ECE 4100/ECE 6100
Advanced Computer Architecture
Prerequisite Self-Assessment Test
(Not to be submitted)

This self-assessment test is intended to help you determine your level of preparation for ECE 4100/ECE 6100 by going through some of the background material we expect you to have seen already. We also hope working through the homework problem set will help refresh your memory on these topics. We will only have a short review of this material in class.

For each question, we ask that you fill out the table at the end of the problem set handout indicating your level of confidence with each assigned problem. If you have never seen the material before, then please enter “0”. If you have seen the material, and think you should know it, but can’t answer the question without spending time studying your old notes, then please enter “1”. If you are very comfortable with the material, then enter “2”.

If you have more than 6 or 7 “0”s in the table, or feel uncomfortable with your background preparation or our expectations for the course, then you should arrange a meeting with the instructor and/or TAs to discuss your particular situation before drop date. Our experience is that most students with some background in logic design or assembly-level computer programming can complete the course successfully.

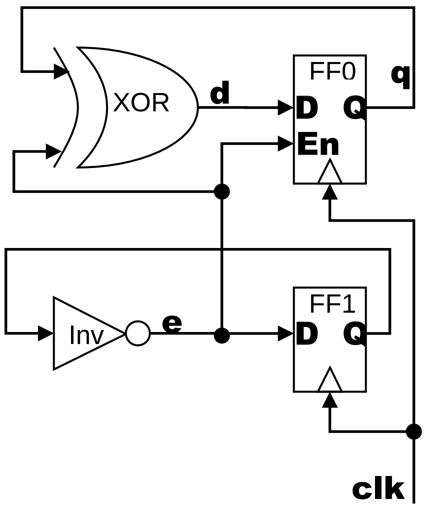
Problem 1

Construct the following logic functions using only two-input NAND gates (please use hierarchy where possible to simplify your designs):

a) inverter
b) two-input XOR gate
c) 2-to-1 multiplexer
d) 2-to-4 decoder

<u>Problem 2</u>

The questions below refer to the following circuit and its associated timing parameters. The flip-flops are positive-edge triggered, and FF0 has an enable input (Q only changes if En is high). Assume that all timing parameters are positive.



Fill out the state transition diagram for the sequential logic drawn above.

FF0	FF1	FF0’	FF1’

Problem 3

The followings are two code segments written in MIPS64 assembly language:

Segment A:

```
Loop:  LD        r5, 0(r1)       # r5 ← Mem[r1+0]
       LD        r6, 0(r2)       # r6 ← Mem[r2+0]
       DADD      r5, r5, r6      # r5 ←  r5 + r6
       SD        r5, 0(r3)       # Mem[r3+0] ← r5
       LD        r5, 0(r1)       # r5 ← Mem[r1+0]
       LD        r6, 0(r2)       # r6 ← Mem[r2+0]
       DSUB      r5, r5, r6      # r5 ←  r5 - r6
       SD        r5, 0(r4)       # Mem[r4+0] ← r5
       DADDUI    r1, r1, 8       # r1 ← r1 + 8
       DADDUI    r2, r2, 8       # r2 ← r2 + 8
       DADDUI    r3, r3, 8       # r3 ← r3 + 8
       DADDUI    r4, r4, 8       # r4 ← r4 + 8
       BNE       r1, r9, Loop    # branch to Loop if r1 ≠ r9
```

Segment B:

```
Loop:  LD        r5, 0(r1)       # r5 ← Mem[r1+0]
       LD        r6, 0(r2)       # r6 ← Mem[r2+0]
       DADD      r7, r5, r6      # r7 ←  r5 + r6
       DSUB      r8, r5, r6      # r8 ←  r5 - r6
       SD        r7, 0(r3)       # Mem[r3+0] ← r7
       SD        r8, 0(r4)       # Mem[r4+0] ← r8
       DADDUI    r1, r1, 8       # r1 ← r1 + 8
       DADDUI    r2, r2, 8       # r2 ← r2 + 8
       DADDUI    r3, r3, 8       # r3 ← r3 + 8
       DADDUI    r4, r4, 8       # r4 ← r4 + 8
       BNE       r1, r9, Loop    # branch to Loop if r1 ≠ r9
```

In both segments, assume r1, r2, r3, r4 initially hold valid memory addresses. Register r9 is pre-computed to be 80 larger than the initial value of r1. All instructions operate on 64-bit doubleword values and the memory address space is byte-addressable.

a) If both segments are expected to perform the same task, can you guess what the task is? You can write the answer in C-like pseudo code.

b) In general, which segment do you expect to perform better when executed?

