UNIVERSIDADE ESTADUAL DO PARANÁ - *CAMPUS* APUCARANA

JOÃO PEDRO DE SOUZA OLIVO TARDIVO

DOCUMENTAÇÃO SOBRE O SIMULADOR DE CICLOS DE INSTRUÇÕES

APUCARANA – PR
2023

JOÃO PEDRO DE SOUZA OLIVO TARDIVO

DOCUMENTAÇÃO SOBRE O SIMULADOR DE CICLOS DE INSTRUÇÕES

Trabalho apresentado à disciplina de Arquitetura e Organização de Computadores, do curso de Bacharelado em Ciência da Computação.

Professor: Guilherme Nakahata

APUCARANA – PR
2023

SUMÁRIO

INSTRUÇÕES DE EXECUÇÃO

Windows

Execute *CPU_instruction_simulator.exe*

Linux

Abra o terminal na pasta que contém o *CPU_instruction_simulator*

Conceda permissão de execução para o arquivo através do comando

```
chmod +x CPU_instruction_simulator
```

Execute o programa através do comando

```
./CPU_instruction_simulator
```

INSTRUÇÕES DE COMPILAÇÃO

Windows

Instale o Python encontrado em:

https://www.python.org/downloads/

Abra o terminal para instalar as dependências:

```
pip install PyQt6 PyInstaller
```

Abra o terminal na pasta com o código fonte

Utilize o comando para gerar o executável na pasta dist

```
pyInstaller main_window.py --onefile --noconsole
--icon=logo.ico --add-data "resources;resources"
```

Após isso siga as instruções de execução

Linux

Abra o terminal e instale o Python

Ubuntu/Debian

```
sudo apt install python3
```

Fedora

```
sudo dnf install python3
```

CentOS

```
sudo yum install centos-release-scl
sudo yum install rh-python36
scl enable rh-python36 bash
```

Arch

```
sudo pacman -S python
```

Instale o Package Installer for Python (pip)

Ubuntu/Debian

```
sudo apt install python3-pip
```

Fedora

```
sudo dnf install python3-pip
```

CentOS

```
sudo yum install python3-pip
```

Arch

```
sudo pacman -S python-pip
```

Ou utilizando o próprio Python

```
python3 get-pip.py
```

Instale as dependências:

```
sudo pip3 install pyinstaller pyqt6
```

Abra o terminal na pasta com o código fonte

Utilize o comando para gerar o arquivo binário na pasta dist

```
python3 -m PyInstaller main_window.py --onefile --noconsole
--icon=logo.ico --add-data "resources:resources"
```

Após isso siga as instruções de execução

DOCUMENTAÇÃO

main_window.py

- **Primeiramente os módulos e classes necessários são importados:**

```
import os
import sys
from PyQt6.QtCore import Qt, QEvent
from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QLabel, QPushButton, QVBoxLayout, QHBoxLayout, QStackedLayout, QMessageBox
from PyQt6.QtGui import QGuiApplication, QIcon
from manual_input_screen import ManualInputScreen
from file_input_screen import FileInputScreen
```

os e **sys**: Usado para manipulação de caminhos de arquivos e operações relacionadas ao sistema.

PyQt6.QtCore, **PyQt6.QtWidgets**, **PyQt6.QtGui**: Módulos do PyQt6 para criação de GUI.

QEvent: Uma classe do PyQt6 usada para lidar com eventos.

QApplication, **QMainWindow**, **QWidget**, **QLabel**, **QPushButton**, **QVBoxLayout**, **QHBoxLayout**, **QStackedLayout**, **QMessageBox**: Classes PyQt6 para criação de elementos GUI, Application gerencia a instanciação ou execução da aplicação da além de ser utilizada dentro da própria MainWindow para obter as dimensões da tela do usuário, MainWindow para definir a janela principal, Widgets são estruturas genéricas do Qt estilo divs do HTML, labels são pequenos textos de títulos ou nomes, button é um botão, VBox é o layout vertical, HBox horizontal e Stacked é o layout em pilha, finalmente MessageBox é uma caixa de pop up utilizada para alertas.

QGuiApplication, **QIcon**: Classes PyQt6 para aplicações GUI e gerenciamento de ícones.

ManualInputScreen, **FileInputScreen**: Classes personalizadas de módulos externos usados como parte da aplicação, elas direcionam o usuário para as telas de entrada manual ou por arquivo e são dinamicamente adicionadas ou retiradas do layout em pilha caso estão em foco ou não.

Nota-se que devido a estrutura de layouts e widgets do PyQt, muitas vezes temos que criar widgets que estão contidos em layouts e posteriormente criamos widgets contêineres para segurar esses layouts para serem incluídos em outros layouts de hierarquia maior.

Isso inicialmente pode parecer bem confuso, mas é a maneira de conseguir resultados mais previsíveis e definidos para a estruturação e posicionamento de cada elemento de uma aplicação.

Novamente remete-se a analogia com o HTML que também segue uma grande estrutura hierárquica de vários componentes para popular o conteúdo de uma página.

- **Manipulação de caminhos do PyInstaller:**

```
## PyInstaller file path handler
if getattr(sys, 'frozen', False):
    # Running as a PyInstaller executable
    base_path = sys._MEIPASS
else:
    # Running as a script
    base_path = os.path.abspath(".")
```

Verifica se o código está sendo executado como um executável PyInstaller ou como um script. Define o base_path de acordo para lidar com caminhos de arquivo, isso previne possíveis erros na execução do arquivo buildado.

- **Criando a janela principal do aplicativo (classe MainWindow):**

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.input_widget = None

        self.setWindowTitle("Simulador de Instrucoes")
        self.setWindowIcon(QIcon(os.path.join(base_path, 'resources', 'logo-unespar.jpg')))
        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)

        ## User screen's dimenions
        self.screen = QGuiApplication.primaryScreen()
        self.screen_size = self.screen.availableSize()

        ## Master Layout
        self.main_layout = QVBoxLayout(self.central_widget)
        self.main_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

É uma subclasse do módulo QMainWindow do PyQt6, ou seja, essa será nossa janela “mestre” onde o layout em pilha meramente vai fazer a adição, remoção e troca do elemento ativo que será mostrado ao usuário.

Definimos o título e o ícone da janela.

Define o widget central da janela principal.

