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

1. Objetivo e requisitos

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 (o uso de um laço `for` para inicializar com zero todos os elementos de um arranjo), mas adotando programas-fonte distintos: o primeiro (procedimento `clear1`) induziria o compilador a gerar código usando **índices**, o segundo (procedimento `clear2`) o induziria a gerar código usando **ponteiros**.

Você vai fazer o papel do compilador e gerar o código *assembly* correspondente a esses dois procedimentos. Você verá que, embora funcionalmente equivalentes, essas alternativas de programação em linguagem de alto nível podem resultar em códigos *assembly* com desempenho bastante diferente, pois uma delas expõe mais oportunidades de otimização para o compilador.

Uma versão de cada um dos procedimentos `clear1` e `clear2` é apresentada nas **páginas 141-144 do livro-texto**. Os experimentos deste laboratório solicitam que você implemente **novas versões** que devem obedecer a um conjunto de **requisitos diferentes dos usados no livro-texto**. Portanto, a **compreensão das versões do livro-texto é prérequisito** para a execução dos experimentos previstos neste roteiro e para responder as perguntas do relatório.

Requisitos gerais para produzir as novas versões

- Os arquivos `experimento1-codigo-base.asm` e `experimento2-codigo-base.asm` fornecem versões incompletas dos programas necessários para os experimentos e especificam requisitos para criar versões completas. Depois de completar instruções e operandos faltantes, você deve gerar dois arquivos `experimento1.asm` e `experimento2.asm` para executar os experimentos e responder às perguntas do relatório.
- Ao completar os programas, você deve usar **exatamente** o número de instruções solicitado nos arquivos de código-base fornecidos (nem mais instruções, nem menos instruções), **sem alterar as instruções já fornecidas**.
- **Use somente instruções nativas** para completar os programas, **exceto por uma pseudo-instrução** necessária para inicializar o endereço-base do arranjo.
- O código *assembly* **não deve usar os registradores $s0-$s7** para evitar as instruções adicionais que seriam necessárias para salvá-los e restaurá-los, como exige a convenção de chamada de procedimentos.
- Adote a seguinte **alocação para os argumentos** dos procedimentos `clear1` e `clear2`: **(array, size) → ($a0, $a1).**

Procedimento de teste

Antes de responder as perguntas do relatório, você deve testar se seu programa está funcionando corretamente. Para facilitar o teste, os estímulos são organizados na área de dados globais da memória. Os N primeiros dados dessa área armazenarão os elementos do arranjo `array`. O dado seguinte armazenará o valor da variável `size` (N). Os arquivos de código-base fornecidos assumem que os elementos do arranjo são inteiros representados em 32 bits. Por isso, eles usam a diretiva `.word`, como ilustrado abaixo:

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

Depois de substituir N por um valor definido no relatório, monte o arquivo *assembly*, execute-o no simulador e verifique se seu programa exibe o **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. Experimento 1: uso de índices

O código abaixo descreve o 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;
}
```

Requisitos específicos para produzir o arquivo `experimento1.asm`

- Adote a seguinte alocação de registradores: **`i`** → **$t0**.
- Assuma que o **endereço de `array[i]`** seja armazenado no registrador **$t2**.
- No código *assembly*, o label `clear1` deve representar a posição de memória contendo a primeira instrução executada dentro do procedimento e o label `Loop1` deve representar a posição de memória contendo a primeira instrução executada dentro do laço.

Produção do arquivo `experimento1.asm`

Complete as instruções e operandos faltantes no arquivo `experimento1-codigo-base.asm`, seguindo rigorosamente os **requisitos gerais** e os **requisitos específicos**. Chame o novo arquivo assim produzido de `experimento1.asm`. Siga as instruções do relatório para definir o valor de N e executar variantes do Experimento 1.

3. Experimento 2: uso de ponteiros

O código abaixo descreve o 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 *assembly* 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 procedimento `clear2`, 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 (ou seja, `array[size-1]` é o último elemento do arranjo).

Requisitos específicos para produzir o arquivo `experimento2.asm`

- Adote a seguinte alocação de registradores: **p** → **$t0**.
- Armazene no registrador **$t2** o **endereço de `array[size]`**.
- No código *assembly*, o label `clear2` deve representar a posição de memória contendo a primeira instrução executada dentro do procedimento e o label `Loop2` deve representar a posição de memória contendo a primeira instrução executada dentro do laço.

Produção do arquivo `experimento2.asm`

Complete as instruções e operandos faltantes no arquivo `experimento2-codigo-base.asm`, seguindo rigorosamente os **requisitos gerais** e os **requisitos específicos**. Chame o novo arquivo assim produzido de `experimento2.asm`. Siga as instruções do relatório para definir o valor de N e executar variantes do Experimento 2.