isep Instituto Superior de Engenharia do Porto

DEI / Licenciatura em Engenharia Informática
Arquitectura de Computadores
Exame Época Normal – Janeiro 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 III, IV e V em folhas A4 separadas. No final, deverá entregar 4 folhas assinadas, indicando a versão do enunciado.

[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 `unsigned int` altera o padrão de bits da variável	☐	☒
2) Em C, numa expressão com variáveis do tipo `int` com e sem sinal, todas as variáveis são convertidas para valores sem sinal	☒	☐
3) Em C, se tivermos um `char` com representação binária de `10011010`, o *cast* para um `short` resulta no valor `1111111110011010`	☒	☐
4) Admita um inteiro *x* com valor `0x01234567` e um valor dado por `&x` de `0x100`. Logo, o valor presente no byte `0x101` é `0x45`	☒	☐
5) Em C, a adição de duas variáveis do tipo `int` com valores positivos nunca pode resultar num valor negativo	☐	☒
6) Em C, a operação `u<<k` resulta em $u*2^k$, independentemente da variável `u` ter ou não sinal	☒	☐
7) Em Assembly, a instrução `movl %eax, (%esp)` pode ser usada para escrever o valor de `%eax` no topo da stack	☒	☐
8) Em Assembly, a instrução `cmp` não altera os seus parâmetros nem os bits do registo EFLAGS	☐	☒
9) Admita que o endereço de uma variável `x` está armazenado em `%edi`. É possível alterar o valor de `x` executando `movl $15,%edi`	☐	☒
10) A instrução `movl 4(%ebp),%eax` armazena em `%eax` o valor antigo de `%ebp` numa função que inicia com o prólogo estudado	☐	☒
11) Admita que o endereço de um vetor do tipo `int` está armazenado em `%esi` e que o valor de `%ecx` é 2. A instrução `leal (%esi,%ecx,4),%esi` armazena em `%esi` o endereço do terceiro elemento do vetor.	☒	☐
12) A execução da instrução `call` não implica a alteração do valor do registo `%esp`.	☐	☒
13) Não é possível passar o endereço de uma variável local para outra função chamada pela primeira usando a *stack*	☐	☒
14) Depois de um `call` de uma função com 3 parâmetros inteiros podemos executar `subl $12, %esp` para retirar os parâmetros da stack	☐	☒
15) A convenção de salvaguarda de registos indica que o registo `%esi` deve ser gerido pela função invocada	☒	☐
16) Admita a declaração da matriz `int m[4][5]`. Em Assembly, o endereço do elemento `m[i][j]` é dado pela expressão `m+20*i+4*j`	☒	☐
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 8 bytes.	☐	☒
18) O espaço ocupado por uma estutura alinhada é sempre o mesmo, independemente da ordem dos seus campos	☐	☒
19) O espaço ocupado por uma *union* é sempre o mesmo, independentemente da ordem dos seus campos	☒	☐
20) O compilador não tem qualquer dificuldade em mover a invocação de funções para outro local do programa com vista à sua optimização	☐	☒

[2v] **Grupo II – Para cada uma das expressões seguintes em C indique se a expressão é sempre verdadeira (uma resposta errada desconta 50% de uma correcta).**

Considere as seguintes declarações:

```
int x = f(); /* Retorna valor em [0, INT_MAX] */
int y = f(); /* Retorna valor em [0, INT_MAX] */

/* Conversão para outros formatos */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
double   dx = (double) x;
double   dy = (double) y;
```

Expressão	Sempre verdade?
`x*x >= 0`	**N**
`x+y == uy+ux`	**S**
`x == (int)(float)x`	**N**
`(double)(float)x == (double)ux`	**N**
`dx + dy == (double)(y+x)`	**S**
`(dx + dy) - dx == dy`	**N**

