Teste 2 A de Arquitetura de Computadores

15/6/2019 Duração 1h45 *sem consulta*

As perguntas de escolha múltipla têm de ser respondidas na folha de respostas anexa.
Respostas erradas descontam 1/5 da cotação da pergunta ou alínea.

Número: _____ *Nome:* _____

1 (1,5 val) 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.

2 (1,5 val) Considere a arquitectura intel 32bits estudada nas aulas e a seguinte declaração de um vetor:

```
.data
v: .int  10, 11, 12, 13, 14, 15   # v está no endereço de memória 0
```

a) Indique os valores dos registos após a execução da seguinte sequência de instruções:

```
mov $v, %ebx
mov $2, %ecx
mov 4(%ebx, %ecx, 2), %eax
```

A. eax=13, ebx=0, ecx=2 B. eax=14, ebx=10, ecx=2
C. eax=12, ebx=0, ecx=2 D. eax=11, ebx=10, ecx=2 E. nenhuma das anteriores

b) Indique os valores do registo EAX após a execução da seguinte sequência de instruções:

```
pushl v
pushl $9
pushl $18
mov 4(%esp), %eax
```

A. eax=10 B. eax=9 C. eax=18 D. eax=4 E. nenhuma das anteriores

3 (0,75 val) 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.

4 (0,75 val) 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.

5 (0,8 val) 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

6 (4,2 val) 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.

7 (0,7 val) 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.

8 (0,7 val) 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.

9 (0,7 val) 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.

10 (0,7 val) 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.

11 (0,7 val) 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

12 (0,75 val) Considere o seguinte programa em C que guarda num ficheiro o conteúdo de um vetor. Assuma que não há erros.

```
int main() {
   float x[]={ 1.3, 4.5, 24.9, 6 };
   FILE*f = fopen("fich.txt", "w");
   fwrite( x, sizeof(float), 4, f);
   return 0;
}
```

Se abrir com um editor de texto o ficheiro "fich.txt" após a execução completa do programa qual será o seu conteúdo?

A. Os números do vetor, um por linha
B. Os números do vetor, todos na mesma linha separados por espaço
C. Os números do vetor, todos colados
D. Símbolos estranhos e/ou não visíveis pois não contém texto
E. O editor mostra uma página em branco pois o ficheiro não foi fechado

13 (2,7 val) 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

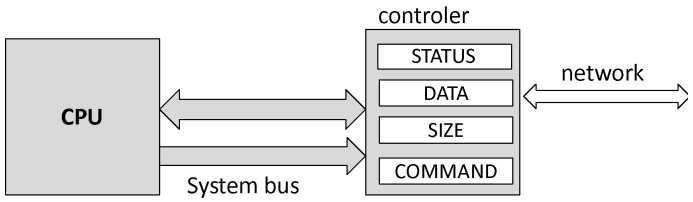
Indique se, conhecendo unicamente o conteúdo do TLB acima indicado, é possível obter o endereço real para cada um dos acessos aos endereços virtuais que se seguem (circule a palavra Não ou Sim). Em caso afirmativo, indique qual o endereço real obtido:

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

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.
E. A página não tem frame atribuído e o CPU vai parar com um *segmentation fault*.

14 (1,5 val) Considere uma arquitetura onde uma placa de rede permite enviar e receber mensagens com até 255 bytes. Esta é controlada por um controlador com uma interface pelos seguintes registos acessíveis por portos de IO.



Sempre que chega uma mensagem tal é assinalado no porto STATUS. Um programa pode obter todos os bytes da mensagem, lendo do porto DATA o número de bytes indicados no porto SIZE. Em mais detalhe:

Todos os registos têm 1byte e são os seguintes, com os endereços de IO (portos) indicados:

- **STATUS (0x90)** - só pode ser lido;
  - o bit 0 está a um se o controlador recebeu uma mensagem; este bit volta a 0 assim que DATA for lido.
- **DATA (0x91)** - pode ser lido e escrito; de cada vez que é lido devolve um novo byte da última mensagem recebida. Se for lido para além da dimensão indicada em SIZE, devolve o valor 0 (zero).
- **SIZE (0x92)** - só pode ser lido; indica quantos bytes tem a última mensagem recebida.
- **COMMAND (0x93)** – 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 char in(unsigned int port);  // IN
void out(unsigned int port, unsigned char value); // OUT
```

Indique das alternativas no fim desta questão qual o código que permite responder a cada uma das seguintes alíneas de acordo com a respetiva descrição:

a) Coloca no vetor `buffer` (variável global) a mensagem recebida ou, se não há nenhuma nova, a próxima mensagem que chegue (ver código abaixo).

A. B. C. D. E.

b) Se a arquitectura suportar um mecanismo de interrupções e este estiver programado para gerar uma interrupção de cada vez que uma nova mensagem chegue, qual será a função mais simples que permite este atendimento (ou tratamento desta interrupção) colocando em `buffer` (variável global) a mensagem que chegou?

A. B. C. D. E.

A.
```
void f1() {
 int s = in(SIZE);
 for (int i=0; i<s; i++)
    buffer[i]=in(DATA);
}
```

B.
```
void f2() {
  int s = in(STATUS);
  while ( (s&1) == 0 )
  ;
  s = in(SIZE);
  for (int i=0; i<s; i++)
    buffer[i]=in(DATA);
}
```

C.
```
void f3() {
  int i=0;
  unsigned char c=in(DATA);
  while (c!=0) {
     buffer[i++]=c;
     c=in(DATA);
  }
}
```

D.
```
void f4() {
 int s = in(SIZE);
 for (int i=0; i<s; i++)
   if ( (in(STATUS)&1)==1 )
      buffer[i]=in(DATA);
}
```

E.
```
void f5() {
  int s=0;
  while ( (s&1) == 0 )
    s=in(STATUS);
  s = in(SIZE);
  for (int i=0; i<s; i++)
    buffer[i]=in(DATA);
}
```

15 (2 val) Uma palavra (ou frase) é um palíndromo se pode ser lida da direita para a esquerda tal como se lê da esquerda para a direita (tal como uma capicua nos números). Exemplos: "ANA", "rir", "salas". A seguinte função recursiva em C testa se um vetor de caracteres contém um palíndromo. Tem como parâmetros esse vetor e a sua dimensão:

```
int isPalind(char* s, int len) {
  if ( len==1 ) return 1;
  if ( s[0]==s[len-1] ) return isPalind( s+1, len-2 );
  else return 0;
}
```

nota: este código só funciona para len ímpar

a) Considerando as seguintes declarações de um vetor e seu tamanho. Indique como, em assembly (Linux/intel 32bits), chamaria a função C anterior para verificar se `str` contém um palíndromo.

```
.data
str:  .ascii "esse"
sz:   .int    4
```

A.
```
pushl str
pushl sz
call isPalind
add $8, %esp
```

B.
```
pushl sz
pushl str
call isPalind
add $8, %esp
```

C.
```
pushl $str
pushl sz
call isPalind
add $8, %esp
```

D.
```
pushl sz
pushl $str
call isPalind
add $8, %esp
```

E.
```
pushl str
pushl sz
call isPalind
add $6, %esp
```

b) Escreva em assembly (Linux/intel 32) uma função recursiva semelhante à `isPalind` e usando as convenções de chamada e retorno usadas nas aulas.

```
isPalind:            # esta é uma possível solução
      push %ebp
      mov %esp, %ebp

      mov 12(%ebp), %eax    # len
      cmp $1, $eax
      je done

      mov 8(%ebp), %edx     # char *s
      dec %eax
      mov (%edx, %eax, 1), %ecl
      cmp (%edx), %ecl
      je cont

      mov $0, %eax
      jmp done

cont: dec %eax
      push %eax
      inc %edx
      push %edx
      call isPalind

