Arquitetura de Computadores

Construindo programas com funções Assembly e C

Luís Nogueira

Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

lmn@isep.ipp.pt

2023/2024

Construindo programas com funções Assembly e C

- Freqüentemente, as rotinas de tempo crítico são escritas em Assembly e o restante do software é escrito em C, portanto, um projeto “misto de Assembly e C” deve ser criado
- Quando você pretende misturar arquivos de origem Assembly e arquivos de origem ANSI-C em um único aplicativo, os seguintes problemas são importantes:
  - Acessando variáveis Assembly em um arquivo fonte ANSI-C
  - Acessando variáveis ANSI-C em um arquivo fonte Assembly
  - Invocando uma função Assembly em um arquivo fonte ANSI-C
  - Esquema de passagem de parâmetros
  - Valor de retorno

Construindo programas com funções Assembly e C

- Iremos (por enquanto) escrever funções em Assembly que não recebem parâmetros e acessam variáveis globais (declaradas em C ou Assembly)
- Nossos programas C chamarão nossas funções Assembly como se fossem funções C nativas
- Para fazer com que nossas funções Assembly retornem:
  - até um valor de 64 bits, deixe esse valor de retorno em%raxregistre (ou partes dele) um
  - valor de 128 bits, deixe o valor de retorno em%rdx:% raxregistros

Nota importante

GCC em execução em x86-64 suporta valores inteiros assinados e não assinados de 128 bits por meio de tipos de dados __int128_te __uint128_t,respectivamente

Compartilhando variáveis entre Assembly e C (1/5)

- Para compartilhar variáveis globais que são declaradas em Assembly, entre C e Assembly, use oexternoPalavra-chave C
- Ele declara ao compilador que uma variável está definida (a memória está reservada) em outro arquivo fonte (no nosso caso, no(s) arquivo(s) fonte(s) do Assembly)

Nota importante

OexternoA palavra-chave C pode ser usada de diferentes maneiras para compartilhar variáveis entre C e Assembly. A seguir está uma prática recomendada, que evita problemas comuns

- Na fonte C:
  1. Declare o*funções e variáveis*implementado em Assembly e usado em C em um arquivo .h-file (muitas vezes chamadoasm.h).Declare essas variáveis Assembly usando o externoPalavra-chave C (funções sãoexternopor padrão):

Listagem 1: asm.h

```
interno asm_funct em();
externo asm_i inteiro;
```

  2. Use a palavra-chave #incluirpara incluir o anterior.h-fiarquivo nos arquivos de origem C (.c-fi arquivos) que usam as funções ou variáveis do Assembly e usam as funções/variáveis do Assembly como funções/variáveis C nativas:

Listagem 2: main.c

```
# incluir "asm. h"
 . . .
interno principal ( ) {
    . . .
    asm_i inteiro r =10;
    asm_funct em(); . . .

}
```

- Na fonte Assembly:
  3. Declare as variáveis e funções usadas pelas fontes C e defina-as como visíveis usando o arquivo .globaldiretiva

Listagem 3: asm.s

```
. seção .dados
asm_inteiro:                    # variável        declaração
        . interno5
. globalasm_inteiro             #define a variável como global

. seção .texto
. globalfunção_asm              #define a função como global
função_asm:                     #início da função
        . . .
        movimentoUS$ 0, %eax    # chegando aqui retornará 0
                                # (rax não será alterado até ret)
        ret
```

Compartilhando variáveis entre Assembly e C (4/5)

1. Para compartilhar variáveis globais que são declaradas em C, entre C e Assembly, basta declará-las como variáveis globais no arquivo fonte C

Listagem 4: main.c

```
# incluir          <stdio. h>
# incluir           "asm. h"

 internoop1=0, op2=0;

 internoprincipal (vazio)     {
      interno     res;
      . . .
      resolução   =    soma ( ) ;
      . . .
      retornar     0;
}
```

2. Na origem Assembly, defina-os como visíveis usando a extensão .globaldiretiva
3. Acesse-os com endereçamento relativo ao% rip

