Arquitetura de Computadores

Perguntas de testes anteriores

→ Na hierarquia de memórias das arquiteturas de computadores existe um ou mais níveis de memória cache.
a) Qual é o objetivo deste tipo de memória?

A. Permitir que os programas usem uma memória central (RAM) maior que a instalada.
B. Otimizar o tempo de acesso à memória central (RAM) especialmente para alguns padrões de acesso.
C. Facilitar a interpretação dos endereços de memória permitindo encontrar os seus dados em menos tempo.
D. Permitir a execução de instruções de acesso a memória central (RAM) sem intervenção do software.
E. Permitir que as leituras de alguns endereços de memória sejam mais eficientes por decomposição do endereço em chave (*tag*) e grupo (*set*).

b) Diga se este tipo de memória é "transparente" para o software e justifique.

A. Sim, porque o software não a vê e só é usada pelo sistema de operação.
B. Sim, porque é usada sem alterações no software, sendo totalmente gerida pelo hardware para ser mais eficiente.
C. Não, porque o sistema de operação (que é software) tem de intervir para resolver as faltas/misses.
D. Não, porque temos de programar no software as entradas e saídas da cache com a memória que queremos para ser eficiente.
E. Não, porque o software tem de ser alterado pelo programador para poder usar a cache.

→ Num programa em C o percorrer uma matriz por linhas, cada linha completamente, terá normalmente melhor desempenho do que percorrer por colunas, devido ao melhor aproveitamento da cache. Porquê?

A. Porque esse código exibe melhor localidade temporal e melhor utilização do pipeline de execução.
B. Porque esse código exibe melhor localidade temporal ao estar constantemente a aceder ao mesmo valor que indexa a linha.
C. Porque esse código exibe melhor localidade espacial ao estar constantemente a aceder a posições de memória contíguas da matriz.
D. Porque esse código exibe melhor localidade espacial sempre que se muda de linha da matriz se volta a zero no índice que indexa a coluna.
E. Porque esse código exibe melhor localidade espacial no acesso às instruções que são executadas.

→Indique o que é conseguido quando uma arquitetura suporta paginação de memória.

A. A unidade de transformação de endereços pode aceder a várias páginas virtuais.
B. Os endereços efetivos usados por um programa são transformados em páginas virtuais.
C. Podemos ter espaços de endereços virtuais para cada programa e espaços maiores que a memória real instalada.
D. Permite uma MMU que transforma segmentos virtuais em reais usando páginas intermédias para essa transformação.
E. O espaço de endereços real pode ser superior ao virtual suportado pelos endereços da arquitetura.

→Indique qual a lista de componentes de uma arquitetura que estão por **ordem crescente** do tempo que demora ao CPU a aceder aos respetivos dados.

A. cache L2; cache L1; disco rígido; memória central
B. disco rígido; memória central; cache L1; cache L2
C. cache L1; cache L2; disco rígido; memória central
D. cache L1; cache L2; memória central; disco rígido
E. disco rígido; memória central; cache L2; cache L1

→ Considere uma arquitetura de cache de um computador com as seguintes características: endereçamento de 20 bits, um nível de cache composto por uma cache associativa por grupos (*sets*) de 8 linhas, com escritas diferidas (*write-back*). Cada endereço é interpretado do seguinte modo:

- Os 4 bits menos significativos como o deslocamento num bloco (ou linha)
- Os 7 bits seguintes (do bit 4 ao 10) como o número do grupo

a) Como serão interpretados os 9 bits mais significativos (do bit 11 ao 19)?

A. Chave (ou *tag*) B. Número de página C. Número de grupo (ou *set*)
D. Deslocamento na linha E. Nenhuma das anteriores

b) Qual é o tamanho em Bytes de um bloco ou linha de cache?

A. 2^4 = 16 bytes B. 2^5 = 32 bytes C. 2^7 = 128 bytes D. 8x2^5 = 256 bytes E. 2^{20} = 1Mbytes

c) Quantos grupos (*sets*) existem nesta cache?

A. 2^4 = 16 B. 2^5 = 32 C. 2^7 = 128 D. 8x2^5 = 256 E. 2^{20} = 1Mega

d) Qual é a capacidade da cache (em bytes)?

A. 2^7 = 128 Kbytes B. 8x 2^4 = 128 bytes C. 8x2^7 = 1 Kbytes
D. 8x2^7 x2^4 = 16 Kbytes E. 8x2^7 x2^5 = 32 Kbytes

e) No acesso ao endereço `0000 0110 0000 1000 1011`, diga em que grupo e linha este é procurado na cache e como é verificado se está ou não (ou seja, se é um *hit* ou um *miss*).

