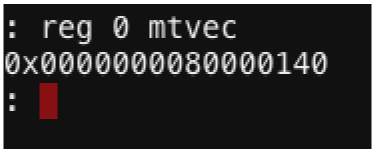
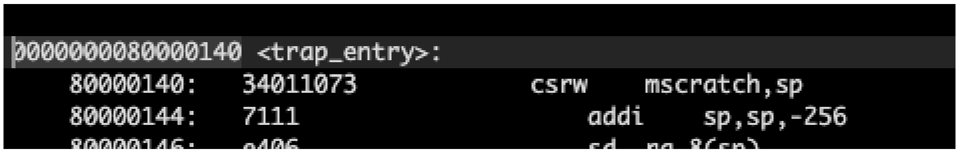
Lab 05 Questions - Leonardo Munoz

a.) Address of the main function is 0x000000008000026c.

b.) The value of the mtvec register: 0000000080000140. The address of the trap_entry procedure is 0000000080000140. As we can see, the contents of register mtvec and the address of trap_entry are the same. They are the same because mtvec is actually used by the hardware to store the address of the first instruction to be executed when a trap is detected. In other words, once the hardware reads a trap, it knows that mtvec will have the address which itself contains the proper handling for that trap.

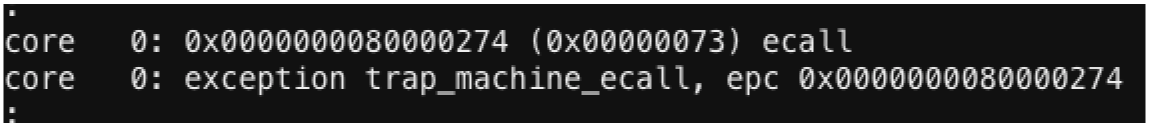


```
: reg 0 mtvec
0x0000000080000140
: 
```



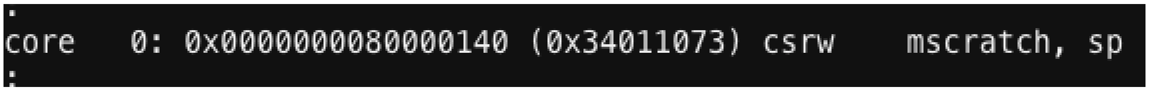
```
0000000080000140 <trap_entry>:
    80000140:   34011073            csrw    mscratch,sp
    80000144:   7111                    addi    sp,sp,-256
    80000146:   e406                    sd  ra,8(sp)
```

c.) When the first ecall instruction is read, the hardware identifies it as such and calls 'exception trap_machine_ecall'



```
:
core   0: 0x0000000080000274 (0x00000073) ecall
core   0: exception trap_machine_ecall, epc 0x0000000080000274
:
```

As we can see, right after the ecall is made, the very next instruction is:



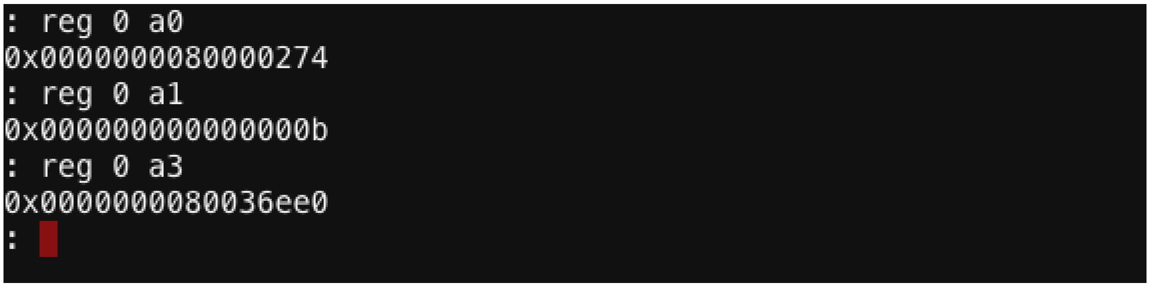
```
:
core   0: 0x0000000080000140 (0x34011073) csrw    mscratch, sp
:
```

Which means that our 'main' has been interrupted and now we will be executing code from 'trap_entry.' We can tell because the address of this instruction is 0000000080000140, which is the address of 'trap_entry.'

