Universidade Federal de Santa Catarina - Centro Tecnológico - Departamento de Informática e Estatística
INE 5411 - Organização de Computadores
Roteiro do Laboratório 4 - Manipulação de arranjos: índices e ponteiros
Pré-requisitos para a compreensão e execução do laboratório: arranjos, ponteiros e estruturas de laços.

1. Objetivo

O objetivo desta aula é estudar alternativas de geração de código para manipulação de arranjos de acordo com dois mecanismos diferentes de acesso aos seus elementos: o uso de índices e o uso de ponteiros. Tais mecanismos são suportados, implícita ou explicitamente, em linguagens de alto nível contemporâneas. Nesse contexto, esta aula propõe dois estudos de caso. Ambos abordam o mesmo problema (a inicialização com zero para todos os elementos de um arranjo), mas adotando programas-fonte distintos: o primeiro induz o compilador a gerar código usando índices, o segundo o induz a gerar código usando ponteiros. Veremos que, embora equivalentes, as alternativas de programação podem resultar em código com desempenho bastante diferente, pois uma delas expõe mais oportunidades de otimização para o compilador.

Convenções para os estudos de caso

- **Definição:** Um laço for, usado em linguagens de programação como C e Java, é composto pelas quatro seguintes partes:

```
for ( PRÓLOGO ; TESTE ; ITERAÇÃO )
{
        CORPO
}
```

- Por simplicidade, mas sem perda de generalidade, vamos nos concentrar no em procedimentos isoladamente, desconsiderando (para os estudos de caso propostos), o código para inicializar salvar e restaurar registradores usados pelos procedimentos.
- Adote a seguinte alocação de registradores: **(array, size) → ($a0, $a1).**
- **Use somente instruções nativas na inicialização e no corpo do laço** (exceto pela pseudo-instrução `la` necessária para inicializar o endereço-base do arranjo).

Procedimento de teste

Para facilitar o teste, os estímulos serão organizados na área de dados globais da memória. As N primeiras palavras dessa área armazenarão os elementos do arranjo de inteiros `array`. A palavra seguinte armazenará o valor da variável `size` (N). Aplique o procedimento abaixo para verificar o funcionamento de cada uma das versões do código solicitadas nos três exercícios desta aula.

```
.data
# Arranjo inicializado com elementos N não nulos. O valor de N é provido no relatório.
_array: .word 3:N            # N palavras com o valor 3
_size: .word N               # tamanho do arranjo
```

Resultado esperado: Todos os N elementos do arranjo, armazenados sequencialmente começando no endereço 0x10010000, devem ter seus valores iguais a **zero**. A posição de memória contendo _size deve continuar com o valor N.

2. Estudo de caso 1: uso de índices

O código abaixo descreve um procedimento `clear1`, escrito em linguagem C, que inicializa com zero todos os elementos de um arranjo de inteiros `array`, acessando cada elemento através do índice `i`. Os parâmetros do procedimento são o endereço-base do primeiro elemento do arranjo (`array[]`) e seu número total de elementos (`size`).

```
void clear1 ( int array[], int size )
{
        int i;
        for ( i = 0 ; i < size ; i+=1 )
                array[i] = 0;
}
```

Convenções específicas

- Adote a seguinte alocação de registradores: **i → $t0**.
- Assuma que o **endereço de `array[i]`** seja armazenado no registrador **$t2**.
- Atribua o label `clear1` à posição de memória contendo a primeira instrução executada dentro do procedimento.
- Atribua o label `Loop1` à posição de memória contendo a primeira instrução executada dentro do laço (após o prólogo).

Exercício 1:

Implemente o trecho de código do estudo de caso 1 em linguagem de montagem do processador MIPS. Armazene-o em um arquivo exercicio1.txt, instrumentado-o com os estímulos do procedimento de teste. O trecho de código abaixo ilustra esquematicamente a estrutura a ser adotada para o arquivo de programa em linguagem de montagem.

- ***Dica 1**: Como seu programa será carregado pelo MARS como uma subrotina `clear1` da rotina principal `main`, inclua um `jr $ra` no final de seu programa para provocar o retorno à rotina principal.*
- ***Dica 2**: **Não se esqueça de inicializar o índice no prólogo.***

```
.data
# Arranjo inicializado com elementos N não nulos. O valor de N é provido no relatório.
_array: .word 3:N           # N palavras com o valor 3
_size: .word N              # tamanho do arranjo

.text
.globl main
main:
jal clear1   # Salto para o endereço do procedimento
li $v0, 10   # Exit syscall
syscall

clear1:
# inicialização dos parâmetros
la $a0, _array
lw $a1, _size
# Prólogo do laço. Deve conter uma única instrução de inicialização do índice.
...
# Teste, corpo e iteração do laço.
Loop1:
slt $t3, $t0, $a1
beq $t3, $zero, Exit  # Se (i>=size) desvia para Exit

...

j Loop1
# Epílogo do procedimento
Exit:
jr $ra  # Retorna ao programa principal
```

a) Carregue e execute o arquivo exercicio1.txt. Modifique o código até que os resultados esperados sejam alcançados.

b) Responda às Questões 1.1 e 1.2 do relatório de aula.

