Arquitectura de Computadores

3º Teste- Versão A *17 de Junho de 2011* *Duração 1h30m*

- Sem esclarecimento de dúvidas. Explicite nas suas respostas as hipóteses que assume
- Sem consulta; não é permitido o uso de calculadoras ou telemóveis
- A detecção de fraude implica a reprovação na cadeira a **todos** os envolvidos

Nome Nº

Perguntas de escolha múltipla- Total 5.4 val – respostas erradas descontam 20% da cotação. A nota mínima nesta parte do teste é 0.0 valores

1- 0.9 val Suponha que na unidade de controlo do Pentium existem dois *indicadores* relacionadas com o sistema de interrupções:
PI : está a 1 se existe uma interrupção pendente, e a 0 se não há nenhuma interrupção pendente; esta flag é posicionada a 1 quando é accionada a linha de interrupção do CPU, quando ocorre uma excepção ou quando é executada uma instrução *INT XX*
IE: está a 1 se o CPU está receptivo a interrupções; é colocada a 1 pela execução da instrução *sti* e colocada a 0 pela instrução *cli*.
Para que se salte para a rotina de tratamento de interrupção é preciso que
1. PI=0 e IE=0
2. PI=0 e IE=1
3. PI=1 e IE=1
4. PI=1 e IE=0

Resposta correcta:

2- 0.9 val Quando se salta para a rotina de tratamento de interrupções é preciso salvaguardar o estado da computação, de forma a que quando a rotina de tratamento de interrupção acaba, o programa interrompido continue a ser executado como se a interrupção não tivesse sido existido. No caso do Pentium, o hardware

1. empilha o *Instrucion Pointer*
2. empilha o *Instruction Pointer* e o registo de *Flags*
3. guarda o *Instruction Pointer* e todos os registos do CPU
4. empilha o *Instruction Pointer,* o registo de *Flags* e todos os registos do CPU

Resposta correcta:

3- 0.9 val Num dado CPU todas as instruções máquina são executadas em 4 fases que duram um ciclo de relógio: *fetch* (obtenção da instrução), obtenção dos operandos, execução da operação e escrita dos resultados. Se a unidade de controlo utilizar *pipelining* na execução de instruções, havendo quatro unidades funcionais que executam cada uma das fases da instrução,
1. todas as instruções máquina passam a ser executadas em um ciclo de relógio
2. as instruções máquina continuam a ser executadas em 4 ciclos de relógio, mas pode-se conseguir completar uma instrução em cada ciclo de relógio
3. algumas instruções passam a ser executadas em 4 ciclos de relógio e outras em um ciclo de relógio
4. todas as instruções passam a ser executadas em dois ciclos de relógio

Resposta correcta:

4- 0.9 val Um CPU com N bits de endereço usa uma cache associativa pura com linhas de 128 bytes; a capacidade total da cache é 128 KBytes. O endereço é dividido em duas partes: a *marca* ou *tag* e o *deslocamento* dentro da linha. Considerando o bit 0 do endereço como o menos significativo e bit N-1 o mais significativo,

1. a marca ou *tag* ocupa os bits de 0 a N-8 e o deslocamento dentro da linha os bits N-7 a N-1
2. a marca ou *tag* ocupa os bits de 0 a N-7 e o deslocamento dentro da linha os bits N-6 a N-1
3. a marca ou *tag* ocupa os bits de N-1 a 7 e o deslocamento dentro da linha os bits 6 a 0
4. a marca ou *tag* ocupa os bits de N-1 a 8 e o deslocamento dentro da linha os bits 7 a 0

Resposta correcta:

5- 0.9 val Considere que entre o CPU e a memória central se encontra uma unidade de gestão de memória (ou MMU - *Memory Management Unit*) que permite a gestão da memória central usando um registo base e um registo limite.
1. A MMU permite a recolocação de programas na memória central e assegura que nenhum programa consegue fazer acesso a memória que não lhe está atribuída pelo SO
2. A MMU não permite a recolocação de programas na memória central, mas assegura que nenhum programa consegue fazer acesso a memória que não lhe está atribuída pelo SO
3. A MMU permite a recolocação de programas na memória central, mas não assegura que um programa não consegue fazer acesso a memória que não lhe está atribuída pelo SO
4. A MMU não permite a recolocação de programas na memória central nem assegura que um programa não consegue fazer acesso a memória que não lhe está atribuída pelo SO

Resposta correcta:

6- 0.9 valores No Pentium a memória física é gerida por uma unidade de gestão de memória (ou MMU - *Memory Management Unit*) que divide a memória física em páginas com 4 Kbytes. Supondo que o bit 0 do endereço é o menos significativo,
1. os bits 31 a 11 do endereço virtual especificam o número da página virtual e os bits 10 a 0 definem o deslocamento dentro da página
2. os bits 0 a 19 do endereço virtual especificam o número da página virtual e os bits 20 a 31 definem o deslocamento dentro da página
3. os bits 31 a 12 do endereço virtual especificam o número da página virtual e os bits 11 a 0 definem o deslocamento dentro da página
4. os bits 0 a 20 do endereço virtual especificam o número da página virtual e os bits 21 a 31 definem o deslocamento dentro da página

Resposta correcta:

Perguntas sobre a parte teórica (4.6 valores)

7- 2.1 valores Considere um controlador de disco que suporta DMA (*Direct Memory Access*) e que assinala o final de uma transferência enviando uma interrupção ao CPU. Suponha que se pretende ler um bloco N do disco com B bytes para o endereço E de memória central. Descreva as várias fases da transferência indicando as acções efectuadas pelo CPU e pelo controlador do disco.

