Repescagem - 2.º Teste de Introdução à Arquitetura de Computadores — IST – LEIC-T

1.º Semestre 2015/2016 — Duração: 60 minutos — 18 janeiro 2016

NOME		NÚMERO	

1. (1 + 2 + 1 valores) Considere o seguinte programa, que usa rotinas de interrupção para alterar o valor de um display hexadecimal.

```
PLACE       1000H
contador:   WORD    6
pilha:      TABLE   100H
fim_pilha:
tab:        WORD    0
            WORD    rot1
            WORD    0
            WORD    rot3
PLACE       0
            MOV     SP, fim_pilha
            MOV     BTE, tab
            MOV     R2, contador
            EI3
            EI
fim:        JMP     fim
```

```
rot1:   PUSH    R1
        MOV     R1, [R2]
        ADD     R1, 2       ; incrementa
        MOV     [R2], R1    ; atualiza contador
        POP     R1
        RFE

rot3:   PUSH    R1
        MOV     R1, [R2]
        EI1
        SUB     R1, 1       ; decrementa
        MOV     [R2], R1    ; atualiza contador
        POP     R1
        RFE
```

a) Do lado esquerdo, complete a zona de dados e o programa principal com o necessário para a pilha e as interrupções 1 e 3 funcionarem corretamente (rotinas no lado direito);

b) Suponha que os pedidos de interrupção 1 e 3 se alternam, com um segundo de intervalo, e que a interrupção 1 é a primeira a ser pedida. Indique de seguida a sequência dos 8 primeiros valores que as rotinas de interrupção escrevem no contador (e o número da interrupção em que cada valor é escrito). Justifique os dois primeiros;

Interrupção	1	3	1	3	1	3	1	3
Valor	8	5	7	4	6	3	5	2

A rotina rot3 é a primeira a ser executada, pois a interrupção 1 não está permitida, pelo que rot1 só é invocada quando EI1 é executada, incrementando o contador de 2 unidades. Mas a rot3 leu o valor original (6) e decrementa-o de 1 unidade, escrevendo 5.
NOTA – A situação repete-se a seguir, pois o RFE de rot3 repõe o estado anterior do RE (a interrupção 1 volta a não estar permitida)

c) Repita a alínea anterior (recomeçando o programa), mas supondo agora que na rotina rot3 a instrução EI1 passa para imediatamente antes do POP R1. Justifique as diferenças face ao caso anterior.

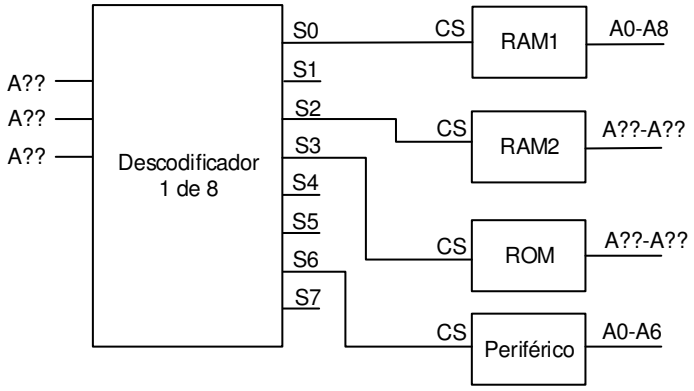
Interrupção	3	1	3	1	3	1	3	1
Valor	5	7	6	8	7	9	8	10

A rotina rot3 continua a ser a primeira a ser executada, mas agora completa a escrita no contador ainda antes da rotina rot1 ser invocada.

2. (2 valores) Uma transmissão de dados é feita por um barramento série assíncrono, com bit de paridade e 2 stop bits. O recetor demora 50 milissegundos a processar cada byte recebido, mas pode ir recebendo o byte seguinte (e apenas esse) durante esse processamento, processando-o imediatamente após processar o anterior. Supondo que o emissor usa um ritmo de transmissão de 1000 bits/seg, indique qual o tempo mínimo, em média, que o emissor tem de esperar entre acabar de enviar um byte e poder começar a enviar o seguinte. Justifique.

