Construção de um compilador de MiniLua para x86 usando Objective Caml

João Paulo Nóbrega Alvim
joaopaulonobregaalvim@gmail.com

Faculdade de Engenharia Elétrica
Graduação em Engenharia de Computação
Universidade Federal de Uberlândia

26 de agosto de 2017

Lista de Listagens

Sumário

Capítulo 1

Introdução

O primeiro passo para a criação de um compilador, é a análise e compreensão do código em Assembly para posteriormente prosseguir para a construção do mesmo em si. Este documento faz a análise de códigos assembly gerados a partir da linguagem MiniLua para a plataforma x86, abordando todas as etapas do processo.

1.1 Compilador e Instruções

O compilador utilizado neste trabalho é o GCC (Gnu Compiler Collection). Este compilador não inclui front-end para Lua ou MiniLua, dessa forma, foi realizada uma etapa intermediária no processo de geração do código assembly: Todos os códigos em MiniLua foram traduzidos para C, e a análise foi feita a partir destes.

1. Para compilar os arquivos criados em C, foi utilizada a seguinte instrução:

```
> gcc nome_do_arquivo.c -o nome_do_executavel
```

2. Para executar este arquivo, foi realizado:

```
> ./nome_do_executavel
```

3. Após esta etapa, foi gerado o assembly destes arquivo, através do comando:

```
> gcc -S nome_do_arquivo.c
```

4. Este processo resultará na criação de um arquivo de extensão .s, onde nele conterá o código em linguagem de máquina. Para compilar e executar um arquivo em assembly, vamos precisar seguir os seguintes passos:

```
> gcc -c nome_do_arquivo.s
```

5. Essa linha de código gerará um arquivo com a extensão .o, que é um código objeto, com isso fazemos:

```
> gcc  nome_do_arquivo.o -o nome_do_arquivo.exe
```

6. Assim temos o arquivo em assembly compilado e pronto para ser executado, a partir do comando já mostrado anteriormente

```
> ./nome_do_arquivo.exe
```

1.2 Mnemônicos e Diretivas

Tanto os Mnemônicos quanto as Diretivas abordadas em todas as codificações presentes nestes documentos podem ser consultados de forma mais detalhada e completa no capítulo Apêndice, onde lá foram tratados com maior atenção.

1. Mnemônicos Mnemônicos são símbolos utilizados para representar os valores brutos, padrões de bits compreendidos pela máquina. Como exemplo podemos citar o mnemônico ADD.

   ADD x1, x2

   o ADD faz a soma dos dois parâmetros passados, ou seja dos valores presentes na posição de memória indicada do registrador.

2. Diretivas As diretivas são comandos que são parte da sintaxe do montador mas que não estão relacionadas com o set de instruções do x86. Estas também são abordadas no apêndice.

Capítulo 2

Codificação e Tradução de Pseudo-Códigos

2.1 Códigos escritos na linguagem Lua e suas traduções em C

Nesta seção, são apresentados os pseudo-códigos propostos escritos na linguagem Lua e em seguida suas respectivas transcrições para linguagem C. Na próximas seções serão abordados os códigos assembly gerados e suas características.

2.1.1 Nano Programas

Nesta subseção são apresentados os nano programas, programas com baixo nível de complexidade.

1. nano01. Módulo mínimo que caracteriza um programa

   Lua:

   Listagem 2.1: nano01.lua

```lua
function main()
end
```

   C:

   Listagem 2.2: nano01.c

```c
#include<stdio.h>

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

2. nano02. Declaração de uma variável

   Lua:

   Listagem 2.3: nano02.lua

```lua
function main()
   local n
end
```

   C:

   Listagem 2.4: nano02.c

```c
#include<stdio.h>

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

3. nano03. Atribuição de um valor à uma variável

   Lua:

   Listagem 2.5: nano03.lua

```lua
function main()
   local n
   n = 1
end

main()
```

   C:

   Listagem 2.6: nano03.c

```c
#include<stdio.h>

int main()
{
  int n;
  n = 1;
  return 0;
}
```

4. nano04. Atribuição de uma soma de valores à uma variável

   Lua:

   Listagem 2.7: nano04.lua

```lua
function main()
   local n
   n = 1 + 2
end

main()
```

C:

Listagem 2.8: nano04.c

```c
#include<stdio.h>

int main()
{
  int n;
  n = 1 + 2;
  return 0;
}
```

5. nano05. Comando de impressão.

   Lua:

Listagem 2.9: nano05.lua

```lua
function main()
   local n
   n = 2
   print(n)
end

main()
```

C:

Listagem 2.10: nano05.c

```c
#include <stdio.h>

int main()
{
  int n;
  n = 2;
  printf("%d",n);
  return 0;
}
```

6. nano06. Atribuição de uma subtração à uma variável

   Lua:

Listagem 2.11: nano06.lua

```lua
function main()
   local n
   n = 1 - 2
   print(n)
end

main()
```

C:

Listagem 2.12: nano06.c

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

```
2
3 int main()
4 {
5   int n;
6   n = 1 - 2;
7   printf("%d",n);
8   return 0;
9 }
```

7. nano07. Inclusão de condicionalidade

   Lua:

   Listagem 2.13: nano07.lua

```
1 function main()
2    local n
3    n = 1
4    if n == 1 then
5       print(n)
6    end
7 end
8
9 main()
```

   C:

   Listagem 2.14: nano07.c

```
1 #include<stdio.h>
2
3 int main()
4 {
5   int n;
6   n = 1;
7   if(n == 1)
8     {
9       printf("%d", n);
10    }
11  return 0;
12 }
```

8. nano08. Inclusão de condicionalidade com 'else'

   Lua:

   Listagem 2.15: nano08.lua

```
1 function main()
2    local n
3    n = 1
4    if n == 1 then
5       print(n)
6    else
7      print(0)
8    end
9 end
10
11 main()
```

C:

Listagem 2.16: nano08.c

```c
#include<stdio.h>

int main()
{
  int n;
  n = 1;
  if(n == 1)
    {
      printf("%d", n);
    }
  else
    {
      printf("0");
    }
  return 0;
}
```

9. nano09. Atribuição de duas operações aritméticas sobre inteiros a uma variável

Lua:

Listagem 2.17: nano09.lua

```lua
function main()
   local n
   n = (1 + 1) / 2
   if n == 1 then
      print(n)
   else
      print(0)
   end
end

main()
```

C:

Listagem 2.18: nano09.c

```c
#include<stdio.h>

int main()
{
  int n;
  n = (1 + 1)/2;
  if(n == 1)
    {
      printf("%d", n);
    }
  else
    {
      printf("0");
    }
}
```

10. nano10. Atribuição de duas variáveis inteiras

Lua:

Listagem 2.19: nano10.lua

```lua
function main()
   local n, m
   n = 1
   m = 2
   if n == m then
      print(n)
   else
      print(0)
   end
end

main()
```

C:

Listagem 2.20: nano10.c

```c
#include<stdio.h>

int main()
{
  int n;
  int m;
  n = 1;
  m = 2;
  if(n == m)
    {
      printf("%d", n);
    }
  else
    {
      printf("0");
    }
}
```

11. nano11. Comando de repetição 'while'

Lua:

Listagem 2.21: nano11.lua

```lua
function main()
   local n, m, x
   n = 1
   m = 2
   x = 5
   while x > n do
         n = n + m
         print(n)
   end
end

main()
```

C:

Listagem 2.22: nano11.c

```c
#include<stdio.h>

int main()
{
  int n;
  int m;
  int x;
  n = 1;
  m = 2;
  x = 5;
  while(x > n)
    {
      n = n + m;
      printf("%d ", n);
    }
}
```

12. nano12. Junção de Condicionalidade com 'while'

Lua:

Listagem 2.23: nano12.lua

```lua
function main()
   local n, m, x
   n = 1
   m = 2
   x = 5
   while x > n do
         if n == m then
            print(n)
         else
            print(0)
         end
         x = x - 1
   end
end

main()
```

C:

Listagem 2.24: nano12.c

```c
#include<stdio.h>

int main()
{
  int n;
  int m;
  int x;
  n = 1;
  m = 2;
  x = 5;
  while(x > n)
    {
      if(n == m)
  {
```

```
15      printf("%d ", n);
16    }
17        else
18    {
19      printf("0 ");
20    }
21        x = x - 1;
22      }
23  }
```

2.1.2 Micro Programas

Nesta subseção serão apresentados os micro programas, programas que apesar de simples, possuem maior grau de complexidade.

1. micro01. Conversão de Celsius para Farenheit.

   Lua:

   Listagem 2.25: micro01.lua

```
1  function main()
2     local cel, far
3     print("Tabela de conversão: Celsius -> Fahrenheit")
4     print("Digite a temperatura em Celsius: ")
5     cel = io
6     .read("*number")
7     far = (9*cel+160)/5
8     print("A nova temperatura eh: "..far.." F")
9  end
10
11 main()
```

   C:

   Listagem 2.26: micro01.c

```
1  #include<stdio.h>
2
3  int main()
4  {
5    float cel;
6    float far;
7
8    printf("Tabela de conversão: Celsius -> Fahrenheit\n");
9    printf("Digite a temperatura em Celsius: ");
10   scanf("%f", &cel);
11   far = ((9*cel+160)/(5));
12   printf("A nova temperatura é: %f F \n",far);
13   return 0;
14 }
```

2. micro02. Ler dois inteiros e ver qual é maior.

   Lua:

Listagem 2.27: micro02.lua

```
function main()
   local num1, num2
   print("Digite o primeiro numero: ")
   num1 = io.read("*number")
   print("Digite o segundo numero: ")
   num2 = io.read("*number")

   if num1 > num2 then
      print("O primeiro número "..num1.." é maior que o segundo "..
          num2)
   else
    print("O segundo número "..num2.." é maior que o primeiro "..num1
        )
   end
end

main()
```

C:

Listagem 2.28: micro02.c

```
#include<stdio.h>

int main()
{
  int num1;
  int num2;

  printf("Digite o primeiro número: ");
  scanf("%d", &num1);
  printf("Digite o segundo número: ");
  scanf("%d", &num2);

  if (num1 > num2)
    {
      printf("O primeiro número %d é maior que o segundo %d", num1,
          num2);
    }
  else
    {
      printf("O segundo número %d é maior que o primeiro %d", num2,
          num1);
    }
}
```

3. micro03. Lê um número e verifica se está entre 100 e 200.

