Computadores de Programação (DCC/UFRJ)

Aula 14: Combinando código assembly com programas C

Prof. Paulo Aguiar

Combinando código assembly em programas C

...antigamente..

- No início da Computação, todos os programas eram escritos em linguagem de montagem
  - Complexidade, propenso a erros, código dependente da arquitetura da máquina
- Os primeiros compiladores de linguagens de alto-nível não geravam código muito eficiente e as linguagens de alto-nível não permitiam acesso a estruturas de dados mais próximas do hardware (necessário para programadores de sistemas)

Os programas que exigiam melhor desempenho e acesso a estruturas de dados mais elementares ainda precisavam ser escritos em linguagem de montagem *(assembly language)*

Combinando código assembly em programas C

...nos dias de hoje..

- A **capacidade de otimização dos compiladores** praticamente eliminou a necessidade de escrever código de montagem para atender requisitos de desempenho
- A **linguagem C** reduziu bastante a necessidade de escrever código de montagem para atender requisitos de acesso a estruturas de dados mais próximas do hardware (a maioria dos SOs, incluindo Linux, Windows e MacOS, são escritos em C)

Combinando código de montagem em programas C

...ainda nos dias de hoje..

Há casos ainda, entretanto, em que **escrever código de montagem é a única opção**

- Para escrever sistemas operacionais há a necessidade de acessar registradores especiais com informações sobre estado dos processos, usar instruções especiais de acesso à memória para implementar E/S, etc.
- Para programadores de aplicações há certas características da máquina, como os flags de condição, que não podem ser acessadas por C diretamente

Combinando código assembly em programas C

...outras motivações..

- Codificar funções fortemente dependentes da arquitetura da máquina
- Otimizar um caminho de código usado com frequência

A estratégia atual para todos esses casos é **integrar código em C com pequenos trechos escritos em linguagem de montagem**

Formas de combinar código de montagem com C

1. Escrever funções separadas em código assembly e ligar com código C
   - Respeitar as convenções para passagem de argumento e uso de registradores seguidas pelo compilador C
2. Embutir código de montagem num programa em C, enxertando-o diretamente no código gerado pelo GCC
   - Usa a diretiva especial provida pelo GCC chamada **asm** ("inline assembly")

Compondo código C e código de montagem

Para compilar basta fazer: `gcc teste.c soma.s`

```
#include <stdio.h>

int soma (int, int);

int main (void) {
   printf("%d\n", soma(-3, 5));
   return 0;
}
```

```
.global  soma

soma:
     pushl   %ebp
     movl    %esp, %ebp
     movl    8(%ebp), %eax
     addl    12(%ebp), %eax
     popl    %ebp
     ret
```

Compondo código C e código de montagem

Resultado da compilação:

```
.LC0: .string   "%d\n"
main:  pushl   %ebp
       movl    %esp, %ebp
       subl    $16, %esp
       movl    $5, 4(%esp)
       movl    $-3, (%esp)
       call    soma
       movl    %eax, 8(%esp)
       movl    $.LC0, 4(%esp)
       movl    $1, (%esp)
       call    __printf_chk
       movl    $0, %eax
       leave
       ret
```

```
.global  soma

soma:
       pushl   %ebp
       movl    %esp, %ebp
       movl    8(%ebp), %eax
       addl    12(%ebp), %eax
       popl    %ebp
       ret
```

Embutindo código de montagem em programas C

"inline assembly": usa a função: `asm (codigo-assembly);`

```c
#include<stdio.h>

int main(void) {

   printf("Ola, mundo!\n");

   // exit(0)
   asm ("movl $1, %eax \n\t"
        "xor %ebx, %ebx \n\t"
        "int $0x80");
}
```

```
.LC0:
        .string "Ola, mundo!"

main:
        pushl   %ebp
        movl    %esp, %ebp
        andl    $-16, %esp
        subl    $16, %esp
        movl    $.LC0, (%esp)
        call    puts
#APP
        movl $1, %eax
        xor %ebx, %ebx
        int $0x80
#NO_APP
        leave
        ret
```

A diretiva **asm** é específica para GCC e incompatível com outros compiladores

Exemplo de uso da diretiva asm

```
asm ( "movl $1, %eax;    #chamada de sistema exit
      xor %ebx, %ebx;  #zera %ebx
      int $0x80"          #gera a interrupção de SO
    );
```

Compilador insere as strings em assembly sem qualquer verificação. Erros serão reportados pelo assembler, se houver.

