Exame de Arquitetura de Computadores **versão A**

Época Normal -- 29/6/2020 -- Duração 2h30

Nas perguntas de escolha múltipla as respostas erradas descontam 1/5 da cotação.

Número: _____ *Nome:* _____

1- (0,7 val) Considere um valor que é representado em binário por 100101.

a) Qual é o número representado?

A. 21 B. 22 C. 37 D. 69 E. nenhum dos anteriores

b) Se esse padrão de bits representar um valor em complemento para 2 numa arquitetura de 6bits, qual o valor representado?

A. -43 B. -27 C. -26 D. 22 E. 37

2- (0,7 val) Diga quantos bits são necessário no mínimo para que uma arquitectura possa representar a gama de valores seguintes:

a) Os valore inteiros sem sinal de 0 a 257?

A. 6 bits B. 7 bits C. 8 bits D. 9 bits E. 10 bits

b) Os valores em complemento para 2 de -128 a 128 ?

A. 6 bits B. 7 bits C. 8 bits D. 9 bits E. 10 bits

3- (0,5 val) Considere um CPU como o Intel onde foi realizada uma subtração, "`sub a,b`", ficando as flags com os valores seguintes: **`CF=1, OF=1, SF=0, ZF=0.`**

Relacionando `a` com `b`, o que significa este estado das flags considerando que `a` e `b` representam valores inteiros?

A. `a < b` independentemente de serem inteiros com ou sem sinal.

B. `a > b` apenas se os valores são inteiros com sinal.

C. `a > b` apenas se os valores são inteiros sem sinal.

D. `a > b` independentemente de serem inteiros com ou sem sinal.

E. nenhuma das anteriores

4 - (1,1 val) Considere agora a representação de números em vírgula flutuante segundo a norma IEEE754 para precisão simples (1 bit sinal; 8 bits componente de expoente somado de 127; 23 bits parte fracionária da mantissa).

a) Complete a frase: O número real **0.6** é representável na norma em questão ...

A. com erro de arredondamento, pois é representado em binário por uma dízima infinita periódica.

B. com erro de arredondamento, pois é representado em binário por uma dízima infinita não periódica.

C. sem erro de arredondamento, pois é representado em binário por uma dízima finita.

D. sem erro de arredondamento, pois é representado em binário por uma dízima infinita periódica.

E. sem erro de arredondamento, pois em IEEE754 as dízimas são omitidas.

b) Que número é representado pelo padrão de bits `01000001001001000000000000000000`?

A. 10.25 B. 12.5 C. 1.7014118e+38 D. −1.7014118e+38 E. 9.25

5- (0,5 val) Considere o seguinte excerto de código C:

```
float f = 2.0;
int i = * (int *) &f; // fica i=0x40000000
int i2 = i << 1;
float f2 =  * (float *) &i2;
```

Qual é o valor que fica em `f2`?

A. f2 = 0.0 B. f2 = f * 2 C. f2 = f / 2 D. f2 = i * 2 E. f2 = i / 2

6- (0,5 val) Qual é o output do seguinte programa em C?

```
#include <stdio.h>
void do_it( int *a, double b ) {
   *a = (int)b;
}
int main () {
   float x = 3.6;
   int i;
   do_it( &i, x );
   printf ("%f, %d\n", (float)i, i);
   return 0;
}
```

A. `3.6, 3` B. `3, 3` C. `3.00000, 3` D. `4.00000, 4`

E. Nenhum, pois dá um erro na compilação

7- (0,5 val) Considere o seguinte excerto de código que pode ser C ou Java, onde VAL é um qualquer valor de `int`.

```
int x = VAL;
float y = (float)x;
int z = (int)y;
```

Indique a afirmação verdadeira:

A. Este código vai dar erro ao ser compilado porque não se pode atribuir um int a um float.

B. Este código vai dar erro ao ser compilado porque não se pode atribuir um float a um int.

C. Este código pode dar uma excepção ou abortar a execução caso VAL seja um valor que não pode ser atribuído a y.

D. Este código não tem problemas em executar, podendo, no entanto, o valor em z ficar diferente de x, dependendo de VAL.

E. Este código não tem problemas em executar, ficando o valor em z sempre igual ao em x (e a VAL).

8- (0,5 val) Qual das seguintes expressões em C permite obter o valor 1 se o número sem sinal em `X` for ímpar e zero caso contrário? (lembre-se que para os números pares o resto da divisão por 2 é sempre 0)

```
A. ((X <<1) & 2)>>1
B. X >>1
C. (X & 1) <<1
D. (X & 2)>> 1
E. (X & 1) | 0xffffff2
```

9- (0,5 val) Considere que numa arquitectura *little-endian* (p.e. intel) a memória a partir do endereço 100 tem em cada byte cada um os seguintes valores (base 10):

```
1   2   3   4   5   6   7    8    9   10
```

Em *Assembly* para Intel, 32 bits, executam-se as seguintes instruções:

```
mov  (101), %eax
mov  (103), %bx
```

Quais os valores (hexadecimais) nos registos EAX e BX após a execução das instruções anteriores?

