isep Instituto Superior de Engenharia do Porto

DEI / Licenciatura em Engenharia Informática
Arquitectura de Computadores
Exame Época Recurso – Fevereiro 2016

- Consulta apenas das 2 folhas de consulta (C e Assembly).
- A infracção implica, no mínimo, a ANULAÇÃO da prova.
- Quando omissa a arquitectura, considere Linux/IA32.

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

	V	F
1) Em C, o *cast* de uma variável do tipo `int` para uma do tipo `float` altera o padrão de bits da variável	☒	☐
2) Em C, o *cast* implícito em determinadas situações de variáveis com sinal para valores sem sinal pode levar a *bugs* no programa	☒	☐
3) Admita um `int x` com valor `0x01234567` e um valor dado por `&x` de `0x100`. Logo, o valor presente no byte `0x100` é `0x67`	☒	☐
4) Em C, se tivermos uma variável x do tipo `short` com o valor `0x1234`, o valor `-0x1234` pode ser obtido através de `~x + 1`	☒	☐
5) Em C, a adição de duas variáveis `u` e `v` do tipo `int` tem como resultado `(u+v) mod 32`	☒	☐
6) Em C, é garantido que o resultado de uma divisão inteira por 2^k, obtida através de `u >> k`, é correctamente arredondado se `u<0`	☐	☒
7) Admita que `ptr` é uma variável do tipo `char*`. Então, a expressão `(int*)ptr + 7` avança 28 bytes na memória	☒	☐
8) Em Assembly, a instrução `movb (%esi), (%edi)` permite copiar um byte para uma nova posição de memória numa única instrução	☐	☒
9) Em Assembly, o resultado das instruções de salto condicional depende do valor dos bits do registo EFLAGS	☒	☐
10) Admita que `%edi` e `int *ptr` armazenam o endereço do inteiro `x`. Então, `movl $1,(%edi)` é o equivalente a `*ptr = 1` em C	☒	☐
11) Os parâmetros de uma função não podem ser acedidos usando o registo `%esp` em vez do `%ebp` como base do endereçamento	☐	☒
12) Admita `0xF000` e `0x0100` em `%edx` e `%ecx`, respetivamente. `leal (%edx,%ecx,4),%esi` armazena em `%esi` o valor `0xF400`.	☒	☐
13) Em IA32 é usada a *stack* para armazenar o valor de retorno de uma função, à semelhança do que acontece com o seu endereço de retorno	☐	☒
14) Admita que o valor de `%esp` é `0x100C`. A execução da instrução `ret` coloca o valor de `%esp` em `0x1010`	☒	☐
15) Os registo `%eax` é local a cada uma das funções, o que dispensa qualquer cuidado no seu uso entre invocações de funções	☐	☒
16) Admita a matriz global `short int m[5][3]`. Em Assembly, acedemos ao valor de `m[3][0]` avançando 18 bytes a partir de `m`	☒	☐
17) Uma estrutura, alinhada de acordo com as regras estudadas, com um vector de 2 `char`, 1 `int` e 1 `short` (por esta ordem) ocupa 12 bytes.	☒	☐
18) É **sempre** possível diminuir o tamanho de um estrutura alinhada alterando a ordem dos seus campos	☐	☒
19) É possível redimensionar, com a função `realloc`, o tamanho um vetor de inteiros `vec` declarado estaticamente com `int vec[10]`	☐	☒
20) A possibilidade de existirem diversas referências para a mesma posição de memória dificulta a optimização efectuada pelo compilador	☒	☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere o código da função `sum` ao lado que pretende somar os elementos de um vetor `a`. O número de elementos do vetor é passado no parâmetro `unsigned int length`.

Quando invocada com o valor 0 no argumento `length`, a função deveria retornar 0.0. No entanto, é gerado um erro de acesso à memória.

```
float sum(float a[], unsigned int length){
   int i;
   float result = 0.0;

   for(i=0; i<= length-1; i++)
      result += a[i];
   return result;
}
```

[1v] **a)** Explique detalhadamente porque o erro acontece.

Esta função demonstra claramente os *bugs* que podem surgir decorrentes do *cast* implíctio de variáveis *signed* para *unsigned*. É válido (e até frequente) usar o parâmetro *length* como *unsigned* uma vez que não é admissível que sejam uados valores negativos para o tamanho do vetor. Do mesmo modo, a condição de paragem do ciclo (*i<=length-1*) está também correcta. A combinação das duas é que pode levar a um resultado que não o esperado!

Uma vez que o parâmetro *length* é *unsigned*, o cálculo 0-1 na condição de paragem do ciclo é realizado usando aritmética sem sinal, equivalente à adição modular. O resultado é então UMAX. A comparação <= é também realizada usando artimética sem sinal. Uma vez que qualquer número inteiro é sempre menor ou igual a UMAX, a comparação é sempre verdadeira. Logo, o código mostrado tenta aceder a posições inválidas do vetor *a* asism que a variável *i* assumer valores para além dos limites do vetor.

[1v] **b)** Demonstre como o código poderia ser corrigido. **Justifique a sua resposta.**

O código pode ser corrigido declarando *length* como *int* no cabeçalho da função ou alterando a condição de paragem do ciclo para *i* < *length* pelas razões descritas na resposta anterior.

