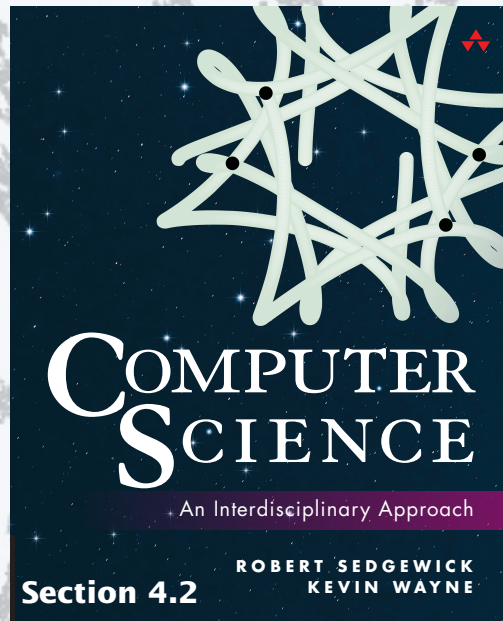


COMPUTER SCIENCE
SEDGEWICK / WAYNE

PART II: ALGORITHMS, THEORY, AND MACHINES



Questions for discussion

Part II

<http://introcs.cs.princeton.edu>



Lecture 11: Searching and Sorting

Search in an array

Suppose that an array of strings contains

a[0]	black raspberry
a[1]	chocolate
a[2]	cookie dough
a[3]	coffee
a[4]	mint
a[5]	strawberry
a[6]	vanilla

Q. How many compares for a successful search for **mint** using:

- binary search?
- sequential search?

Q. How many compares for an unsuccessful search for **pistachio** using:

- binary search?
- sequential search?

Sorting performance

Q. The following tables are performance results for algorithms that sort 10-character strings. Label each as *insertion sort* or *mergesort*.

N	T_N	$T_N/T_{N/2}$
20,000	1	—
40,000	4	—
80,000	35	
160,000	225	

N	T_N	$T_N/T_{N/2}$
1 million	1	—
2 million	2	—
4 million	5	
8 million	10	

Sorting and searching performance

Q. To the right of each option, fill in the circle corresponding to the one-word characterization that best describes the order of growth of the *worst-case* running time.

	logarithmic	linear	linearithmic	quadratic
mergesort	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
merge	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
binary search	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
insertion sort	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
sequential search	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
bubble sort	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>



Lecture 12: Stacks and Queues

Pushdown stack

Q. In the following, interpret

- a letter to mean *push*
- a minus sign to mean *pop*

1. Starting with an empty stack, give the contents of the stack after

a - b c - d e - f g h - - i

2. Suppose that the standard array representation was used for the stack.
Give the contents of the *array* after

a - b c - d e - f g h - - i

Prefix/infix

Q. Give an *infix* expression corresponding to each of the following *prefix* expressions.

3 7 + 4 6 + *

20 35 7 / 4 * +

63 3 / 17 - 0 * 2 +

1 2 4 / + 4 8 / + 6 3 / -



Lecture 13: Symbol Tables

Sorting and searching performance

Q. To the right of each option, fill in the circle corresponding to the one-word characterization that best describes the order of growth of the *worst-case* running time.

	logarithmic	linear	linearithmic	quadratic
mergesort	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
merge	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
binary search	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
BST search	<input type="radio"/> 10%	<input type="radio"/> 23%	<input type="radio"/> 22%	<input type="radio"/> 38%
insertion sort	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
sequential search	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
bubble sort	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Searching scalability

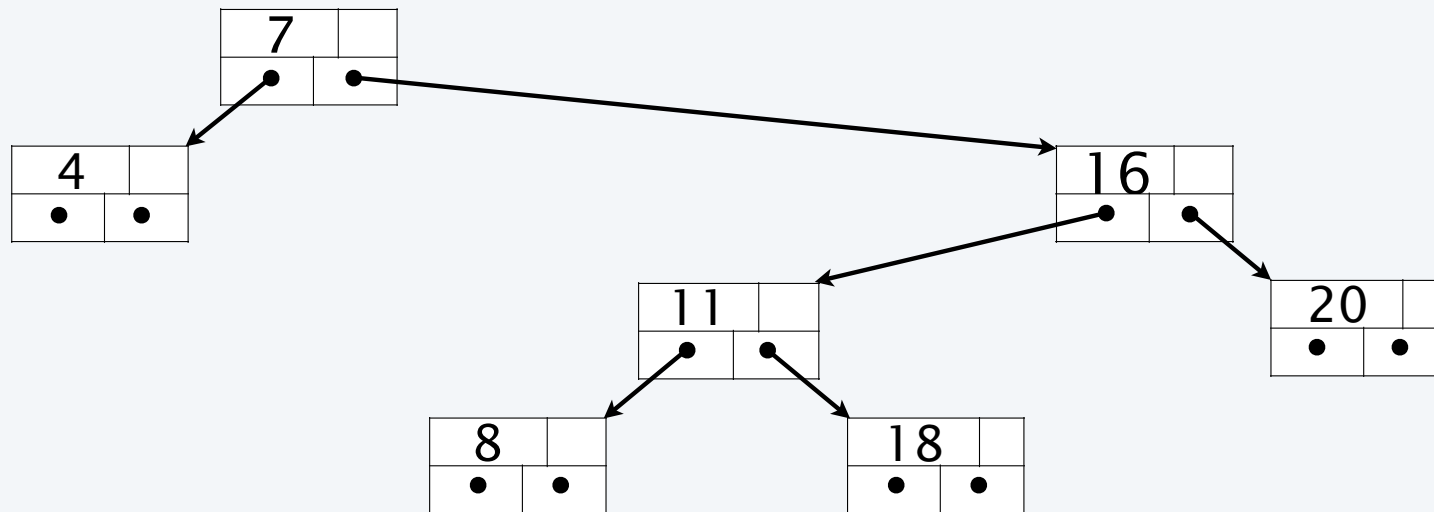
Q. An algorithm is *scalable* if it can accommodate an unpredictable mix of operations for a large amount of data, even as the amount of data grows.

Which of the following data structures admits scalable symbol-table algorithms?

	scalable?
array	<input type="radio"/>
linked list	<input type="radio"/>
BST	<input type="radio"/>

