Mips Processor

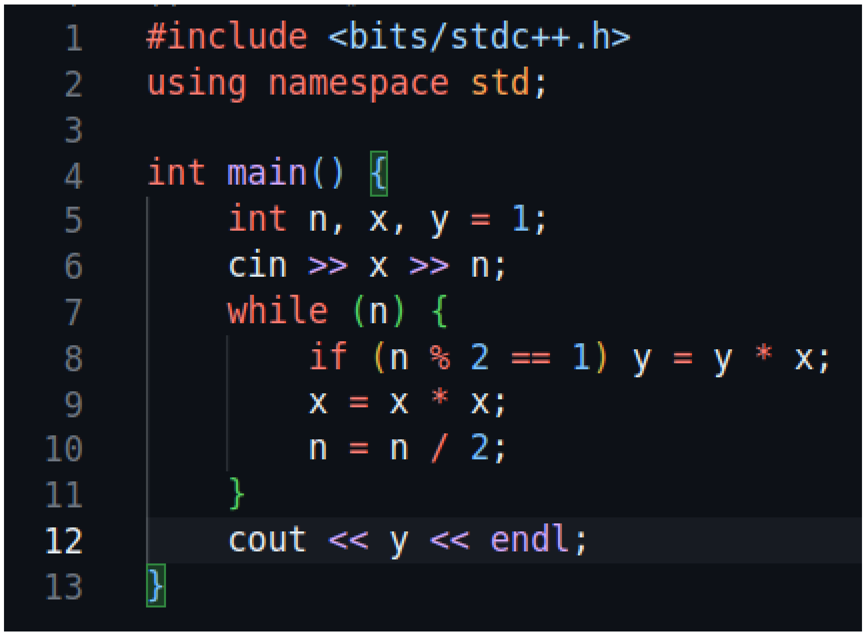
(IMT2023509) Lakshya Kapoor

(IMT2023111) Jayesh Pandit

(IMT2023056) T Shantanu

C++ Code:

Computing x^n

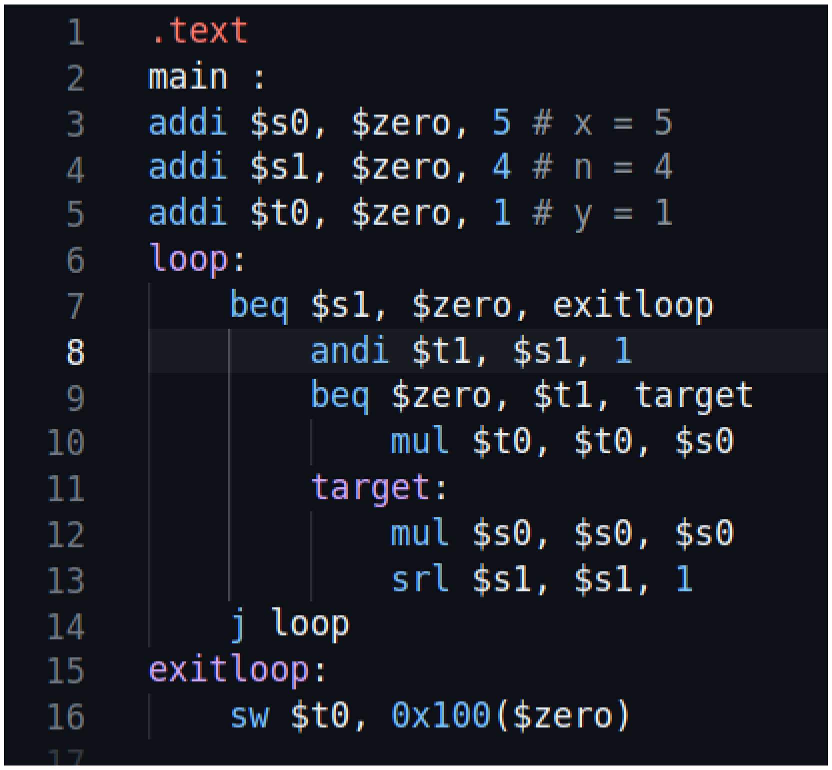


```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, x, y = 1;
    cin >> x >> n;
    while (n) {
        if (n % 2 == 1) y = y * x;
        x = x * x;
        n = n / 2;
    }
    cout << y << endl;
}
```

Calculates x^n in Θ(logn) time.

