TWO PASS ASSEMBLER GUI Documentation

Project Done By: Alstin Gloria Chacko

CSA – 28

User Requirements

Software Requirements

Operating System:

- Windows, macOS, or Linux (Python compatible).

Hardware Requirements

Processor:

- 1 GHz or faster processor.
- Intel Core i3 (3rd generation or newer) or Intel Pentium.
- AMD A4 series or higher.

RAM:

- Minimum 2 GB (4 GB recommended for optimal performance).

Storage:

- At least 100 MB of free disk space for installation and data files.

Display:

- 1024x768 resolution or higher for optimal GUI experience.

Programmer Requirements

Software:

1. **Operating System:**
   - Windows, macOS, or Linux (same as user requirements).
2. **Python:**
   - Python 3.6 or higher, with Tkinter installed.
3. **Development Environment:**
   - An IDE or code editor for development (e.g., PyCharm, Visual Studio Code, or Jupyter Notebook).
4. **Version Control System:**
   - Git for version control (optional but recommended).
5. **Libraries:**
   - No additional libraries are needed beyond Python and Tkinter for this project.

Hardware:

1. **Processor:**
   - Similar to user requirements; any modern processor would suffice.
2. **Memory (RAM):**
   - 8 GB or more recommended for smoother development experience.
3. **Storage:**
   - At least 200 MB of free disk space for the development environment and project files.
4. **Display:**
   - Minimum 1366 x 768 resolution; dual monitors are beneficial for multitasking.

Overview

The TWO PASS ASSEMBLER is a graphical user interface (GUI) application built using Python's Tkinter library. It facilitates the process of assembly language translation by performing two passes over the input assembly code and generating the corresponding output files.

Usage

Note: (This is for developers, not required if using the exe file) To run the application, ensure you have Python installed along with Tkinter. You can install Tkinter using pip if it’s not already included in your Python installation.

pip install tk

1. Upload Input File: Click the "upload input file" button to select the assembly input file. The selected file path will be displayed.

2. Upload Optab File: Click the "upload optab file" button to select the operation table file. The selected file path will be displayed.

3. Run Pass 1: Click the "PASS 1" button to process the input assembly code.

4. Run Pass 2: Click the "PASS 2" button to generate the final object code.

Note: The output files will be generated as files in your working directory.

Pass1 generates output_gui.txt, length_gui.txt, symtab_gui.txt

Pass2 generates object_gui.txt, final_output_gui.txt

Functions

1. uploadInputFile()

- Opens a file dialog to select the input assembly file.
- Updates label1 to show the selected file path.

2. uploadOptabFile()

- Opens a file dialog to select the operation table file.
- Updates label2 to show the selected file path.

3. pass1()

- Reads the input assembly file line by line.
- Generates an output file (output_gui.txt) and a symbol table (symtab_gui.txt).
- Calculates the starting location and length of the program.
- Updates the location counter based on the opcode and directives.

4. pass2()

- Reads the output from Pass 1 and the symbol table to generate the final object code.

- Writes the object code to object_gui.txt and additional details to the final_output_gui.txt.

GUI Components

- Buttons:
    - upload input file: Opens a file dialog to select the assembly code file.
    - upload optab file: Opens a file dialog to select the operation table file.
    - PASS 1: Executes the first pass of assembly.
    - PASS 2: Executes the second pass of assembly.
- Labels:
    - Displays the status of file selections.

Sample Text File to Be Inputted

Note: The delimiter used in the input and optab files should be space (/t-tab spaces or other characters other than a single space are not allowed).

Note: Also ensure that where labels are not present it is replaced by ** symbol.

Input File

** START 2000

** LDA FIVE

** STA ALPHA

** LDCH CHARZ

** STCH C1

ALPHA RESW 2

FIVE WORD 5

CHARZ BYTE C'Z'

C1 RESB 1

** END **

Optab File

LDA 03

STA 0f

LDCH 53

STCH 57

END *

Sample Text Files Outputted

Symtab File

ALPHA 2012

FIVE 2018

CHARZ 2021

C1 2022

Intermediate File

	**	START	2000
2000	**	LDA	FIVE
2003	**	STA	ALPHA
2006	**	LDCH	CHARZ
2009	**	STCH	C1
2012	ALPHA	RESW	2
2018	FIVE	WORD	5
2021	CHARZ	BYTE	C'Z'
2022	C1	RESB	1
2023	**	END	**

