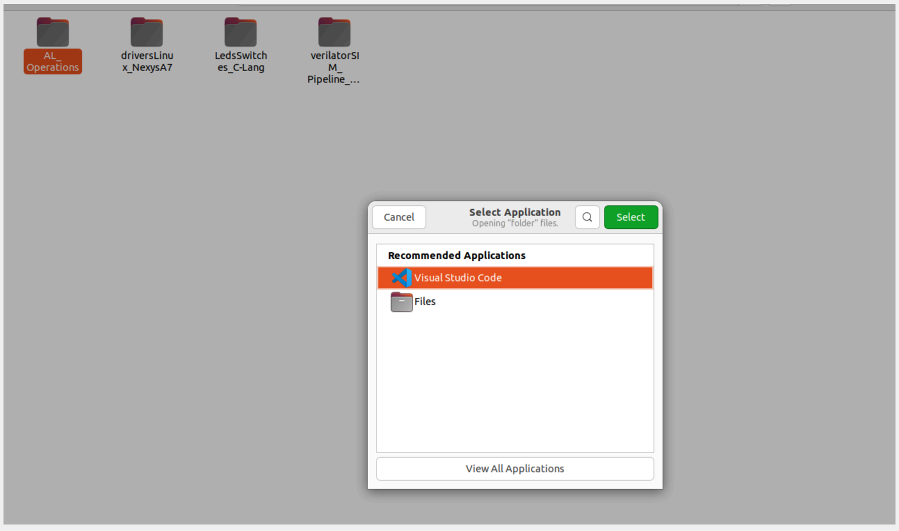
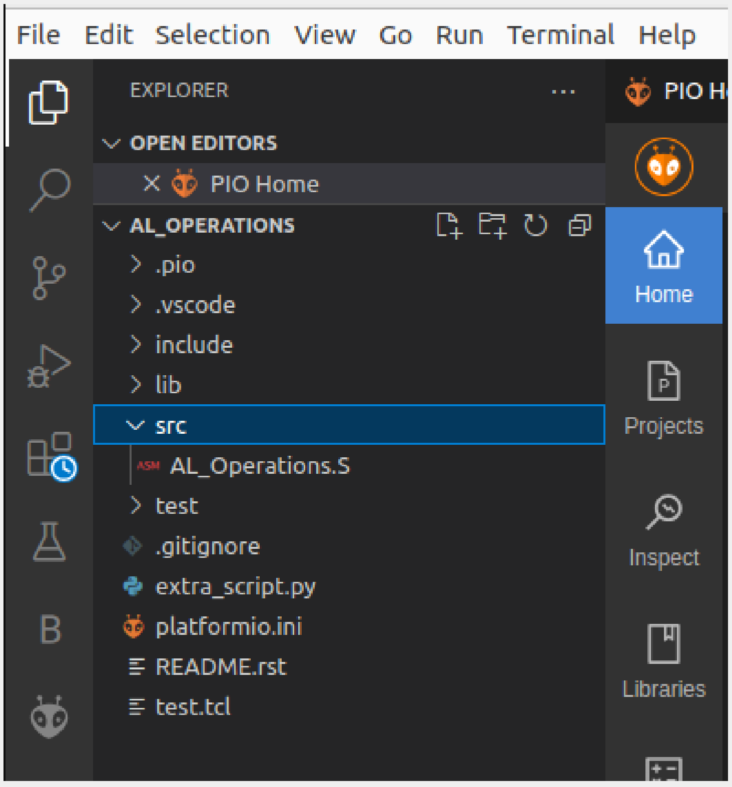
Module: R5: RV-fpga
Section: Installations **Task:** Tools

