Teste 1A de Arquitetura de Computadores

16/4/2018 Duração 2h *sem consulta*

As perguntas com múltiplas respostas têm de ser respondidas na folha de respostas anexa.
Respostas erradas descontam 1/5 da cotação da pergunta ou alínea.

Número: _____ *Nome:* _____

1 — (1,5 val) Considere uma representação de inteiros (sem sinal) usando **6 bits**.

a) Como é codificado o valor 19 ?

A) 01 0110 B) 01 0011 C) 00 1011 D) 11 0110 E) 10 1101

b) Como é codificado −8 (usando a norma complemento para 2)?

A) 11 1110 B) 11 0110 C) 11 1000 D) 10 0110 E) 10 1000

c) Qual é o maior valor que se pode representar (sem sinal, em base 10) ?

A) 64 B) 63 C) 128 D) 127 E) 32

d) Qual é o menor valor que se pode representar usando complemento para 2 (resposta em base 10) ?

A) −64 B) −63 C) −128 D) −127 E) −32

2 — (1,5 val) Considere um CPU que tem uma unidade aritmética e lógica em que as entradas e a saída têm 8 bits.

a) Diga qual é o resultado (em base 2) da soma de $0111\ 1100_2$ com $0001\ 0101_2$

A) 0110 1001 B) 1001 0001 C) 1111 1101 D) 0110 1101 E) 1011 1001

b) Indique os valores das flags CF, OF, ZF e SF, após a realização da operação (soma) anterior.

A) CF=0; OF=1; ZF=0; SF=1 B) CF=1; OF=1; ZF=0; SF=1 C) CF=0; OF=0; ZF=1; SF=1

D) CF=0; OF=1; ZF=1; SF=0 E) CF=1; OF=0; ZF=1; SF=0

c) Assuma que outra operação aritmética entre números com sinal deu a seguinte configuração de *flags*: CF=1; OF=0; ZF=0 e SF=1. O que significa esta configuração de *flags* relativamente à correção do resultado e porquê?

A) O resultado está errado porque a flag CF está a 1.

B) O resultado está errado porque a flag ZF está a 0.

C) O resultado está errado porque a flag SF está a 1 e OF não.

D) O resultado está correto porque ambas as flags SF e CF estão a 1.

E) O resultado está correto porque a flag OF está a 0.

3 – (1 val) A representação da informação/dados por bits nos computadores tem as seguintes consequências (assinale a verdadeira):

A) Representa facilmente valores binários, mas não permite que os nossos programas usem valores em base 10 ou outros tipos de dados como reais, imagens, etc.

B) Representa facilmente valores binários, mas necessitamos cada vez de mais bits para representar valores cada vez maiores.

C) Torna complicado representar valores inteiros e muito difícil de implementar no hardware as operações aritméticas e lógicas.

D) Torna complicado representar valores numéricos, mas permite uma representação compacta (com poucos bits representar qualquer valor da base 10).

E) Representa facilmente os dados como valores binários, mas as instruções executadas pelo CPU têm de ser codificadas em dígitos de base 10.

4 — (1 val) Considere a norma IEEE 754, onde a representação de números reais em precisão simples (32 bits) usa a seguinte interpretação:

- Bit 31: sinal – 0 se maior ou igual a zero, 1 se menor do que zero
- Bits 30 a 23: expoente somado de 127
- Bits 22 a 0: parte fracionária da mantissa (o valor efetivo da mantissa é 1.XXX...XXX)

Indique o valor decimal em base 10 representado por: 1 01111111 11100000000000000000000

A) -1.875×2^{127} B) 1.875×2^{127} C) −1.875 D) 1.875 E) −18.75

5 — (1 val) Sabendo que a variável **unsigned int f** contém a representação binária de um número real codificado na normal IEEE754 de precisão simples, indique qual a expressão que lhe permite obter o valor do expoente.

A) `(f>>23)&0xff+127` B) `(f>>23)&0xff-127` C) `(f&0xff)>>23-127`

D) `(f<<23)&0xff+127` E) `(f<<23)&0x0ff00-127`

6 — (1 val) O processador (CPU) interpreta sequencialmente as instruções máquina de um programa executando repetidamente um ciclo. Escolha a opção que apresenta o processamento realizado nesse ciclo pela ordem correta:

A) obter os operandos; *fetch* da instrução em memória; descodificação da instrução; executar a instrução

B) *fetch* da instrução em memória; obter os operandos; descodificação da instrução; executar a instrução

C) *fetch* da instrução em memória; descodificação da instrução; obter os operandos; executar a instrução

D) *fetch* da instrução em memória; obter os operandos; executar a instrução; descodificação da instrução

E) *fetch* da instrução em memória; descodificação da instrução; executar a instrução; obter os operandos

7 – (1val) Indique qual das seguintes afirmações é verdadeira:

A) Cada arquitetura tem sempre no *bus* um número igual de linhas (bits) para dados e endereços.

