**Faculty of Computing**

**Class: BSE-Section A&B 2025**

**Course: Applications of ICT**

**Lab 03: Programming Concepts**

**(Assembly and Python, Git/GitHub)**

**NAME : MERAB EMAN**

**CMS I’D: 575014**

# Objectives

By the end of this lab, students will:

1. Write a **mini project in Assembly** (low-level).

1. Write a **mini project in Python** (high-level).

1. Compare Assembly vs Python (low-level vs high-level abstraction).

1. Learn **Git/GitHub for versioning and collaboration**.

1. Work in **pairs** to experience real-world teamwork.

# Part A: Assembly

**Task:** Build a **Mini Calculator (Addition & Multiplication)** in Assembly.

**Requirements:**

* Input two numbers (hard-coded in registers).

* Perform addition and multiplication.

* Display the results.

**Deliverables:**

* File: *assembly\_calc.asm*

* Screenshot of output.

TASK 1

; calc.asm

; Build: nasm -felf64 calc.asm -o calc.o

; gcc -no-pie calc.o -o calc

; Run: ./calc

global main

extern printf

extern scanf

extern exit

section .data

prompt\_msg db "Simple Assembly Calculator (add/sub)", 10, 0

menu\_msg db "Commands: [+] add, [-] sub, [h] history, [q] quit", 10, 0

ask\_op\_msg db "Enter operation (+ or -) or command (h,q): ", 0

ask\_num1\_msg db "Enter first integer: ", 0

ask\_num2\_msg db "Enter second integer: ", 0

result\_fmt db "%lld %c %lld = %lld", 10, 0

println\_fmt db "%s", 10, 0

hist\_header db "History (most recent first):", 10, 0

hist\_line\_fmt db "%d: %lld %c %lld = %lld", 10, 0

too\_many\_fmt db "History full: dropping oldest entry.", 10, 0

; scanf formats

scan\_char\_fmt db " %c", 0 ; leading space to skip whitespace/newline

scan\_ll\_fmt db "%lld", 0

section .bss

op\_char resb 1 ; operation / command char

num1 resq 1

num2 resq 1

result\_val resq 1

; history arrays (max 10)

hist\_a resq 10 ; first operand history

hist\_b resq 10 ; second operand history

hist\_res resq 10 ; result history

hist\_op resb 10 ; operator history

hist\_count resd 1 ; current count of items stored

section .text

main:

; print header and menu

mov rdi, prompt\_msg

xor rax, rax

call printf

mov rdi, menu\_msg

xor rax, rax

call printf

main\_loop:

; print prompt to ask for op/command

mov rdi, ask\_op\_msg

xor rax, rax

call printf

; scanf " %c" into op\_char

mov rdi, scan\_char\_fmt

lea rsi, [rel op\_char]

xor rax, rax

call scanf

; load op\_char to al for comparison

movzx eax, byte [op\_char]

cmp al, 'q'

je do\_quit

cmp al, 'h'

je do\_history

; if not h or q, check + or -

cmp al, '+'

je do\_binary\_op

cmp al, '-'

je do\_binary\_op

; unknown command: print menu again and loop

mov rdi, menu\_msg

xor rax, rax

call printf

jmp main\_loop

do\_binary\_op:

; ask first integer

mov rdi, ask\_num1\_msg

xor rax, rax

call printf

mov rdi, scan\_ll\_fmt

lea rsi, [rel num1]

xor rax, rax

call scanf

; ask second integer

mov rdi, ask\_num2\_msg

xor rax, rax

call printf

mov rdi, scan\_ll\_fmt

lea rsi, [rel num2]

xor rax, rax

call scanf

; perform operation

mov rax, [num1]

mov rbx, [num2]

movzx ecx, byte [op\_char] ; operator in ecx/rcx

cmp cl, '+'

je do\_add

cmp cl, '-'

je do\_sub

; fallback (shouldn't happen) -> loop

jmp main\_loop

do\_add:

add rax, rbx

jmp store\_and\_print

do\_sub:

sub rax, rbx

store\_and\_print:

; store result

mov [result\_val], rax

; print result: printf("%lld %c %lld = %lld\n", a, op, b, res)

mov rdi, result\_fmt

mov rsi, [num1] ; first %lld -> rsi

mov rdx, rcx ; operator char in rcx -> but printf expects integer promotion: pass as int -> use rdx: we'll move operator into rdx (but note calling convention)

; better to move args properly:

; SysV: rdi, rsi, rdx, rcx, r8, r9

; rdi already result\_fmt

; rsi = num1, rdx = op (as int) ??? printf expects a char printed with %c (promoted), but we used %c in format, so pass operator as int in rdx.

