«num» «nome» **«lugar»**

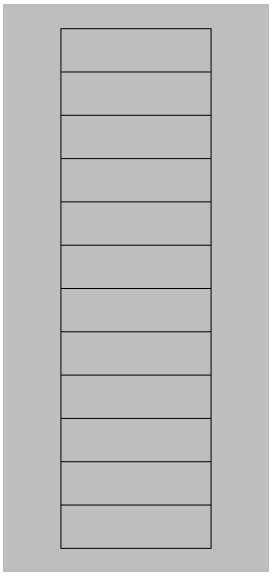
Sistemas de Computação --- LEI --- U Minho --- 2022/23 --- 2023.06.01 --- 2.º TESTE

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) 1.0+1.0+0.75+0.75; (2) 0.75+0.75+0.75; (3) 0.75+0.5; (4) 1.0+0.5+0.75+0.75.

1. [1.0+1.0+0.75+0.75] Considera o código C da função `contaNG` e o código assembly gerado pelo gcc.

(a) **Desenha**, no quadro abaixo, 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 4, 5, 7, 10) **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** duas otimizações que podem ser aplicadas ao código assembly.

```
 1: int contaNG(int ref, int n, int v[])
 2: { int i=1;
 3:   int conta=0;
 4:   if (ref>v[0])
 5:     while (i<n) {
 6:       if (v[i]>=ref)
 7:         conta++;
 8:       ++i;
 9:     }
10:   return (conta);
11: }
```



```
contaNG:
     pushl  %ebp
     movl   %esp, %ebp
     subl   $16, %esp
     movl   $1, -4(%ebp)
     movl   $0, -8(%ebp)
     movl   16(%ebp), %eax
     movl   (%eax), %eax
     cmpl   %eax, 8(%ebp)
     jle    .L2
     jmp    .L3
.L5: movl   -4(%ebp), %eax
     leal   0(,%eax,4), %edx
     movl   16(%ebp), %eax
     addl   %edx, %eax
     movl   (%eax), %eax
     cmpl   %eax, 8(%ebp)
     jg     .L4
     addl   $1, -8(%ebp)
.L4: addl   $1, -4(%ebp)
.L3: movl   -4(%ebp), %eax
     cmpl   12(%ebp), %eax
     jl     .L5
.L2: movl   -8(%ebp), %eax
     leave
     ret
```

---

2. [0.75+0.75+0.75] Um processador tem uma cache *8-way set associative* que contém 8 sets (conjuntos). A memória é endereçada ao byte, tem capacidade de 256KiB e contém 1024 blocos. As palavras têm 32 bits.
(a) **Mostra** o formato dos endereços da memória principal (campos t, s, o) que permite mapeá-los para a cache.
(b) **Calcula**, em bytes, o tamanho da cache, considerando a existência do *valid bit*.
(c) Num dado momento, numa das linhas do conjunto 3, com valid bit ativado, está guardada a tag 73_{16}. **Indica** quais os endereços de memória cujos conteúdos estão armazenados nessa linha.

«num» «nome» **«lugar»**

3. [0.75+0.5] Considera que os registos `eax, ebx, esi, edi` contêm respetivamente os valores $20D04050_{16}$, 7, $8313150C_{16}$ e 38, e que parte da memória tem o conteúdo mostrado na figura ao lado. Considera ainda as duas seguintes instruções:

```
subl _____(%esi), %edi
addl %eax, (%esi,%ebx,4)
```

(a) **Completa** a instrução `subl` acima de modo a ficar armazenado o valor 33 no registo `edi`. **Explica** como chegaste a esse resultado.

endereço (hexadecimal)	conteúdo	novo conteúdo
83131522	0x00	
83131523	0x05	
83131524	0x00	
83131525	0x00	
83131526	0x00	
83131527	0x05	
83131528	0x22	
83131529	0x33	
8313152A	0x55	
8313152B	0x77	

(b) A instrução `addl` altera o conteúdo da memória. **Apresenta**, na coluna "novo conteúdo", quais as células de memória que são modificadas e que valores são lá armazenados.

---

4. [1.0+0.5+0.75+0.75] Considera o seguinte código em C e o respetivo código máquina IA32.

```
#include <stdio.h>
int triplo (int a) {
  return (3*a);
}
int conta(int m, int n, int o) {
  return (triplo(m*n)+o);
}
int main(void)
{ int mult1, mult2, off, res;
  scanf("%d %d %d", &mult1, &mult2, &off);
  res = conta(mult1,mult2,off);
  printf("Result: %d\n",res);
  return 0;
}
```

```
0804837c <triplo>:
804837c: 55              pushl  %ebp
804837d: 89 e5           movl   %esp, %ebp
804837f: 8b 55 08        movl   8(%ebp), %edx
8048382: 89 d0           movl   %edx, %eax
8048384: 01 c0           addl   %eax, %eax
8048386: 01 d0           addl   %edx, %eax
8048388: 5d              popl   %ebp
8048389: c3              ret
0804838a <conta>:
804838a: 55              pushl  %ebp
804838b: 89 e5           movl   %esp, %ebp
804838d: 8b 45 08        movl   8(%ebp), %eax
8048390: 0f af 45 0c     imull  12(%ebp), %eax
8048394: 50              pushl  %eax
8048395: e8 e2 ff ff ff  call   804837c <triplo>
804839a: 83 c4 04        addl   $4,%esp
804839d: 8b 55 10        movl   16(%ebp), %edx
80483a0: 01 d0           addl   %edx, %eax
80483a2: c9              leave
80483a3: c3              ret
080483a4 <main>:
80483a4: 55              pushl  %ebp
80483a5: 89 e5           movl   %esp, %ebp
...
80483d0: ff 75 f4        pushl  -12(%ebp)
80483d3: ff 75 f8        pushl  -8(%ebp)
80483d6: ff 75 fc        pushl  -4(%ebp)
80483d9: e8 ac ff ff ff  call   804838a <conta>
80483de: 50              pushl  %eax
...
80483fc: c9              leave
80483fd: c3              ret
```

(a) **Completa** o esquema da pilha relativo à execução da função `conta(12,10,5)` e consequente chamada da função `triplo`. Assume que o valor de `ebp`, quando a função `conta` é chamada, é `0x32FFD858`. O esquema já contém o primeiro argumento a ser colocado na pilha (5, i.e., o argumento mais à direita).

(b) **Indica** o valor do registo `eax`, imediatamente após a execução da instrução `8048384: addl %eax, %eax`.

`eax` =

(c) **Indica** o valor do registo `esp`, imediatamente após a execução da instrução `804839a: addl $4, %esp`.

`esp` =

(d) A instrução `80483d9: call conta` é codificada com `e8 ac ff ff ff`. **Justifica** essa codificação.

esquema da pilha

Endereço memória (hexadecimal)	conteúdo
32FFD82C	
32FFD830	
32FFD834	
32FFD838	
32FFD83C	
32FFD840	
32FFD844	
32FFD848	
32FFD84C	
32FFD850	5