Conversão entre Níveis

Guião -3r

Resolução com nível de otimização –O2

Exercício 1

$gcc –Wall –O2 main.c imprime.c –S

$ cat main.s
```
main:
      pushl   %ebp
      movl    %esp, %ebp
      subl    $8, %esp
      andl    $-16, %esp
      movl    $10, b
      movl    $20, a
      call    imprime
      leave
      ret
```

$cat imprime.s
```
.LC0:
      .string     "a=%d\tb=%d\tc=%d\n"
      .text
      .p2align 2,,3
.globl imprime
      .type imprime,@function
imprime:
      pushl %ebp
      movl  %esp, %ebp
      subl  $8, %esp
      movl  a, %ecx
      movl  b, %edx
      movl  %ecx, %eax
      subl  %edx, %eax
      pushl %eax
      pushl %edx
      pushl %ecx
      pushl $.LC0
      call  printf
      leave
      ret
```

Exercício 2

a) **Operações aritméticas**:

main.s
A instrução a = b * 2, foi codificada em código assembly por:
```
movl  $20, a
```

Uma vez que se compilou com -02 o compilador conseguiu concluir automaticamente que o valor da variável 'a' seria 20, razão pela qual a operação de multiplicação não se encontra explicitamente codificada em código assembly.

Imprime.s
A instrução c = a – b, foi codificada em:

```
subl  %edx, %eax
```

b) chamadas às funções

main.s :
```
      call imprime
```
imprime.s :
```
      call printf
```

c) O retorno do programa e de funções

main.s
```
      leave ;Prepara o  retorno da função
      ret         ;procede ao retorno função
```

imprime.s
```
      leave ;Prepara o  retorno da função
      ret         ;procede ao retorno da função
```

Exercício 3

a) Variáveis locais

main.s : Não existem variáveis locais;

imprime.s
```
movl  %ecx, %eax ;c  = a (modo endereçamento registo)
…
pushl %ecx ;copia o valor em %ecx (variável c) para  topo da pilha
```

b) Variáveis Globais

main.s:

Dados gravados na própria instrução (endereçamento imediato):
```
movl  $10, b    ; b = 10
movl  $20, a      ; a =  20
```

imprime.s:

Endereços gravados num registo (endereçamento indireto):
```
movl  a, %ecx     ; %ecx = a
```

```
movl  b, %edx     ; %edx = b
```

As variáveis a e b, neste momento estão representadas pelos respectivos símbolos, que durante a execução corresponderão a endereços de memória, usados para ler ou escrever de/para registos da CPU.

c) Constantes

main.s: não definidas.

imprime.s

. LC0:

```
      .string     "a=%d\tb=%d\tc=%d\n"
```

pushl $.LC0

Coloca o endereço da string "a=%d\tb=%d\tc=%d\n" (1º argumento de chamada da função printf) no topo da pilha passado, juntamente com o valor das variáveis a, b e c) que correspondem aos restantes argumentos da função printf.

d) Altere o código C dos módulos acima de forma a que a operação *(c=a-b)* seja realizada na *main.c* e que o valor resultante seja passado como parâmetro a *imprime.c.* Repita o passo 1.

main.c:

```
int a,b;

void imprime(int);

main () {
        int c;
        b = 10; a = b * 2;
        c = a - b;
        imprime(c);
}
```

imprime. c

```
extern int a,b;
#include <stdio.h>
void imprime(int c){

        printf("a=%d\tb=%d\tc=%d\n",a,b,c);
}
```

$ gcc –Wall –O2 main.c imprime.c –S

Exercício 4

main.c

```
subl  $12, %esp   ; 12 octetos extras reservados na stack frame
```

```
pushl $10  ;o valor final do 'c' antes de ser passado como
argumento na função imprime. Com o nível de otimização -02 o
compilador calcula automaticamente  o valor a colocar no  topo,
uma vez que:  c = a – b; c = 20 – 10; c  = 10;
```

imprime.s

```
pushl     8(%ebp) ; coloca no topo da pilha o valor existente
no endereço %ebp + 8, o valor 10 passado como argumento na
chamada à função imprime, na função main.
```

$Introduza na função *main.c* a declaração (*unsigned u=32;*) e seguidamente acrescente as instruções (*a = b * 4; a = b * 8; a = u/2; a = u/4;*). Repita o passo 1.

main.c

```
int a,b;

void imprime(int);

main () {
        int c;
        b = 10; a = b * 2;
        c = a - b;
        imprime(c);
        unsigned u = 32;
        a = b * 4;
        a = b * 8;
        a = u / 2;
        a = u / 4;
}
```

$ gcc –Wall –O2 main.c imprime.c –S

Exercício 5

```
movl  $8, a ;
```

O compilador como sabe que:

1) o valor final tomado pela variável a é 8;
2) que a instrução final só depende da instrução `unsigned u = 32;`
3) e que as instruções a= b * 4; a = b * 8; a = u / 2, não alteram nenhum estado para além da própria variável a;

optimizou o código gerando apenas o valor final de a;

```
a = 32 / 4
a = 8;

movl  $8, a
```

