SISTEMAS COMPUTAÇÃO - RESOLUÇÃO TESTE

① a) 8 hoteis → 3 bits
65 quartos → 7 bits
28 andares -5...22 → 5 bits (0..31)
excesso → permite puxar o limite inferior do intervalo logo excesso 5 (-5..26)

Hotel	Andares	Quarto
011	00011	0011111

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

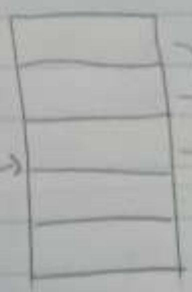
São necessários 4 bits

0000 .. 1001 1111

JA; HHH

1001 0000 1111 0111 0111 0111

② IA-20



a) 0x46800
0100 1100 1000 0000 0000
↳ 2K
256K
↳ 16K
↳ 32K
↳ Px década b)

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

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

③ 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,(0011)₂

0.0001 1001 1001 | 10011

V = 0.0001 1001 1010 × 2^0 → 1.1001 1001 1011₂ × 2^-4

logo

V = 1.F × 2^(Exp − 63)

Exp − 63 = −4

⇒ Exp = 59

0|0111101|

b) 0x8008

1|000 0000| 0000 1000 0000 1/10

Exp frac

↳ Excepção → nº desnormalizado

− 0.0000 1₂ × 2^−62 exp − 63

↳ 1

= −62

= − 1 × 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!!!

0x4054 popl %eax que é o valor que for na %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 é popl %eax → Assume-se que o IP tá aqui ou seja, tem o valor 0x4054

a) Informação operar 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

2ª parte

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

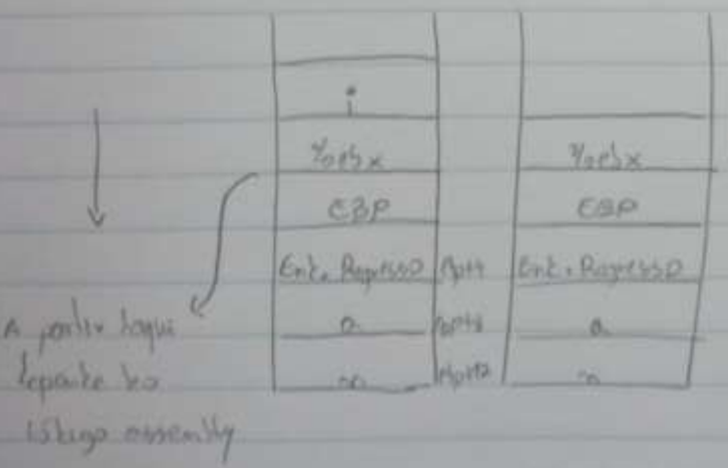
② Paridade de cada elemento a[i] ou seja o &

se o a[i] for par vai dar 0

se o a[i] for ímpar 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, n)

int i; ↳ Isto fica sempre na stack, da direita pa 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
é o por o ebx a 0

⑤ Identifique a próxima instrução a ser executada

PC = 0x8048564 → ver o código e ver o que tem o mesmo valor de eip

Iterações do ciclo for foram feitas e quantas vão ser feitas no total:

a variável i tá no ebx

o edx é o n

2 e 20

Endereço inicial do vetor a

ecx é endereço 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

CP2

1110 1101

?? = E D

nota: leal não vai à memória

incl (%eax) → são 2 acessos à memória!!!

o a[i] por ser impar dá certo, logo só conta se o a[i] for impar que é quando vai à memória se a[i] for par não vai à memória