d.) 'trap_entry' starts off by creating a stack of 256 bytes. It appears that it saves a grand number of registers onto that stack before doing anything else. I presume this is the case because 'trap_entry' would want to keep all the data used by our program intact while it's dealing with the trap.It has one jump statement, which jumps to address 8000028e, which is the address for 'handle_trap.' It then has two different branch statements, one that leads to 'next1', and another that leads to 'next2' which itself leads to 'next3.' In the latter two functions, we can see that the contents which were loaded onto the stack are now being loaded back into the respective registers.

e.) The address for ‘handle_trap’ is 8000028e.

f.) The values for a0, a1 and a3 are as follows:

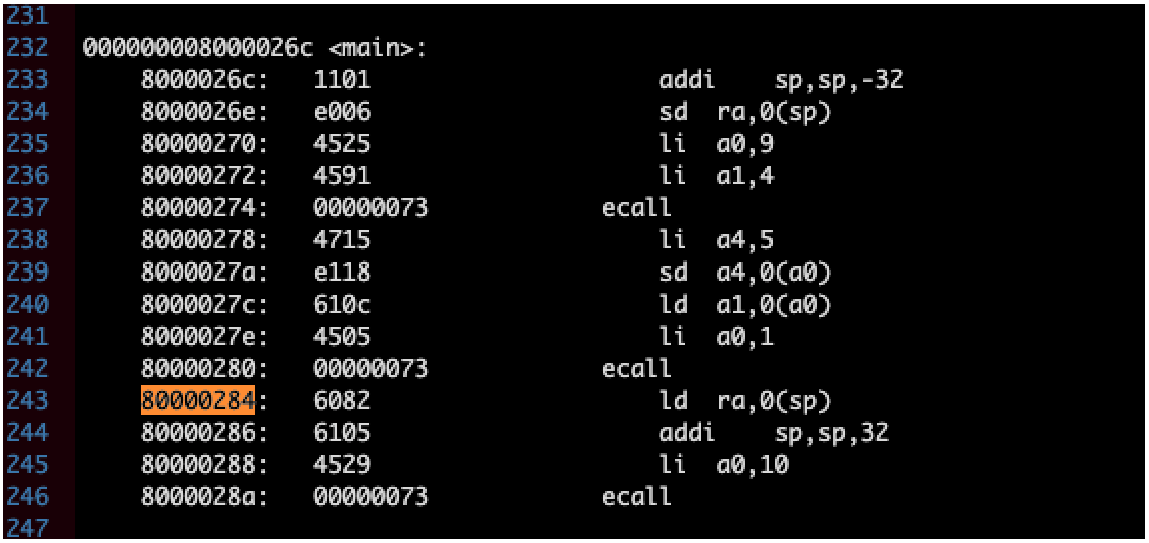


```
: reg 0 a0
0x0000000080000274
: reg 0 a1
0x000000000000000b
: reg 0 a3
0x0000000080036ee0
: 
```

a0 holds the value of the address of the ecall that caused the program to enter trap_handling mode. However, I could not find any connection between the code that had been executed and the values in a1 and a3.

g.) ‘handle_trap’ handles the ecall execution. Inside this function we see that it counts with multiple branch statements that allow the function to skip over parts of itself. Most importantly however, it has a jump to the functions ‘printf’, ‘_sbrk’ and ‘_exit.’

h.) The value of register mepc is 80000284. This points to the following instruction in the main function:



```

000000008000026c <main>:
    8000026c:   1101                    addi    sp,sp,-32
    8000026e:   e006                    sd  ra,0(sp)
    80000270:   4525                    li  a0,9
    80000272:   4591                    li  a1,4
    80000274:   00000073            ecall
    80000278:   4715                    li  a4,5
    8000027a:   e118                    sd  a4,0(a0)
    8000027c:   610c                    ld  a1,0(a0)
    8000027e:   4505                    li  a0,1
    80000280:   00000073            ecall
    80000284:   6082                    ld  ra,0(sp)
    80000286:   6105                    addi    sp,sp,32
    80000288:   4529                    li  a0,10
    8000028a:   00000073            ecall

```

It represents the program coming back to the main function. It skips over the second ecall since it seems it was already handled.

i.) It returns control to the main function.