2/13/15

6.004 Recitation

Fernando Trujano

Truth Table → Boolean Expression → circuit diagram

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

and

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

+ ← or

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

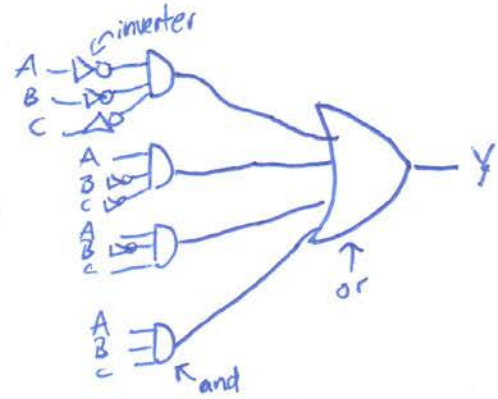
+

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

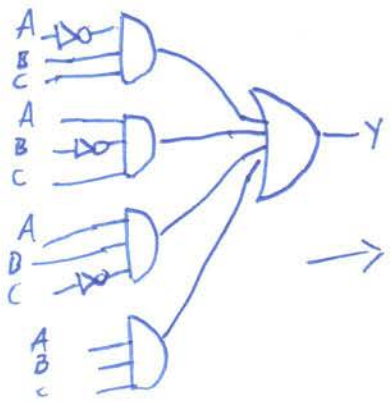
+

$A \cdot B \cdot C$

$\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC$



now the other way...



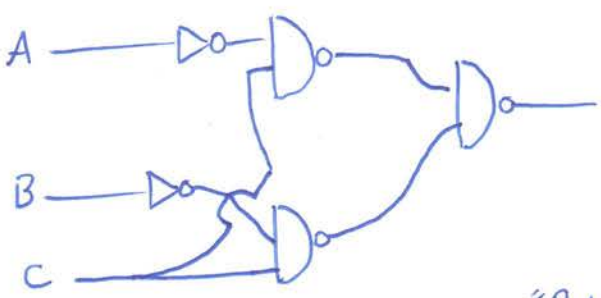
$\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

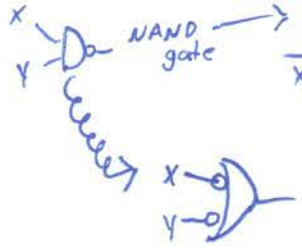
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

circuit diagram → Boolean expression → truth table

Another example with inverting logic

↳ could convert it first ⟹ De Morgans Law

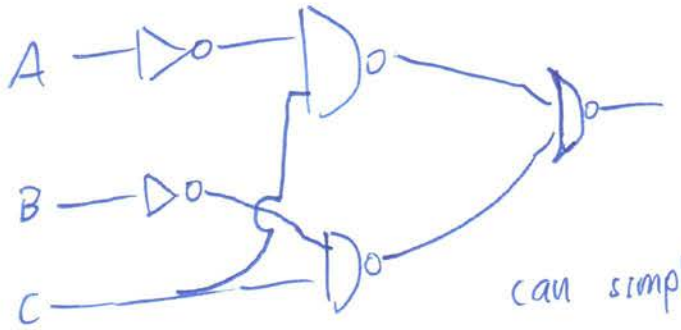




$\overline{XY} = \bar{X} + \bar{Y}$

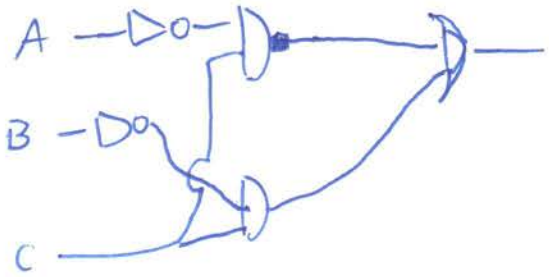
$\overline{X+Y} = \bar{X} \cdot \bar{Y}$

"Pushing bubbles"



can simplify because $\bar{\bar{X}} = X$

pushing bubbles

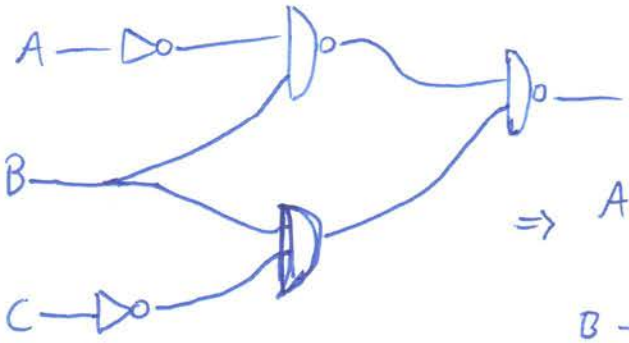


Propagation delays:

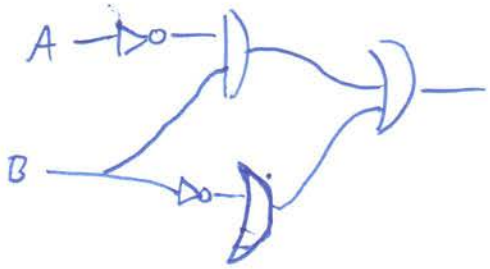
INV	1 μs
AND	4 ns
OR	4 ns
NAND	3 μs

$\boxed{\text{Inverting logic is Faster}}$

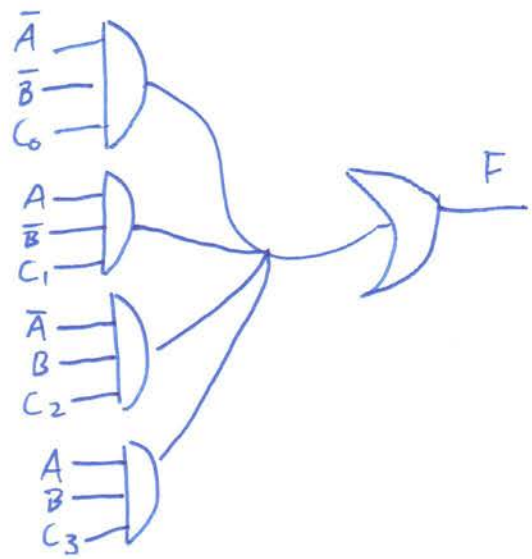
Propagation delay: 8 ns



$\Rightarrow$



To optimize non inverting logic can add 2 bubbles to the end and then push one of them through the circuit



$$F = \bar{A}\bar{B}C_0 + A\bar{B}C_1 + A\bar{B}C_2 + ABC_3$$

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

XOR

Question: Pick C_0, C_1, C_2, C_3 to make $F = A \oplus B$

Assume $A = B = 0$ and C's don't exist.

we get $C_0 = 0$ we want output to be 0

Now $A = 1$ $B = 0$ desired output $= 1$

$C_1 = 1$

$C_2 = 1$

Now $A = 1$ $B = 1$

$C_3 = 0$

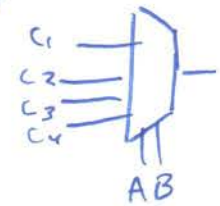
OR Can look at it algebraically.

Sum of Products for $A \oplus B$

$\bar{A}B + A\bar{B}$

$$F = \bar{A}\bar{B}C_0 + A\bar{B}C_1 + \bar{A}B\,C_2 + ABC_3$$

So $C_0 = 0$
$C_1 = 1$
$C_2 = 1$
$C_3 = 0$



We can compute any boolean function by setting C_1 C_2 C_3 C_4

$F = A$
$\Rightarrow C_0 = 0$ $C_2 = 0$
$C_1 = 1$ $C_3 = 1$

6.004 Lecture

Synthesis of Combinational Logic

Sum of products

Take outputs where Y is one. AND all inputs and OR All once

C	B	A	Y	
0	0	0	0	
0	0	1	1	$\bar{C}\bar{B}A$
0	1	0	0	
0	1	1	1	$C\bar{B}A$
1	0	0	0	
1	0	1	0	
1	1	0	1	$CB\bar{A}$
1	1	1	1	CBA

$$Y = \underbrace{\bar{C}\bar{B}A}_{\text{AND}} \underbrace{+}_{\text{OR}} C\bar{B}A + CB\bar{A} + CBA$$

Reduction : $\alpha A + \alpha \bar{A} = \alpha$

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

CB

$\bar{C}A$

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

18/2/15

6.004 Recitation

Fernando Trujano

Logic Synthesis II

AND, OR, INV } Universal. Can combine to make <u>any</u> gate

Simplification

$\cancel{A}B\bar{C} + \cancel{\bar{A}}B\bar{C} + \overline{AB\cancel{C}} + \overline{AB}\cancel{C}$ ← using reduction $\boxed{\bar{a}b + ab = b}$

$= B\bar{C} + \overline{AB}$

← MIN SUM OF PRODUCTS

Karnaugh Maps - another way to do reduction

Ex:

