**과목명: 시스템프로그래밍**

**1분반**

**<<Project #2>>**

**서강대학교 [국제인문학부]**

**[20150038]**

**[김재원]**

목 차

1. **프로그램 개요**

지난 프로젝트 1에서 구현한 Shell에 Assemble 기능을 추가하는 프로젝트 입니다. SIC/XE의 Assembly Program Source 파일을 입력 받아서 object 파일을 생성하고, 어셈블리 과정 중 생성된 symbol table과 결과물인 object 파일을 볼 수 있는 기능을 제공해야 합니다. 위와 같은 기능을 제공하는 프로그램을 작성하기 위해 필요한 자료구조와 알고리즘을 구상하여 전체적인 프로그램을 설계합니다.

1. **프로그램 설명**
   1. 프로그램 흐름도 (\*별첨1 첨부)
2. **모듈 정의**
   1. 정의한 모듈에 대해 간략적인 설명

===================================

**Assemble.C**

|  |  |
| --- | --- |
| **Void assemble\_init(void)** | |
| **Input** | **X** |
| **Func** | Assemble 명령어와 관련해서SICSIM이 실행될 때 초기화되어야 할 일을 수행하고 명령어를 받습니다. 받은 명령어를 적절하게 처리해서 해당 명령어를 수행하는 함수에 넘겨줍니다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void type(char \*)** | |
| **Input** | **Char \*** |
| **Func** | 파일명을 입력받아서 그 파일의 내용을 모두 출력합니다 |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void assemble** | |
| **Input** | **Char \*** |
| **Func** | 매개변수로 들어온 파일이름에 따라 파일을 연다. (.asm 파일일 것) 그 파일을 한 줄씩 읽으면서 Symbol table을 만든다. 이 symbol table을 참고하면서 결과적으로 .lst 파일과 .obj 파일을 만든다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void symbol** | |
| **Input** | **X** |
| **Func** | Symbol() 명령어가 들어오면 작동하는 함수다. Symbol을 alphabetical order로 정렬해서 출력한다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void parse\_line** | |
| **Input** | **Char [3][LINE], char [LINE]\*** |
| **Func** | 입력된 string 한 줄을 Chunk 수에 맞게 쪼갠다. Symbol, operation, operands로 나눈다. |
| **Output** |  |

|  |  |
| --- | --- |
| **Int push\_symbol** | |
| **Input** | **Int, char [60]** |
| **Func** | 매개변수로 들어온 정보를 활용한다. Symbol table에 이미 같은 이름의 symbol이 있으면 에러를 출력하고 assemble을 종료한다. 만약 중복된 symbol이 없다면 address와 함께 symbol table에 저장한다. |
| **Output** | **Error : -1**  **정상 : 1** |

|  |  |
| --- | --- |
| **Void address\_increase** | |
| **Input** | **Int \*, char[LINE], char[LINE]** |
| **Func** | 입력된 operation 과 operand에 따라 주소값을 증가시킨다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Int obj\_make** | |
| **Input** | **Int, char[LINE], char[LINE], char \*, int \*** |
| **Func** | OPCODE, nibpxe, displacement(or address) 총 세 부분으로 나누어서 OBJ Code를 생성한다. |
| **Output** | **objcode** |

|  |  |
| --- | --- |
| **Int symbol\_find** | |
| **Input** | **Char [LINE]** |
| **Func** | 매개변수로 들어온 symbol이 table에 존재하는지 찾는다. 없으면 -1을 return 한다. |
| **Output** | **Symbol의 address 값 혹은 -1** |

|  |  |
| --- | --- |
| **Void dex\_to\_bit** | |
| **Input** | **Char \*, char** |
| **Func** | 입력된 16진수를 4bit binary digit으로 변환한다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Int stoi** | |
| **Input** | **Char \*** |
| **Func** | 입력된 string을 정수값으로 바꾼다. |
| **Output** | **변환된 정수값** |

|  |  |
| --- | --- |
| **Char \* binary\_to\_dex** | |
| **Input** | **Char \*** |
| **Func** | Binary digit을 16진수로 변환한다. |
| **Output** | **변환된 16진수** |

|  |  |
| --- | --- |
| **Char int\_to\_dex** | |
| **Input** | **Int** |
| **Func** | 10진수 정수를 16진수로 변환한다. Return 값은 그 수의 char이다. |
| **Output** | **변환된 character** |

|  |  |
| --- | --- |
| **Void print\_assemble** | |
| **Input** | **FILE \*, int, int, char \*, char \* char \*, char \*** |
| **Func** | .lst 파일 형식에 맞춰서 location, address, objcode 등을 출력한다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void char\_to\_dex** | |
| **Input** | **Int, char \*, int \*** |
| **Func** | 입력된 값을 16진수 두 자리로 변환한다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Char register\_to\_num** | |
| **Input** | **Char** |
| **Func** | 입력된 register에 해당하는 값을 character로 변환해서 return 합니다. |
| **Output** | **Register의 번호** |

|  |  |
| --- | --- |
| **Int instruction\_check** | |
| **Input** | **Char \*** |
| **Func** | 입력된 instruction이 유효한지 check합니다. 유효하지 않다면 -1, 유효하다면 1을 return 합니다. |
| **Output** | **유효하다면 1, 유효하지 않다면 -1** |

