Exame de recurso de Arquitetura de Computadores

21/06/2016 **Duração 3h** *sem consulta*

Número: _____ *Nome:* _____

1- 1 val Considere uma representação de números inteiros num computador usando 8 bits.

a) Quais são o maior e menor valores sem sinal que se podem representar? Responda em base 2 e em base 10 (neste caso pode apresentar a expressão).

b) Quais são o maior e menor valores que se podem representar, assumindo uma representação com sinal (complemento para 2)? Responda em base 2 e em base 10 (neste caso pode apresentar a expressão).

2- 1 val Considere um CPU que tem uma unidade aritmética e lógica em que as entradas e a saída têm 6 bits. Os inteiros negativos são representados em complemento para 2.

a) Diga qual é o resultado da soma de -16 com -8. Apresente o resultado em binário.

b) Indique, justificando com base no resultado anterior, os valores das flags CF, OF, ZF e SF.

3- 1 val Considere a norma IEEE 754 onde a representação de números reais em precisão simples (32 bits) usa a seguinte interpretação:

- Bit 31: sinal – 0 se maior ou igual a zero, 1 se menor do que zero
- Bits 30 a 23: expoente somado de 127
- Bits 22 a 0: parte fracionária da mantissa (o valor efetivo da mantissa é 1.XXX...XXX)

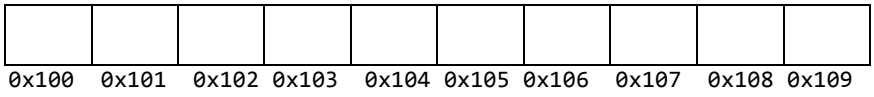
Apresente, em base 10, o valor representado na seguinte sequência de bits:

11000000 11001000 00000000 00000000

4- 1val Indique o conteúdo da memória após a execução das duas instruções seguintes. Admita uma arquitetura ***big-endian*** de 32bits.

```
mov $0x105, %ebx
mov $0x102,(%ebx)
```

bytes de memória e seus endereços:



5- 1 val Ordene por tempo de acesso cada uma das operações abaixo indicadas (atente às condições expressas). Numere-os de 1 (mais rápido) a 4 (mais lento):

inteiro num bloco em disco;
inteiro em memória usando endereçamento direto;
inteiro num endereço de memória presente na cache;
inteiro num registo do CPU.

6- 1 val Considere a arquitetura de Von Neumann, explicada nas aulas. Indique que componentes (CPU, memória, periféricos e bus) são utilizados e qual o seu papel, na execução da instrução **`mov %eax, (100)`**

7- 1 val Considere a seguinte sequência de instruções

```
      movl  $10, %eax
      movl $3, %ebx
      cmp  %ebx, %eax
      ja   L1
      jmp  L2
L1:   movl %eax, %ebx
L2:   inc %ebx
```

Escreva na caixa ao lado o conteúdo dos registos e das *flags*, após a execução destas instruções.

eax
ebx
ZF
OF
SF

8- 2 val Pretende-se que construa uma função chamada `paraZero` que tem a assinatura

```
int paraZero( int ve[], int dim )
```

- Recebe como parâmetros de entrada o endereço inicial de um vetor de inteiros, *ve,* e a sua dimensão, *dim.*
- A função passa todos os elementos do vetor *ve* que não são zero para o valor zero; também devolve quantos inteiros nesse vetor eram diferentes de zero e foram passados a zero.

O exemplo seguinte ilustra o uso da função:

```
int x[ ]= { 7, -15, 0, 7, -2, 0 }; //declaração e inicialização do array x
...
int b = paraZero( x, 6 );   // a variável b fica com o valor 4
                            // x fica com todos os elementos a zero
```

a) Implemente em C a função "`paraZero`" descrita acima.

```
int paraZero ( int *ve, int dim) {









}
```

b) Assuma que a função "paraZero" está implementada. Escreva o código *assembly* correspondente à chamada da função quando são passados como argumentos o array ***x*** e o valor ***6***. Desenhe também o conteúdo da pilha imediatamente após a execução da chamada (antes da execução da primeira instrução da função) e a posição apontada pelo registo %esp.

```
.data
   x:  .int   7, -15, 0, 7, -2, 0
.text
     ...
```

pilha (cada linha são 4 bytes):

9- 2 val Pretende-se que escreva em *assembly* Intel 32 bits usando as mnemónicas e convenções do *as* (**as**sembler), o código equivalente à seguinte função de código C:

```
int contaX( char ss[] ) { // ss é uma string
    int i=0, cont=0;

    while ( ss[i] != '\0' ) {
        if( ss[i] == 'X') cont = cont+1;   // contar o número de caracteres 'X'
        i = i+1;
    }
    return cont;
}
```

Deve seguir as convenções de passagem de argumentos e retorno de resultados usadas nas aulas.

```
.text
...

   # código da sua solução:
contaX:
```

10- 1 val
O Pentium tem uma instrução máquina chamada *return from interrupt*, cuja mnemónica é ***iret***. Esta instrução é usada para terminar rotinas de atendimento de interrupções. Explique, justificando, as acções desencadeadas nos registos do CPU na execução desta instrução que permitem retomar a computação antes interrompida.