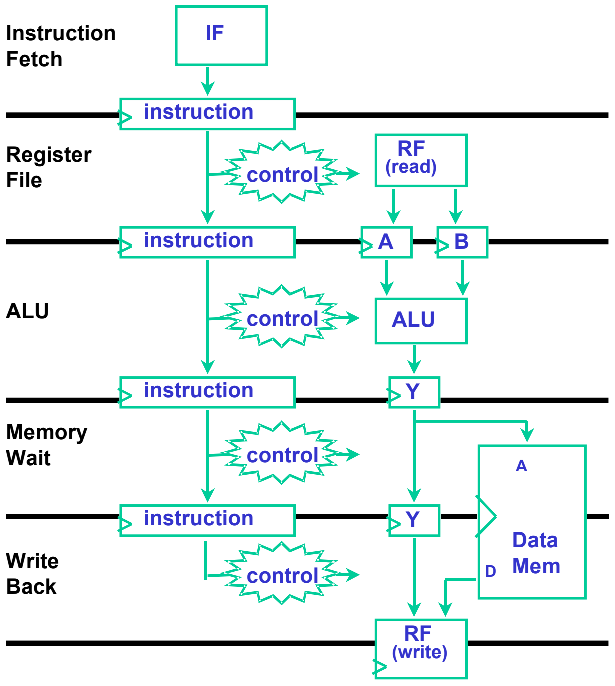
c) Do the two segments always produce the same results in all situations? If not, can you specify a situation which makes them behave differently?

Problem 4

The following figure shows a 5-stage pipelined processor. The pipelined processor should always compute the same results as an unpipelined processor. Answer the following questions for each of the instruction sequences below:

- Why does the sequence require special handling (what could go wrong)?
- What are the minimal hardware mechanisms required to ensure correct behavior?
- What additional hardware mechanisms, if any, could help preserve performance?

Assume that the architecture does not have any branch delay slots, and assume that branch conditions are computed by the ALU.



```
a)  BEQ     r1, r0, 200     # branch to PC+200 if r1 == r0
    DADD    r2, r3, r5      # r2 ← r3 + r5
    DSUB    r4, r5, r6      # r4 ← r5 + r6
    …

b)  DADD    r1, r0, r2      # r1 ← r0 + r2
    DSUB    r4, r1, r2      # r4 ← r1 - r2
    …

c)  LD      r1, 0(r2)       # r1 ← Mem[r2+0]
    DADD    r3, r1, r2      # r3 ← r1 + r2
    …
```

<u>Problem 5</u>

Describe the operation of a data cache. Your description should include discussion of the following:

a) Spatial and temporal locality.
b) Valid bits.
c) Direct mapped versus set-associative structures. Show how cache indexing and tag match works for both direct mapped and 2-way set-associative cache configurations assuming one word per cache line. What are the advantages and disadvantages of direct mapped versus set-associative structures?
d) Multiple-word cache lines. What are the advantages and disadvantages of multiple-word cache lines? Describe how they are implemented for a direct mapped cache.
e) LRU and random replacement policies. What are their relative advantages and disadvantages?

Problem Ratings

		subproblem						
		A	**B**	**C**	**D**	**E**	**F**	**G**
problem	**1**					▨	▨	▨
	2		▨	▨	▨	▨	▨	▨
	3				▨	▨	▨	▨
	4				▨	▨	▨	▨
	5						▨	▨

0	No idea
1	Used to know it
2	Know it