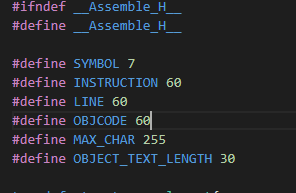
|  |  |
| --- | --- |
| **Void modify\_relocation** | |
| **Input** | **Char \*, int** |
| **Func** | Obj 파일에서 M으로 시작하는 코드를 만든다. (modify, relocation) |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void formatting** | |
| **Input** | **Char \*, int, int** |
| **Func** | 주어진 format과 진법에 맞춰서 입력된 integer를 16진수 문자열로 바꾼다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void format4\_plus\_delete** | |
| **Input** | **Char \*, char \*** |
| **Func** | 함수명처럼 format4에서 앞에 달려있는 +를 제거한다. |
| **Output** | **X** |

**4. 전역 변수 정의**

**Assemble**



**5. 코드 설명**

**Assemble.h**

#ifndef \_\_Assemble\_H\_\_

#define \_\_Assemble\_H\_\_

#define SYMBOL 7

#define INSTRUCTION 60

#define LINE 60

#define OBJCODE 60

#define MAX\_CHAR 255

#define OBJECT\_TEXT\_LENGTH 30

typedef struct sym\_element{

    char \* symbol ;

    int addr;

} element;

typedef struct \_element{

    char \* symbol;

    char \* op;

} op\_element;

element sym\_table[200];

op\_element opcode[200];

//int sym\_index = 0;

/\* filename을 받아서 그 파일을 연다.

만약 열리지 않으면 에러를 출력하고 종료한다.

열린다면 그 파일을 읽어서 Stdout 한다 \*/

// HOMEWORK //

void type(char \* filename);

void assemble(char \* filename);

void symbol();

////////////////////////

int push\_symbol(int addr, char [LINE]);

//check this instruction is valuable

int instruction\_check(char \*);

//Init, Setting objcode.

void assemble\_init();

// according to instruction, increase address.

void address\_increase(int \* addr, char [LINE], char[LINE]);

// print assemble code into FILE pointer

void print\_assemble(FILE \*fp, int addr, int location, char \*symbol, char \*operation, char \*operand, char \*objcode);

// make objcode

int obj\_make(int, char[LINE], char[LINE], char \*, int format);

// parse instruction line by three chunk

void parse\_line(char[3][LINE], char[LINE]);

void formatting(char \*, int digit, int);

void modify\_relocation(char \*, int);

void format4\_plus\_delete(char \*dest, char \*src);

void dex\_to\_bit(char \*, char);

void char\_to\_dex(int , char \*, int \*);

char register\_to\_num(char);

char int\_to\_dex(int);

char \*binary\_to\_dex(char \*);

int stoi(char \*);

//find the symbol and return it's address

int symbol\_find(char[LINE]);

#endif

**Assemble.C**

Assemble.C#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

#include "Assemble.h"

int sym\_index, last\_address, op\_index ;

int base;

/\* initializing을 해줍니다.

asm 파일이 달라지면 sym\_table 등은 다시 초기화시켜줘야합니다

다른 전역변수 값도 초기화되어야 합니다 \*/

void assemble\_init()  {

    int i, k;

sym\_index = 0;

last\_address = 0;

op\_index = 0;

base = 0;

    for (i = 0; i < 200; i++)  {

        sym\_table[i].symbol = (char \*)malloc(sizeof(char) \* SYMBOL);

        sym\_table[i].addr = 0;

        opcode[i].op = (char \*)malloc(sizeof(char) \* SYMBOL);

        opcode[i].symbol = (char \*)malloc(sizeof(char) \* SYMBOL);

    }

    FILE \*fps = fopen("opcode.txt", "r");

    if(fps == NULL){

        printf("ERROR ASSEMBLE INIT\n");

        return;

    }

    char opc[255];

    k = 0;

    while ( !feof(fps) )  {

        fscanf(fps, "%s %s %s\n", opcode[k].op, opcode[k].symbol, opc);

        k++;

    }

    op\_index = k;

    fclose(fps);

}

/\*매개변수로 들어온 파일을 열어서 그 내용을 출력합니다. \*/

void type(char \* filename){

    FILE \* fp = fopen(filename, "r");

    if(fp == NULL){

        printf("TYPE ERROR\n");

        return;

    }

    char c;

    while((int)(c = fgetc(fp)) != EOF){

        printf("%c", c);

    }

    fclose(fp);

    return;

}

/\* 입력된 filename을 assemble 한다. \*/

void assemble(char \* filename){

    int address, line\_number, i;

    char list\_filename[255], obj\_filename[255];

    // 파일을 연다. 문제가 있으면 에러 출력

    FILE \*fp = fopen(filename, "r");

    if (fp == NULL)  {

        printf("assemble error\n");

        return;

    }

    for (i = 0; filename[i] != '.'; i++){

        list\_filename[i] = filename[i];

        obj\_filename[i] = filename[i];

