Análise do código gerado

Código obtido a partir dos pontos 1,2 e 3.

```
0x080483c4 <contar_valores+0>:  push   %ebp
0x080483c5 <contar_valores+1>:  mov    %esp, %ebp
0x080483c7 <contar_valores+3>:  push   %edi
0x080483c8 <contar_valores+4>:  push   %esi
0x080483c9 <contar_valores+5>:  push   %ebx
0x080483ca <contar_valores+6>:  mov    0x8(%ebp), %esi
0x080483cd <contar_valores+9>:  mov    0xc(%ebp), %edi
0x080483d0 <contar_valores+12>: mov    0x10(%ebp), %ecx
0x080483d3 <contar_valores+15>: xor    %ebx, %ebx
0x080483d5 <contar_valores+17>: xor    %edx, %edx
0x080483d7 <contar_valores+19>: nop
0x080483d8 <contar_valores+20>: movl   $0x0, (%ecx, %edx,4)
0x080483df <contar_valores+27>: inc    %edx
0x080483e0 <contar_valores+28>: cmp    $0xc, %edx
0x080483e3 <contar_valores+31>: jle    0x80483d8 <contar_valores+20>
0x080483e5 <contar_valores+33>: xor    %edx, %edx
0x080483e7 <contar_valores+35>: cmp    %edi, %edx
0x080483e9 <contar_valores+37>: jge    0x8048405 <contar_valores+65>
0x080483eb <contar_valores+39>: nop
0x080483ec <contar_valores+40>: mov    0x4(%esi, %edx,8), %eax
0x080483f0 <contar_valores+44>: incl   (%ecx, %eax,4)
0x080483f3 <contar_valores+47>: mov    0x4(%esi, %edx,8), %eax
0x080483f7 <contar_valores+51>: mov   (%ecx, %eax,4), %eax
0x080483fa <contar_valores+54>: cmp    %ebx, %eax
0x080483fc <contar_valores+56>: jle    0x8048400 <contar_valores+60>
0x080483fe <contar_valores+58>: mov    %eax, %ebx
0x08048400 <contar_valores+60>: inc    %edx
0x08048401 <contar_valores+61>: cmp    %edi, %edx
0x08048403 <contar_valores+63>: jl     0x80483ec <contar_valores+40>
0x08048405 <contar_valores+65>: mov    %ebx, %eax
0x08048407 <contar_valores+67>: pop    %ebx
0x08048408 <contar_valores+68>: pop    %esi
0x08048409 <contar_valores+69>: pop    %edi
0x0804840a <contar_valores+70>: leave
0x0804840b <contar_valores+71>: ret
```

5. Tabela de alocação de registos.

%esi	mão
%edi	tam
%ecx	*valores
%edx	i
%ebx	maior*
%eax	mao[i]

*No entanto, no final faz-se um mov para passar o %ebx para o %eax para retornar o valor da variável maior.

7.
Realizou-se um breakpoint em 0x08048407.

Área da memória:

0xbfffe800	2	1	1	1
0xbfffe810	1	2	0	3
0xbfffe820	0	0	0	1
0xbfffe830	1			

Visto que cada inteiro ocupa 4 bytes, o espaço ocupado na memória foi de 4 x 13 =52 bytes.

9.

0x080483ca mov 0x8(%ebp), %esi 0x080483cd mov 0xc(%ebp), %edi 0x080483d0 mov 0x10(%ebp), %ecx	Vai à stack buscar os parâmetros
0x080483d3 xor %ebx, %ebx 0x080483d5 xor %edx, %edx	Inicializa o “maior” e o “i” , ficando estes a zero.
0x080483d8 movl $0x0, (%ecx, %edx,4)	Representa o “valores[i]=0”
0x080483df inc %edx	“i++” incrementa mais 1 cada vez que ocorre o ciclo
0x080483e0 cmp $0xc, %edx 0x080483e3 jle 0x80483d8 <contar_valores+20>	“i<13”, pois é a parte da condição do ciclo for, se isto se verificar salta-se para a instrução 0x80483d8
0x080483ec mov 0x4(%esi, %edx,8), %eax	“mão[i].valor”
0x080483f0 incl (%ecx, %eax,4)	“i++”
0x080483fa cmp %ebx, %eax	“valores[mao[i].valor] > maior”
0x08048405 mov %ebx, %eax	Como o %ebx é responsável pelo return, tem se que passar o valor do ”maior” do registo %ebx para o registo %eax.