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

EECS151/251A
Spring 2020

J. Wawrzynek
4/30/20

Final Exam

Name: _____

Student ID number: _____

Class (EECS151 or EECS251A): _____

You have three hours to take the exam. This exam comprises a set of questions with 1 point per expected minute of completion with a total of approximately 100 points. You have three hours to complete the exam. As with homework problems, submit your solutions using Gradescope. At the end of the exam time, we will give you a few extra minutes for you to submit your answers.

You are allowed to refer to your notes, the class lecture notes, and any other reference materials that you have available. You are not allowed to speak or communicate with anyone on any topic related to the exam during the exam period. After completing the exam, sign the following statement attesting that you did not discuss the exam problems with anyone else. You may either scan this page or copy the statement word-for-word.

I hereby declare that I have not spoken with nor otherwise communicated with anybody regarding the content of this exam while taking the exam:

(sign here): _____

For each problem if you find yourself taking excessive time to work out a solution consider skipping the problem or a fresh approach. Also, start by answering the easier questions and then move on to the more difficult ones.

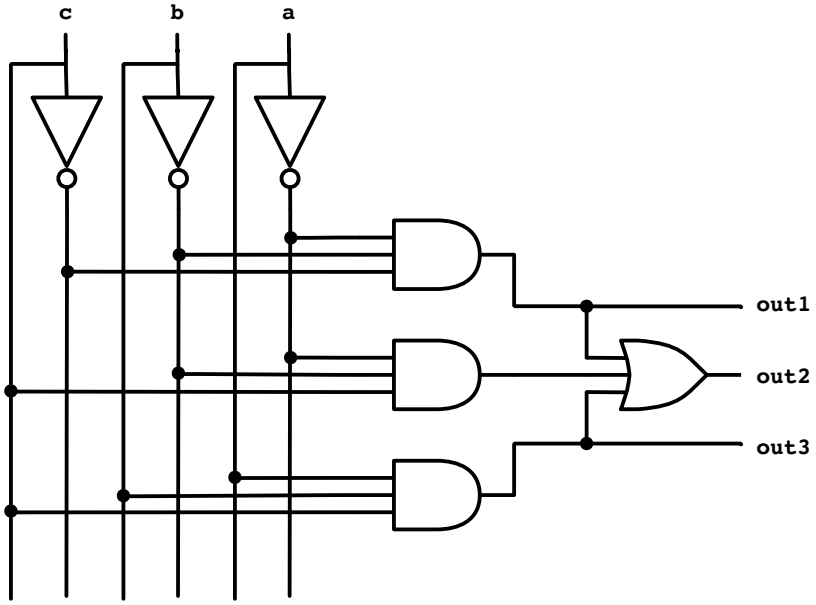
Neatness counts. We will deduct points if we need to work hard to understand your answer.

Before you turn in your exam, write your student ID number on all pages.

Version: 1 - 2020-05-04 13:36:43-07:00

1. **LUT mapping [5 pts]:**

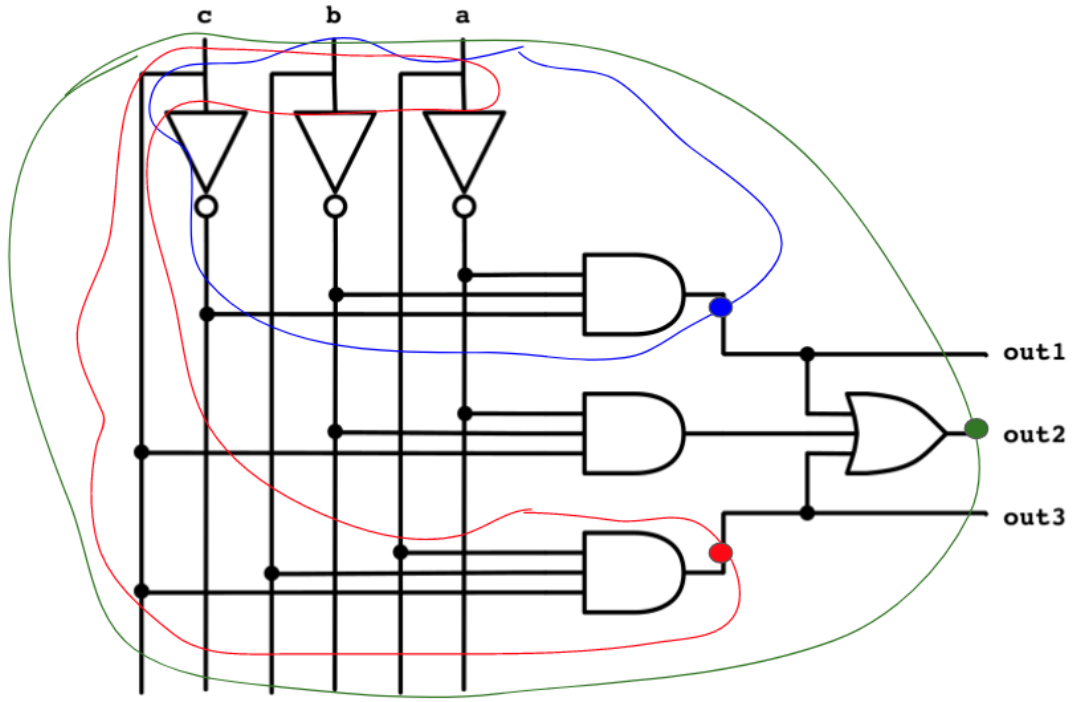
   a) Without trying to simplify the circuit, and by drawing on the circuit diagram below, demonstrate how to map these functions to the minimum number of 3-LUTs.