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

$F = \overline{ABC} + \cancel{\bar{A}}B\bar{C} + \cancel{A}B\bar{C} + ABC$

$= \overline{AC} + AB$

K-MAP

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

- Make table
- join adjacent is in powers of two
- For each group see what remained constant

$F = \overline{CA} + AB$

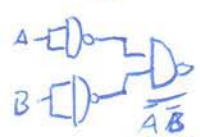
Universality

NAND GATE IS UNIVERSAL

aka can make AND, OR, INV out of NAND

INV

OR



$= \bar{\bar{A}} + \bar{\bar{B}} = A + B$

AND

Example

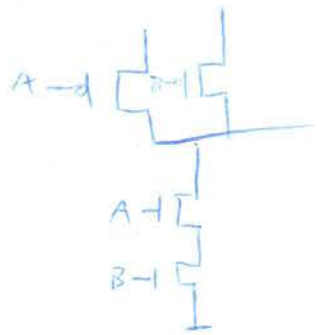
$F = \bar{A} + \bar{B}A + \bar{B}\bar{A}$

A) Draw single CMOS gate that computes F

$F = \bar{A} + \bar{B}A + \bar{B}\bar{A}$

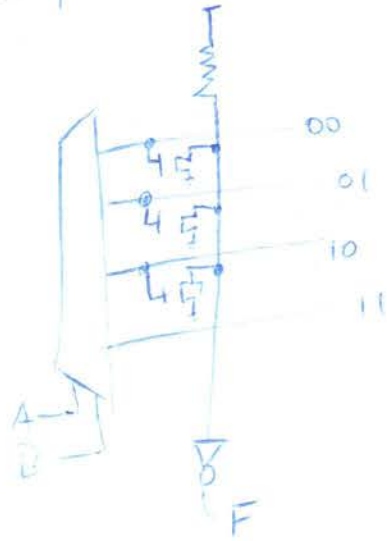
$= \bar{A} + \bar{B}$ ⇐ NAND

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



B) Make a ROM that computes F

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



ROM # ROWS = $2^{\#\text{inputs}}$

Functions = $2^{\#\text{rows}}$

2/19/15

6.004 Lecture

Fernando Trujano

Sequential Logic: Adding a little State

2/20/15

6.004 Recitation

Fernando Trujano

Combinational vs Sequential logic → state

↳ no outputs depend on previous outputs

CLK (clock)

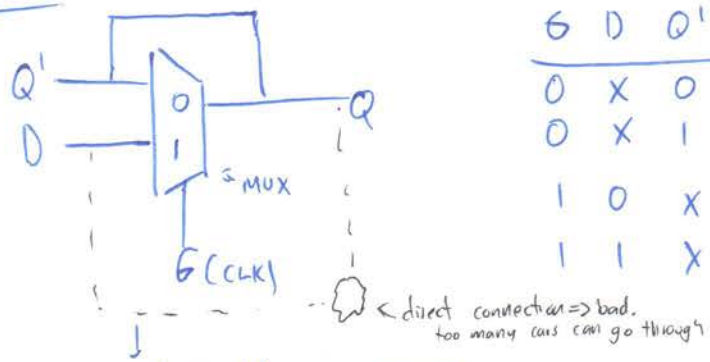
CLK 

Storing state

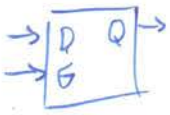
Capacitors - need to keep refreshing, not reliable

Feedback -  Could get stuck in a third "in between" state.

Latch



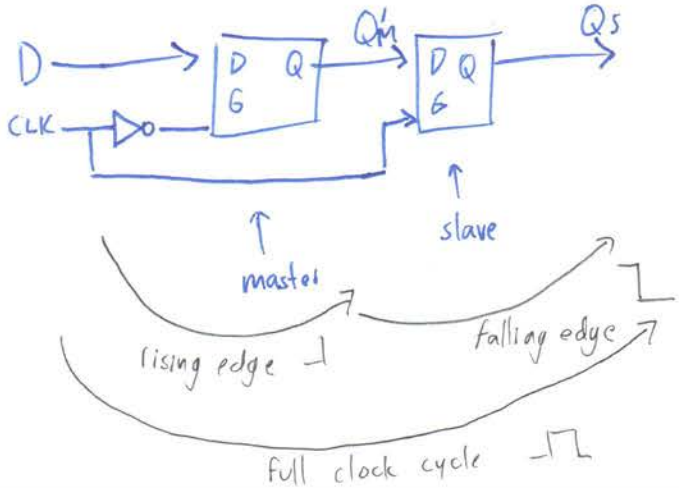
G	D	Q'	Q
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

Drawn like: 

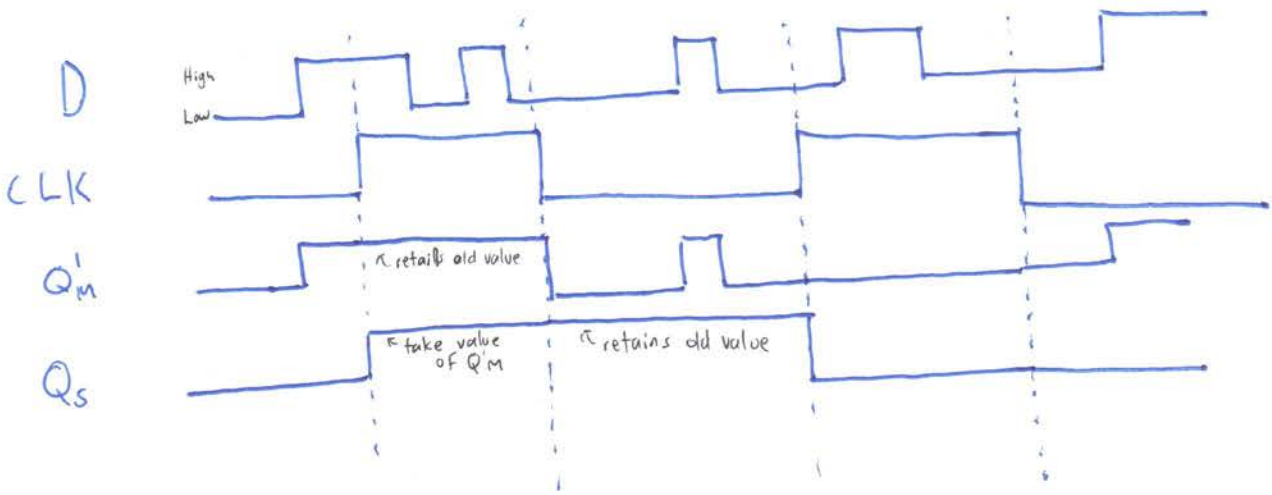
G controls weather Q = D or Q loops internally.

Flip-Flop 

- Made out of two latches. Prevents cycle ← think car gate example from lecture

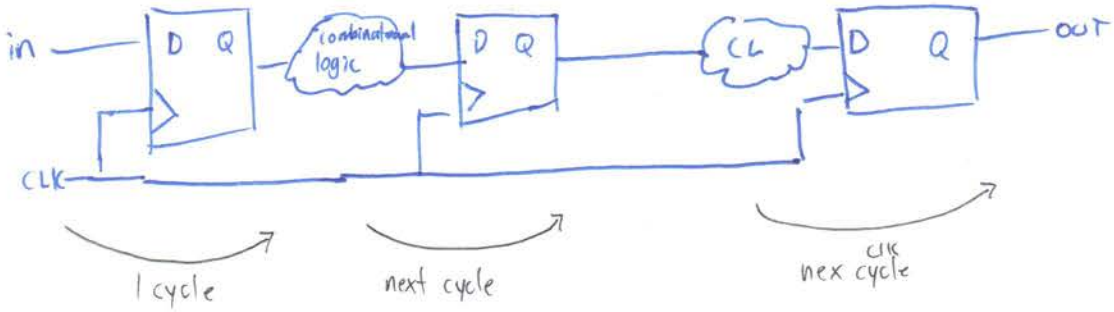


Never a direct connection from D to Q



<u>Register</u>: 1+ flip flops that work as a group to store a set of bits

1bit register = Flip Flop 8 bit register



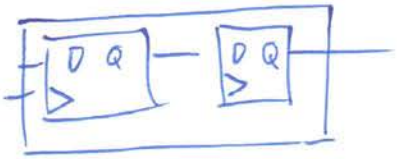
"clock skew"

Cycle here could be recieved erlier here than here.

Because the signal takes some time to travel...

Just be aware of this... not super important for 6.004

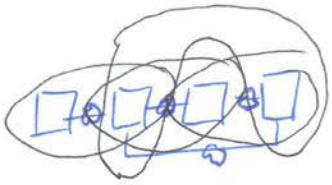
Important Equations



Flip Flops

T_{setup}: How long before rising edge m input must be held valid

T_{hold}: " after "



For all connected pairs of registers in circuits:

Memorize These!

