Teste 2 A de Arquitetura de Computadores

15/6/2018 Duração 2h *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 — (0,7 val) 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

2 — (0,7 val) Na organização interna de um CPU, como se pode aumentar o número de instruções executadas por unidade de tempo?

A. Aumentando a dimensão da memória central.
B. Aumentando a dimensão do disco.
C. Diminuindo o número de bits nos registos.
D. Diminuindo a frequência de trabalho do CPU.
E. Incluindo um *pipeline* de execução de vários estágios.

3 — (0,7 val) Se tentar compilar e executar o seguinte programa em C, o que acontece?

```
#include <stdio.h>
int main() {
    FILE *fp = stdout;
    int n = 45;
    fprintf(fp, "%d\n", n);
    return 0;
}
```

A. Dá erro de compilação na linha "FILE *fp = stdout;"
B. O programa compila sem problema e ao executar apresenta "45" no ecrã
C. O programa compila sem problema, mas ao executar não aparece nada ecrã
D. O programa é abortado com um *segmentation fault* devido ao uso da variável "n"
E. O programa compila sem problema e ao executar apresenta "lixo" no ecrã

4 — (0,7 val) Qual o efeito da função "`fflush(FILE *fp)`", quando usada num programa em C?

A. Esta função não faz parte da biblioteca do C.
B. Limpar (ignorando) o conteúdo do buffer do canal (stream) **`fp`**.
C. Enviar o conteúdo de um buffer (**`fp`**) para o ecrã.
D. Enviar o conteúdo do buffer do canal (stream) **`fp`** para o respetivo ficheiro ou ecrã.
E. Ler o conteúdo do respetivo ficheiro (**`fp`**) para o buffer associado ao canal (stream).

5 — (1,4 val) Numa arquitetura com suporte para memória virtual paginada a MMU usa uma TLB para acelerar a conversão de páginas virtuais para *frames*. Na execução de determinado programa, em 1000 acessos a memória 900 foram resolvidos com recurso à TLB e 100 obrigaram a consultar a tabela de páginas.

a) Qual a taxa de sucesso da TLB (*hit ratio*)?

A. 80% B. 90% C. 9% D. 10% E. Nenhuma das anteriores

b) Se o tempo de acesso a memória for de 10ns, indique o tempo médio de acesso a memória para esses 1000 acessos (assuma o tempo de resolução do endereço via TLB de 0ns).

A. 10ns B. 11ns C. 19ns D. 19,1ns E. Nenhuma das anteriores

6 — (0,7 val) Quando é feita a leitura de um sector (p.e. 1Kbytes) de um disco magnético, o tempo dessa operação é semelhante ao da cópia desse mesmo número de bytes entre duas zonas de memória? Justifique.

A. Sim, pois essa operação envolve aproximadamente o mesmo número de instruções, variando no uso de IN ou OUT em vez de MOV.

B. Não, pois o disco internamente efetua operações de *seek* para encontrar o sector, as quais são muito demoradas comparativamente aos acessos a memória.

C. Sim, no caso dos disco de >10000r.p.m, pois o disco internamente usa operações de leitura que só dependem dessa velocidade de rotação.

D. Não, pois é necessário programar o controlador de DMA e essa operação limita o tempo da leitura a poucos bytes por segundo.

E. Sim, porque a cópia dos bytes do disco para memória será também feita com operações MOV entre o controlador dos disco e a memória, demorando aproximadamente o mesmo tempo.

7 — (0,7 val) Quando um controlador de um periférico fica disponível para ser usado pelo *software (CPU)*, este envia um sinal ao processador de que tipo?

A. Sinal de interrupção (*interrupt*)
B. Sinal de paragem (*halt*)
C. Sinal de processamento (*handler*)
D. Sinal de execução (*execution*)
D. Sinal de exceção (*exception*)

8 — (0,7 val) Que instruções apenas podem ser executadas em modo supervisor?

