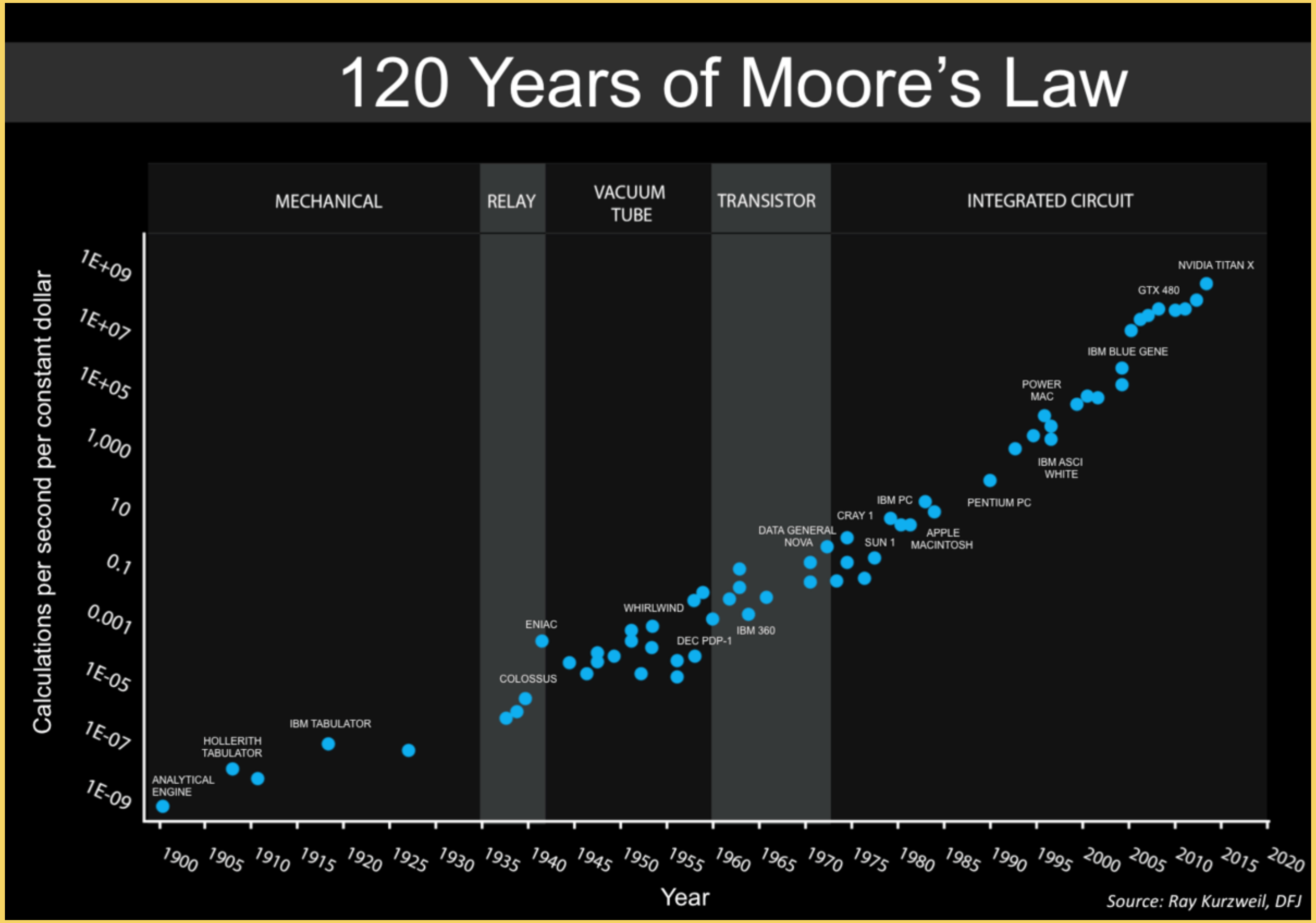
Problem Set 2 is due
This Friday, Jan 31 (10pm)



Graph by Steve Jurvetson, extending a prior graph of Ray Kurzweil.

