SISTEMAS COMPUTAÇÃO - RESOLUÇÃO TESTE

① a) 8 hoteis → 3 bits
66 quartos → 7 bits
28 andares -5...22 → 5 bits (0..31)

excesso → permite passar o limite inferior do intervalo logo excesso 5 (-5..26)

Hotel	Andares	Quarto
011	00011	0011111

b) 11 caracteres
↳ as 10 primeiras letras + "; "

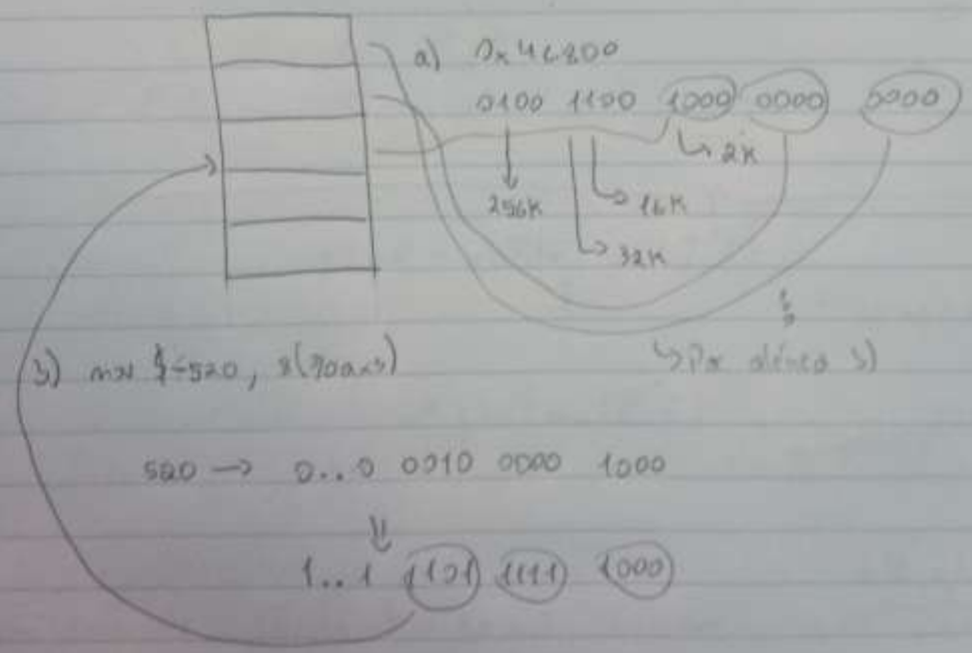
São necessários 4 bits

0000 .. 1001 1111

JA; HHH

1001 0000 1111 0111 0111 0111

② IA-20



a) 0x46800
0100 1100 1000 0000 0000
↓ 256K, 16K, 32K, 2K

↳ (Px dúvida b))

b) mov $-520, 8(%eax)

520 → 0..0 0010 0000 1000
⇓
1..1 1101 1111 1000

(a) 0,1

× 2

0,2

× 2

0,4

× 2

0,8

× 2

1,6

× 2

1,2

× 2

0,4

E agora vai repetir-se sempre

$0,0(0011)_2$

$0.0001\ 1001\ 1001 | 10011$

$V = 0.0001\ 1001\ 1010 \times 2^0$ → $1.1001\ 1001\ 1011_2 \times 2^{-4}$

logo

$V = 1.F \times 2^{Exp-63}$

$Exp - 63 = -4$

$\Rightarrow Exp = 59$

$0 | 0111011 |$

b) 0x80040

$1 | 000\ 0000 | 0000\ 1000\ 0000$ 1/12

Exp frac

↳ Excepção → Nº desnormalizado

$-0.00001_2 \times 2^{-62}$

exp − 63
↳ 1
= −62

$= -1 \times 2^{-67}$ //

(4) a) %eax no final da execução da função

%eax = 0x00.. 100   %esp = 0x00.. 6FFC

%eip = 0x00.. 4051

código:

⋮

0x4051 addl %eax, 12(%esp) → esta operação já foi feita!!! que é o valor que for na %eax

0x4054 popl %eax

Memória

0x0 6FFC: 0xAA 0x00 0x00 0x01

logo o %eax vai ter que como é little vai ser 0x010000AA

b) %esp ← stack pointer

↳ O esp aponta sempre pa posição de memória que chamou esta função!!! não esquecer que é little endian a ler da memória!!!