3. Estudo de caso 3: uso de ponteiros

O código abaixo descreve um procedimento clear2, escrito em linguagem C, que inicializa com zero todos os elementos de um arranjo de inteiros *array, acessando cada elemento através do ponteiro p. Os parâmetros do procedimento são o ponteiro para o primeiro elemento do arranjo (*array) e seu número total de elementos (size).

```
void clear2 ( int *array, int size )
{
        int *p;
        for ( p = &array[0]; p < &array[size]; p = p + 1 )
                *p = 0;
}
```

Revisão conceitual

Em linguagem C, o endereço de uma variável é indicado por & e a referência a uma variável apontada por um ponteiro é denotada por *. Por exemplo, suponha que uma variável v seja declarada do tipo inteiro (int). Após a execução do comando p = &v, temos:

- p aponta para a variável v (p contém o endereço de memória onde reside a variável v);
- *p é uma representação alternativa da variável v (*p é o conteúdo do endereço representado por p);
- Quando p é incrementado de 1 no programa-fonte, o endereço de memória é incrementado no código gerado de um valor igual ao número de bytes em que a variável v é representada (como v é um inteiro, o incremento é de 4)

Portanto, no programa anterior, p é inicializado para apontar para o primeiro elemento do arranjo e o laço termina quando p estiver apontando para a primeira posição fora do arranjo (array[size-1] é o último elemento do arranjo).

Convenções específicas

- Adote a seguinte alocação de registradores: **p → $t0**.
- Armazene no registrador **$t2** o **endereço de array[size]**.
- Atribua o label `clear2` à posição de memória contendo a primeira instrução executada dentro do procedimento.
- Atribua o label `Loop2` à posição de memória contendo a primeira instrução executada dentro do laço (após o prólogo).

Exercício 2:

Implemente o trecho de código do estudo de caso 2 em linguagem de montagem do processador MIPS. Armazene-o em um arquivo exercicio2.txt, instrumentado-o com os estímulos do procedimento de teste. Limite-se a traduzir o programa-fonte para linguagem de máquina. **Não faça otimizações no interior do laço** (você vai notar a existência de instrução(ões) que pode(m) ser movida(s) para junto do prólogo, mas você não deve fazê-lo aqui, **pois essa otimização é o objetivo do Exercício 3**). O trecho de código abaixo ilustra esquematicamente a estrutura a ser adotada para o arquivo de programa em linguagem de montagem.

- ***Dica: Não se esqueça de inicializar o índice no prólogo.***

```
.data
# Arranjo inicializado com elementos N não nulos. O valor de N é provido no relatório.
_array: .word 3:N           # N palavras com o valor 3
_size: .word N              # tamanho do arranjo

.text
.globl main
main:
jal clear2   # Salto para o endereço do procedimento
li $v0, 10   # Exit syscall
syscall

clear2:
# inicialização dos parâmetros
la $a0, _array
lw $a1, _size
# Prólogo do laço. Deve conter uma única instrução de inicialização de p.
...
# Teste, corpo e iteração do laço.
Loop2:
slt $t3, $t0, $t2
beq $t3, $zero, Exit  # Se (p>=&array[size]) desvia para Exit

...

j Loop2
# Epílogo do procedimento
Exit:
jr $ra  # Retorna ao programa principal
```

a) Carregue e execute o arquivo exercicio2.txt. Modifique o código até que os resultados esperados sejam alcançados.

b) Responda às questões 2.1 e 2.2 do relatório de aula.

Exercício 3:

No código do Exercício 2, identifique as instruções do corpo do laço que são independentes do ponteiro. Em um novo arquivo exercicio3.txt, otimize o código do Exercício 2, movendo as instruções identificadas para antes do laço (prólogo) porém preservando a semântica do programa-fonte original (**não faça qualquer outra alteração no código além de mover as instruções solicitadas**).

a) Carregue e execute o arquivo exercicio3.txt. Modifique o código até que os resultados esperados sejam alcançados.

b) Responda às questões 3.1 a 3.4 do relatório de aula.