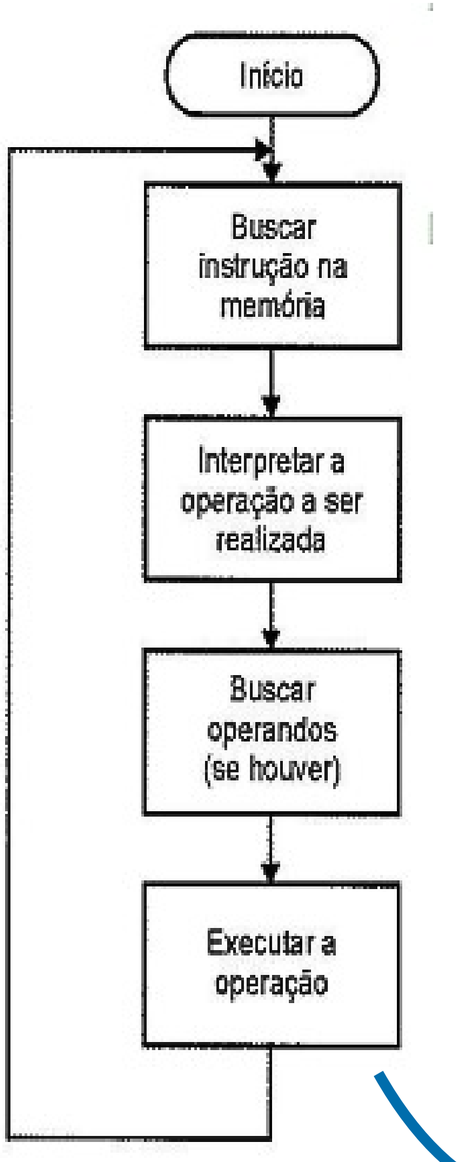
APRESENTANDO O

SIMULADOR MIC-1

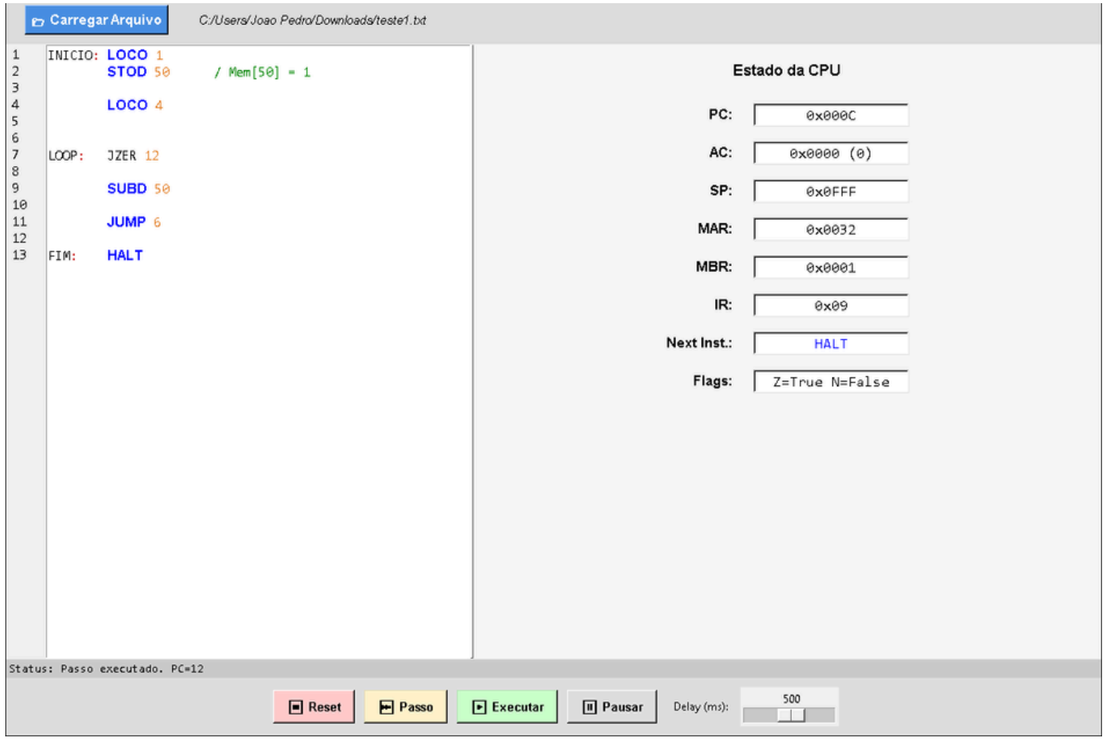
- Yago do Nascimento Santos
- Pedro Henrique Neves Sixel
- João Pedro Saliveros Cordeiro
- Rafael Ribeiro Ramos Pinto

SUMÁRIO



NOSSO OBJETIVO

Com o intuito de representar didaticamente a operação de um processador simples, baseado no modelo MIC-1, nosso simulador oferece uma interface que permite carregar e executar programas, convertidos em instruções. Ele dispõe de registradores, memória e instruções, e o usuário consegue visualizar a execução passo a passo. Isso torna mais fácil entender o fluxo de controle, as operações aritméticas e como os dados são manipulados na memória.



LINGUAGEM DE IMPLEMENTAÇÃO

Nosso programa é desenvolvido em Python, uma escolha estratégica devido à sua simplicidade técnica e à disponibilidade de recursos amplamente conhecidos para aprendizado e pesquisa. Ferramentas, como o Tkinter que foi empregado na interface gráfica, exemplificam essa acessibilidade.