   Lua:

Listagem 2.29: micro03.lua

```
function main()
   local numero
   print("Digite um número: ")
   numero = io.read("*number")
   if numero >= 100 then
      if numero <= 200 then
```

```
7           print("O número está no intervalo entre 100 e 200")
8       else
9     print("O número não está no intervalo entre 100 e 200")
10      end
11    else
12       print("O número não está no intervalo entre 100 e 200")
13    end
14 end
15
16 main()
```

C:

Listagem 2.30: micro03.c

```
1 #include<stdio.h>
2
3 int main()
4 {
5   int numero;
6
7   printf("Digite um número: ");
8   scanf("%d", &numero);
9   if (numero >= 100)
10     {
11       if (numero <= 200)
12   {
13     printf("O número está no intervalo entre 100 e 200\n");
14   }
15       else
16   {
17     printf("O número não está no intervalo entre 100 e 200\n");
18   }
19     }
20   else
21     {
22       printf("O número não está no intervalo entre 100 e 200\n");
23     }
24 }
```

4. micro04. Lê números e fala quais estão entre 10 e 150.

   Lua:

Listagem 2.31: micro04.lua

```
1 function main()
2    local x, num, intervalo
3    intervalo = 0
4    for x=1, 5, 1 do
5        print("Digite um número: ")
6        num = io.read("*number")
7        if num >= 10 then
8      if num <= 150 then
9         intervalo = intervalo + 1
10     end
11       end
12   end
13   print("Ao total, foram digitados "..intervalo.." números no
       intervalo entre 10 e 150")
```

```
14
15 end
16
17 main()
```

C:

Listagem 2.32: micro04.c

```
1 #include<stdio.h>
2
3 int main()
4 {
5   int x;
6   int num;
7   int intervalo = 0;
8
9   for(x = 0; x < 5; x++)
10    {
11      printf("Digite um número: ");
12      scanf("%d", &num);
13      if (num >= 10)
14  {
15    if (num <= 150)
16      {
17        intervalo = intervalo + 1;
18      }
19  }
20    }
21  printf("Ao total, foram digitados %d números no intervalo entre 10
      e 150\n", intervalo);
22  return 0;
23 }
```

5. micro05. Lê strings e caracteres.

   Lua:

Listagem 2.33: micro05.lua

```
1 function main()
2    local nome, sexo
3    local x, h, m = 1, 0, 0
4    for x=1, 5, 1 do
5        print("Digite o nome: ")
6        nome = io.read("*line")
7        print("H - Homem ou M = Mulher: ")
8      sexo = io.read("*line")
9      if sexo == 'H' then
10        h = h + 1
11     elseif sexo == 'M' then
12        m = m + 1
13     else
14        print("Sexo so pode ser H ou M!")
15        end
16    end
17    print("Foram inseridos "..h.." Homens")
18    print("Foram inseridos "..m.." Mulheres")
19 end
20
```

```
21 main()
```

C:

Listagem 2.34: micro05.c

```
1  #include<stdio.h>
2
3  int main()
4  {
5    char nome[50];
6    int x;
7    int h = 0;
8    int m = 0;
9    int op;
10
11   for(x = 0; x < 5; x++)
12     {
13       printf("Digite o nome: ");
14       scanf("%s", nome);
15       printf("1 - Homem ou 2 - Mulher: ");
16       scanf("%d", &op);
17       switch(op)
18   {
19   case 1:
20     h = h + 1;
21     break;
22   case 2:
23     m = m + 1;
24     break;
25   default:
26     printf("Sexo só pode ser Homem(1) ou Mulher(2)!\n");
27     break;
28   }
29     }
30   printf("Foram inseridos %d Homens.\n", h);
31   printf("Foram inseridos %d Mulheres.\n", m);
32
33   return 0;
34 }
```

6. micro06. Escreve um número lido por extenso.

   Lua:

Listagem 2.35: micro06.lua

```
1  function main()
2     local num
3     print("Digite um numero de 1 a 5: ")
4     num = io.read("*number")
5     if num == 1 then
6        print("Um")
7     elseif num == 2 then
8           print("Dois")
9     elseif num == 3 then
10       print("Tres")
11    elseif num == 4 then
12       print("Quatro")
13    elseif num == 5 then
```

```
14        print("Cinco")
15     else
16        print("Numero invalido!!!")
17     end
18 end
19
20 main()
```

C:

Listagem 2.36: micro06.c

```
1  #include<stdio.h>
2
3  int main()
4  {
5    int numero;
6    printf("Digite um número de 1 a 5: ");
7    scanf("%d", &numero);
8    switch(numero)
9      {
10     case 1:
11       printf("Um\n");
12       break;
13     case 2:
14       printf("Dois\n");
15       break;
16     case 3:
17       printf("Três\n");
18       break;
19     case 4:
20       printf("Quatro\n");
21       break;
22     case 5:
23       printf("Cinco\n");
24       break;
25     default:
26       printf("Número inválido!!!\n");
27       break;
28     }
29   return 0;
30 }
```

7. micro07. Verifica se os números são >0 , <0 , ou =0.

   Lua:

Listagem 2.37: micro07.lua

```
1  function main()
2     local programa, numero, opc
3     programa = 1
4     while programa == 1 do
5           print("Digite um numero: ")
6           numero = io.read("*n")
7           if numero > 0 then
8              print("Positivo")
9           else
10             if numero == 0 then
11          print("O numero é igual a zero")
```

```
12        end
13        if numero < 0 then
14           print("Negativo")
15        end
16     end
17     print("Deseja finalizar? (S-1/N-2) ")
18     opc = io.read("*n")
19     if opc == 1 then
20        programa = 0
21     end
22   end
23 end
24
25 main()
```

C:

Listagem 2.38: micro07.c

```
1  #include<stdio.h>
2  #include<string.h>
3
4  int main()
5  {
6    int programa;
7    int numero;
8    char opc[5];
9
10   programa = 1;
11   do
12     {
13       printf("Digite um número: ");
14       scanf("%d", &numero);
15       if (numero > 0)
16   {
17     printf("Positivo\n");
18   }
19       else if (numero == 0)
20   {
21     printf("O número é igual a 0\n");
22   }
23       else
24   {
25     printf("Negativo\n");
26   }
27       printf("Deseja finalizar? (sim/nao) ");
28       scanf("%s", opc);
29       if(strcmp (opc, "sim") == 0)
30   {
31     programa = 0;
32   }
33     }
34   while(programa == 1);
35   return 0;
36 }
```

8. micro08. Verifica se um número é maior ou menor que 10.

   Lua:

Listagem 2.39: micro08.lua

```
function main()
   local numero
   numero = 1
   while numero ~= 0 do
         print("Digite um número: ")
         numero = io.read("*n")
         if numero > 10 then
            print("O número "..numero.." é maior que 10")
         else
            print("O número "..numero.." é menor que 10")
         end
          end
end

main()
```

C:

Listagem 2.40: micro08.c

```
#include<stdio.h>

int main()
{
  int numero;
  numero = 1;
  do
    {
      printf("Digite um número: ");
      scanf("%d", &numero);
      if(numero > 10)
  {
    printf("O numero %d é maior que 10 \n", numero);
  }
      else
  {
    printf("O numero %d é menor que 10 \n", numero);
  }
    }
  while(numero != 0);
  return 0;
}
```

9. micro09. Calcula preços.

   Lua:

Listagem 2.41: micro09.lua

```
function main()
   local preco, venda, novo_preco
   print("Digite o preço: ")
   preco = io.read("*n")
   print("Digite a venda: ")
   venda = io.read("*n")
   if venda < 500 or preco < 30 then
      novo_preco = preco + 10/100 * preco
```

```
9     else if (venda >= 500 and venda < 1200) or (preco >= 30 and preco
          < 80) then
10       novo_preco = preco + 15/100 * preco
11    else if venda >= 1200 or preco >= 80 then
12       novo_preco = preco - 20/100 * preco
13    end
14      end
15   end
16   print("O novo preço é "..novo_preco)
17 end
18
19 main()
```

C:

Listagem 2.42: micro09.c

```
1 #include<stdio.h>
2
3 int main()
4 {
5   float preco;
6   float venda;
7   float novo_preco;
8
9   printf("Digite o preço: ");
10  scanf("%f", &preco);
11  printf("Digite a venda: ");
12  scanf("%f", &venda);
13  if((venda < 500) || (preco < 30))
14    {
15      novo_preco = (preco + 0.1*preco);
16    }
17  else if((venda >= 500 && venda < 1200) || (preco >= 30 && preco <
        80))
18    {
19      novo_preco = (preco + 0.15*preco);
20    }
21  else if(venda >= 1200 || preco >= 80)
22    {
23      novo_preco = (preco - 0.2*preco);
24    }
25  printf("O novo preço é: %f", novo_preco);
26  return 0;
27 }
```

10. micro10. Cálculo de fatorial.

    Lua:

Listagem 2.43: micro10.lua

```
1 function fatorial(n)
2    if n <= 0 then
3       return 1
4    else
5       return n * fatorial(n-1)
6       end
7    end
8
```

```lua
function main()
   local numero, fat
   print("Digite um número: ")
   numero = io.read("*n")
   fat = fatorial(numero)
   print("O fatorial de ")
   print(numero)
   print(" é ")
   print(fat)
end

main()
```

C:

Listagem 2.44: micro10.c

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

int fator(int numero)
{
  if(numero > 1)
    return numero * fator(numero - 1);
  else
    return 1;
}

int main()
{
  int numero= 0;
  int fat;
  printf("Digite um numero: ");
  scanf("%d", &numero);
  fat = fator(numero);
  printf("O fatorial de %d é %d\n",numero, fat);
  return 0;
}
```

11. micro11. Exemplo com chamada de função.

    Lua:

Listagem 2.45: micro11.lua

```lua
function verifica(n)
   local res
   if n > 0 then
      res = 1
   else if n < 0 then
      res = -1
   else
      res = 0
      end
   end
return res
end

function main()
   local numero
```

```
16    local x
17    print("Digite um número: ")
18    numero = io.read("*n")
19    x = verifica(numero)
20    if x == 1 then
21       print("Número positivo")
22    else if x == 0 then
23       print("Zero")
24    else
25       print("Número negativo")
26       end
27    end
28 end
29
30 main()
```

C:

Listagem 2.46: micro11.c

```
1  #include<stdio.h>
2
3  int verifica(int n)
4  {
5    int res;
6    if (n > 0)
7      {
8        res = 1;
9      }
10   else if (n < 0)
11     {
12       res = -1;
13     }
14   else
15     {
16       res = 0;
17     }
18   return res;
19 }
20
21 int main()
22 {
23   int numero;
24   int x;
25
26   printf("Digite um número: ");
27   scanf("%d", &numero);
28   x = verifica(numero);
29
30   if (x == 1)
31     {
32       printf("Número positivo\n");
33     }
34   else if (x == 0)
35     {
36       printf("Zero\n");
37     }
38   else
39     {
40       printf("Número negativo\n");
```