A. EAX = 0x1234, BX = 0x34 B. EAX = 0x4321, BX = 0x43 C. EAX = 0x2345, BX = 0x45

D. EAX = 0x5432, BX = 0x54 E. nenhuma das anteriores

10- (0,5 val) Na arquitectura Intel IA-32 diga qual das seguintes ações ocorre na execução da instrução **`call 100`.**

A. Copia o conteúdo da memória no endereço 100 para o registo EIP (instruction pointer)
B. Coloca o valor 100 no registo EIP (instruction pointer)
C. Coloca o valor 100 no registo ESP (stack pointer)
D. Coloca o valor 100 no topo da pilha de execução do programa
E. Retira o valor no topo da pilha de execução para a posição de memória de endereço 100

11- (0,5 val) No contexto da arquitectura Intel IA-32, descreva as ações efetuadas ao executar a instrução **`ret`**.

A. Retorna de uma função C sem devolver resultado.
B. Coloca no EIP o endereço de uma subrotina, subtrai 4 a ESP
C. Coloca no ESP o endereço de uma subrotina.
D. Coloca no EIP o valor no topo da pilha, soma 4 a ESP
E. Coloca no ESP o valor no topo da pilha.

12- (0,5 val) Considere o seguinte excerto de código *assembly* IA-32. Qual o conteúdo dos registos após a sua execução?

```
            movl  $1, %eax
            cmpl  $3, %eax
            jge   L1
            movl  $-5, %ebx
            jmp   L2
      L1:   movl $10, %ebx
      L2:   cmp   %eax, %eax
```

A. EAX=0 EBX=-5
B. EAX=1 EBX=-5
C. EAX=0 EBX=10
D. EAX=1 EBX=10
E. nenhuma das anteriores

13- (0,5 val) Considere o seguinte excerto de código *assembly* IA-32. Qual o conteúdo dos registos e das flags após a sua execução?

```
            movl  $-1, %eax
            pushl %eax
            add   $2, %eax
            jmp   L2
      L2:   pop   %eax
```

A. EAX=-1 CF=0 OF=0 ZF=0
B. EAX=1 CF=1 OF=0 ZF=0
C. EAX=-1 CF=1 OF=0 ZF=0
D. EAX=1 CF=0 OF=1 ZF=1
E. nenhuma das anteriores

14- (0,5 val) Na arquitectura de um computador o bus de sistema serve para quê?

A. Para, no CPU, ligar os registos à ALU (unidade aritmética e lógica).
B. Para ligar os discos ao controlador de discos.
C. Para ligar a memória ao ecrã e teclado.
D. Para ligar o CPU à memória do computador e aos periféricos.
E. nenhuma das anteriores.

15- (1 val) As arquiteturas de computador usam memórias cache (um ou mais níveis) entre o CPU e a memória central.

a) Estas caches organizam-se em linhas com blocos de memória em vez de bytes arbitrários da memória. Porquê?

A. Porque depois de um acesso há uma grande probabilidade de aceder aos bytes vizinhos no mesmo bloco.
B. Porque quando é necessário remover bytes para dar lugar a novos acessos, assim remove-se logo um bloco.
C. Porque sendo as caches associativas por grupos, cada grupo é um bloco.
D. Porque os acessos ao mesmo byte não tendem a acontecer mas a outros bytes de outros blocos, sim.
E. Porque os acessos a bytes do mesmo bloco não tendem a acontecer mas ao mesmo byte, sim.

b) Considere que numa determinada arquitectura observou-se na execução de um programa um *hit ratio* na cache de 90%. Se o tempo de acesso à cache for de 2ns e o de acesso à memória central for de 20ns, qual é o tempo médio de acesso, para o programa?

