MATA49
Programação de Software Básico

Leandro Andrade
leandrojsa<at>dcc.ufba.br

Assembly e C

Assembly e C

- Atualmente poucos programas são escritos completamente em assembly
- Compiladores são bons na conversão de uma linguagem de alto nível em código de máquina eficiente
- Linguagens de alto nível são muito mais portáveis
  - E populares!

Assembly e C

- O uso de assembly, na prática, normalmente resume-se a pequenos blocos de código
  - Isso pode ser feito de duas maneiras:
    - Chamadas assembly de sub-rotinas em C
    - Código assembly embutido em programas em C (Assembly inline)

Assembly e C

- Chamadas de sub-rotinas em C:
  - A maioria dos compiladores C utilizam os nomes as funções com “_”
    - Ex: printf → _printf
  - Conserva os valores dos registradores de uso interno

Assembly e C

- Passagem de parâmetros segue o “C calling convention”
  - Os parâmetros são empilhados na ordem inversa da ordem na função em C
  - Exemplo:

EBP + 12	value of x
EBP + 8	address of format string
EBP + 4	Return address
EBP	saved EBP

Assembly e C

- Valores de retorno das funções
  - 32 bits em EAX
  - 64 bits EAX:EDX
  - Valores float STO

Assembly e C

```
1  segment .data
2  format        db "%d", 0
3
4  segment .text
5  ...
6        lea     eax, [ebp-16]
7        push    eax
8        push    dword format
9        call    _scanf
10       add     esp, 8
11 ...
```

Assembly e C

- Assembly inline
  - O compilador GCC para Linux usa a sintaxe AT&T/UNIX
    - Similar a usada para Windows
  - Possui algumas diferenças em relação ao assembly da sintaxe Intel

Assembly e C

- Assembly inline:
  - Alguns problemas podem ser resolvidos mais eficientemente com códigos assembly
  - Exemplo:

```
asm ("bsrl %1, %0" : "=r" (position) : "r" (number));
```

One way you could implement the same operation in C is using this loop:

```
long i;
for (i = (number >> 1), position = 0; i != 0; ++position)
  i >>= 1;
```

Assembly e C

- Características da sintaxe AT&T:
  - Ordem de operandos “fonte, destino”
  - Nome dos registradores possui como prefiro “%”
    - %eax, %esi, %ebp...
  - Operandos imediatos são precedidos por $
    - Ex: valor 1 → $1 ; valor 10h → $10h
  - Tamanho dos operandos é definido pelo último caractere da instrução
    - 'b' → 8-bit ; 'w' → 16-bit ; 'l' → 32-bit
    - Ex: movb foo, %al

Assembly e C

- Características da sintaxe AT&T:
  - Acesso a memória é feito por parêntesis
    - Exemplo: [esi] → (esi)

      [base + index*scale + disp] → (base, index, scale)

Assembly e C

Intel Code	AT&T Code
mov eax,1	movl $1,%eax
mov ebx,0ffh	movl $0xff,%ebx
int 80h	int $0x80
mov ebx, eax	movl %eax, %ebx
mov eax,[ecx]	movl (%ecx),%eax
mov eax,[ebx+3]	movl 3(%ebx),%eax
mov eax,[ebx+20h]	movl 0x20(%ebx),%eax
add eax,[ebx+ecx*2h]	addl (%ebx,%ecx,0x2),%eax
lea eax,[ebx+ecx]	leal (%ebx,%ecx),%eax
sub eax,[ebx+ecx*4h-20h]	subl -0x20(%ebx,%ecx,0x4),%eax

Assembly e C

- Para executar uma instrução de assembly em C deve usar o comando:
  - asm(“<codigo assembly>”); ou __asm__(“<codigo assembly>”);
  - Exemplo:

```
asm("movl %ecx, %eax");
/* moves the contents of ecx to eax */
```

    - __asm__("movb %bh, (%eax)");

```
/*moves the byte from bh to the memory pointed by eax */
```

Assembly e C

- Asm estendido
  - Permite especificar os operandos
  - Estrutura:

```
asm ( assembler template
   : output operands                  /* optional
*/      : input operands                   /* optional */
      : list of clobbered registers      /* optional */
   ;
```

Assembly e C

- Asm estendido:
  - Exemplo:

```
int a=10, b;
asm ("movl %1, %%eax;
      movl %%eax, %0;"
     :"=r"(b)          /* output */
     :"r"(a)           /* input */
     :"%eax"           /* clobbered register */
     );
```

Assembly e C

- Asm estendido:
  - Exemplo:
    - 'b' é o operando de saída e é referido por %0
    - 'a' é o operando de entrada e é referido por %1
    - =r e r referem se para o gcc usar registradores para armazenar os operandos
    - %% é para ajudar o gcc a distinguir a diferença entre operandos e registradores
    - :"%eax" indica a modificação feita no registrador será interna ao asm, não se estendendo ao GCC

Assembly e C

- Assembly inline
  - Referências:
    - http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html#s4
    - http://www.advancedlinuxprogramming.com/alp-folder/alp-ch09-inline-asm.pdf
    - http://www.ibm.com/developerworks/library/l-ia/index.html