Construção de Compiladores

Pedro Henrique Faria Teixeira

1ª Etapa do Projeto

Uberlândia - MG
2019

Execução dos Scripts e respectivas saídas:

Main.c # **Programa com somente o main estruturado:**

```c
#include <stdio.h>

int main(){

}
```

Main.txt # **Saída após a execução do código com o Clang:**

```
; ModuleID = 'main.c'
source_filename = "main.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 sspstrong uwtable
define dso_local i32 @main() #0 {
 ret i32 0
}

attributes #0 = { noinline nounwind optnone sspstrong uwtable
"correctly-rounded-divide-sqrt-fp-math"="false"
"disable-tail-calls"="false" "less-precise-fpmad"="false"
"min-legal-vector-width"="0" "no-frame-pointer-elim"="true"
"no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
"no-jump-tables"="false" "no-nans-fp-math"="false"
"no-signed-zeros-fp-math"="false" "no-trapping-math"="false"
"stack-protector-buffer-size"="8" "target-cpu"="x86-64"
"target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false"
"use-soft-float"="false" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{!"clang version 8.0.1 (tags/RELEASE_801/final)"}
```

- Podemos observar que para este código somente o main e o return foram declarados. O return por padrão ja vem declarado no corpo da função main().

Int.c # **Programa que inclui uma variável do tipo int no corpo da main():**

```c
#include <stdio.h>

int main(){
   int i = 0;
}
```

Int.txt # **Saída após a execução do código com o Clang:**

```
; ModuleID = 'int.c'
source_filename = "int.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 sspstrong uwtable
define dso_local i32 @main() #0 {
 %1 = alloca i32, align 4
 store i32 0, i32* %1, align 4
 ret i32 0
}

attributes #0 = { noinline nounwind optnone sspstrong uwtable
"correctly-rounded-divide-sqrt-fp-math"="false"
"disable-tail-calls"="false" "less-precise-fpmad"="false"
"min-legal-vector-width"="0" "no-frame-pointer-elim"="true"
"no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
"no-jump-tables"="false" "no-nans-fp-math"="false"
"no-signed-zeros-fp-math"="false" "no-trapping-math"="false"
"stack-protector-buffer-size"="8" "target-cpu"="x86-64"
"target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false"
"use-soft-float"="false" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}
!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{!"clang version 8.0.1 (tags/RELEASE_801/final)"}
```

- Podemos observar que foi acrescentado 2 novas linhas na saída.
- 1ª alocando uma variável do tipo int .
- 2ª alocando o valor 0 na variável inteira.

If.c # **Programa que acrescenta um if ao corpo da main e com a várivel int declarada e faz operações de comparação e aritmética:**

```c
#include <stdio.h>

int main(){
   int i = 0;

   if (i > 0) {
       i = 1 + 2;
   }
}
```

If.txt **# Saída após a execução do código com o Clang:**

```
; ModuleID = 'if.c'
source_filename = "if.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 sspstrong 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 0, i32* %2, align 4
 %3 = load i32, i32* %2, align 4
 %4 = icmp sgt i32 %3, 0
 br i1 %4, label %5, label %6

; <label>:5:                                      ; preds = %0
 store i32 3, i32* %2, align 4
 br label %6

; <label>:6:                                      ; preds = %5, %0
 %7 = load i32, i32* %1, align 4
 ret i32 %7
```

```
}

attributes #0 = { noinline nounwind optnone sspstrong uwtable
"correctly-rounded-divide-sqrt-fp-math"="false"
"disable-tail-calls"="false" "less-precise-fpmad"="false"
"min-legal-vector-width"="0" "no-frame-pointer-elim"="true"
"no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
"no-jump-tables"="false" "no-nans-fp-math"="false"
"no-signed-zeros-fp-math"="false" "no-trapping-math"="false"
"stack-protector-buffer-size"="8" "target-cpu"="x86-64"
"target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false"
"use-soft-float"="false" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{!"clang version 8.0.1 (tags/RELEASE_801/final)"}
```

- Podemos observar que bastante coisa foi acrescentada.
- 1ª Foi alocado mais dois espaços na memória para o número 1 e 2 que se encontram dentro do if para ser adicionados a variável inteira.
- 2ª Ele faz o load da variável int para comparação no if.
- 3ª Chama o sinal de > para comparação.
- 4ª Chama os labels que foram declarados abaixo para fazer a operação de adição, e o load das variáveis 1 e 2.