Legal BST

Q. Is this a BST?



A. No.

BST construction

Q. Match the key sequences with the BST produced by inserting them in order into an initially empty tree.

Z B P F M G H



G Z F B M P H



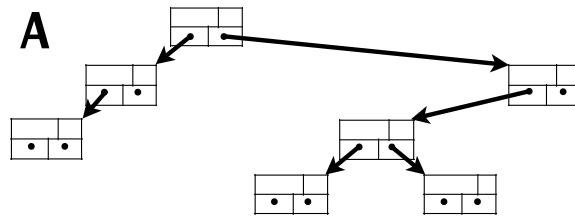
F P Z H G M B



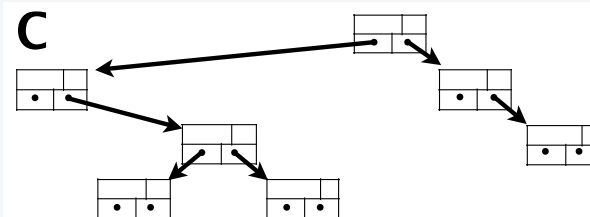
M P Z B G F H



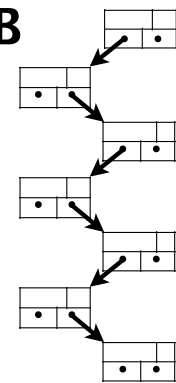
A



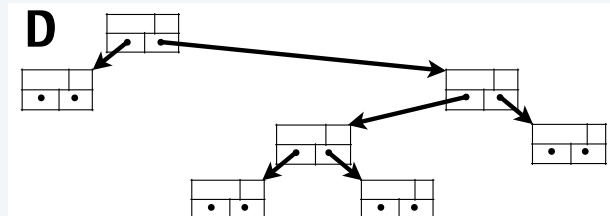
C



B



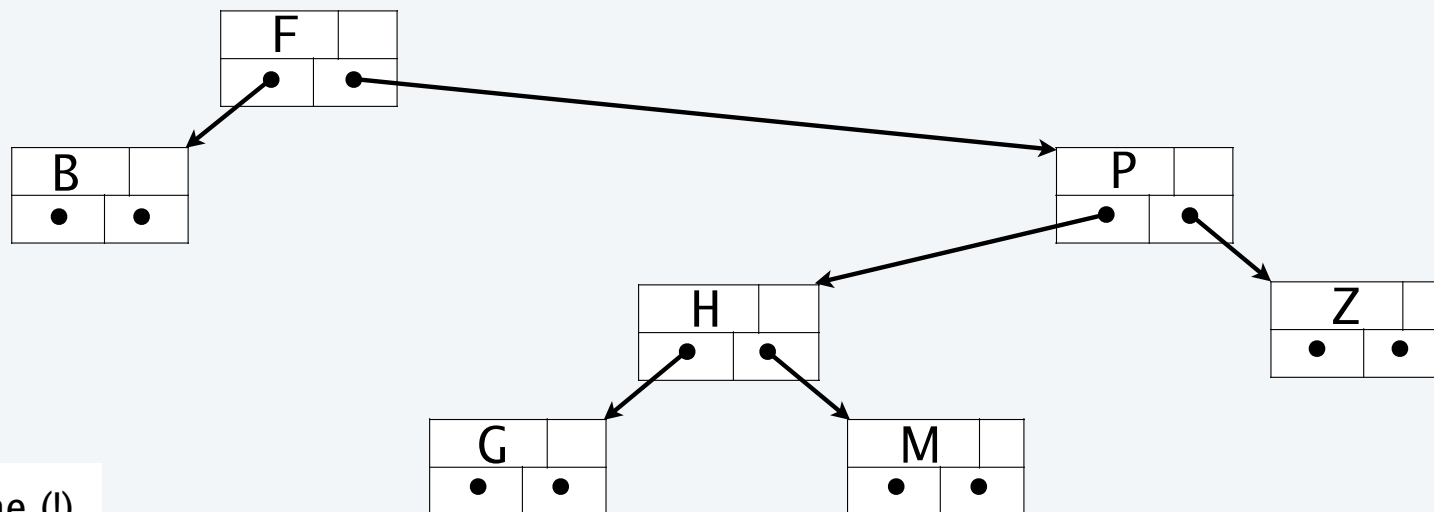
D



Tree labeling

Q. How many ways to assign N keys to a give N -node tree?

B F G H M P Z



A. Only one (!)

Tree height

Q. Give the *height* of the BST produced by inserting keys in the order given into an initially empty BST.

10, 11, 20, 30, 32, 48, 60

7

10, 20, 32, 11, 30, 48, 60

30, 10, 20, 11, 60, 48, 32

20, 11, 10, 30, 32, 60, 48

Tree sequences

Q. Which of the following could *not* have been the sequence of keys examined to search for 70 in a BST?

77, 41, 99, 20, 85, 70

☐

99, 10, 80, 20, 60, 70

☐

5, 10, 80, 40, 32, 50, 70

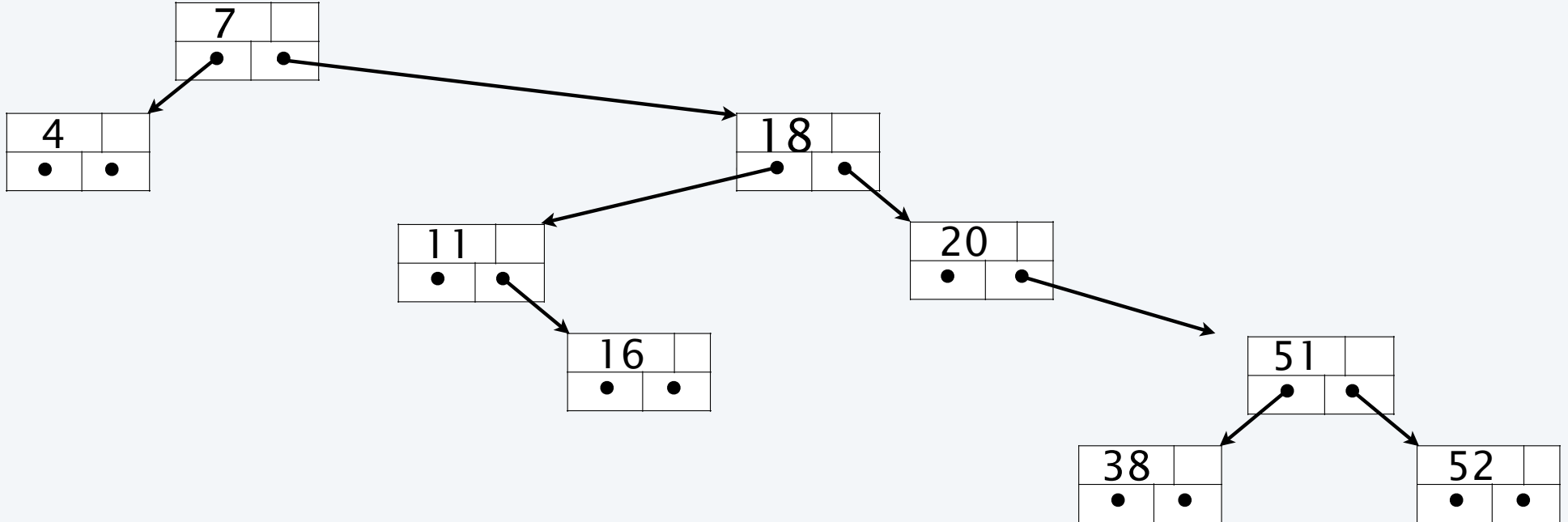
☐

22, 58, 81, 70

☐

Order statistics

Q. Identify the smallest, largest, and *median*.



Tree enumeration

Q. How many different trees of 3 nodes? 4?





Lecture 14: Intro to Theory of Computing

REs

Q. How do know what language is described by an RE?

Ex. Fall 2014 Question 5.

Let $L = \{ \mathbf{ab}, \mathbf{aaab}, \mathbf{aaaab}, \mathbf{aabaab}, \mathbf{aabaaab} \}$.

Write 1, 2, 3, or 4 to indicate whether the RE

1. Matches *no* strings in L .
2. Matches *only some strings in L* and some other strings.
3. Matches all strings in L and some other strings.
4. Matches *all strings in L and no other strings*.

$(aa^*b)^*$ 3

a^*b^* 2

$(a|b)^*ab$ 3

$a^*baba^*b^*$ 1

$(ab) | (a(a|aba)(a|aa)b)$ 4

$a^*baaa^*b^*$ 2

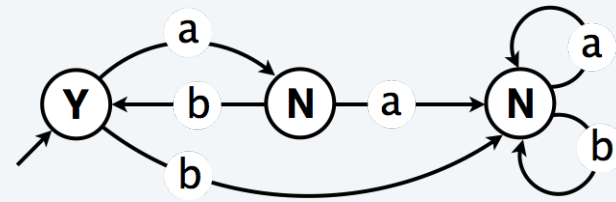
Q. Match the REs with the DFAs.

A $a^* | (a^*ba^*ba^*ba^*)^*$

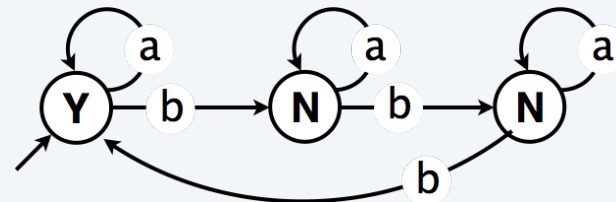
B $(ab)^*$

C $(a^*b)^+$

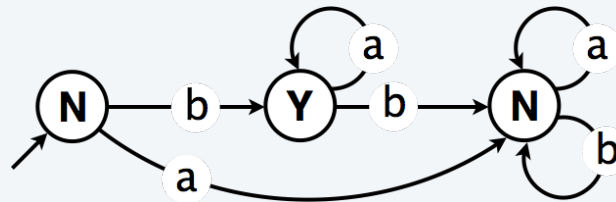
D ba^*



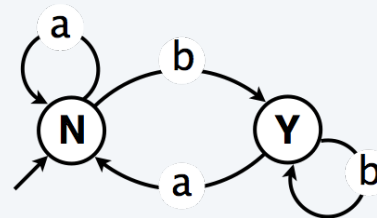
B



A



D



C

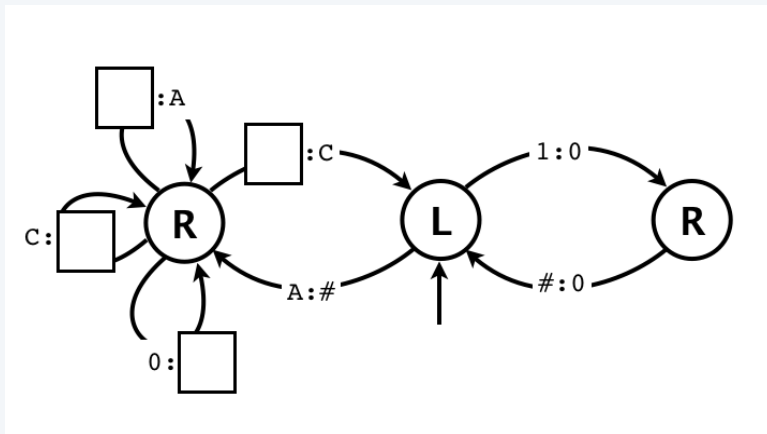


Lecture 15: Turing Machines

Tracing TMs

Q. How do we trace a Turing Machine?

Ex. Fall 2014 Question 5.



The incomplete TM at left is supposed to write onto its tape the Fibonacci sequence 1, 1, 2, 3, 5, 8, 13, ..., but in unary notation: 1, 1, 11, 111, 11111, 11111111, 11111111111111.

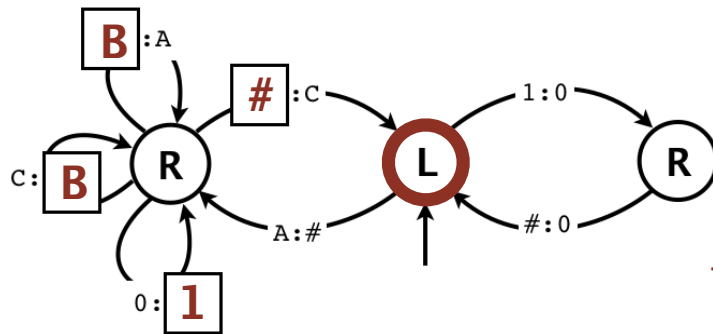
This TM uses the symbols A, B, and C to delineate the two most recently computed Fibonacci numbers. Here is the tape before computing Fibonacci number 5, with the tape head positioned at the C:

1 # 1 A 1 1 B 1 1 1 **C** # # # # #

Fill in exactly one symbol in each of the 4 empty boxes to complete the design of this TM. You may assume that the initial contents of the tape and the position of the tape head are as given above and that the machine starts in the middle state.

