

Teste 2A de Arquitetura de Computadores 24/06/2021

Duração 2h

- *Teste sem consulta e sem esclarecimento de dúvidas*
- *A deteção de fraude conduz à reprovação de todos os envolvidos*

N.º:

Nome:

Q1 - 2,0 valores Para implementar sistemas operativos que suportam múltiplos processos em simultâneo, é necessário que o hardware tenha um certo número de características. Nas alíneas seguintes apresentam-se três dessas características; para cada uma delas, justifique porque é que essa característica é necessária.

a) CPU com dois modos de funcionamento: utilizador e sistema.

b) Unidade de gestão de memória (MMU).

c) Sistema de interrupções.

d) Porque é que as instruções máquina, que permitem programar a MMU e modificar o comportamento do sistema de interrupções, apenas podem ser executadas quando o CPU está em modo sistema?

Q2- 2,0 valores Suponha um sistema computacional com apenas um CPU e todas as características hardware apresentadas na pergunta Q1. O sistema operativo suporta múltiplos processos independentes, através da repartição no tempo do CPU físico pelos vários processos. Cada processo P efetua uma computação que, em cada momento, tem um dado estado S. Que elementos constituem o estado da computação S de um processo?

Q3- 2.5 valores Considere uma máquina com 36 bits de endereço virtual e páginas de 64 Kbytes (2^{16} bytes).

- a) Considere a figura seguinte, em que o bit 35 do endereço virtual é mais significativo e o bit 0 o menos significativo. Preenche os quadrados em branco que indicam qual
- o último bit usado para especificar o deslocamento dentro da página;
 - o primeiro bit do número da página virtual.

35			0
N.º da página virtual		deslocamento	

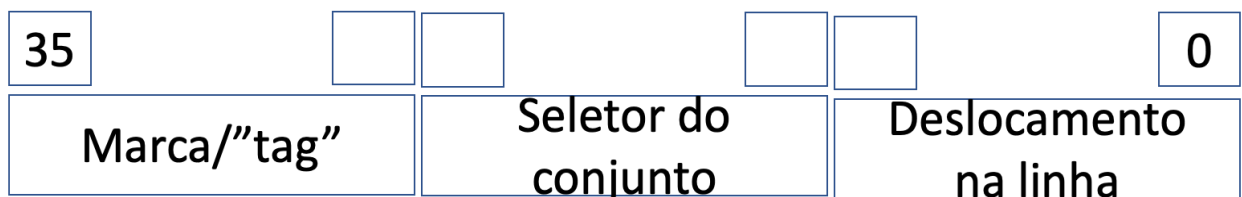
- b) Diga qual é o número de entradas da tabela de páginas de cada processo, expresso como uma potência de 2.

- c) A tabela de páginas está em memória. Explique a conveniência da existência de um TLB (Translation Lookaside Buffer) para converter o número da página virtual em n° da página física.

Q4- 2 valores Considere um sistema informático em que o CPU, a MMU e o sistema operativo suportam paginação a pedido. Explique o que acontece quando um processo P referencia uma página virtual PV e, na entrada PV da tabela de páginas de P, o bit de validade está a 0.

Q5- 2 valores Considere um CPU que tem um endereço físico de 36 bits. O sistema tem um nível de cache único e que é comum ao código e aos dados; o tamanho de uma linha da cache é de 128 bytes (2^7 bytes). A cache é associativa por grupos / conjuntos, tendo cada conjunto 8 linhas, e tem a capacidade de 2 Megabytes (2^{21} bytes). Preencha, justificando, a figura seguinte, indicando quais são os bits usados:

- para indicar o deslocamento dentro da linha da cache;
- para escolher o conjunto em que poderá estar a linha pretendida;
- para a marca ou *tag*.



Q6- 2 valores Considere o seguinte fragmento de programa em C :

```
#define N 10000
int i, j;
float *mat = malloc( N*N*sizeof(float));
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        mat[ i ] [ j ] = 3.1416;
```

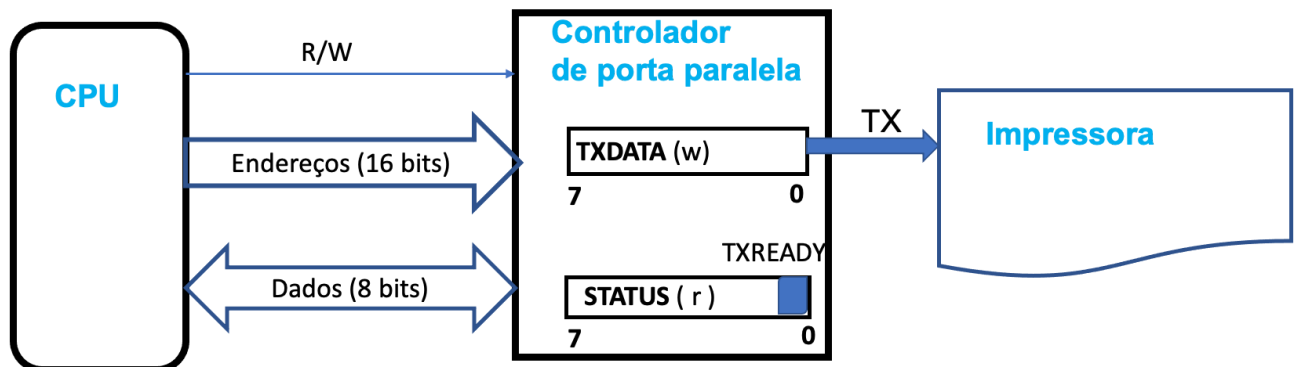
Suponha que este programa é executado num CPU com um único nível de cache, em que a cache de dados e instruções está separada e cada uma tem 1 Mbyte; cada linha da cache tem 64 bytes. Suponha que, como se acabou de reservar espaço para a matriz `mat`, nunca houve acesso às posições de memória onde ela está armazenada e que o endereço inicial de `mat` é um múltiplo de 64; admita ainda que o compilador colocou as variáveis `i` e `j` em registos do CPU. Diga, justificando, qual é o *hit rate* na cache de dados durante a execução dos ciclos **for**. Recorde que um *float* ocupa 4 bytes.

