«num» «nome» **«lugar»**

Sistemas de Computação --- LEI --- U Minho --- 2022/23 --- 2023.06.22 --- EXAME

Notas: Para cada pergunta, apresenta a justificação da solução, incluindo o raciocínio ou os cálculos. Podes dar como resposta um valor numérico não simplificado (exemplo 15^{33}+73x18). Não são permitidas máquinas de calcular, computadores, telemóveis, tablets, etc. Os testes são de resolução individual. Qualquer tentativa de fraude académica pode implicar a abertura de um processo disciplinar. Ao realizares este teste, estás a aceitar esta possível implicação. Cotações (1) 2.75; (2) 2.5; (3) 3.0; (4) 1.75; (5) 3.25; (6) 2.5; (7) 1.5; (8) 2.75.

1. [1.0+1.0+0.75] Considera que um dado número natural é representado pelo padrão 001 0010 1101_2 (em binário natural).

(a) **Representa** esse número em base 8 (octal).

(b) **Representa** esse número em base 5.

(c) **Calcula** o valor correspondente a esse padrão, considerado que representa um número em excesso de 1024.

---

2. [1.25+1.25] Um repositório digital, usado por um jornalista profissional, contém 25.000 ficheiros com fotografias relativas aos concertos dos Coldplay em Coimbra. As fotografias têm todas as mesmas dimensões, numa proporção 3/2, ou seja, a largura é 1.5 vezes maior que a altura. As fotografias têm todas a mesma resolução, sendo que cada pixel da imagem ocupa 16 bits.

(a) Sabendo que as fotografias no total ocupam 3 GB, **indica** quais são a altura e a largura das fotografias.

(b) Para fazer caber todas as fotografias num dispositivo de armazenamento com 300MiB de capacidade, as fotografias foram redimensionadas para um formato quadrado (128*128) e cada pixel passou a ocupar 6 bits. **Indica** se esse dispositivo pode (ou não) armazenar TODAS as fotografias.

3. [1.0+1.0+1.0] Considera a versão com 12 bits para representação de números em vírgula flutuante (que segue os mesmos princípios da norma IEEE 754): 6 bits para o expoente (excesso de 31), 5 bits para a mantissa e 1 bit para o sinal.

(a) **Calcula** o valor (em decimal) do número representado pelo padrão hexadecimal $FC0_{16}$

(b) **Indica** se o valor $+6{,}125_{10}$ pode ser representado exatamente nesta versão da norma.

(c) **Representa e calcula** o valor (em decimal) do menor número positivo que pode ser representado nesta versão da norma.

4. [1.75] Explica o conceito de "*stored program computer*" e indica duas consequências práticas que dele decorrem.

«num» «nome» **«lugar»**

5. [1.0+0.75+1.0+0.5] Considera o código C da função `sc` e o respetivo código assembly gerado pelo gcc.

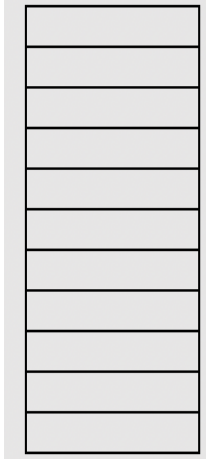
(a) **Desenha**, no quadro ao lado, o activation record (stack frame) que é criado quando se invoca a função. **Identifica** todos os campos, os respetivos offsets em relação a `ebp` e a posição para onde aponta este registo.

(b) Para as instruções C (linhas 2, 4, 5), **relaciona**-as com as instruções assembly. **Anota** o contributo de cada instrução assembly para a concretização da instrução C.

(c) **Explica** como a instrução C na linha 6 foi compilada.

(d) **Sugere** uma otimização que pode ser aplicada ao código assembly gerado para a linha 6.

```
 1: int sc(int n, int v[])
 2: { int i=n;
 3:   int soma=0;
 4:   while (i>=0) {
 5:     if (soma<=v[i])
 6:       soma+=v[i];
 7:     i--;
 8:   }
 9:   return (soma);
10: }
```



```
sc:   pushl  %ebp
      movl   %esp, %ebp
      subl   $16, %esp
      movl   8(%ebp), %eax
      movl   %eax, -4(%ebp)
      movl   $0, -8(%ebp)
      jmp    .L2
.L4:  movl   -4(%ebp), %eax
      leal   (,%eax,4), %edx
      movl   12(%ebp), %eax
      addl   %edx, %eax
      movl   (%eax), %eax
      cmpl   %eax, -8(%ebp)
      jg     .L3
      movl   -4(%ebp), %eax
      leal   (,%eax,4), %edx
      movl   12(%ebp), %eax
      addl   %edx, %eax
      movl   (%eax), %eax
      addl   %eax, -8(%ebp)
.L3:  subl   $1, -4(%ebp)
.L2:  cmpl   $0, -4(%ebp)
      jns    .L4
      movl   -8(%ebp), %eax
      leave
      ret
```

---

6. [0.5+0.75+0.75+0.5] Um processador tem endereços de 32 bits e contém blocos com 32 bytes. Tem também uma cache com 16 KiB. A memória é endereçada ao byte.

