Midterm Review

Introduction to Computer Systems, Fall 2022

Instructor: Travis McGaha

TAs:

Ali Krema	Andrew Rigas	Anisha Bhatia
Audrey Yang	Craig Lee	Daniel Duan
David LuoZhang	Eddy Yang	Ernest Ng
Heyi Liu	Janavi Chadha	Jason Hom
Katherine Wang	Kyrie Dowling	Mohamed Abaker
Noam Elul	Patricia Agnes	Patrick Kehinde Jr.
Ria Sharma	Sarah Luthra	Sofia Mouchtaris

When poll is active, respond at **pollev.com/tqm**

Text **TQM** to **37607** once to join

Midterm Review: Choose what we do next

- Binary & 2C practice questions
- Floating Point practice question
- CMOS, PLA, Gate circuits practice questions
- Gate Delay practice questions
- LC4 Programming practice question
- Control Signals practice quetsion
- General Q & A

Logistics

- Midterm Exam: **This Wednesday "in lecture"**
  - Details released on the course website
- Midterm Review in recitation
  - Tuesdays 6:30 – 8 pm @ Moore 100a
  - Wednesdays 12 - 1:30 pm @ Moore 100c
- Instructor OH shifted to 1:30 – 4:30 pm on Wednesday
- HW03 Sample Solutions (and grades probably) posted tonight

Binary & 2C

- There are about 195 students in the class and 22 staff. If we wanted to assign each of these individuals a unique numerical ID, how many bits would each ID need to be?

- Translate:
  - -1 into 8-bit 2C
  - 7 into 4-bit 2c
  - 5 into 3bit unsigned

Floats

- A common way of checking for equality between floats in code is to see if the difference between the two floats exceeds a certain magnitude instead of checking for exact equality. E.g.:

```
bool float_equals(float a, float b){
  double delta = 0.01f;
  if (abs(a - b) < delta) {
    return true;
  }
  return false;
}
```

- Why is this the case?

CMOS, PLAS, GATES

- Create a circuit that takes in an unsigned 4-bit input I ($I_3I_2I_1I_0$), and outputs a 1 if and only if the 4-bit input is a non-zero multiple of 7
  - List the outputs that result in a 1 for the output
  - Create a corresponding CMOS circuit
    - Can assume you have the inverses of the Input bits
  - Create a corresponding PLA circuit
  - Create a corresponding gate level non-PLA circuit

CMOS, PLAS, GATES

- Create a circuit that takes in an unsigned 4-bit input I ($I_3I_2I_1I_0$), and outputs a 1 if and only if the 4-bit input is a non-zero multiple of 7
  - List the outputs that result in a 1 for the output
  - 7 (0b0111) and 14 (0b1110)

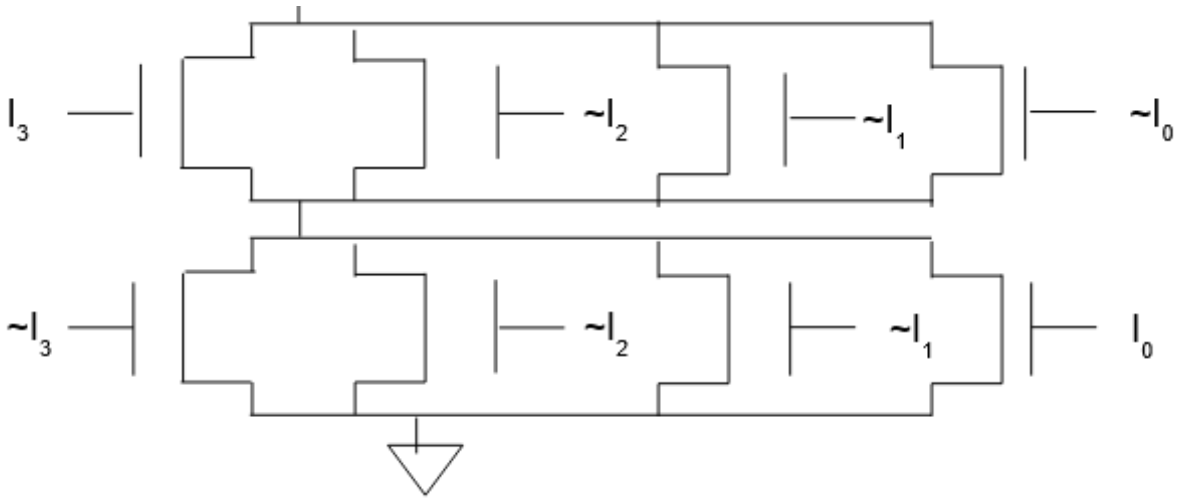
CMOS

- Create a circuit that takes in an unsigned 4-bit input I ($I_3I_2I_1I_0$), and outputs a 1 if and only if the 4-bit input is a non-zero multiple of 7. You can assume you have inverse of the input signals.
  - Overall Expression: $(\sim I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, I_0) \mid (I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, \sim I_0)$