Recuperamos as dimensões da tela para ajustes de layout.

Criamos o layout principal como um layout vertical (QVBoxLayout) e definimos seu alinhamento.

- **Cabeçalho da aplicação:**

```
37        ## Application Header
38        title_label = QLabel("<h1>Simulador de Instrucoes</h1>")
39        title_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
40        self.main_layout.addWidget(title_label)
```

Criamos um rótulo de título e o adicionamos ao layout principal.

Definimos o alinhamento do rótulo do título para o centro.

- **Layout em pilha:**

```
43        ## Stacked Layout
44        self.stacked_layout_container = QWidget()
45        self.stacked_layout = QStackedLayout()
46        self.stacked_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

Configuramos um widget de contêiner (stacked_layout_container) e um layout em pilha (stacked_layout) para gerenciar múltiplas visualizações.

Garantimos que o alinhamento do layout empilhado esteja centralizado.

- **Primeira tela da pilha: Boas-vindas:**

```
49        ## First stacked view: Welcome Screen
50        self.welcome_layout_container = QWidget()
51        self.welcome_layout_container.setMinimumWidth(int(self.screen_size.width() * 0.70))
52        self.welcome_layout_container.setMinimumHeight(int(self.screen_size.height() * 0.70))
53
54        self.welcome_layout = QVBoxLayout()
55        self.welcome_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
56
57
```

```
58          welcome_label = QLabel("<h2>Bem vindo!</h2>")
59          welcome_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
60          self.welcome_layout.addWidget(welcome_label)
61
62          self.welcome_buttons_container = QWidget()
63          self.welcome_buttons_layout = QHBoxLayout()
64
65          self.button_script_manual = QPushButton("Manual")
66          self.button_script_manual.clicked.connect(self.show_manual_input_screen)
67          self.welcome_buttons_layout.addWidget(self.button_script_manual)
68          self.button_script_arquivo = QPushButton("Arquivo")
69          self.button_script_arquivo.clicked.connect(self.show_file_input_screen)
70          self.welcome_buttons_layout.addWidget(self.button_script_arquivo)
71
72          self.welcome_buttons_container.setLayout(self.welcome_buttons_layout)
73          self.welcome_layout.addWidget(self.welcome_buttons_container)
74
75          self.welcome_layout_container.setLayout(self.welcome_layout)
```

Criamos um widget de contêiner (welcome_layout_container) com dimensões mínimas.

Configuramos um layout vertical (welcome_layout) para esta tela e garantimos que seu alinhamento esteja centralizado.

Criamos um rótulo de boas-vindas e o adicionamos ao layout de boas-vindas.

Criamos botões para telas de entrada manual e de arquivos e os conecta às suas respectivas funções.

Adicionamos os botões a um layout horizontal (welcome_buttons_layout) dentro de um contêiner (welcome_buttons_container).

Adicionamos o contêiner ao layout de boas-vindas.

Definimos o layout de boas-vindas para o contêiner de layout de boas-vindas.

Adicionamos o contêiner de layout de boas-vindas ao layout em pilha.

- **Finalizando o layout em pilha:**

```
78          ## Finalizing the stacked layout
79          self.stacked_layout.addWidget(self.welcome_layout_container)
80
81          self.stacked_layout_container.setLayout(self.stacked_layout)
82
83          self.main_layout.addWidget(self.stacked_layout_container)
```

Definimos o layout empilhado para o contêiner de layout empilhado.

Adicionamos o contêiner de layout empilhado ao layout principal.

- **Personalização de layout:**

```
85          self.main_layout.setSpacing(20)
86          self.main_layout.setContentsMargins(30, 30, 30, 30)
87
88          self.showMaximized()
```

Definimos espaçamento e margens para o layout principal.

Definimos que a aplicação será inicializada de forma maximizada.

- **Funções dessa classe principal:**

```
90      def changeEvent(self, event):
91          if event.type() == QEvent.Type.WindowStateChange:
92              if self.windowState() & Qt.WindowState.WindowMaximized:
93                  self.isMaximized = True
94              else:
95                  if self.isMaximized:
96                      self.center_on_screen()
97                  self.isMaximized = False
```

Definimos uma função changeEvent para lidar com a alteração no estado da janela de maximizado para janela.

```
 99      def center_on_screen(self):
100          screen_geometry = QApplication.primaryScreen().availableGeometry()
101
102          center_x = int((screen_geometry.width() - self.width()) / 2)
103          center_y = int((screen_geometry.height() - self.height()) / 2.5)
104
105          self.move(center_x, center_y)
```

Definimos uma função center_on_screen para centralizar a janela na tela calculando a diferença entre o espaço total da tela do usuário e o ocupado pela janela da aplicação.

```
107     def show_alert_box(self, title, text):
108         alert=QMessageBox()
109         alert.setIcon(QMessageBox.Icon.Information)
110         alert.setWindowIcon(QIcon(os.path.join(base_path, 'resources', 'logo-unespar.jpg')))
111         alert.setWindowTitle(title)
112         alert.setText(text)
113         alert.setStandardButtons(QMessageBox.StandardButton.Ok)
114         alert.exec()
```

Definimos uma função show_alert_box para exibir uma caixa de mensagem informativa com título e texto que são passados como argumentos, desta forma toda vez que precisarmos de uma mensagem customizada de alerta na aplicação podemos invocar esta função.

```
116     def show_welcome_screen(self):
117         self.stacked_layout.setCurrentWidget(self.welcome_layout_container)
118         if not self.isMaximized:
119             self.center_on_screen()
120         self.destroy_input_widget()
121 
122     def show_manual_input_screen(self):
123         self.input_widget = ManualInputScreen(self.show_welcome_screen, self.show_alert_box, self.center_on_screen)
124         self.stacked_layout.addWidget(self.input_widget)
125         self.stacked_layout.setCurrentWidget(self.input_widget)
126 
127     def show_file_input_screen(self):
128         self.input_widget = FileInputScreen(self.show_welcome_screen, self.show_alert_box, self.center_on_screen)
129         self.stacked_layout.addWidget(self.input_widget)
130         self.stacked_layout.setCurrentWidget(self.input_widget)
131 
132     def destroy_input_widget(self):
133         if self.input_widget:
134             self.input_widget.deleteLater()
135             self.stacked_layout.removeWidget(self.input_widget)
136             self.input_widget.deleteLater()
137             self.input_widget = None
```

Definimos funções para mostrar a tela de boas-vindas e passar para as telas de entrada manual e de arquivos.

Definimos uma função destroy_input_widget para remover e excluir o widget de entrada atual ao alternar entre telas, poupando memória.

- **Finalmente, o bloco if __name__ == "__main__":**