- **DEMONSTRAÇÃO DO INÍCIO DA CLASSE SIMULADOR**
- **DEFININDO O TAMANHO DA MEMÓRIA**
- **DEFININDO OS REGISTRADORES**

```python
#Simulador MIC-1
class MIC1Simulator:  2 usages
    def __init__(self):
        self.tamanhoDaMemoria = 4096
        self.memoria = [0] * self.tamanhoDaMemoria

        #Registradores
        self.pc = 0 #Program Counter
        self.ac = 0 #Accumulador
        self.sp = self.tamanhoDaMemoria - 1 #Stack Pointer
        self.mar = 0 #Registrador de End. de Mem.
        self.mbr = 0 #Memory Buffer
        self.ir = 0 #Registrador de instrução
```

DEFININDO OS OPCODES DE CADA INSTRUÇÃO

```
#Instruções
self.instructions = {
    0x00: ("HALT", self.halt),
    0x01: ("LODD", self.load),
    0x02: ("STOD", self.store),
    0x03: ("ADDD", self.add),
    0x04: ("SUBD", self.sub),
    0x05: ("JPOS", self.jpos),
    0x06: ("STODL", self.stodl),
    0x07: ("LOCO", self.loadcons),
    0x08: ("JUMP", self.jump),
    0x09: ("JZER", self.jumpZERO),
    0x0A: ("STODL", self.jneg),
    0x0B: ("ADDL", self.call),
    0x0C: ("INSP", self.insp),
    0x0D: ("PUSH", self.push),
    0x0E: ("POP", self.pop),
}
```

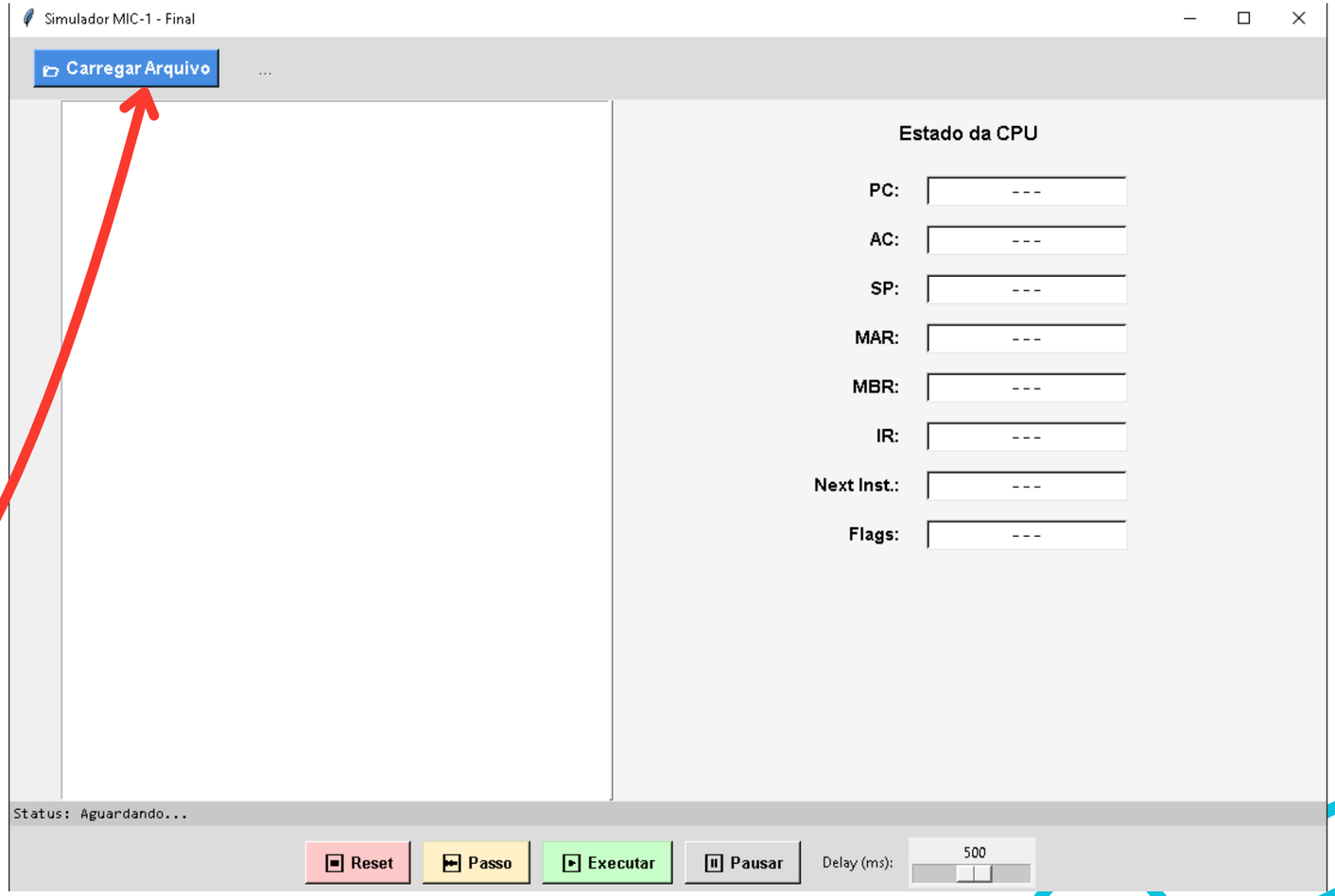
EXEMPLO DE EXECUÇÃO

Vamos utilizar o seguinte arquivo de texto como instruções:

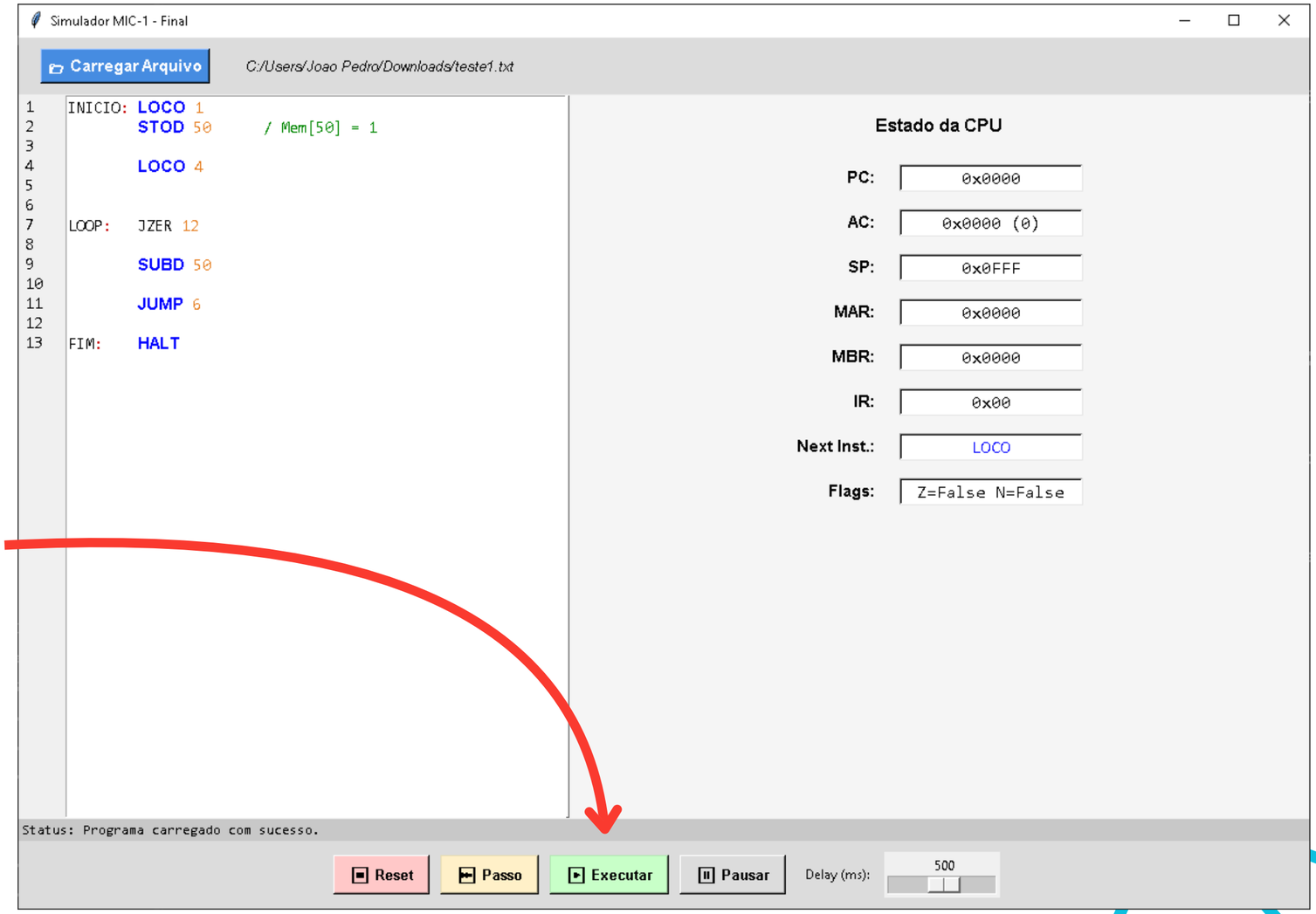
teste.txt

```
/ Inicializacao: Cria a constante 1 na memoria (endereco 50)
INICIO: LOCO 1
        STOD 50      / Mem[50] = 1

/ Configura o contador inicial no AC (comeca com 4)
        LOCO 4

/ --- INICIO DO LOOP (Endereco de memoria 6) ---
LOOP:   JZER 12      / Se AC for Zero, pula para o final (endereco
12)

        SUBD 50      / Subtrai o valor que esta no endereco 50 (4 -
1)

        JUMP 6       / Volta para o inicio do LOOP (endereco 6)

/ --- FIM (Endereco de memoria 12) ---
FIM:    HALT
```

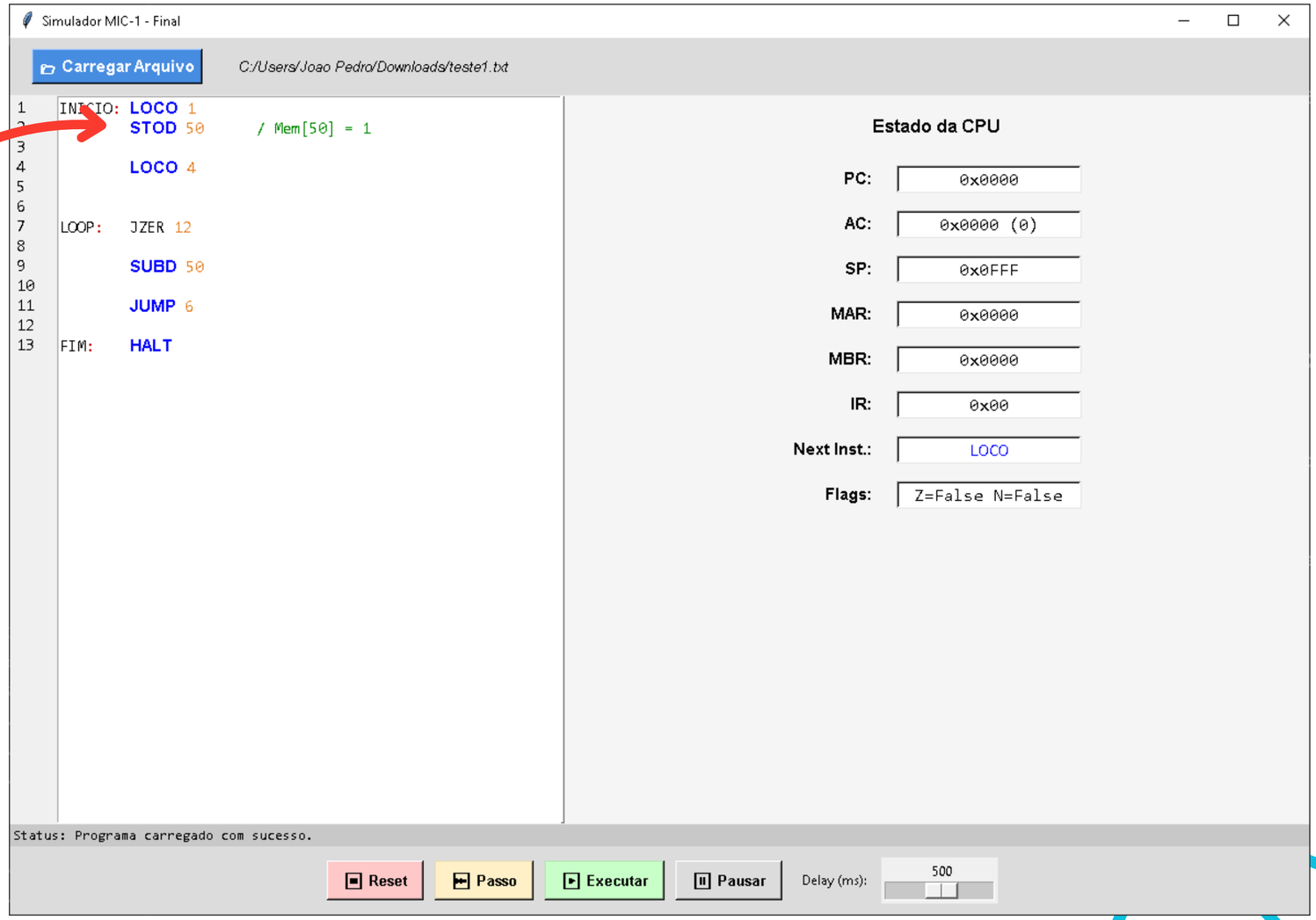
Ao abrir o simulador, precisamos carregar um arquivo .txt com as instruções escritas



