WEEK 1 TASK 1

- For the counter task, I changed the code a little bit and observed the changes in the RISC-V machine code.

```c
#include <stdio.h>
#include <time.h>

void delay(int n) {
    int us = n; // microseconds
    clock_t start_time = clock();
    while (clock() < start_time + (us * CLOCKS_PER_SEC / 1000000));
}

int main()
{
    int count = 0x00000000;
    while (1)
    {
        printf("Count value is: %d\n", count);
        count++;
        if(count==16){
            count=0;
        }
        delay(500);   // delay by 0.5 microseconds
    }
}
```

```c
#include <stdio.h>
#include <time.h>

void delay(int n) {
    int us = n; // microseconds
    clock_t start_time = clock();
    while (clock() < start_time + (us * CLOCKS_PER_SEC / 1000000));
}

int main()
{
    int count = 0x00000002;
    while (1)
    {
        printf("Count value is: %d\n", count);
        count++;
        count++;
        if(count==16){
            count=2;
        }
        delay(500);   // delay by 0.5 microseconds
    }
}
```

Figure 1: counter C codes

In the Figure, the code on the left hand side counts one by one and starts at zero, while the code on the right hand side, counter is incremented by twice at each step and starts from 2.

```
1  delay(int):
2          addi    sp,sp,-16
3          sw      ra,12(sp)
4          sw      s0,8(sp)
5          mv      s0,a0
6          call    clock
7          add     s0,s0,a0
8  .L2:
9          call    clock
10         blt     a0,s0,.L2
11         lw      ra,12(sp)
12         lw      s0,8(sp)
13         addi    sp,sp,16
14         jr      ra
15 .LC0:
16         .string "Count value is: %d\n"
17 main:
18         addi    sp,sp,-16
19         sw      ra,12(sp)
20         sw      s0,8(sp)
21         sw      s1,4(sp)
22         sw      s2,0(sp)
23         li      s0,0
24         lui     s2,%hi(.LC0)
25         li      s1,16
26 .L7:
27         mv      a1,s0
28         addi    a0,s2,%lo(.LC0)
29         call    printf
30         addi    s0,s0,1
31         sub     a5,s0,s1
32         snez    a5,a5
33         neg     a5,a5
34         and     s0,s0,a5
35         li      a0,500
36         call    delay(int)
37         j       .L7
```

Figure 2: RISC-V GCC

```
1  delay(int):
2          addi    sp,sp,-16
3          sw      ra,12(sp)
4          sw      s0,8(sp)
5          mv      s0,a0
6          call    clock
7          add     s0,s0,a0
8  .L2:
9          call    clock
10         blt     a0,s0,.L2
11         lw      ra,12(sp)
12         lw      s0,8(sp)
13         addi    sp,sp,16
14         jr      ra
15 .LC0:
16         .string "Count value is: %d\n"
17 main:
18         addi    sp,sp,-16
19         sw      ra,12(sp)
20         sw      s0,8(sp)
21         sw      s1,4(sp)
22         sw      s2,0(sp)
23         li      s0,2
24         lui     s2,%hi(.LC0)
25         li      s1,16
26         j       .L7
27 .L6:
28         li      a0,500
29         call    delay(int)
30 .L7:
31         mv      a1,s0
32         addi    a0,s2,%lo(.LC0)
33         call    printf
34         addi    s0,s0,2
35         bne     s0,s1,.L6
36         li      s0,2
37         j       .L6
```

Figure 3: RISC-V GCC

We can observe that initial value of the count variable is loaded on the 23rd line in figure 2 and figure 3. ‘li’ instruction loads initial value to s2 register.

Increment of the count variable is done by ‘addi’ instruction. It can be seen that while in figure 2 on 30th line, it is incremented by one, and in figure 3 on 34th line it is incremented by two.

When I changed the loaded value in the if statement, ‘bne’ instruction is used and can be seen in figure 3 on 35th line. It compares register s0 with s1.

```
Count value is: 0
Count value is: 1
Count value is: 2
Count value is: 3
Count value is: 4
Count value is: 5
Count value is: 6
Count value is: 7
Count value is: 8
Count value is: 9
Count value is: 10
Count value is: 11
Count value is: 12
Count value is: 13
Count value is: 14
Count value is: 15
Count value is: 0
```

```
Count value is: 2
Count value is: 4
Count value is: 6
Count value is: 8
Count value is: 10
Count value is: 12
Count value is: 14
Count value is: 2
Count value is: 4
```

Figure 4: Executor Result

[Godbolt link to increment 1 counter]

[Godbolt link to increment by 2 counter]