Experiment 9 **Surya Narayanan**

Pass two of two pass assembler

s5 csb 48

Step 1: START

Step 2: Include necessary header files.

Step 3: Declare necessary file pointers.

Step 4: Declare and initialize necessary variables and arrays.

Step 5: Open the input file in readmode and output and symbol file in write mode.

Step 6: Read the contents of input file.

Step 7: Perform the following if opcode is 'START'

Step 7.1: Get the operand value as integer (start)

Step 7.2: Set LOCCTR as start.

Step 7.3: Store the label, opcode and operand to output file.

Step 7.4: Read the next line.

Step 8: If opcode ≠ START, set LOCCTR as 0.

Step 9: Perform the following till opcode is not found as 'END'

Step 9.1 - Set j=0

Step 9.2 - Store LOCCTR to output file.

Step 9.3 - if label is not '* *'

Step 9.3.1 - write the label and its LOCCTR to symtab file.

Step 9.4 :- Perform the following if mnemonic [j] ≠ END

Step 9.4.1 - If opcode and mnemonic [j] are found to be equal

Step 9.4.1.1 : Increment LOCCTR by 3, and display its value.

Step 9.4.2 - Increment j and go to step 9.4.

Step 9.5 - of opcode = WORD

Step 9.5.1 - increment LOCCTR by 3

Step 9.6 - of opcode = RESW

Step 9.6.1 - Set LOCCTR as LOCCTR + 3* value of operand

Step 9.6.2 - Set count as count + 3* operand value.

Step 9.7 - of opcode = BYTE

Step 9.7.1 - set LOCTR as LOCTR + length of operand.

Step 9.8 - Else print a blank space

Step 9.9 - store the label, opcode and opcode to output file, Read the next input line Go to Step 9.

Step 10 - Store the value of LOCCTR, label operand to output file.

Step 11 - Display program length as LOCCTR - start location

Step 12 - set length as LOCCTR - start location. and LOCCTR as the final address.

Step 13 - Store the value of program length count.

Step 14 - Close all the files.

Step 15 - Open output file in read mode and final file in write mode using pointers INTERMEDIATE and final.

Step 16 - Read the contents of output file and set start address as operand value

Step 17 - If opcode = START

Step 17.1 - Store the label, opcode, address and operand to final file

Step 17.2 - Read the next line and go to step 17.

Step 18 - perform the following till opcode ≠ END.

Step 18.1 : If '*' is encountered.

Step 18.1.1 store the address operand, opcode, label and return value from functions get Mnemonic code() and findSYMTAB into final file.

Step 18.2 - Get the next input line and go to step 18.

Step 18.3 - If opcode = BYTE

Step 18.3.1 – the values of address, label, operand opcode to final file

Step 18.3.2 – Do the following till i < Length of operand - 1

Step 18.3.2.1: Store the value of corresponding operand to final file.

Step 18.3.3: Move to a new line and read the next input line from output file.

Step 18.4 – If opcod = WORD

Step 18.4.1 – Store the address, label, opcode, operand to final file.

Step 18.4.2 – Read the next line of output file.

Step 18.5 – Else read the next line of output line.

Step 19 – store the address, label, operand and opcode to final file.

Step 20: close the files

Step 21: STOP

Find SYMTABL)

Step 1: START

Step 2: Declare a file pointer for symtab file

Step 3: Declare arrays to store label and address

Step 4: Open symtab file in read mode

Step 5: Read the contents of file

Step 6. Do the following till conditions is true

Step 61 – if file end is reached close the symtab file and go to step 7.

Step 62 – if symtab label and mnemonic label is same, close the symtab file and return the corresponding address.

Step 6.3 - Read the next input line of syntax. Go to 6.

Step 7: STOP.

get mnemonic code()

Step 1 : start()

Step 2 : if mnemonic is LDA

Step 2.1 - return 00

Step 3 : if mnemonic = STA

Step 3.1 - Return 23

Step 4. if mnemonic = LDCL

Step 4.1 : return 01

Steps 5 . if mnemonic = ADD

Step 5.1 - return 18

Step 6: . Otherwise return -1

Step . 7 : STOP.

Program

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int findSYMTAB(char findLabel[])
{
    FILE *SYMTAB;
    char label[30], addr[30];
    SYMTAB = fopen("symtab.txt", "r");
    fscanf(SYMTAB, "%s%s", label, addr);

    while (1)
    {
        if (feof(SYMTAB))
        {
            fclose(SYMTAB);
            break;
        }
        if (strcmp(findLabel, label) == 0)
        {
            fclose(SYMTAB);
            return atoi(addr);
        }
        fscanf(SYMTAB, "%s%s", label, addr);
    }
}

int getMnemonicCode(char mnemonic[])
{

    if (strcmp(mnemonic, "LDA") == 0)
        return 0x00;
    else if (strcmp(mnemonic, "STA") == 0)
        return 0x0C;
    else if (strcmp(mnemonic, "LDCH") == 0)
        return 0x01;
    else if (strcmp(mnemonic, "ADD") == 0)
        return 0x18;
    else
```

