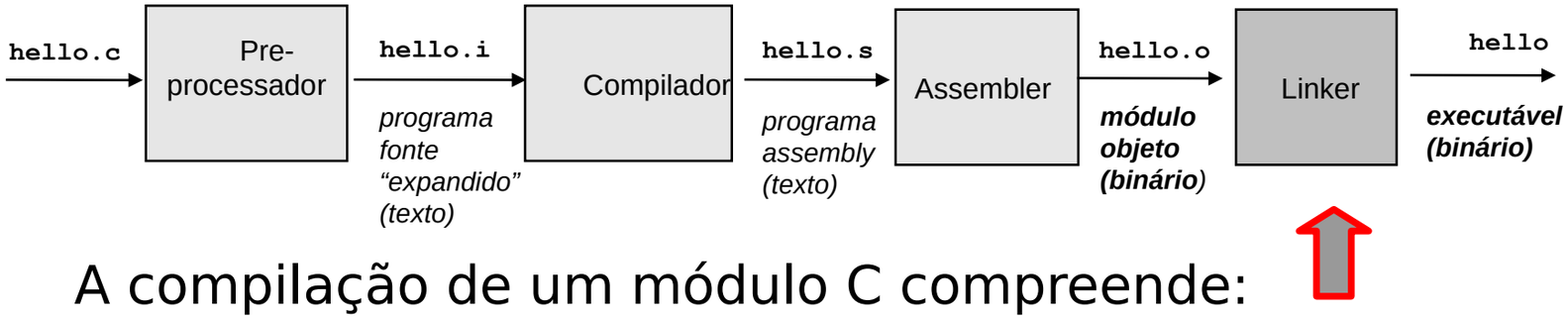
Ligação e Relocação

Noemi Rodriguez
Ana Lúcia de Moura
Raúl Renteria
Alexandre Meslin

http://www.inf.puc-rio.br/~inf1018

Compilação e Ligação



A compilação de um módulo C compreende:

- pré-processamento
- geração de código *assembly*
- geração do módulo objeto

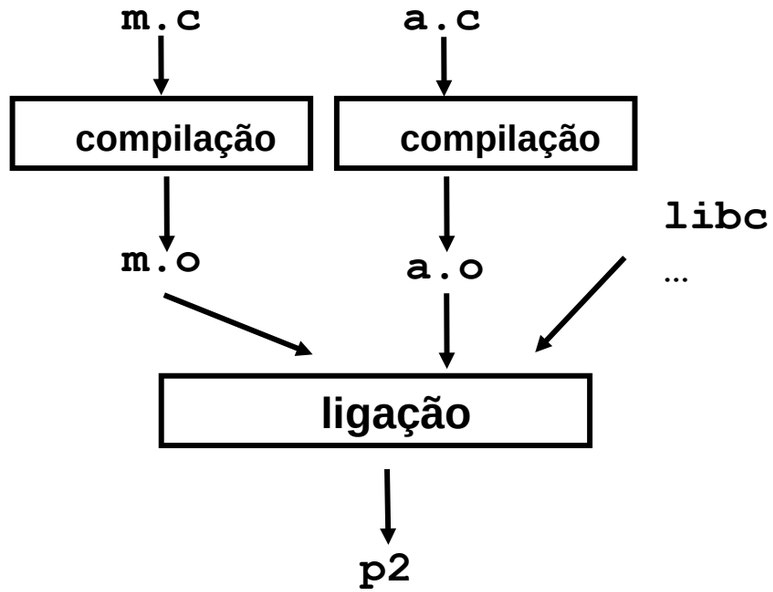
Um módulo objeto não pode ser executado:

- dependência de outros módulos (e bibliotecas): **ligação**
- endereços são relativos: **relocação**

Ligador (*linkeditor*)

Combina módulos objeto em arquivo executável

- união dos módulos (código e dados)
- resolução das referências externas
- relocação das referências à memória




```
gcc -o p2 m.c a.c
```

Módulos de bibliotecas estáticas são incluídos

- uma biblioteca reúne **módulos objeto** (libc, libm, ...)