Class 6: *Finite Computation*

University of Virginia
cs3120: DMT2
Wei-Kai Lin

Recap: Boolean logical ‘gates’

– OR(a, b): outputs 1 iff a=1 **or** b=1

– AND(a, b): outputs 1 iff a=1 **and** b=1

– NOT(b): outputs 1 iff b=0

Output 0 otherwise

Median using **And/Or/N** ot

- Still a “math”-ish def/algorithm for 3-bit MED:

```python
def MED(X[0],X[1],X[2]):
    firstpair = AND(X[0],X[1])
    secondpair = AND(X[1],X[2])
    thirdpair = AND(X[0],X[2])
    temp = OR(secondpair,thirdpair)
    return OR(firstpair,temp)
```

A formal programming language

- **AON Straightline** programs
  - Python-like language
  - Define "_functions_" that take Boolean inputs
  - Use AND/OR/NOT within
  - Assign results of AND/OR/NOT to variables
  - The result of variables can be used later as inputs
  - Return some of the obtained result(s) as output

c = AND(a, b)
c = OR(a, b)
c = NOT(a)

(PS2) More things to program

- NAND

a	b	NAND(a, b)
0	0	1
0	1	1
1	0	1
1	1	0

c = AND(a, b)

~~ret~~ NOT(c)

- XOR

truth table

a	b	XOR(a, b)
0	0	0
0	1	1
1	0	1
1	1	0

OR(AND(NOT(a), b),
AND(a, NOT(b)))

More things to program

- 1-bit addition (mod 2) XOR(a, b)

~~AOR~~
AON

- 1-bit addition with carry

AND(a, b)

a	b	carry	ADD(a, b)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

NAND Straightline Programs

- Like AON straightline programs
- Difference: we can only use NAND

~~Make AND using NAND~~

OR

NOT

$\mathrm{NAND}(a, \cancel{b}\, a) = \mathrm{NOT}(a)$

$\mathrm{AND}(a,b)$

$\mathrm{NOT}(\mathrm{NAND}(a,b))$

$\mathrm{NAND}((\quad),(\quad))$

NAND Straightline = AON Straightline

- What does it even mean? How to prove it?

Equivalence of Computing Models?

- To show model 1 is "equivalent" to model 2
AON NAND
  - Show any algorithms implemented with model 1 can be converted to an equivalent algorithm written in model 2
  - Show any algorithms implemented with model 2 can be converted to an equivalent algorithm written in model 1

NAND Straightline $\Rightarrow$ AON Straightline

Converting NAND to AON

NAND Straightline $\Leftarrow$ AON Straightline

Converting AON to NAND

NAND Straightline = AON Straightline

$f:\{\ \}^0 \to \{0,1\}^1$

AON to NAND

$f:\{\ \}^3 \to \{0,1\}^1$

x = NAND(a,b)

Becomes

temp = AND(a,b) $f(x) = 1$

x = NOT(temp)

$1(a) =$

NAND(NAND(a,a), a)

NAND to AON

x = NOT(a)

Becomes

x= NAND(a,a)

NAND(a,1)

x = AND(a,b)

Becomes

temp= NAND(a,b)

x=NAND(temp,temp)

1

~~AON, 1~~

NAND, 1

x = OR(a,b)

Becomes

t1 = NAND(a,a)

t2 = NAND(b,b)

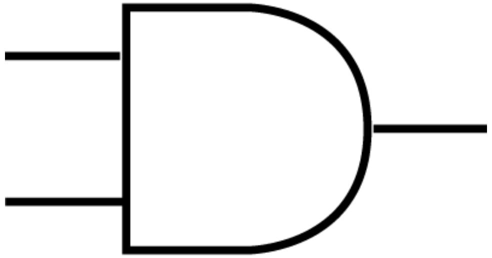
x= NAND(t1,t2)

NAND Straightline = AON Straightline

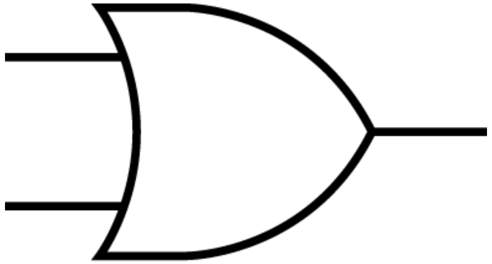
- What could be benefits of each of them?

