

Sistemas Operacionais

Vitor Hugo Garcez, João Pedro Martins, Alberto Rocha

1. Implementação:

Para compilar e executar o projeto basta instalar um pacote JDK em sua máquina, e por meio de uma IDE (de sua preferência) executar o programa.

2. Programas:

Os programas foram feitos dentro da classe **Programas** e são executados através da classe **Computer** utilizando o método **main**. Esse método chama métodos separados dentro da própria classe, para que as saídas no terminal tenham informações importantes sobre suas execuções, por exemplo: quais instruções foram alocadas na memória, valores inseridos nelas e a quantidade de espaços na memória que mostraremos de uma vez só.

2.1 Bubble Sort:

Para executa-lo basta remover as barras duplas na linha 17 da classe computer.

```
----- programa carregado
0: [ LDI, 0, -1, 15 ]
1: [ STD, 0, -1, 60 ]
2: [ LDI, 0, -1, 7 ]
3: [ STD, 0, -1, 61 ]
4: [ LDI, 0, -1, 3 ]
5: [ STD, 0, -1, 62 ]
6: [ LDI, 0, -1, 9 ]
7: [ STD, 0, -1, 63 ]
8: [ LDI, 0, -1, 1 ]
9: [ STD, 0, -1, 64 ]
10: [ LDI, 0, -1, 60 ]
11: [ STD, 0, -1, 50 ]
12: [ STD, 0, -1, 51 ]
13: [ STD, 0, -1, 52 ]
14: [ LDI, 0, -1, 61 ]
15: [ STD, 0, -1, 53 ]
16: [ LDI, 0, -1, 65 ]
17: [ STD, 0, -1, 54 ]
18: [ LDD, 0, -1, 51 ]
19: [ LDD, 1, -1, 52 ]
20: [ LDD, 2, -1, 53 ]
21: [ LDX, 3, 1, -1 ]
22: [ LDX, 4, 2, -1 ]
23: [ SUB, 3, 4, -1 ]
24: [ LDI, 4, -1, 31 ]
25: [ JMPIL, 4, 3, -1 ]
26: [ LDX, 3, 1, -1 ]
27: [ LDX, 4, 2, -1 ]
28: [ SWAP, 3, 4, -1 ]
29: [ STX, 1, 3, -1 ]
30: [ STX, 2, 4, -1 ]
31: [ ADDI, 1, -1, 1 ]
32: [ ADDI, 2, -1, 1 ]
33: [ LDD, 3, -1, 54 ]
34: [ SUB, 3, 2, -1 ]
35: [ LDI, 4, -1, 21 ]
37: [ ADDI, 0, -1, 1 ]
38: [ LDD, 3, -1, 54 ]
39: [ SUB, 3, 0, -1 ]
40: [ LDI, 4, -1, 19 ]
41: [ JMPIL, 4, 3, -1 ]
42: [ STOP, -1, -1, -1 ]
43: [ NULL, 0, 0, 0 ]
44: [ NULL, 0, 0, 0 ]
45: [ NULL, 0, 0, 0 ]
46: [ NULL, 0, 0, 0 ]
47: [ NULL, 0, 0, 0 ]
48: [ NULL, 0, 0, 0 ]
49: [ NULL, 0, 0, 0 ]
50: [ NULL, 0, 0, 0 ]
51: [ NULL, 0, 0, 0 ]
52: [ NULL, 0, 0, 0 ]
53: [ NULL, 0, 0, 0 ]
54: [ NULL, 0, 0, 0 ]
55: [ NULL, 0, 0, 0 ]
56: [ NULL, 0, 0, 0 ]
57: [ NULL, 0, 0, 0 ]
58: [ NULL, 0, 0, 0 ]
59: [ NULL, 0, 0, 0 ]
60: [ NULL, 0, 0, 0 ]
61: [ NULL, 0, 0, 0 ]
62: [ NULL, 0, 0, 0 ]
63: [ NULL, 0, 0, 0 ]
64: [ NULL, 0, 0, 0 ]
```

