Conversão entre Níveis

Guião III - Resolução

Resolução com nível de otimização -O2

Exercício 1

```
$ gcc -Wall -S -O2 main.c imprime.c
```

```
$ cat main.s
```

```
main:
      pushl   %ebp        ; guarda EBP antigo na pilha
      movl    %esp, %ebp  ; define o novo EBP
      subl    $8,   %esp
      andl    $-16, %esp
      movl    $10,  b     ; guarda 10 na variável b
      movl    $20,  a     ; a=b*2 otimizado (a=b*2=10*2=20)
      call    imprime     ; chama a função imprime
      leave               ; Copia EBP para ESP, repõe EBP com valor antigo
      ret                 ; Coloca em EIP o endereço de retorno guardado
                          ; na pilha
```

```
$ cat imprime.s
```

```
.LC0:
      .string     "a=%d\tb=%d\tc=%d\n"
      .text
      .p2align 2,,3
.globl imprime
      .type imprime,@function
imprime:
      pushl %ebp          ; guarda EBP antigo na pilha
      movl  %esp, %ebp    ; define o novo EBP
      subl  $8,   %esp    ; reserva 8 bytes na pilha para variáveis locais
      movl  a,    %ecx    ; ecx <=> a
      movl  b,    %edx    ; edx <=> b
      movl  %ecx, %eax    ; eax=a
      subl  %edx, %eax    ; eax=eax-edx=a-b <=> c
      pushl %eax          ; 4º argumento de printf (c)
      pushl %edx          ; 3º argumento de printf (b)
      pushl %ecx          ; 2º argumento de printf (a)
      pushl $.LC0         ; 1º argumento de printf (string formatação .LC0)
      call  printf        ; chama a função printf
      leave               ; Copia EBP para ESP, repõe EBP com valor antigo
      ret                 ; Coloca em EIP o endereço de retorno guardado
                          ; na pilha
```

Exercício 2

a) Operações aritméticas:

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

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

```
subl  %edx, %eax  ; eax = eax-edx = a-b <=> c
```

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 (movl %ebp, %esp ; popl %ebp)
      ret   ; procede ao retorno função
```

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

Exercício 3

a) Acesso a variáveis locais

main.s
```
      Não existem variáveis locais.
```

imprime.s

```
      movl  %ecx, %eax ; eax=a (endereçamento por registo)
      subl  %edx, %eax ; eax=eax-edx=a-b <=> c
      pushl %eax       ; copia o valor de %eax (c) para o topo da pilha
```

b) Acesso a variáveis globais

main.s

Os valores a escrever e o endereço de memória são incluídos na própria instrução (**endereçamento imediato e absoluto**):

```
movl   $10, b        ; b = 10
movl   $20, a        ; a = 20
```

imprime.s

O endereço de memória (variável) a ler está incluído na instrução (**endereçamento absoluto e por registo**):

```
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 do CPU.

c) Acesso a constantes

main.s

```
       Não definidas.
```

imprime.s

```
.LC0:
       .string     "a=%d\tb=%d\tc=%d\n"
       ...
       pushl   $.LC0
```

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

Altere o código C dos módulos acima de modo a que a operação ***c=a-b*** seja realizada no ***main*** e o valor resultante seja passado como parâmetro a ***imprime***. 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){
      int c;
      c = a - b;
      printf("a=%d\tb=%d\tc=%d\n",a,b,c);
}
```

```
$ gcc -Wall -O2 -S main.c imprime.c
```

Exercício 4

main.c

```
subl  $12, %esp   ; 12 bytes extra reservados na stack frame

pushl $10   ; O valor de c é passado como argumento à função
            ; imprime.
            ; Com o nível de otimização -O2 o compilador calcula
            ; automaticamente o valor a colocar no topo da pilha,
            ; uma vez que: c = a - b, c = 20 - 10, c  = 10.
```

imprime.s

Em vez de se passar o 4º argumento ao **printf** da seguinte forma:

```
   pushl  %eax    ; coloca no topo da pilha o 4º arg. do printf (c)
```

passa a ser:

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

Introduzir na função ***main*** a declaração ***unsigned u=32;*** e seguidamente acrescentar as instruções: ***a = b*4; a = b*8; a = u/2; a = u/4;***. Repetir 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 -S main.c imprime.c
```

Exercício 5

```
movl   $8, a
```

Como o compilador sabe que:

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

otimizou 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 -S -O0 main.c imprime.c
$ cat main.s
```

```
main:
      pushl  %ebp       ; guarda EBP antigo na pilha
      movl   %esp, %ebp ; define o novo EBP
      subl  $8, %esp
      andl  $-16, %esp
      movl  $0, %eax
      subl  %eax, %esp
      movl  $10, b      ; b=10
      movl  b, %eax     ; eax=b
      sall  $1, %eax    ; eax = eax<<1 = b<<1 = b*2
      movl  %eax, a     ; a = eax = b*2
      call  imprime     ; chama a função imprime
      leave             ; Copia EBP para ESP, repõe EBP com valor antigo
      ret               ; Coloca em EIP o endereço de retorno guardado
                        ; na pilha
```

