EECS151/251A Midterm 2

Review Session

Zhenghan Lin & Harrison Liew

Midterm Logistics Updates

Re-submit your Zoom ID & pwd if different from midterm 1

Changelog from Exam Policy:

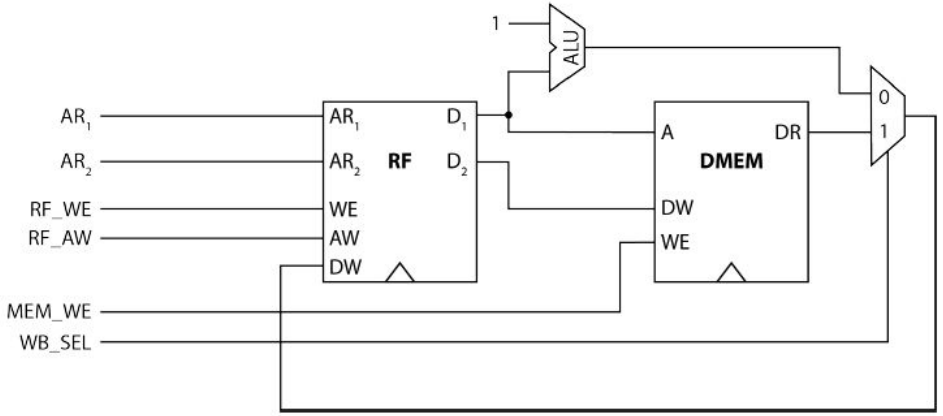
- You may start your Zoom meeting/recording, do your room tour, and set up your camera before the exam period.
  - This change is reflected in the re-ordering of some steps.
  - This also means you no longer need to record your exam printing.
  - This is to alleviate any stress of the camera setup being correct (just set it & forget it!).
- The submission time is extended to 30 minutes after the exam stop time to account for slow printing/scanning.
  - Scanning no longer needs to be recorded, but your screen share shall remain active until after you submit your exam to Gradescope. Again, this is to avoid moving your camera until you are finished.
  - Any amount of printing time beyond 10 minutes should be announced and extended to your exam stop time as a minor disruption.
- Added guidance for addressing environmental disruptions during the exam, similar to a bathroom break.
  - Clarification that this is considered a minor disruption, but bathroom break is not.
  - Removed the requirement to immediately report it to teaching staff.

Midterm Scope

5 questions on topics from Lecture 11 through Lecture 18

1. RISC-V Pipeline (counting stalls, forwarding)
2. CMOS Gates (sizing, LE, parasitics)
3. Path Delay (sizing, branching)
4. Elmore Delay (wires, model)
5. Power/Energy (calculation, optimization)

RISC-V Pipelining (FA18)



Consider the diagram above of a simple processor for storing arrays of data into memory. As discussed in a previous homework, the **swinc** instruction is well-suited for this task. For this architecture, the **swinc** instruction is as follows:

```
swinc rd, rs1, rs2
```

which stores the value of register `rs2` into the memory at the address given by register `rs1`. The address stored in `rs1` is then incremented and written back to register `rd`. A typical use case would be:

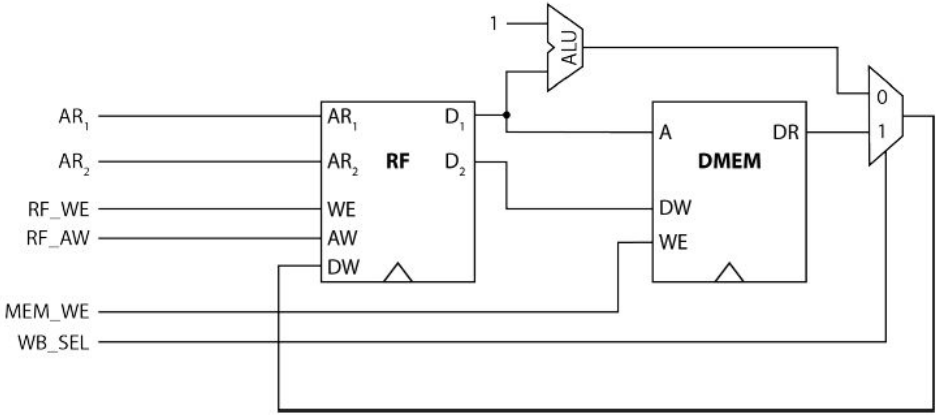
```
swinc x1, x1, x2
```

where the register `x1` is used to loop through an array of memory addresses, storing the value of `x2` and incrementing x1 by 1 each time.