```
41      }
42    return 0;
43 }
```

2.2 Análise dos Códigos Gerados em Assembly

É importante fazermos um breve comentário sobre alguns comandos presentes no código gerado em assembly: > As instruções que começam com um .cfi são "instruções de quadro de chamadas"com significado para o desenrolamento de pilha e o tratamento de exceções. Outras instruções que começam com um ponto são as diretivas de montagem que anotam como o arquivo deve ser montado.

> As instruções como main: , .LFB0: , LFE0: são “labels”, e não contém instruções. Dessa forma, vamos focar nos resultados mais importantes.

> Todos os valores que apresentam rbp ou eax são registros. Eles são pedaços de memória presentes na CPU. Registros que começam com um r são 64 bits, e os registros que começam com e têm 32 bits de largura. Os sufixos q nas instruções em si referem-se a "quad-words"indicando que é uma instrução de 64 bits. Os sufixos l indicam instruções de 32 bits.

> Parênteses indicam um local de memória, enquanto o número na frente dos parênteses indica um deslocamento dessa localização de memória.

> O símbolo ’por cento’ indica um nome do registro, enquanto que o ’cifrão’ indica um valor literal.

> As diretrizes apresentadas no código serão tratadas na secão de Apêndice, juntamente com a explicação genérica dos comandos encontrados.

2.2.1 Nano Programas

Esta subseção apresenta o arquivo assembly do código, em sequência a explicação de alguns termos.

1. nano01. Módulo mínimo que caracteriza um programa

Listagem 2.47: nano01.s

```
1   .file "nano01.c"
2   .text
3   .globl  main
4   .type main, @function
5 main:
6 .LFB0:
7   .cfi_startproc
8   pushq %rbp
9   .cfi_def_cfa_offset 16
10  .cfi_offset 6, -16
```

```
11   movq  %rsp, %rbp
12   .cfi_def_cfa_register 6
13   movl  $0, %eax
14   popq  %rbp
15   .cfi_def_cfa 7, 8
16   ret
17   .cfi_endproc
18 .LFE0:
19   .size main, .-main
20   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
21   .section  .note.GNU-stack,"",@progbits
```

main: pushq %rbp salva rbp na pilha
movq %rsp, %rbp guarda o valor de rsp em rbp
movl $0, %eax guarda o valor de 0 em eax
popq %rbp restaura rbp com o valor salvo na pilha
ret indica o retorno desta função

2. nano02. Declaração de uma variável

Listagem 2.48: nano02.s

```
1   .file "nano02.c"
2   .text
3   .globl  main
4   .type main, @function
5 main:
6 .LFB0:
7   .cfi_startproc
8   pushq %rbp
9   .cfi_def_cfa_offset 16
10  .cfi_offset 6, -16
11  movq  %rsp, %rbp
12  .cfi_def_cfa_register 6
13  movl  $0, %eax
14  popq  %rbp
15  .cfi_def_cfa 7, 8
16  ret
17  .cfi_endproc
18 .LFE0:
19  .size main, .-main
20  .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
21  .section  .note.GNU-stack,"",@progbits
```

Apesar da declaração da variável local n, nenhuma mudança foi notada no código gerado.

3. nano03. Atribuição de um valor à uma variável

Listagem 2.49: nanos.s

```
1   .file "nano03.c"
2   .text
3   .globl  main
4   .type main, @function
5 main:
6 .LFB0:
```

```
7   .cfi_startproc
8   pushq %rbp
9   .cfi_def_cfa_offset 16
10  .cfi_offset 6, -16
11  movq  %rsp, %rbp
12  .cfi_def_cfa_register 6
13  movl  $1, -4(%rbp)
14  movl  $0, %eax
15  popq  %rbp
16  .cfi_def_cfa 7, 8
17  ret
18  .cfi_endproc
19 .LFE0:
20  .size main, .-main
21  .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
22  .section  .note.GNU-stack,"",@progbits
```

main: pushq %rbp
movq %rsp, %rbp
movl $1, -4(%rbp) faz a atribuição do valor 1 para a posição de memória rbp com deslocamento de memória de 4
movl $0, %eax
popq %rbp
ret

4. nano04. Atribuição de uma soma de valores à uma variável

Listagem 2.50: nano04.s

```
1   .file "nano04.c"
2   .text
3   .globl  main
4   .type main, @function
5  main:
6  .LFB0:
7   .cfi_startproc
8   pushq %rbp
9   .cfi_def_cfa_offset 16
10  .cfi_offset 6, -16
11  movq  %rsp, %rbp
12  .cfi_def_cfa_register 6
13  movl  $3, -4(%rbp)
14  movl  $0, %eax
15  popq  %rbp
16  .cfi_def_cfa 7, 8
17  ret
18  .cfi_endproc
19 .LFE0:
20  .size main, .-main
21  .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
22  .section  .note.GNU-stack,"",@progbits
```

main: pushq %rbp
movq %rsp, %rbp
movl $3, -4(%rbp) faz a atribuição do valor 3 para a posição de memória rbp com deslocamento de memória de 4. Dessa forma faz-se a atribuição n = 1 + 2

movl $0, %eax
popq %rbp
ret

5. nano05. Comando de impressão.

Listagem 2.51: nano05.s

```
1   .file "nano05.c"
2   .section  .rodata
3 .LC0:
4   .string "%d"
5   .text
6   .globl  main
7   .type main, @function
8 main:
9 .LFB0:
10  .cfi_startproc
11  pushq %rbp
12  .cfi_def_cfa_offset 16
13  .cfi_offset 6, -16
14  movq  %rsp, %rbp
15  .cfi_def_cfa_register 6
16  subq  $16, %rsp
17  movl  $2, -4(%rbp)
18  movl  -4(%rbp), %eax
19  movl  %eax, %esi
20  movl  $.LC0, %edi
21  movl  $0, %eax
22  call  printf
23  movl  $0, %eax
24  leave
25  .cfi_def_cfa 7, 8
26  ret
27  .cfi_endproc
28 .LFE0:
29  .size main, .-main
30  .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
31  .section  .note.GNU-stack,"",@progbits
```

main: pushq %rbp
movq %rsp, %rbp
subq $16, %rsp Subtrai 16 de %rsp. Empilha-se o ponteiro movido (alocando espaço na pilha) quando uma função precisa reservar dados locais.
movl $2, -4(%rbp) faz a atribuição do valor 2 para a posição de memória rbp com deslocamento de memória de 4. Dessa forma faz-se a atribuição n = 2.
movl -4(%rbp), %eax
movl %eax, %esi
movl $.LC0, %edi
movl $0, %eax
call printf utilizando quando fazemos chamadas de funções, neste caso, a função printf.
movl $0, %eax
leave Libera as variáveis locais criadas pela instrução de entrada anterior, restaurando a pilha para sair do procedimento.
ret

6. nano06. Atribuição de uma subtração à uma variável

Listagem 2.52: nano06.s

```
1   .file "nano06.c"
2   .section  .rodata
3 .LC0:
4   .string "%d"
5   .text
6   .globl  main
7   .type main, @function
8 main:
9 .LFB0:
10  .cfi_startproc
11  pushq %rbp
12  .cfi_def_cfa_offset 16
13  .cfi_offset 6, -16
14  movq  %rsp, %rbp
15  .cfi_def_cfa_register 6
16  subq  $16, %rsp
17  movl  $-1, -4(%rbp)
18  movl  -4(%rbp), %eax
19  movl  %eax, %esi
20  movl  $.LC0, %edi
21  movl  $0, %eax
22  call  printf
23  movl  $0, %eax
24  leave
25  .cfi_def_cfa 7, 8
26  ret
27  .cfi_endproc
28 .LFE0:
29  .size main, .-main
30  .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
31  .section  .note.GNU-stack,"",@progbits
```

main: pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $-1, -4(%rbp) faz a atribuição do valor -1 para a posição de memória rbp com deslocamento de memória de 4. Dessa forma faz-se a atribuição n = 1- 2. movl -4(%rbp), %eax
movl %eax, %esi
movl $0, %eax
call printf
movl $0, %eax leave
ret

7. nano07. Inclusão de condicionalidade

Listagem 2.53: nano07.s

```
1   .file "nano07.c"
2   .section  .rodata
3 .LC0:
```

```
4   .string "%d"
5   .text
6   .globl  main
7   .type main, @function
8 main:
9 .LFB0:
10  .cfi_startproc
11  pushq %rbp
12  .cfi_def_cfa_offset 16
13  .cfi_offset 6, -16
14  movq  %rsp, %rbp
15  .cfi_def_cfa_register 6
16  subq  $16, %rsp
17  movl  $1, -4(%rbp)
18  cmpl  $1, -4(%rbp)
19  jne .L2
20  movl  -4(%rbp), %eax
21  movl  %eax, %esi
22  movl  $.LC0, %edi
23  movl  $0, %eax
24  call  printf
25 .L2:
26  movl  $0, %eax
27  leave
28  .cfi_def_cfa 7, 8
29  ret
30  .cfi_endproc
31 .LFE0:
32  .size main, .-main
33  .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
34  .section  .note.GNU-stack,"",@progbits
```

main: pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $1, -4(%rbp)
cmpl $1, -4(%rbp) Compara o valor 1 com o valor contido na posição de memória de %rbp, realiza a função if.
jne .L2 Faz com que a execução "pule" a intrução .L2 caso não for igual, "JNE – Jump Not Equal", prossegue a execução após o if.
movl 4(%rbp), %eax
movl %eax, %esi
movl $.LC0, %edi
movl $0, %eax
call printf

8. nano08. Inclusão de condicionalidade com 'else'

Listagem 2.54: nano08.s

```
1   .file "nano08.c"
2   .section  .rodata
3 .LC0:
4   .string "%d"
5   .text
6   .globl  main
```