B) Cada componente de um computador tem uma ligação (*bus*) dedicada ao CPU.

C) Numa arquitetura com memória endereçável ao byte, o *bus* transfere apenas um byte de cada vez da memória para o CPU

D) A memória real endereçável por um CPU é limitada pela capacidade (número de bits) do *bus* de endereços.

E) O *bus* permite fazer múltiplas transferências simultâneas de dados entre vários dispositivos.

8 — (1 val) A arquitetura de um computador é constituída pelos seguintes componentes genéricos (escolha a verdadeira):

A) Memória, periféricos e CPU (sendo que este contém registos e bus de sistema)

B) Memória, bus de sistema e CPU (sendo que este contém registos, ALU e periféricos)

C) Registos, periféricos e CPU (sendo que este contém ALU, bus de sistema e memória)

D) Periféricos, memória, bus de sistema e CPU (sendo que este contém registos e ALU)

E) Memória, registos, periféricos e CPU (sendo que este contém ALU e bus de sistema)

9 — (1,5 val) Num programa (C ou Java) pretende-se fazer determinados cálculos envolvendo números inteiros e reais. Para avaliar a melhor maneira de o fazer testou-se o seguinte troço de código que produziu o output indicado:

```
int i = 200000011;
float f = i;
double result1 = i/2;
double result2 = f/2;
```

O resultado foi 100000005.0 e 100000008.0 respetivamente. Ambos errados, pois devíamos obter 100000005.5.

a) Indique uma possível explicação para tanto `result1` como `result2` estarem errados:

A) Em ambos os casos foi efetuada uma divisão de inteiros e não obtemos as décimas

B) i/2 não obtém as décimas e o float f não tem precisão para representar corretamente 200000011

C) O valor 200000011 é maior que o maior int e fica logo errado em i, daí os erros seguintes

D) i/2 não obtém as décimas e ao atribuir i a f são copiados os bits sem qualquer conversão ficando f com "lixo"

b) Numa tentativa de melhorar os resultados, alguém propôs o seguinte código:

```
int i = 200000011;
double f = i;
double result = f/2;
```

Indique a argumentação correta para aceitar ou recusar este código:

A) Recusar porque a divisão continua a ser inteira, não nos permitindo assim obter as casas decimais necessárias

B) Recusar porque terá a mesma precisão que a anterior (o double tem os mesmos bits que o float para representar a parte inteira)

C) Aceitar porque f como double já tem precisão suficiente para representar o valor 200000011

D) Aceitar porque o código efetua a divisão em double o que permite obter a casa decimal que faltava

10 — (1 val) Considere o seguinte programa em C:

```
#include <stdio.h>
int main() {
    char c = 'a';
    int i[3] = {123, 124, 125};
    char *p = &c;
    int *pp = i;

    *p = 'b';
    p++;
    pp++;
    (*pp) = (*pp)+1
    printf( "%c, %d, %d, %d\n", c, i[0], i[1], i[2]);
}
```

qual dos seguintes resultados obtemos?

A) b 125 124 125 B) b 123 125 125

C) c 125 124 125 D) c 123 125 125 E) c 123 124 126

11 — (1 val) Qual das afirmações referentes ao seguinte código C é verdadeira?

```
char a[] = "hello";
for (char* p = a; *p != '\0'; p++)
  *p = (*p) + 1;
```

A) A expressão inicial do ciclo (`char* p = a`) está incorreta, devia ser `char* p = &a`

B) A expressão inicial do ciclo (`char *p = a`) está incorreta, devia ser `char[] p = a`

C) O ciclo executa normalmente, incrementando o valor guardado em cada posição do array `a`, por exemplo se `a[i]` tiver `'a'` então o novo valor de `a[i]` será `'b'`

D) O ciclo executa normalmente, colocando em cada posição do array `a` o valor do elemento seguinte, por exemplo coloca em `a[1]` o valor de `a[2]`

E) O ciclo executa indefinidamente, pois a condição de paragem está mal definida, devia ser `p != '\0'`

12 — (1 val) Considere um programa em *assembly* que escreve a cadeia *msg* no ecrã.

```
EXIT = 1
WRITE = 4
LINUX_SYSCALL = 0x80
.data
msg: .ascii "Hello, world!\n"  # um vetor de caracteres
msglen = . - msg               # msglen representa o tamanho do vetor
.text
.global _start
_start:    movl $msglen,%edx
           movl $msg,%ecx
           movl $1,%ebx
           movl $WRITE,%eax
           int $LINUX_SYSCALL
           movl $0, %ebx
           movl $EXIT, %eax
           int LINUX_SYSCALL
```

Assinale a afirmação **falsa**:

A) A instrução "`movl $WRITE,%eax`" coloca no registo %eax a *string* WRITE.