Another approach: Boolean Circuits

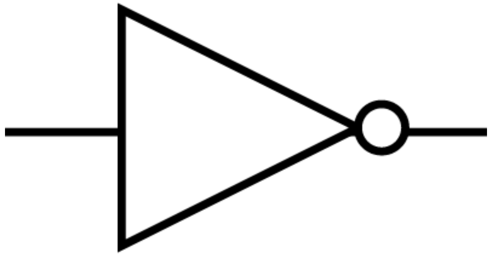
- AND



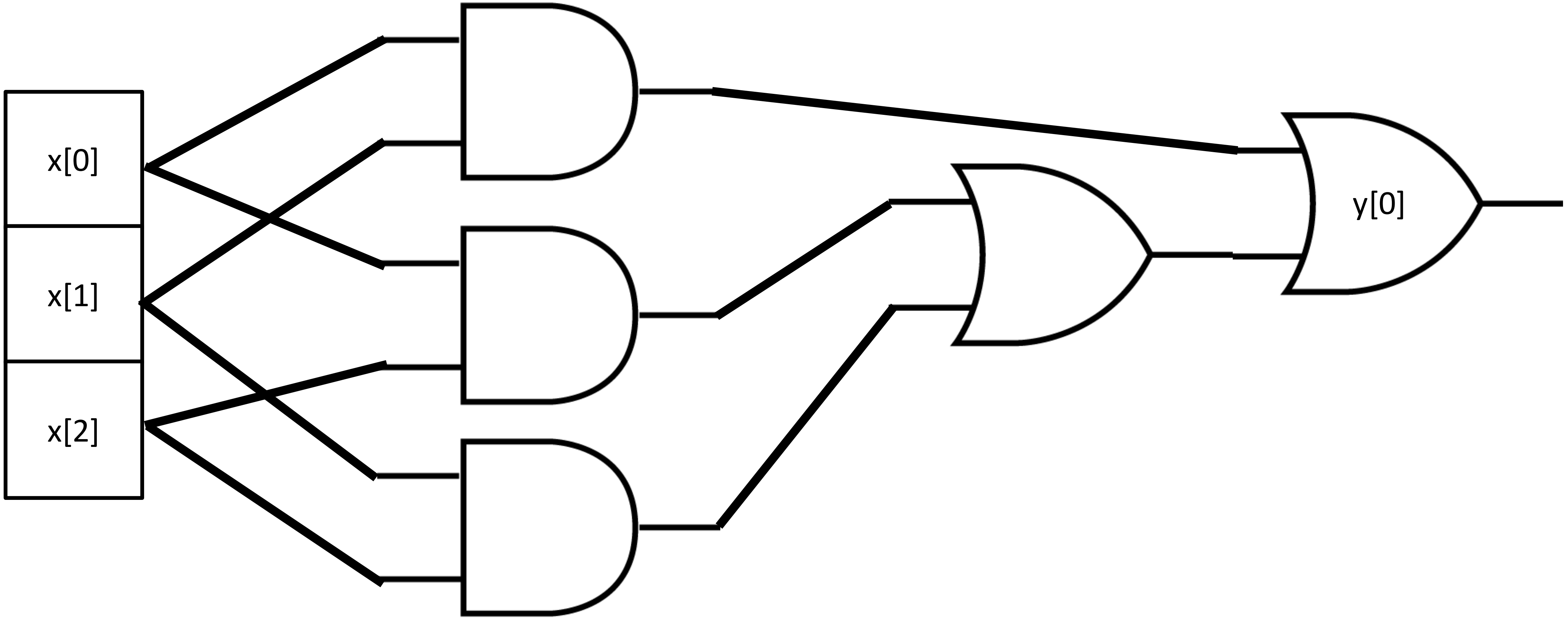
- OR



- NOT



Median with Boolean Circuits

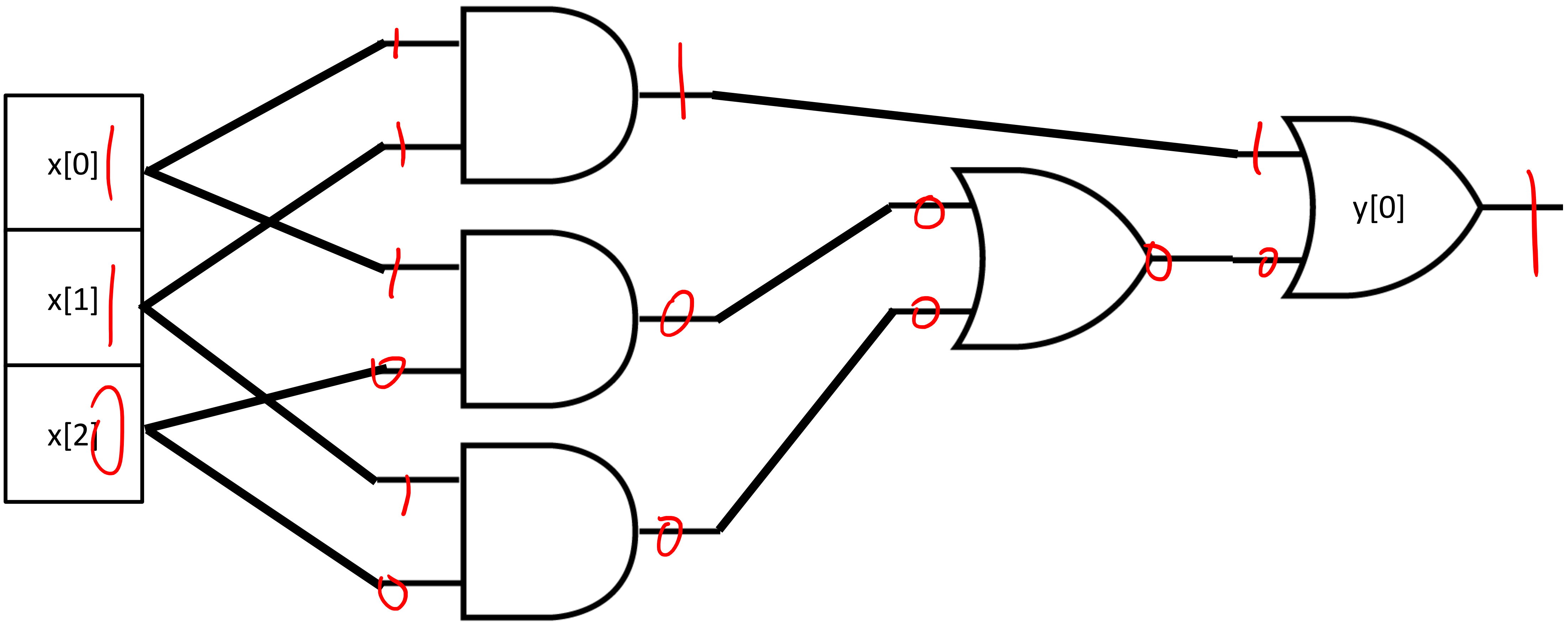


Formal Definition of Boolean Circuits

- A Boolean circuit with *n* inputs, *m* outputs, and *s* gates is a *directed acyclic graph*
- Exactly *n* nodes have no in-neighbors (these are inputs, label them *x[0], …, x[n-1]*)
- All remaining *s* nodes have a label AND, OR, NOT. AND and OR gates have two in-neighbors, NOT gates have one in-neighbor
- Exactly *m* gates are denoted as outputs (label them *y[0], …, y[m-1]*)