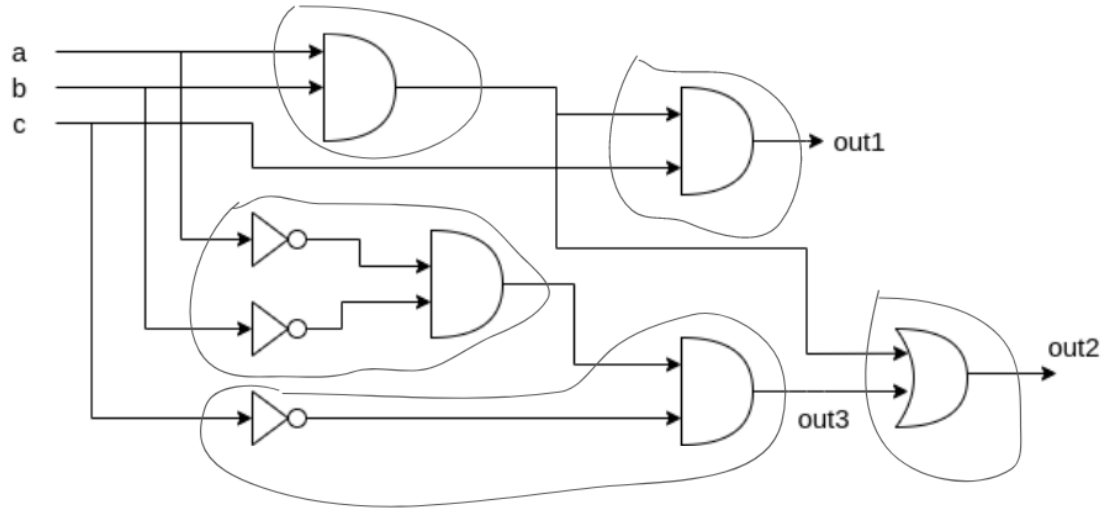
   b) Redraw the circuit and show how to map the function to the minimum number of 2-LUTs.

Solution:

(a) 3-LUT mapping



(b) 2-LUT mapping

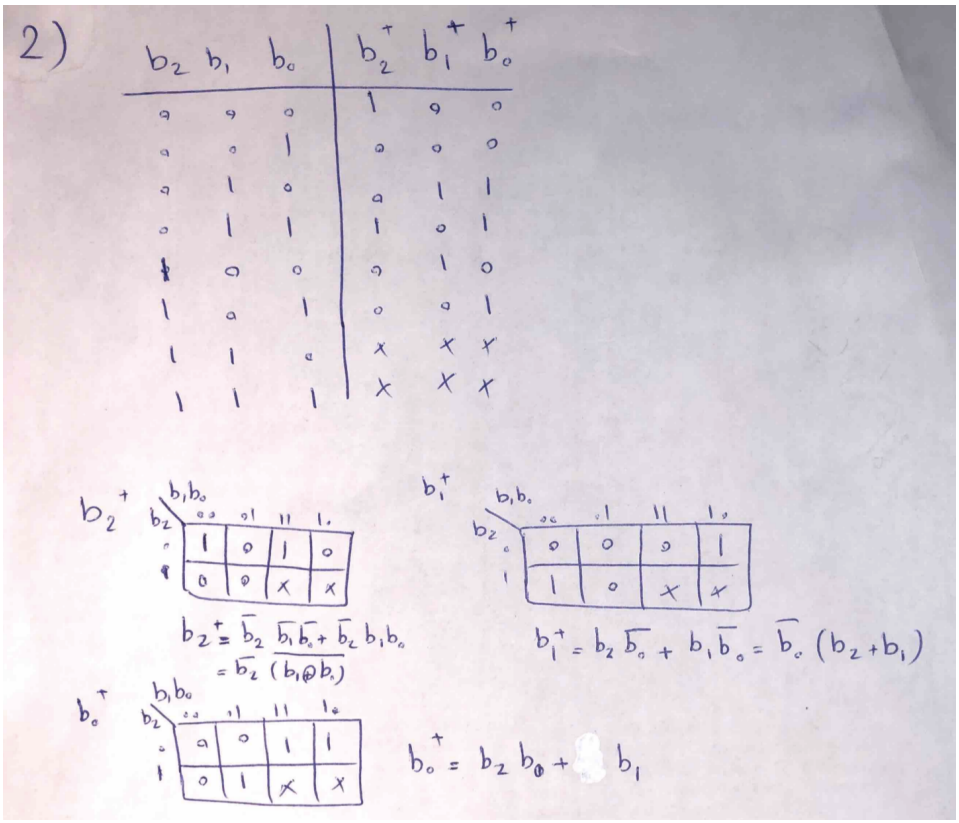


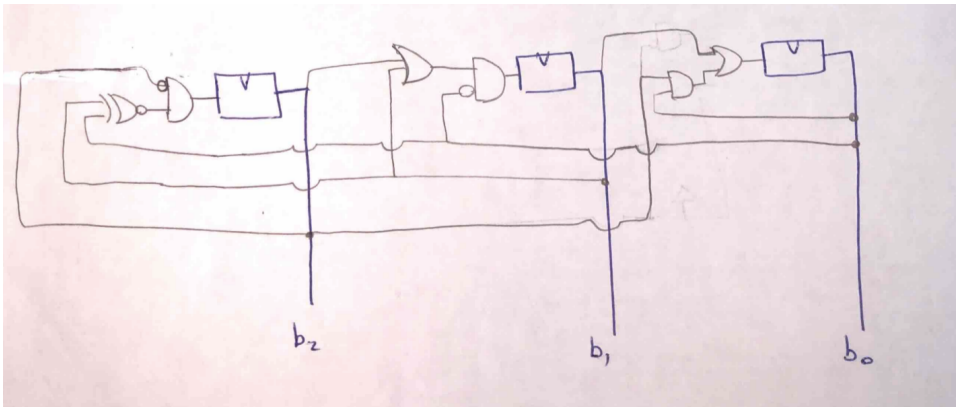
2. **Counter design [6 pts]:**

   A synchronous counter has the following sequence of output values: 000, 100, 010, 011, 101, 001, 000, ...

