NOME: N.º: v1

Sistemas de Computação ::: LEI ::: U Minho ::: 2021/22 ::: 2022.05.24 ::: TESTE v1

Notas: Coloca o teu nome e número nas quatro páginas de 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 simplifica (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.

[1.5 valores] 1. O pipelining é um mecanismo que pode ser usado para explorar o paralelismo ao nível das instruções. Explica de forma sucinta como é que esse mecanismo acelera a execução das instruções e indica um problema que advém da sua utilização.

[2 valores] 2. Considera uma imagem digital em tons de cinza, de dimensões 1024*512, que utiliza 6 bits para codificar a informação de cada pixel.
[1 val.] (a) Calcula em KiB (kibibytes) o tamanho dessa imagem.
[1 val.] (b) Através de uma aplicação tipo PhotoShop, as dimensões da imagem foram reduzidas de forma proporcional e passou a usar-se apenas 3 bits para codificar a informação dos pixels. Qual a dimensão final se a imagem passou a ocupar 12 KiB?

[2 valores] 3. Uma instituição bancária aceita depósitos ao balcão até 10 mil euros. O montante (em cêntimos) de cada depósito é codificado em binário, pelo sistema informático.
[1 val.] (a) Quantos bits são necessários para representar esses montantes se se usar complemento para 2?
[1 val.] (b) A legislação mudou e, para evitar lavagens de dinheiro, limitou esses depósitos a 3 mil euros. A instituição aproveitou esta alteração para assumir que os montantes (em cêntimos) são representados por números naturais (inteiros sem sinal). Quantos bits são agora necessários para representar um desses montantes?

NOME: N.º: v1

[3 valores] 4. Considera uma versão reduzida da norma IEEE 754 com 10 bits (4 bits para o expoente em excesso de 7, 5 bits para a mantissa e 1 bit para o sinal; não esquecer os casos de exceção). O valor decimal de um número normalizado representado com este formato vem dado por $V = (-1)^s \times (1+f) \times 2^{e-7}$.
[1 val.] (a) É possível representar exatamente (i.e., sem arredondamentos) o valor $+88_{10}$?
[1 val.] (b) Qual é o valor (em decimal) do número representado pelo padrão hexadecimal $2E8_{16}$?
[1 val.] (c) Qual é o valor (em decimal) do maior número positivo subnormal?

[3.5 valores] 5. Um processador tem uma cache *4-way set associative* que contém 8 conjuntos. A memória contém 256 blocos, cada um com 32 palavras. As palavras têm 32 bits, mas são endereçadas ao byte.
[0.75 val.] (a) Mostra o formato dos endereços da memória principal (incluindo os campos t, s, o) que permite mapeá-los para a cache.
[1 val.] (b) Qual é, em bytes, o tamanho da cache, considerando a existência do *valid bit*?
[0.75 val.] (c) Para que conjunto é mapeado o endereço de memória 671_8?
[1 val.] (d) Qual o miss ratio de um programa que itera cinco vezes nas instruções que estão armazenadas nas posições 108 a 155. Assume que as instruções ocupam quatro bytes e que nenhuma delas acede à memória para ler/escrever dados.

NOME: N.º: v1

[2 valores] 6. Considera o código C da função contaPrimeiros e o código assembly gerado pelo gcc. Identifica 5 otimizações relacionadas com argumentos e variáveis locais que o compilador aplicou com a opção -O2 (e não fez com -O0). A identificação deve ser feita indicando as linhas do código do lado esquerdo e as linhas do código do lado direito e referindo que tipo de otimização foi feita.

```
int contaPrimeiros(int n, int val[])
{ int i, soma=0;
  if(n<10) {
    for(i=0; i<n; ++i)
      soma += val[i];
    }
  return soma;
}
```

```
# === gcc -O0 -S ... ===
 1  contaPrimeiros:
 2       pushl   %ebp
 3       movl    %esp, %ebp
 4       subl    $8, %esp
 5       movl    $0, -8(%ebp)
 6       cmpl    $9, 8(%ebp)
 7       jg      .L2
 8       movl    $0, -4(%ebp)
 9  .L3: movl    -4(%ebp), %eax
10       cmpl    8(%ebp), %eax
11       jl      .L6
12       jmp     .L2
13  .L6: movl    -4(%ebp), %eax
14       leal    0(,%eax,4), %edx
15       movl    12(%ebp), %eax
16       movl    (%eax,%edx), %edx
17       leal    -8(%ebp), %eax
18       addl    %edx, (%eax)
19       leal    -4(%ebp), %eax
20       incl    (%eax)
21       jmp     .L3
22  .L2: movl    -8(%ebp), %eax
23       leave
24       ret
```

```
# === gcc -O2 -S ... ===
25  contaPrimeiros:
26       pushl   %ebp
27       movl    %esp, %ebp
28       movl    8(%ebp), %ecx
29       xorl    %eax, %eax
30       cmpl    $9, %ecx
31       pushl   %ebx
32       movl    12(%ebp), %ebx
33       jg      .L2
34       xorl    %edx, %edx
35       cmpl    %ecx, %eax
36       jge     .L2
37  .L7: addl    (%ebx,%edx,4), %eax
38       incl    %edx
39       cmpl    %ecx, %edx
40       jl      .L7
41  .L2: popl    %ebx
42       leave
43       ret
```