Resolução com nível de otimização –O0

Exercício 1

$gcc –Wall main.c imprime.c –S
$ cat main.s

```
main:
      pushl %ebp
      movl  %esp, %ebp
      subl  $8, %esp
      andl  $-16, %esp
      movl  $0, %eax
      subl  %eax, %esp
      movl  $10, b
      movl  b, %eax
      sall  $1, %eax
      movl  %eax, a
      call  imprime
      leave
      ret
```

$cat imprime.s

```
.LC0:
      .string     "a=%d\tb=%d\tc=%d\n"
      .text
...
imprime:
      pushl %ebp
      movl  %esp, %ebp
      subl  $8, %esp
      movl  b, %eax
      movl  a, %edx
      subl  %eax, %edx
      movl  %edx, %eax
      movl  %eax, -4(%ebp)
      pushl -4(%ebp)
      pushl b
      pushl a
      pushl $.LC0
      call  printf
      addl  $16, %esp
      leave
      ret
```

Exercício 2

a) Operações aritméticas:

main.s
A instrução a = b * 2, foi codificada em código assembly por:

```
movl  b, %eax    ; copia a variável global b para %eax
sall  $1, %eax   ; %eax = %eax << 1; shift left de 1 bit que
                   corresponde a multiplicar por 2;
movl  %eax, a    ; copia o registo %eax para a variável a;
```

Imprime.s
A instrução c = a – b, foi codificada em:

```
movl  b, %eax       ; copia o valor de b para  %eax
movl  a, %edx       ; copia o valor de a para %edx
subl  %eax, %edx    ; subtrai %edx de %eax, o que equivale a
subtrair a de b
movl  %edx, %eax    ; copia %edx para %eax
movl  %eax, -4(%ebp); passar o valor de %eax para a posição na pilha
                      reservada à variável local c.
```

b) e c) solução igual à obtida com -02.

Exercício 3

a) Variáveis locais

main.s : Não existem variáveis locais;

imprime.s
```
movl  %eax, -4(%ebp) ; c  = %eax  copia %eax para o endereço de
                       memória %ebp - 4
…
pushl -4(%ebp) ; copia a variável 'c' para o topo da pilha para ser
                 usada a seguir como argumento da função printf
```

b) Variáveis Globais

main.s:
Dado contido na própria instrução(endereçamento imediato):
```
movl  $10, b            ; b = 10;
```

Endereços gravados num registo (modo de endereçamento registo):
```
movl  b, %eax     ; %eax = b;
movl  %eax, a     ; %eax = a;
```

imprime.s:
Endereços gravados num registo (endereçamento indireto):
```
movl  b, %eax     ; %eax = b
movl  a, %edx     ;%edx = a
```
…

c) solução igual à obtida com -02.

d) Altere o código C dos módulos acima de forma a que a operação (c=a-b) seja realizada na main.c e que o valor resultante seja passado como parâmetro a imprime.c. Repita o passo 1. ….

```
$ gcc –Wall main.c imprime.c –S
```

Exercício 4

Nesta versão temos novas instruções IA32, nomeadamente:
main.c

```
….
movl  %eax, a         ; a = %eax
movl  b, %edx         ; b = %edx
movl  a, %eax         ; %eax = a;
subl  %edx, %eax      ; %eax = %eax - %edx ; 'a' = 'a' - b
movl  %eax, -4(%ebp)  ; %ebp – 4 = %eax;     ;  'c' = a;
subl  $12, %esp       ; %esp = %esp - 12
pushl -4(%ebp)        ; coloca o valor da operação a – b, ou seja, 'c'
                        no topo da pilha
….
pushl –4(%ebp) ; Esta instruções foi adicionada para permitir passar
como argumento à função imprime, o  valor do 'c' que se encontra na
posição %ebp – 4 da pilha .
```

imprime.s

```
pushl 8(%ebp) ;Coloca no topo da pilha o valor existente no endereço
%ebp + 8 que corresponde ao endereço  da variável 'c' utilizada na
função main (valor passado como argumento na chamada à função imprime).
```

Introduza na função *main.c* a declaração (*unsigned u=32;*) e seguidamente acrescente as instruções (*a = b * 4; a = b * 8; a = u/2; a = u/4;*). Repita o passo 1.

main.c : (...) o código é o mesmo que o usado na secção -02.

$ gcc –Wall main.c imprime.c –S

Exercício 5

```
movl  $32, -8(%ebp) ; guarda (32) da variável u na posição %ebp - 8
movl  b, %eax       ; guarda o valor da variável b em registo
sall  $2, %eax      ; %eax = %eax << 2, shift 2 bits à esquerda; b * 4;
movl  %eax, a       ; a = %eax;
movl  b, %eax       ; %eax = b;
sall  $3, %eax      ; %eax = %eax << 3, shift 3 bits à esquerda; b * 8;
movl  %eax, a       ; a = %eax;
movl  -8(%ebp),%eax ; guarda o valor 32 (variável u) no registo %eax
shrl  $1, %eax      ; shift 1 bit à direita; u / 2;
movl  %eax, a       ; a = %eax
movl  -8(%ebp),%eax  ; guarda o valor 32 (variável u) no registo %eax
shrl  $2, %eax      ; shift 2 bits à direita; u / 4;
movl  %eax, a       ; a = %eax
```