Teste 2A de Arquitetura de Computadores
12/6/2017 Duração 2h *sem consulta*

Número: _____ *Nome:* _____

Em cada pergunta de respostas múltiplas, respostas erradas descontam ⅓ da cotação da pergunta/alínea.

Q1- 0,6 val. A organização interna do CPU num pipeline de execução, permite:
A. Diminur o tempo de execução de cada instrução
B. Aumentar o número de instruções executadas por unidade de tempo
C. Aumentar o número de acessos a memória
D. Diminuir o número de acessos a memória por unidade de tempo
E. Tornar os acessos à memória mais rápidos

Q2- 0,6 val. No processamento de um conjunto de instruções no pipeline de um CPU podem ocorrer problemas (*pipeline hazzards*) e provocar o seu *stall*. Qual das situações apresentadas NÃO gera problemas?
A. Uma instrução usa como operando fonte um registo que foi usado como destino (resultado) noutra instrução que já foi totalmente executada.
B. Uma das instruções é um salto (*jump*) para uma instrução distante desse *jump*.
C. Uma instrução usa como operando um registo que é o destino (resultado) na instrução anterior.
D. Uma instrução de salto condicional depende de uma condição que é avaliada na instrução anterior.
E. O processador termina a execução de uma subrotina (*ret*).

Q3- 0,6 val. A programação de entradas/saídas por interrupções:
A. É menos eficiente do que a espera activa porque usa *buffers*.
B. Permite a transferência de dados entre os dispositivos periféricos e a memória sem intervenção do CPU.
C. É mais eficiente do que a espera activa porque é executada em modo supervisor.
D. Permite gerir múltiplos periféricos de forma mais eficaz do que a espera activa.
E. Permite gerir um ou mais periféricos mantendo o CPU em modo utilizador.

Q4- 0,6 val. Quando a linha de interrupção do CPU é accionada por um controlador (como acontece no caso dos processadores Intel), o CPU…
A. Salta imediatamente para a rotina de tratamento da interrupção.
B. Salta imediatamente para a rotina de tratamento da interrupção, mas só se as interrupções estiverem desinibidas (ligadas).
C. Acaba a execução da instrução corrente e salta para a rotina de tratamento da interrupção.
D. Acaba a execução da instrução corrente e salta para a rotina de tratamento da interrupção, mas só se as interrupções estiverem desinibidas (ligadas).
E. Acaba a execução da instrução corrente e salta para a rotina de tratamento da interrupção, mas só se as interrupções estiverem desinibidas (ligadas) e o os registos de uso geral livres.

Q5- 0,6 val. Considere o trabalho prático de transferência de ficheiros com interrupções na recepção. Diga qual a afirmação correta:
A. O programa vai chamar explicitamente a rotina de tratamento de interrupções para ler um byte.
B. Caso o buffer esteja cheio, a rotina de tratamento de interrupções vai esperar num ciclo até o programa retire um byte do buffer, e só depois coloca o byte no buffer.
C. O programa, sempre que quer receber um byte, vai testar primeiro o dispositivo para saber se existe algum byte disponível para ser lido.
D. A rotina de tratamento de interrupções deve esperar até que seja possível ler o byte do dispositivo.
E. O programa, sempre que quer receber um byte, irá lê-lo do buffer.

Q6- 0,6 val. O GPU consiste em:
A. O conjunto de operações do CPU que lidam com operações gráficas.
B. Um co-processador para instruções vetoriais de uso geral.
C. Um *bus* que permite ligar várias placas gráficas num único computador.
D. Um co-processador dedicado a operações gráficas.
E. Uma placa gráfica que permite utilizar vários ecrãs num único computador.

Q7- 0,6 val. Assuma que se pretende ler um ficheiro de pequenas dimensões (e.g., um ficheiro com 300 bytes, que ocupa apenas um bloco/sector) num disco magnético (HDD). Esta operação pode ser subdividida em várias suboperações (algumas realizadas por *software* e outras por *hardware*). Qual das suboperações seguintes, tem normalmente, maior impacto no tempo de execução da operação total? (Ou seja, normalmente qual das suboperações seguintes demora mais tempo a ser executada?)

A. Posicionar a cabeça de leitura sobre a pista apropriada (tempo de seek).
B. Esperar que o bloco pretendido chegue à cabeça de leitura (tempo de rotação).
C. Leitura do bloco pretendido (tempo de leitura).
D. Transferência do bloco lido, do disco para a memória do computador (tempo de transferência).
E. Execução do código que atende a interrupção e pede os bytes ao controlador do disco.

