

# IA32 (Little Endian)

0x 01234567

$$\rightarrow 2^{\text{num}} = 0x100$$

0x100	67
0x101	45
0x102	23
0x103	01

Teste

a) +

0x08 e 20484

9

lê a instrução  
apontada para it

0x8e28f0c

9

lê -12(%esp)

0x8e28f0c

9

escreve  
em -12(%esp)

b) (?) O jeif também foi alterado para apontar para a  
próxima instrução

①

T →	1	1	1
V →	1	1	0
X →	1	0	1
L →	1	0	0
C →	0	1	1
D →	0	1	0
n →	0	0	1

675 → 010011100101101110

2 2 2

②

movl \$0, %ebx # n=0

cmpl \$8, %ecx

je .L2

addl %ebx, %ecx

diel %ebx

jmp .L3

③ 5) 0 0 0 0 0 0 0 0 / 1 1 0 1 / 1 0 0

⑦ 6) value is positive! au (16 bits)

④

⑤

1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 ... 0

⑦ Esta codificación representa o  
- ∞.

$$U - 127 = -30$$

$$\therefore U = -30 + 127$$

$$\therefore U = 97$$

$$\begin{array}{r} 97 \\ \hline 12 \\ 98 \end{array}$$
  
$$\begin{array}{r} 97 \\ \hline 12 \\ 84 \end{array}$$
  
$$\begin{array}{r} 97 \\ \hline 12 \\ 12 \end{array}$$
  
$$\begin{array}{r} 97 \\ \hline 12 \\ 0 \end{array}$$
  
$$\begin{array}{r} 97 \\ \hline 12 \\ 0 \\ 6 \end{array}$$
  
$$\begin{array}{r} 97 \\ \hline 12 \\ 0 \\ 3 \end{array}$$
  
$$\begin{array}{r} 97 \\ \hline 12 \\ 0 \\ 1 \end{array}$$
  
$$\begin{array}{r} 97 \\ \hline 12 \\ 0 \\ 0 \end{array}$$

a) ⑦

$$97 = 1100001$$

0	10
1	11
2	100
3	101
4	110
5	111
6	1000
7	1001
8	1010
9	1011
10	1100
11	1101
A	1110
B	1111
C	10000
D	10001
E	10010
F	10011

(5)  $\%ebh \rightarrow 0x5f1f7e878$

$\%edu \rightarrow 0x1$

$\%ebx \rightarrow 0x80484e0$

		Variável Inicial	
①	$\%ebx$	8	$\star$ a, sotoco
	$\%al$	$\star$	'a' 061
	$\%ecx$	point	0
	$\%edu$	?	0

$$\begin{aligned}0x61 &= 97 \rightarrow 'a' \\0x69 &= 108 \rightarrow 'i'\end{aligned}$$

adicionalmente não utilizados o `leave` para retornar o resultado da função.

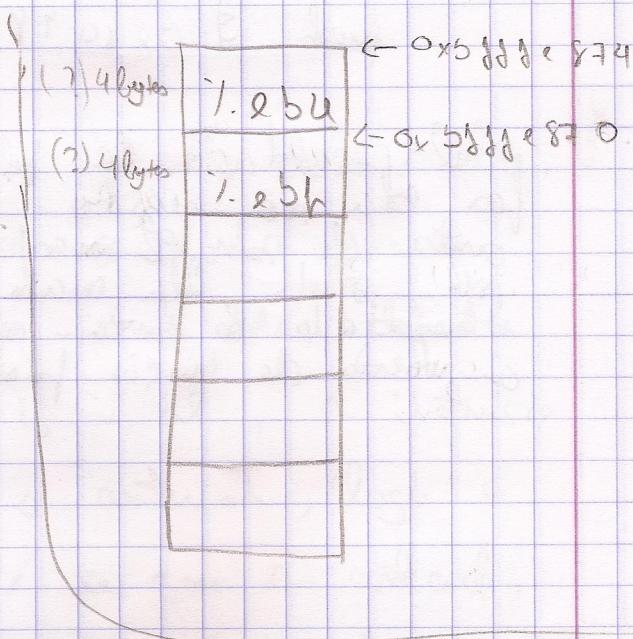
•  $\%esi \rightarrow ?$

②  
%esp  
%ebp  
%ebx  
%al  
%ecx

Quando está a executar o

, `Mov 0x8(%ebx), %ebx`  
 $0x8 = 0x5f1f7e874$

alterei as células de memória.  
de  $0x5f1f7e870$  a  $0x5f1f7e874$



③

```

    mov 0x8(%1,%1), %1.%bl      1
    mov (%1,%1), %1.%al        2
    xor %1.%cu, %1.%cu        3
    xor %1.%du, %1.%du        4
    test %1.%al, %1.%al       5
    je 8048330                 6
  
```

1 - Faz a o argumento da função para um registo %1.%bl

2 - Coloca o valor apontado por %1.%bl em %1.%al

3 - inicializa o %1.%cu a zero (count=0)

4 - inicializa o %1.%du a zero (i=0)

5&6 - verifica se a string é nulla (%al != '\0').  
Se já salta para 8048330.

④ A função conta-ai é chamada pela instrução

call 80483ac, os argumentos da mesma  
foram feitos antes na stack pelas instruções

sub \$0xc,%1.%sp → reserva espaço na stack

push \$0x80484c0

O resultado da função conta-ai foi colocado  
no registo %1.%cu e mesmo  
antes de chamar a função final, foi colocada  
na stack pela main, o valor de %1.%cu, ou seja,  
o resultado do conta-ai, fará que seja fornecido como  
argumento da função final.

