isep Instituto Superior de Engenharia do Porto

DEI / Licenciatura em Engenharia Informática
Arquitectura de Computadores
Exame Época Normal – Janeiro 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) Admita a variável `unsigned char x` em C. O valor armazenado em `x` depois de executar "`x = -1; x = x >> 1;`" é 127.	☒	☐
2) As operações aritméticas de soma e subtracção de inteiros têm uma implementação diferente em *hardware* para valores com e sem sinal.	☐	☒
3) Em IA32, uma arquitetura *little-endian*, considerando o vetor `short x[10]`, o elemento `x[1]` está num endereço menor que `x[0]`	☐	☒
4) O vetor "`int *vec = (int*)malloc(16);`" pode armazenar 16 inteiros tal como se tivesse sido definido como "`int vec[16];`".	☐	☒
5) Em C, a multiplicação de duas variáveis `u` e `v` do tipo `int` pode resultar num valor menor do que os armazenados em `u` ou `v`	☒	☐
6) Em C, as expressões "`x * 35`" e "`(x<<5) + (x<<2) - x`" são sempre equivalentes para qualquer valor de `unsigned int x`	☒	☐
7) Admita que `ptr` é uma variável do tipo `char*`. Então, em C, a expressão `(short*)ptr + 7` avança 14 bytes na memória	☒	☐
8) Admita que declara a variável `int x` na função *main* em C. O compilador pode atribuir `x` a um registo ou a um endereço na *heap*	☐	☒
9) Em Assembly, a instrução "`imull %edx`" duplica o valor do registo usado como argumento.	☐	☒
10) Em Assembly, a instrução "`pushl %eax`" é equivalente a "`movl %eax, (%esp)`" seguido de "`addl $-4, %esp`"	☐	☒
11) A adição de dois bytes com sinal com valores 0xAC e 0x8A deixa as *flags* do registo EFLAGS com os valores ZF=0, SF=1, CF=1, OF=1	☐	☒
12) Em IA32, a instrução `test` compara o valor dos seus operandos através de um subtração.	☐	☒
13) Em IA32, a *stack* é usada para suportar a invocação de funções e o retorno para a função invocadora com `call` e `ret`, respetivamente.	☒	☐
14) Admita que o valor de `%esp` é `0x1000`. A execução da instrução `jmp` coloca o valor de `%esp` em `0xFFC`	☐	☒
15) De acordo com a convenção usada em Linux/IA32, a responsabilidade de salvaguarda e restauro de `%esi` é da função invocada	☒	☐
16) Admita a matriz global `short m[10][3]`. Em Assembly, acedemos ao valor de `m[3][1]` avançando 20 bytes a partir de `m`	☒	☐
17) Uma estrutura, alinhada de acordo com as regras estudadas, com 2 `int`, um vetor de 7 `char` e 1 `short` (por esta ordem) ocupa 20 bytes	☒	☐
18) O tamanho de uma *union* sujeita a alinhamento pode ser menor se indicarmos os seus campos por ordem crescente do seu tamanho	☐	☒
19) A fragmentação da *heap* pode impedir a alocação de um novo bloco mesmo que exista esse número de bytes livres	☒	☐
20) A invocação de funções introduz *overhead* e limita as possibilidades de otimização dos programas pelo compilador	☒	☐

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

Pediram-lhe para implementar uma função que determine se a primeira *string* é maior do que a segunda. Decidiu usar a função `strlen` definida na biblioteca "`string.h`" com a seguinte declaração:

```
size_t strlen(const char *s);
```

A sua primeira tentativa resultou na função `strlonger` descrita ao lado. Quando a testou, verificou que os resultados nem sempre são os esperados. Depois de alguma investigação, descobriu que o tipo `size_t` está definido em "`stdio.h`" como sendo `unsigned int`.

```
int strlonger(char *s, char *t){

   return strlen(s) - strlen(t) > 0;

}
```

[1v] **a)** Em que casos irá a função definida por si produzir um resultado incorreto? Explique como é que esse resultado incorreto é possível.

A função irá retornar erradamente 1 quando *s* for menor do que *t*. Quando s é menor do que t, a diferença *strlen(s) – strlen(t)*, que deveria ser negativa, é interpretada com um valor positivo grande sem sinal, o que é maior do que zero.

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

Bastaria alterar o teste para *return strlen(s) > strlen(t)*, por exemplo.