A. 3,8ns B. 4ns C. 18,2ns D. 22ns E. 38ns

16 - (1 val) Admita uma arquitectura de um computador com as seguintes características: endereçamento de 20 bits e uma cache de 128KBytes (2^{17}) de capacidade, associativa por grupos (conjuntos) de 8 linhas e onde cada linha tem 128 Bytes. (lembre-se que os zeros à esquerda de um número podem ser omitidos)

a) Para o **byte** com endereço 0111 0011 0001 1001 0101, indique qual é o número do bloco que o contém e qual o valor deslocamento do byte dentro dentro desse bloco.

A. bloco=111 0011 0001 100 deslocamento=1 0101
B. bloco=111 0011 0001 10 deslocamento=1 0101
C. bloco=111 0011 0001 1 deslocamento=1 0101
D. bloco=111 0011 0001 deslocamento=1001 0101
E. bloco=111 0011 000 deslocamento=1 1001 0101

b) Para o **bloco** com endereço 110 0011 0101, indique para que grupo este mapeia e qual é a sua chave.

A. grupo=1 1001 chave=1 0101 B. grupo=1 0101 chave=1 1001
C. grupo=1 100 chave=11 0101 D. grupo=11 0101 chave=1 100
E. grupo=1 10 chave=11 0101 ~~F. grupo=11 0101 chave=1 10~~

17- (0,5 val) Indique porque os operandos na seguinte sequência de instruções podem ser um problema para a eficiente utilização do pipeline do CPU?

```
mov $3, %eax
add %eax, %ebx
mov %ebx, (var)
```

A. Os operandos obrigam sempre a acessos a memória e atrasam assim a execução do pipeline.
B. Devido ao uso de variáveis, a instrução que entra no pipeline pode não ser a próxima a ser executada, tendo nesses casos de se recomeçar com uma nova instrução.
C. A soma (add) obriga a ir à ALU o que provoca atrasos no pipeline.
D. Algumas instruções dependem do término da anterior para poderem obter um operando e serem então executadas.
E. Os operandos que são registos não podem ser obtidos no pipeline do CPU, tendo de se executar fora desse pipeline.

18- (0,5 val) Podemos construir um programa usando funções implementadas em diferentes linguagens, incluindo *assembly*?

A. Não porque cada compilador reconhece uma linguagem e não existe um que compile todas as linguagens.
B. Não porque uma função implementada numa linguagem não pode ser usada por outra linguagem.
C. Sim desde que os vários compiladores tornem públicos os nomes das funções implementadas.
D. Sim desde que os vários compiladores partilhem as convenções de chamada e implementação das funções.
E. Sim desde que sejam apenas C e *assembly*.

19- (0,5 val) Um processador, aquando de uma interrupção do *hardware* (ou de *software* como na instrução INT do Intel), comuta para modo supervisor?

A. Não porque vai apenas executar o código de tratamento da interrupção que executa sempre em modo utilizador.
B. Sim porque necessita de guardar o estado corrente das flags antes de executar o código de tratamento da interrupção.
C. Sim para poder executar todas as instruções necessárias (p.ex. IN/OUT) no código de tratamento da interrupção.
D. Apenas se as instruções no código de tratamento da interrupção o pedirem.
E. Não pois é igual a um *call* para o código de tratamento de interrupção.

20- (0,5 val) Qual a vantagem dos mecanismos de DMA (*Direct Memory Access*) e interrupções para os controladores de IO?

A. Permitem a transferência de blocos de bytes do CPU para o controlador e a interrupção do controlador quando a transferência terminar.
B. Permitem que aquando de uma interrupção se executem as instruções no CPU que permitem transferir um byte para o controlador.
C. Permitem a transferência de um bloco de bytes, sendo que o CPU acede diretamente à memória a cada interrupção para enviar o próximo byte.
D. Permitem que o controlador aceda à memória para a transferência de bytes, interrompendo o CPU a cada byte transferido.
E. Permitem que o controlador aceda à memória e efetue a transferência de bytes, em vez de ser o CPU a fazê-lo, gerando uma interrupção para assinalar o fim da transferência.

21- (0,5 val) Qual é a necessidade de um mecanismo de memória virtual numa arquitetura de computador?

A. Para permitir a paginação da memória.
B. Para que cada programa em memória tenha endereços privados e independentes dos endereços dos outros programas.
C. Para que num CPU com múltiplos *cores* se possa ter caches.
D. Para que os endereços usados por cada programa não levem a fragmentação e a desperdício de memória.
E. Para usar caches e acelerar o acesso a memória por cada programa.

22- (0,5 val) Porque é que num sistema multiprogramado as instruções de acesso aos periféricos (IN, OUT, etc) e de configuração da MMU não podem ser executadas pelos programas normais (dos utilizadores) mas apenas pelo kernel do SO?

