1. [2 points] State 2 examples for each of the following algorithm types. So, all in all: 4 examples. Examples cited in the course slides are not counted.

Unsolvable:
- How many prime numbers do we have?
- Cite all floating point numbers between 1.0 and 2.0

Undecidable:
- What will be the temperature in 1 month from now?
- Predict the age of a person with 100% accuracy.

2. [1 point] State two differences between machine code and assembly language code.
- Machine code is in binary while assembly is in text.
- Assembly is a superset of the instruction set, while machine code contains only native instructions.
- Machine code does not contain labels but addresses.

3. For the following binary number: 10110
a. [1] What is the decimal equivalent if we interpret it as unsigned int?
   22
b. [1] What is the decimal equivalent if we interpret it as signed int?
   -10
c. [2] Take the number you calculated in “b” above and present it in IEEE 754 binary presentation of floating point single-precision.

\[-10 = - 01010 = - 1.01 \times 2^3 \text{ (exponent will be } 127 + 3 = 130 \rightarrow 10000010)\]

So: \[1 10000010 \ 010000...0\]

d. [1] Why 2’s complement is signed number presentation used in almost all computers?

- Easier implementation in hardware (can implement both adder and subtracter with the same circuit and few extra gates).
- More efficient representation of numbers (e.g. with 3 bits we can present from -4 to 3, which is better than 1s complement or sign-magnitude).


No, how can we determine the control signals that need to be generated without knowing the datapath?

b. [1] Can we design the control unit without knowing the instruction set? Justify.

No, how will the control unit be able to decode an instruction without knowing the instruction set?

c. [1] Can we design the control unit without knowing the high-level language that will be used to write programs on that computer? Justify

Yes, because the compiler/assembler shield the control unit from knowing the high-level language. Moreover, there are many HLLs and they evolve.
5. Given the following logic circuit:

![Logic Circuit Diagram]

a. [1] What is the logic expression (i.e. Boolean expression) that represents the above circuit?

\[ f = abc + ad + a'c \]

b. [1] What are all the values of \( a \) and \( c \) for which the value of \( f \) is 1?

From the 3\(^{rd} \) term in the Boolean equation, \( a=0 \) and \( c=1 \) will make \( f = 1 \). Note that we didn’t use the first term because it also depends on \( b \).

c. [1] Take only the AND gate with \( a, b, \) and \( c \) as input from the above figure. Substitute that gate with one or more NAND gates, yet give the same result.

Simply connect \( a, b, \) and \( c \) to a NAND gate, then connect the output of this NAND gate to the input of another NAND gate.

6. [1] Why do we need both ebp and esp to manage the stack in IA-32 ISA?

One keeps track of the top of the stack (esp), and the other (ebp) keeps track of the bottom of the current active stack frame. So, both esp and ebp determine the part in the memory that represents the active stack frame.

7. [1] After the following piece of C code, what will be printed?

```c
int a =0x20, b = 02;
a >>= b;
printf("a=%d, b=%d\n", a, b);
```

\( a= 8 \quad b=2 \)
8. [2] Implement, in C, the function whose definition is shown below (assume m and n are integers larger or equal to 0):

\[
A(m, n) = \begin{cases} 
  n + 1 & \text{if } m = 0 \\
  A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0 \\
  A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0 
\end{cases}
\]

```c
int A(int m, int n)
{
    if (!m)
        return n+1;
    if (!n) // Remember that we assume m and n larger or equal to 0
        A(m-1, 1);
    else
        A(m-1, A(m, n-1));
}
```

9. [1] In algorithm analysis, we mostly care about large values of n and worst-case scenario. Why is that?

- For small values of n, it doesn’t matter which algorithm to use, as current machines are already fast enough to provide adequate performance.
- The worst case scenario is the conservative approach by which we can decide which machine to use given an algorithm, or to compare two algorithms. The best case scenario seldom happens and the average case is hard to calculate.

10. [1] List a scenario where executing an O(n) algorithm can be faster than O(logn) algorithm even if the same machine and same input are used.

If the o(n) algorithm accesses the memory way more frequently than the o(logn) algorithm. Memory is orders of magnitude slower than computation these days, especially if computations are simple.