(1) $T_{hold\ reg2} < T_{CD\ reg1} + T_{CD\ logic}$

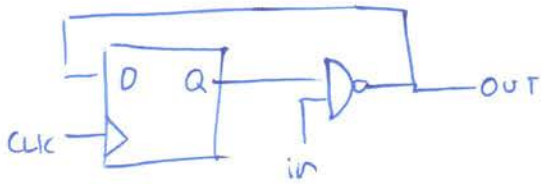
contamination delay

(2) $T_{CLK} > T_{PD\ reg1} + T_{PD\ logic} + T_{setup\ reg2}$

For all paths from external input to reg

(3) $T_{setup\ in} = T_{setup\ reg} + T_{PD\ logic}$

(4) $T_{hold\ in} = T_{hold\ reg} - T_{CD\ logic}$



	T_{CD}	T_{PD}	T_S	T_H
A and Z	?	5ns	–	–
d-reg	1	3	2	2

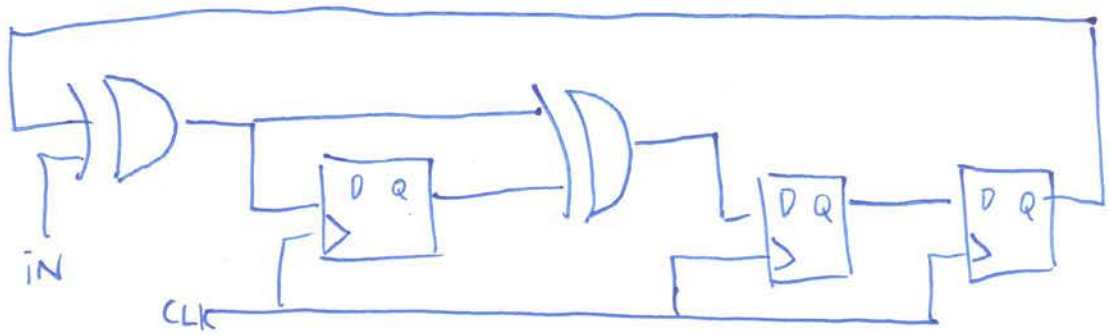
Find T_{CD}: $2 < 1 + ? = 1$ Equation (1)

min CLK: $T_{CLK} > 3 + 5 + 2 = 10$ Equation (2)

input: $T_S = 5 + 3 = 8$

$T_H = 2 - 1 = 1$

reg holds for T_{CD}



	T_{CD}	T_{PD}	T_S	T_H
xor 2	.4	2.1		
dreg	.2	1.8	.8	.15
dreg 2	.1	.4	.8	.15

two options

Fin CLK cycle:

Chose longest possible PD between any two pairs of registers

$$T_{CLK} > (2.1)(2) + 1.8 + \overset{\text{setup}}{.8}$$

$$T_{SIN} = .8 + (2.1)(2)$$

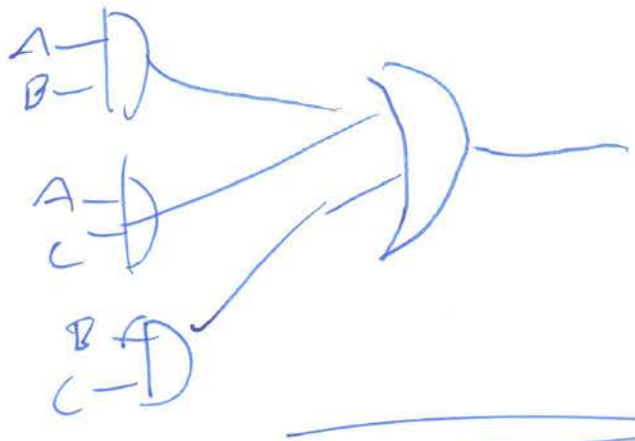
longest PD from input to register could be different

$$T_{H_{IN}} = .15 - (.4 \times 2)$$

would it still work if we use dreg 2?

idk do it later

6.004 Checkoff



$$\overline{\overline{C_{OUT} = AB + AC_{IN} + BC_{IN}}}$$

$$\overline{\overline{A^* + B^* + C^*}}$$

$$\overline{\overline{AB}\ \overline{AC_{IN}}\ \overline{BC_{IN}}}$$

$$\overline{A^* B^* C^*}$$

6.004 Recitation

Fernando Trujano

Sequencial Logic

For all pairs of connected registers ($reg_1 \rightarrow reg_2$)

$$T_{hold\ reg_2} \leq T_{CD\ reg1} + T_{CD\ logic}$$

$$T_{CLK} \geq T_{PD\ reg1} + T_{PD\ logic} + T_{setup\ reg2}$$

$T_{CD} \rightarrow$ shortest path

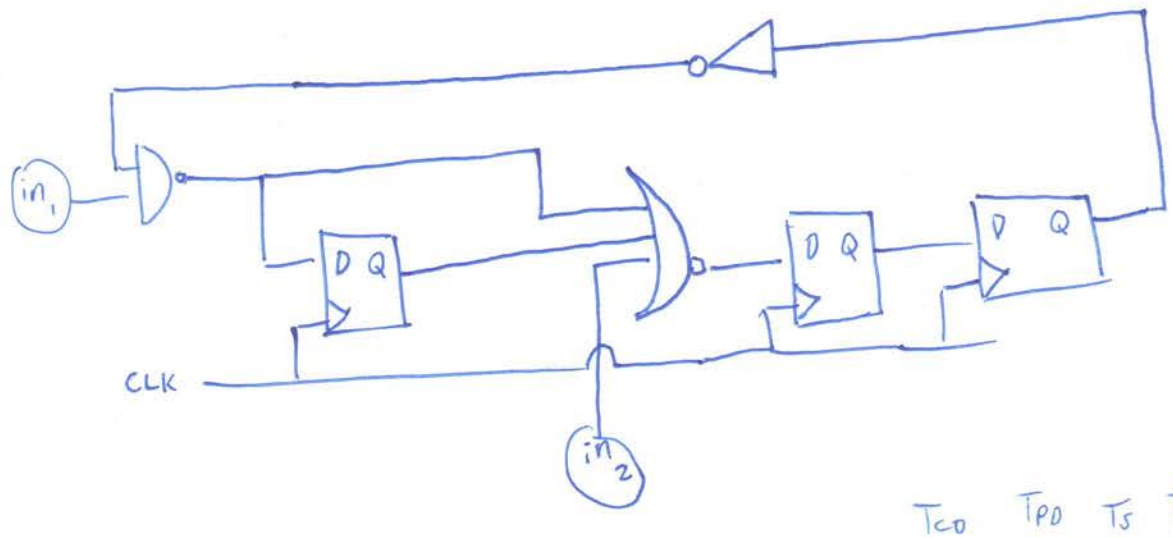
For all paths from an external input to register

$T_{PD} \rightarrow$ longest path

$$T_{setup\ in} \geq T_{setup\ reg} + T_{PD\ logic}$$

$$T_{hold\ in} \geq T_{hold_{reg}} - T_{CD_{logic}}$$

Ex:



	T_{CD}	T_{PD}	T_S	T_H
inv	.1	.2	–	–
nand 2	.2	1	–	–
nor3	.4	2.2	–	–
dreg	.2	2	.6	?

$$T_H \leq .2 + 0$$

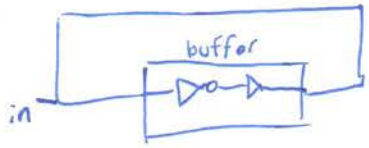
$$T_{CLK} \geq 2ns + (.2 + 1 + 2.2) + .6 = 6$$

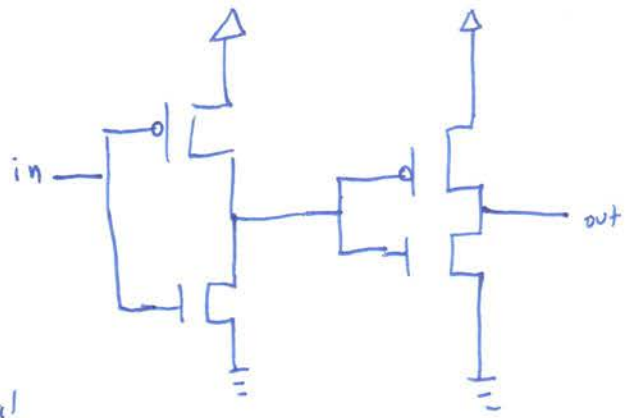
$$T_{Sin} \geq .6ns + (1 + 2.2) = 3.8$$

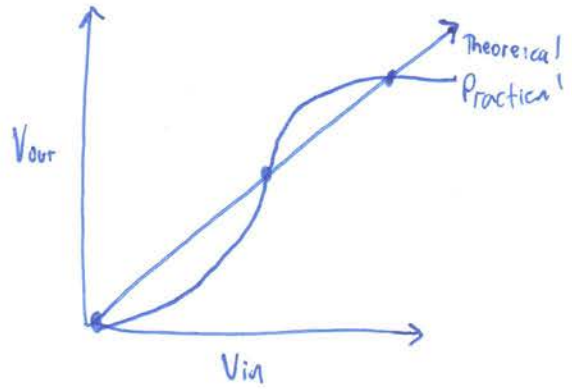
$$T_{Hin} \geq .2 - .2 - 0$$

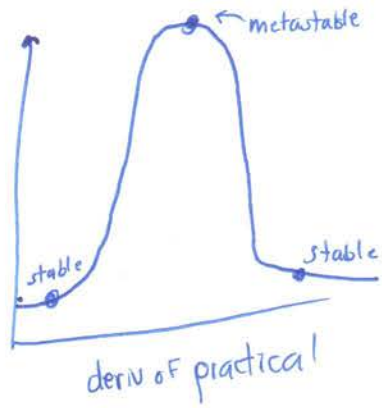
This not on quiz!