```
        return -1;
}

int main()
{
    FILE *INPUT, *OUTPUT, *SYMTAB, *INTERMEDIATE, *FINAL;
    char mnemonic[10][10] = {"START", "LDA", "STA", "ADD", "END"};
    int LOCCTR, start = 0, j = 0, i, length, Tlength, count = 0,
finalAddress, startAddr;
    char label[20], opcode[20], operand[20], address[20];
    INPUT = fopen("input.txt", "r");
    OUTPUT = fopen("output.txt", "w");
    SYMTAB = fopen("symtab.txt", "w");
    fscanf(INPUT, "%s%s%s", label, opcode, operand);
    if (strcmp(opcode, "START") == 0)
    {

        start = atoi(operand);
        LOCCTR = start;
        fprintf(OUTPUT, "\t%s\t%s\t%s\n", label, opcode, operand);
        fscanf(INPUT, "%s%s%s", label, opcode, operand);
    }
    else
        LOCCTR = 0;
    while (strcmp(opcode, "END") != 0)
    {

        j = 0;
        fprintf(OUTPUT, "%d", LOCCTR);
        if (strcmp(label, "**") != 0)
            fprintf(SYMTAB, "\t%s\t%d\n", label, LOCCTR);
        while (strcmp(mnemonic[j], "END") != 0)

        {

            if (strcmp(mnemonic[j], opcode) == 0)

            {

                LOCCTR += 3;
```

```
            }
            j++;
        }
        if (strcmp(opcode, "WORD") == 0)
            LOCCTR += 3;
        else if (strcmp(opcode, "RESW") == 0)

        {

            LOCCTR = LOCCTR + (3 * atoi(operand));
            count += (3 * atoi(operand));
        }
        else if (strcmp(opcode, "RESB") == 0)

        {

            LOCCTR = LOCCTR + atoi(operand);
            count += atoi(operand);
        }
        else if (strcmp(opcode, "BYTE") == 0)

        {

            LOCCTR = LOCCTR + (strlen(operand));
        }
        else

            printf(" ");

        fprintf(OUTPUT, "\t%s\t%s\t%s\n", label, opcode, operand);
        fscanf(INPUT, "%s%s%s", label, opcode, operand);
    }

    fprintf(OUTPUT, "%d", LOCCTR);
    fprintf(OUTPUT, "\t%s\t%s\t%s\n", label, opcode, operand);
    printf("\n\n THE LENGTH OF THE PROGRAM IS %d", LOCCTR - start);
    length = LOCCTR - start;
    finalAddress = LOCCTR;
    Tlength = length - count;
    fclose(INPUT);
```

```
    fclose(OUTPUT);
    fclose(SYMTAB);

    INTERMEDIATE = fopen("output.txt", "r");
    FINAL = fopen("final.txt", "w");
    fscanf(INTERMEDIATE, "%s%s%s", label, opcode, operand);
    startAddr = atoi(operand);
    if (strcmp(opcode, "START") == 0)
    {

        fprintf(FINAL, "%s\t%s\t%s\t\n", label, opcode, operand);
        fscanf(INTERMEDIATE, "%s%s%s%s", address, label, opcode, operand);
    }
    while (strcmp(opcode, "END") != 0)
    {

        if (strcmp(label, "**") == 0)

        {

            fprintf(FINAL, "%s\t%s\t%s\t%x%d\n", address, label,
opcode, operand, getMnemonicCode(opcode), findSYMTAB(operand));
            fscanf(INTERMEDIATE, "%s%s%s%s", address, label, opcode,
operand);
        }

        else if (strcmp(opcode, "BYTE") == 0)

        {

            fprintf(FINAL, "%s\t%s\t%s\t%s", address, label, opcode,
operand);