Q8- 0,6 val. Para que o Sistema de Operação possa suportar múltiplos processos em execução sem que estes interfiram entre si, a arquitectura *hardware* deve suportar alguns mecanismos. Tal inclui:

A. Dois modos de execução, para que algumas instruções só possam ser executadas pelo SO, como por exemplo o CALL e o RET.
B. Duas instruções (IN/OUT) para que só o SO possa aceder aos periféricos em representação dos processos.
C. Dois modos de execução para que algumas instruções só possam ser executadas pelo SO, como por exemplo o CLI e o STI.
D. Duas instruções (CLI/STI) para que só o SO possa aceder aos periféricos em representação dos processos.
E. Uma MMU (*memory management unit*) para que o SO tenha uma TLB para uso dos processos.

Q9- 0,6 val. Considere uma hierarquia de memória com uma cache e memória central RAM, em que o tempo médio de acesso à memória observado é 2,8 ns. Indique qual das seguintes expressões é verdadeira:

A. A cache tem tempo de acesso 3 ns e um hit ratio de 85 % e o tempo de acesso a RAM é 25 ns.
B. A cache tem tempo de acesso 2 ns e um hit ratio de 90 % e o tempo de acesso a RAM é 20 ns
C. A cache tem tempo de acesso 2 ns e um hit ratio de 95 % e o tempo de acesso a RAM é 15 ns
D. A cache tem tempo de acesso 2 ns e um hit ratio de 90 % e o tempo de acesso a RAM é 10 ns
E. A cache tem tempo de acesso 2 ns e um hit ratio de 95 % e o tempo de acesso a RAM é 10 ns

Q10- 0,6 val. Considere uma arquitetura com dois níveis de cache (L1 e L2). Diga qual das seguintes afirmações é verdadeira:

A. O acesso à cache L2 é feito após um miss no acesso à cache L1 e à memória central.
B. O *Hit rate* (taxa de acertos) da cache L2 representa a taxa com que os dados de um determinado endereço é encontrado na memória central.
C. O acesso à cache L2 é sempre efectuado se o endereço não se encontrar na L1 (miss)
D. O acesso à cache L1 é sempre seguido do acesso à L2 e à memória para garantir a atualização da L1.
E. A cache L1 normalmente tem maior capacidade que a cache L2.

Q11- 0,6 val. As caches organizadas por mapa directo:

A. Podem introduzir muitas falhas (*misses*) por conflito.
B. Necessitam de muito *hardware* para gestão/pesquisa em cada linha.
C. Um endereço poderá ser encontrado em qualquer linha, se estiver na cache.
D. Têm menos falhas (*misses*) por conflito que as caches associativas puras pois um endereço será sempre encontrado na mesma linha, se estiver na cache.
E. De todo o tipo de caches são as que têm tempos de acesso mais lentos.

Q12- 3 val. Uma arquitetura de um computador tem as seguintes características: endereçamento de 32 bits; um nível de cache com organização associativa por conjuntos/grupos e as escritas na cache são diferidas (*write-back*); nesta cache, os endereços são interpretados do seguinte modo:

- Os 5 bits menos significativos (do bit 0 ao 4) referem o deslocamento num bloco (ou linha)
- Os 8 bits seguintes (do bit 5 ao 12) referem o número do grupo/conjunto
- Os restantes (do bit 13 ao 31) servem de chave/tag

a) O tamanho em Bytes de um bloco ou linha é

A. 16 B. 24 C. 32 D. 64 E. 128

b) O número de grupos (*sets*) na cache é

A. 128 B. 256 C. 512 D. 1024 E. 4K

c) O endereço `1001 0110 0001 0000 1000 0000 0101 1001` deve estar no grupo:
A. 0 B. 1 C. 2 D. 9 E. 11

d) O endereço `1001 0110 0001 0000 1000 0000 0101 1001` corresponde a um cache hit se a cache contiver no grupo correto a seguinte tag e bit de validade:
A. `1001011000010000100 bit val = 0`
B. `0001000000001011001 bit val = 1`
C. `1001011000010000100 bit val = 1`
D. `0001000000001011001 bit val = 0`
E. `0001000010000000010 bit val = 1`

e) Caso um endereço corresponda a um cache hit e o acesso seja de escrita
A. a linha da cache é imediatamente atualizada na memória
B. é necessário chamar um algoritmo de substituição da linha para actualizar a linha correspondente ao endereço
C. não há necessidade de actualizar o campo "dirty" da linha da cache
D. quando a linha da cache for seleccionada para conter um novo bloco/linha de memória será necessário escrever o seu conteúdo em memória
E. é atualizado o bit "dirty" e a escrita efectuada imediatamente na memória central

Q13- 0,6 val. No contexto da gestão de memória virtual de programas em execução, o conceito de paginação consiste em:
A. Dividir tanto a memória real como as memórias cache em múltiplos blocos do mesmo tamanho.
B. Dividir o espaço de endereçamento dos programas em secções para código e para dados.
C. Desfragmentar a memória real.
D. Dividir o espaço de endereçamento dos programas em páginas de tamanho fixo.
E. Dividir o espaço de endereçamento dos programa em páginas de tamanho variável.

