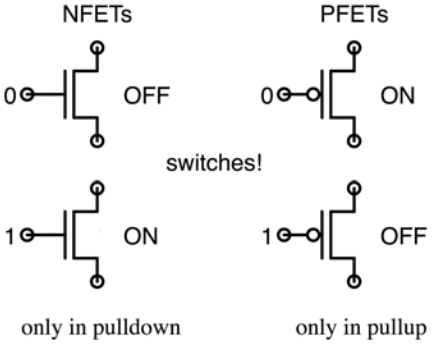
Computation Structures

CMOS Technology Worksheet

Concept Inventory:

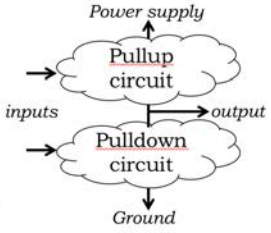
- PFET, NFET: voltage controlled switches
- CMOS composition rules: complementary pullup and pulldown
- CMOS gates are naturally inverting
- t_{PD} and t_{CD} timing specifications
- Lenient gates

Notes:



We want *complementary* pullup and pulldown logic, i.e., the pulldown should be "on" when the pullup is "off" and vice versa.

pullup	pulldown	F(inputs)
on	off	driven "1"
off	on	driven "0"
on	on	driven "X"
off	off	no connection



CMOS gates are naturally inverting:

- Rising input (0 to 1): NFETs turn on, PFETs turn off; if output changes, it falls (1 to 0)
- Falling input (1 to 0): NFETs turn off, PFETs turn on; if output changes, it rises (0 to 1)

Timing:

- t_{PD} (propagation delay): how long after inputs are stable and valid until outputs are stable and valid = max over all paths from input to output (sum of component t_{PD} along path)
  - t_{PD} specification is an upper bound on all measured propagation delays
- t_{CD} (contamination delay): how long output stays valid after inputs go invalid = min over all paths from input to output (sum of component t_{CD} along path)
  - t_{CD} specification is a lower bound on all measured contamination delays

Lenient gate:

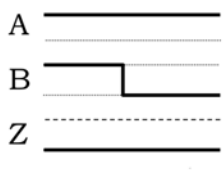
- If a subset of a lenient gate's inputs is suffice to guarantee an specific output value (i.e., the values of the other inputs don't matter in this case), then the output will remain valid and stable by transitions on the irrelevant inputs.
- CMOS gates are naturally lenient

NOR:

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

Lenient NOR:

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



Problem 1.

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

(A) Which of the above CMOS pulldown circuits would implement F if the corresponding complementary pullup circuit was also provided? For each pulldown, select Yes if it is a valid pulldown for F, and No if it is not a valid pulldown for F.

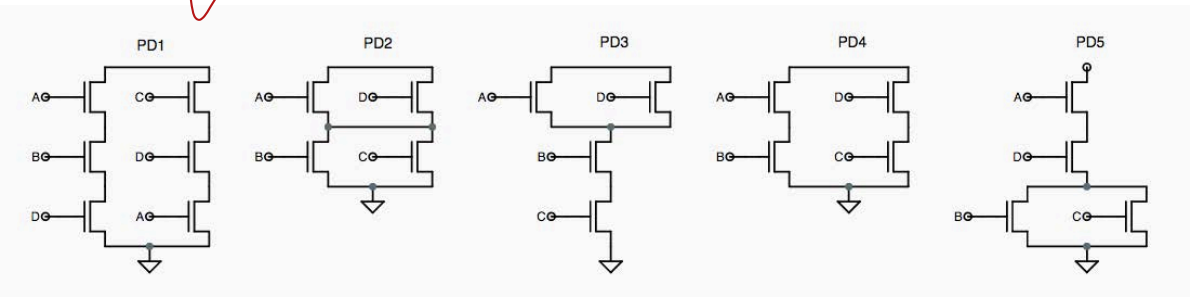
PD1 (Yes/No): Yes

PD2 (Yes/No): No

PD3 (Yes/No): No

PD4 (Yes/No): No

PD5 (Yes/No): Yes



(B) Are all the implementations you selected for part (A) lenient?

All lenient (Yes/No): Yes.

Problem 2.

(A) A single CMOS gate, consisting of an output node connected to a single PFET-based pullup circuit and a single NFET-based pulldown circuit (as described in lecture) computes F(A, B, C, D). It is observed that F(1, 0, 1, 0) = 1. What can you say about the following values?

1->0. **(circle one) F(0, 0, 1, 0) = : 0 … 1 … (can't say)**

0->1 **(circle one) F(1, 1, 1, 0) = : 0 … 1 … (can't say)**

(circle one) F(1, 1, 1, 1) = : 0 … 1 … (can't say)

all (1) => 0.

(B) The Boolean function F(A,B,C) can be implemented using a *single* CMOS gate operating as a combinational device that obeys the static discipline. It's known that F(1,1,0) = 1 and F(0,1,1) = 0. What can be determined about the value of F in the following cases? Please circle one of "0", "1" or "Can't tell".

(circle one) F(1,0,0) = 0 … 1 … Can't tell

(circle one) F(1,0,1) = 0 … 1 … Can't tell

(circle one) F(1,1,1) = 0 … 1 … Can't tell

(C) A single CMOS gate, consisting of an output node connected to a single pullup circuit containing one or more PFETs and a single pulldown circuit containing one or more NFETs (as described in lecture), computes $F(A,B)$. F has the property that for all A, $F(A,0) = \overline{F(A,1)}$. What can you say about the value of $F(1,0)$?

$F(1,0) = \overline{F(1,1)}$.

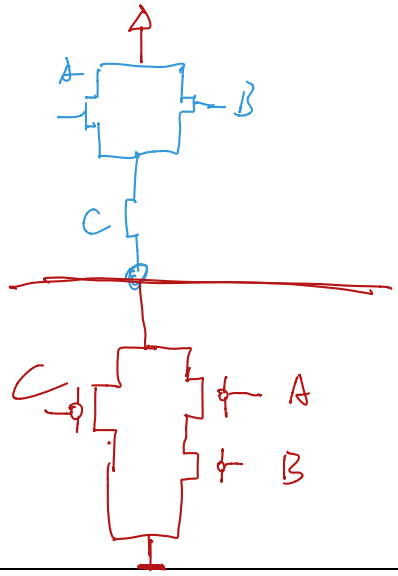
(circle one): F(1,0) = 1 … 0 … can't tell

$= 1.$ $= 0.$

Problem 3.

For each of the functions F and G, if the function can be implemented using a **single CMOS gate**, please draw the corresponding single CMOS gate. If it cannot be implemented using a single CMOS gate, then write NONE. **For full credit use a minimum number of FETs.**

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

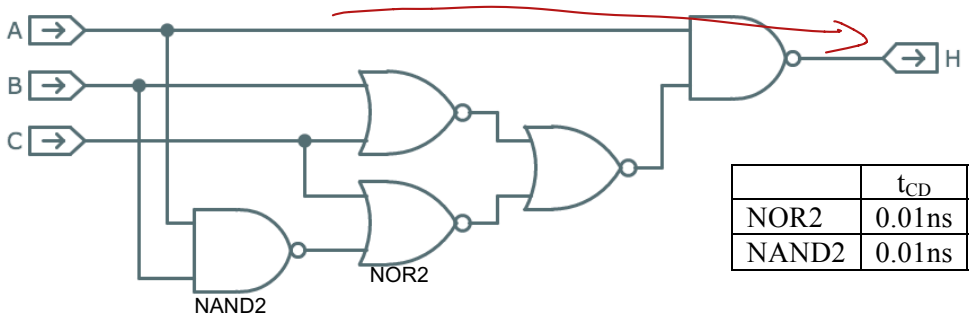
Draw CMOS implementation of F(A,B,C) below or write NONE if F cannot be implemented as single CMOS gate.	**Draw CMOS implementation of G(A,B,C) below or write NONE if G cannot be implemented as single CMOS gate.**
No No ? F(1,1,1) => 1. Not a single CMOS.	

B 0→1 (pull down ?).
F (1→0).
C 0→1 pull down
F(1→0)

Problem 4.

Consider the Boolean function that has the truth table shown to the right; a possible implementation as a combinational circuit is shown in the schematic below. You may assume that the NOR2 and NAND2 components are combinational.

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



	t_{CD}	t_{PD}
NOR2	0.01ns	0.05ns
NAND2	0.01ns	0.03ns

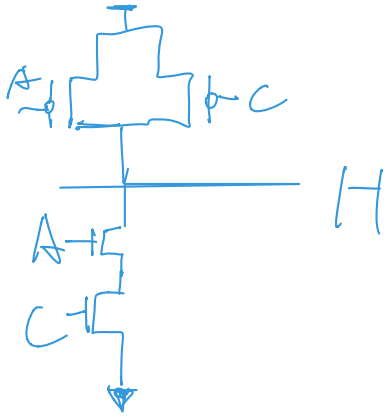
(A) Using the timing specifications shown above for NOR2 and NAND2, compute the contamination and propagation delay for the implementation of H shown above.

为什么是0.01呢，
先等A变化，最短时间对H也会变化。

timing for H (ns): t_{CD}= 0.02 0.01 t_{PD}= 0.16

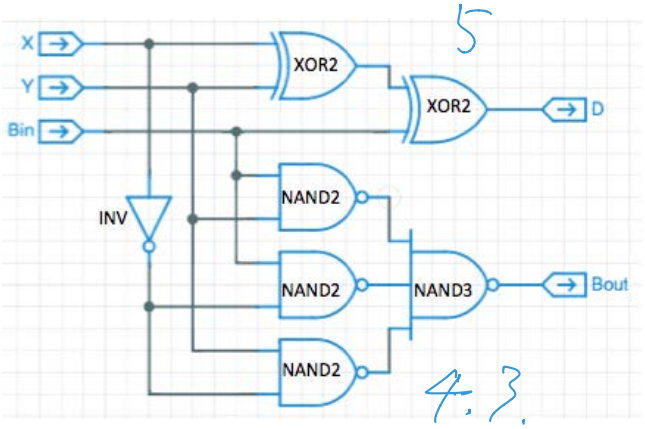
(B) Can H be implemented as a single CMOS gate (only PFETs in the pullup circuit, only NFETs in the pulldown circuit)? If so draw the MOSFET schematic for H to the right, otherwise write "NO".

Draw schematic or write "NO"



Problem 5.

A gate-level schematic is shown below. Using the t_{CD} and t_{PD} information for the gate components shown in the table below, compute t_{CD} and t_{PD} for the circuit.



Compute timing specs:

t_{CD} = 0.5 ns

t_{PD} = 5 ns

Gate	t_{CD}	t_{PD}
INV	0.1ns	1.0ns
NAND2	0.2ns	1.5ns
NAND3	0.3ns	1.8ns
XOR2	0.6ns	2.5ns

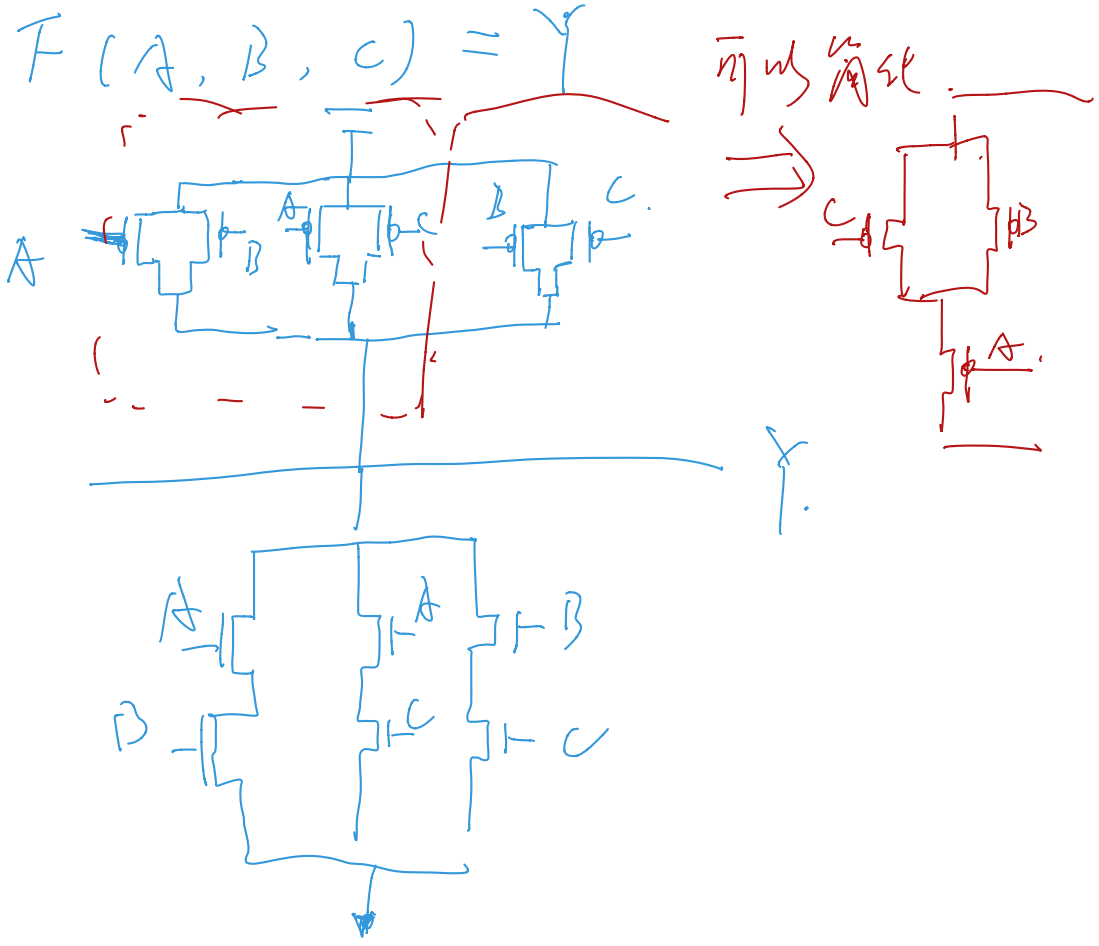
Problem 6.

A minority gate has three inputs (call them A, B, C) and one output (call it Y). The output will be 0 if two or more of the inputs are 1, and 1 if two or more of the inputs are 0.

In the space below, draw the *pulldown* circuit for a single CMOS gate that implements the minority function, using the minimum number of NFETs. You needn't draw the *pullup* circuit.

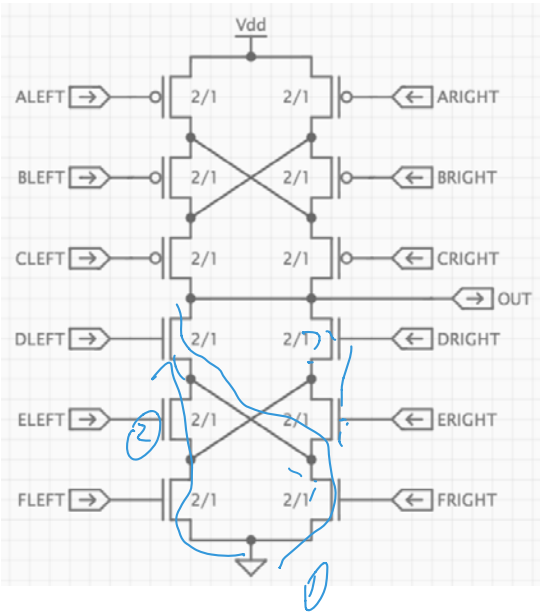
If you're convinced that the function cannot be implemented as a single CMOS gate, give a brief, convincing explanation.

Can it be implemented as single CMOS gate? Circle one: YES can't tell NO



Problem 7.

