

Tutorial 6
CSE 112 Computer Organization

Q1.

High-level code

```
m=input()
```

```
for(int i=1;i!=(m+1);i++){
    for(int j=0;j!=i;j++)"{
        cout<<" # "<<" " <<;
    }
}
cout<<endl;
-----
in x1          # Read input m into x1
addi x2, x0, 1  # i = 1

outer_loop:
bgt x2, x1, end  # if i > m, exit
addi x3, x0, 0    # j = 0

inner_loop:
bge x3, x2, next_line # if j >= i, go to next line
out "# "          # Print "# "
addi x3, x3, 1      # j++
beq x0, x0, inner_loop # Repeat inner loop

next_line:
out "\n"          # Print new line
addi x2, x2, 1      # i++
beq x0, x0, outer_loop # Repeat outer loop

end:
```

```
    hlt      # Halt execution
```

Q2:

```
input n
switch(n):
    case 5: print 5;
    case 4: print 4;
    case 3: print 3;
    case 2: print 2;
    case 1: print 1;
    case 0: print 0;
```

```
in x1      # Read input n into x1
```

```
beq x1, x0, case_0  # if n == 0, go to case_0
beq x1, x2, case_1  # if n == 1, go to case_1
beq x1, x3, case_2  # if n == 2, go to case_2
beq x1, x4, case_3  # if n == 3, go to case_3
beq x1, x5, case_4  # if n == 4, go to case_4
beq x1, x6, case_5  # if n == 5, go to case_5
beq x0, x0, invalid # If n is not in range, go to invalid
```

case_5:

```
    out "5 "
```

case_4:

```
    out "4 "
```

case_3:

```
    out "3 "
```

case_2:

```
    out "2 "
```

case_1:

```
    out "1 "
```

case_0:

```
    out "0 "
```

```
    hlt      # Halt execution
```

invalid:

```
out "Invalid"      # Print invalid message
hlt                  # Halt execution
```

Q3

```
n = input()
switch(n) {
    case 0: print 0;
              break;
    case 1: print 1;
              break;
    case 2: print 2;
              break;
    case 3: print 3;
              break;
    case 4: print 4;
              break;
    case 5: print 5;
              break;
    default : print Invalid; break;
}
```

```
in x1      # Read input n into x1
```

```
beq x1, x0, case_0  # if n == 0, go to case_0
beq x1, x2, case_1  # if n == 1, go to case_1
beq x1, x3, case_2  # if n == 2, go to case_2
beq x1, x4, case_3  # if n == 3, go to case_3
beq x1, x5, case_4  # if n == 4, go to case_4
beq x1, x6, case_5  # if n == 5, go to case_5
beq x0, x0, invalid # If n is not in range, go to invalid
```

```
case_5: out "5 "
        beq x0, x0, exit
case_4: out "4 "
        beq x0, x0, exit
```

```

case_3: out "3 "
    beq x0, x0, exit
case_2: out "2 "
    beq x0, x0, exit
case_1: out "1 "
    beq x0, x0, exit
case_0: out "0 "
    beq x0, x0, exit
invalid:
    out "Invalid"      # Print invalid message
    exit: hlt           # Halt execution

```

Q4.

High level code:

```

n = input();
fact = 1;

```

```

if (n == 1) {
    print(fact);
} else {
    while (n > 1) {
        product = 0;
        count = n;
        while (count > 0) {
            product = product + fact;
            count = count - 1;
        }
        fact = product;
        n = n - 1;
    }
    print(fact);
}

```

Assembly code:

```
in x10      # Read input n into x10
    addi x11, x0, 1  # Load 1 into x11 (for comparison and result initialization)
    beq x10, x11, base_case # If n == 1, jump to base case
    add x12, x10, x0  # Copy n into x12 (counter)
    add x13, x11, x0  # Initialize factorial result as 1

loop:
    beq x12, x11, end  # If counter == 1, end loop
    add x14, x0, x0  # Initialize product accumulator (x14 = 0)
    add x15, x13, x0  # Copy result into x15 (to multiply)
    add x16, x12, x0  # Copy counter into x16 (multiplier)

mul_loop:
    beq x16, x0, mul_done # If multiplier is 0, multiplication is done
    add x14, x14, x15  # Accumulate product
    addi x16, x16, -1  # Decrement multiplier
    beq x0, x0, mul_loop # Loop until multiplier is 0

mul_done:
    add x13, x14, x0  # Store multiplication result in x13
    addi x12, x12, -1  # counter--
    beq x0, x0, loop  # Unconditional jump to loop

base_case:
    add x13, x11, x0  # Set result = 1 for n = 1

end:
    out x13      # Print result
    hlt          # Halt program
```