P1 de Software Básico – 2003.1

Departamento de Informática — PUC-Rio

1. (3.0 pontos) Considere o programa em C a seguir:

```
#include <stdio.h>

void dump (int n, void* p) {
   unsigned char* pc = p;
   int i;
   printf("==========================\n");
   printf("N. de bytes: %d\n",n);
   for (i = 0; i < n; i++,pc++)
      printf("%p - %02x\n", pc, *pc);
   printf("==========================\n");
}

struct {
   char c1;
   int i;
   char c2;
} x[] = { {72, 648, -112}, {-512, -1300, 68}};

int main (void) {
   dump(sizeof(x), x);
   return 0;
}
```

   Considerando que `x` será carregado a partir do endereço 0x00237F00, mostre qual será a saída desse programa, **justificando** com os cálculos necessários.

   Obs.: assuma a arquitetura de computador que temos visto no laboratório (`int` de 4 bytes, *little-endian* e alinhamento "usual").

2. (3.0 pontos) Considerando as seguintes variáveis:

```
int i, total;
struct parcela {
   char tipo;
   int valor;
} pagamento[10];
```

   Traduza o trecho de código em C a seguir para a linguagem Assembly da família de processadores Intel Pentium:

```
total = 0;
for (i = 0; i < 10; i++)
   if (pagamento[i].tipo == 2)
      total += pagamento[i].valor;
```

Observações:

- use os registradores que julgar necessário para a tradução;
- a rotina em assembly deve retornar o total no registrador EAX
- assuma que um ponteiro para a variável `pagamento` é fornecido em EBX;
- não é necessário traduzir as declarações das variáveis globais;
- utilize as convenções "usuais" de representação de dados em memória e de alinhamento, e comente o seu código.

3. (4.0 pontos) O código de Huffman é utilizado na compactação de dados. Ele se baseia na freqüência de ocorrência de um determinado dado para gerar códigos menores para dados mais freqüentes, e assim oferecer uma representação mais compacta do conjunto de dados.

   Para uma determinada seqüência de texto, foram calculados os seguintes códigos para as letras a seguir:

   ```
   a = 0
   b = 10
   c = 1100
   d = 1101
   e = 1110
   ```

   E o código `1111` é usado como marcador do final de uma seqüência de caracteres.

   Portanto, exemplos de possíveis seqüências são:

   ```
   10011001111 -> "bac"
   01101011101111 -> "adae"
   ```

   Considerando seqüências codificadas de no máximo 32 bits, escreva uma função que coloque na string `saida` o código lido em `entrada`:

   ```
   #define MAX 29
   char saida [MAX];

   void decode (unsigned int entrada);
   ```

   Observações:

   - a sua função pode ser feita em C ou assembly;
   - a string resultante deve ser terminada pelo caractere `'\0'`;
   - se você optar por assembly, considere a existência do label `SAIDA` para o endereço da string `saida`, e que o parâmetro `entrada` é passado em EAX;
   - o código ASCII do caractere `'a'` é 97.

Boa Sorte!