Arquivo Objeto

Código, dados e informações para a “ligação”

- texto (código de máquina), dados (ro, inicializados e não inicializados/inicializados com 0)
- **Tabela de Símbolos**
  - símbolos **definidos** pelo módulo ("exportações")
  - símbolos **referenciados** pelo módulo ("importações")
  - símbolos “locais” (privados) ao módulo (*static*)
- **Dicionário de Relocação**
  - localização de referências a memória a serem preenchidas ou corrigidas (instruções e dados)

Exemplo Simplificado

```
extern int var1;
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var1);
  …
}
```

Exemplo Simplificado

```
extern int var1;
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var1);
  …
}
```

```
.data
.globl var2
var2: int 0

.text
.globl fun2
fun2: pushq %rbp
    …
    movl var1, %edi

    call fun1
```

Exemplo Simplificado

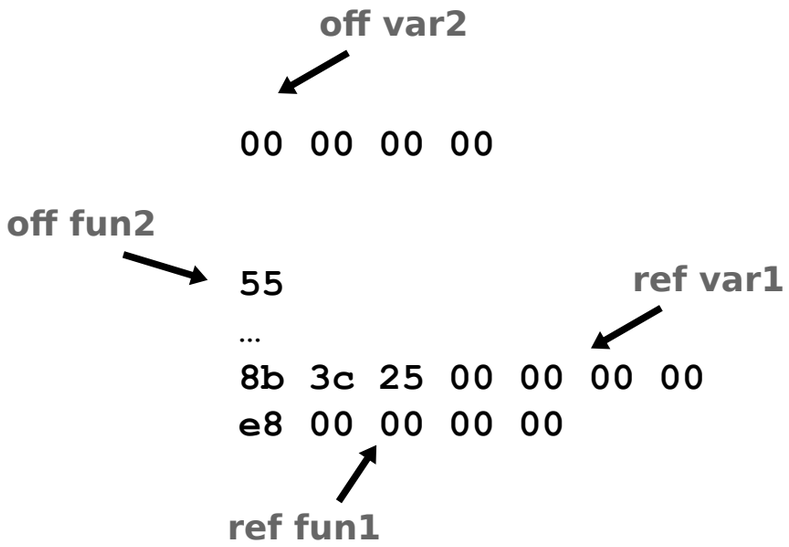
```
extern int var1;
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var1);
  …
}
```

```
.data
.globl var2
var2: int 0

.text
.globl fun2
fun2: pushq %rbp
    …
    movl var1, %edi

    call fun1
```



Exemplo Simplificado

```
extern int var1;
void fun1 (int x);

int var2 = 1;
int fun2() {
  …
  fun1(var1);
  …
}
```

```
.data
.globl var2
var2: int 0

.text
.globl fun2
fun2: pushq %rbp
    …
    movl var1, %edi

    call fun1
```

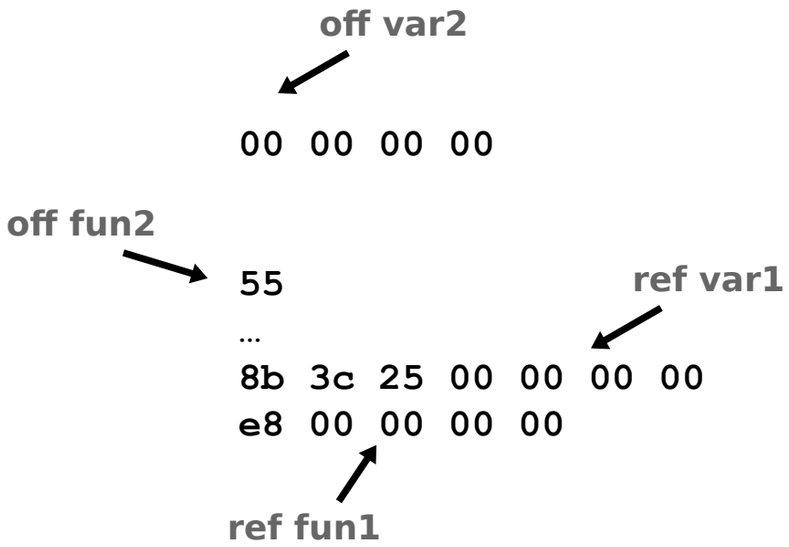


Tabela de Símbolos

var2	D	<off var2>
fun2	T	<off fun2>
var1	U	
fun1	U	

Exemplo Simplificado

```
extern int var1;
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var1);
  …
}
```

```
.data
.globl var2
var2: int 0

.text
.globl fun2
fun2: pushq %rbp
    …
    movl var1, %edi

    call fun1
```