Fig 1: programa carregado

```

----- após execucao
vm.interrupts.list.StopInterrupt happen due to STOP word called
System must halt? YES
0: [ LDI, 0, -1, 15 ]
1: [ STD, 0, -1, 60 ]
2: [ LDI, 0, -1, 7 ]
3: [ STD, 0, -1, 61 ]
4: [ LDI, 0, -1, 3 ]
5: [ STD, 0, -1, 62 ]
6: [ LDI, 0, -1, 9 ]
7: [ STD, 0, -1, 63 ]
8: [ LDI, 0, -1, 1 ]
9: [ STD, 0, -1, 64 ]
10: [ LDI, 0, -1, 60 ]
11: [ STD, 0, -1, 50 ]
12: [ STD, 0, -1, 51 ]
13: [ STD, 0, -1, 52 ]
14: [ LDI, 0, -1, 61 ]
15: [ STD, 0, -1, 53 ]
16: [ LDI, 0, -1, 65 ]
17: [ STD, 0, -1, 54 ]
18: [ LDD, 0, -1, 51 ]
19: [ LDD, 1, -1, 52 ]
20: [ LDD, 2, -1, 53 ]
21: [ LDX, 3, 1, -1 ]
22: [ LDX, 4, 2, -1 ]
23: [ SUB, 3, 4, -1 ]
24: [ LDI, 4, -1, 31 ]
25: [ JMPIL, 4, 3, -1 ]
26: [ LDX, 3, 1, -1 ]
27: [ LDX, 4, 2, -1 ]
28: [ SWAP, 3, 4, -1 ]
29: [ STX, 1, 3, -1 ]
30: [ STX, 2, 4, -1 ]
31: [ ADDI, 1, -1, 1 ]
32: [ ADDI, 2, -1, 1 ]
33: [ LDD, 3, -1, 54 ]
34: [ SUB, 3, 2, -1 ]
35: [ SUB, 3, 2, -1 ]
36: [ LDI, 4, -1, 21 ]
37: [ JMPIG, 4, 3, -1 ]
38: [ ADDI, 0, -1, 1 ]
39: [ LDD, 3, -1, 54 ]
40: [ SUB, 3, 0, -1 ]
41: [ LDI, 4, -1, 19 ]
42: [ JMPIG, 4, 3, -1 ]
43: [ STOP, -1, -1, -1 ]
44: [ NULL, 0, 0, 0 ]
45: [ NULL, 0, 0, 0 ]
46: [ NULL, 0, 0, 0 ]
47: [ NULL, 0, 0, 0 ]
48: [ NULL, 0, 0, 0 ]
49: [ NULL, 0, 0, 0 ]
50: [ DATA, -1, -1, 60 ]
51: [ DATA, -1, -1, 60 ]
52: [ DATA, -1, -1, 60 ]
53: [ DATA, -1, -1, 61 ]
54: [ DATA, -1, -1, 65 ]
55: [ NULL, 0, 0, 0 ]
56: [ NULL, 0, 0, 0 ]
57: [ NULL, 0, 0, 0 ]
58: [ NULL, 0, 0, 0 ]
59: [ NULL, 0, 0, 0 ]
60: [ DATA, -1, -1, 1 ]
61: [ DATA, -1, -1, 3 ]
62: [ DATA, -1, -1, 7 ]
63: [ DATA, -1, -1, 9 ]
64: [ DATA, -1, -1, 15 ]

