1) Recalculate the various propagation delays in a 4-bit carry lookahead adder and a ripple adder assuming that XOR gates have twice the delay of all other gates.

For the ripple adder, we have:

\[ S_0 = A \text{ xor } B \text{ xor } C_{in} = 4 \text{ gate delays} \]
\[ C_1 = BC_i + AC_i + AB = 2 \text{ gate delays} \]
\[ S_i = 2 \text{ past } C_{i-1} \]
\[ C_i = 2 \text{ past } C_{i-1} \]

For the carry lookahead adder, we have:  
For the basic 1-bit circuit, we have:

\[ P_i @ 2 \text{ gate delays} \]
\[ S_i @ 4 \text{ gate delays} \]
\[ G_i @ 1 \text{ gate delay} \]

Therefore, \( C_1 - C_4 @ 2 \text{ past } P_i \) (since \( C_i \) is two level AND/OR logic) = 4 gate delays

\[ S_0 @ 4 \text{ gate delays} \]
\[ S_1 - S_3 @ 2 \text{ past } C_i = 6 \text{ gate delays} \]
2) The easiest way to perform arithmetic operations using the sign-and-magnitude system is to convert to 2’s complement for your calculations. You are given the following components:
- A 4-bit universal shift register
- As many 1-bit full adders as you need
- As many 2-input XOR gates as you need
- As many 2-input NAND gates as you need

a) Design a circuit to convert a 4-bit sign-and-magnitude number to 2’s complement number.

b) Design a circuit to convert a 4-bit 2’s complement number to a sign-and-magnitude number.
3) Trace the execution of the current instruction in the following microprocessor datapath. The instruction syntax is given on slide 22 of the class notes on computer organization.

<table>
<thead>
<tr>
<th>PC: 38H</th>
<th>REG[000]: 0000H</th>
<th>REG[010]: 0040H</th>
<th>REG[001]: 00F0H</th>
<th>REG[011]: 0100H</th>
<th>REG[100]: 0000H</th>
<th>REG[101]: 1100H</th>
<th>REG[110]: FF00H</th>
<th>REG[111]: FF00H</th>
<th>MEM[38H]: 2502H</th>
<th>MEM[F0H]: 1264H</th>
<th>MEM[F1H]: 0000H</th>
<th>MEM[F2H]: 1000H</th>
<th>MEM[F3H]: FFFFH</th>
<th>MEM[F4H]: 0000H</th>
<th>MEM[FFH]: 0000H</th>
</tr>
</thead>
</table>

a) List the sequence of transfer operations and control signals (see slide 28 for an example of what is expected from you).

Instruction fetch:
- mabus PC; move PC to memory address bus (PCmaEN, ALUmaEN)
- memory read; assert memory read signal (mr, RegBmdEN)
- IR memory; load IR from memory data bus (IRld)
- op add; send PC into A input, 1 into B input (srcA, srcB0, scrB1, op)
- PC ALUout; load result of incrementing in ALU into PC (PCld, PCsel)

Instruction decode: (IR = 0010010100000010)
- IR to controller (instruction is a LD instruction, rt is REG[010], rs is REG[001], offset = 0000010)
- values of A and B read from register file (rs, rt)

Instruction execution:
- op add; send regA into A input, Inst into B input, add (srcA, srcB0, scrB1, op)
- ALU ALUout; clock ALU output into ALU Buffer
- mabus ALU; move ALU to memory address bus (PCmaEN, ALUmaEN)
- memory read; assert memory read signal (mr, RegBmdEN)
- MBR memory; load MBR from memory data bus (MBRld)
- rt MBR; store MBR into destination register (regWrite, wrDataSel, wrRegSel)

b) After the instruction execution is complete, list the contents of the following, in hexadecimal:

- PC: 39H
- REG[000]: 0000H
- REG[001]: 00F0H
- REG[010]: 1000H
- REG[011]: 0100H
- REG[100]: 0000H
- REG[110]: FF00H
- REG[111]: FF00H

<table>
<thead>
<tr>
<th>PC: 38H</th>
<th>REG[000]: 0000H</th>
<th>REG[010]: 1000H</th>
<th>REG[001]: 00F0H</th>
<th>REG[011]: 0100H</th>
<th>REG[100]: 0000H</th>
<th>REG[101]: 1100H</th>
<th>REG[110]: FF00H</th>
<th>REG[111]: FF00H</th>
<th>MEM[38H]: 2502H</th>
<th>MEM[F0H]: 1264H</th>
<th>MEM[F1H]: 0000H</th>
<th>MEM[F2H]: 1000H</th>
<th>MEM[F3H]: FFFFH</th>
<th>MEM[F4H]: 0000H</th>
<th>MEM[FFH]: 0000H</th>
</tr>
</thead>
</table>

c) Suppose you wanted to display the results of the execution of your instruction on the display (which is connected to MEM[39H]). Write the contents of MEM[39H].

