University of California
College of Engineering
Department of Electrical Engineering
and Computer Sciences

J. Rabaey

G. Alexandrov, N. Narevsky, V. Iyer

MoWe 4-5:30pm

Mo, Oct. 2, 6:00-7:30pm

EECS 151/251A: SPRING 17—MIDTERM 2

SOLUTIONS

NAME	Last First

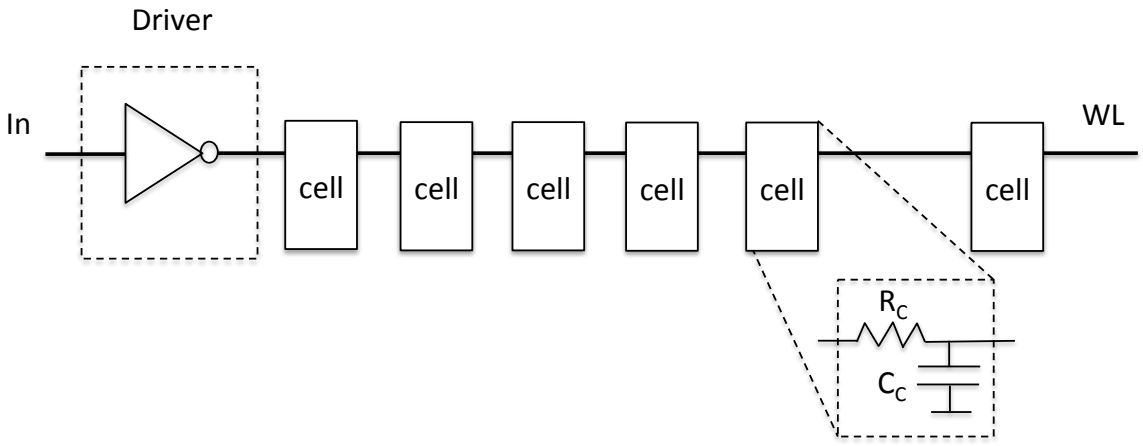
SID	

Problem 1 (16):
Problem 2 (17):
Problem 3 (15):

Total (48)	

[PROBLEM 1] Logic and Wire optimization (16 + 1 Pts)

a) A designer at a memory company is in charge of developing the circuitry to drive the wordline of an SRAM module as fast as possible. An initial design is shown below. It consists of an inverting driver and a wordline wire connecting to 256 SRAM cells. The contribution of each cell to the wordline can be modeled as a series resistance R_C of 5 Ω and a load capacitance C_C of 2fF. Assume at first that the driver is a minimum-sized inverter with an input capacitance C_D of 5fF and a driver resistance R_D of 10 KΩ. You may assume that $\gamma = 1$.
In a first step, determine the **worst-case propagation delay of the circuit** (from *In* to any cell connected to the wordline) (5 Pts)



tp = 0.69 Rd * (Cd + 256 Cc) + 0.38 * (256 * Rc) * (256*Cc) = 3.57 + 0.25 nsec = 3.82 nsec

1 pt – ln(2)
1 pt – (Cd+256Cc)
1 pt – distributed wire model (e.g. 0.38)
2 pt – correct wire delay (proportional to 256^2)

b) One way to reduce the delay is to optimize the driver. Determine the optimal number of stages to minimize the delay. The function of the driver cannot be changed – that is, **it has to remain inverting**. (2 Pts)

F = (256 * 2 fF) / (5 fF) = 102.4

Optimal # of stages = log4(F) = 3.34

Given that the circuit has to remain inverting, 3 stages is the optimal answer

1 pt – log4(F) calculation
1 pt – 3 stages

c) For the chosen number of stages, size each stage so that the delay is minimized, and determine the new value of the delay. (3 Pts)

Optimal sizing factor: f = F^(1/3) = 4.68
Hence: stage 1: 1; stage 2: 4.68; stage 3: 21.88

