DEI / Licenciatura em Engenharia Informática
Arquitectura de Computadores
Exame Época Recurso – Fevereiro 2022

- Autorizada apenas a consulta da folha de consulta oficial.
- A infração implica, no mínimo, a **ANULAÇÃO** da prova.
- Quando omissa a arquitectura, considere Linux/x86-64.

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 correta).**

	V	F
1) Em C, se tivermos um `char` com representação binária de `10011010`, o *cast* para um `short` resulta no valor `1111111110011010`	☒	☐
2) Em C, considere "`int x=0x01234567;`" com o endereço de `x` em `0x100`. Logo, o valor presente no byte `0x101` é `0x23`	☐	☒
3) Em C, considere "`short x=0x1234;`". O resultado da operação "`x && 0x0F0F`" é `0x0204`	☐	☒
4) Em C, a avaliação de expressões com variáveis com e sem sinal interpreta todas as variáveis como sendo valores com sinal	☐	☒
5) Em C, admita um vetor "`int vec[10];`" e um apontador "`short *ptr = (short*) vec`". Então, `ptr + 4` avança para `vec[2]`	☒	☐
6) Em C, quando a soma de duas variáveis "`unsigned char u,v;`" é igual ou superior a 2^8 o valor obtido é equivalente a `u + v - 2^8`	☒	☐
7) Em C, executar "`malloc(strlen("arqcp"))`" permite-nos reservar na *heap* os bytes suficientes para armazenar a string "`arqcp`"	☐	☒
8) Em x86-64, a instrução "`popq %rax`" é o equivalente a "`movq %rax,(%rsp)`" seguido de "`subq $8,%rsp`"	☐	☒
9) Em x86-64, se atribuirmos valores com sinal aos registos a somar, o resultado será incorreto se a *flag* de *carry* estiver ativa após a soma	☐	☒
10) Em x86-64, "`idivq %rcx`" efetua a divisão (com sinal) entre `%rax` e `%rcx` colocando o quociente em `%rax` e o resto em `%rdx`	☐	☒
11) Em x86-64, ao contrário das operações de deslocamento de bits, as rotações nunca perdem os bits da informação original	☒	☐
12) Em x86-64, a instrução "`leaq (%rax,%rax,6),%rax`" pode ser usada para multiplicar por sete o valor presente em `%rax`	☐	☒
13) Em x86-64, é possível obter o mesmo resultado com "`imull $-8,%eax`" e "`shll $3,%eax; notl %eax; incl %eax`"	☒	☐
14) Em x86-64, admita que o valor de `%rsp` é `0x1008`. A execução da instrução `ret` coloca o valor de `%rsp` em `0x1000`	☐	☒
15) Em x86-64, o equivalente a "`*ptr1 = *ptr2`", apontadores do tipo `int*` em C, pode ser obtido com "`movl (%rax),(%rcx)`"	☐	☒
16) Em x86-64, de acordo com a convenção de salvaguarda e restauro de registos estudada nas aulas, `%r10` é um registo *caller saved*	☒	☐
17) Em x86-64, o endereço inicial de uma *struct* alinhada depende das restrições de alinhamento dos seus campos	☒	☐
18) Em x86-64, o espaço ocupado por uma *union* é sempre o mesmo, independentemente da ordem dos seus campos	☒	☐
19) Em x86-64, a *stack* é usada para suportar o retorno do valor de saída de uma função, tal como acontece com o controlo de fluxo	☐	☒
20) O bloco de código "`for(j=0;j<N;j++)for(i=0;i<M;i++)sum+=m[j][i];`" exibe boa localidade espacial e temporal	☒	☐

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

[1,5v] **a)** Cada uma das seguintes linhas de código gera um erro quando invocamos o *assembler*. Explique o que está errado em cada uma delas.

```
1. movb $0xF, (%ebx)
2. movl %rax, (%rsp)
3. movw (%rax), 4(%rsp)
4. movq %rax, $0x123
5. movl %eax, %rdx
6. movw %si, 8(%rdi,%rcx,9)
```

1. Um endereço nunca pode ser um valor de 32 bits armazenado em %ebx
2. Diferença entre a variante usada na instrução mov (4 bytes) e o tamanho do valor que se pretende copiar (8 bytes)
3. Não é possível ter duas referências para a memória na mesma instrução
4. Não é possível ter um valor imediato (constante) como destino
5. Diferença entre o tamanho do valor que se pretende copiar (4 bytes) e o destino (8 bytes)
6. Valor errado (9) no parâmetro "tamanho" na expressão de endereçamento. Apenas são permitidos os valores 1, 2, 4, ou 8

[1,5v] **b)** Assuma os apontadores `src_t *sp` e `dest_t *dp`, em que `src_t` e `dest_t` são tipos de dados declarados com `typedef`. Assuma que os endereços `sp` e `dp` são passados por parâmetro a uma função e estão, portanto, armazenados nos registos `%rdi` e `%rsi`, respetivamente. Para cada uma das entradas seguintes da tabela indique **as duas instruções em Assembly** que implementam o equivalente à operação `*dp = (dest_t) *sp` realizada dentro da função em C.