    }

    list\_filename[i] = '\0';

    obj\_filename[i] = '\0';

    strcat(list\_filename, ".lst");

    strcat(obj\_filename, ".obj");

    // 파일을 쭈우우욱 읽어들인다. 한 줄씩 읽을 건데 넉넉하게 60개 잡아놓자

    /\* 파일이 끝날 때 까지 읽어들여서 .lst 파일과 .obj 파일을 만든다. \*/

    FILE \*f\_lst = fopen((const char \*)list\_filename, "w+");

    FILE \*f\_obj = fopen((const char \*)obj\_filename, "w+");

    if (f\_lst == NULL || f\_obj == NULL)  {

        fclose(f\_lst);

        fclose(f\_obj);

        remove(list\_filename);

        remove(obj\_filename);

        printf("ERROR LST OR OBJ FILE\n");

        return;

    }

    /\* PASS 1 : 한 줄씩 읽으면서 각 Symbol의 Location Counter를 만든다.

    => Symbol Table을 만든다.

    Format 1:  ...  - few used

    Format 2: 2byte - register

    Format 3: 3byte - standard

    Format 4: 4byte

    \*/

    address = 0;

    line\_number = 0;

    while (1) {

        // Local Variable로 선언함으로써 하나하나 grabage 생각하지 않아도됨

        char src\_line[LINE], parsed\_line[3][LINE];

        line\_number += 5;

        if (fgets(src\_line, LINE, fp) == NULL)  {

            // NULL 인 경우 왠만하면 파일 끝에 다다랐다는 것이므로 while 문을 종료한다.

            break;

        }

        // 소스 파일에서 한 줄 읽었으면 명령어로 바꿔줘야한다.

        if (strstr(src\_line, "."))

            continue;

        parse\_line(parsed\_line, src\_line);  // |ㅁ|ㅁ|ㅁ| OR |ㅁ|ㅁ| |

        if (parsed\_line[0][0] == 'N')  {   // No Symbol => No need to into

            if (parsed\_line[1][0] == '+')  {// format4

            /\* Format 4는 좀 특이하게 생겼으므로 특이 취급을 해준다. \*/

                address += 4;

                char for\_check[10];

                format4\_plus\_delete(for\_check, parsed\_line[1]);

                if (instruction\_check(for\_check) == -1)  {

                    // 에러가 터지면 어디서 에러가 났는지 보고하고

                    // lst 파일과 obj 파일을 닫고, 이미 만든 건 삭제한다.

                    printf("%s\n", for\_check);

                    printf("ERROR : %d : %s\n", line\_number, src\_line);

                    fclose(f\_lst);

                    fclose(f\_obj);

                    remove(list\_filename);

                    remove(obj\_filename);

                    return;

                }

                continue;

            }

            else {  // '+' 가 아니다!

                if(instruction\_check(parsed\_line[1]) == -1)  {

                    printf("Not plus ERROR : %d : %s\n", line\_number, src\_line);

                    fclose(f\_lst);

                    fclose(f\_obj);

                    remove(list\_filename);

                    remove(obj\_filename);

                    return;

                }

                if (!strcmp(parsed\_line[1], "BASE"))

                continue;

            else if (!strcmp(parsed\_line[1], "END"))

                break;

            else

                address\_increase(&address, parsed\_line[1], parsed\_line[2]);

            }

        }

        else  { // Yes Symbol => need to Symbolize.

            if (parsed\_line[1][0] == '+') {// format4

            /\* Format 4는 좀 특이하게 생겼으므로 특이 취급을 해준다. \*/

                char for\_check[10];

                format4\_plus\_delete(for\_check, parsed\_line[1]);

                if (instruction\_check(for\_check) == -1)  {

                    printf("SYMBOL PLUS ERROR : %d : %s\n", line\_number, src\_line);

                    fclose(f\_lst);

                    fclose(f\_obj);

                    remove(list\_filename);

                    remove(obj\_filename);

                    return;

                }

            }

            else {

                if(instruction\_check(parsed\_line[1]) == -1){

                    printf("SYMBOL NO PLUS ERROR : %d : %s\n", line\_number, src\_line);

                    fclose(f\_lst);

                    fclose(f\_obj);

                    remove(list\_filename);

                    remove(obj\_filename);

                    return;

                }

            }

            if (!strcmp(parsed\_line[1], "START"))  {

                int i, temp = 0;

                for (i = 0; parsed\_line[2][i] != '\0' && parsed\_line[2][i] != '\n'; i++)  {

                    temp \*= 16;

                    temp += (parsed\_line[2][i] - '0');

                }

                address = temp;

            }

            else {

                if(parsed\_line[0][0] >= 'A' && parsed\_line[0][0] <= 'Z'){

                    // -1이 나오면 중복 삽입이므로 오류!

                    if(push\_symbol(address, parsed\_line[0]) == -1){

                        printf("ERROR at %d LINE\n", line\_number);

                        printf("%s\n", src\_line);

                        return;

                    }

                }

                address\_increase(&address, parsed\_line[1], parsed\_line[2]);

            }

        }

    }

    fclose(fp);

    /////////////////////////////////////////////////////////////////////////////////

    /\*

    PASS2 : Although sleep come... but We must make OBJCODE!!!!!

