1. Preparation [Total max. mark: 1]

**2. Explain stored-program concept [0.5].**

The stored program concept extends from the idea that instructions and data of many types can be stored in memory as binary numbers. This will allow the computer to hold a program in memory also, that can be accessed, with the intention to perform a variety of tasks.

**3. Consider three types of information: *calculated values*, *ASCII characters*, and *instructions* to be executed. Explain what out of the three types of information (e.g. some of them; all of them; or none of them) can be found in the memory of MIPS computers? Justify your answer [0.5].**

Answer: All three.

For the stored program concept to be possible, all these things must be found in the memory of MIPs computers.   
  
1) Calculated values – calculated values can be stored in memory by instructions like sw.

2) ASCII characters – strings are stored in memory when the program codes are loaded.

3) Instructions – instructions are stored in memory when the program codes are loaded.

2. Workshop Tasks [Total max. mark: 2]

**2. Open *simplecalc.s* in a text editor and analyse the program. Identify its text segment, data segment, and the registers used in it. Run the program in PCSpim, and observe its results in the console window.  
Question (0.5 marks):  
a. Write on a piece of paper the formula for the calculation performed by the program *simplecalc.s*,  
and perform this calculation manually, on paper, using number values defined in *simplecalc.s.***

**F = (a+b) - (c-d)**

f = (12-2) – (13-3)  
  
f = 0  
  
**b. write what the program prints in the console window (before the errors corrected in d).**

“26850099200.00000000

... Thank you :)”  
  
Not what I have calculated.  
  
**c. are the results from (a) and (b) identical? If not, explain why.**

No. The calculation is correct; and we can find in the Qtspim simulator that the calculated value is the same; however, when we look at the console, the person observing the program cannot see the correct answer in the console, and in this way it is different. The reason this is occurring is because “system call” operation is not being executed correctly. In case 1; we notice that $v0 was being loaded with 2 and this will cause it to print a floating point integer; in this case it will print the address. In the second case, the value of “1” was placed into register $v0(when it should have been four), causing the system to mistakenly print an integer (0).

# print string

li $v0, 4 # ERROR HEJA NOTE: It should be 4; since we want to print a string not 2

la $a0, extra # \_\_?? Prints extra at message section

syscall

# print string

li $v0, 4 # ERROR HEJA NOTE: it should be 4 not 1;

la $a0, message # \_\_?? ## Prints Message from Syscall at the top

syscall

**d. Two lines of the code are marked “ERROR“. Correct the errors in both lines.**

The code is not identical because the system call is executing the wrong instruction. The value which was loaded immediately was incorrectly assigned; in one case; the system call executed the instruction to print a floating point number. And we can see that it printed a floating point number.   
 **3. Question (1 mark): add ALL missing comments (everywhere you see “\_\_??”). You need to  
understand what the program does (the formula for the calculation performed by the program), and what is the purpose of each line of the code.**

Completed.

**4. Question (0.5 marks): Write a list of all registers used in the program, and write next to each register what was its role (for example: $ra – used to store return address). Also, compare the registers used in the program with “Policy of Registers Use Convention” table (see the lecture notes and the textbook). Is the program using the registers in accordance with the convention?**

Yes.

$s0 Stores the value of f (the final calculation) #Used in accordance with convention

$s1 Stores the value of a single variable (12) #Used in accordance with convention

$s2 Stores the value of a single variable (-2) #Used in accordance with convention

$s3 Stores the value of a single variable (13) #Used in accordance with convention

$s4 Stores the value of a single variable (3) #Used in accordance with convention

$t0 Adds the values of $s1 and $s2; and places it into $t0 #Used in accordance with convention

$t1 Adds the values of $s2and $s3; and places it into $t1 #Used in accordance with convention

$s0 subs the values of the temporaries in $t0, $t1 #Used in accordance with convention

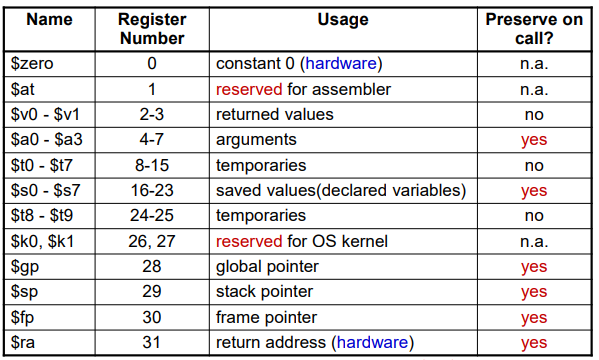
$s7 Stores return Address #Used in accordance with convention

$v0 loads an immediate value for system call [execution] #Used in accordance with convention

$a0 load address with values in messages #Used in accordance with convention

$ra Return Address (to return to point of origin) #Used in accordance with convention

$0 A register which has the value of 0(zero) #Used in accordance with convention

****