Para transmitir um byte, para além dos 8 bits de dados é preciso enviar um start bit, um bit de paridade e dois stop bits, ou seja, 12 bits no total.
Logo, a uma taxa de transmissão de 1000 bits/seg, o envio de um byte gasta 12 milissegundos. Como o recetor demora 50 milissegundos a processar um byte, o emissor terá de esperar pelo menos 38 milissegundos até poder começar a enviar o seguinte.

3. (3 valores) Considere o seguinte sistema de descodificação de endereços utilizado por um processador de bus de dados de 8 bits e bus de endereços de 16 bits. Preencha a tabela com os bits de endereço a que cada dispositivo deve ligar, a sua capacidade (decimal) e os endereços de início e de fim (em hexadecimal) em que esse dispositivo está ativo (não considerando endereços de acesso repetido - espelhos).



Dispositivo	Bits de endereço	Capacidade (bytes) (decimal)	Início (hexadecimal)	Fim (hexadecimal)
Descodificador	**A11-A13**			
RAM1	A0-A8	**512**	**0000H**	**01FFH**
RAM2	**A0-A9**	1 K	1000H	**13FFH**
ROM	**A0-A10**	**2 K**	**1800H**	1FFFH
Periférico	A0-A6	**128**	**3000H**	**307FH**

4. (2 valores) Considere a seguinte tabela de verdade, relativa a uma função de quatro entradas e uma saída. Simplifique a respetiva função, preenchendo a tabela de Karnaugh e escrevendo a expressão algébrica simplificada.

A	B	C	D	Z
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**1**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**0**
1	1	0	0	**1**
1	1	0	1	**1**
1	1	1	0	**1**
1	1	1	1	**0**

		CD			
		00	01	11	10
AB	00		**1**	**1**	**1**
	01		**1**	**1**	**1**
	11	**1**	**1**		**1**
	10	**1**			**1**

Z = $A\overline{D} + AB\overline{C} + \overline{A}C + \overline{A}D$

5. (1 + 2 valores) Suponha que a *cache* do PEPE (processador com 16 bits de endereço, endereçamento de byte) é de mapeamento direto, com uma capacidade de 512 palavras (blocos de 8 palavras).

   a) Quantos blocos tem a cache?

   64

   b) Suponha que um programa num PEPE sem *cache* gasta 20% do seu tempo de execução T em acessos de dados à memória e que num PEPE com *cache* os acessos à memória em caso de *cache hit* demoram 10% do tempo dos acessos em caso de *cache miss*. Assumindo uma *hit rate* de 80%, qual será o tempo de execução T_C do mesmo programa no PEPE com *cache*, em função de T? Justifique.

O tempo de execução T do programa no PEPE sem cache será:

$T = 0{,}8 * T + 0{,}2 * T$ (20% gasto em acessos à memória)

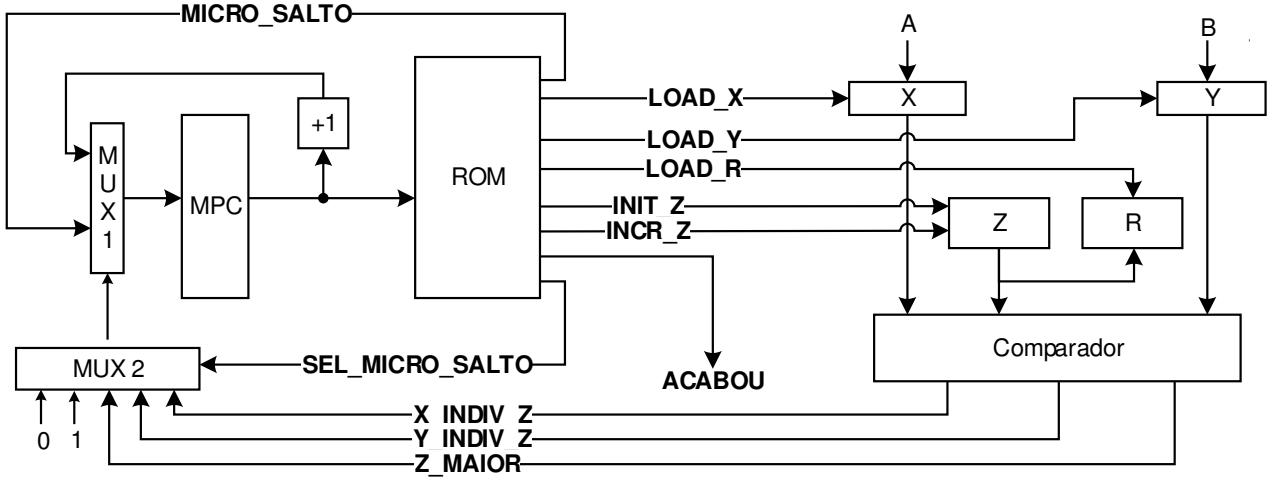
Com cache, o tempo médio t_m de acesso à memória será:

$t_m = (0{,}8 * 0{,}1 + 0{,}2 * 0{,}9)\ t_M = 0{,}26\ t_M$ (t_M: tempo de cada acesso, sem cache)

Assim, o tempo de execução T_C no PEPE com cache será:

$T_C = 0{,}8 * T + 0{,}2 * T * 0{,}26 = 0{,}852\ T$ (melhoria apenas no tempo gasto em acessos à memória)

6. (2 + 1 valores) Considere o circuito seguinte, que permite obter no registo Z o mdc (máximo divisor comum) entre dois números A e B (positivos e maiores que 1). Depois de carregar os valores A e B nos registos X e Y, o algoritmo inicializa Z e R a 1 e vai testando sucessivamente se X e Y são divisíveis por Z, caso em que atualiza R com o valor de Z. Termina quando Z for maior que X ou que Y, altura em que R conterá o mdc, o sinal ACABOU é ativado e o programa fica em salto infinito na mesma microinstrução.



a) Preencha a tabela seguinte com os sinais necessários para implementar o algoritmo. Indique apenas os sinais relevantes em cada ciclo de relógio e deixe em branco as restantes células.

Endereço	Microinstrução (RTL)	LOAD_X	LOAD_Y	INIT_Z	INCR_Z	LOAD_R	ACABOU	SEL_MICRO_SALTO	MICRO_SALTO
0	X ← A; Y ← B ; Z ← 1;	SIM	SIM	SIM					
1	R ← Z					SIM			
2	(Z_MAIOR = 1) : MPC ← 7							Z_MAIOR	7
3	(X_INDIV_Z = 1) : MPC ← 6							X_INDIV_Z	6
4	(Y_INDIV_Z = 1) : MPC ← 6							Y_INDIV_Z	6
5	R ← Z					SIM			
6	Z ← Z + 1; MPC ← 2				SIM			1	2
7	ACABOU ← 1; MPC ← 7						SIM	1	7

b) Se usarmos uma ROM de microprograma com 16 bits de largura, quantos bits **NÃO** serão necessários? Justifique.

Os primeiros 6 sinais são de um bit cada um.

O sinal SEL_MICRO_SALTO gasta 3 bits (5 hipóteses).

O sinal MICRO_SALTO gasta 3 bits (8 microinstruções).

Logo, precisamos de uma ROM com pelo menos 12 bits de largura, pelo que ficarão 4 bits livres.

7. (1,5 + 1,5 valores) Imagine um processador com endereçamento de byte, capaz de endereçar um espaço virtual de 000 000H até FFF FFFH, enquanto o espaço de endereçamento físico vai de 00000H até FFFFFH. As páginas físicas têm uma dimensão de 4 Kbytes.

a) Preencha a tabela seguinte com os valores que decorrem desta informação.

N.º bits do espaço virtual	**24**
N.º bits do espaço físico	**20**
N.º páginas virtuais	**4 K**

b) Suponha que a TLB é totalmente associativa de 8 entradas e há apenas 64 Kbytes de memória física, localizada a partir do endereço 10000H. Após reset, o processador acede aos seguintes endereços virtuais:

207 3B8H
4B8 35AH
0A2 1A0H
1EF 5BEH
4B8 FFEH
207 8FCH

Indique, na tabela a seguir, um possível estado do conteúdo da TLB imediatamente após estes acessos. Arbitre o que for necessário (não há solução única) e preencha apenas o que for relevante.

Posição da TLB	Bit validade	N.º página virtual (hexadecimal)	N.º página física (hexadecimal)
0	**1**	**207**	**10**
1	**1**	**4B8**	**11**
2	**1**	**0A2**	**12**
3	**1**	**1EF**	**13**
4	**0**		
5	**0**		
6	**0**		
7	**0**		