While.c # **Programa que acrescenta um while e uma operação aritmética no código anterior If.c:**

```
#include <stdio.h>

int main(){
   int i = 0;
   if (i == 0) {
       i = 1 + 2;
   }
   while(i != 0) {
       i--;
   }
```

```
}
```

While.txt **# Saída após a execução do código com o Clang:**

```
; ModuleID = 'while.c'
source_filename = "while.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 sspstrong 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 0, i32* %2, align 4
 %3 = load i32, i32* %2, align 4
 %4 = icmp eq i32 %3, 0
 br i1 %4, label %5, label %6

; <label>:5:                                      ; preds = %0
 store i32 3, i32* %2, align 4
 br label %6

; <label>:6:                                      ; preds = %5, %0
 br label %7

; <label>:7:                                      ; preds = %10, %6
 %8 = load i32, i32* %2, align 4
 %9 = icmp ne i32 %8, 0
 br i1 %9, label %10, label %13

; <label>:10:                                     ; preds = %7
 %11 = load i32, i32* %2, align 4
 %12 = add nsw i32 %11, -1
 store i32 %12, i32* %2, align 4
 br label %7

; <label>:13:                                     ; preds = %7
 %14 = load i32, i32* %1, align 4
 ret i32 %14
```

```
}

attributes #0 = { noinline nounwind optnone sspstrong uwtable
"correctly-rounded-divide-sqrt-fp-math"="false"
"disable-tail-calls"="false" "less-precise-fpmad"="false"
"min-legal-vector-width"="0" "no-frame-pointer-elim"="true"
"no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
"no-jump-tables"="false" "no-nans-fp-math"="false"
"no-signed-zeros-fp-math"="false" "no-trapping-math"="false"
"stack-protector-buffer-size"="8" "target-cpu"="x86-64"
"target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false"
"use-soft-float"="false" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{!"clang version 8.0.1 (tags/RELEASE_801/final)"}
```

- Podemos observar que novamente muitas coisas foram acrescentadas.
- As mais importantes são:
- 1ª A operação de subtração que é o comando de adição com um -1 acrescentado que faz o inverso da adição.
- 2 ª Os novos labels criados para fazer a operação de desigualdade, load do i e o store para guardar na variável a cada iteração do while.

If_else.c # **Programa que acrescenta um if e um else com operações aritméticas básicas:**

```c
#include <stdio.h>

int main(){
   int i = 0;
   int j = 2;
   if (i > 0) {
       j = 1 - i;
   } else if(i == 0) {
```

```
        i = j + 2;
    }
}
```

If_else.txt # **Saída após a execução do código com o Clang:**

```
; ModuleID = 'if_else.c'
source_filename = "if_else.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 sspstrong uwtable
define dso_local i32 @main() #0 {
 %1 = alloca i32, align 4
 %2 = alloca i32, align 4
 %3 = alloca i32, align 4
 store i32 0, i32* %1, align 4
 store i32 0, i32* %2, align 4
 store i32 2, i32* %3, align 4
 %4 = load i32, i32* %2, align 4
 %5 = icmp sgt i32 %4, 0
 br i1 %5, label %6, label %9

; <label>:6:                                      ; preds = %0
 %7 = load i32, i32* %2, align 4
 %8 = sub nsw i32 1, %7
 store i32 %8, i32* %3, align 4
 br label %16

; <label>:9:                                      ; preds = %0
 %10 = load i32, i32* %2, align 4
 %11 = icmp eq i32 %10, 0
 br i1 %11, label %12, label %15

; <label>:12:                                     ; preds = %9
 %13 = load i32, i32* %3, align 4
 %14 = add nsw i32 %13, 2
 store i32 %14, i32* %2, align 4
 br label %15
```

```
; <label>:15:                                     ; preds = %12, %9
 br label %16

; <label>:16:                                     ; preds = %15, %6
 %17 = load i32, i32* %1, align 4
 ret i32 %17
}

attributes #0 = { noinline nounwind optnone sspstrong uwtable
"correctly-rounded-divide-sqrt-fp-math"="false"
"disable-tail-calls"="false" "less-precise-fpmad"="false"
"min-legal-vector-width"="0" "no-frame-pointer-elim"="true"
"no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
"no-jump-tables"="false" "no-nans-fp-math"="false"
"no-signed-zeros-fp-math"="false" "no-trapping-math"="false"
"stack-protector-buffer-size"="8" "target-cpu"="x86-64"
"target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false"
"use-soft-float"="false" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{!"clang version 8.0.1 (tags/RELEASE_801/final)"}
```

- Como anteriormente já havíamos visto como o if( ) funciona, agora a única diferença é que foi acrescentado o else if( ).
- As diferenças são:
- 1ª Foi alocado mais espaço a uma nova variável.
- 2 ª O comando de subtração na linha 20 do código, junto com o load dos inteiros.
- 3ª A chamada de mais uma operação de comparação que agora é a igualdade.
- 4 ª O comando de adição junto com o load da variável J + um inteiro 2 e o store na variável i.