Arquitetura de Computadores 2017-18

Ficha 5

Tópico: *Introdução ao assembly.*

Programação em *assembly*

1 - Escreva um programa em *assembly* Intel para Linux 32bits que faz a soma de duas variáveis em memória e deixa o resultado numa terceira. Use como ponto de partida as seguintes variáveis e veja o programa *hello.s* da ficha anterior:

```
.data
  var1:     .int 35  # valores exemplo (experimente com outros)
  var2:     .int 10
  result:   .int 0   # deve ficar com o resultado da soma
```

Não se esqueça de terminar corretamente o programa com o pedido EXIT ao sistema de operação.
Use o *debugger* para executar o programa passo-a-passo e confirme que a variável `result` fica com o valor correto imediatamente antes do programa terminar.

2 - Escreva um programa em *assembly* Intel para Linux 32bits que faz a soma dos elementos do vetor indicado. Use como ponto de partida as seguintes variáveis:

```
.data
  vetor:    .int  -1, 5, 1, 1, 4    # um vetor de inteiros
  soma:     .int 0   # deve ficar com a soma dos valores em 'vetor'
```

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

3 - Altere agora o programa hello.s da ficha anterior para que, antes de imprimir a mensagem, percorra o vetor de caracteres e, caso o carácter seja uma letra minúscula (e só neste caso), converte o carácter numa letra maiúscula. Execute e verifique a mensagem que aparece no terminal. Se necessário recorra ao *debugger* para verificar o seu funcionamento.

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