   Minimize the logic and draw the diagram of a circuit that implements this counter, using flip-flops and logic gates.

Solution:



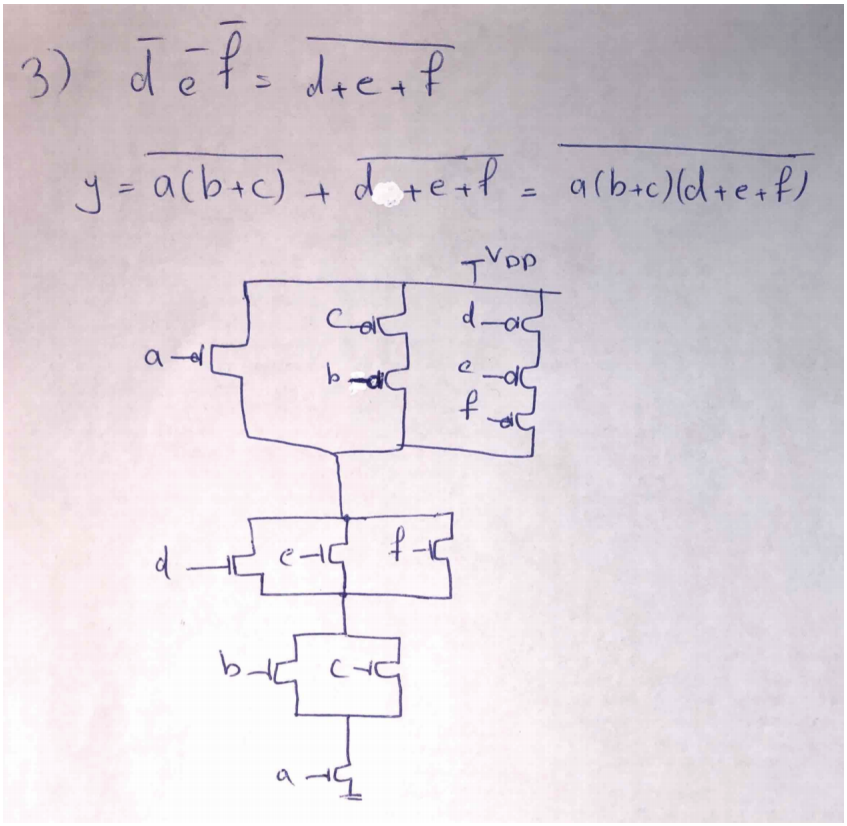


3. **CMOS gate [4 pts]:**

   Without assuming that inverted inputs are available, derive a single logic gate with the following function:

$$y = (a(b + c))' + d'e'f'$$

Solution:



4. **Multi-core Power and Energy [15 pts]:**

   Your new job is at ManyCores Inc. and you are asked to help design a embedded product that uses your 8 core chip.

   You measure the target workload and find that, with all 8 cores running, the chip consumes 10W of power while running at 1 GHz and Vdd = 1V. You would like to get the average power consumption down below 7.75W.

   You notice that the cores are only 80% busy during the measurements (they spend 20% of the time running, but just waiting for input).

   Looking at the datasheet you see that the chip is capable of running with Vdd in the range of 0.8-1.2V.

   You also see that the chip has a controller that can turn off all power (both switching and leakage power) to any or all of the cores.

   After some testing you determine that 50% of the total power is in leakage.

You ask you co-workers what to do to get to the target average power while maintaining the desired workload performance.

Co-worker #1 says: Just lower the clock frequency.

Co-worker #2 says: Lower Vdd.

Co-worker #3 says: Raise Vdd and "race to halt".

Who is right? (Evaluate all three options as a technique to save average power, while maintaining the desired performance. Show your work for each option.)

Solution:

$P_{tot} = 10W$ @ $f = 1GHz$, $V_{dd} = 1V$

$P_{tot} = P_{sw} + P_{leak}$

$= 5W + 5W$

$P_{avg} = (5W + 5W) \cdot 0.8 + 5W \cdot 0.2 = 9W$

#1 lower clock

$P_{sw} = \alpha C (0.8 V_{dd})^2 (0.8 f)$

$= 0.512 \alpha C V_{dd}^2 f$

$= 0.512 \cdot 5W$

$= 2.56 W$

$P_{leak} = 5W$

$P_{tot} = 7.56 W$

$P_{avg} = 7.56 W$

(if Vdd not changed)

