Design of Compiler

Assembly Level Design

Sumit Agarwal 13CS10061

Department of Computer Science
Indian Insitute of Technology, Kharagpur

Assgn 6 (Compilers Lab),2015

P.S.- Read the instruction_read_first.txt file to run the codes !

Outline

1. tinyC Specs
   - What does the compiler support ?
   - What doesn't the compiler support ?
2. Target Architecture Specs
3. Implementation
   - Changes for target Code
   - Activation Record
   - Use of Library Functions
   - Generate Labels
   - Global Assembly Code
   - Target Assembly Code
4. Error
   - When can you get an error in tinyC ?

tinyC Specs

What does it support ?

- Supports only int and char data types.
- Supports only void* , int* , char * as pointer data types
- Supports only one dimensional arrays
- Supports function calls involving parameters that are specified
- Supports recursive function calls as well

tinyC Specs

What doesn't it support ?

- No scope for double pointers and multi dimensional arrays.
- No scope for double or long int or unsigned int data type
- No scope for structs, classes and user defined operators
- No scope for type conversion as well

Target Architecture Specs

- Target Architecture is x86-64
- Memory address is 8 bytes long
- Has 8 bytes long registers namely rax, rdx, rdi, etc. whose last 4 bytes are namely eax, edx, edi (x86-32 counterparts)
- All assembly commands have two options l and q denoting 32 bit and 64 bit operations respectively.

Example

```
movl -20(%rbp),%eax
movq -20(%rbp),%rax
```

- Only pushq has no pushl counterpart.

Implementation

Changes for target Code

- All pointers are considered 8 bytes long.
- All references to one dimensional arrays are considered 8 bytes long.
- Integers are 4 bytes long.
- All variables be it local or parameters are stored onto stack and all function parameters are passed through stack.

Implementation

Activation Record

- We create a memory binding for all the local as well parameter variables.
- For a given function we first calculate the offset of all the variables (the parameters of it) and the variables locally defined with respect to the function's base pointer.
- For function calls we store the parameters onto the stack and pass the arguments through thr stack always !
- All variables be it local or parameters are stored onto stack and all function parameters are passed through stack so that the callee can access them through the stack.

Implementation

Global Assembly Code

- We have already made library functions for printi, readi and prints in assignment 2.
- We now use those which uses syscall for system in and system out (reading and printing parameters and strings).
- Introduced two new functions readc and printc for reading and printing a character respectively to support the input and output and therefore operations on strings as well

Implementation

Global Assembly Code

- As we do not have quad numbers like we had in the quad Array we have labels to take care of the jump statements.
- As we counter any such jump operator we create a simultaneous label for that and store in an array.

Implementation

Global Assembly Code

- As we do not have scope for string data type. As and when we enconter a string through the lex detection we store it in an array.
- Now using those strings are printed globally to be used by 'prints'.
- All global variables are handled by their name and they are not present onto the stack but are accessed through rip (respect to instruction pointer).

Implementation

Target Assembly Code

- Depending on the operator of the quad Array we create a map regarding the assembly instructions that should be generated with respect to a particular quad.
- Note all pointer and array parameter operations are 'q' (movq/leaq) and rest are 'l'.
- The function prologue is printed on encountering a function's quad array and its epilogue during returning.
- All variables (except global ones) are accessed through stack. Only global variabes are accessed through their names.

Error

When can you get an error in tinyC ?

- When the written code is not in tinyC format.
- To avoid errors you have to declare every function that you use with their parameters as well.
- To avoid errors never do an assignment while declaration
- If any of the following error persists it will generate Segmentation Fault (core dumped)

For Further Reading I


Prof. Partha Pratim Das
Run Time Environment.


Prof. Partha Pratim Das
Target Code Generation.