Q14- 0,6 val. Na gestão da memória virtual baseada em paginação a pedido:…
A. Cada processo tem a sua tabela de páginas.
B. Todos os processos partilham a mesma tabela de páginas e TLB.
C. Todos os processos partilham a mesma tabela de páginas, mas não o TLB.
D. O TLB guarda as tabelas de páginas de todos os processos.
E. A tabela de páginas contém uma entrada para cada endereço virtual possível.

Q15- 0,7 val. Suportando paginação por pedido, a tradução de um endereço virtual num endereço real…
A. Pode gerar uma interrupção por falta de página e o *hardware* (CPU e controlador de disco) irá automaticamente ao disco buscar a página em falta.
B. Pode gerar uma interrupção por falta de página e o Sistema de Operação terá de verificar se o processo pode ou não continuar.
C. Nunca gera uma falta de página, pois caso não exista uma entrada para a respectiva página no TLB esta entrada tem de estar na RAM.
D. Nunca gera uma falta de página, pois não é possível a um processo aceder a páginas inválidas.
E. Só gera uma interrupção por falta de página se um processo aceder a uma página inválida.

Q16- 1 val. Suponha que um dado CPU emite endereços virtuais com 36 bits e que a unidade de transformação de endereços gere a memória física dividindo-a em páginas de 16 KBytes.
A. O número de bits necessário para identificar o número da página virtual é 22 e o número de bits necessário para o deslocamento dentro da página é 14.
B. O número de bits necessário para identificar o número da página virtual é 24 e o número de bits necessário para o deslocamento dentro da página é 12.
C. O número de bits necessário para identificar o número da página virtual é 12 e o número de bits necessário para o deslocamento dentro da página é 24.
D. O número de bits necessário para identificar o número da página virtual é 20 e o número de bits necessário para o deslocamento dentro da página é 16.
E. O número de bits necessário para identificar o número da página virtual é 16 e o número de bits necessário para o deslocamento dentro da página é 1024 (1K).

Q17- 3 val. Considere uma arquitectura com endereços virtuais de 32 bits, suporta páginas com dimensão 4 KBytes (2^{12}) e permite paginação a pedido. Preencha o conteúdo do TLB e das primeiras 6 entradas da tabela de páginas que, para a sequência seguinte de acessos a memória, produz os resultados indicados:

acesso: 0x00002004 — resultado: hit no TLB, endereço real gerado: 0xA00DA004
acesso: 0x00003003 — resultado: miss no TLB, página válida, endereço real gerado: 0x00000003
acesso: 0x00000003 — resultado: miss no TLB, página inválida
acesso: 0x00005004 — resultado: hit no TLB, endereço real gerado: 0x0AF0F004
acesso: 0x00004004 — resultado: hit no TLB, endereço real gerado: 0x0A102004
acesso: 0x00006003 — resultado: miss no TLB, página válida, endereço real gerado: 0xAAA0E003

Caso não tenha informação suficiente para preencher alguma entrada, coloque um traço ("—").

TLB:

página virtual	página física

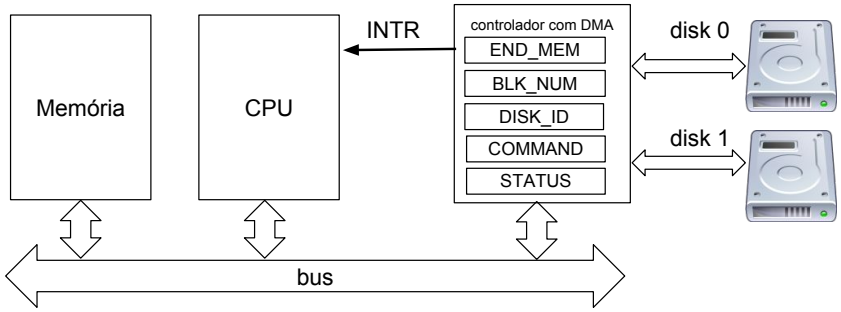
Tabela de Páginas:

página virtual	página física
0	
1	
2	
3	
4	
5	
6	

Q18- 4,5 val. As várias alíneas desta pergunta supõem um ambiente, do ponto de vista do software, semelhante ao usado nas aulas práticas e no mini-projecto, mas com as seguintes funções:

```
unsigned char inportb(unsigned int portid);  // IN de um byte
unsigned int inport32(unsigned int portid);  // IN de 4 bytes
void outportb(unsigned int portid, unsigned char value); // OUT de um byte
void outport32(unsigned int portid, unsigned int value); // OUT de 4 bytes
void enable(void);  // STI - ligar flag de interrupções
void disable(void);  // CLI - desligar a flag de interrupções
setvect(int intrnum, void interrupt(*isr)()); // afetar o vetor de interrupções
```