Task 1.5

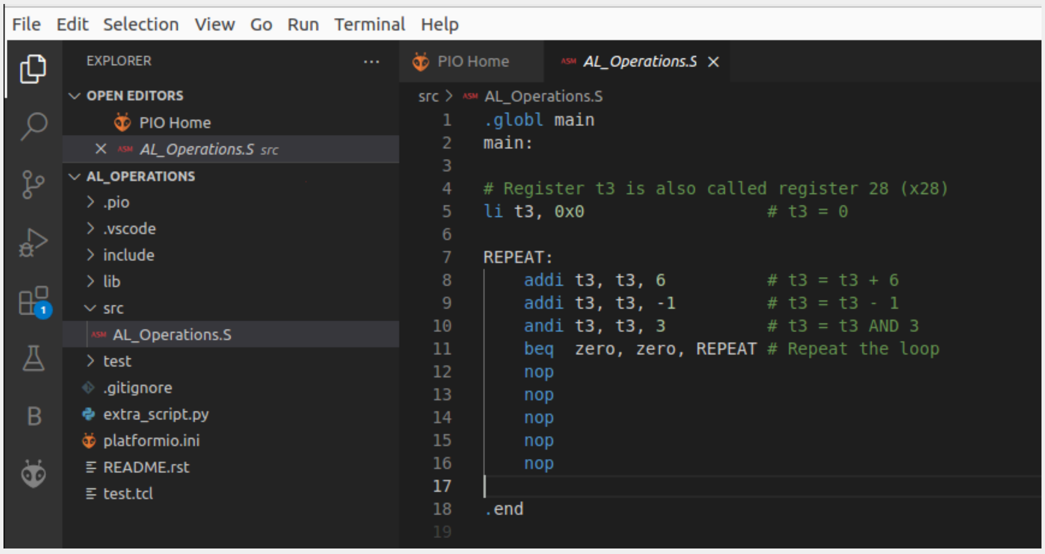
RVfpga-Whisper

- ➢ **Testing:**
  - ■ **RVfpga-Whisper**
    1. Open the specified folder of example program in VS Code:



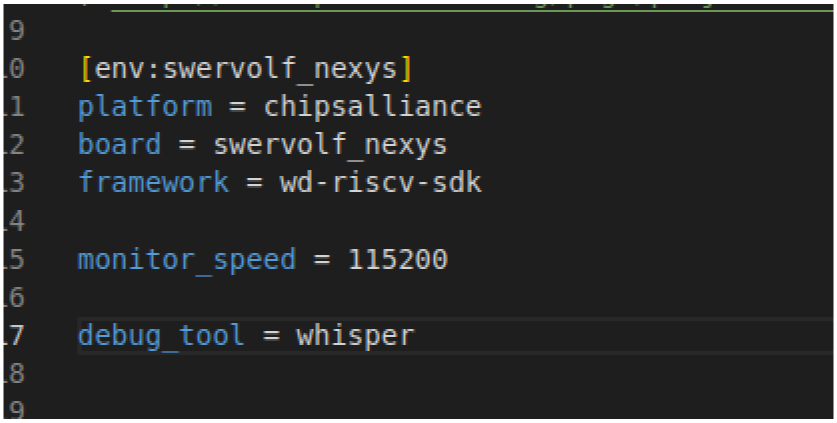


2. PlatformIO will now open this program, which includes three assembly arithmetic-logic instructions (addition, subtraction, and logical and) on the same register, t3 (also called x28), within an infinite loop. We can view the program by expanding the src folder and double-clicking on **AL_Operations.S**.



