Trabalho 1 (Fases 1, 2 e 3)

Edson Ricardo da Costa, Jonatas Van Groll Lemos e Lourenço Souza

[1] Sistemas Operacionais - Turma 031

[2] Graduação de Engenharia de Software – Escola Politécnica

[3] Pontifícia Universidade Católica do Rio Grande do Sul(PUCRS)

{Edson.Costa, Jonatas.Lemos, Lourenco.Souza} @edu.pucrs.br;

Resumo. *Este pequeno resumo descreve como foi desenvolvido o primeiro trabalho da disciplina de Sistemas Operacionais, no 2º semestre de 2021. Mostraremos como foi feita a implementação do trabalho, como se deve executar o programa para poder testar os oito casos de teste e mostraremos também o resultado de cada um dos casos de teste desenvolvidos.*

DESENVOLVIMENTO DA SOLUÇÃO

Com relação a solução do problema proposto pelas fases 1, 2 e 3, o grupo optou por uma abordagem simples e mais vantajosa. Como o problema era relativamente pequeno, o grupo decidiu pegar o código base disponibilizado pelo professor e apenas acrescentar o código com a solução para os problemas propostos. Seguindo essa abordagem, o grupo teve vantagens em entender o código e aplicar o conteúdo aprendido em aula, além do mais, isso facilitará a correção para o professor.

Neste trabalho, desenvolvemos as exceções propostas pelo professor no enunciado e também desenvolvemos a chamada de sistema. Isso ocorre no código por meio dos manipuladores, que chamamos de manipulador de exceção e manipulador de chamada de sistema, respectivamente nomeados no código como "*interruptionsHandler*" e "*trapHandler*".

Quando estamos executando algum programa de teste, o sistema carrega o programa na VM e então passa a executar o programa, caso em algum momento tenhamos passado uma instrução inválida, ou uma instrução tenta acessar um endereço de memória inválido, ou ocorra algum problema em alguma operação matemática, ou ocorra o término do programa nós lançamos as respectivas interrupções e o manipulador de interrupções cuida desses problemas e ao final da execução informa ao usuário qual interrupção aconteceu. Caso tenhamos alguma chamada de sistema, o manipulador de chamadas entra em ação e cuida do resto, podemos ter uma chamada de leitura("*trap input*") ou uma chamada de exibição("*trap output*"), em ambas, quando ocorrerem o manipulador trata e faz o que precisa ser feito.

Além dos problemas desenvolvidos, pensamos também em casos de teste, desenvolvemos oito casos no total para verificar se a solução estava correta, porém iremos falar desses casos de teste nos próximos tópicos.

RODANDO O PROGRAMA

Para executar o sistema e poder realizar os testes, primeiro você precisa localizar a função “*main*” escrita no arquivo “*Sistema.java*”. É por ela que você executará todos os testes criados pelo grupo, essa função é responsável por controlar o sistema, ou seja, no início da função nós criamos uma nova instância do sistema e logo abaixo chamamos todos os seus respectivos testes.

Para rodar cada teste você precisa primeiramente escolher um dos testes que deseja rodar, pois você só pode rodar um teste por vez, após efetuar a escolha do teste, basta apenas descomentar a linha que está chamando ele e comentar a linha do outro teste que estava sendo chamada.

Por exemplo, nessa imagem abaixo nós temos a criação da instância do sistema e o teste 1 descomentado (“*s.test1();*”), isso significa que ao rodarmos o “*main*” ele irá rodar o primeiro teste, caso eu queira trocar o teste para rodar o teste 6, que é o teste de chamada de sistema de exibição(“*trap output*”), eu preciso comentar a linha “*s.test1()*;” e descomentar a linha “*s.test6();*”.

```
public static void main(String args[]) {
    //cria uma nova instancia do sistema
    Sistema s = new Sistema();

    //test1 - programa que testa fibonnaci
    s.test1();

    //test2 - programa que testa progminimo
    //s.test2();

    //teste3 - programa que testa fatorial
    //s.test3();

    //test4 - programa que testa interrupções de endereço invalido
    //s.test4();

    //teste5 - programa que testa manipulador de chamada de sistema(trap 1 - input)
    //s.test5();

    //teste6 - programa que testa manipulador de chamada de sistema(trap 2 - output)
    //s.test6();

    //test7 - programa que testa interrupções de intrução invalida
    //s.test7();

    //teste8 - programa que testa interrupções de overflow de operações matematicas
    //s.test8();
}
```