```
7   .type main, @function
8 main:
9 .LFB0:
10   .cfi_startproc
11   pushq %rbp
12   .cfi_def_cfa_offset 16
13   .cfi_offset 6, -16
14   movq  %rsp, %rbp
15   .cfi_def_cfa_register 6
16   subq  $16, %rsp
17   movl  $1, -4(%rbp)
18   cmpl  $1, -4(%rbp)
19   jne .L2
20   movl  -4(%rbp), %eax
21   movl  %eax, %esi
22   movl  $.LC0, %edi
23   movl  $0, %eax
24   call  printf
25   jmp .L3
26 .L2:
27   movl  $48, %edi
28   call  putchar
29 .L3:
30   movl  $0, %eax
31   leave
32   .cfi_def_cfa 7, 8
33   ret
34   .cfi_endproc
35 .LFE0:
36   .size main, .-main
37   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
38   .section  .note.GNU-stack,"",@progbits
```

main: pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $1, -4(%rbp)
cmpl $1, -4(%rbp)
jne .L2
movl -4(%rbp), %eax
movl %eax, %esi
movl $.LC0, %edi
movl $0, %eax
call printf
jmp .L3 transfere incondicionalmente o controle para L3 para execução de seus comandos. “jmp – inconditional jump”.

9. nano09. Atribuição de duas operações aritméticas sobre inteiros a uma variável

Listagem 2.55: nano09.s

```
1   .file "nano09.c"
2   .section  .rodata
3 .LC0:
4   .string "%d"
5   .text
```

```
6   .globl  main
7   .type main, @function
8 main:
9 .LFB0:
10  .cfi_startproc
11  pushq %rbp
12  .cfi_def_cfa_offset 16
13  .cfi_offset 6, -16
14  movq  %rsp, %rbp
15  .cfi_def_cfa_register 6
16  subq  $16, %rsp
17  movl  $1, -4(%rbp)
18  cmpl  $1, -4(%rbp)
19  jne .L2
20  movl  -4(%rbp), %eax
21  movl  %eax, %esi
22  movl  $.LC0, %edi
23  movl  $0, %eax
24  call  printf
25  jmp .L3
26 .L2:
27  movl  $48, %edi
28  call  putchar
29 .L3:
30  movl  $0, %eax
31  leave
32  .cfi_def_cfa 7, 8
33  ret
34  .cfi_endproc
35 .LFE0:
36  .size main, .-main
37  .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
38  .section  .note.GNU-stack,"",@progbits
```

main: pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $1, -4(%rbp) atribuição do valor a n;
cmpl $1, -4(%rbp) comparação do if;
jne .L2 pula para a instrução L2;
movl -4(%rbp), %eax
movl %eax, %esi
movl $.LC0, %edi
movl $0, %eax
call printf
jmp .L3 pula para instrução L3.

10. nano10. Atribuição de duas variáveis inteiras

Listagem 2.56: nano10.s

```
1   .file "nano10.c"
2   .section  .rodata
3 .LC0:
4   .string "%d"
5   .text
```

```
6   .globl  main
7   .type main, @function
8 main:
9 .LFB0:
10  .cfi_startproc
11  pushq %rbp
12  .cfi_def_cfa_offset 16
13  .cfi_offset 6, -16
14  movq  %rsp, %rbp
15  .cfi_def_cfa_register 6
16  subq  $16, %rsp
17  movl  $1, -8(%rbp)
18  movl  $2, -4(%rbp)
19  movl  -8(%rbp), %eax
20  cmpl  -4(%rbp), %eax
21  jne .L2
22  movl  -8(%rbp), %eax
23  movl  %eax, %esi
24  movl  $.LC0, %edi
25  movl  $0, %eax
26  call  printf
27  jmp .L3
28 .L2:
29  movl  $48, %edi
30  call  putchar
31 .L3:
32  movl  $0, %eax
33  leave
34  .cfi_def_cfa 7, 8
35  ret
36  .cfi_endproc
37 .LFE0:
38  .size main, .-main
39  .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
40  .section  .note.GNU-stack,"",@progbits
```

main:

pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $1, -8(%rbp) n =1
movl $2, -4(%rbp) m =2
movl -8(%rbp), %eax move n para eax
cmpl -4(%rbp), %eax compara m com eax (n).
jne .L2 pula para instrução L2.
movl -8(%rbp), %eax
movl %eax, %esi
movl $.LC0, %edi
movl $0, %eax
call printf
jmp .L3 pula para instrução L3.

11. nano11. Comando de repetição 'while'

Listagem 2.57: nano11.s

```
1   .file "nano11.c"
2   .section  .rodata
3 .LC0:
4   .string "%d "
5   .text
6   .globl  main
7   .type main, @function
8 main:
9 .LFB0:
10  .cfi_startproc
11  pushq %rbp
12  .cfi_def_cfa_offset 16
13  .cfi_offset 6, -16
14  movq  %rsp, %rbp
15  .cfi_def_cfa_register 6
16  subq  $16, %rsp
17  movl  $1, -12(%rbp)
18  movl  $2, -8(%rbp)
19  movl  $5, -4(%rbp)
20  jmp .L2
21 .L3:
22  movl  -8(%rbp), %eax
23  addl  %eax, -12(%rbp)
24  movl  -12(%rbp), %eax
25  movl  %eax, %esi
26  movl  $.LC0, %edi
27  movl  $0, %eax
28  call  printf
29 .L2:
30  movl  -4(%rbp), %eax
31  cmpl  -12(%rbp), %eax
32  jg  .L3
33  movl  $0, %eax
34  leave
35  .cfi_def_cfa 7, 8
36  ret
37  .cfi_endproc
38 .LFE0:
39  .size main, .-main
40  .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
41  .section  .note.GNU-stack,"",@progbits
```

main: pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $1, -12(%rbp)
movl $2, -8(%rbp)
movl $5, -4(%rbp)
jmp .L2
.L3:
movl -8(%rbp), %eax
addl %eax, -12(%rbp) Soma o conteúdo de eax com de rbp, e substitui o valor original de rbp, ambos os operandos são binários
movl -12(%rbp), %eax
movl %eax, %esi
movl $.LC0, %edi

movl $0, %eax
call printf
.L2:
movl -4(%rbp), %eax
cmpl -12(%rbp), %eax
jg .L3 "jg – jump greater", representa o sinal de comparação maior (>), e passa para a intrução L3.
movl $0, %eax
leave
ret

12. nano12. Junção de Condicionalidade com 'while'

Listagem 2.58: nano12.s

```
1   .file "nano12.c"
2   .section  .rodata
3 .LC0:
4   .string "%d "
5 .LC1:
6   .string "0 "
7   .text
8   .globl  main
9   .type main, @function
10 main:
11 .LFB0:
12   .cfi_startproc
13   pushq %rbp
14   .cfi_def_cfa_offset 16
15   .cfi_offset 6, -16
16   movq  %rsp, %rbp
17   .cfi_def_cfa_register 6
18   subq  $16, %rsp
19   movl  $1, -8(%rbp)
20   movl  $2, -4(%rbp)
21   movl  $5, -12(%rbp)
22   jmp .L2
23 .L5:
24   movl  -8(%rbp), %eax
25   cmpl  -4(%rbp), %eax
26   jne .L3
27   movl  -8(%rbp), %eax
28   movl  %eax, %esi
29   movl  $.LC0, %edi
30   movl  $0, %eax
31   call  printf
32   jmp .L4
33 .L3:
34   movl  $.LC1, %edi
35   movl  $0, %eax
36   call  printf
37 .L4:
38   subl  $1, -12(%rbp)
39 .L2:
40   movl  -12(%rbp), %eax
41   cmpl  -8(%rbp), %eax
42   jg  .L5
```

```
43   movl  $0, %eax
44   leave
45   .cfi_def_cfa 7, 8
46   ret
47   .cfi_endproc
48 .LFE0:
49   .size main, .-main
50   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
51   .section  .note.GNU-stack,"",@progbits
```

Basta seguir a lógica e os comandos já apresentados acima para se entender o funcionamento do programa.

2.2.2 Micro Programas

1. micro01. Conversão de Celsius para Farenheit.

Listagem 2.59: micro01.s

```
1   .file "micro01.c"
2   .section  .rodata
3   .align 8
4 .LC0:
5   .string "Tabela de convers\303\243o: Celsius -> Fahrenheit"
6   .align 8
7 .LC1:
8   .string "Digite a temperatura em Celsius: "
9 .LC2:
10  .string "%f"
11 .LC6:
12  .string "A nova temperatura \303\251: %f F \n"
13  .text
14  .globl  main
15  .type main, @function
16 main:
17 .LFB0:
18  .cfi_startproc
19  pushq %rbp
20  .cfi_def_cfa_offset 16
21  .cfi_offset 6, -16
22  movq  %rsp, %rbp
23  .cfi_def_cfa_register 6
24  subq  $16, %rsp
25  movq  %fs:40, %rax
26  movq  %rax, -8(%rbp)
27  xorl  %eax, %eax
28  movl  $.LC0, %edi
29  call  puts
30  movl  $.LC1, %edi
31  movl  $0, %eax
32  call  printf
33  leaq  -16(%rbp), %rax
34  movq  %rax, %rsi
35  movl  $.LC2, %edi
36  movl  $0, %eax
37  call  __isoc99_scanf
```

```
38   movss -16(%rbp), %xmm1
39   movss .LC3(%rip), %xmm0
40   mulss %xmm1, %xmm0
41   movss .LC4(%rip), %xmm1
42   addss %xmm1, %xmm0
43   movss .LC5(%rip), %xmm1
44   divss %xmm1, %xmm0
45   movss %xmm0, -12(%rbp)
46   cvtss2sd  -12(%rbp), %xmm0
47   movl  $.LC6, %edi
48   movl  $1, %eax
49   call  printf
50   movl  $0, %eax
51   movq  -8(%rbp), %rdx
52   xorq  %fs:40, %rdx
53   je  .L3
54   call  __stack_chk_fail
55 .L3:
56   leave
57   .cfi_def_cfa 7, 8
58   ret
59   .cfi_endproc
60 .LFE0:
61   .size main, .-main
62   .section  .rodata
63   .align 4
64 .LC3:
65   .long 1091567616
66   .align 4
67 .LC4:
68   .long 1126170624
69   .align 4
70 .LC5:
71   .long 1084227584
72   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
73   .section  .note.GNU-stack,"",@progbits
```

Neste exemplo aparece o pós-fixo ‘s’, que representa signal bit, ou seja, bits de sinal, como por exemplo em no comando movss , (movs + s).

As strings digitadas no codigo são representadas após o sufixo .string, entre aspas, como mostrado neste código e se repetindo nos códigos posteriores. .LC0:
.string "Tabela de convers303243o: Celsius -> Fahrenheit"
.align 8
.LC1:
.string "Digite a temperatura em Celsius: "
.LC2:
.string "%f"
.LC6:
.string "A nova temperatura 303251: %f F "
.text
.globl main
.type main, @function
main: pushq %rbp
movq %rsp, %rbp
subq $16, %rsp

movq %fs:40, %rax
movq %rax, -8(%rbp) este comando executa um OR exclusivo bit a bit dos operandos e retorna o resultado no destino.
xorl %eax, %eax
movl $.LC0, %edi
call puts chama a função puts
movl $.LC1, %edi
movl $0, %eax
call printf
leaq -16(%rbp), %rax Load effective address, esse comando transfere o endereço de rbp para o destino, no caso rax.
movq %rax, %rsi
movl $.LC2, %edi
movl $0, %eax
call scanf
movss -16(%rbp), %xmm1 movs signifca 'move string', tem a mesma função do mov só que para strings, byte ou palavra
movss .LC3(%rip), %xmm0
mulss %xmm1, %xmm0 faz a multiplicação sinalizada do acumulador com a fonte, no caso, xmm1 com xmm0.
movss .LC4(%rip), %xmm1
addss %xmm1, %xmm0
movss .LC5(%rip), %xmm1
divss %xmm1, %xmm0 faz a divisão sinalizada binária dos dois parâmetros passados.
movss %xmm0, -12(%rbp)
cvtss2sd -12(%rbp), %xmm0
movl $.LC6, %edi
movl $1, %eax
call printf
movl $0, %eax
movq -8(%rbp), %rdx
xorq %fs:40, %rdx
je .L3

.L3:
leave
ret

2. micro02. Ler dois inteiros e ver qual é maior.

Listagem 2.60: micro02.s