Length File

23

Output File

```
        **      START   2000
2000    **      LDA     FIVE    332018
2003    **      STA     ALPHA   442012
2006    **      LDCH    CHARZ   532021
2009    **      STCH    C1      572022
2012    ALPHA   RESW    2
2018    FIVE    WORD    5       000005
2021    CHARZ   BYTE    C'Z'    5a
2022    C1      RESB    1
2023    **      END     **
```

Object File

```
H^***^002000^002023
T^002000^10^332018^442012^532021^572022^000005^5a
E^002000
```

Program

```
# import files

import tkinter as tk
from tkinter import filedialog                  # importing filedialog for getting filename of selected file



# Function for showing which file is selected (input file)
def uploadInputFile():
    global input_file_path
    input_file_path = filedialog.askopenfilename()

    # checking if file is selected
    if input_file_path:
        label1.config(text=f"Selected File: {input_file_path}")


# Function for showing which file is selected (optab file)
def uploadOptabFile():
    global optab_file_path
    optab_file_path = filedialog.askopenfilename()

    # checking if file is selected
    if optab_file_path:
        label2.config(text=f"Selected File: {optab_file_path}")

# pass 1 function
def pass1():
    global loctor,start,length,count
```

```
      label3.config(text=f"{current_text1}{loctor}\t{inp[0]}\t{inp[1]}\t{inp[2]}")

      if inp[0] == "**":
          count += 3
      # to check if it is a label and then write it to symtab file
      if inp[0] != "**":
          f4.write(f"{inp[0]}\t{loctor}\n")
          current_text2 = label4.cget("text")
          label4.config(text=f"{current_text2}{inp[0]}\t{loctor}\n")

      f2 = open(optab_file_path,"r")
      inp2 = []
      lines2 = f2.readline()
      fields2 = lines2.split(' ')
      inp2.extend(fields2)

      while inp2[0] != "END":
          if inp[1] == inp2[0]:
              loctor += 3
              break
          inp2 = []
          lines2 = f2.readline()
          fields2 = lines2.split(' ')
          inp2.extend(fields2)


      f2.close()                                      # file is closed here to ensure that the optab file is read from start in the ne

      if inp[1] == "WORD":
```

```
            loctor += 3
            count += 3
        elif inp[1] == "RESW":
            loctor +=(3*(int(inp[2].rstrip())))
        elif inp[1] == "BYTE":
            loctor += 1
            count += 1
        elif inp[1] == "RESB":
            loctor += int(inp[2].rstrip())

        inp = []
        lines = f1.readline()
        fields = lines.split(' ')
        inp.extend(fields)

    f3.write(f"{loctor}\t{inp[0]}\t{inp[1]}\t{inp[2]}")
    current_text1 = label3.cget("text")
    label3.config(text=f"{current_text1}{loctor}\t{inp[0]}\t{inp[1]}\t{inp[2]}")


    # closing files
    f1.close()
    f3.close()
    f4.close()

    length = loctor - start                                  # calculating the length
    f5.write(str(length))
    f5.close()
```

```
# pass 2 function
def pass2():
    global prevaddr,finaddr,diff,start,leng,actual_len,ad,fields2
    mnemonic = ["LDA","STA","LDCH","STCH"]
    code = ["33","44","53","57"]
    j = 0

    # opening files
    f1 = open("final_output_gui.txt","w")
    f2 = open("symtab_gui.txt","r")
    f3 = open("output_gui.txt","r")
    f4 = open("object_gui.txt","w")

    inp = []
    lines = f3.readline()
    fields = lines.split('\t')
    inp.extend(fields)

    inp = []                                                 # reading the second line since we need second line first
    lines = f3.readline()
    fields = lines.split('\t')                               # seperating each item in file using tab space as the delimiter
    inp.extend(fields)

    while inp[2] != "END":
        prevaddr = int(inp[0])
        inp = []
        lines = f3.readline()
        fields = lines.split('\t')
        inp.extend(fields)
```

```
    finaddr = inp[0]
    f3.close()
    f3 = open("output_gui.txt","r")

    inp = []
    lines = f3.readline()
    fields = lines.split('\t')
    inp.extend(fields)

    if inp[2] == "START":
        f1.write(f"\t{inp[1]}\t{inp[2]}\t{inp[3]}")
        label3.config(text=f"\t{inp[1]}\t{inp[2]}\t{inp[3]}")
        f4.write(f"H^{inp[1]}^00{inp[3]}^00{finaddr}\n")
        label4.config(text=f"H^{inp[1]}^00{inp[3].rstrip()}^00{finaddr}\n")
        inp = []
        lines = f3.readline()
        fields = lines.split('\t')
        inp.extend(fields)
        start = int(inp[0])
        diff = count
        f4.write(f"T^00{inp[0]}^{hex(diff)[2:]}")
        current_text2 = label4.cget("text")
        label4.config(text=f"{current_text2}T^00{inp[0]}^{hex(diff)[2:]}")
    while inp[2] != "END":
        if inp[2].rstrip() == "BYTE":                                   # rstrip is used to remove the \n at the end
            f1.write(f"{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3].rstrip()}\t")
            current_text1 = label3.cget("text")
            label3.config(text=f"{current_text1}{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3].rstrip()}\t")
            leng = len(inp[3])
```

