isep Instituto Superior de Engenharia do Porto

DEI / Licenciatura em Engenharia Informática
Arquitectura de Computadores
Exame Época Recurso – Fevereiro 2017

- 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 maior valor que podemos armazenar numa variável do tipo `int` é 2^{32}	☐	☒
2) Admita a variável `unsigned int a = 0xFFFFFFFF` em C. Então, a variável `unsigned int b = a + 1` tem o valor zero	☒	☐
3) Admita a variável `short a` = `0x0123` e um valor dado por `&a` de `0x200` em C. Então, o valor presente no byte `0x201` é `0x01`	☒	☐
4) Em C, ao truncarmos uma variável do tipo `int` que armazena um valor positivo para um `short` podemos ficar com um valor negativo	☒	☐
5) Em C, o operador lógico || (OR) termina a avaliação da expressão se encontrar uma condição que seja avaliada como verdade	☒	☐
6) Em C, as divisões de inteiros usando deslocamentos podem exigir a alteração do dividendo para que o arredondamento seja correto	☒	☐
7) Admita que `int *ptr` armazena endereço inicial de um vetor de `int`. Logo, `(short*)ptr + 8` aponta para o quinto elemento	☒	☐
8) Admita que declara a variável `int x` como variável global em C. Logo, os 4 bytes são reservados na *stack*	☐	☒
9) Em C, é possível retornar como valor de saída de uma função o endereço de um vetor `short *vec = (short*)malloc(20)`	☒	☐
10) Em Assembly, a instrução `popl %eax` é equivalente a executar `movl (%esp),%eax` seguido de `addl $4, %esp`	☒	☐
11) Em Assembly, a instrução `leal (%eax,%eax,4),%eax` pode ser usada para multiplicar por 5 o valor em `%eax`	☒	☐
12) Se `%ecx` for 4 e `%esi` o endereço inicial de um vetor de `short`, `movw 6(%esi,%ecx,2),%ax` coloca em `%ax` o oitavo elemento.	☒	☐
13) Em IA32, a *stack* é usada para suportar a passagem do valor de retorno de uma função invocada à função invocadora	☐	☒
14) Admita que o valor de `%esp` é `0x1004`. A execução da instrução `ret` coloca o valor de `%esp` em `0x1000`	☐	☒
15) Em IA32, reservamos espaço para as variáveis locais de uma função subtraindo o número de bytes necessários ao valor atual de `%ebp`	☐	☒
16) Admita a matriz global `int m[4][5]`. Em Assembly, acedemos ao endereço de `m[i]` calculando `m + i*20`	☒	☐
17) Uma estrutura, alinhada de acordo com as regras estudadas, com 1 `char`, 1 `double` e um 1 `char*` (por esta ordem) ocupa 16 bytes	☒	☐
18) O tamanho de uma estrutura é garantidamente menor se indicarmos os seus campos por ordem decrescente do seu tamanho	☐	☒
19) A fragmentação interna dos blocos reservados na *heap* é originada pelas regras de alinhamento e *overhead* da gestão dos blocos	☒	☐
20) A possibilidade de existirem diversas referências para a mesma posição de memória em C dificulta a otimização efetuada pelo compilador	☒	☐

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

No seguinte excerto de código em C foram omitidos os valores das constantes `M` e `N`:

```
#define M       /* Número mistério 1 */
#define N       /* Número mistério 2 */

int arith(int x, int y) {
  int result = 0;
  result = x*M + y/N;
  return result;
}
```

Admita agora que a função foi compilada para valores específicos de `M` e `N`. O compilador otimizou a multiplicação e divisão pelas constantes `M` e `N` usando deslocamentos, tal como estudado nas aulas. O código seguinte em C é uma tradução do código Assembly gerado:

```
int optarith(int x, int y) {
  int t = x;
  x <<= 5;
  x -= t;
  if (y < 0) y += 7;
  y >>= 3;
  return x+y;
}
```

Quais os valores de `M` e `N`? **Justifique a sua resposta**

A primeira metade da função mostra-nos que `M = 31`; `x*M` é calculado como `(x<<5) - x`.

A segunda metade mostra-nos que `N = 8`; `y/N` é calculado como `y>>3`. Como `y` pode ser negativo (é um `int`), a divisão através de deslocamentos para a direita exige que se altere, nesses casos, o dividendo para `x + (1<<k) - 1` de modo a que o arredondamento seja corretamente efetuado na direção do zero e não de menos infinito.