A. Porque qualquer programa poderia aceder a todos os periféricos, ou à memória dos outros programas.
B. Porque os programas normais não necessitam de usar periféricos.
C. Para permitir maior desempenho na execução das instruções no CPU.
D. Porque os programas normais necessitam de executar mais instruções.
E. Para ser possível as chamadas de funções e rotinas de tratamento de interrupções.

23- (1,5 val) Uma arquitetura tem endereços virtuais de 20 bits e páginas com a dimensão de 256 Bytes (2^8).
a) Qual o tamanho máximo da tabela de páginas de um processo?

A. 256 M páginas (2^{28})
B. 4 K páginas (2^{12})
C. 64 K páginas (2^{16})
D. 256 páginas (2^8)
E. nenhuma das anteriores

b) Para um dado processo em execução, o conteúdo da tabela de páginas é o seguinte (vista parcial):

0	1
1	0x60
2	0x0A
3	0xFF
...	...

Indique para cada endereço virtual qual o endereço real (físico) obtido após a transformação:

`0x00101 -> endereço real =` 0x6001

`0x00330 -> endereço real =` 0xFF30

`0x00011 -> endereço real =` 0x111

`0 -> endereço real =` 0x100

c) Uma MMU pode incluir uma TLB para acelerar a transformação dos endereços. Indique o conteúdo da TLB após os acessos aos endereços anteriores, assumindo que esta tem apenas 2 entradas e uma política LRU para a substituição.

3	0xff
0	1

24- (2 val)

a) Implemente uma função na linguagem C que, dados dois vetores de tipo double, devolve o seu produto interno. Recorde que o produto interno de dois vetores $\boldsymbol{v} \cdot \boldsymbol{w}$ num espaço euclidiano de dimensão *n*, é obtido pela soma dos produtos das suas coordenadas: $\sum v_i w_i$ para *i=1..n* (a coordenada 1 está na posição 0 do *array, etc*).

```
double dotProd(double v[], double w[], int n) {
        double dp = 0.0;
        for (int i=0; i<n; i++ )
                dp += v[i] * w[i];
        return dp;
}
```

b) Implemente agora a função main de um programa C que pede dois vetores ao utilizador (vet1 e vet2) e, usando a função anterior, escreve no ecrã o seu produto interno. Este programa deve começar por pedir a dimensão *n* e depois os *n* valores de cada vetor. Assuma que os vetores terão sempre dimensão inferior a MAXDIM.

```
#include <stdio.h>
#define MAXDIM 1000
float vet1[MAXDIM], vet2[MAXDIM];
//nota: o tipo devia ser o mesmo que no dotProd
//assumindo que existe um dotProd de floats
//(podia assumir que aqui eram doubles ou qualquer outra solucao)

int main() {
        int n; // dimensao

        scanf("%d", &n);
        for (int i=0; i<n; i++ )
                scanf("%f", vet1+i);    // ou &vet1[i]
        for (int i=0; i<n; i++ )
                scanf("%f", vet2+i);
        printf("prod int: %f\n", dotProd(vet1, vet2, n) );
        return 0;
}
```

c) Suponha agora que dentro da função `main` anterior, pretende guardar o conteúdo do vetor `vet1` num ficheiro de nome "`vetor1.data`". Escreva as instruções que em C lhe permitem fazer isso (note que o ficheiro deve conter o conteúdo do vetor e não uma representação em texto desses valores).

```
FILE *f = fopen("vetor1.data", "w");
fwrite( vet1, sizeof(float), n, f);
fclose(f);
```

25- (2 val) Considere a seguinte função recursiva escrita em C.

```
int fun( int n, int sum ) {
        if ( n==0 ) return sum;
        else return fun( n-1, sum+n );
}
```

a) Escreva em *assembly* IA-32 e usando as convenções usadas nas aulas a chamada desta função com os argumentos 3 e 0, respectivamente, colocando o resultado em `x`. Equivalente a `x=fun( 3, 0 )` que atribui a `x` o resultado 6.

