Universidade Federal de Santa Catarina - CTC - Departamento de Informática e Estatística
INE 5411 - Organização de Computadores
Roteiro do Laboratório 3 - Implementação de Laços de Iteração

1. Objetivo

O objetivo geral desta aula é estudar alternativas eficientes para a geração de código associado a laços de iteração (while, repeat...until, do...while e for) suportados em linguagens de alto nível contemporâneas. Nesse contexto, esta aula propõe um estudo de caso de um pequeno trecho de programa contendo uma construção while. O objetivo específico é mostrar que a escolha de linguagens de programação de alto nível, tais como C ou Java, resulta em códigos com diferentes números de instruções, o que afeta o desempenho.

2. Estudo de caso

O código abaixo descreve o laço de iteração usado para o estudo de caso. Na situação anômala em que o índice i resulte fora dos limites do arranjo save, o mau funcionamento do programa pode ser mais difícil de depurar quando o programa foi originalmente escrito em C, linguagem que não possui mecanismo algum de teste automático de limites, ao contrário de Java, a qual requer que esse teste seja feito em tempo de execução (mesmo que a pessoa programadora não o tenha inserido no código-fonte). Em outras palavras, em C, a pessoa programadora precisaria incluir explicitamente (manualmente) o teste de limites em seu código (se quiser que seja efetuado), enquanto que, em Java, a pessoa programadora não precisa se preocupar em fazê-lo (mas também não pode evitá-lo), pois as instruções de teste serão geradas automaticamente pelo compilador. No primeiro caso, gera-se um código-objeto potencialmente mais rápido (assumindo um risco), enquanto que, no segundo caso, gera-se um código-objeto mais robusto (aceitando uma degradação de desempenho).

```
i = 0;
while ( save[i] == k )
        i += 1;
```

Resultado esperado: em condições normais, ao final da execução, a variável `i` deve conter o número de elementos **sucessivos** do arranjo – a partir do **primeiro** -- que são iguais a k.

Requisitos para os experimentos

- Adote a seguinte alocação de registradores: **(i,k) → ($s3,$s5)**.
- Assuma que o **endereço da base do arranjo save** está armazenado no registrador **$s6**.
- Atribua labels Loop e Exit às posições de memória contendo, respectivamente, a primeira instrução executada dentro do laço e a primeira instrução executada à saída do laço.
- **Use somente instruções nativas na inicialização e no corpo do laço** (exceto pela **pseudoinstrução `la`** necessária para inicializar o endereço-base do arranjo e pela **pseudoinstrução `lw`** necessária para inicializar o valor de k).

3. Experimento 1: Código *assembly* sem teste de limites do arranjo

Você deverá completar o arquivo disponibilizado no Moodle (**experimento1-codigo-base.asm**), conforme instruções abaixo. Serão realizados dois testes com o mesmo código *assembly*, mas com estímulos diferentes. Os estímulos serão organizados na área de dados globais da memória para inicializar o arranjo com diferentes conteúdos. As primeiras palavras dessa área armazenarão os elementos do **arranjo de inteiros** save. A palavra seguinte armazenará o valor da **variável inteira** k. **Não se esqueça** de montar novamente o programa para cada novo estímulo. Para facilitar o preenchimento do relatório, recomenda-se que você crie dois arquivos-texto **experimento1-1.asm** e **experimento1-2.asm** com o mesmo código, mas estímulos diferentes, como indicado abaixo:

Experimento 1.1: Estímulos: save = [...] e k = 6:

```
        .data
_save:  .word Estímulo 1.1 (disponível no relatório)
_k:     .word 6
```

Experimento 1.2: Estímulos: save = [...] e k = 6:

```
        .data
_save:  .word Estímulo 1.2 (disponível no relatório)
_k:     .word 6
```

Para a observação do resultado esperado, aparece ao final do programa, uma sequência de instruções que imprime no console o valor de $s3 à saída do laço (não se preocupe, você não precisa entendê-la para executar os experimentos):

```
Exit:
addi $v0, $zero, 1
add $a0, $zero, $s3
syscall
```

Passos do experimento

a) Crie um arquivo **experimento1-1.asm** com a implementação em linguagem de montagem do trecho de programa escrito em linguagem C. Instrumente-o com o Estímulo 1.1. No MARS, abra, monte e execute esse arquivo.

b) Responda às Questão 1a, 1b, 1c do Relatório 3.

c) Crie um arquivo **experimento1-2.asm** com a implementação em linguagem de montagem do trecho de programa escrito em linguagem C. Instrumente-o com o Estímulo 1.2. No MARS, abra, monte e execute esse arquivo.

d) Responda às Questão 1d, 1e, 1f do Relatório 3.