In his bid for the Lemelson Prize, Ben Bitdiddle has invented the "flexible gate," a single CMOS gate that implements different functions depending how its inputs are wired up. The FlexGate® (see figure at right) uses 6 PFETs in its pullup circuit and 6 NFETs in its pulldown circuit..



Each of the FlexGate's twelve inputs can connected to an input signal (X, Y, …), GND (logical "0") or VDD (logical "1"). To show off its versatility, Ben has asked you to show how to hook up the inputs so the FlexGate computes several different functions whose Boolean equations are given below. Associated with each equation is a table with 12 entries; in each cell of the table please write an input name, GND or VDD as appropriate. Note that there may be several possible implementations for each of the three functions – any correct answer will be acceptable. Hint: there should be an entry in each cell, i.e., a connection should be specified each input!

If the desired function cannot be implemented, please draw a big "X" through the table.

Fill in tables below or mark with "X"

$OUT = \overline{X \cdot Y}$

input	LEFT	RIGHT
A	G	G
B	V	V
C	X	Y
D	X	G
E	G	G
F	G	Y

$OUT = \overline{X + Y + Z}$

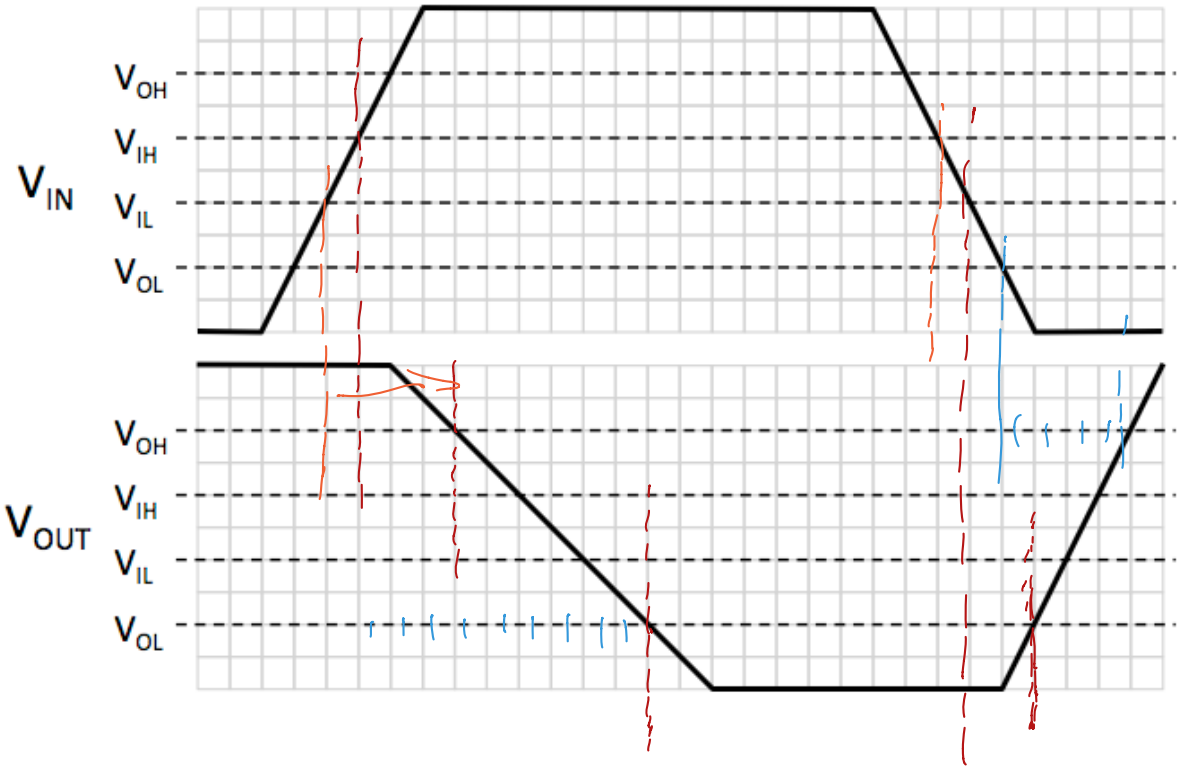
input	LEFT	RIGHT
A	X	V
B	V	Y
C	Z	V
D	X.	X VDD V
E	G.	Y. V
F	Z	X VDD

$OUT = \overline{X + Y \cdot Z}$

input	LEFT	RIGHT
A	X	V
B	V	G.
C	Y	Z
D	X	V
E	G.	Y
F	Z	V

Problem 8.

The response of a combinational gate to a test input waveform is shown below. Each horizontal division of the plot represents 10 ps.



(A) Based on the figure below, what is an appropriate choice for the contamination delay of the gate?

30 ps or 20 ps. ⇒ 30 ps.

invalid ⇒ invalid.

min (40 ps, 30 ps.) .

(B) Based on the figure below, what is an appropriate choice for the propagation delay of the gate?

valid ⇒ valid.

90 ps. or 50 ps ⇒ 90 ps

max (90 ps, 50 ps) ⇒ 90 ps.

Lec. 4.

Computation Structures

Combinational Logic Worksheet

Concept Inventory:

- Truth tables ↔ sum-of-products equations
- implementation using NOT/AND/OR
- Demorgan's Law, implementation using NAND/NOR
- Simplification, truth tables w/ don't cares
- Karnaugh maps
- Implementation using MUXes and ROMs

Here's a Design Approach

Truth Table

C	B	A	Y
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

1. Write out our functional spec as a truth table
2. Write down a Boolean expression with terms covering each '1' in the output:

$$Y = \bar{C}B\bar{A} + \bar{C}BA + C\bar{B}\bar{A} + CBA$$

3. We'll show how to build a circuit using this equation in the next two slides.

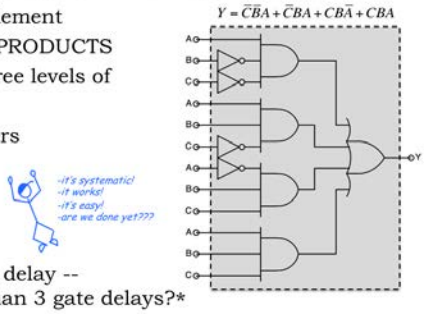
This approach will always give us Boolean expressions in a particular form: SUM-OF-PRODUCTS

6.004 Computation Structures — L04: Logic Synthesis, Slide #3

Straightforward Synthesis

We can implement SUM-OF-PRODUCTS with just three levels of logic:

1. Inverters
2. ANDs
3. OR



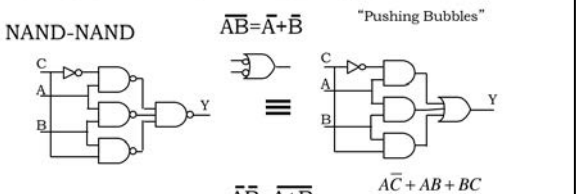
Propagation delay -- No more than 3 gate delays?*

*assuming gates with an arbitrary number of inputs, which, as we'll see, isn't a good assumption!

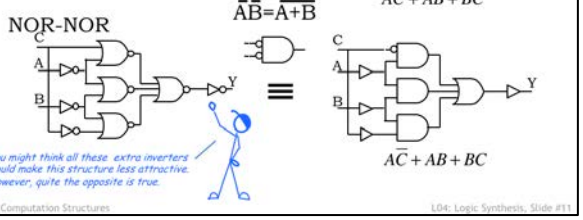
6.004 Computation Structures — L04: Logic Synthesis, Slide #5

CMOS Sum-of-products Implementation

NAND-NAND



NOR-NOR



6.004 Computation Structures — L04: Logic Synthesis, Slide #11

Finding Implicants

An implicant

- is a rectangular region of the K-map where the function has the value 1 (i.e., a region that will need to be described by one or more product terms in the sum-of-products)
- has a width and length that must be a power of 2: 1, 2, 4
- can overlap other implicants
- is a prime implicant if it is not completely contained in any other implicant.

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

$A\bar{C}$, $\bar{A}\bar{B}C$

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

$\bar{A}\bar{C}$, $\bar{B}$

- can be uniquely identified by a single product term. The larger the implicant, the smaller the product term.

6.004 Computation Structures — L04: Logic Synthesis, Slide #18

Write Down Equations

Picking just enough prime implicants to cover all the 1's in the KMap, combine equations to form minimal sum-of-products.

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

$Y = A\bar{C} + BC$

We're done!

\AB CD\	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

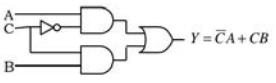
$Y = D + B\bar{C} + A\bar{C} + \bar{B}C$

Minimal SOP is not necessarily unique!

6.004 Computation Structures — L04: Logic Synthesis, Slide #20

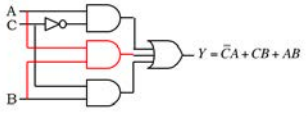
Prime Implicants, Glitches & Leniency

This circuit produces a glitch on Y when A=1, B=1, C: 1→0



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

To make the circuit lenient, include product terms for ALL prime implicants.

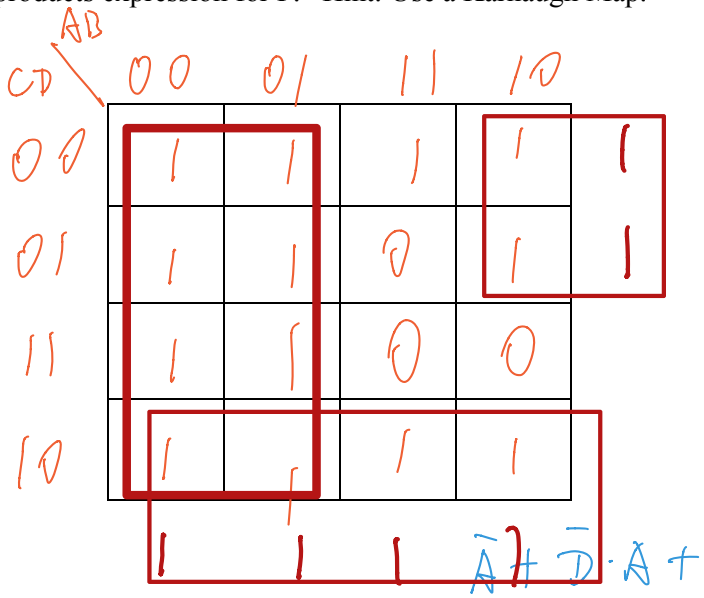


6.004 Computation Structures — L04: Logic Synthesis, Slide #21

Problem 1.

Given a function F defined by the truth table to the right, provide a minimal sum-of-products expression for F. Hint: Use a Karnaugh Map.

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

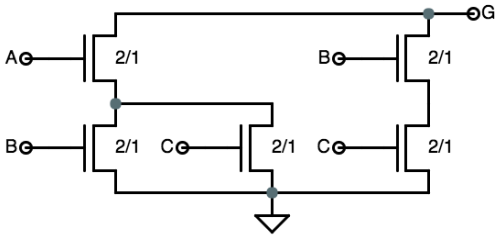


Minimal Sum-of-products Expression for F: _____

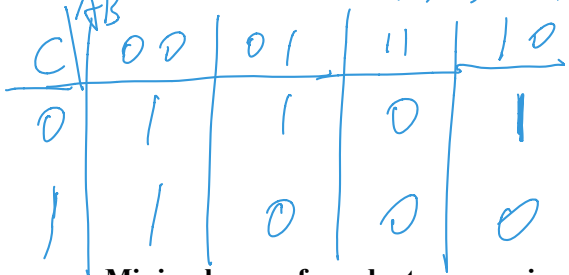
Problem 2.

$F = \bar{A} + \bar{D} + \bar{C}\cdot\bar{A}\cdot\bar{B} + \bar{C}\cdot B$

(A) A correctly-formed CMOS gate implementing G(A,B,C) uses the pulldown circuit shown on the right. Please give a *minimal sum-of-products expression* for G(A,B,C). Hint: use a Karnaugh map!



$G(A,B,C) = A\cdot(B+C) + B\bar{C}$



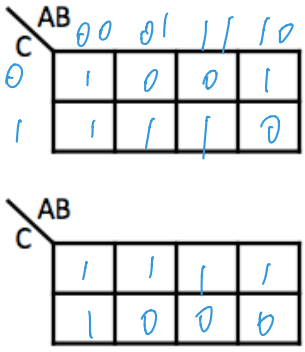
Minimal sum-of-products expression for G(A,B,C): $\bar{C}\cdot\bar{A} + \bar{A}\cdot\bar{B} + \bar{C}\cdot\bar{B}$

(B) Is the function G(A,B,C) from part (B) universal? In other words, can we implement any Boolean function using combinational circuits built only from G gates and the Boolean constants 0 and 1?

Problem 3.

Consider the following truth table which defines two functions F and G of three input variables (A, B, and C).

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



Give the **minimal sum of products** (minimal SOP) logic equation for each of the two functions. Then determine if the minimum sum of products expression would result in a **lenient** implementation of the function. If it does, then enter "**SAME**" for the lenient SOP expression. **If not**, specify what sum of products expression would result in a lenient implementation. Hint: Use Karnaugh maps above to determine the minimal sum of products.

B → 0, 1 ⇒

Minimal sum of products F(A,B,C) = $\bar{C}\cdot\bar{B} + \bar{A}\cdot\bar{B} + C\cdot B$

怎么看这个呢?

当 $\bar{A}=0$, $C=1$ 时, $F(A,B,C)=1$.

Does minimal SOP for F result in a lenient circuit (circle one)? Yes No

If "No", give lenient SOP expression for F(A,B,C) = $\bar{C}\cdot\bar{B} + \bar{A}\cdot\bar{B} + C\cdot B + \bar{A}\cdot C$

感觉是选择. 下一个值在某个变量从 1→0 时 0→1 这时候, 值不变, 但又依赖这个变量,

Minimal sum of products G(A,B,C) = $\bar{C} + \bar{A}\cdot\bar{B}$

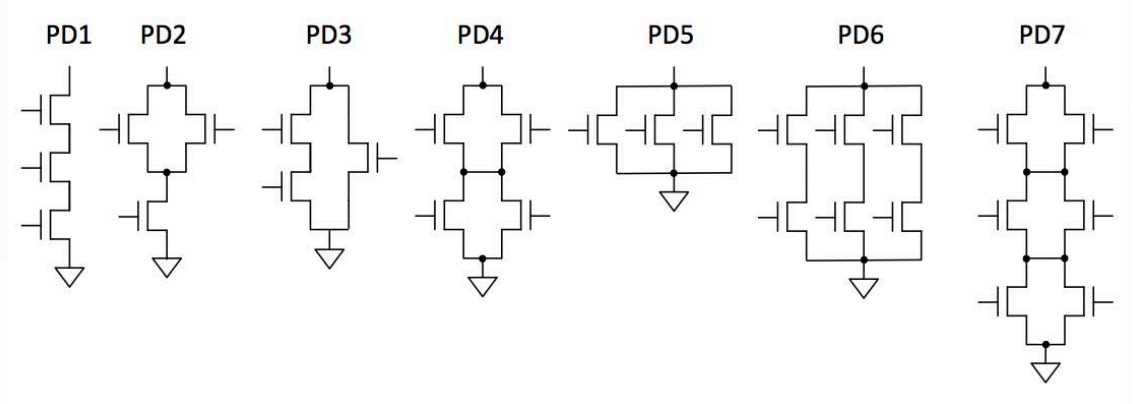
这时候由于 inverter 一延迟, 就会出现 unlenient.

Does minimal SOP for G result in a lenient circuit (circle one)? Yes No

If "No", give lenient SOP expression for G(A,B,C) = _____

Problem 4.

