**EXPERIMENT -6**

**Single Pass Assembler**

## Aim

To implement a single pass assembler

**Program**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct

{

    int address;

    char label[40], opcode[40], operand[40], objcode[40];

} pgm;

struct

{

    char opcode[40];

    int value;

} optab[1000];

int optab\_len = 0;

struct

{

    char label[40];

    int value;

} symtab[1000];

int symtab\_len = 0;

struct LocationPtr

{

    int location;

    struct LocationPtr \*next;

};

struct ForwardRef

{

    char label[40];

    struct LocationPtr \*head;

} forward\_ref[1000];

int forward\_ref\_len = 0;

int searchOptab(char opcode[40])

{

    for (int i = 0; i < optab\_len; i++)

    {

        if (strcmp(opcode, optab[i].opcode) == 0)

            return i;

    }

    return -1;

}

int searchSymtab(char label[40])

{

    for (int i = 0; i < symtab\_len; i++)

    {

        if (strcmp(label, symtab[i].label) == 0)

            return i;

    }

    return -1;

}

void writeTextRecord(FILE \*temp\_ptr, FILE \*record\_ptr, int \*text\_len, int \*text\_starting\_addr, int new\_starting\_addr)

{

    if (\*text\_len == 0)

        return;

    char str[100];

    fclose(temp\_ptr);

    fprintf(record\_ptr, "\nT^%06X^%02X", \*text\_starting\_addr, \*text\_len);

    temp\_ptr = fopen("op/temp.txt", "r");

    fscanf(temp\_ptr, "%s", str);

    fprintf(record\_ptr, "%s", str);

    fclose(temp\_ptr);

    \*text\_len = 0;

    temp\_ptr = fopen("op/temp.txt", "w");

    \*text\_starting\_addr = new\_starting\_addr;

}

void checkTextRecordLimit(FILE \*temp\_ptr, FILE \*record\_ptr, int \*text\_len, int \*text\_starting\_addr, int offset, int new\_starting\_addr, int \*obj\_staring\_addr)

{

    if (\*text\_starting\_addr == -1)

        \*text\_starting\_addr = new\_starting\_addr;

    if (\*obj\_staring\_addr == -1)

        \*obj\_staring\_addr = new\_starting\_addr;

    if ((\*text\_len + offset) \* 2 > 60)

    {

        writeTextRecord(temp\_ptr, record\_ptr, text\_len, text\_starting\_addr, new\_starting\_addr);

    }

}

int addToForwardRef(char label[40], int location)

{

    int i;

    struct LocationPtr \*newLocationPtr = (struct LocationPtr \*)malloc(sizeof(struct LocationPtr));

    newLocationPtr->location = location + 1;

    newLocationPtr->next = NULL;

    for (i = 0; i < forward\_ref\_len; i++)

    {

        if (strcmp(forward\_ref[i].label, label) == 0)

        {

            struct LocationPtr \*temp = forward\_ref[i].head;

            while (temp->next != NULL)

                temp = temp->next;

            temp->next = newLocationPtr;

            break;

        }

    }

    if (i == forward\_ref\_len)

    {

        strcpy(forward\_ref[i].label, label);

        forward\_ref[i].head = newLocationPtr;

        forward\_ref\_len++;

    }

}

int checkAndRemoveForwardRef(char label[40], int location, FILE \*temp\_ptr, FILE \*record\_ptr, int \*text\_len, int \*text\_starting\_addr)

{

    for (int i = 0; i < forward\_ref\_len; i++)

    {

        if (strcmpi(forward\_ref[i].label, label) == 0)

        {

            writeTextRecord(temp\_ptr, record\_ptr, text\_len, text\_starting\_addr, location);

            struct LocationPtr \*temp = forward\_ref[i].head;

            while (temp != NULL)

            {

                fprintf(record\_ptr, "\nT^%06X^%02X^%X", temp->location, 2, location);

                temp = temp->next;

            }

            break;

        }

    }

}

void main()