Q. How do we trace a TM?

Ex. Fall 2014 Question 5.



*scan left
for 1 or A*

1 # 1 A 1 1 B 1 1 1 **C** # # # #

*scan right
for #*

1 # 1 A 1 1 B 1 1 **0** C # # # #

*scan left
for 1 or A*

1 # 1 A 1 1 B 1 1 0 C **0** # # # #

. . .

*scan left
for 1 or A*

1 # 1 A 0 0 B 0 0 0 C 0 0 0 0 **0**

*scan right for
0, B, or C
change 0 to 1
change B to A
change C to B*

1 # 1 **#** 0 0 B 0 0 0 C 0 0 0 0 0

. . .

*scan left
for 1 or A*

1 # 1 # 1 1 A 1 1 1 B 1 1 1 1 1 **C**

Computability

Spring 2015 Q5

	Turing Machine	TOY with sufficient memory	DFA	None of these
Can perform any possible computation (if the Church-Turing thesis holds).	✓	✓	✗	✗
Cannot express some Java programs	✗	✗	✓	✗
Cannot be simulated in Java	✗	✗	✗	✓
Always halts on all finite inputs	✗	✗	✓	✗
Can always correctly check whether an arbitrary Java program goes into a loop.	✗	✗	✗	✓

Computability

Spring 2011 Q8

A known to be true

There exists a mathematical function that can be computed in Java, but *cannot* be computed on a Turing machine.

B

B known to be false

There exists a mathematical function that can be computed in polynomial time on a quantum computer, but cannot be computed in polynomial time on a Turing machine. *Assume that quantum computers can be built.*

D

C if true would falsify the Church-Turing thesis

D if true would falsify the *extended* Church-Turing thesis

There exists a mathematical function that can be computed in polynomial time in Java, but *cannot* be computed in polynomial time on a Turing machine.

B

E if true would prove the Church-Turing thesis

There exists a Turing machine that can simulate the behavior of any other Turing machine.

A



Lecture 16: Intractability

Polynomial vs. Exponential

Circle the largest value, for $N = 10$

$$N^{100}$$

$$1000N^3$$

$$2^N$$

Circle the largest value, for $N = 100$

$$N^{100}$$

$$1000N^3$$

$$2^N$$

Circle the largest value, for $N = 1000$

$$N^{100}$$

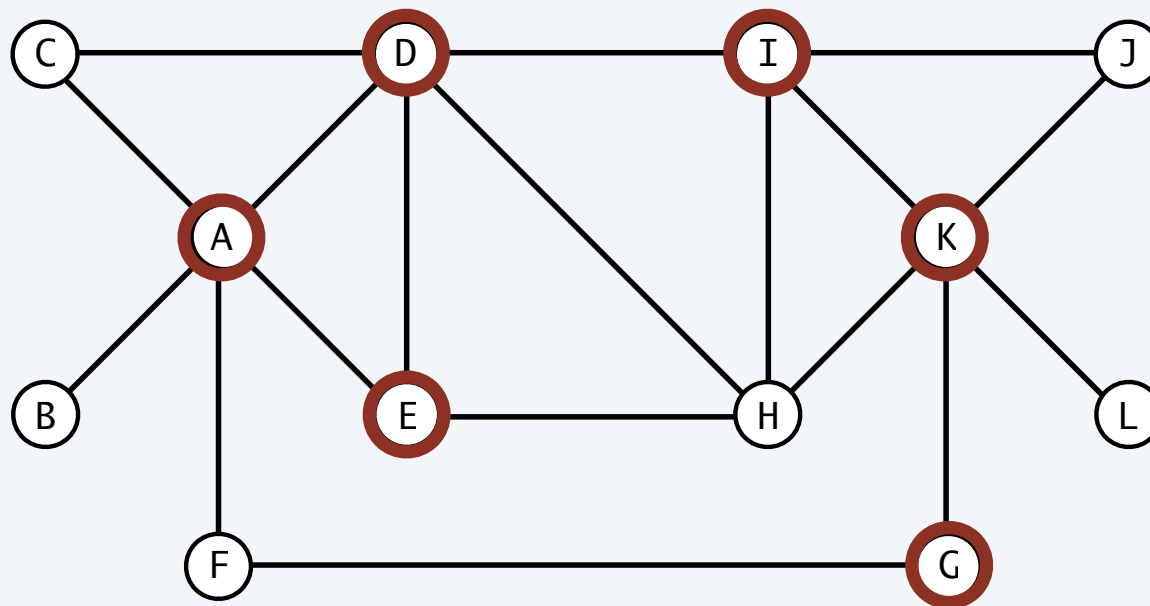
$$1000N^3$$

$$2^N$$

Vertex cover

Exercise 5.5.6

Find a minimum-cardinality vertex cover for this graph.



NP-hard vs. NP-complete

Q. If someone finds a polynomial-time algorithm for FACTOR, would that prove that $P = NP$?

A. No. (No reduction from an-NP-complete problem is known.)

Q. If someone finds a polynomial-time algorithm for MIN VERTEX COVER, would that prove that $P = NP$?

A. Yes. Such a solution would give a solution to VERTEX COVER, which is NP-complete.

Q. If someone proves that $P=NP$, would that give a polynomial-time algorithm for FACTOR?

A. No. It would prove that one exists, not necessarily exhibit one.

Q. If someone proves that $P=NP$, would that give a polynomial-time algorithm for MIN VERTEX COVER?







A. No. (It is “NP-hard” but not known to be in NP.)

NP-completeness

Exercise 5.5.25

Which of the following can we infer from the fact that TSP is NP-complete, if we assume that $P \neq NP$?








- a. No algorithm exists that solves arbitrary instances of TSP.
- b. No algorithm exists that *efficiently* solves arbitrary instances of TSP.
- c. There exists an algorithm that efficiently solves arbitrary instances of TSP, but no one has been able to find it.
- d. TSP is not in P.
- e. All algorithms that are guaranteed to solve TSP run in polynomial time for some family of inputs.
- f. All algorithms that are guaranteed to solve TSP run in exponential time for all families of inputs.

-  *exponential algorithm would do*
-  *that's the point*
-  *it would prove $P = NP$*
-  *same as b.*
-  *nonsense distractor*
-  *could be between poly and exp*

NP-completeness

Exercise 5.5.29

Let A and B be two *decision* problems. Suppose we know that A polynomial-time reduces to B. Which of the following can we infer?

- | | | |
|--|---|------------------------------------|
| a. If B is NP-complete then so is A. |  | <i>A might not be in NP</i> |
| b. If A is NP-complete then so is B. |  | <i>B might not be in NP</i> |
| c. If B is NP-complete and A is in NP then A is NP-complete. |  | <i>wrong way</i> |
| d. If A is NP-complete and B is in NP then B is NP-complete. |  | <i>an implication that matters</i> |
| e. A and B cannot both be NP-complete. |  | <i>why not?</i> |
| f. If A is in P, then B is in P. |  | <i>wrong way</i> |
| g. If B is in P, then A is in P. |  | <i>an implication that matters</i> |

P vs. NP

Q. Assume that $P = NP$. Check all correct options.

	halting problem	TSP	SAT	FACTOR	ILP
is computable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
is intractable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
is in P	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
is in NP	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
is in NPC	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

P vs. NP

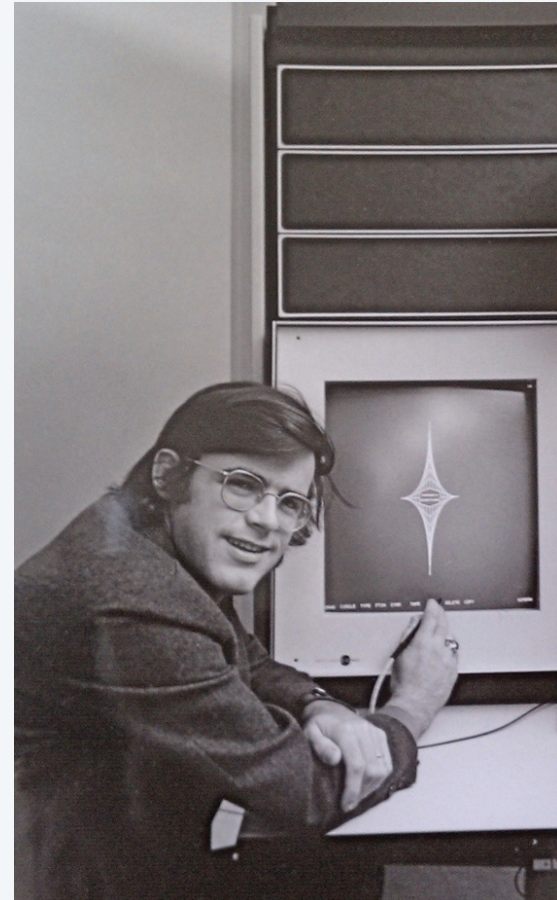
Q. Assume that $P \neq NP$. Check all correct options.

	halting problem	TSP	SAT	FACTOR	ILP
is computable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
is intractable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
known to be in P	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
known to be in NP	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
known to be NPC	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>



Lecture 17: A Computing Machine

1970



Binary operations

Q. Why is ~ 0 equal to -1 and not 1 ? (Fall 2014 Q1B)

A (wrong).

\sim is "not"

0 is "false"

"not false" is "true"

"true" is 1

A (correct).

