**Sistemas de Computação ::: LEI ::: U Minho ::: 2023/24 ::: 2024.05.23 ::: 2.º TESTE**

**Notas**: Coloca o teu nome e número nas quatro páginas que compõem este teste.
Para cada pergunta, apresenta a justificação da solução, incluindo o raciocínio ou os cálculos que efetuares. Podes dar como resposta um valor numérico não simplificado (exemplo $15^{33}$+73/18). 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.

**[2.5 valores] E.** Um processador tem uma cache de mapeamento direto, com 32 linhas, cada uma com 32 palavras. A memória contém 2048 blocos. As palavras são endereçadas ao byte e ocupam 2 bytes.
(E1) **Qual é o tamanho da memória em bytes**? [0.5 val.]
(E2) **Desenha o formato dos endereços da memória principal** (incluindo os campos t, s, o) que permite mapeá-los para a cache. [0.5 val.]
(E3) **Qual é, em <u>bytes</u>, o tamanho da cache**, assumindo que cada linha tem dois bits de controlo (valid e dirty)? [0.75 val.]
(E4) **Indica dois endereços de memória seguidos**, que sejam mapeados para duas linhas diferentes da cache. [0.75 val.]

---

**[1 valor] F.** Pergunta sobre otimizações.
Considere a seguinte função, escrita em C, que multiplica duas matrizes quadradas A e B de tamanho NxN para produzir uma matriz C. Identifica e explica as possíveis ineficiências relacionadas com a cache, nomeadamente como os padrões de acesso às matrizes A, B e C afetam o desempenho da cache.

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

**[2.5 valores] G.** Considera a função `cntRange` codificada em C e o código assembly gerado pelo gcc (opção de optimização -O0).
(G1) **Desenha o stack frame desta função**, até ao instante em que as variáveis locais são alocadas. [1 val.]
(G2) **Relaciona as partes (1), (2), (3) e (4)** (marcadas no código C a vermelho) **com as respetivas instruções do código assembly**. Faz o relacionamento no código assembly, indicando junto das linhas respetivas que partes dessas instruções C são abrangidas. [1.5 val.]

```c
int cntRange(int li, int top, int *val) {
  int i, count=0;1
  for (i=0; i<=li;2 i++)
    if (val[i]>=0 && val[i]<=top)3
      count++;
  return count;4
}
```

```
# === gcc -O0 -S ... ===
cntRange(int, int, int*):
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        movl    16(%ebp), %eax
        movl    12(%ebp), %eax
        movl    8(%ebp), %eax
        movl    $0, -8(%ebp)
        movl    $0, -4(%ebp)
.LBB0_1: movl   -4(%ebp), %eax
        cmpl    8(%ebp), %eax
        jg      .LBB0_7
        movl    16(%ebp), %eax
        movl    -4(%ebp), %ecx
        cmpl    $0, (%eax,%ecx,4)
        jl      .LBB0_5
        movl    16(%ebp), %eax
        movl    -4(%ebp), %ecx
        movl    (%eax,%ecx,4), %eax
        cmpl    12(%ebp), %eax
        jg      .LBB0_5
        movl    -8(%ebp), %eax
        addl    $1, %eax
        movl    %eax, -8(%ebp)
.LBB0_5: jmp    .LBB0_6
.LBB0_6: movl   -4(%ebp), %eax
        addl    $1, %eax
        movl    %eax, -4(%ebp)
        jmp     .LBB0_1
.LBB0_7: movl   -8(%ebp), %eax
        addl    $8, %esp
        popl    %ebp
        ret
```

**[1.5 valores] H.** Considera que parte da memória tem os valores mostrados na figura ao lado, que o registo `%edi` tem o valor $20_{16}$ (hexadecimal) e ainda as seguintes instruções em *assembly*:

```
movl (%esi,%edi,4), %eax
movl $30, %ecx
subl %ecx, %eax
```

| endereço | conteúdo |
|---|---|
| 0x12AA0082 | 0x83 |
| 0x12AA0083 | 0x42 |
| 0x12AA0084 | 0x77 |
| 0x12AA0085 | 0xAA |
| 0x12AA0086 | 0x77 |
| 0x12AA0087 | 0xAA |
| 0x12AA0088 | 0x77 |
| 0x12AA0089 | 0x52 |
| 0x12AA008A | 0x83 |
| 0x12AA008B | 0x15 |

(H1) **Qual é o conteúdo do registo `%esi`**, se, no final da execução da 1.ª instrução `movl`, o registo `%eax` tem o valor `0x77AA77AA`, que foi obtido na parte da memória mostrada? Apresenta o teu raciocínio. [0.75 val.]