Tentativa de inserir código de montagem que falha

Pretende: retornar 1 se OK, retornar 0 se multiplicação dá overflow

Overflow na multiplicação detetado com o flag CF

/* Primeira tentativa que não funciona */

```
int tmult_ok1(int x, int y, int *dest)
{
int result = 0;
*dest = x*y;
asm("setae %al");
return result;
}
```

- setae copia ∼CF para o LSB de %eax, que deveria ter sido zerado antes já que é o registrador default de retorno

Tentativa de inserir código assembly que falha

x em %ebp+8, y em %ebp +12, dest em %ebp+16

Código assembly gerado

```
1 tmult_ok1:
2               pushl   %ebp
3               movl    %esp, %ebp
4               movl    12(%ebp), %eax      pega y
5               imull   8(%ebp), %eax       multiplica por x
6               movl    16(%ebp), %edx      pega dest
7               movl    %eax, (%edx)        aramazena produto em dest
                        código asm
8               setae   %al
                        fim do código asm
9               movl    $0, %eax            faz result = 0
10              popl    %ebp
11              ret
```

GCC somente zera result ao final e não no início como seria esperado!

Construtor **asm** (ou __asm__)

Registradores alocados randomicamente pelo compilador

```
int main (void) {
   int x=1, y=2, z;
   z = x + y;
   asm("addl %2, %0 \n\t"              // z += y
       "movl %2, %%eax \n\t"           // salva y em %eax
       "movl %1, %2 \n\t"              // y = x
       "movl %%eax, %1 \n\t"           // x = %eax
       : "=r"(z)                       // z é um operando de saida
       : "r" (x) , "r" (y)             // x e y são operandos de entrada
       : "%eax");                      // registradores modificados
   printf("%d %d %d\n ", x, y, z);
   return z;}
```

- Os campos **operandos de saída** e **operandos de entrada** devem consistir de uma **string de restrição de operandos**, seguida de uma expressão C entre parênteses
- A restrição do operando de saída deve ser precedida por "="
- Podemos ter vários operandos de entrada e de saída e registradores afetados, cada entrada deve ser separada por ","

Construtor **asm**

Registradores podem ser abreviados

- %eax - a
- %ebx - b
- %ecx - c
- %edx - d
- %esi - S
- %edi - D
- r - registrador randomicamente alocado

Duas formas

- Nomes referenciados como %0 a %9 na ordem de citação, considerando as duas listas (para versão antiga de GCC)
- Por nome (% [nome] como %%eax) (para GCC 3.3 ou superior)

Exemplo de asm com referência numérica

Registradores alocados randomicamente pelo GCC, sem operandos de saída

```
int main (void) {
   int x=1, y=2, z;
   z = x + y;
   asm("addl %2, %0 \n\t"              // z += y
       "movl %2, %%eax \n\t"           // salva y em %eax
       "movl %1, %2 \n\t"              // y = x
       "movl %%eax, %1 \n\t"           // x = %eax
       :                               // operandos de saída
       : "r"(z), "r" (x) , "r" (y)     // observar a ordem aqui
       : "%eax");                      // registradores modificados
   printf("%d %d %d\n ", x, y, z);
   return z;}
```

- %0 referencia z, %1 referencia x, %2 referencia y
- Nas instruções assembly usa-se %%eax
- Na lista de registradores afetados, todavia, usa-se %eax apenas
- Lista de saída vazia

Exemplo de asm com referência nominal: melhor para evitar erros

Registradores alocados randomicamente pelo GCC, sem operandos de saída

```
int main (void) {
   int x=1, y=2, z;
   z = x + y;
   asm("addl %[y], %[z] \n\t"          // z += y
     "movl %[y], %%eax \n\t"           // salva y em %eax
     "movl %[x], %[y] \n\t"            // y = x
     "movl %%eax, %[x] \n\t"           // x = %eax
     :                                 // operandos de saída
     : [z]"r"(z), [x]"r"(x) ,[y]"r"(y) // operandos de entrada
     : "%eax");                        // registradores modificados
   printf("%d %d %d\n ", x, y, z);
   return z;}
```

