IR(Intermediate Representation)

List

IR(Intermediate Representation)

- IR(Intermediate Representation)은 소스 코드를 표현하기 위해 컴파일러 또는 가상 시스템에서 내부적으로 사용하는 데이터 구조 또는 코드입니다.
- IR은 최적화와 번역 등 추가 처리를 위해 도움이 되도록 설계되었습니다.
- IR중 우수한 IR은 정보의 손실 없이 소스코드를 나타낼수 있고, 특정 소스, 대상언어와 무관하게 소스 코드를 나타낼 수 있습니다.
- IR은 다음과 같은 여러 가지 형태가 있습니다.
    - 메모리 내 데이터 구조
    - 프로그램이 읽을 수 있는 특별한 튜플 또는 스택 기반 코드(중간 언어,Intermediate language)

Intermediate language

- **중간 언어는 컴퓨터 프로그램의 분석을 돕기위해 설계된 추상적 기계 언어입니다.**
- 프로그램의 소스코드가 대상 기계의 오브젝트 또는 기계 코드를 생성하기 전에 변환을 개선하는 코드에 더 적합한 형식으로 변환합니다.
- 중간 언어의 설계는 일반적으로 3가지 기본적인 방법으로 실용적인 기계어 언어와 다릅니다.
    - 각 명령은 하나의 기본 동작을 나타냅니다.
    - 제어 흐름 정보는 명령 집합에 포함되지 않을 수 있습니다.
    - 사용 가능한 프로세서 레지스터 수는 크기가 크거나 무제한 될 수 있습니다.
- 중간 언어의 일반적인 형식은 Three-address code(TAC or 3AC) 입니다.

> **추상적 기계(Abstract machine)**
>
> - 물리적인 컴퓨터 자체나 프로그램 언어 같은 기능까지 포함하며, 컴퓨터 시스템의 기능을 추상화한 모델 기계.

Three-address code

- Three-address code는 TAC 또는 3AC 으로 표시하기도 합니다.
- TAC는 향상된 코드 변환을 위해 컴파일러의 최적화하는데 사용되는 중간 코드(intermediate code) 입니다.
- 각 TAC 명령어는 최대 세 개의 피연산자를 가지며 일반적으로 대입 연산자와 이진 연산자를 조합합니다.
- TAC는 컴파일러에서 중간언어로 사용되므로, 피연산자는 구체적인 메모리 주소나 프로세서 레지스터가 아니라 레지스터 할당 중 실제 주소로 변환될 수 있는 기호적 주소를 의미합니다.

Example

- "x = (a + b * c) / 2" 연산이 다음과 같은 형태로 구현됩니다.

x = (a + b * c) / 2

```
t1 := b * c
t2 := a + t1
t3 := t2 / 2
x := t3
```

- 아래 코드가 다음과 같은 형태로 구현됩니다.

Sample code

```
for(i = 0; i < 100; i++){
        val[i] = i;
}
```

Three-address code

```
     t1 := 0
L1:  if t1 >= 100 goto L2
     t2 := t1 * 4
     t3 := val + t2
     *t3 := t1
     t1 := t1 + 1
     goto L1
L2:
```

Intermediate Representation

QEMU의 TCG(Tiny code generator)

- TCG는 QEMU에서 실행할 바이너리를 TCG를 이용해 IR(Intermediate Representation) 코드변환하는 코드 생성기 입니다.
- TCG에 의해 생성된 바이트 코드를 TCL을 이용해 실행 함으로써 거의 모든 아키텍처의 실행을 지원할 수 있습니다.
- QEME에서 지원가능한 아키텍처
    - arm, i386, ia64, ppc, ppc64, s390, sparc, x86_64

TCG Readme

- https://git.qemu.org/?p=qemu.git;a=blob_plain;f=tcg/README;hb=HEAD
- http://repo.or.cz/w/qemu/ar7.git/blob_plain/HEAD:/tcg/tci/README

Valgrind의 VEX

- VEX는 TCG와 비슷하게 거의 모든 아키텍처를 지원하며, 여러 기계 언어의 표현이 가능합니다.
- VEX IR에 중요한 클래스의 객체는 다음과 같습니다.
    - Expressions
    - Operations
    - Temporary variables.
    - Statements
    - Blocks

Example

- 다음과 같이 프로그램 분석을 쉽게하기 위해 기계코드의 표현을 추상화합니다.

The following ARM instruction

```
subs R2, R2, #8
```

Becomes this VEX IR

```
t0 = GET:I32(16)
t1 = 0x8:I32
t3 = Sub32(t0,t1)
PUT(16) = t3
PUT(68) = 0x59FC8:I32
```

- https://docs.angr.io/docs/ir.html

LLVM(Low Level Virtual Machine) IR:

- LLVM 의 IR은 가볍고 Low-level의 표현과 형태 및 확장성을 동시에 유지 합니다.
- LLVM IR은 고정된 이름의 레지스터 세트를 사용하지 한고 % 문자를 이용해 임시 테이블 세트를 표현합니다.
- LLVM 은 다음과 같은 작업을 지원합니다.
    - C 코드로 IR 생성
    - IR을 이용해 기계 코드 생성
    - 자동 최적화

Example

- 다음과 같이 IR을 생성할 수 있습니다.

Install the llvm

```
$ sudo aptitude install llvm clang
```

Sample code

```
int sample(int a,int b) {
            return a*b;
}
```

Create the llvm IR

```
lazenca0x0@ubuntu:~$ cat sample.ll
; ModuleID = 'test.c'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

; Function Attrs: norecurse nounwind optsize readnone uwtable
define i32 @sample(i32 %a, i32 %b) #0 {
  %1 = mul nsw i32 %b, %a
  ret i32 %1
}

attributes #0 = { norecurse nounwind optsize readnone uwtable "disable-tail-calls"="false" "less-precise-fpmad"
="false" "no-frame-pointer-elim"="false" "no-infs-fp-math"="false" "no-nans-fp-math"="false" "stack-protector-
buffer-size"="8" "target-cpu"="x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2" "unsafe-fp-math"="false" "use-
soft-float"="false" }

!llvm.ident = !{!0}

!0 = !{!"clang version 3.8.0-2ubuntu4 (tags/RELEASE_380/final)"}
lazenca0x0@ubuntu:~$
```

Related info

- https://en.wikipedia.org/wiki/Intermediate_representation
- https://wiki.qemu.org/Documentation/TCG
- http://repo.or.cz/w/qemu/ar7.git/blob_plain/HEAD:/tcg/tci/README
- https://en.wikipedia.org/wiki/Three-address_code