COMP520 GoLite Compiler AMD64 Implementation (Intel Syntax)

CHEN Hongji 260610116

Background

AMD64 Registers

@http://www.x86-64.org/documentation/abi.pdf

Register name	usage in GoLite Implementation
rax	temporary register; return register; store the number of vector registers used across function calls (float number)
rbx	base registers, used in calculate the address of the field of a struct.
rcx	temporary register; 4th argument passing register
rdx	temporary register; 3rd argument passing register
rsp	stack pointer
rsi	temporary register; 2nd argument passing register
rdi	temporary register; 1st argument passing register
r8	5th argument passing register
r9	6th argument passing register
xmm0	temporary register; return register
xmm1-7	temporary register

During the expression evaluation, rax will always store the evaluation result for the int, rune, bool and reference type, and xmm0 will always store the evaluation result for float64 type.

Calling Convention

In order to make the code generation easier, and allow the goLite program to interact with static C library, it will follow the standard calling convention.

Memory Frame

Address	Usage
rbp + 8	return address
rbp	previous rbp address
rbp - 8*number of vector parameter	shadow memory for the lower 8 byte of vector parameter (float64)
rbp - 8*number of regular parameter	shadow memory for the regular parameter
rbp - 16*k	used for stack alignment
	local variable area
	evaluation area

The size of the local variable area can be determined by,

$$\text{local size} = \sum \text{size of variable in current scope} + \sup\{\text{local size in next layer of scope}\}$$

When

Primitive Data Types Memory Implementation

GoLite Primitive Type	Memory Size (byte)	Equivalence in C	Registers for evaluated type
int	4	int64_t	rax
float64	4	double	xmm0
*rune	4	char(uint64_t)	rax
*bool	4	uint64_t	rax

* using 4 bytes for rune and bool to avoid the padding problem, and make it simpler for implementation.

Reference Data Types Memory Implementation

Unlink primitive data types, we can not have a fixed size for the reference type in GoLite, and most of them are supported by the library. This include string type, struct type, array type and slice type.

String Type

All string typed variable will be a pointer which points to a contiguously allocated memory, and terminate by '\0'. Also, the string will be **immutable** in our implementation. The concatenation of string will return a new contiguous block of memory which contains the result of concatenation.

Array Type

Array type will also be implemented using library. During the code generation, the generated code will use the beginning of the a library implemented array type as identifier. The library should provide the function of access a specific index of an array, create a slice reference to the array, and create a new array.

Slice Type

Slice type is a bit tricky. If we create a new slice type without binding it on a array. It will behave like a dynamic array (double its size every time when it reach its boundary). If we bind to a array it will become a alias to the array at first, and if we attempt to append to the slice, it will get copied, and become a dynamic array. Thus, the library should also support this feature.

The library should provide the function of access an index of a slice, create a slice reference from a slice, and create a new slice, and append a slice.(Note: the size checking will become quite difficult)

Struct Type

By our definition on memory implementation, we can treat the struct type like a array type. As every type have exactly 4 byte, we can access its entry by calculate the address. The library should provide with the function of access a field of a struct, create a new struct.