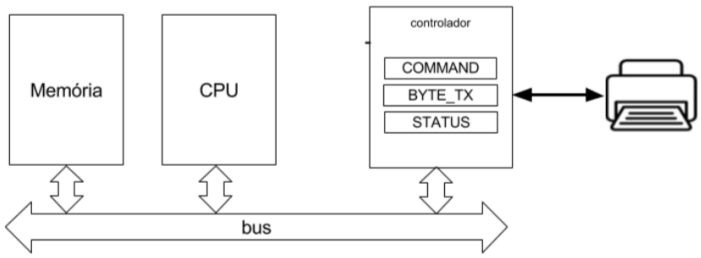
```
.data
        x:   .int   0
.text

        pushl $0
        pushl $3
        call fun
        add $8, %esp
        mov %eax, x
```

b) Implemente agora em *assembly* IA-32 usando as convenções usadas nas aulas, a função recursiva anterior.

```
fun:  push %ebp
      mov %esp, %ebp
      mov 12(%ebp), %eax
      cmp $0, 8(%ebp)
      je fim
      add 8(%ebp), %eax
      push %eax
      mov 8(%ebp), %eax
      dec %eax
      push %eax
      call fun
      add $8, %esp
fim:  pop %ebp
      ret
```

26- (1,5 val) Considere o controlador de uma impressora cuja documentação do *hardware* apresenta a seguinte interface, usando *ports* para registos de comando, estado e dados:



A descrição indica que quando se escreve o valor 1 em COMMAND se inicia o envio para a impressora do byte anteriormente guardado (escrito) em BYTE_TX. Em STATUS, o bit 0 fica a 1 se o controlador/impressora está ocupado a enviar o byte; o bit 2 fica a 0 se a impressora está off-line. Note que enquanto um envio está a decorrer ou quando a impressora está off-line, qualquer byte enviado para BYTE_TX será ignorado e a escrita em COMMAND não tem qualquer efeito. Como esperado, a impressora pode ficar off-line em qualquer altura (p.ex. alguém desliga a impressora a meio de uma impressão).
As instruções de IN e OUT do CPU, para ler e escrever nos *ports*, estão acessíveis em C usando as funções seguintes:

```
unsigned char in(unsigned int portid); // IN de 1 byte
void out(unsigned int port, unsigned char data ); // OUT de 1 byte
```

As constantes STATUS, BYTE_TX e COMMAND representam os respetivos *ports*.

a) O código seguinte do *device driver* pretende enviar um byte para a impressora mas não se comporta como pretendido. Pretende-se enviar o byte para a impressora assim que tal seja possível, tendo em conta a descrição anterior. Esta função só deve retornar quando:

- este byte acabar de ser impresso, devolvendo neste caso 1;
- ou quando a impressora está ou fica off-line, devolvendo neste caso 0.

Corrija esta função!

```
int sendByte( unsigned char byte ) {

  if ( (in(STATUS)& 4)==0 ) return 0;
  out( BYTE_TX, byte );
  while ( (in(STATUS)& 1)==1 )
   ; // waiting....
  out( COMMAND, 1 );
  if ( (in(STATUS)& 4)==0 ) return 0;
  else return 1;
}
```

```
int sendByte( unsigned char byte ) {
  unsigned st;
  while ((st=in(STATUS)) & 5)==5)
    ;     // busy && online
  if ( (st & 4)==0 ) return 0;
  out( BYTE_TX, byte );
  out( COMMAND, 1 );
  while ((st=in(STATUS)) & 5)==5)
    ;
  if ( (st & 4)==0 ) return 0;
  else return 1;
}
```

b) Considere agora que este controlador suporta DMA, permitindo pedir logo a impressão de uma linha de texto, usando os dois *ports* seguintes:

ADDRESS - endereço de memória onde começa a linha de texto a imprimir (assuma que nesta arquitetura os endereços têm a dimensão de um byte);

SIZE - número de bytes (caracteres) a imprimir a partir do endereço indicado em ADDRESS.

No *port* COMMAND pode agora ser dado um novo comando com o valor 2, o que desencadeia a impressão por DMA do texto indicado pelos valores nos registos dos *ports* anteriores. O *port* STATUS continua a reportar a mesma informação, sendo que agora o controlador vai ser dado como ocupado durante a transferência de toda a linha. Implemente agora a função seguinte, responsável por imprimir uma linha de texto, e que deve retornar 1 caso imprima corretamente toda a linha ou 0 caso a impressora esteja/fique off-line antes de terminar a linha.

```
int sendLine( char *line, int linesize ) {
```

```
    unsigned st;
    while ((st=in(STATUS)) & 5)==5)
      ;     // busy && online
    if ( (st & 4)==0 ) return 0;
    out( ADDRESS, line );
    out( SIZE, linesize );
    out( COMMAND, 2 );
    while ((st=in(STATUS)) & 5)==5)
      ;
    if ( (st & 4)==0 ) return 0;
    else return 1;
  }
```