

PUC-Rio – Software Básico – INF1612
Prova 1 – 29/09/05

1. (2,5 pontos) Considere a função `unpack` pedida no primeiro trabalho:

```
void unpack (char* formatos, char *buffer, long numeros[]);
```

relembrando: A função `unpack` realiza a descompactação do buffer recebido, preenchendo o array `numeros` com os números extraídos do `buffer`. O primeiro caracter da cadeia `formatos` indica se os números estão armazenados no buffer como *little* ou *big-endian*, e os demais indicam o tamanho de cada inteiro no buffer e se ele corresponde a uma quantidade com ou sem sinal. E na cadeia `formatos` o significado dos caracteres é o seguinte:

- (a) < ou > – indica se o buffer é little-endian (<) ou big-endian (>)
- (b) b ou B – indica um byte com sinal (b) ou sem sinal (B)
- (c) s ou S – indica um inteiro de dois bytes com sinal (s) ou sem sinal (S)
- (d) t ou T – indica um inteiro de três bytes com sinal (t) ou sem sinal (T)
- (e) l ou L – indica um inteiro de quatro bytes com sinal (l) ou sem sinal (L)
- (f) x – indica que deve ser deixado um byte de padding nessa posição

Suponha que façamos a chamada:

```
long nums[MAX];  
char buffer[] = {0xf0, 0xf3, 0xff, 0xff, 0xf4, 0xff,  
                 0xff, 0xff, 0x03, 0x04, 0xfa, 0x07};  
unpack ("<Bxxlslbx", buffer, nums);
```

Diga quais seriam os quatro primeiros valores do array `nums`, *em decimal*, logo após a chamada.

2. Traduza as funções `bar` e `zee` abaixo para assembly IA-32 (o assembly visto em sala), utilizando as regras usuais de alinhamento, passagem de parâmetros e resultados em C. Use registradores para abrigar as variáveis locais que aparecem nessas funções. Comente seu código.

(Não se preocupe se você não souber o que as funções fazem, apenas traduza-as literalmente!)

- (a) (2,5 pontos)

```
struct T {  
    int v;  
    struct T *next;  
};  
int bar (struct T *lista) {  
    int s = 0; struct T *p = lista;  
    while (p!=0) {  
        s += p->v;  
        p = p->next;  
    }  
    return s;  
}
```

- (b) (2,5 pontos)

```
int zee (int *a) {  
    int i = 0;  
    while (a[i]!=0)  
        i = a[i];  
    return i;  
}
```

3. (2,5 pontos) Considere um inteiro cujo bits de 26 a 28 representam um contador de 3 bits, ou seja, um número de 0 a 7.

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
-----
                |contador|
-----
```

Escreva em assembly uma função que receba um inteiro neste formato e o retorne com o campo de 3 bits incrementado de 1. Caso ocorra “overflow”, o valor do campo deve voltar para 0 (isto é, o incremento de 7 dá 0). A função não deve alterar nenhum outro bit do inteiro passado.

Sua função deve seguir a convenção de passagem de parâmetros do C. Considere o seguinte protótipo:

```
int incCampo(int pacote);
```