You are trying to select pulldowns for several 3- and 4-input CMOS gate designs. The Pulldowns-R-Us website offers seven different pulldowns, given names PD1 through PD7, diagrammed below:



The web site explains that the customer can choose which inputs or constants (GND, VDD) are connected to each NFET, allowing their pulldowns to be used in various ways to build gates with various numbers of inputs. Since Pulldowns-R-Us charges by transistor, you are interested in selecting pulldowns using the minimum number of transistors for each of the 3-input gates you are designing.

For each of the following 3- and 4-input Boolean functions, choose the appropriate pulldown design, i.e., the one which, properly connected, implements that gate's pulldown using the *minimum number* of transistors. This may require applying Demorgan's Laws or minimizing the logic equation first. If none of the above pulldowns meets this goal, write "NONE".

pull down
1 ⇒ 0.
0 ⇒ 1.
∴ F() は反.

(A) $\overline{F(A,B,C)} = \overline{A + (B \cdot C)}$ **Choice or NONE:** 3.

(B) $F(A,B,C) = A + \overline{B \cdot C}$ **Choice or NONE:** _____

$= \bar{A} \cdot \overline{(\overline{B \cdot C})} = \bar{A} \cdot (B \cdot C)$.

(C) $F(A,B,C) = (\overline{A} \cdot \overline{B}) + \overline{C}$ **Choice or NONE:** 2.

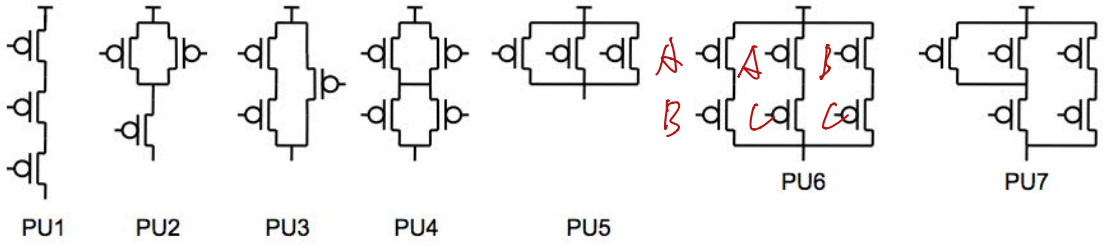
$= C \cdot (\bar{A} \cdot \bar{B}) = C \cdot (A + B)$.

(D) $F(A,B,C,D) = \overline{A + C \cdot (B + D)}$ **Choice or NONE:** 4.

$= \bar{A} \cdot (C \cdot (B + D))$.

Problem 5.

You are trying to select pullups for several 3-input CMOS gate designs. The Pullups Galore web site offers seven different pullups, given names PU1 through PU7, diagrammed below:



The web site explains that the customer can choose which inputs are connected to each PFET, allowing their pullups to be used in various ways to build gates with various numbers of inputs. Since Pullups Galore charges by transistor, you are interested in selecting pullups using the minimum number of transistors for each of the 3-input gates you are designing.

For each of the following 3-input Boolean functions, choose the appropriate pullup design, i.e., the one which, properly connected, implements that gate's pullup using the *minimum number* of transistors. This may require minimizing the logic equation first. If none of the above pullups meets this goal, write "NONE".

(A) $F(A,B,C) = \overline{A} + \overline{B} + \overline{C}$ $A=0 \,||\, B=0 \,||\, C=0$. **Choice or NONE:** 5.

(B) $F(A,B,C) = \overline{A} + \overline{B \cdot C}$ $A=0 \,||\, B \cdot C = 0$. **Choice or NONE:** 5.

(C) $F(A,B,C) = \overline{A + B \cdot C}$ $\overline{A} \cdot (\overline{B \cdot C})$ **Choice or NONE:** 2.

(D) $F(A,B,C) = A + \overline{B \cdot C}$ $A=1 \,||\, B \cdot C = 0$. **Choice or NONE:** _____

(E) $F(A,B,C) = \overline{(A+B)} + \overline{(B+C)} + \overline{(A+C)}$ ⇒ 7用的比6个多，所以选了. **Choice or NONE:** 6. ??

(F) $F(A,B,C) = \overline{(A+C) \cdot B}$ $= \overline{A} \cdot \overline{B} + \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{C}$ **Choice or NONE:** 3.

$= \overline{B} + \overline{(A+C)}$ $B=0 \,||\, A=0 \,||\, C=0$.

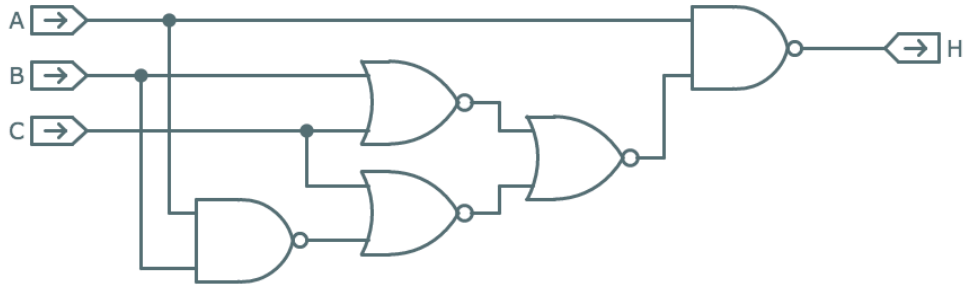
$= \overline{B} + (\overline{A} \cdot \overline{C})$. $(A=0 \,||\, C=0)$.

$\overline{A}(\overline{B} + \overline{C}) + \overline{B}\,\overline{C}$

Problem 6.

Consider the Boolean function $H(A,B,C) = \overline{A} + \overline{B}\cdot\overline{C} + A\cdot B\cdot\overline{C}$. Its truth table is shown to the right and a possible implementation is shown in the schematic below.

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



(A) Give a minimal sum-of-products expression for H. A couple of scratch 3-input Karnaugh map templates are provided for your convenience.

minimal sum-of-products expression for H: $\overline{C} + \overline{A}$

	00	01	11	10
0	1	1	1	1
1	1	1	0	0

	00	01	11	10
0	1	0	1	0
1	0	1	0	1

No simplify.

(B) What is the largest number of product terms possible in a minimal sum-of-products expression for a 3-input, 1-output Boolean function?

Largest number of product terms possible: 4

max = 4.

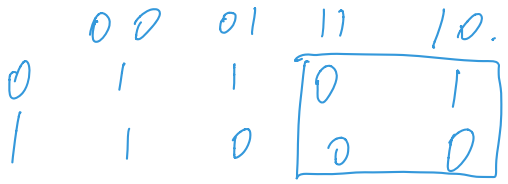
	00	01	11	10
0	1	1	0	1
1	0	1	1	1

Problem 7.

A minority gate has three inputs (call them A, B, C) and one output (call it Y). The output will be 0 if two or more of the inputs are 1, and 1 if two or more of the inputs are 0.

(A) Give a *minimal* **sum-of-products** Boolean expression for the minority gates output Y, in terms of its three inputs A, B, and C.

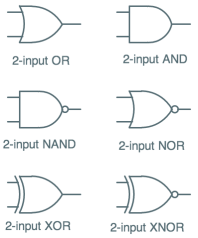
Minimal SOP expression: Y = $\bar{C}\cdot\bar{A} + \bar{C}\cdot\bar{B} + \bar{A}\cdot\bar{B}$

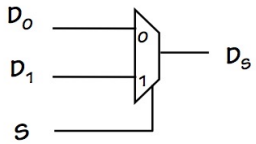


(B) Is a minority gate *universal*, in the sense that using only minority gates (along with constants 0 and 1) its possible to implement arbitrary combinational logic functions?

Universal? Circle one: **YES** **can't tell** **NO**

Y(1, B, C).

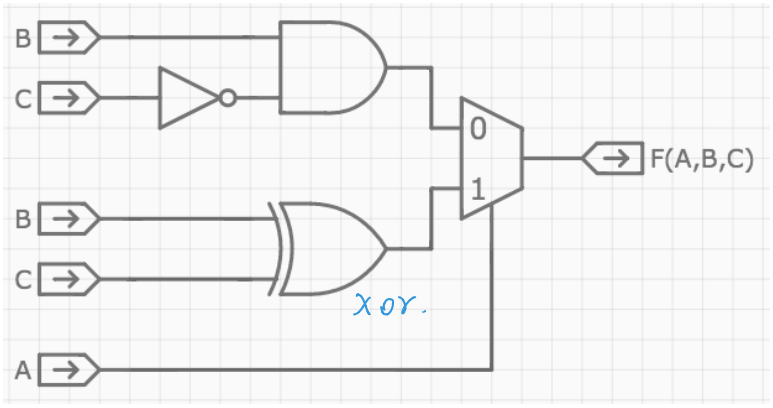




S	D_1	D_0	D_S
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Problem 8.

A 6.004 intern at Intel has designed the combinational circuit shown below. His boss can't figure out what it does and has asked for your help.



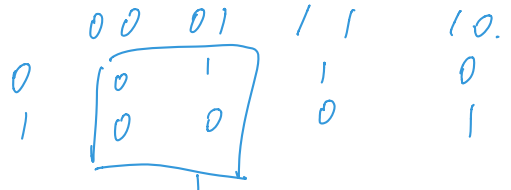
A	*B*	*C*	*F(A,B,C)*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0.

(A) Please fill in the truth table for F(A,B,C) above.

Fill in truth table above

(B) Express F(A,B,C) in minimal sum-of-products form. Hint: use a Karnaugh map!

minimal sum-of-products expression for F(A,B,C) = $\bar{C} \cdot B + C \cdot A \cdot \bar{B}$



(C) The boss isn't quite sure what it means but he knows his engineers are always impressed if he asks "is the circuit universal?" Is it? Circle YES or NO.

F(A,B,C) universal? YES … NO

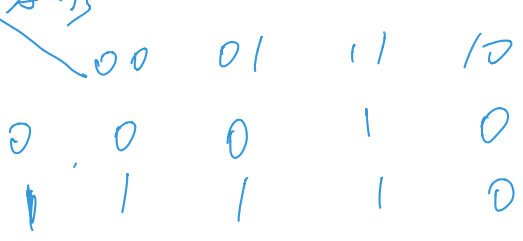
NAND

F(0, B, C).

the selector.

not lenient.

C. A-B



$D_S(\ ) = A \cdot B + C \cdot \bar{A}$

Problem 9.

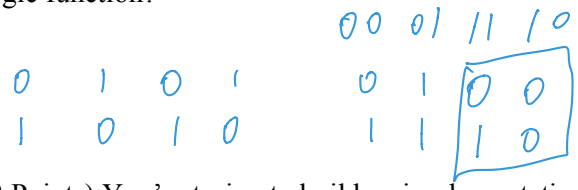
The full subtractor (FS) implements a one-column binary subtraction of two bits (X and Y) producing their difference (D), accepting a borrow-in (B_{IN}) from the previous column and producing a borrow-out (B_{OUT}) for the next column. Numerically FS computes $X-Y-B_{IN}$ and encodes the possible answers (1, 0, −1, −2) using D and B_{OUT} as shown in the truth table to the right.

X	Y	B_{IN}	D	B_{OUT}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(A) (2 Points) Give a sum-of-products expression for B_{OUT}.

Sum of products expression: B_{OUT} = $\bar{X} \cdot B_{in} + Y \cdot B_{in} + \bar{X}\,Y$

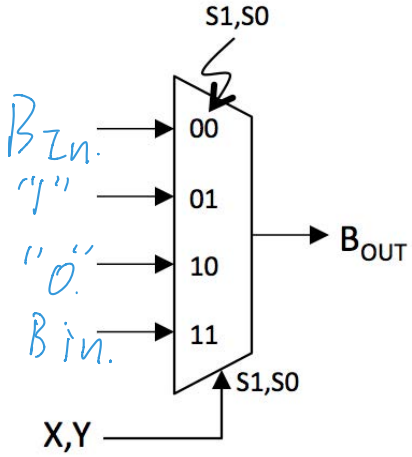
(B) (1 Point) Is the FS(X,Y,B_{IN}) circuit universal in the sense that 2-input NOR and 2-input NAND are universal? In other words, using only acyclic networks of FS circuits (perhaps with one or more of their inputs tied to "0" or "1"), can one implement any combinational logic function?



FS Universal: … YES … NO …

(C) (2 Points) You're trying to build an implementation for the B_{OUT} part of the FS circuit (see truth table above) but discover that the NITGFOC supply room only has 4-to-1 multiplexors in stock. In desperation, you call up your 6.004 TA who says "No problem! In fact, you can produce B_{OUT} with just a single 4-to-1 mux: connect X to S1 and Y to S0, then hook each of the data inputs to the appropriate choice of '0', '1' or B_{IN}." Using this hint, finish off the implementation shown below.

Show connections for data inputs using only '0', '1' or B_{IN}



Problem 10.

The 3-input Boolean function G(A,B,C) computes $\overline{A}\cdot\overline{C}+A\cdot\overline{B}+\overline{B}\cdot\overline{C}$.

(A) How many 1's are there in the output column of G's 8-row truth table?

4.

(B) Give a minimal sum-of-products expression for G.

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

(C) There's good news and bad news: the bad news is that the stockroom only has G gates. The good news is that it has as many as you need. Using only combinational circuits built from G gates, one can implement (choose the best response)

(A) only inverting functions
(B) only non-inverting functions
(C) any function (G is universal)
(D) only functions with 3 inputs or less
(E) only functions with the same truth table as G

(D) Can a sum-of-products expression involving 3 input variables with greater than 4 product terms *always* be simplified to a sum-of-products expression using fewer product terms?

Yes. Max (...) = 4.

Lec. 4 . 7.

Computation Structures

Sequential Logic Worksheet

Concept Inventory:

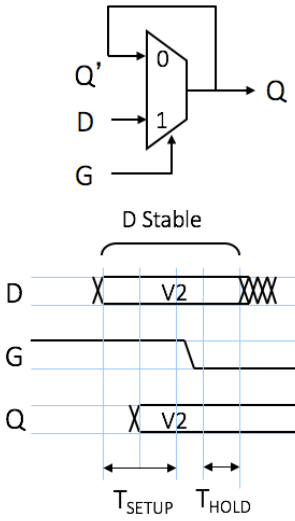
- D-latch & the Dynamic Discipline
- D-register
- Timing constraints for sequential circuits
- Set-up and hold times for sequential circuits

很多理论in timing分析.