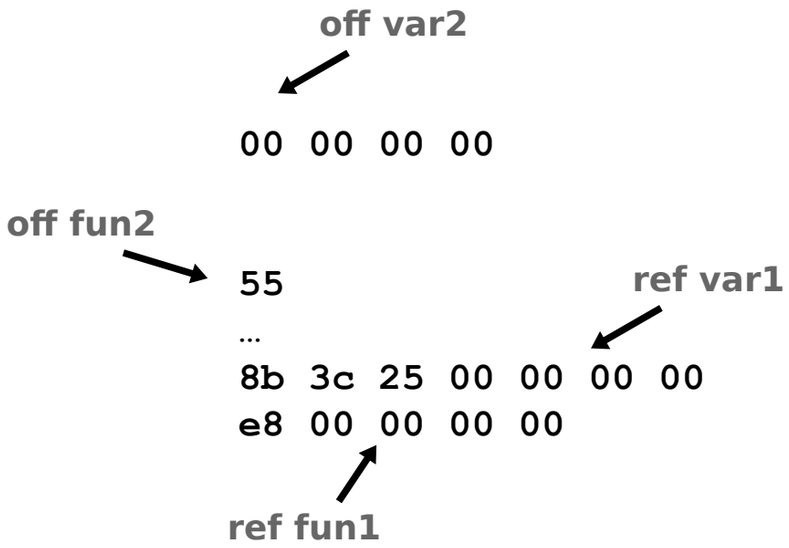


Tabela de Símbolos

var2	D	<off var2>
fun2	T	<off fun2>
var1	U	
fun1	U	

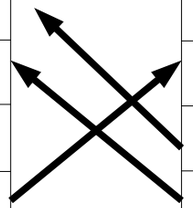
Dicionário de Relocação

<ref var1>	ref	var1
<ref fun1>	ref rel	fun1

Resolução de Referências

Módulo 1

var1	D	<off var1>
fun1	T	<off fun1>
main	T	<off main>
fun2	U	



Módulo 2

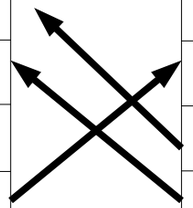
var2	D	<off var2>
fun2	T	<off fun2>
var1	U	
fun1	U	

Cada referência externa deve ser associada a uma definição única (exportação) de um símbolo **com o mesmo nome**

Resolução de Referências

Módulo 1

var1	D	<off var1>
fun1	T	<off fun1>
main	T	<off main>
fun2	U	



Módulo 2

var2	D	<off var2>
fun2	T	<off fun2>
var1	U	
fun1	U	

Cada referência externa deve ser associada a uma definição única (exportação) de um símbolo **com o mesmo nome**

- se nenhuma definição com esse nome é encontrada, o ligador termina com erro
- se há duas (ou mais) definições com o mesmo nome, o ligador também termina com erro

Erros de Ligação

```
void foo(void);

int main() {
  foo();
  return 0;
}
```

```
/tmp/cc0cCWXd.o: In function `main':
teste.c:(.text+0x7): undefined reference to `foo'
collect2: ld returned 1 exit status
```

```
int i = 1;

int foo(int j) {
  return i + j;
}
```

```
int foo(int);
int i = 0;

int main() {
  i = foo(3);
  return 0;
}
```

```
gcc -o t foo.o m.o
m.o:(.bss+0x0): multiple definition of `i'
foo.o:(.data+0x0): first defined here
collect2: ld returned 1 exit status
```

Referências em C

É importante distinguir **definição** e **referência**

- uma definição estabelece o módulo que “possui” o símbolo, e a representação na memória (localização, tamanho, ...)