```
139 if __name__ == "__main__":
140     app = QApplication(sys.argv)
141     window = MainWindow()
142     sys.exit(app.exec())
```

Inicializa a aplicação PyQt6 (app).

Cria uma instância da classe MainWindow (janela).

Inicia o loop de eventos da aplicação com app.exec().

- **Em resumo, main_window.py:**

Cria um aplicativo GUI com um layout empilhado que alterna entre uma tela de boas-vindas, uma tela de entrada manual e uma tela de entrada de

arquivo. Ele também lida com alterações de estado da janela e fornece funções para exibir mensagens de alerta e centralizar a janela na tela.

manual_input_screen.py

- **Primeiramente os módulos e classes necessários são importados:**

```
1  from PyQt6.QtCore import Qt, QTimer
2  from PyQt6.QtWidgets import QWidget, QPushButton, QVBoxLayout, QLabel, QHBoxLayout, QLineEdit, QScrollArea, QComboBox, QGridLayout
3  from instruction import Instruction
4  from instruction_list_processor import run_instructions
```

PyQt6.QtCore, PyQt6.QtWidgets: Módulos do PyQt6 para criação de GUI.

QTimer: Uma classe do PyQt6 usada para lidar com a contagem de tempo.

QWidget, QPushButton, QVBoxLayout, QLabel, QHBoxLayout, QLineEdit, QScrollArea, QComboBox, QGridLayout: Classes PyQt6 para criação de elementos GUI, similar a explicação de main_window.py, as únicas novidades aqui são a ScrollArea que como o nome indica cria uma área de espaço definido que é expandida através do uso de barras de rolagem e o GridLayout, que utiliza um sistema de coordenadas para posicionar elementos.

instruction: Classe personalizada para armazenar as informações de cada instrução de uma forma mais organizada, como se fosse um struct de outras linguagens.

instruction_list_processor, **run_instructions**: Arquivo com a lógica da execução do ciclo de instruções, ele é importado pois também é reutilizado na entrada por arquivo.

- **Criando o widget da tela de entrada manual (classe ManualInputScreen):**

```
6  class ManualInputScreen(QWidget):
7      def __init__(self, show_welcome_screen_callback, show_alert_box_callback, center_on_screen_callback):
8          super().__init__()
9
10         self.show_welcome_screen_callback = show_welcome_screen_callback
11         self.show_alert_box_callback = show_alert_box_callback
12         self.center_on_screen_callback = center_on_screen_callback
13
14         self.instructions_array = []
15
16         ## Master Layout
17         self.manual_input_layout = QVBoxLayout()
18         self.manual_input_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

É uma subclasse do módulo widget do PyQt, como se fosse uma div do HTML, extremamente customizável.

Iniciamos a classe passando algumas funções da classe principal como argumento na forma de “callbacks” para que a mesma função seja invocada. Essas funções são para voltar a tela de boas vindas, criar uma mensagem de alerta e centralizar a aplicação na tela.

Definimos uma lista vazia de instruções.

Criamos o layout principal como um layout vertical (QVBoxLayout) e definimos seu alinhamento.

- **Tela de visualização, adição e remoção de instruções:**

```
21          ## Add instructions master layout
22          self.add_instructions_layout_container = QWidget()
23          self.add_instructions_layout_container.setVisible(False)
24          self.add_instructions_layout = QVBoxLayout()
```

```
27          ### Add instructions header
28          self.add_instructions_label = QLabel("<h3>Inserir instrucoes</h3>")
29          self.add_instructions_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
30          self.add_instructions_layout.addWidget(self.add_instructions_label)
31          self.add_instructions_layout.addSpacing(10)
32
33
34          ### Add instructions scroll area
35          self.add_instructions_scroll_area = QScrollArea()
36          self.add_instructions_scroll_area.setWidgetResizable(True)
37
38          self.add_instructions_scroll_layout_container = QWidget()
39          self.add_instructions_scroll_layout = QVBoxLayout()
40          self.add_instructions_scroll_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

Criamos um layout como contêiner desta tela, inicialmente invisível.

Criamos o cabeçalho desta dela com um QLabel e adicionando espaçamento.

Criamos uma ScrollArea para mostrar a lista de instruções, isso depende muito da resolução da tela do usuário mas em algum momento se existem muitas instruções será criada uma barra de rolamento.

```
        #### Add instructions list
        self.instructions_list_container = QWidget()
        self.instructions_list_layout = QGridLayout()
        self.instructions_list_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)

        self.instructions_list_container.setLayout(self.instructions_list_layout)
        self.add_instructions_scroll_layout.addWidget(self.instructions_list_container)

        self.add_instructions_scroll_layout_container.setLayout(self.add_instructions_scroll_layout)
        self.add_instructions_scroll_area.setWidget(self.add_instructions_scroll_layout_container)
        self.add_instructions_layout.addWidget(self.add_instructions_scroll_area)
        self.add_instructions_layout.addSpacing(10)
```

Criamos um layout Grid inicialmente vazio.

```
        ### Add instruction input
        self.instructions_list_add_item_container = QWidget()
        self.instructions_list_add_item_layout = QHBoxLayout()
        self.instructions_list_add_item_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)

        self.add_instructions_list_add_item_header_label = QLabel(f"{len(self.instructions_array)+1}")
        self.instructions_list_add_item_layout.addWidget(self.add_instructions_list_add_item_header_label)
```

```
        self.code_choice_box = QComboBox()
        self.code_choice_box.addItems(["000001", "000010", "000011", "000100", "000101",
        "000110", "000111", "001000", "001001", "001010", "001011", "001111", "001100"])
        self.instructions_list_add_item_layout.addWidget(self.code_choice_box)
        self.code_choice_box.currentIndexChanged.connect(self.on_main_combobox_changed)
```

```
        self.value_a_input = QLineEdit()
        self.value_a_input.setPlaceholderText("#pos")
        self.instructions_list_add_item_layout.addWidget(self.value_a_input)

        self.value_b_input = QLineEdit()
        self.value_b_input.setDisabled(True)
        self.instructions_list_add_item_layout.addWidget(self.value_b_input)

        self.confirm_add_list_add_item_button = QPushButton("Inserir")
        self.confirm_add_list_add_item_button.clicked.connect(self.insert_new_instruction)
        self.instructions_list_add_item_layout.addWidget(self.confirm_add_list_add_item_button)

        self.instructions_list_add_item_container.setLayout(self.instructions_list_add_item_layout)
        self.add_instructions_layout.addWidget(self.instructions_list_add_item_container)
```

Criamos um layout horizontal abaixo da Grid e ScrollArea para manipular a entrada do usuário composto de uma Combo Box com todos os códigos de instrução, dois QLineEdits para receber os valores dos operandos e um botão de inserção.

