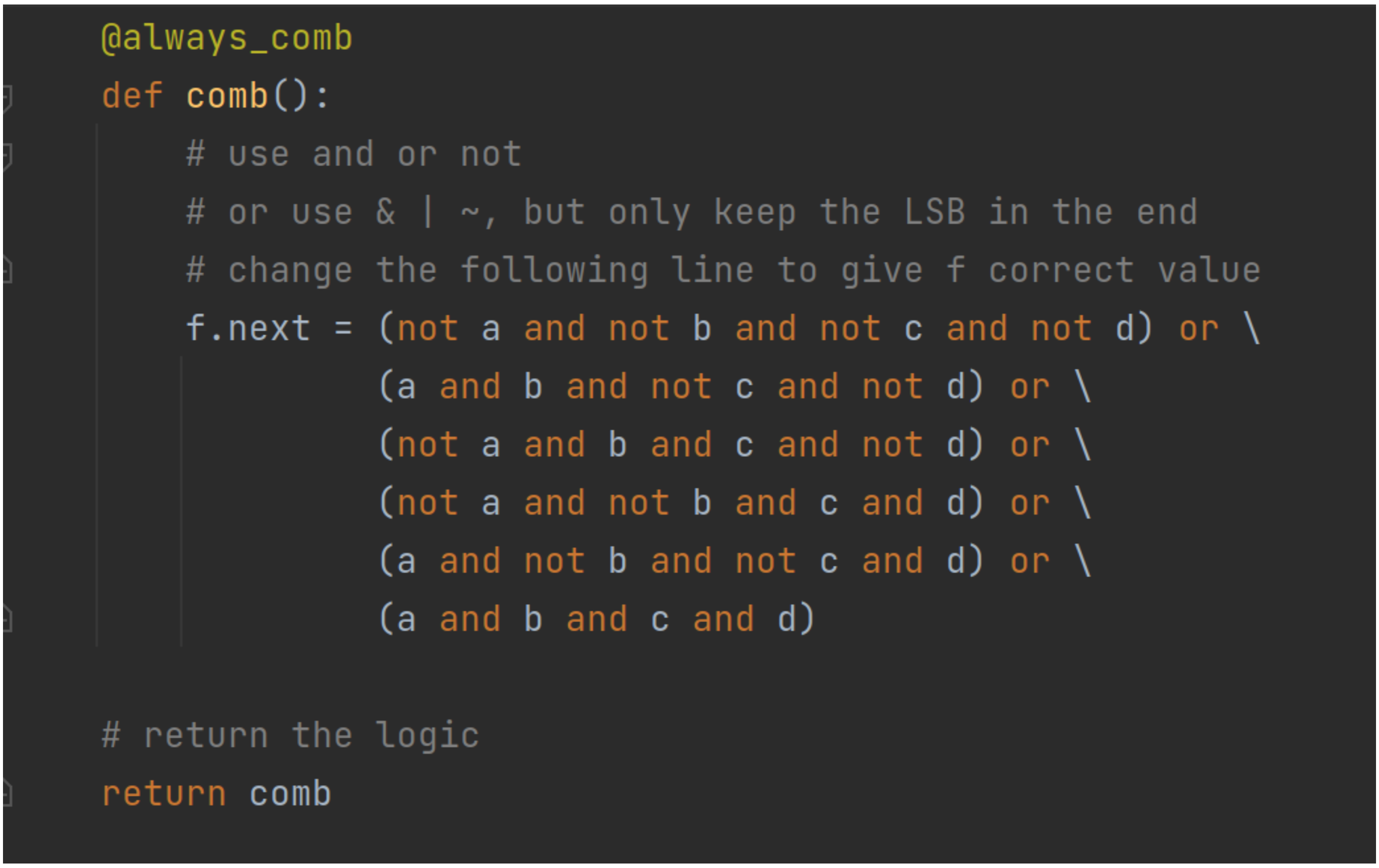
HW4

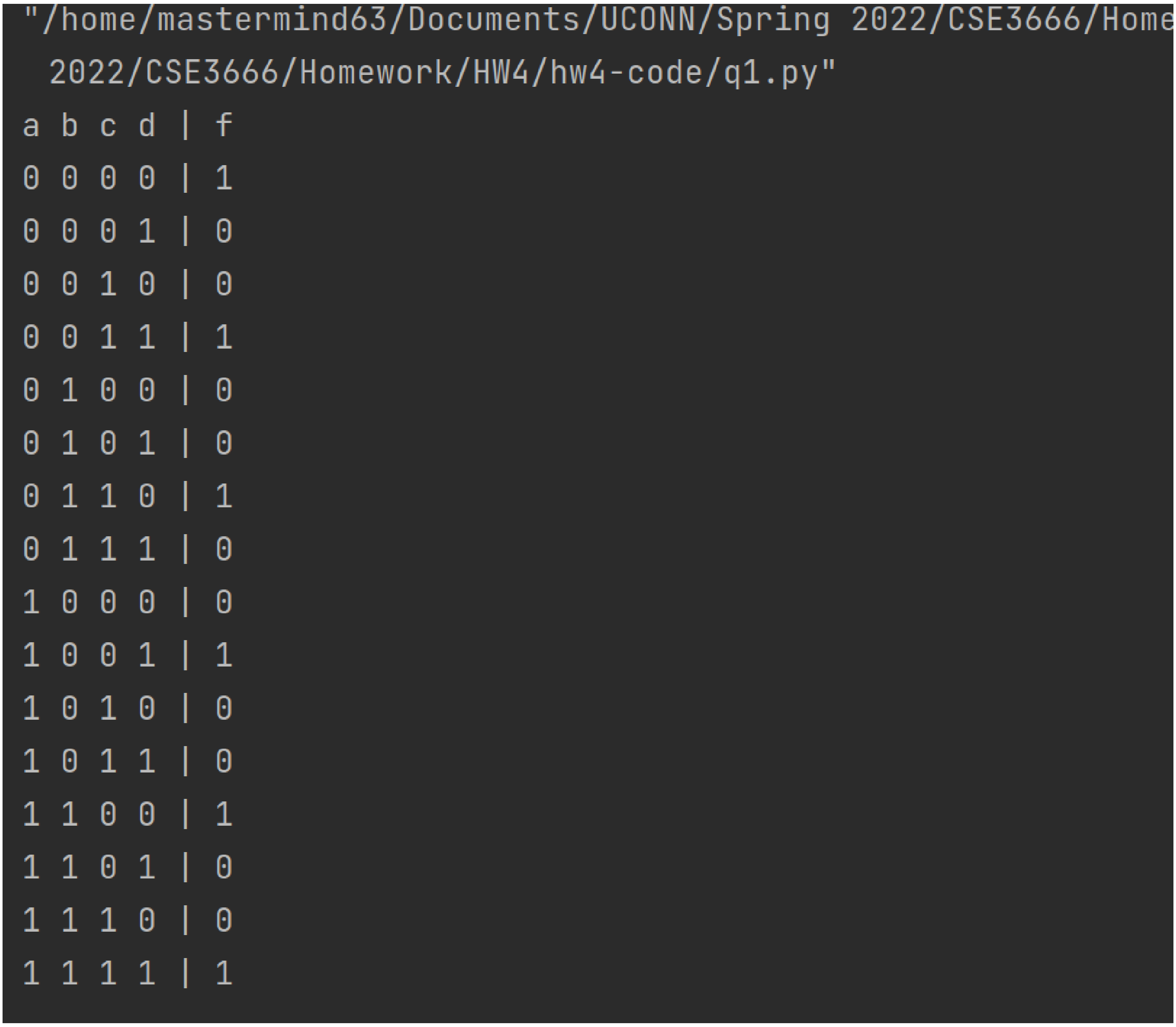
Question 1

Combinational logic



```
@always_comb
def comb():
    # use and or not
    # or use & | ~, but only keep the LSB in the end
    # change the following line to give f correct value
    f.next = (not a and not b and not c and not d) or \
             (a and b and not c and not d) or \
             (not a and b and c and not d) or \
             (not a and not b and c and d) or \
             (a and not b and not c and d) or \
             (a and b and c and d)

# return the logic
return comb
```

