8086 Trainer Kit

1a. *Addition*

```
MOV CX,0000
MOV AX,[0500]
MOV BX,[0600]
ADD AX,BX
JNC 200F
INC CX
MOV [0700],AX
MOV [0800],CX
HLT

STARTING ADDR: 1000
I/P
0500: 82
0501: 12
0600: 84
0601: 12

O/P
0700: 06
0701: 25
```

1b. *Substraction*

```
MOV CX,0000
MOV AX,[0500]
MOV BX,[0600]
SUB AX,BX
JNC 200F
INC CX
MOV [0700],AX
MOV [0800],CX
HLT

STARTING ADDR: 1000
I/P
0500: 2D
0501: FE
0600: AD
0601: BC
```

```
O/P
0700: 80
0701: 41
0800: 00
```

1c. *Multiplication*

```
MOV AX,[0300]
MOV BX,[3002]
MUL BX
MOV [3004],AX
MOV AX,BX
MOV [3006],AX
HLT

STARTING ADDR:1000
I/P
3000: 03
3001: 04
3002: 08
3003: 07
3004: 18

O/P
3005: 75
3006: 1C
3007: 00
```

1d. *Division*

```
MOV SI,0500
MOV DI,0600
MOV BL,[SI]
INC SI
MOV AX,[SI]
DIV BL
MOV [DI],AX
HLT
STARTING ADDR: 1000
I/P
0500: 06
0501: 20
0502: 00
```

```
O/P
0600: 05
0601: 02
```

2. *Greatest of 2 Numbers*

```
MOV SI,0500
MOV AL,[SI]
INC SI
MOV BL,[SI]
CMP AL,BL
JGE 040E
MOV AL,BL
MOV [0502],AL
HLT

STARTING ADDR: 0400
I/P
0500: 25
0501: 72

O/P
0502: 72
```

2 Pass Assembler

Pass 1

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX_SYM 50
#define BUF 20

typedef struct symtab {
    char lab[20];
    int addr;
} stab;

typedef struct optab {
    char opnd[20];
    int addr;
} optb;
```

```c
int st_cnt = 0, otsize = 0, start = 0, locctr = 0;
stab st[MAX_SYM];
optb otab[MAX_SYM];

void pass1 (FILE* inter, char opcode[], char label[], char op[])
{
    if (strcmp(opcode, "START") == 0) {
        start = locctr = (int) strtol(op, NULL, 16);
        fprintf(inter, "%X\t%s\t%s\t%s\n", start, label, opcode,
op);
        return;
    }
    int inc;
    if (strcmp(label, "**")) {
        int found = 0;
        for (int i = 0; i < st_cnt; i++) {
            if (!strcmp(st[i].lab, label)) {
                found = 1;
            }
        }
        if (found) {
            printf("Symbol %s Already Declared!!!\n", label);
            exit(1);
        } else {
            strcpy(st[st_cnt].lab, label);
            st[st_cnt++].addr = locctr;
        }
    }
    if (!strcmp(opcode, "BYTE"))
        inc = strlen(op);
    else if (!strcmp(opcode, "WORD"))
        inc = 3;
    else if (!strcmp(opcode, "RESB"))
        inc = (int) strtol(op, NULL, 10);
    else if(!strcmp(opcode, "RESW"))
        inc = 3 * (int) strtol(op, NULL, 10);
    else {
        int found = 0;
        for (int i = 0; i < otsize; i++) {
            if (!strcmp(opcode, otab[i].opnd)) {
                found = 1;
                break;
            }
        }
        if (found)
            inc = 3;
        else if (!strcmp(opcode, "END"))
            inc = 0;
        else {
            printf("OPCODE %s NOT FOUND!!!\n", opcode);
            exit(1);
        }
    }
    fprintf(inter, "%X\t%s\t%s\t%s\n", locctr, label, opcode,
op);
    locctr += inc;
}
int main() {
    FILE* src = fopen("../../ipop/input.sic", "r");
    FILE* optab = fopen("../../ipop/optab", "r");
    FILE* inter = fopen("../../ipop/intermediate.sic", "w");
    FILE* symtab = fopen("../../ipop/symtab", "w");

    if (!src || !optab || !inter || !symtab) {
        printf("Couldnot Open File...\n");
        return 0;
    }
    char label[BUF], opcode[BUF], op[BUF];
    int opint;
    while (fscanf(optab, "%s\t%d", opcode, &opint) > 0) {
        strcpy(otab[otsize].opnd, opcode);
        otab[otsize++].addr = opint;
    }               // to fetch optab into memory...