Q7 1.5 valores Considere a figura seguinte que mostra a interação entre um CPU, um controlador de entrada e saída e uma impressora. O CPU pode ler e escrever em registos dos controladores de entrada / saída através de instruções de **in** e **out** que estão acessíveis num programa em C através das funções:

```
void outPort( unsigned short int port, unsigned char val);
unsigned char inPort( unsigned short int port);
```

O controlador tem os seguintes registos que estão acessíveis ao CPU:

- TXDATA (endereço 0x3C8, o CPU pode apenas escrever): quando é escrito um byte neste endereço, esses 8 bits são transferidos em paralelo para a impressora.
- STATUS (endereço 0x3C9, o CPU pode apenas ler): indica o estado em que se encontra a transferência entre o registo TXDATA e a impressora. O bit 0 deste registo (TXREADY) estará a 1 se já acabou a transferência do byte anterior e a 0 se a transferência ainda estiver a decorrer. Sempre que o CPU escreve um byte no endereço TXDATA, este bit fica a 0.



Pretende-se que escreva o código de uma função `void escreverImpressora(unsigned char val)` que envia um byte para a impressora, usando espera ativa.

```
void escreverImpressora( unsigned char val ){
```

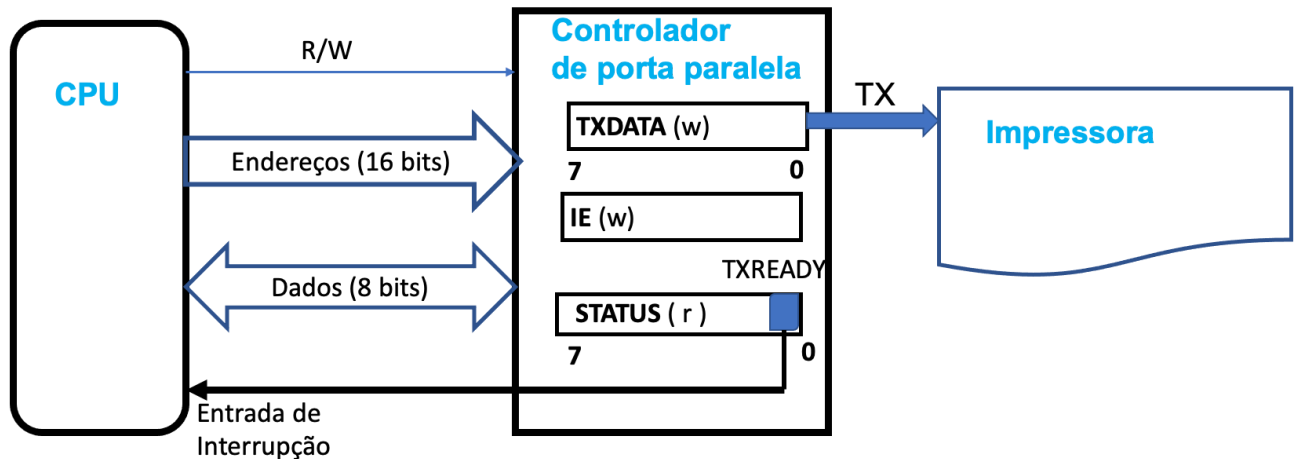
```
}
```

Q8 2.0 valores

Considere a mesma situação descrita na pergunta 7, mas em que ao controlador da porta paralela foi acrescentada a possibilidade de enviar uma interrupção ao CPU, tal como mostra a figura abaixo. O controlador tem um novo registo chamado *IE* (*InterruptEnable*) no endereço 0x3CA que funciona da seguinte forma:

- quando o CPU quer autorizar o controlador a efetuar interrupções escreve o valor 0xFF no registo IE
- quando o CPU quer impedir que o controlador realize interrupções escreve 0x00 no registo IE

Admita que quando é aplicada energia ao controlador, ele fica com a possibilidade de fazer interrupções desativada.