Do ponto de vista do *hardware*, considere um controlador de discos com capacidade de acesso directo à memória (DMA-Direct Memory Access). O controlador pode controlar até 8 discos, organizados em blocos de 512 bytes (os sectores) e numerados por endereços de 32 bits. Assuma que este é o único controlador ligado à linha de interrupção do CPU. Considere um mecanismo de interrupções vectorizadas e para este periférico deve usar a entrada número 9 para identificar a interrupção.



Todos os registos do controlador têm 8 ou 32 bits. Os registos relevantes são os seguintes, com a dimensão e os endereços de IO (ports) indicados:

- **END_MEM (32bits - 0x300)** — só pode ser escrito e indica o endereço de memória a partir do qual lê ou escreve um sector por DMA.
- **BLK_NUM (32bits - 0x304)** — só pode ser escrito e indica o número do bloco (sector) a ser lido ou escrito no disco indicado em DISK_ID.
- **DISK_ID (8bits - 0x308)** — só pode ser escrito e identifica o disco a ser lido ou escrito.

- **COMMAND (8bits - 0x309)** — só pode ser escrito e desencadeia uma ação no controlador de cada vez que é escrito. Os registos anteriores têm de ter o endereço de memória, o disco e respectivo sector envolvidos na ação. O valor escrito descreve qual a ação a executar, onde cada bit tem o seguinte significado:
  - bit 0 - colocando o valor 1 significa começar uma transferência de 512 bytes da memória para o BLK_NUM do disco DISK_ID; o valor a zero significa transferir do disco para memória.
  - bit 1 - colocando o valor 1 significa que uma interrupção deve ser gerada após concluir a transferência; a zero significa que não é gerada interrupção.
- **STATUS (8bits - 0x30A)** - só pode ser lido;
  - o bit 0 está a um se o controlador está ocupado com uma transferência; a zero se está livre
  - bit 1 fica a 1 se houve um erro durante a ação e 0 se tudo correu bem.

Assuma que não existe mais nenhum *software* a manipular este controlador.

a) Escreva, usando C, a função escreveBloco que pede a escrita do bloco *block* no sector *sect* do disco *disk*. **Deve aguardar** a conclusão da escrita e devolver 0 se houver um erro ou 1 se conseguiu escrever sem problemas.

```
int escreveBloco( unsigned char block[512], long sect, unsigned char disk )
{









}
```

b) Escreva, usando C, a função escreveTudo que pede a escrita de *size* bytes de memória a partir do endereço indicado em *data*, para o disco *disk* no sector *sect* e sectores seguintes. Assuma que *size* é sempre múltiplo de 512 e maior ou igual a 512. Note que **deve usar a função da alínea anterior** (se não a fez assuma que está implementada) e deve devolver um valor indicando erro ou não, como na alínea anterior.

```
int escreveTudo(unsigned char*data, int size, long sect, unsigned char disk)
{









}
```

c) Escreva, em C (tipo turbo C), as duas funções seguintes que realizam a escrita de uma sequência de sectores **usando interrupções**:

`void escreveTudoInt(unsigned char*data, int size, long sect, unsigned char disk)`
Esta função deve iniciar a escrita de *size* bytes de memória a partir do endereço indicado em *data*, para o disco *disk* e sector *sect*, usando o mecanismo de interrupções. Assuma que *size* é sempre múltiplo de 512 e maior ou igual a 512. Nesta função deve programar o vetor de interrupções e o controlador para que uma interrupção seja gerada quando cada sector é escrito. A rotina de atendimento deverá desencadear o envio do próximo sector.

`void interrupt meuISR( )`
Esta função realizará o atendimento de interrupções deste controlador e deve pedir a escrita do próximo sector, ou não fazer nada, caso não haja mais dados para escrever.

```
/* Variáveis globais partilhadas por todas as funções do programa */
unsigned char *xData;
int xSize;
long xSectX;
unsigned char xDisk;

void escreveTudoInt(unsigned char*data, int size, long sect, unsigned char disk)
{
      xData=data; xSize=size; xSect=sect; xDisk=disk;









}
.....

void interrupt meuISR( )
{









}
```