CMOS Strategy 1 (Starting with PDN)

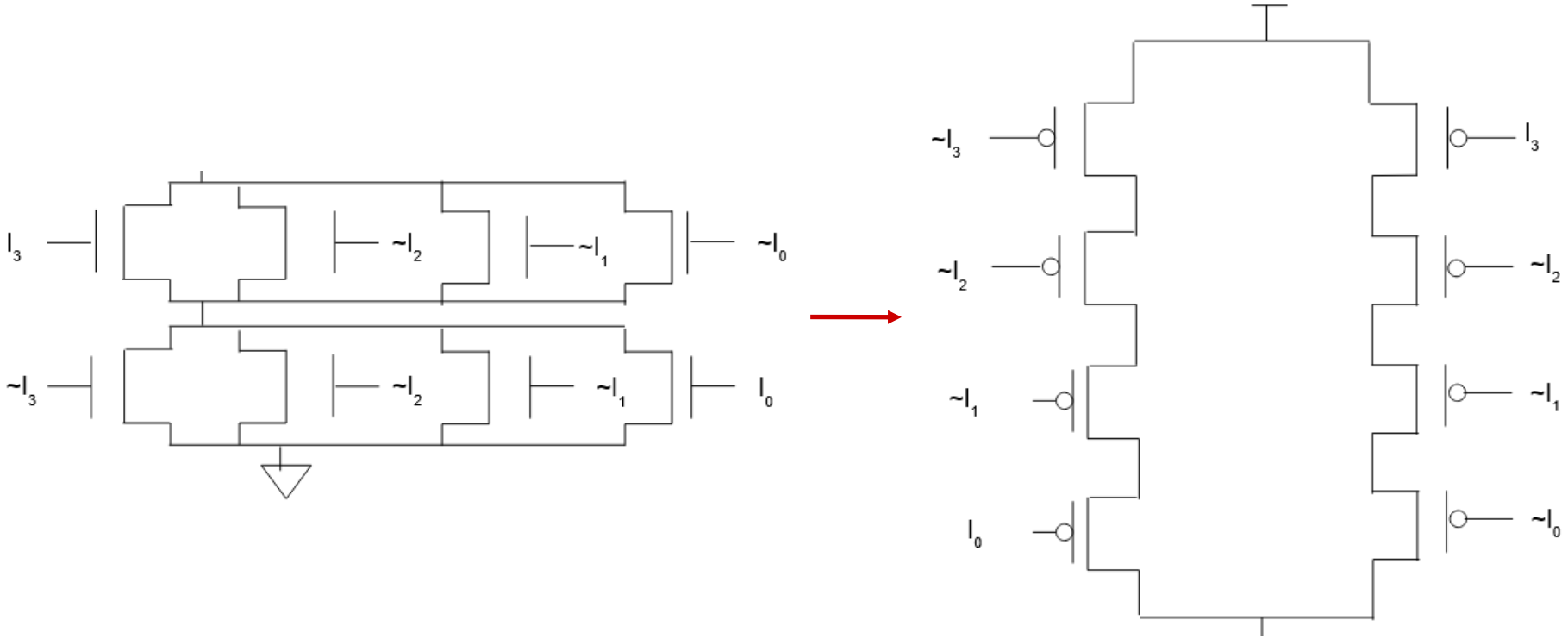
- Overall Expression: $(\sim I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, I_0) \mid (I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, \sim I_0)$
- PDN Expression:
  - $\sim((\sim I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, I_0) \mid (I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, \sim I_0))$ // negate
  - $\sim(\sim I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, I_0) \,\&\, \sim(I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, \sim I_0)$ // De Morgan's
  - $(I_3 \mid \sim I_2 \mid \sim I_1 \mid \sim I_0) \,\&\, (\sim I_3 \mid \sim I_2 \sim I_1 \mid I_0)$ // De Morgan's

- Translated to PDN:



CMOS Strategy 1 (Starting with PDN)