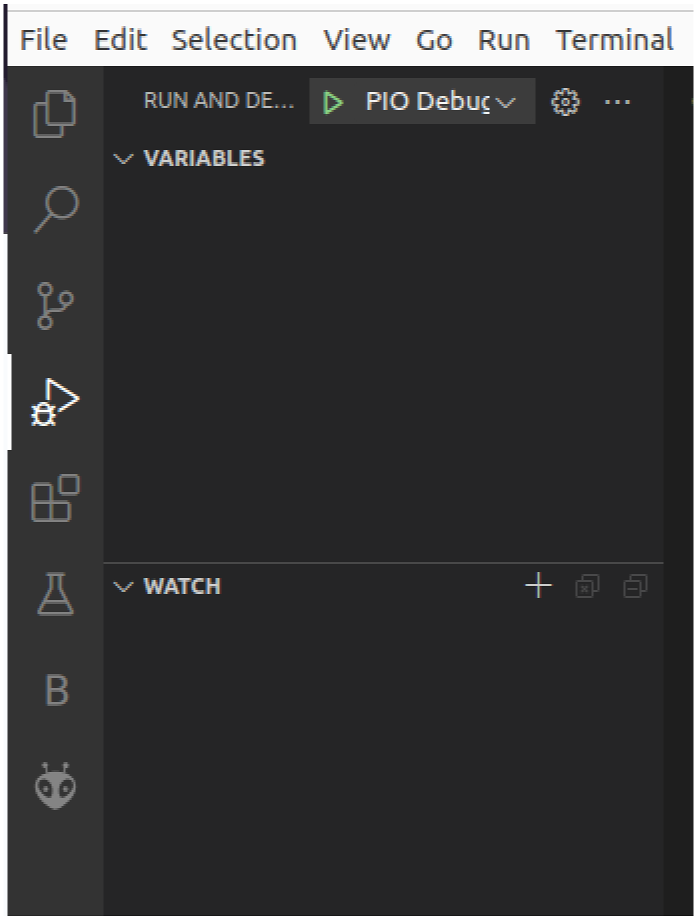
```
.globl main
main:

# Register t3 is also called register 28 (x28)
li t3, 0x0                  # t3 = 0

REPEAT:
    addi t3, t3, 6          # t3 = t3 + 6
    addi t3, t3, -1         # t3 = t3 - 1
    andi t3, t3, 3          # t3 = t3 AND 3
    beq  zero, zero, REPEAT # Repeat the loop
    nop
    nop
    nop
    nop
    nop

.end
```

3. Open file **platformio.ini**. Set Whisper as the simulation tool to use the debug tool by uncommenting line 17. Save the file.

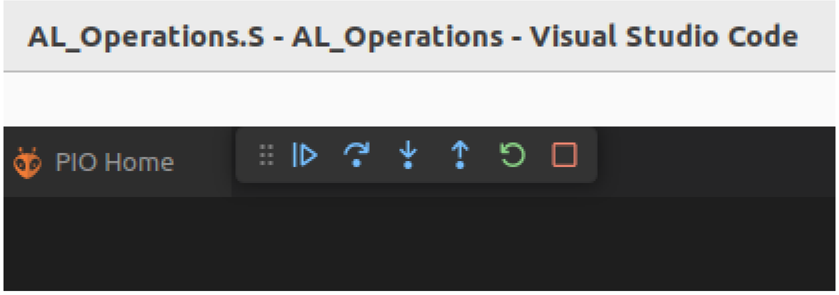


```
9
0    [env:swervolf_nexys]
1    platform = chipsalliance
2    board = swervolf_nexys
3    framework = wd-riscv-sdk
4
5    monitor_speed = 115200
6
7    debug_tool = whisper
8
9
```

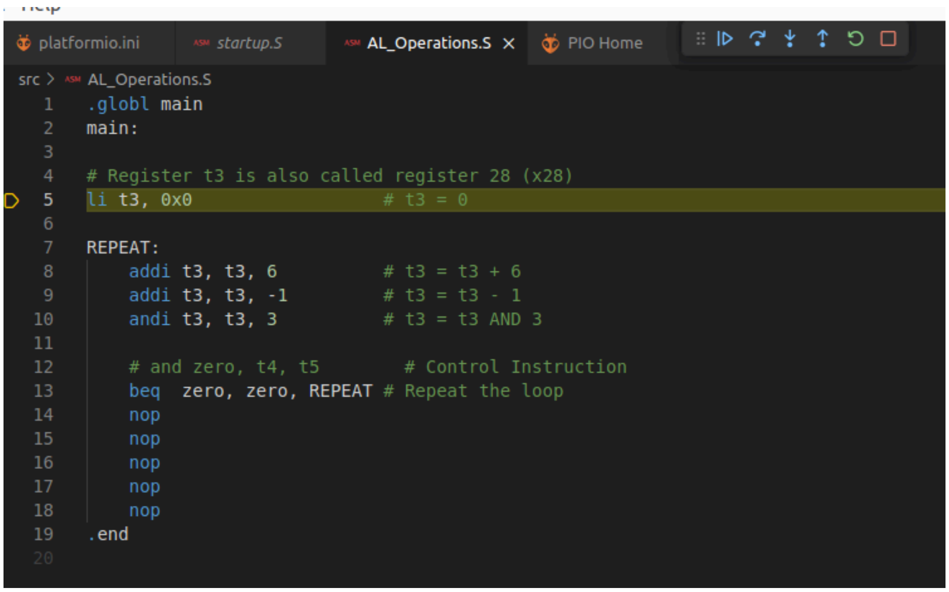
4. Click on the RUN AND DEBUG button in the bar on the left-hand side.
   - Start the debugger by clicking on the Play button PIO Debug (make sure that the "PIO Debug" option is selected).