~~Majority~~ with Boolean Circuits

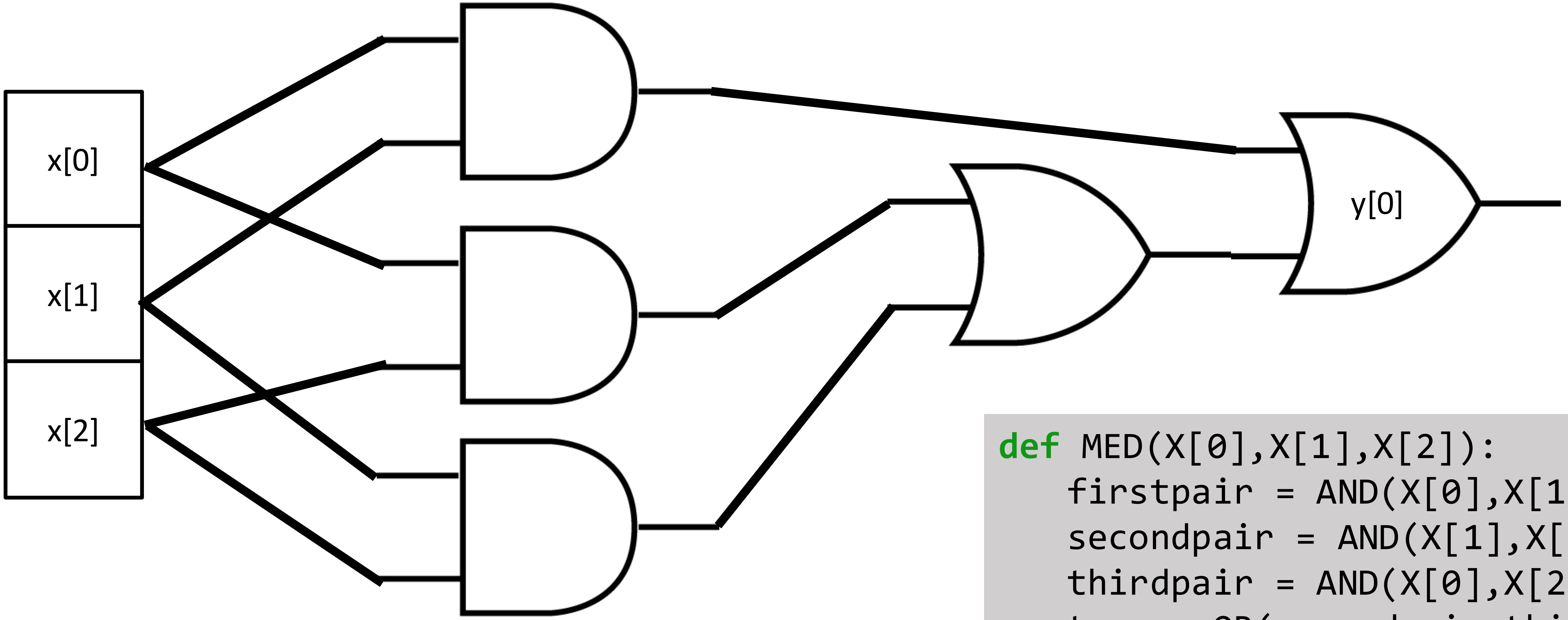
Median



Computing with a Boolean Circuit

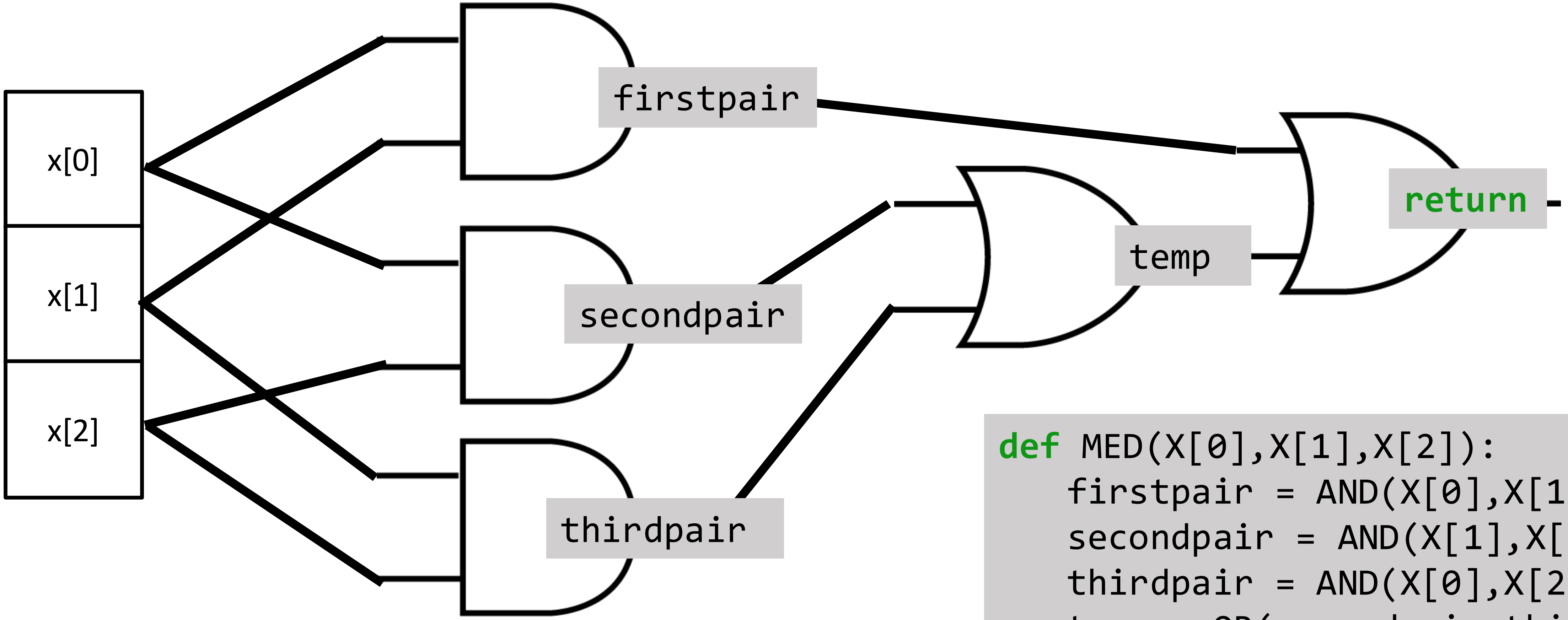
- Assign gates into *layers* such that gate *x* appears before gate *y* whenever x has an outgoing edge that's an incoming edge of *y*
- For each input node *x[i]*, assign its value to be bit *i* in the input
- For each gate (considered in order of layers), assign as its value the result of its labelled operation applied to its in-neighbor(s)
- The result is the bit-string given by the values of the output gates

Median with Boolean Circuits → AON straightline program




```
def MED(X[0],X[1],X[2]):
    firstpair = AND(X[0],X[1])