[5v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere as seguintes declarações:

```
typedef struct {
    short int code;
    long int start;
    char raw[3];
    double data;
} OldSensor;
```

```
typedef struct {
    short int code;
    short int start;
    char raw[5];
    short int sense;
    short int ext;
    double data;
} NewSensor;
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `OldSensor`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas mas não usadas para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

```
typedef struct {
    short int code;         0x100 : 2 bytes
    [gap]                   0x102 : 2 bytes
    long int start;         0x104 : 4 bytes
    char raw[3];            0x108 : 3 bytes
    [gap]                   0x10B : 1 byte
    double data;            0x10C : 8 bytes
} OldSensor;

Tamanho da estrutura: 20 bytes
```

[1.5v] **b)** Se definirmos os campos da estrutura `OldSensor` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta** e indique, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho, o novo endereço de cada campo e das partes alocadas mas não usadas, bem como o novo tamanho total da estrutura.

```
typedef struct {
    double data;            0x100 : 8 bytes
    long int start;         0x108 : 4 bytes
    short int code;         0x10C : 2 bytes
    char raw[3];            0x10E : 3 bytes
    [gap]                   0x111 : 3 bytes
} OldSensor;

Tamanho da estrutura: 20 bytes
```

Não é possível diminuir o tamanho desta estrutura, mesmo ordenando de forma decrescente os seus campos em função do seu tamanho, uma vez que o seu tamanho total tem de ser múltiplo de k=4.

[2v] **c)** Considere o seguinte fragmento de código em C, respeitando as declarações iniciais das estruturas.

```
void xpto(OldSensor *oldData){
   NewSensor *newData;

   /* zeros out all the space of oldData */
   bzero((void *)oldData,sizeof(OldSensor));

   oldData->code = 0x104f;
   oldData->start = 0x80501ab8;
   oldData->raw[0] = 0xe1;
   oldData->raw[1] = 0xe2;
   oldData->raw[2] = 0x8f;
   oldData->data = 1.5;

   newData = (NewSensor *) oldData;
   ...
}
```

Admita que após estas linhas de código começamos a aceder aos campos da estrutura `NewSensor` através da variável `newData`. Indique, em hexadecimal, o valor de cada um dos campos de `newData` indicados a seguir. Tenha em atenção a ordenação dos bytes em memória em Linux/IA32!

a) `newData->code` = **0x104f**
b) `newData->raw[0]` = **0xb8**
c) `newData->raw[2]` = **0x50**
d) `newData->raw[4]` = **0xe1**
e) `newData->sense` = **0x008f**

```
typedef struct {
    short int code;         0x100 : 2 bytes
    short int start;        0x102 : 2 bytes
    char raw[5];            0x104 : 5 bytes
    [gap]                   0x109 : 1 byte
    short int sense;        0x10A : 2 bytes
    short int ext;          0x10C : 2 bytes
    double data;            0x10E : 8 bytes
    [gap]                   0x116 : 2 bytes
} NewSensor;
```

[2v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Admita a existência de um vetor `v` preenchido com um número arbitrário de inteiros positivos e cuja última posição preenchida tem o valor -1. O código seguinte em C determina a soma de todos os seus valores positivos.

```
void sum_elements(int *v, int *sum){
    int i, val;
    *sum = 0;
    for(i = 0; i < vec_length(v); i++){
       get_element(v, i, &val);
       *sum += val;
    }
}
```

```
void get_element(int *v, int i, int *val){
   *val = v[i];
}

int vec_length(int *v){
   int i=0, length=0;
   while(v[i++]!= -1)
      length++;
   return length;
}
```

Reescreva a função `sum_elements` em C usando as técnicas de optimização estudadas nas aulas. **Indique claramente cada uma das optimizações usadas sob a forma de comentário no código.**

```
void sum_elements(int *v, int *sum){
    int i, val;
    int length = vec_length(v);  /* code motion: move call to vec_length out of loop */
    int temp   = 0;              /* accumulate result in temporary,
                                    eliminating unneeded memory references*/
    for(i = 0; i < length; i++){
       val = v[i];               /* eliminate functions calls within loop */
       temp += val;
    }
    *sum = temp;
}
```

[3v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere o seguinte código Assembly:

```
loop_func:
   pushl %ebp
   movl %esp,%ebp
   pushl %ebx
   movl 8(%ebp),%ecx
   movl 12(%ebp),%ebx
   movl $1, %eax
   cmpl %ecx,%ebx
   jle .L4
.L6:
   leal (%ebx,%ecx),%edx
   imull %edx,%eax
   shll %ecx
   cmpl %ecx,%ebx
   jg .L6
.L4:
   movl $0, %edx
   idivl %ebx
   popl %ebx
   movl %ebp,%esp
   popl %ebp
   ret
```

Com base no código Assembly à esquerda, preencha os espaços em branco no código correspondente em C. Apenas pode usar as variáveis `a`, `b` e `result` nas expressões (*não use nomes de registos!*) (**escreva a função completa na folha A4**).

```
int loop_func(int a, int b){
    int result = __1__;

    while(__a < b___){
        __result *= (a+b)__;
        __a *= 2__;
    }

    __result /= b__;

    return result;
}
```