Assembly do IA-32 em ambiente Linux

Trabalho para Casa: **TPC7**

Alberto José Proença

Prazos

Entrega **impreterível**: semana de 25-Nov-02, na hora e local da sessão Teórico-Prática.

Não serão aceites trabalhos entregues depois deste prazo.

Metodologia

Idêntica aos trabalhos anteriores.

Introdução

A lista de exercícios que se apresenta segue directamente o material apresentado nas aulas teóricas da semana 9 (ver sumários na página da disciplina na Web), requerendo os conceitos básicos adquiridos em aulas anteriores.

Exercícios

Localização de elementos de um *array*

1. Considere as seguintes declarações:

```
short         S[7];
short        *T[3];
short       **U[6];
long double   V[8];
long double  *W[4];
```

Preencha a tabela seguinte, com os valores do tamanho de cada elemento, o tamanho total, o endereço do elemento `i`, para cada um destes *arrays*.

Array	Dim. Elem.	Dim. Total	Ender. início	Ender. Elem. `i`
S			x_S	
T			x_T	
U			x_U	
V			x_V	
W			x_W	

Aritmética com apontadores

2. Considere que o apontador para o início do *array* `S` (de inteiros `short`)e que o índice inteiro `i` estão armazenados nos registos `%edx` e `%ecx`, respectivamente. Para cada uma das expressões seguintes, indique o seu tipo, a fórmula de cálculo do seu valor, e uma implementação em código *assembly*. O resultado deverá ser colocado no registo `%eax` se for um apontador, e em `%ax` se for um inteiro `short`.

Expressão	Tipo	Valor	Código *Assembly*
`S+1`			
`S[3]`			
`&S[i]`			
`S[4*i+1]`			
`S+i-5`			

Matriz de 2 dimensões

3. Considere o seguinte código fonte, onde `M` e `N` são constantes declarados com `#define`.

```
1 int mat1[M][N];
2 int mat2[N][M];
3
4 int sum_element(int i, int j)
5 {
6    return mat1[i][j] + mat2[j][i];
7 }
```

Na compilação deste programa, o `gcc` gera o seguinte código *assembly*:

```
1  movl  8(%ebp),%ecx
2  movl  12(%ebp),%eax
3  leal  0(,%eax,4),%ebx
4  leal  0(,%ecx,8),%edx
5  subl  %ecx,%edx
6  addl  %ebx,%eax
7  sall  $2,%eax
8  movl  mat2(%eax,%ecx,4),%eax
9  addl  mat1(%ebx,%edx,4),%eax
```

Use técnicas de *reverse engineering* para determinar os valores de `M` e `N` baseados neste código *assembly*.

Structures

4. Considere a seguinte declaração de uma estrutura (*`structure`*).

```
struct prob {
   int *p;
   struct {
        int x;
        int y;
   } s;
   struct prob *next;
};
```

Esta declaração mostra que uma estrutura pode estar incluida numa outra, tal como *arrays* podem também estar incluidas em estruturas, estruturas em *arrays*, e *arrays* em *arrays*.

O procedimento seguinte (com algumas expressões omitidas) trabalha com esta estrutura:

```
void sp_init(struct prob *sp)
{
     sp->s.x = ________;
     sp->p = ________;
     sp->next = ________;
}
```

a) Qual o valor dos deslocamentos (em *bytes*) dos seguintes campos:

```
    p:
  s.x:
  s.y:
 next:
```

b) Quantos *bytes* ao todo necessita esta estrutura?

c) O compilador gera o seguinte código *assembly* para o corpo de `sp_init`:

```
1    movl    8(%ebp),%eax
2    movl    8(%eax),%edx
3    movl    %edx,4(%eax)
4    leal    4(%eax),%edx
5    movl    %edx,(%eax)
6    movl    %eax,12(%eax)
```

Com base neste código, preencha as expressões em falta no código C de `sp_init`.

Anexo (Resumo de Bry Ch.3.9.1)

Structures

The C `struct` declaration creates a data type that groups objects of possibly different types into a single object. The different components of a structure are referenced by names. The implementation of structures is similar to that of arrays in that all of the components of a structure are stored in a contiguous region of memory, and a pointer to a structure is the address of its first byte. The compiler maintains information about each structure type indicating the byte offset of each field. It generates references to structure elements using these offsets as displacements in memory referencing instructions.

As an example, consider the following structure declaration:

```
struct rec {
     int i;
     int j;
     int a[3];
     int *p;
};
```

This structure contains four fields: two 4-byte `int`'s, an array consisting of three 4-byte `int`'s, and a 4-byte integer pointer, giving a total of 24 bytes:

Offset	0	4	8			20
Contents	`i`	`j`	`a[0]`	`a[1]`	`a[2]`	`p`

Observe that array `a` is embedded within the structure. The numbers along the top of the diagram give the byte offsets of the fields from the beginning of the structure.

To access the fields of a structure, the compiler generates code that adds the appropriate offset to the address of the structure. For example, suppose variable `r` of type `struct rec *` is in register `%edx`. Then the following code copies element `r->i` to element `r->j`:

```
1    movl    (%edx),%eax              Get r->i
2    movl    %eax,4(%edx)             Store in r->j
```

Since the offset of field `i` is `0`, the address of this field is simply the value of `r`. To store into field `j`, the code adds offset `4` to the address of `r`.

To generate a pointer to an object within a structure, we can simply add the field's offset to the structure address. For example, we can generate the pointer `&(r->a[1])` by adding offset `8 + 4*1 =12`. For pointer `r` in register `%edx` and integer variable `i` in register `%eax`, we can generate the pointer value `&(r->a[i])` with the single instruction:

```
   r in %eax, i in %edx
1    leal    8(%eax,%edx,4),%ecx      %ecx = &r->a[i]
```

As a final example, the following code implements the statement:

```
r->p = &r->a[r->i + r->j];
```

starting with `r` in register `%edx`:

```
1    movl    4(%edx),%eax             Get r->j
2    addl    (%edx),%eax              Add r->i
3    leal    8(%edx,%eax,4),%eax      Compute &r->[r->i + r->j]
4    movl    %eax,20(%edx)            Store in r->p
```

As these examples show, the selection of the different fields of a structure is handled completely at compile time. The machine code contains no information about the field declarations or the names of the fields.

Nº	**Nome:**	**Turma:** Seg1 - Seg2 - Ter

Resolução dos exercícios

1. Localização de elementos de um *array*

Array	Dim. Elem.	Dim. Total	Ender. início	Ender. Elem. `i`
S			x_S	
T			x_T	
U			x_U	
V			x_V	
W			x_W	

2. Aritmética com apontadores

Expressão	Tipo	Valor	Código *Assembly*
`S+1`			
`S[3]`			
`&S[i]`			
`S[4*i+1]`			
`S+i-5`			

3. Matriz de 2 dimensões

M = _____ N = _____

4. Structures

a)

```
    p: _________
  s.x: _________
  s.y: _________
 next: _________
```

b)

c)

```
1    movl   8(%ebp),%eax
2    movl   8(%eax),%edx
3    movl   %edx,4(%eax)
4    leal   4(%eax),%edx
5    movl   %edx,(%eax)
6    movl   %eax,12(%eax)
```

```
void sp_init(struct prob *sp)
{
     sp->s.x = _________;
     sp->p =   _________;
     sp->next = _________;
}
```