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

Considere as seguintes declarações:

```
typedef struct {
    char c;
    double *p;
    int i;
    double d;
    short s;
} struct1;
```

```
typedef struct {
    float x;
    struct1 *s1;
    char e;
    int v[4];
} struct2;
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `struct1` em Linux/IA32. Indique claramente, para cada campo, o seu tamanho, o seu endereço, assim como para as partes alocadas mas não usadas para satisfazer as restrições de alinhamento. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

```
typedef struct {
    char c;          0x100 : 1 byte
    [gap]            0x101 : 3 bytes
    double *p;       0x104 : 4 bytes
    int i;           0x108 : 4 bytes
    double d;        0x10C : 8 bytes
    short s;         0x114 : 2 bytes
    [gap]            0x116 : 2 bytes
} struct1;
```

Tamanho da estrutura: 24 bytes

[1.5v] **b)** Se definirmos os campos da estrutura `struct1` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? Justifique a sua resposta, indicando, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho.

```
typedef struct {
    double d;        0x100 : 8 bytes
    double *p;       0x108 : 4 bytes
    int i;           0x10C : 4 bytes
    short s;         0x110 : 2 bytes
    char c;          0x112 : 1 byte
    [gap]            0x113 : 1 byte
} struct1;
```

Tamanho da estrutura: 20 bytes

[2v] **c)** Para cada sequência de código em Assembly à esquerda, complete a função correspondente em C à direita, considerando as declarações iniciais das estruturas (**escreva a função completa na folha A4**).

```
f1:
   pushl %ebp
   movl %esp, %ebp
   movl 8(%ebp),%eax
   movl 16(%eax), %eax
   movl %ebp, %esp
   popl %ebp
   ret
```

```
_int__ f1(struct2 *s2)
 {
   return ___s2->v[1]_____;
 }
```

```
f2:
   pushl %ebp
   movl %esp, %ebp
   movl 8(%ebp),%eax
   movl 4(%eax), %eax
   movl 8(%eax),%eax
   movl %ebp, %esp
   popl %ebp
   ret
```

```
 _int__ f2(struct2 *s2)
 {
   return ___s2->s1->i_____;
 }
```

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

O código seguinte em C preenche a diagonal de uma matriz N*N com o valor passado no argumento `val`.

```
void fix_set_diag(int m[N][N], int val){
    int i;
    for(i = 0; i < N; i++)
        m[i][i] = val;
}
```

```
void fix_set_diag(int m[N][N], int val){
    int i;
    int *ptr = &m[0][0];
    i = 0;
    do{
       *(ptr+i) = val;
       i += (N+1);
    }while(i != (N+1)*N);
}
```

Quando compilado para Assembly, o GCC gera o seguinte código para um valor de N = 4

```
fix_set_diag:
   pushl %ebp
   movl %esp, %ebp
   movl 8(%ebp), %ecx
   movl 12(%ebp), %edx
   movl $0, %eax
.L14:
   movl %edx, (%ecx,%eax)
   addl $20, %eax
   cmpl $80, %eax
   jne .L14
   ret
```

Reescreva a função `fix_set_diag` em C usando uma optimização similar à demonstrada no código Assembly. Use expressões que recorram à variável N em vez de valores inteiros constantes, para que o seu código continue a funcionar se o valor de N for alterado.

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

Considere o seguinte código Assembly:

```
loop:
   pushl %ebp
   movl %esp,%ebp
   movl 8(%ebp),%ecx
   movl 12(%ebp),%edx
   xorl %eax,%eax
   cmpl %edx,%ecx
   jle .L4
.L6:
   decl %ecx
   incl %edx
   incl %eax
   cmpl %edx,%ecx
   jg .L6
.L4:
   incl %eax
   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 `x`, `y` e `result` nas expressões (*não use nomes de registos!*) (**escreva a função completa na folha A4**).

```
int loop(int x, int y){
    int result;

    for(__result=0__; __x>y__; result++){
        ___x--____;
        ___y++_____;
    }

    _result++_;

    return result;
}
```