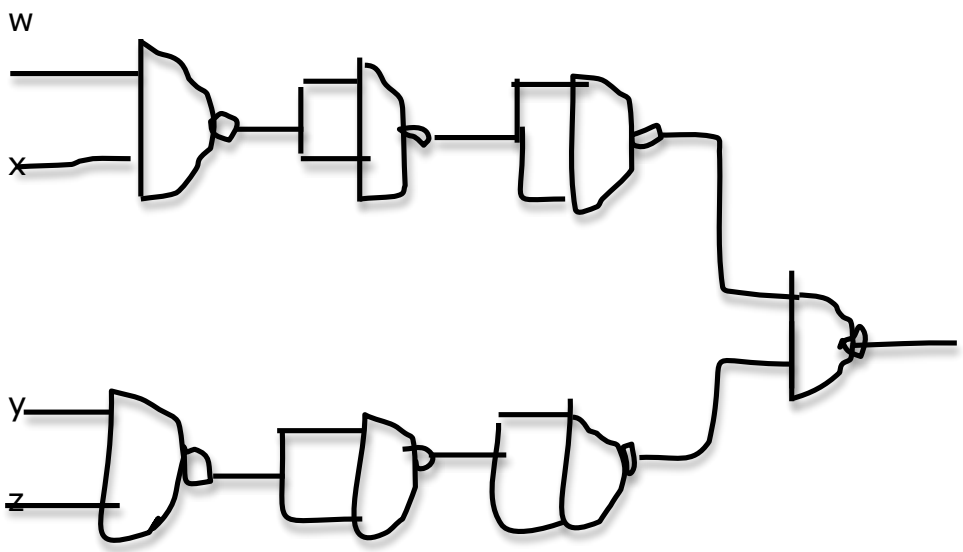
0 1 1 1
1 0 1 1
1 1 0 0

(a'*b')'=a+b

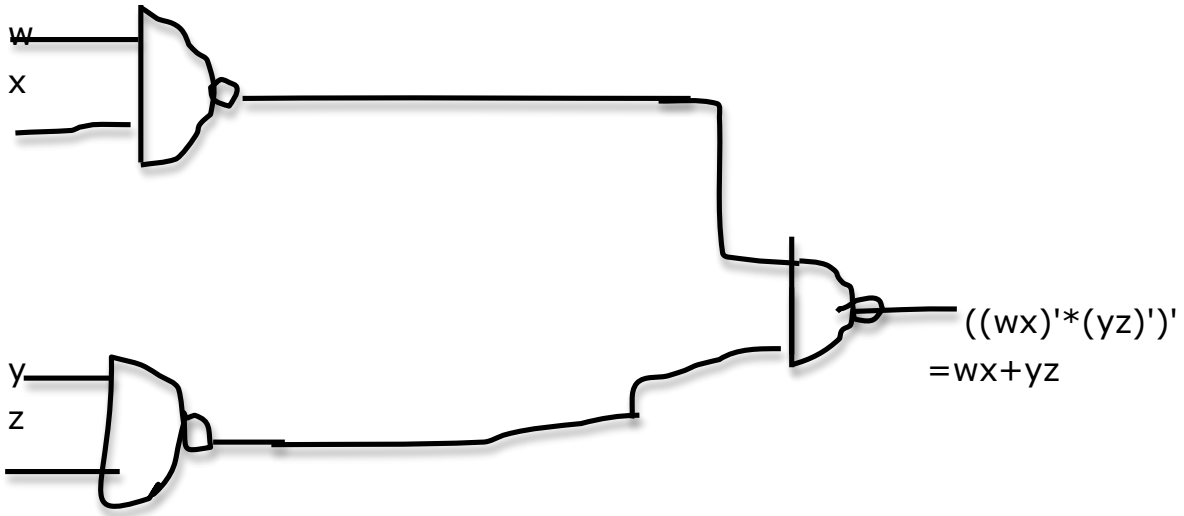


f(w,x,y,z)
=wx+yz





NAND OR

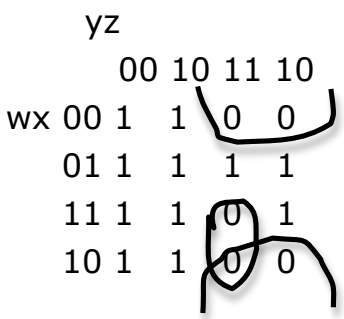


f(w,x,y,z)=(w+x)(y+z)

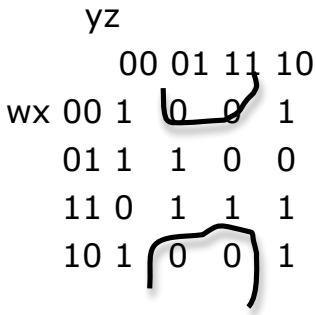
POS

OR-AND

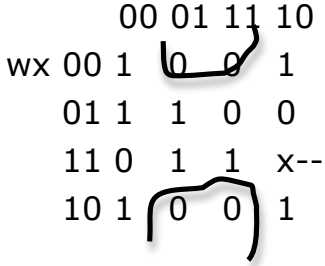
NOR



(x + y')(w'+y'+z')
0 + 1'=0



(x+z')(w+x'+y')(w'+x'+y+z)



Lab 5 is due on Friday in lecture. All office hours from 9/25 to 10/3 will be in 4022.
Pick up lab 6 kit on Th, F in office hours

dec digits BCD
0 0000
1 0001
...
9 1001

1010
...
1111

A B C f
0 0 0 0
0 0 1 1
0 1 0 0 1 01 x
0 1 1 0 1 10 x
1 0 0 0
1 0 1 1
1 1 0 0
1 1 1 1

AB

	00	01	11	10
C 0	0	1	x	x
1	0	1	1	0

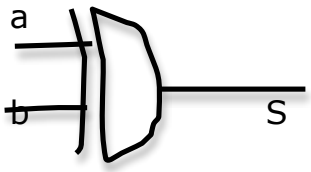
0 1 1 0
0 1 1 0

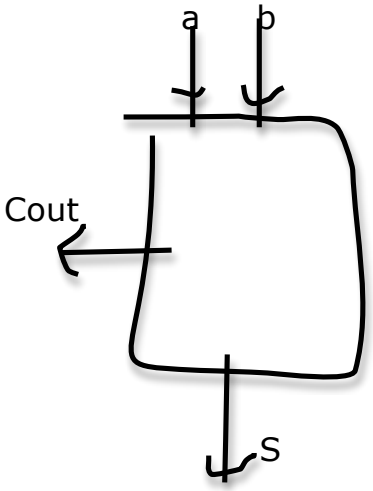
F=A'B+BC 00
t=A'B+BC+AB'C' 01
F=B 10
F=B+AC' 11

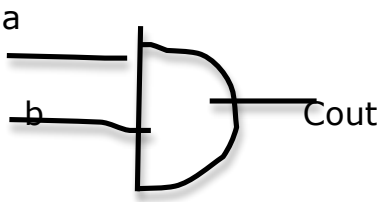
A B Sum S Cout
0 0 0 0 0
0 1 1 1 0
1 0 1 1 0
1 1 1 0 0 1

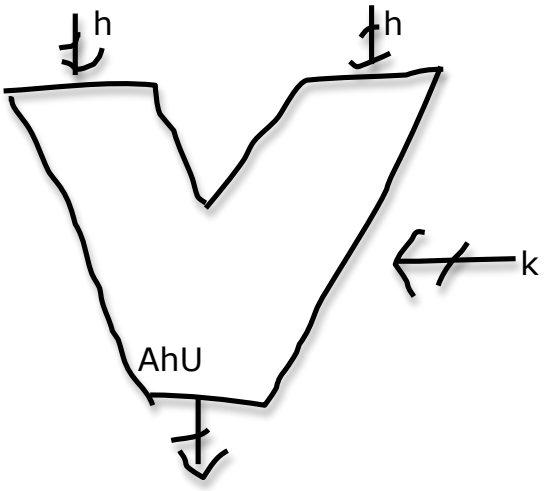
Cn Cn-1 C2 C1
an-1 an-2...a2a1a0
\+ bn-1 bn-2 b2b1b0
_____
Sn-1 Sn-2 S1 S0

S=a' b + a b'=a⊕b
Cout=ab









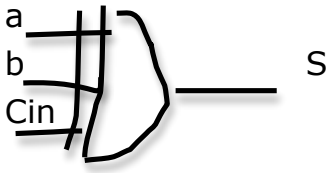
h

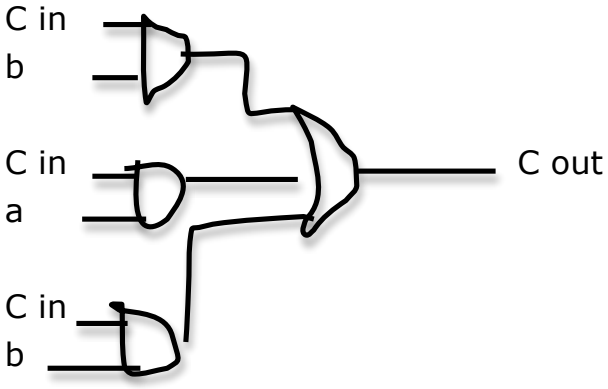
Cin	ab = 00	01	11	10
0	0	1	0	1
1	1	0	1	0

S=C_ in a' b' + C_ in' a' b + C_ in a b + C_ in a b' = C_ in ⊕ a ⊕ b

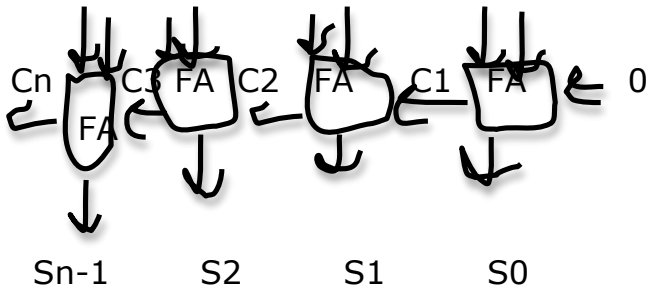
Cin	ab = 00	01	11	10
0	0	0	1	0
1	0	1	1	1

C_ out=C_ in b + C_ in a + a b



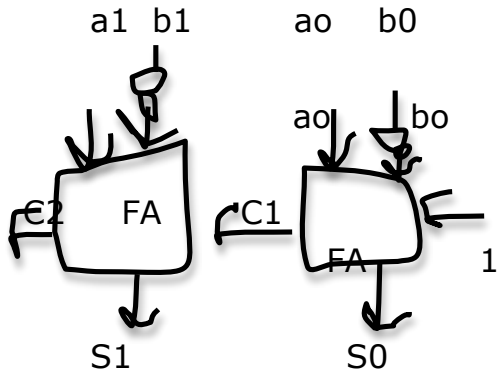


an-1 bn-1 a2b2 a1b1 a0b0

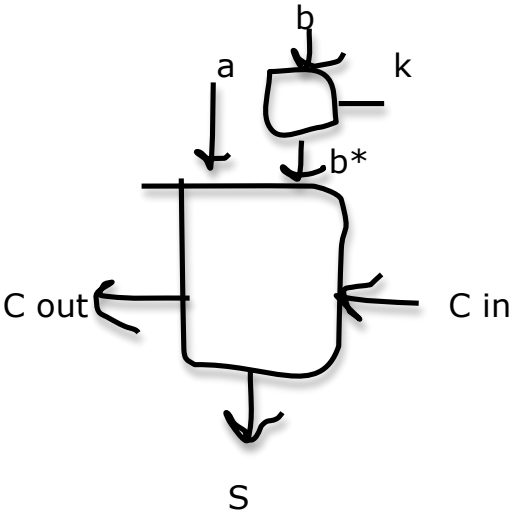


A-B=A+(-B)

