

Arquitetura de Computadores - Teste 2 (Versão C)

21/6/2022 Duração 1h45 sem Consulta

Responda às questões na folha de resposta. Respostas erradas descontam $\frac{1}{3}$ da cotação.

1 (0.8 val.) Qual das seguintes afirmações é verdadeira?

- A) Um endereço físico representa uma localização física na memória RAM.
- B) Um endereço virtual representa uma localização física na memória RAM.
- C) O CPU emite endereços físicos.
- D) Quando, em C, declaramos uma variável `int* x`: o valor de `x` é um endereço virtual e `&x` é um endereço físico.
- E) O tamanho de um endereço virtual é sempre menor do que o tamanho de um endereço físico.

→ Considere um computador com uma hierarquia de memória de 3 níveis:

- 1- **memória cache** com tempo de acesso de t_1 nano-segundos e uma capacidade de c_1 bytes
- 2 - **memória RAM** com tempo de acesso de t_2 nano-segundos e uma capacidade de c_2 bytes
- 3 - **disco rígido** com tempo de acesso de t_3 nano-segundos e uma capacidade de c_3 bytes

2 (0.8 val.) Qual das seguintes afirmações é verdadeira?

- A) t_1 é menor do que t_2 ; t_2 é da mesma ordem de grandeza de t_3 , se o disco rígido for SSD
- B) t_3/t_2 é diretamente proporcional a c_3/c_2 , ou seja, para discos pequenos t_2 e t_3 são da mesma ordem de grandeza.
- C) $c_1 + c_2$ é normalmente da mesma ordem de grandeza de c_3
- D) $t_2 < t_3$ e $c_3 < c_2$
- E) t_1 é menor do que t_2 ; t_2 é muito menor do que t_3 ; $c_1 < c_2 < c_3$

3 (0.8 val.) Assumindo que t_1 é muito menor que t_2 e que c_1 é muito menor do que c_2 , se uma data computação for caracterizada por ter boa localidade temporal e espacial, o seu tempo médio de acesso à memória:

- A) aproxima-se de t_1 , mas só se a memória necessária for menor do que c_1
- B) aproxima-se da média aritmética de t_1 e t_2
- C) aproxima-se de t_2 se a memória necessária for maior do que c_1
- D) aproxima-se $t_1 + t_2$
- E) aproxima-se de t_1 , mesmo que a memória necessária seja consideravelmente maior do que c_1

4 (0.8 val.) As caches organizadas de forma associativa pura são pouco utilizadas porque:

- A) descobrir se um dado bloco está ou não na cache é demasiado demorado (procura em todas as linhas).
- B) porque a quantidade de hardware necessária para encontrar um bloco na cache é diretamente proporcional ao número de linhas.
- C) podem gerar muitas falhas (misses) por conflito.
- D) limitam o tamanho do bloco ao número de linhas da cache.
- E) podem gerar muitas falhas mesmo quando há espaço livre na cache.

5 (0.8 val.) As rotinas conhecidas como "rotina de atendimento de interrupções" (interrupt service routine) são executadas:

- A) pelo periférico que gera a interrupção
- B) pelo PIC (Programmable Interrupt Controller)
- C) pelo controlador do periférico que gera a interrupção
- D) pelo processador (CPU)
- E) pelo TLB (Translate Look-aside Buffer)

Questão 6: Tamanho do bloco = $128 = 2^7 \rightarrow$ 7 bits para o deslocamento: 0 ... 6

Nº conjuntos = Tamanho cache/tamanho conjunto = $2^{21}/(2^7 \cdot 2^3) = 2^{11} \rightarrow$ 11 bits para o conjunto: 7 ... 17

Bits para a chave: 18 ... 31

→ Considere um CPU que gera endereços com 32 bits, que entre o CPU e a RAM está uma cache associativa por grupos (ou conjuntos) e que a memória está dividida em blocos de 128 bytes.

6 (0.8 val.) Para uma cache com 2 MBytes de capacidade e conjuntos de 8 linhas, indique quais são os bits utilizados para identificar a chave (ou tag) de um dado endereço.