    secondpair = AND(X[1],X[2])
    thirdpair = AND(X[0],X[2])
    temp = OR(secondpair,thirdpair)
    return OR(firstpair,temp)
```

Median with Boolean Circuits → AON straightline program




```
def MED(X[0],X[1],X[2]):
    firstpair = AND(X[0],X[1])
    secondpair = AND(X[1],X[2])
    thirdpair = AND(X[0],X[2])
    temp = OR(secondpair,thirdpair)
    return OR(firstpair,temp)
```

Circuits equivalent to AON Straightline

- How do we show this?
  - Show how to convert any circuit to an AON straightline that computes the same function
  - Show how to convert any AON straightline to a circuit that computes the same function

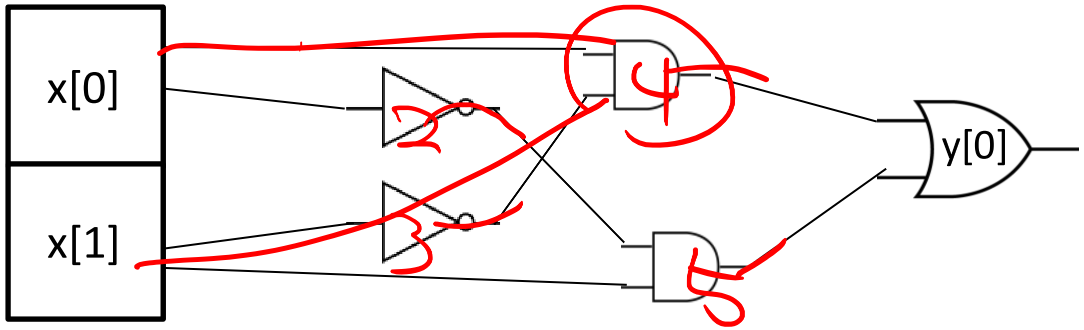
Circuit to Straightline (saw an example)