Abordagem adotada por sistemas Linux, em geral:

- declaração **com inicialização** é **definição** (símbolo “forte”)

```
int i = 1024;
```

- declaração com classe **extern** é **referência**

```
extern int i;
```

- declaração sem inicialização (e sem *extern*) é símbolo “fraco”

```
int i;
```

Definições com mesmo nome: bug

```
#include <stdio.h>
void f(void);

int x = 15212;
int y = 15213;

int main() {
  f();
  printf("%d %d\n",x,y);
  return 0;
}
```

```
gcc -Wall -o f f.c f1.c
/usr/bin/ld: Warning: alignment 4 of symbol `x' in
   /tmp/ccP4WG2t.o is smaller than 8 in
   /tmp/ccobMeSD.o

./f
0 0
```

```
double x;

void f() {
    x = 0.0;
  }
```

Cuidados e Boas Práticas

modulo1.h

```
int foo(void);
```

modulo1.c

```
#include "modulo1.h"
#include "modulo2.h"

int foo() {
  contador++;
  …
}
```

modulo2.h

```
extern int contador;
```

modulo2.c

```
#include "modulo1.h"
#include "modulo2.h
int contador = 0;
int main (void) {
  int l = foo();
  …
}
```

Um símbolo global deve ser declarado **com o mesmo tipo** em todos os módulos

- a resolução de referências é feita com base apenas no **nome**!
- o compilador não tem como garantir consistência a não ser com o uso de arquivos de cabeçalho ("interface")

Relocação de Referências

O Dicionário de Relocação indica que referências à memória do módulo devem ser preenchidas/corrigidas

- referências externas ou a endereços que dependem da localização "final" do trecho de código/dados correspondente

```
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var2);
  …
}
```

```
.data
.globl var2
var2: int 0
.text
.globl fun2
fun2: pushq %rbp
   …
   movl var2, %edi

   call fun1
```

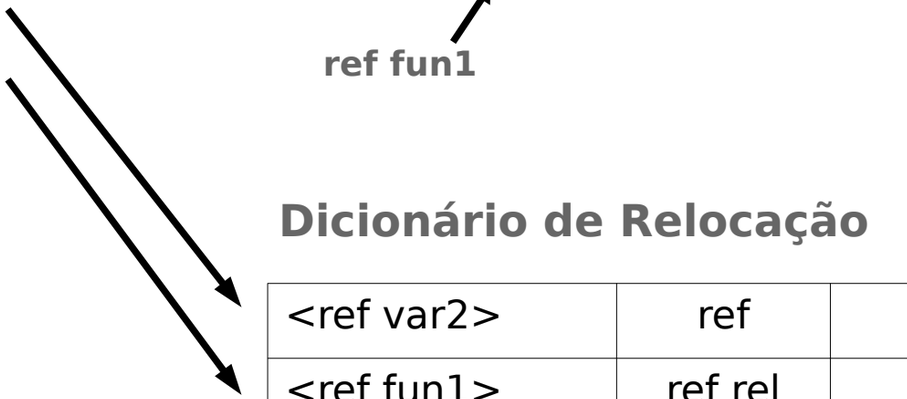
```
55                    ref var2
…                        ↙
8b 3c 25 00 00 00 00
e8 00 00 00 00
   ↖
ref fun1
```

Relocação de Referências

O ligador produz uma tabela com os endereços dos símbolos globais definidos por todos os módulos

Tabela de Símbolos

var1	D	<end var1>
var2	D	<end var2>
fun1	T	<end fun1>
main	T	<end main>
fun2	T	<end fun2>



Dicionário de Relocação

<ref var2>	ref	var2
<ref fun1>	ref rel	fun1

Carga e Relocação

Na ligação, as referências à memória são calculadas em relação ao endereço "virtual" do programa

- quando o programa é carregado na memória, essas referências devem ser relocadas em relação ao endereço "real"
- essa relocação pode ser feita através de uma tradução de endereços feita pelo hardware