{

    FILE \*pgm\_ptr, \*optab\_ptr, \*record\_ptr, \*temp\_ptr, \*symtab\_ptr;

    int location\_ctr = 0, text\_len = 0, text\_starting\_addr = -1, program\_obj\_staring\_addr = -1, program\_starting\_addr;

    record\_ptr = fopen("op/record.txt", "w");

    temp\_ptr = fopen("op/temp.txt", "w");

    symtab\_ptr = fopen("op/symtab.txt", "w");

    pgm\_ptr = fopen("input/source\_code.txt", "r");

    printf("Reading Optab\n");

    optab\_ptr = fopen("input/optab.txt", "r");

    while (fscanf(optab\_ptr, "%s%X", optab[optab\_len].opcode, &optab[optab\_len].value) != EOF)

        optab\_len++;

    printf("Reading Program\n");

    printf("Generating record.txt\n");

    printf("Generating symtab.txt\n");

    while (fscanf(pgm\_ptr, "%s%s%s", pgm.label, pgm.opcode, pgm.operand) != EOF)

    {

        pgm.address = location\_ctr;

        if (strcmp(pgm.opcode, "START") == 0)

        {

            location\_ctr = strtol(pgm.operand, NULL, 16);

            program\_starting\_addr = strtol(pgm.operand, NULL, 16);

            pgm.address = location\_ctr;

        }

        else if (searchOptab(pgm.opcode) != -1 || strcmp(pgm.opcode, "WORD") == 0)

        {

            location\_ctr += 3;

        }

        else if ((strcmp(pgm.opcode, "RESW") == 0))

        {

            location\_ctr = location\_ctr + 3 \* strtol(pgm.operand, NULL, 10);

        }

        else if ((strcmp(pgm.opcode, "RESB") == 0))

        {

            location\_ctr += strtol(pgm.operand, NULL, 10);

        }

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

        {

            if (pgm.operand[0] == 'X')

                location\_ctr += (strlen(pgm.operand) - 3) / 2;

            else if (pgm.operand[0] == 'C')

                location\_ctr += (strlen(pgm.operand) - 3);

        }

        else if (strcmp(pgm.opcode, "END") == 0)

        {

        }

        else

        {

            printf("\nERROR: INVALID OPCODE, %s", pgm.opcode);

            continue;

        }

        if (searchSymtab(pgm.label) == -1 && !(strcmp(pgm.label, "\*\*") == 0))

        {

            checkAndRemoveForwardRef(pgm.label, pgm.address, temp\_ptr, record\_ptr, &text\_len, &text\_starting\_addr);

            strcpy(symtab[symtab\_len].label, pgm.label);

            symtab[symtab\_len].value = pgm.address;

            fprintf(symtab\_ptr, "%-7s %6X\n", symtab[symtab\_len].label, symtab[symtab\_len].value);

            symtab\_len++;

        }

        else if (!(strcmp(pgm.label, "\*\*") == 0))

        {

            printf("ERROR: DUPLICATE LABEL, %s\n", pgm.label);

        }

        if (strcmp(pgm.opcode, "START") == 0)

        {

            fprintf(record\_ptr, "H^%6.6s^%06s^%06X", pgm.label, pgm.operand, 0x0);

        }

        else if (searchOptab(pgm.opcode) != -1)

        {

            checkTextRecordLimit(temp\_ptr, record\_ptr, &text\_len, &text\_starting\_addr, 3, pgm.address, &program\_obj\_staring\_addr);

            int op\_pos = searchOptab(pgm.opcode);

            int sym\_pos = searchSymtab(pgm.operand);

            if (sym\_pos != -1)

            {

                fprintf(temp\_ptr, "^%02X%X", optab[op\_pos].value, symtab[sym\_pos].value);

            }

            else if (strstr(pgm.operand, ",X"))

            {

                char label[40] = "";

                strncpy(label, pgm.operand, strlen(pgm.operand) - 2);

                int sym\_pos = searchSymtab(label);

                if (sym\_pos != -1)

                {

                    int value = symtab[sym\_pos].value + 0x8000;

                    fprintf(temp\_ptr, "^%02X%X", optab[op\_pos].value, value);