    WE finally enter the final line...!

    \*/

    fp = fopen(filename, "r");

    if(fp == NULL){

        printf("assemble PASS 2 error : We cant open the file\n");

        return;

    }

    // .asm 파일을 한 줄 씩 읽어들인다.

    last\_address = address;

    address = 0;

    line\_number = 0;

    char ts[2], objfile[255], modifys[32768], now[12];

    objfile[0] = '\0';

    int start\_address, rs;

    ts[0] = ' ';

    ts[1] = '\0';

    now[0] = '\0';

    while (1)  {

        char src\_line[LINE], parsed\_line[3][LINE], obj[INSTRUCTION];

        line\_number += 5;

        if (fgets(src\_line, LINE, fp) == NULL)

            // NULL 인 경우 왠만하면 파일 끝에 다다랐다는 것이므로 while 문을 종료한다.

            break;

        // 소스 파일에서 한 줄 읽었으면 명령어로 바꿔줘야한다.

        if (strstr(src\_line, "."))  {

            fprintf(f\_lst, "%-4d     %s", line\_number, src\_line);

            continue;

        }

        parse\_line(parsed\_line, src\_line);

        // |ㅁ|ㅁ|ㅁ| OR |ㅁ|ㅁ| |

        if (parsed\_line[0][0] == 'N')  {                                   // No Symbol => No need to into

            if(parsed\_line[1][0] == '+')  { // format4

                char t\_inst[20];

                format4\_plus\_delete(t\_inst, parsed\_line[1]);

                if(obj\_make(address, t\_inst, parsed\_line[2], obj, 4) == -1){

                    printf("NO SYMBOL PLUS format 4 ERROR %d %s\n", line\_number, src\_line);

                    fclose(f\_lst);

                    fclose(f\_obj);

                    remove(list\_filename);

                    remove(obj\_filename);

                    return;

                };

                if(!(parsed\_line[2][0] == '#' && (parsed\_line[2][1] >= '0' && parsed\_line[2][1] <= '9'))){

                    modify\_relocation(now, address);

                    strcat(modifys, now);

                }

                print\_assemble(f\_lst, line\_number, address, " ", parsed\_line[1], parsed\_line[2], obj);

                address += 4;

            }

            else if(!strcmp(parsed\_line[1], "BASE"))  { // BASE No OBJ code

                base = symbol\_find(parsed\_line[2]);

                if(base == -1){

                    printf("%s ERROR\n", src\_line);

                    fclose(f\_lst);

                    fclose(f\_obj);

                    remove(list\_filename);

                    remove(obj\_filename);

                    return;

                }

                fprintf(f\_lst, "%-5d %4s     %-6s   %-7s  %-10s  %-10s\n",

                line\_number, ts, ts, parsed\_line[1], parsed\_line[2], ts);

            }

            else if(!strcmp(parsed\_line[1], "END")){

                if((int)strlen(objfile) > 0){

                    fprintf(f\_obj, "T%06X%02X%s\n", start\_address, (int)strlen(objfile), objfile);

                }

                fprintf(f\_obj, "%s", modifys);

                fprintf(f\_obj, "E%06X\n", rs);

                fprintf(f\_lst, "%-5d %4s     %-6s   %-7s  %-10s  %-10s\n",

                    line\_number, ts, ts, parsed\_line[1], parsed\_line[2], ts);

                break;

            }

            else {

                if(obj\_make(address, parsed\_line[1], parsed\_line[2], obj, 3) == -1){

                    printf("NO SYMMMMBOL ERROR %d %s\n", line\_number, src\_line);

                    fclose(f\_lst);

                    fclose(f\_obj);

                    remove(list\_filename);

                    remove(obj\_filename);

                    return;

                }

                if(!strcmp(parsed\_line[1], "RSUB")){

                    print\_assemble(f\_lst, line\_number, address, " ", parsed\_line[1], " ", obj);

                }

                else {

                    print\_assemble(f\_lst, line\_number, address, " ", parsed\_line[1], parsed\_line[2], obj);

                }

                address\_increase(&address, parsed\_line[1], parsed\_line[2]);

            }

        }

        else  {  // YES Symbol !

            if (!strcmp(parsed\_line[1], "START"))  {

                start\_address = address;

                rs = address;

                fprintf(f\_obj, "H%-6s %06X%06X\n", parsed\_line[0], address, last\_address);

                print\_assemble(f\_lst, line\_number, address, parsed\_line[0], parsed\_line[1], parsed\_line[2], " ");

                continue;

            }

            else if(parsed\_line[1][0] == '+')  { // format 4

                char t\_inst[20];

                format4\_plus\_delete(t\_inst, parsed\_line[1]);

                if (obj\_make(address, t\_inst, parsed\_line[2], obj, 4) == -1)  {

                    printf("ERROR %d %s\n", line\_number, src\_line);

                    fclose(f\_lst);

                    fclose(f\_obj);

                    remove(list\_filename);

                    remove(obj\_filename);

                    return;