8- 2.5 val Suponha um CPU que emite endereços com 5 bits e usa uma cache de mapa directo em que cada linha tem 4 bytes. A cache tem 4 linhas. O CPU emitiu a seguinte sequência de endereços 00010 (2), 00011 (3) , 00101 (5), 00100 (4), 10010 (18), 00001 (1). Preencha o quadro seguinte indicando para cada linha se há um *hit* ou um *miss*. Justifique brevemente.

Endereço	Hit	Miss	Justificação
00010		X	Inicialmente a cache está vazia
00011			
00101			
00100			
10010			
00001			

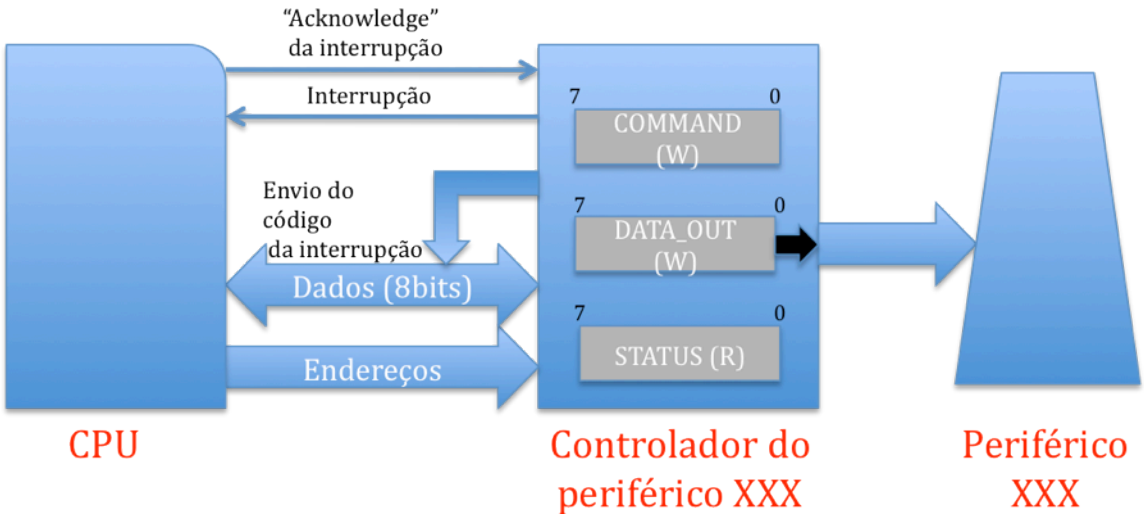
Perguntas sobre a parte prática (10.0 valores)

As perguntas de 9 a 13 pressupõem um ambiente, do **ponto de vista de software**, semelhante ao utilizado nas aulas práticas:
- sistema operativo *FreeDos*
- compilador *turboC* com as seguintes extensões ao C
  - `unsigned char inportb( unsigned int portid)`
  - `void outportb(unsigned int portid, unsigned char value)`
  - `void disable (void)`
  - `void enable(void)`

Supondo a existência de uma rotina de rotina de tratamento de interrupções `void my_isr()` o endereço dessa subrotina é colocada na entrada **5** do vector de interrupções através da invocação da função `setvect( 5, my_isr ).`

- um módulo de software que manipula um *buffer* circular e que exporta as seguintes operações
  - `void buffer_init()`: inicializa a estrutura de dados que descreve o *buffer*
  - `int buffer_full()`: retorna 1 se não há espaço e 0 se há
  - `int buffer_empty()`: retorna 1 se o *buffer* está vazio e 0 se não está
  - `void buffer_put(unsigned char c)`: coloca `c` na 1ª posição livre do *buffer*; assume que há espaço
  - `unsigned char buffer_get()`: retorna o elemento que está há mais tempo no *buffer*; assume que este tem pelo menos um elemento

Do ponto de vista hardware: O controlador do periférico XXX é o único controlador ligado à linha de interrupção do CPU. Depois do controlador do periférico accionar a linha de interrupção do CPU, aguarda que este active a linha de "Acknowledge" da interrupção e coloca no bus de dados o **código 0x0F**; ao mesmo tempo desactiva a linha de interrupção.



Todos os registos do controlador têm 8 bits e o bit 7 é o mais significativo.
O *controlador do periférico* XXX tem os seguintes registos:
- **0x200: DATA_OUT**: registo que só se pode escrever; o valor a 8 bits escrito neste registo é enviado para o exterior pelo controlador
- **0x201 STATUS:** registo só de leitura; o bit 3 a 1 indica que o controlador está pronto para enviar mais um byte; quando o CPU escreve no registo DATA_OUT este bit fica a 0
- **0x202 COMMAND:** registo só de escrita: quando o bit 7 é colocado a 1 o controlador interrompe o CPU quando pode enviar mais dados; quando está a 0 não interrompe o CPU

9- 2.0 valores Pretende-se que escreva o código de uma rotina em C com o seguinte protótipo
`void  out_XXX( unsigned char c)`
que envia para o periférico XXX o byte `c` utilizando espera activa.

10- 1.5 valores Supondo disponível a função out_XXX() da alínea anterior, escreva o código de uma função

```
void out_file_XXX( char *filename)
```

que envia todos os bytes do ficheiro `filename` para o periférico XXX

11- 2.5 valores Escreva o código da função
`void  out_XXX_int(  unsigned char c)`
que envia um byte para o periférico XXX utilizando interrupções e um buffer circular.

12- 2.5 valores Escreva o código da rotina de atendimento de interrupções enviadas pelo controlador do periférico XXX que corresponde à rotina `out_XXX_int( ...)`, o que quer dizer que utiliza o buffer circular .

13- 1.5 valores Escreva o código da rotina de inicialização do sistema de interrupções do controlador do periférico XXX; esta rotina será invocada antes da primeira chamada de `out_XXX_int( ...)`