NOT(B)+1



S=A-B

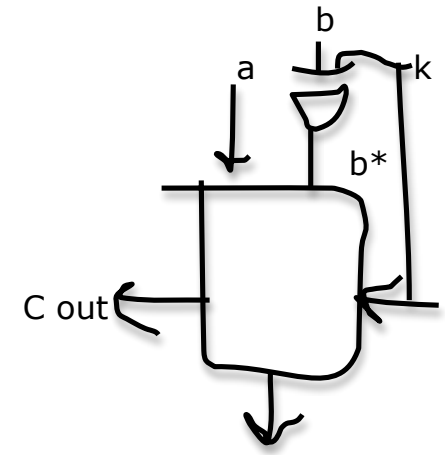


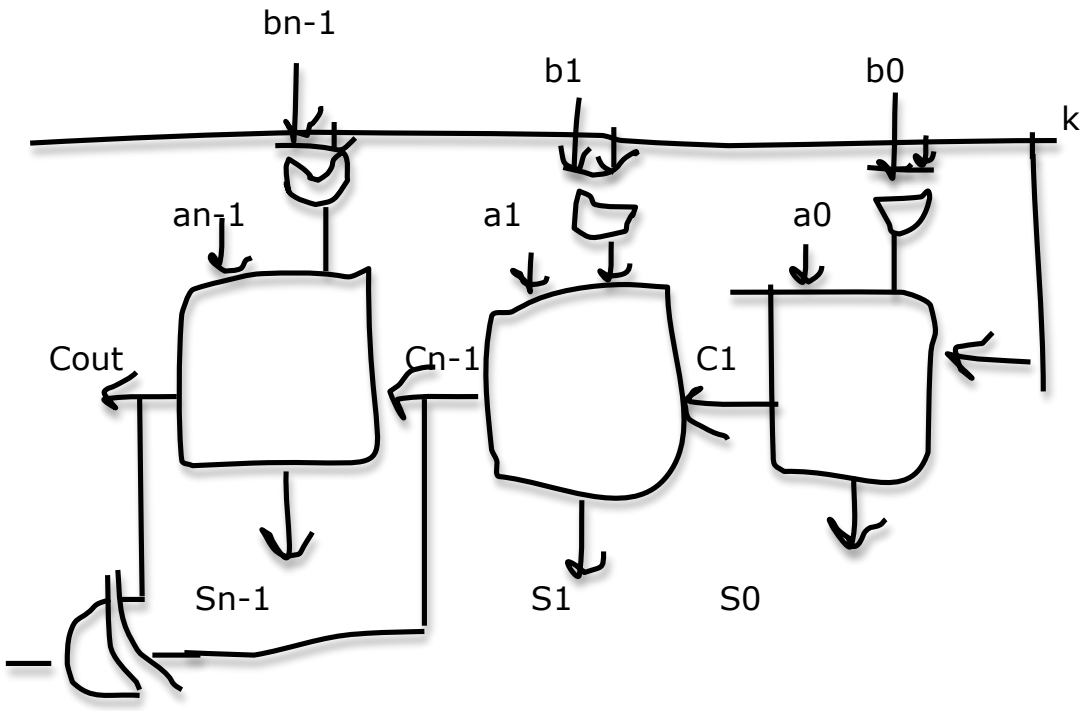
b k b*
0 0 0

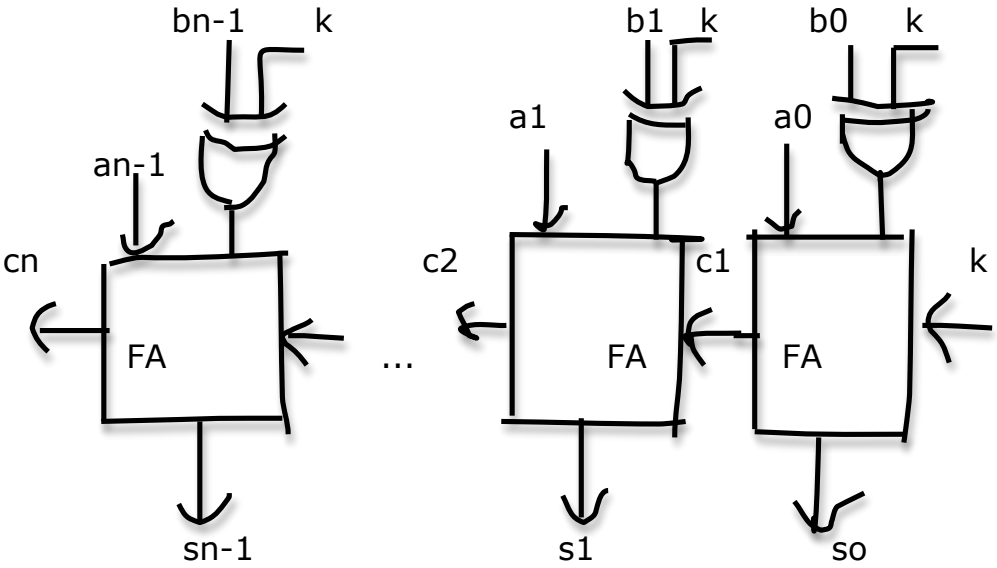
1 0 1
0 1 1
1 1 0

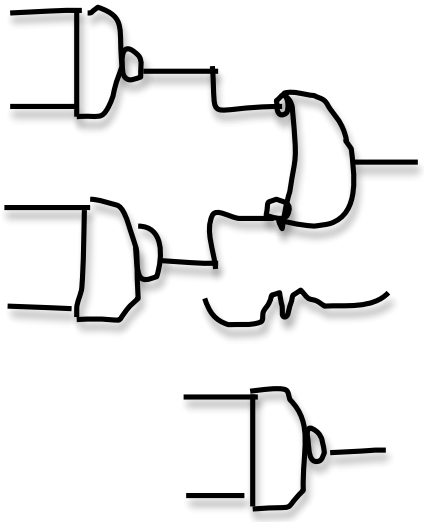
k b*
0 b
1 b'

b*=b ⊕ k







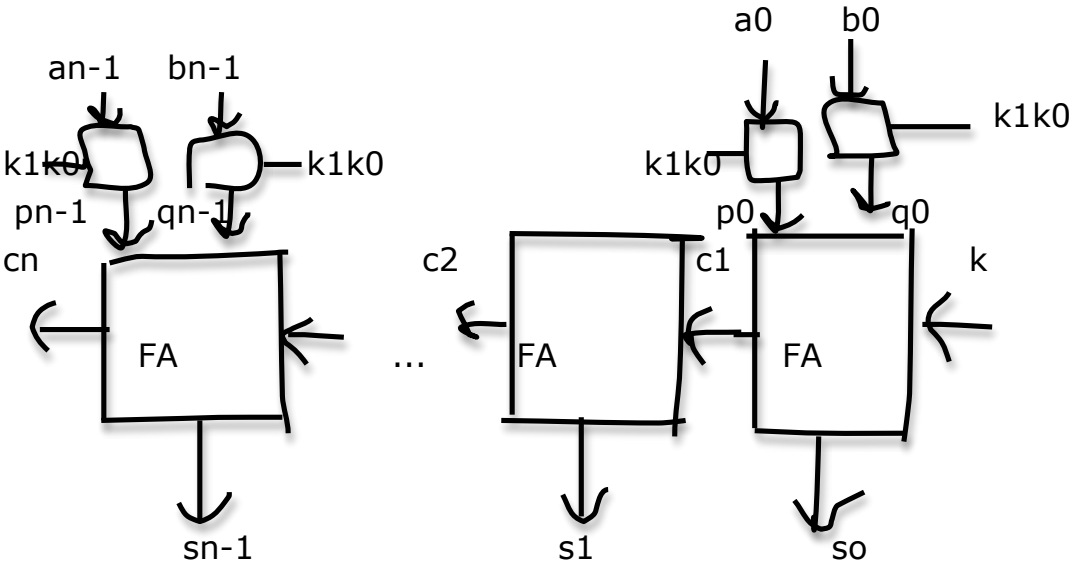


a' + b'=(a b)'

k=0, ADD
k=1, SUB

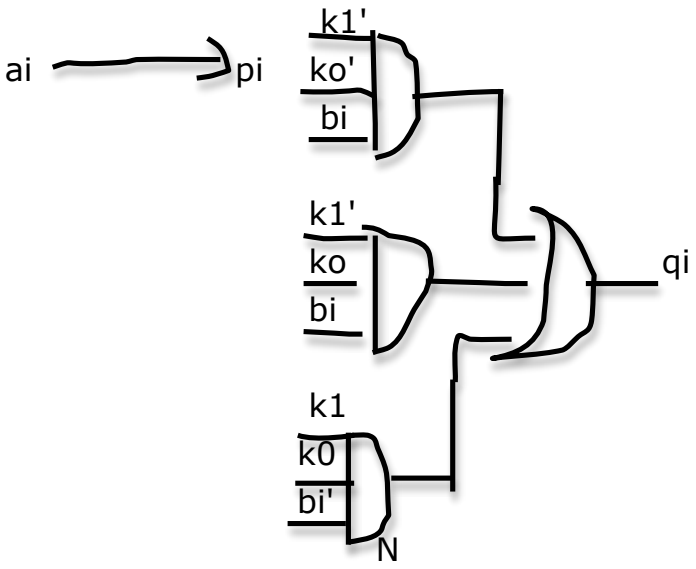
{A plus B
{A plus B plus 1
F= {A
{A minus B

K1	K0
0	0
0	1
1	0
1	1

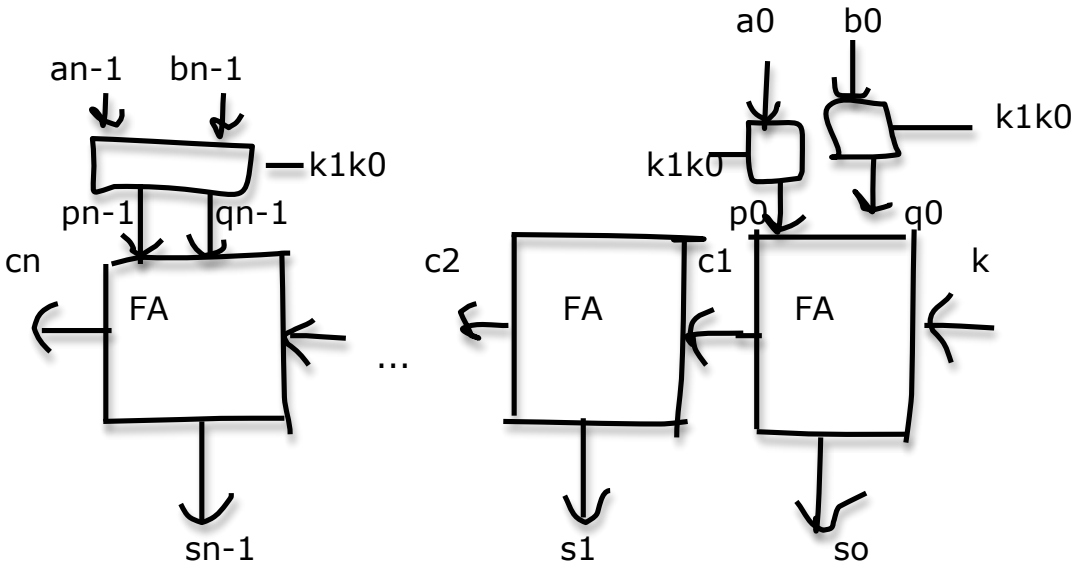


k1	k0		pi	qi	co
0	0	A+B	ai	bi	0
0	1	A+B+1	ai	bi	1
1	0	A	ai	0	0
1	1	A-B	ai	bi'	1

pi=ai
qi=k1'k0'bi+k1'k0bi+k1k0bi'
co=ko

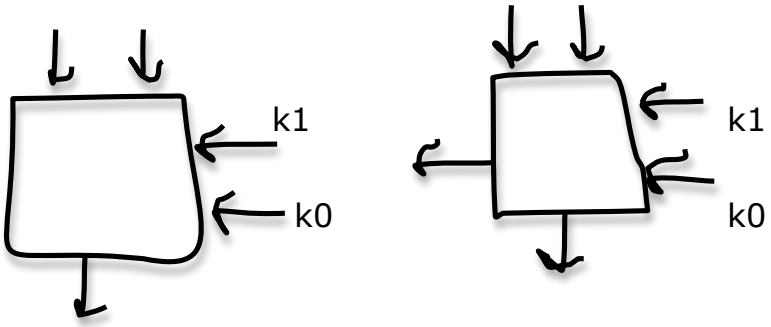


p(ai,bi,k1,ko)
q(ai,bi,k1,k0)



F={A AND B
{A OR B
{A NAND B
{0

an-1 bn-1 a0 b0

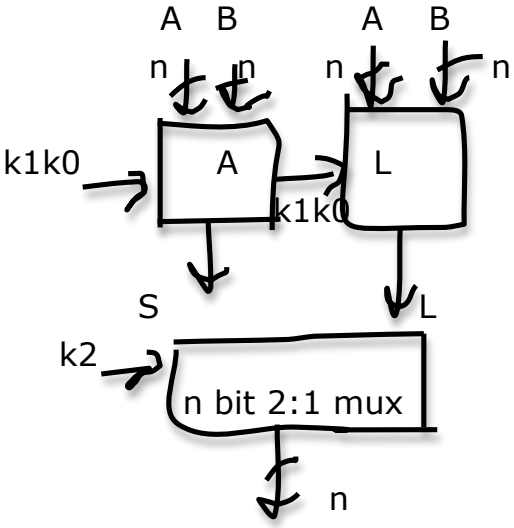


l(ai,bi,k1,k0)

