

## 11 - arrays

Sistemas Hardware-Software - 2020/2

Igor Montagner

### Arrays

A parte final de nossas atividades com Assembly é entender *Arrays*. Já estudamos todo o necessário para lidar com eles:

1. um array é representado por um apontador para seu primeiro elemento. Já estudamos ponteiros e a escrita em memória com `MOV`.
2. ao acessar o elemento `i` de `long *vec` estamos acessando o endereço `&vec[0] + sizeof(long) * i`. A notação de cálculos de endereço de memória faz exatamente isso.

Para iniciar vamos revisar a notação de cálculo de endereços de memória: `D(%reg1, %reg2, S)`, onde

- `D` é um número inteiro que representa um deslocamento constante a partir do endereço guardado em `%reg1`
- `%reg1` contém o endereço do primeiro elemento do vetor
- `%reg2` contém o índice do elemento a ser acessado
- `S` contém o tamanho de cada elemento do vetor e pode ser `1`, `2`, `4` ou `8`.

O acesso é feito ao endereço `D + %reg1 + S * %reg2`. Ou seja, primeiro computamos o endereço e depois acessamos a memória *no endereço computado*.

**Exemplo:** dado o array `int *vec` (guardado em `%rdx`). A atribuição de 5 ao elemento `i` (guardado em `%ecx`) seria traduzida como

```
MOVL $0x5, (%rdx, %rcx, 4)
```

**Importante:** lembre-se de que apontadores são tipos de dados com tamanho *8 bytes*.

---

**Exercício 1:** Considerando um vetor `short *vec` e que o endereço do primeiro elemento de `vec` esteja em `%rdi`

1. qual a instrução usada para mover o conteúdo de `vec[i]` para o registrador `%eax`? (supondo que o valor de `i` esteja em `%esi`)
2. qual a instrução usada para mover `&vec[i]` para o registrador `%eax`? **Dica:** como você implementava o operador `&` com variáveis locais?

## Loops e arrays

**Exercício 2:** veja o código abaixo e responda as perguntas.

```
0000000000000000 <soma>:
0:  ba 00 00 00 00      mov    $0x0,%edx
5:  b8 00 00 00 00      mov    $0x0,%eax
a:  eb 09               jmp    15 <soma+0x15>
c:  48 63 ca           movslq %edx,%rcx
f:  03 04 8f           add    (%rdi,%rcx,4),%eax
12: 83 c2 01           add    $0x1,%edx
15: 39 f2              cmp    %esi,%edx
17: 7c f3              jl     c <soma+0xc>
19: f3 c3              repz retq
```

1. A função acima usa vários registradores. Para facilitar a descoberta de quais são parâmetros da função anote abaixo cada registrador usado e, ao lado, a linha do primeiro uso e se esse uso foi leitura ou escrita.

2. Se o primeiro acesso a registrador é de escrita então ele provavelmente não é um parâmetro. Com base nisto, escreva abaixo a declaração da função acima.

3. Sempre que escrevemos a notação de acesso à memória `D(%reg1, %reg2, S)` precisamos usar registradores de `64` bits nos lugares de `reg1` e `reg2`. Com base nisto, explique qual o uso feito do registrador `%edx` e porquê usamos a instrução `movslq` na linha `c`.

4. Faça uma versão em *C* do código acima usando somente `if-goto`. Escreva, então, ao lado desta versão um código legível em *C*.

## Acesso a elementos constantes

O acesso a elementos “constantes”, como `long v[10]; v[5] = 0;`, não é feito usando a notação acima, pois o compilador já sabe **em tempo de compilação**, qual é o deslocamento necessário para encontrar a posição 6 de `v`.

**Exercício:** Considerando o exemplo acima, responda.

1. Supondo que `v=0x100`, qual o é o endereço de `v[5]`?
2. Escreva a instrução usada para mover o valor `0` para `v[5]` (supondo que o endereço do primeiro elemento esteja em `%rdi`).

---

```
0000000000000000 <func_que_recebe_array>:
0:  8b 47 04          mov     0x4(%rdi),%eax
3:  03 07            add     (%rdi),%eax
5:  3b 47 08          cmp     0x8(%rdi),%eax
8:  0f 9c c0          setl    %al
b:  0f b6 c0          movzbl  %al,%eax
e:  c3              retq
```

1. Temos acessos à memória relativos ao endereço passado em `%rdi` nas linhas `0, 3` e `5`. Isto significa que `%rdi` é um ponteiro. Pelos tipos de acessos feitos, você consegue identificar para qual tipo de variável ele aponta?
2. Traduza os acessos de memória feitos nas linhas citadas acima para a notação de acesso a arrays em *C*.
3. Com base no respondido acima, faça uma versão em *C* legível do código assembly acima. Se ajudar, faça uma tradução linha a linha do Assembly e depois torne-a legível.

## Exercício final

**Exercício:** Veja agora o código abaixo e responda.

```
0000000000000000 <first_neg>:
 0:  b8 00 00 00 00      mov     $0x0,%eax
 5:  39 f0                cmp     %esi,%eax
 7:  7d 0e                jge     17 <first_neg+0x17>
 9:  48 63 d0            movslq  %eax,%rdx
 c:  83 3c 97 00          cmpl    $0x0,(<%rdi,%rdx,4>)
10:  78 05                js      17 <first_neg+0x17>
12:  83 c0 01            add     $0x1,%eax
15:  eb ee                jmp     5 <first_neg+0x5>
17:  f3 c3                repz retq
```

1. A função acima usa vários registradores. Para facilitar a descoberta de quais são parâmetros da função anote abaixo cada registrador usado e, ao lado, a linha do primeiro uso e se esse uso foi leitura ou escrita.

2. Desenhe setas indicando o destino dos pulos no código acima. Você consegue identificar quais estruturas de controle? Entre quais linhas?

3. Faça uma versão em *C* usando `if-goto` do código acima.

4. Transforme seu código em uma versão legível.