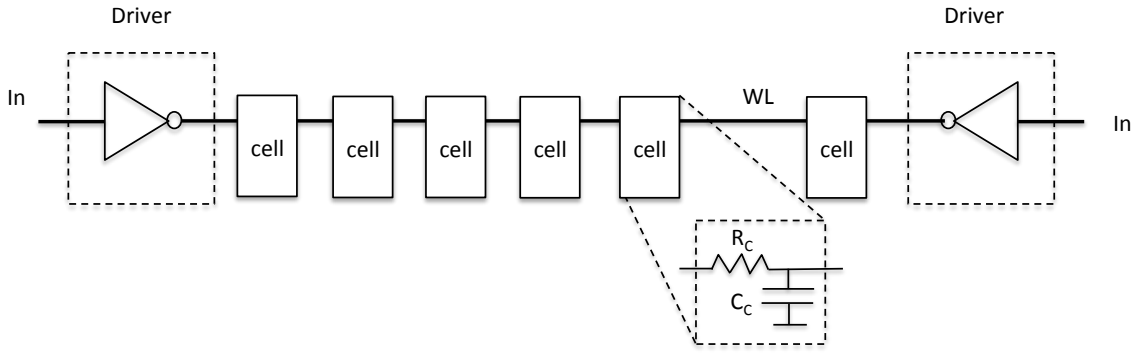
Delay: Term 2 (disturbed array) is unchanged . The first term (driver) is reduced to:
0.69* 3 * 10kOhm * (5 + 4.68 * 5) fF = 0.59 nsec

The total delay is: 0.59 nsec + 0.25 nsec = 0.84 nsec

While the first factor still dominates – the are now almost equal

2 pt – f and sizing
1 pt - delay

d) If you did the sizing right, you may observe that the overall delay is now dominated by the memory array. Our designer has come up with two approaches to address this. In a first approach, he decides to drive the WL form both sides (see Figure). Determine how this impacts the worst-case delay and determine the resulting value. To make things easy, keep the drivers the same as in part c) (3 Pts)

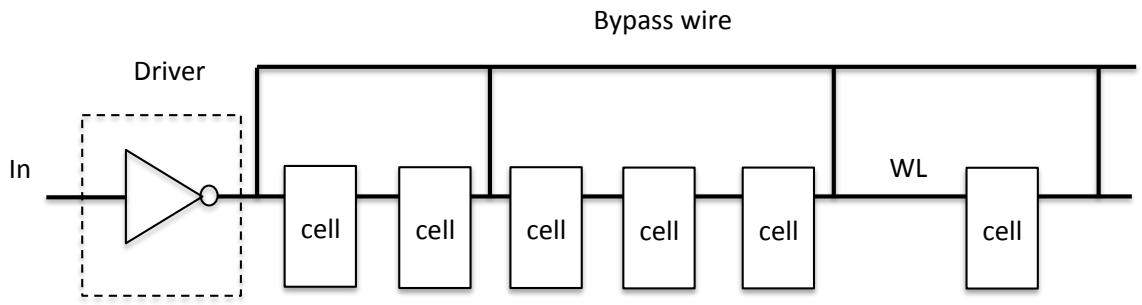


Driving from both sides reduces the "effective distributed length" of the memory array by a factor of 2. In a distributed system this effectively reduces the delay by a factor of 4. The overall delay is now:

tp = 0.59 nsec + 0.25nsec/4 = 0.65 nsec

1 pt – driver delay
2 pt – wire delay reduced by factor of 4

e) Another approach that he is considering is to put a metal wire (called a bypass) with lower resistance (located on the higher levels of the interconnect stack) in parallel with the wordline (see Figure). The bypass wire connects to the wordline every 64 cells (4 connections in total, or one connection every 64 cells). You may assume that the capacitance of bypass wire per cell is equal to the wordline, but that its resistance is 10 times lower. Determine again the impact on delay. (3 Pts)



The longest connection now would be 3 * 64 cells over the bypass wire + 32 memory cells. This translates into a delay:
0.38 * (192 * (Rc/10) * 192* Cc) + (32* Rc) * 32*Cc) = 0.018 nsec

The total delay is now : 0.61 nsec

2 pt – correct critical path
1 pt – delay calculation

f) Bonus question: Since most of the wordlines of a memory are always at 0, and only one line goes high with very cycle, it seems that the low-to-high transition is the most important one and making that one faster is the primary goal. Discuss **qualitatively** how you would change the driver characteristics to exploit this feature.
(1 Pt)

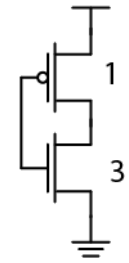
Speed up the 0->1 transition by making the PMOS of the final driver stage larger than the normal ratio, actually reducing the switching threshold of the device.