A. Procura no grupo 11, na linha 8 se o bit validade está a 1. Se sim é um *hit* e lê da cache; se não é um *miss*.
B. Procura no grupo 8, na linha 11 se o bit *dirty* está a 1. Se sim é um *hit* e lê da cache; se não é um *miss*.
C. Procura no grupo 8 a chave 12. Se a encontra e a linha tem o bit validade a 1 é um *hit* e lê da cache; se não é um *miss*.
D. Procura no grupo 12 a chave 8. Se a encontra e a linha tem o bit validade a 1 é um *hit* e lê da cache; se não é um *miss*.
E. Procura no grupo 11 a chave 12. Se a encontra e a linha tem o bit *dirty* e o bit validade a 1 é um *hit* e lê da cache; se não é um *miss*.

f) Se nesta cache a política de escrita passar a imediata (*write-through*), o que podemos esperar em termos do número de leituras e de escritas na memória central, relativamente à política anterior, para a mesma sequência de acessos a memória.

A. O número de leituras de memória central vai aumentar.
B. O número de leituras de memória central vai diminuir.
C. O número de escritas de memória central vai aumentar.
D. O número de escritas de memória central vai diminuir.
E. O número de escritas de memória central vai se manter.

→ Suponha que num sistema de operação dois programas diferentes estão a ser executados. Em ambos existe uma variável no endereço 10000 e usam a instrução: `mov (10000), %eax`. Qual das situações seguintes é verdade:

A. Os dois programas têm a mesma variável e lêem a mesma memória central física e, logo, o mesmo valor.
B. Quando estão em execução a memória central, física, tem duas posições diferentes com o mesmo endereço, uma para cada programa.
C. O endereço é virtual e, aquando da execução da instrução, é transformado num endereço real distinto nos dois programas.
D. Quando os programas foram carregados em memória para executarem, foram criadas duas caches diferentes, para cada programa ter o seu próprio endereço.
E. Cada programa tem de ser executado à vez, só depois de um terminar completamente é que o outro pode começar a executar, usando os mesmos endereços.

→ Considere uma arquitetura com memória paginada, com TLB (*translation lookaside buffer*) e cache de endereços reais. Indique qual das seguintes situações é possível nesta arquitectura num acesso à memória.

A. O endereço virtual não é resolvido na TLB mas está na cache e pode-se assim obter o conteúdo da memória.
B. O endereço virtual tem de ser resolvido em endereço real pela consulta da tabela de páginas em memória e, depois, obtido da TLB.

C. O endereço virtual é resolvido em endereço real por consulta da TLB e depois procurado na cache, podendo resultar num hit ou miss.
D. O endereço virtual tem de ser resolvido em endereço real pela consulta da cache, sendo depois a página obtida da tabela de páginas em memória.
E. O endereço real é procurado na tabela de páginas em memória e transformado em endereço virtual sendo depois o frame copiado para a TLB.

→ Para programar entradas/saídas (I/O) mas evitando que o programa tenha de ficar num ciclo de espera ativa, onde o CPU executa as instruções que testam se o controlador está apto a efetuar a operação pretendida, o *hardware* pode suportar um determinado mecanismo. Qual e como funciona?

A. Trata-se do mecanismo de pipeline, onde o CPU vai executando instruções até poder efectuar a operação pretendida.

B. Trata-se do mecanismo de DMA (Direct Memory Access - acesso direto a memória), onde a memória interrompe o CPU para executar a operação pretendida.

C. Trata-se do mecanismo de interrupções, onde o controlador desencadeia a interrupção do que o CPU está a efectuar para que CPU execute as instruções que desencadeiam a transferência necessária.

D. Trata-se do mecanismo de DMA, onde o controlador comunica com o CPU para negociar a transferência de bytes da memória para o periférico.

E. Trata-se do mecanismo de interrupções, onde o CPU desencadeia a interrupção do controlador para que o controlador execute a transferência necessária.

→ Indique justificando como o pipeline de execução de um CPU pode melhorar débito de instruções completadas por unidade de tempo.

A. As instruções que estão no pipeline têm acesso privilegiado à cache melhorando o tempo de execução face às restantes instruções.
B. As instruções que estão no pipeline têm acesso privilegiado à memória central melhorando o seu tempo de execução.
C. O pipeline executa várias instruções em simultâneo graças a cada estágio executar completamente uma instrução.
D. O pipeline executa as instruções aritméticas e lógicas, acelerando estas face às de jump/call.
E. As instruções que estão no pipeline são executadas em simultâneo, onde cada estágio executa uma das fases de cada instrução.

→ Indique qual poderá ser a estimativa do tempo médio para efectuar a leitura de um setor de um disco magnético com as características seguintes:

velocidade de rotação: 10000 rpm
tempo médio de seek: 5ms
setores por pista: 500

A. 8ms B. 50ms C. 6ns D. 5ms E. 50µs

→ Uma arquitetura tem endereços virtuais de 24 bits e páginas com a dimensão de 8 KBytes (2^{13}).
a) Qual o número máximo de páginas que um processo pode ocupar?