A função “*main()*” está bem intuitiva, tentamos fazer ela o mais simples e organizada possível para facilitar a compreensão de quem estiver testando o “*Sistema.java*”, deixamos pré escrito cada caso de teste e também deixamos um

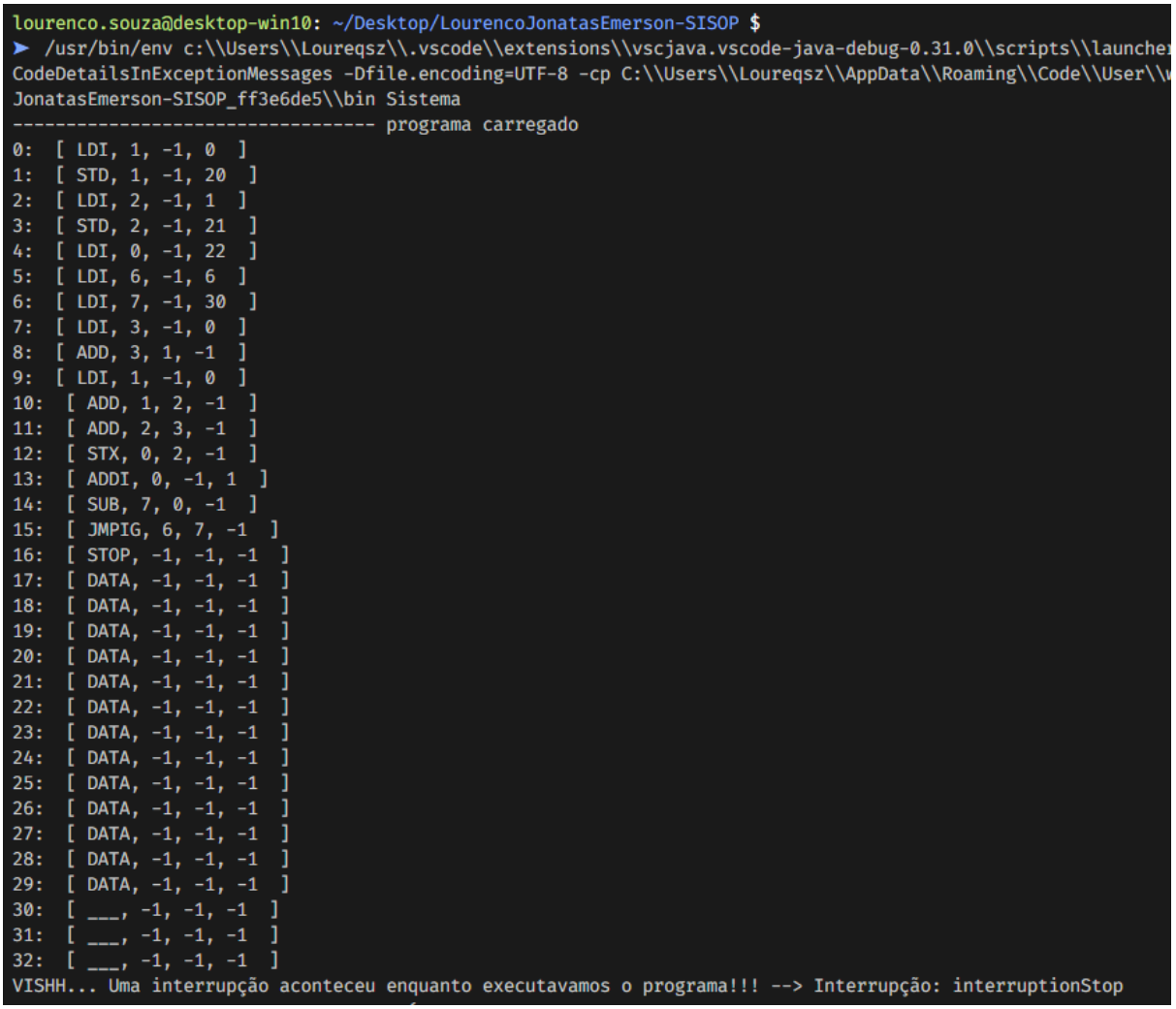
comentário falando o que cada caso de teste realmente executa, ou seja, falando de forma simples, porém redundante, escrevemos o que o teste irá testar.