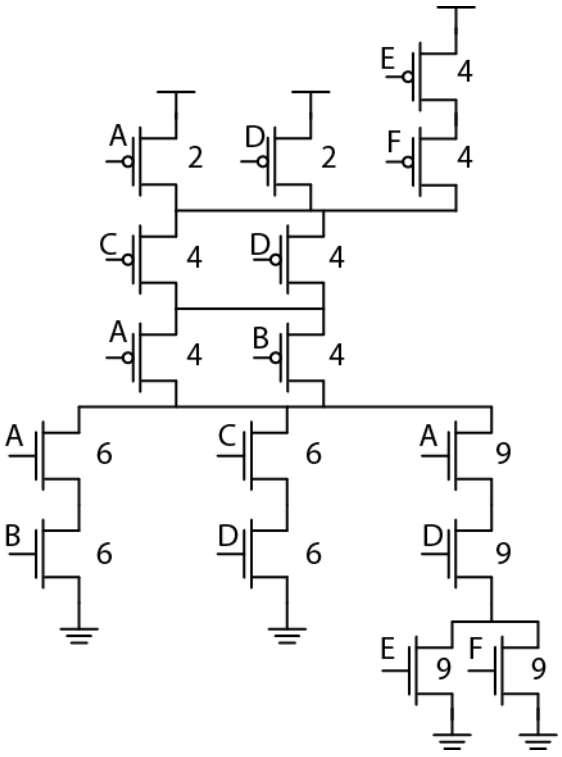
[Problem 2] Logical Effort (17 Pts)

Problem 2: Logical Effort Solutions

a) (4 pts) Implement the function $Out = \overline{(A \cdot B) + (C \cdot D) + \big(A \cdot D \cdot (E + F)\big)}$ with a complex static CMOS gate. Assuming that for this process $R_P = \frac{1}{3} * R_N$ (i.e. for the same width, a PMOS has one third the resistance of an NMOS). Size your gate such that the worst-case pull up resistance is equal to the worst-case pull-down resistance, assuming that the minimum transistor width is 1.

Ref Inverter





1 pt – correct pull-up functionality
1 pt – correct pull-down functionality
1 pt – correct pull-up sizing (multiple answers)
1 pt – correct pull-down sizing (multiple answers)
½ pt taken off if inverter had transistor size 1/3
1 pt taken off if p-n ratio was flipped

b) (3 pts) What is the logical effort of this gate from the A, C and E inputs?

For input A: $LE_A = {12 * C * R}/{4 * C * R} = {21}/{4} = 5.25$

For input C: $LE_A = {10 * C * R}/{4 * C * R} = {10}/{4} = 2.5$

For input E: $LE_A = {13 * C * R}/{4 * C * R} = {13}/{4} = 3.25$

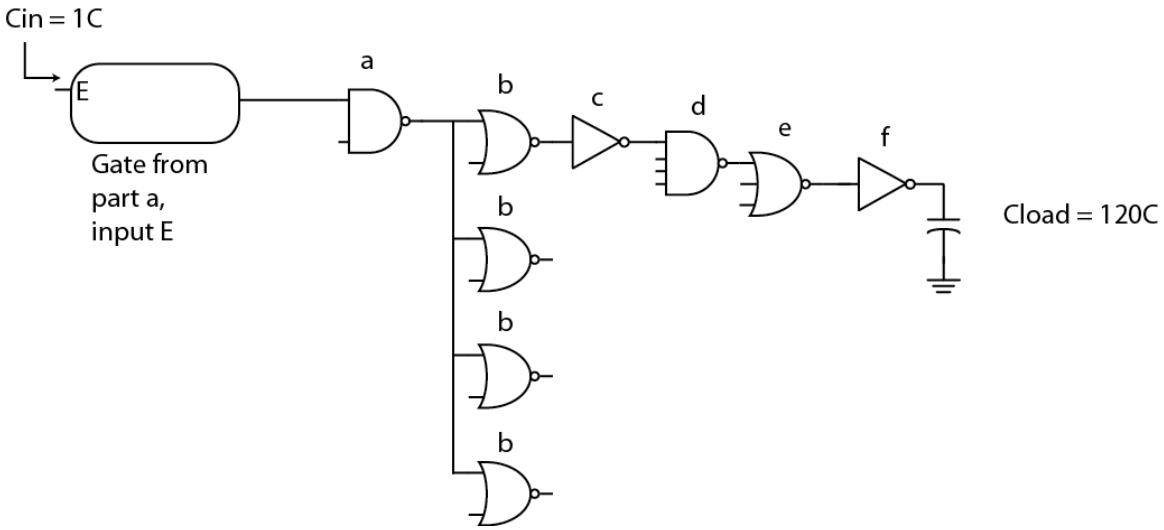
1 pt each LE

c) (2 pts) Using the same gate as previous, connect the A and B inputs together without resizing any of the transistors. What is the logical effort for this new input? Calculate the logical effort for both a rising transition as well as a falling transition

