Aluno: Daniel Flavio Volski Daum

Compilação e geração de arquivo em Assembly

O código abaixo escrito em linguagem C realiza a alocação dinâmica de memória e utiliza-se da aritmética de ponteiros para armazenar 10 números da sequência de fibonacci.

```
1   #include<stdio.h>
2   #include<malloc.h>
3
4   int main()
5   {
6       int *p;
7       p = (int *) malloc(10 * sizeof(int));
8
9       if(p == NULL) {
10          printf("Erro ao alocar memória!");
11      } else {
12          //Premissas
13          *(p+0) = 1;
14          *(p+1) = 2;
15          //Preenche o ponteiro p com 10 valores da sequência Fibonacci
16          for(int i=2; i<10; i++) {
17              *(p+i) = *(p+(i-1)) + *(p+(i-2));
18          }
19          //Imprime os valores na tela
20          for(int i=0; i<10; i++) {
21              printf("%d\n", *(p+i));
22          }
23          //Libera o espaço de memória alocada
24          free(p);
25      }
26  }
```

O mesmo código quando compilado da utilizando a linha de comando “gcc -S exercicio1.c” gera o seguinte código na linguagem Assembly.

```
1               .file   "exercicio1.c"
2               .text
3               .section .rodata
4   .LC0:
5               .string "Erro ao alocar mem\303\263ria!"
6   .LC1:
7               .string "%d\n"
8               .text
9               .globl  main
10              .type   main, @function
11  main:
12  .LFB0:
13              .cfi_startproc
14              endbr64
15              pushq   %rbp
16              .cfi_def_cfa_offset 16
17              .cfi_offset 6, -16
18              movq    %rsp, %rbp
19              .cfi_def_cfa_register 6
20              subq    $16, %rsp
21              movl    $40, %edi
```

```
22              call    malloc@PLT
23              movq    %rax, -8(%rbp)
24              cmpq    $0, -8(%rbp)
25              jne     .L2
26              leaq    .LC0(%rip), %rdi
27              movl    $0, %eax
28              call    printf@PLT
29              jmp     .L3
30      .L2:
31              movq    -8(%rbp), %rax
32              movl    $1, (%rax)
33              movq    -8(%rbp), %rax
34              addq    $4, %rax
35              movl    $2, (%rax)
36              movl    $2, -16(%rbp)
37              jmp     .L4
38      .L5:
39              movl    -16(%rbp), %eax
40              cltq
41              salq    $2, %rax
42              leaq    -4(%rax), %rdx
43              movq    -8(%rbp), %rax
44              addq    %rdx, %rax
45              movl    (%rax), %ecx
46              movl    -16(%rbp), %eax
47              cltq
48              salq    $2, %rax
49              leaq    -8(%rax), %rdx
50              movq    -8(%rbp), %rax
51              addq    %rdx, %rax
52              movl    (%rax), %edx
53              movl    -16(%rbp), %eax
54              cltq
55              leaq    0(,%rax,4), %rsi
56              movq    -8(%rbp), %rax
57              addq    %rsi, %rax
58              addl    %ecx, %edx
59              movl    %edx, (%rax)
60              addl    $1, -16(%rbp)
61      .L4:
62              cmpl    $9, -16(%rbp)
63              jle     .L5
64              movl    $0, -12(%rbp)
65              jmp     .L6
66      .L7:
67              movl    -12(%rbp), %eax
68              cltq
69              leaq    0(,%rax,4), %rdx
70              movq    -8(%rbp), %rax
71              addq    %rdx, %rax
72              movl    (%rax), %eax
73              movl    %eax, %esi
74              leaq    .LC1(%rip), %rdi
75              movl    $0, %eax
76              call    printf@PLT
77              addl    $1, -12(%rbp)
78      .L6:
79              cmpl    $9, -12(%rbp)
80              jle     .L7
81              movq    -8(%rbp), %rax
82              movq    %rax, %rdi
83              call    free@PLT
```

```
84  .L3:
85          movl    $0, %eax
86          leave
87          .cfi_def_cfa 7, 8
88          ret
89          .cfi_endproc
90  .LFE0:
91          .size   main, .-main
92          .ident  "GCC: (Ubuntu 10.3.0-1ubuntu1) 10.3.0"
93          .section.note.GNU-stack,"",@progbits
94          .section.note.gnu.property,"a"
95          .align 8
96          .long   1f - 0f
97          .long   4f - 1f
98          .long   5
99  0:
100         .string "GNU"
101 1:
102         .align 8
103         .long   0xc0000002
104         .long   3f - 2f
105 2:
106         .long   0x3
107 3:
108         .align 8
109 4:
```