\sim is **BITWISE** "not"

0 is 00000000000000000000000000000000

~ 0 is 11111111111111111111111111111111

11111111111111111111111111111111 is -1 (2s complement)

Representing numbers

Q. Fill in the blanks in this table.

unsigned base 10	16-bit binary	4-digit hex
100	000000001100100	0064
$4096+256+16+1 = 4369$	0001000100010001	1111
12	0000000000001100	000C

Representing information

Q. Fill in the blanks in this table.

hex	TOY instruction	TOY integer
0064	halt	100
1A00	$R[A] = R[0] + R[0]$	6,656
FFF0	$R[F] = PC + 1; PC = F0$	-16
2111	$R[1] = R[1] - R[1]$	8,465
DEEF	if ($R[E] > 0$) $PC = EF$	-8,465

opcode	operation	format	pseudo-code
0	halt	—	halt
1	add	RR	$R[d] = R[s] + R[t]$
2	subtract	RR	$R[d] = R[s] - R[t]$
3	bitwise and	RR	$R[d] = R[s] \& R[t]$
4	bitwise xor	RR	$R[d] = R[s] \wedge R[t]$
5	shift left	RR	$R[d] = R[s] \ll R[t]$
6	shift right	RR	$R[d] = R[s] \gg R[t]$
7	load addr	A	$R[d] = \text{addr}$
8	load	A	$R[d] = M[\text{addr}]$
9	store	A	$M[\text{addr}] = R[d]$
A	load indirect	RR	$R[d] = M[R[t]]$
B	store indirect	RR	$M[R[t]] = R[d]$
C	branch zero	A	if ($R[d] == 0$) $PC = \text{addr}$
D	branch positive	A	if ($R[d] > 0$) $PC = \text{addr}$
E	jump register	RR	$PC = R[d]$
F	jump and link	A	$R[d] = PC + 1; PC = \text{addr}$

TOY blocking and tackling I

Q. Give seven instructions (all having different opcodes) to put 0000 in R[A].

1A00 $R[A] = R[0] + R[0]$

2Axx $R[A] = R[x] - R[x]$

3A0x $R[A] = R[0] \& R[x]$

4Axx $R[A] = R[x] \wedge R[x]$

5A0x $R[A] = R[0] \ll R[0]$

6A0x $R[A] = R[0] \gg R[x]$

7A00 $R[A] = 0000$

<i>opcode</i>	<i>operation</i>	<i>format</i>	<i>pseudo-code</i>
0	halt	—	halt
1	add	RR	$R[d] = R[s] + R[t]$
2	subtract	RR	$R[d] = R[s] - R[t]$
3	bitwise and	RR	$R[d] = R[s] \& R[t]$
4	bitwise xor	RR	$R[d] = R[s] \wedge R[t]$
5	shift left	RR	$R[d] = R[s] \ll R[t]$
6	shift right	RR	$R[d] = R[s] \gg R[t]$
7	load addr	A	$R[d] = \text{addr}$
8	load	A	$R[d] = M[\text{addr}]$
9	store	A	$M[\text{addr}] = R[d]$
A	load indirect	RR	$R[d] = M[R[t]]$
B	store indirect	RR	$M[R[t]] = R[d]$
C	branch zero	A	if $(R[d] == 0)$ PC = addr
D	branch positive	A	if $(R[d] > 0)$ PC = addr
E	jump register	RR	PC = R[d]
F	jump and link	A	$R[d] = \text{PC} + 1$; PC = addr

TOY blocking and tackling II

Q. Which of the following put FFFF in R[A]?

7AFF	✗	$R[A] = 00FF$
7B01	✓	$R[B] = 0001$
2A0B		$R[A] = R[0] - R[B]$
7A01	✓	$R[A] = 0001$
2A0A		$R[A] = R[0] - R[A]$
7AFF		$R[A] = 00FF$
7B08	✓	$R[B] = 0008$
5AA8		$R[A] = R[A] \ll R[B]$
6AA8		$R[A] = R[A] \gg R[B]$

opcode	operation	format	pseudo-code
0	halt	—	halt
1	add	RR	$R[d] = R[s] + R[t]$
2	subtract	RR	$R[d] = R[s] - R[t]$
3	bitwise and	RR	$R[d] = R[s] \& R[t]$
4	bitwise xor	RR	$R[d] = R[s] \wedge R[t]$
5	shift left	RR	$R[d] = R[s] \ll R[t]$
6	shift right	RR	$R[d] = R[s] \gg R[t]$
7	load addr	A	$R[d] = \text{addr}$
8	load	A	$R[d] = M[\text{addr}]$
9	store	A	$M[\text{addr}] = R[d]$
A	load indirect	RR	$R[d] = M[R[t]]$
B	store indirect	RR	$M[R[t]] = R[d]$
C	branch zero	A	if $(R[d] == 0)$ PC = addr
D	branch positive	A	if $(R[d] > 0)$ PC = addr
E	jump register	RR	PC = $R[d]$
F	jump and link	A	$R[d] = PC + 1$; PC = addr

TOY code I

Q. What does this TOY program do?

```
10: 82FF  R[2] = stdin
11: 7B01  R[B] = 0001
12: 2A0B  R[A] = FFFF
13: 432A  R[3] = R[2] ^ R[A]
14: 133B  R[3] = R[3] + 1
15: 93FF  stdout = R[3]
16: 0000  halt
```

<i>opcode</i>	<i>operation</i>	<i>format</i>	<i>pseudo-code</i>
0	halt	—	halt
1	add	RR	$R[d] = R[s] + R[t]$
2	subtract	RR	$R[d] = R[s] - R[t]$
3	bitwise and	RR	$R[d] = R[s] \& R[t]$
4	bitwise xor	RR	$R[d] = R[s] \wedge R[t]$
5	shift left	RR	$R[d] = R[s] \ll R[t]$
6	shift right	RR	$R[d] = R[s] \gg R[t]$
7	load addr	A	$R[d] = \text{addr}$
8	load	A	$R[d] = M[\text{addr}]$
9	store	A	$M[\text{addr}] = R[d]$
A	load indirect	RR	$R[d] = M[R[t]]$
B	store indirect	RR	$M[R[t]] = R[d]$
C	branch zero	A	if $(R[d] == 0)$ PC = addr
D	branch positive	A	if $(R[d] > 0)$ PC = addr
E	jump register	RR	PC = $R[d]$
F	jump and link	A	$R[d] = \text{PC} + 1$; PC = addr

TOY code II

Q. What does this TOY program do?

```
10: 8210    R[2] = 8210
11: 7D01    R[D] = 0001
12: 2A0D    R[A] = FFFF
13: 432A    R[3] = 7DEF
14: 9315    M[15] = R[3]
15: 93FF    stdout = R[3]
16: 9DFF    stdout = R[D]
17: 0000    halt
```

<i>opcode</i>	<i>operation</i>	<i>format</i>	<i>pseudo-code</i>
0	halt	—	halt
1	add	RR	$R[d] = R[s] + R[t]$
2	subtract	RR	$R[d] = R[s] - R[t]$
3	bitwise and	RR	$R[d] = R[s] \& R[t]$
4	bitwise xor	RR	$R[d] = R[s] \wedge R[t]$
5	shift left	RR	$R[d] = R[s] \ll R[t]$
6	shift right	RR	$R[d] = R[s] \gg R[t]$
7	load addr	A	$R[d] = \text{addr}$
8	load	A	$R[d] = M[\text{addr}]$
9	store	A	$M[\text{addr}] = R[d]$
A	load indirect	RR	$R[d] = M[R[t]]$
B	store indirect	RR	$M[R[t]] = R[d]$
C	branch zero	A	if $(R[d] == 0)$ PC = addr
D	branch positive	A	if $(R[d] > 0)$ PC = addr
E	jump register	RR	PC = $R[d]$
F	jump and link	A	$R[d] = \text{PC} + 1$; PC = addr



Lecture 18: van Neumann Machines

TOY blocking and tackling III

Q. (F11) Suppose that R[2] contains a small integer x . Match each instruction with a description of the value of R[2] after it is executed.

1000 **G**

4222 **A**

1222 **B**

5222 **E**

E022 **G**

7022 **G**

1202 **G**

A. 0

B. $2x$

C. x^2

D. 2^x

E. $x2^x$

F. $x - 2$

G. x

H. No match

opcode	operation	format	pseudo-code
0	halt	—	halt
1	add	RR	$R[d] = R[s] + R[t]$
2	subtract	RR	$R[d] = R[s] - R[t]$
3	bitwise and	RR	$R[d] = R[s] \& R[t]$
4	bitwise xor	RR	$R[d] = R[s] \wedge R[t]$
5	shift left	RR	$R[d] = R[s] \ll R[t]$
6	shift right	RR	$R[d] = R[s] \gg R[t]$
7	load addr	A	$R[d] = \text{addr}$
8	load	A	$R[d] = M[\text{addr}]$
9	store	A	$M[\text{addr}] = R[d]$
A	load indirect	RR	$R[d] = M[R[t]]$
B	store indirect	RR	$M[R[t]] = R[d]$
C	branch zero	A	if $(R[d] == 0)$ PC = addr
D	branch positive	A	if $(R[d] > 0)$ PC = addr
E	jump register	RR	PC = $R[d]$
F	jump and link	A	$R[d] = \text{PC} + 1$; PC = addr

TOY code with a loop I

Q. [S11] Consider this TOY program.

```
20: 2AAB  R[A] = R[A] - R[B]
21: DA20  if (R[A] > 0) PC = 20
22: CA24  if (R[A] == 0) PC = 24
23: 1AAB  R[A] = R[A] + R[B]
24: 0000  halt
```