src_t	dest_t	Instruções Assembly
long	long	`movq (%rdi), %rax` `movq %rax, (%rsi)`
char	int	`movsbl (%rdi), %eax` `movl %eax, (%rsi)`
int	unsigned long	`movl (%rdi), %eax` `movq %rax, (%rsi)`

unsigned char	long	movzbl (%rdi), %eax movq %rax, (%rsi)
int	char	movl (%rdi), %eax movb %al, (%rsi)
unsigned int	unsigned short	movl (%rdi), %eax movw %ax, (%rsi)

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

```
struct s1{
    char a;
    short b;
    struct s2 *c;
    union u1 d;
    struct s2 e;
};
```

```
struct s2{
   long f;
   struct s1 *g;
   struct s2 *h;
   char i;
   char j[3];
};
```

```
union u1 {
  int *k;
  char l;
  long m[2];
  struct s1 *n;
};
```

[1,5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo **`struct s1`**. 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`.**

```
struct s1{
    char a;                 0x100 : 1 byte
    [gap]                   0x101 : 1 byte
    short b;                0x102 : 2 bytes
    [gap]                   0x104 : 4 bytes
    struct s2 *c;           0x108 : 8 bytes
    union u1 d;             0x110 : 16 bytes
    struct s2 e;            0x120 : 32 bytes
};
```

Tamanho da estrutura: 64 bytes

[1,5v] **b)** Considere que a função `init` opera sobre uma estrutura do tipo `struct test` e que o compilador gerou o seguinte código Assembly. Com base nesta informação, preencha as expressões em falta no código em C para a função `init`. **Justifique as suas escolhas**.

```
struct test{
    short *p;
    struct s{
      short x;
      short y;
    }
    struct test *next;
};
```

```
void init(struct test *st){
   st->s.y  = st->s.x;
   st->p    = &(st->s.y);
   st->next = st;
}
```

```
init:
   movw 8(%rdi), %ax
   movw %ax, 10(%rdi)
   leaq 10(%rdi), %rax
   movq %rax, (%rdi)
   movq %rdi, 16(%rdi)
   ret
```

```
init:
   movw 8(%rdi), %ax    # %ax = st->s.x
   movw %ax, 10(%rdi)   # st->s.y = %ax
   leaq 10(%rdi), %rax  # %rax = &(st->s.y)
   movq %rax, (%rdi)    # st->p = %rax
   movq %rdi, 16(%rdi)  # st->next = st
   ret
```

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

[1,5v] **a)** No seguinte excerto de código em C foi omitido o valor da constante `M`:

```
#define M       /* Número mistério */

void transpose(long A[M][M]) {
  long i, j;
  for(i=0; i<M; i++){
   for(j=0; j<i; j++){
     long t =  A[i][j];
     A[i][j] = A[j][i];
     A[j][i] = t;
   }
}
```

Admita que a função foi compilada para um valor específico de `M` e o compilador gerou o seguinte código otimizado em Assembly para o ciclo interior da função:

```
.L6:
  movq (%rdx), %rcx
  movq (%rax), %rsi
  movq %rsi, (%rdx)
  movq %rcx, (%rax)
  addq $8, %rdx
  addq $120, %rax
  cmpq %rdi, %rax
  jne .L6
```

Qual o valor de `M`? **Justifique a sua resposta.**

Podemos ver que os registos `%rdx` e `%rax` estão a ser usados como apontadores. `%rdx` está a ser incrementado em 8, o que só pode ser uma referência a um incremento para o próximo elemento da matriz (8 bytes é o tamanho de um `long`). Logo, `%rdi = &A[i][j]`, o que nos deixa `%rax = &A[j][i]`. `%rax` está a ser incrementado em 120, o que só pode ser uma referência ao número de bytes que ocupa uma linha. Assim, temos **M = 120/8 = 15**.

[1,5v] **b)** Use os seguintes valores iniciais em memória e nos registos da arquitetura para responder a cada uma das questões (isto é, cada questão não é afetada pela execução das instruções anteriores).

Endereço	Valor
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Registo	Valor
%rax	0x100
%rcx	0x1
%rdx	0x3
%rbx	0x4

A. Qual o novo valor de `%eax` após "`movl 0x100,%eax`"? __0xFF__

B. Qual o novo valor de `%ecx` após "`movl (%rax,%rdx,4),%ecx`"? __0x11__

C. Qual o endereço que é alterado com "`subl %edx,4(%rax)`"? __0x104__

D. Qual o novo valor de `%rbx` após "`leaq 0x100(,%rbx,2),%rbx`"? __0x108__

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

```
long f1(long a, long b, long c,
        long d, long e, long f){
  return a + f2(a*b,b,c,d,e,f,10,-20);
}
```

Com base no código C acima, preencha os espaços em branco no código correspondente em Assembly ao lado. (**escreva a função completa na folha A4**).

```
f1:
  pushq %rdi
  movq %rdi, %rax
  imulq %rsi
  movq %rax, %rdi
  pushq $-20
  pushq $10
  call f2
  addl $16, %rsp
  popq %rdi
  addl %rdi, %rax
  ret
```