                }

                if(!(parsed\_line[2][0] == '#' && (parsed\_line[2][1] >= '0' && parsed\_line[2][1] <= '9'))){

                    modify\_relocation(now, address);

                    strcat(modifys, now);

                }

                print\_assemble(f\_lst, line\_number, address, parsed\_line[0], parsed\_line[1], parsed\_line[2], obj);

                address += 4;

            }

            else {

                if(!strcmp(parsed\_line[1], "RESW") || !strcmp(parsed\_line[1], "RESB")) {

                    if((int)strlen(objfile) > 0)  {

                        fprintf(f\_obj, "T%06X%02X%s\n", start\_address, (int)strlen(objfile) ,objfile);

                        start\_address = address;

                        strcpy(objfile, "");

                    }

                }

                if(obj\_make(address, parsed\_line[1], parsed\_line[2], obj, 3) == -1){

                    printf("ERROR %d %s\n", line\_number, src\_line);

                    fclose(f\_lst);

                    fclose(f\_obj);

                    remove(list\_filename);

                    remove(obj\_filename);

                    return;

                } // obj code make

                print\_assemble(f\_lst, line\_number, address, parsed\_line[0], parsed\_line[1], parsed\_line[2], obj);

                address\_increase(&address, parsed\_line[1], parsed\_line[2]);

            }

        }

        if(!strcmp(parsed\_line[1], "RESW") || !strcmp(parsed\_line[1], "RESB"))

            continue;

        strcat(objfile, obj);

        if((int)strlen(objfile) > OBJECT\_TEXT\_LENGTH){ // Dont over Mine.

            fprintf(f\_obj, "T%06X%02X%s\n", start\_address, (int)strlen(objfile) ,objfile);

            start\_address = address;

            strcpy(objfile, "");

        }

    }

    fclose(fp);

    fclose(f\_lst);

    fclose(f\_obj);

    printf("SUCCESSFULLY ASSEMBLED!\n");

}

void symbol()  {

   /\* 정렬하는 부분 \*/

   /\* 알파벳 순으로 정렬하자... \*/

   // Ordering by alphabetical order

    int i, j;

    for(i = 0; i < sym\_index; i++){

        for(j = i + 1; j < sym\_index; j++){

            if(strcmp(sym\_table[i].symbol,sym\_table[j].symbol) > 0){

              char temp[8];

              int int\_temp;

              strcpy(temp, sym\_table[i].symbol);

              strcpy(sym\_table[i].symbol, sym\_table[j].symbol);

              strcpy(sym\_table[j].symbol, temp);

              int\_temp = sym\_table[i].addr;

              sym\_table[i].addr = sym\_table[j].addr;

              sym\_table[j].addr = int\_temp;

            }

        }

    }

   /\* 출력하는 부분 \*/

    /\* 출력은 끝에 다다를 때 까지 하면 그만 \*/

    for(i = 0; i < sym\_index; i++){

        printf("        "); // space 8

        printf("%-7s  ", sym\_table[i].symbol);

        printf("%04X\n", sym\_table[i].addr);

    }

    return;

}

/\* 파일에서 읽은 한 줄은 공백으로 가득 차 있다.

내가 원하는 건 그 중 Symbol, operation, operand 이다.

이들을 분리해내는 함수 \*/

void parse\_line(char parse[3][LINE], char origin[LINE]){

    int i = 0, j = 0;

    if(origin[0] == ' ')  { // No Symbol => 1 or 2 chunk :)

    //If we don't have operand... then we get 'N'

        parse[0][0] = 'N';

        parse[0][1] = '\0';

        parse[2][0] = 'N';

        parse[2][1] = '\0';

        while (origin[i] == ' ')

            i++;

        while(origin[i] != ' ' && origin[i] != '\n'){

            parse[1][j] = origin[i];

            j++;

            i++;

        }

        parse[1][j] = '\0';  //instruction part

        while (origin[i] == ' ')

            i++;

        if (origin[i] == '\n' || origin[i] == '\0')  {

            parse[2][0] = 'N';

            parse[2][1] = '\0';

        }

        else  {

            j = 0;

            while (origin[i] != '\n' && origin[i] != '\0')

            {

                parse[2][j] = origin[i];

                j++;

                i++;

            }

            parse[2][j] = '\0';

        }

    }

    else  {  // Yes Symbol -> 3 chunk!

        while(origin[i] != ' '){

            parse[0][j++] = origin[i];

            i++;

        }

        parse[0][j] = '\0'; //symbol part

        j = 0;

        while (origin[i] == ' ')

            i++;

        while(origin[i] != ' '){

            parse[1][j++] = origin[i];

            i++;

        }

        parse[1][j] = '\0'; // instruction part

        j = 0;

        while(origin[i] == ' ')

            i++;

        while(origin[i] != '\n'){

            parse[2][j++] = origin[i];

            i++;

        }

        parse[2][j] = '\0'; // operand part

    }

    return;

}

/\*Symbol과 그 address를 symbol table에 집어넣어주는 함수 \*/

int push\_symbol(int addr, char inst[60]){

    int i;

    for (i = 0; i < sym\_index; i++){

        if(!strcmp(sym\_table[i].symbol, inst)){

            return -1;

        }

    }

    // symbol 넣어주기

    strcpy(sym\_table[sym\_index].symbol, inst);

    // addr 넣기

    sym\_table[sym\_index].addr = addr;

    sym\_index++;

    return 1;

}

/\* 명령어 종류에 따라 address를 증가시킨다. \*/

/\* format3 : opcode (6) nixbpe dis(12)

10진수 4096 -> 16진수 1000

format4 : opcode 6 nixbpe dis 20.