	AND	OR	0	NAND
	00	01	11	10
aibi 00	0	0	0	1
01	0	1	0	1
11	1	1	0	0
10	0	1	0	1

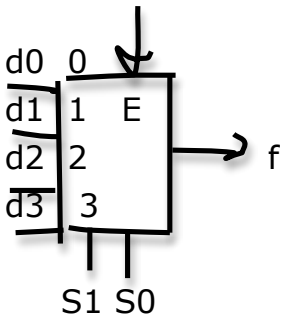
l=aibik1'+aik1'k0+bik1'k0+ai'k1ko'+bi'k1k0'

k2 k1 k0
0 arithmatic
1 logical

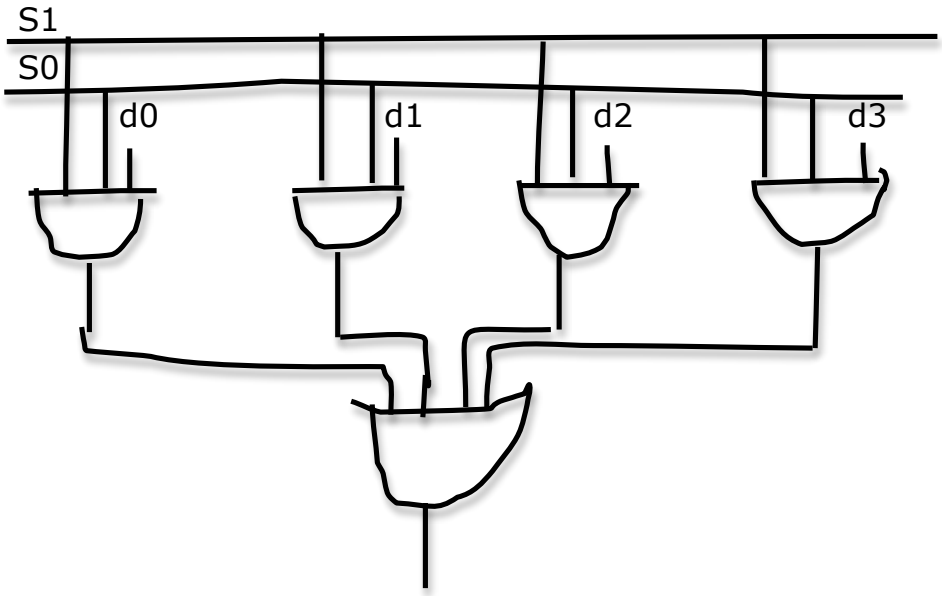


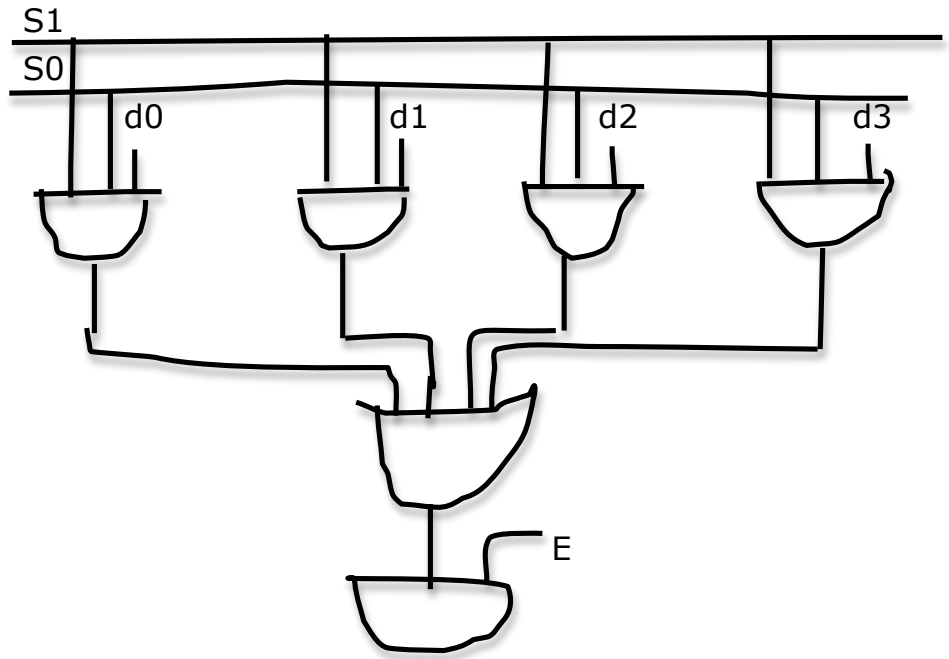
1 bit 4:1 mux

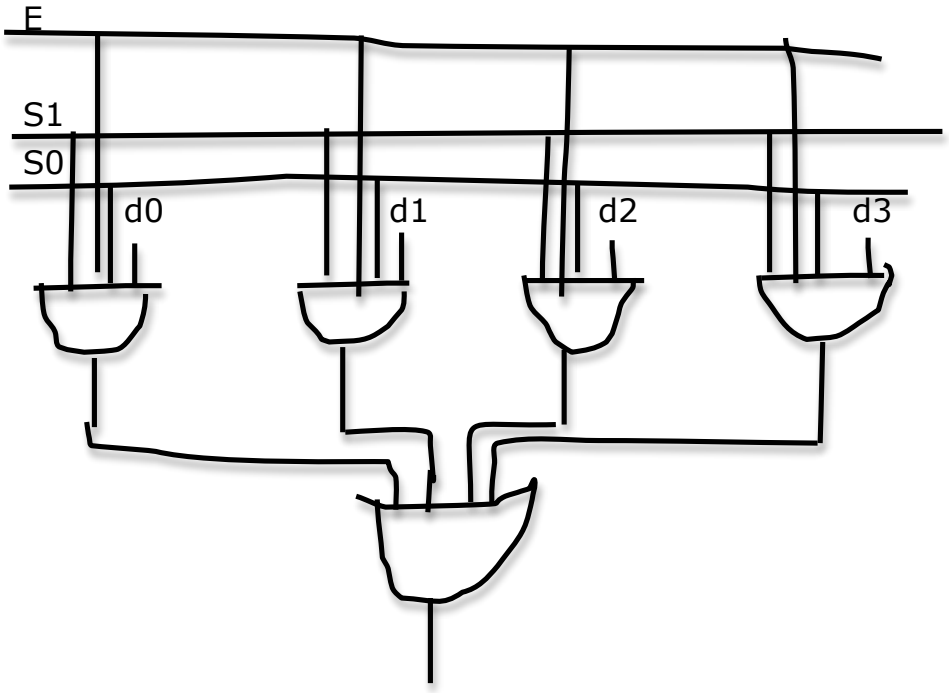
E	s1	s0	f
1	0	0	do
1	0	1	d1
1	1	0	d2
1	1	1	d3
0	x	x	0



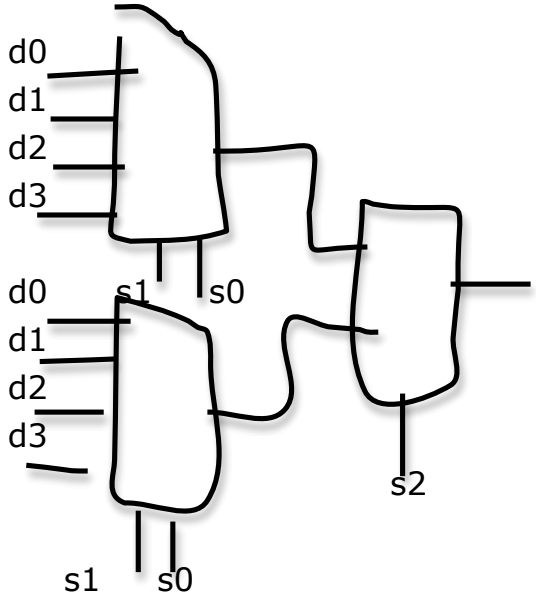
f=S1'S0'd0+s1's0d1+s1s0'd2+s1s0d3







2X 4:1 mux
1X 2:1 mux
-->8:1 mux


d0
d1
d2
d3
s1
s0
d0
d1
d2
d3
s2
s1
s0

This week's TA's office hours are in 4022
Lab6 is due in office hours
Demo your working circuit to a TA

x=xn-1xn-2...x2x1x0
y=yn-1yn-2...y2y1y0

_____

xn-1<yn-1: x<y
xn-1>yn-1: y<x
xn-1=yn-1:

x 010
y 011
  x?y
x0=0
y0=1

x0<y0

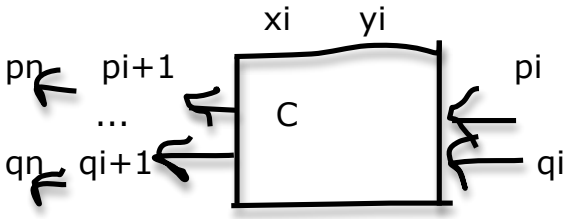
x<y

x=xn-1xn-2...x1x2
y=yn-1yn-2...y1y0

x0=y0
xo<y0
xo>y0

x1<y1
x>y

x1>y1
y>x



pn qn pi qi
0 0
0 1 x<y
1 0 y<x
1 1 x=y

piqi \ xiyi	00	01	11	10
00	x	x	x	x
01	<	<	<	>
11	=	<	=	>
10	>	<	>	>

x<y
<> =
xi=0
yi=1

xi>yi

piqi \ xiyi	00	01	11	10
00	x	x	x	x
01	0	0	0	1
11	1	0	1	1
10	1	0	1	1

pi+1

piqi	xiyi 00	01	11	10
00	x	x	x	x
01	1	1	1	0
11	1	1	1	0
10	0	1	0	0

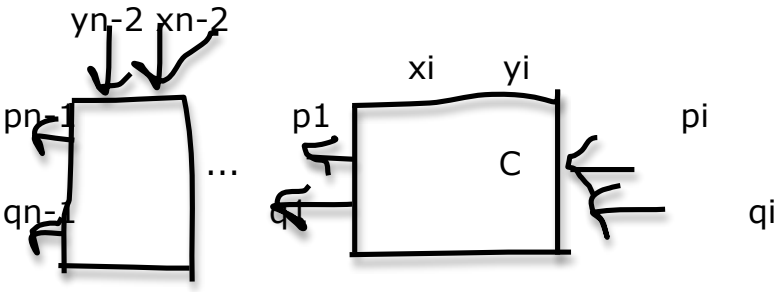
qi+1

pi=3 product terms
qi=3 product terms

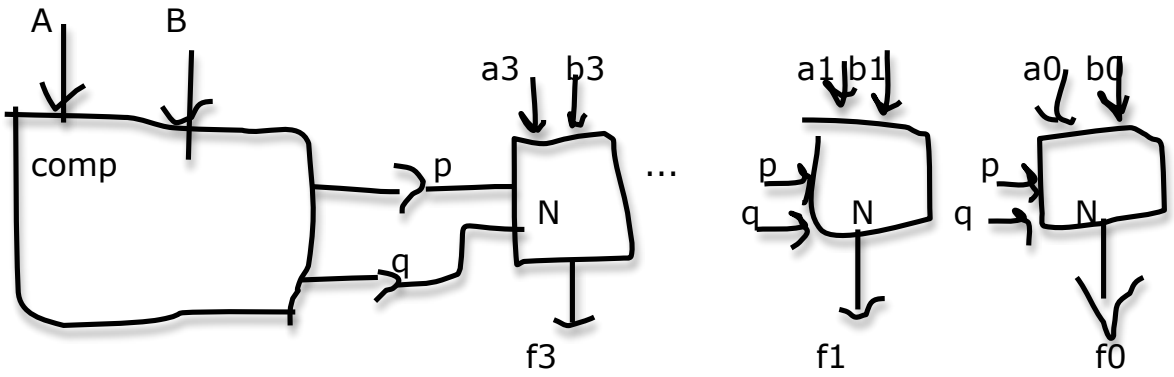
xn-1 yn-2 ...
yn-1 yn-2 ...

xn-1 yn-1
0 0 xn-2...x0 ? yn-2...y0
0 1 x>y
1 0 y>x
1 1

x+2^(n-1) ? y+2^(n-1)
if x+2^(n-1)<y+2^(n-1)
x<y



A=a3a2a1a0
B=b3b2b1b0
F={A NAND B if A<B
{0 A>B
{A OR B A=B



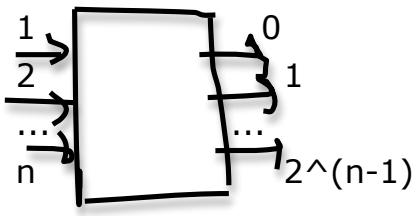
pq
00 A<B
01 A=B
11 A>B

pq	aibi 00	01	11	10	
00	1	1	0	1	NAND
01	0	x	1	x	OR
11	0	0	0	0	
10	x	x	x	x	

q'b'