```
$ cat imprime.s
```

```
.LC0:
      .string     "a=%d\tb=%d\tc=%d\n"
      .text
...
imprime:
      pushl %ebp
      movl  %esp, %ebp
      movl  b, %eax           ; eax=b
      movl  a, %edx           ; edx=a
      subl  %eax, %edx        ; edx=a-b
      movl  %edx, %eax        ; eax=a-b
      movl  %eax, -4(%ebp)    ; c=a-b (c está na pilha em EBP-4)
      pushl -4(%ebp)          ; coloca na pilha 4º arg do printf (c)
      pushl b                 ; coloca na pilha 3º arg do printf (b)
      pushl a                 ; coloca na pilha 2º arg do printf (a)
      pushl $.LC0             ; coloca na pilha 1º arg do printf (form.)
      call  printf            ; chama a função 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; equivale a subtrair a de b
movl  %edx, %eax      ; copia %edx (=a-b) para %eax
movl  %eax, -4(%ebp)  ; passar o valor de %eax para a posição na pilha
                      ; reservada à variável local c
```

b) chamadas às funções

Chamada e retorno de funções são iguais ao que se obtém com **-O2**.

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

c) O retorno do programa e de funções

main.s
```
      leave ; prepara o retorno da função <=> movl %ebp, %esp; popl %ebp
      ret   ; procede ao retorno função
```

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

Exercício 3

a) Acesso a variáveis locais

main.s : `Não existem variáveis locais.`

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

b) Acesso a variáveis globais

main.s:

Valor e endereço de memória contidos na instrução (endereçamento imediato e absoluto):

```
movl  $10, b       ; b = 10
```

Endereço contido na própria instrução (endereçamento absoluto e por registo):

```
movl  b, %eax      ; %eax = b
...
movl  %eax, a      ; a = %eax
```

imprime.s:

Endereço contido na própria instrução (endereçamento absoluto e por registo):

```
movl  b, %eax      ; %eax = b
movl  a, %edx      ; %edx = a
...
pushl b            ; coloca na pilha 3º argumento do printf (b)
pushl a            ; coloca na pilha 2º argumento do printf (a)
```

c) Acesso a constantes

Igual ao que se obtém com **-02**.

main.s

```
      Não definidas.
```

imprime.s

```
.LC0:
      .string     "a=%d\tb=%d\tc=%d\n"
      ...
      pushl  $.LC0
```

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

Altere o código C dos módulos acima de forma a que a operação ***c=a-b*** seja realizada no ***main*** e que o valor resultante seja passado como parâmetro a ***imprime***. 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){
      int c;
      c = a - b;
      printf("a=%d\tb=%d\tc=%d\n",a,b,c);
}
```

```
$ gcc -Wall -S -O0 main.c imprime.c
```

Exercício 4

Novas instruções IA32 em relação ao **main.s** obtido no passo 1:

main.s:

```
...
movl b, %edx          ; %edx = b
movl a, %eax          ; %eax = a
subl %edx, %eax       ; %eax = %eax - %edx = a-b
movl %eax, -4(%ebp)   ; M[%ebp-4] = %eax <=> c = a-b
subl $12, %esp        ; %esp = %esp - 12
pushl     -4(%ebp)    ; passar o valor de c como argumento à função
                      ; imprime. O valor do c fica guardado
                      ; na posição %ebp-4 da pilha
...
```

imprime.s:

Nesta versão temos uma instrução nova:

```
pushl 8(%ebp)      ; Corresponde a colocar no top da pilha o valor existente no endereço
                   ; %ebp + 8. Que corresponde à localização do valor da variável c
                   ; utilizada na função main (valor que foi passado como argumento na
                   ; chamada à função imprime).
```

que substitui as instruções:

```
movl  b, %eax
movl  a, %edx
subl  %eax, %edx        ; edx=a-b
movl  %edx, %eax
movl  %eax, -4(%ebp)    ; c=a-b
pushl -4(%ebp)          ; coloca argumento c de printf na pilha
```

Introduza na função ***main*** 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 -S -O0 main.c imprime.c
```

Exercício 5

main.s:

Novas instruções em relação ao **main.s** obtido no passo 4:

```
...
movl    $32, -8(%ebp)    ; guarda o valor da variável local u (32) na posição %ebp - 8
movl    b, %eax          ; guarda o valor da variável b em registo
sall    $2, %eax         ; %eax = %eax << 2, shift à esquerda de 2 bits <=> b * 4
movl    %eax, a          ; a = %eax = b * 4
movl    b, %eax          ; %eax = b;
sall    $3, %eax         ; %eax = %eax << 3, shift à esquerda de 3 bits <=> b * 8
movl    %eax, a          ; a = %eax = b * 8
movl    -8(%ebp), %eax   ; copia a variável local u (32) para o registo %eax
shrl    $1, %eax         ; shift à direita de 1 bit <=> u / 2
movl    %eax, a          ; a = %eax = u / 2
movl    -8(%ebp), %eax   ; copia a variável u (32) para o registo %eax
shrl    $2, %eax         ; shift à direita de 2 bits <=> u / 4
movl    %eax, a          ; a = %eax = u / 4
...
```

Exercício 6

```
gcc -Wall -O0 main5.c imprime5.c -o prog_o0
gdb ./prog_o0
```

a)

```
(gdb) dissas main
(gdb) dissas imprime
```

b) c)

```
(gdb) break main
(gdb) break imprime
…
(gdb) run
(gdb) stepi ou nexti
…
(gdb) print /x $Registo
…
(gdb) info registers
…
(gdb) print *(int *) 0xEndereçoVariável
…
(gdb) print *(int *) ($registo+ValorImediato)
…
(gdb) continue
…
```