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

Considere as seguintes declarações:

```
typedef struct {
    short x;
    int y;
} structA;
```

```
typedef struct {
    structA a;
    structA *b;
    int x;
    char c;
    int y;
    char e[3];
    short z;
} structB;
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `structB`. 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 {
    structA a;              0x100 : 8 bytes
    structA *b;             0x108 : 4 byte
    int x;                  0x10C : 4 bytes
    char c;                 0x110 : 1 byte
    [gap]                   0x111 : 3 bytes
    int y;                  0x114 : 4 bytes
    char e[3];              0x118 : 3 bytes
    [gap]                   0x11B : 1 byte
    short z;                0x11C : 2 bytes
    [gap]                   0x11E : 2 bytes
} structB;
```

Tamanho da estrutura: 32 bytes

[1.5v] **b)** Se definirmos os campos da estrutura `structB` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta.** 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.

Sim, ordenando os seus campos por ordem decrescente de tamanho.

```
typedef struct {
    structA a;              0x100 : 8 bytes
    structA *b;             0x108 : 4 byte
    int x;                  0x10C : 4 bytes
    int y;                  0x110 : 4 bytes
    short z;                0x114 : 2 bytes
    char e[3];              0x116 : 3 bytes
    char c;                 0x119 : 1 byte
    [gap]                   0x11A : 2 bytes
} structB;
```

Tamanho da estrutura: 28 bytes

[2v] **c)** Considere as seguintes funções em C, respeitando as declarações iniciais das estruturas:

```
short fun1(structB *s){
  return s->a.x;
}

short* fun2(structB *s){
  return &s->z;
}
```

```
short fun3(structB *s){
  return s->z;
}

short fun4(structB *s){
  return s->b->x;
}
```

Indique a que funções (`fun1`, `fun2`, `fun3` ou `fun4`) correspondem os seguintes excertos de código em Assembly**. Escreva as funções completas na folha A4.**

```
_fun2_:
  pushl   %ebp
  movl    %esp,%ebp
  movl    8(%ebp),%eax
  addl    $28,%eax
  popl    %ebp
  ret
```

```
_fun4_:
  pushl   %ebp
  movl    %esp,%ebp
  movl    8(%ebp),%eax
  movl    8(%eax),%eax
  movswl  (%eax),%eax
  popl    %ebp
  ret
```

```
_fun3_:
  pushl   %ebp
  movl    %esp,%ebp
  movl    8(%ebp),%eax
  movswl  28(%eax),%eax
  popl    %ebp
  ret
```

```
_fun1_:
  pushl   %ebp
  movl    %esp,%ebp
  movl    8(%ebp),%eax
  movswl  (%eax),%eax
  popl    %ebp
  ret
```

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

Admita a seguinte função em C que recebe como primeiro parâmetro o endereço de uma *string* `str` e como segundo parâmetro o endereço de um inteiro `hash` no qual a função armazena o resultado.

```
void calcHash(char *str, int *hash){
 int i, j;

 *hash = 0;
 for(i = 0; i < strlen(str); i++){
   j = strlen(str) / 2;
   *hash += secret(str,i) * 32 + j);
 }
}
```

```
int secret(char *str, int pos){
  return str[pos] % 26;
}
```

Apresente uma segunda versão da função `calcHash` em C com a mesma funcionalidade, mas melhor desempenho. Admita que o compilador que é usado não efetua nenhuma otimização. **Indique claramente cada uma das otimizações usadas sob a forma de comentário no código.**

```
void calcHash(char *str, int *hash){
  int i, j, length, tmp;

  /* accumulate results in local variable */
  tmp = 0;

  /* code motion and strength reduction*/
  length = strlen(str);
  j = length >> 1;

  for(i = 0; i < length; i++){
    /* removing unnecessary procedure calls */
    tmp += ((str[i] % 26) << 5) + j;
  }

  /* Eliminating unneeded memory references */
  *hash = tmp;
}
```

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

```
fun:
  pushl %ebp
  movl %esp,%ebp
  movl 16(%ebp),%ecx
  movl 12(%ebp),%eax
  movl 8(%ebp),%edx
  cmpl %ecx,%edx
  jl .L1
.L2:
  addl %edx,%eax
  decl %edx
  cmpl %ecx,%edx
  jge .L2
.L1:
  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`, `z`, `i` e `result` nas expressões (*não use nomes de registos!*) (**escreva a função completa na folha A4**).

```
int fun(int x, int y, int z){

  int i, result = __y__;

  for(i=_x_; _i >= z_; _i--_){
    result = _result + i_;
  }

  return _result_;
}
```