Arquitetura de Computadores 2017-18

Ficha 6

Tópico: *Introdução ao assembly e invocação de subrotinas.*

Programação em *assembly*

1 - Escreva um programa em *assembly* Intel para Linux 32bits que faz a soma dos elementos de mesmo índice de dois vetores *vetor1* e *vetor 2* e deixa o resultado das somas no vetor2. Use como ponto de partida as seguintes variáveis:

```
.data
  vetor1:    .int  -1, 5, 1, 1, 4     # um vetor de inteiros
  vetor2:    .int  1, -3, 1, -5, 4    # vetor que fica com a soma dos dois
```

Use o *debugger* para executar o programa passo-a-passo e confirmar que calcula os valores corretos.

2 – Escreva um programa em *assembly* Intel para Linux 32bits para comparar os caracteres de duas cadeias cad1 e cad2, usando os seus códigos ASCII. A comparação será da esquerda para a direita e considera-se uma cadeia maior se o primeiro caracter diferente tiver um código maior que o da outra ou se a outra cadeia terminou. O resultado deve ser colocado num inteiro na memória -- caso as cadeias sejam iguais, o resultado é zero; caso a primeira cadeia seja maior, o resultado é 1; caso a segunda seja maior que a primeira, o resultado é -1.

Use como ponto de partida os exemplos seguintes:

```
.data           # secção de dados (variaveis)
 cad1:      .ascii   "hellu\n"
 cad2:      .ascii   "hello\n"
 res:       .int      0             # o resultado vai ser 1

.data           # secção de dados (variaveis)
 cad1:      .ascii   "flor"
 cad2:      .ascii   "floresta"
 res:       .int      0             # o resultado vai ser -1
```

Use o *debugger* para executar o programa passo-a-passo e confirmar que calcula o valor correto.

3 - Escreva um programa em *assembly* Intel para Linux 32bits que implementa uma subrotina que recebe dois números como argumento e devolve o maior dos dois. Na linguagem C, a subrotina corresponde à assinatura seguinte:

```
int maior( int n1, int n2 );
```

4 - Escreva um programa em *assembly* Intel para Linux 32bits que implementa uma subrotina que recebe uma cadeia como argumento e devolve o último carácter da cadeia. Na linguagem C, a subrotina corresponde à assinatura seguinte:

```
char ultimo( char *seq );
```

5 - Escreva um programa em *assembly* Intel para Linux 32bits que implementa uma subrotina que inverte os elementos de uma cadeia de caracteres. A sua subrotina deve receber: o endereço da sequência e o seu número de elementos. Na linguagem C, a subrotina corresponde à assinatura seguinte:

```
void inverte( void *seq, int len );
```

Complete o código seguinte onde o programa, depois de inverter as duas cadeias indicadas, deve afixar cada uma na saída standard (veja a chamada ao sistema WRITE em exemplos anteriores).

```
.data
  seq1:     .ascii    "ola"
  seq2:     .ascii    "hello"

.text
 .global _start
 _start: (...)
            # passa os argumentos da sequência seq1
            call inverte

            # passa os argumentos da sequência seq2
            call inverte

            # chamada ao sistema para escrever seq1
            # chamada ao sistema para escrever seq2
            # chamada ao sistema para terminar

inverte:
            ( ... )
            ret
```

Mais informação

(ver página seguinte)

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

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