Inline Assembly em Linguagem C

Mateus Maruzka Roncaglio

Programa de Pós-graduação em Ciência da Computação
Universidade Estadual do Oeste do Paraná - UNIOESTE

March 30, 2022

Índice

Índice

Inline Assembly: O que é?

Inline Assembly

- É uma extensão do compilador que permite utilizar Assembly e linguagem C dentro do mesmo código, sem a necessidade de outros métodos
- É um recurso nativo ao compilador *GCC*
  - o compilador *Clang* possui sintaxe compatível com a do *GCC*
- Torna o programa dependente do processador, uma vez que diferentes processadores podem ter diferentes instruções assembly;
  - Entretanto, Intel e AMD trabalham em conjunto

PPGComp

Por que e quando utilizar?

- Tempo de execução pode ser menor (Apesar de ser difícil de competir com o compilador)
  - Em especial, funções matemáticas, que geralmente são custosas computacionalmente;
- Necessidade de manusear algum *hardware* específico;
- Controlar o conteúdo dos registradores

Sintaxe

Para utilizar código assembly na linguagem C é necessário utilizar a palavra-chave `asm`:

```
// asm pode ser substituído por __asm__
asm [qualificadores] ("String contendo o código assembly");

// Os qualificadores podem ser volatile e/ou inline, ou também pode ser omitido
```

Exemplo 1:

```
asm(
"MEU_INLINE_ASSEMBLY:\n\t"
"mov rax, 0x5\n\t"
)
```

Sintaxe - Inline Assembly Básico

Exemplo 1:

- Execute o seguinte comando:
  - gcc intro.c -o intro.asm -S -masm=intel -fno-asynchronous-unwind-tables
  - Aqui vamos gerar um código assembly utilizando o compilador *gcc*.
- Observe que comentários são precedidos por `\#` e não por ;
- Observe que as instruções assembly foram colocadas literalmente no arquivo .asm entre #APP e #NOAPP
- Para separar as instruções pode-se utilizar ; ou `\n\t`

Para criar o executável utilize: `gcc intro.c -o intro -masm=intel`

A diretriz `-masm=intel` diz ao compilador que estamos utilizando a sintaxe intel

Índice

Sintaxe - Inline Assembly Estendido

- Até o momento apenas inserimos um código assembly, mas sua interligação com o restante do código C é difícil ou até mesmo impossível.
- Para contornar isso, há o *Inline Assembly estendido*

Inline Assembly Estendido

```
asm qualificadores (
  "instruções assembly"
  : operandos de saida
  : operandos de entrada
  : clobbers
  : rótulos/labels para goto
)
```

Qualificadores:

- *volatile*
- *inline*
- *goto*

Função

Desabilita a otimização de código

Inline Assembly Estendido

```
asm qualificadores (
  "instruções assembly"
  : operandos de saida
  : operandos de entrada
  : clobbers
  : rótulos/labels para goto
)
```

Qualificadores:

- *volatile*
- *inline*
- *goto*

Função

Indica ao compilador para expandir a função como *inline*

Inline Assembly Estendido

```
asm qualificadores (
  "instruções assembly"
  : operandos de saida
  : operandos de entrada
  : clobbers
  : rótulos/labels para goto
)
```

Qualificadores:

- *volatile*
- *inline*
- *goto*

Função

Indica que o assembly pode realizar um salto para uma das *labels* indicadas. Quando usa-se o goto, não é possível utilizar operandos de saída. A recomendação é passar como operando de entrada e passar clobber "memory".

Inline Assembly Estendido

```
asm qualificadores (
  "instruções assembly"
  : operandos de saida
  : operandos de entrada
  : clobbers
  : rótulos/labels para goto
)
```

Operandos de Saída:

Função

Os operandos de saída especificam as variáveis que podem ser lidas e/ou alteradas pelo código assembly. São os únicos operandos que são obrigatórios no assembly estendido

Sintaxe

: [nome] "parâmetros(leitura, escrita, armazenamento)" (myVar)

• [nome] $\rightarrow$ É um apelido para *myVar*, que foi definida no código C, e que será utilizada no assembly (é opcional). A mesma variável, pode ser referenciada utilizando %*n*, em que *n* é a posição que o operando foi colocado. O primeiro operando é referenciado com %0, o segundo com %1 e assim por diante (Essa regra leva em conta a posição dos operandos de entrada, de saída e dos rótulos/labels e são limitados à no máximo 30 operandos no total). Para referenciar utilizando o apelido usa-se %[*nome*]

Inline Assembly Estendido

```
asm qualificadores (
  "instruções assembly"
  : operandos de saida
  : operandos de entrada
  : clobbers
  : rótulos/labels para goto
)
```

Operandos de Saída:

Função

Os operandos de saída especificam as variáveis que podem ser lidas e/ou alteradas pelo código assembly. São os únicos operandos que são obrigatórios no assembly estendido

Sintaxe

: [nome] "parâmetros(leitura, escrita, armazenamento)" (myVar)

• "parâmetros..." → Indicam onde o operando deve ser armazenado e se ele será apenas lido, modificado ou ambos. A *string* deve sempre começar com as permissões de acesso e em seguida o armazenamento.