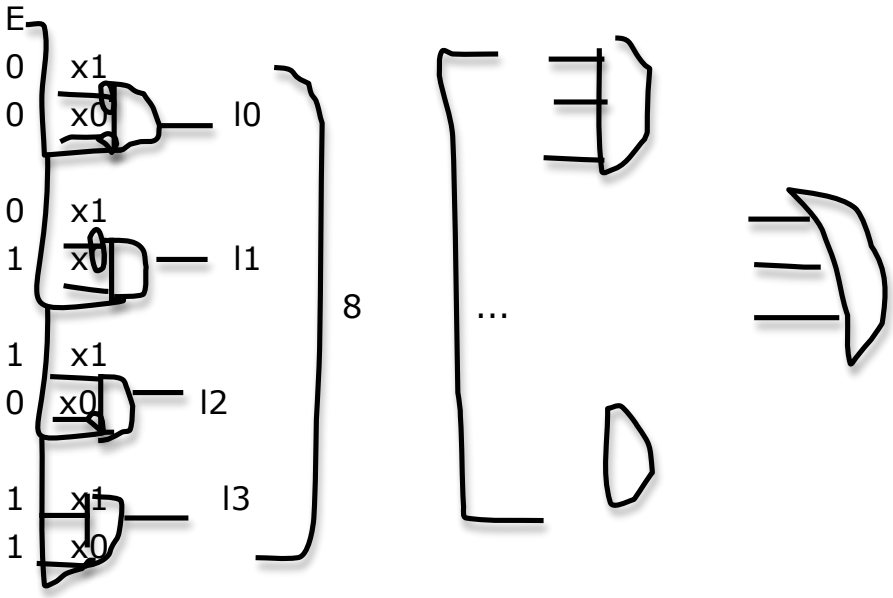
x1	x0	l0	l1	l2	l3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

decoder



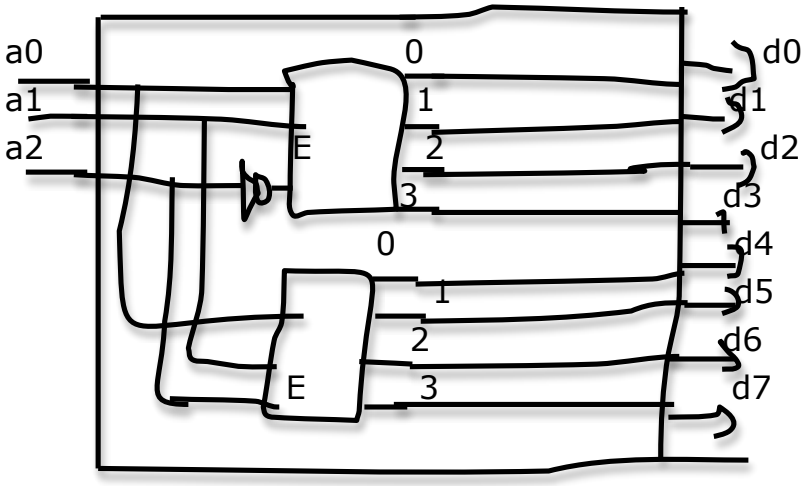
2^n outputs

n=2



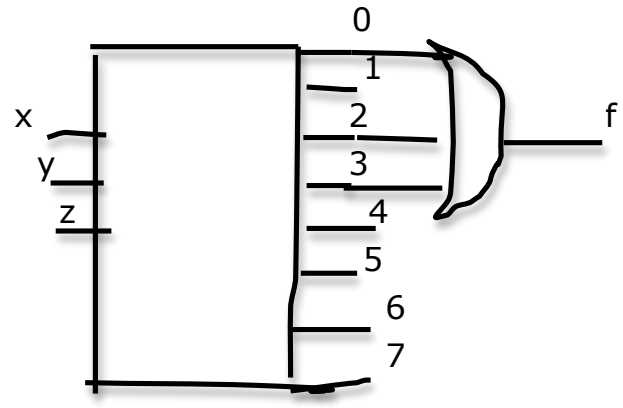
3-8 decoder

2-4



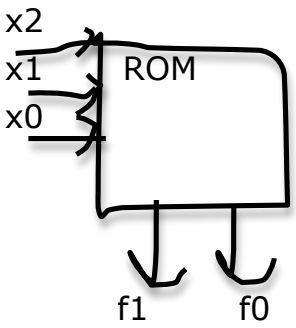
f(x,y,z)=x'y'z'+x'yz'+x'yz

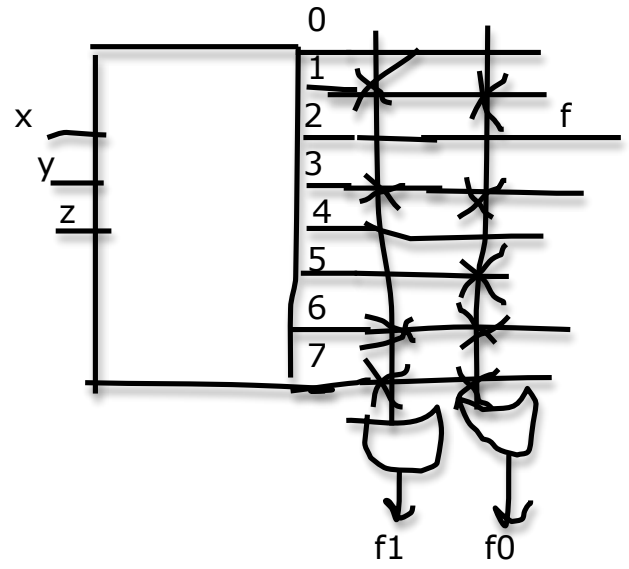
m0 m2 m3



x2	x1	x0	f1	f0		
0	0	0	0	0		
0	0	1	1	1	m1	m1
0	1	0	0	0		
0	1	1	1	1	m3	m3
1	0	0	0	0		
1	0	1	0	1	m5	
1	1	0	1	1	m6	m6
1	1	1	1	1	m7	m7

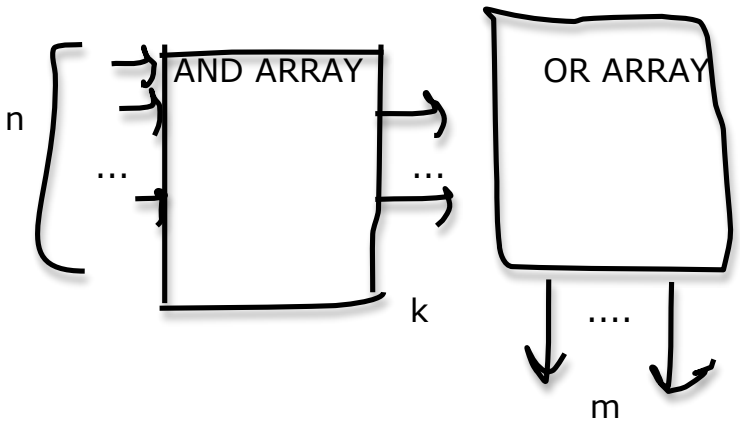
8 2 bit

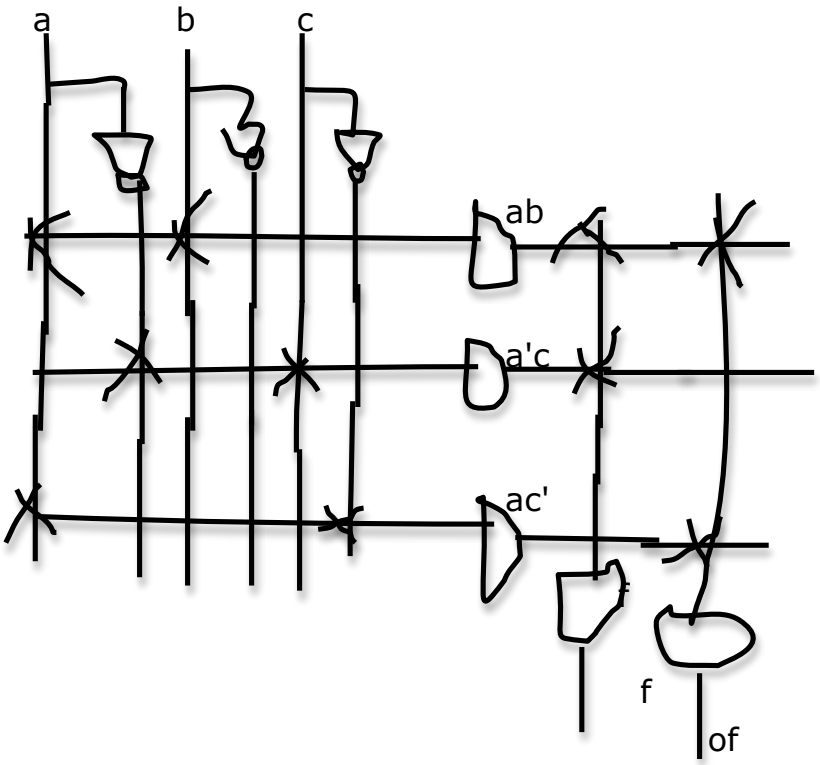




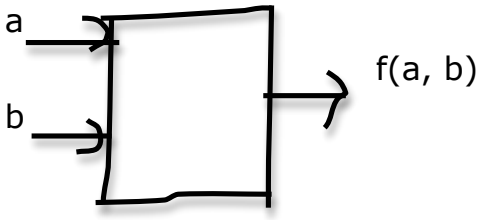
f(a, b, c)=m1 + m3 + m6 + m7=ab+a'c

PLA

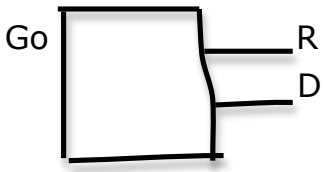


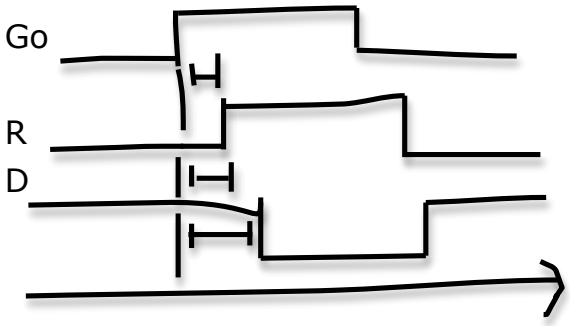

a
b
c
ab
a'c
ac'
f
of

Midterm 2 is on 10/14
Conflict reporting due day: 10/7
No lecture next Friday :)
7-9 pm

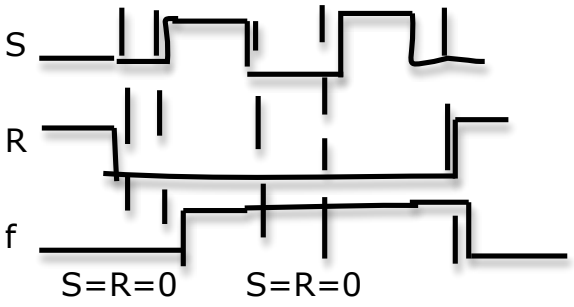


g(inputs, state)





inputs: S R
output: f{=1 iff S was 1 more recently than R
{=0 otherwise



f=0 f=1

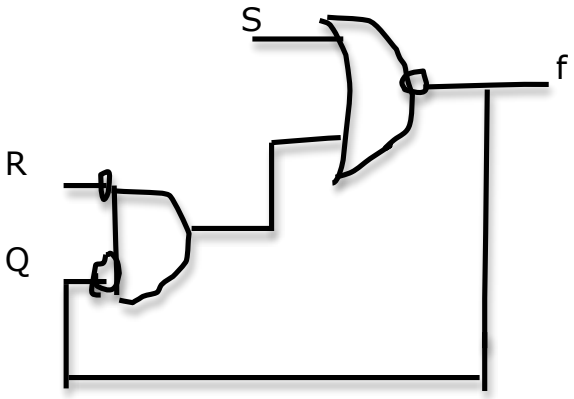
Q=1 iff S was 1 more recently than R

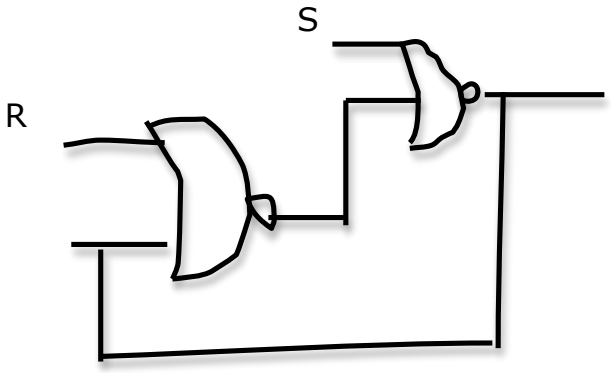
S, R, Q

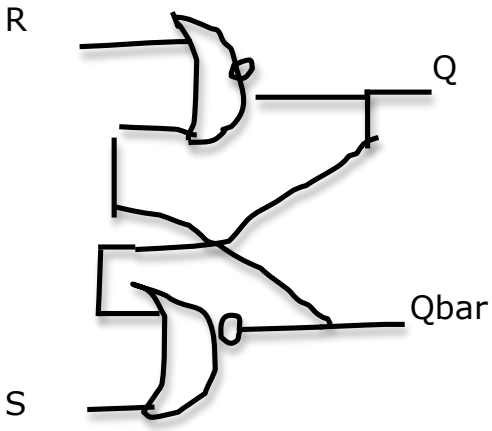
Q	S, R = 00	01	11	10
0	0	0	x	1
1	1	0	x	1

