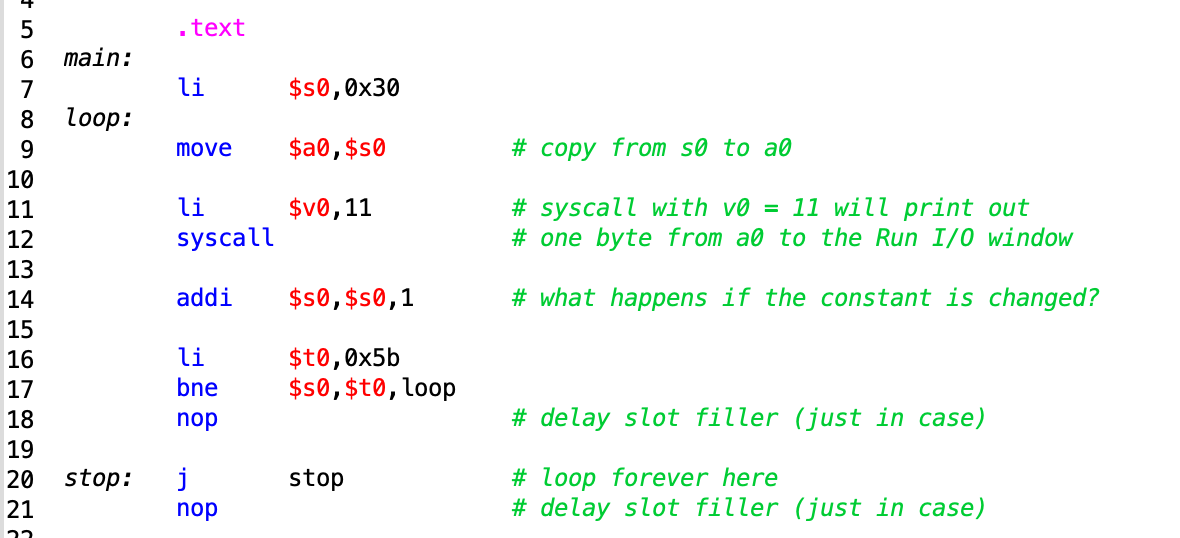
Laboratory 1 tasks

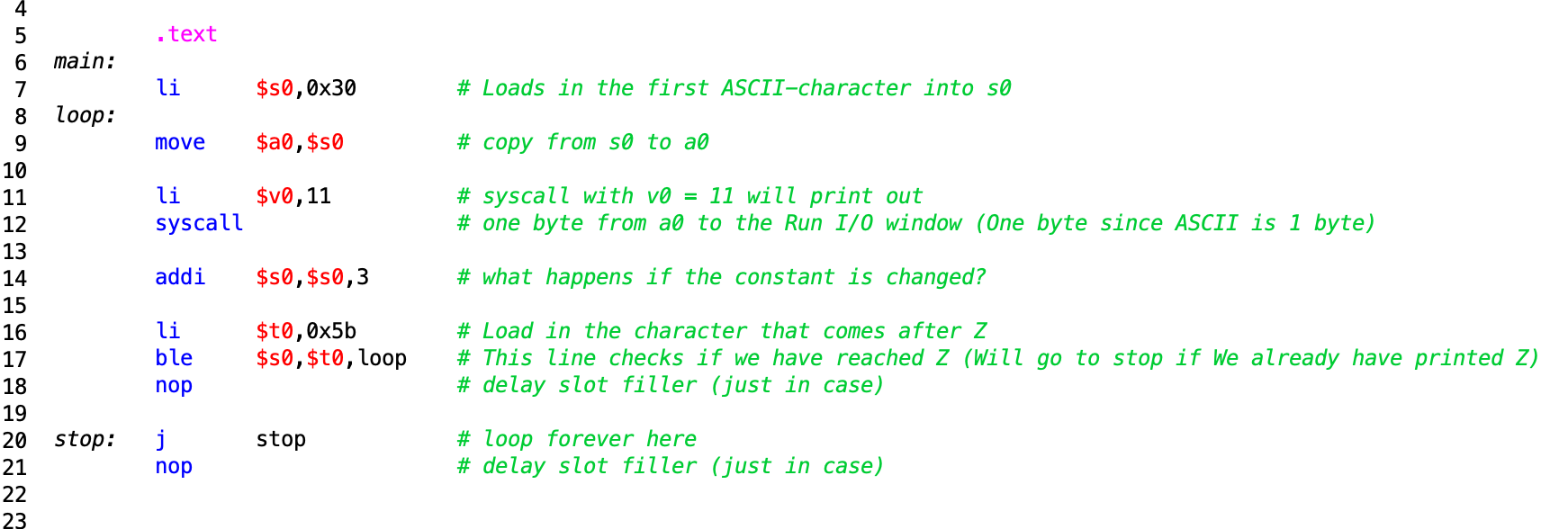
Alexander Lundqvist

**Task 1**

Which lines of code had to be changed? Why?