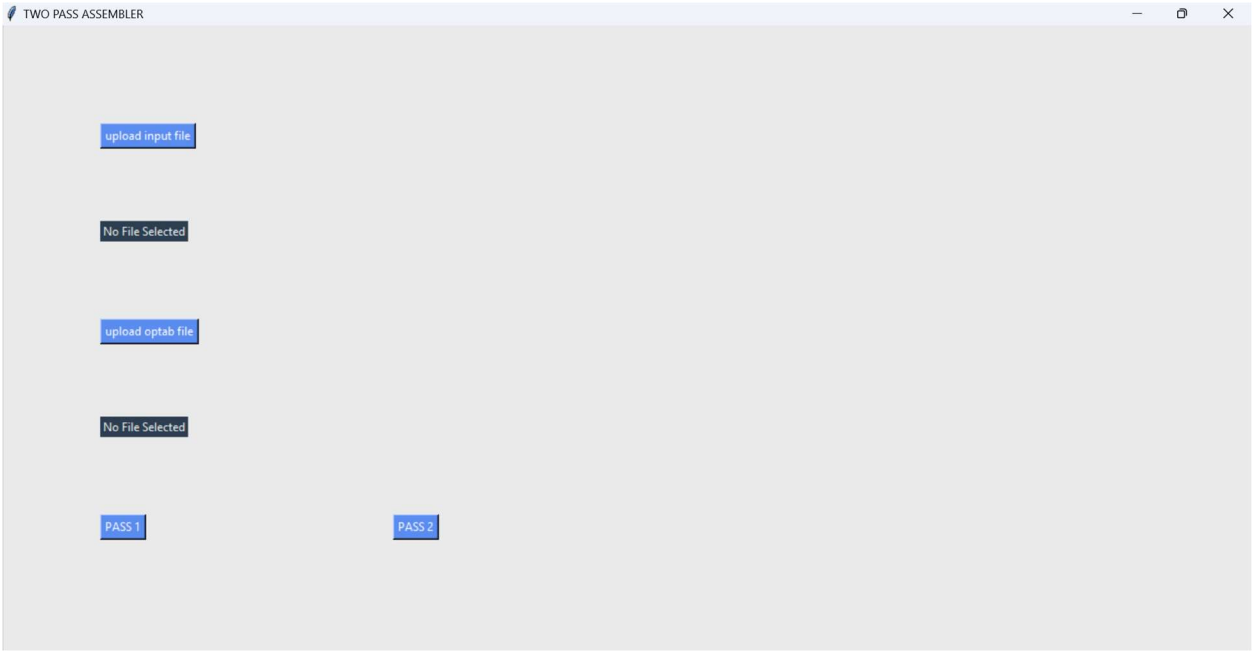
```
            actual_len = leng - 3
            f4.write("^")
            current_text2 = label4.cget("text")
            label4.config(text=f"{current_text2}^")
            for i in range(2,actual_len+1):
                ad = str(hex(ord(inp[3][i]))[2:])                       # hex used for hex conversion, ord for ascii value,[2:] is
                f1.write(ad)
                current_text1 = label3.cget("text")
                label3.config(text=f"{current_text1}{ad}")
                f4.write(ad)
                current_text2 = label4.cget("text")
                label4.config(text=f"{current_text2}{ad}")
            f1.write("\n")
            current_text1 = label3.cget("text")
            label3.config(text=f"{current_text1}\n")
        elif inp[2] == "WORD":
            leng = len(inp[2])
            a = str(int(inp[3]))
            f1.write(f"{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3].rstrip()}\t00000{a}\n")
            current_text1 = label3.cget("text")
            label3.config(text=f"{current_text1}{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3].rstrip()}\t00000{a}\n")
            f4.write(f"^00000{a}")
            current_text2 = label4.cget("text")
            label4.config(text=f"{current_text2}^00000{a}")
        elif (inp[2] == "RESB" or inp[2] == "RESW"):
            f1.write(f"{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3]}")
            current_text1 = label3.cget("text")
            label3.config(text=f"{current_text1}{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3]}")
        else:
```

```
            while inp[2] != mnemonic[j]:
                j += 1
            if inp[2] == "COPY":
                f1.write(f"{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3].rstrip()}\t{code[j]}")
                current_text1 = label3.cget("text")
                label3.config(text=f"{current_text1}{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3].rstrip()}\t{code[j]}")
            else:
                f2.seek(0)                                          # seek(0) is used to change the file pointer to start of
                inp2 = []
                lines2 = f2.readline()
                fields2 = lines2.split('\t')
                inp2.extend(fields2)
                while inp[3].rstrip() != inp2[0]:
                    inp2 = []
                    lines2 = f2.readline()
                    fields2 = lines2.split('\t')
                    inp2.extend(fields2)



                f1.write(f"{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3].rstrip()}\t{code[j]}{inp2[1]}")
                current_text1 = label3.cget("text")
                label3.config(text=f"{current_text1}{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3].rstrip()}\t{code[j]}{inp2[1]}")
                f4.write(f"^{code[j]}{inp2[1].rstrip()}")
                current_text2 = label4.cget("text")
                label4.config(text=f"{current_text2}^{code[j]}{inp2[1].rstrip()}")

        inp = []
        lines = f3.readline()
```