(H2) **Qual é, em hexadecimal, o valor armazenado no registo `%eax`**, após a execução da instrução `subl`. Assume que no início da execução da instrução, o valor de `%eax` tem é o indicado em H1. Apresenta os cálculos efetuados. [0.75 val.]

**[2.5 valores] I.** Considera duas funções C e o respetivo código máquina IA32.

```
int f2(int n) {
  if (n==0)
    return 0;
  return n*34;
}

int f1(int n) {
  if (n>5)
    value=f2(n+1);
  else
    value=0;
  return value;
}
```

| **Endereço** (hexadecimal) | **conteúdo** |
|---|---|
| FF0FC4 | |
| FF0FC8 | |
| FF0FCC | |
| FF0FD0 | |
| FF0FD4 | |
| FF0FD8 | |
| FF0FDC | |
| FF0FE0 | |
| FF0FE4 | |
| FF0FE8 | |
| FF0FEC | |
| FF0FF0 | |
| FF0FF4 | |
| FF0FF8 | |
| FF0FFC | |
| FF1000 | |
| FF1004 | |

```
0x080483e4:  f2:   pushl   %ebp
0x080483e5:        movl    %esp, %ebp
0x080483e7:        pushl   %eax
0x080483e8:        movl    8(%ebp), %eax
0x080483eb:        cmpl    $0, 8(%ebp)
0x080483ef:        jne     .L02
0x080483f1:        movl    $0, -4(%ebp)
0x080483f7:        jmp     .L03
0x080483f9:  .L02: imull   $34, 8(%ebp), %eax
0x080483ff:        movl    %eax, -4(%ebp)
0x08048402:  .L03: movl    -4(%ebp), %eax
0x08048405:        addl    $4, %esp
0x08048409:        popl    %ebp
0x0804840a:        ret
0x0804840b:  f1:   pushl   %ebp
0x0804840c:        movl    %esp, %ebp
0x0804840e:        pushl   %ebx
0x0804840f:        subl    $20, %esp
0x08048412:  .Lt5: movl    8(%ebp), %eax
0x08048415:        cmpl    $5, 8(%ebp)
0x08048418:        jle     .L12
0x0804841a:        movl    -12(%ebp), %ebx
0x0804841d:        movl    8(%ebp), %eax
0x08048420:        addl    $1, %eax
0x08048422:        movl    %eax, (%esp)
0x08048425:        call    f2
0x0804842a:        movl    %eax, -8(%ebp)
0x0804842d:        jmp     .L13
0x0804842f:  .L12: movl    $0, -8(%ebp)
0x08048435:  .L13: movl    -8(%ebp), %eax
0x08048438:        addl    $20, %esp
0x0804843b:        popl    %ebx
0x0804843c:        popl    %ebp
0x0804843d:        ret
```

(I1) Assume que foi feita uma chamada a `f1(9)`, que vai desencadear a chamada a `f2(10)` e por aí fora. Para `f1(9)`, os valores de `%ebx`, `%esp` e `%ebp`, antes da invocação, são `4`, `0xFF1004` e `0xFFFF1022` e o endereço de regresso é `0x080455C9`. **Preenche o conteúdo da pilha** para parte da execução dessa sequência de chamadas, ou seja, até ao momento que antecede a execução da instrução `return n*34` no contexto de `f2`. [1.5 val.]

(I2) **Quais são os valores de `%esp` e `%ebp`**, imediatamente antes da execução da instrução `call f2` no contexto `f1(9)`? [1.0 val.] `%esp` = 0x_____ `%ebp` = 0x_____