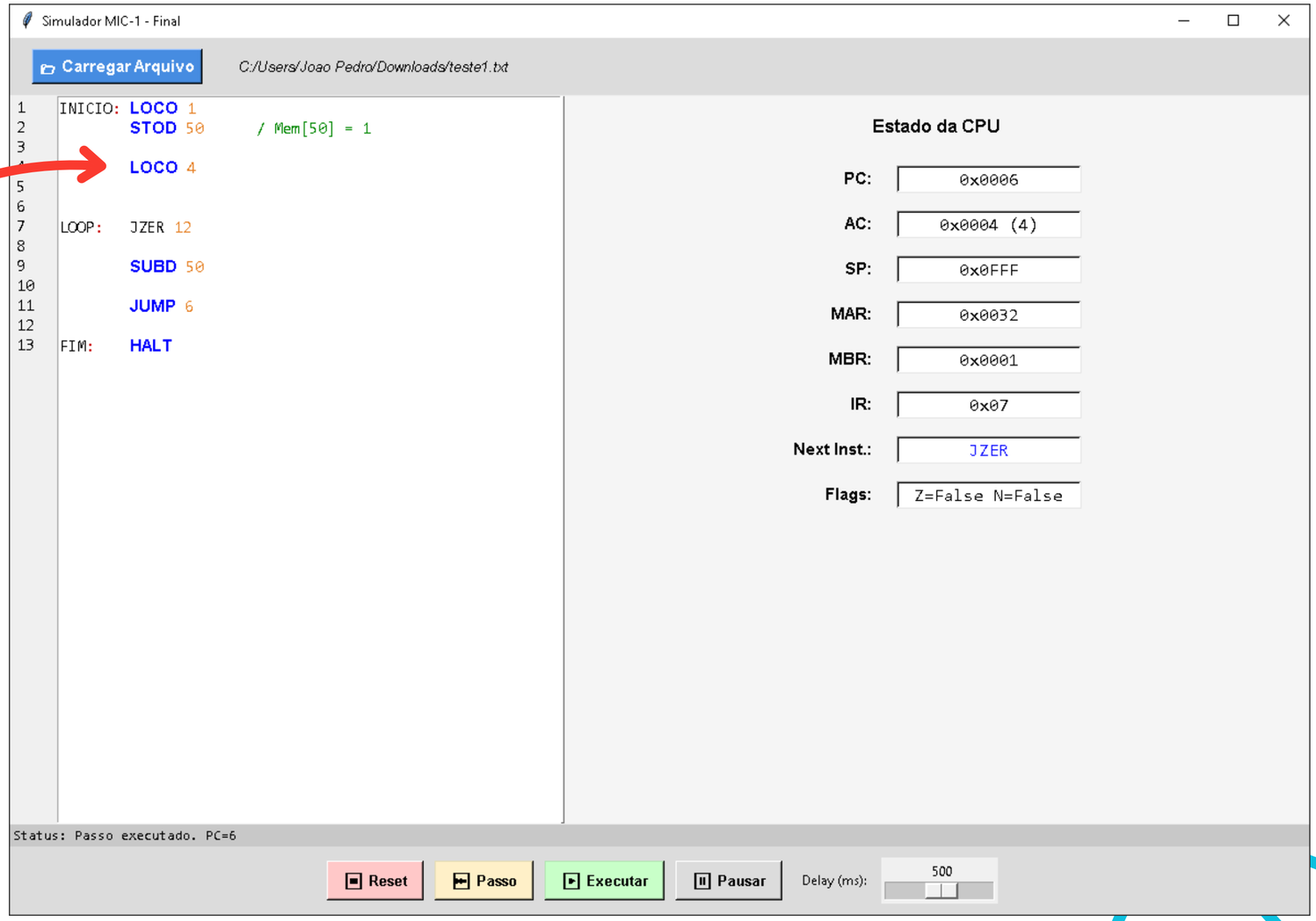
Com o arquivo carregado, podemos iniciar a execução.