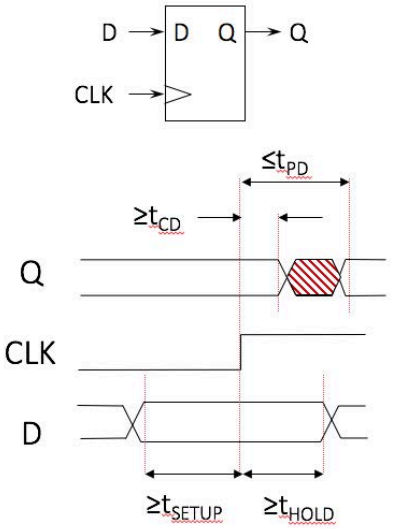
不懂。

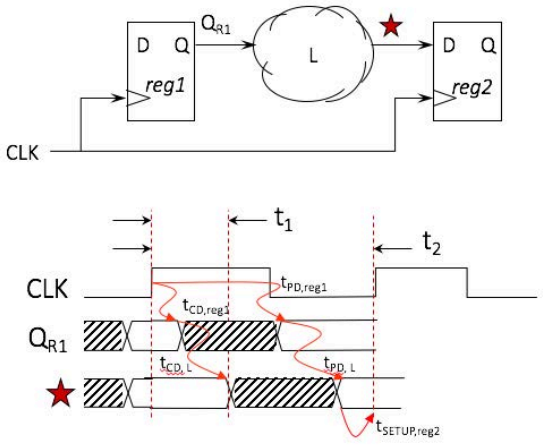
Notes:

D latch



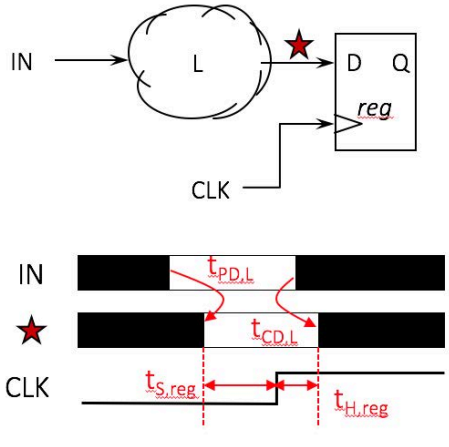
D register





$t_1 = t_{CD,reg1} + t_{CD,L} \geq t_{HOLD,reg2}$

$t_2 = t_{PD,reg1} + t_{PD,L} + t_{SETUP,reg2} \leq t_{CLK}$

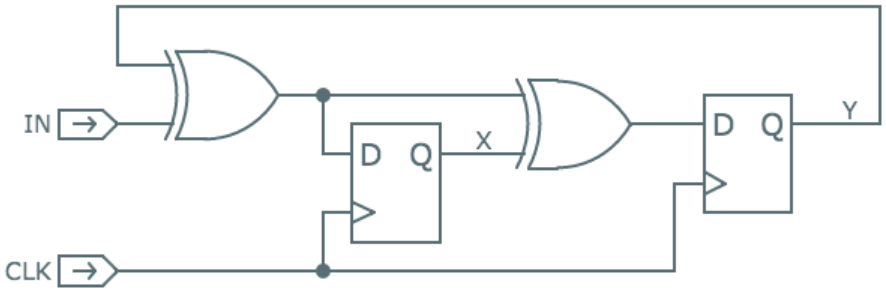


$t_{S,INPUT} = t_{PD,L} + t_{S,R}$

$t_{H,INPUT} = t_{H,R} - t_{CD,L}$

Problem 1.

Consider the following sequential logic circuit. It consists of one input IN, a 2-bit register that stores the current state, and some combinational logic that determines the state (next value to load into the register) based on the current state and the input IN.



(A) Using the timing specifications shown below for the XOR and DREG components, determine the shortest clock period, t_{CLK}, that will allow the circuit to operate correctly or write NONE if no choice for t_{CLK} will allow the circuit to operate correctly and briefly explain why.

Component	t_{CD}	t_{PD}	t_{SETUP}	t_{HOLD}
XOR2	0.15ns	2.1ns	–	–
DREG	0.1ns	1.6ns	0.4ns	0.2ns

1.6 + 0.4 + 2 × tpd. 6.2

Minimum value for t_{CLK} (ns): ~~8.2ns~~.
or explain why none exists

1.为什么 DREG.
让 tpd 和 tsetup 不是加起来?

(B) Using the same timing specifications as in (A), determine the setup and hold times for IN with respect to the rising edge of CLK.

2 × 2.1 + 0.4.

t_{SETUP} for IN with respect to CLK↑ (ns): ~~5.8~~ 4.6 ns.

t_{HOLD} for IN with respect to CLK↑ (ns): 0.05.

0.2 − 0.15.

(C) One of the engineers on the team suggests using a new, faster XOR2 gate whose t_{CD} = 0.05ns and t_{PD} = 0.7ns. Determine a new minimum value for t_{CLK} or write NONE and explain why no such value exists.

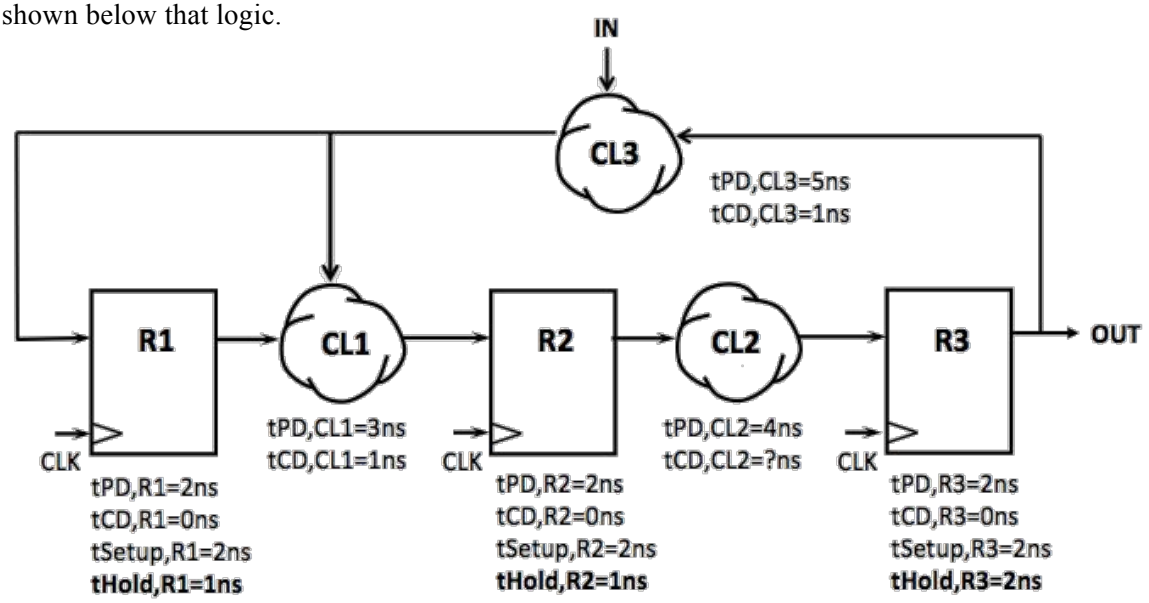
Minimum value for t_{CLK} (ns): No.
or explain why none exists

0.15 < 0.2.

sum(tcd) < t Hold..

Problem 2.

Consider the following sequential logic circuit. It consists of three D registers, three different pieces of combinational logic (CL1, CL2, and CL3), one input IN, and one output OUT. The propagation delay, contamination delay, and setup time of the registers are all the same and are specified below each register. **The hold time for the registers is NOT the same** and is specified in bold below each register. The timing specification for each combinational logic block is shown below that logic.



(A) (1 point) What is the smallest value for the t_{CD} of CL2 that will guarantee the dynamic discipline is obeyed for all the registers in the circuit?

$R_2 - CL_2 \rightarrow R_3$

$0 + ? \geq 2$

Smallest value for t_{CD} of CL2 (ns): 0ns. 2ns.

(B) (2 points) What is the smallest value for the period of CLK (i.e., t_{CLK}) that will guarantee the dynamic discipline is obeyed for all the registers in the circuit?

$R_3 - R_2 = 2+5+3+2.$

Smallest value for t_{CLK} (ns): ~~11ns~~ 12.

(C) (2 points) What are the smallest values for the setup and hold times for IN relative to the rising edge of CLK that will guarantee the dynamic discipline is obeyed for all the registers in the circuit?

Setup time for IN (ns): 10.

Hold time for IN (ns): 0

(D) (2 points) What are the propagation delay and contamination delay of the output, OUT, of this circuit relative to the rising edge of the clock?

t_{PD} for OUT (ns): ~~10.~~ 2

t_{CD} for OUT (ns): ~~0~~ 0.

use R_3.

完全搞不懂.....

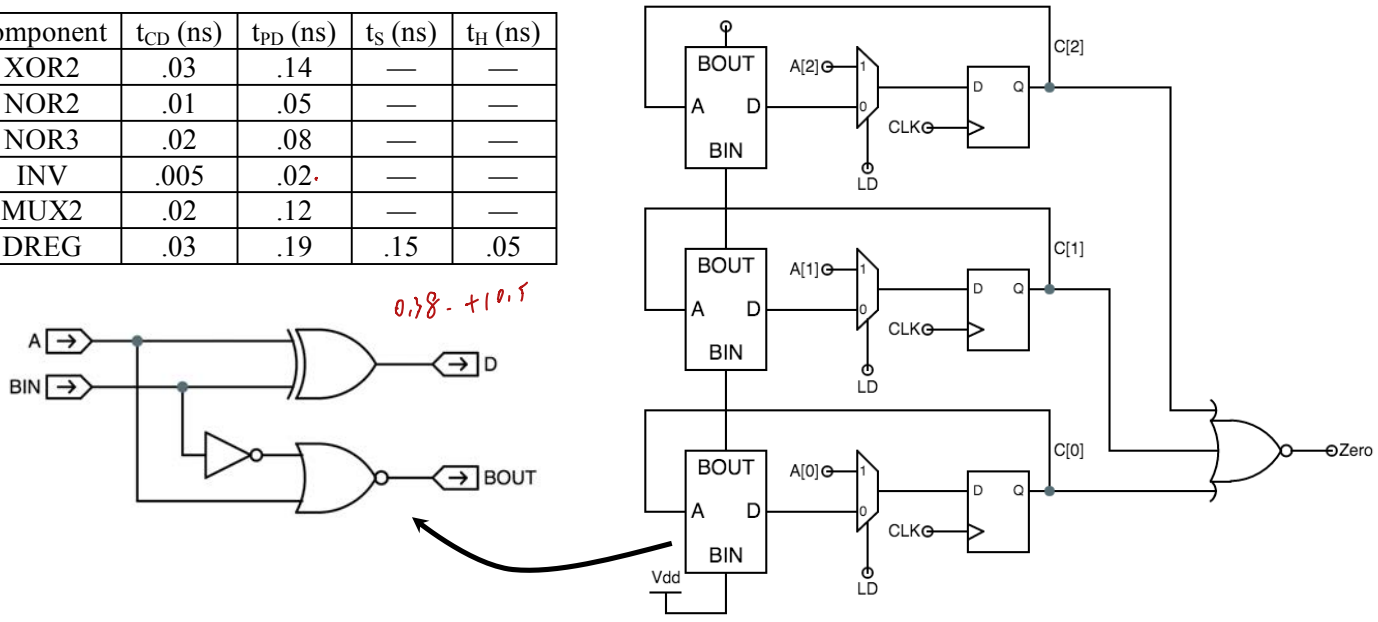
为什么不是计算.

$CL_3 - CL_1 \rightarrow R_2 \rightarrow CL_2 - R_3$ 呢?

Problem 3.

Here's a schematic for a 3-bit loadable down-counter, which uses a ripple decrementer as a building block:

component	t_{CD} (ns)	t_{PD} (ns)	t_S (ns)	t_H (ns)
XOR2	.03	.14	—	—
NOR2	.01	.05	—	—
NOR3	.02	.08	—	—
INV	.005	.02	—	—
MUX2	.02	.12	—	—
DREG	.03	.19	.15	.05



(A) Using the contamination delays (t_{CD}), propagation delays (t_{PD}), setup times (t_S), and hold times (t_H) shown in the table above, please compute the minimum value for the clock period (t_{CLK}) for which the circuit will work correctly.

minimum value for t_{CLK} (ns): _____

(B) What are the appropriate values for the setup (t_S) and hold (t_H) times for the **LD** input with respect to the rising edge of the clock?

setup time (t_S) for LD: _____

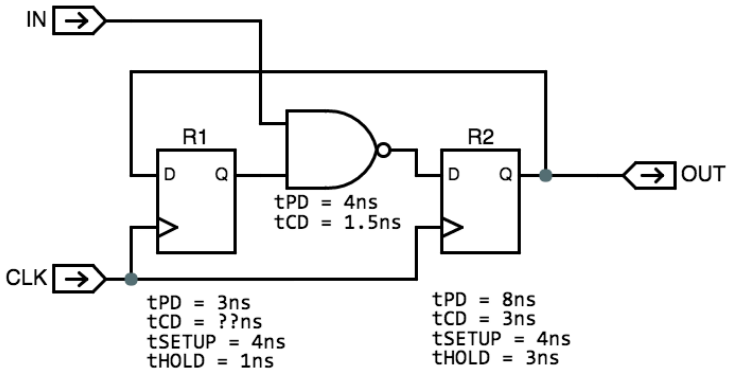
hold time (t_H) for LD: _____

(C) What is the t_{PD} for the Zero output with respect to the rising edge of CLK?

t_{PD} for Zero (ns): _____

Problem 4.

Consider the following sequential logic circuit. The timing specifications are shown below each component. Note that the two registers do NOT have the same specifications.



(A) What are the smallest values for the setup and hold times for IN relative to the rising edge of CLK that will guarantee the dynamic discipline is obeyed for all the registers in the circuit?

Setup time for IN (ns): 8ns.

Hold time for IN (ns): 1.5.

(B) What is the smallest value for the period of CLK (i.e., tCLK) that will guarantee the dynamic discipline is obeyed for all the registers in the circuit?

12.

Smallest value for tCLK (ns): ~~16ns~~

为什么要.
寄存器之间呢?

$\begin{cases} t_{R_2} - R_1 = 8 + 4 = 12 \\ t_{R_1} - R_2 = 3 + 4 + 4 = 11. \end{cases}$

(C) What is the smallest for the tCD of R1 that will guarantee the dynamic discipline is obeyed for all the registers in the circuit?

Smallest value for tCD of R1 (ns): 1.5 ?

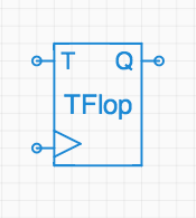
(D) Suppose two of these sequential circuits were connected in series, with the OUT signal of the first circuit connected to the IN signal of the second circuit. The same CLK signal is used for both circuits. Now what is the smallest value for the period of CLK (i.e., tCLK) that will guarantee the dynamic discipline is obeyed for all the registers in the circuit?

Smallest value for tCLK (ns): 16

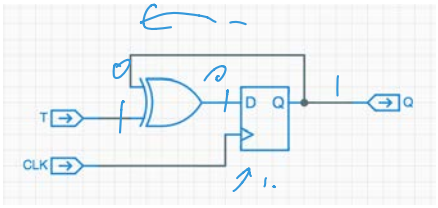
longest path

$R_2 \rightarrow In \rightarrow NAND \rightarrow R_2$

Problem 5.



It is often useful to make clocked devices that count in binary, and a simple building block for such binary counters is the toggle flipflop whose symbol is shown on the right. It is a clocked device, hence the clock input indicated by the triangle on its lower-left edge. The other input, T (for *toggle*), may be set to one to cause the TFlop to flip its state (the *Q* output) from 0 to 1 or vice versa on the next active (positive) clock edge. If T is zero at an active clock edge, the state of the TFlop remains unchanged. We assume that the initial state of each TFlop at power-up is *Q=0*; more sophisticated versions might feature a *Reset* input to force a *Q=0* state.



A TFlop may be implemented using a D flipflop like the ones developed in lecture together with an XOR2 gate, as shown to the left.

As is our convention for clocked devices, we would like to specify timing specs for the TFlop as t_{CD}, t_{PD}, t_{SETUP}, and t_{HOLD}, all measured relative to the active (positive) clock edge.

(A) The timing specifications for the components are shown in the table below. Give appropriate values for the timing specifications of the TFlop implementation shown above.

Component	t_{CD}	t_{PD}	t_{SETUP}	t_{HOLD}
XOR2	40ps	400ps	–	–
DREG	100ps	300ps	80ps	40ps

t_{CD}: 100 **ps**

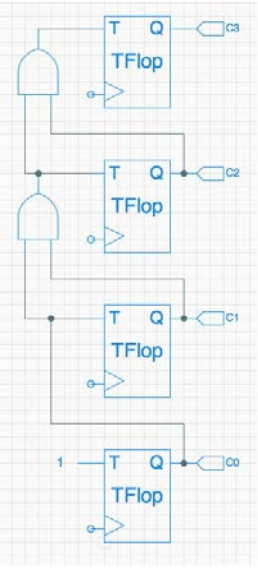
t_{PD}: ~~200~~ 300 **ps**

t_{SETUP}: 80. **ps**

t_{HOLD}: 0. **ps**

(B) Suppose we connect the T input of a single TFlop to 1 (i.e., V_{DD}) and try to clock it at its maximum rate. What is the minimum clock period we can use and expect the TFlop to perform properly?