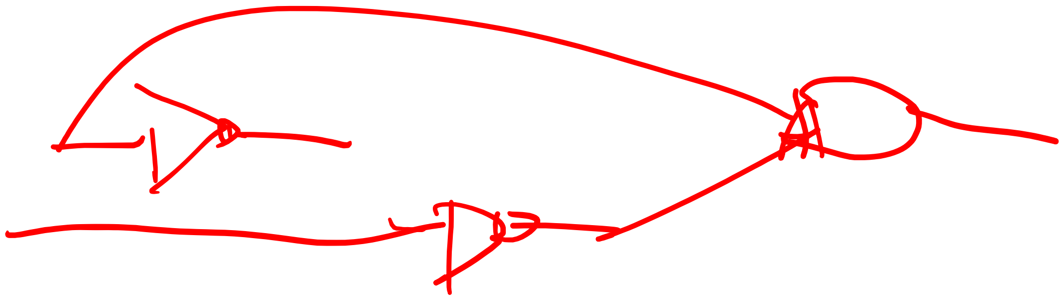
Topological Sort on gates of circuit

- Circuit inputs are straightline inputs already
- Each gate gets a variable
- Value of that variable is result of applying the operation of that gate to the variables of the in-neighbors
- Output is variable values of the output gates

Another example: XOR

Circuit





Program

v2 = NOT(x[0])

v3 = NOT(x[1])

v4 = AND(x[0], v3)

v5 = AND(x[1], v2)

ret OR(v4, v5)

AON-Straightline to Circuit

- Straightline inputs become circuit inputs
- Each variable becomes a gate
- The type of gate is given by the RHS of the assignment in the AON program
- The in-neighbors of the gate are the gates represented by the operand variables
- The output gates are the ones represented by return variables

Observations (2nd one as important)

- Everything function computable by a circuit is also computable by a straightline program (and vice-versa)
- Every function computable by a straightline program with s variables is computable by a circuit with s gates (and the converse)

Universality

What does it mean for a Gate Set to be *Universal*?

Theorem 3.12 (NAND is a universal operation)

For every Boolean circuit C of s gates, there exists a NAND circuit C' of at most $3s$ gates that computes the same function as C.

Definition 3.20 (General straight-line programs)

Let $\mathcal{F} = \{f_0, \ldots, f_{t-1}\}$ be a finite collection of Boolean functions, such that $f_i : \{0,1\}^{k_i} \to \{0,1\}$ for some $k_i \in \mathbb{N}$. An $\mathcal{F}$ *program* is a sequence of lines, each of which assigns to some variable the result of applying some $f_i \in \mathcal{F}$ to k_i other variables. As above, we use `X[` i `]` and `Y[` j `]` to denote the input and output variables.

We say that $\mathcal{F}$ is a *universal set of operations* (also known as a universal gate set) if there exists a $\mathcal{F}$ program to compute the function $NAND$.

(Informal) Definition. We say a computation model is ***universal*** if for **any** finite function $f: \{0,1\}^n \to \{0,1\}^m$, there is an "instance" of the model that computes f.

Theorem:
AON circuits is universal.

Corollary:
NAND is universal.

Goal: Compute $f: \{0,1\}^n \to \{0,1\}^m$

- Let $f_1, f_2, \ldots, f_m$ be functions such that
  - $f_i: \{0,1\}^n \to \{0,1\}$
  - $f_i(x)$ is the ith bit of $f(x)$

Goal: Compute $f: \{0,1\}^n \rightarrow \{0,1\}$

(Also called "Boolean" functions) output 1 bit.

- Represent f as a string s_f:

$$s_f = b_0 b_1 \dots b_{2^n-1}, \qquad b_i = f(i)$$

- We have $f(i) = s_f[i]$

- Can we implement "array" using AON gates?
- Want: LOOKUP(s, i) outputs s[i]

AON

Gate: IF(cond, a, b)

- Output a if cond = 1
- Output b if cond = 0

cond	a	b	IF(cond, a, b)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Array: $\text{LOOKUP}_k(s, i)$

$2^k + k$

- k: 1,2,3,…
- s: 2^k-bit string, s = $b_0 b_1 \ldots b_{2^k-1}$ $\quad 2^k$
- i: k-bit string, representing $0, 1, \ldots, 2^k - 1$ $\quad k$
- $\text{LOOKUP}_k(\text{s}, i)$ outputs s[i] = b_i