Listagem 5: asm.s

```
. seção .dados
      . global      op1
      . global      op2

. seção .texto
      . globalsoma
      soma:
          . . .
          movimentoop1(%rasgar), %      #copia o valor de op1 para ecx
          ecx movlop2(%rasgar),%eax     #copia o valor de op2 para eax
          . . .
          ret
```

A interface binária do aplicativo x86-64

- A interface binária de aplicativo (ABI) x86-64 descreve as convenções para código x86-64 em execução em sistemas Linux
- Isso inclui regras sobre como os argumentos da função são colocados, para onde vão os valores de retorno, quais registros as funções podem usar, como elas podem alocar variáveis locais e assim por diante.
- As convenções de chamada restringem ambos *chamadores* e *chamados*. Um chamador é uma função que chama outra função; um receptor é uma função que foi chamada (a função atualmente em execução é um receptor)
- Discutiremos vários detalhes ao longo do semestre, mas por enquanto basta considerar o valor de retorno e quais registros podem ser usados livremente por uma função

Nota importante sobre como escrever funções Assembly

- Até detalharmos o uso da pilha *NÃO* use qualquer um desses *receptor salvo* registra em suas funções:% rbx,% rbp,% r12,% r13,% r14,% r15
- Isto é particularmente importante se você chamar suas funções do código Assembly ou C de outro programador (por exemplo, testes de unidade)

Exemplo: Somar duas variáveis – fonte C

Listagem 6: main.c

```
# incluir<stdio.h>
# incluir"asm.h"//define op1 , op2 e sum_op1_op2(void)

interno  principal(vazio) {
         longores =0;
         printf("Valor de          op1?:");
         scanf("%d",&op1);
         printf("Valor de          op1?:");
         scanf("%hd",&op2);

         /* res = op1 + op2; */
         res=soma_op1_op2();

         printf("%ld = %d + %d\n", res, op1, op2); retornar0;

}
```

Listagem 7: asm.h

```
soma longa_op1_op2(vazio);
externo   internoop1;
externo   curtoop2;
```

Exemplo: Somar duas variáveis – Fonte de montagem

Listagem 8: asm.s

```
. seção .dados
    operação1:                          #declarar      op1,    op2
            . interno0
    operação2:
            . curto     0
    . globalop1, op2                    #define op1 e op2 como globais

. seção .texto

    . globalsoma_op1_op2# define a função global long sum_op1_op2(void)

soma_op1_op2:
        movimento op1(%rasgar), %ecx        # coloque op1 em ecx
        movslq%ecx,%movimento rcx           # sinal estendido para palavra quádrupla
                    op2(%rasgar),%          # coloque op2 em rax
        machado movswq%machado,%rax         # sinal estendido para palavra quádrupla
        addq        %RCX,%rax               # adiciona rcx ao rax, o resultado está em rax
                                            # e será nosso valor de retorno
        ret                                 # retorna para a função chamadora
```

Exemplo: Somar duas variáveis - Makefile

Listagem 9: Makefile

```
principal: principal. ó  asm. ó
        gcc  principal . ó  asm. o −znoexecstack −o principal

principal . ó:  principal . c  asm. h
        gcc-g  − Parede l  − Wextra −fanalyzer −c principal . c −o principal. ó

asm. o: asm. é
        gcc −g −Wal l −Wextra −fanalyzer −c asm . s −o asm . ó

executar: principal
        . / principal

limpar :
        rm ∗ . o principal
```

Nota importante

O sinalizador do vinculador -z noexecstackké necessário em versões recentes do gcc para silenciar um aviso de pilha executável

Prática

- Escreva um programa em C que chameincremento(),uma função implementada em Assembly
- Funçãoincremento()incrementa o valor da variável inteira globalnúmero_g e retorna este valor (após o incremento)
- O programa C deve atribuir um valor de teste paranúmero_g,chamarincremento()e depois imprima ambosnúmero_ge o valor retornado pela função
- Escreva umMakefilepara compilar seu programa