A. 1024 páginas (2^{10}) B. 2 K páginas(2^{11}) C. 8 K páginas (2^{13})
D. 16 K páginas (2^{14}) E. nenhuma das anteriores

b) Para um dado processo em execução, o conteúdo da TLB na MMU é o seguinte (endereços em hexadecimal):

TLB:

página virtual	*página física (frame)*
0x12	0x100
0x50	0x45A
0x0	0x0

Indique se, conhecendo unicamente o conteúdo do TLB acima indicado, é possível obter o endereço real para cada um dos acessos aos endereços virtuais que se seguem (circule a palavra Não ou Sim). Em caso afirmativo, indique qual o endereço real obtido:

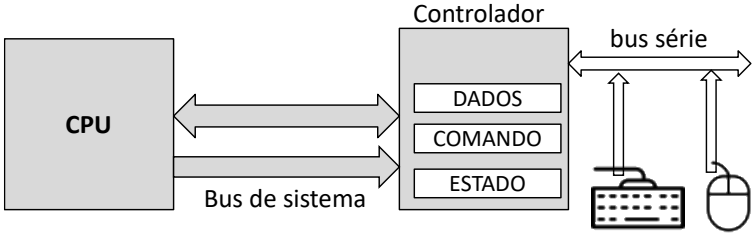
```
0x050010    Não  /  Sim -> endereço real =
0x013001    Não  /  Sim -> endereço real =
0x500111    Não  /  Sim -> endereço real =
0x00009A    Não  /  Sim -> endereço real =
```

c) Nos casos em que não é possível obter logo na TLB o endereço real, tal corresponde a que situação?

A. A página não tem frame atribuída e vai produzir um *page-fault*.
B. A página vai ter o seu frame na tabela de páginas em memória.
C. A página pode ter o seu frame na tabela de páginas em memória ou, se não tiver, produz um *page-fault*.
D. A página não tem frame atribuído mas pode ser atribuído e resolvido pela MMU.

→ Que unidade da arquitetura, como estudada nas aulas, desempenha um papel fundamental na conversão de um endereço virtual num endereço real?
A. ALU – *Arithmetic and Logic Unit*
B. MMU – *Memory Management Unit*
C. Memória Cache
D. *Pipeline* de execução
E. *Bus* de sistema

→ As várias alíneas desta pergunta supõem um ambiente, do ponto de vista do software, semelhante ao usado nas aulas práticas e TPC. Considere uma arquitetura com um controlador de um *bus* série externo onde se ligam ratos e teclados, entre outros periféricos (um pouco à semelhança do USB).



O funcionamento deste controlador e periféricos é o seguinte:

De cada vez que uma tecla é premida, o rato se move ou um dos seus botões é premido, um evento é enviado ao controlador e fica codificado num registo de DADOS. Este tem 2 bytes sendo os 3 bits menos significativos o número do periférico (de 0 a um máximo de 7) e, os restantes 13 bits, a descrição do evento (qual a tecla do teclado, qual o botão ou as novas coordenadas do rato, etc). Quando há um novo evento em DADOS, o bit 0 no registo de ESTADO fica com o valor 1. Os registos relevantes têm 2bytes e são os seguintes, com os endereços de IO (ports) indicados:

- **ESTADO (0x100)** - só pode ser lido;
  - o bit 0 está a um se o controlador recebeu um evento; este bit volta a 0 assim que DADOS for lido.
- **DADOS (0x101)** — só pode ser lido e indica o evento recebido dos periféricos (teclado, rato, etc.)
- **COMANDO (0x102)** — só pode ser escrito (não usado neste exercício)

Assuma que não existe mais nenhum *software* a manipular este controlador e que o seu código executa em modo supervisor. Para efetuar as operações IN e OUT do processador, tem disponíveis as funções C (à semelhança das

usadas nas aulas práticas):

```
unsigned short in(unsigned int portid);  // IN de uma word (2 bytes)
void out(unsigned int portid, unsigned short value); // OUT de uma word (2 bytes)
```

Um determinado sistema, deve ler os eventos que chegam ao controlador e distribuir pelas aplicações que os consomem (p.e. um programa lê o teclado e o gestor gráfico controla a seta do rato no ecrã, etc). Para tal dispomos de 8 filas de eventos onde devem ser colocados cada um dos eventos recebidos no controlador de acordo com o identificador do periférico. Estas oferecem a seguinte interface (API):

```
void addEvent(int queueID, unsiged int event); //acrescentar evento à fila queueID
unsigned int getEvent( int queueID );  // retirar 1º evento da fila queueID
```

a) Usando esta interface, implemente em C a função seguinte que deve ficar num **ciclo infinito** recebendo os eventos e colocando-os na fila respetiva de acordo com o número do periférico. Cada fila é identificada pelo número do periférico que lhe diz respeito (exemplo: eventos do periférico 2 vão para a fila 2). Note que o evento são apenas os 13 bits antes descritos. Não se preocupe como estes eventos serão depois lidos ou usados pelas aplicações.

```
void receiveEvents() {









}
```

b) Admita agora que o controlador foi programado para gerar uma interrupção na chegada de cada evento e que a programação necessária para a chamada da respetiva rotina de tratamento para essas interrupções já foi feita para chamar a função seguinte. Implemente-a por forma a permitir a mesma funcionalidade que na alínea anterior.

```
void IntrHandler() {









}
```