RESULTADO DOS TESTES

Este tópico é destinado apenas a apresentação dos resultados obtidos na execução de cada um dos testes desenvolvidos, para compreender melhor o código você precisará abrir o "*Sistema.java*" e analisar como montamos cada um dos códigos.

Primeiro Programa de Teste

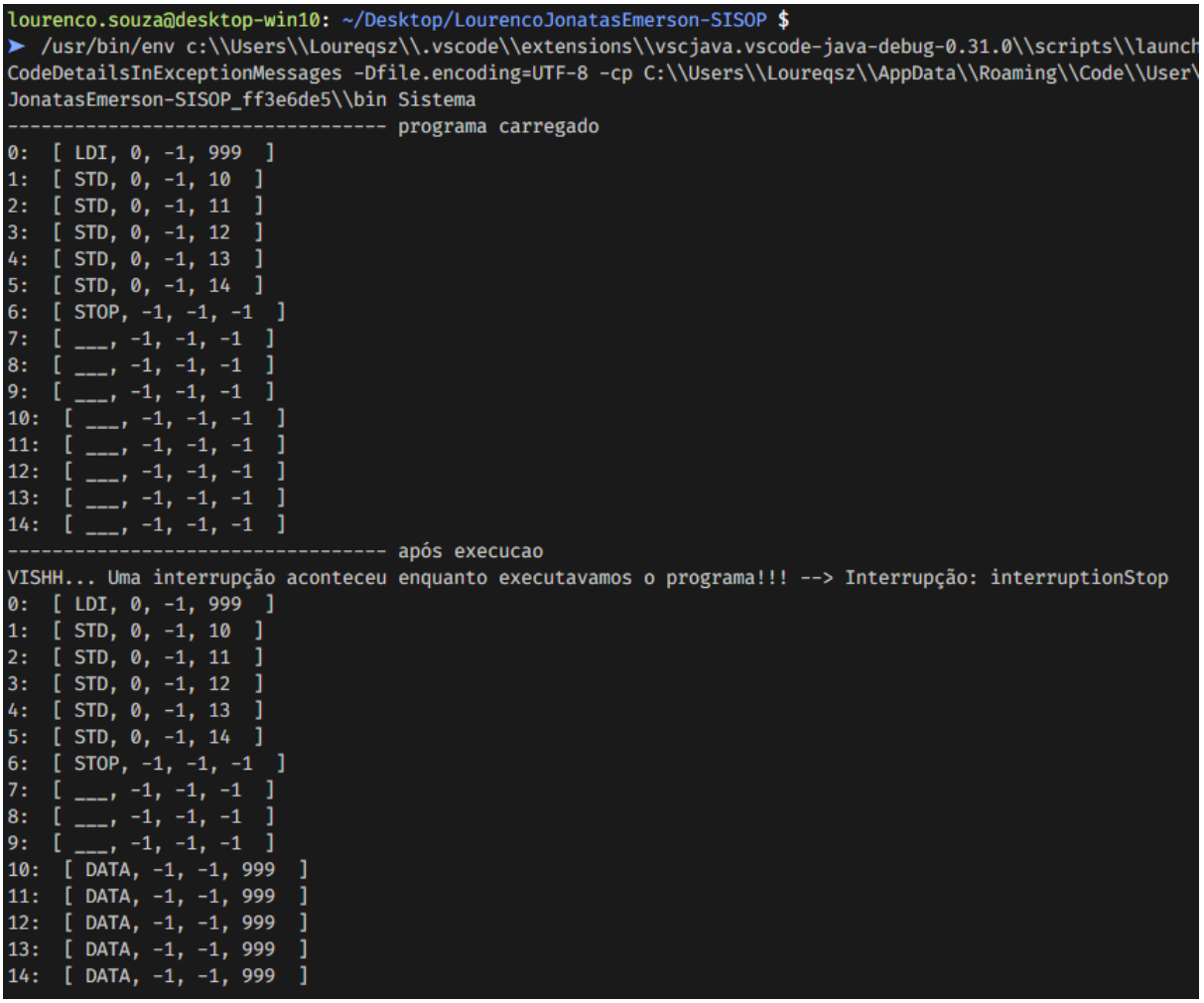
Esse teste consiste na execução do programa de *fibonnaci*, fornecido pelo professor.



```
-------------------------------- após execucao
0:  [ LDI, 1, -1, 0  ]
1:  [ STD, 1, -1, 20  ]
2:  [ LDI, 2, -1, 1  ]
3:  [ STD, 2, -1, 21  ]
4:  [ LDI, 0, -1, 22  ]
5:  [ LDI, 6, -1, 6  ]
6:  [ LDI, 7, -1, 30  ]
7:  [ LDI, 3, -1, 0  ]
8:  [ ADD, 3, 1, -1  ]
9:  [ LDI, 1, -1, 0  ]
10:  [ ADD, 1, 2, -1  ]
11:  [ ADD, 2, 3, -1  ]
12:  [ STX, 0, 2, -1  ]
13:  [ ADDI, 0, -1, 1  ]
14:  [ SUB, 7, 0, -1  ]
15:  [ JMPIG, 6, 7, -1  ]
16:  [ STOP, -1, -1, -1  ]
17:  [ DATA, -1, -1, -1  ]
18:  [ DATA, -1, -1, -1  ]
19:  [ DATA, -1, -1, -1  ]
20:  [ DATA, -1, -1, 0  ]
21:  [ DATA, -1, -1, 1  ]
22:  [ DATA, -1, -1, 1  ]
23:  [ DATA, -1, -1, 2  ]
24:  [ DATA, -1, -1, 3  ]
25:  [ DATA, -1, -1, 5  ]
26:  [ DATA, -1, -1, 8  ]
27:  [ DATA, -1, -1, 13  ]
28:  [ DATA, -1, -1, 21  ]
29:  [ DATA, -1, -1, 34  ]
30:  [ ___, -1, -1, -1  ]
31:  [ ___, -1, -1, -1  ]
32:  [ ___, -1, -1, -1  ]
```

