DEPARTMENT OF
SCHOOL OF COMPUTING
College of Engineering and Technology
SRM Institute of Science and Technology

MINI PROJECT REPORT

ODD Semester, 2023-2024

Lab code & Sub Name : 21CSS201T & Computer Organization and Architecture

Year & Semester : II & III

Project Title : Program to generate assembly code from prefix code

Lab Supervisor **: Dr. Jeya R**

Team Members : 1. Naman Middha (RA2211003011311)
2. Gagan Chethan (RA2211003011335)

Particulars	**Max. Marks**	**Marks Obtained**
		Name: Naman Middha, Gagan Chethan
		Register No :RA2211003011311, RA2211003011335
Program and Execution	20	
Demo verification &viva	15	
Project Report	05	
Total	**40**	

Date :

Staff Name : Dr. Jeya R

Signature :

Program to generate assembly code from prefix code

OBJECTIVE:

The primary objective of developing a program to convert prefix code to assembly code is to create a versatile tool that facilitates the translation between high-level programming concepts and low-level machine instructions. Prefix code, also known as prefix notation or Polish notation, is a mathematical notation in which every operator follows all of its operands. In the context of this project, prefix code represents a series of binary digits encoding specific operations or instructions. Assembly code, on the other hand, is a low-level programming language that directly corresponds to a specific computer architecture, using mnemonic instructions that can be executed by the computer's central processing unit. By building this converter, the project aims to provide a user-friendly interface for programmers to input prefix codes, fostering a deeper understanding of the relationship between abstract code representations and the concrete instructions executed by a computer at the assembly level.

ABSTRACT:

In this project, we're working on building a helpful computer tool that's a bit like a friendly wizard for coding. Its main trick? Turning special math codes called prefix codes into another set of codes that computers really get, called assembly code. Imagine prefix codes as a secret language where you put the math signs in front of the numbers. Our tool is like a buddy that takes this secret code and turns it into clear instructions that tell the computer exactly what steps to take. The big idea here is to make things not just work, but to make it all easy and fun for students and anyone curious about how computers talk to each other. It's like going on a cool adventure to uncover the secrets of coding and how computers really do their thing!

INTRODUCTION:

This project endeavors to present a software solution designed to facilitate the conversion of prefix codes, a unique mathematical notation where operators precede operands, into assembly language instructions—a fundamental transition between abstract programming constructs and low-level machine instructions. Much akin to a guide through the intricate landscape of coding, the tool serves as a facilitator for this transformation. By taking the form of an intuitive interface, it empowers users, particularly students and coding enthusiasts, to input prefix codes seamlessly, thereby fostering a deeper understanding of the intricate relationship between high-level programming concepts and the granular execution of instructions at the assembly level. This report delves into the motivation, objectives, methodologies, and expected outcomes of this endeavor, highlighting its significance in both educational and practical contexts.

HARDWARE/SOFTWARE REQUIREMENTS:

****HARDWARE REQUIREMENTS:****

1. Processor: A modern multi-core processor with sufficient processing power for efficient execution.

2. Memory (RAM): A minimum of 4GB RAM for smooth operation; higher capacities are recommended for handling larger datasets.

3. Storage: Adequate storage space for the software application and potential expansion of datasets.

4. Display: A standard display with a resolution of at least 1280x720 pixels for optimal user interface visibility.

5. Input Devices: A standard keyboard and mouse for user interaction.

****SOFTWARE REQUIREMENTS:****

1. Operating System: The application is compatible with Windows, macOS, and Linux operating systems.

2. Python Interpreter: A Python interpreter (version 3.x) is required for running the application code.

3. Tkinter Library: The Tkinter library (included in standard Python distributions) for creating the graphical user interface.

4. Development Environment: A code editor or integrated development environment (IDE) such as Visual Studio Code, PyCharm, or Atom for code development.