+ 있으면 format 4이다.

format2 : op 8, r1 4 r2 4 ( register가 9개 )

\*/

void address\_increase(int \*addr, char inst[LINE], char operand[LINE]){

    if(inst[0] == '+'){

        (\*addr) += 4;

    }

    else if(!strcmp(inst, "BYTE")){

        int i = 0, count = 0;

        char mode = operand[0];

        while(operand[i++] != '\'')

            ;

        while(operand[i] != '\''){

            i++;

            count++;

        }

        if(mode == 'C'){

            (\*addr) += count;

        }

        else if(mode == 'X'){

            (\*addr) += (count / 2);

        }

    }

    else if(!strcmp(inst, "WORD")){

        (\*addr) += 3;

    }

    else if(!strcmp(inst, "RESW")){

        (\*addr) += ((int)(stoi(operand)) \* 3);

    }

    else if(!strcmp(inst, "RESB")){

        (\*addr) += ((int)(stoi(operand)));

    }

    else if(!strcmp(inst, "CLEAR") || !strcmp(inst, "COMPR") || !strcmp(inst, "ADDR") ||

    !strcmp(inst, "DIVR") || !strcmp(inst, "MULR") || !strcmp(inst, "TIXR"))   {

        (\*addr) += 2;

    }

    else {

        (\*addr) += 3;

    }

}

/\*

step 1 : OPCODE

step 2 : nixbpe

    -> simple : n=1, i=1

    -> immediate : n=0, i=1 #

    -> indirect : n=1, i=0 @

    ->

    x = Array

    b = base register

    p = program counter(default)

    e = format 4

step 3 : Displacement or address

    -> format3 : 12bit

    -> format4 : 20bit : add 0 in front of Locnumber and Displacment

etc : format 2-> opcode + RegNumber + 0

\*/

/\*object code를 만드는 함수입니다. \*/

int obj\_make(int PC, char operation[LINE], char operand[LINE], char \* objcode, int format){

// PC already increased(latest)

    int k, next\_line;

    char n, i, x, b, p, e;

    char first[5], second[5], \*operands, third[5];

    next\_line = PC;

    /\* Handle Constant \*/

    if(!strcmp(operation, "RESW") || !strcmp(operation, "RESB")){

        strcpy(objcode, ""); // 얘들은 objcode가 없습니다.

        return 1;

    }

    if (!strcmp(operation, "BYTE"))

    { // 상수는 특별하게 처리해주겠어

        char new\_op[MAX\_CHAR], resulted[MAX\_CHAR];

        int index = 0;

        if (operand[0] == 'C')  { // byte

            for (k = 2; operand[k] != '\''; k++)

                new\_op[k - 2] = operand[k];

            new\_op[k - 2] = '\0';

            for (k = 0; k < strlen(new\_op); k++)  {

                char\_to\_dex(new\_op[k],resulted, &index);

            }

            strcpy(objcode, resulted);

            return 1;

        }

        if(operand[0] == 'X'){

            for (k = 2; operand[k] != '\''; k++)

                new\_op[k - 2] = operand[k];

            new\_op[k-2] = '\0';

            strcpy(objcode, new\_op);

            return 1;

        }

    }

/////////////////////////////////////////////////////////////////////////

    address\_increase(&next\_line, operation, operand);

    // update next\_line  for pc relative

    /\* Step 1: Opcode \*/

    // Geting a Opcode...

    for (k = 0; k < op\_index; k++)  {

        if(!strcmp(opcode[k].symbol, operation)){

            operands = opcode[k].op;

            break;

        }

    }

    dex\_to\_bit(first, operands[0]);

    dex\_to\_bit(second, operands[1]);

    strcpy(objcode, binary\_to\_dex(first));

/////////////////////////////////////////////////////////////////////////////////////////////////////////////

    /\* Step 2: nixbpe \*/

    // Find Symbol... If result is -1, there isn't.

    if(!strcmp(operation, "RSUB")){

        n = '1';

        i = '1';

        x = '0';

        b = '0';

        p = '0';

        e = '0';

    }

    else if (format == 4)  { // format 4 : address : 20bit

        if(operand[0] == '#')  {

            n = '0';

            i = '1';

            x = '0';

            b = '0';

            p = '0';

            e = '1';

            /\*if (operand[1] >= '0' && operand[1] <= '9') // Immediate addressing n = 0, i = 1

                p = '0';

            else  // simple addressing n=0, i=1.