            for (i = 2; i < (strlen(operand) - 1); i++)
            {

                fprintf(FINAL, "%d", operand[i]);
            }
            fprintf(FINAL, "\n");
```

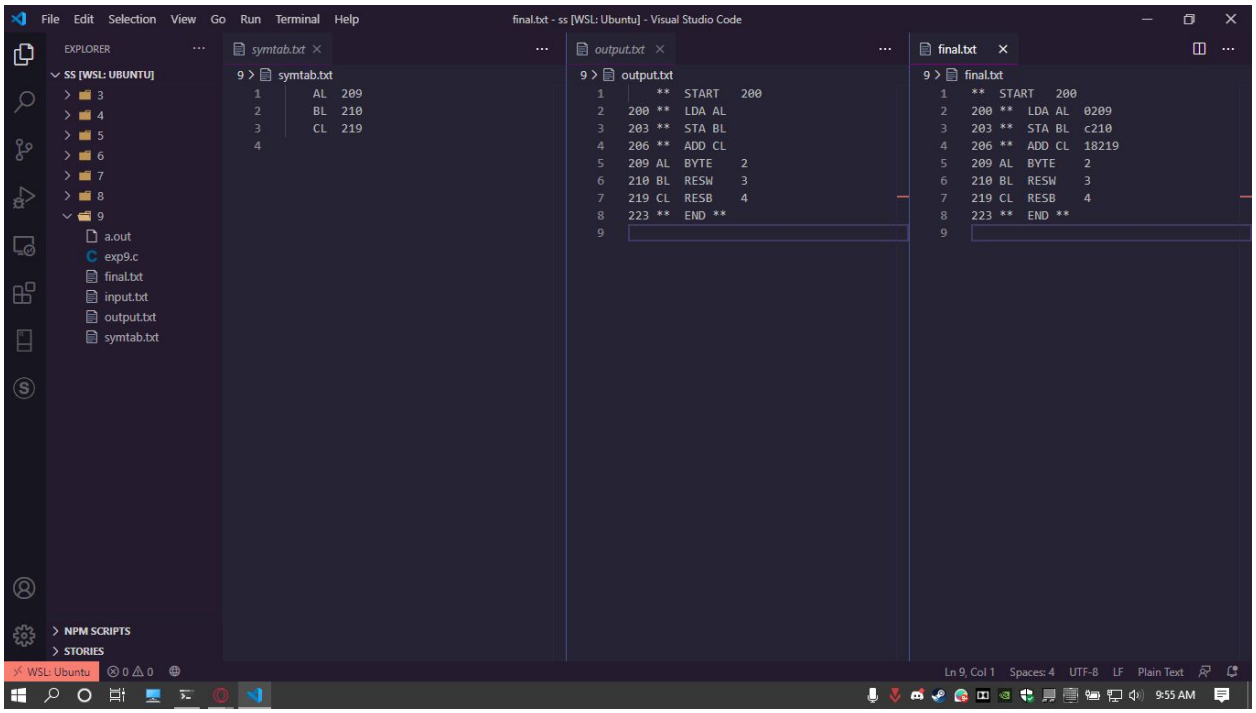
```
            fscanf(INTERMEDIATE, "%s%s%s%s", address, label, opcode,
operand);
        }
        else if (strcmp(opcode, "WORD") == 0)

        {

            fprintf(FINAL, "%s\t%s\t%s\t%s\n", address, label, opcode,
operand);

            fscanf(INTERMEDIATE, "%s%s%s%s", address, label, opcode,
operand);
        }
        else if (strcmp(opcode, "RESW") == 0)

        {

            fprintf(FINAL, "%s\t%s\t%s\t%s\n", address, label, opcode,
operand);

            fscanf(INTERMEDIATE, "%s%s%s%s", address, label, opcode,
operand);
        }
        else if (strcmp(opcode, "RESB") == 0)

        {

            fprintf(FINAL, "%s\t%s\t%s\t%s\n", address, label, opcode,
operand);

            fscanf(INTERMEDIATE, "%s%s%s%s", address, label, opcode,
operand);
        }
        else

            fscanf(INTERMEDIATE, "%s%s%s%s", address, label, opcode,
operand);
    }
    fprintf(FINAL, "%s\t%s\t%s\t%s\n", address, label, opcode, operand);
    fclose(INTERMEDIATE);
```

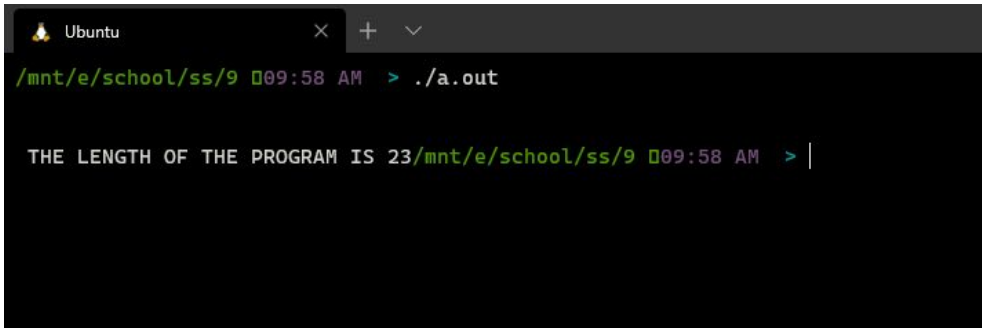
```
    fclose(FINAL);
    return (0);
}
```

Input

```
9 > input.txt
1   **  START   200
2   **  LDA AL
3   **  STA BL
4   **  ADD CL
5   AL  BYTE    2
6   BL  RESW    3
7   CL  RESB    4
8   **  END **
```

Output




Ubuntu
/mnt/e/school/ss/9 09:58 AM > ./a.out
THE LENGTH OF THE PROGRAM IS 23/mnt/e/school/ss/9 09:58 AM >