```
1   .file "micro02.c"
2   .section  .rodata
3 .LC0:
4   .string "Digite o primeiro n\303\272mero: "
5 .LC1:
6   .string "%d"
7 .LC2:
8   .string "Digite o segundo n\303\272mero: "
9   .align 8
10 .LC3:
```

```
11    .string "O primeiro n\303\272mero %d \303\251 maior que o segundo %
          d"
12    .align 8
13  .LC4:
14    .string "O segundo n\303\272mero %d \303\251 maior que o primeiro %
          d"
15    .text
16    .globl  main
17    .type main, @function
18  main:
19  .LFB0:
20    .cfi_startproc
21    pushq %rbp
22    .cfi_def_cfa_offset 16
23    .cfi_offset 6, -16
24    movq  %rsp, %rbp
25    .cfi_def_cfa_register 6
26    subq  $16, %rsp
27    movq  %fs:40, %rax
28    movq  %rax, -8(%rbp)
29    xorl  %eax, %eax
30    movl  $.LC0, %edi
31    movl  $0, %eax
32    call  printf
33    leaq  -16(%rbp), %rax
34    movq  %rax, %rsi
35    movl  $.LC1, %edi
36    movl  $0, %eax
37    call  __isoc99_scanf
38    movl  $.LC2, %edi
39    movl  $0, %eax
40    call  printf
41    leaq  -12(%rbp), %rax
42    movq  %rax, %rsi
43    movl  $.LC1, %edi
44    movl  $0, %eax
45    call  __isoc99_scanf
46    movl  -16(%rbp), %edx
47    movl  -12(%rbp), %eax
48    cmpl  %eax, %edx
49    jle .L2
50    movl  -12(%rbp), %edx
51    movl  -16(%rbp), %eax
52    movl  %eax, %esi
53    movl  $.LC3, %edi
54    movl  $0, %eax
55    call  printf
56    jmp .L3
57  .L2:
58    movl  -16(%rbp), %edx
59    movl  -12(%rbp), %eax
60    movl  %eax, %esi
61    movl  $.LC4, %edi
62    movl  $0, %eax
63    call  printf
64  .L3:
65    movl  $0, %eax
66    movq  -8(%rbp), %rcx
67    xorq  %fs:40, %rcx
```

```
68   je  .L5
69   call  __stack_chk_fail
70 .L5:
71   leave
72   .cfi_def_cfa 7, 8
73   ret
74   .cfi_endproc
75 .LFE0:
76   .size main, .-main
77   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
78   .section  .note.GNU-stack,"",@progbits
```

3. micro03. Lê um número e verifica se está entre 100 e 200.

Listagem 2.61: micro03.s

```
1   .file "micro03.c"
2   .section  .rodata
3 .LC0:
4   .string "Digite um n\303\272mero: "
5 .LC1:
6   .string "%d"
7   .align 8
8 .LC2:
9   .string "O n\303\272mero est\303\241 no intervalo entre 100 e 200"
10  .align 8
11 .LC3:
12  .string "O n\303\272mero n\303\243o est\303\241 no intervalo entre
        100 e 200"
13  .text
14  .globl  main
15  .type main, @function
16 main:
17 .LFB0:
18  .cfi_startproc
19  pushq %rbp
20  .cfi_def_cfa_offset 16
21  .cfi_offset 6, -16
22  movq  %rsp, %rbp
23  .cfi_def_cfa_register 6
24  subq  $16, %rsp
25  movq  %fs:40, %rax
26  movq  %rax, -8(%rbp)
27  xorl  %eax, %eax
28  movl  $.LC0, %edi
29  movl  $0, %eax
30  call  printf
31  leaq  -12(%rbp), %rax
32  movq  %rax, %rsi
33  movl  $.LC1, %edi
34  movl  $0, %eax
35  call  __isoc99_scanf
36  movl  -12(%rbp), %eax
37  cmpl  $99, %eax
38  jle .L2
39  movl  -12(%rbp), %eax
40  cmpl  $200, %eax
41  jg  .L3
42  movl  $.LC2, %edi
```

```
43   call  puts
44   jmp .L5
45 .L3:
46   movl  $.LC3, %edi
47   call  puts
48   jmp .L5
49 .L2:
50   movl  $.LC3, %edi
51   call  puts
52 .L5:
53   movl  $0, %eax
54   movq  -8(%rbp), %rdx
55   xorq  %fs:40, %rdx
56   je  .L7
57   call  __stack_chk_fail
58 .L7:
59   leave
60   .cfi_def_cfa 7, 8
61   ret
62   .cfi_endproc
63 .LFE0:
64   .size main, .-main
65   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
66   .section  .note.GNU-stack,"",@progbits
```

Neste caso não há novos comandos comparados com os anteriores já apresentados, basta então seguir as explicações apresentadas para a compreensão deste código.

4. micro04. Lê números e fala quais estão entre 10 e 150.

Listagem 2.62: micro04.s

```
1   .file "micro04.c"
2   .section  .rodata
3 .LC0:
4   .string "Digite um n\303\272mero: "
5 .LC1:
6   .string "%d"
7   .align 8
8 .LC2:
9   .string "Ao total, foram digitados %d n\303\272meros no intervalo
       entre 10 e 150\n"
10  .text
11  .globl  main
12  .type main, @function
13 main:
14 .LFB0:
15  .cfi_startproc
16  pushq %rbp
17  .cfi_def_cfa_offset 16
18  .cfi_offset 6, -16
19  movq  %rsp, %rbp
20  .cfi_def_cfa_register 6
21  subq  $32, %rsp
22  movq  %fs:40, %rax
23  movq  %rax, -8(%rbp)
24  xorl  %eax, %eax
25  movl  $0, -12(%rbp)
```

```
26   movl  $0, -16(%rbp)
27   jmp .L2
28 .L4:
29   movl  $.LC0, %edi
30   movl  $0, %eax
31   call  printf
32   leaq  -20(%rbp), %rax
33   movq  %rax, %rsi
34   movl  $.LC1, %edi
35   movl  $0, %eax
36   call  __isoc99_scanf
37   movl  -20(%rbp), %eax
38   cmpl  $9, %eax
39   jle .L3
40   movl  -20(%rbp), %eax
41   cmpl  $150, %eax
42   jg  .L3
43   addl  $1, -12(%rbp)
44 .L3:
45   addl  $1, -16(%rbp)
46 .L2:
47   cmpl  $4, -16(%rbp)
48   jle .L4
49   movl  -12(%rbp), %eax
50   movl  %eax, %esi
51   movl  $.LC2, %edi
52   movl  $0, %eax
53   call  printf
54   movl  $0, %eax
55   movq  -8(%rbp), %rdx
56   xorq  %fs:40, %rdx
57   je  .L6
58   call  __stack_chk_fail
59 .L6:
60   leave
61   .cfi_def_cfa 7, 8
62   ret
63   .cfi_endproc
64 .LFE0:
65   .size main, .-main
66   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
67   .section  .note.GNU-stack,"",@progbits
```

Neste caso não há novos comandos comparados com os anteriores já apresentados, basta então seguir as explicações apresentadas para a compreensão deste código.

5. micro05. Lê strings e caracteres.

Listagem 2.63: micro05.s

```
1   .file "micro05.c"
2   .section  .rodata
3 .LC0:
4   .string "Digite o nome: "
5 .LC1:
6   .string "%s"
7 .LC2:
8   .string "1 - Homem ou 2 - Mulher: "
```

```
.LC3:
  .string "%d"
  .align 8
.LC4:
  .string "Sexo s\303\263 pode ser Homem(1) ou Mulher(2)!"
.LC5:
  .string "Foram inseridos %d Homens.\n"
.LC6:
  .string "Foram inseridos %d Mulheres.\n"
  .text
  .globl  main
  .type main, @function
main:
.LFB0:
  .cfi_startproc
  pushq %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset 6, -16
  movq  %rsp, %rbp
  .cfi_def_cfa_register 6
  subq  $80, %rsp
  movq  %fs:40, %rax
  movq  %rax, -8(%rbp)
  xorl  %eax, %eax
  movl  $0, -72(%rbp)
  movl  $0, -68(%rbp)
  movl  $0, -76(%rbp)
  jmp .L2
.L7:
  movl  $.LC0, %edi
  movl  $0, %eax
  call  printf
  leaq  -64(%rbp), %rax
  movq  %rax, %rsi
  movl  $.LC1, %edi
  movl  $0, %eax
  call  __isoc99_scanf
  movl  $.LC2, %edi
  movl  $0, %eax
  call  printf
  leaq  -80(%rbp), %rax
  movq  %rax, %rsi
  movl  $.LC3, %edi
  movl  $0, %eax
  call  __isoc99_scanf
  movl  -80(%rbp), %eax
  cmpl  $1, %eax
  je  .L4
  cmpl  $2, %eax
  je  .L5
  jmp .L10
.L4:
  addl  $1, -72(%rbp)
  jmp .L6
.L5:
  addl  $1, -68(%rbp)
  jmp .L6
.L10:
  movl  $.LC4, %edi
```