```

Figura 2: Programa pós execução

2.2 Fatorial

Para executa-lo basta remover as barras duplas na linha 16 da classe computer.

```

----- programa carregado
0: [ LDI, 0, -1, 5 ]
1: [ STD, 0, -1, 29 ]
2: [ LDD, 0, -1, 29 ]
3: [ LDI, 1, -1, 18 ]
4: [ JMPIL, 1, 1, -1 ]
5: [ STD, 0, -1, 30 ]
6: [ LDI, 4, -1, 30 ]
7: [ LDX, 1, 4, -1 ]
8: [ SUBI, 1, -1, 1 ]
9: [ LDI, 3, -1, 10 ]
10: [ MULT, 0, 1, -1 ]
11: [ SUBI, 1, -1, 1 ]
12: [ JMPIG, 3, 1, -1 ]
13: [ LDI, 3, -1, 18 ]
14: [ STD, 0, -1, 23 ]
15: [ JMPIE, 3, 2, -1 ]
16: [ LDI, 0, -1, -1 ]
17: [ STD, 0, -1, 23 ]
18: [ STOP, -1, -1, -1 ]
19: [ NULL, 0, 0, 0 ]
20: [ NULL, 0, 0, 0 ]
21: [ NULL, 0, 0, 0 ]
22: [ NULL, 0, 0, 0 ]
23: [ NULL, 0, 0, 0 ]

```

Figura 3: Programa Carregado Fatorial.

```

----- após execucao
vm.interrupts.list.StopInterrupt happen due to STOP word called
System must halt? YES
0: [ LDI, 0, -1, 5 ]
1: [ STD, 0, -1, 29 ]
2: [ LDD, 0, -1, 29 ]
3: [ LDI, 1, -1, 18 ]
4: [ JMPIL, 1, 1, -1 ]
5: [ STD, 0, -1, 30 ]
6: [ LDI, 4, -1, 30 ]
7: [ LDX, 1, 4, -1 ]
8: [ SUBI, 1, -1, 1 ]
9: [ LDI, 3, -1, 10 ]
10: [ MULT, 0, 1, -1 ]
11: [ SUBI, 1, -1, 1 ]
12: [ JMPIG, 3, 1, -1 ]
13: [ LDI, 3, -1, 18 ]
14: [ STD, 0, -1, 23 ]
15: [ JMPIE, 3, 2, -1 ]
16: [ LDI, 0, -1, -1 ]
17: [ STD, 0, -1, 23 ]
18: [ STOP, -1, -1, -1 ]
19: [ NULL, 0, 0, 0 ]
20: [ NULL, 0, 0, 0 ]
21: [ NULL, 0, 0, 0 ]
22: [ NULL, 0, 0, 0 ]
23: [ DATA, -1, -1, 120 ]

```

Figura 4: Fatorial Pós execução

2.3 Fibonacci

Para executa-lo basta remover as barras duplas na linha 15 da classe computer.

```

----- programa carregado
0: [ LDI, 1, -1, 10 ]
1: [ STD, 1, -1, 50 ]
2: [ LDD, 7, -1, 50 ]
3: [ LDI, 0, -1, 52 ]
4: [ LDI, 4, -1, -1 ]
5: [ LDI, 2, -1, 27 ]
6: [ JMPIL, 2, 1, -1 ]
7: [ LDI, 1, -1, 0 ]
8: [ LDI, 2, -1, 1 ]
9: [ LDI, 6, -1, 28 ]
10: [ STD, 1, -1, 50 ]
11: [ SUBI, 7, -1, 1 ]
12: [ JMPIE, 6, 7, -1 ]
13: [ STD, 2, -1, 51 ]
14: [ SUBI, 7, -1, 1 ]
15: [ JMPIE, 6, 7, -1 ]
16: [ LDI, 4, -1, 16 ]
17: [ LDI, 3, -1, 0 ]
18: [ ADD, 3, 1, -1 ]
19: [ LDI, 1, -1, 0 ]
20: [ ADD, 1, 2, -1 ]
21: [ ADD, 2, 3, -1 ]
22: [ STX, 0, 2, -1 ]
23: [ ADDI, 0, -1, 1 ]
24: [ SUBI, 7, -1, 1 ]
25: [ JMPIG, 4, 7, -1 ]
26: [ JMPIE, 6, 7, -1 ]
27: [ STD, 4, -1, 50 ]
28: [ STOP, -1, -1, -1 ]