- Acesso: '=' indica que o valor será modificado e '+' indica que o valor será modificado e lido.
- Armazenamento: "r" indica que será armazenado em um registrador de propósito geral "m" indica que será armazenado na memória "a", "b", "c" e "d" indicam os registradores RAX, RBX, RCX e RDX

PPGComp

Inline Assembly Estendido

```
asm qualificadores (
  "instruções assembly"
  : operandos de saida
  : operandos de entrada
  : clobbers
  : rótulos/labels para goto
)
```

Operandos de Saída:

Função

Os operandos de saída especificam as variáveis que podem ser lidas e/ou alteradas pelo código assembly. São os únicos operandos que são obrigatórios no assembly estendido

Sintaxe

: [nome] "parâmetros(leitura, escrita, armazenamento)" (myVar)

- A variável *myVar* é a variável, que foi declarada em C. Ela deve ser posta entre parênteses.

Inline Assembly Estendido

```
asm qualificadores (
  "instruções assembly"
  : operandos de saida
  : operandos de entrada
  : clobbers
  : rótulos/labels para goto
)
```

Operandos de Entrada:

Função

Operandos de entrada especificam quais variáveis serão apenas lidas (apesar de ser possível modifica-las, mas isso não é recomendado)

Sintaxe

: [nome] "parâmetros(leitura, escrita, armazenamento)" (myVar)

- A sintaxe é a mesma dos operandos de saída e segue as mesmas regras, com exceção dos parâmetros, que não deve ser passados os parâmetros de acesso ("=", "+").

PPGComp

Inline Assembly Estendido

```
asm qualificadores (
  "instruções assembly"
  : operandos de saida
  : operandos de entrada
  : clobbers
  : rótulos/labels para goto
)
```

Clobbers

Função

É utilizado para indicar ao compilador alterações indiretas que o código assembly pode, ou irá realizar. Por exemplo, ao realizar uma soma o registrador EEFLAGS é alterado

Sintaxe

: "parametro-1", "parametro-2", ... , "parametro-N"
Cada parâmetro é uma string contendo o alvo das alterações.
Por exemplo: "m", "rax"

Parâmetros Clobbers

Parâmetro	Descrição
cc	Indica que as flags serão modificadas pelo assembly
m	Indica que alguma posição da memória será lida/escrita e que não está listada nos operandos de entrada e saída. Exemplo: O valor de um ponteiro passado nos operandos de saída.
rax	Indica que o registrador rax será modificado
rbx	Indica que o registrador rbx será modificado
A mesma regra vale para os demais registradores, com excessão ao Stack Pointer (RSP)	

Inline Assembly Estendido

```
asm qualificadores (
  "instruções assembly"
  : operandos de saida
  : operandos de entrada
  : clobbers
  : rótulos/labels para goto
)
```

Rótulos para goto

Função

É utilizado para indicar ao compilador qual rótulo, definido no código em linguagem C. Ele é utilizado em conjunto com qualificador goto.

Sintaxe

Os rótulos que o código assembly pode realizar os saltos são colocados, separados por vírgula. Não são colocados entre aspas (" ").
Exemplo: : label0, label1 , ... ,labelN
Para referenciar o rótulo no código ASM é feita utilizando %*l*[*label*0] ou %ln, em que n é a posição em que a label foi colocada na lista de rótulos, levando em conta também os operadores de entrada e de saída.

Exemplo - Rótulos GOTO

```
int par(int value) {
  asm goto( "mov eax, %[a];"
            "and eax, 0x1;"
            "jz %[par];"
            "jmp %[impar];"
           :/*Sem operandos de saída */: [res] "+m" (result)
           : "cc", "memory"
           : par, impar);
par:
    return 1;
impar:
    return 0;
}
```

Definindo um registrador à uma variável

Quando utilizamos a *keyword* `register` podemos definir uma variável e escolher qual registrador ela será armazenada.

```
register int myVar asm("r12") = 5;
```

Com a *keyword* `static` podemos definir o nome da variável no assembly.

```
static int myVar asm("x") = 5;
```

Índice

Exercícios Propostos

- Escreva uma função que calcule 2^N (Dica: utilize o deslocamento de bits)
- Escreva uma função que retorne se o número é par ou não, apenas utilizando operadores bit a bit.
- Escreva uma que retorne se o numero é par ou não, utilizando a instrução IDIV
- Escreva uma função que calcule o n-ésimo número da função de fibonacci. Lembre-se que $F_n = F_{n-1} + F_{n-2}$