Bibliotecas Dinâmicas

Carregadas na memória e "ligadas" em tempo de execução a um programa

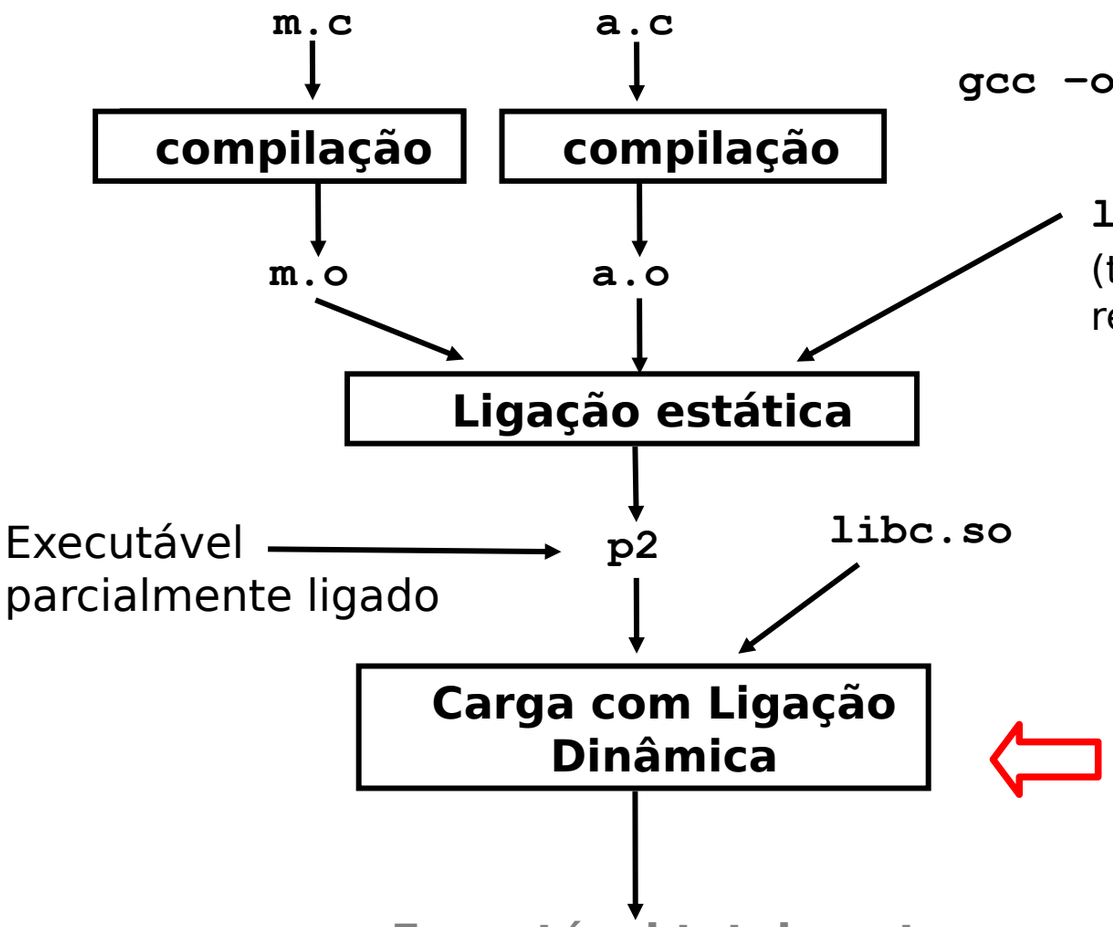
- *shared objects* (.so), *dynamic link libraries* (.dll)

Melhor aproveitamento de espaço

- bibliotecas estáticas são incorporadas no programa
- bibliotecas dinâmicas (código) são compartilhadas

A carga e ligação da biblioteca é realizada por um *ligador dinâmico*

Ligação com Bibliotecas Dinâmicas




```
gcc -o p2 m.c a.c
```

libc.so
(tabela de símbolos e informações para relocação)

módulos de libc.so são carregados na memória

referências a símbolos de libc em p2 são relocadas

Exemplo:

Meu include:

```
#define INICIO 10
extern int x;
int calcula(int x);
int valor(int x);
```

Exemplo:

Módulo 1:

```
#include <stdio.h>
#include "meuInclude.h"
int calcula(int x);
int main(void) {
  printf("%d\n", calcula(x));
  return 0;
}
int valor(int x) {
  return x + 2;
}
```

Exemplo:

Módulo 2:

```
#include "meuInclude.h"
int x = INICIO;
int calcula(int x) {
        x = valor(x);
        return x + 4;
}
```

Exemplo:

Módulo 2: `$ gcc -o modulo2.i modulo2.c -E`

```
#include "meuInclude.h"
int x = INICIO;
int calcula(int x) {
        x = valor(x);
        return x + 4;
}
```

```
# 1 "modulo2.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "modulo2.c"
# 1 "meuInclude.h" 1

extern int x;
int calcula(int x);
int valor(int x);
# 2 "modulo2.c" 2
int x = 10;
int calcula(int x) {
 x = valor(x);
 return x + 4;
}
```

Exemplo:

Módulo 2: `$ gcc -o modulo2.s modulo2.c -S`

```
#include "meuInclude.h"
int x = INICIO;
int calcula(int x) {
        x = valor(x);
        return x + 4;
}
```

```
    .file   "modulo2.c"
    .text
    .globl  x
    .data
    .align 4
    .type   x, @object
    .size   x, 4
x:
    .long   10
    .text
    .globl  calcula
    .type   calcula, @function
calcula:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq%rsp, %rbp
    .cfi_def_cfa_register 6
    subq$16, %rsp
    movl%edi, -4(%rbp)
    movl-4(%rbp), %eax
    movl%eax, %edi
```

DEPARTAMENTO DE INFORMÁTICA PUC·RIO

Exemplo:

Módulo 1: `$ gcc -o modulo1.o modulo1.c -c`

```c
#include <stdio.h>
#include "meuInclude.h"
int calcula(int x);
int main(void) {
  printf("%d\n", calcula(x));
  return 0;
}
int valor(int x) {
  return x + 2;
}
```

```
Disassembly of section .text:

0000000000000000 <main>:
   0:   f3 0f 1e fa           endbr64
   4:   55                    push   %rbp
   5:   48 89 e5              mov    %rsp,%rbp
   8:   8b 05 00 00 00 00     mov    0x0(%rip),%eax        # e <main+0xe>
   e:   89 c7                 mov    %eax,%edi
  10:   e8 00 00 00 00        callq  15 <main+0x15>
  15:   89 c6                 mov    %eax,%esi
  17:   48 8d 3d 00 00 00 00  lea    0x0(%rip),%rdi        # 1e <main+0x1e>
  1e:   b8 00 00 00 00        mov    $0x0,%eax
  23:   e8 00 00 00 00        callq  28 <main+0x28>
  28:   b8 00 00 00 00        mov    $0x0,%eax
  2d:   5d                    pop    %rbp
  2e:   c3                    retq

000000000000002f <valor>:
  2f:   f3 0f 1e fa           endbr64
  33:   55                    push   %rbp
  34:   48 89 e5              mov    %rsp,%rbp
  37:   89 7d fc              mov    %edi,-0x4(%rbp)
  3a:   8b 45 fc              mov    -0x4(%rbp),%eax
  3d:   83 c0 02              add    $0x2,%eax
  40:   5d                    pop    %rbp
```

DEPARTAMENTO DE INFORMÁTICA
PUC·RIO

Exemplo:

Módulo 2: `$ gcc -o modulo2.o modulo2.c -c`

```
#include "meuInclude.h"
int x = INICIO;
int calcula(int x) {
        x = valor(x);
        return x + 4;
}
```

```
Disassembly of section .text:

0000000000000000 <calcula>:
   0:    f3 0f 1e fa           endbr64
   4:    55                    push   %rbp
   5:    48 89 e5              mov    %rsp,%rbp
   8:    48 83 ec 10           sub    $0x10,%rsp
   c:    89 7d fc              mov    %edi,-0x4(%rbp)
   f:    8b 45 fc              mov    -0x4(%rbp),%eax
  12:    89 c7                 mov    %eax,%edi
  14:    e8 00 00 00 00        callq  19 <calcula+0x19>
  19:    89 45 fc              mov    %eax,-0x4(%rbp)
  1c:    8b 45 fc              mov    -0x4(%rbp),%eax
  1f:    83 c0 04              add    $0x4,%eax
  22:    c9                    leaveq
  23:    c3                    retq

Disassembly of section .data:

0000000000000000 <x>:
   0:    0a 00                 or     (%rax),%al
     ...
```