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

Número:_____ **Nome:**_____

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

Considere as seguintes declarações:

```
typedef struct {
    char a[3];
    short int b;
    long long int c;
    int d;
    structB *ptrB;
    char e;
} structA;
```

```
typedef struct {
    int a;
    char b;
    short c;
    int d;
} structB;
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `structA`. 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 {
    char a[3];              0x100 : 3 bytes
    [gap]                   0x103 : 1 byte
    short int b;            0x104 : 2 bytes
    [gap]                   0x106 : 2 bytes
    long long int c;        0x108 : 8 bytes
    int d;                  0x110 : 4 bytes
    structB *ptrB;          0x114 : 4 bytes
    char e;                 0x118 : 1 byte
    [gap]                   0x119 : 3 bytes
} structA;
```

Tamanho da estrutura: 28 bytes

[1.5v] **b)** Se definirmos os campos da estrutura `structA` 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, o novo endereço de cada campo e das partes alocadas mas não usadas, bem como o novo tamanho total da estrutura.

Sim, ordenando os campos por ordem decrescente de tamanho.

```
typedef struct {
    long long int c;        0x100 : 8 bytes
    int d;                  0x108 : 4 bytes
    structB *ptrB;          0x10C : 4 bytes
    short int b;            0x110 : 2 bytes
    char a[3];              0x112 : 3 bytes
    char e;                 0x115 : 1 byte
    [gap]                   0x116 : 2 bytes
} structA;
```

Tamanho da estrutura: 24 bytes

[2v] **c)** Considere o seguinte fragmento de código em C:

```
structA matrix[4][5];

int return_structB_d(int i, int j){
   return matrix[i][j].ptrB->d;
}
```

Reescreva a função `return_structB_d` em Assembly. Na sua resolução tenha em consideração que a matriz `matrix` é global e respeite as declarações iniciais das estruturas. **Comente o seu código.**

```
return_structB_d:               # Abordagem simples (código pode ser optimizado)
  pushl %ebp
  movl  %esp,%ebp
  pushl %ebx
  pushl %esi
  movl  8(%ebp), %eax           #i
  imull $5,%eax                 #5*i (número de elementos por linha)
  imull $28,%eax                #5*i*28 (tamanho da estrutura)
  movl  12(%ebp), %ebx          #j
```

```
  imull $28, %ebx                 #j*28
  addl  %ebx,%eax                 #5*i*28+j*28
  movl  $20,%ebx                  #offset do campo ptrB
  movl  matrix(%eax,%ebx),%esi    #endereço da structB apontada por ptrB
  movl  8(%esi),%eax              #offset de 8 bytes para o campo d
  popl  %esi
  popl  %ebx
  movl  %ebp,%esp
  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 um vetor `strs` de apontadores para *strings* e como segundo parâmetro o endereço de um inteiro `res` no qual a função armazena o resultado.

```
void handle_strs(char *strs[20], int *res){
 int i, j;
 *res = 0;
 for(i = 0; i < 20; i++){
  if(strlen(strs[i]) < i*10)
   *res += strlen(strs[i]);
  else
   for(j=0; j < strlen(strs[i]); ++j)
    *res +=(16*i + get_char_at(strs[i],j));
 }
}
```

```
int get_char_at(char *str, int pos){
  return str[pos];
}
```

Apresente uma segunda versão da função `handle_strs` 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 handle_strs(char *strs[20], int *res){
  int i, j, length,tmp, local_res;

  local_res = 0;                  /* accumulate results in local variable */
  for(i = 0; i < 20; i++){
     length = strlen(strs[i]);    /* code motion and strength reduction*/
     if(length < (i<<3 + i<<1))
       *res += length;
     else{
       tmp = i<<4;
       for(j=0; j < length; ++j){
         local_res += tmp + strs[i][j]; /* removing unnecessary procedure calls */
       }
    }
  }
  *res = local_res;                    /* Eliminating unneeded memory references */
}
```

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

```
fun:
  pushl %ebp
  movl  %esp,%ebp
  pushl %esi
  pushl %ebx
  movl  8(%ebp),%ebx              #n
  movl  12(%ebp),%esi             #*a
  xorl  %edx,%edx                 #x=0
  xorl  %ecx,%ecx                 #i=0
  cmpl  %ebx,%ecx                 #i<n?
  jge   .L3
.L1:
  movl  (%esi,%ecx,4),%eax        #a[i]
  cmpl  %edx,%eax                 #x<a[i]?
  jle   .L2
  movl  %eax,%edx                 #x=a[i]
.L2:
  incl  %edx                      #x++
  incl  %ecx                      #i++
  cmpl  %ebx,%ecx                 #i<n?
  jl    .L1
.L3:
  movl  %edx,%eax
  popl  %ebx
  popl  %esi
  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 `n`, `a`, `i` e `x` nas expressões (*não use nomes de registos!*) (**escreva a função completa na folha A4**).

```
int fun(int n, int *a){
  int i;
  int x = __0__;

  for(i=_0_; _i<n_; i++){
     if(___x<a[i]___)
        x = __a[i]__;
     __x++__;
  }
  return x;
}
```