                }

                else

                {

                    fprintf(temp\_ptr, "^%02X%04X", optab[op\_pos].value, 0x0);

                    addToForwardRef(pgm.operand, pgm.address);

                }

            }

            else

            {

                fprintf(temp\_ptr, "^%02X%04X", optab[op\_pos].value, 0x0);

                addToForwardRef(pgm.operand, pgm.address);

            }

            text\_len += 3;

        }

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

        {

            float temp\_len = 0;

            for (int i = 0; i < strlen(pgm.operand); i++)

            {

                if (pgm.operand[i] == 'C' || pgm.operand[i] == 'X' || pgm.operand[i] == '\'')

                    continue;

                if (pgm.operand[0] == 'C')

                    temp\_len += 2;

                else if (pgm.operand[0] == 'X')

                    temp\_len += 1;

            }

            checkTextRecordLimit(temp\_ptr, record\_ptr, &text\_len, &text\_starting\_addr, temp\_len / 2, pgm.address, &program\_obj\_staring\_addr);

            fprintf(temp\_ptr, "%s", "^");

            for (int i = 0; i < strlen(pgm.operand); i++)

            {

                if (pgm.operand[i] == 'C' || pgm.operand[i] == 'X' || pgm.operand[i] == '\'')

                    continue;

                if (pgm.operand[0] == 'X')

                    fprintf(temp\_ptr, "%C", pgm.operand[i]);

                else if (pgm.operand[0] == 'C')

                    fprintf(temp\_ptr, "%X", pgm.operand[i]);

            }

            text\_len += temp\_len / 2;

        }

        else if (strcmp(pgm.opcode, "WORD") == 0)

        {

            checkTextRecordLimit(temp\_ptr, record\_ptr, &text\_len, &text\_starting\_addr, 3, pgm.address, &program\_obj\_staring\_addr);

            fprintf(temp\_ptr, "^%06X", strtol(pgm.operand, NULL, 10));

            text\_len += 3;

        }

    }

    if (text\_len != 0)

        writeTextRecord(temp\_ptr, record\_ptr, &text\_len, &text\_starting\_addr, pgm.address);

    fprintf(record\_ptr, "\nE^%06X", program\_obj\_staring\_addr);

    printf("Computing Program Length");

    int program\_length = location\_ctr - program\_obj\_staring\_addr;

    fseek(record\_ptr, 16, SEEK\_SET);

    fprintf(record\_ptr, "%06X", program\_length);

    fclose(pgm\_ptr);

    fclose(optab\_ptr);

    fclose(symtab\_ptr);

    fclose(record\_ptr);

    fclose(temp\_ptr);

    printf("\nRemoving temp file");

    remove("op/temp.txt");

    printf("\n\nOne Pass Assembler done...");

}

**Input**

**source\_code.txt**

\*\*      START   2000

\*\*      LDA     FIVE

\*\*      STA     ALPHA

\*\*      LDCH    STRING

\*\*      STCH    C1

ALPHA   RESW    1

FIVE    WORD    5

STRING  BYTE    C'HELLO'

C1      RESB    1

\*\*      END     \*\*

**optab.txt**

LDA 00

LDX 04

ADD 18

COMP 28

DIV 24

JEQ 30

JGT 34

JLT 38

LDCH 50

MUL 20

RD  D8

STA 0C

STCH 54

STX 10

SUB 1C

TD  E0

TIX 2C

WD  DC

JSUB 48

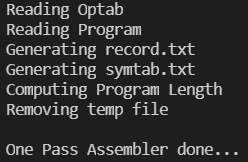
RSUB 4C

J   3C

STL 14

**Output**

**Output**



**symtab.txt**

ALPHA     200C

FIVE      200F

STRING    2012

C1        2017

**record.txt**

H^    \*\*^002000^000018

T^002000^0C^000000^0C0000^500000^540000

T^002004^02^200C

T^002001^02^200F

T^00200C^03^000005

T^002007^02^2012

T^002012^05^48454C4C4F

T^00200A^02^2017

E^002000