

Curso: LCC
Disciplina: Sistemas de Computação

2ª Teste 23/Jun/08
Duração (máx): 1h30m

Avisos

1. Este Teste vale 10 valores e é constituído por 2 partes:
 - **Parte 1: – obrigatória**
3 questões sobre execução de instruções numa HLL numa arquitectura tipo IA-32.
Falhando mais de 2 questões no conjunto dos 2 testes não satisfaz os resultados de aprendizagem requeridos para aprovação na UC; falhando 1 a 2 haverá lugar a reapreciação (analisado caso a caso).
 - **Parte 2** Classificativa – **opcional**,
Para definição da classificação final (até 7 valores neste Teste, na escala 10 a 20), apenas para quem satisfizer os critérios mínimos.
2. Apresente sempre o raciocínio ou os cálculos que efectuar; o não cumprimento desta regra equivale à não resolução do exercício. Use o verso das folhas do enunciado do exame como papel de rascunho.

Parte 1

1. (A) Considere o fragmento de código *assembly* apresentado em baixo, representando uma função no IA-32 após ter sido compilado pelo `gcc`, e as seguintes notas:
 - a função recebe 2 argumentos: o 1º é um apontador para a variável `loc`, e o 2º argumento é o valor da variável `n`, do tipo inteiro;
 - a função implementa uma das seguintes estruturas de controlo do C: "`if <cond> then <statement_1> else <statement_2>;`" "`do <statement_3> while <cond>;`" "`while <cond> <statement_4>;` ou "`for (<cond_ini>, <end_cntrl>, <cntrl_update>) <statement_5>.`"

```
    pushl    %ebp
    movl     %esp, %ebp
    pushl    %ebx
    xorl     %edx, %edx
    movl     12(%ebp), %ecx
    xorl     %eax, %eax
    movl     8(%ebp), %ebx
.L2:
    addl     (%ebx,%edx,4), %eax
    incl     %edx
    cmpl     %ecx, %edx
    jl       .L2
    popl     %ebx
    popl     %ebp
    ret
```

Nº**Nome**

- a) (A) Nem todo o código gerado pelo `gcc` corresponde ao corpo da função; parte dele tem a ver com a gestão do contexto relacionado com a invocação e regresso de uma função. **Identifique** essas instruções no código *assembly* e **indique** qual a utilidade de cada uma delas.
- b) (A) **Indique** qual a **estrutura de controlo** presente, e **mostre** as instruções em *assembly* que estão directamente relacionadas com a implementação daquela estrutura.
- c) (A) **Mostre** a estrutura da *stack* associada a esta função na arquitectura IA-32, após a invocação da função e imediatamente antes da execução do corpo da função, **indicando a dimensão e conteúdo** de toda a informação relevante, bem como a localização do *stack pointer* e do *base pointer*.

Parte 2

2. Considere de novo a função do exercício anterior.

- a) ^(R) Se se considerar essa função como uma "caixa negra", que recebe como argumentos apenas 2 valores (um apontador e um inteiro), **descreva** numa só expressão matemática o *output* desta função (pode usar texto para explicar o raciocínio e/ou detalhes da expressão).

- b) ^(R) Considere a seguinte alteração ao código dessa função: passa a conter operações sobre os elementos de um *array* bi-dimensional de inteiros, 64x64 (uma variável local estruturada da função, por exemplo `vec[i][j]`). Imagine o código que o `gcc` iria gerar e **modifique** as suas respostas às alíneas **a)** e **c)** do exercício anterior de acordo com esta alteração.

Nota: Considere que o `gcc` otimiza a utilização da *stack*, i.e., nenhuma célula é desaproveitada.

- c) ^(B) **Calcule** os valores de `i` e `j` que definem o elemento do vector que está armazenado nas célula da *stack* que estão à distância de 269-272 posições da célula que contém o início do endereço de regresso da função. Não se esqueça de apresentar todos os cálculos.

Nº	Nome
----	------

- d) (B/E) Supondo que esta era uma função-folha, aponte as diferenças que deveria encontrar entre a estrutura da *stack* que mostrou em b) e a que encontraria numa arquitectura RISC de 32 bits.

3. (R) Comente a seguinte afirmação:

"Quando se tenta otimizar o desempenho na execução de uma aplicação, a opção de loop unroll num ciclo com muitas iterações deve ser evitado, senão, ao desenrolar por completo o ciclo, arrisco-me a ficar com um pedaço de código muito grande que não cabe todo na cache, e portanto sou fortemente penalizado com excessivos acessos à memória com cache miss."

4. (B) (**Nota: esta questão vale por 2**) A seguinte função calcula a soma dos elementos numa lista ligada:

```
1 int soma_lista(list_ptr ls)
2 {
3     int soma=0;
4
5     for (; ls; ls=ls->next)
6         soma += ls->data;
7     return soma
8 }
```

O código *assembly* que um compilador geraria para este ciclo, e respectiva conversão para operações do P6 (IA-32), teria o seguinte aspecto:

Instruções em <i>assembly</i>	Operações da <i>Execution Unit</i> do P6
.L43: addl 4(%edx),%eax movl (%edx),%edx testl %edx,%edx jne .L43	. load 4(%edx,0) → t.1 addl t.1,%eax.0 → %eax.1 load (%edx.0) → %edx.1 testl %edx.1,%edx.1 → cc.1 jne-taken cc.1

Nº

Nome

Recordar que:

- a unidade de execução do P6 tem várias unidades funcionais, sendo apenas algumas replicadas;
- a latência dessas unidades não é igual para todas elas, e que algumas permitem execução encadeada no seu interior;
- o P6 permite a execução de instruções fora de ordem e que pode, em cada instante, enviar simultaneamente 3 instruções para a unidade de execução (i.e., é superescalar nível 3, ou em terminologia inglesa, *3-way superscalar*).

Desenhe um diagrama que mostre a sequência de operações para as primeiras 3 iterações do ciclo (use um tipo de diagrama semelhante ao utilizado nos slides das aulas), **identificando** os factores que limitam o desempenho em cada iteração.