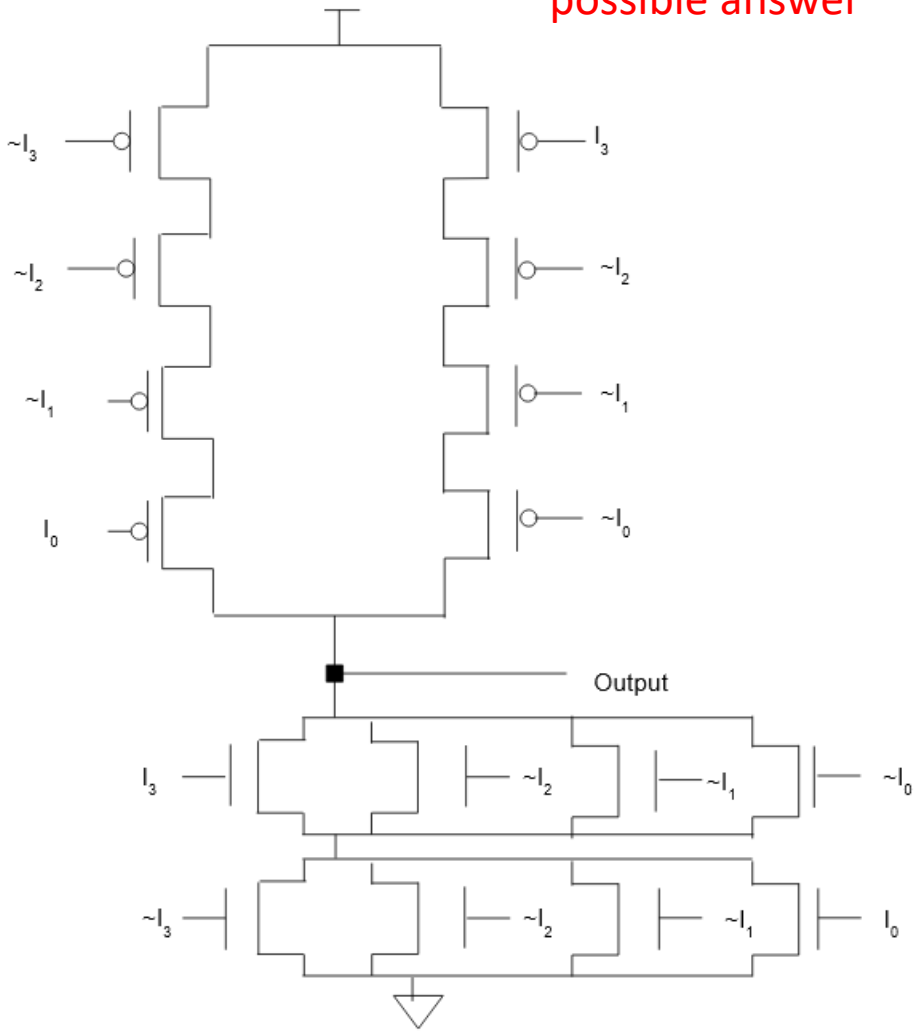
- Flip PDN into PUN:



CMOS Strategy 1 (Starting with PDN)

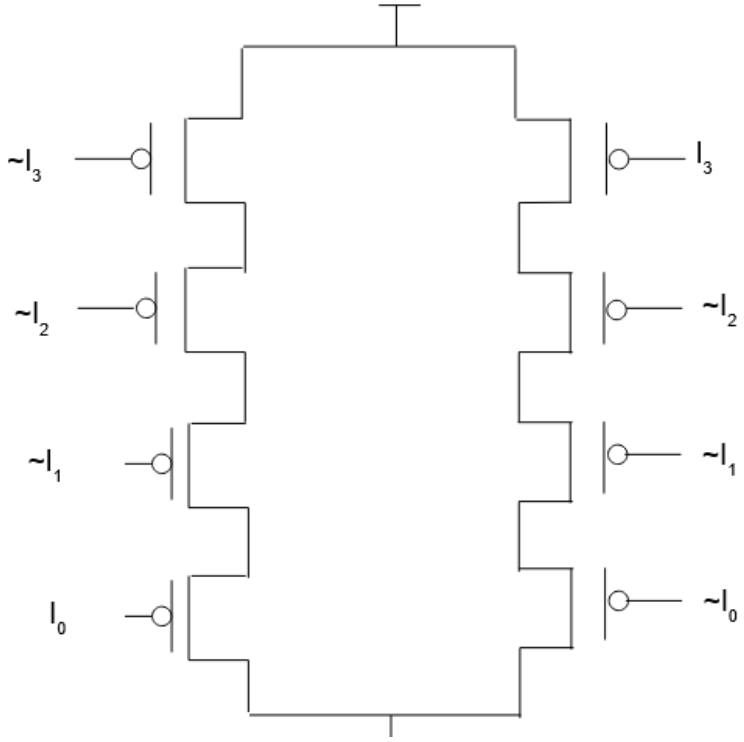
- Connect PDN and PUN:

Not the only possible answer



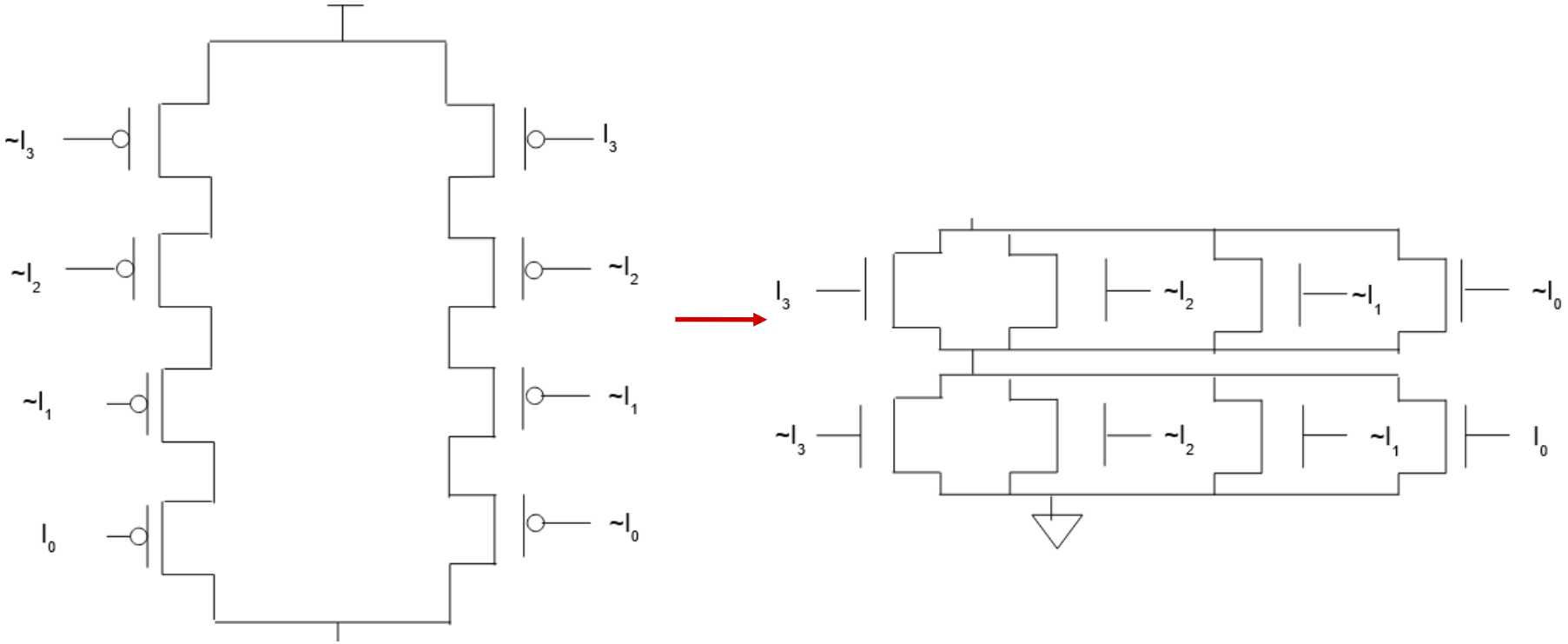
CMOS Strategy 2 (Starting with PUN)

- Take the original expression:
  - $(I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, \sim I_0) \mid (\sim I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, I_0)$
- Translate it directly into PDN but add a negation to each input
  - This is because PMOS transistors are "naturally negating"
  - E.g., $\sim I_3$ becomes $\sim\sim I_3 == I_3$



CMOS Strategy 2 (Starting with PUN)