11- 1 val
Numa determinada arquitectura, o CPU tira partido do *pipelining* das várias fases de execução das instruções. Descreva os potenciais problemas para o *pipelining*, que podem surgir na execução de uma sequência de instruções envolvendo registos

12- 2 val
Admita uma arquitectura de um computador com as seguintes características: endereçamento de 20 bits, um nível de cache de 64Kbytes composta por uma cache associativa pura com linhas de 64 bytes, e com escritas diferidas (*write-back*).

a) Indique que bits do endereço são interpretados como chave (tag) e quais representam o deslocamento na linha (ou bloco).

b) Indique que outros bits estão associados a cada linha para a gestão da cache incluindo, por exemplo, a detecção de *hits* e *misses*. Justifique.

c) Existe alguma interrupção associada ao *cache miss* para que o software trate desse situação? Justifique.

d) Para uma sequência de 10000 acessos à memória, 100 não puderam ser logo satisfeitos pela cache e levaram a acessos à memória central. Calcule o *hit ratio* nesta sequência e o tempo médio destes acessos, considerando que o tempo de acesso à cache é de 1ns e à memória central de 10ns.

13- 2,5 val

Uma arquitectura com endereços virtuais de 32 bits suporta páginas com dimensão 64 KBytes (2^{16}) e permite paginação a pedido.

a) Qual o número máximo de entradas que pode ter a tabela de páginas de um processo? Justifique.

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

TLB:

página virtual	página física
3	0x00
1	0xAF
2	0xAD
6	0xEA

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. Em caso afirmativo, indique qual o endereço real obtido; caso contrário, explique porque não pode obter o endereço real.

`0x00020000` →

`0x00061000` →

`0x00058010` →

`0x00000020` →

c) Nos casos em que não é possível obter logo o endereço real, indique que acções o *hardware* toma para tentar obter esse endereço e em que condições produz um *page-fault*.

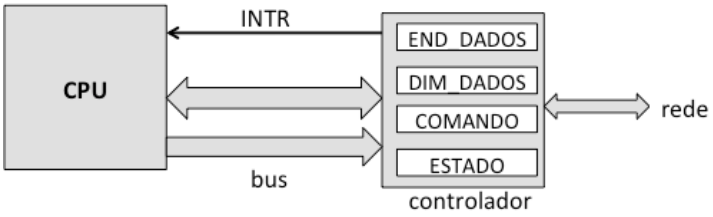
14- 2,5 val

As várias alíneas desta pergunta supõem um ambiente, do ponto de vista do software, semelhante ao usado nas aulas práticas e no mini-projecto, incluindo o TurboC com as seguintes funções:

```
unsigned int inport(unsigned int portid)
void outport(unsigned int portid, unsigned int value)
void enable(void)
void disable(void)
setvect( int intrnum, void interrupt(*isr)() )
```

Do ponto de vista de *hardware*, considere o controlador de uma interface de comunicação com outros computadores (tipo rede), capaz de enviar e receber blocos até 1500 bytes. Considere apenas a operação de envio de blocos de dados por esta interface.

Os dados são copiados usando DMA, onde o controlador acede diretamente à memória para ler os bytes a enviar. Assuma que este é o único controlador ligado à linha de interrupção do CPU para assinalar o fim de uma transmissão. O CPU usa interrupções vectorizadas, sendo para este periférico usado o número 5 para identificar estas interrupções.



Os registos do controlador são os seguintes:

- **endereço 0x30 END_DADOS**: registo com 16 bits que só pode ser escrito. O valor neste registo indica o endereço de memória do bloco a enviar.
- **endereço 0x31 DIM_DADOS**: registo só de escrita; Número de bytes a enviar.
- **endereço 0x32 COMANDO**: registo só de escrita com 2 bits relevantes; Quando o CPU escreve neste registo o valor 0x01, o controlador desencadeia o envio do bloco de dimensão dada por DIM_DADOS, a partir do endereço de memória indicado em END_DADOS, sem usar interrupções. Quando é escrito o valor 0x02 é também desencadeado o envio de um bloco de bytes mas, quando termina, o controlador envia uma interrupção ao CPU. Note que, se o controlador já estiver ocupado a enviar um bloco, estes comandos são ignorados.
- **endereço 0x33 ESTADO**: registo só de leitura com o bit menos significativo relevante. Depois de ser pedido o envio de um bloco, o controlador colocará o bit 0 a 1 indicando que está ocupado; quando termina a transmissão de todos os bytes, este bit passa a zero.

a) Escreva, usando o Turbo C, uma rotina com o protótipo

```
void  EnviaBloco( unsigned char *block, int dim )
```

Esta pede a transmissão do bloco no endereço indicado e dimensão 'dim' bytes. Esta função tem que garantir que o pedido de transmissão não é ignorado e é corretamente efectuado.

b) Considere que pretende enviar uma sequencia de mais de 1500 bytes por esta interface. Para tal terá de o fazer em sub-blocos de 1500 bytes cada. Programe uma função que, usando a função anterior, envia os dados pretendidos:

```
void EnviaTodosOsDados( unsigned char *data, int dim )
// dim pode ser > que 1500
```

c) Descreva as desvantagens duma solução usando espera ativa face à utilização do mecanismo de interrupções. Indique ainda como as interrupções poderiam ser usadas na situação da alínea anterior, indicando o que deveria ser feito para iniciar o processo de transmissão do primeiro bloco de 1500 bytes e o que deveria ser feito pela rotina de atendimento a cada interrupção gerada por este controlador, para enviar todos os dados.

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

Folha para rascunho.
Pode destacar depois do início do teste.