Arquitetura de Computadores
Exame de recurso - 03/07/2015 - Duração 3h00m

- *Teste sem consulta e sem esclarecimento de dúvidas*
- *A detecção de fraude conduz à reprovação de todos os envolvidos*
- *Todas as perguntas têm a cotação de 1 valor*

Nome ***Nº***

Questão 1 Considere uma representação de inteiros sem sinal usando 16 bits. Qual é o maior valor que se pode representar? Deve apresentar o resultado como uma expressão numérica que envolva uma potência de 2.

Questão 2 Considere uma representação de inteiros com sinal em complemento para 2 em que estão disponíveis 16 bits. Como é representado -17 (base 10) ?

Questão 3 Considere um CPU que tem uma unidade aritmética e lógica em que as duas entradas têm 12 bits e a saída tem 12 bits. Qual é o resultado (em base 2) da soma de $1000\ 1110\ 0000_2$ com $1100\ 0011\ 0001_2$? Diga, justificando, se o resultado é válido, supondo que os valores operados são números inteiros com sinal , usando uma representação em complemento para 2.

Questão 4 Considere a norma para representação de números reais em precisão simples IEEE 754 em que um número real é representado em 32 bits (bit 31 é o mais significativo e bit 0 é o menos significativo) com a seguinte interpretação:

- Bit 31: sinal – 0 maior ou igual a zero, 1 menor do que zero
- Bits 30 a 23: expoente. Sendo X o valor representado, o valor do expoente é X-127
- Bits 22 a 0: mantissa. Sendo a configuração dos bits da mantissa xxxxx...xxx_2, o valor efectivo da mantissa é 1.xxxx...xx_2

Apresente, justificando, a representação do número real *-0.375* (em base 10)

Questão 5- Considere a seguinte sequência de instruções

```
        movl  $3, %eax
        cmpl  $7, %eax
        jle  l1
        movl $-5, %ebx
        jmp l2
l1:     movl $10, %ebx
l2:     movl %ebx, %ecx
```

Escreva no espaço ao lado o conteúdo dos registos *eax* e *ebx* após a execução da última instrução.

eax

ebx

Questão 6 Considere a seguinte sequência de instruções

```
        movl  $7, %eax
        movl  $3, %ebx
        pushl %eax
        pushl %ebx
       popl  %eax
       popl  %ecx
```

Escreva no espaço ao lado o conteúdo dos registos *eax* e *ecx* após a execução da última instrução.

eax

ecx

Questão 7 Considere a seguinte sequência de instruções

```
        movl  $0, %eax
        xorl  %ebx, %ebx
l1:     incl %eax
        addl  $eax,$ebx
        cmpl  $5, %eax
        jz l1
        movl $5, %ecx
```

Escreva no espaço ao lado o conteúdo dos registos *eax* e *ebx* após a execução da última instrução.

eax

ebx

Questão 8 A instrução **leal especificação de endereço, registo** carrega no registo que é o 2º operando da instrução o endereço que é o 1º operando da instrução. Considere a seguinte sequência de instruções

```
movl  $1000, %eax
movl  $3, %ebx
leal $1000(%eax, %ebx, $2), %ecx
```

Escreva no espaço ao lado o conteúdo do registo ecx após a execução da última instrução.

ecx

Questão 9 Pretende-se que construa em C uma função chamada *troca* que tem a assinatura

```
void troca( int *v1, int *v2)
```

- Recebe como parâmetros de entrada os endereços de 2 inteiros
- Durante a execução os conteúdos dos dois inteiros apontados por v1 e v2 são trocados

Ver a seguir um exemplo do uso da função:

```
int x = 3;
int y = 5; // declaração e inicialização de x e y
...
troca( &x, &y);   // após a execução da função o conteúdo de x é 5 e o de y é 3
```

Implemente em C a função `troca` descrita acima.

```
void troca( int *v1, int * v2) {









}
```

