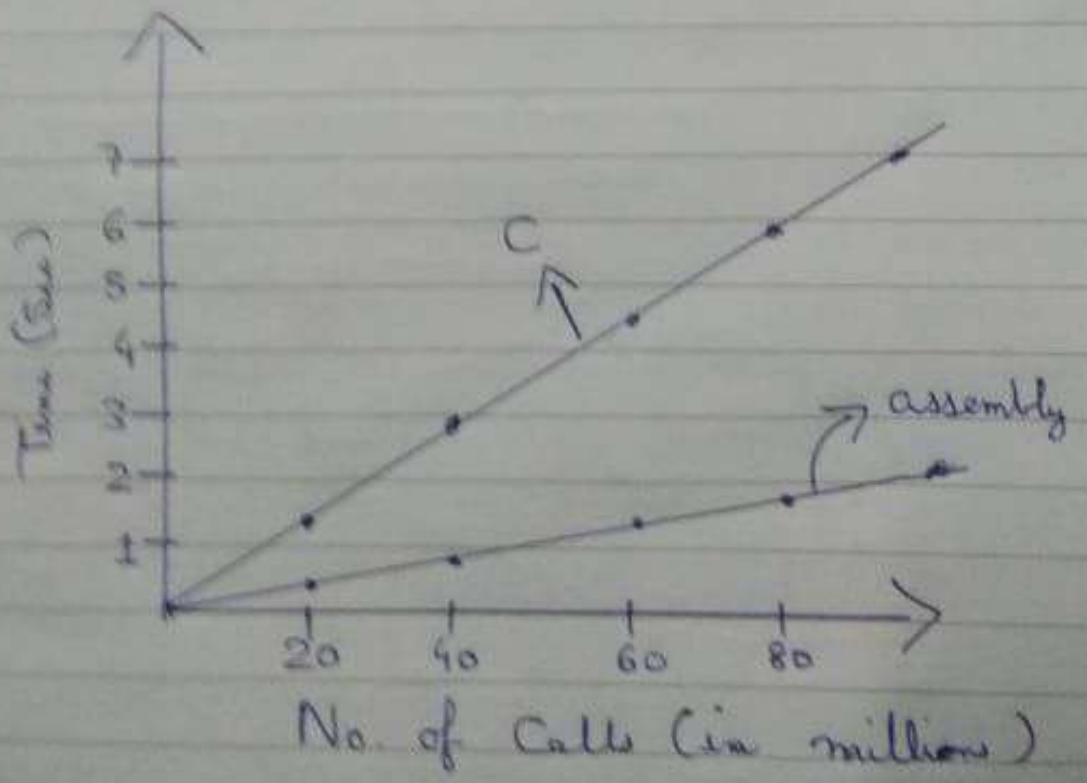
Name Ammaar Ahmad Roll No - 18010508

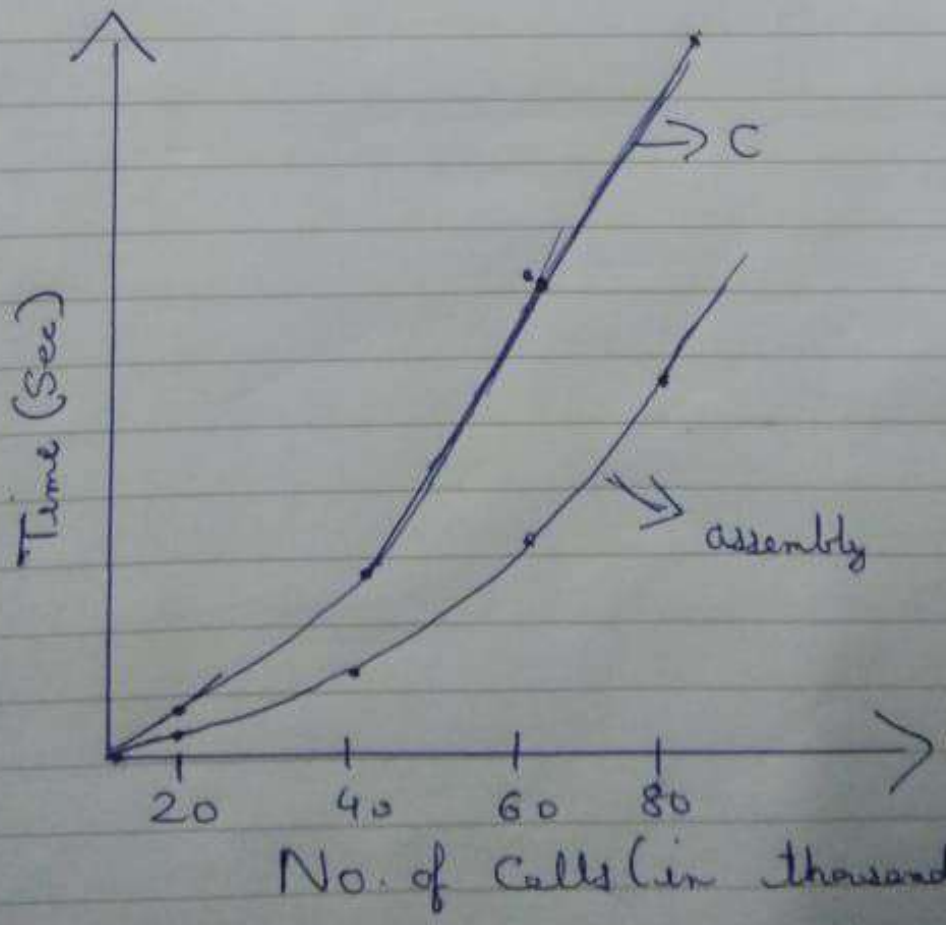
1 of Multiplication

| No of Calls (in million) | C/Mlt | asm/Mlt |
|---|---|---|
| 20 | 1.40 | 0.45 |
| 40 | 2.79 | 0.89 |
| 60 | 4.21 | 1.34 |
| 80 | 5.60 | 1.78 |
| 100 | 7.01 | 2.24 |



b) Factorial

| No. of Calls (in thousand) | C / factorial | asm / factorial |
|---|---|---|
| 20 | 1.22 | 0.71 |
| 40 | 6.08 | 2.84 |
| 60 | 20.46 | 6.62 |
| 80 | 38.46 | 14.86 |
| 100 | 65.75 | 23.72 |



# NAME: AMMAAR AHMAD

# ROLL NO: - 1801CS08

# PURPOSE OF THE LAB:

TO FAMILIARIZE WITH NASM 64 BIT ASSEMBLY LANGUAGE AND C COMMANDS TO RUN IT AND IT'S DIFFERENT INSTRUCTIONS.

# ABOUT THE GAME

IT'S A SIMPLE NUMBER GAME WHERE IN EACH TURN EITHER COMPUTER OR USER CAN WITHDRAW ANY NUMBER OF BALLS PROVIDED NUMBER = 2^X FOR SOME X BELONGS TO WHOLE NUMBER AND LOG(NUMBER)>=X.