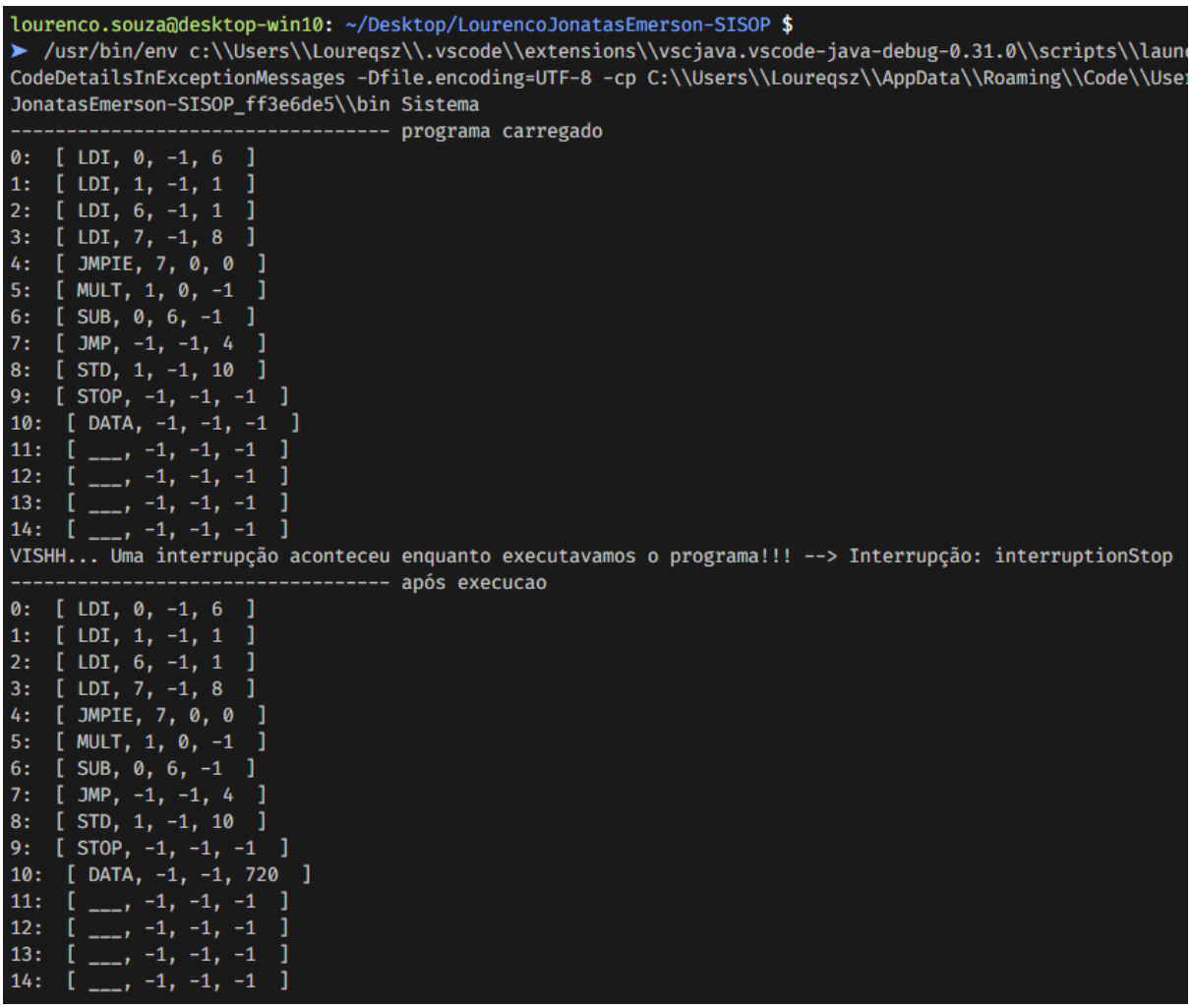
Segundo Programa de Teste

Esse teste consiste na execução do programa *progminimo*, fornecido pelo professor.



Terceiro Programa de Teste

Esse teste consiste na execução do programa *fatorial*, fornecido pelo professor.



Quarto Programa de Teste

Esse teste consiste na execução do programa *testExercicio*, que foi a tentativa de resolver um dos exercícios, porém no final ele serviu para demonstrar uma interrupção de endereço inválido.

```
13:  [ STD, 1, -1, 51  ]
14:  [ STD, 2, -1, 52  ]
15:  [ STD, 3, -1, 53  ]
16:  [ STD, 4, -1, 54  ]
17:  [ STD, 5, -1, 55  ]
18:  [ STD, 6, -1, 56  ]
19:  [ STD, 7, -1, 57  ]
20:  [ LDI, 1, -1, 59  ]
21:  [ STD, 1, -1, 1024  ]
22:  [ STOP, 1, -1, 0  ]
23:  [ ___, -1, -1, -1  ]
24:  [ ___, -1, -1, -1  ]
25:  [ ___, -1, -1, -1  ]
26:  [ ___, -1, -1, -1  ]
27:  [ ___, -1, -1, -1  ]