- A) 18 a 31 B) 17 a 31 C) 15 a 31 D) 16 a 31 E) 19 a 31

7 (0.8 val.) Assuma agora uma cache que requer em 14 bits para identificar o número de um dos seus conjuntos, dada a instrução `mov (5000), %AX` que bytes são transferidos para o registo AX? **Nota:** $5000_{(10)} = 1001110001000_{(2)}$

- A) os bytes 8 e 9 do bloco com chave 0 que se encontra no conjunto 78
B) o byte 8 do bloco com chave 0 que se encontra no conjunto 78
C) o byte 8 do bloco com chave 0 que se encontra no conjunto 39
D) os bytes 8 e 9 do bloco com chave 39 que se encontra no conjunto 0
E) os bytes 8 e 9 do bloco com chave 0 que se encontra no conjunto 39

7 bits para o deslocamento: $000\ 1000 = 8$
14 bits para o conjunto: $00\ 0000\ 0010\ 0111 = 39$
11 bits para a chave: $000 \dots 000 = 0$

8 (0.8 val.) Uma chamada ao sistema é um evento desencadeado:

- A) pela execução de uma instrução INT e o seu tratamento consiste em alterar o estado do processo de *utilizador* para *administrador*.
B) por um periférico e cujo tratamento, em vez de ser executado pelo próprio periférico, é efectuado por uma rotina de atendimento de interrupção.
C) pela execução de uma instrução INT e o seu tratamento é em tudo idêntico ao de uma interrupção hardware.
D) quando a execução de uma instrução atinge um estado de exceção (como uma divisão por 0).
E) pelo PIC (Programmable Interrupt Controller) e cujo tratamento é em tudo idêntico ao de uma interrupção hardware

9 (0.8 val.) Que hardware permite ao SO obter acesso ao CPU sem depender do comportamento do(s) processo(s) em execução:

- A) a UART B) o PIC C) o timer D) hyper-threading E) múltiplos cores

10 (0.8 val.) A unidade de transformação de endereços - Memory Management Unit (MMU)

- A) É um dispositivo hardware que faz a correspondência entre endereços virtuais de um processo e endereços físicos recebidos pela memória central (RAM).
B) É software, parte do SO, que faz a correspondência entre endereços físicos da memória central (RAM) e endereços virtuais dos processos que aí se encontram.
C) É um dispositivo hardware que faz a correspondência entre endereços físicos da memória central (RAM) e endereços virtuais dos processos que aí se encontram.
D) É um dispositivo hardware que faz a correspondência entre endereços físicos de um processo e endereços físicos recebidos pela memória central (RAM)
E) É software, parte do SO, que faz a correspondência entre endereços virtuais e físicos.

11 (0.8 val.) Suponha que um dado CPU emite endereços virtuais com 32 bits e que a unidade de transformação de endereços (MMU) gere a memória central (RAM) dividindo-a em páginas de 8 KBytes. Quantos bits **P** são utilizados para identificar o número da página virtual e quantos bits **D** são utilizados para o deslocamento dentro da página?

- A) $P = 24$ e $D = 8$ B) $P = 13$ e $D = 19$ C) $P = 8$ e $D = 24$ D) $P = 29$ e $D = 3$ E) $P = 19$ e $D = 13$

8 K = $2^{13} \rightarrow$ 13 bits deslocamento
19 bits para o número da página

12 (0.8 val.) A conversão de um endereço virtual num endereço físico:

- A) é efetuada pela MMU, recorrendo ao SO quando o bit de validade da entrada da página na tabela de páginas é 0.
- B) é sempre efetuada pela MMU sem intervenção do SO.
- C) é efetuada pela MMU, recorrendo ao SO para saber onde está a tabela de páginas
- D) é efetuada pelo SO, recorrendo à MMU para decompor o endereço virtual no par (página, deslocamento).
- E) é efetuada pelo SO, recorrendo à MMU para carregar a página para a RAM sempre que esta está em disco.

13 (0.8 val.) Como é que a MMU sabe qual é a localização na memória RAM da tabela de páginas de um dado processo?

- A) Gera uma interrupção para que o SO lhe indique.
- B) Só existe uma tabela de páginas para todos os processos, cuja localização é definida à partida.
- C) Utiliza dois registos (geridos pelo SO) que contêm o endereço de início e o tamanho da tabela.
- D) A tabela de páginas do processo corrente está guardada no TLB.
- E) Utiliza o registo EBP que tem de ser gerido explicitamente pelo programador, com as instruções push e pop sobre a pilha (stack).