```

Figura 5: Fibonacci Carregado em memoria

```

----- após execucao
vm.interrupts.list.StopInterrupt happen due to STOP word called
System must halt? YES
0: [ LDI, 1, -1, 10 ]
1: [ STD, 1, -1, 50 ]
2: [ LDD, 7, -1, 50 ]
3: [ LDI, 0, -1, 52 ]
4: [ LDI, 4, -1, -1 ]
5: [ LDI, 2, -1, 27 ]
6: [ JMPIL, 2, 1, -1 ]
7: [ LDI, 1, -1, 0 ]
8: [ LDI, 2, -1, 1 ]
9: [ LDI, 6, -1, 28 ]
10: [ STD, 1, -1, 50 ]
11: [ SUBI, 7, -1, 1 ]
12: [ JMPIE, 6, 7, -1 ]
13: [ STD, 2, -1, 51 ]
14: [ SUBI, 7, -1, 1 ]
15: [ JMPIE, 6, 7, -1 ]
16: [ LDI, 4, -1, 16 ]
17: [ LDI, 3, -1, 0 ]
18: [ ADD, 3, 1, -1 ]
19: [ LDI, 1, -1, 0 ]
20: [ ADD, 1, 2, -1 ]
21: [ ADD, 2, 3, -1 ]
22: [ STX, 0, 2, -1 ]
23: [ ADDI, 0, -1, 1 ]
24: [ SUBI, 7, -1, 1 ]
25: [ JMPIG, 4, 7, -1 ]
26: [ JMPIE, 6, 7, -1 ]
27: [ STD, 4, -1, 50 ]
28: [ STOP, -1, -1, -1 ]

29: [ NULL, 0, 0, 0 ]
30: [ NULL, 0, 0, 0 ]
31: [ NULL, 0, 0, 0 ]
32: [ NULL, 0, 0, 0 ]
33: [ NULL, 0, 0, 0 ]
34: [ NULL, 0, 0, 0 ]
35: [ NULL, 0, 0, 0 ]
36: [ NULL, 0, 0, 0 ]
37: [ NULL, 0, 0, 0 ]
38: [ NULL, 0, 0, 0 ]
39: [ NULL, 0, 0, 0 ]
40: [ NULL, 0, 0, 0 ]
41: [ NULL, 0, 0, 0 ]
42: [ NULL, 0, 0, 0 ]
43: [ NULL, 0, 0, 0 ]
44: [ NULL, 0, 0, 0 ]
45: [ NULL, 0, 0, 0 ]
46: [ NULL, 0, 0, 0 ]
47: [ NULL, 0, 0, 0 ]
48: [ NULL, 0, 0, 0 ]
49: [ NULL, 0, 0, 0 ]
50: [ DATA, -1, -1, 0 ]
51: [ DATA, -1, -1, 1 ]
52: [ DATA, -1, -1, 1 ]
53: [ DATA, -1, -1, 2 ]
54: [ DATA, -1, -1, 3 ]
55: [ DATA, -1, -1, 5 ]
56: [ DATA, -1, -1, 8 ]
57: [ DATA, -1, -1, 13 ]
58: [ DATA, -1, -1, 21 ]
59: [ DATA, -1, -1, 34 ]

```

Figura 6: Fibonacci pós execução

2.4 teste de chamada de sistema

Para executa-lo basta remover as barras duplas na linha 18 da classe computer.

```

----- programa carregado
0: [ LDI, 7, -1, 1 ]
1: [ TRAP, 7, 8, -1 ]
2: [ LDI, 7, -1, 2 ]
3: [ TRAP, 7, 8, -1 ]
4: [ NULL, 0, 0, 0 ]
5: [ NULL, 0, 0, 0 ]
6: [ NULL, 0, 0, 0 ]
7: [ NULL, 0, 0, 0 ]
8: [ NULL, 0, 0, 0 ]
9: [ NULL, 0, 0, 0 ]
10: [ NULL, 0, 0, 0 ]
11: [ NULL, 0, 0, 0 ]
12: [ NULL, 0, 0, 0 ]
13: [ NULL, 0, 0, 0 ]
14: [ NULL, 0, 0, 0 ]