A. Give the result when R[A] is 001A and R[B] is 0008.

R[A]: **0002** R[B]: **0008**

B. Give the result when R[A] is 5EAB and R[B] is 0010.

R[A]: **000B** R[B]: **0010**

C. Give equivalent Java code.

a = a % b ;

opcode	operation	format	pseudo-code
0	halt	—	halt
1	add	RR	R[d] = R[s] + R[t]
2	subtract	RR	R[d] = R[s] - R[t]
3	bitwise and	RR	R[d] = R[s] & R[t]
4	bitwise xor	RR	R[d] = R[s] ^ R[t]
5	shift left	RR	R[d] = R[s] << R[t]
6	shift right	RR	R[d] = R[s] >> R[t]
7	load addr	A	R[d] = addr
8	load	A	R[d] = M[addr]
9	store	A	M[addr] = R[d]
A	load indirect	RR	R[d] = M[R[t]]
B	store indirect	RR	M[R[t]] = R[d]
C	branch zero	A	if (R[d] == 0) PC = addr
D	branch positive	A	if (R[d] > 0) PC = addr
E	jump register	RR	PC = R[d]
F	jump and link	A	R[d] = PC + 1; PC = addr

TOY code with a loop II

Q. [S2012] What does this TOY program punch?

```
0F: 000N
10: 840F    R[4] = M[0F]
11: 4555    R[5] = 0000
12: 7601    R[6] = 0001
13: 1554    R[5] = R[5] + R[4]
14: 2446    R[4] = R[4] - 1
15: D413    if (R[4] > 0) PC = 13
16: 95FF    stdout = R[5]
17: 0000    halt
```

A. $N + (N - 1) + (N - 2) + \dots + 1 = N(N + 1)/2$

Q. Largest value in 0F with no overflow?

A. 0100

opcode	operation	format	pseudo-code
0	halt	—	halt
1	add	RR	$R[d] = R[s] + R[t]$
2	subtract	RR	$R[d] = R[s] - R[t]$
3	bitwise and	RR	$R[d] = R[s] \& R[t]$
4	bitwise xor	RR	$R[d] = R[s] \wedge R[t]$
5	shift left	RR	$R[d] = R[s] \ll R[t]$
6	shift right	RR	$R[d] = R[s] \gg R[t]$
7	load addr	A	$R[d] = \text{addr}$
8	load	A	$R[d] = M[\text{addr}]$
9	store	A	$M[\text{addr}] = R[d]$
A	load indirect	RR	$R[d] = M[R[t]]$
B	store indirect	RR	$M[R[t]] = R[d]$
C	branch zero	A	if $(R[d] == 0)$ PC = addr
D	branch positive	A	if $(R[d] > 0)$ PC = addr
E	jump register	RR	PC = $R[d]$
F	jump and link	A	$R[d] = PC + 1$; PC = addr

TOY code with a loop III

Q. [Ex. 6.3.21] What does this TOY program punch?

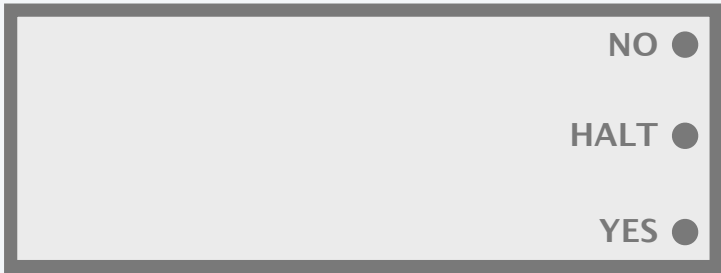
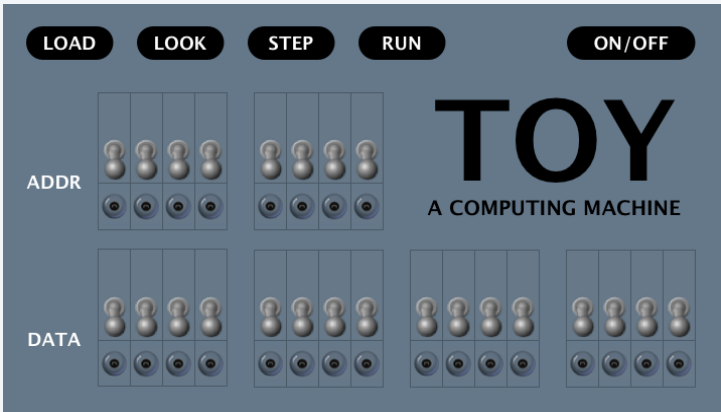
10: 7101	$R[1] = 0001$	D0: 0001
11: 72D0	$R[2] = 00D0$	D1: 00D6
12: 1421	$R[4] = R[2] + R[1]$	D2: 0005
13: A302	$R[3] = M[R[2]]$	D3: 00D8
14: 93FF	$stdout = R[3]$	D4: 0004
15: A204	$R[2] = M[R[4]]$	D5: 00D2
16: D212	$if (R[2] > 0) PC = 12$	D6: 0002
17: 0000	halt	D7: 00DA
		D8: 0006
		D9: 0000
		DA: 0003
		DB: 00D4

A. 0001 0002 0003 0004 0005 0006

opcode	operation	format	pseudo-code
0	halt	—	halt
1	add	RR	$R[d] = R[s] + R[t]$
2	subtract	RR	$R[d] = R[s] - R[t]$
3	bitwise and	RR	$R[d] = R[s] \& R[t]$
4	bitwise xor	RR	$R[d] = R[s] \wedge R[t]$
5	shift left	RR	$R[d] = R[s] \ll R[t]$
6	shift right	RR	$R[d] = R[s] \gg R[t]$
7	load addr	A	$R[d] = \text{addr}$
8	load	A	$R[d] = M[\text{addr}]$
9	store	A	$M[\text{addr}] = R[d]$
A	load indirect	RR	$R[d] = M[R[t]]$
B	store indirect	RR	$M[R[t]] = R[d]$
C	branch zero	A	$if (R[d] == 0) PC = \text{addr}$
D	branch positive	A	$if (R[d] > 0) PC = \text{addr}$
E	jump register	RR	$PC = R[d]$
F	jump and link	A	$R[d] = PC + 1; PC = \text{addr}$

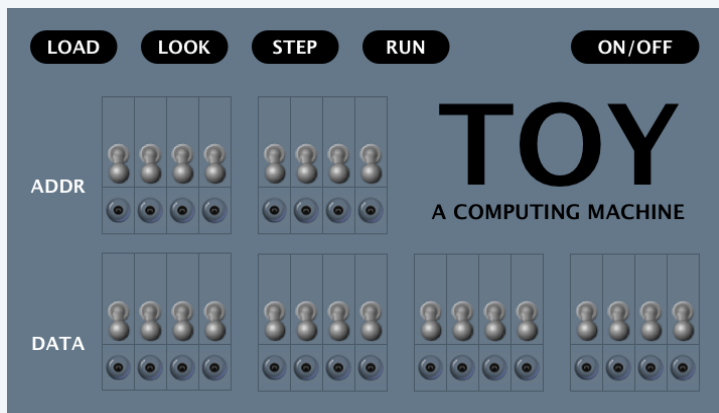
Key idea I

Q. Can a TOY do anything that a UTM can (and vice-versa)?



Key idea II

Q. Can a TOY do anything that *your computer* can (and vice-versa)?





Lecture 19: Combinational Circuits

Boolean logic I

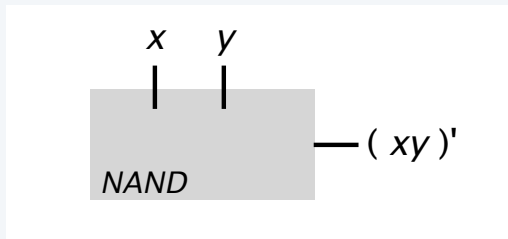
Q. [Ex. 7.1.3] Give a truth-table proof showing that $x + yz = (x + y)(x + z)$.

A.

x	y	z	yz	$x + yz$	$x + y$	$x + z$	$(x + y)(x + z)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

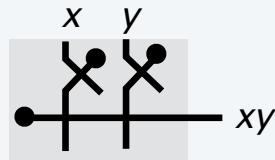
NAND gate

Q. Give a circuit that implements the NAND function.

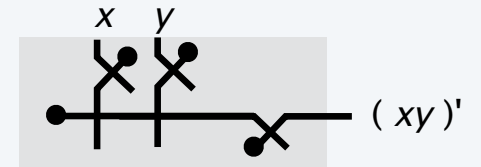


x	y	$NAND$
0	0	1
0	1	1
1	0	1
1	1	0

Hints.

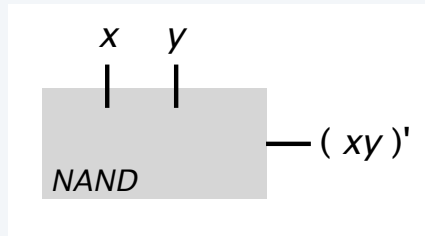


A.

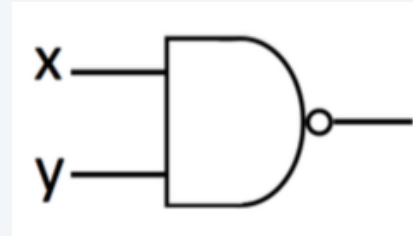


Notation

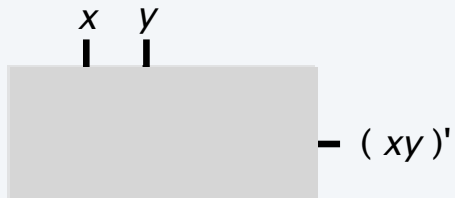
Q. Why this?



and not this?



