1. Considere a figura anexa com um programa C desenvolvido e executado no servidor de apoio às sessões laboratoriais. A função `para_par(a, n)` adiciona uma unidade a todos os elementos ímpares do vetor `a[]`.

2. As 11 questões nesta prova para 14 valores estão cotadas para **1 valor**, com exceção das questões **3**, **5** e **6** que valem **2 valores** cada.

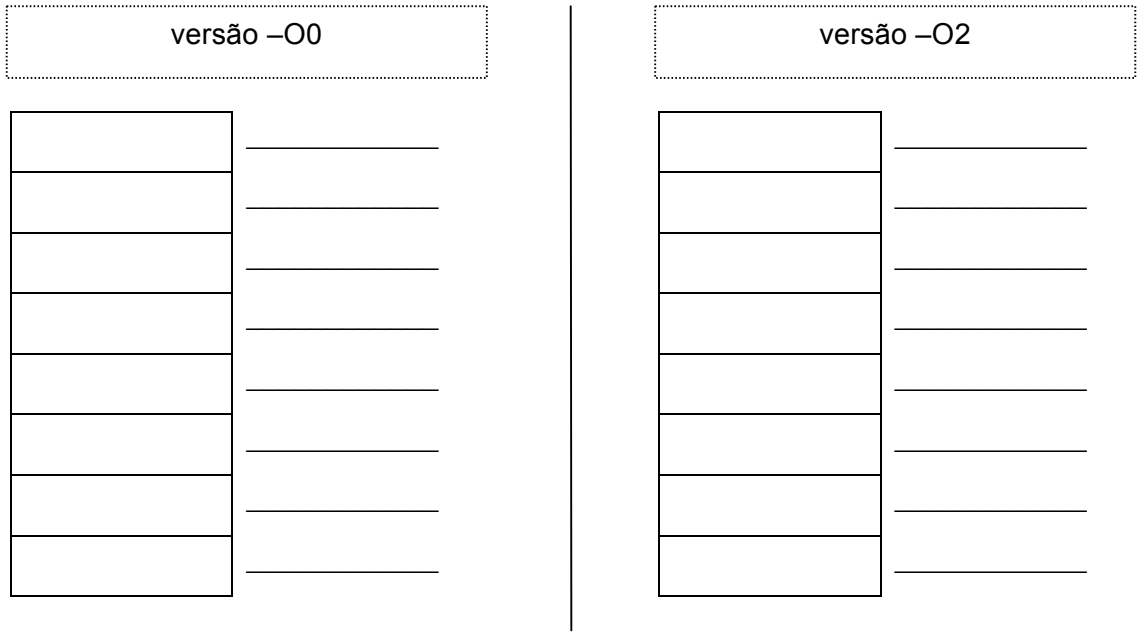
3. É permitido o uso de uma **nota auxiliar de memória**, manuscrita, com dimensão máxima de 1 folha A4.

1. Considerando o código otimizado `(-O2)` da função `para_par()`, **complete** a tabela considerando que os registos `ecx`, `edx` e `eax` contêm, respetivamente, o endereço do início do vetor `a(int)`, o índice `i` (inteiro) e o registo de destino na instrução na 3ª coluna (i.e., onde vai ficar o valor da expressão indicada na 1ª coluna).

Expressão (C)	Explicação	Instrução
`a`	`Endereço base de a`	`movl %ecx , %eax`
`a[4]`		`mov , %eax`
`a[i]`		
`&a[i]`		

2. Explique a operação lógica utilizada no código em C para verificar a paridade de cada elemento `a[i]` e indique como é realizada no código otimizado (`-O2`) em *assembly*.

3. Usando as figuras abaixo, **ilustre** o quadro de ativação da função `para_par` (*stack frame*), para cada uma das versões compiladas. **Indique** claramente todos os campos pertinentes, mas não necessita de identificar nem os endereços de memória onde cada campo fica armazenado nem os respectivos conteúdos. Considere que cada célula da tabela corresponde a 4 *bytes* e indique recorrendo a uma seta a posição apontada pelo *frame pointer* `(%ebp)`.



4. Dentro do corpo do ciclo `for`, na versão otimizada do código *assembly*, pode encontrar as instruções abaixo. Explique claramente a que sequência do código em C correspondem e justifique de que forma suportam essa funcionalidade:

```
        movl    (%ecx,%edx,4), %eax
        …
        incl    %eax
        movl    %eax, (%ecx,%edx,4)
```

Considere na figura anexa a informação obtida através do `objdump` e do `gdb`.
Responda, **justificando**, a cada uma das alíneas das questões 5 e 6.

5. a) **Identifique** a próxima instrução a ser executada.

5. b) **Indique** quantas iterações do ciclo `for` já foram executadas e quantas serão executadas no total.

5. c) **Indique** qual o endereço inicial do vetor `a`.

6. a) Sabendo que a instrução `call` que invocou a função `para_par()` tem 5 *bytes* de tamanho, qual o endereço da posição de memória onde se encontra armazenado o primeiro *byte* desta instrução?

6. b) Quantos elementos do vetor `a` *são* ímpares?

7. Indique, justificando, o valor em hexadecimal do 2º *byte* da instrução de salto que começa no endereço `0x8048375` (ver figura anexa, onde este *byte* está representado por **??**).