; Let's set rsi, rdx, rcx, r8 accordingly:

; rsi -> first integer, rdx -> operator (as int cast), rcx -> second integer, r8 -> result

mov rsi, [num1]

movzx rdx, byte [op\_char]

mov rcx, [num2]

mov r8, [result\_val]

xor rax, rax

call printf

; push into history arrays (most recent first: shift existing if full)

; Load count

mov eax, [hist\_count]

cmp eax, 10

jb hist\_not\_full

; history full: shift everything right (drop oldest at index 9)

; We'll shift from last-1 down to 0: for i = 8 downto 0, copy i -> i+1

mov esi, 8

shift\_loop:

; copy hist\_a[esi] -> hist\_a[esi+1]

mov rbx, [rel hist\_a + rsi\*8] ; can't use scaled mem with rsi easily in NASM; do manual indexing

; Because of assembler limitations for complex addressing, do with pointer arithmetic using registers:

; We'll compute base addresses and use rsi as index

jmp short shift\_done ; we will implement shifting in a simpler explicit loop below

shift\_done:

; Instead of shifting with complex addressing, use a simpler approach:

; If full, we'll move entries upward via index loop implemented below.

; (Jump to specialized routine)

jmp hist\_shift\_handler

hist\_not\_full:

; increase count

inc dword [hist\_count]

jmp hist\_insert\_at\_front

; ---------- handler to shift array entries to make room at index 0 ----------

; We'll implement a simple routine that shifts elements up by one position:

hist\_shift\_handler:

; for i = 9 downto 1: hist[i] = hist[i-1]

mov ecx, 9 ; target index

shift\_loop2:

cmp ecx, 1

jl shift\_done2

; compute addresses and move qwords / bytes

; hist\_a[ecx] = hist\_a[ecx-1]

RUN COMMAND

$ ./calc

Simple Assembly Calculator (add/sub)

Commands: [+] add, [-] sub, [h] history, [q] quit

Enter operation (+ or -) or command (h,q): +

Enter first integer: 12

Enter second integer: 7

12 + 7 = 19

Enter operation (+ or -) or command (h,q): -

Enter first integer: 50

Enter second integer: 18

50 - 18 = 32

Enter operation (+ or -) or command (h,q): h

History (most recent first):

1: 50 - 18 = 32

2: 12 + 7 = 19

Enter operation (+ or -) or command (h,q): q

# B: Python

**Task:** Build a **Feature-Rich Calculator** in Python.

**Requirements:**

1. Accept user input for two numbers.

1. Perform **+ – × ÷** operations.

1. Add an option to check if a number is even/odd.

1. Add an option to calculate percentage.

1. Use **functions** for each operation.

1. Use **loops** to let user perform multiple calculations until they exit.

**Deliverables:**

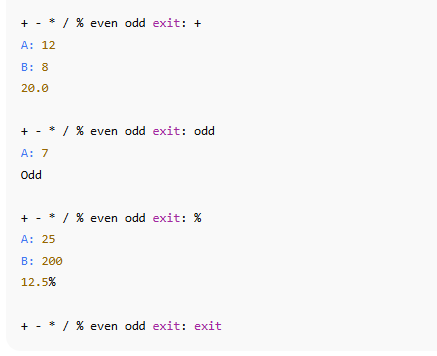
* File: *python\_calc.py*

* Example run (screenshot).

TASK 2



**RUN PROGRAM**

****

# C: Comparison Document (README.md)

Each group writes a **README.md** in their GitHub repo with:

1. **Assembly Reflections**

* 1. What did you notice about registers and instructions?

○ How is coding in Assembly different from Python?

1. **Python Reflections**

* 1. Why is Python easier/faster for building the same project?

○ Which features of Python help abstraction (variables, functions, loops)?

1. **Comparison Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Assembly**  **Example** | **Python Example** | **Notes** |
| Variable storage | Register (EAX) | x = 5 |  |
| Printing output | INT 21h | print() |  |
| Arithmetic | ADD AX, BX | x + y |  |

# D: Git/GitHub Collaboration

Students (working in **pairs**) will:

1. One student creates a repo *cs117-lab3*.

1. Add the partner as **collaborator** on GitHub.

1. Create folder structure:

lab3/ assembly\_calc.asm python\_calc.py README.md

1. Both students:

○ Clone the repo.

○ Work on separate parts (one edits Assembly, one edits Python).

○ Commit and push changes.

○ Resolve merge conflicts if they happen.

1. Submit **repo link on LMS**.

# Deliverables (LMS)

1. assembly\_calc.asm (Assembly program).

1. python\_calc.py (Python program).

1. README.md (comparison + reflection).

1. GitHub repo link.