$P_{sw} = 4W$

$P_{avg} = 9W$

#2 lower Vdd

$P_{sw} = \alpha C (0.8 V_{dd})^2 (0.8 f)$

$= 0.512 \cdot 5W$

$= 2.56$

$P_{tot} = 7.56 W$

$P_{avg} = 7.56 W$

#3 race to halt

$P_{sw} = \alpha C (1.2 V_{dd})^2 (1.2 f)$

$= 1.728 \cdot 5W$

$= 8.64 W$

$P_{tot} = 13.64 W$

$P_{avg} = 13.64 \cdot 0.667$

$= 9.09 W$

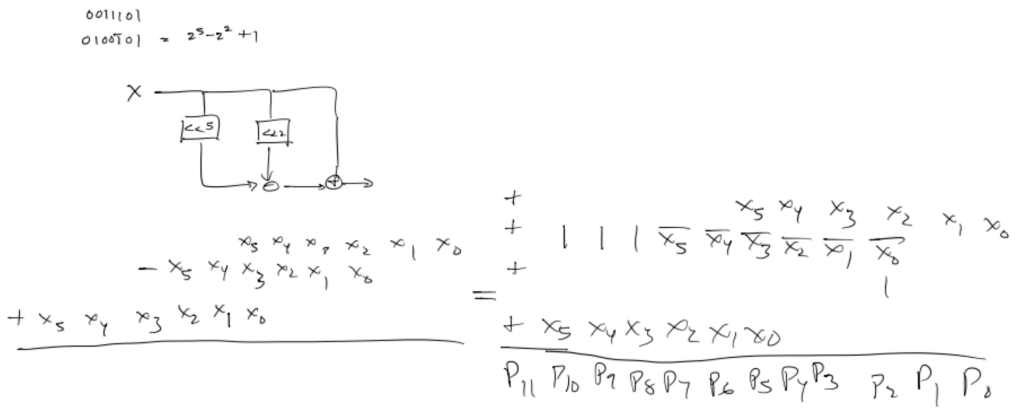
Lower Vdd best option

5. **Constant Multiplication [10 pts]:**

   Design a circuit for multiplying an unsigned 6-bit integer X by the constant, 6b'011101.

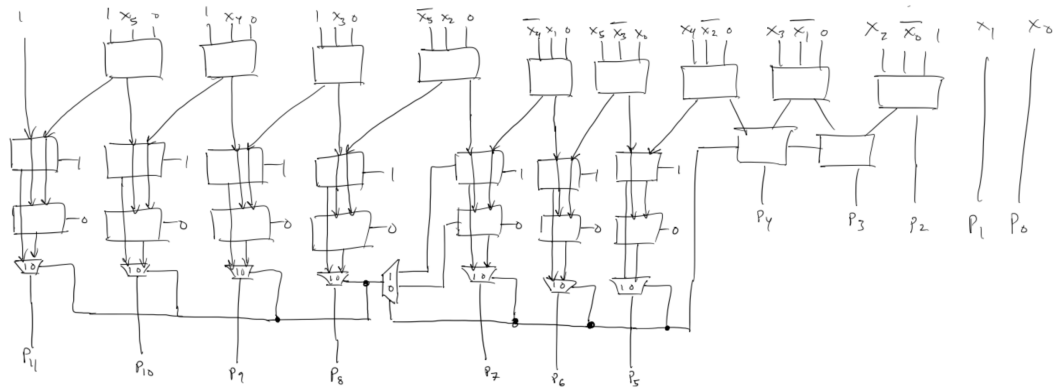
   (a) Draw the block diagram of your design.

Solution:



   (b) Draw a detailed circuit diagram using full-adder cells (FA) and, if needed, simple logic gates and multiplexors. Minimize cost, and use the *carry-save* technique where possible and *carry-select* where needed to speed up the circuit.

Solution:

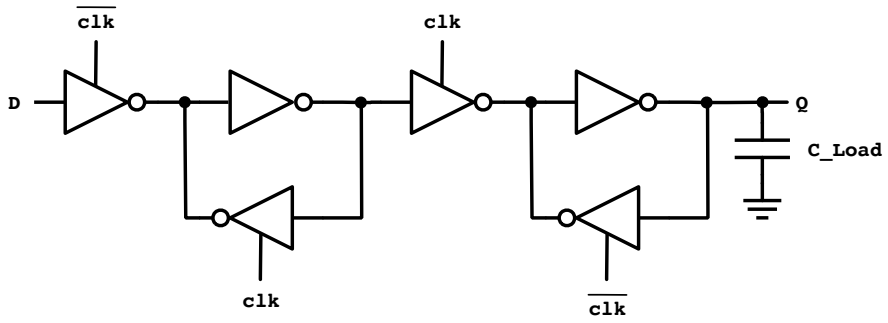


6. **Flip-flop Timing [10 pts]:**

   Calculate the clock-to-q time as a function of fanout for the tri-state based flip-flop design shown below.

   Assume that $\gamma = 1$ and that the strength per unit width for pFETs is half of nFETs. Recall (from the midterm) that for the tri-state buffer: $t_p = 2t_{p0}(1 + f)$. Also assume that the inverters are unit size.

   Show your work.



Solution:

The clock-to-q delay is the time from the rising edge of the clock the output change. It is the delay from the forward tristate buffer of the output latch to the flip-flop output, q. This delay comprises the delay of the tristate buffer plus the delay of the output inverter.

The tristate delay is $2t_{p0}(1+f)$. Its fanout include the input to the inverter ($3C_g$) plus the total drain C of the feedback tristate ($3C_d = 3C_g$, since gamma = 1). Therefore the forward tristate delay is

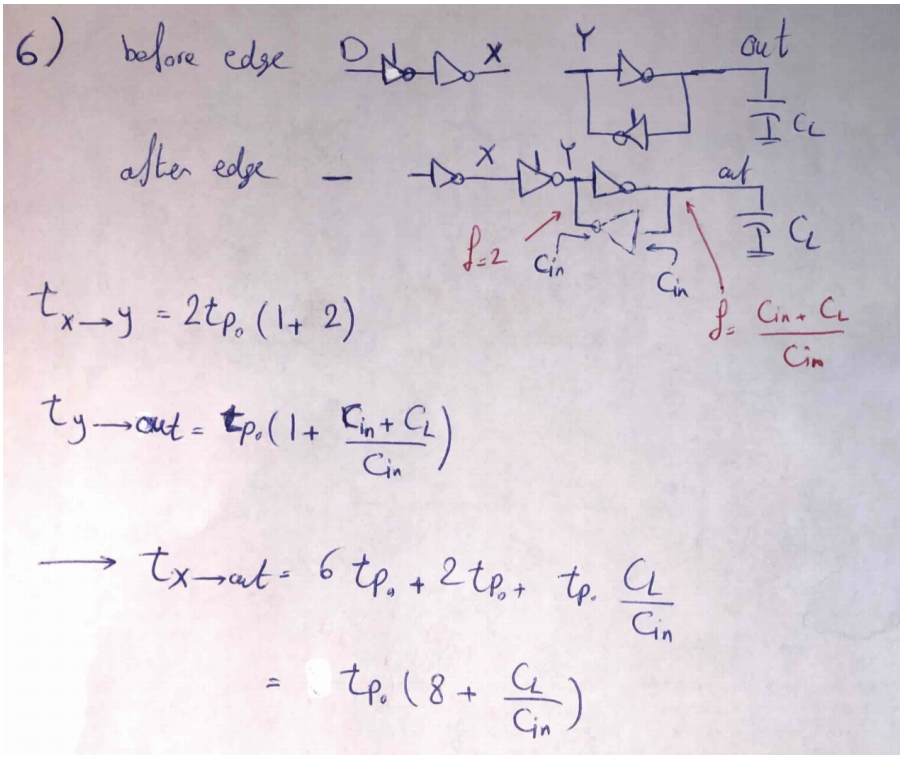
$$2t_{p0}\left(1+\frac{6C_g}{3C_g}\right) = 6t_{p0}$$