```
68   call  puts
69   nop
70 .L6:
71   addl  $1, -76(%rbp)
72 .L2:
73   cmpl  $4, -76(%rbp)
74   jle .L7
75   movl  -72(%rbp), %eax
76   movl  %eax, %esi
77   movl  $.LC5, %edi
78   movl  $0, %eax
79   call  printf
80   movl  -68(%rbp), %eax
81   movl  %eax, %esi
82   movl  $.LC6, %edi
83   movl  $0, %eax
84   call  printf
85   movl  $0, %eax
86   movq  -8(%rbp), %rdx
87   xorq  %fs:40, %rdx
88   je  .L9
89   call  __stack_chk_fail
90 .L9:
91   leave
92   .cfi_def_cfa 7, 8
93   ret
94   .cfi_endproc
95 .LFE0:
96   .size main, .-main
97   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
98   .section  .note.GNU-stack,"",@progbits
```

Neste caso não há novos comandos comparados com os anteriores já apresentados, basta então seguir as explicações apresentadas para a compreensão deste código.

6. micro06. Escreve um número lido por extenso.

Listagem 2.64: micro06.s

```
1   .file "micro06.c"
2   .section  .rodata
3 .LC0:
4   .string "Digite um n\303\272mero de 1 a 5: "
5 .LC1:
6   .string "%d"
7 .LC2:
8   .string "Um"
9 .LC3:
10  .string "Dois"
11 .LC4:
12  .string "Tr\303\252s"
13 .LC5:
14  .string "Quatro"
15 .LC6:
16  .string "Cinco"
17 .LC7:
18  .string "N\303\272mero inv\303\241lido!!!"
19  .text
```

```
20   .globl  main
21   .type main, @function
22 main:
23 .LFB0:
24   .cfi_startproc
25   pushq %rbp
26   .cfi_def_cfa_offset 16
27   .cfi_offset 6, -16
28   movq  %rsp, %rbp
29   .cfi_def_cfa_register 6
30   subq  $16, %rsp
31   movq  %fs:40, %rax
32   movq  %rax, -8(%rbp)
33   xorl  %eax, %eax
34   movl  $.LC0, %edi
35   movl  $0, %eax
36   call  printf
37   leaq  -12(%rbp), %rax
38   movq  %rax, %rsi
39   movl  $.LC1, %edi
40   movl  $0, %eax
41   call  __isoc99_scanf
42   movl  -12(%rbp), %eax
43   cmpl  $5, %eax
44   ja  .L2
45   movl  %eax, %eax
46   movq  .L4(,%rax,8), %rax
47   jmp *%rax
48   .section  .rodata
49   .align 8
50   .align 4
51 .L4:
52   .quad .L2
53   .quad .L3
54   .quad .L5
55   .quad .L6
56   .quad .L7
57   .quad .L8
58   .text
59 .L3:
60   movl  $.LC2, %edi
61   call  puts
62   jmp .L9
63 .L5:
64   movl  $.LC3, %edi
65   call  puts
66   jmp .L9
67 .L6:
68   movl  $.LC4, %edi
69   call  puts
70   jmp .L9
71 .L7:
72   movl  $.LC5, %edi
73   call  puts
74   jmp .L9
75 .L8:
76   movl  $.LC6, %edi
77   call  puts
78   jmp .L9
```

```
79 .L2:
80   movl  $.LC7, %edi
81   call  puts
82   nop
83 .L9:
84   movl  $0, %eax
85   movq  -8(%rbp), %rdx
86   xorq  %fs:40, %rdx
87   je  .L11
88   call  __stack_chk_fail
89 .L11:
90   leave
91   .cfi_def_cfa 7, 8
92   ret
93   .cfi_endproc
94 .LFE0:
95   .size main, .-main
96   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
97   .section  .note.GNU-stack,"",@progbits
```

Vamos fazer uma análise desta diretiz que aqui ocorre :
.L4:
.quad .L2
.quad .L3
.quad .L5
.quad .L6
.quad .L7
.quad .L8
.text

A diretiz .quad trunca os valores de expressão especificados na lista separada por vírgula para valores de 64 bits e reune os valores em locais sucessivos. Estes valores de expressão podem ser relocáveis.

7. micro07. Verifica se os números são >0 , <0 , ou =0.

Listagem 2.65: micro07.s

```
1   .file "micro07.c"
2   .section  .rodata
3 .LC0:
4   .string "Digite um n\303\272mero: "
5 .LC1:
6   .string "%d"
7 .LC2:
8   .string "Positivo"
9 .LC3:
10  .string "O n\303\272mero \303\251 igual a 0"
11 .LC4:
12  .string "Negativo"
13 .LC5:
14  .string "Deseja finalizar? (sim/nao) "
15 .LC6:
16  .string "%s"
17 .LC7:
```

```
18   .string "sim"
19   .text
20   .globl  main
21   .type main, @function
22 main:
23 .LFB0:
24   .cfi_startproc
25   pushq %rbp
26   .cfi_def_cfa_offset 16
27   .cfi_offset 6, -16
28   movq  %rsp, %rbp
29   .cfi_def_cfa_register 6
30   subq  $32, %rsp
31   movq  %fs:40, %rax
32   movq  %rax, -8(%rbp)
33   xorl  %eax, %eax
34   movl  $1, -20(%rbp)
35 .L6:
36   movl  $.LC0, %edi
37   movl  $0, %eax
38   call  printf
39   leaq  -24(%rbp), %rax
40   movq  %rax, %rsi
41   movl  $.LC1, %edi
42   movl  $0, %eax
43   call  __isoc99_scanf
44   movl  -24(%rbp), %eax
45   testl %eax, %eax
46   jle .L2
47   movl  $.LC2, %edi
48   call  puts
49   jmp .L3
50 .L2:
51   movl  -24(%rbp), %eax
52   testl %eax, %eax
53   jne .L4
54   movl  $.LC3, %edi
55   call  puts
56   jmp .L3
57 .L4:
58   movl  $.LC4, %edi
59   call  puts
60 .L3:
61   movl  $.LC5, %edi
62   movl  $0, %eax
63   call  printf
64   leaq  -16(%rbp), %rax
65   movq  %rax, %rsi
66   movl  $.LC6, %edi
67   movl  $0, %eax
68   call  __isoc99_scanf
69   leaq  -16(%rbp), %rax
70   movl  $.LC7, %esi
71   movq  %rax, %rdi
72   call  strcmp
73   testl %eax, %eax
74   jne .L5
75   movl  $0, -20(%rbp)
76 .L5:
```

```
77   cmpl  $1, -20(%rbp)
78   je  .L6
79   movl  $0, %eax
80   movq  -8(%rbp), %rdx
81   xorq  %fs:40, %rdx
82   je  .L8
83   call  __stack_chk_fail
84 .L8:
85   leave
86   .cfi_def_cfa 7, 8
87   ret
88   .cfi_endproc
89 .LFE0:
90   .size main, .-main
91   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
92   .section  .note.GNU-stack,"",@progbits
```

O que aparece de novo neste código é o comando test; testl %eax, %eax Executa um AND lógico dos dois operandos atualizando o registro de sinalizadores sem salvar o resultado.

8. micro08. Verifica se um número é maior ou menor que 10.

Listagem 2.66: micro08.s

```
1   .file "micro08.c"
2   .section  .rodata
3 .LC0:
4   .string "Digite um n\303\272mero: "
5 .LC1:
6   .string "%d"
7 .LC2:
8   .string "O numero %d \303\251 maior que 10 \n"
9 .LC3:
10  .string "O numero %d \303\251 menor que 10 \n"
11  .text
12  .globl  main
13  .type main, @function
14 main:
15 .LFB0:
16  .cfi_startproc
17  pushq %rbp
18  .cfi_def_cfa_offset 16
19  .cfi_offset 6, -16
20  movq  %rsp, %rbp
21  .cfi_def_cfa_register 6
22  subq  $16, %rsp
23  movq  %fs:40, %rax
24  movq  %rax, -8(%rbp)
25  xorl  %eax, %eax
26  movl  $1, -12(%rbp)
27 .L4:
28  movl  $.LC0, %edi
29  movl  $0, %eax
30  call  printf
31  leaq  -12(%rbp), %rax
32  movq  %rax, %rsi
33  movl  $.LC1, %edi
34  movl  $0, %eax
```

```
35   call  __isoc99_scanf
36   movl  -12(%rbp), %eax
37   cmpl  $10, %eax
38   jle .L2
39   movl  -12(%rbp), %eax
40   movl  %eax, %esi
41   movl  $.LC2, %edi
42   movl  $0, %eax
43   call  printf
44   jmp .L3
45 .L2:
46   movl  -12(%rbp), %eax
47   movl  %eax, %esi
48   movl  $.LC3, %edi
49   movl  $0, %eax
50   call  printf
51 .L3:
52   movl  -12(%rbp), %eax
53   testl %eax, %eax
54   jne .L4
55   movl  $0, %eax
56   movq  -8(%rbp), %rdx
57   xorq  %fs:40, %rdx
58   je  .L6
59   call  __stack_chk_fail
60 .L6:
61   leave
62   .cfi_def_cfa 7, 8
63   ret
64   .cfi_endproc
65 .LFE0:
66   .size main, .-main
67   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
68   .section  .note.GNU-stack,"",@progbits
```

Neste caso não há novos comandos comparados com os anteriores já apresentados, basta então seguir as explicações apresentadas para a compreensão deste código.

9. micro09. Calcula preços.

Listagem 2.67: micro09.s

```
1   .file "micro09.c"
2   .section  .rodata
3 .LC0:
4   .string "Digite o pre\303\247o: "
5 .LC1:
6   .string "%f"
7 .LC2:
8   .string "Digite a venda: "
9 .LC10:
10  .string "O novo pre\303\247o \303\251: %f"
11  .text
12  .globl  main
13  .type main, @function
14 main:
15 .LFB0:
16  .cfi_startproc
```