B) A instrução "`movl $msg, %ecx`" move o endereço do primeiro byte do vetor para o registo %ecx.

C) A etiqueta "`msg:`" representa o endereço do primeiro byte do vetor.

D) Cada byte na cadeia referida por "`msg`" terá um valor entre 0 e 255.

E) Depois da instrução "`movl $msglen,%edx`", o registo %edx fica com o valor 14.

13 – (1 val) Dadas as declarações de variáveis `a` e `b`, num Intel 32 bits, escolha a opção que apresenta as instruções corretas considerando o tamanho de dados utilizado:

```
a: .int  321
b: .byte 57

1. movb a, %ax
2. movl a, %eax
3. mov b, a
4. incb %ebx
5. movb b, %al
```

A) Apenas 2 e 5 B) Apenas 2, 3 e 5 C) Apenas 1, 2, 3, e 5 D) Apenas 1, 2, 4 e 5 E) Todas são corretas

14 — (1 val) Considere uma arquitetura de 32 bits *little-endian* onde declarou, em C, as seguintes varáveis:

```
int a;
int *b;
char c;
char *d;
```

Sabendo que estas estão sequencialmente em memória, respetivamente nos endereços 100, 104, 108 e 112, indique qual das seguintes figuras é uma representação válida da memória do computador (cada célula representa um byte), após executar as seguintes instruções C:

```
a = 257;
c = 48;
b = &a;
d = &c;
*b = 9;
*d = 50;
```

A)

End.	*Cont.*
100	9
101	0
102	0
103	0
104	100
105	0
106	0
107	0
108	50
109	0
110	0
111	0
112	108
113	0
114	0
115	0

B)

End.	*Cont.*
100	257
101	0
102	0
103	0
104	9
105	0
106	0
107	0
108	48
109	0
110	0
111	0
112	50
113	0
114	0
116	0

C)

End.	*Cont.*
100	0
101	0
102	0
103	9
104	0
105	0
106	0
107	100
108	0
109	0
110	0
111	50
112	0
113	0
114	0
115	108

D)

End.	*Cont.*
100	257
101	0
102	0
103	0
104	100
105	0
106	0
107	0
108	48
109	0
110	0
111	0
112	108
113	0
114	0
116	0

E)

End.	*Cont.*
100	0
101	0
102	0
103	257
104	0
105	0
106	0
107	9
108	0
109	0
110	0
111	48
112	0
113	0
114	0
115	50

15 — (1,5 val) Pretende-se que implemente uma função na linguagem C que, dada uma cadeia de caracteres (*string*) não vazia, devolve o carácter na última posição (o carácter imediatamente antes do código de fim de cadeia '\0'). Pode usar funções em string.h.

```
char ultimo( char cadeia[] )
{
        int i = 1;
        while ( cadeia[i] != '\0' )
           i++;
        return cadeia[i-1];

                                outra solução: return cadeia[strlen(cadeia)-1];

}
```

16 — (1,5 val) Um polígono regular tem todos os lados e ângulos iguais. Pretende-se que implemente uma função na linguagem C que, dado os lados e ângulos de um polígono, retorna true (1) se o polígono é regular (i.e. tem todos os lados e ângulos iguais) e false (0) caso contrário.

```
int polRegular( int numL, float *lados, float *angulos )
{
      for ( int i = 1; i<numL; i++ )
         if ( lados[i]!=lados[0] || angulos[i]!=angulos[0] )
            return 0;

      return 1;









}
```

17 — (1,5 val) Pretende-se que escreva, em C uma função que permita inverter a ordem dos elementos de qualquer vetor de ints, sem usar vetores auxiliares (se usar desconta metade da cotação). Esta terá a seguinte assinatura:

```
void inverte( int *vet, int len );
```

Esta recebe como parâmetros de entrada o endereço inicial de um vetor, vet, com len elementos. Exemplos de utilização:

```
int x[]={ 1, 2, 4, 5, 6 };
inverte ( x, 5 );
```

x fica depois com { 6, 5, 4, 2, 1 }

```
void inverte( int *vet, int len ) {
   for ( int i=0; i<len/2; i++ ) {
       int tmp = vet[i];
       vet[i] = vet[len-i-1];
       vet[len-i-1] = tmp;
   }
}
```

página para rascunho.