In this implementation, the register file (RF) and data memory (DMEM) are **both synchronous read and synchronous write**. The ALU is used for incrementing the register value from the D_1 output of the register file, and a mux enables writing back the incremented address or a value loaded from memory (a function to be added later).

a) The processor, as shown, does not function properly for the swinc instruction. Explain what the error is, and add any components necessary to correct the design **to enable the swinc instruction only**. When correcting the design, do not add forwarding for any data hazards, but **minimize the number of stall cycles required**. (2 pts)

RISC-V Pipelining (FA18)



Consider the diagram above of a simple processor for storing arrays of data into memory. As discussed in a previous homework, the **swinc** instruction is well-suited for this task. For this architecture, the **swinc** instruction is as follows:

```
swinc rd, rs1, rs2
```

which stores the value of register `rs2` into the memory at the address given by register `rs1`. The address stored in `rs1` is then incremented and written back to register `rd`. A typical use case would be:

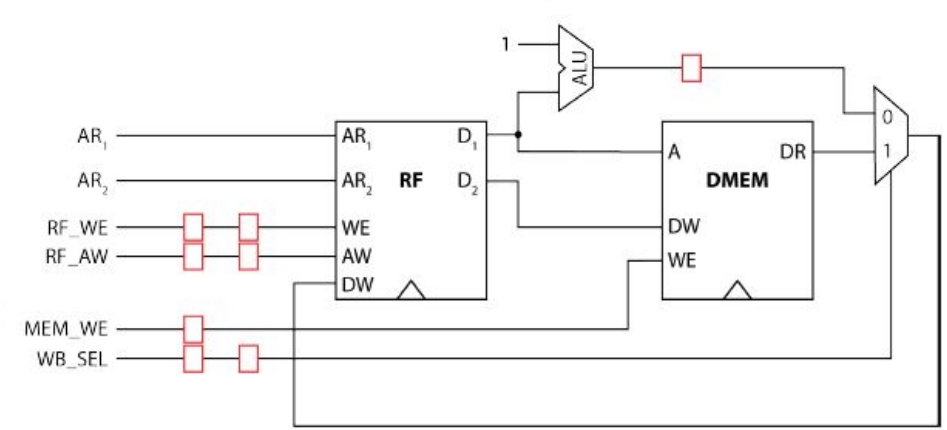
```
swinc x1, x1, x2
```

where the register `x1` is used to loop through an array of memory addresses, storing the value of `x2` and incrementing x1 by 1 each time.

In this implementation, the register file (RF) and data memory (DMEM) are **both synchronous read and synchronous write**. The ALU is used for incrementing the register value from the D_1 output of the register file, and a mux enables writing back the incremented address or a value loaded from memory (a function to be added later).

b) Now improve the design to also implement load instructions. In particular, add the instruction `lw rd, rs1` where the value in memory at address `rs1` is loaded into register `rd`. Again, explain the hardware additions, and there is no need to solve data hazards. (3 pts)

RISC-V Pipelining (FA18)



Consider the diagram above of a simple processor for storing arrays of data into memory. As discussed in a previous homework, the **swinc** instruction is well-suited for this task. For this architecture, the **swinc** instruction is as follows:

```
swinc rd, rs1, rs2
```

which stores the value of register `rs2` into the memory at the address given by register `rs1`. The address stored in `rs1` is then incremented and written back to register `rd`. A typical use case would be:

```
swinc x1, x1, x2
```

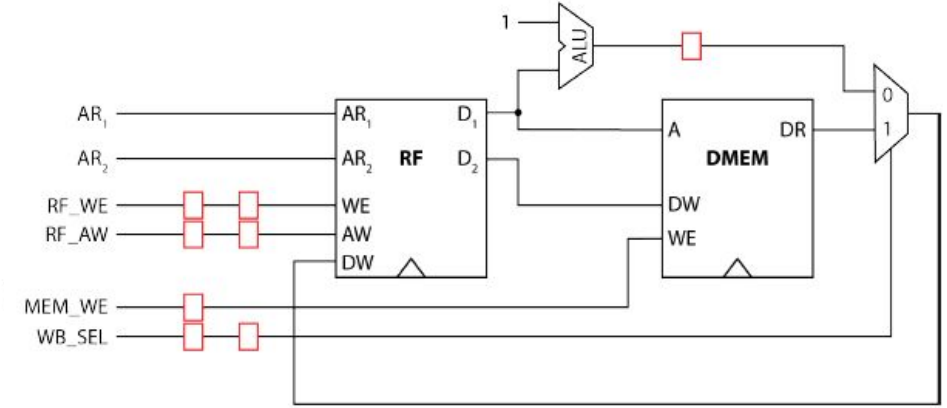
where the register `x1` is used to loop through an array of memory addresses, storing the value of `x2` and incrementing x1 by 1 each time.