NOME: N.º: v1

[2 valores] 7. Considera que os registos `%esi, %edi` contêm respetivamente os valores `0x8313150F` e `6`, e que parte da memória tem os valores mostrados na figura ao lado. Considera ainda as seguintes instruções em *assembly*:

```
subl $3, %edi
movl (%esi,%edi,2), %eax
movl %edi, 9(%esi)
```

endereço	conteúdo	novo conteúdo
0x83131512	0x22	
0x83131513	0x33	
0x83131514	0x66	
0x83131515	0x88	
0x83131516	0xAA	
0x83131517	0xCC	
0x83131518	0x22	
0x83131519	0x33	
0x8313151A	0x55	
0x8313151B	0x77	

[1 val.] (a) Qual é o conteúdo do registo de `%eax` após a execução destas instruções? Apresenta os cálculos que efetuares.

[1 val.] (b) A terceira instrução 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 valores] 8. Considera o seguinte código em C e o respetivo código máquina IA32.

```
int bar (int a, int b) {
  return a + b;
}
int foo(int n, int m, int c) {
  c += bar(m, n);
  return c;
}
```

```
08048374 <bar>:
8048374: 55                pushl %ebp
8048375: 89 e5             movl %esp,%ebp
8048377: 8b 45 0c          movl 12(%ebp),%eax
804837a: 03 45 08          addl 8(%ebp),%eax
804837d: 5d                popl %ebp
804837e: c3                ret
0804837f <foo>:
804837f: 55                pushl %ebp
8048380: 89 e5             movl %esp,%ebp
8048382: 83 ec 08          subl $8,%esp
8048385: 8b 45 08          movl 8(%ebp),%eax
8048388: 89 44 24 04       movl %eax,4(%esp)
804838c: 8b 45 0c          movl 12(%ebp),%eax
804838f: 89 04 24          movl %eax,(%esp)
8048392: e8 dd ff ff ff    call 8048374 <bar>
8048397: 03 45 10          addl 16(%ebp),%eax
804839a: c9                leave
804839b: c3                ret
```

ESQUEMA DA PILHA (MEMÓRIA)

Endereço memória (hexadecimal)	**conteúdo**
FFFFD82C	
FFFFD830	
FFFFD834	
FFFFD838	
FFFFD83C	
FFFFD840	
FFFFD844	
FFFFD848	
FFFFD84C	
FFFFD850	6

[2 val.] (a) Mostra o conteúdo completo da pilha (stack) para a execução da função `foo(4,5,6)`. Assume que o valor de `%ebp` quando foi chamada é `0xFFFFD858` e que o endereço de regresso na função que chamou `foo` é `0x080483C9`. O esquema já contém o primeiro argumento (6, i.e. o argumento mais à direita) a ser colocado na pilha para a execução da função `foo`.

[0.5 val.] (b) Qual é o valor de `%esp`, imediatamente antes da execução da instrução `ret` em `bar`?

`%esp` = 0x_____

[0.5 val.] (c) Qual é o valor de `%ebp`, imediatamente antes da execução da instrução `ret` em `bar`?

`%ebp` = 0x_____

[1 val.] (d) Justifica por que razão a instrução `call 8048374` (que permite chamar a função `bar`) é codificada com os bytes `e8 dd ff ff ff` (notação hexadecimal).