```

Figura 7: Teste de chamada de sistema carregado em memoria

```

----- após execucao
Insira um numero para preencher o registrador:
3554
R9 content: 3554
vm.interruptions.list.InvalidRuleInterruption happen due to Should not call word instruction of type: NULL
System must halt? YES
0: [ LDI, 7, -1, 1 ]
1: [ TRAP, 7, 8, -1 ]
2: [ LDI, 7, -1, 2 ]
3: [ TRAP, 7, 8, -1 ]
4: [ NULL, 0, 0, 0 ]
5: [ NULL, 0, 0, 0 ]
6: [ NULL, 0, 0, 0 ]
7: [ NULL, 0, 0, 0 ]
8: [ NULL, 0, 0, 0 ]
9: [ NULL, 0, 0, 0 ]
10: [ NULL, 0, 0, 0 ]
11: [ NULL, 0, 0, 0 ]
12: [ NULL, 0, 0, 0 ]
13: [ NULL, 0, 0, 0 ]
14: [ NULL, 0, 0, 0 ]
PS D:\Drive\Curso\Semestre4\SO\ICPU\T1\VmOs>

```

Figura 8: Teste de chamada de sistema pós execução

2.5 teste de interrupção de sistema

Para executa-lo basta remover as barras duplas na linha 19 da classe computer.

```

----- programa carregado
0: [ LDD, 0, -1, 2000 ]
1: [ LDD, 0, -1, -5 ]
2: [ STOP, -1, -1, -1 ]
3: [ NULL, 0, 0, 0 ]
4: [ NULL, 0, 0, 0 ]
5: [ NULL, 0, 0, 0 ]
6: [ NULL, 0, 0, 0 ]
7: [ NULL, 0, 0, 0 ]
8: [ NULL, 0, 0, 0 ]
9: [ NULL, 0, 0, 0 ]
10: [ NULL, 0, 0, 0 ]
11: [ NULL, 0, 0, 0 ]
12: [ NULL, 0, 0, 0 ]
13: [ NULL, 0, 0, 0 ]
14: [ NULL, 0, 0, 0 ]

```

Figura 9: teste de Interrupção de sistema carregado em memoria

```

----- após execucao
vm.interruptions.list.MemoryOutOfBoundsInterruption happen due to Memory has 1024 positions, and counter tried to get position: 2000
System must halt? YES
0: [ LDD, 0, -1, 2000 ]
1: [ LDD, 0, -1, -5 ]
2: [ STOP, -1, -1, -1 ]
3: [ NULL, 0, 0, 0 ]
4: [ NULL, 0, 0, 0 ]
5: [ NULL, 0, 0, 0 ]
6: [ NULL, 0, 0, 0 ]
7: [ NULL, 0, 0, 0 ]
8: [ NULL, 0, 0, 0 ]
9: [ NULL, 0, 0, 0 ]
10: [ NULL, 0, 0, 0 ]
11: [ NULL, 0, 0, 0 ]
12: [ NULL, 0, 0, 0 ]
13: [ NULL, 0, 0, 0 ]
14: [ NULL, 0, 0, 0 ]

```

Figura 10: teste de interrupção de sistema pós execução

3. Considerações parciais

O trabalho foi feito utilizando a linguagem de programação Java, para fins de estudos estávamos discutindo o desenvolvimento usando a linguagem GO. Entretanto como as horas de trabalhos estavam puxando muito tempo dos

integrantes do grupo optamos por deixar o código base como estava, apenas remodelando-o para aplicar a engenharia de software dentro da VM.

O trabalho no geral, em certa parte, foi bem divertido. Conseguimos aprender o funcionamento do computador Viking que permite programarmos em Assembly.