***Minimum* clock period for correct operation:** 780 **ps**



We next consider the use of four TFlops to make a 4-bit ripple-carry counter as shown to the left. Assume that the TFlops share a common clock input (not shown) with an appropriate period, and that all TFlops have an initial Q=0 state.

(C) Suppose we run this circuit for a large number, N, of clock cycles. For approximately how many of the N active clock edges would you expect the T input to the topmost TFlop to be 1?

***Topmost* T=1 occurrences in N cycles:** N/2

(D) If the AND2 gates have t_{PD} =200ps and t_{CD} =40ps, what is the minimum clock period we can use for the 4-bit counter?

***Minimum* clock period for correct operation:** 1560 **ps**

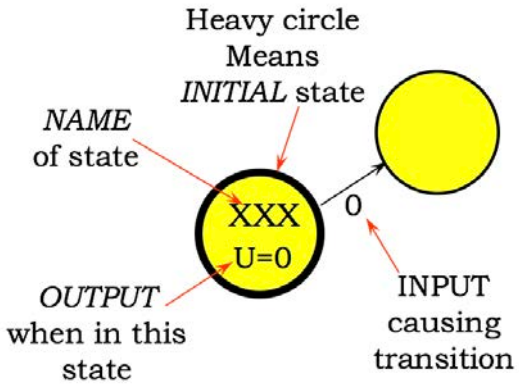
Computation Structures

Finite State Machines Worksheet

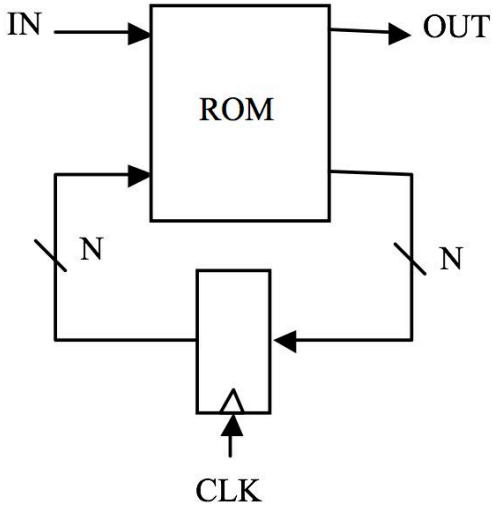
Concept Inventory:

- State transition diagrams & FSM truth tables
- Register & ROM implementation
- Equivalent FSMs; equivalent state reduction
- Metastability: causes and cures

Notes:



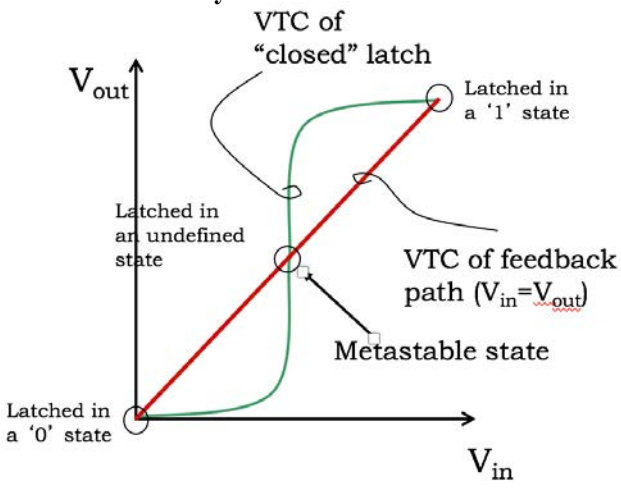
Arcs leaving a state must be (1) mutually exclusive, and (2) collectively exhaustive.



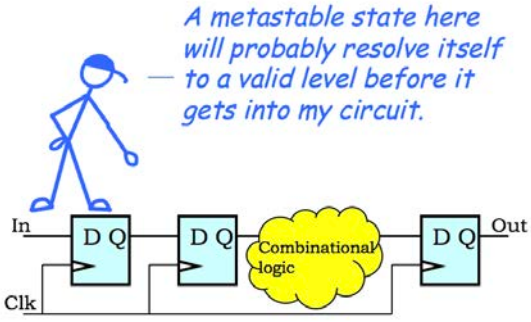
FSMs are EQUIVALENT if and only if every inputs sequence yields identical output sequences.

Two states are equivalent if
1. both states have identical outputs, AND
2. every input transitions to equivalent states.

Metastability:

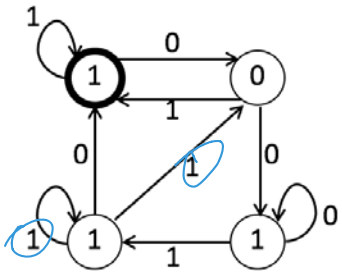


Quarantine time reduces p(metastable)



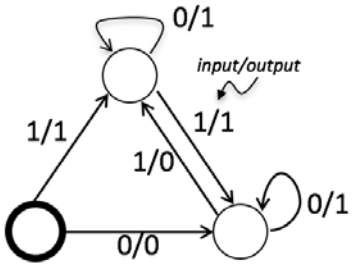
Problem 1.

(A) For each of the following FSMs please indicate if they are or are not well formed. Note that the state names have been omitted for clarity; you may assume the state names are unique.

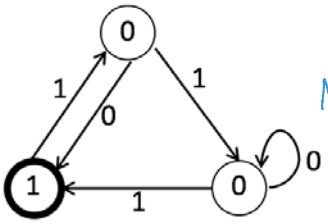


(A)

Not mutually exclusive.



(B)



(C)

Not exhaustive.

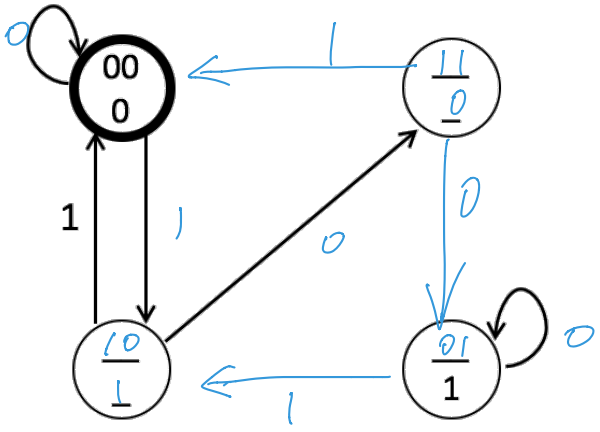
FSM A (circle one): Well Formed / Not Well Formed

FSM B (circle one): Well Formed / Not Well Formed

FSM C (circle one): Well Formed / Not Well Formed

(B) Given the partially completed truth table and FSM diagram below. **Complete all the missing entries** in the truth table and the FSM diagram. The FSM is a Moore machine, i.e., the Out signal is determined only by the current state. In each state circle, the top entry is S_1S_0 and the bottom entry is the value of Out. Make sure that you have labeled all missing states, inputs, and outputs, and that you have added and labeled any missing transitions in the FSM.

S1	S0	In	S1'	S0'	Out
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	1	1	1
1	0	1	0	0	1
1	1	0	0	1	0
1	1	1	0	0	0



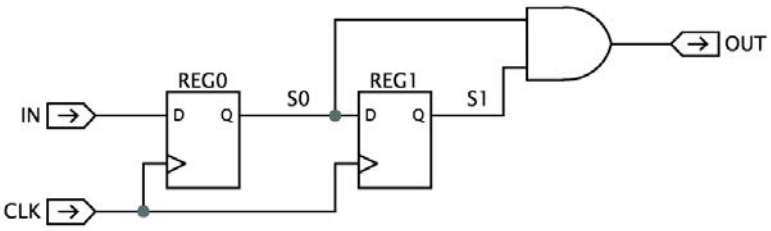
(C) If this FSM is implemented using a 2-bit state register and a ROM, what size ROM would be needed? Please specify the number of locations (entries) of the ROM, and the width of each entry.

Number of locations in ROM: 8

Width of each ROM entry (bits): 3

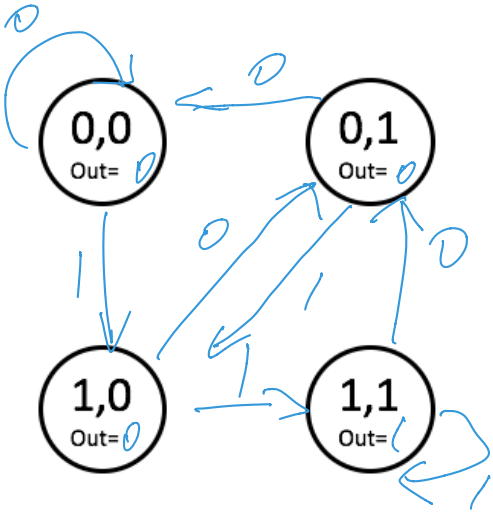
Problem 2.

Consider the sequential logic circuit to the right, which implements an FSM with a single data input IN and single data output OUT. Assume that all signal transitions are timed so that the dynamic discipline is satisfied at each register.



Please describe the operation of the FSM by filling in both the state transition diagram and the truth table shown below. The two-digit state names in the state transition diagram are S0,S1, the logic values present at the outputs of REG0 and REG1 after the rising edge of the clock. In the truth table, S0' and S1' are the values that will loaded into REG0 and REG1 at the next rising clock edge.

Fill in state transition diagram and truth table



S0	S1	IN	S0'	S1'	OUT
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

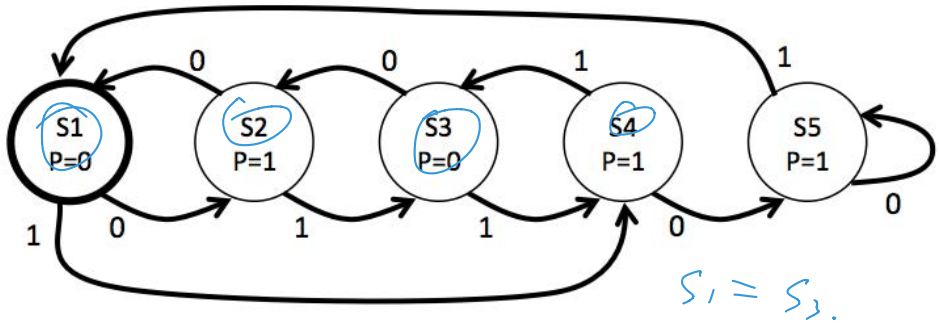
Problem 3.

A "Thingee" is a clocked device built out of 3 interconnected components, each of which is known to be a 4-state FSM. What bound, if any, can you put on the number of states of a Thingee?

Max # of states, or "Can't Tell": ≤ 64

Problem 4.

Consider the 1-input, 1-output finite state machine with the state transition diagram shown below. Note that the single output P only depends on the current state of the FSM.



$S_1 \equiv S_3$. $S_4 \equiv S_5$.

(A) (1 Point) The FSM has been processing inputs for a while and we would like to determine its current state. After entering three additional inputs "000", we observe that we have reached a state where P=0. Please circle the possible values for the state *before* the additional three inputs were entered.

Possible values for state before: **S1** **S2** **S3** **S4** **S5**

(B) (2 Points) Assume that the states are represented by the 3-bit binary values given on the left below. Please fill in the appropriate entries for the partial truth table shown on the right where S is the current state, I is the input value, S' is the next state, P is the output value

State	*Encoding*
S1	001
S2	010
S3	011
S4	100
S5	101

Fill in partial truth table

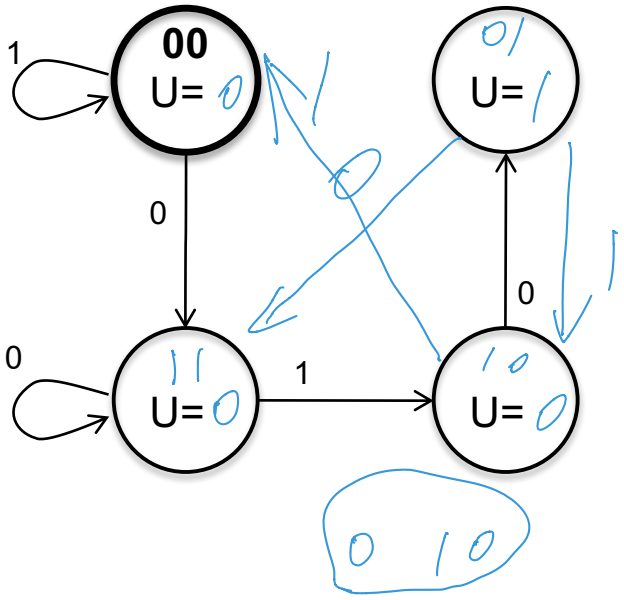
	S	*I*	*S'*	*P*
S_3	011	0	010	0
S_3	011	1	100	0
S_4	100	0	101	1
S_4	100	1	011	1

(C) Please identify which, if any, states are equivalent. For example, if states S1, S2, and S4 are equivalent, please write "(S1,S2,S4)". You may need multiple parenthesized lists if more than one set of states is equivalent.

Equivalent states: (S_1, S_3) (S_4, S_5)

Problem 5.

Perfectly Perplexing Padlocks makes an entry-level electronic lock, the P3b, built from an FSM with two bits of state. The P3b has two buttons ("0" and "1") that when pressed cause the FSM controlling the lock to advance to a new state. In addition to advancing the FSM, each button press is encoded on the B signal (B=0 for button "0", B=1 for button "1"). The padlock unlocks when the FSM sets the UNLOCK output signal to 1, which it does **whenever—and only whenever—the last 3 button presses correspond to the 3-digit combination**. The combination is unique, and will open the lock independently of the starting state. Unfortunately the design notes for the P3b are incomplete.



S_1	S_0	B	S'_1	S'_0	U
0	0	0	1	1	0
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	0	0	0
1	1	0	1	1	0
1	1	1	1	0	0

(A) (1 Point) What is the 3-bit combination for the lock?

lock combination: 0 1 0

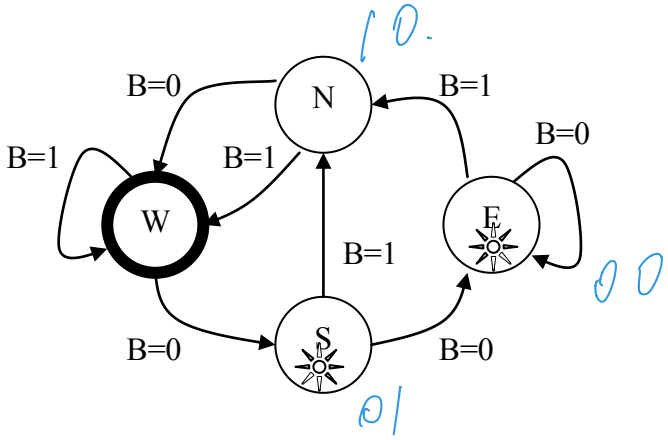
(B) (5 Points) Using the specification and clues from the partially completed diagrams above fill in the information that is missing from the state transition diagram and its accompanying truth table. When done:

- each state in the transition diagram should be assigned a 2-bit state name S_1S_0 (note that in this design the state name is *not* derived from the combination that opens the lock),
- the arcs leaving each state should be mutually exclusive and collectively exhaustive,
- the value for **U** should be specified for each state, and
- the truth table should be completed.

(complete above transition diagram and table)

Problem 6.

Below is a state transition diagram for a 4-state FSM with a single binary input B. The FSM has single output – a light that is "on" when the FSM is in states "E" or "S". The starting state, "W", is marked by the heavy circle.