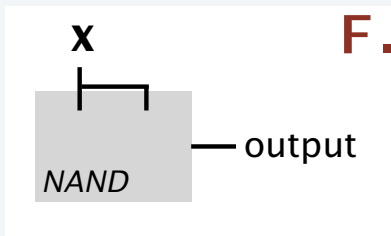
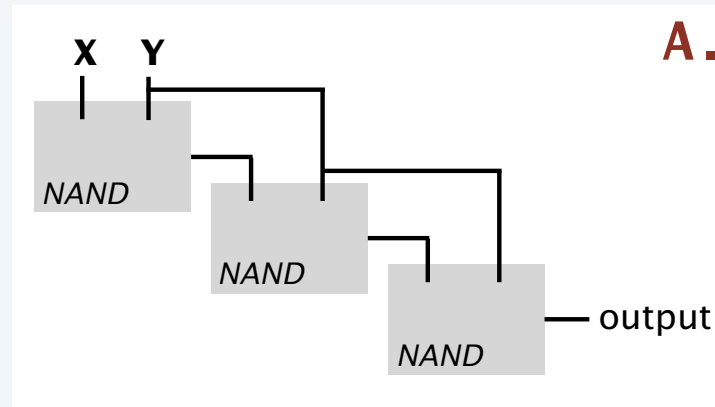
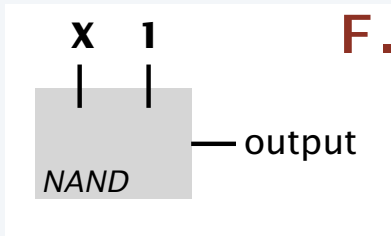
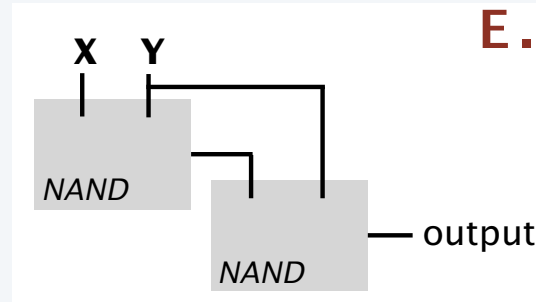
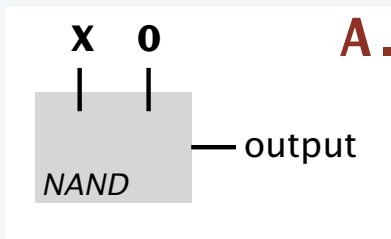
A.



A. Eliminates a layer of abstraction (stay tuned).

Simple circuits I

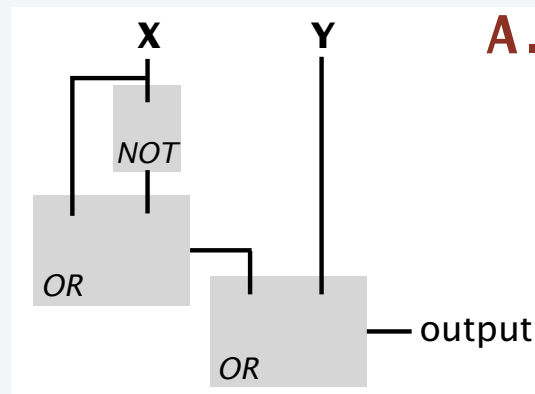
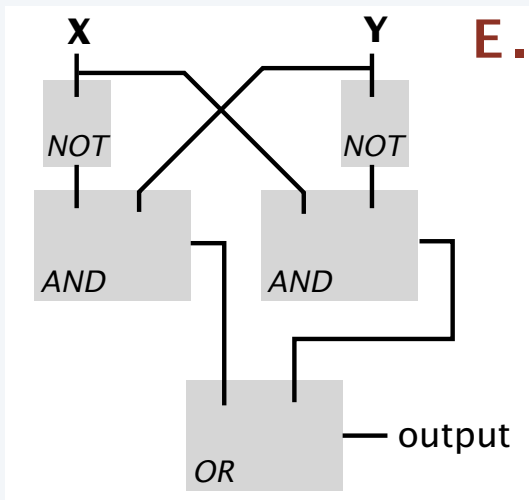
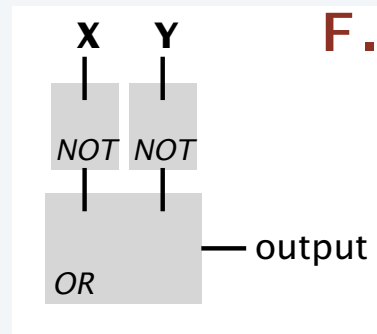
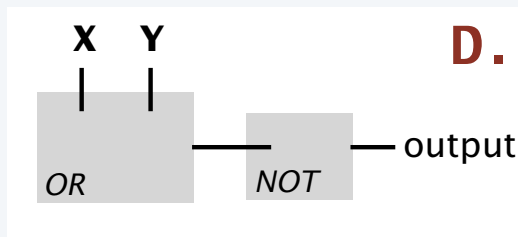
Q. [Spring 2014] Label each circuit with the letter corresponding to its output.



- A.** 1
- B.** 0
- C.** xy
- D.** $x'y'$
- E.** $xy + y'$
- F.** x'

Simple circuits II

Q. [Spring 2013] Label each circuit with the letter corresponding to its output.



- A.** always 1
- B.** always 0
- C.** 1 iff X and Y are equal
- D.** 1 iff X and Y are both 0
- E.** 1 iff X and Y are not equal
- F.** 0 iff X and Y are both 1

Boolean logic II

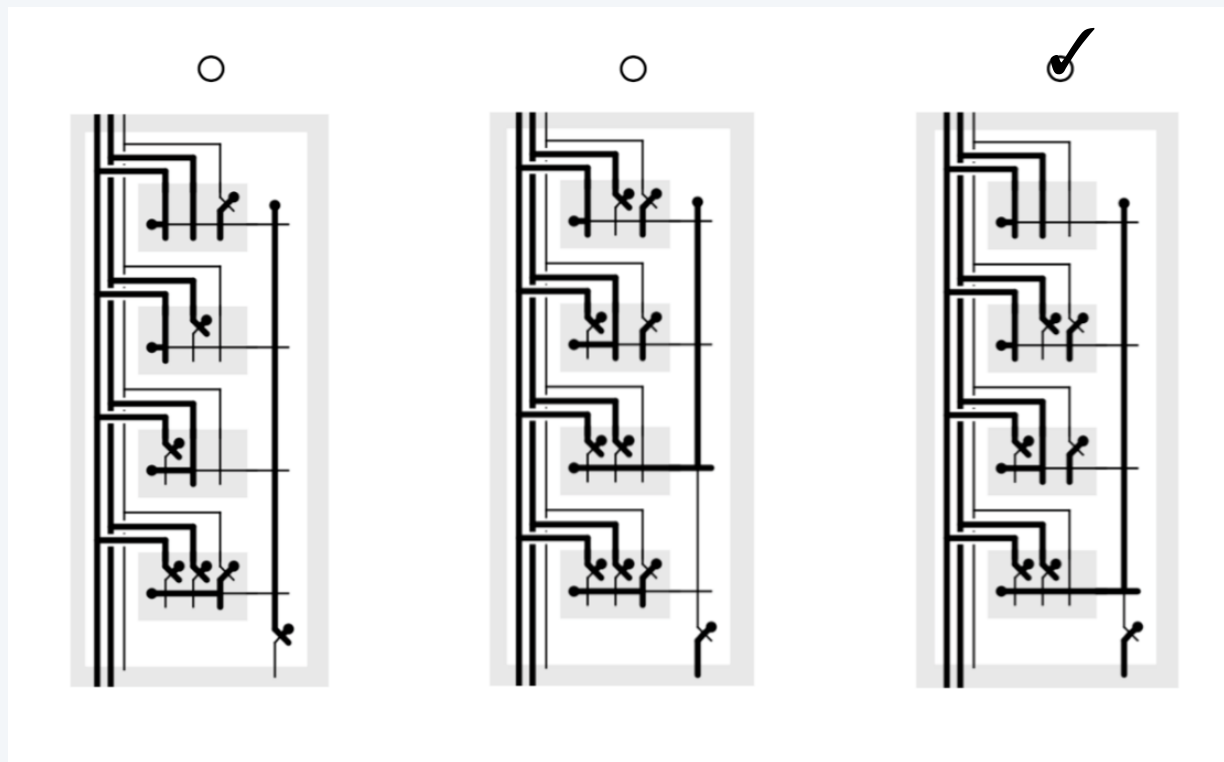
Q. [F 2014] Give a truth table for *EVEN PARITY* (number of 1s in inputs is even).

A.

x	y	z	<i>EVEN PARITY</i>
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Combinational circuit

Q. [F 2014] Which circuit is computing *EVEN PARITY* for 1 1 0?



Incrementer

Q. [7.3.18] Draw a circuit that *increments* a 4-bit number.

A.