Inicialização: Cria a constante 1 na memoria (endereço 50)

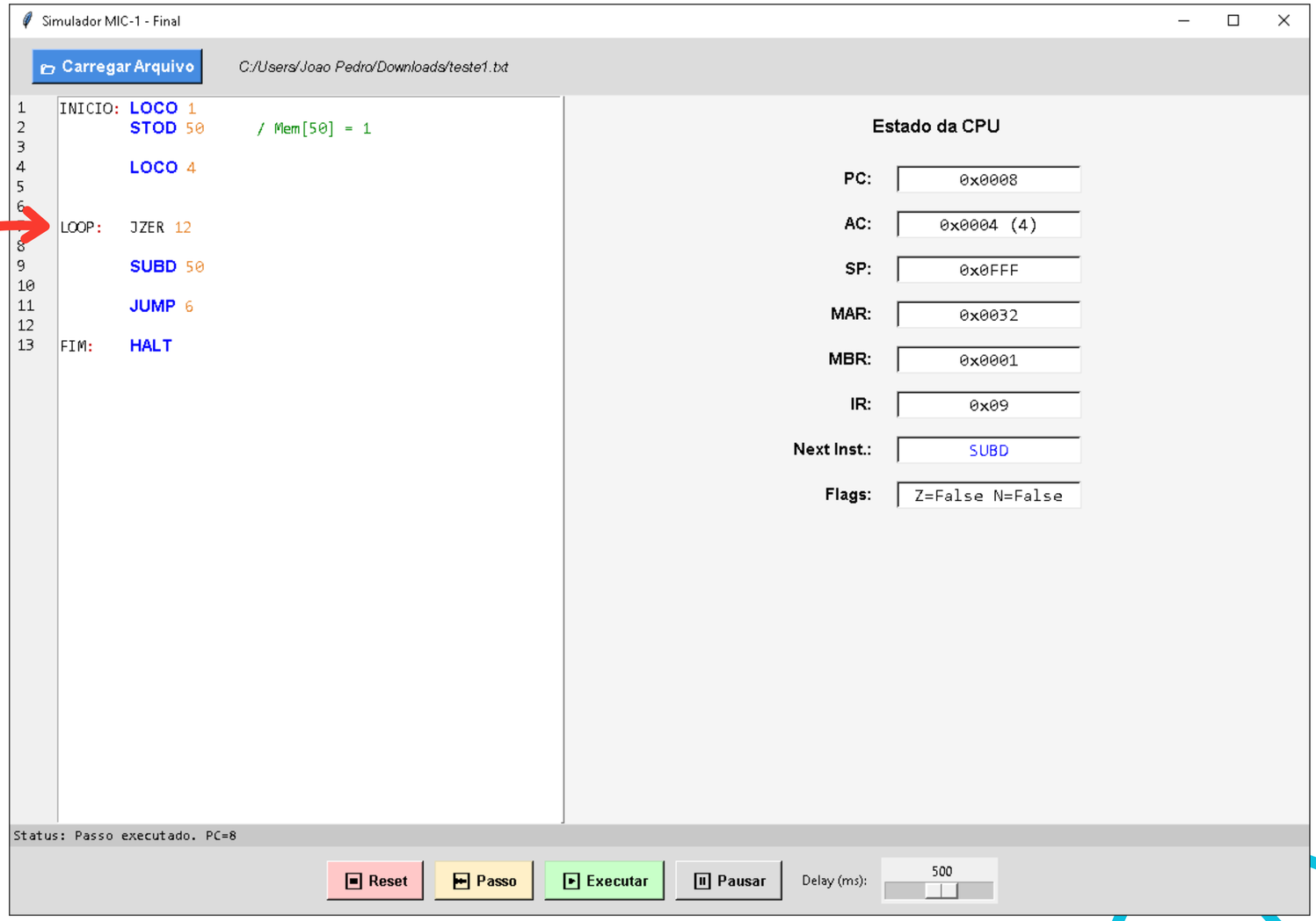


Configura o contador inicial no AC (começa com 4)