```
86          ### Add instructions back button
87          self.add_instructions_back_button = QPushButton("Voltar")
88          self.add_instructions_back_button.clicked.connect(self.show_add_instructions_view)
89          self.add_instructions_layout.addWidget(self.add_instructions_back_button)
90
91          self.add_instructions_layout_container.setLayout(self.add_instructions_layout)
92          self.manual_input_layout.addWidget(self.add_instructions_layout_container)
```

Criamos um botão de voltar abaixo de todos esses elementos.

Definimos a hierarquia entre todos esses layouts com o contêiner no topo.

- **Tela inicial da entrada manual:**

```
95          ## Menu view master layout
96          self.menu_view_layout_container = QWidget()
97          self.menu_view_layout_container.setVisible(True)
98          self.menu_view_layout = QVBoxLayout()
```

Criamos um layout e contêiner para a tela inicial da entrada manual.

```
101         ### Menu view header
102         self.menu_view_input_label = QLabel("<h2>Entrada manual de valores</h2>")
103         self.menu_view_input_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
104         self.menu_view_layout.addWidget(self.menu_view_input_label)
105
106
107         ### Insert instructions button
108         self.menu_view_input_insert_instructions_button = QPushButton("Inserir instrucoes")
109         self.menu_view_input_insert_instructions_button.clicked.connect(self.show_add_instructions_view)
110         self.menu_view_layout.addWidget(self.menu_view_input_insert_instructions_button)
111
112
113         ### Run instructions button
114         self.menu_view_run_instructions_button = QPushButton("Executar instrucoes")
115         self.menu_view_run_instructions_button.clicked.connect(self.run_instructions_function)
116         self.menu_view_layout.addWidget(self.menu_view_run_instructions_button)
117
118
119         ### Back button
120         self.menu_view_input_back_button = QPushButton("Voltar")
121         self.menu_view_input_back_button.clicked.connect(self.show_welcome_screen_callback)
122         self.menu_view_layout.addWidget(self.menu_view_input_back_button)
123
124         self.menu_view_layout_container.setLayout(self.menu_view_layout)
125         self.manual_input_layout.addWidget(self.menu_view_layout_container)
```

Criamos o cabeçalho e os 3 botões com as opções do usuário de ver a tela de visualização, adição e remoção definida anteriormente, a tela de resultado da execução das instruções, e voltar para a tela de boas vindas.

Definimos a hierarquia entre todos esses layouts com o contêiner no topo.

- **Tela de resultados:**

```
128         ## Results view master layout
129         self.results_view_layout_container = QWidget()
130         self.results_view_layout_container.setVisible(False)
131         self.results_view_layout = QVBoxLayout()
```

Criamos um layout e contêiner para a tela de resultado da execução das instruções, inicialmente invisível.

```
134         ### Results view scroll area
135         self.instructions_result_scroll_area = QScrollArea()
136         self.instructions_result_scroll_area.setWidgetResizable(True)
137
138         self.instructions_result_layout_container = QWidget()
139         self.instructions_result_layout = QVBoxLayout()
140         self.instructions_result_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

Criamos uma ScrollArea para caso o conteúdo seja muito extenso uma barra de rolagem seja criada.

```
143         #### MBR result
144         self.mbr_result_label = QLabel("<h2>MBR Final</h2>")
145         self.mbr_result_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
146         self.instructions_result_layout.addWidget(self.mbr_result_label)
147         self.instructions_result_layout.addSpacing(10)
148
149         self.mbr_result_value = QLabel()
150         self.mbr_result_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
151         self.instructions_result_layout.addWidget(self.mbr_result_value)
152         self.instructions_result_layout.addSpacing(20)
153
154
155         #### Instructions log output
156         self.instructions_log_output_label = QLabel("<h2>Instrucoes realizadas</h2>")
157         self.instructions_log_output_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
158         self.instructions_result_layout.addWidget(self.instructions_log_output_label)
159         self.instructions_result_layout.addSpacing(10)
160
161         self.instructions_log_output_value = QLabel()
162         self.instructions_log_output_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
163         self.instructions_result_layout.addWidget(self.instructions_log_output_value)
164         self.instructions_result_layout.addSpacing(20)
165
```

```
167         #### Tape result output
168         self.tape_result_output_label = QLabel("<h2>Fita final</h2>")
169         self.tape_result_output_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
170         self.instructions_result_layout.addWidget(self.tape_result_output_label)
171         self.instructions_result_layout.addSpacing(10)
172 
173         self.tape_result_output_value = QLabel()
174         self.tape_result_output_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
175         self.instructions_result_layout.addWidget(self.tape_result_output_value)
176 
177 
178         self.instructions_result_layout_container.setLayout(self.instructions_result_layout)
179         self.instructions_result_scroll_area.setWidget(self.instructions_result_layout_container)
180         self.results_view_layout.addWidget(self.instructions_result_scroll_area)
```

Criamos os 3 headers para as seções de log, valor do MBR, instruções realizadas e a fita dos endereços de memória e seus valores.

```
183         ## Results view back button
184         self.results_view_back_button = QPushButton("Voltar")
185         self.results_view_back_button.clicked.connect(self.show_menu_view)
186         self.results_view_layout.addWidget(self.results_view_back_button)
187 
188 
189         self.results_view_layout_container.setLayout(self.results_view_layout)
190         self.manual_input_layout.addWidget(self.results_view_layout_container)
```

Criamos um botão de voltar abaixo.

