# LAB – C

## C3a.asm

ECALL for input

ecall aka environment call exception is used to invoke the OS. When compiled and run the code none of the register values change, this is because the ecall instruction waits till a value is input during run time, and not compile time.

When the program runs, the program waits till the 1st input value is entered to the input/output window. When the 1st value is entered, it goes to the second instruction and waits for the second input value, and then adds the 2 values

Format of the instruction

**ecall <rd – destination reg>, <rs source reg>, <syscall code>**

in input rs is taken as 0.

Syscall codes

5 – integer

6 – float

7 – hex

8 – char

9 – string

Eclass x6, x0, 5

Means the input expected is integer, the input will be saved in reg x6.

## C3b.asm

ECALL for output

Ecall can be used to print the output to the output window

**ecall <rd – destination reg>, <rs source reg>, <syscall code>**

Syscall codes

0 – integer

1 – float

2 – hex

3 – char

4 – string

The rd register will just be used to take the program to a new line after the output is printed. To take it to the new line use x0 as the rd register.

Note that now output is also printed in the IO window

## C3c.asm

Here beq is used to take the program to run as a loop

C3d.asm

DC – define characters. This is similar to printf in C, you can display a string of characters using DC and outputting the message on the IO window.

Text

Description automatically generated

C1 – has the address of the memory location which holds the LSB of the string “integer: “

C2 – has the address of the memory location which holds the LSB of the string “sum: “

S1 – has the address of the memory location which holds the LSB of the string “Inputs two integers\nand prints the sum.\0: “

Since this is little endian format the LSB is placed on the byte with lowest address

Eg: if we take the string “integer: ” “i” is the first letter in the string which goes to the LSB in memory, the <space> character is going to be the MSB

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | : | r | e | g | e | t | n | i | Character string |
| MSB |  |  |  |  |  |  |  | LSB |  |
| 20 | 3A | 72 | 65 | 67 | 65 | 74 | 6e | 69 | ASCII of the string |

3A to 69 is saved on memory address 0x000, the 20 is saved on the next byte 0x008

The addi x30, x0, s1 is used to put in the address of the LSB to the register x30

Next we have ecall x0, x30, 4 meaning this is a output environment call, tells the program to print the value pointed by x30 as a string. X30 points to the LSB of s1, and it prints the whole string to IO window until the end of string \0 is reached.

Then we have ecall x6, x28, 5 which is a input ecall. This says to get the input and save it to x6, the rs register x28 is used as an input prompt if its not 0. So here the pointer in x28 points to c1, so that is printed on to the IO window.

Same happens in the next line, the input value is saved to x7

Add x5, x6, x7 is a regular add operation

Ecall x1, x29, 3

Here the instruction is a char output type instruction. The value in x29 is converted to char and printed on IO window. So “sum: “ is printed on IO window. Note that the rd is x1, and not x0. So here if we put x0 instead of x1, the output will be printed on the next line, but x1 is

Ecall x0, x5, 0

Here the instruction is an output, prints the value at x5 into IO window and goes to a newline