The inverter delay is $t_{p0}(1+f)$. The output inverter fanout includes the input to the feedback tristate + f, therefore the output inverter delay is

$$t_{p0}\left(1+\frac{C_{load}+3C_g}{3C_g}\right) = t_{p0}(2+f)$$

Therefore the total delay is

$$t_{p0}(8+f)$$



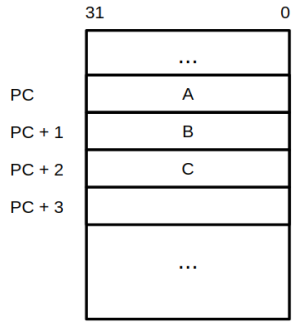
7. **Problem 7. Single Instruction Computer Design [20 pts]:**

   Several proposals in the past have been made for Turing complete computers that operate with only a single type of instruction. One such proposal is for a computer based on the "SubLeq" (subtract and branch if less than or equal) instruction.

   A SubLeq instruction has 3 32-bit operands: `A B C`.

   Execution of one instruction `A B C` subtracts the value in the memory location at the address stored in A from the content of a memory location at the address stored in B and then writes the result back into the location with the address in B. If the value after subtraction in B is less or equal to zero, the execution jumps to the address specified in C; otherwise execution continues to the next instruction, i.e. the address of the memory location following C.

   For this problem we assume that all instructions and all data is stored in a single memory module. It is a simple dual-port (one read port, one write port) asynchronous read and synchronous write memory. Unlike the RISC-V, memory is word-addressed (32-bit words), not byte addressed. Instructions are stored in the memory as illustrated below:



To complete this problem, design a datapath circuit and specify the controller for this computer. Your goal is to maximize the performance while keeping the amount of hardware to a minimum (priority on performance). (Hint: first, generate a design without worrying about cost and performance, then later think about optimizations to improve cost and performance. Turn in both versions.)

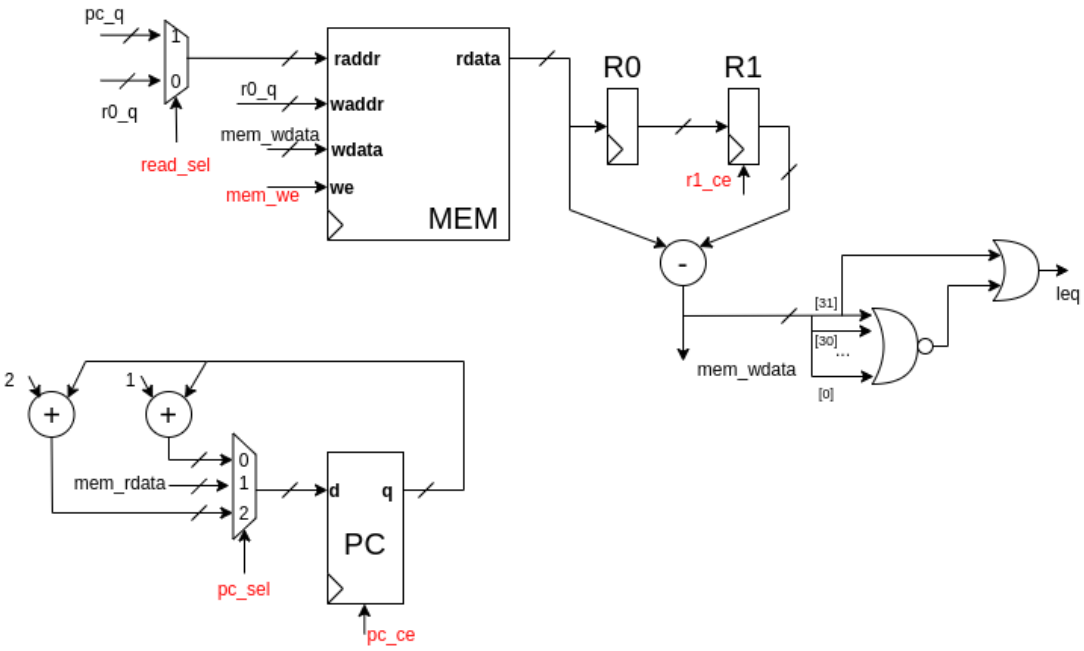
a) Show your datapath design in the form of a block diagram. You may use registers, arithmetic units, multiplexors, simple logic gates, and the memory module described above. Label all control points.

b) For the controller portion, write the RT Language description.

Solution:

Optimization for clock cycles

Datapath.



The control signals are highlighted in red. The critical path is the path from asychronous read port from the memory block to the subtractor to the write data port of the memory block.

