SIC/XE Disassembler

201820873

Test.obj

```
HCOPY  000000001077
T0000001D17202D69202D4B10103603202629000033200 74B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T0010361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T001070073B2FEF4F000005
M00000705
M00001405
M00002705
E000000
```

Object code 를 .obj 형식으로 저장하였다.

Objcode.h

```c
#define MAXLINE 128
#define MAXLENGTH 128

typedef struct
{
    int obj_line_length;
    char objt_buf[MAXLENGTH][MAXLINE];
} OBJ;

OBJ get_obj()
{
    OBJ obj;
    FILE *objcode = NULL;
    int i = 0;
    int obj_line_length = 0;

    objcode = fopen("./test.obj", "r");
    while (feof(objcode) == 0)
    {
        fgets(obj.obj_buf[i], MAXLENGTH, objcode);
        i++;
        obj.obj_line_length = i;
    }

    return obj;
}
```

Test.obj 파일을 열어서 각 줄을 OBJ 구조체에 저장하였다.

Inst_reg.h

```c
typedef struct
{
    char Mnemonic[6];
    int format;
    unsigned short int opcode;
} instruction_set;
instruction_set instruction[] = {
    {"ADD", 3, 0x18},
    {"ADDF", 3, 0x58},
    {"ADDR", 2, 0x90},
    {"AND", 3, 0x40},
    {"CLEAR", 2, 0xB4},
```

```c
typedef struct
{
    char Mnemonic_R[2];
    int number;

} Register;

Register R[] = {
    {"A", 0},
    {"X", 1},
    {"L", 2},
    {"B", 3},
    {"S", 4},
    {"T", 5},
    {"F", 6},
    {"PC", 8},
    {"SW", 9}};
```

Inst_reg.h 에는 opcode 의 이름과, 포맷과, hex 값을 가지고 있는 구조체와 레지스터의 이름과 숫자를 가지고 있는 구조체를 구성하였다.

Objcode_hex.h

```c
#define MAXLINE 128
#define MAXLENGTH 128



typedef struct
{
    char type;
    int size;
    unsigned start_addr_line_length[10];
    unsigned code_hex[MAXLENGTH];
}obj_code_line;



typedef struct
{
    char program_name[10];
    char program_starting_addr[10];
    char program_length[10];
    char E_starting_addr[10];
    int program_line;
    obj_code_line OBJ_LINE[MAXLINE];

}OBJ_HEX;

unsigned int ascii_to_hex(const char* str, int size, unsigned* hex)
{
    unsigned int i, h, high, low;
    for (h = 0, i = 0; i < size; i += 2, ++h) {
        high = (str[i] > '9') ? str[i] - 'A' + 10 : str[i] - '0';
        low = (str[i + 1] > '9') ? str[i + 1] - 'A' + 10 : str[i + 1] - '0';
        hex[h] = (high << 4) | low;
    }
    return h;
}
```

Objcode_hex.h 에는 전체 프로그램에 대한 구조체와 각 라인에 대한 구조체가 선언되어 있고,
ascii_to_hex 함수는 문자열로 이루어진 hex 를 실제 hex 값으로 바꾸어 주는 함수이다.
예를 들어 “12CD”의 문자열을 받으면 반환값은 0x12CD 이다.
OBJ_HEX 에는 첫번째 줄에서 얻을 수 있는 프로그램의 이름, 시작주소, 길이를 알 수 있고,
마지막 줄에서 얻을 수 있는 시작주소, 전체 프로그램의 라인 수, 각 줄에 대한 구조체를 포함하고 있다.

Obj_code_line 구조체에는 첫번째 문자로 알 수 있는 각 줄의 역할인 type, 줄의 길이, start_addr_line_length 는 [0]~[2]는 시작주소, [3]은 라인 길이를 저장한다. 그 후 나머지 opcode 와 operand 는 code_hex 에 저장된다.

Print_objcode

```
void print_objcode(OBJ_HEX object_code_hex){
    printf("H %s ", object_code_hex.program_name);
    printf("%s ", object_code_hex.program_starting_addr);
    printf("%s\n", object_code_hex.program_length);
    for(int i=0; i<object_code_hex.program_line; i++){
        printf("%c ", object_code_hex.OBJ_LINE[i].type);
        int size = object_code_hex.OBJ_LINE[i].size;
        for(int k=0; k<4; k++){
            if(k==3)printf(" ");
            printf("%02X", object_code_hex.OBJ_LINE[i].start_addr_line_length[k]);
        }printf(" ");

        for(int k=0; k<size/2; k++){
            printf("%02X", object_code_hex.OBJ_LINE[i].code_hex[k]);

        }printf("\n");
    }
    printf("========================================\n");
}
```

Print_objcode 함수는 OBJ_HEX 구조체에 저장되어 있는 모든 object code 를 출력하기 위한 함수이다.

