Assembly do IA-32 em ambiente Linux

Exercícios de Programação 1

(Adaptados do livro de Randal E. Bryant and David R. O'Hallaron)
Alberto J. Proença e António M. Pina

Prazos

Entrega **impreterível**: segunda 19-Nov-01, às 11h00, no Lab. da disciplina.

Não serão aceites trabalhos entregues depois deste prazo.

Metodologia

O material a entregar nesta semana é similar ao das semanas que se seguirão, e constará essencialmente da(s) folha(s) anexa(s) ao enunciado, preenchidas à mão (manuscrito), excepto quando indicado em contrário. A validação do conteúdo será feita nas sessões teórico-práticas da semana em que forem entregues os trabalhos.

A penalização por fraude será a atribuição de classificação com valor <0.

Introdução

A lista de exercícios que se apresenta segue directamente o material apresentado nas aulas teóricas e práticas da semana 7 (ver sumários na página da disciplina na Web), requerendo os conceitos básicos adquiridos em aulas anteriores.

Exercício 1.1 *(Acesso a operandos)*:

Considere que os seguintes valores estão armazenados em registos e em endereços de memória:

Endereço	Valor
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Registo	Valor
%eax	0x100
%ecx	0x1
%edx	0x3

Preencha a seguinte tabela mostrando os valores para os operandos indicados:

Operando	Valor
%eax	
0x104	
$0x108	
(%eax)	
4(%eax)	
9(%eax,%edx)	
260(%ecx,%edx)	
0xFC(,%ecx,4)	
(%eax,%edx,4)	

Exercício 1.2 *(Transferência de informação em funções)***:**

Considere que a seguinte função, cuja assinatura (*prototype*) vem dada por

```
void decode1(int *xp, int *yp, int *zp);
```

é compilada para o nível do *assembly*. O corpo da função fica assim codificado:

```
1    movl   8(%ebp),%edi
2    movl   12(%ebp),%ebx
3    movl   16(%ebp),%esi
4    movl   (%edi),%eax
5    movl   (%ebx),%edx
6    movl   (%esi),%ecx
7    movl   %eax,(%ebx)
8    movl   %edx,(%esi)
9    movl   %ecx,(%edi)
```

Os parâmetros `xp`, `yp`, e `zp` estão armazenados nas posições de memória com um deslocamento de 8, 12, e 16, respectivamente, relativo ao endereço no registo `%ebp`.

Escreva código C para `decode1` que tenha um efeito equivalente ao programa em *assembly* apresentado em cima. Pode verificar a sua resposta compilando com o *switch* –S. O compilador que usar poderá eventualmente gerar código com uma utilização diferente dos registos ou de ordenação das referências à memória, mas deverá ser funcionalmente equivalente.

Exercício 1.3 *(Load effective address)***:**

Suponha que o registo `%eax` contém o valor *x* e que `%ecx` contém o valor *y*. Preencha a tabela seguinte, com expressões (fórmulas) que indiquem o valor que será armazenado no registo `%edx` para cada uma das seguintes instruções em *assembly*:

Instrução	Valor
`leal  6(%eax), %edx`	
`leal  (%eax,%ecx), %edx`	
`leal  (%eax,%ecx,4), %edx`	
`leal  7(%eax,%eax,8), %edx`	
`leal  0xA(,%ecx,4), %edx`	
`leal  9(%eax,%ecx,2), %edx`	
`leal  (%eax,%ecx,4), %edx`	

Exercício 1.4 *(Operações aritméticas)***:**

Considere que os seguintes valores estão armazenados em registos e em endereços de memória:

Endereço	Valor
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Registo	Valor
%eax	0x100
%ecx	0x1
%edx	0x3

Preencha a seguinte tabela mostrando os efeitos das instruções seguintes, quer em termos de localização dos resultados (em registo ou endereço de memória), quer dos respectivos valores:

Instrução	Destino	Valor
`addl %ecx,(%eax)`		
`subl %edx,4(%eax)`		
`imull $16,(%eax,%edx,4)`		
`incl 8(%eax)`		
`decl %ecx`		
`subl %edx,%eax`		

Exercício 1.5 *(Operações de deslocamento)*:

Suponha que se pretende gerar código *assembly* para a seguinte função C:

```
int shift_left2_rightn(int x, int n)
{
   x <<= 2;
   x >>= n;
   return x;
}
```

Apresenta-se de seguida uma porção do código assembly que efectua as operações de deslocamento e deixa o valor final em `%eax`. Duas instruções chave foram retiradas. O parâmetros `x` e `n` estão armazenados nas posições de memória com um deslocamento relativo ao endereço no registo `%ebp` de, respectivamente, 8 e 12 células.

```
1    movl    8(%ebp),%ecx          Get x
2    movl    12(%ebp),%eax         Get n
3    _______________               x <<= 2
4    _______________               x >>= n
```

Complete o programa com as instruções em falta, de acordo com os comentários à direita. O *right shift* deverá ser realizado aritmeticamente.

Exercício 1.6 *(Operações aritméticas/lógicas)*:

Na compilação do seguinte ciclo:

```
for (i = 0; i < n; i++)
   v += i;
```

encontrou-se a seguinte linha de código *assembly*:

```
xorl  %edx,%edx
```

Explique a presença desta instrução, sabendo que não há operadores de OU-EXCLUSIVO no código C. Que operação do programa em C, implementa esta instrução *assembly*?