A. As privilegiadas, para uso exclusivo pelo Sistema de Operação.
B. As aritméticas de vírgula flutuante, para garantir a precisão dos resultados.
C. As call e ret, para garantir a chamada e reposição da pilha de execução (stack).
D. As normais, para uso de todos os programas do utilizador.
E. As de acesso a memória, a quando da programação do DMA (Direct Memory Access).

9 — (0,7 val) Uma rotina de tratamento de interrupções deve terminar com uma instrução:

A. IRET - retorna à instrução interrompida e repõe as FLAGS
B. IRET - retorna à rotina de tratamento da interrupção
C. INT - retorna à instrução interrompida e repõe as FLAGS
D. INT - retorna à rotina de tratamento da interrupção
E. INT - retorna à rotina de tratamento da interrupção e repõe as FLAGS

10 — (0,7 val) Quando o CPU está a executar um programa e chega uma interrupção de um controlador, o que é que acontece?

A. Continua a executar instruções do programa até este executar INT para chamar a interrupção
B. É executado um salto para o endereço presente no registo %eax
C. É executado um salto para o endereço presente na respetiva posição de um vetor de interrupções
D. O programa é terminado para atender a interrupção
E. É ignorada até ao final da execução da subrotina corrente (até ser executada uma instrução RET).

11 — (0,7 val) Considere um computador, a correr um sistema operativo com suporte para memória virtual, com 2GB (2^{31} bytes) de memória física (real) e espaço de endereçamento virtual de 32 bits. Suponha que pretende executar o programa P nesse computador. Assinale a afirmação verdadeira:

A. O tamanho da memória virtual de P pode exceder os 2GB
B. Os endereços de memória usados por P só podem variar entre 0 e $2^{31} - 1$
C. A soma dos tamanhos dos dados e do código de P não pode exceder os 2GB
D. São precisos 32 bits para endereçar a memória física do computador
E. As operações aritméticas e lógicas têm de usar operandos de 32 bits

12 — (0,7 val) As caches organizadas por mapa direto não são normalmente utilizadas porque:

A. O número de blocos por cada linha é no máximo 4

B. Descobrir se um dado bloco está ou não na cache é uma operação demorada

C. Podem introduzir muitas falhas (*misses*) por conflito mesmo com linhas livres

D. A dimensão de cada linha tem de ser igual a 32 ou 64 bytes

E. Têm pouca capacidade, pois cada linha necessita de muito hardware para efetuar as pesquisas

13 — (0,7 val) Considere todos os passos de tradução de um programa escrito numa linguagem de alto nível (como C) até à geração de um executável numa arquitetura *hardware* particular. Estes podem envolver várias ferramentas como compilador, assembler, linker, etc. Diga qual das seguintes afirmações é verdadeira.

A. A função de um assembler é traduzir código C dando origem diretamente a um executável.

B. O linker (ou ligador) é a primeira etapa da tradução, tendo como função ligar o programa C com o código disponibilizado noutros módulos (e.g. em bibliotecas).

C. A função de um compilador é traduzir código C dando origem diretamente a um executável.

D. A função de um linker (ligador) é ligar o código máquina de diferentes módulos binários, tentando gerar um executável completo.

E. Os programas envolvidos na tradução são invocados pela seguinte ordem: compilador, linker e assembler.

14 — (0,7 val) — Considere a seguinte sequência de instruções:

```
.data
vec   .int 2, 3, 4
.text
_start: movl $vec, %ebx
        movl ???, %ecx
        movl (%ebx, %ecx, ??? ), %eax
```

Selecione a opção para os dois casos "???", que permite carregar o registo %eax com o valor 3 de vec.

A. `movl $1, %ecx  / movl(%ebx, %ecx, 1), %eax`

B. `movl $1, %ecx  / movl(%ebx, %ecx, 2), %eax`

C. `movl $2, %ecx  / movl(%ebx, %ecx, 1), %eax`

D. `movl $1, %ecx  / movl(%ebx, %ecx, 4), %eax`

E. `movl $4, %ecx  / movl(%ebx, %ecx, 2), %eax`

15 — (0,7 val) Em que situação se usa DMA (*Direct Memory Access*) para melhorar a taxa de transferência de dados?