14 (0.8 val.) A técnica de programação de periféricos conhecida com o nome "espera activa" baseia-se em

- A) ler continuamente um registo do controlador do periférico até que a indicação de que o periférico está livre seja activada
- B) esperar, adormecido (sem gastar CPU), que um registo do controlador do periférico indique que este está livre
- C) esperar, adormecido (sem gastar CPU), que uma interrupção gerada pelo periférico indique que este está livre
- D) ler continuamente um registo do controlador do periférico até que uma interrupção, gerada pelo periférico quando fica livre, quebre o ciclo
- E) ler continuamente um registo do controlador do periférico até que uma interrupção, gerada pelo PIC (Programmable Interrupt Controller), quebre o ciclo

15 (0.8 val.) A diferença entre a instrução `in 0x123,%al` e a instrução `mov 0x123,%al` é a seguinte

- A) `in 0x123,%al` é uma instrução mal codificada - faltam parêntesis, devia ser `in (0x123),%al` - enquanto `mov 0x123,%al` coloca no registo al do processador o valor 0x123
- B) `in 0x123,%al` lê um registo (de um periférico) cujo endereço é 0x123, enquanto `mov 0x123,%al` coloca no registo al do processador o valor 0x123
- C) `in 0x123,%al` escreve o conteúdo do registo al do processador para o registo (de um periférico) cujo endereço é 0x123, enquanto `mov 0x123,%al` coloca no registo al do processador o valor 0x123
- D) `in 0x123,%al` coloca no registo al do processador o valor lido da posição de memória cujo endereço é 0x123, enquanto `mov 0x123,%al` coloca no registo al do processador o valor 0x123
- E) As duas instruções estão mal codificadas, pelo que nenhuma das respostas anteriores está correcta

16 (0.8 val.) O mecanismo de acesso directo à memória - DMA (Direct Memory Access)

- A) Permite ao controlador DMA aceder à i) memória central (RAM) e à ii) memória do periférico e assim efectuar transferências entre i) e ii)
- B) Permite ao controlador DMA aceder à i) memória cache e à ii) memória do periférico e assim efectuar transferências entre i) e ii)
- C) Permite ao processador (CPU) aceder à memória do periférico e assim efectuar transferências entre esta e os seus registos.
- D) Permite ao processador (CPU) aceder à memória do periférico e assim efectuar transferências entre esta e a memória central (RAM)
- E) Permite ao driver do periférico aceder à memória do periférico e assim efectuar transferências entre esta e a memória central (RAM).

17 (0.8 val.) Cada núcleo (core) de um processador multicore

- A) é capaz de ler e decodificar instruções independentemente dos restantes núcleos mas partilha, com estes, as unidades de execução (como a ALU)
- B) executa, em cada instante, a mesma instrução que os restantes núcleos, mas sobre dados distintos.
- C) partilha, obrigatoriamente, todos os níveis da hierarquia de memória (cache L1, L2, L3, ..., memória central, disco) com os restantes núcleos.
- D) tem a sua própria memória central (RAM)
- E) **executa o seu próprio ciclo *fetch-decode-execute* independentemente dos restantes núcleos.**

18 (0.8 val.) Um processador com arquitetura RISC diferencia-se de um outro com arquitetura CISC por:

- A) permitir que as operações aritméticas possam aceder a valores em memória.
- B) ter instruções para mover dados entre memória e registos.
- C) ter instruções de tamanho pequeno.
- D) permitir que as operações aritméticas possam aceder a valores em registo.
- E) **ter instruções de tamanho fixo.**

19 (0.8 val.) Considere que o processo *fetch-decode-execute* de um CPU está implementado como um pipeline com 5 estágios: Obter instrução → Descodificar instrução → Obter operandos → Executar instrução → Escrever resultado

- A) **Não é possível concluir a execução das instruções `add $4,%eax` e `add %eax,%ebx` em dois ciclos de relógio consecutivos.**
- B) Não é possível concluir a execução das instruções `add %eax,%ecx` e `add %eax,%ebx` em dois ciclos de relógio consecutivos.
- C) Este pipeline não pode executar instruções de salto (jumps).
- D) Este CPU tem um débito de 5 instruções por ciclo de relógio.
- E) Neste CPU uma instrução demora um ciclo de relógio a executar os 5 estágios.

20 (0.8 val.) Indique qual das seguintes afirmações é falsa.