Suponha que está disponível uma estrutura de dados buffer circular, declarado da seguinte forma

```
typedef struct {
    unsigned int put;
    unsigned int get;
    unsigned count;
    unsigned char buf[1024];
} bufcirc;

e com as seguintes operações definidas:
int bufFull( bufcirc *b); // retorna 1 se o buffer estiver cheio; 0 se não estiver
int bufEmpty( bufcirc *b); // retorna 1 se o buffer estiver vazio; 0 se não estiver
unsigned char bufGet(bufcirc *b); // devolve o byte que está há mais tempo no buffer;
// quando é chamada *bc não pode estar vazio
void bufPut( bufcirc *bc, unsigned char b); // coloca o byte b na 1a posição
// livre buffer; assume que *bc não está cheio
```

Pretende-se que escreva o código da função `escreverImpressora(bufcirc *bc, unsigned char val)` que, em vez de usar espera ativa, usa interrupções. Esta função, coloca o byte `val` no buffer circular `*bc` e retorna sem esperar que a transmissão acabe. Se não houver espaço no buffer, retorna -1. Esta função, em conjunto com a rotina de tratamento de interrupções, deve garantir que, quando há bytes para transmitir, as interrupções da porta paralela estão ligadas; quando não há bytes no buffer para enviar, as interrupções da porta paralela devem estar desligadas.

```
int escreverImpressora( bufcirc *bc, unsigned char val ){
```

```
}
```

```
void interruptHandlerImpressora( bufcirc *bc ){

interrupt_return; // executa a instrução máquina que termina uma rotina de
                // tratamento de interrupções

}
```

Q9- 2.0 valores - Sobre o TPC3

```
void * my_malloc( unsigned int size) {
void myFree(void *ptr)
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

#define BLOCK_SIZE sizeof(struct s_block)

typedef struct s_block *t_block;

struct s_block {
    size_t size; // current size of block
    t_block next; // pointer to next block
    int free; // indicates that the block is free (1) or occupied (0)
};

t_block head = NULL; // points to the beginning of the list
t_block tail = NULL; // points to the last element of the list
```

```
t_block find_block(size_t size) {
```



```
}
```

```
t_block extend_heap(size_t s) {  
    t_block b = sbrk(BLOCK_SIZE+s);  
    if (b == (void *)-1)  
        return NULL; /* if sbrk fails, return NULL pointer*/
```

```
    b->size = s;  
    b->next = NULL;  
    b->free = 0;  
    if (head==NULL) head = b;  
    else tail->next = b;  
    tail = b;  
    return b;    // with metadata
```

```
}
```

```
void *myMalloc(size_t size) {
```



```
}
```

```
void myFree(void *ptr) {
```



```
}
```

Q10 – 2.0 valores - Sobre o TPC4

Considere um simulador de cache semelhante aos usados no TPC4 e Ficha 9. Nesta pergunta vai ser desenvolvido um simulador de forma a que a cache simulada seja uma cache associativa pura, ou seja com apenas um conjunto ou grupo. A cache é unificada, isto é, usada para as instruções e os dados. As únicas operações que aparecem no ficheiro são 'R' e 'W'.

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>
```

```
#define HIT 1  
#define MISS 0
```



```


#define LINESIZE      64
#define CACHESIZE     (1024 * 64)

typedef struct {
    unsigned char valid;
    unsigned int tag;
    // a few more fields, if this was a real cache...
} cacheLine;

typedef struct {
    cacheLine lines[CACHESIZE / LINESIZE];
    int next;
    unsigned int Raccesses;
    unsigned int Waccesses;
    unsigned int misses;
} cache;

cache realCache;

void nextLine( cache *Acache) {
    // TODO: update to the next index in the cache, using a FIFO policy
    
}
/* find tag in cache
   returns: line were found or -1 if not found
*/
int findInCache( cache *Acache, unsigned int Atag ) {
    
}

void addMiss(cache *Acache, unsigned int Atag, char mode) {
    // TODO: update the number of read or write accesses, and misses
    // update the line with the tag and valid "bit"
    
    nextLine(Acache); // update the next index
}

```

```

void addHit(cache *Acache, char mode) {
    if (mode == 'R')
        Acache->Raccesses++;
    else Acache->Waccesses++;
}

/**  simulates an access using pure associative cache with write-through
 *  check if addr is in cache and, if not, update cache,
 *  update hit/miss and write counters
 **/

void simulateOneStep(cache *theCache, unsigned int addr, unsigned char mode)
{
    unsigned int tag =  ;

    int cacheLinePosition =  ;

    if (cacheLinePosition < 0)
        addMiss( theCache, tag, mode);
    else
        addHit( theCache, mode);
}

void simulateAllSteps(FILE *tf) {
    unsigned addr;
    char mode;
    while (fscanf(tf, "%x %c", &addr, &mode) == 2)
        simulateOneStep(&realCache, addr, mode); // simulate
                                                    // one cache/memory access
}

int main(int argc, char *argv[]) {
    // código para testar se o número de argumentos está correto omitido
    if ((traceFile = fopen(argv[1], "r")) == NULL) {
        printf("Trace-file '%s' error\n", argv[1]); exit(1);
    }
    memset(&realCache, 0, sizeof(cache));
    simulateAllSteps(traceFile);
    return 0;
}

```