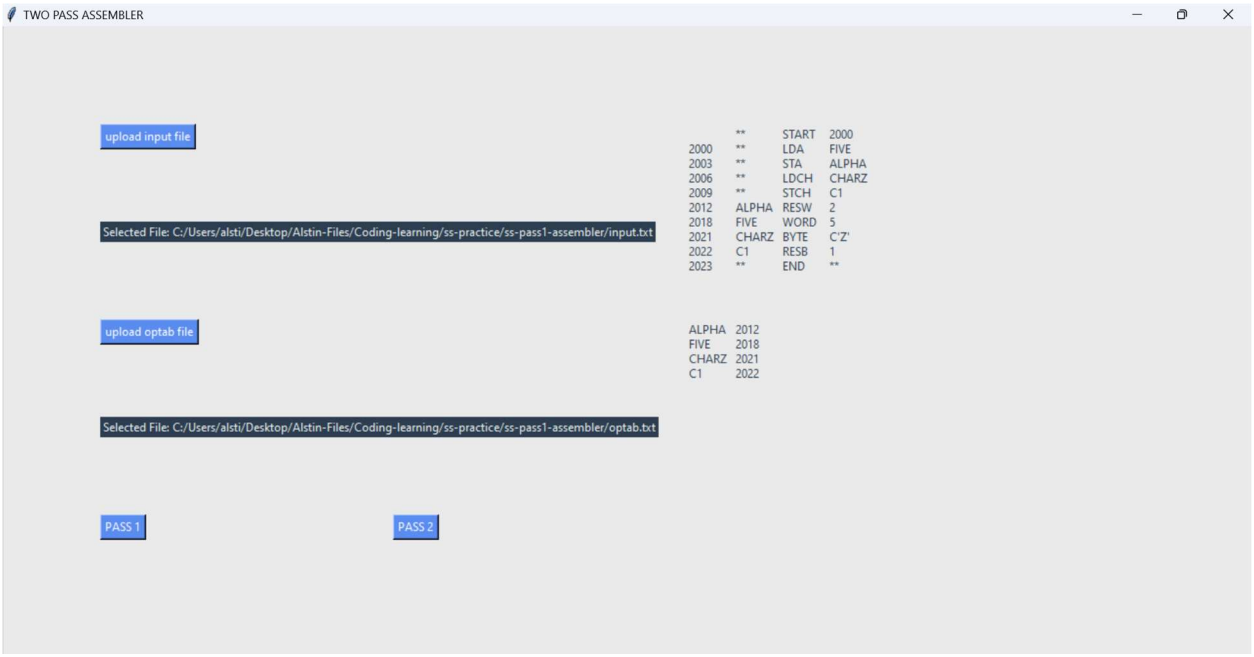
```
        fields = lines.split('\t')
        inp.extend(fields)

    (variable) current_text1: Any                    inp[2]}\t{inp[3]}")
    current_text1 = label3.cget("text")
    label3.config(text=f"{current_text1}{inp[0]}\t{inp[1]}\t{inp[2]}\t{inp[3]}")
    f4.write(f"\nE^00{start}")
    current_text2 = label4.cget("text")
    label4.config(text=f"{current_text2}\nE^00{start}")

    # closing files
    f1.close()
    f2.close()
    f3.close()
    f4.close()




# Creating the window
root = tk.Tk()
root.title("TWO PASS ASSEMBLER")                # Title for the Application
root.config(bg="#EAEAEA")                       # backgroundcolor for the window

#creating the Buttons and Labels
inputFileButton = tk.Button(root,text="upload input file",command=uploadInputFile)
label1 = tk.Label(root,text="No File Selected")
optabFileButton = tk.Button(root,text="upload optab file",command=uploadOptabFile)
label2 = tk.Label(root,text="No File Selected")
```

```
passbutton = tk.Button(root,text="PASS 1",command=pass1)
passbutton2 = tk.Button(root,text="PASS 2",command=pass2)
label3 = tk.Label(root,text="",justify="left")
label4 = tk.Label(root,text="",justify="left")

# Positioning the Buttons and Labels
inputFileButton.place(x=100,y=100)
optabFileButton.place(x=100,y=300)
passbutton.place(x=100,y=500)
passbutton2.place(x=400,y=500)
label1.place(x=100,y=200)
label2.place(x=100,y=400)
label3.place(x=700,y=100)
label4.place(x=700,y=300)

# setting the background and foreground colors for buttons and labels
inputFileButton.config(bg="#5B8DF1", fg="#FFFFFF")
optabFileButton.config(bg="#5B8DF1", fg="#FFFFFF")
passbutton.config(bg="#5B8DF1", fg="#FFFFFF")
passbutton2.config(bg="#5B8DF1", fg="#FFFFFF")
label1.config(bg="#2C3E50", fg="#ECF0F1")
label2.config(bg="#2C3E50", fg="#ECF0F1")
label3.config(bg="#EAEAEA", fg="#2C3E50")
label4.config(bg="#EAEAEA", fg="#2C3E50")


root.mainloop()                                # starts the Tkinter event loop
```