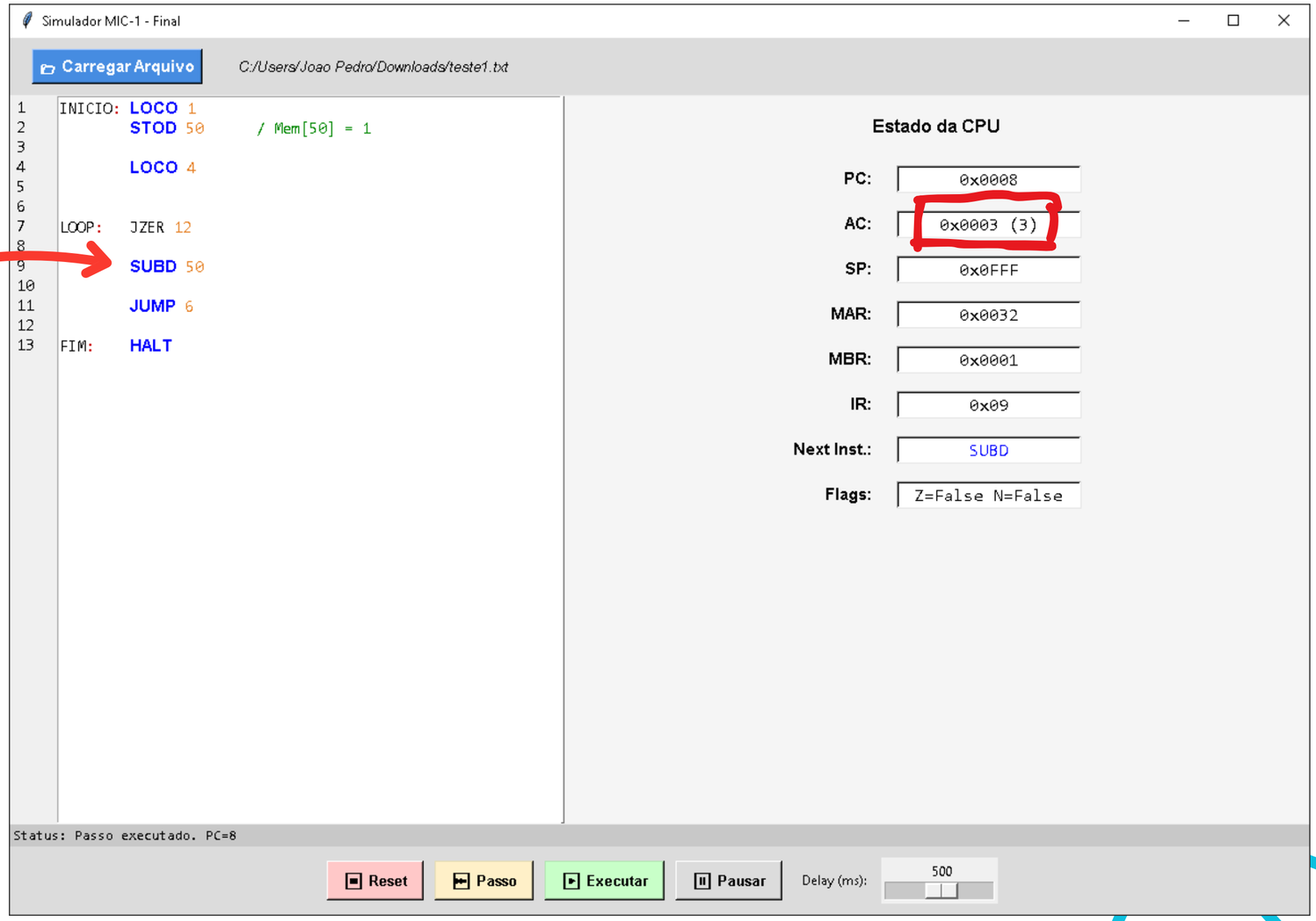


INICIO DO LOOP (Endereço de memória 6)

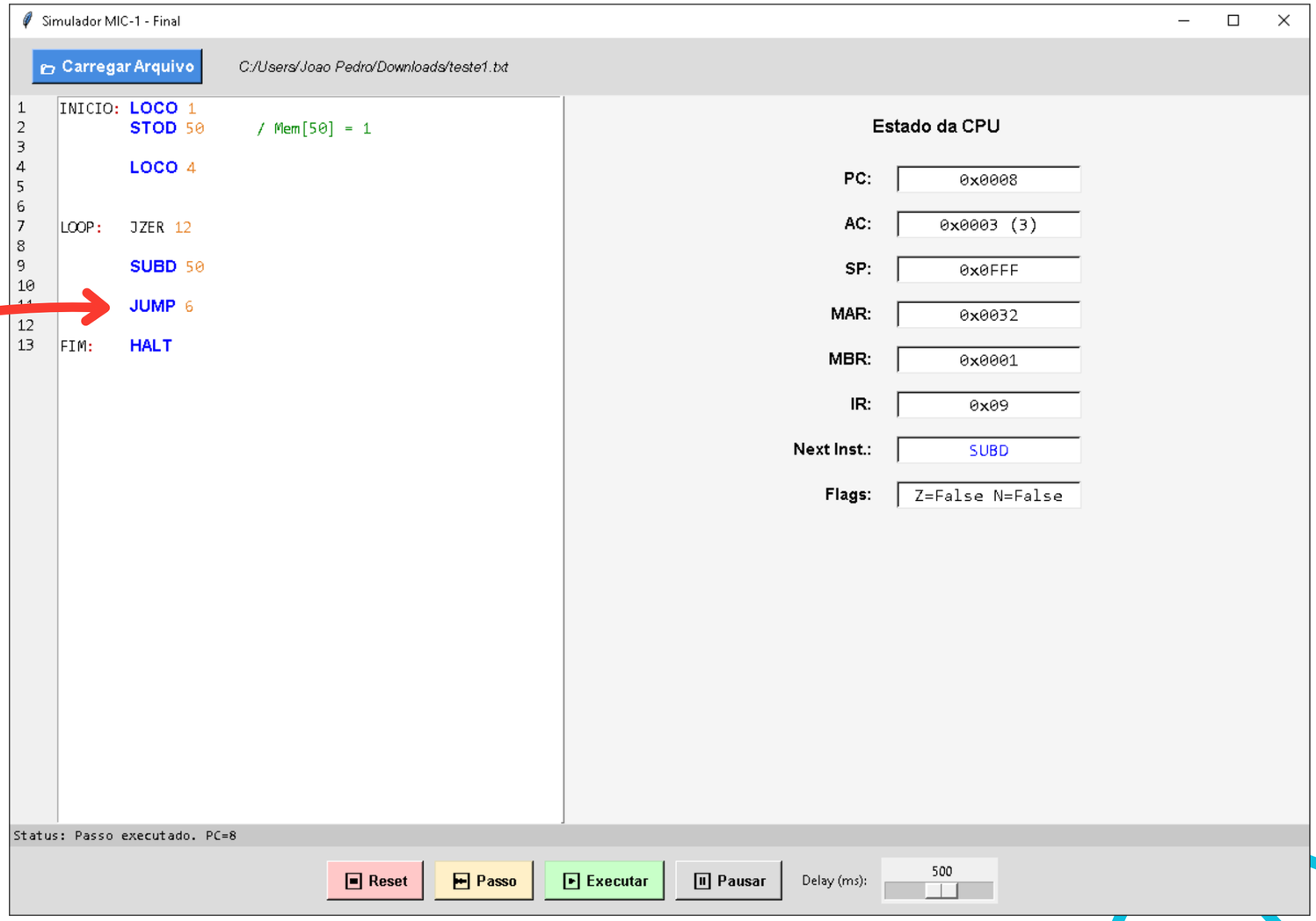
Se AC for Zero, pula para o final (endereço 12)