Exemplo de asm com registradores alocados randomicamente: código de montagem

Registradores alocados randomicamente pelo GCC, sem operandos de saída

```
main:
1   pushl  %ebp            // salva %ebp na pilha
2   movl   %esp, %ebp      // inicializa a base do registro de ativacão
3   pushl  %ebx            // salva %ebx (obrigacão da rotina chamada)
4   andl   $-16, %esp      // alinha em fronteira de 16 bytes (desempenho)
5   subl   $32, %esp       // abre 8 espacos na pilha
6   movl   $2, %ebx        // aloca %ebx para y, fazendo y = 2
7   movl   $1, %ecx        // aloca %ecx para x, fazendo x = 1
8   movl   $3, %edx        // aloca %edx para z, fazendo z = x+y = 3
#APP
9   addl   %ebx, %edx      // z = z + y
10  movl   %ebx, %eax      // %eax = y (salva y)
11  movl   %ecx, %ebx      // y = x
12  movl   %eax, %ecx      // x = valor salvo de y
#NO_APP
.
.
.
```

Exemplo de asm com registradores alocados randomicamente: código de montagem

Registradores alocados randomicamente pelo GCC, sem operandos de saída

```
main:...

#APP
9   addl   %ebx, %edx      // z = z + y
10  movl   %ebx, %eax      // %eax = y (salva y)
11  movl   %ecx, %ebx      // y = x
12  movl   %eax, %ecx      // x = valor salvo de y
#NO_APP
13  movl   $3, 16(%esp)    // passa z=3, indica que z não está na saída
14  movl   $2, 12(%esp)    // passa y=2, indica que y não está na saída
15  movl   $1, 8(%esp)     // passa x=1, indica que x não está na saída
16  movl   $.LC0, 4(%esp) // lista de printf
17  movl   $1, (%esp)      // parâmetro de printf
18  call __printf_chk      // chamada de printf
19  movl   $3, %eax        // retorna 3, indica que z não está na saída
20  movl   -4(%ebp), %ebx // restaura valor de %ebx
21  leave                  // prepara o retorno
22  ret                    // retorna
```

Exemplo de asm com referência nominal

Registradores alocados pelo programador e x, y e z listados como saída

```
int main (void) {
   int x=1, y=2, z;
   z = x + y;
   asm("addl %[y], %[z] \n\t"              // z += y
    "movl %[y], %%eax \n\t"                // salva y em %eax
    "movl %[x], %[y] \n\t"                 // y = x
    "movl %%eax, %[x] \n\t"                // x = %eax
    :[z]"=d"(z), [x]"=c"(x), [y]"=b"(y)    // operandos de saida
    :"d"(z),"c"(x),"b"(y)                  // operandos de entrada
    : "%eax");                             // registradores modificados
   printf("%d %d %d\n ", x, y, z);
   return z;}
```

Ao relacionar x, y e z como saída, como o código de montagem é afetado?

Programador quer z em %edx, x em %ecx e y em %ebx

Exemplo de asm com registradores alocados pelo programador: código de montagem

Registradores x, y e z listados como saída

```
main:
1   pushl %ebp              // salva %ebp na pilha
2   movl  %esp, %ebp        // inicializa a base do registro de ativacao
3   pushl %esi              // salva %esi (obrigacao da rotina chamada)
4   pushl %ebx              // salva %ebx (obrigacao da rotina chamada)
5   andl  $-16, %esp        // alinha em fronteira de 16 bytes
6   subl  $32, %esp         // abre 8 espacos na pilha
7   movl  $2, %ebx          // aloca %ebx para y (coincide com programador)
8   movl  $1, %ecx          // aloca %ecx para x (coincide com programador)
9   movl  $3, %esi          // aloca %esi para z (escolha do compilador)
10  movl  %esi, %edx        // copia z para %edx (escolha do programador)
#APP
11  addl  %ebx, %edx        // z = z + y
12  movl  %ebx, %eax        // %eax = y (salva y)
13  movl  %ecx, %ebx        // y = x
13  movl  %eax, %ecx        // x = valor salvo de y
#NO_APP
15  movl  %edx, %esi        // z copiado para %esi
```

Exemplo de asm com registradores alocados pelo programador randomicamente: código de montagem

Registradores x, y e z listados como saída

```
main:
...
#NO_APP
15  movl  %edx, %esi      // copia z para %esi
16  movl  %edx, 16(%esp)  // indica que z está na lista de saída
17  movl  %ebx, 12(%esp)  // indica que y está na lista de saída
18  movl  %ecx, 8(%esp)   // indica que x está na lista de saia
19  movl  $.LC0, 4(%esp)  // lista de printf
20  movl  $1, (%esp)      // parâmetro de printf
21  call __printf_chk     // chamada de printf
22  movl  %esi, %eax      // copia z como retorno da rotina
23  leal  -8(%ebp), %esp  // aponta para posicao onde %ebx foi salvo
24  popl  %ebx            // restaura %ebx
25  popl  %esi            // restaura %esi
26  popl  %ebp            // restaura %ebp
27  ret                   // retorna
```

