Curso: LMCC **Exame 1ª Chamada** 15/Jun/05
Disciplina: Arquitectura de Computadores **Duração:** 2h30m

Nota: Apresente sempre o raciocínio ou os cálculos que efectuar; o não cumprimento desta regra equivale à não resolução do exercício. Use o verso do enunciado do exame como papel de rascunho.

1. Analisando o conteúdo de várias células de memória num PC, a partir do endereço `0x8c10280`, encontraram-se os seguintes valores (aqui representados em vários sistemas de numeração e separados por vírgulas):

$33,\ 107_{11},\ 63,\ 377_{8},\ 0x4d,\ 96,\ 164_{8},\ 1211_{4},\ 0x6d,\ 0_{6},\ 100_{2},\ 10,\ 2_{7},\ 86_{9},\ 12,\ 0x2a$

a) **(A/R)** Considere que o CPU (IA32) tinha acabado de executar um `pop %cx` quando se fez essa análise das células de memória (valor actualizado de `%esp`: `0x8c10284`). Sabendo que o compilador destinou o registo `%cx` para conter uma variável do tipo `short int` **indique, justificando**, qual o valor, em decimal, que essa variável toma após a execução dessa instrução de `pop`.

b) **(A)** Se agora o CPU executar uma instrução de `push %ebx`, **indique, justificando,** quais as células de memória que irão ser modificadas.

c) **(A/R)** Sabendo que no topo da *stack* (`%esp`: `0x8c10284`) se encontra uma *string*, **indique, justificando**, a dimensão da *string* e o seu conteúdo (o texto).

H	e	l	l	o		w	o	r	l	d	!
48	65	6c	6c	6f	20	77	6f	72	6c	64	21

d) **(R)** Considere a operação de *casting* do C, na conversão para inteiro do valor real (tipo *float*) armazenado em binário num registo com a seguinte combinação de bits: `0xf02c`. Sem efectuar qualquer operação aritmética, **mostre** qual o resultado da operação. **Apresente** o raciocínio seguido.

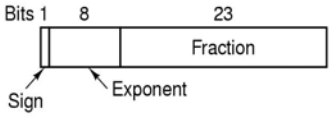
Nº	**Nome**	**Turma/Grupo:**

e) **(A/R)** O valor "1/10" é bastante comum em cálculo científico. **Mostre**, apresentando os cálculos que efectuar, a sua representação em binário num registo de FP do PC (valor do tipo *float*).

Notas de apoio (norma IEEE 754)

Normalized	±	0 < Exp < Max	Any bit pattern
Denormalized	±	0	Any nonzero bit pattern
Zero	±	0	0
Infinity	±	1 1 1…1	0
Not a number	±	1 1 1…1	Any nonzero bit pattern

Sign bit



(a)

Valor decimal de um fp em binário:

- precisão simples, normalizado: $V = (-1)^S * (1.F) * 2^{E-127}$
- precisão simples, desnormalizado: $V = (-1)^S * (0.F) * 2^{-126}$

2. Considere a função de soma dos elementos de um *array*, com 2 variantes no algoritmo/código em C (a que designaremos de `soma1` e `soma2`), e respectivo resultado da passagem por um compilador (as listagens em *assembly* estão na mesma folha anexa que o código em C).

a) **(A/R)** Considere `soma1`. **Identifique** o registo que o compilador reservou para conter o argumento que representa o índice do *array*, **comentando/anotando** as instruções em *assembly* em que este valor é usado.

b) **(R)** Considere `soma2`. **Identifique** no código *assembly* as instruções que foram geradas para executar o controlo do ciclo `for`, **comentando/anotando** essas instruções em *assembly*.

c) **(R)** Uma das tarefas associadas à gestão da invocação/retorno de funções numa linguagem imperativa está associada à salvaguarda/recuperação de registos. **Justifique** esta afirmação, **identifique** os registos que são modificados na função `soma1` e **explique** onde/como é feita a sua salvaguarda.