- Flip PUN to get PDN



PLA

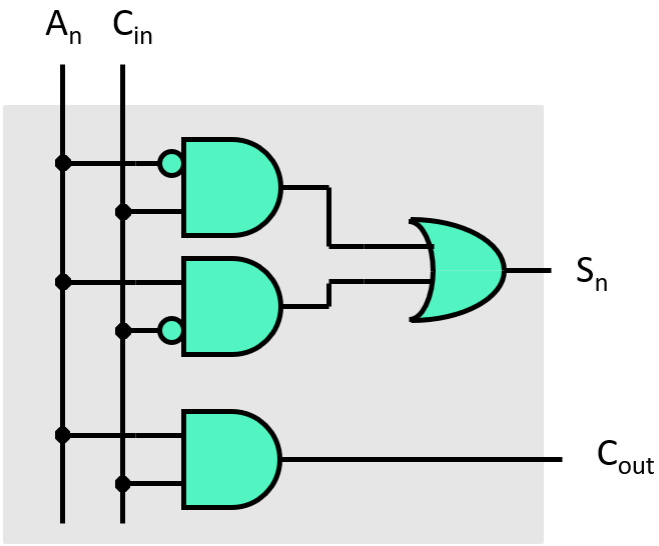
- Create a circuit that takes in an unsigned 4-bit input I ($I_3I_2I_1I_0$), and outputs a 1 if and only if the 4-bit input is a non-zero multiple of 7

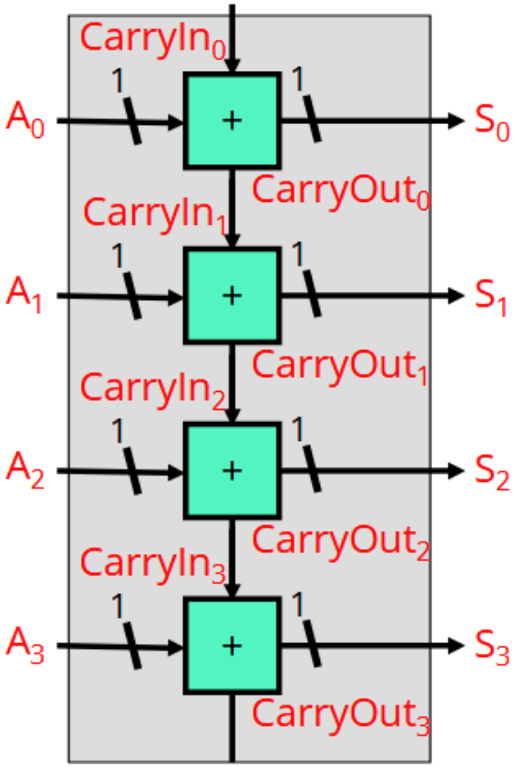
Non-PLA

- Create a circuit that takes in an unsigned 4-bit input I ($I_3I_2I_1I_0$), and outputs a 1 if and only if the 4-bit input is a non-zero multiple of 7

Gate delay pt.1

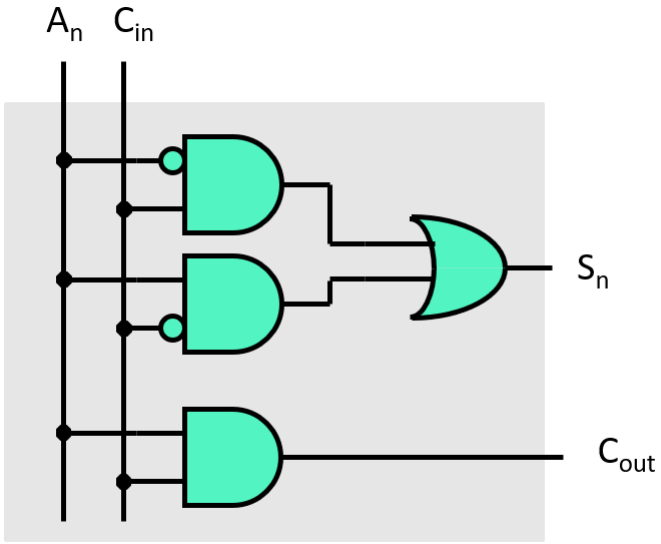
- Given the 4-bit incrementor that we created in lecture, how long do we have to wait to make sure that the output of the incrementor matches the input?
  - Assume that each gate has a 1ns delay
  - You can ignore delay from inverters

