

Solution:

Solution:

1 Binary Warmup (10 points)

For each of the following statements, determine whether it is either **true** or **false**. (1 point each)

1. All combinatorial circuits can be implemented purely out of NAND gates (using no other gates).

Solution: True. The NAND operation is a complete boolean base.

2. Abstraction means ignoring certain details in order to focus more clearly on important issues.

Solution: True. Not sure what else to say...

3. Revision control (“Subversion”) for a geographically distributed team should be based on locking.

Solution: False. If you’re “far from each other” locking is a bad idea since it may be hard to contact someone who has a file locked and ask them to unlock it in a timely manner.

4. Turing machines (defined in the 1930s) are strictly more powerful than today’s computers (2005).

Solution: True. Turing machines are a theoretical model with an **infinite** memory tape, something physical machines will never have.

5. The number 10001011 in binary is -116 in decimal (signed twos-complement interpretation).

Solution: False. It’s 117, so what can I say?

6. The drawback of iterative development processes is that customers are involved very little.

Solution: False. That’s the drawback of waterfall-style processes, iterative ones explicitly involve the customer in some regular fashion.

7. The λ -calculus expression $(\lambda x.xx)(\lambda x.xx)$ diverges, i.e. its evaluation continues “forever.”

Solution: True. What can I say?

8. Python’s motto “Batteries included!” alludes to the fact that Python programs are energy efficient.

Solution: False. It alludes to the fact that Python comes with a pretty big standard library that includes lots of stuff available only as “add ons” for other languages.

9. Runlength encoding leads to good compression only if the data in question is highly regular.

Solution: True. If things change a lot from byte to byte (or pixel to pixel, or whatever we’re compressing) runlength encoding never finds a big enough “stretch” to compress.

10. The multiplication of two n -bit numbers produces at most an $n + 1$ -bit result.

Solution: False. Addition leads to one “travelling” bit resulting from overflows that can eventually propagate all the way out. Multiplication, however, has n such potential travellers, so we get at most a $2n$ -bit result.

Solution:

2 Tough Choices

(8 points)

For each of the following questions, circle **one** answer out of the choices given. (2 points each)

1. Assume that `a = [1, 2, 3, 4, 5, 6, 7, 8, 9]` was entered into the Python interpreter. What is the result of evaluating `a[1:-1]` next?

- (a) `[]`
- (b) `[9]`
- (c) `[2, 3, 4, 5, 6, 7, 8]`
- (d) `[2, 3, 4, 5, 6, 7, 8, 9]`
- (e) None of the above.

Solution: That would be (c).

2. Consider the **exclusive or** (XOR) gate. Its output A is high if and only if its two inputs X and Y are different. Which of the following is the **disjunctive normal form** (DNF) for the XOR?

- (a) $A = X \cdot Y' + X' \cdot Y$
- (b) $A = (X + Y)'$
- (c) $A = (X' + Y') \cdot (X + Y)$
- (d) $A = (X \cdot Y)'$
- (e) None of the above.

Solution: That would be (a).

3. According to the Church-Turing thesis, which of the following is **not** a universal model of computation (i.e. not capable of expressing **all** computable functions)?

- (a) Turing machines
- (b) finite automata
- (c) λ -calculus

- (d) random access machines
- (e) None of the above.

Solution: That would be (b).

4. Which of the following is the encoding of the boolean function AND into λ -calculus? **Hint:** Remember that IF is encoded as $\lambda c.\lambda t.\lambda e.cte$ into λ -calculus!

- (a) $\lambda x.\lambda y.x$
- (b) $\lambda x.\lambda y.y$
- (c) $pq(\lambda x.\lambda y.x)$
- (d) $\lambda p.\lambda q.pq(\lambda x.\lambda y.y)$
- (e) None of the above.

Solution: That would be (b).

Solution:

3 Short Answer (8 points)

For each of the following questions, answer in **one to three** sentences, the shorter the better. (2 points each)

1. Write a Python function `reverse` that takes a list and returns the **reversed** list. For example, `reverse([1, 2, 3])` returns `[3, 2, 1]` while `reverse([])` returns `[]`.

Solution: `def reverse(ls): return ls[::-1]`

2. Explain the notion of **sign extension** as well as you can. When is it relevant? How is it performed? Give an example as well.

Solution: Sign extension is necessary when we want to move (or otherwise use) a quantity from a smaller to a bigger register. For a negative quantity, all “new” bits should be one, for a positive quantity all “new” bits should be zero. For example, when adding -1 stored in 8 bits to 10 stored in 32 bits, we need to “pad” the representation of -1 with 24 one bits before performing a 32 bit addition.

3. Explain the differences between the **blackbox** and **whitebox** (aka **glassbox**) approaches to software testing. Give an example as well.

Solution: In blackbox testing we rely on just the specification of a piece of code to derive test cases. For a sorting function, we know that any list of comparable elements needs to be changed into a list with (a) the same elements and (b) in sorted order; we design test cases accordingly. In whitebox testing we rely on the actual code to derive test cases. For example, if we find the number 42 in our sorting function, we know that for some reason the code will do something “special” for 42 and we design test cases around that number.

4. Design a **finite automaton** that accepts strings of the form $a(a|b)^*$, that is the symbol a followed by any number of the symbols a or b . **Hint:** You don't have to include an explicit error state.

Solution: Sorry, no time to draw this...

Solution:

4 Python Hacking

(14 points)

Remember the Python program to list all the **anagrams** of a given word using an existing dictionary? Develop, in as much detail as you have time for, a Python program that does exactly that. Explain briefly how your program is organized, but try to give as much “actual” code as possible as well. **You don't have to follow the design we did in class if you have a shorter/better/neater solution.**

Solution: See the solution we developed in class and posted later.

Solution: