Logo, company name

Description automatically generated



**COMSATS University Islamabad (CUI)**

**Lab terminal**

**Submitted to:**

**Submitted By: ANIS MAJID**

**Reg No: FA21-BCS-024**

**Course: COMPILER CONSTRUCTION**

**Date: 3 JANUARY 2025**

**Question No 04)**

**Input: Mini-c Source Code**

* **File Format:** The compiler takes as input a text file (typically with a .c extension) containing source code written in the simplified mini-c language.
* **Language Syntax:** The mini-c language is a subset of C, with specific restrictions and simplifications:
  + **Typelessness:** All variables are implicitly 32-bit signed integers. There are no explicit type declarations (e.g., int, float, char) except when declaring. Pointers and integers are treated interchangeably.
  + **Basic Control Flow:** Supports if, else, while, do...while, and return statements.
  + **Functions:** Allows definition and calls of functions, with support for parameters.
  + **Operators:** Supports standard arithmetic operators (+, -, \*), comparison operators (==, !=, <, >=), logical operators (||, &&), and assignment (=), also pre/post ++ and --, ! and - (unary), ? :, [] and ().
  + **Literals:** Supports integer literals, character literals (e.g., 'a'), string literals (e.g., "hello"), and boolean literals (true, false). String literals are concatenated if contiguous.
  + **Comments:** Supports single-line comments starting with # or //.
  + **No Structures or Pointers:** Structs or unions are not part of the language. Pointers are treated as integers. The only valid "pointer" operations are dereference and array indexing.
  + **Single Line Declarations:** You cannot declare multiple variables on the same line.
  + **No function pointers:** No support for function pointers.

**Output: x86 Assembly Code**

* **File Format:** The compiler generates a text file (typically with an .s extension) containing x86 assembly language code (32-bit).
* **Assembly Syntax:** The generated assembly code follows Intel syntax (as specified by .intel\_syntax noprefix).
* **Assembly Directives:**
  + .intel\_syntax noprefix: Sets the assembly syntax to Intel style.
  + .section .text: Marks the beginning of the code section.
  + .global <function\_name>: Declares a function as global, making it visible to the linker.
  + .section .data: Marks the beginning of the initialized data section.
  + .quad <value>: Declares and initializes a 8-byte (quad word) of data.
  + .section .rodata: Marks the start of read-only data section.
  + .ascii <string> : Declares a string, and does not add a null terminator.
  + .byte 0: Adds a null terminator in the string.
  + offset \_<label> : Access address of a given label.
* **Assembly Instructions:** The compiler uses a subset of x86 instructions:
  + mov: Move data between registers, memory locations, and immediates.
  + lea: Load effective address (calculates and stores a memory address into a register)
  + add, sub, imul: Arithmetic operations.
  + cmp: Compare two operands.
  + jmp: Unconditional jump.
  + je, jne, jl, jge, jnz, jz: Conditional jumps based on flags.
  + call: Call a function.
  + push, pop: Push and pop data from the stack.
  + sete, setne, setl, setge: Set a byte based on flag.
  + neg: Negate the content of a register.
  + ret return from a function.
* **Function Calling Convention (cdecl):** The generated code adheres to the cdecl calling convention:
  + Parameters are pushed onto the stack in reverse order.
  + The caller is responsible for cleaning up the stack after the call.
  + The return value is placed in the eax register.
  + The base pointer ebp and stack pointer esp must be adjusted to allocate local variables and restore the old base pointer.