Programação em IA32
(Semana-8)

Exercícios adaptados do livro CSPP
Randal E. Bryant e David R. O'Hallaron

A lista de exercícios que se apresenta segue directamente o material apresentado nas aulas teóricas e práticas da semana 6 (ver sumários na página da disciplina na Web), requerendo os conceitos básicos adquiridos em aulas anteriores. O objectivo é apresentar aos alunos um conjunto de exercícios que sustentem a compreensão dos princípios gerais de programação de processadores convencionais. A linguagem de montagem usa um conjunto restrito de instruções ao nível máquina baseado no IA32.

Os exercícios deverão ser resolvidos em grupos de 2/3 pessoas e a resolução condensada numa única folha fornecida para o efeito a entregar no final da aula.

Exercício 1 *(Acesso a operandos)***:**
Considere os seguintes quadros de valores:

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

Registo IA32	Valor
%eax	0x100
%ecx	0x1
%edx	0x3

Preencha a tabela de valores para as entradas correspondentes:

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

Exercício 2 *(Transferência de informação em funções)***:**

Considere o corpo de função para a qual é válida a assinatura (*prototype*) e a respectiva representação em linguagem de montagem, que segue. De acordo com a convenção adoptada, pelo GCC, para a representação de uma função na memória, os parâmetros de entrada `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 o código C que supostamente estará na origem daquela função `decode1`.

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

```
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)
```

Números de alunos: 1ª _____ 2ª _____ 3ª _____ Turno ___

Exercício 3 *(Load effective address)***:**

Supondo que o registo `%eax` e `%ecx` contém, respectivamente, os valores *x* e *y*, preencha os valores na tabela seguinte, com expressões (fórmulas) que indiquem o valor do registo no destino (`%edx`)

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 4 *(Operações aritméticas)***:**

Considere os seguintes quadros de valores:

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, 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 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 em seguida uma porção do código que efectua 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),%eax   ; Get x
2    movl    12(%ebp),%ecx  ; 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* é aritmético.

Exercício 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?

Exercício 7 *(Comparações)***:**

Nas alíneas seguintes, substituíram-se alguns dos operadores de comparação e declarações de tipos de dados por "__". A partir da listagem, abaixo, produzida pelo GCC, preencha com os operadores ou declarações em falta no texto fonte em C.

a)

```
char ctest(int a, int b) {
       char t1 = a    b;
       return t1;
}
```

```
movl    12(%ebp), %eax
cmpl    %eax, 8(%ebp)
setl    %al
movsbl  %al,%eax
```

b)

```
char ctest(  a, int b) {
       char t1 = a    b;
       return t1;
}
```

```
movl    12(%ebp), %eax
cmpl    %eax, 8(%ebp)
setb    %al
movsbl  %al,%eax
```

c)

```
char ctest(int a) {
       char t1 = a   0;
       return t1;
}
```

```
movl    8(%ebp), %eax
testl   %eax, %eax
setg    %al
movsbl  %al,%eax
```

d)

```
char ctest (int a, int b) {
       char t1 = a   b;
       char t2 = a _ b;
       char t3 = t1 + t2;
       return t3;
}
```

```
movl    12(%ebp), %eax
movl    8(%ebp), %ecx
cmpl    %eax, %ecx
setne   %dl
cmpl    %eax, %ecx
setge   %al
addb    %al, %dl
movsbl  %dl,%eax
```

Exercício 8 *(Controlo de fluxo)*:

Nos seguintes excertos de programas desmontados, alguns itens de informação foram substituídos 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. Qual o endereço especificado na instrução `jmp`?

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

d) Explique a incongruência do seguinte código desmontado.

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

Números de alunos: 1ª _____ 2ª _____ 3ª _____ Turno ___

Exercício 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 2 *(Transferência de informação em funções)***:**

Exercício 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 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		

Exercício 5 *(Operações de deslocamento)***:**

```
3     _____________________________     x <<= 2

4     _____________________________     x >>= n
```

Exercício 6 *(Operações aritméticas/lógicas)***:**

Exercício 7 *(Comparações)***:**

a)

```
char ctest(int a, int b) {
       char t1 = a    b;
       return t1;
}
```

b)

```
char ctest(__a, int b) {
       char t1 = a    b;
       return t1;
}
```

c)

```
char ctest(int a) {
       char t1 = a   0;
       return t1;
}
```

d)

```
char ctest (int a, int b) {
       char t1 = a   b;
       char t2 = a _ b;
       char t3 = t1 + t2;
       return t3;
}
```

Exercício 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)