Análise do ISA do IA-32 e avaliação de desempenho

Teste 2

Nº	**Nome**

Nota: Apresente sempre o raciocínio ou os cálculos que efectuar; o não cumprimento desta regra equivale à não resolução do exercício.

1. (A) Uma matriz 10x12 de valores reais (*float*) foi declarada como variável local `C` de uma função. Considere que o compilador reservou espaço na *stack frame* para esta matriz a começar no endereço de memória à distancia de 512 células do *frame pointer*. **Indique**, apresentando e justificando os cálculos que efectuar, a que distância do *frame pointer* se encontra o início do bloco de memória contendo o elemento `C[1][2]`.

2. Considere o código *assembly* da função *loaddefault* da prova laboratorial, listado no fim deste enunciado, resultante da compilação no servidor, com e sem a opção de optimização `-O2`.

a) (A) **Identifique** na versão não-optimizada desse código, o conjunto de instruções que implementa a estrutura de controlo do 1º ciclo `for` (e apenas estas), e **indique** a possível correspondência ao código fonte. Escreva aqui a sua resolução.

Nº	**Nome**

b) **(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 na versão do código *assembly* optimizado pelo `gcc`, e **indique** qual a função de cada uma delas. Indique aqui as instruções.

c) **(A)** Considere o momento durante a execução desta função (compilada sem `-O2`) em que o IP está a apontar para a instrução que inicializa a variável de controlo do 1º ciclo `for` da função. **Identifique** claramente os principais componentes da *stack frame* associada a essa função, nesse momento.

d) **(R)** **Represente** graficamente a estrutura e o conteúdo da *stack frame* da função nesse momento, com o máximo detalhe que for possível (incluindo a localização dos apontadores relevantes).

Nº	**Nome**

e) **(R) Faça uma análise** comparativa qualitativa entre as 2 versões.

f) **(B) Faça uma estimativa** do nº de acessos à memória adicionais que a versão compilada sem optimização requer em cada iteração do 1º ciclo `for` dessa função (relativamente à versão compilada com -O2).

3. Considere a seguinte função para o cálculo da multiplicação de 2 matrizes quadradas:

```
// Calcula C=AxB, sendo A, B e C matrizes de inteiros com dimensão nxn
// Este produto de matrizes nao esta' optimizado

void mult_MxM(int* A, int* B, int* C) {

   for (int i=0; i<dim_M(int* C); i++) {
         for (int j=0; j<dim_M(int* C); j++) {
               for (int k=0; k<dim_M(int* C); k++) {
                     C[i*N+j] += A[i*N+k] * B[k*N+j];
               }
         }
   }
}
```

Nº	**Nome**

a) **(R)** **Modifique** este código introduzindo os 2 tipos de optimização mais óbvios e eficientes para melhorar consideravelmente o seu desempenho. **Justifique** a sua introdução.

b) **(B/E)** **Indique**, justificando, que outra modificação poderia ainda inserir para minimizar a *miss rate* da *cache* em matrizes grandes, sem recorrer à matriz transposta.

c) **(B/E)** Considere que o processador tem associado na mesma *chip* uma *cache* L1 de 16KB, só para dados, com linhas de 16 *bytes* e que não introduz atrasos no *pipeline* do CPU; mas quando há uma *cache miss*, são necessários 20 *clock cycles* para ler uma linha para a *cache* L1. Faça uma estimativa do impacto desta organização de memória no CPE da matriz C, para uma multiplicação de matrizes de 32x32 inteiros.

Código *assembly* da função compilada sem –O2:

```
loaddefault:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        movl    $0, -4(%ebp)
.L104:
        cmpl    $8, -4(%ebp)
        jle     .L107
        jmp     .L105
.L107:
        subl    $4, %esp
        movl    -4(%ebp), %eax
        movswl  default_reg(%eax,%eax),%eax
        pushl   %eax
        pushl   -4(%ebp)
        pushl   $1
        call    reg
        addl    $16, %esp
        leal    -4(%ebp), %eax
        incl    (%eax)
        jmp     .L104
.L105:
        movl    $0, -4(%ebp)
.L108:
        cmpl    $2, -4(%ebp)
        jle     .L111
        jmp     .L103
.L111:
        movl    $0, -8(%ebp)
.L112:
        cmpl    $9, -8(%ebp)
        jle     .L115
        jmp     .L110
.L115:
        subl    $4, %esp
        movl    -4(%ebp), %edx
        movl    %edx, %eax
        sall    $1, %eax
        addl    %edx, %eax
        sall    $2, %eax
        addl    -8(%ebp), %eax
        movswl  default_mem+4(%eax,%eax),%eax
        pushl   %eax
        movl    -4(%ebp), %edx
        movl    %edx, %eax
        sall    $1, %eax
        addl    %edx, %eax
        leal    0(,%eax,8), %edx
        movl    -8(%ebp), %eax
        sall    $1, %eax
        addw    default_mem(%edx), %ax
        movzwl  %ax, %eax
        pushl   %eax
        pushl   $1
        call    mem
        addl    $16, %esp
        leal    -8(%ebp), %eax
        incl    (%eax)
        jmp     .L112
.L110:
        leal    -4(%ebp), %eax
        incl    (%eax)
        jmp     .L108
.L103:
        leave
        ret
```

Código *assembly* da função compilada com –O2:

```
loaddefault:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %edi
        pushl   %esi
        pushl   %ebx
        subl    $12, %esp
        xorl    %edi, %edi
.L135:
        movswl  default_reg(%edi,%edi),%eax
        pushl   %ebx
        pushl   %eax
        pushl   %edi
        pushl   $1
        incl    %edi
        call    reg
        addl    $16, %esp
        cmpl    $8, %edi
        jle     .L135
        xorl    %edi, %edi
        xorl    %esi, %esi
.L145:
        xorl    %ebx, %ebx
.L144:
        leal    (%ebx,%esi,4), %eax
        movswl  default_mem+4(%eax,%eax),%eax
        pushl   %ecx
        pushl   %eax
        movw    default_mem(,%esi,8), %ax
        leal    (%eax,%ebx,2), %eax
        movzwl  %ax, %eax
        pushl   %eax
        pushl   $1
        incl    %ebx
        call    mem
        addl    $16, %esp
        cmpl    $9, %ebx
        jle     .L144
        incl    %edi
        addl    $3, %esi
        cmpl    $2, %edi
        jle     .L145
        leal    -12(%ebp), %esp
        popl    %ebx
        popl    %esi
        popl    %edi
        leave
        ret
```