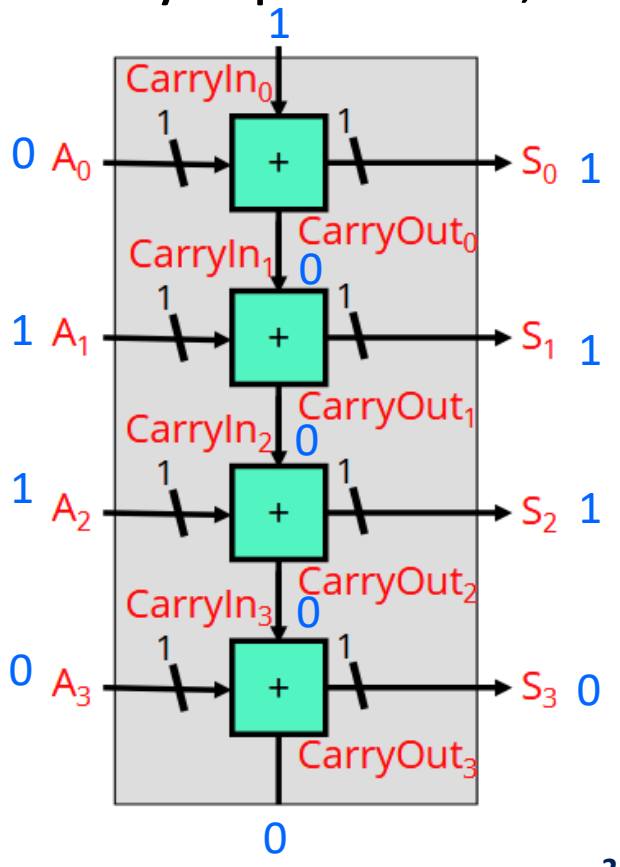




Gate delay pt.2

- The 4-bit incrementor that we created in lecture is currently in a stable state showing the output of 1 + 0110. If the A input signals were to simultaneously flip to 0111, what would all signals be after 2ns
  - Assume that each gate has a 1ns delay
  - Ignore delays from inverters
  - $CarryIn_n$ stays the same





LC4 Programming

- Write an asm program that you can assume has:
  - R0 = addr of start of array
  - R1 = length of array that R0 refers to
  - R2 = addr of start of destination array (you can assume there is enough memory locations "free" to store the resulting array)
  - R3 = Filter integer
- Your asm should make a copy of the array referred to by R0 into the memory referred to by R1. However, you should not include any value that is less than or equal to the filter integer put in R3.
- You are free to update the registers as long as the memory referred to by R2 is properly updated and the length of the new array is stored in R4 by the end.