(A) (1 Point) Does this FSM have a set or sets of equivalent states that can be merged to yield an equivalent FSM with fewer states?

List set(s) of states that can be merged or write NONE: (S, E)

(B) (5 Points) Please fill in as many entries as possible in the following truth table for the FSM. The *light* output is a function of the current state and should be 1 when the light is "on" and 0 when it's "off."

S1	S0	B	S1'	S0'	light
0	0	0	0	0	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	1	1	0
1	0	1	1	1	0
1	1	0	0	1	0
1	1	1	1	1	0

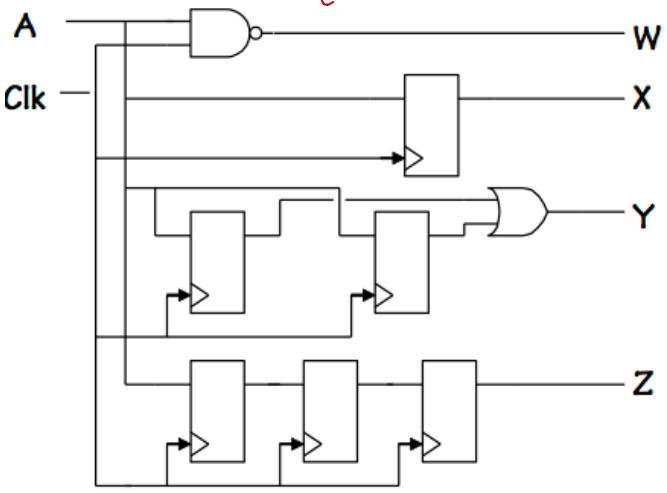
Problem 7.

The following circuit has two inputs (A, CLK) and four outputs (W, X, Y Z). The CLK signal is square wave with a period t_{CLK}=1us. The A signal makes a single 0→1 transition but the timing of the transition is close to (within a few ns of) the active CLK edge, ignoring dynamic discipline. All the devices are lenient and have the same propagation delay t_{PD}=10ns.

In a test involving a large number of trials, the **Z** output has been examined 100ns after an active CLK edge (and when *both CLK and A have been stable for many propagation delays*); at this time, **Z** was found to be invalid P times. In the same test, what would you expect to observe at the other outputs 100ns after the CLK edge? For each output, circle one of

LESS RELIABLE if you would expect the output to be **invalid** appreciably **more** than P times;
EQUALLY RELIABLE if you would expect the output to be **invalid** about P times; or
MORE RELIABLE if you would expect the output to be **invalid** appreciably **less** than P

combinational logic never goes to metastable.



W reliability: **LESS** **EQUAL** **MORE**

X reliability: **LESS** **EQUAL** **MORE**

Y reliability: **LESS** **EQUAL** **MORE**

Z was found to be invalid P times……

Computation Structures

Pipelined Circuits Worksheet

Concept Inventory:

- Latency & throughput
- Pipelined circuits & conventions
- Pipeline diagrams
- Pipelining methodology: contours
- Pipelined components; interleaving

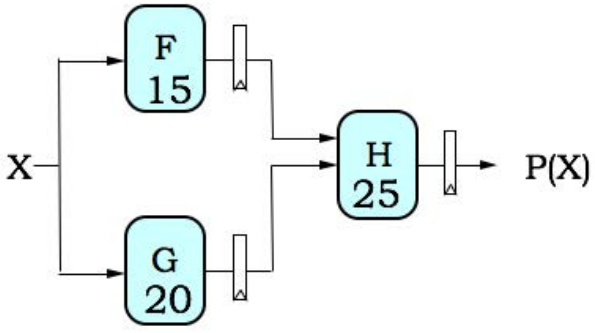
Notes:

Latency: the delay from when an input is established until the output associated with the input becomes valid.

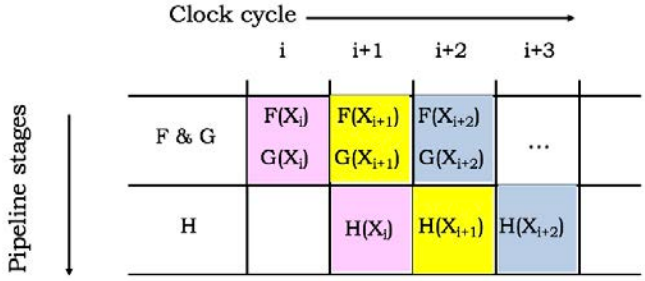
- Combinational circuits: $L = t_{PD}$
- K-pipeline: $L = K*t_{CLK}$

Throughput: the rate at which inputs or outputs are processed.

- Combinational circuits: $T = 1/L$
- K-pipeline: $T = 1/t_{CLK}$

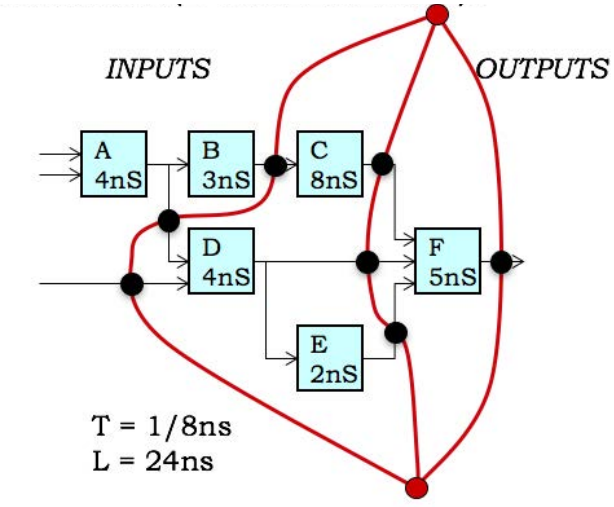


Unpipelined:
L = 45ns, T = 1/L = 1/(45ns)
2-stage pipeline [t_{CLK}=25ns]:
L = 2*25 = 50 ns, T = 1/(25ns)



Pipelining methodology:

- Form 1-pipeline by adding registers to all outputs
- To add a pipeline stage, draw contour across all paths from inputs to outputs such that it doesn't cross other contours and all input-output paths cross the contour in the same direction. This ensures the pipeline is well-formed (same # of registers on all input-output paths). A K-pipeline has K registers on all input-output paths.
- Contours must take into account pipelined or interleaved components. An N-way interleaved component behaves like N-pipeline.



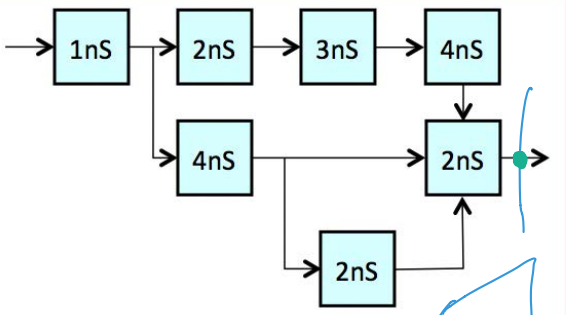
Problem 1.

A simple combinational circuit is to be pipelined for **maximum throughput using a minimal number of registers**. For each of the questions below, please create a valid K-stage pipeline. **Show your pipelining contours** and place large black circles (●) on the signal arrows to **indicate the placement of ideal pipeline registers** (tPD=0, tSETUP=0). Give the latency and throughput for each design. Remember that our convention is to place a pipeline register on each output.

(A) (1 point). Show the maximum-throughput 1-stage pipeline.

Latency (ns): 12

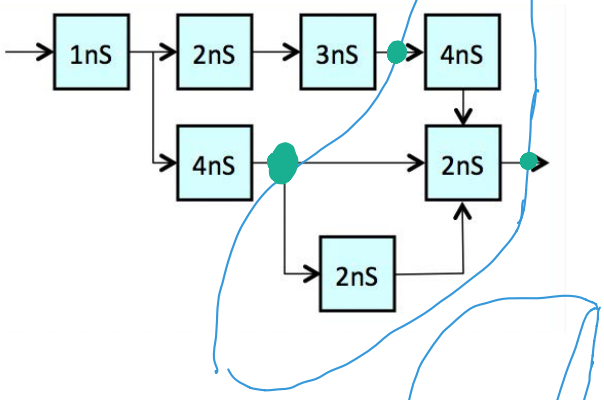
Throughput (ns-1): 1/12



(B) (2 points). Show the maximum-throughput 2-stage pipeline using a minimal number of registers.

Latency (ns): 12

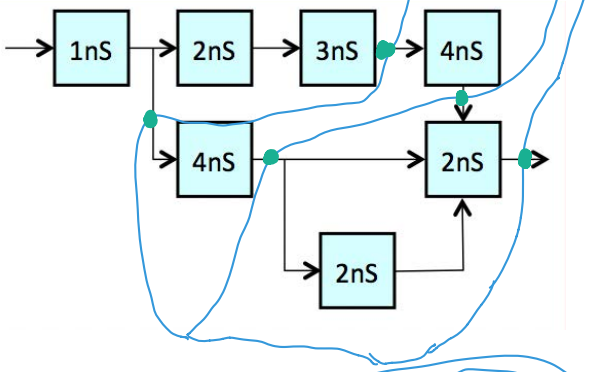
Throughput (ns-1): 1/6



(C) (2 points). Show the maximum-throughput 3-stage pipeline using a minimal number of registers.

Latency (ns): 18

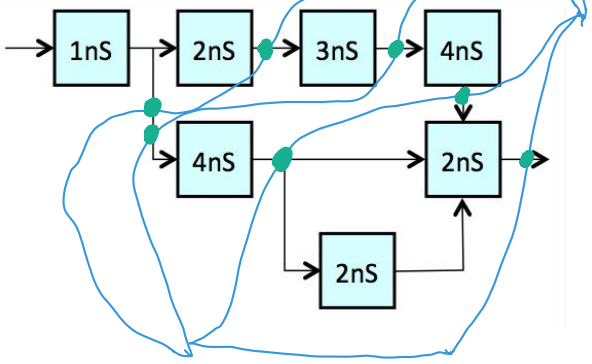
Throughput (ns-1): 1/6



(D) (2 points). Show the maximum-throughput 4-stage pipeline using a minimal number of registers.

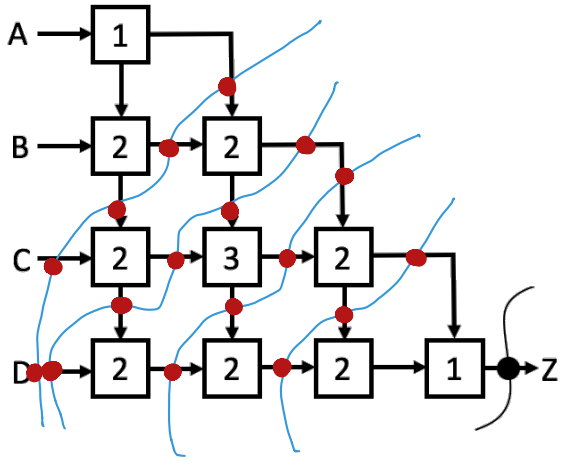
Latency (ns): 16

Throughput (ns-1): 1/4



Problem 2.

The following 1-stage pipelined circuit computes Z from the four inputs A, B, C, and D. Each component is annotated with its propagation delay in ns.



(A) Please pipeline the circuit above for maximum throughput with the minimum possible latency using ideal pipeline registers ($t_{PD} = 0$, $t_{SETUP} = 0$). Show the location of pipeline registers in the diagram above using filled-in circles, like the one shown on the Z output. Please give the latency and throughput of the resulting pipelined circuit.

15

5. ~~6~~ x3 = ~~18~~

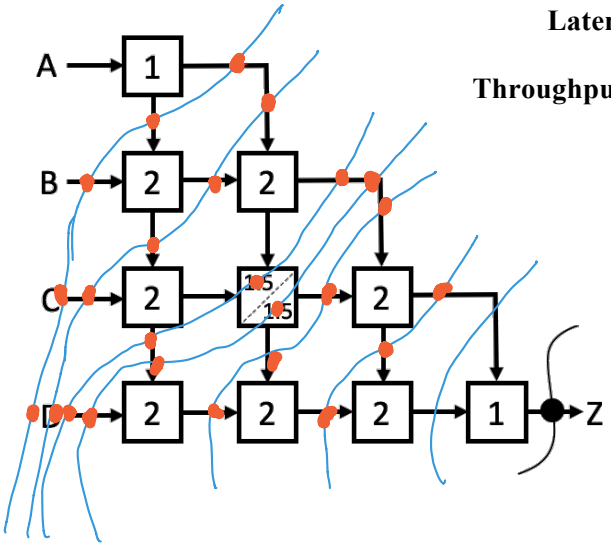
Latency (ns)? _____

Throughput (1/ns)? 1/3

(B) Now suppose the "3" component is replaced by a two-way interleaved component with a minimum t_{CLK} of 1.5ns. Recall that a two-way interleaved component behaves like a 2-stage pipelined component. Again, please pipeline the circuit below for maximum throughput with the minimum possible latency using ideal pipeline registers. Show the location of pipeline registers in the diagram below using filled-in circles, like the one shown on the Z output. Please give the latency and throughput of the resulting pipelined circuit.

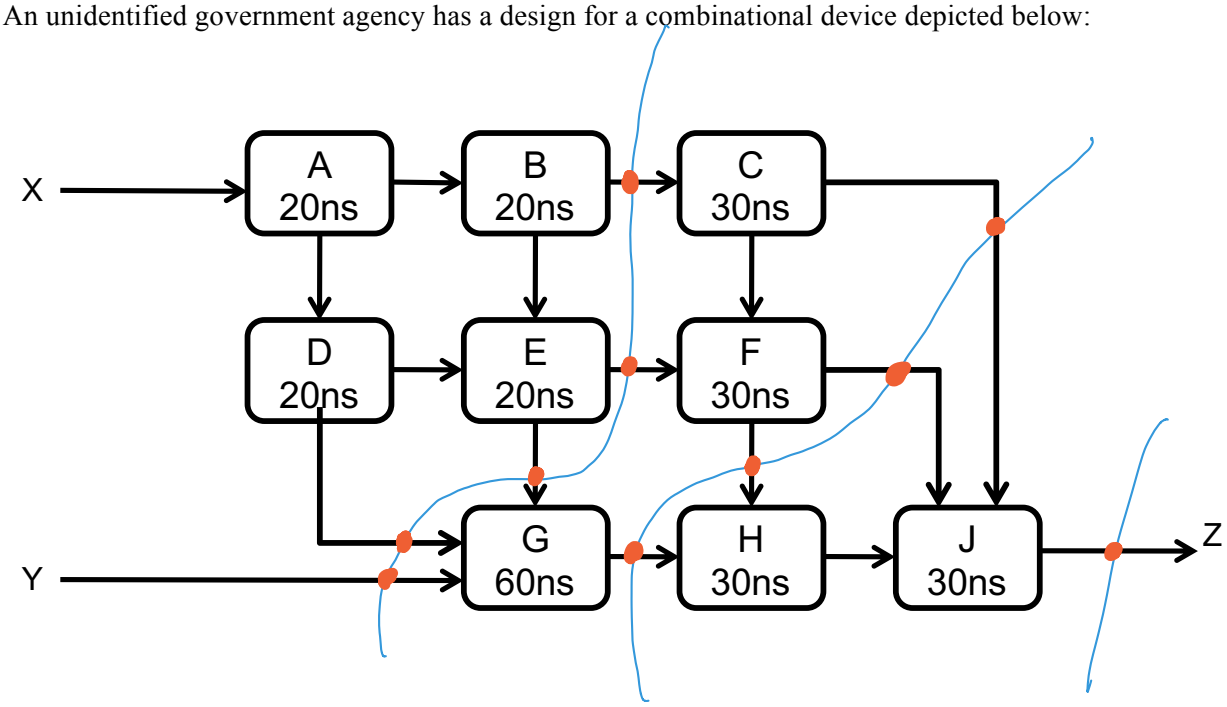
Latency (ns)? 2x8 = 16 ns

Throughput (1/ns)? 1/2



Problem 3.

