Questão 1: Quais são as semelhanças e as diferenças entre "Ola-add.c" e "a.out"? Sabe-se que há algumas equivalências entre elementos de "Ola-add.c" e "a.out" pois são diferentes representações do mesmo algoritmo. Apenas com a informação da captura de tela é possível mostrar algum elemento correspondente? No contexto do processador, como é possível saber se um arquivo pode ser executado? No contexto do computador, o que nos impede de executar "Ola-add.c"? No contexto do processador, "Ola-add.c" poderia ser executado? Elabore sobre isso em sua resposta.

“Ola-add.c” é o código-fonte que um compilador utilizou para gerar o binário executável “a.out”, ou seja, eles são equivalentes, porém “Ola-add.c” é um arquivo de texto-plano e “a.out” é um arquivo em linguagem de máquina.

Não é possível apontar elementos correspondentes apenas com a captura de tela, pois mesmo que a coluna à direita mostre os dados compilados traduzidos para algo mais próximo da linguagem humana, o texto não é compreensível ou de fácil apontamento para o código-fonte.

No contexto do processador, tudo é a mesma coisa; um processador entende dados como informações, isto é, não é distinguível uma fita de bits “executável” de uma “não executável”.

O sistema operacional nos impede de executar “Ola-add.c”, porque “Ola-add.c” não tem extensão de um arquivo executável e por não ser possível validar os metadados de seu cabeçalho, até porque ele não tem um.

No contexto do processador, “Ola-add.c” poderia sim ser executado.

Questão 2: Que partes desses arquivos correspondem à função "main" de "Ola-add.c"? (copie e cole as linhas dos respectivos arquivos, indicando a que arquivo pertencem e explique como você chegou a essa correspondência). Arquivos: Ola-add.s e Ola-add.dump

A seguinte parte de “Ola-add.s” corresponde à função “main” de “Ola-add.c”:

```
main:
.LFB6:
        .cfi_startproc
        endbr64
        pushq  %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq   %rsp, %rbp
        .cfi_def_cfa_register 6
        movl   $32, -12(%rbp)
        movl   $-4, -8(%rbp)
        movl   -12(%rbp), %eax
        subl    -8(%rbp), %eax
        movl   %eax, -4(%rbp)
        nop
        popq   %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

A próxima parte de “Ola-add.dump” corresponde à função “main” de “Ola-add.c”:

```
        0000000000001129 <main>:
```

```
1129:       f3 0f 1e fa             endbr64
112d:       55              push   %rbp
112e:       48 89 e5                mov    %rsp,%rbp
1131:       c7 45 f4 20 00 00 00    movl   $0x20,-0xc(%rbp)
1138:       c7 45 f8 fc ff ff ff    movl   $0xfffffffc,-0x8(%rbp)
113f:       8b 45 f4                mov    -0xc(%rbp),%eax
1142:       2b 45 f8                sub    -0x8(%rbp),%eax
1145:       89 45 fc                mov    %eax,-0x4(%rbp)
1148:       90              nop
1149:       5d              pop    %rbp
114a:       c3              ret
```

Eu cheguei nessa conclusão pela comparação dos “nomes das seções”: “*main:*” em “Ola-add.s”, e “*0000000000001129 <main>:*” em “Ola-add.dump”. É possível ver que os nomes acompanham o nome original do função “main” escrita em C.

Questão 3: Que correspondências há entre a saída de "objdump" , o enunciado do EP1 (seções: 3, 4, 5 e 7) e o IAS (figuras 1.6, 1.7, Tabela 1.1)

As correspondências são que: na saída de “objdump”, possuímos os conceitos de endereços, palavras, códigos de instrução e quais peças no hardware estão sendo usadas para o programa escrito originalmente em C executar corretamente.
Para isso, é importante exemplificar, logo, usaremos o bloco de “Ola-add.dump” que foi colocado na questão anterior:

1. Primeiro o programa começa na posição 0x1129 com a instrução *endbr64*
2. Depois vem *push* na posição 0x112d, que coloca dados no topo da pilha,
3. E então *mov* na posição 0x112e, que move os dados do registrados em %rbp para %rsp

Neste pequeno trecho de código assembly, é possível verificar os comandos e suas traduções em hexadecimal, que chegarão para o processador em binário. Também é possível ver o endereço do primeiro bit de cada palavra – a primeira palavra tem 4 bytes. Por fim, a linguagem de montagem, como mostrado no item 3, utiliza os registradores para armazenar dados temporariamente, assim como era no IAS.