⑤ Variável   Local	
i	-4(%ebp)
Count	-8(%ebp)

Como o código m̄o foi optimizado as variáveis locais ficaram arrançadas na memória.

Na instrução `push $8, %ebp` estamos a reservar espaço no stack para os colocarmos. Logo a seguir inicializámos a zero. Tudo isto a um o uma distância de -4 e -8 da base pointer.

É fácil vermos que a variável guardada em `-8(%ebp)` (o count) tem sempre que o valor do array é 97 ('a') ou 105 ('i'), incrementando-a sempre.

PS: outro elemento indicador é se fizermos movermos para `%eax` o valor de `-8(%ebp)`, sendo que o registo `%eax` vai conter o resultado da função.

⑥ A última instrução executada foi o `push %ebp`. Isto é `%ebp` está sempre a apontar para a próxima instrução, e como o `%ebp` está a apontar para a instrução mora (`rebu`), `%al`, a instrução que está a ser executada é a anterior, logo a última instrução executada é a que a antecede, que é, como já referido o `push %ebp`.

$$\textcircled{7} \quad 7.81f = 80483d0$$

base de andar para traz 16 bytes

$$16 = \underline{0} \underline{0} \underline{0} \underline{1} \underline{0} \underline{0} \underline{0} \underline{0}$$

$$\begin{array}{r} 11101111 \\ + 1 \\ \hline - 16 = 11110000 \end{array}$$

$$= 0x\text{F}0$$

(?)

$$\textcircled{8} \quad 0xFFFFFFF0$$

$$= -2$$

?

\textcircled{9}

afgs

1254

0xb382978

and 50.25h

end. return  $\rightarrow 804839a$

funcion  $\rightarrow 0x80484e0$

$\rightarrow$  fara uniendo de registros

citebh

200

$$\textcircled{10} \quad 20 \times 4 = 80 \text{ bytes}$$

$$\begin{array}{r} 1012 \\ \textcircled{0} 4012 \\ \textcircled{0} 2012 \end{array}$$

$$\begin{array}{r} 1012 \\ \textcircled{0} 0212 \\ \textcircled{0} 0112 \\ \textcircled{0} 0012 \end{array}$$

7)  $7 \cdot 8 = 80$  bytes

em de andar para traz 16 bytes

$$16 = 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0$$

$$\begin{array}{r} 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1 \\ + \qquad \qquad \qquad 1 \\ \hline 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0 \end{array}$$

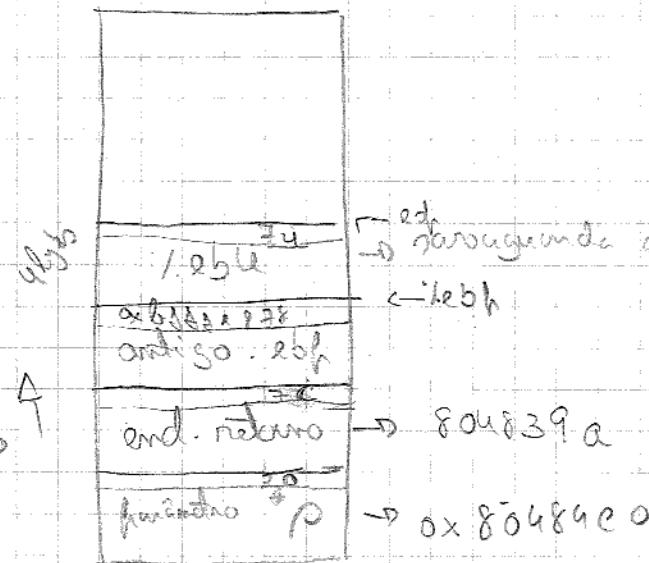
$$= 0x\text{F0} \quad ?$$

8)  $0x\text{FFFF FFF0}$

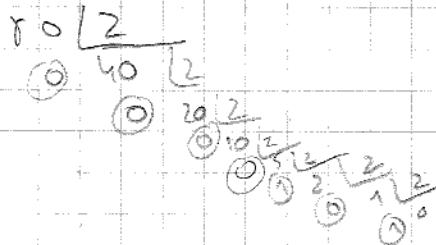
$$= -2$$

?

9)



10)  $20 \times 4 = 80$  bytes



$$80 = 01\ 01\ 00\ 00$$

$$= 0x5\ 0$$

$$\begin{array}{r} 0\ 3\text{FFF}\ 5\ 874 \\ - 8\ 0\ 0\ 0\ 0\ 0\ 0\ 5\ 0 \\ \hline 0\ 3\text{FFF}\ E\ 82\ 4 \end{array}$$

O segundo elemento fixaria em 0x BFFF E8 24+4

= 0x BFFF E8 28

(Vou sei se estou certo!)

- ⑪ Chamar funções é um processo que infiltra muitas informações e muitas ideias à memória (referências a regiões, pensar argumentos, guardar base pontos, etc), logo gente que podemos devemos adicionar a uma variável o nome da função, pena que todo este trabalho excepcional seja evitado. Este processo não pode ser elaborado pelo compilador pois este não sabe qual é o resultado de função. Imaginemos uma função que nos dá as horas exatas naquelle instante. O valor dessa função vai variar dentro de um ciclo. Logo o compilador tem de salvaguardar estas exceções e não adicionar para nós o código. Pode ser uma variável e depois usá-la sempre. Esta é uma otimização que só nós (humanos) podemos fazer.