Exemplo:

```
$ objdump -Ds modulo1.o | less
$ objdump -Ds modulo2.o | less
```

Módulo 1

```
Disassembly of section .text:

0000000000000000 <main>:
   0:    f3 0f 1e fa           endbr64
   4:    55                    push %rb
   5:    48 89 e5              mov %rsp
   8:    8b 05 00 00 00 00     mov 0x0(
   e:    89 c7                 mov %eax
  10:    e8 00 00 00 00        callq 15
  15:    89 c6                 mov %eax
  17:    48 8d 3d 00 00 00 00  lea 0x0(
  1e:    b8 00 00 00 00        mov $0x0
  23:    e8 00 00 00 00        callq 28
  28:    b8 00 00 00 00        mov $0x0
  2d:    5d                    pop %rbp
  2e:    c3                    retq

000000000000002f <valor>:
  2f:    f3 0f 1e fa           endbr64
  33:    55                    push %rb
  34:    48 89 e5              mov %rsp
  37:    89 7d fc              mov %edi
  3a:    8b 45 fc              mov -0x4
  3d:    83 c0 02              add $0x2
  40:    5d                    pop %rbp
  41:    c3                    retq
```

Módulo 2

```
Disassembly of section .text:

0000000000000000 <calcula>:
   0:    f3 0f 1e fa           endbr64
   4:    55                    push %rbp
   5:    48 89 e5              mov %rsp,
   8:    48 83 ec 10           sub $0x10
   c:    89 7d fc              mov %edi,
   f:    8b 45 fc              mov -0x4(
  12:    89 c7                 mov %eax,
  14:    e8 00 00 00 00        callq 19
  19:    89 45 fc              mov %eax,
  1c:    8b 45 fc              mov -0x4(
  1f:    83 c0 04              add $0x4,
  22:    c9                    leaveq
  23:    c3                    retq

Disassembly of section .data:

0000000000000000 <x>:
   0:    0a 00                 or (%rax
     ...
```

Exemplo:

Executável: `$ gcc -Wall -o exemplo modulo1.c modulo2.c`

```
0000000000001149 <main>:
    1149:       f3 0f 1e fa             endbr64
    114d:       55                      push   %rbp
    114e:       48 89 e5                mov    %rsp,%rbp
    1151:       8b 05 b9 2e 00 00       mov    0x2eb9(%rip),%eax        # 4010 <x>
    1157:       89 c7                   mov    %eax,%edi
    1159:       e8 2d 00 00 00          callq  118b <calcula>
    115e:       89 c6                   mov    %eax,%esi
    1160:       48 8d 3d 9d 0e 00 00    lea    0xe9d(%rip),%rdi        # 2004 <_IO
    1167:       b8 00 00 00 00          mov    $0x0,%eax
    116c:       e8 df fe ff ff          callq  1050 <printf@plt>
    1171:       b8 00 00 00 00          mov    $0x0,%eax
    1176:       5d                      pop    %rbp
    1177:       c3                      retq

0000000000001178 <valor>:
    1178:       f3 0f 1e fa             endbr64
    117c:       55                      push   %rbp
    117d:       48 89 e5                mov    %rsp,%rbp
    1180:       89 7d fc                mov    %edi,-0x4(%rbp)
    1183:       8b 45 fc                mov    -0x4(%rbp),%eax
    1186:       83 c0 02                add    $0x2,%eax
    1189:       5d                      pop    %rbp
    118a:       c3                      retq

000000000000118b <calcula>:
    118b:       f3 0f 1e fa             endbr64
    118f:       55                      push   %rbp
```

DEPARTAMENTO DE INFORMÁTICA
PUC·RIO

PIC: Position-Independent Code

Código que pode ser carregado sem modificação pelo ligador (relocação de referências a variáveis globais e funções)

- acesso a referências externas é “indireto”

Acesso a dados: GOT (Global Offset Table)

- código gerado obtem o endereço do símbolo na GOT e o acessa “indiretamente”
- o ligador dinâmico reloca apenas as entradas da GOT

Chamada de funções: PLT (Procedure Linkage Table)

- trechos de código que usam entradas na GOT para chamar (indiretamente) funções externas

PIC: Position-Independent Code

```
#include <stdio.h>

int main(void) {
        int n;
        puts("Começando:");
        printf("Entre com um número: ");
        scanf("%d", &n);
        printf("%d\n", n);
        puts("FIM!");
        return 0;
}
```