```
17   pushq %rbp
18   .cfi_def_cfa_offset 16
19   .cfi_offset 6, -16
20   movq  %rsp, %rbp
21   .cfi_def_cfa_register 6
22   subq  $32, %rsp
23   movq  %fs:40, %rax
24   movq  %rax, -8(%rbp)
25   xorl  %eax, %eax
26   movl  $.LC0, %edi
27   movl  $0, %eax
28   call  printf
29   leaq  -20(%rbp), %rax
30   movq  %rax, %rsi
31   movl  $.LC1, %edi
32   movl  $0, %eax
33   call  __isoc99_scanf
34   movl  $.LC2, %edi
35   movl  $0, %eax
36   call  printf
37   leaq  -16(%rbp), %rax
38   movq  %rax, %rsi
39   movl  $.LC1, %edi
40   movl  $0, %eax
41   call  __isoc99_scanf
42   movss -16(%rbp), %xmm1
43   movss .LC3(%rip), %xmm0
44   ucomiss %xmm1, %xmm0
45   ja  .L2
46   movss -20(%rbp), %xmm1
47   movss .LC4(%rip), %xmm0
48   ucomiss %xmm1, %xmm0
49   jbe .L18
50 .L2:
51   movss -20(%rbp), %xmm0
52   cvtss2sd  %xmm0, %xmm1
53   movss -20(%rbp), %xmm0
54   cvtss2sd  %xmm0, %xmm0
55   movsd .LC5(%rip), %xmm2
56   mulsd %xmm2, %xmm0
57   addsd %xmm1, %xmm0
58   cvtsd2ss  %xmm0, %xmm3
59   movss %xmm3, -12(%rbp)
60   jmp .L5
61 .L18:
62   movss -16(%rbp), %xmm0
63   ucomiss .LC3(%rip), %xmm0
64   jb  .L6
65   movss -16(%rbp), %xmm1
66   movss .LC6(%rip), %xmm0
67   ucomiss %xmm1, %xmm0
68   ja  .L8
69 .L6:
70   movss -20(%rbp), %xmm0
71   ucomiss .LC4(%rip), %xmm0
72   jb  .L9
73   movss -20(%rbp), %xmm1
74   movss .LC7(%rip), %xmm0
75   ucomiss %xmm1, %xmm0
```

```
  jbe .L9
.L8:
  movss -20(%rbp), %xmm0
  cvtss2sd  %xmm0, %xmm1
  movss -20(%rbp), %xmm0
  cvtss2sd  %xmm0, %xmm0
  movsd .LC8(%rip), %xmm2
  mulsd %xmm2, %xmm0
  addsd %xmm1, %xmm0
  cvtsd2ss  %xmm0, %xmm4
  movss %xmm4, -12(%rbp)
  jmp .L5
.L9:
  movss -16(%rbp), %xmm0
  ucomiss .LC6(%rip), %xmm0
  jnb .L12
  movss -20(%rbp), %xmm0
  ucomiss .LC7(%rip), %xmm0
  jb  .L5
.L12:
  movss -20(%rbp), %xmm0
  cvtss2sd  %xmm0, %xmm0
  movss -20(%rbp), %xmm1
  cvtss2sd  %xmm1, %xmm1
  movsd .LC9(%rip), %xmm2
  mulsd %xmm2, %xmm1
  subsd %xmm1, %xmm0
  cvtsd2ss  %xmm0, %xmm5
  movss %xmm5, -12(%rbp)
.L5:
  cvtss2sd  -12(%rbp), %xmm0
  movl  $.LC10, %edi
  movl  $1, %eax
  call  printf
  movl  $0, %eax
  movq  -8(%rbp), %rdx
  xorq  %fs:40, %rdx
  je  .L15
  call  __stack_chk_fail
.L15:
  leave
  .cfi_def_cfa 7, 8
  ret
  .cfi_endproc
.LFE0:
  .size main, .-main
  .section  .rodata
  .align 4
.LC3:
  .long 1140457472
  .align 4
.LC4:
  .long 1106247680
  .align 8
.LC5:
  .long 2576980378
  .long 1069128089
  .align 4
.LC6:
```

```
135   .long 1150681088
136   .align 4
137 .LC7:
138   .long 1117782016
139   .align 8
140 .LC8:
141   .long 858993459
142   .long 1069757235
143   .align 8
144 .LC9:
145   .long 2576980378
146   .long 1070176665
147   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
148   .section  .note.GNU-stack,"",@progbits
```

Apareceram esses 2 novos mnemônicos no micro09, que são descritos da seguinte maneira: ucomiss %xmm1, %xmm0
Esse comando está fazendo uma comparação de valores do tipo float, xmm1 e xmm0 e define os registradores de acordo com o resultado, maior que, menor que, igual, desordenado
ja .L2 Jump above- ramifica a instrução, tem basicamente função similar aos jumps já apresentado, direciona para a próxima instrução

10. micro10. Cálculo de fatorial.

Listagem 2.68: micro10.s

```
1   .file "micro10.c"
2   .text
3   .globl  fator
4   .type fator, @function
5 fator:
6 .LFB2:
7   .cfi_startproc
8   pushq %rbp
9   .cfi_def_cfa_offset 16
10  .cfi_offset 6, -16
11  movq  %rsp, %rbp
12  .cfi_def_cfa_register 6
13  subq  $16, %rsp
14  movl  %edi, -4(%rbp)
15  cmpl  $1, -4(%rbp)
16  jle .L2
17  movl  -4(%rbp), %eax
18  subl  $1, %eax
19  movl  %eax, %edi
20  call  fator
21  imull -4(%rbp), %eax
22  jmp .L3
23 .L2:
24  movl  $1, %eax
25 .L3:
26  leave
27  .cfi_def_cfa 7, 8
28  ret
29  .cfi_endproc
30 .LFE2:
31  .size fator, .-fator
```

```
32   .section  .rodata
33 .LC0:
34   .string "Digite um numero: "
35 .LC1:
36   .string "%d"
37 .LC2:
38   .string "O fatorial de %d \303\251 %d\n"
39   .text
40   .globl  main
41   .type main, @function
42 main:
43 .LFB3:
44   .cfi_startproc
45   pushq %rbp
46   .cfi_def_cfa_offset 16
47   .cfi_offset 6, -16
48   movq  %rsp, %rbp
49   .cfi_def_cfa_register 6
50   subq  $16, %rsp
51   movq  %fs:40, %rax
52   movq  %rax, -8(%rbp)
53   xorl  %eax, %eax
54   movl  $0, -16(%rbp)
55   movl  $.LC0, %edi
56   movl  $0, %eax
57   call  printf
58   leaq  -16(%rbp), %rax
59   movq  %rax, %rsi
60   movl  $.LC1, %edi
61   movl  $0, %eax
62   call  __isoc99_scanf
63   movl  -16(%rbp), %eax
64   movl  %eax, %edi
65   call  fator
66   movl  %eax, -12(%rbp)
67   movl  -16(%rbp), %eax
68   movl  -12(%rbp), %edx
69   movl  %eax, %esi
70   movl  $.LC2, %edi
71   movl  $0, %eax
72   call  printf
73   movl  $0, %eax
74   movq  -8(%rbp), %rcx
75   xorq  %fs:40, %rcx
76   je  .L6
77   call  __stack_chk_fail
78 .L6:
79   leave
80   .cfi_def_cfa 7, 8
81   ret
82   .cfi_endproc
83 .LFE3:
84   .size main, .-main
85   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
86   .section  .note.GNU-stack,"",@progbits
```

Neste caso não há novos comandos comparados com os anteriores já apresentados, basta então seguir as explicações apresentadas para a compreensão deste código.

11. micro11. Exemplo com chamada de função.

Listagem 2.69: micro11.s

```
1   .file "micro11.c"
2   .text
3   .globl  verifica
4   .type verifica, @function
5 verifica:
6 .LFB0:
7   .cfi_startproc
8   pushq %rbp
9   .cfi_def_cfa_offset 16
10  .cfi_offset 6, -16
11  movq  %rsp, %rbp
12  .cfi_def_cfa_register 6
13  movl  %edi, -20(%rbp)
14  cmpl  $0, -20(%rbp)
15  jle .L2
16  movl  $1, -4(%rbp)
17  jmp .L3
18 .L2:
19  cmpl  $0, -20(%rbp)
20  jns .L4
21  movl  $-1, -4(%rbp)
22  jmp .L3
23 .L4:
24  movl  $0, -4(%rbp)
25 .L3:
26  movl  -4(%rbp), %eax
27  popq  %rbp
28  .cfi_def_cfa 7, 8
29  ret
30  .cfi_endproc
31 .LFE0:
32  .size verifica, .-verifica
33  .section  .rodata
34 .LC0:
35  .string "Digite um n\303\272mero: "
36 .LC1:
37  .string "%d"
38 .LC2:
39  .string "N\303\272mero positivo"
40 .LC3:
41  .string "Zero"
42 .LC4:
43  .string "N\303\272mero negativo"
44  .text
45  .globl  main
46  .type main, @function
47 main:
48 .LFB1:
49  .cfi_startproc
50  pushq %rbp
51  .cfi_def_cfa_offset 16
52  .cfi_offset 6, -16
53  movq  %rsp, %rbp
54  .cfi_def_cfa_register 6
55  subq  $16, %rsp
56  movq  %fs:40, %rax
```