A unidade de processamento gráfico (GPU) de um computador atual

- A) expõe um conjunto de instruções que inclui operações para o desenho de formas geométricas, como elipse.
- B) pode ser um componente hardware externo à placa (pacote) do processador (que inclui os cores, caches, ...).
- C) é um componente hardware programável, ou seja pode executar programas.
- D) **pode ser incorporada no pipeline de um core do processador, ao nível de uma ALU.**
- E) pode ser um componente hardware interno à placa (pacote) do processador (que inclui os cores, caches, ...).

21 Considere um cache associativa por grupos (ou conjuntos) semelhante à usada no TI3

```
unsigned block_size;    // tamanho de um bloco em bytes
unsigned number_lines_per_set; // número de linhas por conjunto

typedef struct {        // tipo de uma linha
    unsigned char valid;
    unsigned char dirty;
    unsigned char lru;
    unsigned int tag;
    unsigned char* block;
} line_t;

typedef line_t* set_t; // tipo de um conjunto

set_t* cache;          // a cache. Assuma que foi inicializada previamente.

unsigned char* memory; // a memória central (RAM). Assuma que foi inicializada previamente.
```

a) (1 val.) Implemente a função `find_in_set` que pesquisa num dado conjunto (`aset`) da cache se está presente o bloco com a chave `atag`. Caso a pesquisa tenha sucesso, a função deve retornar o número da linha onde o bloco se encontra, caso contrário deve retornar -4.

```
int find_in_set(unsigned aset, unsigned atag) {

    for (int line = 0; line < number_lines_per_set; line++) {
        if (cache[aset][line].valid && cache[aset][line].tag == atag)
            return line;
    }

    return -4;

}
```

b) (1 val.) Implemente a função `copy_to_cache` que copia o bloco `ablock` da memória para a linha `aline` do conjunto `aset` da cache. **Nota:** pode utilizar a função `void *memcpy(void *dest, const void * src, size_t n)`

```
void copy_to_cache(unsigned aset, unsigned aline, unsigned ablock) {

    memcpy(cache[aset][aline].block, &memory[ablock*block_size], block_size);

    ou

    int mem_block_index = ablock * block_size;
    for (int i = 0; i < block_size; i++)
        cache[aset][aline].block[i] = memory[mem_block_index + i];

}
```

(vire, se faz favor)

22 Considere uma versão simplificada do controlador da porta série e do PIC utilizados no TI4. Pretende-se configurar o controlador e o PIC para receber um fluxo de bytes do dispositivo. À semelhança do TI4, à medida que vão sendo lidos do controlador, os bytes são guardados num buffer intermédio. Para tal assuma que tem à sua disposição as funções

```
int bufFull() // retorna 1 se o buffer está cheio e 0 caso contrário
void bufPut(unsigned char byte) // coloca o byte no buffer (não testa se está cheio)
```

Os registos do controlador do dispositivo são:

DADOS (endereço **0x106**) - contém o último byte recebido pelo controlador.

ESTADO (endereço **0x107**) - contém o estado atual do controlador.

CONF (endereço **0x108**) - contém a configuração atual do controlador. O bit 3 a 1 indica que o controlador gerará interrupções sempre que haja um novo byte no registo de DADOS para ser lido.

Os registos do PIC são

MASCARA (endereço **0x204**) – indica quais das 8 linhas da interrupção estão ativas (podem interromper o CPU). O controlador deste exercício está ligado à linha 2.

COMANDO (endereço **0x205**) – registo para onde devem ser enviados comandos dirigidos ao PIC, por exemplo EOI (End Of Interrupt) com o valor 0x20.

a) (1.0 val.) Implemente a rotina de tratamento de interrupção `isr` para a receção de um byte do dispositivo. O byte deve ser lido do controlador e apenas colocado no buffer se não tiver sido encontrado um erro (ou seja se o bit 7 do registo de ESTADO estiver a 0).

```
void isr() {

    unsigned char state = in(0x107); // read state
    unsigned char byte = in(0x106); // read data
    if ( ((state & 0b10000000) == 0) && !bufFull() )
        bufPut(byte);

    out(0x205, 0x20);

}
```

b) (1.0 val.) Apresente a sequência de instruções necessárias para configurar: (1) o controlador do dispositivo, de forma a que este gere interrupções sempre que há um novo byte no registo de DADOS para ser lido, e (2) o PIC para que este deixe passar a referida interrupção para o CPU.

```
out(0x108, in(0x108) | 0b00001000);
out(0x204, in(0x204) & 0b11111011);
```