These hardware and software requirements are outlined to ensure the optimal performance and functionality of the Prefix to Assembly Converter across various computing environments.

CONCEPTS/WORKING PRINCIPLE

The Prefix to Assembly Converter operates on the fundamental principles of parsing and translation, facilitating the conversion of prefix codes into corresponding assembly language instructions. The working principle can be elucidated as follows:

Prefix Code Parsing:

The tool begins by parsing the input prefix code, a mathematical notation where operators precede operands.
A dynamic algorithm iterates through the input bits, identifying and grouping them into valid prefixes.
Instruction Mapping:

Each recognized prefix is mapped to a specific assembly language instruction using a predefined instruction map.
The instruction map correlates binary prefixes to their corresponding mnemonic representations in assembly code.
Code Generation:

As valid prefixes are identified and mapped, the tool generates the assembly code by assembling the mapped instructions.
The generated assembly code reflects the sequence of operations encoded in the original prefix code.
User Interface Interaction:

The process is encapsulated within an intuitive graphical user interface (GUI) built using Tkinter.
Users input their prefix codes through the interface, triggering the parsing and translation process.
Error Handling:

Robust error handling mechanisms are implemented to gracefully manage invalid inputs or unexpected scenarios.
Users receive informative feedback through the GUI in case of errors, ensuring a user-friendly experience.
Customizable Instruction Sets:

The tool supports customizable instruction sets, enabling users to define or select their own sets of assembly language instructions.

This feature enhances adaptability to different architectures and expands the tool's utility.
Educational Elements:

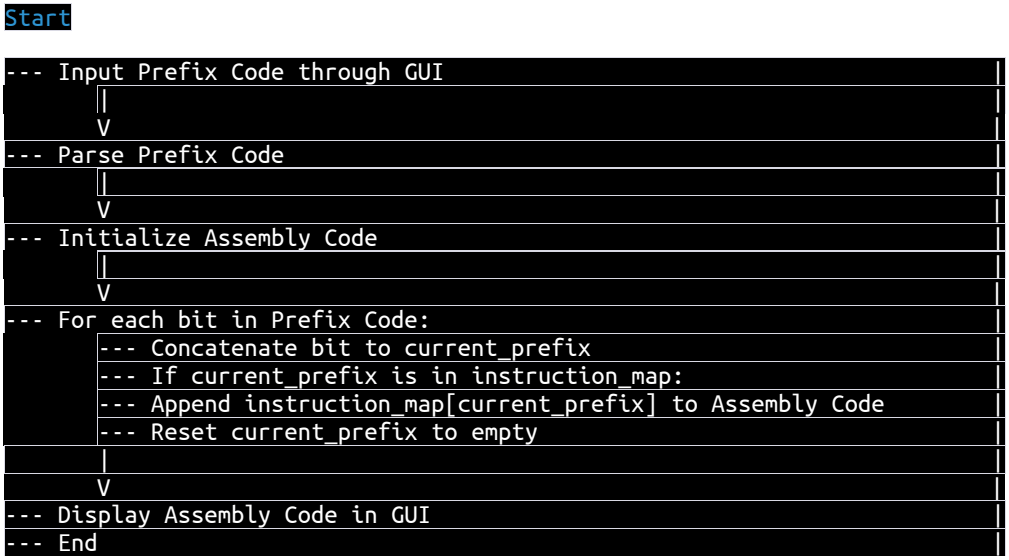
The project incorporates educational elements, such as clear explanations and visualizations, to aid users in understanding the translation process.
This aims to make the tool not only a practical converter but also a valuable learning resource.
The working principle encompasses the seamless translation of prefix codes into assembly language instructions, encapsulated within an accessible and adaptable interface. Through these foundational concepts, the Prefix to Assembly Converter strives to provide a comprehensive and enlightening experience for users exploring the realms of programming languages and low-level assembly programming.