SR=11-->X

f=S+QR'

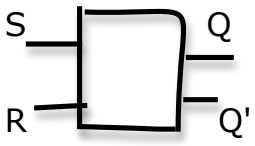






R	S	Q	Q bar	
0	1	1	0	set
1	0	0	1	reset
0	0	{1 {0		hold
		{0 {1		
1	1	x	x	

SR latch



next state

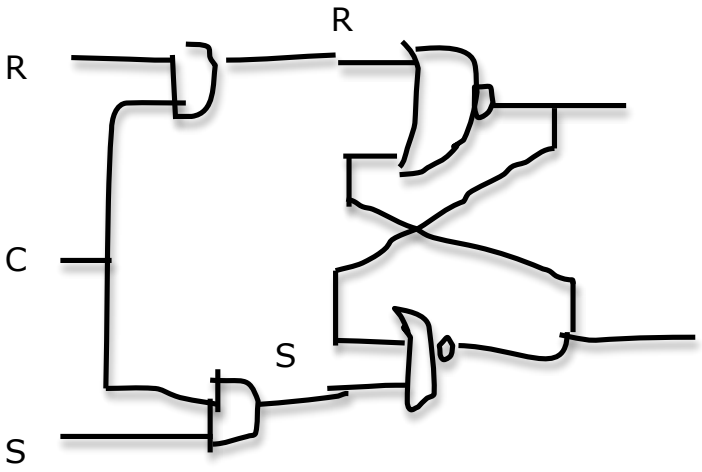
S	R	Q+	
0	0	Q	
0	1	0	reset
1	0	1	set
1	1	forbidden	

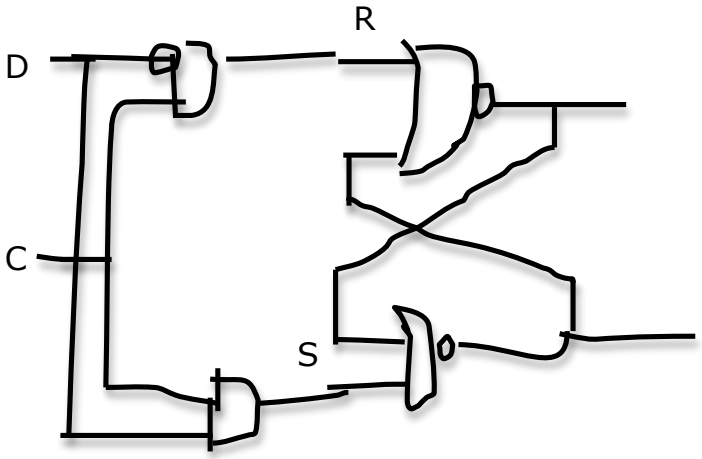
11-->00

excitation

Q-->Q+	S	R

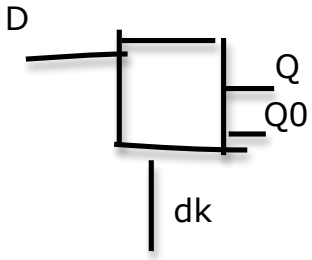
0-->0 0 x
0-->1 1 0
1-->0 0 1
1-->1 x 0





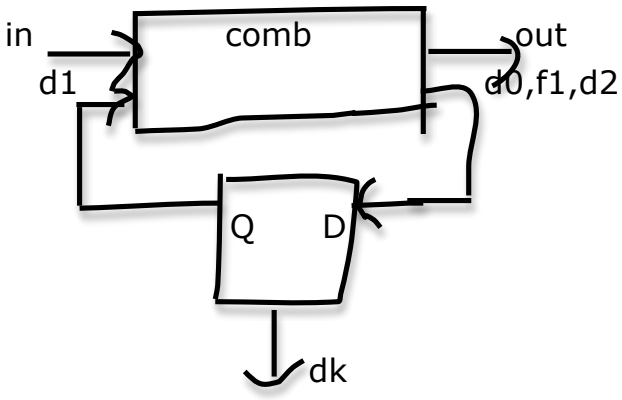
D R S Q
0 1 0 0
1 0 1 1

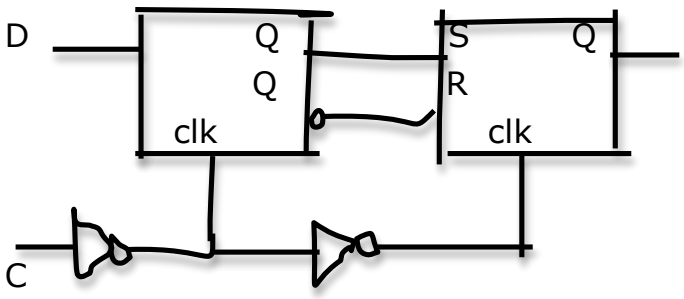
gated D latch



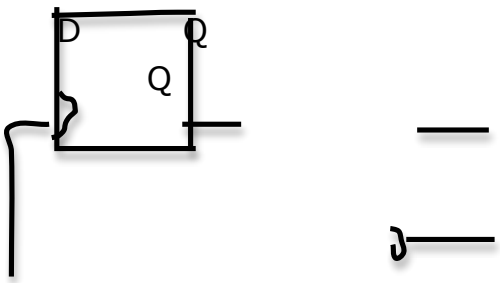
C	D	Q+
1	0	0
1	1	1
0	x	Q

Q	Q+	D
0	0	0
0	1	1
1	0	0
1	1	1

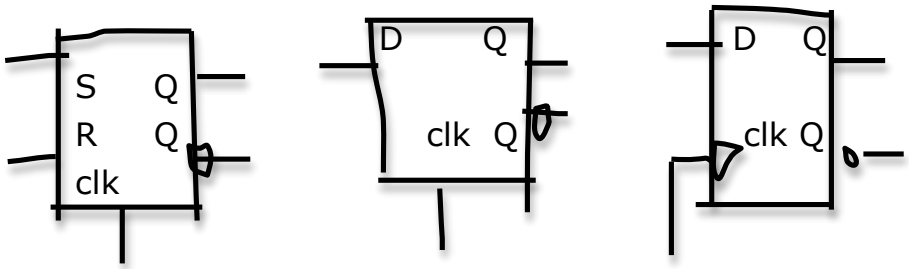




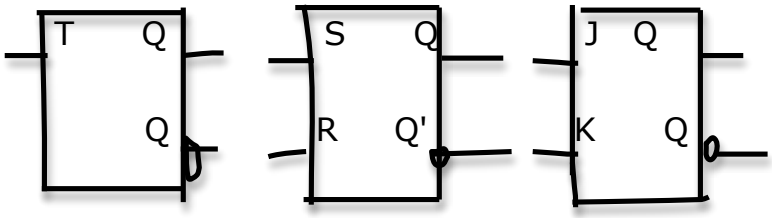
D=flip flop



Conflict exam notification due 10/7
no lecture on Friday
lab 7 is due on Friday by 10am in room 4034



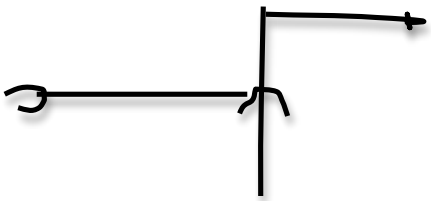
D Q+
0 0
1 1



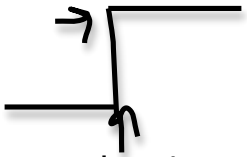
T Q+
0 Q
1 Q'

S R Q+
0 0 Q
0 1 0
1 0 1
1 1 FORBIDDEN!!!

J K Q+
0 0 Q
0 1 0
1 0 1
1 1 Q'