Questão 10 Assuma que a função `troca` anterior está implementada. Complete o código *assembly* apresentado a seguir correspondente à invocação da função `troca( &x, &y )`:

```
.data
   x:  .int   3
   y:  .int   5

.text
     ...









     call troca
```

Questão 11 Escreva, em *assembly* do Pentium (versão 32 bits) e as mnemónicas e convenções do *gas* (**g**nu **as**sembler) o código da função *troca*.

```
.text
      ...

 troca:

       ret
```

Questão 12 Considere a um sistema operativo que suporta multi-programação, isto é, N programas carregados em RAM e em que existe 1 processo para executar o código associado a cada um dos N programas. Explique, como é que o sistema operativo garante que os N processos executam de forma independente, isto é, produzem os mesmos resultados sem serem afetados pelo que os outros programas carregados em RAM fazem.

Questão 13 Considere que um dispositivo de saída XYZ transfere 8 bits de cada vez. Este dispositivo é visto pelo CPU através dos dois seguintes endereços do espaço de endereçamento de entradas / saídas que permitem ler e escrever 8 bits de cada vez:

- Endereço 0x200 DATA_OUT (write only): quando o CPU escreve um valor neste endereço ele é transferido para o periférico
- Endereço 0x201 STATUS (read only): o único bit relevante é o bit 0 que quando está a 0 indicado que o periférico está pronto a transmitir mais um byte e que quando está a 1 indica que o periférico está ocupado. Quando o CPU escreve um byte no endereço DATA_OUT este bit fica a 1.

Escreva o código que permite enviar bytes através deste dispositivo:

```
void out_byte_xyz( unsigned char c){




}
```

Questão 14 Suponha que se acrescenta agora ao controlador XYZ a capacidade de fazer interrupções. Para esse efeito existe um registo COMANDO com o endereço 0x202 cujo único papel é ligar / desligar as interrupções geradas. Quando o CPU escreve 0xFF no registo COMANDO o periférico interrompe o CPU sempre que é possível enviar mais um byte; após o CPU escreve 0x00 no mesmo registo este deixa de fazer interrupções. Suponha que tem um *buffer* circular com as operações habituais:

```
void initBuffer();      // inicializa o buffer
int isBufferFull();     // retorna 1 se o buffer estiver cheio e 0 se tal não acontecer
int isBufferEmpty();    // retorna 1 se o buffer estiver vazio e 0 se tal não acontecer
void putBuffer(unsigned char c);// coloca c na 1ª posição livre do buffer; assume que
                                // há casas livres
unsigned char getBuffer();      // retorna o byte há mais tempo no buffer; assume que
                               // há casas ocupadas
```

Apresente o código da rotina de tratamento da interrupção que é executada quando o dispositivo está pronto a transmitir mais um byte.

Questão 15 Quando uma rotina de tratamento de interrupções como a da pergunta anterior está a ser executada, o CPU pode aceitar interrupções ou não? Diga especificamente quando é que a possibilidade de aceitar interrupções existe e não existe. Inclua na sua resposta a existência de outros periféricos que também podem enviar interrupções ao CPU.

Questão 16 Um dado CPU emite endereços com 40 bits. Entre o CPU e a memória central (RAM) existe uma cache associativa por grupos com as seguintes características:

- Uma linha tem 256 (2^8) bytes
- Cada conjunto ou grupo tem 16(2^4) linhas
- Há 1024 (2^{10}) conjuntos

Diga que que forma é interpretado o endereço emitido pelo CPU indicando como é que o endereço é dividido e para que é que servem os vários campos em é dividido.

Questão 17 Um sistema em que os endereços virtuais têm 40 bits, usa uma MMU que suporta páginas com uma dimensão de 8 KBytes (2^{13}). Diga, justificando, como é interpretado um endereço virtual, isto é como é que é obtido o endereço físico a partir do endereço virtual.