This is the starting code of analyze.asm. It prints out all the ASCII-characters from 0-Z in the run I/O console. The task is to change the code so that only each third character is printed.

ASCII-characters take up one byte (8 bits or 2 hex numbers). The hex range is from 0x00000030 to 0x0000005A. To produce the output specified in the task, we will change line 14 and line 17.

This is because in line 14, we previously added only one, and therefore incremented the ASCII-character to the next. (0x30 = 0, 0x31 = 1,…, 0x59 = Y, 0x5A = Z). So to get the each third ASCII-character we add 3 instead.

The previous line at 17 checked if the character was equal to the character after Z, which would signal a stop. But by taking each third we would actually skip the instance that would signal a stop and the program would loop forever. By instead changing it to ble instead, we will check if the current char is less than our “stop” char ( 0x5B = [ ).

**Task 2**

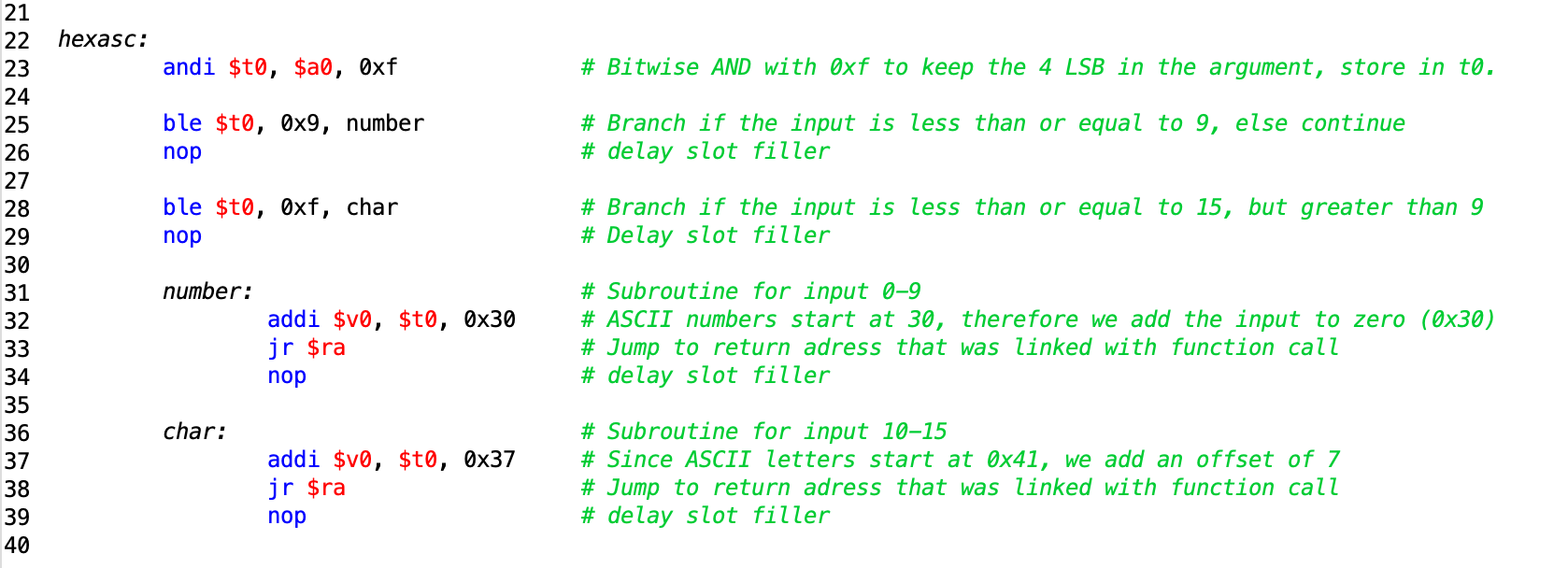
Note to teachers and students: No s-registers may be used, and no registers should be saved.