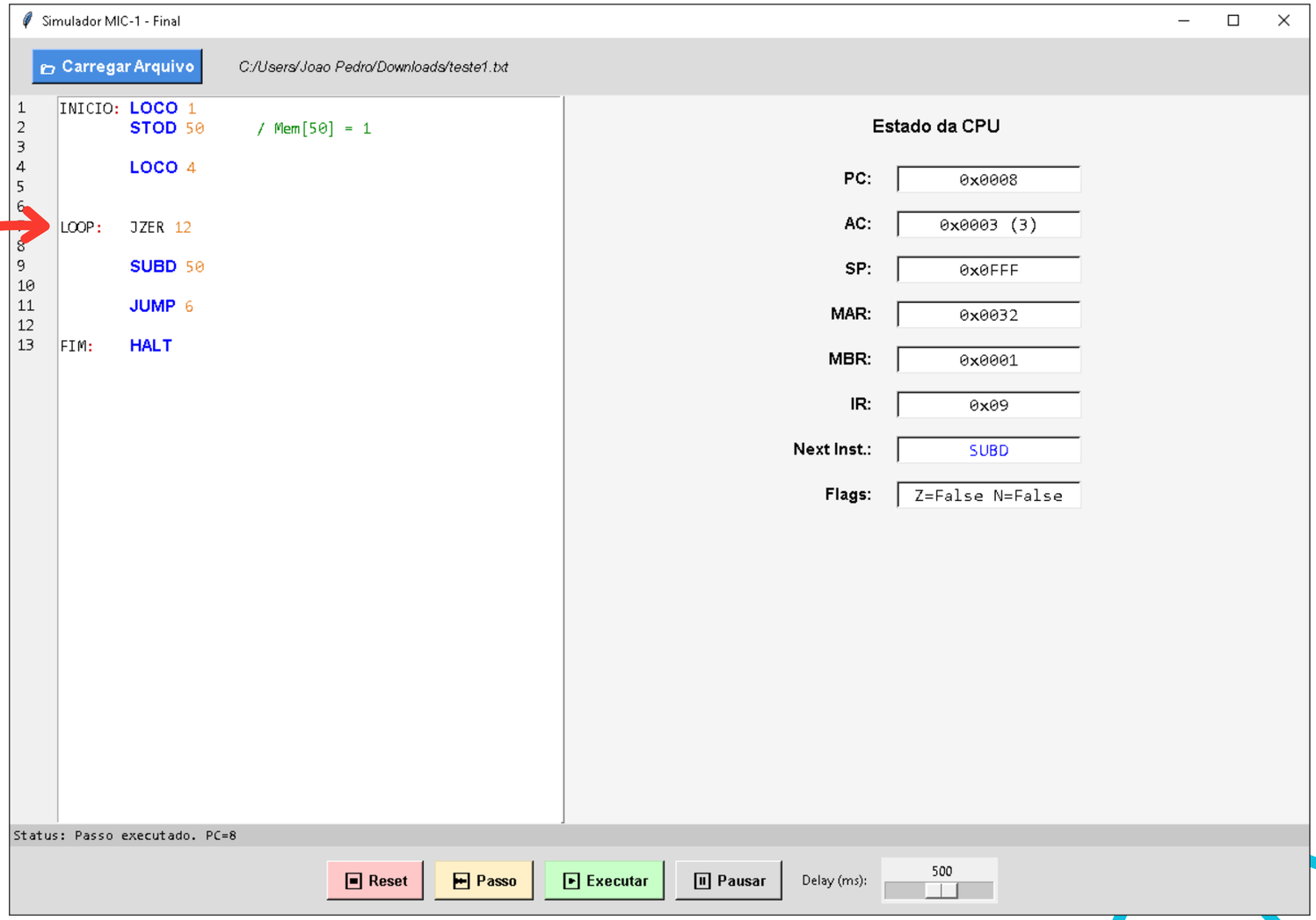
Subtrai o valor que esta no endereco 50 (4 - 1)



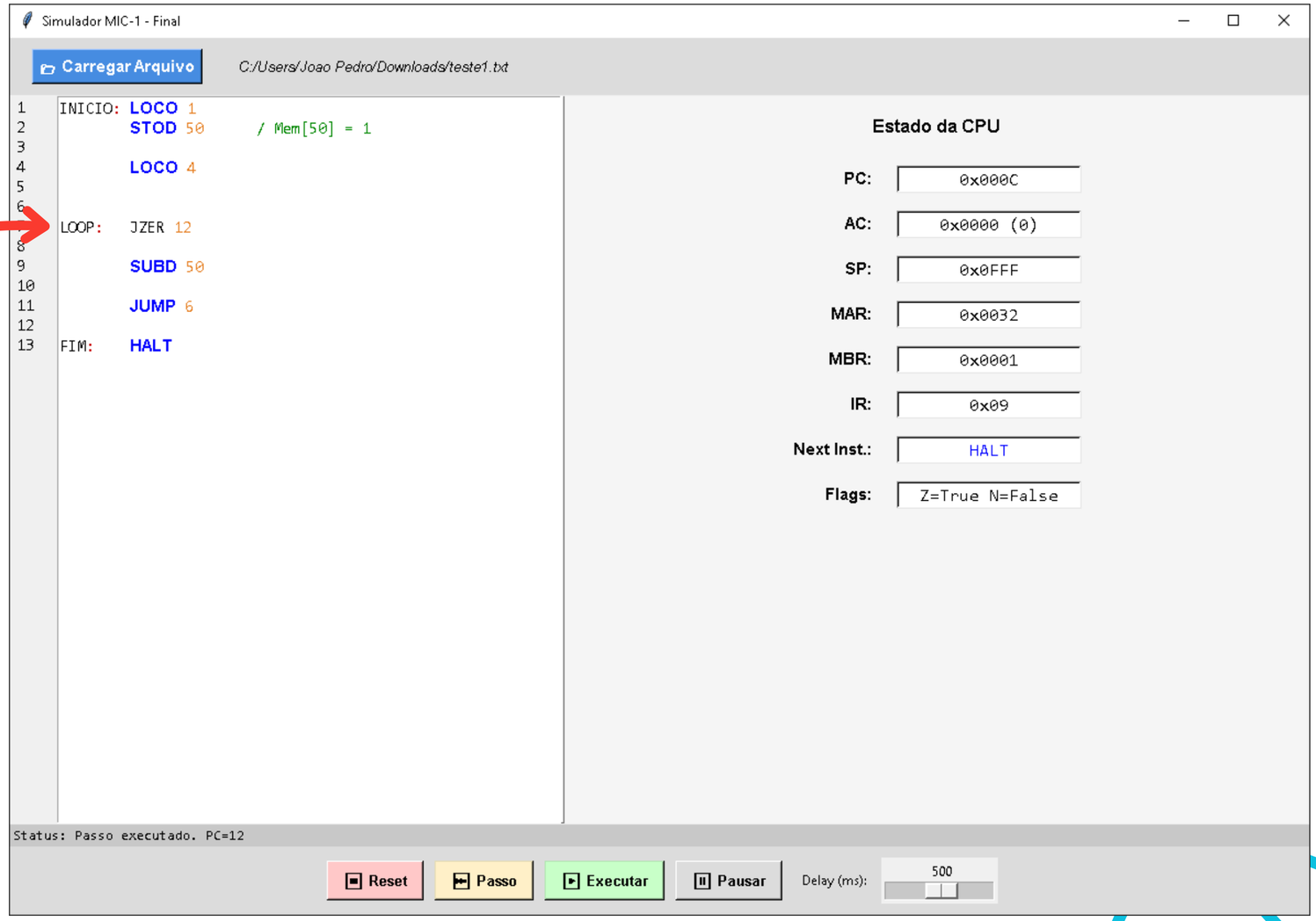
Volta para o inicio do LOOP (endereço 6)