Com o código Assembly gerado acima é possível realizar algumas observações as quais são expressas na tabela abaixo, que correlaciona a linha de algumas instruções do código *Assembly* com uma breve descrição e sua possível equivalência no código escrito em C.

Linha	Instrução Assembly	Descrição	Instrução C
1	.file "exercicio1.c"	Identifica o nome do arquivo original a partir do qual foi gerado o Assembly.	*
11 12	main: .LFB0	Indica o início da função local main.	Int main()
24	cmpq $0, -8(%rbp)	Realiza a comparação entre 0 e o endereço indicado.	(p == NULL)
25	jne .L2	Caso o valor da comparação seja diferente de 0, pula para o bloco de indtruções .L2	else
26 28	leaq .LC0(%rip), %rdi call printf@PLT	Carrega a constante local (String) e então chama a função printf.	printf("Erro ao alocar memória!");
29	jmp .L3	Pula diretamente para o bloco de instruções com rótulo (*label*) .L3	Seria o equivalente a finalização do programa
30	.L2:	Indica o início do bloco de instruções rotulado .L2.	Equivale ao bloco de instruções indicados pela palavra-chave *else* da linha 11.
32 35	movl $1, (%rax) movl $2, (%rax)	Realiza a manipulação das constantes 1 e 2 para o registrador %*rax*	*(p+0) = 1; *(p+1) = 2;
37	jmp .L4	Pula diretamente para o bloco de instruções rotulado .L4.	for(int i=2; i<10; i++)
62 63	cmpl $9, -16(%rbp) jle .L5	Realiza mais uma comparação e caso o valor seja menor do que o especificado	Equivale a comparação feita no laço de

		pula para o bloco de instruções rotulado .L5.	repetição for da linha 17
38	.L5:	Indica o início do bloco de instruções rotulado .L5.	O rótulo .L7 equivale a linha 17.
65	jmp .L6	Pula para o bloco de instruções rotulado .L6	O rótulo .L6 equivale as linhas 20 e 24.
79 80	cmpl $9, -12(%rbp) jle .L7	Realiza mais uma comparação e caso o valor seja menor do que o especificado pula para o bloco de instruções rotulado .L7.	Equivale a comparação feita no laço de repetição for da linha 20
66	.L7	Indica o início do bloco de instruções rotulado .L7.	O bloco de instruções .L7 corresponde a linha 21,
74 76	leaq .LC1(%rip), %rdi call printf@PLT	Carrega a constante local que armazena o texto que deve ser exibido, e logo chama a função printf()	printf(“%d\n”, *(p+1))
83	call free@PLT	Realiza a chamada da função free()	free(p)
90	.LFE0:	Indica o fim da função local.	Finalização do programa.

É válido ressaltar a função de algumas instruções. A próxima tabela mostra a sintaxe de algumas instruções junto a sua descrição. As letras O e D indicam respectivamente origem e destino e B indica o rótulo de um bloco de instrução:

Sintaxe	**Descrição**
mov O, D	Realiza a transferência de dados da origem para o destino.
add O, D	Realiza a adição da origem para o destino.
sub O, D	Realiza a subtração da origem para o destino.
call O	Realiza a chamada de uma função.
jmp B	Redireciona o fluxo para o rótulo indicado.
cmp O, D	Realiza a comparação entre a origem e destino.
jne B	Redireciona o fluxo para o rótulo indicado caso não seja igual a zero.
leaq O, D	Carrega o endereço da origem para o destino.