Definimos a hierarquia entre todos esses layouts com o contêiner no topo.

- **Bloco if not self.isMaximized:**

```
195         if not self.isMaximized:
196             QTimer.singleShot(0, self.center_on_screen_callback)
```

Centralizamos a aplicação caso esteja minimizada para melhor visualização.

- **Funções dessa classe de entrada manual:**

```
198     def show_menu_view(self):
199         self.results_view_layout_container.setVisible(False)
200         self.menu_view_layout_container.setVisible(True)
```

Definimos uma função show_menu_view para voltar à tela inicial desta classe de entrada manual manipulando a visibilidade dos grandes contêineres da classe.

```
202     def show_add_instructions_view(self):
203         if(self.add_instructions_layout_container.isVisible()):
204             self.add_instructions_layout_container.setVisible(False)
205             self.menu_view_layout_container.setVisible(True)
206         else:
207             self.menu_view_layout_container.setVisible(False)
208             self.add_instructions_layout_container.setVisible(True)
209             self.update_add_instructions_view()
210 
211             self.code_choice_box.setCurrentIndex(0)
212 
```

```
213             self.value_a_input.clear()
214             self.value_a_input.setDisabled(False)
215             self.value_a_input.setPlaceholderText("#pos")
216 
217             self.value_b_input.clear()
218             self.value_b_input.setDisabled(True)
219             self.value_b_input.setPlaceholderText("")
```

Definimos uma função show_add_instructions_view para mostrar a tela de visualização, adição e remoção de instruções. Esta função realiza a verificação da sequência atual de instruções e sempre inicializa a Combo Box na primeira instrução.

```
221     def update_add_instructions_view(self):
222         while self.instructions_list_layout.count():
223             item = self.instructions_list_layout.takeAt(0)
224             if item.widget():
225                 item.widget().deleteLater()
226 
227         self.add_instructions_list_header_instruction_label = QLabel(f"Instrucao")
228         self.add_instructions_list_header_instruction_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
229         self.instructions_list_layout.addWidget(self.add_instructions_list_header_instruction_label, 0, 0)
230 
231         self.add_instructions_list_header_code_label = QLabel(f"Codigo da Instrucao")
232         self.add_instructions_list_header_code_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
233         self.instructions_list_layout.addWidget(self.add_instructions_list_header_code_label, 0, 1)
234 
235         self.add_instructions_list_header_operand_label = QLabel(f"Operandos")
236         self.add_instructions_list_header_operand_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
237         self.instructions_list_layout.addWidget(self.add_instructions_list_header_operand_label, 0, 2, 1, 2)
238 
239         self.add_instructions_list_header_delete_label = QLabel(f"Remover?")
240         self.add_instructions_list_header_delete_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
241         self.instructions_list_layout.addWidget(self.add_instructions_list_header_delete_label, 0, 4)
```

```
243         for i in range(len(self.instructions_array)):
244             self.add_instructions_list_item_pos_label = QLabel(f"{i+1}")
245             self.add_instructions_list_item_pos_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
246             self.instructions_list_layout.addWidget(self.add_instructions_list_item_pos_label, (i+1), 0)
247 
248             self.add_instructions_list_item_code_label = QLabel(f"{self.instructions_array[i].code}")
249             self.add_instructions_list_item_code_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
250             self.instructions_list_layout.addWidget(self.add_instructions_list_item_code_label, (i+1), 1)
251 
252             if(self.instructions_array[i].code not in ["001010", "001011", "001100"]):
253 
254                 operand_title = ""
255 
256                 if(self.instructions_array[i].code in ["000001", "000010", "000011", "000100", "000101", "000110", "001111"]):
257                     operand_title = "pos"
258                 else:
259                     operand_title = "lin"
260 
261                 self.add_instructions_list_item_operand_a_label = QLabel(f"#{operand_title} {self.instructions_array[i].value_a}")
262                 self.add_instructions_list_item_operand_a_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
263                 self.instructions_list_layout.addWidget(self.add_instructions_list_item_operand_a_label, (i+1), 2)
264 
265                 if(self.instructions_array[i].code == "000010"):
266                     self.add_instructions_list_item_operand_b_label = QLabel(f"#dado {self.instructions_array[i].value_b}")
267                     self.add_instructions_list_item_operand_b_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
268                     self.instructions_list_layout.addWidget(self.add_instructions_list_item_operand_b_label, (i+1), 3)
269 
270             self.add_instructions_list_item_delete_button = QPushButton("Remover")
271             self.add_instructions_list_item_delete_button.clicked.connect(self.create_delete_button_callback(i))
272             self.instructions_list_layout.addWidget(self.add_instructions_list_item_delete_button, (i+1), 4)
273 
274         self.add_instructions_list_add_item_header_label.setText(f"{len(self.instructions_array)+1}")
```

Definimos uma função auxiliar update_add_instructions_view que utiliza o sistema de Grid Layout para popular a Grid vazia criada quando a classe foi inicializada. O Grid é útil aqui para estruturar as instruções em uma forma de tabela, com os headers indicando a linha da instrução, seu código, operandos se existirem, além de um botão que possibilita a remoção desta instrução específica.

A função também estabelece o label apropriado para os operandos considerando o código de cada instrução (lin, pos, dado ou nenhum) bem como a sua posição na sequência que corresponde ao índice do array que armazena as instruções.

```
276     def on_main_combobox_changed(self):
277         selected_option = self.code_choice_box.currentText()
278 
279         if(selected_option not in ["001010", "001011", "001100"]):
280             self.value_a_input.clear()
281             self.value_a_input.setDisabled(False)
282 
283             operand_title = ""
284 
285             if(selected_option in ["000001", "000010", "000011", "000100", "000101", "000110", "001111"]):
286                 operand_title = "pos"
287             else:
288                 operand_title = "lin"
289 
290             self.value_a_input.setPlaceholderText(f"#{operand_title}")
291 
292             if(selected_option == "000010"):
293                 self.value_b_input.clear()
294                 self.value_b_input.setDisabled(False)
295                 self.value_b_input.setPlaceholderText(f"#dado")
296             else:
297                 self.value_b_input.clear()
298                 self.value_b_input.setDisabled(True)
299                 self.value_b_input.setPlaceholderText("")
```

```
300         else:
301             self.value_a_input.clear()
302             self.value_a_input.setDisabled(True)
303             self.value_a_input.setPlaceholderText("")
304 
305             self.value_b_input.clear()
306             self.value_b_input.setDisabled(True)
307             self.value_b_input.setPlaceholderText("")
```