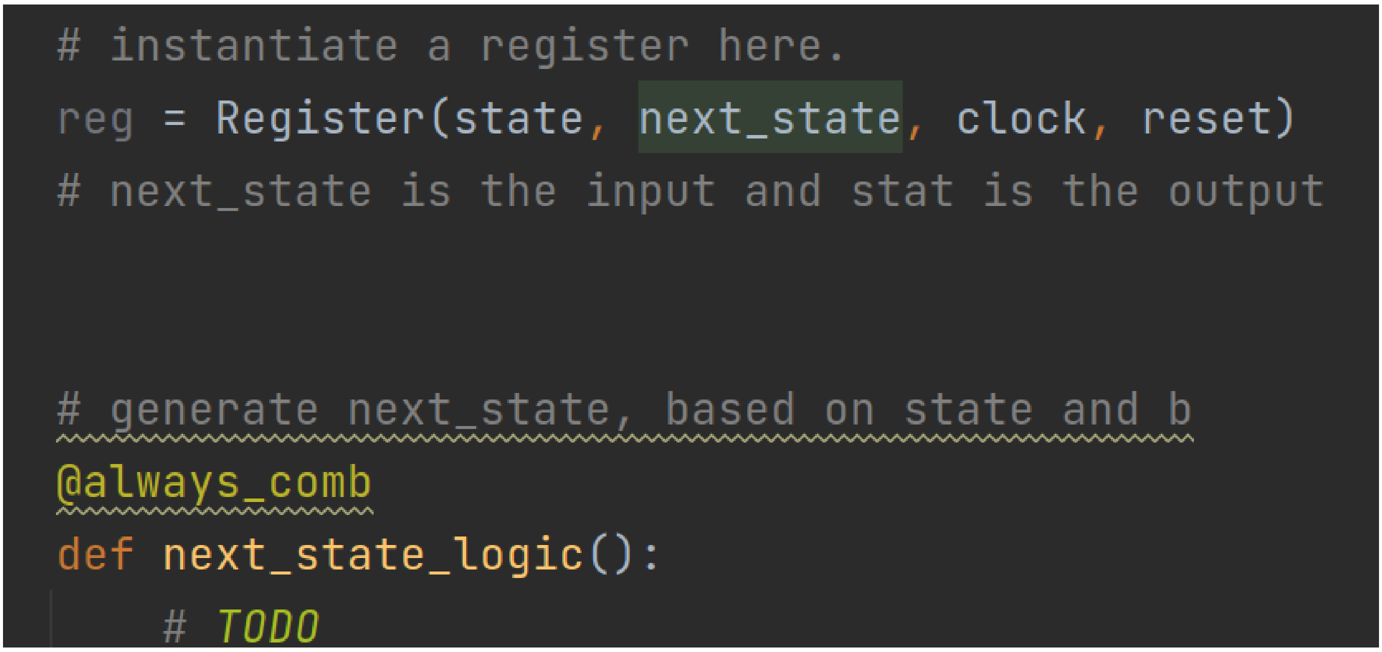
Test Cases



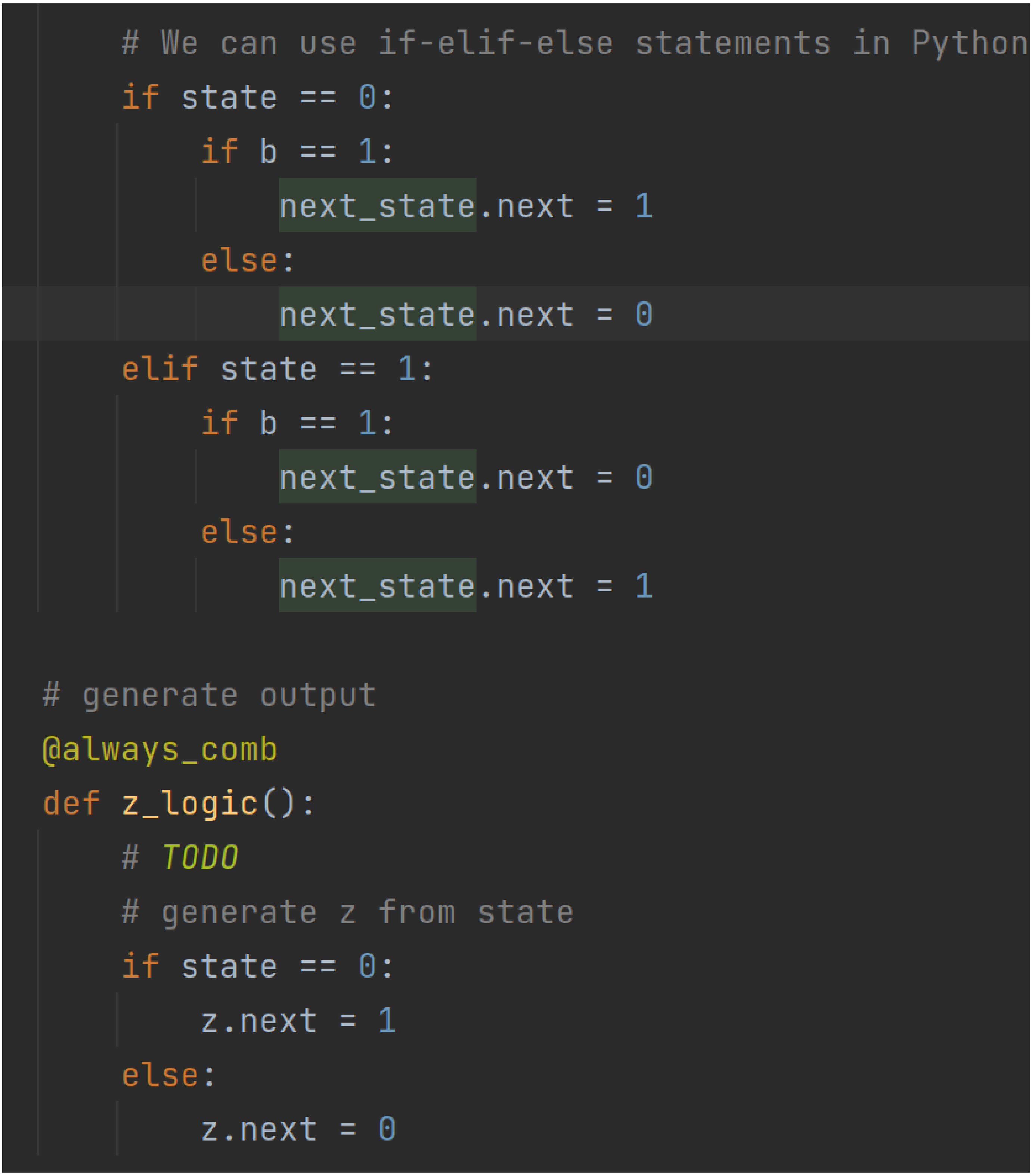
```
"/home/mastermind63/Documents/UCONN/Spring 2022/CSE3666/Home
 2022/CSE3666/Homework/HW4/hw4-code/q1.py"
a b c d | f
0 0 0 0 | 1
0 0 0 1 | 0
0 0 1 0 | 0
0 0 1 1 | 1
0 1 0 0 | 0
0 1 0 1 | 0
0 1 1 0 | 1
0 1 1 1 | 0
1 0 0 0 | 0
1 0 0 1 | 1
1 0 1 0 | 0
1 0 1 1 | 0
1 1 0 0 | 1
1 1 0 1 | 0
1 1 1 0 | 0
1 1 1 1 | 1
```