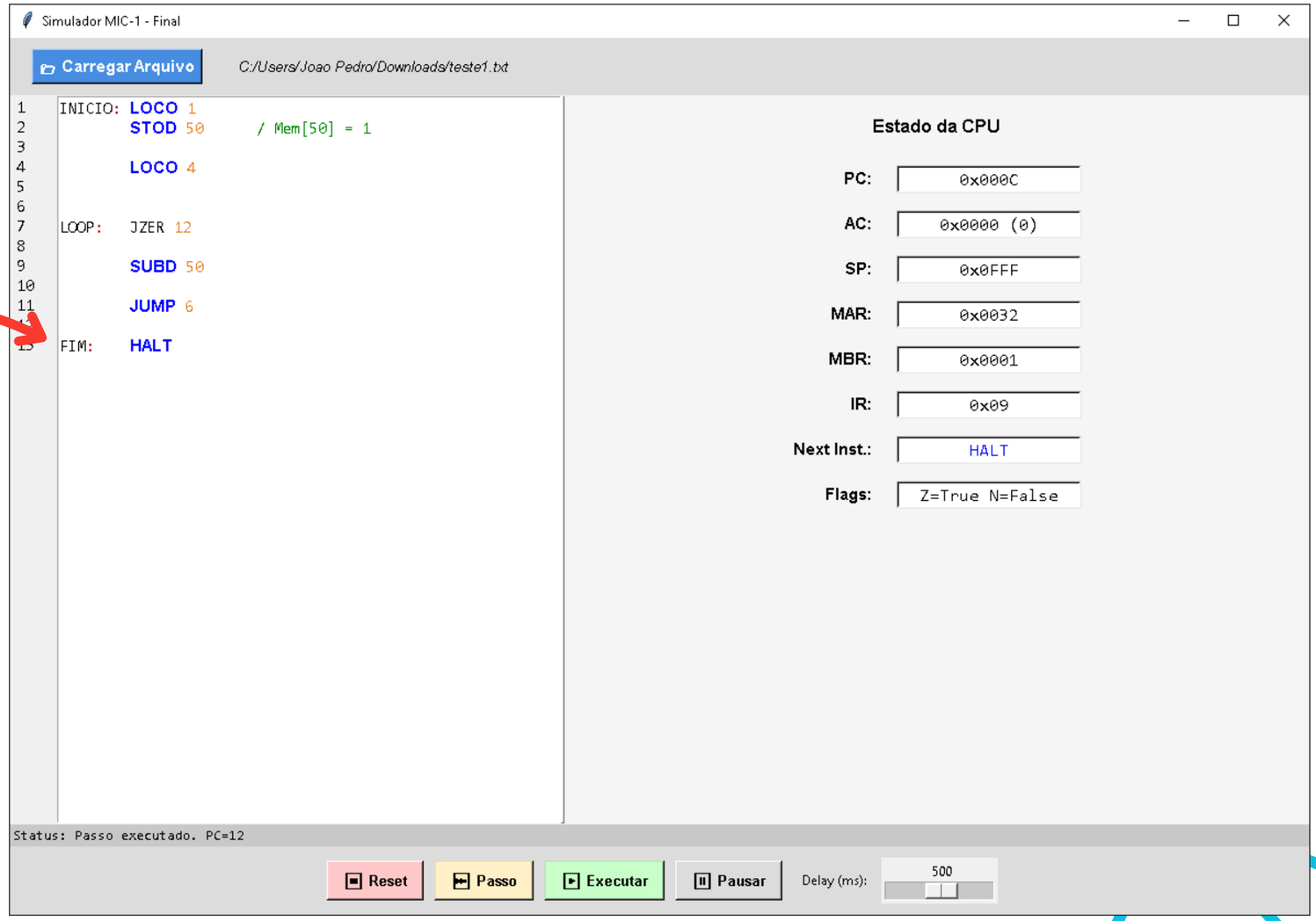
Repete o LOOP até AC = 0



AC = 0, pula para o final (endereço 12)



Fim do programa



FIM DA APRESENTAÇÃO

Obrigado pela atenção de todos!