Nº	**Nome**	**Turma/Grupo:**

d) **(R)** As operações aritméticas/lógicas na arquitectura do IA32 apenas permitem especificar até 2 operandos. Contudo, no código *assembly* de `soma1` encontra-se uma instrução que efectua uma operação aritmética do tipo `a=b<op>c` (especificando explicitamente os 3 operandos). **Identifique-a (i)** no código em *assembly* e **(ii)** no código em C.

e) **(A)** Considere a execução da instrução de `soma1` que efectua em *assembly* a actualização do valor de retorno da função (soma do novo elemento do *array*), desde o fim da instrução anterior, e que:
- o conteúdo de cada célula de memória é igual ao *byte* menos significativo do seu endereço;
- os barramentos são de 32 bits;
- os registos `%eax,%ebx,%esi,%eip,%esp,` e `%ebp` contêm, respectivamente, os seguintes valores em hexadecimal: `0x8`, `0x202`, `0x8401220`, `0x8f04802`, `0x8c12004`, `0x8c12010` .

Mostre: **(i)** **(A/R)** todo o tráfego que circula no barramento de dados (cronologicamente, em hexadecimal, e com que finalidade), e **(ii)** **(B)** as alterações que deveriam existir em relação a esta instrução em *assembly* para que ela pudesse ser executada por uma arquitectura típica RISC (use a sintaxe do *assembler* do IA-32 da Gnu).

(i) **(A/R)**

(ii) **(B)**

Nº	**Nome**	**Turma/Grupo:**

f) **(R/B)** Represente a estrutura da *stack frame* - incluindo a localização e uma descrição sumária dos conteúdos - imediatamente antes de executar a instrução de `call` (conteudo de `%ebp` na alínea anterior).

g) **(R)** **Compare e comente** a provável diferença de desempenho entre estas 2 versões da função `soma`.

h) **(B)** **Identifique e liste**, para cada um dos elementos do array, as instruções que são executadas pelo CPU em cada uma das 2 versões da função `soma`.

i) **(B)** **Quantifique, justificando**, o nº de acessos à memória de dados (e apenas a esta) que o CPU vai necessitar em média, por cada elemento do *array*, nas 2 versões da função `soma`.

Nº	**Nome**	**Turma/Grupo:**

```
int soma1 (int *arr, int ind, int nelem)
{
   int ret=0;

   if (ind<nelem)
     ret = arr[ind] + soma(arr, ind+1, nelem);
   return (ret);
}
```

```
int soma2 (int *arr, int nelem)
{
   int i,ret=0;

   for (i=0;i<nelem;i++ )
   {
      ret += arr[i];
   }
   return (ret);
}
```

```
       .file  "soma1.c"
       .text
.globl _soma
       .def   _soma; .scl   2;     .type  32;
       .endef
_soma1:
       pushl  %ebp
       movl   %esp, %ebp
       subl   $20, %esp
       movl   %ebx, -8(%ebp)
       movl   %esi, -4(%ebp)
       movl   8(%ebp), %esi
       movl   12(%ebp), %ebx
       movl   16(%ebp), %edx
       movl   $0, %eax
       cmpl   %edx, %ebx
       jge    L2
       movl   %edx, 8(%esp)
       leal   1(%ebx), %eax
       movl   %eax, 4(%esp)
       movl   %esi, (%esp)
       call   _soma
       addl   (%esi,%ebx,4), %eax
L2:
       movl   -8(%ebp), %ebx
       movl   -4(%ebp), %esi
       movl   %ebp, %esp
       popl   %ebp
       ret
```

```
       .file  "soma2.c"
       .text
.globl _soma
       .def   _soma; .scl   2;     .type
       32;    .endef
_soma2:
       pushl  %ebp
       movl   %esp, %ebp
       pushl  %ebx
       movl   8(%ebp), %ebx
       movl   12(%ebp), %ecx
       movl   $0, %eax
       movl   $0, %edx
       cmpl   %ecx, %eax
       jge    L8
L6:
       addl   (%ebx,%edx,4), %eax
       incl   %edx
       cmpl   %ecx, %edx
       jl     L6
L8:
       popl   %ebx
       popl   %ebp
       ret
```

Nº	**Nome**	**Turma/Grupo:**