Main.c (objcode -> hex)

```c
int main(void)
{
    OBJ objcode_string;
    OBJ_HEX object_code_hex;
    ASMC assembly_code;
    objcode_string = get_obj();

    if(objcode_string.obj_buf[0][0] == 'H'){
        strncpy(object_code_hex.program_name, *(objcode_string.obj_buf) + 1, 6);
        strncpy(object_code_hex.program_starting_addr, *(objcode_string.obj_buf) + 7, 6);
        strncpy(object_code_hex.program_length, *(objcode_string.obj_buf) + 13, 6);
    }
    for (int j = 1; j < objcode_string.obj_line_length; j++)
    {
        char code_line_str[128];
        int size;
        char type = objcode_string.obj_buf[j][0];
        object_code_hex.OBJ_LINE[j-1].type = type;
        if(objcode_string.obj_buf[j][0] == 'E'){
            object_code_hex.program_line = j-1;
            strncpy(object_code_hex.E_starting_addr, *(objcode_string.obj_buf) + j*MAXLENGTH+1, 6);
        }else{
            strncpy(code_line_str,objcode_string.obj_buf[j]+1, 8);
            ascii_to_hex(code_line_str, strlen(code_line_str), object_code_hex.OBJ_LINE[j-1].start_addr_line_length);

            if(type == 'T'){
                strncpy(code_line_str,objcode_string.obj_buf[j] + 9, sizeof(objcode_string.obj_buf[j]));
                size = strlen(code_line_str);
                object_code_hex.OBJ_LINE[j-1].size = size;
                ascii_to_hex(code_line_str, strlen(code_line_str), object_code_hex.OBJ_LINE[j-1].code_hex);
            }
        }

    }
```

Get_obj 함수로부터 objcode 에 대한 모든 정보를 불러온 후 각 줄을 분석한다.
맨 처음 문자열을 통해 라인의 특성을 파악한 후 H, T, M, E 에 나누어 분리하여 동작하게 하였다.
문자열 복사를 위해 strncpy 함수를 사용하였으며 ascii_to_hex 함수를 통해 문자열을 hex 값으로 바꾼 후 구조체에 저장하였다.

Main.c (object code -> assembly code(address, opcode)

```c
for(int i=0; i<object_code_hex.program_line; i++){
    if(object_code_hex.OBJ_LINE[i].type == 'T'){
        unsigned int a, b, c;
        a = object_code_hex.OBJ_LINE[i].start_addr_line_length[0] << 16;
        b = object_code_hex.OBJ_LINE[i].start_addr_line_length[1] << 8;
        c = object_code_hex.OBJ_LINE[i].start_addr_line_length[2];
        for(int j=0; j<object_code_hex.OBJ_LINE[i].start_addr_line_length[3]; j++){

            for(int k = 0; k<INST_SIZE; k++){
                if(object_code_hex.OBJ_LINE[i].code_hex[j] >>2 == instruction[k].opcode>>2){

                    if(instruction[k].format == 1){
                        printf("(%04X %s\n", j + a+b+c, instruction[k].Mnemonic);
                        break;
                    }
                    else  if(instruction[k].format == 2){

                        printf("%04X %s\n",j+a+b+c,  instruction[k].Mnemonic);
                        j +=1;
                        break;
                    }
                    else if(instruction[k].format == 3){

                        printf("%04X ", j+a+b+c);
                        if(object_code_hex.OBJ_LINE[i].code_hex[j+1]>>4 % 0x10 == 1){
                            j+=1;
                            printf("+");
                        }
                        printf("%s\n",instruction[k].Mnemonic);
                        j +=2;
                        break;
                    }
                }
            }
    }printf("\n");
    }
}
```

Object code 를 통해 어셈블리 코드로 변환하는 것을 구현하였다. Opcode 를 instruction 구조체를 반복문을 통해 매칭시킨 후 문자열로 변환하여 출력하였다. 그 후 포맷에 따라 반복문의 인자를 변화시켜서 operand 가 잘 들어가도록 하였다. 그 후 주소값도 일치시켰다.

결과

```
25%  6%  4.2 GI
/mnt/c/U/h/Doc/Micro_computer_Architecture ./MCD
H COPY   000000 001077
T 000000 1D 17202D69202D4B1010360320262900003320074B10105D3F2FEC032010FFFFFFDA
T 00001D 13 0F20160100030F200D4B10105D3E2003454F46FFFFFFDA
T 001036 1D B410B400B44075101000E32019332FFADB2013A00433200857C003B850FFFFFFDA
T 001053 1D 3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850FFFFFFDA
T 001070 07 3B2FEF4F000005FFFFFFDA
M 000007 05
M 000014 05
M 000027 05
========================================
0000 STL
0003 LDB
0006 +JSUB
000A LDA
000D COMP
0010 JEQ
0013 +JSUB
0017 J
001A LDA

001D STA
0020 LDA
0023 STA
0026 +JSUB
002A J
002D OR

1036 CLEAR
1038 CLEAR
103A CLEAR
103C +LDT
1040 TD
1043 JEQ
1046 RD
1049 COMPR
104B JEQ
104E STCH
1051 TIXR
1053 JLT
1056 STX
1059 RSUB
105C SIO
105D CLEAR
105F LDT
1062 TD
1065 JEQ
1068 LDCH
106B WD
106E TIXR

1070 JLT
1073 RSUB
1076 LDX
```