Circuit: LOOKUP_1(s, i)

k: 1,2,3,…

s = $b_0 b_1 \dots b_{2^k-1}$

i represent $0, 1, \dots, 2^k - 1$

outputs s[i] = b_i

2 1

- LOOKUP_1(s, i) = $\text{LOOKUP}_1(b_0 b_1$, i)

$\bar{i} = 0$ b_0

$\bar{i} = 1$ b_1

IF(cond = $\bar{i}$, b_1 , b_0)

Circuit: LOOKUP_2(s, i)

k: 1,2,3,…

s = $b_0 b_1 \ldots b_{2^k-1}$

i represent $0, 1, \ldots, 2^k - 1$

outputs s[i] = b_i

- LOOKUP_2(s, i) = $\text{LOOKUP}_2(b_0 b_1 b_2 b_3$, i)

Circuit: LOOKUP_k(s, i)

k: 1,2,3,…

s = $b_0 b_1 \ldots b_{2^k-1}$

i represent $0, 1, \ldots, 2^k - 1$

outputs s[i] = b_i

- LOOKUP_k(s, i) = $\text{LOOKUP}_k(b_0 b_1 \ldots b_{2^k-1}$, i)
- Recurse!

Circuit: $\text{LOOKUP}_k(s, i)$

k: 1,2,3,…

s = $b_0 b_1 \ldots b_{2^k-1}$

i represent $0, 1, \ldots, 2^k - 1$

outputs s[i] = b_i

$\text{LOOKUP}_k(s, i)$:

first_half = LOOKUP_{k-1}(s[0:2^{k-1}], i[1:k])

second_half = LOOKUP_{k-1}(s[2^{k-1}:2^k], i[1:k])

return IF(i[0], second_half, first_half)

Theorem:
AON circuit is universal.

PS2: Autograder Updated

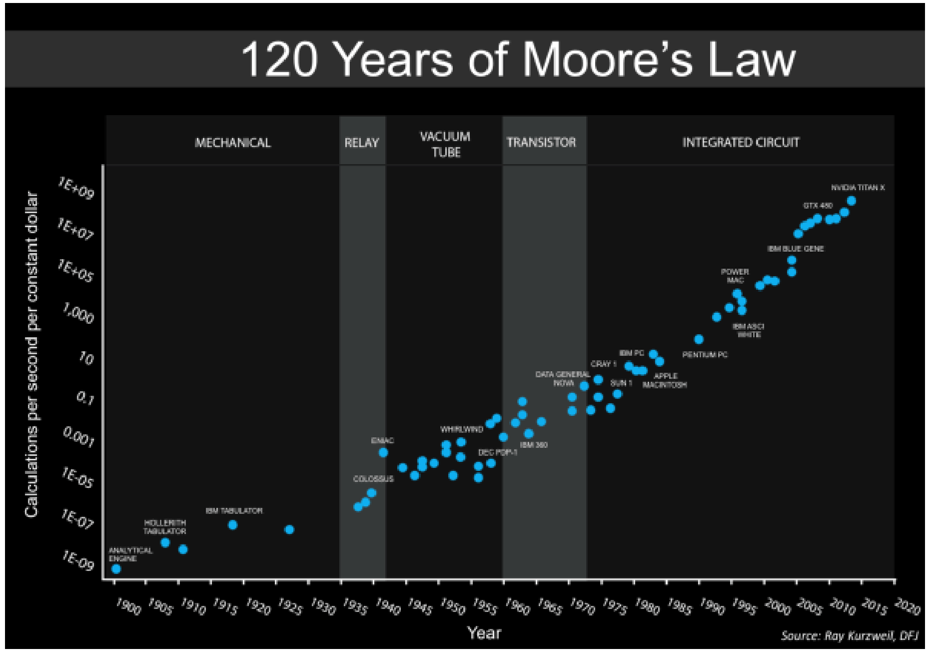
Test cases shall be public.

If you passed all tests in the notebook and on Gradescope, you are good.

Charge

Computation Model

AND, OR, NOT
Universality



PS2: due this Friday 10:00pm

PS3: to be released today, due next Friday 10:00pm
PRR3: due next Monday 10:00pm