    do {
        fscanf(src, "%s\t%s\t%s", &label, &opcode, &op);
        pass1(inter, opcode, label, op);
    } while (strcmp(opcode, "END"));
    for (int i = 0; i < st_cnt; i++) {
        fprintf(symtab, "%s\t%X\n", st[i].lab, st[i].addr);
    }
    printf("Intermediate file saved as \"intermediate.sic\" in
ipop\n");
    printf("SYMTAB saved as \"symtab\" in ipop\n");
    fclose(src); fclose(optab); fclose(symtab); fclose(inter);
    return 0;
}
```

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

Output
Intermediate file saved as "intermediate.sic" in ipop
SYMTAB saved as "symtab" in ipop
    1. intermediate.sic
        1000    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
        1012    **  END FIRST
    2. symtab
        FIRST   1000
        ALPHA   1009
        BETA    100C
        GAMMA   100F
```

Pass 2

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define BUF 20
#define MAX 100
#define REC_BUFF 70
#define REC_BYTES 30

typedef struct tab {
    char symbol[BUF];
    int address;
} tab;
typedef struct inter {
    int addr;
    char label[BUF];
    char opcode[BUF];
    char opnd[BUF];
} inter;
typedef struct record {
    int start_addr;
    int len;
    char rec[REC_BUFF];
} rec;
int otsize = 0, stsize = 0, lcnt = 0;
tab optab[BUF];
tab symtab[BUF];
inter itr[MAX];
rec text;
void flush_text(FILE *out) {
    if (text.len > 0) {
        fprintf(out, "T %06X %02X %s\n", text.start_addr,
text.len, text.rec);
        text.len = 0;
        text.rec[0] = '\0';
        text.start_addr = -1;
    }
}

void start_text_if_needed(int addr) {
    if (text.len == 0) {
        text.start_addr = addr;
        text.rec[0] = '\0';
    }
}

void text_write_bytes(const char *hex_no_spaces, int nbytes, int
locctr, FILE *out) {
    if (text.len + nbytes > REC_BYTES) {
        flush_text(out);
    }
    start_text_if_needed(locctr);
    if (text.rec[0] != '\0') strcat(text.rec, " ");
    strcat(text.rec, hex_no_spaces);
    text.len += nbytes;
}
```

```
void int_to_hex(int value, int width_bytes, char *outhex) {
    char fmt[8];
    sprintf(fmt, "%%0%dX", width_bytes * 2);
    sprintf(outhex, fmt, value & ((1 << (width_bytes * 8)) -
1));
}
int parse_BYTE(const char *opnd, char *hex_out) {
    int nbytes = 0;
    if (opnd[0] == 'C' && opnd[1] == '\'' &&
opnd[strlen(opnd)-1] == '\'') {
        const char *p = opnd + 2;
        size_t len = strlen(opnd);
        size_t end = len - 1;
        hex_out[0] = '\0';
        for (; p < opnd + end; ++p) {
            char tmp[3*2];
            sprintf(tmp, "%02X", (unsigned char)*p);
            strcat(hex_out, tmp);
            nbytes++;
        }
    } else if (opnd[0] == 'X' && opnd[1] == '\'' &&
opnd[strlen(opnd)-1] == '\'') {
        size_t len = strlen(opnd);
        size_t datalen = len - 3;
        const char *p = opnd + 2;
        char buf[REC_BUFF];
        strncpy(buf, p, datalen);
        buf[datalen] = '\0';
        buf[datalen - 1] = '\0';
        for (char *q = buf; *q; ++q) {
            if (!isxdigit((unsigned char)*q)) return -1;
            *q = (char)toupper((unsigned char)*q);
        }
        if (strlen(buf) % 2 != 0) return -1;
        strcpy(hex_out, buf);
        nbytes = (int)(strlen(buf) / 2);
    } else {
        int val = atoi(opnd);
        if (val < 0 || val > 255) return -1;
        sprintf(hex_out, "%02X", val & 0xFF);
        nbytes = 1;
    }
    return nbytes;
}