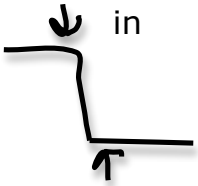


positive-edge triggered flip-flops



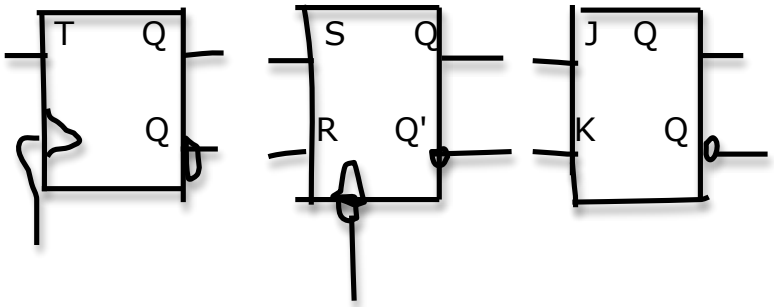
input

negative edge



in

1 1

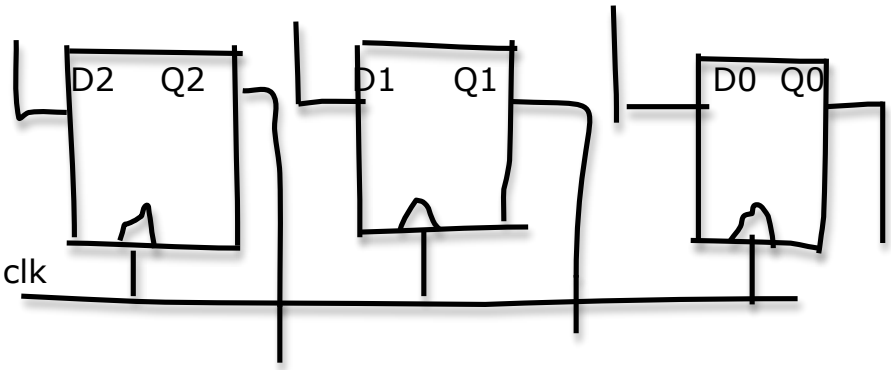


active high vs active low

0 false

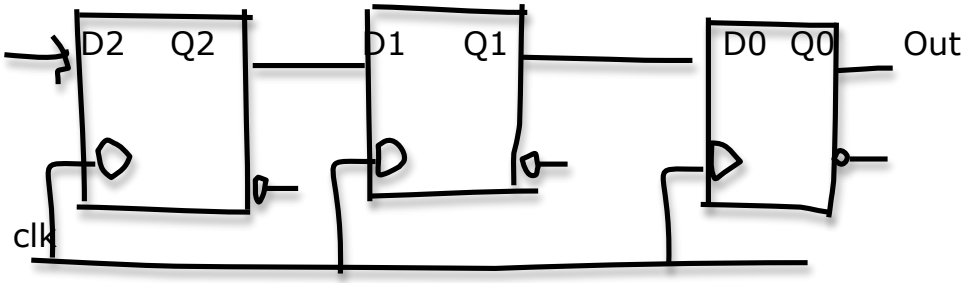
1 true

LSI

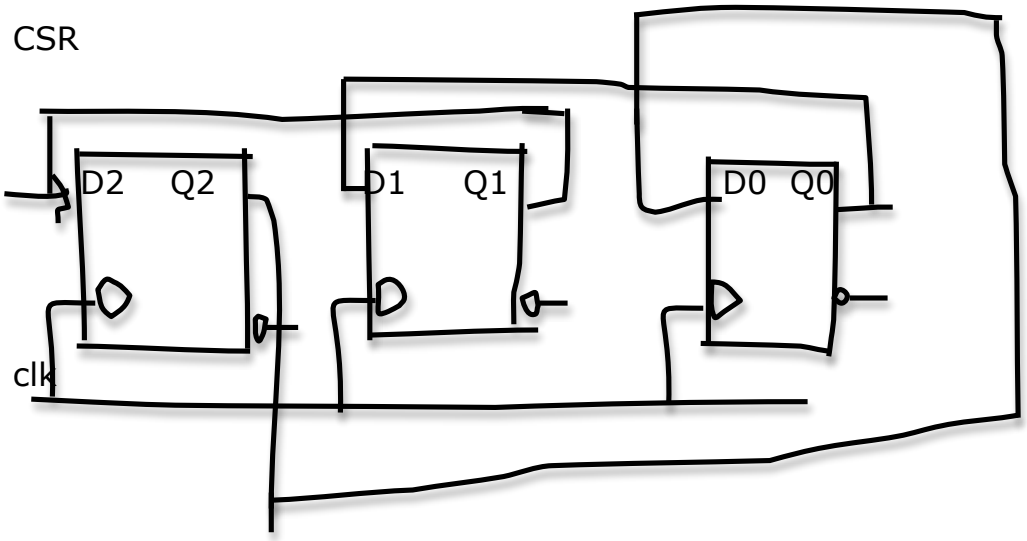


register:
shift

LSR

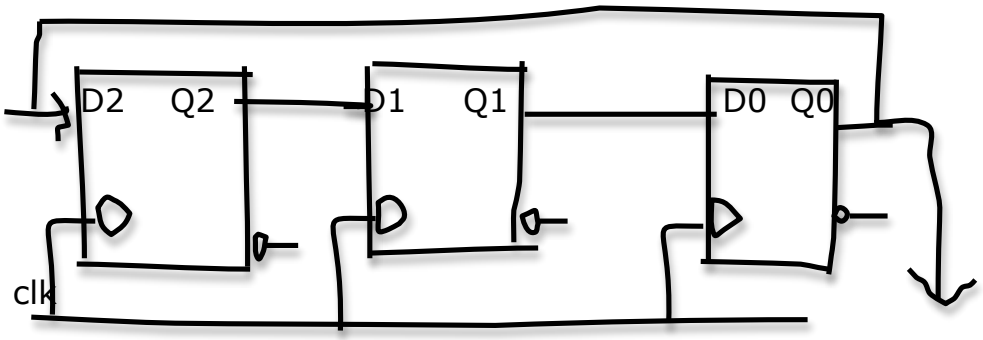


CSR



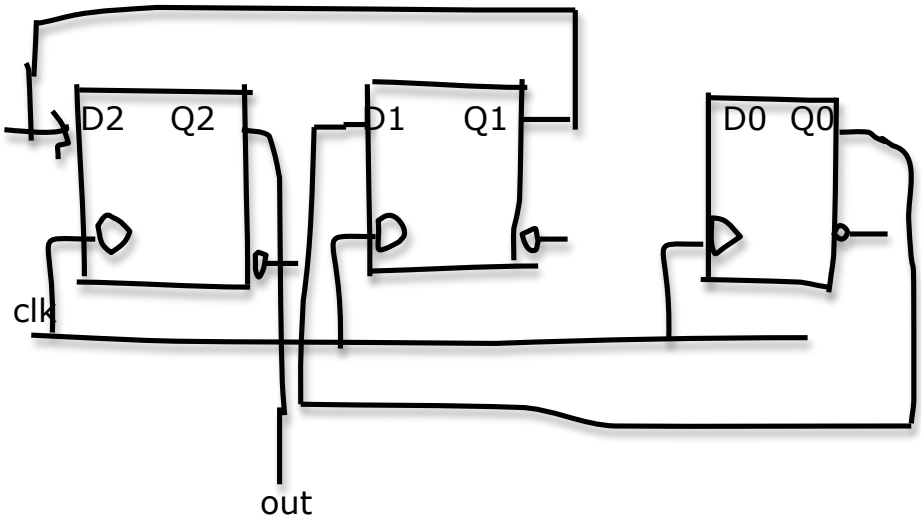
Out

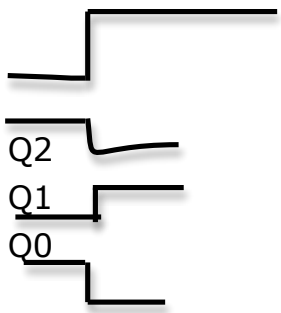
CSL



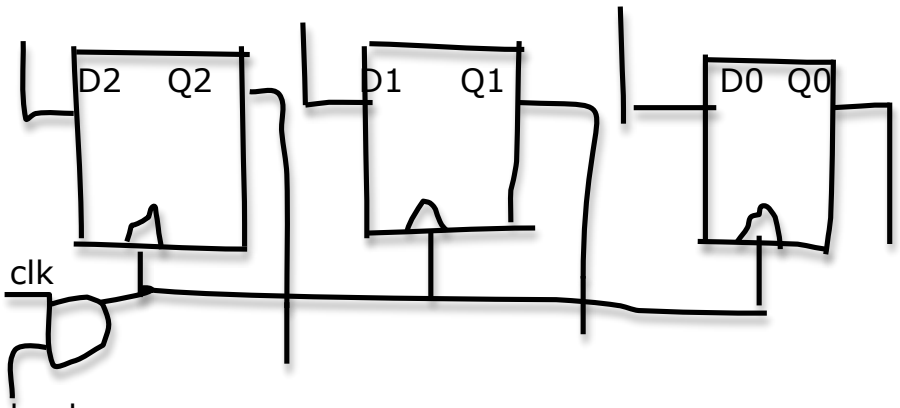
out

ASL





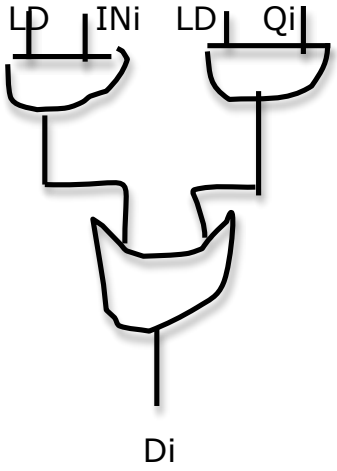
LSI

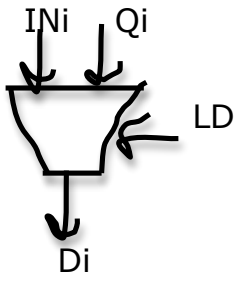


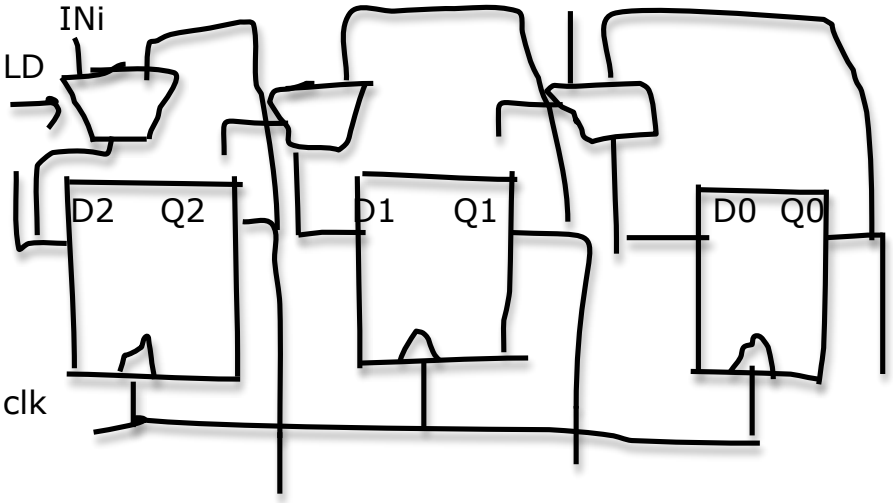
load=1: store
load=0: read

clk	Load	OP
0	0	Read
0	1	Read
1	0	Read
1	1	Write

LD*IN+LD'*Q



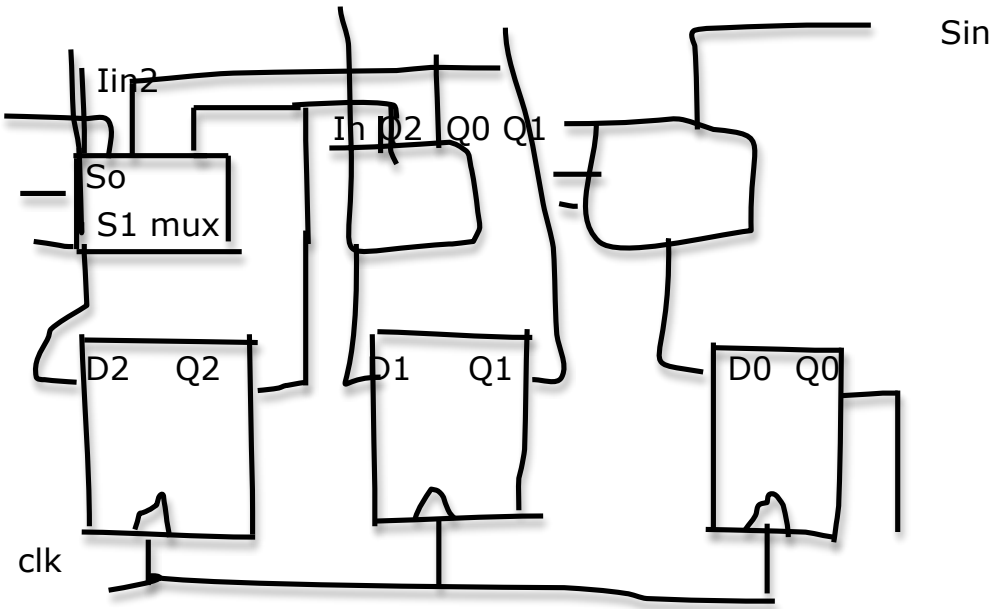


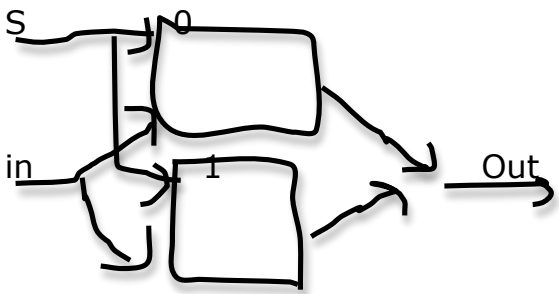


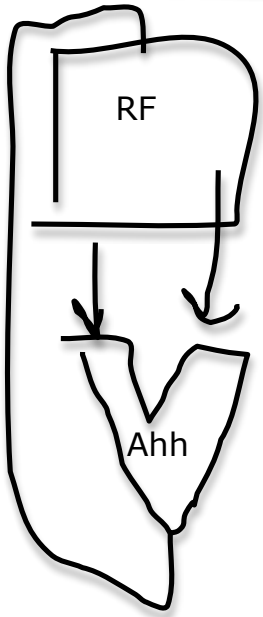
load

74194

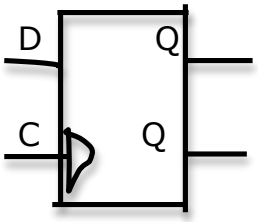
So	S1	
0	0	Hold
0	1	Shift + left
1	0	shift + right
1	1	parallel load


Sin
Iin2
In
Q2
Q0
Q1
So
S1 mux
D2
Q2
D1
Q1
D0
Q0
clk


S
0
in
1
Out

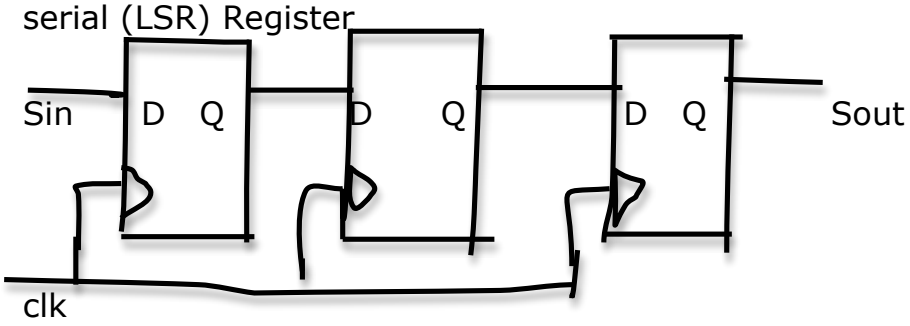

RF
Ahh

No lecture on Friday
Lab 7 is due on Fr by 10am in 4034
check grades in compass
Exam review sessions on Sunday
HW7 is due on Monday
Lecture on Monday is exam review

