

Análise do código gerado

Grupo 41

1 Código *assembly* da função contar_valores

```
0x080483c4: push    %ebp
0x080483c5: mov     %esp,%ebp
0x080483c7: push    %edi
0x080483c8: push    %esi
0x080483c9: push    %ebx
0x080483ca: mov     0x8(%ebp),%esi
0x080483cd: mov     0xc(%ebp),%edi
0x080483d0: mov     0x10(%ebp),%ecx
0x080483d3: xor     %ebx,%ebx
0x080483d5: xor     %edx,%edx
0x080483d7: nop
0x080483d8: movl    $0x0, (%ecx,%edx,4)
0x080483df: inc     %edx
0x080483e0: cmp     $0xc,%edx
0x080483e3: jle     0x80483d8
0x080483e5: xor     %edx,%edx
0x080483e7: cmp     %edi,%edx
0x080483e9: jge     0x8048405
0x080483eb: nop
0x080483ec: mov     0x4(%esi,%edx,8),%eax
0x080483f0: incl    (%ecx,%eax,4)
0x080483f3: mov     0x4(%esi,%edx,8),%eax
0x080483f7: mov     (%ecx,%eax,4),%eax
0x080483fa: cmp     %ebx,%eax
0x080483fc: jle     0x8048400
0x080483fe: mov     %eax,%ebx
0x08048400: inc     %edx
0x08048401: cmp     %edi,%edx
0x08048403: jl      0x80483ec
0x08048405: mov     %ebx,%eax
0x08048407: pop     %ebx
0x08048408: pop     %esi
0x08048409: pop     %edi
0x0804840a: leave
0x0804840b: ret
```

2 Tabela de alocação de registos

Variável	Registo
mao	%ebx
tam	%edi
valores	%ecx
i	%edx
maior	%ebx
c	Não existe pois esta variável não é utilizada

3 Área de memória associada à variável valores

Através do *debugger* foi possível descobrir que a variável **valores** apontava para o endereço de memória 0xbfffe9a0. Logo, sendo o processador IA-32 *little endian*, como um inteiro ocupa 4 *bytes* e cada célula de memória possui de espaço 1 *byte*, são necessários 4 células de memória para cada valor da variável. Assim sendo, como o *array* tem tamanho 13, irá ocupar 52 *bytes* de espaço na memória. Importante referir

Endereço
0xbfffe9a0 <— valores
0xbfffe9a4 <— valores[1]
0xbfffe9a8 <— valores[2]
0xbfffe9ac <— ...
0xbfffe9b0
0xbfffe9b4
0xbfffe9b8
0xbfffe9bc
0xbfffe9c0
0xbfffe9c4
0xbfffe9c8
0xbfffe9cc
0xbfffe9d0
...

4 Indexação da matriz

A indexação da matriz é realizada agrupando os valores das cartas da **mao** na respetiva posição do *array valores*. Ao mesmo tempo, é guardado o maior valor que será devolvido no fim da função. As linhas de assembly que correspondem a esta operação são:

```
0x080483e5: xor    %edx,%edx
0x080483e7: cmp    %edi,%edx
0x080483e9: jge    0x8048405
0x080483eb: nop
0x080483ec: mov    0x4(%esi,%edx,8),%eax
0x080483f0: incl   (%ecx,%eax,4)
0x080483f3: mov    0x4(%esi,%edx,8),%eax
0x080483f7: mov    (%ecx,%eax,4),%eax
0x080483fa: cmp    %ebx,%eax
0x080483fc: jle    0x8048400
0x080483fe: mov    %eax,%ebx
0x08048400: inc    %edx
0x08048401: cmp    %edi,%edx
0x08048403: jl     0x80483ec
```

5 Correspondência entre as instruções em C e as instruções em *assembly*

```
int contar_valores(MAO mao, int tam, int *valores) {  
    int i, c;  
    int maior = 0;
```

```
0x080483c4:  push    %ebp  
0x080483c5:  mov     %esp,%ebp  
0x080483c7:  push    %edi  
0x080483c8:  push    %esi  
0x080483c9:  push    %ebx  
0x080483ca:  mov     0x8(%ebp),%esi // argumento mao  
0x080483cd:  mov     0xc(%ebp),%edi // argumento tam  
0x080483d0:  mov     0x10(%ebp),%ecx //argumento *valores  
0x080483d3:  xor     %ebx,%ebx // i = 0  
0x080483d5:  xor     %edx,%edx // maior = 0
```

```
    for(i = 0; i <13; i++)  
        valores[i] = 0;
```

```
0x080483d8:  movl    $0x0, (%ecx,%edx,4) // valores[i] = 0  
0x080483df:  inc     %edx // i++  
0x080483e0:  cmp     $0xc,%edx // compara i com 13  
0x080483e3:  jle     0x80483d8
```

```
    for(i = c = 0; i <tam; i++) {  
        valores[mao[i].valor]++;  
        if(valores[mao[i].valor] >maior)  
            maior = valores[mao[i].valor];  
    }
```

```
0x080483e5:  xor     %edx,%edx // i = 0  
0x080483e7:  cmp     %edi,%edx // compara i com tam  
0x080483e9:  jge     0x8048405  
0x080483eb:  nop  
0x080483ec:  mov     0x4(%esi,%edx,8),%eax // mao[i].valor  
0x080483f0:  incl    (%ecx,%eax,4) // valores[mao[i].valor]++  
0x080483f3:  mov     0x4(%esi,%edx,8),%eax // mao[i].valor  
0x080483f7:  mov     (%ecx,%eax,4),%eax // valores[mao[i].valor]  
0x080483fa:  cmp     %ebx,%eax // compara valores[mao[i].valor] com maior  
0x080483fc:  jle     0x8048400  
0x080483fe:  mov     %eax,%ebx // maior = valores[mao[i].valor]  
0x08048400:  inc     %edx // i++  
0x08048401:  cmp     %edi,%edx // compara i com tam  
0x08048403:  jl      0x80483ec
```