Calculator emulator using Python + Assembly backend.

Computer Organization and Architecture

ECE2002

Name -S SREE SAI VARDHAN

Registration no- 23BCE8186

Assembly-Python Calculator Integration

Project Overview

This project implements a simple calculator that uses assembly language for its core arithmetic operations, which are then accessed through a Python interface. By combining the efficiency of low-level assembly code with the user-friendly interface of Python, this calculator demonstrates cross-language integration and provides insights into how high-level applications can leverage low-level code for performance-critical operations.

Technical Implementation

Architecture

The calculator consists of two key components:

- Assembly language implementation of basic arithmetic operations (add, subtract, multiply, divide)
- Python wrapper that provides a user interface and connects to the assembly code via ctypes

This design separates the computation layer (assembly) from the interface layer (Python), demonstrating a modular architecture approach.

Assembly Implementation

The core calculator functions are implemented in x86-64 assembly language. The assembly code:

- Takes advantage of CPU registers for efficient computation
- Follows Windows x64 calling convention (arguments in RCX, RDX registers)
- Returns results in the RAX register

- Implements four basic operations: addition, subtraction, multiplication, and division

Python Interface

The Python component:

- Uses ctypes to load and interact with the compiled assembly code
- Defines proper function signatures to ensure correct data type handling
- Provides a simple command-line interface for user interaction
- Handles input validation and error cases (such as division by zero)

Components

calculator.asm

```
section .text
    global add, sub, mul, div
add:
    mov rax, rcx        ; first argument (a)
    add rax, rdx        ; second argument (b)
    ret
sub:
    mov rax, rcx
    sub rax, rdx
    ret
mul:
    mov rax, rcx
    imul rax, rdx
    ret
```

```
div:
    mov rax, rcx
    cqo
    idiv rdx
    ret
```

The assembly file exports four functions that implement the basic arithmetic operations using x86-64 assembly. Each function follows the Windows x64 calling convention, taking parameters in RCX and RDX registers and returning results in RAX.

calculator.py

```
import ctypes


# Load your compiled shared library
calc = ctypes.CDLL('./libcalc.dll')


# Define argument and return types
calc.add.argtypes = [ctypes.c_long, ctypes.c_long]
calc.add.restype = ctypes.c_long


calc.sub.argtypes = [ctypes.c_long, ctypes.c_long]
calc.sub.restype = ctypes.c_long


calc.mul.argtypes = [ctypes.c_long, ctypes.c_long]
calc.mul.restype = ctypes.c_long
```

```python
calc.div.argtypes = [ctypes.c_long, ctypes.c_long]
calc.div.restype = ctypes.c_long

# Simple calculator
def calculator():
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    op = input("Choose operation (+ - * /): ")

    if op == '+':
        print("Result:", calc.add(a, b))
    elif op == '-':
        print("Result:", calc.sub(a, b))
    elif op == '*':
        print("Result:", calc.mul(a, b))
    elif op == '/':
        if b != 0:
            print("Result:", calc.div(a, b))
        else:
            print("Error: Division by zero!")
    else:
        print("Invalid operation!")
calculator()
```

The Python script loads the compiled assembly functions as a shared library and provides type information for the function parameters and return values. It then implements a simple command-line interface for the calculator.

Build and Execution Process

The project is built and executed using the following steps:

1. **Assemble the calculator.asm file to create an object file**
2. nasm -f win64 calculator.asm -o calculator.obj
3. **Compile the object file into a shared library (DLL)**
4. gcc -shared -o libcalc.dll calculator.obj
5. **Run the Python script**
6. python calculator.py

Example Execution

C:\Users\srees\OneDrive\Desktop\CalculatorEmulator>nasm -f win64 calculator.asm -o calculator.obj

C:\Users\srees\OneDrive\Desktop\CalculatorEmulator>gcc -shared -o libcalc.dll calculator.obj

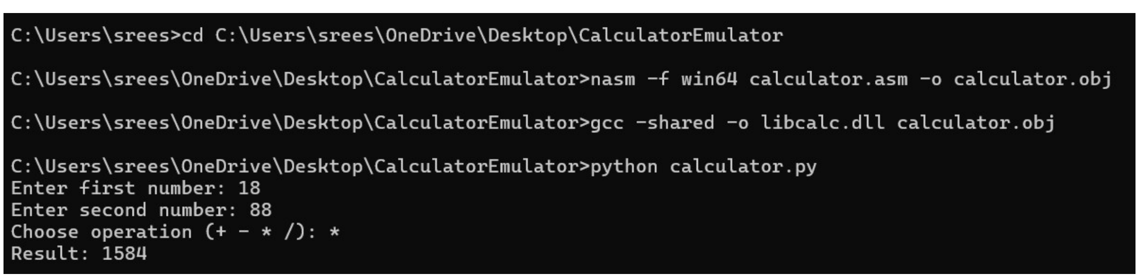
C:\Users\srees\OneDrive\Desktop\CalculatorEmulator>python calculator.py

Enter first number: 18

Enter second number: 88

Choose operation (+ - * /): *

Result: 1584



```
C:\Users\srees>cd C:\Users\srees\OneDrive\Desktop\CalculatorEmulator

C:\Users\srees\OneDrive\Desktop\CalculatorEmulator>nasm -f win64 calculator.asm -o calculator.obj

C:\Users\srees\OneDrive\Desktop\CalculatorEmulator>gcc -shared -o libcalc.dll calculator.obj

C:\Users\srees\OneDrive\Desktop\CalculatorEmulator>python calculator.py
Enter first number: 18
Enter second number: 88
Choose operation (+ - * /): *
Result: 1584
```

Technical Challenges and Solutions

1. Windows x64 Calling Convention

The project required understanding and implementing the Windows x64 calling convention for assembly language, where:

- The first argument is passed in RCX
- The second argument is passed in RDX
- The return value is expected in RAX

2. Foreign Function Interface

Using ctypes to bridge Python and assembly required careful definition of argument and return types to ensure proper data conversion between languages.

3. Division Implementation

Division in x86-64 assembly requires the use of the CQO instruction to extend the sign bit of RAX into RDX for proper handling of signed division with the IDIV instruction.

Learning Outcomes

This project demonstrates:

- Practical application of assembly language programming
- Integration of low-level code with high-level languages
- Foreign function interface usage in Python
- Cross-language development techniques
- Building and using shared libraries

Future Enhancements

Potential improvements to this project could include:

- Adding more complex mathematical operations (square root, power, etc.)
- Implementing floating-point arithmetic
- Creating a graphical user interface
- Adding memory functions similar to a standard calculator
- Implementing error handling for arithmetic exceptions

Conclusion

This Assembly-Python Calculator Integration project successfully demonstrates how to leverage the strengths of both low-level and high-level programming languages in a single application. The assembly code provides efficient computation, while Python offers a user-friendly interface, resulting in a simple yet educational example of cross-language development.