One pass Assembler

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

#define MAX_SYM 50
#define MAX_OBJ 4096
#define MAX_LINE 256
#define MAX_ADDR 0xFFFFF

typedef struct Ref {
    int pos;
    struct Ref *next;
} Ref;

typedef struct {
    char name[10];
    int addr;
    int defined;
    Ref *refs;
} Symbol;

Symbol symtab[MAX_SYM];
int nsym = 0;

unsigned char objbuf[MAX_OBJ];
int objidx = 0;
int locctr = 0;
int start_addr = 0;
char progname[10] = "PROG";

int hex_to_int(char *s) {
    int val = 0;
    for (int i = 0; s[i]; i++) {
        val <<= 4;
        if (s[i] >= '0' && s[i] <= '9') val |= s[i] - '0';
        else if (s[i]>='A' && s[i]<='F') val |= s[i] - 'A' + 10;
        else if (s[i]>='a' && s[i]<='f') val |= s[i] - 'a' + 10;
    }
    return val;
}

int find_sym(char *name) {
    for (int i = 0; i < nsym; i++) {
        if (strcmp(symtab[i].name, name) == 0) return i;
    }
    return -1;
}

void insert_forward(char *name) {
    int idx = find_sym(name);
    if (idx == -1) {
        if (nsym >= MAX_SYM) return;
        strcpy(symtab[nsym].name, name);
        symtab[nsym].addr = 0;
        symtab[nsym].defined = 0;
        symtab[nsym].refs = NULL;
        nsym++;
    }
}

int insert_sym(char *name, int addr) {
    int idx = find_sym(name);
    if (idx != -1) {
        if (symtab[idx].defined) {
            printf("Error: %s redefined\n", name);
            return -1;
        }
        symtab[idx].addr = addr;
        symtab[idx].defined = 1;
        Ref *r = symtab[idx].refs;
        while (r) {
            int p = r->pos;
            objbuf[p] = (addr >> 8) & 0xFF;
            objbuf[p + 1] = addr & 0xFF;
            Ref *tmp = r;
            r = r->next;
            free(tmp);
        }
        symtab[idx].refs = NULL;
        return idx;
    }
    if (nsym >= MAX_SYM) return -1;
    strcpy(symtab[nsym].name, name);
    symtab[nsym].addr = addr;
    symtab[nsym].defined = 1;
```

```
    symtab[nsym].refs = NULL;
    return nsym++;
}

void add_ref(int symidx, int pos) {
    Ref *ref = malloc(sizeof(Ref));
    ref->pos = pos;
    ref->next = symtab[symidx].refs;
    symtab[symidx].refs = ref;
}

int get_operand_addr(char *opnd) {
    char *end;
    long num = strtol(opnd, &end, 10);
    if (*end == '\0') return (int)num;

    num = strtol(opnd, &end, 16);
    if (*end == '\0') return (int)num;

    int idx = find_sym(opnd);
    if (idx == -1) {
        insert_forward(opnd);
        idx = find_sym(opnd);
        add_ref(idx, objidx);
        return 0;
    }
    if (symtab[idx].defined) return symtab[idx].addr;
    add_ref(idx, objidx);
    return 0;
}

void append_addr(int addr) {
    objbuf[objidx++] = (addr >> 8) & 0xFF;
    objbuf[objidx++] = addr & 0xFF;
}
void append_byte(unsigned char b) {
    objbuf[objidx++] = b;
}
void append_word(int val) {
    objbuf[objidx++] = (val >> 16) & 0xFF;
    objbuf[objidx++] = (val >> 8) & 0xFF;
    objbuf[objidx++] = val & 0xFF;
}
```

```
void handle_instruction(char *op, char *opnd) {
    unsigned char opc = 0;
    if (strcmp(op, "LDA") == 0) opc = 0x00;
    else if (strcmp(op, "ADD") == 0) opc = 0x18;
    else if (strcmp(op, "STA") == 0) opc = 0x0C;
    else return;
    append_byte(opc);
    int addr = get_operand_addr(opnd);
    append_addr(addr);
}