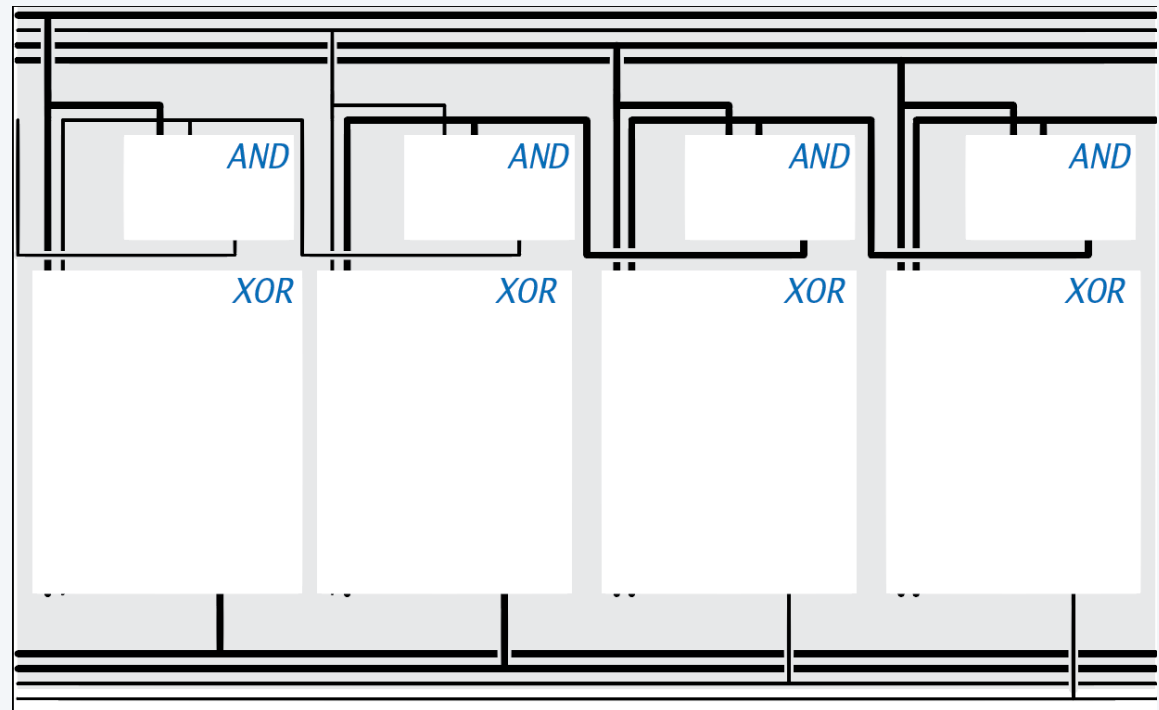
C_4	C_3	C_2	C_1	1
	X_3	X_2	X_1	X_0
	Z_3	Z_2	Z_1	Z_0

carry bit

X_i	C_i	C_{i+1}
0	0	0
0	1	0
1	0	0
1	1	1

sum bit

X_i	C_i	Z_i
0	0	0
0	1	1
1	0	1
1	1	0



Incrementer

Q. [7.3.18] Draw a circuit that *increments* a 4-bit number.

A.

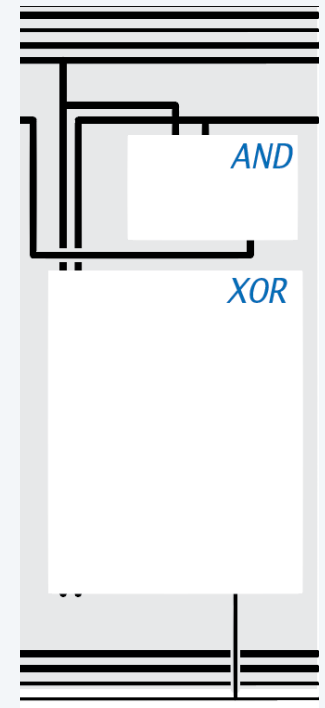
C_4	C_3	C_2	C_1	1
	X_3	X_2	X_1	X_0
	Z_3	Z_2	Z_1	Z_0

carry bit

X_i	C_i	C_{i+1}
0	0	0
0	1	0
1	0	0
1	1	1

sum bit

X_i	C_i	Z_i
0	0	0
0	1	1
1	0	1
1	1	0





Lecture 20: CPU

Number representation

Q. (Spring 2013 Q1) Why can $3/2$ be represented as an exact double in Java but $1/10$ cannot?

A. *No questions on floating point representation in this exam.*

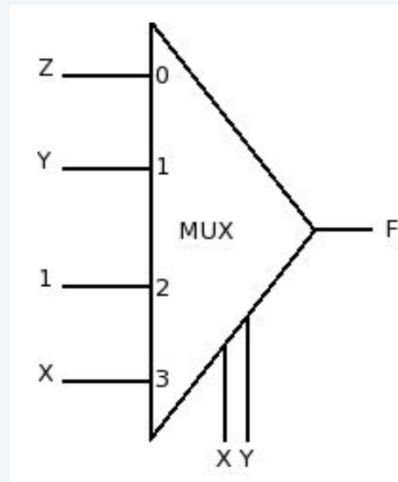
A. $3/2 = 1 + 1/2$, but no way to represent $1/10$ as sum of decreasing powers of $1/2$.

Circuit components

Q. (Spring 2012 Q5) How to write the truth table?

A. Try all possibilities.

A *multiplexer* switches the addressed input value to the output (p. 942)



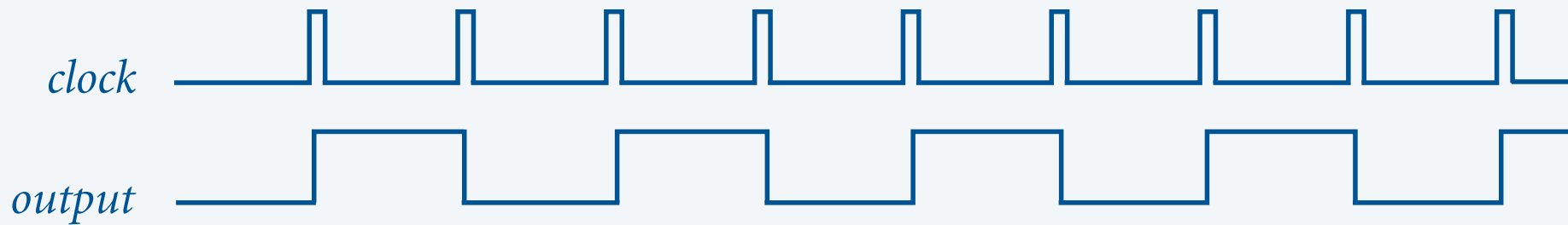
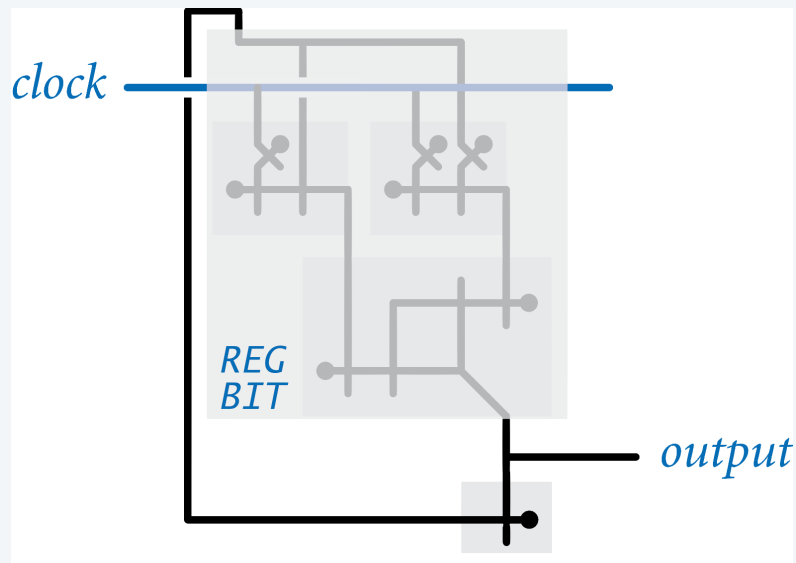
x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Q. Do we need to memorize what the complex circuits are composed of?