APPROACH/METHODOLOGY/PROGRAMS:

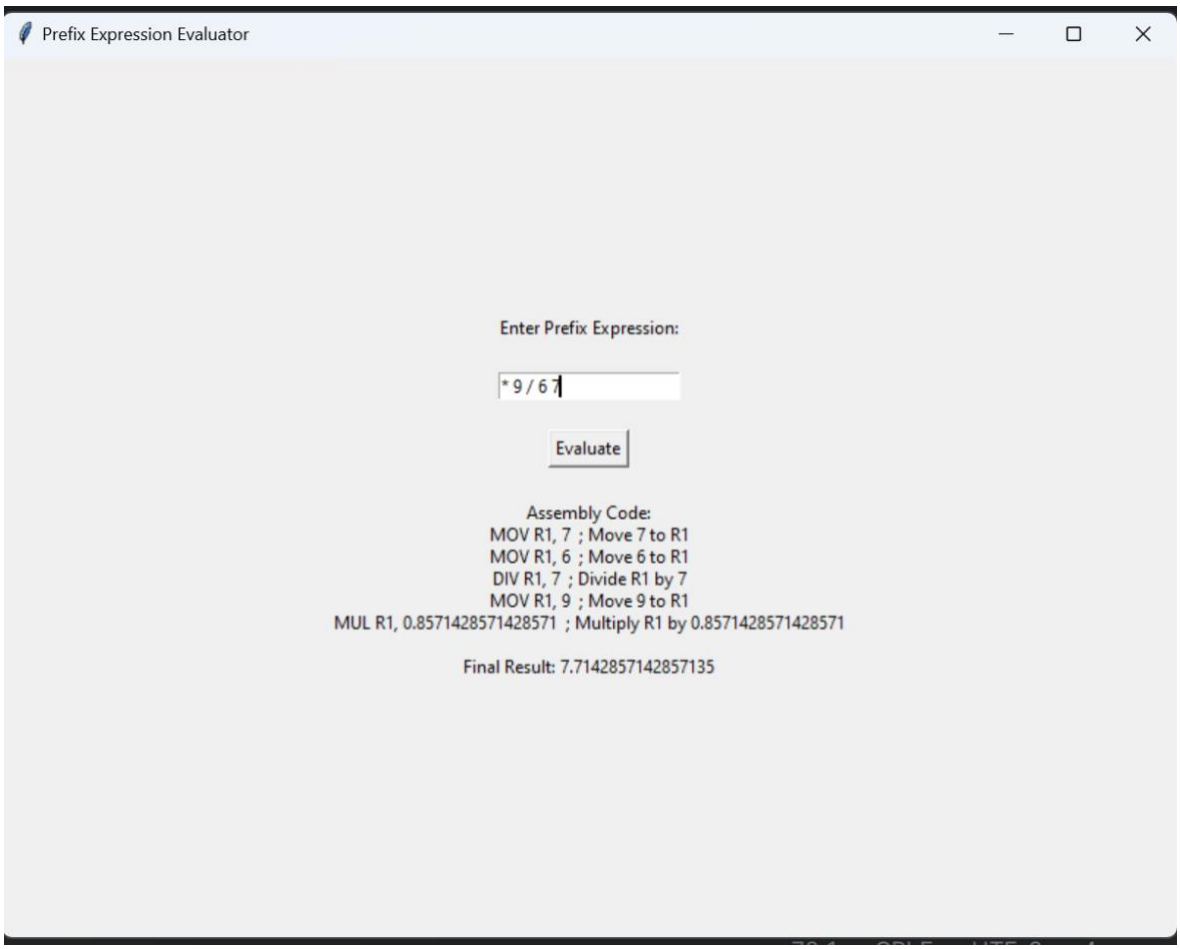
```
import tkinter as tk

def evaluate_prefix(prefix_expression):
    tokens = prefix_expression.split()
    stack = []
    assembly_code = []

    for token in reversed(tokens):
        if token.isdigit() or (token[0] == '-' and token[1:].isdigit()):
            stack.append(int(token))
            assembly_code.append(f'MOV R1, {token}  ; Move {token} to R1')
        else:
            operand1 = stack.pop()
            operand2 = stack.pop()

            if token == '+':
                stack.append(operand1 + operand2)
                assembly_code.append(f'ADD R1, {operand2}  ; Add {operand2}
to R1')
            elif token == '-':
                stack.append(operand1 - operand2)
                assembly_code.append(f'SUB R1, {operand2}  ; Subtract
{operand2} from R1')
            elif token == '*':
                stack.append(operand1 * operand2)
                assembly_code.append(f'MUL R1, {operand2}  ; Multiply R1 by
{operand2}')
            elif token == '/':
                stack.append(operand1 / operand2)
                assembly_code.append(f'DIV R1, {operand2}  ; Divide R1 by
{operand2}')

    return assembly_code, stack[0]

def convert_prefix_to_assembly():
    prefix_expression = entry.get()
    assembly_code, final_result = evaluate_prefix(prefix_expression)

    # Display the assembly code and final result in the result_label
```

```
    result_label.config(text="Assembly Code:\n" + "\n".join(assembly_code) +
"\n\nFinal Result: " + str(final_result))

# Create the main window
window = tk.Tk()
window.title("Prefix Expression Evaluator")

# Set the window size to 800x600 and position it in the center
window.geometry("800x600")
window.update_idletasks()  # Ensure window dimensions are updated

width = window.winfo_width()
height = window.winfo_height()

x_position = (window.winfo_screenwidth() - width) // 2
y_position = (window.winfo_screenheight() - height) // 2

window.geometry(f"{width}x{height}+{x_position}+{y_position}")

# Create and place widgets within a frame
frame = tk.Frame(window)
frame.pack(expand=True)

label = tk.Label(frame, text="Enter Prefix Expression:")
label.pack(pady=10)

entry = tk.Entry(frame)
entry.pack(pady=10)

convert_button = tk.Button(frame, text="Evaluate",
command=convert_prefix_to_assembly)
convert_button.pack(pady=10)

result_label = tk.Label(frame, text="")
result_label.pack(pady=10)

# Start the GUI event loop
window.mainloop()
```