void handle_directive(char *op, char *opnd, int *loc_change) {
    *loc_change = 0;
    if (strcmp(op, "WORD") == 0) {
        int val = get_operand_addr(opnd);
        append_word(val);
        *loc_change = 3;
    } else if (strcmp(op, "RESW") == 0) {
        int n = atoi(opnd);
        *loc_change = 3 * n;
    } else if (strcmp(op, "RESB") == 0) {
        int n = atoi(opnd);
        *loc_change = n;
    } else if (strcmp(op, "BYTE") == 0) {
        if (opnd[0] == 'C' && opnd[1] == '\'') {
            int len = strlen(opnd) - 3;
            *loc_change = len;
            for (int k = 2; k < strlen(opnd) - 1; k++) {
                append_byte(opnd[k]);
            }
        } else if (opnd[0] == 'X' && opnd[1] == '\'') {
            char hex[3] = {0};
            int len = (strlen(opnd) - 3) / 2;
            *loc_change = len;
            for (int k=2, j=0; k<strlen(opnd)-1; k+=2, j++) {
                hex[0] = opnd[k];
                hex[1] = opnd[k+1];
                append_byte(hex_to_int(hex));
            }
        }
    }
}
```

```c
int main() {
    FILE *in = fopen("../../ipop/input.sic", "r");
    FILE *listf = fopen("../../ipop/listfile-op", "w");
    FILE *objf = fopen("../../ipop/objcode-op", "w");
    if (!in || !listf || !objf) {
        printf("Error opening file\n");
        return 1;
    }
    char line[MAX_LINE];
    int program_len = 0;

    while (fgets(line, sizeof(line), in)) {
        char *comm = strchr(line, ';');
        if (comm) *comm = '\0';

        char label[20] = "", op[10] = "", opnd[20] = "";
        sscanf(line, "%s %s %s", label, op, opnd);

        if (strlen(label) > 0 && label[0] == '*') label[0] =
'\0';

        if (strlen(op) == 0) continue;
        int save_loc = locctr;
        int loc_change = 0;

        if (strcmp(op, "START") == 0) {
            strcpy(progname, label);
            start_addr = hex_to_int(opnd);
            locctr = start_addr;
            save_loc = 0;
        } else if (strcmp(op, "END") == 0) {
            break;
        } else {
            if (strlen(label) > 0) {
                insert_sym(label, locctr);
            }
            if (strcmp(op, "LDA") == 0 || strcmp(op, "ADD") == 0
|| strcmp(op, "STA") == 0) {
                handle_instruction(op, opnd);
                loc_change = 3;
            } else {
                handle_directive(op, opnd, &loc_change);
            }
        }
        locctr += loc_change;
        fprintf(listf, "%04X\t%-8s %-8s %s\n", save_loc, label,
op, opnd);
    }
    program_len = locctr - start_addr;
    fprintf(objf, "H %6s %06X %06X\n", progname, start_addr,
program_len);
    int i = 0;
    int current_addr = start_addr;
    while (i < objidx) {
        int chunk_size = (objidx - i > 30) ? 30 : objidx - i;
        fprintf(objf, "T %06X %02X", current_addr, chunk_size);
        for (int b = 0; b < chunk_size; b += 3) {
            if (b + 2 < chunk_size) {
                int byte1 = objbuf[i + b];
                int byte2 = objbuf[i + b + 1];
                int byte3 = objbuf[i + b + 2];
                fprintf(objf," %02X%02X%02X",byte1,byte2,byte3);
            }
        }
        fprintf(objf, "\n");
        i += chunk_size;
        current_addr += chunk_size;
    }
    fprintf(objf, "E %06X\n", start_addr);
    for (int k = 0; k < nsym; k++) {
        Ref *r = symtab[k].refs;
        while (r) {
            Ref *tmp = r;
            r = r->next;
            free(tmp);
        }
        if (!symtab[k].defined) {
            printf("Warning: Undefined symbol %s\n",
symtab[k].name);
        }
    }
    fclose(in);
    fclose(listf);
    fclose(objf);
    printf("Assembly complete. Object code in objcode-op,
listing in listfile-op\n");
    return 0;
}
```

```
Output
input.sic
    COPY    START   1000
    FIRST   LDA     ALPHA
    **      ADD     BETA
    **      STA     GAMMA
    ALPHA   WORD    5
    BETA    WORD    3
    GAMMA   RESW    1
    **      END     FIRST

Assembly complete. Object code in objcode-op, listing in
listfile-op

listfile-op
    0000    COPY     START    1000
    1000    FIRST    LDA      ALPHA
    1003             ADD      BETA
    1006             STA      GAMMA
    1009    ALPHA    WORD     5
    100C    BETA     WORD     3
    100F    GAMMA    RESW     1

objcode-op
    H   COPY 001000 000012
    T 001000 0F 001009 18100C 0C100F 000005 000003
    E 001000
```