Question 2

State Machine

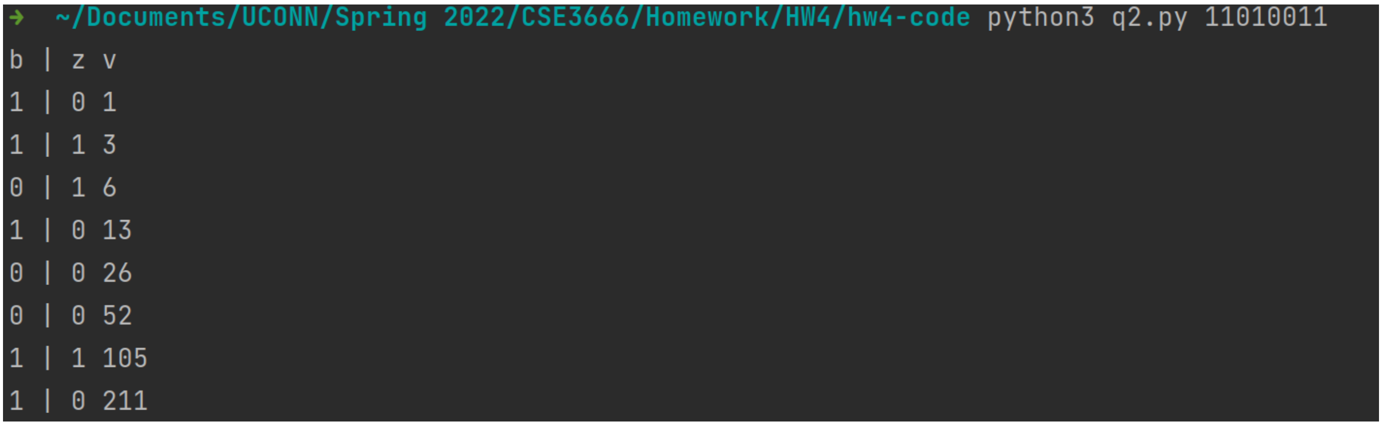


```
# instantiate a register here.
reg = Register(state, next_state, clock, reset)
# next_state is the input and stat is the output



# generate next_state, based on state and b
@always_comb
def next_state_logic():
    # TODO
```



```
    # We can use if-elif-else statements in Python
    if state == 0:
        if b == 1:
            next_state.next = 1
        else:
            next_state.next = 0
    elif state == 1:
        if b == 1:
            next_state.next = 0
        else:
            next_state.next = 1

# generate output
@always_comb
def z_logic():
    # TODO
    # generate z from state
    if state == 0:
        z.next = 1
    else:
        z.next = 0
```