Metastability









metastability will always resolve → not resutant to noise.
↳ not in bounded time

Asynchronous Arbiter

- not synchronized with clock ≠ bounded time

* which button was pressed first

Pulse synchronizer: Registers in a row ⇒ delay
↳ the more you wait, exponentially more likely to resolve metastable

FSM

Finite state machines

- Moore
- Mealy
- Equivalence

2/25/15

6.004 Quiz I Review

Fernando Trujano

Topics + key Terms

Information

- Info resolves uncertainty
- Huffman Code
- Two's complement
- Hamming Distance
- Error correction + detection

EX: F2013 P1, S2014 P2, S2012 P1(0)

Digital Abstraction

- Signal validity
- Noise margins
- Buffer + Inverter
  ↳ non-linear VTC
  gain > 1
- Static Discipline

Ex. F2013 P2, F2014 P3

CMOS

- FET ≈ Voltage controlled switch
- NFETs in pulldown
- PFETs in pullup
- Naturally inverting
- t_{PD}, & t_{CD}
- Leniance

Ex S2012 P4, S2012 P1(E) S2014 P1(A)
F2009 P4

Logic Synthesis

- Truth Tables
- Sum of products
- Table Lookup
  ↳ w/ MUX
  ↳ w/ ROM

Ex 2013 P5(A,C)

Sequential Logic

- Latches
- Registers
  ↳ T_{PD}, T_{CD}
- Dynamic discipline
  ↳ T_{setup}, T_{Hold}
- Clock constraints

EX F2013 P4, S2013 P5(E)

Sample Problems + Notes

Information

1) Write -37 in 2's complement

① Write 37 in binary : 00100101
② Negate 37 ↳ Flip every bit and add 1 — 11011011 = -37

2) Optimal Encoding:

Huffman Encoding - lowest probability get longer encoding

① Take two events with lowest prob and link together
- Add their probabilities in new event
Repeat

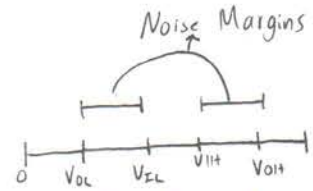
② Assign bits for each branch

*Events always at leaves

Entropy := ($Pr_1 \cdot$ # bits to encode$_1$ + Pr_2 # " $_2$ . . . .)
↳ Average # bits for ~~fixed~~ variable length encoding

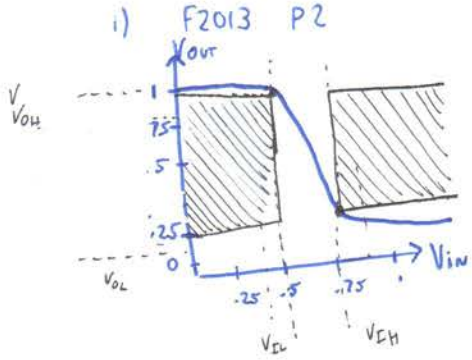
Hamming Distance - # different bits ← Take min of all pairs
↳ how many 1 bit changes to have another valid signal

detect N bits : N+1 hamming code
correct N bits : 2N+1 "

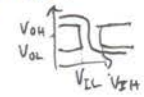


Digital Abstraction

1) F2013 P2



Pick values that obey static discipline and have widest margins.
↳ inputs more lenient than outputs.

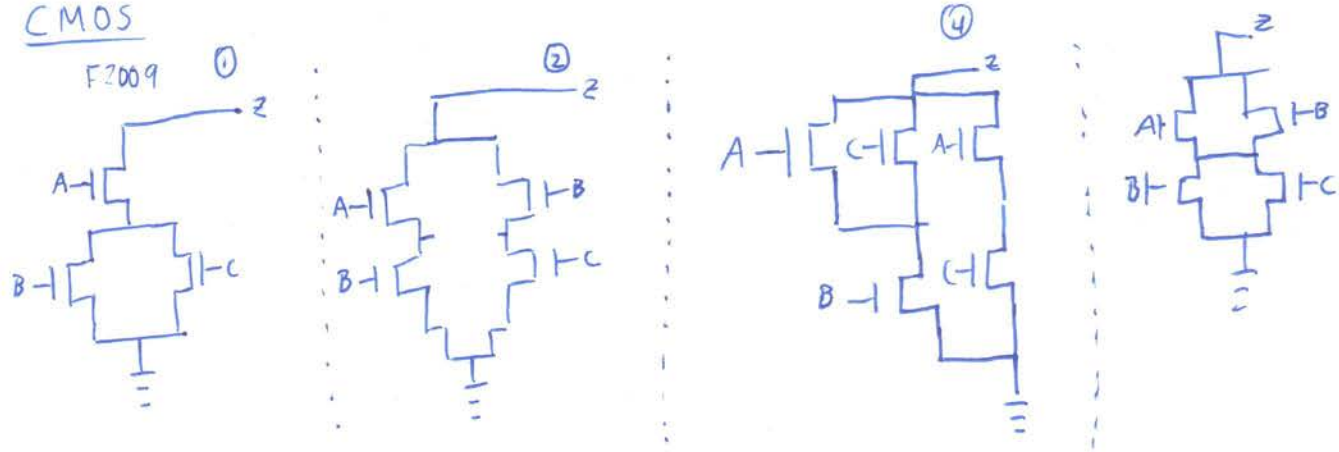


$V_{OH} = 1$
$V_{IH} = .75$
$V_{IL} = .5$
$V_{OL} = .25$

CMOS

F2009

All Pulldowns



PFETS : conduct when 0 Pull up

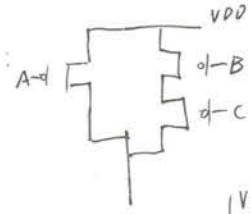
Write functions for each :

① $\overline{A \cdot (B + C)}$ (AND, OR) — negate b/c pull down

DeMorgans : $\overline{A(B+C)}$ => Pullup :

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

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

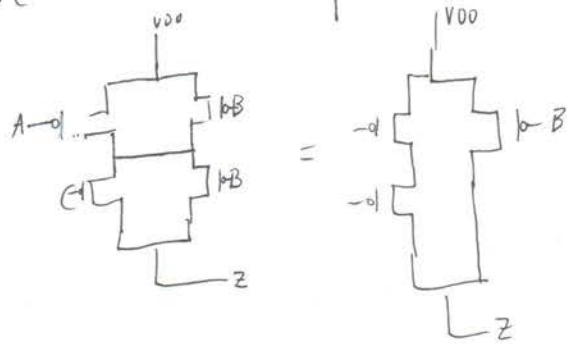


② Pullup

$Z = \overline{AB + BC}$

$\overline{AB} \cdot \overline{CB}$

$(\bar{A} + \bar{B}) \cdot (\bar{C} + \bar{B})$ ← formula for pullup circuit



Sequential Logic

Register :

T_{SETUP} : before rising edge : hold D input stable and valid

T_{HOLD} : after rising edge : "

6.004 Notes Review Quiz 1

Fernando Trujano

Information

Information recieved: $I(x_i) = \log_2(1/p_i)$ bits

if data reduces N equally likely → M $I(\text{data}) = \log_2(N/M)$ bits

Entropy: Avg amount of info recieved $H(X) = E(I(X)) = \sum_{i=1}^{N} p_i \underbrace{\log_2\left(\frac{1}{p_i}\right)}_{\#\text{ bits}}$

Avg < Entropy => not enough info

" = " => perfect

" > " => OK, inneficient

Encoding Numbers: 2's complement: high order bit carries negative weight

$-A = \sim A + 1$ ← flip all bits

$$\begin{array}{r} 0011 \\ +\ 0011 \\ \hline 0110 \end{array} \qquad \begin{array}{l} 1111=10 \\ 1111+1=11 \end{array}$$

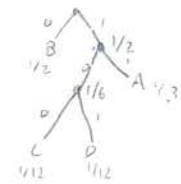
Range N-bits $\left[-2^{N-1} \ldots +2^{N-1}-1\right]$

