Intermediate Code of TeaPL

Basic Instructions and Components

Identifiers

There are three types of identifiers:

- global variables: @a, %b
- local variables: %a, %b, ...
- temporal variables: %r1, %r2, ..., %1, %2, ...(pure numbers should start from %0 and be continuous)

Requirements of naming:

- SSA form: each variable can be defined only once
- Continuous: pure numbers should start from %0 and be continuous

Scala Types

- void
- integer: i32, i8, i1
- pointer: i32*, i8*

Compound Types

- array: [i32 x 10]
- struct:

```
%mystruct = type { i32, i32 }
```

Global Variable Declaration and Initialization

```
@g = global i32 10
```

Local Variable Declaration

- alloca: the instruction for allocating spaces on the stack for local variables

```
%x = alloca i32
%ar = alloca [2 x i32]
%st = alloca %mystruct
```

Data Load and Store

- store: the instruction storing data to a memory address
- load: the instruction loading data from an address

```
store i32 %0, %x
%r1 = load i32, i32* %x
```

Data Load and Store for Compound Types

- getelementptr: obtain the address of a target element within an object of compound types

```
%ar = alloca [2 x i32]
%r1 = getelementptr [2 x i32], [2 x i32]* %ar, i32 0, i32 0
store i32 99, i32* %r1
%st = alloca %mystruct
%r2 = getelementptr %mystruct, %mystruct* %st, i32 0, i32 0
store i32 1, i32* %r2
```

Type Conversion

- trunc: instruction for data truncation
- zext: instruction for expanding data with zero

```
%r2 = trunc i8 %r1, i1
%r3 = zext i1 %r2, i8
```

Function Definition

- define: define a new function
- ret: return from a function call
- call: make a function call

```
define i32 @bar(i32 %r0){
    ...
    %r2 = call i32 @bar(i32 %r1)
    ret i32 %r2;
}
```

Binary Operations

- add: instruction of adding two integers
- sub: instruction of subtraction
- mil: instruction of multiplication
- div: instruction of signed division

```
%r3 = add i32 %r2, 1
%r4 = sub i32 %r3, 2
%r5 = mul i32 %r3, 3
%r6 = sdiv i32 %r4, 4
```

Comparison Operations

```
%5 = icmp sgt i32 %4, 0
%6 = icmp sge i32 %4, 0
%7 = icmp slt i32 %4, 0
%8 = icmp sle i32 %4, 0
%9 = icmp eq i32 %4, 0
%10 = icmp ne i32 %4, 0
```

Logical Operation: Not

- xor: exclusive or

```
%7 = xor i1 %6, true
```

Control Flow

- br: instruction for both direct jump and conditional jump

```
 br label %bb3
%bb1:
  %4 = icmp sgt i32 %3, 0
  br i1 %4, label %bb2, label %bb3
bb2:

bb3:
```

Data Flow
-phi: instruction of conditional value depending on its previous code block

```
bb1:

bb2:

bb3:
  %r0 = phi i32 [ %r2, %bb1 ], [ %r3, %bb2 ]
```