(b) Controller (RT Language).

Here are the steps to execute an instruction.

- Read operand `A` from the memory and store it to `R0`;
- Read `mem[A]` from the memory and store it to `R0`,
  Increment `PC` to read the next operand;
- Read operand `B` from the memory and store it to `R0`,
  Store `mem[A]` to `R1`;
- Setup a write to memory,
  If `leq` == 0, increase PC by 2 to read the next instruction (branch is not-taken),
  otherwise increase PC by 1 to read the next operand;
- Read operand `C` from the memory and store `C` to `PC` (branch is taken);

RTL code (only need to write assignments to synchronous elements, the control signals are implied from the code):

```
repeat {
    R0 <- Mem[PC];
    R0 <- Mem[R0], PC <- PC + 1;
    R0 <- Mem[PC], R1 <- R0;
```

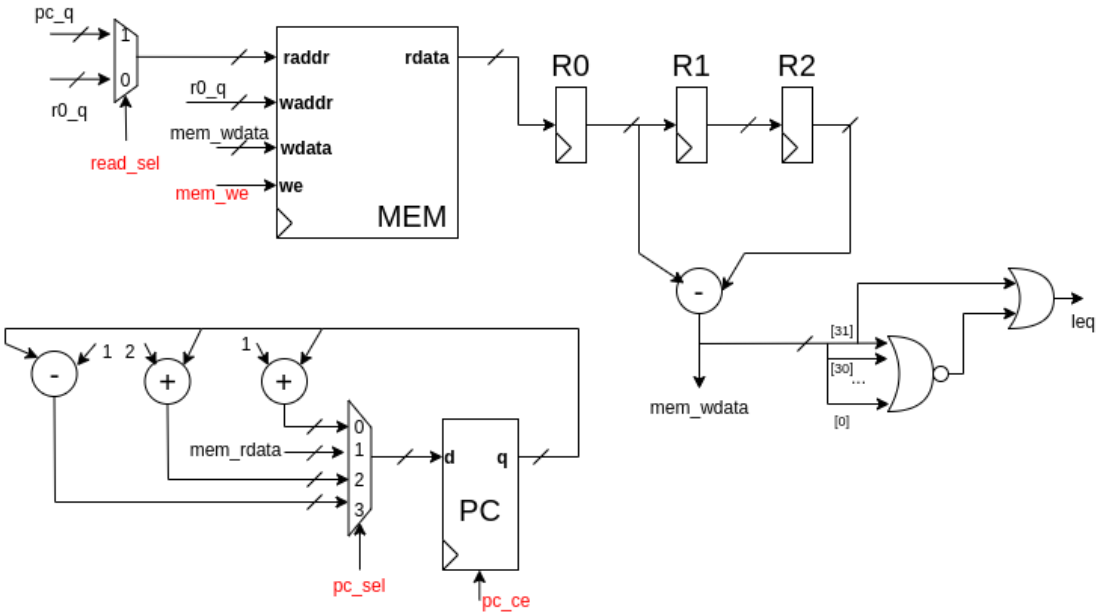
```
        R0 <- Mem[R0], Mem[R0] <- Mem[R0] - R1,
        if (leq == 0) {
            PC <- PC + 2,
            continue;
        else {
            PC <- PC + 1;
        }
        PC <- Mem[PC];
    } until (1);
```

The number of cycles to execute an instruction: best case is 4 cycles (branch is taken) , worst case is 5 cycles (branch is not taken).

Solution:

Optimization for clock frequency

Datapath. We predict branches not taken. If the prediction is wrong, we roll back to the previous memory location to read operand C.



The control signals are highlighted in red.

(b) Controller (RT Language).

RTL code (non-CE `R1`, `R2` register assigments are omitted for brevity)

```
repeat {
    R0 <- Mem[PC],

STATE1:
    R0 <- Mem[R0],
    PC <- PC + 1;
```

```
        R0 <- Mem[PC];

        R0 <- Mem[R0],
        PC <- PC + 2;

        Mem[R1] <- R2 - R0,
        if (leq == 0) {
            R0 <- Mem[PC],
            j STATE1;
        }
        else {
            PC <- PC - 1,
        }

        PC <- Mem[PC];
    }
```

The number of cycles to execute an instruction: best case is 4 cycles, worst case is 6 cycles. This is optimal in performance if we assume that branches are usually not taken. Alternatively, if no branch prediction is performed, it always takes 5 cycles to execute an instruction which is also a valid solution.

8. **Problem 8. Stack Design [15 pts]:**

   A hardware stack implements a last-in-first-out (LIFO) data structure.

   For this problem, we are interested in designing the circuit for implementation of a stack of 512 elements with the following inputs and outputs:

   `clk`: clock signal,

   `push`: when asserted on rising edge of clock, data input is placed on the stack,

   `pop`: when asserted on rising edge of clock, data is removed from stack,

   `din`: 8-bit wide signal for data push,

   `dout`: 8-bit wide signal for data pop,

   `full`: output signal that indicates that a push operation would result in overflow,

   `empty`: output signal that indicates that a pop operation would result in underflow.