Definimos uma função on_main_combobox_changed conectada a Combo Box que permite o usuário selecionar uma instrução para adicionar à sequência. Esta função altera dinamicamente quais caixas de entrada ficam ativas ou não baseadas nos operandos de cada instrução.

```
309     def insert_new_instruction(self):
310         selected_option = self.code_choice_box.currentText()
311         invalid_value = False
312 
313         if(selected_option not in ["001010", "001011", "001100"]):
314             num_a = 0
315             num_b = 0
316 
317             try:
318                 num_a = int(self.value_a_input.text())
319             except ValueError:
320                 invalid_value = True
321 
322             if(not invalid_value):
323                 if(selected_option == "000010"):
324                     try:
325                         num_b = float(self.value_b_input.text())
326                     except ValueError:
327                         invalid_value = True
328 
329                     if(not invalid_value):
330                         self.instructions_array.append(Instruction(self.code_choice_box.currentText(), num_a, num_b))
331                 else:
332                     self.instructions_array.append(Instruction(self.code_choice_box.currentText(), num_a))
333         else:
334             self.instructions_array.append(Instruction(self.code_choice_box.currentText()))
335 
336         if(invalid_value):
337             self.show_alert_box_callback("Alerta!", f"Instrucao invalida," +
338             "parametros devem ser numeros inteiros, ou decimal apenas no caso do dado em '000010'.")
339         else:
340             self.update_add_instructions_view()
```

Definimos uma função insert_new_instruction que cuida da adição de uma instrução para a sequência, verificando se a entrada foi um valor válido ou não. Apenas valores inteiros são operandos válidos, com exceção do código “000010” que permite float em seu segundo operando.

```
341     def remove_instruction(self, index):
342         self.instructions_array.pop(index)
343 
344         self.update_add_instructions_view()
```

Definimos uma função remove_instruction que utiliza o índice armazenado no botão de remoção para deletar o índice certo da instrução clicada pelo usuário.

```
346     def create_delete_button_callback(self, index):
347         def callback():
348             self.remove_instruction(index)
349         return callback
```

Definimos uma função create_delete_button_callback que dinamicamente cria funções únicas de remoção para cada botão de remoção, desta forma o seu índice sempre corresponderá ao item correto.

```
351     def run_instructions_function(self):
352         self.menu_view_layout_container.setVisible(False)
353         self.results_view_layout_container.setVisible(True)
354 
355         mbr, log, tape_display, out_of_bounds_error = run_instructions(self.instructions_array)
356 
357         if(out_of_bounds_error):
358             self.show_alert_box_callback("Alerta!", f"Sequencia de instrucoes parada antes de " +
359                                          "sua conclusao, jump realizado para linha inexistente!")
360 
361         self.mbr_result_value.setText(str(mbr))
362         self.tape_result_output_value.setText(''.join(tape_display))
363 
364         if(len(log) == 0):
365             self.instructions_log_output_value.setText("VAZIO")
366         else:
367             self.instructions_log_output_value.setText(''.join(log))
368 
369         self.results_view_layout_container.setVisible(True)
```

Finalmente, definimos uma função run_instructions_function que utiliza a array de instruções armazenada na classe para executar a sequência de instruções atual. Esta função está ligada a tela de resultado, recebe os logs e os mostra para o usuário saber o que aconteceu. Esta função utiliza a lógica importada da função run_instructions do arquivo instruction_list_processor.py.

- **Em resumo, manual_input_screen.py:**

Cria um widget que pode ser instanciado e adicionado para o layout em pilha da classe principal. Este widget contém uma lógica similar ao layout em pilha com a visibilidade de sub layouts que direcionam o usuário para as funções de inserção ou execução de uma sequência de instruções. Esta classe tem o objetivo de proporcionar uma boa experiência de entrada manual de dados ao usuário.

file_input_screen.py

- **Primeiramente os módulos e classes necessários são importados:**

```
1  from PyQt6.QtCore import Qt, QTimer
2  from PyQt6.QtWidgets import QWidget, QPushButton, QVBoxLayout, QLabel, QTextEdit, QScrollArea, QFileDialog
3  from PyQt6.QtGui import QGuiApplication
4  from instruction import Instruction
5  from instruction_list_processor import run_instructions
```

PyQt6.QtCore, **PyQt6.QtWidgets**, **PyQt6.QtGui**: Módulos do PyQt6 para criação de GUI.

QTimer: Uma classe do PyQt6 usada para lidar com a contagem de tempo.

QWidget, QPushButton, QVBoxLayout, QLabel, QTextEdit, QScrollArea, QFileDialog: Classes PyQt6 para criação de elementos GUI, similar a explicação de main_window.py e manual_input_screen.py, as únicas novidades aqui são o QTextEdit que é basicamente um QLineEdit só que é uma caixa de texto ao invés de apenas uma linha de entrada e o QFileDialog que permite o usuário explorar a estrutura de pastas de seu computador para abrir um arquivo de texto com instruções pré-definidas.

QGuiApplication, **QIcon**: Classes PyQt6 para aplicativos GUI e gerenciamento de ícones.

instruction: Classe personalizada para armazenar as informações de cada instrução de uma forma mais organizada, como se fosse um struct de outras linguagens.

instruction_list_processor, **run_instructions**: Arquivo com a lógica da execução do ciclo de instruções, ele é importado pois também é reutilizado na entrada manual.