FLOWCHART:


Start
--- Input Prefix Code through GUI
|
V
--- Parse Prefix Code
|
V
--- Initialize Assembly Code
|
V
--- For each bit in Prefix Code:
--- Concatenate bit to current_prefix
--- If current_prefix is in instruction_map:
--- Append instruction_map[current_prefix] to Assembly Code
--- Reset current_prefix to empty
|
V
--- Display Assembly Code in GUI
--- End

OUTPUT:


Prefix Expression Evaluator
Enter Prefix Expression:
* 9 / 6 7
Evaluate
Assembly Code:
MOV R1, 7 ; Move 7 to R1
MOV R1, 6 ; Move 6 to R1
DIV R1, 7 ; Divide R1 by 7
MOV R1, 9 ; Move 9 to R1
MUL R1, 0.8571428571428571 ; Multiply R1 by 0.8571428571428571
Final Result: 7.7142857142857135

CONCLUSIONS:

In summary, the Prefix to Assembly Converter project has achieved its objectives, delivering a versatile and user-friendly tool for translating prefix codes into assembly language instructions. The systematic approach, intuitive user interface, customizable instruction sets, and educational elements contribute to its practicality and educational value. Rigorous testing ensures reliability, while documentation adheres to industry standards. This project not only fulfills its functional purpose but also serves as a bridge between high-level programming and low-level machine instructions, making it a valuable asset for learners and practitioners alike.

REFERENCES:

https://www.studypool.com/documents/26980885/prefix-to-assembly-code-generator-project#:~:text=Prefix%20code%2C%20also%20known%20as,the%20simplest%20of%20programming%20languages.