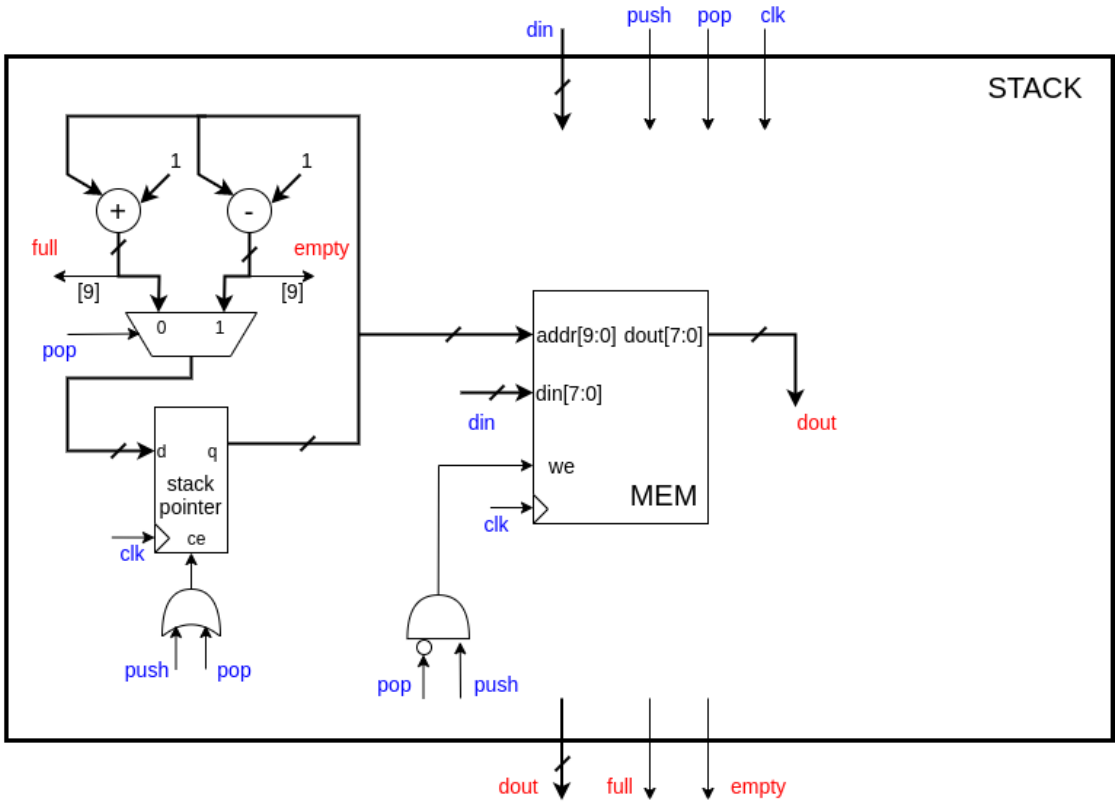
   If both `push` and `pop` are asserted on the same cycle, `pop` gets priority. You don't need to ensure correct behavior after an overflow or underflow occurs.

   The set of components you are allowed to use is: simple logic gates, multiplexors, tri-state buffers, arithmetic units, registers, and an asynchronous read synchronous write 1024x8 memory module.

   Draw a circuit diagram with your complete design. Use hierarchy for clarity.

Solution:

Input signals are highlighted in red, output signals are blue. When **push** and **pop** occur in the same cycle, the stack only executes **pop**.



9. **Bit Reversal Circuit [10 pts]:**

   Devise a combinational circuit with the following specifications. The data input is 16-bits wide, and the output is 16-bits wide with some bits reversed. Depending on the control input, the circuit can group the input as a 16-bit word, or as 2 8-bit bytes, or as 4 4-bit nibbles, or as 8 2-bit pairs. For each of these four options, the circuit can independently reverse the bits within each of the groups.

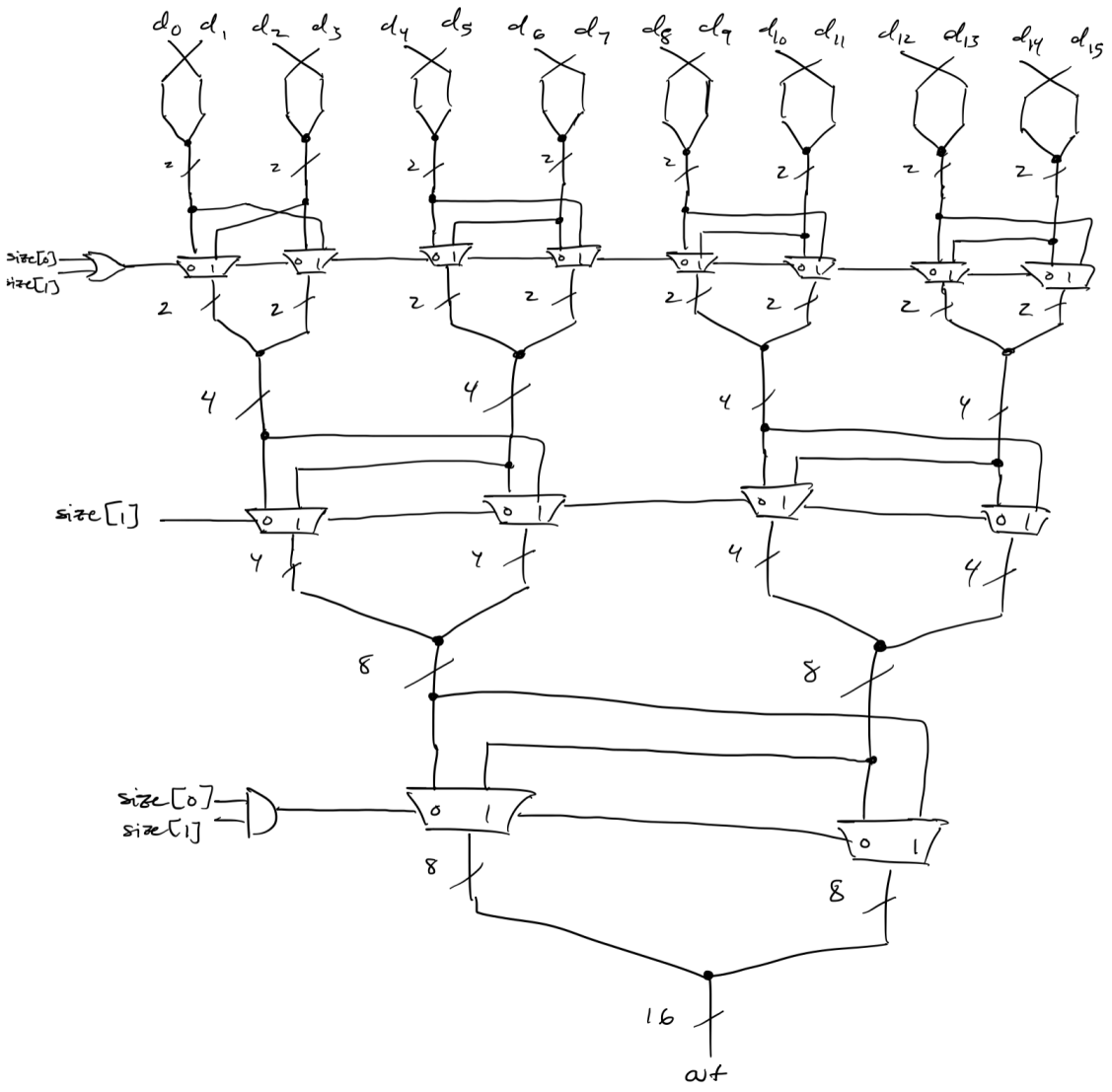
   The grouping is controlled by the 2-bit control signal, "`size`" as follows: `size` = 0 (pairs), 1 (nibbles), 2 (bytes), 3 (word).