28:  [ ___, -1, -1, -1  ]
29:  [ ___, -1, -1, -1  ]
--------------------------------- após execucao
VISHH... Uma interrupção aconteceu enquanto executavamos o programa!!! --> Interrupção: interruptionInvalidAddress
0:  [ LDI, 1, -1, 50  ]
1:  [ LDI, 7, -1, 7  ]
2:  [ JMPIG, 7, 1, -1  ]
3:  [ LDI, 7, -1, 69  ]
4:  [ STD, 7, -1, 60  ]
5:  [ STOP, 1, -1, 0  ]
6:  [ LDI, 2, -1, 0  ]
7:  [ ADD, 2, 1, -1  ]
8:  [ LDI, 6, -1, 1  ]
9:  [ SUB, 1, 6, -1  ]
10:  [ LDI, 7, -1, 8  ]
11:  [ JMPIG, 7, 1, -1  ]
12:  [ STD, 0, -1, 50  ]
13:  [ STD, 1, -1, 51  ]
14:  [ STD, 2, -1, 52  ]
15:  [ STD, 3, -1, 53  ]
16:  [ STD, 4, -1, 54  ]
17:  [ STD, 5, -1, 55  ]
18:  [ STD, 6, -1, 56  ]
19:  [ STD, 7, -1, 57  ]
20:  [ LDI, 1, -1, 59  ]
21:  [ STD, 1, -1, 1024  ]
22:  [ STOP, 1, -1, 0  ]
23:  [ ___, -1, -1, -1  ]
24:  [ ___, -1, -1, -1  ]
25:  [ ___, -1, -1, -1  ]
26:  [ ___, -1, -1, -1  ]
```