USER WILL ENTER THE NUMBER OF BALLS TO START THE GAME

COMPUTER WILL HAVE THE FIRST TURN

# SOURCE CODE:

```
; Simple Game using Nasm

; CS/Algo used: Simple Application of Number Theory


extern printf

extern scanf

extern exit



SECTION .DATA

Start_msg     db      "Please follow the rules of the games provided below",10,0

Rule_1                db      "In each step only 2^x balls can be taken away where 0<=x<=log(No. of balls left)",10,0

Rule_2                db      "Player who cannot take the ball at any step has lost the game",10,0
```

```
Player_Won  db      "You Won",10 ,0
Computer_Won        db      "Computer Won",10 ,0
Restart_msg db      "Do you want to play again: Y for Yes, N for No: ",0
Game_msg    dd      "Enter the number of balls to start with: ",0
Playgame_msg        db      "Press any key to continue: ",0
Ingame_msg db       "No of balls left: ",0
Continue_msg        db      "Pick the balls: ",10,0
Error_msg   db      "Wrong Input, Please try Again: ",10 ,0
Character   db      "%c", 0
C           db      0
Input       db      "%d", 0
Output              db      "%d",10,0
NUM         dw      0
USER        dw      0

SECTION .CODE
GLOBAL main
main:
        ;Instructions          ;Printing Message for user
        MOV   RAX, 0
        MOV   RDI, Start_msg
        CALL  printf

        ;Printing Rules
        MOV   RAX, 0
        MOV   RDI, Rule_1
        CALL  printf
```

```
        MOV   RAX, 0
        MOV   RDI, Rule_2
        CALL  printf

        ;Main Game          ;Asking any key to start game
        MOV   RAX, 0
        MOV   RDI, Playgame_msg
        CALL  printf

        MOV RAX, 0
        MOV RDI, Character
        MOV RSI, C
        CALL scanf

        ;Start
NEW:    MOV   RAX, 0
        MOV   RDI, [Game_msg]
        CALL  printf

        MOV   RAX, 0
        MOV   RDI, [Input]
        MOV   RSI, [NUM]
        CALL  scanf

        MOV   AX, [NUM]             ;Number of balls
        MOV   BX, 03                ;BX=3

GAME:       ;Checking if remaining number of balls is 2^x
```

```
        PUSH  AX            ;Store AX value in stack
        MOV   DX, AX                ;DX=AX
        DEC   DX            ;Decrementing DX
        AND   DX, AX                ;Checking whether AX is 2^x
        JZ    LOST          ;IF ZF=1 Player Lost

        ;If not 2^x continue game
        POP   AX            ;AX = top of stack
        PUSH  AX            ;Store AX value in stack
        XOR   RDX, RDX      ;Initialing EDX=0
        AND   RAX, 0FFH     ;Initialing EAX = AX
        DIV   BX            ;Divide BX
        MOV   CX, DX                ;Copying Remainder to ECX

        ;Remaining Number of balls in AX
        POP   AX            ;AX = top of stack

        ;Computer Move
        CMP   CX, 00                ;Check if CX=0
        JNZ   NEXT          ;If ZF=1 then jump to next
        DEC   AX            ;Else decrement AX
        JMP   STEP          ;Unconditional jump to Step
NEXT:   SUB   AX, CX                ; AX = AX - CX
        JZ    LOST          ; IF AX=0 jump to LOST label

        ;Present Game Situation
STEP:   MOV [NUM], AX
```

```
MOV   RAX, 0
MOV   RDI, [Ingame_msg] ;Printing Current number of balls
CALL  printf

MOV   RAX, 0
MOV   RDI, [Output]
MOV   RSI, [NUM]
CALL  printf

MOV   RAX, 0
MOV   RDI, [Continue_msg]       ;Asking user for their turn
CALL  printf

MOV   RAX, 0
MOV   RDI, [Input]          ;User choice
MOV   RSI, [USER]
CALL  scanf

MOV   AX,[NUM]
MOV   DX,[USER]
CMP   AX,DX         ;Comparing AX with DX
JZ    WON           ;If equal then User Won
JNC   SKIP          ;If DX<AX Jump to Skip

;Printint Error Message if DX>AX
MOV   RAX, 0
MOV   RDI, [Error_msg]
CALL  printf
```

```
        JMP     STEP

        ;Checking if DX is of 2^x form
SKIP:   MOV     CX, DX              ;Copying DX to CX
        DEC     CX              ;Decrement CX
        AND     CX, DX              ;num & (num-1)
        JZ      AHEAD               ;if ZF=1 it is of 2^x form, jump to Ahead

        ;If not 2^x print error message
        MOV     RAX, 0
        MOV     RDI, Error_msg
        CALL    printf

        JMP     STEP            ;If error go back for user choice

        ;Game Update after User Choice
AHEAD:      SUB     AX, DX                  ;AX=AX-DX
        JMP     GAME            ;Game continues

        ;Player won the Game
WON:    MOV     RAX, 0
        MOV     RDI, Player_Won
        CALL    printf

        JMP     RESTART             ;Jump to restart

        ;Computer won the Game
```

```
LOST:  MOV   RAX, 0
       MOV   RDI, Computer_Won
       CALL  printf

       ;Restart Game
RESTART:
       MOV   RAX, 0
       MOV   RDI, Restart_msg
       CALL  printf

       MOV RAX, 0
       MOV RDI, Character
       MOV RSI, C
       CALL scanf

       MOV   AL, [C]
       CMP   AL, 89          ;Checking if AL ='Y'
       JE    NEW             ;If equal restart->NEW

       ;Exit Game
EXIT:  MOV   RAX, 0
       MOV   RDI, 0
       CALL  exit
```