

6.5900 Computer System Architecture
Prerequisite Self-Assessment Test

Assigned Sept. 06, 2023

Due Sept. 08, 2023

<http://csg.csail.mit.edu/6.5900/>

This self-assessment test is intended to help you determine your level of preparation for 6.5900 by going through some of the background material we expect you to have seen already. We also hope working through the problem set will help refresh your memory on these topics. We will only have a short review of this material in 6.5900.

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 and hand this in with your solutions. 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”. You should turn in solutions for problems where you entered “1” and “2”, but do not have to turn in solutions for problems for which you entered a “0”.

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 a lecturer to discuss your particular situation before the third week of class. Our experience is that most students with some background in logic design or assembly-level computer programming can complete the course successfully.

For this test only, you must work individually and turn in your own solutions. Do not discuss the problems with others.

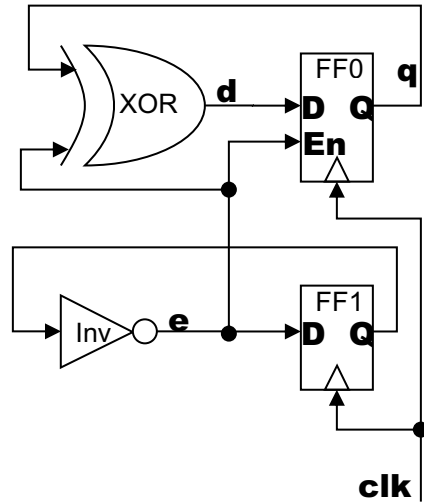
Problem 1

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

Problem 2

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



Draw a state transition diagram or fill out the state-transition table below for the sequential logic drawn above.

FF0	FF1	FF0'	FF1'

Problem 3

The followings are two code segments written in RISC-V 32-bit (RV32I) assembly language (refer to the accompanying 6.191 handout for information on the RV32I instruction set):

Segment A:

```
Loop: lw      x5, 0(x1)      # x5 ← Mem[x1+0]
      lw      x6, 0(x2)      # x6 ← Mem[x2+0]
      add     x5, x5, x6     # x5 ← x5 + x6
      sw      x5, 0(x3)     # Mem[x3+0] ← x5
      lw      x5, 0(x1)     # x5 ← Mem[x1+0]
      lw      x6, 0(x2)     # x6 ← Mem[x2+0]
      sub     x5, x5, x6     # x5 ← x5 - x6
      sw      x5, 0(x4)     # Mem[x4+0] ← x5
      addi    x1, x1, 8      # x1 ← x1 + 8
      addi    x2, x2, 8      # x2 ← x2 + 8
      addi    x3, x3, 8      # x3 ← x3 + 8
      addi    x4, x4, 8      # x4 ← x4 + 8
      bne     x1, x9, Loop   # branch to Loop if x1 ≠ x9
```

Segment B:

```
Loop: lw      x5, 0(x1)      # x5 ← Mem[x1+0]
      lw      x6, 0(x2)      # x6 ← Mem[x2+0]
      add     x7, x5, x6     # x7 ← x5 + x6
      sub     x8, x5, x6     # x8 ← x5 - x6
      sw      x7, 0(x3)     # Mem[x3+0] ← x7
      sw      x8, 0(x4)     # Mem[x4+0] ← x8
      addi    x1, x1, 8      # x1 ← x1 + 8
      addi    x2, x2, 8      # x2 ← x2 + 8
      addi    x3, x3, 8      # x3 ← x3 + 8
      addi    x4, x4, 8      # x4 ← x4 + 8
      bne     x1, x9, Loop   # branch to Loop if x1 ≠ x9
```

In both segments, assume x_1, x_2, x_3, x_4 initially hold valid memory addresses. Register x_9 is pre-computed to be 80 larger than the initial value of x_1 . All instructions operate on 32-bit values and the memory address space is byte-addressable.

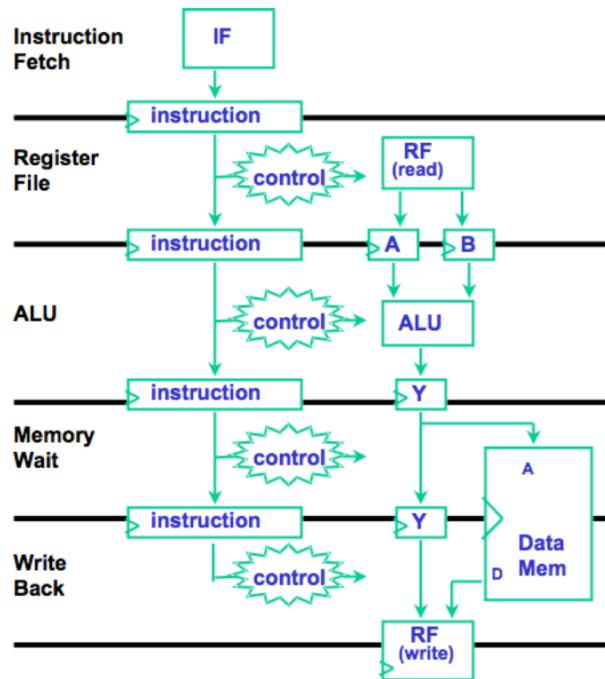
- If both segments are expected to perform the same task, can you guess what the task is? You can write the answer in C-like pseudo code.
- In general, which segment do you expect to perform better when executed?
- Can you specify initial conditions for x_1, x_2, x_3 , and x_4 which makes the two segments behave differently?

Problem 4

The following figure shows a 5-stage pipelined RISC-V 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 x1, x0, Label # branch to Label if x1 == x0`
 `add x2, x3, x5 # x2 ← x3 + x5`
 `sub x4, x5, x6 # x4 ← x5 - x6`

...

Label:

- b) `add x1, x0, x2 # x1 ← x0 + x2`
 `sub x4, x1, x2 # x4 ← x1 - x2`

...

- c) `lw x1, 0(x2) # x1 ← Mem[x2+0]`
 `add x3, x1, x2 # x3 ← x1 + x2`

...

Problem 5

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