The program will first compile and then debugging will start. To control your debugging session, you can use the debugging toolbar which appears near the top of the editor.

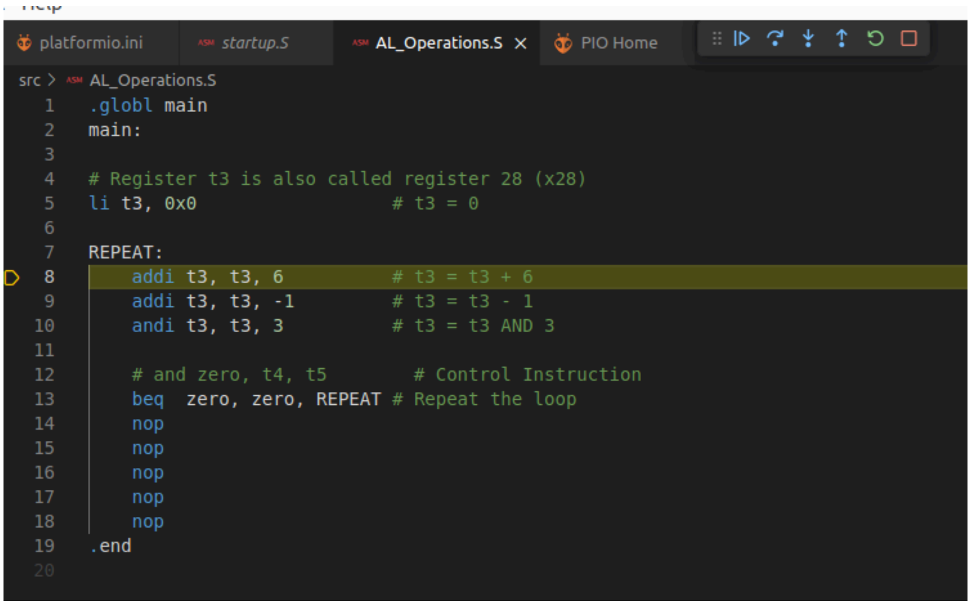


5. PlatformIO will set a temporary breakpoint at the beginning of the main function and we can continue execution step by step and analyze the three arithmetic-logic instructions.