                p = '1';\*/

        }

        else  {    // simple addressing.

            n = '1';

            i = '1';

            x = '0';

            b = '0';

            p = '0';

            e = '1';

        }

    }

    else if(!strcmp(operation, "CLEAR") || !strcmp(operation, "COMPR") || !strcmp(operation, "TIXR")

    || !strcmp(operation, "RSUB")) { // format 2-> opcode + RegNumber + 0

        n = '0';

        i = '0';

        x = '0';

        b = '0';

        p = '0';

        e = '0';

        switch (operand[0]) {

        case 'B':

            p = '1';

            e = '1';

            break;

        case 'S':

            b = '1';

            break;

        case 'T':

            b = '1';

            e = '1';

            break;

        case 'F':

            b = '1';

            p = '1';

            break;

        case 'A':

            e = '0';

            break;

        case 'X':

            e = '1';

            break;

        case 'L':

            p = '1';

            break;

        }

    }

    else if(!strcmp(operation, "STCH") || !strcmp(operation,"LDCH")){

        n = '1';

        i = '1';

        x = '0';

        if(strstr(operand, "BUFFER")){

            int distance = symbol\_find("BUFFER");

            if(distance == -1){

                return -1;

            }

            if(distance < next\_line){

                b = '1';

                p = '0';

            }

            else {

                b = '0';

                p = '1';

            }

            x = '1';

        }

        e = '0';

    }

    else { // format 3

        if(operand[0] == '@')  { // indirect addressing n = 1, i = 0;

            n = '1';

            i = '0';

            x = '0';

            b = '0';

            p = '1';

            e = '0';

        }

        else if (operand[0] == '#')  { // immediate addressing n = 0, i = 1;

            n = '0';

            i = '1';

            x = '0';

            b = '0';

            if (operand[1] >= '0' && operand[1] <= '9')

                p = '0';

            else

                p = '1';

            e = '0';

        }

        else  { // simple addressing. n = 1, i = 1;

            n = '1';

            i = '1';

            x = '0';

            b = '0';

            p = '1';

            e = '0';

            int distance = symbol\_find(operand);

            if(distance == -1){

                return -1;

            }

            if (PC - distance >= 4096){

                b = '1';

                p = '0';

            }

            if (strcmp(operand, "BUFFER") && strstr(operand, "BUFFER"))  { // Buffer Access

                x = '1';

            }

        }

    }

    second[2] = n;

    second[3] = i;

    second[4] = '\0';

    third[0] = x;

    third[1] = b;

    third[2] = p;

    third[3] = e;

    third[4] = '\0';

    strcat(objcode, binary\_to\_dex(second));

    strcat(objcode, binary\_to\_dex(third));

    ////////////////////////////////////////////////////////////////////////////////////////////

    /\* Step 3 : Displacement or Address \*/

    if (format == 4)  { // format 4

        int kk, temp = 0;

        char format4\_addr[7];

        for (kk = 0; kk < 6; kk++)

            format4\_addr[kk] = '0';

        if(operand[0] == '#')  { // Immediate Addressing n=0, i=1

            if(operand[1] >= '0' && operand[1] <= '9')  {

                for (k = 1; operand[k] != '\0'; k++)  {

                    temp \*= 10;

                    temp += (operand[k] - '0');

                }

            }

            else {

                char new\_op[LINE];

                format4\_plus\_delete(new\_op, operand);

                temp = symbol\_find(new\_op);

            }

        }

        else

            temp = symbol\_find(operand);

        if(temp == -1)

            return -1;

        if (p == 1)

            temp -= next\_line; // 어떤 mode냐~

        /\* Int Address Value => Char Address Value \*/

        formatting(format4\_addr, 4, temp);

        strcat(objcode, format4\_addr); //objcode에 더해준다.

    }

    else if(!strcmp(operation, "RSUB"))

        strcat(objcode, "000");

    else if (!strcmp(operation, "CLEAR") || !strcmp(operation, "TIXR"))

        strcat(objcode, "0");

    else if(!strcmp(operation, "COMPR")){ // format 2

        int tts = 0;

        char tt[2];

        while (tts < strlen(operand) && operand[tts] != ',')

            tts++;

        tts++;

        while(operand[tts] <= 'A' || operand[tts] >= 'Z')

            tts++;

        e = register\_to\_num(operand[tts]);

        tt[0] = e; tt[1] = '\0';

        strcat(objcode, tt);

    }

    else if(!strcmp(operation, "STCH") || !strcmp(operation, "LDCH"))  {

        char format3\_addr[4];

        int dist, temp;

        for (k = 0; k < 4; k++)

            format3\_addr[k] = '0';

            // x 가 1일 경우 buffer에 access 합니다

            // buffer의 위치에 따라 값을 더해줘야할 수도 있습니다.

        if(x == '1'){

            dist = symbol\_find("BUFFER");

            if(dist == -1)

                return -1;

        }

        if (b == '1')  {

            if ( dist < base ){

                dist += (int)(pow(16, 3));

            }

            dist -= base;

            temp = dist;