   So for instance, if the control input is `size` = 2, then the circuit will reverse the order of the top 8 bits and independently reverse the order of the bottom 8 bits. On the other hand, if the control input is `size` = 3, then the circuit will perform one 16-bit reversal.

   You may use simple logic gates and 2-input multiplexors. Your goal is to find a good design in terms of cost and performance. Give cost priority.

   Draw your circuit design.

Solution:



We also accepted a 4-1 MUX solution, provided you show the implementation for a 4-1 MUX using 2-1 MUXes.

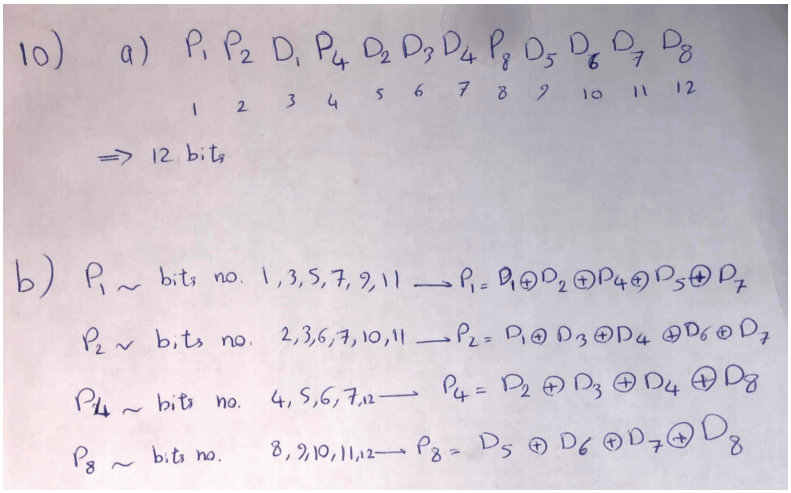
10. **Error Correction Code [8 pts]:**

Derive the Hamming code for an 8-bit data word.

a) Express your answer by numbering your code-word bits from left to right starting at "1" and indicating what bit occupies each position.

b) Write an expression for calculation of each of the parity bits on a *write* operation.

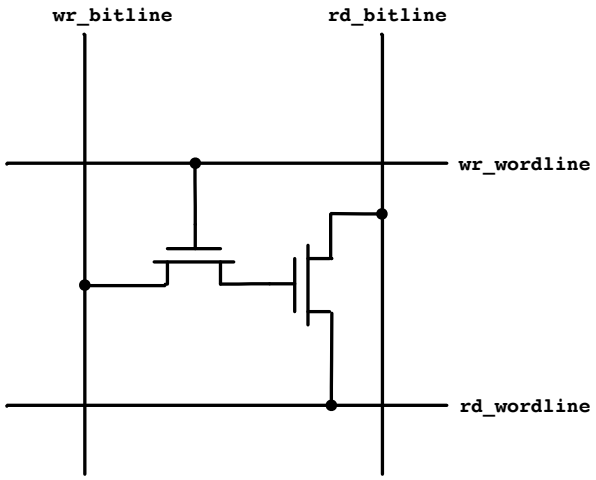
Solution:



11. **2-Transistor DRAM, *251A Only* [6 pts]:**

John has a new idea for a DRAM that can be built without a special IC process (unlike normal DRAM).

The storage cell uses 2 transistors in the configuration shown below.



a) Explain how it would work (the steps involved for reading and writing).

Solution:

For write: assert `wr_wordline` and drive the proper value on `wr_bitline`.

For read, the most effective way to use this cell is the following is: Assert all the `rd_wordlines` in array, then precharge the `rd_bitlines` high, then de-assert the `rd_wordline` for the selected row. The cells in the selected row storing a 1 will pull down on the `rd_bitline`. A sensing circuit looks for a lowering of the prechared `rd_bitline`.

b) Identify and explain any potential problems with this design.

Solution:

During the read operation, there is a sneak path for current onto the `rd_bitline` from cells in the same column above and below the selected row. If any of the other cells in the column are storing a 1, then they will try to pull the `rd_bitline` high while the cell in the selected row will try to pull it low. The result will be that the `rd_bitline` will only drop to Vdd-Vt.

Changing the sense of `rd_bitline` to precharge low and force the cell storing a 1 to pull it up doesn't help. It makes the problem worse.