Teste 1A de Arquitetura de Computadores

16/04/2016 Duração 2h *sem consulta*

Número: _____ *Nome:* _____

1- 1,5 val Considere uma representação de inteiros com sinal (complemento para 2) usando 6 bits.

a) Como é codificado o valor 17 ?

010001

b) Como é codificado -5 ?

111011

c) Qual é o maior valor que se pode representar (responda em base 10) ?

$2^5 - 1 = 31$

d) Qual é o menor valor que se pode representar (responda em base 10) ?

$-2^5 = -32$

2- 1,5 val Considere um CPU que tem uma unidade aritmética e lógica em que as entradas e a saída têm 8 bits. Os inteiros negativos são representados em complemento para 2.

a) Diga qual é o resultado (em base 2) da soma de 01111110_2 com 00010010_2

10010000

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

CF=0 não há carry, o resultado cabe nos 8 bits
OF=1 há overflow, o resultado não cabe nos 7 bits + 1 de sinal (soma de dois positivos deu negativo)
ZF=0 o resultado não deu zero
SF=1 o bit de sinal é 1

c) O que significa a configuração de flags da alínea b) relativamente à correção do resultado?

O resultado está correto se os números somados forem do tipo inteiro sem sinal;
O resultado está errado se os números somados forem do tipo inteiro com sinal

3- 1 val Assuma que não dispõe de instruções para comparar valores inteiros e verificar se um número é positivo ou negativo. No entanto sabe que na representação de números com sinal o bit mais significativo representa o sinal (1=negativo; 0=positivo). Escreva no código C seguinte a expressão que lhe permite avaliar se um número (int de 4 bytes) é positivo ou negativo.

```
int numero = ???;

if (_____( numero & 0x80000000 ) != 0____) printf("numero negativo");
else printf( "numero positivo");
```

4- 1,5 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 a representação do valor decimal **5.25** nesta representação.

0 10000001 01010000000000000000000

5- 1.5 val Considere o código Java seguinte:

```
float f = 1/10;
System.out.println( f );
System.out.printf( "%.10f\n", 1.0/10.0 );  //escrever com 10 casas decimais
System.out.println( 50000 * 50000 );
```

Ao executar o programa contendo este código no ecrã apareceram os resultados:

```
0.0
0.10000000015
-1794967296
```

Estes estão matematicamente errados. Explique cada um destes resultados à luz dos seus conhecimentos em arquitetura de computadores.

1/10 é uma divisão de inteiros cujo resultado é 0, que só depois é atribuído a f, convertido para float;

com muitas casas decimais vemos que a representação de 0.1 como um float não é exata, mas sim uma aproximação. Nem é possível representar 0.1 em binário com um número finito de casas decimais;

o resultado 50000 * 50000 é superior ao maior inteiro positivo representável no tipo int. Existe overflow e, como o bit mais significativo representa o sinal, aparece assim um valor negativo.

6- 1val Indique o conteúdo da memória após a execução das duas instruções seguintes. Admita uma arquitetura de 32bits ***little-endian***.

```
mov $0x100, %eax
mov $0x105,(%eax)
```

bytes de memória e seus endereços:

0x05	0x01	0	0						
0x100	0x101	0x102	0x103	0x104	0x105	0x106	0x107	0x108	0x109

7- 1 val Ordene por ordem crescente dos tempos de acesso pelo CPU, cada um dos dispositivos/componentes de armazenamento de dados da arquitetura de um computador:

Disco; memória usando endereçamento direto; memória usando um endereço também em memória; registo.

registo; memória usando endereçamento direto; memória usando um endereço também em memória; disco

8- 1 val Descreva o papel do *assembler* (as) e do *linker* (ld) na construção de um programa executável. Inclua na explicação a resolução de símbolos (etiquetas) realizada pelo *linker* aquando da ligação de vários ficheiros objeto (.o).

9- 1,5 val Na arquitetura de Von Neumann, explicada nas aulas, os diversos componentes estão interligados por um barramento (bus) de sistema. Explique a natureza da informação transportada por este bus. Exemplifique o seu papel descrevendo como se obtém o conteúdo de memória quando é executada a instrução: **`mov (100), %eax`**

o Bus transporta informação de endereços, dados e controlo, entre os vários componentes.
Para o exemplo, leva do CPU para a memória o endereço 100 e, no controlo, o pedido de leitura da memória;
Depois recebe da memória e leva ao CPU a resposta a essa leitura, o seja, o conteúdo do endereço 100 de memória.

10- 1 val Considere a seguinte sequência de instruções

```
        mov  $5, %eax
        cmp  $6, %eax
        ja   L1
        movl $3, %ebx
        jmp  L2
L1:     movl %eax, %ebx
L2:     movl %ebx, %ecx
```

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

eax	5
ebx	3
ecx	3
ZF	0
OF	0
SF	1

11- 1 val Considere a seguinte sequência de instruções

```
        mov $5, %eax
        mov $6, %ebx
        push %eax
        push %ebx
        pop %eax
        pop %ebx
```

Escreva no espaço ao lado o conteúdo dos registos após a execução desta sequência.

eax	6
ebx	5

12- Pretende-se que construa uma função chamada `parapositivos` que tem a assinatura

```
int parapositivos( int *end, int dim )
```

- Recebe como parâmetros de entrada o endereço inicial de um vetor de inteiros, *end,* e a sua dimensão, *dim.*
- A função muda cada inteiro negativo existente no vetor para o respetivo valor absoluto (positivo) e devolve o número de mudanças efetuadas.

O exemplo seguinte ilustra o uso da função:

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

a) 2,5 val Implemente em C a função "`parapositivos`" descrita acima.

```
int parapositivos( int *end, int dim) {


   int  i,  c = 0;

   for ( i = 0;  i < dim;  i++ )
      if ( end[i] < 0 )  {
          end[i] = - end[i];
          c = c + 1;
      }

   return c;



}
```

b) 1,5 val Assuma que a função "parapositivos" está implementada. Escreva o código *assembly* correspondente à chamada da função passando o array x e o valor 6 como argumentos. Desenhe também o conteúdo da pilha imediatamente após a execução da chamada.

```
.data
   x:  .int   7, -15, 4, 7, -2, 0
.text
     ...

   pushl  $6
   pushl  $x
   call  parapositivos
```

pilha (cada linha são 4 bytes):

6
endereço de x
endereço de retorno

13- 2,5 val Pretende-se que escreva em *assembly* Intel 32 bits usando as mnemónicas e convenções do *as* (**as**sembler), o código equivalente ao seguinte fragmento de código C:

```
int ss[10] = { ... }; // vetor ja' inicializado
int i;
for ( i=0; i<9; i++ )
   ss[i] = ss[i] + ss[i+1];
```

Considere que as variáveis já estão declaradas e inicializadas.

```
.data
   ss:  .int ... ~~.ascii "............" # reserva de espaço para string ss (já inicializado)~~
   i:   .int  0    # reserva de espaço para variável i ~~cont~~
...
.text
   # código da sua solução:

          mov  i, %ecx
   myfor: cmp  $9, %ecx
          je fim
          inc %ecx
          mov ss( , %ecx, 4 ), %eax
          dec %ecx
          add %eax, ss( , %ecx, 4 )
          jmp myfor
   fim:
          mov %ecx, i
```

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.