Huffman Algorithm: Create optimal encoding: ↑P => shorter encodings

↓P => longer "

- connect two symbols/sub trees with ↓P
  - add P's
- Repeat
- Label branches

A = 1/3
B = 1/2
C = 1/12
D = 1/12



Error Detection + Correction:

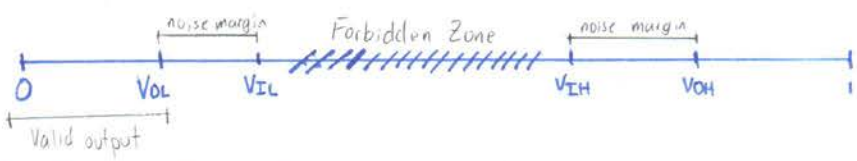
Hamming distance: # different bits

Detect E errors: HD ≥ E+1 Correct E errors: HD ≥ 2E+1

Digital Abstraction

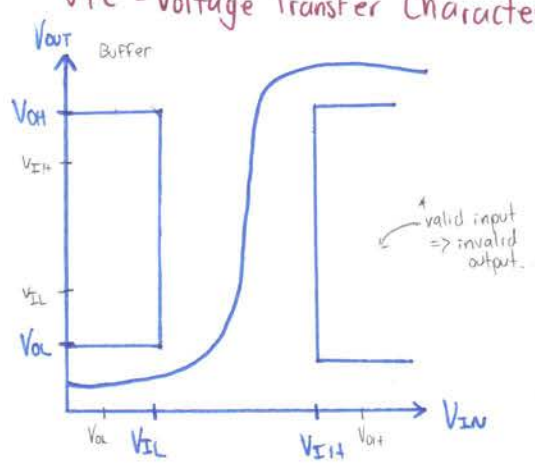
Voltage signals easily affected by noise

- allow inputs to have more "freedom" than outputs to account for noise



Static discipline: valid input → valid output*

VTC - Voltage Transfer Characteristic - measures static behavior



Slope - gain of combinational device

CMOS

NFets: 

G	"switch"
0	OFF
1	connected

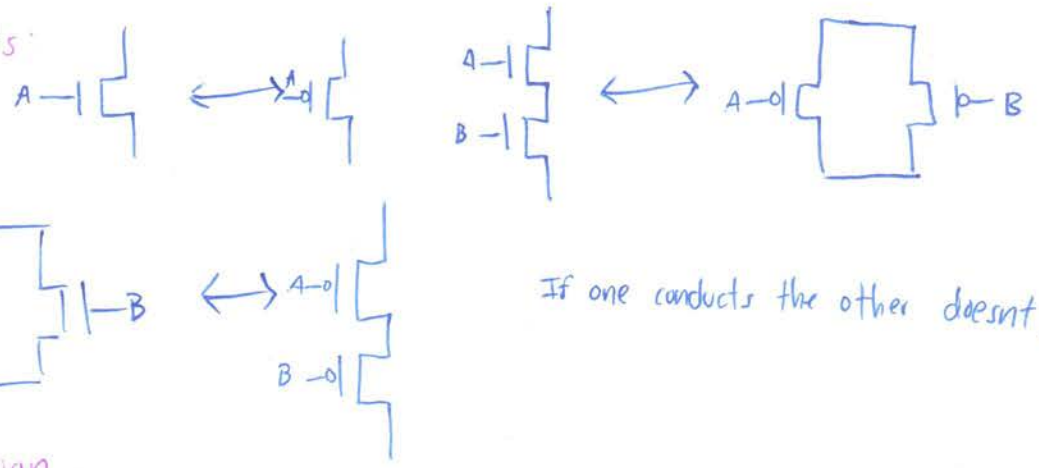
only use in pulldown

PFets: 

G	switch
0	connected
1	OFF

only use in pullup

Complements:



If one conducts the other doesnt!

How to design

- Construct pulldown for when $F = 0$
  OR
- Construct pullup for when $F = 1$

then build complement

Timing: Propagation delay T_{PD}: Upperbound valid inputs → valid outputs
Contamination delay T_{CD}: Lowerbound invalid inputs → invalid outputs

Leniance: can produce valid outputs when a subset of inputs have been stable
All 1 input gates are leniant

Naturally Inverting: Rising input can only cause falling output. (Single CMOS gate)
$0 \to 1 \Rightarrow 0 \to 1$ ← not a single CMOS Gate
input output

For a single gate: changing input to 1 can only change output to 0

Logic Synthesis

Boolean Logic: NOT: $\bar{A}$
OR: $A+B$
AND: AB

De-Morgan's Law: $\overline{A+B} = \bar{A} \cdot \bar{B}$
$\overline{A \cdot B} = \bar{A} + \bar{B}$

Inverter: 

AND: 

OR: 

Universality: Can use gate to make inverter, and, or => can make any SUM-OF-PRODUCTS

Logic Simplification: $ab + \bar{a}b = b$ $\quad (a+b)(\bar{a}+b) = b$

6.004

R-R

$T_{HOLD} R_2 = T_{CD} R_1 + T_{CD}\ \text{logic}$

$T_{CLK} = T_{PD} R_1 + T_{PD}\ \text{logic} + T_{SETUP} R_2$

I→R

$T_{SETUP} I = T_{SETUP} R + T_{PD}\ \text{logic}$

$T_{HOLD} I = T_{HOLD} R - T_{CD}\ \text{logic}$

$T_{HOLD} R_2 \leq T_{CD} R_1 + T_{CD}\ \text{logic}$

$R_1 \to R_2$

$T_{HOLD} R_2 \leq T_{CO} R_1 + T_{CO}\ logic$

$T_{CLK} \geq T_{PD} R_1 + T_{PD}\ logic + T_{SET} R_2$

$I \to R$

$T_{SETUP} I = T_{SETUP} R + T_{PD}\ logic$

$T_{HOLD} I = T_{HOLD} R - T_{CO}\ logic$

$R_1 \to R_2$

$T_{HOLD} R_2 = T_{CO} R_1 + T_{CO}\ logic$

$T_{CLK} = T_{PD} R_1 + T_{PD}\ logic + T_{SET} R_2$

$I \to R$

$T_{SETUP} I = T_{SETUP} R + T_{PD}\ logic$

$T_{Hold} I = T_{HOLD} R - T_{CO}\ logic$

2/26/15

6.004 Lecture #7

Fernando Trujano

Pipelining

Processing several generations of inputs in parallel

For combinational

Latency -> how long for input to get through system $= t_{PD}$

Throughput -> rate at which outputs produced $= 1/t_{PD}$

Check for well-formed pipeline.

- Every path from input -> output should have k registers

Quiz I Hint

- In Problem 4 one of the answers is none

6.004 Recitation

Fernando Truja

★ Do lab 4 before Quiz II

Beta Architecture

Types of Memory:

Registers

- 32 registers. Each 32 bits ~~right~~ wide
- Labled with names: R0, R1, R2 ... R31
- Special Registers: R27 ... R31 ← discards what you write always get 0.
- small, Fast, expensive
- temporary "scratch space"

Memory (RAM)

- stores program instructions
  program variables (brg)
- stack

Disk

- cheap, slow
- store Files, apps

16-bits

Memory

address	value
	~~00100011~~ = 0xFF
	0xFFFF0011 ← 32 bit word (1 2 3 4)
0x0	
0x4	0x11111111

starts at byte address 4

1 hex digit = 4 bit
1 byte = 8 bits
32-bit = 4 bytes

2^{10} = kB
2^{20} = MB
2^{30} = GB

byte addresses = 2^{16}

word-aligned = $2^{16}/4 = 2^{14}$

how many bytes memory = 2^{16} bytes

how many 32-bit words can memory hold = 2^{14}

Compiling

Java
↓
Assembly Language
↓
10110 Machine Code

LD(R31, 0x4, R1)
R1 ← 0x11111111

translations of app

```
.=0x0
ADDC(R31, N, R0)
X: LD(R0, 8, R31)        0x84
                         ↳ since X is on 0x84
ADDC(R31, X, R2)
SRA(R1, R2, R3)
ST(R3, 0x4, R0)
ST(R0, 0, R31)
HALT()
                         // skip to address 0x2000
=0x2000
N: LONG(0x12345678)
   LONG(0xDEADBEEF)
   LONG(0xEDEDEDED
```

Label

Registers

R0 ← 0
R1 ← 0
R2 ← 0
R3 ← 0

Memory

address	values
0x0	0x181F2000
0x4	0x00200008
⋮	
0x2000	0x12345678
0x2004	0xDEADBEEF
0x2008	0xEDEDEDED

/12

3/11/15

6.004 Recitation

Fernando Trujano

PC (program counter) - which address in memory the processor is currently interpreting as a program instruction

label = PC + 4 + 4×SEXT (literal)