A execução do teste 4 é muito extensa, como o programa acaba fazendo dump e exibindo as 1024 posições da memória, mostramos apenas o momento que exibe a interrupção para facilitar a exibição no documento, caso tenha curiosidade, para analisar o log completo você precisará rodar o teste 4 na sua máquina local.

Quinto Programa de Teste

Esse teste consiste na execução do programa *testTrapHandlerInput*, que foi desenvolvido para testar a ação do manipulador de chamada de sistema, nesse teste, como o próprio nome já insinua, testamos a leitura(trap input).

```
lourenco.souza@desktop-win10: ~/Desktop/LourencoJonatasEmerson-SISOP $
➤ /usr/bin/env c:\\Users\\Loureqsz\\.vscode\\extensions\\vscjava.vscode-java-debug-0.31.0\\scripts\\launch
CodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 -cp C:\\Users\\Loureqsz\\AppData\\Roaming\\Code\\User\
JonatasEmerson-SISOP_ff3e6de5\\bin Sistema
--------------------------------- programa carregado
0:  [ LDI, 8, -1, 1  ]
1:  [ LDI, 9, -1, 8  ]
2:  [ TRAP, -1, -1, -1  ]
3:  [ STOP, 1, -1, 0  ]
4:  [ ___, -1, -1, -1  ]
5:  [ ___, -1, -1, -1  ]
6:  [ ___, -1, -1, -1  ]
7:  [ ___, -1, -1, -1  ]
8:  [ ___, -1, -1, -1  ]
9:  [ ___, -1, -1, -1  ]
--------------------------------- após execucao
Opa... Uma chamada de sistema ocorreu!!! -->  | 1 | 8 |
 --> Por favor digite um valor, apenas inteiros!!!
113
 --> Valor armazenado na posição: 8 --> Valor armazenado: [ DATA, -1, -1, 113  ]
VISHH... Uma interrupção aconteceu enquanto executavamos o programa!!! --> Interrupção: interruptionStop
0:  [ LDI, 8, -1, 1  ]
1:  [ LDI, 9, -1, 8  ]
2:  [ TRAP, -1, -1, -1  ]
3:  [ STOP, 1, -1, 0  ]
4:  [ ___, -1, -1, -1  ]
5:  [ ___, -1, -1, -1  ]
6:  [ ___, -1, -1, -1  ]
7:  [ ___, -1, -1, -1  ]
8:  [ DATA, -1, -1, 113  ]
9:  [ ___, -1, -1, -1  ]
```