4. Experimento 2: Código *assembly* com teste de limites do arranjo

Informações adicionais

Quando um programa é escrito em uma linguagem orientada a objetos, o compilador encapsula cada objeto na forma de uma estrutura de dados, onde o primeiro elemento contém o endereço de uma tabela que permite invocar métodos do objeto. Por isso, em Java arrays, a primeira palavra armazenada na estrutura de dados é reservada para armazenar o endereço da correspondente tabela. Além disso, a segunda palavra da estrutura de dados é reservada para armazenar um atributo: o número total de elementos do arranjo. Vamos usar uma representação similar a Java arrays no Experimento 2. Portanto, lembre-se que os elementos propriamente ditos do arranjo save são armazenados a partir da terceira palavra.

Lembre também que, se o arranjo contém N elementos, índices válidos estão no intervalo [0, N).

Procedimento de teste

Você deverá completar o arquivo disponibilizado no Moodle (**experimento2-codigo-base.asm**), conforme instruções abaixo. Os testes usados neste experimento são similares ao do experimento anterior, mas usam a nova representação para o arranjo: as duas primeiras palavras armazenadas em memória são utilizadas para armazenar atributos do arranjo (para aderir à representação Java). Em particular, a segunda palavra contém o tamanho do arranjo (a primeira é irrelevante para nosso código e o valor 9999 lhe será atribuído só para distingui-la das demais). Para facilitar o preenchimento do relatório, recomenda-se que você crie dois arquivos-texto **experimento2-1.asm** e **experimento2-2.asm** com o mesmo código, mas estímulos diferentes, como indicado abaixo:

Procedimento de teste 2.1: Estímulos: save = [...] e k = 6:

```
.data
_save: .word Estímulo 2.1 (disponível no relatório)
_k: .word 6
_error: .asciiz "Index Out of Bounds Exception"
```

Procedimento de teste 2.2: Estímulos: save = [...] e k = 6:

```
.data
_save: .word Estímulo 2.2 (disponível no relatório)
_k: .word 6
_error: .asciiz "Index Out of Bounds Exception"
```

Para a observação do resultado esperado ou indicação de erro em tempo de execução, aparece ao final do programa uma sequência de instruções que imprime no console o valor de $s3 à saída do laço, caso o índice esteja nos limites do arranjo e, em caso contrário, imprime uma mensagem de erro (não se preocupe, você não precisa entendê-la para executar os experimentos):

```
Exit:
addi $v0, $zero, 1
add $a0, $zero, $s3
syscall
j End

IndexOutOfBounds:
addi $v0, $zero, 4
la $a0, _error
syscall
End:
```

Requisitos adicionais para este experimento

- Insira **uma instrução** -- imediatamente antes do label Loop -- que carregue em $t2 o número de elementos do arranjo (tamanho). Esta instrução deve ser nativa.
- Quando o índice não está dentro dos limites do arranjo, o programa deve desviar para um endereço representado pelo label IndexOutOfBounds, onde há rotina de indicação de erro (já disponibilizada). Insira **duas instruções** nativas: uma para verificar se $0 \leq i < N$, (onde N é tamanho do arranjo), outra para desviar condicionalmente para a rotina de indicação de erro.

Passos do experimento

a) Crie um arquivo **experimento2-1.asm** com a implementação em linguagem de montagem do trecho de programa escrito em linguagem C, adapte-o para a nova representação adotada para o arranjo save e inclua as três instruções nativas especificadas acima. Instrumente-o com o Estímulo 2.1. No MARS, abra, monte e execute esse arquivo. Se você não tiver obtido o resultado esperado, revise seu programa até que isso ocorra, antes de prosseguir no roteiro.
Dica: use o deslocamento ("offset") da instrução lw para compensar o efeito dos atributos armazenados nas duas primeiras palavras reservadas da estrutura de dados.

b) Responda às Questões 3a e 3b do Relatório 3.

c) Crie um arquivo **experimento2-2.asm** com a mesma implementação do item acima, mas agora instrumente-o com o Estímulo 2.2. No MARS, abra, monte e execute esse arquivo.

d) Responda às Questões 3c e 3d do Relatório 3.