Example:

```
    . = 0x0
    LDR(a, R0)
    JMP(R0, R1)
a:  LONG(0xC)
    BEQ(R31, end, R31)
    ADDC(R0, 1, R0)
end: HALT()
```

memory

address	value
0	LDR
4	JMP
8	0x000000C
C	BEQ
10	ADDC
14	HALT
18	
1C	

Registers

R0 ← ~~0~~ ~~0x8~~ 0xC

R1 ← ~~0~~ ~~0x8~~ 0x10

① Translate instructions and write to memory

② Run instructions

Memory

address	value

Registers

R0 ←

R1 ←

R2 ←

Code:

```
a = [1, 2, 3]
For i in range(len(a))
        a[i] = a[i] + 1
if a[1] == 1 : a[0] = 0
else : a[2] = 0
```

Instructions:

```
    . = 0xF0   ← some location in memory
a:  LONG(0x1)
    LONG(0x2)   initialize array
    LONG(0x3)
    . = 0x0
    :
```

OR

```
R0 ← 0xF0
ADDC(R31, 1, R1)
ST(R1, 0, R0)
ADDC(R31, 1, R1)
ST(R1, 4, R0)      // store at R0+4
ADDC(R1, 1, R1)
ST(R1, 8, R0)
```

★ DONT FORGET

```
then:
    // index i = R1
    // array base R0 = 0xF0
    // array len = 3 R2

    ADDC(R31, 3, R2)        // len = 3
    ADDC(R31, 0, R1)        // i=0

    CMPEQ(R1, R2, R3)       // R3 ← (i == len)
beg-loop: BNE(R3, end-loop, R31)
For Loop  MULC(R1, 4, R3)       // R3 ← i×4   ← don't care about storing PC+4
          LD(R3, 0xF0, R4)      // R4 ← a[i]
          ADDC(R4, 1, R4)       // R4 ← a[i] + 1
          ST(R4, 0xF0, R3)
          ADDC(R1, 1, R1)       // i++
          BEQ(R31, beg-loop, R31)   // can also use JMP(beg-loop, R31)
end-loop: LD(R0, 4, R1)         // R1 ← a[1]   ← register
          CMPEQC(R1, 1, R2)     // R2 ← a[1] == 1
          BEQ(R2, else, R31)
iF:
          ST(R31, 0, R0)
          JMP(end, R31)
else:     ST(R31, 0, R0)

end:      HALT()
```

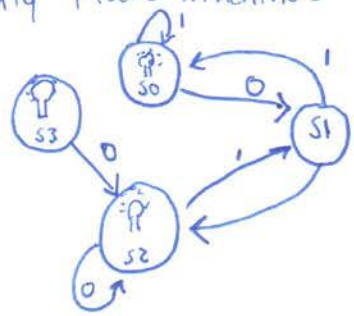
3/18/15

6.004 Recitation

Fernando Trujano

Quiz:

Only Moore machines



S	in	S'	out
S_0	0	S_1	1
S_0	1	S_0	1
S_1	0	S_2	0
S_1	1	S_0	0
S_2	0	S_2	1
S_2	1	S_1	1

Stack Detective

- Think about c and trace execution
- Assembly - think about pattern it creates on stack
- label given stack w/pattern
- Address stack w/ BP's
- determine which activation records go w/ which function calls

Factorial:

```
PUSH(LP)
PUSH(BP)
MOVE(SP, BP)
ALLOC(1)
PUSH(R1)

LD(BP, -12, R0)      // R0 ← x
BEQ(R0, rtn1, R31)
LD(BP, -12, R1)      // R1 ← x
SUBC(R0, 1, R0)      // R0 ← x-1
ST(R0, 0, BP)
LD(BP, 0, R0)
PUSH(R0)             // R0 → for next call arg 0 = x-1
BR(Factorial, BP)
DEALLOC(1)

MUL(R0, R1, R0)      // R0 ←
BEQ(R31, rtn, R31)
ADDC(R31, 1, R0)
POP(R1)
a: DEALLOC(1)
MOVE(BP, SP)
b: POP(BP)
POP(LP)
```

Stack Pattern ↓ (args pushed in reverse order)

arg0 (x)
LP
BP
BP → x-1 // y
R1
SP → x
LD
BD

stack

404	3	X	f(3)
408	11C	LP	
40C	380	BP	
BP → 410	2	Y	
414		R1	
418	2	X	LP's usually repeated
41C	168	LP	
420	410	BP	f(2)
BP → 424	1	Y	
428	3	R1	
42C	1	X	
430	168	LP	
434	424	BP	f(1)
BP → 438	BP → 0	Y	
43C	2	R1	
440	0	X	f(0)

BP? 0x438
SP? 0x440
LP? 0x168
R0? 0x1
↑ return value

address of original call f(3)? PC+4 ↓ 0x11C-4 = 0x118

```
int FFO (unsigned v, int b) {
    if (v==0)
        ????
    else
        return FFO( v>>1, b+1) ;
}
```

a? 0x190
b? 0x190

(Recall PUSH, POP take 2 instructions each

```
FFO: PUSH(LP)
     PUSH(BP)
     MOVE(SP,BP)
     PUSH(R1)
     LD(BP, -16, R0) //R0←B
     LD(BP, -12, R1) //R1←V
     BEQ(R1, rtn, R31) //rtnb:
     ADDC(R0,1,R0)
     PUSH(R0)
     SHRC(R1,1,R1)
     PUSH(R1)
     BR(ffo, LP)
     DEALLOC(2)
rtn: POP(R1)
     MOVE(BP,SP)
     POP(BP)
     POP(LP)
     JMP(LP)
```

stack bottom

b	
V	
LP	
BP	
R1	

original call from 30C

stack

310	LP	154	BP	
100	BP BP → 006			
FFF	R1	012		F(3,16)
00F	B	003		
01B	V			
208	LP			
12C	BP			
01B	R1			
01D	B			
00D	V			
209	LP			
140	BP			
154 00D	R1			
011	B			
006	V			
208	LP			f(6,17)

R1?
BP? 0x168
LP? 0x208
SP? 0x16C
PC? 0x00C

2/2

6.004 Quiz II Review

FSMs

- Moore FSM (state → output)
- Mealy FSM (state + input → output)
- Represent states w/ binary
- State equivalence
- Synchronizing sequence

Synchronizing Inputs

- arbitrer vs combinational
- can't time-bound arbiter
- meta stable state

Pipelining

- Latency and throughput
- K-stage pipeline
- K reg on each data path
- Interleaving

Beta ISA Assembly

- Instructions and formats
- Assembler: text file → 0's 1's
- Assembly language
  - ↳ values, symbols, labels, macros
  - ↳ pseudo instructions
  - ↳ dot notation
  - ↳ defining memory values
    - ↳ (LONG)

Stacks + Procedures

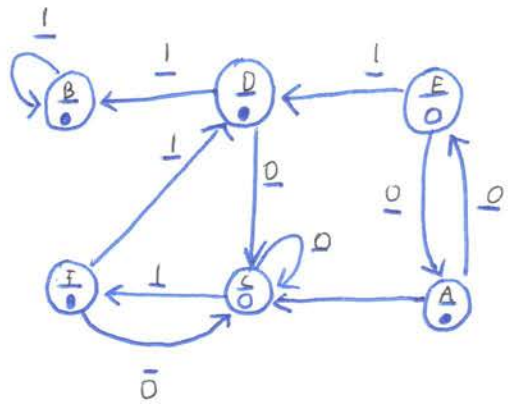
- Just a block of memory
- SP, BP help keep track
- SP(R29) = next usable loc
- BP(R27) = first stack loc
- Stack Macros
  - ↳ PUSH, POP, (DE)ALLOCATE
- Linkage contract: LP = R29
  - ↳ use calling procedure

Quiz stuff
- Hard
- Do stacks + procedures online practice problems

Finite state machine

Sample problem F2013 Q2 P2

State	Input	Next St.	Output
A	0	E	[illegible]
A	1	C	[illegible]
B	0	C	[illegible]
B	1	B	[illegible]
C	0	C	0
C	1	F	0
D	0	C	[illegible]
D	1	B	[illegible]
E	0	A	0
E	1	D	0
F	0	C	[illegible]
F	1	D	[illegible]



1) Fill in the blanks

2) Is there a synchronizing sequence

↳ Sequence that reaches 0 from any state!

Sequence : 10

3) Merge?