Exercício 1.7 *(Comparações)*:

No código C a seguir, substituiu-se alguns dos operadores de comparação por "__" e retiraram-se os tipos de dados nas conversões de tipo (*cast*).

```
1 char ctest(int a, int b, int c)
2 {
3    char t1 = a __ b;
```

```
4    char t2 = b __ ( ) a;
5    char t3 = ( ) c __ ( ) a;
6    char t4 = ( ) a __ ( ) c;
7    char t5 = c __ b;
8    char t6 = a __ 0;
9    return t1 + t2 + t3 + t4 + t5 + t6;
10 }
```

A partir do código original em C, o GCC gera o seguinte código *assembly*:

```
1    movl   8(%ebp),%ecx          Get a
2    movl   12(%ebp),%esi         Get b
3    cmpl   %esi,%ecx             Compare a:b
4    setl   %al                   Compute t1
5    cmpl   %ecx,%esi             Compare b:a
6    setb   -1(%ebp)              Compute t2
7    cmpw   %cx,16(%ebp)          Compare c:a
8    setge  -2(%ebp)              Compute t3
9    movb   %cl,%dl
10   cmpb   16(%ebp),%dl          Compare a:c
11   setne  %bl                   Compute t4
12   cmpl   %esi,16(%ebp)         Compare c:b
13   setg   -3(%ebp)              Compute t5
14   testl  %ecx,%ecx             Test a
15   setg   %dl                   Compute t4
16   addb   -1(%ebp),%al          Add t2 to t1
17   addb   -2(%ebp),%al          Add t3 to t1
18   addb   %bl,%al               Add t4 to t1
19   addb   -3(%ebp),%al          Add t5 to t1
20   addb   %dl,%al               Add t6 to t1
21   movsbl %al,%eax              Convert sum from char to int
```

Baseado neste programa em *assembly*, preencha as partes em falta (as comparações e as conversões de tipo) no código C.

Exercício 1.8 *(Controlo de fluxo)*:

Nos seguintes excertos de programas desmontados do binário (*disassembled binary*), alguns ítens de informação foram substituidos por `X`'s. Responda às seguintes questões.

a) Qual o endereço destino especificado na instrução `jbe`?

```
8048d1c: 76 da                jbe XXXXXXX
8048d1e: eb 24                jmp 8048d44
```

b) Qual o endereço em que se encontra o início da instrução `mov`?

```
XXXXXXX: eb 54                jmp 8048d44
XXXXXXX: c7 45 f8 10 00       mov $0x10,0xfffffff8(%ebp)
```

c) Nesta alínea, o endereço da instrução de salto é especificado no modo relativo ao IP/PC, em 4 *bytes*, codificado em complemento para 2. Os *bytes* são apresentados do menos para o mais significativo, de acordo com o modelo de representação *little endian*, característico do IA-32. Qual o endereço especificado na instrução `jmp`?

```
8048902: e9 cb 00 00 00       jmp XXXXXXX
8048907: 90 nop
```

d) Explique a relação que existe entre a anotação à direita, e a codificação dos *bytes* à esquerda. Ambas linhas fazem parte da codificação da instrução `jmp`?

```
80483f0: ff 25 e0 a2 04       jmp *0x804a2e0
80483f5: 08
```

Nº _____ Nome _____

Exercício 1.1 *(Acesso a operandos)***:**

Operando	Valor
%eax	
0x104	
$0x108	
(%eax)	
4(%eax)	
9(%eax,%edx)	
260(%ecx,%edx)	
0xFC(,%ecx,4)	
(%eax,%edx,4)	

Exercício 1.2 *(Transferência de informação em funções)***:**

Exercício 1.3 *(Load effective address)***:**

Instrução	Valor
leal 6(%eax), %edx	
leal (%eax,%ecx), %edx	
leal (%eax,%ecx,4), %edx	
leal 7(%eax,%eax,8), %edx	
leal 0xA(,%ecx,4), %edx	
leal 9(%eax,%ecx,2), %edx	
leal (%eax,%ecx,4), %edx	

Exercício 1.4 *(Operações aritméticas)***:**

Instrução	Destino	Valor
addl %ecx,(%eax)		
subl %edx,4(%eax)		
imull $16,(%eax,%edx,4)		
incl 8(%eax)		
decl %ecx		
subl %edx,%eax		

Nº _____ Nome _____

Exercício 1.5 *(Operações de deslocamento)***:**

```
3    _____________________________    x <<= 2
4    _____________________________    x >>= n
```

Exercício 1.6 *(Operações aritméticas/lógicas)***:**

Exercício 1.7 *(Comparações)***:**

```
1 char ctest(int a, int b, int c)
2 {
3    char t1 = a __ b;
4    char t2 = b __ (    ) a;
5    char t3 = (    ) c __ (    ) a;
6    char t4 = (    ) a __ (    ) c;
7    char t5 = c __ b;
8    char t6 = a __ 0;
9    return t1 + t2 + t3 + t4 + t5 + t6;
10 }
```

Exercício 1.8 *(Controlo de fluxo)***:**

a) `8048d1c: 76 da jbe XXXXXXX` _____

b) `XXXXXXX: c7 45 f8 10 00 mov…` _____

c) `8048902: e9 cb 00 00 00 jmp XXXXXXX` _____

d)