An unidentified government agency has a design for a combinational device depicted below:



Although you don't know the function of each of the component modules, they are each combinational and marked with their respective propagation delays. You have been hired to analyze and improve the performance of this device.

(A) (1 Point) What are the throughput and latency for the unpipelined combinational device?

Latency: 180 **ns; Throughput:** 1/180 **ns^{-1}**

(B) (4 Points) Show how to pipeline the above circuit for maximum throughput, by marking locations in the diagram where registers are to be inserted. **Use a minimum number of registers, but be sure to include one on the output**. Assume that the registers have 0 t_{PD} and t_{SETUP}.

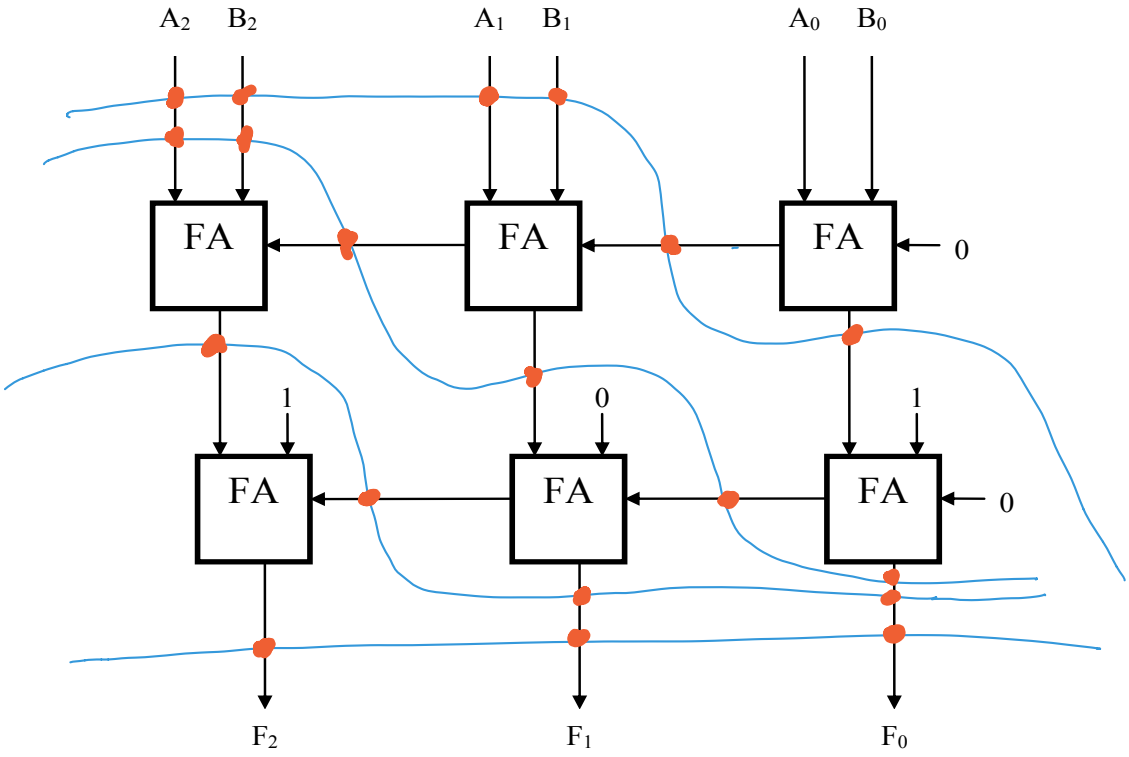
(mark register locations in diagram above)

(C) (1 Point) What are the latency and throughput of your pipelined circuit?

Latency: 180 **ns; Throughput:** 1/60 **ns^{-1}**

Problem 4.

The following circuit uses six full adder modules (as you've seen in lecture and lab) arranged in a combinational circuit that computes a 3-bit value **F=A+B+5** for 3-bit inputs **A** and **B**:



The full adders have a t_{PD} of 6ns.

(A) Give the latency and throughput of the combinational circuit.

Latency: 24 **ns; Throughput:** 1/24.

(B) Indicate, on the above diagram, appropriate locations to place ideal (zero-delay) registers to pipeline the circuit for *maximum throughput* using a *minimum number of registers*. Be sure to include a register on each output.

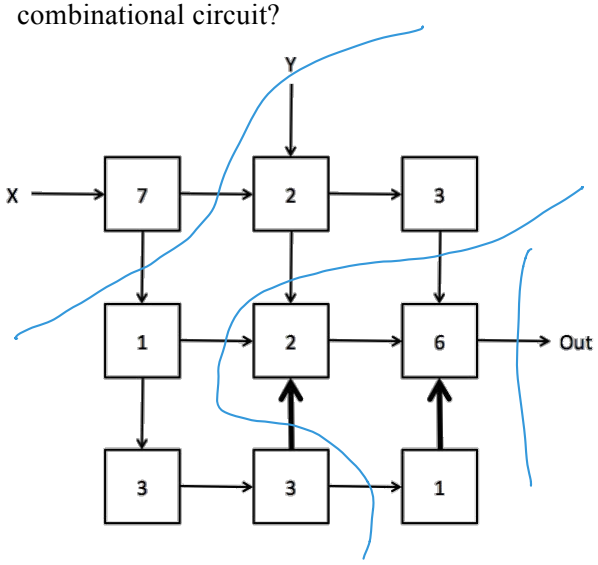
(mark circuit above)

(C) Give the latency and throughput of your pipelined circuit.

Latency: 24 **ns; Throughput:** 1/6

Problem 5.

(A) You are provided with the circuit shown below. Each box represents some combinational logic. The number in each box is the t_{PD} of that combinational logic. The circuit has two inputs, X and Y, and one output Out. Pay close attention to the direction of the arrows especially the arrows shown in **bold**. What is the latency and throughput of this combinational circuit?



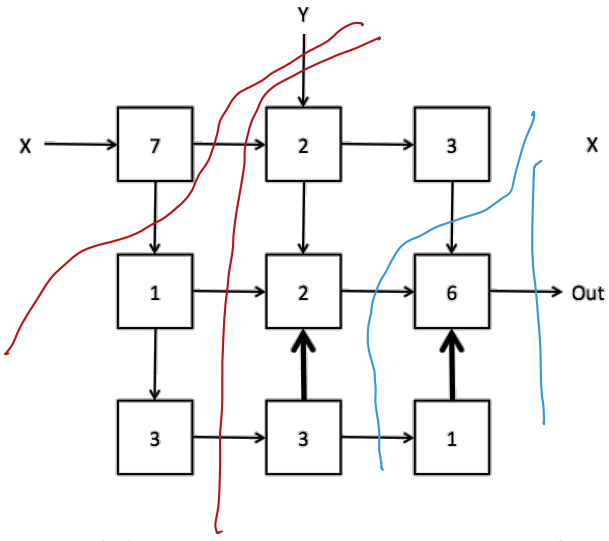
Latency (ns): 22

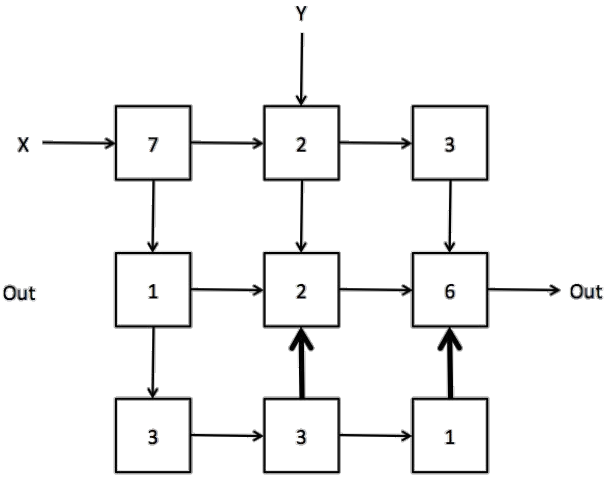
Throughput (1/ns): 1/22

(B) Draw contours through the circuit above to produce a valid pipelined circuit **whose t_{CLK} = 9ns with minimum latency**. Extra copies of the diagram are included below. Please use a large dot to **indicate the location of each pipeline register**. Assume that you have ideal pipeline registers (t_{PD}=t_{CD}=t_{Setup}=t_{Hold}=0 ns). Pay close attention to the direction of each arrow to ensure that you produce a valid pipeline. What is the latency and throughput of this pipelined circuit?

Latency (ns): 27

Throughput (1/ns): 1/9

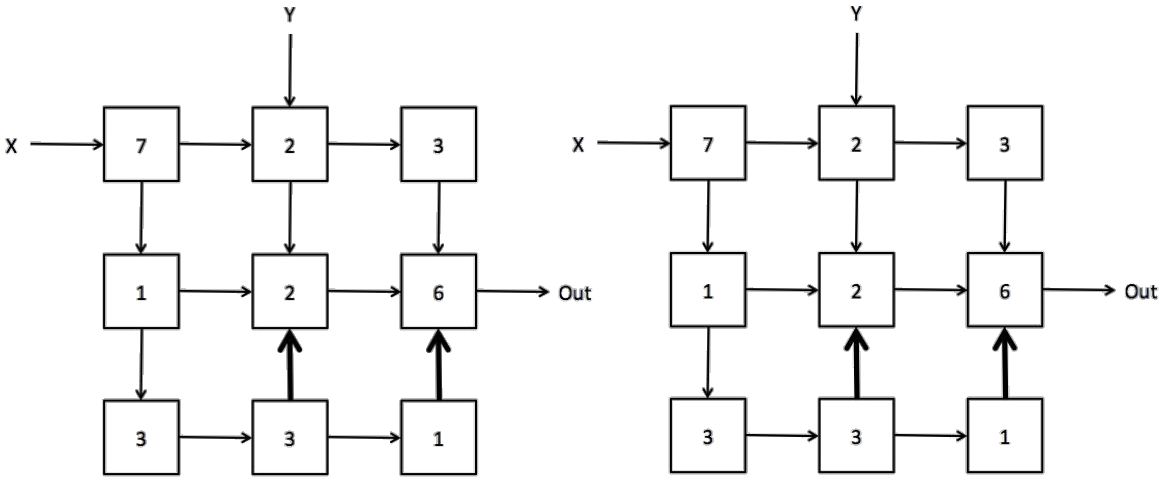




(C) You are now asked to consider the performance of this circuit using different clock periods while achieving the minimum latency. For each suggested t_{CLK}, specify whether or not you can create a **valid** pipelined circuit using that clock period. If you can, then provide the latency and throughput of the resulting circuit and specify the number of registers at each input. If it results in an invalid pipeline, enter NA for the rest of the row.

Extra copies of the circuit diagram are provided below.

t_{CLK}	Valid/Invalid	Latency (ns)	Throughput (1/ns)	Pipeline registers at input X	Pipeline registers at input Y
6 ns	NA.				
7 ns	Valid.	28	1/7	0	2.



Problem 6.

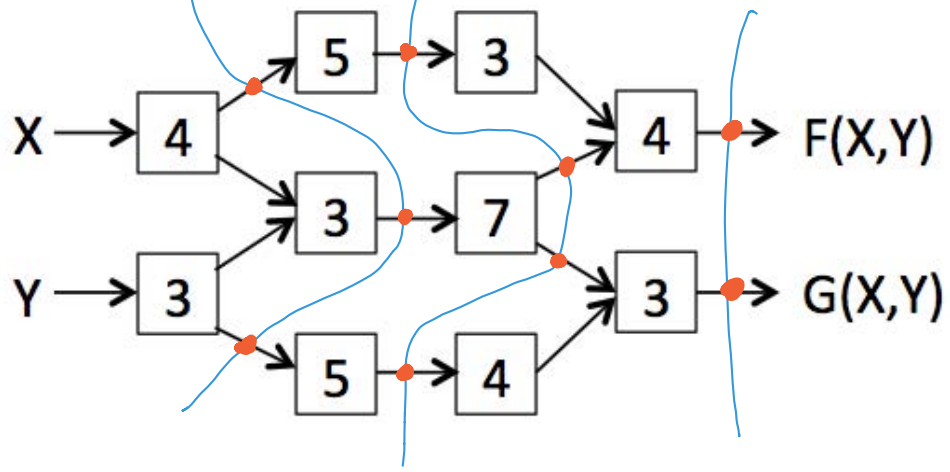
A complex combinational circuit is constructed entirely from 2-input NAND gates having a propagation delay of 1 ns. If this circuit is pipelined for maximal throughput by adding (non-ideal) registers whose setup time and propagation delay are each 1 ns, what is the throughput of the resulting pipeline? Enter a number, a formula, or "CAN'T TELL".

Throughput (ns^{-1}): 1/3.

3.

Problem 7.

The following combinational circuit computes F(X,Y) and G(X,Y) from inputs X and Y. The t_{PD} (in ns) of each individual component is shown inside its box.

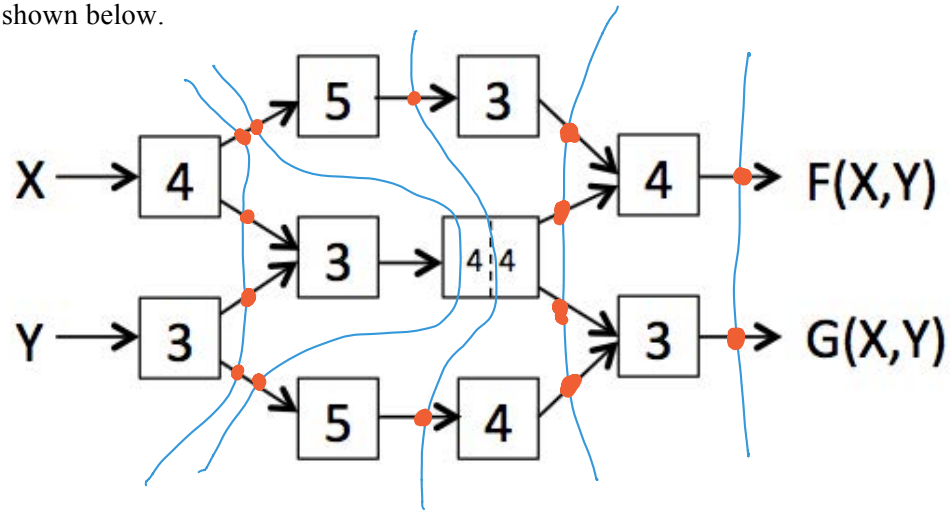


(A) Using ideal zero-delay registers, mark the location of the minimal number of registers necessary to achieve maximum throughput. Give the latency and throughput of your pipelined circuit.

mark diagram above

Latency: 2 **(ns); Throughput:** 1/2 **(1/ns)**

Rummaging through the stockroom you find a pipelined component with two pipeline stages that can replace the "7" module. The minimum t_{CLK} for the new component is 4ns. The updated circuit is shown below.



