CSA1455

Compiler Design For Lexical Analysis

Assignment - 1

Name: Sai Lokesh Nalabothu

Regd. No: 192365023

Branch: IInd year, CSE (Cyber security)

Date: 10 - Feburary - 2025

Serial. No: 27

CODE GENERATION FOR FUNCTION CALLS AND PARAMETER PARAMETERS:

FUNCTION CALLS AND RETURN VALUE HANDLING

When generating code for function calls, a Compiler Must handle:

1. Parameter Passing

2. Stack Management

3. Return value Handling

Parameter Passing Mechanisms

There are three Main ways to Pass Parameters:

A. Pass by value

* The actual value is Copied to the Function's stack frame.

Pass by Reference

- * A Pointer (address) to the actual data is Passed.
- * The Function Modifies the Caller's variable directly.

C. Pass by Register

- * Parameters are Passed in CPU Registers for efficiency.
- * Used in Architectures like X86-64 and Arm.

Stack frame and Memory Allocation

When a Function is Called, the system allocates a Stack frame to Store Parameters, Return addresses, local variables, and Saved Registers.

1. Structure of a Stack frame

   1. Return Address
   2. Saved Base Pointer.

3. Function Arguments

4. Local variables

5. Saved Registers

2. Stack Memory Layout

Arguments Passed to Function
Return Address
Saved Base Pointer (EBP)
Local variables
Saved Registers
Lower Memory Addresses

* Stack frame is created for each function Call and destroyed on Return.

* ESP (Stack Pointer) tracks the top of the stack.

* Stack Allocation is faster but limited in size.

* Heap Allocation is flexible.

. Explain how function calls are handled in Code Generation.

Function calls are a critical Part of Code generation in a Compiler.

1. Steps in Function Call Handling

a. Callers Responsibilitiy

1. Pass Parameters
2. Save Caller - Saved Registers
3. Push Return Address
4. Jump to Function.

Ex:

```
int add (int a, int b) {
    return a+b;
}
 int main () {
  int Result = add (10, 5);
  return Result;
 }
```

Given C Code:

```
int add (int a,int b){
    return a + b;
}
 int Main () {
   int x = 5 , y = 10;
   int result = add (x,y);
   }
```

TAC Representation:

```
L1 : PARAM a
L2 : PARAM b
L3 : t1 = a + b
L4 : Return t1
```

Function:

```
L5 : x = 5
L6 : y = 10
L7 : PARAM X
L8 : PARAMY
L9 : t2 = Call add
```

Summary:

* TAC breaks down the Function call into low- level questions.

* Temporary variables hold intermediate Results.

* Explicit PARAM and CALL statements handle Function Calls.

Discuss different Parameter Passing techniques.

Parameter Passing techniques:

1. Call by value.
2. Call by Reference
3. Call by name.
4. Call by value-Result and Call by Need

Parameter Passing	Modification Allowed?	Efficiency	Used In
Call by value	No	Slower	C, Java
Call by Reference	yes	Fast	C++
Call by Name	Yes	Can be Slow	Macros
Call by Value-Result	Yes	Moderate	Ada
Call by Need	Yes	Fast	Haskell.

* Call by value: Safe but inefficient

* Call by Reference: Fast but Risky

* Call by Name: Avoids unnesscarry

* Call by Need: optimizes Computation in Functional Programming

How is Stack Memory used for storing Function Parameters and Return addresses?

The Stack is Critical ComPonent in function Calls, Used to Store Parameters, Return addresses, local variables, and Saved Registers.

Stack Frame Layout.

Stack Address	Content
Previous Stack frame	Caller's data
Return Address	Address to Return after Function
Saved Base Pointer	old base Pointer
Function Parameters	Arguments Passed to function
Local variables	Space allocated for function
Saved Registers	Registers that Need to Preserved.

Summary:

* Function Parameters are Stored on the Stack.

* The Return address is Pushed to the Stack before the function executes.

* The Return value is Stored in EAX OR RAX.

Generate an assembly-like target Code for Function Call Handling.

Below is an assembly-like RePresentation of how Function Calls are handled using the Stack.

- Parameters are Passed on the Stack.
- The Return value is Stored in EAX.
- The Caller cleans UP the stack after the Function Call.

A. Stack setoP for add(int a, int b)

Stack Layout After CALL Add
Return Address
Argument 1 ($x = 5$)
Argument 2 ($y = 10$)

- Stack based function Calls use Push to Pass arguments.
- Callers cleans UP the stack in cdecl, while Stdcall has Calle Clean UP.
- Returns valuy are Stored in EAX