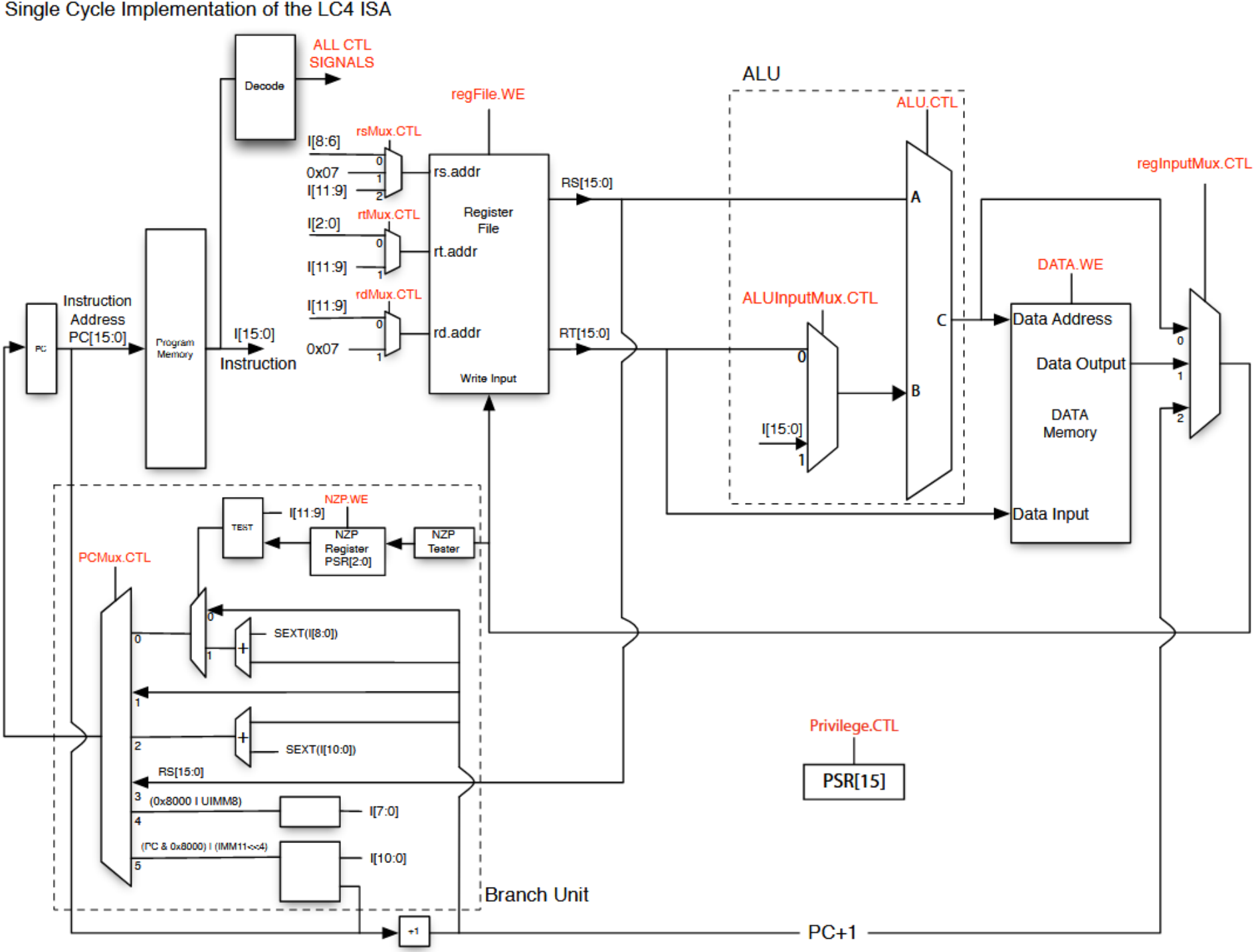
Control Signals

- Bit-wise XOR operations can be used to test for equality. If A XOR B is 0, then the two values contained the same bit pattern. As a result, Travis wants to add the CMPX instruction, which updates the NZP based off of the XOR of two register values.
  - Mnemonic: CMPX Rs, Rt
  - Semantics: NZP = sign (Rs ^ Rt)
  - Encoding: 0011 ttts ssxx xxxx
- What are the control signals for this instruction?

General Questions & Answers

- Take questions/requests from students

Mnemonic	Semantics	Encoding
NOP	PC = PC + 1	0000 000x xxxx xxxx
BRp <Label>	(P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 001i iiii iiii
BRz <Label>	(Z) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 010i iiii iiii
BRzp <Label>	(Z|P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 011i iiii iiii
BRn <Label>	(N) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 100i iiii iiii
BRnp <Label>	(N | P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 101i iiii iiii
BRnz <Label>	(N|Z) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 110i iiii iiii
BRnzp <Label>	(N|Z|P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 111i iiii iiii
ADD Rd Rs Rt	Rd = Rs + Rt	0001 ddds ss00 0ttt
MUL Rd Rs Rt	Rd = Rs * Rt	0001 ddds ss00 1ttt
SUB Rd Rs Rt	Rd = Rs - Rt	0001 ddds ss01 0ttt
DIV Rd Rs Rt	Rd = Rs / Rt	0001 ddds ss01 1ttt
ADD Rd Rs IMM5	Rd = Rs + sext(IMM5)	0001 ddds ss1i iiii
MOD Rd Rs Rt	Rd = Rs % Rt	1010 ddds ss11 xttt
AND Rd Rs Rt	Rd = Rs & Rt	0101 ddds ss00 0ttt
NOT Rd Rs	Rd = ~Rs	0101 ddds ss00 1xxx
OR Rd Rs Rt	Rd = Rs | Rt	0101 ddds ss01 0ttt
XOR Rd Rs Rt	Rd = Rs ∧ Rt	0101 ddds ss01 1ttt
AND Rd Rs IMM5	Rd = Rs & sext(IMM5)	0101 ddds ss1i iiii
LDR Rd Rs IMM6	Rd = dmem[Rs + sext(IMM6)]	0110 ddds ssii iiii
STR Rt Rs IMM6	dmem[Rs + sext(IMM6)] = Rt	0111 ttts ssii iiii
CONST Rd IMM9	Rd = sext(IMM9)	1001 dddi iiii iiii
HICONST Rd UIMM8	Rd = (Rd & 0xFF) | (UIMM8 << 8) [1]	1101 dddx uuuu uuuu
CMP Rs Rt	NZP = sign(Rs - Rt) [2]	0010 sss0 0xxx xttt
CMPU Rs Rt	NZP = sign(uRs - uRt) [3]	0010 sss0 1xxx xttt
CMPI Rs IMM7	NZP = sign(Rs - IMM7)	0010 sss1 0iii iiii
CMPIU Rs UIMM7	NZP = sign(uRs - UIMM7)	0010 sss1 1uuu uuuu
SLL Rd Rs UIMM4	Rd = Rs << UIMM4	1010 ddds ss00 uuuu
SRA Rd Rs UIMM4	Rd = Rs >>> UIMM4	1010 ddds ss01 uuuu
SRL Rd Rs UIMM4	Rd = Rs >> UIMM4	1010 ddds ss10 uuuu
JSRR Rs	R7 = PC + 1; PC = Rs	0100 0xxs ssxx xxxx
JSR <Label>	R7 = PC + 1; PC = (PC & 0x8000) | ((IMM11 offset to <Label>) << 4)	0100 1iii iiii iiii
JMPR Rs	PC = Rs	1100 0xxs ssxx xxxx
JMP <Label>	PC = PC + 1 + (sext(IMM11) offset to <Label>)	1100 1iii iiii iiii
TRAP UIMM8	R7 = PC + 1; PC = (0x8000 | UIMM8); PSR [15] = 1	1111 xxxx uuuu uuuu
RTI	PC = R7; PSR [15] = 0	1000 xxxx xxxx xxxx