- **Criando o widget da tela de entrada por arquivo(classe FileInputScreen):**

```python
class FileInputScreen(QWidget):
    def __init__(self, show_welcome_screen_callback, show_alert_box_callback, center_on_screen_callback):
        super().__init__()

        self.show_welcome_screen_callback = show_welcome_screen_callback
        self.show_alert_box_callback = show_alert_box_callback
        self.center_on_screen_callback = center_on_screen_callback

        self.instructions_array = []

        ## Screen dimensions
        screen = QGuiApplication.primaryScreen()
        screen_size = screen.availableSize()


        ## Master Layout
        self.master_layout = QVBoxLayout()

        self.file_input_scroll_area = QScrollArea()
        self.file_input_scroll_area.setWidgetResizable(True)
        self.file_input_scroll_area.setMinimumWidth(int(screen_size.width() * 0.70))
        self.file_input_scroll_area.setMinimumHeight(int(screen_size.height() * 0.70))

        self.file_input_container = QWidget()
        self.file_input_layout = QVBoxLayout()
        self.file_input_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

```python
        ## Manual Input Screen Header
        self.file_input_label = QLabel("<h2>Entrada de valores por arquivo</h2>")
        self.file_input_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.file_input_layout.addWidget(self.file_input_label)


        ## Text box input
        self.file_input_text_box = QTextEdit()
        self.file_input_text_box.setPlaceholderText(f"Cole o texto da tabela de transicao aqui...")
        self.file_input_text_box.setMinimumHeight(int(screen_size.height() * 0.50))
        self.file_input_layout.addWidget(self.file_input_text_box)


        ## Confirm button
        self.file_input_confirm_button = QPushButton("Confirmar")
        self.file_input_confirm_button.clicked.connect(self.get_data)
        self.file_input_layout.addWidget(self.file_input_confirm_button)

        ## Open file button
        self.file_input_open_file_button = QPushButton("Abrir arquivo")
        self.file_input_open_file_button.clicked.connect(self.open_file_button)
        self.file_input_layout.addWidget(self.file_input_open_file_button)

        ## Back button
        self.file_input_back_button = QPushButton("Voltar")
        self.file_input_back_button.clicked.connect(lambda: self.show_welcome_screen_callback())

        self.file_input_layout.addWidget(self.file_input_back_button)
```

```
        ## Result scroll area
        self.instructions_result_scroll_area = QScrollArea()
        self.instructions_result_scroll_area.setVisible(False)
        self.instructions_result_scroll_area.setWidgetResizable(True)
        self.instructions_result_scroll_area.setMinimumHeight(int(screen_size.height() * 0.50))

        self.instructions_result_layout_container = QWidget()
        self.instructions_result_layout = QVBoxLayout()
        self.instructions_result_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)

        ## MBR result
        self.mbr_result_label = QLabel("<h2>MBR Final</h2>")
        self.mbr_result_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.instructions_result_layout.addWidget(self.mbr_result_label)
        self.instructions_result_layout.addSpacing(10)

        self.mbr_result_value = QLabel()
        self.mbr_result_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.instructions_result_layout.addWidget(self.mbr_result_value)
        self.instructions_result_layout.addSpacing(20)
```

```
        ## Instructions log output
        self.instructions_log_output_label = QLabel("<h2>Instrucoes realizadas</h2>")
        self.instructions_log_output_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.instructions_result_layout.addWidget(self.instructions_log_output_label)
        self.instructions_result_layout.addSpacing(10)

        self.instructions_log_output_value = QLabel()
        self.instructions_log_output_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.instructions_result_layout.addWidget(self.instructions_log_output_value)
        self.instructions_result_layout.addSpacing(20)


        ## Tape result output
        self.tape_result_output_label = QLabel("<h2>Fita final</h2>")
        self.tape_result_output_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.instructions_result_layout.addWidget(self.tape_result_output_label)
        self.instructions_result_layout.addSpacing(10)

        self.tape_result_output_value = QLabel()
        self.tape_result_output_value.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.instructions_result_layout.addWidget(self.tape_result_output_value)


        self.instructions_result_layout_container.setLayout(self.instructions_result_layout)
        self.instructions_result_scroll_area.setWidget(self.instructions_result_layout_container)
        self.file_input_layout.addWidget(self.instructions_result_scroll_area)
```

```
115         self.file_input_container.setLayout(self.file_input_layout)
116         self.file_input_scroll_area.setWidget(self.file_input_container)
117 
118         self.master_layout.addWidget(self.file_input_scroll_area)
119 
120         self.setLayout(self.master_layout)
121 
122         if not self.isMaximized:
123             QTimer.singleShot(0, self.center_on_screen_callback)
```

Estrutura extremamente similar a tela de resultado da classe de entrada manual, porém com a caixa de texto no começo do layout vertical para que o usuário insira várias linhas de texto rapidamente, agilizando a velocidade de teste de instruções diferentes.

Além disso, temos o botão de navegação da estrutura de pastas e arquivos do computador do usuário.

- **Funções dessa classe de entrada por arquivo:**

```
125     def open_file_button(self):
126         file_dialog = QFileDialog()
127         file_dialog.setNameFilter("Sequencias de instrucoes (*.txt)")
128         file_dialog.setFileMode(QFileDialog.FileMode.ExistingFile)
129 
130         if file_dialog.exec() == QFileDialog.DialogCode.Accepted:
131             selected_file = file_dialog.selectedFiles()[0]
132 
133             with open(selected_file, "r") as file:
134                 raw_text = file.read()
135 
136             self.file_input_text_box.setText(raw_text)
137 
138             lines = raw_text.split('\n')
139             self.parse_instructions(lines)
```

Definimos uma função open_file_button que utiliza um FileDialog para mostrar a tela de explorar os arquivos do computador do usuário com o filtro para arquivos do formato .txt. O conteúdo deste arquivo será escrito na caixa TextEdit da interface e pode ser editada pelo usuário a qualquer momento.

```
141     def get_data(self):
142         raw_text = self.file_input_text_box.toPlainText()
143         lines = raw_text.split('\n')
144         self.parse_instructions(lines)
```

Definimos uma função get_data que recebe o texto da QTextEdit para variáveis mais “amigáveis” ao processamento de dados, dividindo o conteúdo em uma array de strings, onde cada índice corresponde a uma linha de texto.