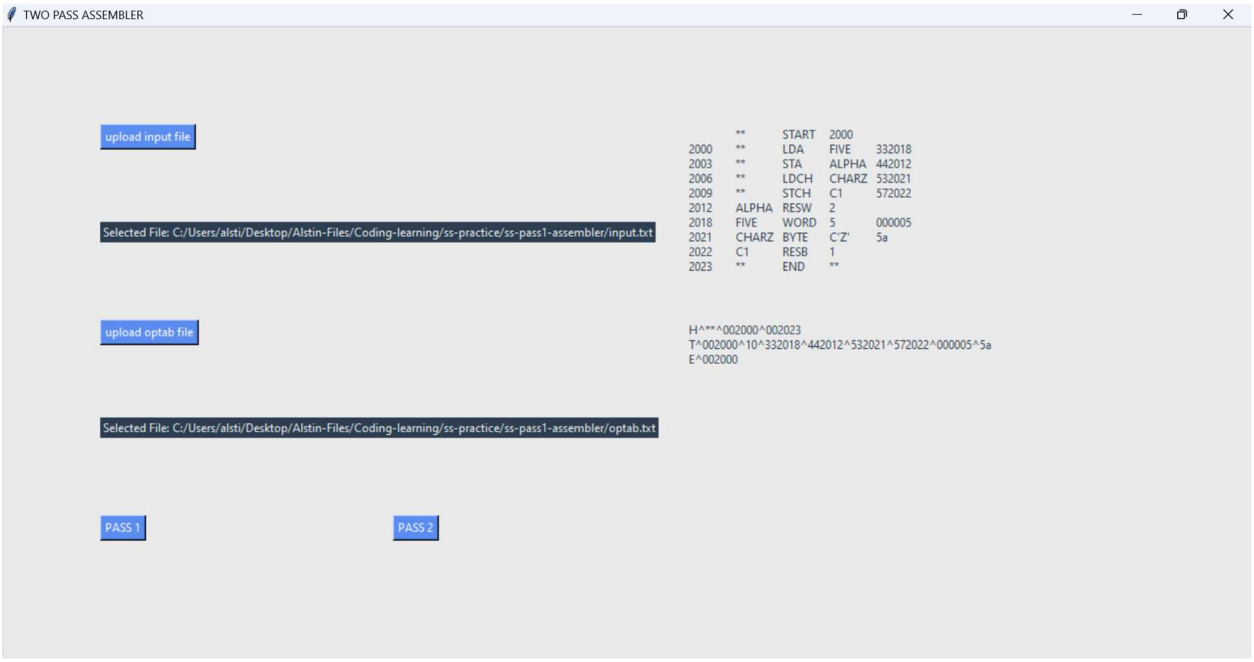
GUI Sample



After Pass 1



After Pass 2



Github Release Link: https://github.com/ALSTINGLORIA/Two-Pass-Assembler-GUI/releases