void pass2(FILE* out, FILE* list) {
    text.len = 0;
    text.rec[0] = '\0';
    text.start_addr = -1;

    int start_addr = itr[0].addr;
    const char *progname = itr[0].label[0] ? itr[0].label :
"NONAME";
    int proglen = (lcnt > 0) ? (itr[lcnt-1].addr - start_addr) :
0;

    fprintf(out,  "H %06s %06X %06X\n", progname, start_addr,
proglen);
    fprintf(list, "%X\t%s\t%s\t%s\t**\n", itr[0].addr,
itr[0].label, itr[0].opcode, itr[0].opnd);

    for (int i = 1; i < lcnt; i++) {
        char objhex[REC_BUFF]; objhex[0] = '\0';
        int objbytes = 0;

        int opi = -1;
        for (int j = 0; j < otsize; j++) {
            if (!strcmp(optab[j].symbol, itr[i].opcode)) { opi =
j; break; }
        }

        if (opi != -1) {
            char opc[3]; int_to_hex(optab[opi].address, 1, opc);
            int addr_val = 0;
            if (strcmp(itr[i].opnd, "**")) {
                int stf = -1;
                for (int j = 0; j < stsize; j++) {
                    if (!strcmp(symtab[j].symbol, itr[i].opnd))
{ stf = j; break; }
                }
                if (stf == -1) {
                    fprintf(stderr, "Undefined symbol: %s at
%X\n", itr[i].opnd, itr[i].addr);
                    exit(1);
                }
```

```c
                addr_val = symtab[stf].address;
            }
            char addrhex[5]; int_to_hex(addr_val, 2, addrhex);
            sprintf(objhex, "%s%s", opc, addrhex);
            objbytes = 3;
            text_write_bytes(objhex, objbytes, itr[i].addr,
out);
        }
        else if (!strcmp(itr[i].opcode, "BYTE")) {
            int nb = parse_BYTE(itr[i].opnd, objhex);
            if (nb < 0) {
                fprintf(stderr, "Invalid BYTE operand: %s at
%X\n", itr[i].opnd, itr[i].addr);
                exit(1);
            }
            objbytes = nb;
            text_write_bytes(objhex, objbytes, itr[i].addr,
out);
        }
        else if (!strcmp(itr[i].opcode, "WORD")) {
            int val = atoi(itr[i].opnd);
            int_to_hex(val, 3, objhex);
            objbytes = 3;
            text_write_bytes(objhex, objbytes, itr[i].addr,
out);
        }
        else if (!strcmp(itr[i].opcode, "RESB") || !
strcmp(itr[i].opcode, "RESW")) {
            flush_text(out);
        } else if (!strcmp(itr[i].opcode, "END")) {
        } else {
            flush_text(out);
        }

        fprintf(list, "%X\t%s\t%s\t%s\t%s\n",
                itr[i].addr, itr[i].label, itr[i].opcode,
itr[i].opnd,
                (objhex[0] ? objhex : "**"));
    }

    flush_text(out);
    fprintf(out, "E %06X\n", start_addr);
}
```

```c
int main() {
    FILE *inter = fopen("../../ipop/intermediate.sic", "r");
    FILE *otab  = fopen("../../ipop/optab", "r");
    FILE *stab  = fopen("../../ipop/symtab", "r");
    FILE *out   = fopen("../../ipop/objcode", "w");
    FILE *list  = fopen("../../ipop/listfile.sic", "w");
    if (!inter || !otab || !stab || !out || !list) {
        printf("Could not open files...\n");
        return 1;
    }
    // read optab
    while (fscanf(otab, "%s %X", optab[otsize].symbol,
&optab[otsize].address) == 2) otsize++;
    // read symtab
    while (fscanf(stab, "%s %X", symtab[stsize].symbol,
&symtab[stsize].address) == 2) stsize++;
    // read intermediate
    while (fscanf(inter, "%X %s %s %s", &itr[lcnt].addr,
itr[lcnt].label, itr[lcnt].opcode, itr[lcnt].opnd) == 4) lcnt++;
    pass2(out, list);
    printf("Object Code Saved in \"objcode\" in ipop\n");
    printf("Listing file Saved in \"listfile.sic\" in ipop\n");

    fclose(inter); fclose(otab); fclose(stab); fclose(out);
fclose(list);
    return 0;
}
```

Output