A. Entre um controlador e o respetivo periférico

B. Entre a memória e controladores de periféricos orientados ao byte

C. Entre a memória e controladores de periféricos orientados ao bloco

D. Entre a memória e o CPU

E. Entre os vários níveis da hierarquia de caches

16 — (0,7 val) Quando a MMU deteta uma falta de página (*Page Fault*):

A. Lança uma interrupção para que o Sistema de Operação se encarregue da ocorrência.

B. Lança uma interrupção "Segmentation Fault" para o processo em execução.

C. Lança uma interrupção "Page Fault" que o processo em execução deve resolver.

D. Lança uma interrupção para que a tabela de páginas se encarregue da ocorrência.

E. Carrega automaticamente a página em falta para a RAM (escolhendo uma vítima para ser removida da RAM, caso necessário).

17 — (1,6 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 4 MBytes.

a) Admitindo que cada entrada da tabela de páginas ocupa 4 bytes, a tabela de páginas de um processo pode ocupar até:

A. 2^{13} bytes B. 2^{15} bytes C. 2^{16} bytes D. 2^{22} bytes E. 2^{24} bytes

b) Na transformação de endereços virtuais em endereços físicos, indique como a MMU divide e interpreta os bits do endereço.

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 23 e o número de bits necessário para o deslocamento dentro da página é 13.
C. O número de bits necessário para identificar o número da página virtual é 14 e o número de bits necessário para o deslocamento dentro da página é 22.
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 24 e o número de bits necessário para o deslocamento dentro da página é 12.

18 — (2 val) Considere uma arquitetura com um nível de cache, associativa por grupos de 4 linhas cada (*4 way set associative*). Os endereços são de 16 bits, e blocos (linhas) de 16 bytes cada (2^4).

a) Assumindo que a cache tem 8 grupos, como é a interpretação do endereço para procurar cada endereço na cache?

A. tag: 10 bits mais significativos; nº grupo: 3 bits; deslocamento no bloco: 3 bits menos significativos
B. tag: 9 bits mais significativos; nº grupo: 3 bits; deslocamento no bloco: 4 bits menos significativos
C. tag: 8 bits mais significativos; nº grupo: 4 bits; deslocamento no bloco: 4 bits menos significativos
D. tag: 9 bits mais significativos; nº grupo: 2 bits; deslocamento no bloco: 5 bits menos significativos
E. tag: 10 bits mais significativos; nº grupo: 4 bits; deslocamento no bloco: 2 bits menos significativos

b) Considere agora que a cache tem apenas dois grupos. Para a sequência seguinte de acessos a memória e para o conteúdo da cache na figura seguinte, indique para cada endereço se há um cache hit ou miss. Para os casos de hit assinale na figura, com uma seta, em que linha se encontram os bytes acedidos, como no exemplo.