The instruction is sw, REG[001], REG[010], 000010, i.e., 0100010100000111
MEM[39H] = 450FH
4) You are going to implement the following machine (which you may remember as the washing machine from midterm I) using a pure jump counter (Note that there are MEDIUM and LARGE inputs and outputs).

![Diagram of the washing machine states and transitions]

a) Determine an optimal encoding for the states.

- WAIT: 000
- RINSE: 010
- DRY2: 100
- WASH: 001
- DRY1: 011

b) Determine the control functions:

\[
\begin{align*}
\text{CNT} &= \text{WAIT} \cdot \text{START} + \text{WASH} \cdot \text{T2DONE} + \text{RINSE} \cdot (\text{T2DONE} \cdot \text{DIRTY}' + \text{T1DONE}) + \text{DRY1} \cdot \text{T2DONE} \cdot \text{WET} \\
\text{LD} &= \text{RINSE} \cdot \text{T2DONE} \cdot \text{DIRTY} \cdot \text{T1DONE}' \\
\text{CLR} &= \text{DRY1} \cdot \text{WET}' + \text{DRY2} \cdot (\text{WET}' + \text{T2DONE})
\end{align*}
\]

c) List the contents of the Jump ROM.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>XXX</td>
<td>010</td>
<td>001</td>
<td>100</td>
<td>XXX</td>
<td>110</td>
<td>XXX</td>
</tr>
<tr>
<td>001</td>
<td>XXX</td>
<td>011</td>
<td>XXX</td>
<td>101</td>
<td>XXX</td>
<td>111</td>
<td>XXX</td>
</tr>
</tbody>
</table>
d) List the output functions:

LARGE = \text{WAIT} \cdot \text{START} \cdot \text{LARGE}

MEDIUM = \text{WAIT} \cdot \text{START} \cdot \text{MEDIUM}

T1START = \text{WAIT} \cdot \text{START}

T2START = \text{WAIT} \cdot \text{START} + \text{WASH} \cdot \text{T2DONE} + \text{RINSE} \cdot (\text{T2DONE} \cdot \text{DIRTY}' + \text{T1DONE}) + \text{RINSE} \cdot \text{DIRTY} \cdot \text{T2DONE} + \text{DRY1} \cdot \text{T2DONE} \cdot \text{WET}

WASH = \text{WAIT} \cdot \text{START} + \text{WASH} \cdot \text{T2DONE}'

RINSE = \text{WASH} \cdot \text{T2DONE} + \text{RINSE} \cdot \text{T2DONE}'

DRY = \text{RINSE} \cdot (\text{T2DONE} \cdot \text{DIRTY}' + \text{T1DONE}) + \text{DRY1} \cdot \text{WET} + \text{DRY2} \cdot \text{T2DONE}' \cdot \text{WET}

e) Draw a circuit diagram for the jump counter. You do not need to show the output function implementations. Use the PLA below to implement your control functions, and use a 74163 4-bit counter as the state storage element.
5) You are going to re-implement the washing machine controller from the previous problem using vertical microcode.

\[ \text{WAIT} \]
\[ \text{WASH} \]
\[ \text{START} \]
\[ \text{RESET} \]
\[ \text{START.MEDIUM/START.LARGE} \]
\[ \text{T2DONE.DIRTY/T2DONE} \]
\[ \text{T2DONE/RINSE} \]
\[ \text{T2DONE.WET/DRY} \]
\[ \text{DRY} \]

a) List all sets of mutually exclusive outputs

(MEDIUM, LARGE) and (WASH, RINSE, DRY)

b) Assuming you use 1-hot encoding for all non-mutually exclusive output groupings, what is the minimum number of bits you need to encode the outputs?

4 (T1START, T2START, MEDIUM/LARGE, WASH/RINSE/DRY)

c) Assume you need a 5-bit ROM address to encode the machine. Write out the format (as on slide 39 of the controller implementation notes) of the RT and BJ instructions for this machine assuming (b).

BJ = Type (1 bit): Condition Select (3 bits): Compare (1 bit): Address (5 bits)
Condition Selects: START=000, MEDIUM=001, LARGE=010, T2DONE=011, T1DONE=100, DIRTY=101, WET=110

RT = Type (1 bit): Output (4 bits): Unused (5 bits)
Output = T1START:T2START: MEDIUM/LARGE: WASH/RINSE/DRY