(5) 0x4054 : popl %eax → Assume-se que o IP até aqui seja, tem o valor 0x4054

a) Informação operas no barramento endereços pa esta instrução

1º Fazer a busca da instrução pelo b. endereços 0x4054

2º Como é um pop tem de se ler o topo da stack %esp = 0x00.. 6FFC

b) Todos os registos e células memória alterados

IP, SP, EAX

IP ↳ 0x4055

EAX ↳ 0x010000AA

SP ↳ por ser um pop soma-se 4 ao esp

2º parte

	↱ Endereço base	
a	movl %ecx, %eax	int → 4bytes
a[4]	mov 16(%ecx), %eax	
a[i]	mov (%ecx, %edx, 4), %eax ↳ base ↳ índice ↳ fator de escala	a[4] → (16bytes + endereço base
&a[i]	leal (%ecx, %edx, 4), %eax	Diz que no enunciado o i tá no %edx

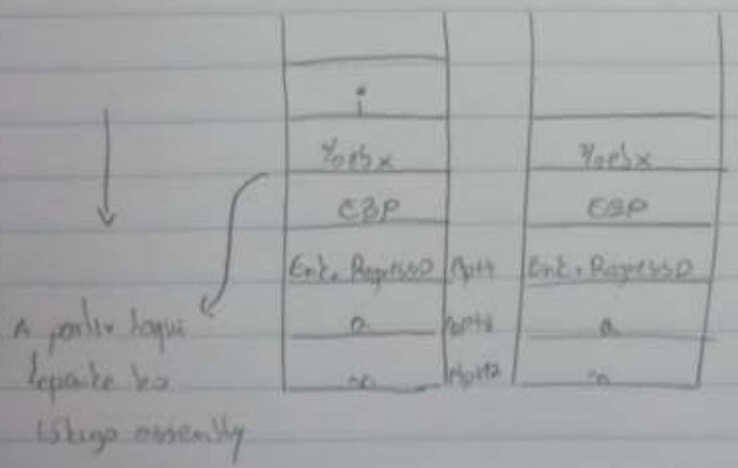
② Paridade de cada elemento a[i] ou seja o &

se o a[i] for par vai dar 0

se o a[i] for impar vai dar 1

A instrução é o test → testa mas não guarda o resultado em lado nenhum. Isto só vai alterar as flags, neste caso a zero flag.

③ Versão – 01 Versão – 02



para-par (int a[], int n)

int i; ↳ Isto fica sempre na stack, da direita para esquerda!

Código otimizado tem sempre lá as variáveis!!!

④ movl (%ecx, %edx, 4), %eax → Tá a aceder-se ao a[i]

...

incl %eax → Soma 1 ao valor de a[i]

movl %eax, (%ecx, %edx, 4) → Escreve o a[i]+1

nota: xorl %ebx, %ebx
põe o ebx a 0

⑤ Identifique a próxima instrução a ser executada

PC = 0x8048544 → ver o código e é o que tem o mesmo valor do eip

Iterações do ciclo for foram feitas e quantas vão ser feitas no total?
a variável i tá no ebx
o ebx é o n

2 e 20

Endereço inicial do vetor a
ecx é onde começa o vetor – ver o valor do endereço ecx

⑥ Funções que fez o call

|________| xx xx xx xx xx call

⑦ 0x..: 7e ?? jl 8048564
0x..77: 53

67 – 77 = –13

–0x13

0001 0011
CP 2
1110 1101

| ?? = E D |

nota: leal não vai à memória
incl (%eax) → são 2 acessos à memória!!!

o ACD por ex impor 1S cada, lipos só conta se o ACD por impor que é quanto vai à memória se ACD por par não vai à memória