Exemplo**: acesso 0x0000 --- hit/miss: **HIT

acesso: 0x0013 — hit/miss: _____hit_____

acesso: 0x0023 — hit/miss: _____hit_____

acesso: 0x0101 — hit/miss: _____miss_____

acesso: 0x0038 — hit/miss: _____miss_____

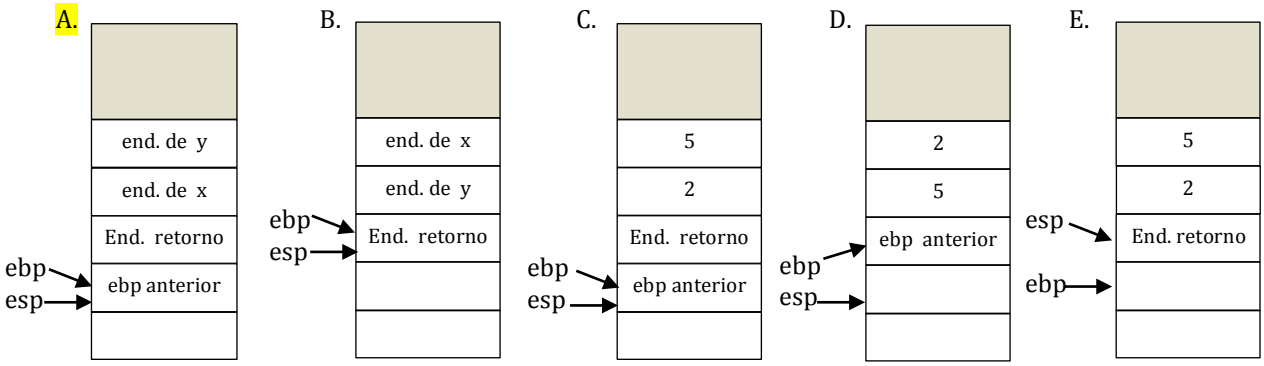
grp	valid	chave	Blocos (16 bytes cada)
0	1	0	…
	1	1	…
	1	2	…
	0	10	…
1	1	3	…
	1	0	…
	0	1	…
	0	0	…

19 — (2,5 val) Considere o seguinte código C que chama a função **troca** para trocar os valores nas variáveis x e y; e a respetiva implementação desta função:

```
…
int x=2; y=5;
troca(&x, &y);
…  // x fica com 5 e y com 2
```

```
void troca(int* a, int* b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

a) Indique, na folha de respostas, qual das seguintes configurações da pilha (*stack*) está correta para a chamada indicada, a quando da execução da primeira instrução da função troca (a pilha cresce de cima para baixo).



b) Implemente em *assembly* a chamada da função de acordo com o código anterior e seguindo as convenções de chamada de funções da linguagem C usadas nas aulas.

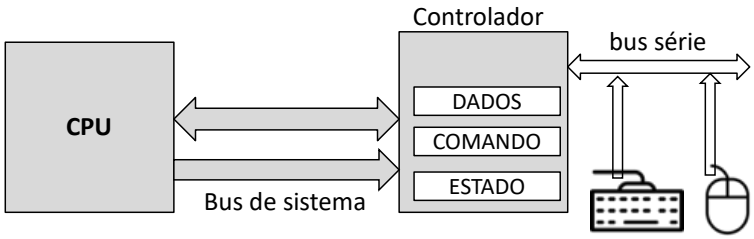
```
.data
x: .int 2
y: .int 5
.text
…
        pushl   $y
        pushl   $x
        call    troca
        addl    $8, %esp
```

c) Implemente a função `troca` em *assembly* IA-32, com a mesma funcionalidade da função C com o mesmo nome, seguindo as convenções de chamada de funções da linguagem C usadas nas aulas:

```
troca:

        push    %ebp
        mov     %esp, %ebp
        mov     8(%ebp), %eax       # get &x
        mov     (%eax), %ecx        # get x
        mov     12(%ebp), %eax      # get &y
        mov     (%eax), %edx        # get y
        mov     %ecx, (%eax)        # store x in y
        mov     8(%ebp), %eax       # get &x
        mov     %edx, (%eax)        # store y in x
        pop     %ebp
        ret
```

20 — (2 val) 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() {
                                                    // assumindo:
                                                    #define DADOS 0x101
                                                    #define ESTADO 0x100
        while(1) {
                unsigned int e = receive();
                int dev = e & 7;
                unsigned ev = e>>3;
                addEvent( dev, ev );    // acrescenta na fila respetiva
        }
}


unsigned int receive() {
        while( in(ESTADO) & 1 == 0 )  // espera por novo evento
        ;
        return in( DADOS );


}
```

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() {
        unsigned e = in(DADOS);  // se IntrHandler esta' a correr -> ha' novo evento
        addEvent(  e&7,  e>>3 ); // acrescenta na fila respetiva









}
```

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