A. No, you need to know what they *do*.

Sequential circuit I

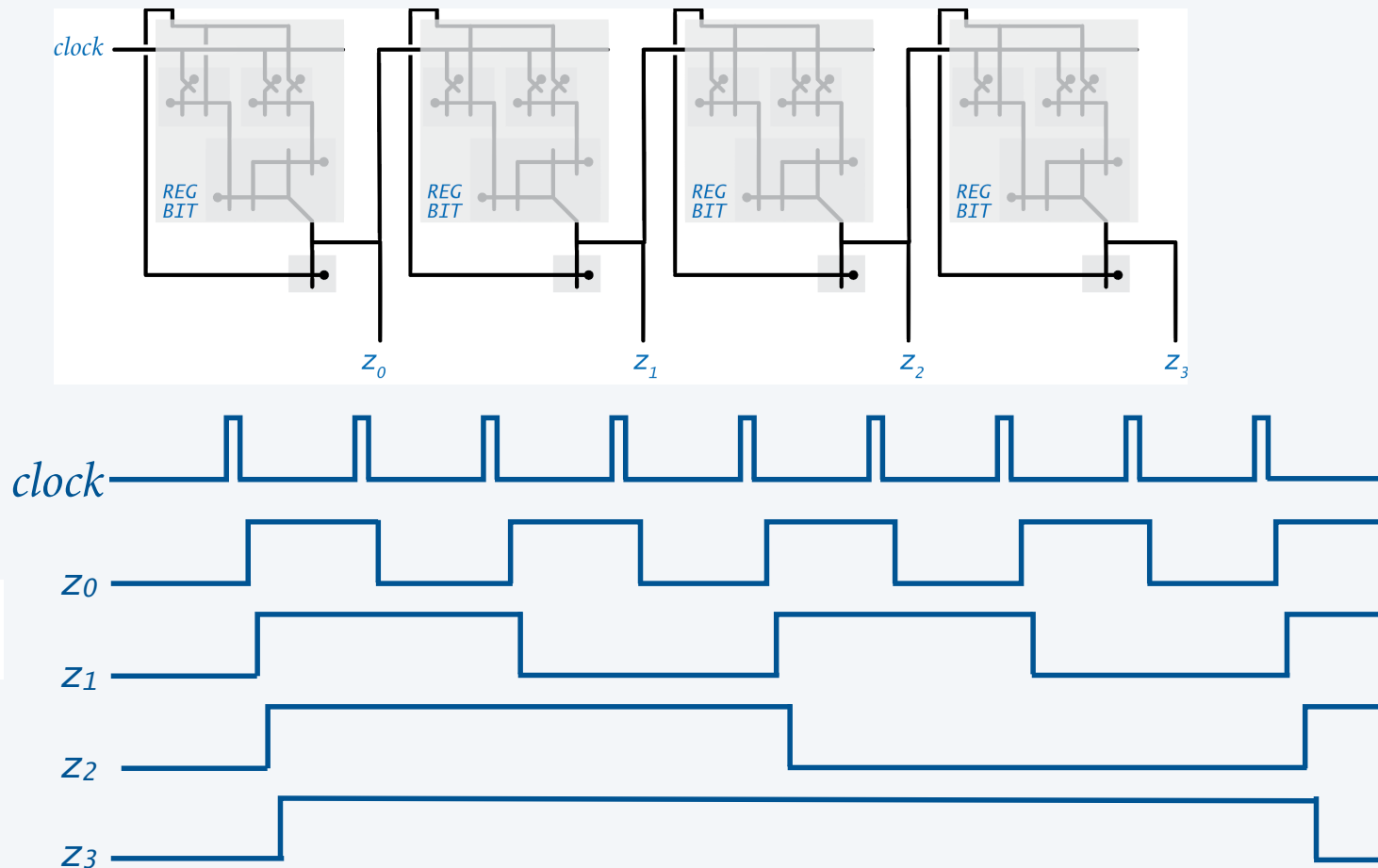
Q. (Ex. 7.4.3) Draw the response of this circuit to the given clock signal.



A.

Sequential circuit II

Q. (Ex. 7.4.16) Draw the response of this circuit to the given clock signal.



$\sim z_3$	$\sim z_2$	$\sim z_1$	$\sim z_0$
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0

A.

Matching I

Q. (Fall 2014 Q10) Identify each of the CPU components as *combinational* or *sequential*.

	<i>combinational</i>	<i>sequential</i>
ADDR MUX	<input checked="" type="radio"/>	<input type="radio"/>
ALU	<input checked="" type="radio"/>	<input type="radio"/>
CLOCK	<input type="radio"/>	<input checked="" type="radio"/>
CONTROL	<input checked="" type="radio"/>	<input type="radio"/>
IR	<input type="radio"/>	<input checked="" type="radio"/>
MEMORY	<input type="radio"/>	<input checked="" type="radio"/>
PC	<input type="radio"/>	<input checked="" type="radio"/>
R	<input type="radio"/>	<input checked="" type="radio"/>
R MUX	<input checked="" type="radio"/>	<input type="radio"/>

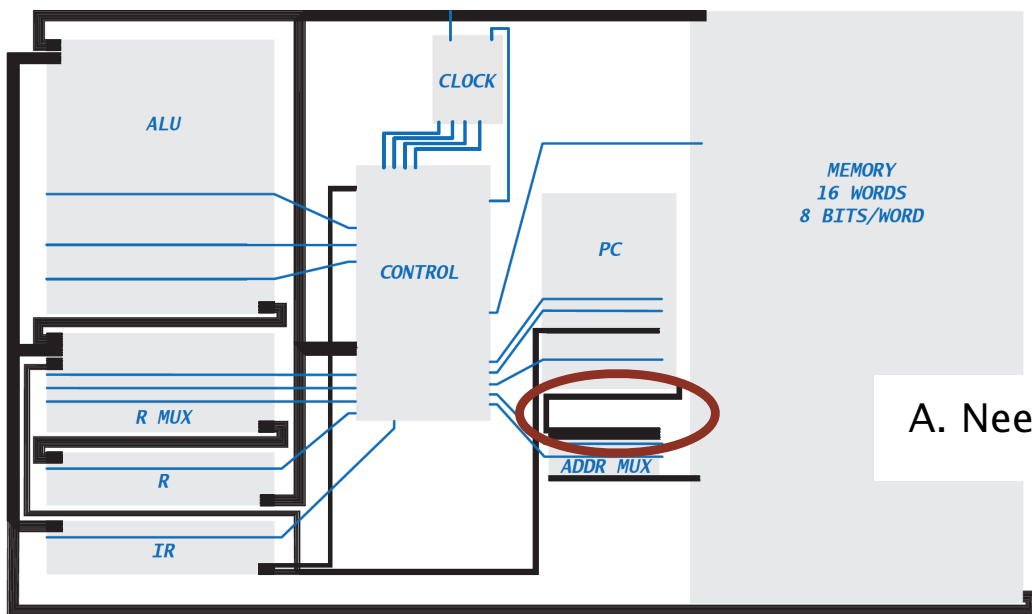
Matching II

Q. (Fall 2014 Q10, modified) Match each CPU component to a description.

ADDR MUX	B.	A. holds address of current instruction
ALU	I.	B. inputs from IR and PC
CLOCK	H.	C. contents ultimately come from one of three sources
CONTROL	D.	D. decodes instructions
IR	G.	E. selects register inputs
MEMORY	F.	F. big sequential circuit
PC	A.	G. holds current instruction
R	C.	H. sequential circuit that produces periodic pulses
R MUX	E.	I. computes boolean function values

Data paths I

Q. (Spring 2015 Q7) Why is A checked as a needed data path for every instruction?



OLD NOTATION

A is the path from the PC to the MA mux

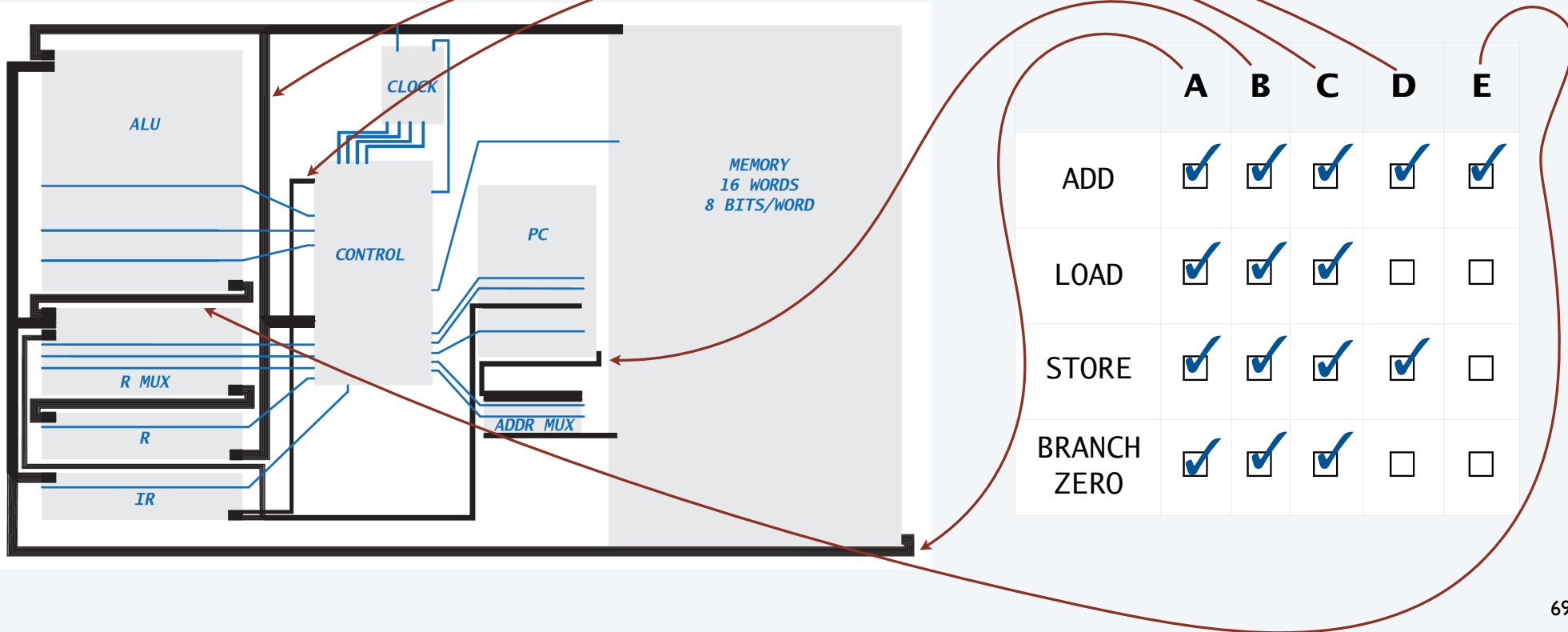
A. Need this path to fetch *the instruction*.

Q. How do we study from Lecture 20?

A. Read Chapter 7.

Data paths

Q. (Spring 2015 Q7) Check all the boxes that correspond to the datapaths needed to fetch and execute the following TOY-8 instructions.



TOY-8 CPU