Questão 18 Considere o seguinte fragmento de programa em C

```
#define NSTEPS 1024*1024
#define STRIDE 8
int i;
int *vec = malloc( NSTEPS*STRIDE*sizeof(int));
for( i = 0; i < NSTEPS; i++)
       vec[i*STRIDE] = 5;
```

No CPU que está a executar o programa:

- As linhas da cache têm 128 bytes
- As páginas têm 1024 bytes

Admita que o endereço de `vec[0]` é um múltiplo de 128 e que o programa não fez acesso anteriores ao vector `vec`. Diga, justificando em ambos os casos, qual é o *hit rate* na cache e no TLB (*translation lookaside buffer*) durante a execução do ciclo *for*.

Questão 19 Considere que se pretende construir um ficheiro executável *p* a partir do código fonte dos ficheiros em C *p1.c* e *p2.c*. Diga quais são as ferramentas envolvidas na obtenção de *p* e faça um resumo da forma como elas funcionam para este caso concreto

---

Questão 20 Os endereços virtuais gerados pelo programa contido no ficheiro executável *p* referido na pergunta anterior vão desde 0 até um dado valor máximo. No entanto, o programa *p* pode, em ocasiões diferentes, ser carregado em diferentes posições na memória física.
Suponha que este programa é carregado numa RAM que é gerida por uma MMU (Memory Management Unit) que contém um registo base e um registo limite. Explique como é que se garante que o programa executa corretamente mesmo quando é carregado a partir da posição de memória 0x10000.

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

Constant (decimal or hex): $10 or $0xff
Fixed address: (2000) or (0x1000+53)
Register: %eax %bl
Dynamic address: (%eax) or 16(%esp)
or 200(%edx, %ecx, 4)

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

Instruction	Effect	Examples
Copying Data		
mov *src,dest*	Copy src to dest	mov $10,%eax movw %eax,(2000)
Arithmetic		
add *src,dest*	dest = dest + src	add $10, %esi
sub *src,dest*	dest = dest – src	sub %eax,%ebx
mul *reg*	edx:eax = eax * reg	mul %esi
div *reg*	eax = edx:eax / reg edx = edx:eax **mod** reg	div %edi
inc *dest*	Increment destination	inc %eax
dec *dest*	Decrement destination	dec (0x1000)
Function Calls		
call *label*	Push eip, transfer control	call format_disk
ret	Pop eip and return	ret
push *item*	Push item (constant or register) to stack	pushl $32 push %eax
pop *[reg]*	Pop item from stack; optionally store to register	pop %eax popl
Bitwise Operations		
and *src,dest*	dest = src & dest	and %ebx, %eax
or *src,dest*	dest = src | dest	orl (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)
Conditionals and Jumps		
cmp *arg1,arg2*	Compare arg1 to arg2; must immediately precede any of the conditional jump instructions	cmp $0,%eax
je *label*	Jump to label if arg1 == arg2	je endloop
jne *label*	Jump to label if arg1 != arg2	jne loopstart
jg *label*	Jump to label if arg2 > arg1	jg exit
jge *label*	Jump to label if arg2 >= arg1	jge format_disk
jl *label*	Jump to label if arg2 < arg1	jl error
jle *label*	Jump to label if arg2 <= arg1	jle finish
jmp *label*	Unconditional relative jump	jmp exit

Diretivas (exemplos):

.data – inicio da zona de dados/variávais globais
.int – reserva de memória para variáveis de 32bits
.byte – reserva de memória para bytes (8bits)
.text – inicio da zona de código
.comm *label*, *length* – reserva de length bytes
.ascii – reserva de memória para sequências de chars

Convenções de chamada de funções:

Invocador:
- empilha parâmetros da direita para a esquerda
- call função
- recupera espaço ocupado pelos parâmetros antes empilhados

Dimensões dos tipos C em ia32:
char 1 byte
short 2 bytes
int, float, long e *pointer* 4 bytes
double 8 bytes

Invocado (função):
- inicia ebp: push %ebp
  mov %esp, %ebp
  sub 4, %esp #se necessário var. local
- usa ebp para endereçar os argumentos, p.e. 4(%ebp)
- resultado fica em %eax (se inteiro ou endereço)
- acaba com: mov %ebp, %esp #liberta var. local
  pop %ebp
  ret