Single Cycle Implementation of the LC4 ISA
Decode
ALL CTL SIGNALS
regFile.WE
ALU
ALU.CTL
rsMux.CTL
I[8:6]
0x07
I[11:9]
rtMux.CTL
I[2:0]
I[11:9]
rdMux.CTL
I[11:9]
0x07
rs.addr
Register File
rt.addr
rd.addr
Write Input
RS[15:0]
RT[15:0]
regInputMux.CTL
A
B
C
ALUInputMux.CTL
I[15:0]
DATA.WE
Data Address
Data Output
DATA Memory
Data Input
PC
Instruction Address PC[15:0]
Program Memory
I[15:0]
Instruction
NZP.WE
I[11:9]
TEST
NZP Register PSR[2:0]
NZP Tester
PCMux.CTL
SEXT(I[8:0])
SEXT(I[10:0])
RS[15:0]
(0x8000 | UIMM8)
I[7:0]
(PC & 0x8000) | (IMM11<<4)
I[10:0]
Branch Unit
+1
PC+1
Privilege.CTL
PSR[15]

Signal Name	# of bits	Value	Action
PCMux.CTL	3	0	Value of NZP register compared to bits I[11:9] of the current instruction if the test is satisfied then the output of TEST is 1 and NextPC = BRANCH Target, (PC+1) + SEXT(IMM9); otherwise the output of TEST is 0 and NextPC = PC + 1
		1	Next PC = PC+1
		2	Next PC = (PC+1) + SEXT(IMM11)
		3	Next PC = RS
		4	Next PC = (0x8000 | UIMM8)
		5	Next PC = (PC & 0x8000) | (IMM11 << 4)
rsMux.CTL	2	0	rs.addr = I[8:6]
		1	rs.addr = 0x07
		2	rs.addr = I[11:9]
rtMux.CTL	1	0	rt.addr = I[2:0]
		1	rt.addr = I[11:9]
rdMux.CTL	1	0	rd.addr = I[11:9]
		1	rd.addr = 0x07
regFile.WE	1	0	Register file not written
		1	Register file written: rd.addr indicates which register is updated with the value on the Write Input
regInput.Mux.CTL	2	0	Write Input = ALU output
		1	Write Input = Output of Data Memory
		2	Write Input = PC + 1
NZP.WE	1	0	NZP register not updated
		1	NZP register updated from Write Input to register file
DATA.WE	1	0	Data Memory not written
		1	Data Input written into location on Data Address lines
Privilege.CTL	2	0	PSR[15] = 0 – Clear privilege bit
		1	PSR[15] = 1 – Set privilege bit
		2	PSR[15] unchanged – no change to privilege bit

Signal Name	# of bits	Value	Action
ALU.CTL	6		
Arithmetic Ops		0	C = A + B : Addition
		1	C = A * B : Multiplication
		2	C = A - B : Subtraction
		3	C = A / B : Division
		4	C = A % B : Modulus
		5	C = A + SEXT(B[4:0])
		6	C = A + SEXT(B[5:0])
Logical Ops		8	C = A AND B : Bitwise Logical Product
		9	C = NOT A: Bitwise Negation
		10	C = A OR B: Bitwise Logical Sum
		11	C = A XOR B: Bitwise Exclusive OR
		12	C = A AND SEXT(B[4:0])
Comparator Ops		16	C = signed-CC(A-B) [-1, 0, +1]
		17	C = unsigned-CC(A-B) [-1, 0, +1]
		18	C = signed-CC(A-SEXT(B[6:0])) [-1, 0, +1]
		19	C = unsigned-CC(A-SEXT(B[6:0])) [-1, 0, +1]
Shifter Ops		24	C = A << B[3:0] : Shift Left Logical
		25	C = A >>> B[3:0] : Shift Right Arithmetic
		26	C = A >> B[3:0] : Shift Right Logical
Constant Ops		32	C = SEXT(B[8:0])
		33	C = (A & OxFF) | (B[7:0] << 8)