Your subroutine hexasc is called with an integer-value as an argument in register $a0, and

returns a return-value in register $v0. If the argument is 17, what is the return-value? Why?

If your solution contains a conditional-branch instruction: which input values cause the

instruction to actually branch to another location? This is called a taken branch.

To keep the last for bits (4 LSB), we perform a immediate and operation on our “input” register $a0 with the value 0x0000000F, since bitwise AND operations keep the value of a bit if it is ANDed with a one, and discard the value if it is a zero.

If we input anything higher than 15, we will get wrong answer. 17 for example will become 1. This is because we are only considering the 4 LSB in the input. And because 17 is 11 in hex, we get 1. 76 (0100**1100**) would become **C.**

We use conditional branching (ble) to determine where to continue the program. If the “input” is less than or equal to 9, we go to the label number. If it is less than or equal to 15, we jump to label char.

**Task 3**

Note to teachers and students: check register-usage and saving/restoring carefully.

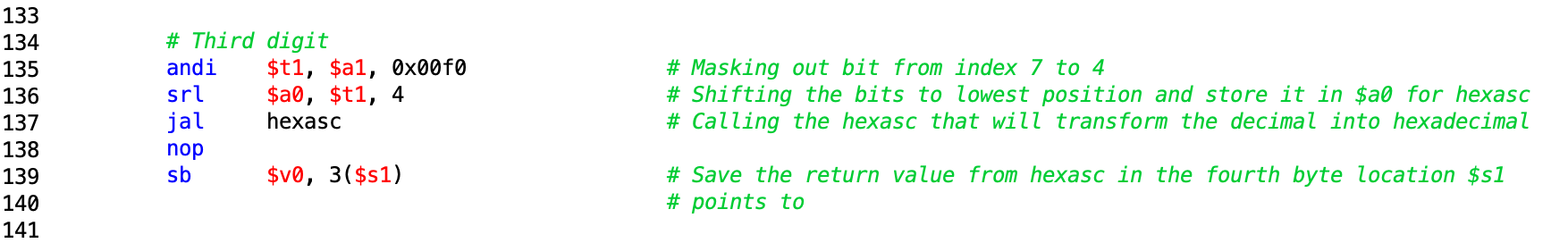
Which registers are saved and restored by your subroutine? Why?

Which registers are used but not saved? Why are these not saved?

Assume the time is 16:53. Which lines of your code handle the '5'?

Register usage

Saved no saved

This is the code segment that handles the 3 digit.

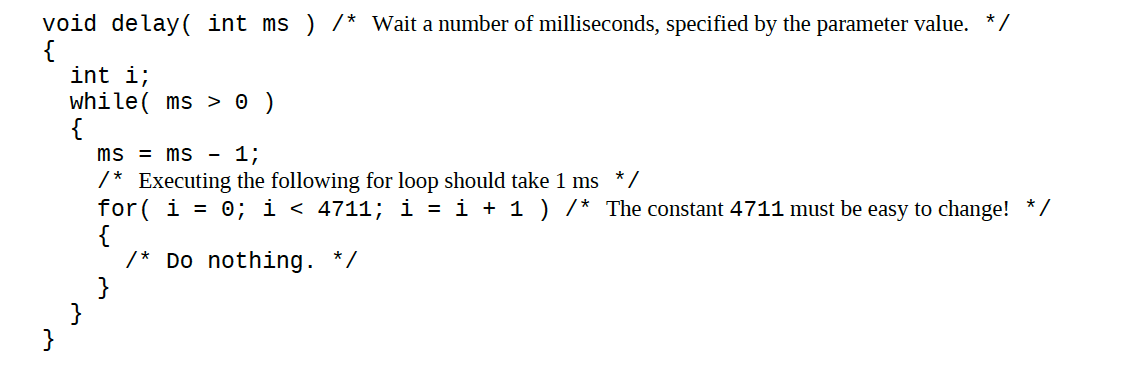
**Task 4**

Note to teachers and students: check that the assembly code matches the C code.

If the argument value in register $a0 is zero, which instructions in your subroutine are

executed? How many times each? Why?

Repeat the previous question for a negative number: -1.

The following C-code could be written in assembly as following:

INSERT IMAGE

Asskjnflölm

**Task 5**

**Task 6**

What is the effect of the assembler directive .global? Why is the directive particularly

important in this assignment? The teachers will help you with this if necessary.

**Task 7**

When you move your code from the simulator to the lab-board, you have to change the

value of the constant in the delay subroutine to get correct timing. Why?