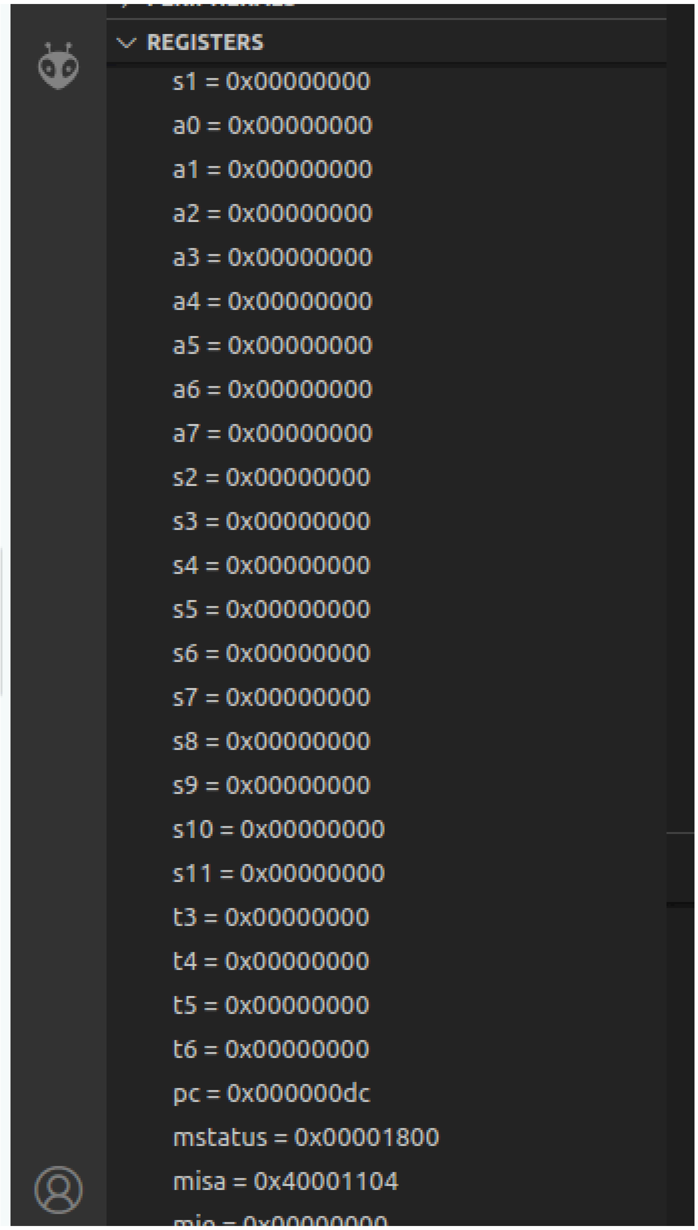
```
.globl main
main:

# Register t3 is also called register 28 (x28)
li t3, 0x0                  # t3 = 0

REPEAT:
    addi t3, t3, 6          # t3 = t3 + 6
    addi t3, t3, -1         # t3 = t3 - 1
    andi t3, t3, 3          # t3 = t3 AND 3

    # and zero, t4, t5        # Control Instruction
    beq  zero, zero, REPEAT # Repeat the loop
    nop
    nop
    nop
    nop
    nop
.end
```

6. Let's step over. This is the first arithmetic-logic instruction, which adds t3, which is initially zero, plus 6.



```
.globl main
main:

# Register t3 is also called register 28 (x28)
li t3, 0x0                  # t3 = 0

REPEAT:
    addi t3, t3, 6          # t3 = t3 + 6
    addi t3, t3, -1         # t3 = t3 - 1
    andi t3, t3, 3          # t3 = t3 AND 3

    # and zero, t4, t5        # Control Instruction
    beq  zero, zero, REPEAT # Repeat the loop
    nop
    nop
    nop
    nop
    nop
.end
```

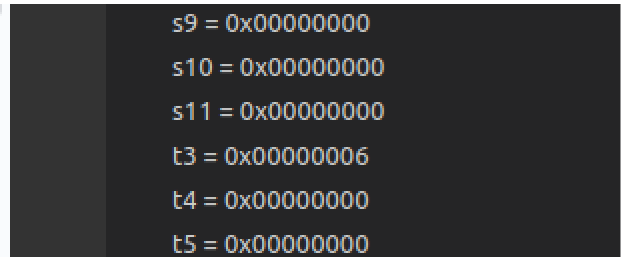
7. Let's open the Registers view, so that we can analyze the values of the t3 register.



REGISTERS

```
s1 = 0x00000000
a0 = 0x00000000
a1 = 0x00000000
a2 = 0x00000000
a3 = 0x00000000
a4 = 0x00000000
a5 = 0x00000000
a6 = 0x00000000
a7 = 0x00000000
s2 = 0x00000000
s3 = 0x00000000
s4 = 0x00000000
s5 = 0x00000000
s6 = 0x00000000
s7 = 0x00000000
s8 = 0x00000000
s9 = 0x00000000
s10 = 0x00000000
s11 = 0x00000000
t3 = 0x00000000
t4 = 0x00000000
t5 = 0x00000000
t6 = 0x00000000
pc = 0x000000dc
mstatus = 0x00001800
misa = 0x40001104
mie = 0x00000000
```

8. When we step over this instruction, t3 goes to 6, then it is updated to 5, and finally, the and instruction sets t3 to 1.



```
s9 = 0x00000000
s10 = 0x00000000
s11 = 0x00000000
t3 = 0x00000006
t4 = 0x00000000
t5 = 0x00000000
```

9. We can continue, and the second iteration will execute.