In this implementation, the register file (RF) and data memory (DMEM) are **both synchronous read and synchronous write**. The ALU is used for incrementing the register value from the D_1 output of the register file, and a mux enables writing back the incremented address or a value loaded from memory (a function to be added later).

c) With your implementation from part (b) assume that a controller is in place to handle stalling for data hazards. How many cycles does your processor take to execute the following assembly code? (2 pts)

```
swinc x1, x1, x2
swinc x1, x1, x3
swinc x1, x1, x4
swinc x1, x1, x5
swinc x1, x1, x6
```

RISC-V Pipelining (FA18)



Consider the diagram above of a simple processor for storing arrays of data into memory. As discussed in a previous homework, the **swinc** instruction is well-suited for this task. For this architecture, the **swinc** instruction is as follows:

```
swinc rd, rs1, rs2
```

which stores the value of register `rs2` into the memory at the address given by register `rs1`. The address stored in `rs1` is then incremented and written back to register `rd`. A typical use case would be:

```
swinc x1, x1, x2
```

where the register `x1` is used to loop through an array of memory addresses, storing the value of `x2` and incrementing `x1` by 1 each time.

In this implementation, the register file (RF) and data memory (DMEM) are **both synchronous read and synchronous write**. The ALU is used for incrementing the register value from the D_1 output of the register file, and a mux enables writing back the incremented address or a value loaded from memory (a function to be added later).

c) With your implementation from part (b) assume that a controller is in place to handle stalling for data hazards. How many cycles does your processor take to execute the following assembly code? (2 pts)

```
swinc x1, x1, x2
swinc x1, x1, x3
swinc x1, x1, x4
swinc x1, x1, x5
swinc x1, x1, x6
```

d) Now add a single data forwarding path to minimize the number of cycles it takes to execute the same segment of assembly code as in part (c). How many cycles does your new implementation take? (2 pts)

CMOS Gate (FA19, 1a & 1b)

a) The following function F that implements an OR-AND-Invert OAI21 gate.

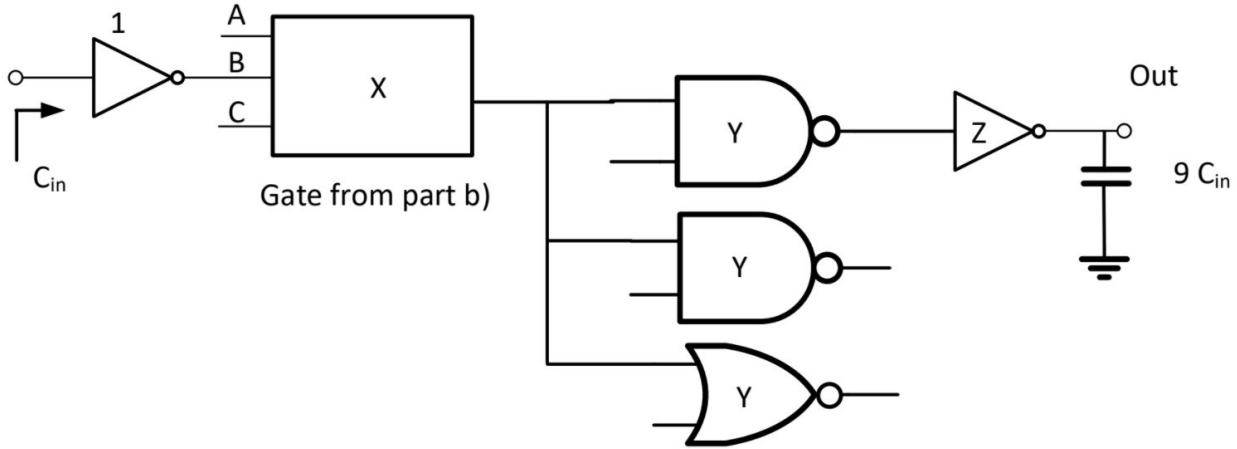
$$F = \overline{(A+B)C}$$

Implement the function F as standard, complementary CMOS logic gate. Size your transistors to match the pull-up/pull-down resistances to those of a unit inverter. You can assume NMOS and PMOS devices have equal strength in this technology.

b) Find the logical effort for the input B for the gate from part a).