```
    def parse_instructions(self, lines):
        self.instructions_array.clear()
        error_triggered = False

        for i in range(len(lines)):
            line_value = lines[i].split(",")
            num_a = 0
            num_b = 0

            if(line_value[0] not in ["000001", "000010", "000011", "000100", "000101", "000110", "000111", "001000", "001001", "001010", "001011", "001111", "001100"]):
                self.show_alert_box_callback("Alerta!", f"Sequencia de instrucoes invalida! Instrucao da linha {(i+1)} nao possui codigo valido!")
                error_triggered = True
                break

            if(line_value[0] in ["000001", "000010", "000011", "000100", "000101", "000110", "000111", "001000", "001001", "001111"]):
                try:
                    num_a = int(line_value[1])
                except ValueError:
                    self.show_alert_box_callback("Alerta!", f"Sequencia de instrucoes invalida! Instrucao da linha {(i+1)} nao possui valor valido!")
                    error_triggered = True
                    break

                if(line_value[0] == "000010"):
                    try:
                        num_b = float(line_value[2])
                    except ValueError:
                        self.show_alert_box_callback("Alerta!", f"Sequencia de instrucoes invalida! Instrucao da linha {(i+1)} nao possui valor valido!")
                        error_triggered = True
                        break

                    self.instructions_array.append(Instruction(line_value[0], num_a, num_b))
                else:
                    self.instructions_array.append(Instruction(line_value[0], num_a))
            else:
                self.instructions_array.append(Instruction(line_value[0]))

        if(not error_triggered):
            mbr, log, tape_display, out_of_bounds_error = run_instructions(self.instructions_array)

            if(out_of_bounds_error):
                self.show_alert_box_callback("Alerta!", f"Sequencia de instrucoes parada antes de sua conclusao, jump realizado para linha inexistente!")

            self.mbr_result_value.setText(str(mbr))
            self.tape_result_output_value.setText(''.join(tape_display))

            if(len(log) == 0):
                self.instructions_log_output_value.setText("VAZIO")
            else:
                self.instructions_log_output_value.setText(''.join(log))

            self.instructions_result_scroll_area.setVisible(True)
        else:
            self.instructions_result_scroll_area.setVisible(False)
```

Definimos uma função parse_instructions para verificar se as instruções foram inseridas com a formatação correta, "[CODE],[VALUE_A],[VALUE_B]" em uma linha separada, dividindo cada valor com uma vírgula, com a mesma verificação de número, inteiro ou float da entrada manual bem como uma verificação adicional se o código em questão existe ou não. Caso nenhum problema seja encontrado, as instruções são executadas com a lógica

importada da função run_instructions do arquivo instruction_list_processor.py e o log resultante é mostrado ao usuário.

- **Em resumo, file_input_screen.py:**

Cria um widget que pode ser instanciado e adicionado para o layout em pilha da classe principal. Este widget é bem similar ao manual_input_screen.py porém com algumas nuances para lidar com a entrada de um grande string de texto ao invés de uma entrada mais controlada como a manual. Nota-se que ambas as classes utilizam a mesma lógica importada de outro arquivo.

instruction.py

- **Criando a classe Instruction**:

```python
class Instruction:
    def __init__(self, code, value_a=None, value_b=None):
        self.code = code
        self.value_a = value_a
        self.value_b = value_b
```

Esta classe é extremamente simples, quase como um struct de outras linguagens, definindo alguns valores a serem armazenados para uma conveniência um pouco maior do desenvolvimento da aplicação.

Em síntese, o código da instrução é sempre armazenado e os valores a e b são opcionais, tendo em vista que dependem do próprio código de instrução. Desta forma a lógica para a adição de uma instrução cuida desta verificação e inserção correta na array de sequência de instruções.

instruction_list_processor.py

- **Primeiramente os módulos e classes necessários são importados:**

```
1   from math import sqrt
```

math com a função sqrt: Realiza a operação de raiz quadrada da instrução “001010”.

- **Criando a função run_instructions**:

Esta função é a espinha dorsal de toda a aplicação e contém a lógica que executa a sequência de instruções.

```
3   def run_instructions(instructions_array):
4       greatest_pos_value = 0
5
6       for i in range(len(instructions_array)):
7           if(instructions_array[i].code in ["000001", "000010", "000011", "000100", "000101", "000110", "001111"]):
8               if(instructions_array[i].value_a > greatest_pos_value):
9                   greatest_pos_value = instructions_array[i].value_a
```

De início, ela verifica qual é a maior posição de memória para retornar uma fita de endereços de memória mais “verídica”.

```
11      mbr = 0
12      tape = [0.0] * (greatest_pos_value + 1)
13      log = []
14      tape_display = []
15      current_instruction = 0
16
17      out_of_bounds_error = False
```

Todas as variáveis auxiliares são inicializadas em zero, vazias ou falsas.

```
19      while(current_instruction < len(instructions_array)):
20          current_instruction_code = instructions_array[current_instruction].code
```

Um loop while é utilizado para executar toda a sequência de instruções devido a possibilidade de existir JUMPs que modificam a ordem de execução.

```
22      match current_instruction_code:
23          case "000001":
24              mbr = tape[instructions_array[current_instruction].value_a]
25              log.append(f"Operacao {instructions_array[current_instruction].code} realizada\n")
26              log.append(f"MBR recebeu o valor {tape[instructions_array[current_instruction].value_a]:.1f} da posicao {instructions_array[current_instruction].value_a}\n")
27              log.append(f"MBR atual e {mbr:.1f}\n\n")
28          case "000010":
29              tape[instructions_array[current_instruction].value_a] = instructions_array[current_instruction].value_b
30              log.append(f"Operacao {instructions_array[current_instruction].code} realizada\n")
31              log.append(f"A posicao {instructions_array[current_instruction].value_a} recebeu o valor {instructions_array[current_instruction].value_b:.1f}\n\n")
32          case "000011":
```

Cada iteração cai em um switch case, ou match case em Python que verifica o código da instrução e executa as operações apropriadas, bem como armazena o que aconteceu no log.

```
96          if(current_instruction < 0 or current_instruction > len(instructions_array)):
97              out_of_bounds_error = True
98              break
```

Como uma estrutura de array é utilizada, existe a possibilidade de um JUMP ser executado para um índice fora de seu escopo, desta forma a booleana out_of_bounds_error serve como um gatilho para interromper a execução e informar o usuário caso isso aconteça.

```
100     for i in range(len(tape)):
101         tape_display.append(str(str(i) + ": " + str(tape[i]) + " - "))
102         if(i % 15 == 0):
103             tape_display.append("\n")
104 
105     if(out_of_bounds_error):
106         log.append("Operacao parada neste ponto. Linha do jump inexistente\n")
107         return mbr, log, tape_display, True
108 
109     return mbr, log, tape_display, False
```

Por fim, o MBR, a fita e o log são retornados pela função, dados que são utilizados nas classes ManualInputScreen e FileInputScreen para serem mostrados na tela da aplicação.