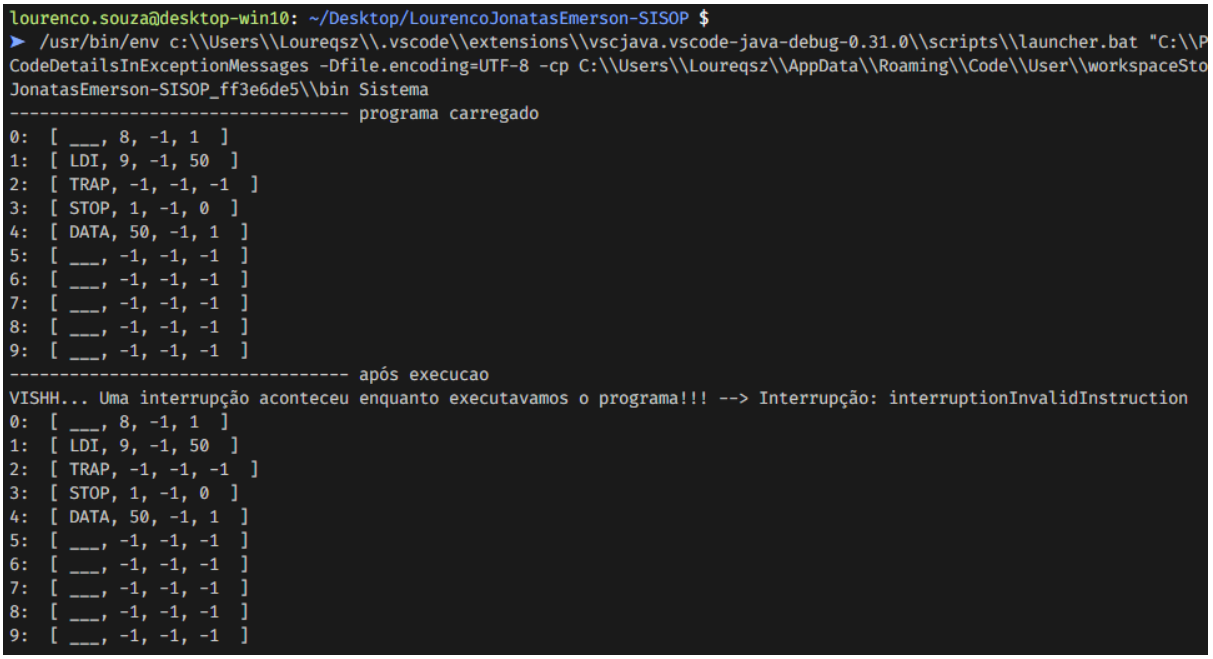
Sexto Programa de Teste

Esse teste consiste na execução do programa *testTrapHandlerOutput*, que foi desenvolvido para testar a ação do manipulador de chamada de sistema, nesse teste, como o próprio nome já insinua, testamos a exibição(trap output).

```
lourenco.souza@desktop-win10: ~/Desktop/LourencoJonatasEmerson-SISOP $
➤ /usr/bin/env c:\\Users\\Loureqsz\\.vscode\\extensions\\vscjava.vscode-java-debug-0.31.0\\scripts\\launc
CodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 -cp C:\\Users\\Loureqsz\\AppData\\Roaming\\Code\\User
JonatasEmerson-SISOP_ff3e6de5\\bin Sistema
--------------------------------- programa carregado
0:  [ LDI, 8, -1, 2  ]
1:  [ LDI, 9, -1, 8  ]
2:  [ STD, 9, -1, 8  ]
3:  [ TRAP, -1, -1, -1  ]
4:  [ STOP, 1, -1, 0  ]
5:  [ ___, -1, -1, -1  ]
6:  [ ___, -1, -1, -1  ]
7:  [ ___, -1, -1, -1  ]
8:  [ ___, -1, -1, -1  ]
9:  [ ___, -1, -1, -1  ]
---------------------------------- após execucao
Opa... Uma chamada de sistema ocorreu!!! -->  | 2 | 8 |
 --> Output do sistema: [ DATA, -1, -1, 8  ]
VISHH... Uma interrupção aconteceu enquanto executavamos o programa!!! --> Interrupção: interruptionStop
0:  [ LDI, 8, -1, 2  ]
1:  [ LDI, 9, -1, 8  ]
2:  [ STD, 9, -1, 8  ]
3:  [ TRAP, -1, -1, -1  ]
4:  [ STOP, 1, -1, 0  ]
5:  [ ___, -1, -1, -1  ]
6:  [ ___, -1, -1, -1  ]
7:  [ ___, -1, -1, -1  ]
8:  [ DATA, -1, -1, 8  ]
9:  [ ___, -1, -1, -1  ]
```

Sétimo Programa de Teste

Esse teste consiste na execução do programa *testInstructionsInvalid*, que foi desenvolvido para testar o manipulador de interrupções, nesse teste nós testamos uma interrupção de instrução inválida.



Oitavo Programa de Teste

Esse teste consiste na execução do programa *testInstructionsOverflow*, que foi desenvolvido para testar o manipulador de interrupções, nesse teste nós testamos uma interrupção de overflow de operação matemática.