Inserção de código assembly OK

Pretende: retornar 1 se OK, retornar 0 se multiplicação dá overflow

```
/* Código OK */
int tmult_ok2(int x, int y, int *dest)
  {
      int result;
      *dest = x*y;
      asm("setae %%bl          # seta LSB \n\t"
        mozbl %%bl, %[val]     # estende 0 para resultado"
        : [val] "=r" (result)  /* saida */
        :                      /* sem entradas */
        : "%bl"                /* registrador alterado */
        );
      return result;
  }
```

- Nome `val` escolhido para armazenar o resultado do código e associado à variável `result` no C
- Indica-se que o registrador %bl será sobrescrito

Inserção de código assembly OK

Código assembly para tmult_ok2

```
x em %ebp+8, y em %ebp+12, dest em %ebp+16

1 tmult_ok2:
2   pushl   %ebp
3   movl    %esp, %ebp
4   pushl   %ebx              Salva na pilha, pois é reescrito pelo asm
5   movl    12(%ebp), %eax    Obtém y
6   imull   8(%ebp), %eax     Multiplica por x
7   movl    16(%ebp), %edx    Obtém ponteiro dest
8   movl    %eax, (%edx)      Armazena produto em MEM[dest]
      codigo asm
9   setae   %bl               Seta LSB de %ebx
10  movzbl  %bl, %eax         Estende com 0, associa result (val) a %eax
      fim do codigo asm
11  popl    %ebx              Restaura da pilha
12  popl    %ebp
13  ret
```

Inserção de código assembly OK

Versão usando multiplicação sem sinal umult_ok

```
int umult_ok(unsigned x, unsigned y, unsigned *pdest) {
    unsigned char result;

    asm("movl %[x], %%eax          # Pega x \n\t"
        "mull %[y]                 # Multiplica sem sinal por y\n\t"
        "movl %%eax, %[dest]       # Armazena 4bytes menores em dest\n\t"
        "setae %[b]                # Seta result"
        : [dest] "=r" (*pdest), [b] "=r" (result) /* saida */
        : [x] "r" (x), [y] "r" (y)                 /* entrada */
        : "%eax", "%edx"                           /* reescrito */
        );
    return (int) result;
    }
```

Inserção de código assembly OK

Código para `umult_ok`: x em %ebp+8, y em %ebp+12, pdest em %ebp+16

```
1 umult_ok:
2   pushl   %ebp
3   movl    %esp, %ebp
4   pushl   %ebx                Salva na pilha, pois é reescrito pelo asm
5   movl    12(%ebp), %ebx      Obtem y
6   movl    8(%ebp), %ecx       Obtem x
      codigo asm
7   movl    %ecx, %eax          Copia x
8   mull    %ebx                Multiplica sem sinal por y
9   movl    %eax, %ecx          Copia os 4 bytes menores
10  setae   %bl                 Seta result
      fim do codigo asm
11  movl    16(%ebp), %eax      Obtem pdest
12  movl    %ecx, (%eax)        Armazena produto em dest (saída)
13  movzbl  %bl, %eax           Copia result (saída) para %eax (retorno),
                                estendendo com zero
14  popl    %ebx                Restaura da pilha
15  popl    %ebp
16  ret
```

Exemplo de código asm com troca de valores

```
void main(void) {
   int x = 10, y;
   asm ("movl %1, %0"
        :"=d"(y) // y é op de saída
        :"c"(x)  // x é op de entrada
       );
}
```

```
main: pushl   %ebp
      movl    %esp, %ebp
      subl    $16, %esp
      movl    $10, -4(%ebp)
      movl    -4(%ebp), %eax
      movl    %eax, %ecx
#APP
# 3 "ex36.c" 1
      movl %ecx, %edx
# 0 "" 2
#NO_APP
      movl    %edx, %eax
      movl    %eax, -8(%ebp)
      leave
      ret
```

Outros exemplos

(*Mostrar programas em anexo*)

Referências bibliográficas

- CS:APP2e Web Aside ASM:EASM: *Combining Assembly Code with C Programs*, R.E.Bryant e D.R.O'Hallaron, 2010.
- http://asm.sourceforge.net