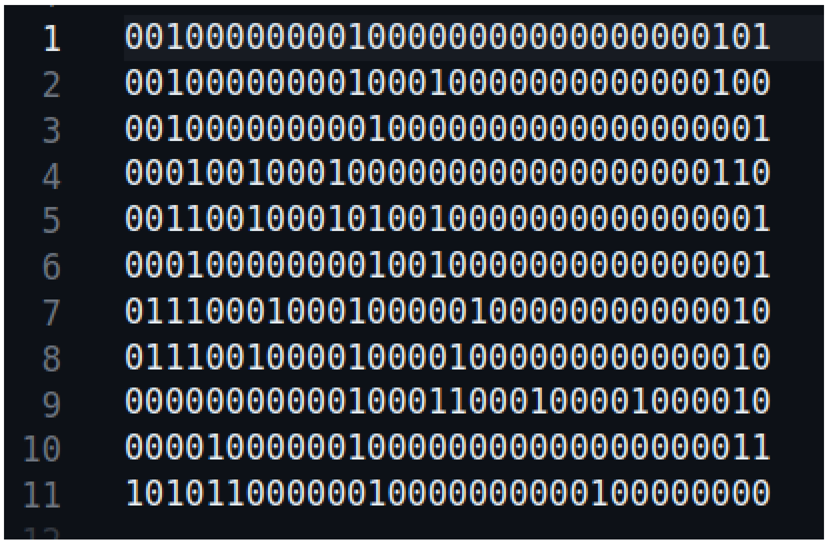
Assembly Code:



```
.text
main :
addi $s0, $zero, 5 # x = 5
addi $s1, $zero, 4 # n = 4
addi $t0, $zero, 1 # y = 1
loop:
    beq $s1, $zero, exitloop
        andi $t1, $s1, 1
        beq $zero, $t1, target
            mul $t0, $t0, $s0
        target:
            mul $s0, $s0, $s0
            srl $s1, $s1, 1
    j loop
exitloop:
    sw $t0, 0x100($zero)
```

- Store the value of x in a saved register. For e.g. - $s0
- Similarly store n in $s1
- Now save y in a temporary register as it is only going to be used for calculations.
- Now define "loop”, in which the first “beq” compares if n is equal to zero or not. If it is then it exits the loop to the “exitloop” target.
- Else it continues to next command and checks if s1 is odd or not using “andi” command.
- Then the next “beq” checks if s1 is odd or not. If it is, then it gets multiplied by y. Thus, doing x*y.
- Whether it is odd or not, the next instruction is executed anyway. x is squared using “mul” and n is halved using “srl”.
- Now to jump back to loop to check if n is equal to 0 or not and the cycle continues till it happens.
- Finally, the result is stored in the memory.

Machine code:



```
00100000000100000000000000000101
00100000000100010000000000000100
00100000000010000000000000000001
00010010001000000000000000000110
00110010001010010000000000000001
00010000000010010000000000000001
01110001000100000100000000000010
01110010000100001000000000000010
00000000000100011000100001000010
00001000000100000000000000000011
10101100000010000000000100000000
```

Mars assembler converts the mips assembly code to the above machine code. We then feed this into the instruction memory.

Processor:

We used OOPS to implement the processor. It has four child classes for the Instruction memory, Data memory, Register file and ALU. The ‘run’ method in the processor class simulates the five stages of a non-pipelined mips processor i.e., instruction fetch, instruction decode, execute, memory access and register writeback. To generate the control signals of the processor control unit and the alu control unit we used K-maps to simplify the process.