8. Calcule, mostrando os cálculos, o nº de acessos adicionais à memória na versão de código compilada sem qualquer otimização, quando comparada com a versão compilada com `-O2`.

9. Considere uma função semelhante à da figura anexa, que recebe como argumentos o mesmo vetor `a[]` e uma variável inteira por referência (`int *sum`), e calcula o somatório de todos os inteiros do vetor, ficando o resultado no local indicado no 2º argumento. No interior do único ciclo `for` está a instrução em C `sum += a[i]`. **Apresente**, **justificando**, sugestões de alteração a este código para tornar a sua execução mais eficiente.

10. Considere o desenvolvimento de código científico em C, cuja especificação impõe que as variáveis do tipo `real` seja representadas com pelo menos 8 algarismos significativos. **Indique, justificando**, se consegue representar essas variáveis como `float` ou se tem de as representar como `double`.

11. Considere a execução destas 2 instruções (do código otimizado da figura anexa) numa arquitetura típica RISC com apenas um modo de endereçamento (*Reg + Offset*).

```
804836e:        40              inc    %eax
804836f:        89 04 91        mov    %eax,(%ecx,%edx,4)
```

Explique, justificando, como seria compilada para esta arquitetura a operação codificada por estas 2 instruções do IA-32, usando a sintaxe do IA-32 para mostrar esse código.

Comente a dimensão deste pedaço de código na arquitetura RISC face à dimensão no IA-32.

```
void para_par (int a[], int n) {
  int i;

  for (i=0 ; i<n ; i++) {
    if (a[i] & 0x01) {
      a[i] += 1;
    }
  }
}
```

---- código assembly obtido com gcc -S -O0 ----

```
...
para_par:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $4, %esp
        movl    $0, -8(%ebp)
.L2:
        movl    -8(%ebp), %eax
        cmpl    12(%ebp), %eax
        jl      .L5
        jmp     .L1
.L5:
        movl    -8(%ebp), %eax
        leal    0(,%eax,4), %edx
        movl    8(%ebp), %eax
        movl    (%eax,%edx), %eax
        testl   $1, %eax
        je      .L4
        movl    -8(%ebp), %eax
        leal    0(,%eax,4), %ebx
        movl    8(%ebp), %ecx
        movl    -8(%ebp), %eax
        leal    0(,%eax,4), %edx
        movl    8(%ebp), %eax
        movl    (%eax,%edx), %eax
        incl    %eax
        movl    %eax, (%ecx,%ebx)
.L4:
        leal    -8(%ebp), %eax
        incl    (%eax)
        jmp     .L2
.L1:
        addl    $4, %esp
        popl    %ebx
        leave
        ret
```

---- código assembly obtido com gcc -S -O2 ----

```
...
para_par:
        pushl   %ebp
        movl    %esp, %ebp
        xorl    %edx, %edx
        pushl   %ebx
        movl    12(%ebp), %ebx
        cmpl    %ebx, %edx
        movl    8(%ebp), %ecx
        jge     .L9
.L7:
        movl    (%ecx,%edx,4), %eax
        testl   $1, %eax
        je      .L4
        incl    %eax
        movl    %eax, (%ecx,%edx,4)
.L4:
        incl    %edx
        cmpl    %ebx, %edx
        jl      .L7
.L9:
        popl    %ebx
        leave
        ret
```

---- executável com -O2 após objdump -d -----

```
...
08048354 <para_par>:
8048354:  55              push %ebp
8048355:  89 e5           mov  %esp,%ebp
8048357:  31 d2           xor  %edx,%edx
8048359:  53              push %ebx
804835a:  8b 5d 0c        mov  0xc(%ebp),%ebx
804835d:  39 da           cmp  %ebx,%edx
804835f:  8b 4d 08        mov  0x8(%ebp),%ecx
8048362:  7d 13           jge  8048377
8048364:  8b 04 91        mov(%ecx,%edx,4),%eax
8048367:  a9 01 00 00 00  test $0x1,%eax
804836c:  74 04           je   8048372
804836e:  40              inc  %eax
804836f:  89 04 91        mov %eax,(%ecx,%edx,4)
8048372:  42              inc  %edx
8048373:  39 da           cmp  %ebx,%edx
8048375:  7c ??           jl   8048364
8048377:  5b              pop  %ebx
8048378:  c9              leave
8048379:  c3              ret
```

- breakpoint em para_par (executável com -O2) _

```
(gdb) info registers
eax            0x2         2
ecx            0x8049620
edx            0x2         2
ebx            0x14        20
esp            0xbfffe8a4
ebp            0xbfffe8a8
eip            0x8048364

----------------------

(gdb) x/5wx $esp
0xbfffe8a4:
   0x007d3ff4 0xbfffe8c8 0x080483a4  0x08049620
0xbfffe8b4:
   0x00000014

----------------------

(gdb) x/20wb $ecx
0x08049620:
   0x00000000 0x00000002 0x00000002 0x00000003
0x08049630:
   0x00000004 0x00000005 0x00000006 0x00000007
0x08049640:
   0x00000008 0x00000009 0x0000000a 0x0000000b
0x08049650:
   0x0000000c 0x0000000d 0x0000000e 0x0000000f
0x08049660:
   0x00000010 0x00000011 0x00000012 0x00000013
```