done: mov %ebp, %esp
      pop %ebp
      ret
```

Intel x86 (IA32) Assembly Language Cheat Sheet

Suffixes: b=byte (8 bits); w=word (16 bits); l=long (32 bits). Optional if instruction is unambiguous.

Operands: immediate/constant (not as *dest*): $10, $0xff ou $0b01101 (decimal, hex or bin)

32-bit registers: %eax, %ebx, %ecx, %edx, %esi, %edi, %esp, %ebp

16-bit registers: %ax, %bx, %cx, %dx, %si, %di, %sp, %bp

8-bit registers: %al, %ah, %bl, %bh, %cl, %ch, %dl, %dh

direct addr: (2000) or (0x1000+53)

indirect addr: (%eax) or 16(%esp) or 200(%edx, %ecx, 4)

Note that it is not possible for **both** *src* and *dest* to be memory addresses.

Instruction	Effect	Examples
Copying Data		
mov *src,dest*	Copy src to dest	mov $10,%eax movw %ax,(2000)
Arithmetic		
add *src,dest*	dest = dest + src	add $10, %esi
sub *src,dest*	dest = dest – src	sub %eax,%ebx
cmp *src,dest*	Compare using sub (dest is not changed)	cmp $0,%eax
inc *dest*	Increment destination	inc %eax
dec *dest*	Decrement destination	decl (0x1000)
Bitwise and Logic Operations		
and *src,dest*	dest = src & dest	and %ebx, %eax
test *src,dest*	Test bits using and (dest is not changed)	test $0xffff,%eax
or *src,dest*	dest = src | dest	or (0x2000),%eax
xor *src,dest*	dest = src ^ dest	xor $0xffffffff,%ebx
shl *count,dest*	dest = dest << count	shl $2,%eax
shr *count,dest*	dest = dest >> count	shr $4,(%eax)
sar *count,dest*	dest = dest >> count (preserving signal)	sar $4,(%eax)
Jumps		
je/jz *label*	Jump to label if dest == src /result is zero	je endloop
jne/jnz *label*	Jump to label if dest != src /result not zero	jne loopstart
jg *label*	Jump to label if dest > src	jg exit
jge *label*	Jump to label if dest >= src	jge format_disk
jl *label*	Jump to label if dest < src	jl error
jle *label*	Jump to label if dest <= src	jle finish
ja *label*	Jump to label if dest > src (unsigned)	ja exit
jae *label*	Jump to label if dest >= src (unsigned)	jae format_disk
jb *label*	Jump to label if dest < src (unsigned)	jb error
jbe *label*	Jump to label if dest <= src (unsigned)	jbe finish
jz/je *label*	Jump to label if all bits zero	jz looparound
jnz/jne *label*	Jump to label if result not zero	jnz error
jmp *label*	Unconditional jump	jmp exit
Function Calls / Stack		
call *label*	Call (Push eip and Jump)	call format_disk
ret	Return to caller (Pop eip and Jump)	ret
push *src*	Push item to stack	pushl $32
pop *dest*	Pop item from stack	pop %eax

Directives (examples):

.data – data section (global variables)

.int – 32bits space(s) for integer value(s)

.ascii – char sequence

.global *label* -- export *label* symbol/address

.text – text section (code)

.comm *label*, *length* – length bytes space

Functions Linux/32bits:

caller:

- push args (right to left)

- call function

- free stack space used with args

C types:

char 1 byte

short 2 bytes

int, float, long and *pointer* 4 bytes

double 8 bytes

callee (function):

- initialise: push %ebp
  mov %esp, %ebp
  sub $4, %esp #space for local var.
- use ebp based address, e.g.: movl 8(%ebp), %eax
- result at %eax
- finalise: mov %ebp, %esp #free local var.
  pop %ebp
  ret

INSERIR CABULA ASSEMBLY
FAZER FOLHA DE RESPOSTAS

8 (0,7 val) Muitas arquiteturas suportam DMA (Direct Memory Access - acesso direto a memória) nos controladores dos periféricos. Em que consiste esta tecnologia e qual é a vantagem?

A. Permite a transferência de bytes diretamente entre controlador e memória sem necessidade de intervenção do CPU.
B. Permite o acesso direto a memória sem transformação de endereços ou cache, de forma mais eficiente que o normal.
C. Permite o acesso dos controladores diretamente à memória para receber/enviar bytes, mantendo o CPU ocupado com essa transferência.
D. Permite ao CPU e aos periféricos o acesso direto a memória sem passar pela cache o que permite acessos mais eficientes.
E. Permite a transferência de um bloco de bytes com o controlador, ao mesmo tempo que este envia outros dados para o periférico.