Path Delay (FA19, 1c)

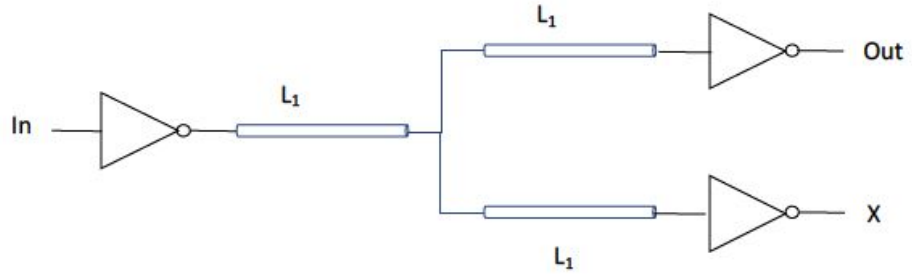
c) The gate from part a) is used in the following circuit. Determine the gate sizes, X, Y, Z, that minimize the delay in the path below. If you're not confident about your answer to part c, assume the logical efforts of the new gate is 3. Gates sized as '1' have the equivalent driving resistance and input capacitance equal to a unit-sized inverter.



Elmore Delay (FA 18)

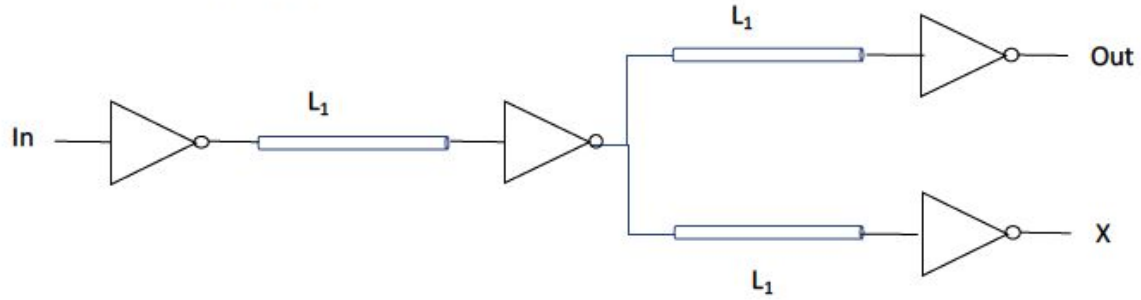
[Problem 2] Wires and Repeaters (10 Pts)

Consider the circuit shown below featuring a number of inverters connected by wires. For a minimum-sized inverter assume the following parameters: Driver resistance R_d, input capacitance C_{in} and intrinsic output capacitance $C_{int.}$ The following parameters hold for the wires (per unit length): c_w, r_w .



Elmore Delay (FA 18)

d) To reduce the delay, we add a repeater (also minimum sized) as shown in the Figure below. Derive an expression for the propagation delay of the revised circuit. (2 Pts)



Power/Energy (FA19, 3)

3) *Energy and Performance* (12 points, 12 minutes)

We would like to examine some properties of a single-cycle RISC-V datapath. The datapath presents a total load capacitance of 5pF to the supply, operates at 500 MHz, and has an activity factor of $\alpha_{\{0\to1\}}=0.1$. V_{DD} = 1V. The CPI is 1. You may find the following two equations useful for this problem.

$$\text{CPI}=\frac{\text{Clocks}}{\text{Instruction}}$$

$$\frac{\text{Seconds}}{\text{Program}}=\frac{\frac{\text{Instructions}}{\text{Program}}*\text{Cycles}}{\frac{\text{Instruction}}{\text{Cycle}}}*\text{Seconds}$$

a) What is the dynamic power consumption?

b) What is the average energy per instruction?

Power/Energy (FA19, 3)

c) How much energy is drawn from the supply to run a program with 1000 executed instructions?

d) What is the energy-delay product for the case in c) ?

e) You are able to pipeline this design so a 5-stage pipeline operates at 1.5GHz, at the same supply V_{DD} = 1V, has the same activity factor, and has a CPI of 3. The load capacitance has increased by 50% compared to the non-pipelined baseline. Calculate the energy-delay product when running the same program as in c)