Rising transition: $LE = {31 * C * R}/{4 * C * R} = {31}/{4} = 7.75$

Falling transition: $LE = {31 * C * (7R/8)}/{4 * C * R} = {15}/{4} = 6.78125$

1 pt – correct capacitance
1 pt – correct pull-up resistance



d) (4 pts) Using the gate from part a) as well as other standard logic gates, what is the path effort of the following chain of gates? All gates are also implemented in this same technology with $R_P = \frac{1}{3} * R_N$.

$LE_E = 3.25, LE_{NAND2} = 7/4, LE_{NAND4} = 13/4, LE_{NOR2} = 5/4, LE_{NOR3} = 6/4, B = 4$

$PE = LE_E * LE_{NAND2} * B * LE_{NOR2} * LE_{INV} * LE_{NAND4} * LE_{NOR3} * F$

$= 3.25 * \frac{7}{4} * 4 * \frac{5}{4} * 1 * \frac{13}{4} * \frac{6}{4} * 120$

$= 16635.6$

1 pt – all LE. -1/4 for each one wrong
1 pt – correct branching
1 pt – correct PE
1 pt - calculation

e) (1 pts) What is the EF/stage that minimizes the delay?

$EF = \sqrt[7]{16635.6} = 4.008 \approx 4$

f) (3 pts) Compute the sizes for the gates in the chain to minimize the delay

Size	Value
a	1.23
b	0.702
c	2.24
d	8.96
e	11.02
f	29.33

$$\frac{a}{1} * LE_E = EF => a = \frac{4}{3.25} = 1.23$$
$$\frac{b}{a} * 4 * LE_{NAND2} = EF => b = \frac{4 * 1.23}{1.75 * 4} = 0.702$$
$$\frac{c}{b} * LE_{NOR2} = EF => c = \frac{4 * .702}{1.25} = 2.24$$
$$\frac{d}{c} * LE_{INV} = EF => d = \frac{4 * 2.24}{1} = 8.96$$
$$\frac{e}{d} * LE_{NAND4} = EF => e = \frac{4 * 8.96}{3.25} = 11.02$$
$$\frac{f}{e} * LE_{NOR3} = EF => f = \frac{4 * 11.02}{1.5} = 29.33$$
$$\frac{C_{load}}{f} * LE_{INV} = EF => \frac{120}{29.33} * 1 \approx 4 \ \therefore$$

½ pt each size

[Problem 3] Processor Architecture (15 pts)

a) (3 pts) Translate the following snippet of C into RISC-V assembly. Assume that 'str' is stored in memory and register x10 contains its base address. Recall that the char datatype is 1 byte. It is OK if you manipulate the str pointer itself.

```
char* str;
for (i = 0; str[i] != 0; i++) {
   str[i] = str[i] - 32;
}
```

There are a few ways to compile this snippet; here's one.

```
loop: lb x1, 0(x10)     ; str[i]
beq x1, x0, done        ; str[i] != 0
addi x1, x1, -32        ; str[i] - 32
sb x1, 0(x10)           ; str[i] = str[i] - 32
addiu x10, x10, 1       ; advance str pointer
jal loop
done: nop
```

1 pt – branch on loaded byte = 0, jump for loop
1 pt – addi -32 and store back into mem
1 pt – pointer arithmetic

b) (2 pts) Assume these instructions are executed on a single cycle RISC-V processor with asynchronous read IMEM, DMEM, and register file. The DMEM and register file have synchronous writes. You can assume that char* str = {'a', '\0'} (i.e. the string is just a single character followed by a null terminator).

How many clock cycles does your code from part a) take to execute? What is the CPI (clock cycles per instruction)?

Since this is a single-cycle processor each instruction takes a single cycle. Since str is only 1 character long, lines 1-6 will be executed, then lines 1-2, after which the snippet is complete. The sequence of instructions takes 8 clock cycles to complete and the CPI is 1.0.

1 pt – # cycles match code in (a)
1 pt – cpi = 1

c) (3 pts) Let's extend the RISC-V ISA with a custom instruction called ‘sbi’.

sbi rs1, rs2, imm will perform Mem[Reg[rs2]] <= Reg[rs1] + imm

What additional hardware and control signals are needed to implement this instruction? (describe verbally, diagram not needed) The added hardware should be minimal. Please be concise.

