This task-based evaluation corresponds to labs #3, 4 and 5.

You have half an hour to solve this sheet.

Q1 [10]: Say you write the following line of code in a C/C++ program:

```
uint16_t x = 241, flag = 0;

if (2 % x == 0)
    flag = 1;
```

What assembly code do you think the above logic would get compiled to, i.e write equivalent assembly to accomplish the above (~7 instructions).

Feel free to assume that: - `flag` is a memory variable - x will get stored in the `si` register - No compiler optimisations are enabled

---

Q2 [10]: Assume you have a list of words in memory, sorted contiguously in ascending order, marked by the label `arr`. A new word is inserted at `arr + si`. Your task is to move this newly 'appended' word to its correct position such that the array maintains its sortedness. You can assume that before `arr` lies 0xFFFF (~ 8 instructions).

Q3 [05]: (BONUS) If the if condition from question #1 were `x % 2` instead, would it get compiled to a more efficient solution? Justify your answer (~2 lines).

Q4 [20]: (BONUS) Below is the code for a 'bogosort' in Python:

```
import random

def bogosort(arr):
    while arr != sorted(arr):
        random.shuffle(arr)
    return arr
```

Recreate it in x86 assembly. You can simulate randomness like so:

```
1. Take an arbitrary number of bits of your choosing from an arbitrary number in the array
2. XOR/OR/AND them together
3. Shift/rotate the result left/right by an arbitary number of bits, and use it as the random index
```

Or like so:

```
1. Select a large, random 16 bit number as a seed
2. Multiply the seed by a large number; storing the result as the new seed
3. Divide the seed by the length of the array, and use the remainder as the random index
```

Any other approach of your own is fine too, as long as you can explain whichever approach you chose produces reasonably random numbers.