Equivalence: Same inputs produce same result

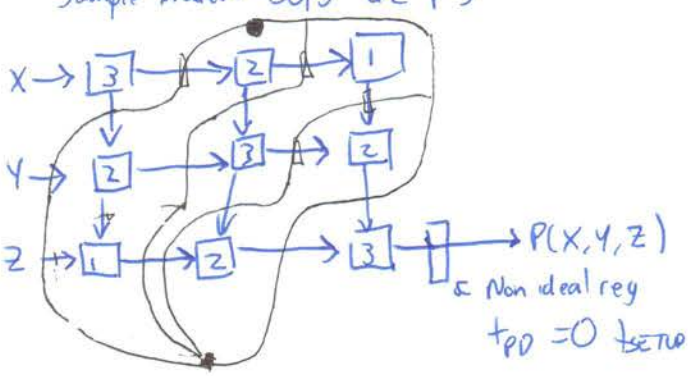
B - D : Same output 1→B 0→C

BD - F :

Merge: BDF

Pipelining

Sample Problem 2013 Q2 P3



$t_{PD} = 0$ $t_{SETUP} = 1$

1) Pipeline ~~through~~ to maximize throughput ~~input~~

max throughput = 1/min ~~all~~ delay for output = 1/4 (3+1)

- isolate components with biggest PD ← Bottle neck

latency 5 × 4 = 20

5 stage pipeline

longest path input → output

has to go through 5 pregisters

No pipelining

Throughput: 1/14

Latency: 14

with p-lining

Throughput: 1/4

latency: 20

Beta ISA Assembly

```
        .=0 // Next line is 0
0       ADDC(r31, N, r0)      // r0 ← 0 + 2000
4       LD(r0, 8, r1)
8       SRAC(r1, 4, r2)
C       ST(r2, 4, r0)
10      HALT()

        .=2000
2000    N: LONG(0x12345678)
2004       LONG(0x DEADBEEF)
2008       LONG(0x EDEDEDED)
200C       LONG(0x 0000004)
```

Stacks and Procedures

6.004 Quiz II Practice

Fall 2012 Q3 P2

```
int ilog2(unsigned x) {
    unsigned y
    if (x==0) return 0
    else {
        y = x>>1
        return ilog2(y)+1
    }
```

```
ilog2: PUSH(LP)
       PUSH(BP)
       MOVE(SP, BP)
       ALLOC(1)
       PUSH(R1)

       LD(BP,-12,R0)
       BEQ(R0, rtn, R31)

       LD(BP,-12,R1)
       SHRC(R1,1,R1)
       ST(R1,0,BP)

       LD(BP,0,R1)
       PUSH(R1)                 168
       BR(ilog2, LP)
168    DEALLOC(1)
16C    ADD(R0,1,R0)
rtn: 170
174    POP(R1)
XXX: 178
       DEALLOC(1)
       MOVE(BP,SP)
       POP(BP)
       POP(LP)
       JMP(LP)
```

STACK

	5	X	callered pushes into stack
	11C	LP	
BP →	4	BP	
10	2	Y	//ALLOC 1
	0	R1	
10	2	X	
1C	168	LP	
20 BP →	10	BP	
24	1	Y	
28	2	R1	
2C	1	X	
30	168	LP	
34 BP →	24	BP	
38	0	Y	
3C	1	R1	
40	0		
SP →			

call ilog2(5)

so: x = 5
y = 5>>1 = 2
return ilog2(2)+1 //=3

x=2
y=2>>1 = 1
return ilog2(1)+1

x=1
y=1>>1=0
return ilog2(0)+1

x=0
return 0

R0 = 1
SP = 40
BP = 38
LP = 168
XXX = 178

3/13/15

6.004 Recitation

Fernando Trujano

Stacks and Procedures
↳ Functions

Stacks

- push - putting thing in stack
- pop - remove most recently pushed element
- stored in dedicated part of RAM

Activation Records/stack Frame
↳ part of stack that "belongs" to particular instance of function
- Holds all info needed by a single procedure call
  - input arguments
  - return addressed
  - local variables / registers

PC (Program counter): which address in memory we are currently interpreting as an instruction

SP (Stack pointer): Points to next open spot on stack

Push (Rx): Mem[SP] ← Rx
SP ← SP+4

Pop SP ← SP-4
Rx ← Mem[SP]

Dealloc (n): SP ← SP-(4*n)
↳ dealocate

BP (Base pointer): Points to beginning of current activation record, value is the address of previous base pointer

LP (Linkage pointer): old PC

caller - doesn't use R0 for anything important
- push arguments in stack on reverse order
- BR (F, LP)

callee - push LP
- push BP
- move BP up to SP
- save registers do procedure -> put answer R0
- restore
- restore all registers
- move SP to BP
- Pop BP
- Pop LP
JMP (LP)

Example

.0x400

stack

```
def sum(n)
    if n==0 return 0
    return 1 + sum(n-1)
sum(2)
```

```
.=0x400        //code

PUSH (LP)
PUSH (BP)
MOVE (SP, BP)
PUSH(R1)
PUSH(R2)
LD (BP,-12,R0)// Load arg1
BNE(R0, skip, R31)  // R0 is already 0
ADDC (R1, 0, R0)
```

save registers

2/3

```
restore registers
    POP(R2)
    POP(R1)
return
    MOVE(BP,SP)      //double check
    POP(BP)
    POP(LP)
    JMP(LP)

skip:  SUBC(R1,1,R2)
       PUSH(R2)
       BR(SUM,LP)
       DEALLOC(1)      // R0 contains answer
       ADD(R1,R0,R0)

restore registers
    POP(R2)
    POP(R1)
    MOVE(BR,SP)
    POP(BP)
    POP(LP)
    JUMP(LP)

.=0x0
0x0    ADDC(R31,2,R1)// R1 ← 2
0x4    PUSH(R1)                  // Push takes 2 lines (it's a macro)
0x8
0xC    BR(sum,LP)
0x10   DEALLOC(1)
0x14   HALT()
```

4/3/15

6.004 Recitation

Fernando Trujano

Cache

registers - scratch space, arithmetic manipulation

cache - small but fast, near CPU. - Store things we might use again.

RAM - stack, program instructions, variables

Disk - everything else

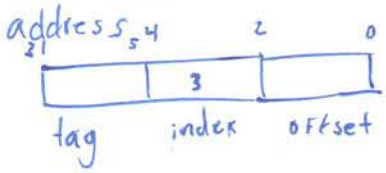
α = hit rate (probability of cache hit)

$$t_{avg} = \alpha t_{cache} + (1-\alpha)(t_{cache} + t_{mem})$$

$\Rightarrow$

★ $$\alpha = 1 - \frac{t_{avg} - t_{cache}}{t_{mem}}$$

Direct Mapped



tag: unique ID
index: what line to look at
offset: block size

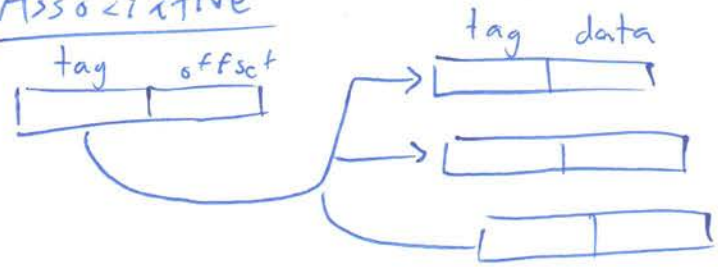
Ex: address 0x2D10

0x2D 0001 0000

index = 4 offset

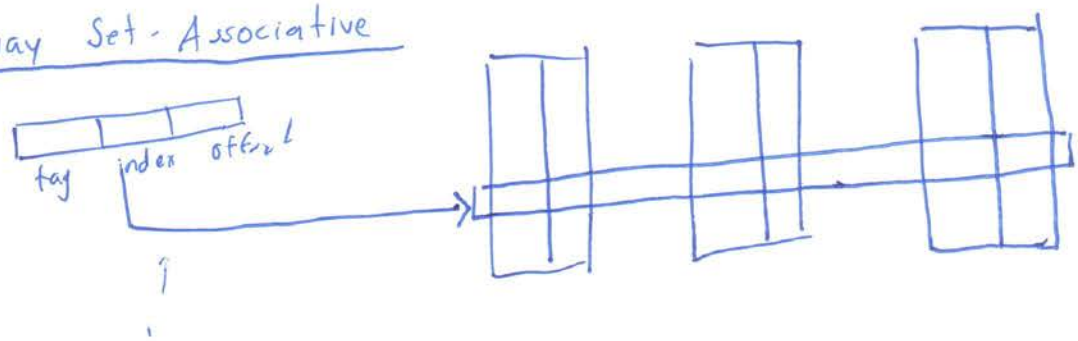
- simple to implement :)
- no way to resolve collisions :(

Fully Associative



- N tag comparator
- collision resolution
- LRU (least-recently-used) replacement

N-way Set-Associative



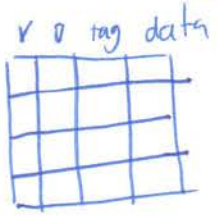
Replacement Strategies

LRU: Least-recently-used N! orderings, for each line $O(\log N!)$
$\approx O(N \log N)$

FIFO: First in First out

random: If full, randomly kick out something in cache

Write Strategies



v = valid bit: means it contains the value corresponding to the tag.

D = dirty bit: means the value in the cache is different from mean

write-through: on cache write, always write to mem (stalls CPU)
↳ dirty bit never set to one

write-behind: writes buffered in order in the background (sometimes has to wait a long time)

write-back: dirty bits, only write to mem when evicting cache line.

Examples

	tag	data
3	0x7	0x11111111
2	0x7	0x22222222
1	0x2	0xFFD00011
0	0x1	0x11213452

block size = 4

0x24

0x10

4 lines -> need 2 bits

index

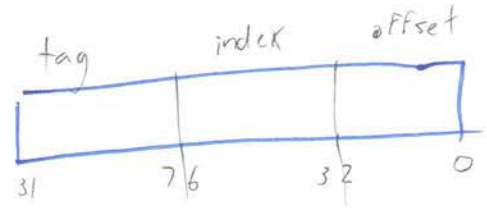
offset - $\log 4 = 2$

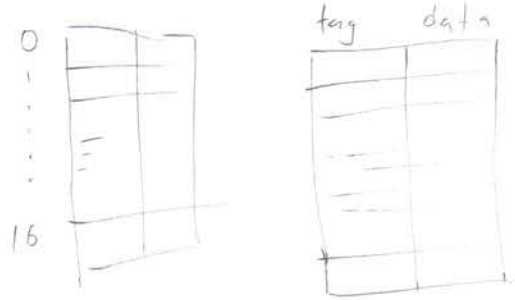
28	2	2

tag, the rest

2-way associative, 16 lines each, 2 word block size

```
. = 0x2000
CMOVE(0x1000, R0)   -> 2000
loop: CD(R0, 0, R1)
      ADDC(R0, 4, R0)
      BNE(R0, loop)
      HALT()
```





Hitrate?

every-instruction 1 lookup

LD and ST take 2 lookups

6.004 Quiz IV Review

entries in page table depends only of VPN

bits in each entry of page table depends only on PPN

4/10/15

6.004 Recitation

Fernando Trujano

β Pipelining I

Stages	Control Signal	Output	Mem
IF = "instruction fetch"	PCSEL	ID	
RF "register file"	ASEL BSEL	A B WD	
ALU	ALUFN	Y=F(A,B) MA=Y	
MEM	MWR	Y	
WB "write back"	WERF WASEL	Mem[MA]	

Data Hazard - Read a changed R that hasn't been written yet

ADD(R0,R1,R3)
SUB(R3,R2,R1)

Solution:

Stall - wait

Bypass - get value before WB

6.004 Recitation

Fernando Trujar

Control Hazards and Data Hazards

Control Hazards

- which branch to take

1) Stall: add NOPs in software or hardware (mux)
2) Bypass
3) Guess - branch predictor

Data Hazard

1) Stall
2) Bypass
3) Rewrite / reorder software to do useful things while writing

Toolbox

- Stall: Repeat instructions in IF/RF stages instead of reading a new one and add NOP
- Bypass: send data from ALU stage or later and send up to RF
- Annul: turn an instruction that shouldn't be executed into a nop
- Flush pipeline: annul all previous instructions in the pipeline

Exceptions or interrupts

Illop → RF stage

(mouse, keyboard, etc) → at any time

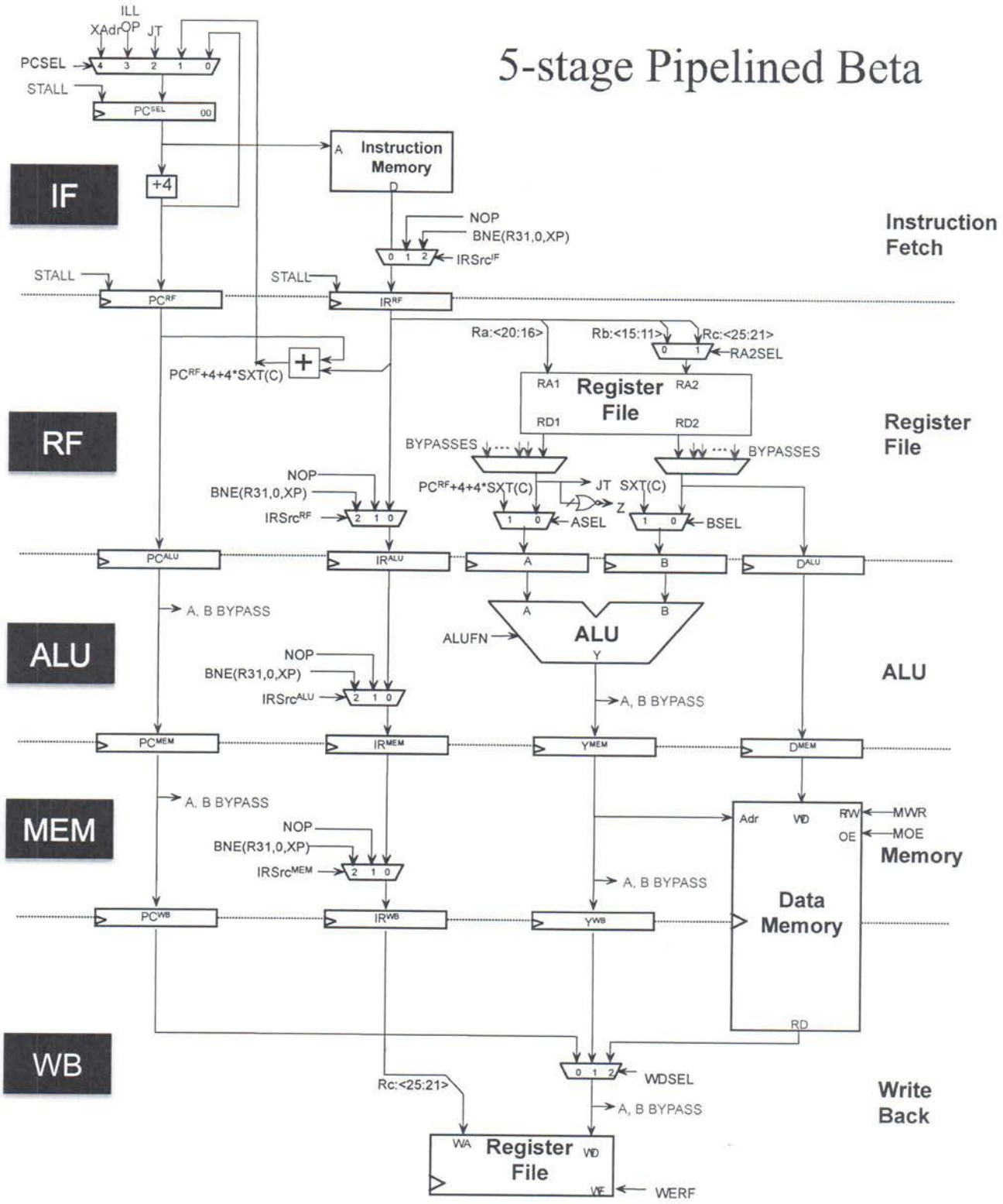
- halt execution, save PC+4 in XP, handle interrupt / exception
  ↳ PXP = PC+4
- Force instruction to be BNE (R31, 0, XP)
  ↳ same logic as annul

Exception => Flush pipeline

Interrupt => No need to flush

ADD	SUB	BNE	Interrupt
	ADD	SUB	BNE
		ADD	⋮

5-stage Pipelined Beta



Examples:

```
loop:  LD (R31, status, R0)
       BEQ (R0, loop, R31)
       ADD (R0, R1, R2)
       SUB (R2, R1, R0)
       XOR (R1, R2, R0)
```

predictor was wrong

stall - IF and RF same

IF	LD	BEQ	ADD	ADD	ADD	LD
RF		LD	BEQ	BEQ	BEQ	NOP_3
ALU			LD	NOP_1	NOP_2	BEQ
MEM				LD	NOP_1	NOP_2
WB					LD	NOP_1

stall

annul

BEQ moved on, must have Bypassed

1) What are $NOP_{1,2,3,4,5}$ used for.
2) Label bypass arrows.
3) Write next two cycles if don't take branch

stall

BEQ	ADD	ADD	ADD
LD	BEQ	BEQ	BEQ
NOP_3	LD	NOP_4	NOP_5
BEQ	NOP_3	LD	NOP_4
NOP_2	BEQ	NOP_3	LD

check if these operations use a register that will be modified later in the pipeline.

IF so → bypass to the RF stage.

SUB	XOR
ADD	SUB
BEQ	ADD
NOP_5	BEQ
NOP_4	NOP_5

6.004 Quiz 3 Review

Pipelining

1) Program takes 1000 cycles on unpipelined det a
   takes N cycles to reach last instruction

   - N can't be less than 1000
   - N can be ≥ 1000 → if stalls || annuls

2) New instruction 5-stage pipe line: JMZ - Jump is memory zero.
   How many delay slots would follow

Stages

IF	RF	ALU	MEM	WB
X	X	X	X	JMZ

← branch decision

$\boxed{4}$

3) How many Fetched instructions are annulled following a BNE instruction

IF	RF
X	BR

← Branch taken

Branch Taken: 1
Not taken: 0

5 min