UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Trabalho de Construção de Compiladores

Estruturas de Representação Intermediária (IR)

Aluno: Marcelo Mendonça Borges
Matrícula: 11611BCC020

2019

Mapeamento de Instruções

Foi utilizado um código em linguagem C para identificar que tipo de alteração é gerada na representação intermediária do Clang do LLVM. Esse código foi iniciado com duas bibliotecas inclusas (stdio.h e stdlib.h), e a função main vazia. Partindo disso foi incrementado as seguintes estruturas básicas da linguagem:

1. Retorno da função main;
2. Declaração de variáveis;
3. Atribuição de Valores;
4. Função printf();
5. Função scanf();
6. Estrutura if-else;
7. Estrutura while;
8. Estrutura do-while;
9. Estrutura for;

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    /*Comentario*/
    int num, leitura;

    num = 100;

    printf("num: %d\n", num);

    printf("Digite um numero inteiro: ");

    scanf("%d", &leitura);

    printf("LEITURA: %d\n", leitura);

    if(leitura >= 0)
    {
        num = num/2;
        leitura = num-1;
    }
    else
    {
        num = num*2;
        leitura = num+1;
    }
```

```c
    while(leitura > 0)
    {
        leitura--;
    }

    do
    {
        leitura++;
    }
    while(leitura < 100);

    for(num = 0; num < 100; num++)
    {
        printf("%d\n",num);
    }

    return 0;
}
```

A seguir será indicado cada alteração feita no código fonte e sua alteração correspondente na representação intermediária:

0) Situação Inicial (Com função main destacada)

```
; ModuleID = 'programa.c'
source_filename = "programa.c"
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @main() #0 {
  ret i32 0
}

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-divide-sqrt-fp-math

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 9.0.0-svn371490-1~exp1~20190910083050.54 (branches/release_90)"}

```

1) Retorno da main

```
%1 = alloca i32, align 4
store i32 0, i32* %1, align 4
ret i32 0
```

2) Declaração de Variável

```
%2 = alloca i32, align 4
```

3) Atribuição de Valor

```
store i32 100, i32* %2, align 4
```

4) Função printf()

No caso da função printf(), antes da estrutura da main é adicionado o valor de @.str, além da alteração dentro da main, com a chamada da função em %4.

```
@.str = private unnamed_addr constant [9 x i8] c"num: %d\0A\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 100, i32* %2, align 4
  %3 = load i32, i32* %2, align 4
  %4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([9 x i8], [9 x i8]* @.str, i64 0, i64 0), i32 %3)
  ret i32 0
}

declare dso_local i32 @printf(i8*, ...) #1

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-c
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise-fpmad"
```

5) Função scanf()

No caso da função scanf(), a mesma estrutura de printf() é adicionada assim como a chamada de função na main.

```
@.str.1 = private unnamed_addr constant [27 x i8] c"Digite um numero inteiro: \00", align 1
@.str.2 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
```

```
%7 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str.2, i64 0, i64 0), i32* %3)
```

6) Estrutura if-else (E operações aritméticas)

No caso dessa estrutura tem-se a parte que verifica a condição, e logo depois os dois trechos de códigos representando o código dentro de if e else, respectivamente. Nota-se também que dentro de cada um desses blocos tem as operações de divisão, subtração, multiplicação e soma, junto com atribuições.

As operações de divisão e subtração são indicadas por sdiv (%12) e sub (%10), dentro do bloco de if (10). As operações de multiplicação e soma são indicadas por mul (%17) e add (%19), dentro do bloco de else (15).

```
  %8 = load i32, i32* %3, align 4
  %9 = icmp sge i32 %8, 0
  br i1 %9, label %10, label %15

10:                                               ; preds = %0
  %11 = load i32, i32* %2, align 4
  %12 = sdiv i32 %11, 2
  store i32 %12, i32* %2, align 4
  %13 = load i32, i32* %2, align 4
  %14 = sub nsw i32 %13, 1
  store i32 %14, i32* %3, align 4
  br label %20

15:                                               ; preds = %0
  %16 = load i32, i32* %2, align 4
  %17 = mul nsw i32 %16, 2
  store i32 %17, i32* %2, align 4
  %18 = load i32, i32* %2, align 4
  %19 = add nsw i32 %18, 1
  store i32 %19, i32* %3, align 4
  br label %20
```

7) Estrutura while (Com decremento)

Nessa estrutura tem-se também o bloco de verificação da condição (23) e logo depois o bloco executável do while (26). Percebe-se também o decremento (%28).

```
23:                                               ; preds = %26, %22
  %24 = load i32, i32* %3, align 4
  %25 = icmp sgt i32 %24, 0
  br i1 %25, label %26, label %29

26:                                               ; preds = %23
  %27 = load i32, i32* %3, align 4
  %28 = add nsw i32 %27, -1
  store i32 %28, i32* %3, align 4
  br label %23
```

8) Estrutura do-while (Com incremento)

Nessa estrutura também tem o bloco de verificação da condição e o bloco executável, entretanto o executável aparece

antes da verificação, como já é esperado. Temos então o executável (30) e a verificação (33). Percebe-se também o incremento (%32)

```
30:                                               ; preds = %33, %29
  %31 = load i32, i32* %3, align 4
  %32 = add nsw i32 %31, 1
  store i32 %32, i32* %3, align 4
  br label %33

33:                                               ; preds = %30
  %34 = load i32, i32* %3, align 4
  %35 = icmp slt i32 %34, 100
  br i1 %35, label %30, label %36
```

9) Estrutura for (Com printf())

Nessa estrutura tem a verificação da condição no início (36 e 37), o bloco de execução (40) em que pode-se perceber a função printf() (%42), e por fim temos o incremento do for (43).

```
36:                                               ; preds = %33
  store i32 0, i32* %2, align 4
  br label %37

37:                                               ; preds = %43, %36
  %38 = load i32, i32* %2, align 4
  %39 = icmp slt i32 %38, 100
  br i1 %39, label %40, label %46

40:                                               ; preds = %37
  %41 = load i32, i32* %2, align 4
  %42 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
  br label %43

43:                                               ; preds = %40
  %44 = load i32, i32* %2, align 4
  %45 = add nsw i32 %44, 1
  store i32 %45, i32* %2, align 4
  br label %37
```