(a) **Calcula** quantos blocos da memória podem ser armazenados na cache.

(b) Se a cache usar mapeamento direto, **mostra** o formato dos endereços da memória principal (campos t, s, o) que permite mapeá-los para a cache.

(c) **Calcula**, em bytes, o tamanho da cache, considerando a existência do *valid bit*.

(d) **Repete** a questão (b), mas agora considera mapeamento 8-way set associative.

«num» «nome» **«lugar»**

7. [1.5] A ferramenta CodePal gera código assembly a partir de descrições textuais em inglês. O seguinte código foi gerado para um pedido de criação de uma função que soma 100 números inteiros (32 bits) de um array, como a figura mostra. O código gerado apresenta algumas divergências face ao que seria expectável. Indica três aspetos do programa que deviam ser modificados/acrescentados para estar em concordância com o que foi pedido.


```
_start:
    movl  $100, %ecx
    movl  arr, %esi
    movl  sum, %ebx
sum_loop:
    addl  (%esi), %eax
    incl  %esi
    decl  %ecx
    jnz   sum_loop
```

8. [1.0+0.5+0.75+0.5] Considera o seguinte código em C e parte do respetivo código máquina IA32.

```
#include <stdio.h>
int cincoX (int a) {
  return (5*a);
}
int opera(int m, int n, int o) {
  if (m<n+o) return (cincoX(m+n));
  return (0);
}
main() {
  int res;
  res = opera(7, 8, 9);
  printf("Result: %d\n", res);
}
```

```
8048344 <cincoX>:
8048344: 55              pushl %ebp
8048345: 89 e5           movl  %esp, %ebp
8048347: 8b 45 08        movl  8(%ebp), %eax
804834a: 8d 04 80        leal  (%eax,%eax,4), %eax
804834d: c9              leave
804834e: c3              ret

804834f <opera>:
804834f: 55              pushl %ebp
8048350: 89 e5           movl  %esp, %ebp
8048352: 83 ec 08        subl  $8, %esp
8048355: 8b 4d 08        movl  8(%ebp), %ecx
8048358: 8b 55 0c        movl  12(%ebp), %edx
804835b: 89 d0           movl  %edx, %eax
804835d: 03 45 10        addl  16(%ebp), %eax
8048360: 39 c1           cmpl  %eax, %ecx
8048362: 7d 0e           jge   8048372 <opera+0x23>
8048364: 83 ec 0c        subl  $12, %esp
8048367: 8d 04 0a        leal  (%edx,%ecx,1), %eax
804836a: 50              pushl %eax
804836b: e8 d4 ff ff ff  call  8048344 <cincoX>
8048370: eb 05           jmp   8048377 <opera+0x28>
8048372: b8 00 00 00 00  movl  $0, %eax
8048377: c9              leave
8048378: c3              ret

8048379 <main>:
8048379: 55              pushl %ebp
804837a: 89 e5           movl  %esp, %ebp
804837c: 83 ec 08        subl  $8, %esp
804837f: 83 e4 f0        andl  $0xfffffff0,%esp
8048382: 83 ec 04        subl  $4, %esp
8048385: 6a 09           pushl $9
8048387: 6a 08           pushl $8
8048389: 6a 07           pushl $7
804838b: e8 bf ff ff ff  call  804834f <opera>
8048390: 83 c4 08        addl  $8,%esp
 ...
804839e: c9              leave
804839f: c3              ret
```

(a) **Constrói**, a partir da posição de memória `0x32F0D858`, o esquema da pilha relativo à execução da função `opera(7,8,9)` e eventual consequente execução da função `cincoX`. Assume que o valor de `ebp`, quando a função `opera` é chamada, é `0x32F0D868`.

esquema da pilha

Endereço memória (hexadecimal)	**conteúdo**
32F0D820	
32F0D824	
32F0D828	
32F0D82C	
32F0D830	
32F0D834	
32F0D838	
32F0D83C	
32F0D840	
32F0D844	
32F0D848	
32F0D84C	
32F0D850	
32F0D854	
32F0D858	

(b) **Indica** o valor do registo `eax`, imediatamente após a execução da instrução `804835d: addl 16(%ebp), %eax`.

`eax =`

(c) **Indica** o valor do registo `eax`, imediatamente após a execução da instrução `804834a: leal (%eax,%eax,4),%eax`.

`eax =`

(d) **Explica** a razão para a instrução `8048362: jge 8048372` ser codificada com `7d 0e`.