PIC: Position-Independent Code

$ objdump -Ds pic | less

pic.c

```
#include <stdio.h>

int main(void) {
    int n;
    puts("Começando:");
    printf("Entre com um número: ");
    scanf("%d", &n);
    printf("%d\n", n);
    puts("FIM!");
    return 0;
}
```

pic

```
00000000000011a9 <main>:
11a9: f3 0f 1e fa             endbr64
11ad: 55                      push   %rbp
11ae: 48 89 e5                mov    %rsp,%rbp
11b1: 48 83 ec 10             sub    $0x10,%rsp
11b5: 64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
11bc: 00 00
11be: 48 89 45 f8             mov    %rax,-0x8(%rbp)
11c2: 31 c0                   xor    %eax,%eax
11c4: 48 8d 3d 39 0e 00 00    lea    0xe39(%rip),%rdi
11cb: e8 b0 fe ff ff          callq  1080 <puts@plt>
11d0: 48 8d 3d 39 0e 00 00    lea    0xe39(%rip),%rdi
11d7: b8 00 00 00 00          mov    $0x0,%eax
11dc: e8 bf fe ff ff          callq  10a0 <printf@plt>
11e1: 48 8d 45 f4             lea    -0xc(%rbp),%rax
11e5: 48 89 c6                mov    %rax,%rsi
11e8: 48 8d 3d 38 0e 00 00    lea    0xe38(%rip),%rdi
11ef: b8 00 00 00 00          mov    $0x0,%eax
11f4: e8 b7 fe ff ff          callq  10b0 <scanf@plt>
11f9: 8b 45 f4                mov    -0xc(%rbp),%eax
11fc: 89 c6                   mov    %eax,%esi
11fe: 48 8d 3d 25 0e 00 00    lea    0xe25(%rip),%rdi
1205: b8 00 00 00 00          mov    $0x0,%eax
120a: e8 91 fe ff ff          callq  10a0 <printf@plt>
120f: 48 8d 3d 18 0e 00 00    lea    0xe18(%rip),%rdi
1216: e8 65 fe ff ff          callq  1080 <puts@plt>
121b: b8 00 00 00 00          mov    $0x0,%eax
```

PIC: Position-Independent Code

$ objdump -Ds pic | less

```
Contents of section .got:
 3fa0 b03d0000 00000000 00000000 00000000  .=..............
 3fb0 00000000 00000000 30100000 00000000  ........0.......        # 3FB8 puts
 3fc0 40100000 00000000 50100000 00000000  @.......P.......        # 3FC8 printf
 3fd0 60100000 00000000 00000000 00000000  `...............        # 3FD0 scanf
 3fe0 00000000 00000000 00000000 00000000  ................
 3ff0 00000000 00000000 00000000 00000000  ................

Disassembly of section .plt.sec:

0000000000001080 <puts@plt>:
    1080:       f3 0f 1e fa             endbr64
    1084:       f2 ff 25 2d 2f 00 00    bnd jmpq *0x2f2d(%rip)        # 3fb8 <puts@GLIBC_2.2.5>
    108b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

0000000000001090 <__stack_chk_fail@plt>:
    1090:       f3 0f 1e fa             endbr64
    1094:       f2 ff 25 25 2f 00 00    bnd jmpq *0x2f25(%rip)        # 3fc0 <__stack_chk_fail@GLIBC_2.4>
    109b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

00000000000010a0 <printf@plt>:
    10a0:       f3 0f 1e fa             endbr64
    10a4:       f2 ff 25 1d 2f 00 00    bnd jmpq *0x2f1d(%rip)        # 3fc8 <printf@GLIBC_2.2.5>
    10ab:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

00000000000010b0 <__isoc99_scanf@plt>:
    10b0:       f3 0f 1e fa             endbr64
    10b4:       f2 ff 25 15 2f 00 00    bnd jmpq *0x2f15(%rip)        # 3fd0 <__isoc99_scanf@GLIBC_2.7>
    10bb:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
```

DEPARTAMENTO DE INFORMÁTICA PUC·RIO

Ligação e Relocação

Noemi Rodriguez
Ana Lúcia de Moura
Raúl Renteria
Alexandre Meslin

http://www.inf.puc-rio.br/~inf1018