```
57   movq  %rax, -8(%rbp)
58   xorl  %eax, %eax
59   movl  $.LC0, %edi
60   movl  $0, %eax
61   call  printf
62   leaq  -16(%rbp), %rax
63   movq  %rax, %rsi
64   movl  $.LC1, %edi
65   movl  $0, %eax
66   call  __isoc99_scanf
67   movl  -16(%rbp), %eax
68   movl  %eax, %edi
69   call  verifica
70   movl  %eax, -12(%rbp)
71   cmpl  $1, -12(%rbp)
72   jne .L7
73   movl  $.LC2, %edi
74   call  puts
75   jmp .L8
76 .L7:
77   cmpl  $0, -12(%rbp)
78   jne .L9
79   movl  $.LC3, %edi
80   call  puts
81   jmp .L8
82 .L9:
83   movl  $.LC4, %edi
84   call  puts
85 .L8:
86   movl  $0, %eax
87   movq  -8(%rbp), %rdx
88   xorq  %fs:40, %rdx
89   je  .L11
90   call  __stack_chk_fail
91 .L11:
92   leave
93   .cfi_def_cfa 7, 8
94   ret
95   .cfi_endproc
96 .LFE1:
97   .size main, .-main
98   .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
99   .section  .note.GNU-stack,"",@progbits
```

Neste caso não há novos comandos comparados com os anteriores já apresentados, basta então seguir as explicações apresentadas para a compreensão deste código.

Capítulo 3

Apêndice

3.1 Mnemônicos

Nesta seção serão apresentados os mnemônicos utilizados nas codificações deste trabalho de forma genérica.

ADD - Arithmetic Addition (Adição Aritmetica)
Adiciona "fonte"a "destino"e substitui o conteúdo original de "destino".
Ambos os operandos são binários.

CMP – Compare (Comparar)
Subtrai a fonte do destino e atualiza os valores mas não salva os resultados. Esses valores posteriormente são usados para comparações.

JA - Jump Above Faz com que a execução se ramifique para "rótulo"se a bandeira Carry e a bandeira zero estiverem claras. Comparação não assinada.

JE - Jump Equal (“Pular Igual”)
Faz com que a execução se ramifique para "rotular"através de comparação de ‘=’ sem sinal.

JG – Jump Greater (“Pular Maior”)
Faz com que a execução se ramifique para ‘’rotular” através da comparação ‘>’ com sinal.

JMP - Unconditional Jump (“Pulo incondicional”)
Transfere incondicionalmente o controle para "rótulo"(instrução). Os saltos por padrão estão dentro de -32768 a 32767 bytes das instruções após o salto.

LEA - Load Effective Address (Carregar Endereço Efetivo)
Transfere o endereço da ‘’fonte” para o registrador de destino.

LEAVE - Restore Stack for Procedure Exit (Restaurar Pilha para procedimento de saída)
Libera as variáveis locais criadas pela instrução de entrada anterior.

MOV - Move Byte or Word (Mover byte ou palavra)
Copia byte ou palavra da fonte para o destino.

MOVS - Move String (Byte or Word) (Mover String)
Basicamente faz o mesmo que MOV.

MUL - Unsigned Multiply (Multiplicação sem sinal)
Multiplica o acumulador pela fonte, multiplicação sem sinal.

TEST - Test for Bit Pattern) Executa um AND lógico dos dois operandos atualizando o registro de sinalizadores sem salvar o resultado.

UNCOMISS- Unordered Compare Scalar Single-Precision Floating- Point Values and Set EFLAGS) Executa e desordenada comparar os valores de ponto flutuante de precisão única nas duplas duplas do operando de origem 1 (primeiro operando) e o operando de origem 2 (segundo operando)
e define os sinalizadores ZF, PF e CF no registro EFLAGS De acordo com o resultado (não ordenado, maior que, menor ou igual).

XOR - Exclusive OR (OU exclusivo)
Executa um OR exclusivo bit a bit dos operandos e retorna o resultado no destino.

3.2 Diretivas

As diretivas são comandos que fazem parte da sintaxe do montador, mas não estão relacionados ao conjunto de instruções do processador x86. As informaçãoes desta seção foram retiradas do Oracle x86 Reference Manual.

.align integer, pad
A diretiva .align faz com que os próximos dados gerados sejam alinhados a bytes de módulo inteiro.
Integer deve ser uma expressão de número inteiro positivo e deve ser uma potência de 2. Se especificado,
pad é um valor de byte inteiro usado para preenchimento. O valor padrão do pad para a seção de texto é 0x90 (nop); Para outras seções, o valor padrão do pad é zero (0).

.ascii "string"
A diretriz .ascii coloca os caracteres em cadeia no módulo de objeto na localização atual,

mas não termina a string com um byte nulo (0).
String deve ser entre aspas duplas (") (ASCII 0x22). A diretiva .ascii não é válida para a seção .bss.

.bcd integer
A diretiva .bcd gera um valor decimal (80 bits) compactado na seção atual. A diretiva .bcd não é válida para a seção .bss.

.bss
A diretiva .bss altera a seção atual para .bss.

.bss symbol integer
Defina o símbolo na seção .bss e adicione bytes inteiros ao valor do contador de localização para .bss. Quando emitido com argumentos,
a diretiva .bss não altera a seção atual para .bss. Integer deve ser positivo.

.byte byte1, byte2, ... byteN
A diretiva .byte gera bytes inicializados na seção atual. A diretiva .byte não é válida para a seção .bss. Cada byte deve ser um valor de 8 bits.

.2byte expression1, expression2, ..., expressionN
Consulte a descrição da diretiva .value.

.4byte expression1, expressão2, ..., expressionN
Consulte a descrição da diretriz .long.

.8byte expression1, expression2, ..., expressionN
Consulte a descrição da diretriz .quad.

.comm
A diretriz .comm aloca o armazenamento na seção de dados. O armazenamento é referenciado pelo nome do identificador.
O tamanho é medido em bytes e deve ser um número inteiro positivo. O nome não pode ser predefinido.
O alinhamento é opcional. Se o alinhamento for especificado, o endereço do nome está alinhado a um múltiplo de alinhamento.

.data
A diretiva .data altera a seção atual para .data.

.double float
A diretiva .double gera uma constante de ponto flutuante de dupla precisão na seção atual.
A diretiva .double não é válida para a seção .bss.

.even
A diretriz .even alinha o contador do programa atual (.) Para um limite par.

.ext expression1, expression2, ..., expressionN
A diretiva .ext gera uma constante de ponto flutuante 80387 de 80 bits para cada expressão na seção atual. A diretiva .ext não é válida para a seção .bss.

.float float
A diretiva .float gera uma constante de ponto flutuante de precisão única na seção atual. A diretriz .float não é válida na seção .bss.

.globl symbol1, symbol2, ..., symbolN
A diretiva .globl declara que cada símbolo da lista é global. Cada símbolo é definido externamente ou definido no arquivo de entrada e acessível em outros arquivos.
As ligações padrão para o símbolo são substituídas. Uma definição de símbolo global em um arquivo satisfaz uma referência indefinida ao mesmo símbolo global em outro arquivo.
Não são permitidas definições múltiplas de um símbolo global definido. Se um símbolo global definido tiver mais de uma definição, ocorre um erro.
A diretriz .globl declara apenas que o símbolo é global, não define o símbolo.

.hidden
A diretiva .hidden declara que cada símbolo na lista deve ter o escopo escondido do vinculador.
Todas as referências ao símbolo dentro de um módulo dinâmico se ligam à definição dentro desse módulo. O símbolo não é visível fora do módulo.

.ident "string"
A diretiva .ident cria uma entrada na seção .comment contendo seqüência de caracteres.
A seqüência é qualquer seqüência de caracteres, não incluindo a cotação dupla (").
Para incluir o caractere de cotação dupla dentro de uma seqüência de caracteres, preceda o caractere de cotação dupla com uma barra invertida (ASCII 0x5C).

.lcomm, nome , tamanho, alinhamento
A diretriz .lcomm aloca armazenamento na seção .bss.
O armazenamento é referenciado pelo nome do símbolo e possui um tamanho de bytes de tamanho.
O nome não pode ser predefinido e o tamanho deve ser um número inteiro positivo.
Se o alinhamento for especificado, o endereço do nome está alinhado com um múltiplo de bytes de alinhamento.
Se o alinhamento não for especificado, o alinhamento padrão é de 4 bytes.

.local symbol1, symbol2, ..., symbolN

A diretriz .local declara que cada símbolo na lista é local. Cada símbolo é definido no arquivo de entrada e não acessível a outros arquivos.
As ligações padrão para os símbolos são substituídas. Os símbolos declarados com a diretiva .local prevalecem sobre símbolos fracos e globais.

.long expression1, expression2, ..., expressionN
A diretiva .long gera um inteiro longo (32 bits, o valor do complemento de dois) para cada expressão na seção atual.
Cada expressão deve ser um valor de 32 bits e deve avaliar para um valor inteiro. A diretriz .long não é válida para a seção .bss.

.popsection
A diretiva .popsection exibe o topo da pilha de seção e continua o processamento da seção surgida.

.previous
A diretiva .previous continua a processar a seção anterior.

.pushsection section
A diretriz .pushsection empurra a seção especificada para a pilha de seção e muda para outra seção.

.quad expression1, expression2, ..., expressionN

A diretiva .quad gera uma palavra inicializada (64 bits, o valor do complemento de dois) para cada expressão na seção atual.
Cada expressão deve ser um valor de 64 bits e deve ser avaliada em um valor inteiro. A diretiva .quad não é válida para a seção .bss.

.rel symbol @ type
A diretiva .rel gera o tipo de entrada de deslocamento especificado para o símbolo especificado.

.section section,attributes
A diretriz de .section faz seção a seção atual. Se a seção não existir, é criada uma nova seção com o nome e os atributos especificados.
Se a seção for uma seção não reservada, os atributos devem ser incluídos, a primeira seção de tempo é especificada pela diretiva .section.

.set symbol, expression
A diretiva .set atribui o valor da expressão ao símbolo.
Expressão pode ser qualquer expressão legal que avalie um valor numérico.

.size symbol, expr
Declara o tamanho do símbolo a ser expr. expr deve ser uma expressão absoluta.

.skip inteiro, valor
Ao gerar valores para qualquer seção de dados, a diretiva .skip faz com que os bytes inteiros sejam ignorados ou, opcionalmente, preenchidos com o valor especificado.

.sleb128 expression
A diretiva .sleb128 gera um número assinado, pequeno-endiante, base 128 da expressão.

.string
A diretriz .string coloca os caracteres em cadeia no módulo de objeto na localização atual e termina a string com um byte nulo (0).
String deve ser entre aspas duplas (") (ASCII 0x22). A diretiva .string não é válida para a seção .bss.

.symbolic simbólico1, símbolo2, ..., símboloN
A diretriz .symbolic declara cada símbolo na lista para o escopo do vinculador havesymbolic. Todas as referências ao símbolo dentro de um módulo dinâmico se ligam à definição dentro desse módulo. Fora do módulo, o símbolo é tratado como global.

.tbss
A diretiva .tbss altera a seção atual para .tbss. A seção .tbss contém objetos de dados TLS não inicializados que serão inicializados a zero pelo vinculador de tempo de execução.

.tcomm
A diretiva .tcomm define um bloco comum TLS.

.tdata
A diretiva .tdata altera a seção atual para .tdata. A seção .tdata contém a imagem de inicialização para objetos de dados TLS inicializados.

.texto
A diretiva de texto define a seção atual como .text.

.type symbol[, symbol, ..., symbol], type[, visibility]
Declara o tipo de símbolo e onde a visibilidade pode ser uma das seguintes:
hidden protected eliminate singleton exportado internal

.uleb128 expression

A diretiva .uleb128 gera um número não assinado, pequeno-endiante, base 128 da expressão.

.value expression1, expression2, ..., expressionN

A diretiva .value gera uma palavra inicializada (16 bits, valor de complemento de dois) para cada expressão na seção atual.
Cada expressão deve ser um valor inteiro de 16 bits. A directiva .value não é válida para a seção .bss.

.weak symbol1, symbol2, ..., symbolN

A diretiva .weak declara que cada símbolo na lista de argumentos deve ser definido externamente ou no arquivo de entrada e acessível a outros arquivos.
As ligações padrão do símbolo são substituídas pela diretriz .weak. Uma definição de símbolo fraco em um arquivo satisfaz uma referência indefinida a um símbolo global do mesmo nome em outro arquivo.
Símbolos fracos não resolvidos têm um valor padrão de zero. O editor de links não resolve esses símbolos.
Se um símbolo fraco tiver o mesmo nome que um símbolo global definido, o símbolo fraco é ignorado e nenhum resultado de erro. A diretriz .weak não define o símbolo.

.zero expression

Ao preencher uma seção de dados, a diretriz .zero preenche o número de bytes especificados pela expressão com zero (0).