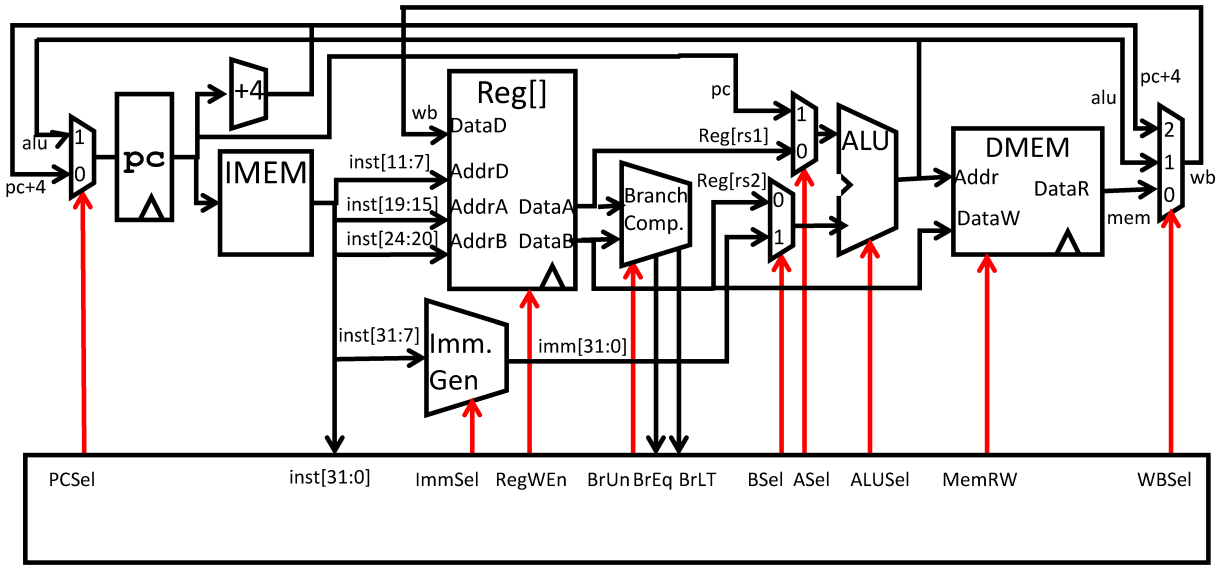
- The ALU can be used to compute the DMEM write data (Reg[rs1] + imm)
- The DMEM address can be fed straight from Reg[rs2]
- The only new hardware needed are two muxes
    - To drive the DMEM write data from the ALU output rather than Reg[rs2]
    - To drive the DMEM address from Reg[rs2] rather than the output of the ALU
- Need two 1-bit control signals for these muxes which go high when sbi is being executed

1 pt – no new adder HW
2 pt – 1 for each mux

d) (2 pts) Write the C snippet in assembly again, but using the sbi custom instruction. You can just rewrite the lines of your original code that would change.

Replace lines 3 and 4 from part a) with sbi x1, x10, -32

e) (5 pts) In the fabrication of any digital circuit, there may be manufacturing defects. One type of defect involves a signal being shorted to GND or VDD (stuck-at-zero or stuck-at-one). For the following stuck signals, specify the RISC-V instructions that will no longer work. Assume the following datapath and control signals.



i. WBSel[1:0] = 2'd1

Memory read operations won't work: **lw, lh, lhu, lb, lbu**.

Also the link operation of jumps won't work: **jal, jalr**.

1 pt – loads
1 pt – jumps

ii. ASel is stuck-at-zero

auipc will no longer work. **branch** and **jal** address calculations won't work.

jalr will still work since the new PC is calculated as rs1 + imm.

1 pt – auipc
1 pt – branch and jal
½ pt taken off if jalr isn't specified as working

iii. BSel is stuck-at-one

Any instructions that use rs2 in the ALU won't work. This includes all R-type instructions. The explicit list is: **add, slt, sltu, and, or, xor, sll, srl, sub, sra**.

Note that branches will still work as the branch computation unit is separate from the ALU. Also note stores will still work since Reg[rs2] is directly passed to the DMEM.

To get credit for this question, you should specify the explicit instruction list or claim that R-type instructions in general won't work.

1 pt – R-type instructions