(B) Using ideal zero-delay registers, mark the location of the minimal number of registers necessary to achieve maximum throughput. Give the latency and throughput of your pipelined circuit.

mark diagram above

Latency: 25 **(ns); Throughput:** 1/5 **(1/ns)**

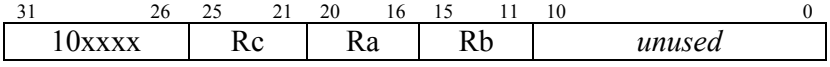
Computation Structures

Instruction Set Architecture Worksheet

Summary of β Instruction Formats

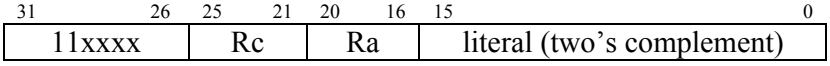
Operate Class:

Register	Symbol	Usage
R31	R31	Always zero
R30	XP	Exception pointer
R29	SP	Stack pointer
R28	LP	Linkage pointer
R27	BP	Base of frame pointer



OP(Ra,Rb,Rc): Reg[Rc] ← Reg[Ra] op Reg[Rb]

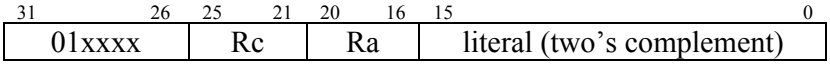
Opcodes: **ADD** (plus), **SUB** (minus), **MUL** (multiply), **DIV** (divided by)
AND (bitwise and), **OR** (bitwise or), **XOR** (bitwise exclusive or), **XNOR** (bitwise exclusive nor),
CMPEQ (equal), **CMPLT** (less than), **CMPLE** (less than or equal) [result = 1 if true, 0 if false]
SHL (left shift), **SHR** (right shift w/o sign extension), **SRA** (right shift w/ sign extension)



OPC(Ra,literal,Rc): Reg[Rc] ← Reg[Ra] op SEXT(literal)

Opcodes: **ADDC** (plus), **SUBC** (minus), **MULC** (multiply), **DIVC** (divided by)
ANDC (bitwise and), **ORC** (bitwise or), **XORC** (bitwise exclusive or), **XNORC** (bitwise exclusive nor)
CMPEQC (equal), **CMPLTC** (less than), **CMPLEC** (less than or equal) [result = 1 if true, 0 if false]
SHLC (left shift), **SHRC** (right shift w/o sign extension), **SRAC** (right shift w/ sign extension)

Other:



LD(Ra,literal,Rc): Reg[Rc] ← Mem[Reg[Ra] + SEXT(literal)]
ST(Rc,literal,Ra): Mem[Reg[Ra] + SEXT(literal)] ← Reg[Rc]
JMP(Ra,Rc): Reg[Rc] ← PC + 4; PC ← Reg[Ra]
BEQ/**BF**(Ra,label,Rc): Reg[Rc] ← PC + 4; if Reg[Ra] = 0 then PC ← PC + 4 + 4*SEXT(literal)
BNE/**BT**(Ra,label,Rc): Reg[Rc] ← PC + 4; if Reg[Ra] ≠ 0 then PC ← PC + 4 + 4*SEXT(literal)
LDR(label,Rc): Reg[Rc] ← Mem[PC + 4 + 4*SEXT(literal)]

Opcode Table: (*optional opcodes)

2:0 *5:3*	*000*	*001*	*010*	*011*	*100*	*101*	*110*	*111*
000								
001								
010								
011	**LD**	**ST**		**JMP**	**BEQ**	**BNE**		**LDR**
100	**ADD**	**SUB**	**MUL***	**DIV***	**CMPEQ**	**CMPLT**	**CMPLE**	
101	**AND**	**OR**	**XOR**	**XNOR**	**SHL**	**SHR**	**SRA**	
110	**ADDC**	**SUBC**	**MULC***	**DIVC***	**CMPEQC**	**CMPLTC**	**CMPLEC**	
111	**ANDC**	**ORC**	**XORC**	**XNORC**	**SHLC**	**SHRC**	**SRAC**	

Problem 1.

An unnamed associate of yours has broken into the computer (a Beta of course!) that 6.004 uses for course administration. He has managed to grab the contents of the memory locations he believes holds the Beta code responsible for checking access passwords and would like you to help discover how the password code works. The memory contents are shown in the table below:

Addr	Contents	Opcode	Rc	Ra	Rb	Assembly
0x100	0xC05F0008	110000	00010	11111	_____	_____
0x104	0xC03F0000	110000	00001	11111	_____	_____
0x108	0xE060000F	111000	00011	00000	_____	_____
0x10C	0xF0210004	111100	00001	00001	_____	_____
0x110	0xA4230800	101001	00001	00011	_____	_____
0x114	0xF4000004	111101	00000	00000	_____	_____
0x118	0xC4420001	110001	00010	00010	_____	_____
0x11C	0x73E20002	011100	11111	00010	_____	_____
0x120	0x73FFFFF9	011100	11111	11111	_____	_____
0x124	0xA4230800	101001	00001	00011	_____	_____
0x128	0x605F0124	011000	00010	11111	_____	_____
0x12C	0x90211000	100100	00001	00001	_____	_____

Further investigation reveals that the password is just a 32-bit integer which is in R0 when the code above is executed and that the system will grant access if R1 = 1 after the code has been executed. What "passnumber" will gain entry to the system?

Problem 2.

SHLC (R0, 3, R1)

(A) What assembly instruction could a compiler use to implement y = x * 8 on the Beta assuming that MUL and MULC are not available? Assume x is in R0 and y is in R1.

ADD(R0, R0, R0)

Equivalent assembly instruction: ADD(R1, R1, R1) or ADD(R1, A, R1).

(B) Assume that the registers are initialized to: R0=8, R1=10, R2=12, R3=0x1234, R4=24 before execution of each of the following assembly instructions. For each instruction, provide the value of the specified register or memory location. **If your answers are in hexadecimal, make sure to prepend them with the prefix 0x.**

8

1. SHL(R3, R4, R5) **Value of R5:** 0x34000000
2. ADD(R2, R1, R6) **Value of R6:** 22
3. ADD(R0, 2, R7) **Value of R7:** ~~10~~ ? 20. (这里2is R2的意思. 0x1238)
4. ST(R1, 4, R3) **Value stored:** ~~0x1234~~ (0) **at address:** ~~0x1400~~

(C) A student tries to optimize his Beta assembly program by replacing a line containing

ADDC(R0, 3*4+5, R1)

by

ADDC(R0, 17, R1)

Is the resulting binary program smaller? Does it run faster?

(circle one) Binary program is SMALLER? yes … no

(circle one) FASTER? yes … no

(D) A BR instruction at location 0x1000 branches to 0x2000. If the binary representation for that BR were moved to location 0x1400 and executed there, where will the relocated instruction branch to?

Branch target for relocated BR (in hex): 0x 2400?

(E) A line in an assembly-language program containing "ADDC(R1,2,R3)" is changed to "ADDC(R1,R2,R3)". Will the modified program behave differently when executed?

change to 2.

Circle best answer: YES … NO … CAN'T TELL

Problem 3.

Each of the following programs is loaded into a Beta's main memory starting at location 0 and execution is started with the Beta's PC set to 0. Assume that all registers have been initialized to 0 before execution begins. Please determine the specified values after execution reaches the HALT() instruction and the Beta stops. Write "CAN'T TELL" if the value cannot be determined. **Please write all values in hex.**

(A)

```
        . = 0
        LD(R31,X+4,R1)
        SHLC(R1,2,R1)
        LD(R1,X,R2)
        HALT()
   X:   LONG(4)
        LONG(3)
        LONG(2)
        LONG(1)
        LONG(0)
```

Value left in R1: 0x_____

Value left in R2: 0x_____

(B)

```
        . = 0
        LD(R31,X,R0)
        CMOVE(0,R1)
L:      CMPLTC(R0,0,R2)
        BNE(R2,DONE)
        ADDC(R1,1,R1)
        SHLC(R0,1,R0)
        BR(L)
DONE:   HALT()
X:      LONG(0x08306352)
```

Value left in R0: 0x_____

Value left in R1: 0x_____

Value left in R2: 0x_____

Value assembler assigns to symbol X: 0x_____

(C)

```
        . = 0
        LD(R31,Z,R1)
        SHRC(R1,26,R1)
Z:      CMPLTC(R1,0x3C,R2)
        HALT()
```

Value left in R1: 0x_____

Value left in R2: 0x_____

(D)

```
        . = 0
        LD(R31,X,R0)
        CMOVE(0,R1)
L:      ADDC(R1,1,R1)
        SHRC(R0,1,R0)
        BNE(R0,L,R2)
        HALT()

        . = 0x100
X:      LONG(5)
```

Value left in R0: 0x_____

Value left in R1: 0x_____

Value left in R2: 0x_____

Value assembler assigns to symbol X: 0x_____

(E)

```
        . = 0
0       LD(r31, X, r0)
4       CMPLE(r0, r31, r1)
8       BNE(r1, L1, r1)
C       ADDC(r31, 17, r2)
10      BEQ(r31, L2, r31)
14  L1: SRAC(r0, 4, r2)
18  L2: HALT()

        . = 0x1CE8
X:      LONG(0x87654321)
```

Value left in R0? 0x_____

Value left in R1? 0x_____

Value left in R2? 0x_____

Value assembler assigns to L1: 0x_____

(F)

```
        . = 0
        LD(R31, i, R0)
        SHLC(R0, 2, R0)
        LD(R0, a-4, R1)
        HALT()
a:      LONG(0xBADBABE)
        LONG(0xDEADBEEF)
        LONG(0xC0FFEE)
        LONG(0x8BADF00D)
i:      LONG(3)
```

Contents of R0 (in hex): 0x_____

Contents of R1 (in hex): 0x_____

(G)

```
        . = 0
        LD(R31,Z,R1)
        SHRC(R1,16,R2)
Z:      SUBC(R2,0x3C,R3)
        HALT()
```

Value left in R1: 0x_____

Value left in R3: 0x_____

Value assembler assigns to symbol Z: 0x_____

(H)

```
        . = 0
0       LD(R31,X,R0)
4       CMOVE(0,R1)

8   L:  ADDC(R1,1,R1)
C       SHRC(R0,1,R0)
10      BNE(R0,L,R2)
14      HALT()

        X: LONG(0xDECAF)
```

Value left in R0: 0x_____

Value left in R1: 0x_____

Value left in R2: 0x_____

1. 可以把 instruction Load 到 Register.

2. BNE, BEQ 指令会把 PC 的值放入 Register.

Computation Structures

Compilation Worksheet

compile_expr(expr) ⇒ Rx

- Constants: `1234` ⇒ Rx
  - `CMOVE(1234,Rx)`
  - `LD(c1,Rx)`
    …
    `c1: LONG(123456)`
- Variables: `a` ⇒ Rx
  - `LD(a,Rx)`
    `...`
    `a: LONG(0)`
- Variables: `b[expr]` ⇒ Rx
  - `compile_expr(expr)⇒Rx`
    `MULC(Rx,bsize,Rx)`
    `LD(Rx,b,Rx)`
    …
    `// reserve array space`
    `b: . = . + bsize*blen`
- Operations: $expr_1$ + $expr_2$ ⇒ Rx
  - `compile_expr(`$expr_1$`)⇒Rx`
    `compile_expr(`$expr_2$`)⇒Ry`
    `ADD(Rx,Ry,Rx)`
- Procedure call: `f(`e_1, e_2, …`)` ⇒ Rx
  next lecture!
- Assignment: `a=expr` ⇒ Rx
  - `compile_expr(expr)⇒Rx`
    `ST(Rx,a)`

compile_statement(…)

- Unconditional: *expr*;
  - `compile_expr(expr)`
- Compound: { *s1*; *s2*; … }
  - `compile_statement(s1)`
    `compile_statement(s2)`
    …
- Conditional: `if (expr) s1;`
  - `compile_expr(expr)⇒Rx`
    `BF(Rx,Lendif)`
    `compile_statement(s1)`
    `Lendif:`
- Conditional: `if (expr) s1; else s2;`
  - `compile_expr(expr)⇒Rx`
    `BF(Rx,Lelse)`
    `compile_statement(s1)`
    `BR(Lendif)`
    `Lelse:`
    `compile_statement(s1)`
    `Lendif:`
- Iteration: while (*expr*) *s1*;
  - `BR(Ltest)`
    `Lwhile:`
    `compile_statement(s1)`
    `Ltest:`
    `compile_expr(expr)⇒Rx`
    `BT(Rx, Lwhile)`
- Iteration: `for (init; test; incr) s1;`
  `init;`
  `while (test) { s1; incr; }`

Problem 1.

Please hand-compile the following snippets of C code into equivalent Beta assembly language statements. Assume that memory locations have been allocated for the all C variables with labels that corresponds to the variable names. So to load the value of the C variable `a` into register `R3`, the appropriate assembly language statement would be `LD(R31,a,R3)`. And to store the value in `R17` to the C variable `b`, the appropriate assembly language statement would be `ST(R17,b,R31)`. Similarly, assume that memory locations have been allocated for each C array, with a label defined whose value is the address of the 0^{th} element of the array.

(A)
```
a = 42;
```

(B)
```
c = 5*x - 13;
```

(C)
```
y = (x - 3)*(y + 123456);
```

(D)
```
if (a == 3) b = b + 1;
```

(E)
```
a[i] = a[i-1];
```

(F)
```
x = y[3] + y[12];
```

(G)
```
if (b == 0 || b < min) {
    min = b;
} else {
    too_big += 1;
}
```

(H)
```
sum = 0;
i = 0;
while (i < 10) {
    sum = sum + i
    i = i + 1;
}
```

Problem 2.

In block-structured languages such as C or Java, the scope of a variable declared locally within a block extends only over that block, i.e., the value of the local variable cannot be accessed outside the block. Conceptually, storage is allocated for the variable when the block is entered and deallocated when the block is exited. In many cases, this means the compiler if free to use a register to hold the value of the local variable instead of a memory location.

Consider the following C fragment:

```
int sum = 0;
{ int i;
  for (i = 0; i < 10; i = i+1) sum += i;
}
```

A. Hand-compile this loop into assembly language, using registers to hold the values of the local variables "i" and "sum".

```
MOV(R31, R0)
ST(R0, sum)
MV(R31, R1)
BR(L2)
L1: ADD(R0, R1, R0)
    ADDC(R1, 1, R1)
L2: CMPLTC(R1, 10, R2)
    BT(R2, L1)
    ST(R0, sum)
```

B. Define a *memory access* as any access to memory, i.e., instruction fetch, data read (LD), or data write (ST). Compare the number of total number of memory accesses generated by executing the optimized loop with the total number of memory access for the unoptimized loop (part G of the preceding problem).

H: 2 instructions
2 memory accesses.

loop(1, 10).
10 instruction fetch.
6 memory accesses.
⇒ 2+2+10×(6+4) = 104.

A: 4+1
Loop. (1, 10)
4
+1 = 46.

C. Some optimizing compilers "unroll" small loops to amortize the overhead of each loop iteration over more instructions in the body of the loop. For example, one unrolling of the loop above would be equivalent to rewriting the program as

```
int sum = 0;
{ int i;
  for (i = 0; i < 10; i = i+2) {
      sum += i; sum += i+1;
  }
}
```

Hand-compile this loop into Beta assembly language and compare the total number of memory accesses generated when it executes to the total number of memory accesses from part (1).

Problem 3.

Which of the following Beta instruction sequences might have resulted from compiling the following C statement? For each sequence describe the value that does end up as the value of y.

```
int x[20], y;
y = x[1] + 4;
```

A.
```
LD (R31, x + 1, R0)
ADDC (R0, 4, R0)
ST (R0, y, R31)
```
exception.

B.
```
CMOVE (4, R0)
ADDC (R0, x + 4, R0)
ST (R0, y, R31)
```
address of X[2]

C.
```
LD (R31, x + 4, R0)
ST (R0, y + 4, R31)
```
⇒ $R_0 \leftarrow X[1]$.
If next address $\leftarrow X[1]$.

D.
```
CMOVE (4, R0)
LD (R0, x, R1)
ST (R1, y, R0)
```
$R_1 \leftarrow Y[1]$.
$Y \leftarrow X[1]$.
Same as C.

E.
```
LD (R31, x + 4, R0)
ADDC (R0, 4, R0)
ST (R0, y, R31)
```

F.
```
ADDC (R31, x + 1, R0)
ADDC (R0, 4, R0)
ST (R0, y, R31)
```
? error