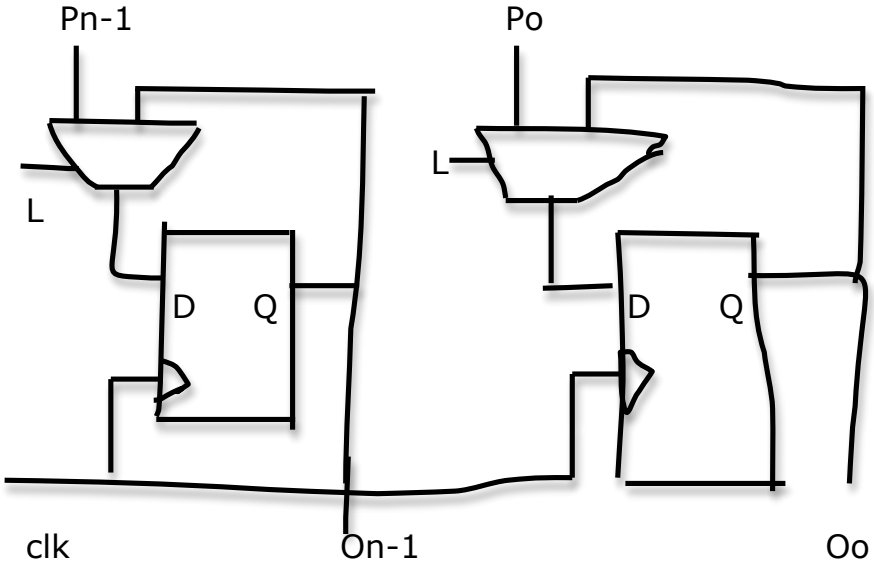


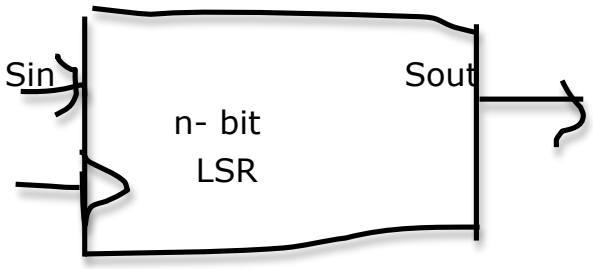
D flip flop

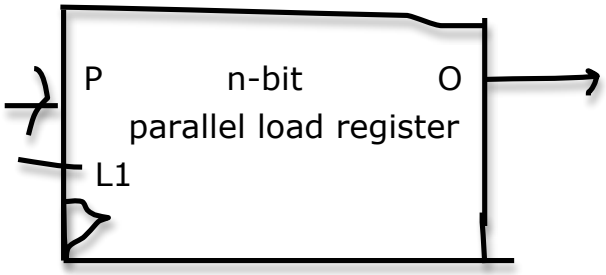
serial (LSR) Register

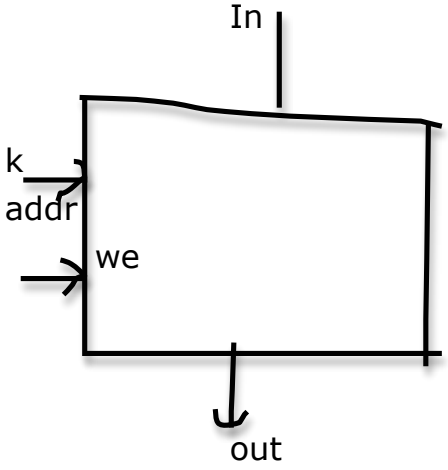


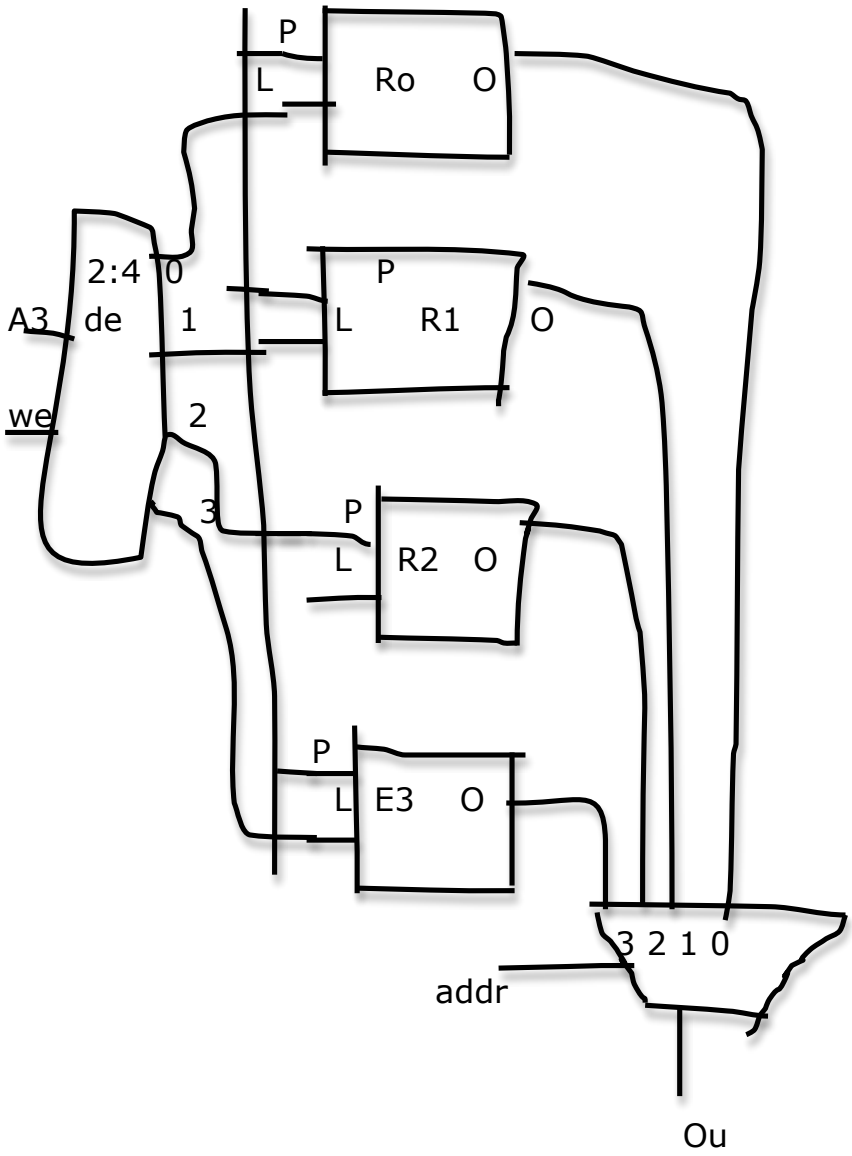
parallel load register

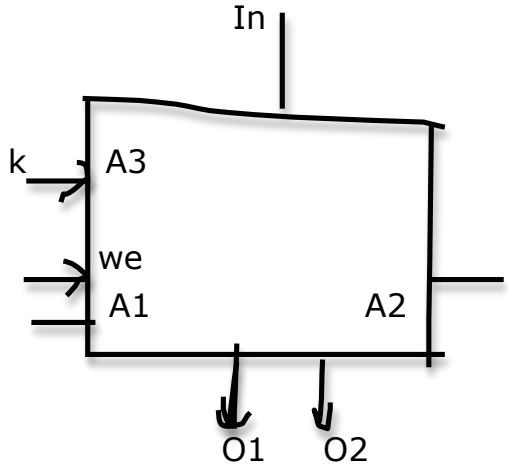


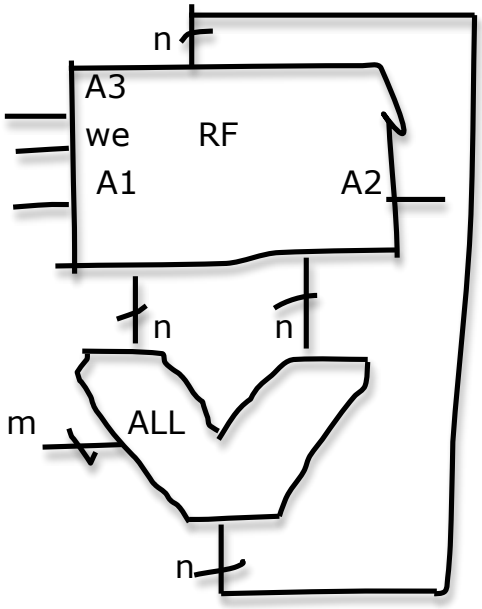

Sin
Sout
n- bit
LSR


P
n-bit
O
parallel load register
L1


In
k
addr
we
out

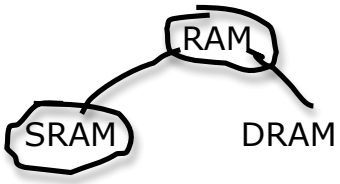

P
L
Ro
O
2:4
de
A3
we
0
1
2
3
P
L
R1
O
P
L
R2
O
P
L
E3
O
3 2 1 0
addr
Ou

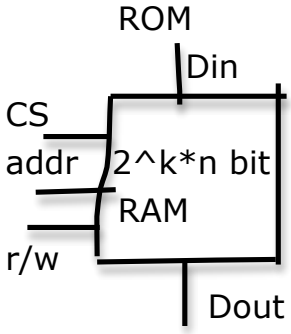

In
k
A3
we
A1
A2
O1
O2



R4<--R2+R3

ROM





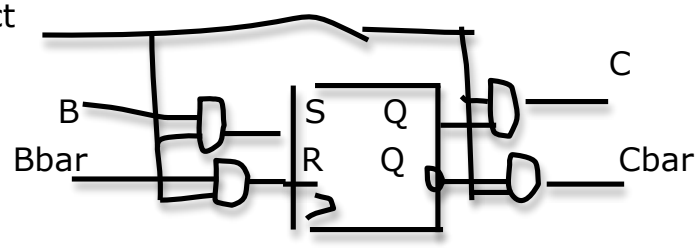
CS R/w' OP

0	x	no operation
1	0	write
1	1	read

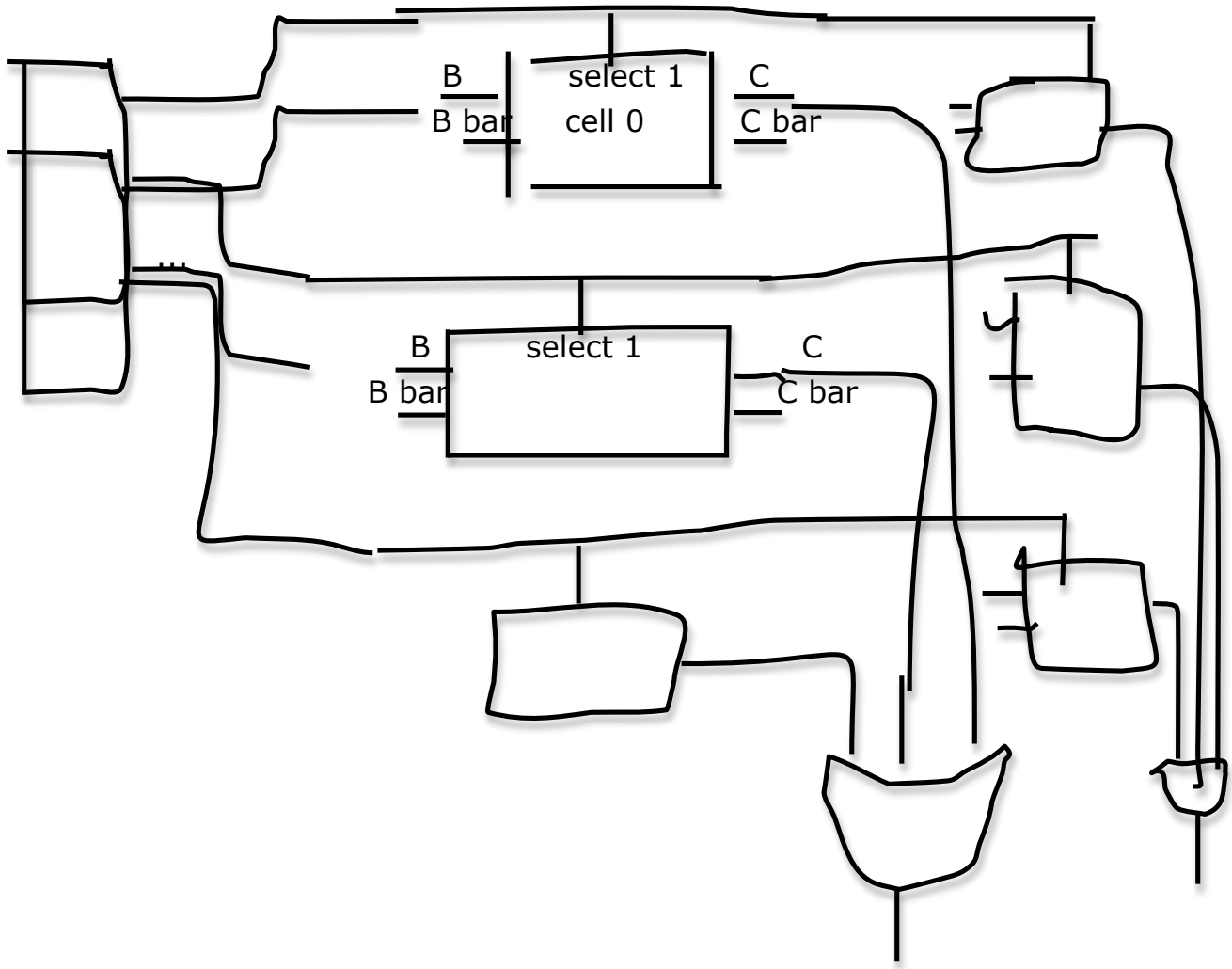
read
CS-->1
r/w'-->1
adder-->address
Dout