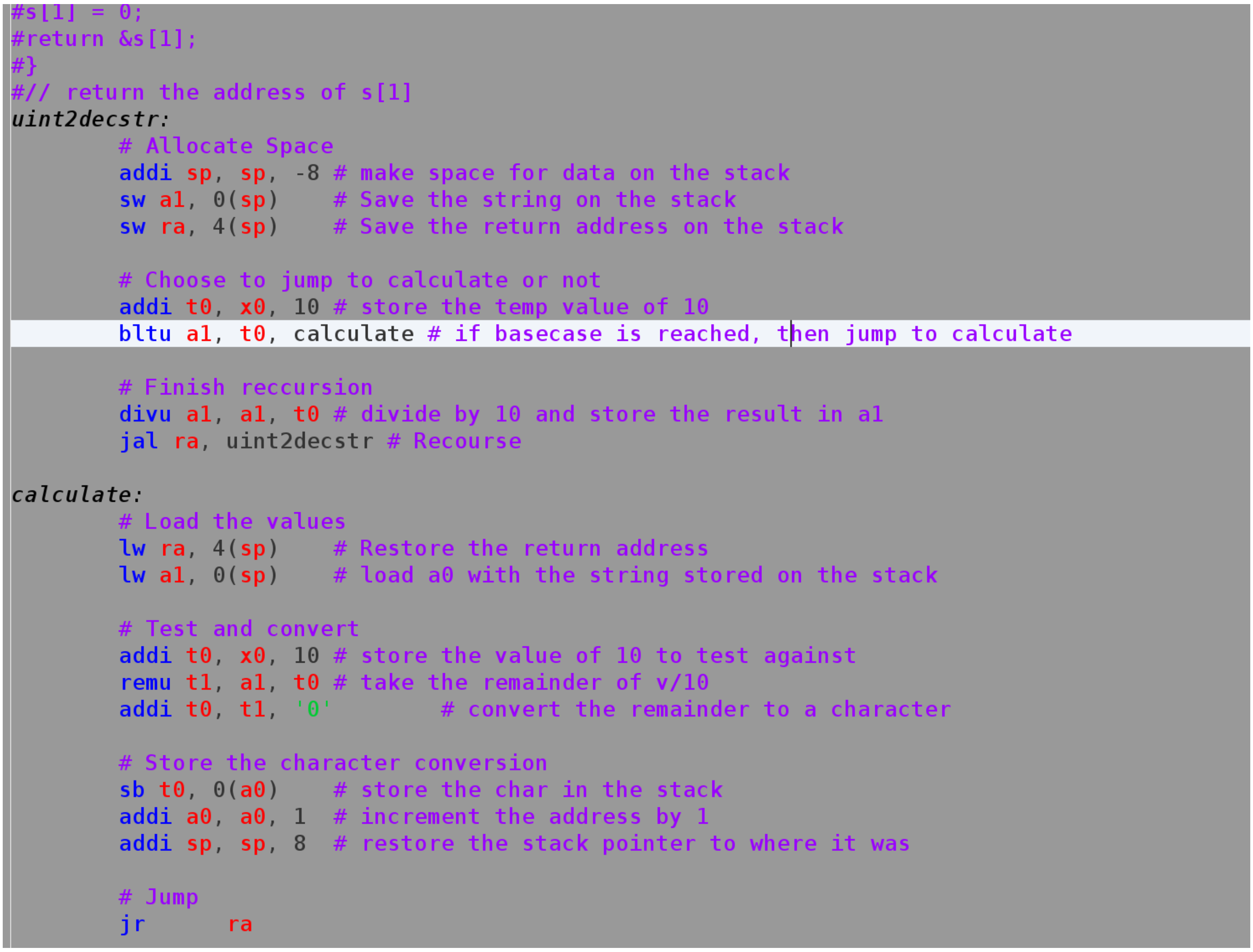
Test Cases



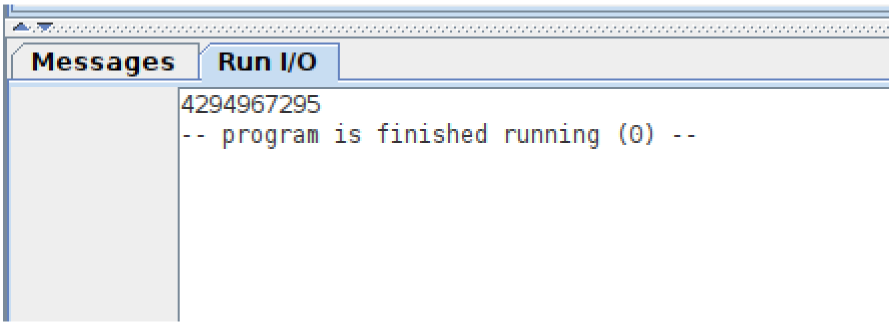
```
➜  ~/Documents/UCONN/Spring 2022/CSE3666/Homework/HW4/hw4-code python3 q2.py 11010011
b | z v
1 | 0 1
1 | 1 3
0 | 1 6
1 | 0 13
0 | 0 26
0 | 0 52
1 | 1 105
1 | 0 211
```

Question 5

Code



```
#s[1] = 0;
#return &s[1];
#}
#// return the address of s[1]
uint2decstr:
        # Allocate Space
        addi sp, sp, -8 # make space for data on the stack
        sw a1, 0(sp)    # Save the string on the stack
        sw ra, 4(sp)    # Save the return address on the stack

        # Choose to jump to calculate or not
        addi t0, x0, 10 # store the temp value of 10
        bltu a1, t0, calculate # if basecase is reached, then jump to calculate

        # Finish reccursion
        divu a1, a1, t0 # divide by 10 and store the result in a1
        jal ra, uint2decstr # Recourse

calculate:
        # Load the values
        lw ra, 4(sp)    # Restore the return address
        lw a1, 0(sp)    # load a0 with the string stored on the stack

        # Test and convert
        addi t0, x0, 10 # store the value of 10 to test against
        remu t1, a1, t0 # take the remainder of v/10
        addi t0, t1, '0'        # convert the remainder to a character

        # Store the character conversion
        sb t0, 0(a0)    # store the char in the stack
        addi a0, a0, 1  # increment the address by 1
        addi sp, sp, 8  # restore the stack pointer to where it was

        # Jump
        jr      ra
```



```
4294967295
-- program is finished running (0) --
```