```
Object Code Saved in "objcode" in ipop
Listing file Saved in "listfile.sic" in ipop
```

1. objcode

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

2. listfile.sic

```
1000    COPY    START   1000    **
1000    FIRST   LDA ALPHA   001009
1003    **  ADD BETA    18100C
1006    **  STA GAMMA   0C100F
1009    ALPHA   WORD    5   000005
100C    BETA    WORD    3   000003
100F    GAMMA   RESW    1   **
1012    **  END FIRST   **
```

Loader

Absolute Loader

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define BUF 1000
int main() {
    FILE* obj = fopen("../ipop/objcode", "r");
    if (!obj) {
        printf("Couldnot open file...");
        return 1;
    }
    int locctr = 0;
    char ch;
    while ((ch = fgetc(obj)) != EOF) {
        if (ch == 'H') {
            while ((ch = fgetc(obj)) == ' ');
            char pname[6];
            int start, length;
            fscanf(obj, "%s %X %X\n", pname, &start, &length);
            locctr = start;
            printf("Program Name: %c%s\n", ch, pname);
            printf("Starting Address: %X\n", start);
            printf("Program Length: %X\n", length);
        }
        else if (ch == 'T') {
            while ((ch = fgetc(obj)) == ' ');
            int start, length;
            fscanf(obj, "%X %X ", &start, &length);
            for (int i = 0; i < length; i++) {
                ch = fgetc(obj);
                if (ch == ' ') {
                    i--;
                    continue;
                }
                char ch1 = fgetc(obj);
                printf("%X: %c%c\n", locctr++, ch, ch1);
            }
        }
    }
    return 0;
}
```

```
I/P
objcode
    H   COPY 001000 000012
    T 001000 0F 001009 18100C 0C100F 000005 000003
    E 001000

O/P
Program Name: COPY
Starting Address: 1000
Program Length: 12
1000: 00
1001: 10
1002: 09
1003: 18
1004: 10
1005: 0C
1006: 0C
1007: 10
1008: 0F
1009: 00
100A: 00
100B: 05
100C: 00
100D: 00
100E: 03
```

Relocation Loader

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

int main() {
    FILE* obj = fopen("../ipop/objcode-rel", "r");
    if (!obj) {
        printf("Couldnot Open File...\n");
        return 1;
    }
    int locctr = 0, start_addr = 0, new_start, addr_diff = 0;
    char ch;
    printf("Enter New Starting Address: ");
    scanf("%X", &new_start);
    locctr = new_start;
    while ((ch = fgetc(obj)) != EOF) {
        if (ch == 'H') {
            while ((ch = fgetc(obj)) == ' ');
            char pname[6];
            int length;
            fscanf(obj, "%s %X %X\n", pname, &start_addr,
&length);
            printf("Program Name: %c%s\n", ch, pname);
            printf("Starting Address: %X\n", start_addr);
            printf("Program Length: %X\n", length);
            addr_diff = new_start - start_addr;
        }
        else if (ch == 'T') {
            while ((ch = fgetc(obj)) == ' ');
            int start, length, rel;
            fscanf(obj, "%X %X %X ", &start, &length, &rel);

            char rel_bits[12];
            int code;
            char ocode[6];
            sprintf(rel_bits, "%012b", rel);
            for (int i = 0; i < length/3; i++) {
                fscanf(obj, "%X ", &code);
                if (rel_bits[i] == '1')
                    code += addr_diff;

                sprintf(ocode, "%06X", code);
                for (int i = 0; i < 6; i += 2)
                    printf("%X: %c%c\n", locctr++, ocode[i],
ocode[i+1]);
            }
        }
    }
    return 0;
}
```

```
I/P
    objcode-rel
        H   COPY 001000 000012
        T 001000 0F E00 001009 18100C 0C100F 000005 000003
        E 001000
```

```
O/P
Enter New Starting Address: 3000
Program Name: COPY
Starting Address: 1000
Program Length: 12
3000: 00
3001: 30
3002: 09
3003: 18
3004: 30
3005: 0C
3006: 0C
3007: 30
3008: 0F
3009: 00
300A: 00
300B: 05
300C: 00
300D: 00
300E: 03
```