write
CS-->1
r/w'-->0
addr-->address
Din-->value

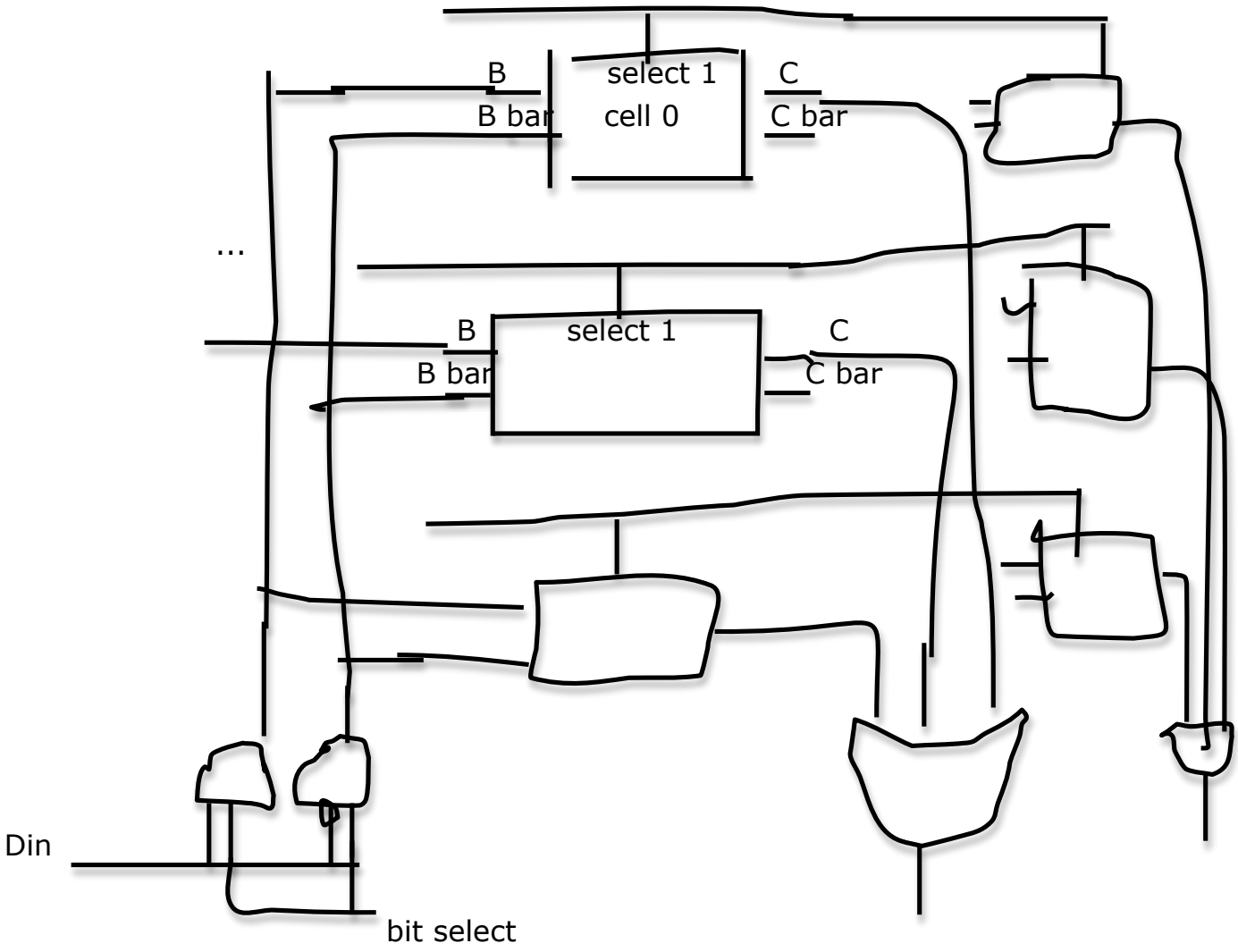
SRAM cell
select



S R Q
0 0 hold
0 1 0
1 0 1
1 1 FORBIDDEN!!!!



nx2 memory


B
B bar
select 1
cell 0
C
C bar
...
B
B bar
select 1
C
C bar
Din
bit select

Comparators

Parallel and series load registers

simplify boolean algebra

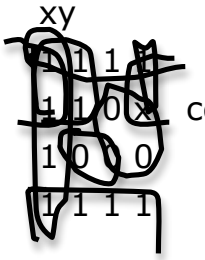
essential and prime implicants

adders

addressibilities

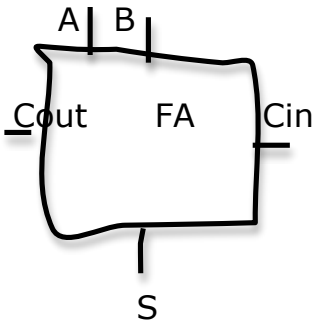
master slave design

perfect induction



EPI: a'b', d', a'c
Min SOP= a'b'+d'+a'c

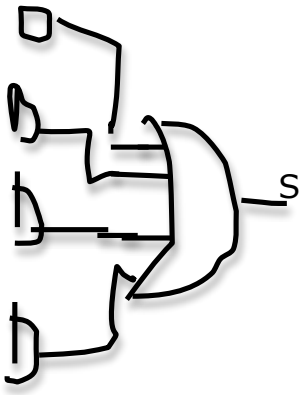
canonincal POS
=(a' + b' + c + d')(a + b' + c' + d')(a' + b' + c' + d')(a' + b + c' + d')

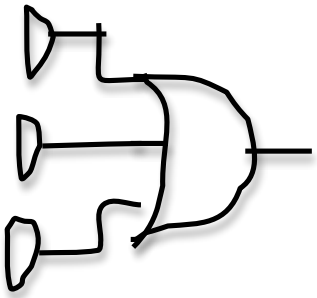


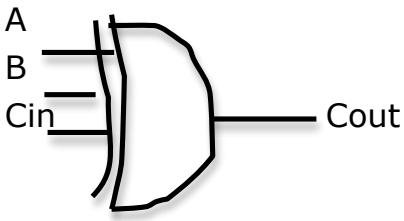
AB
00 01 11 10

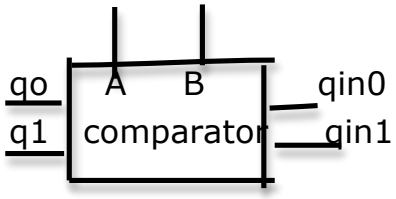
S	C	0	0	1	0	1
	1	1	0	1	0	

Cout	C	AB 00	01	11	10
	0	0	0	1	0
	1	0	1	1	1









q0q1 00 equal
01 A<B
11 don't care
10 A>B

q0

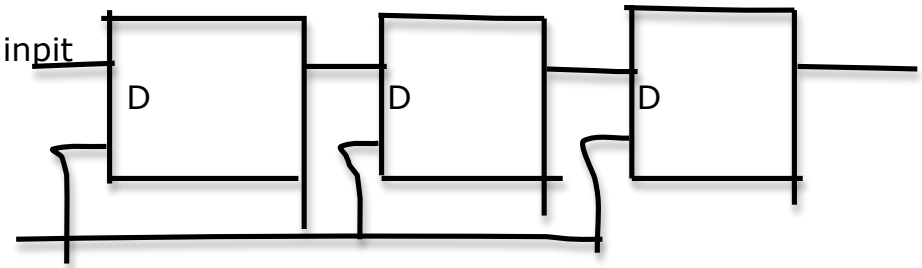
	AB			
	00	01	11	10
qin01 00	0	0	0	1
01	0	0	0	1
11	x	x	x	x
10	1	0	1	1

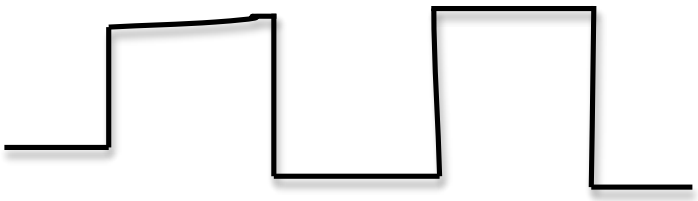
q1

	AB			
	00	01	11	10
qin01 00	0	1	0	1
01	1	1	1	0
11	x	x	x	x
10	0	1	0	0

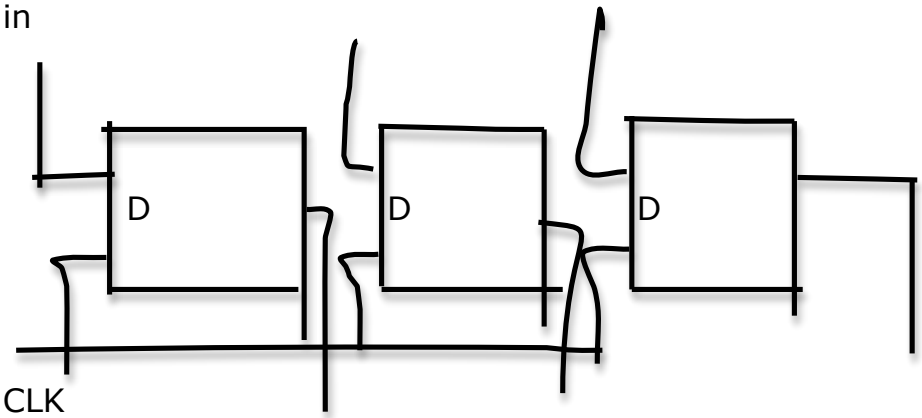
00 equ
01 a<b
11 dont care
10 a>b

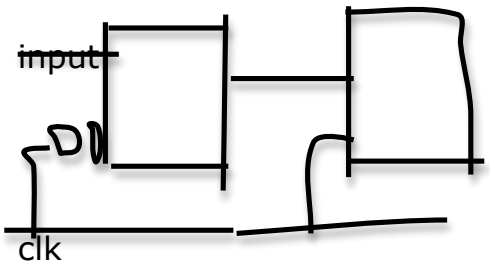
Series

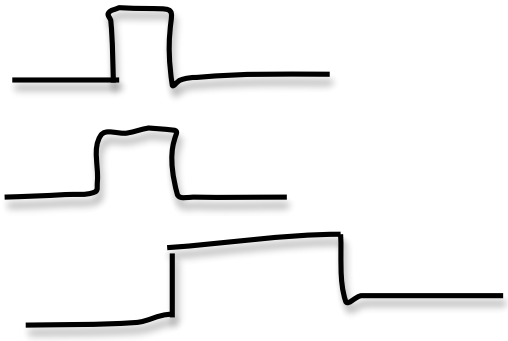


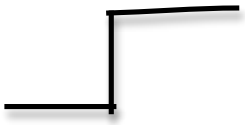


Parallel



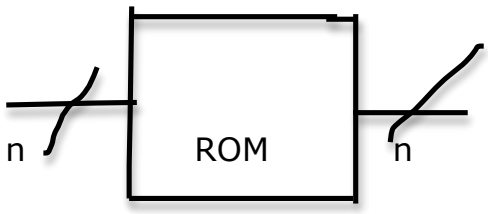






Next State for D latch

Current	CLK
Q	0
Q	1



Word input

out

word: n=adressibility
for n bit input

adress space=adressibility times number of adresses

y'(x' z + y' z)'

=(y + x' z + y' z)'
=(y + z + x' z)'
=(y+z)'
=y'z'

2012
1.
D.
(a' xor b)+c'd'+ac'+ad'+bcd

-->a'b'+ab+c'd'+ac'+ad'+bcd

=AB'+CD+