        }

        else if(p == '1')  {

            if(next\_line > dist){

                dist += (int)(pow(16, 3));

            }

            dist -= next\_line;

            temp = dist;

        }

        formatting(format3\_addr, 2, temp);

        strcat(objcode, format3\_addr);

    }

    else { // format 3 => 12bit

        int temp = 0, dist;

        char format3\_addr[4];

        for (k = 0; k < 4; k++)

            format3\_addr[k] = '0';

        if (operand[0] == '@')  {

            char new\_op[LINE];

            format4\_plus\_delete(new\_op, operand);

            temp = symbol\_find(new\_op);

        }

        else if (operand[0] == '#')  { //immediate addressing n=0, i=1;

            // To decimal.

            if (operand[1] >= '0' && operand[1] <= '9')  {

                for (k = 1; operand[k] != '\0' && operand[k] != '\n'; k++)  {

                    temp \*= 10;

                    temp += (operand[k] - '0');

                }

            }

            else  {

                char new\_op[LINE];

                format4\_plus\_delete(new\_op, operand);

                temp = symbol\_find(new\_op);

            }

        }

        else if(b == '1'){

            dist = symbol\_find(operand);

            if(dist == -1)

                return -1;

            dist -= base;

            temp = dist;

        }

        else   // simple addressing n =1, i = 1

            temp = symbol\_find(operand);

        if(temp == -1)

            return -1;

        if(p == '1'){ // pc Relative

            if (temp < next\_line)  {

                temp += (int)(pow(16, 3));

            }

            temp -= next\_line;

        }

        formatting(format3\_addr, 2, temp);

        strcat(objcode, format3\_addr);

    }

    return 1;

}

/\* Symbol table에서 symbol을 찾아서 그 address를 return 한다 \*/

int symbol\_find(char operation[LINE]){

    int i;

    for (i = 0; i < sym\_index; i++){

        if(!strcmp(operation, sym\_table[i].symbol)){

            return sym\_table[i].addr;

        }

    }

    return -1;

}

/\* 매개변수로 들어온 fbit는 16진수 형식입니다. 이걸 4bit binary digit으로 변환합니다. \*/

void dex\_to\_bit(char \* dest, char fbit){

    switch(fbit){

    case '0':

        dest[0] = '0';

        dest[1] = '0';

        dest[2] = '0';

        dest[3] = '0';

        break;

    case '1':

        dest[0] = '0';

        dest[1] = '0';

        dest[2] = '0';

        dest[3] = '1';

        break;

    case '2':

        dest[0] = '0';

        dest[1] = '0';

        dest[2] = '1';

        dest[3] = '0';

        break;

    case '3':

        dest[0] = '0';

        dest[1] = '0';

        dest[2] = '1';

        dest[3] = '1';

        break;

    case '4':

        dest[0] = '0';

        dest[1] = '1';

        dest[2] = '0';

        dest[3] = '0';

        break;

    case '5':

        dest[0] = '0';

        dest[1] = '1';

        dest[2] = '0';

        dest[3] = '1';

        break;

    case '6':

        dest[0] = '0';

        dest[1] = '1';

        dest[2] = '1';

        dest[3] = '0';

        break;

    case '7':

        dest[0] = '0';

        dest[1] = '1';

        dest[2] = '1';

        dest[3] = '1';

        break;

    case '8':

        dest[0] = '1';

        dest[1] = '0';

        dest[2] = '0';

        dest[3] = '0';

        break;

    case '9':

        dest[0] = '1';

        dest[1] = '0';

        dest[2] = '0';

        dest[3] = '1';

        break;

    case 'A':

        dest[0] = '1';

        dest[1] = '0';

        dest[2] = '1';

        dest[3] = '0';

        break;

    case 'B':

        dest[0] = '1';

        dest[1] = '0';

        dest[2] = '1';

        dest[3] = '1';

        break;

    case 'C':

        dest[0] = '1';

        dest[1] = '1';

        dest[2] = '0';

        dest[3] = '0';

        break;

    case 'D':

        dest[0] = '1';

        dest[1] = '1';

        dest[2] = '0';

        dest[3] = '1';

        break;

    case 'E':

        dest[0] = '1';

        dest[1] = '1';

        dest[2] = '1';

        dest[3] = '0';

        break;

    case 'F':

        dest[0] = '1';

        dest[1] = '1';

        dest[2] = '1';

        dest[3] = '1';

        break;

    }

    dest[4] = '\0';

    return;

}

/\* 우리가 아는 그 stoi 맞습니다... C는 왜 지원하지 않느냐!

지원하지 않길래 제가 만들었습니다. string으로 된 정수를 integer로 바꿉니다. \*/

int stoi(char \* target){

    int i, temp = 0, size = strlen(target);

    for (i = 0; i < size; i++){

        temp \*= 10;

        temp += (target[i] - '0');

    }

    return temp;

}

/\* binary -> Dex change! \*/

char\* binary\_to\_dex(char \* binary){

    int i, digit, temp = 0;

    char \*result;

    if (strlen(binary) == 5){

        digit = 16;

        result = (char \*)malloc(sizeof(char) \* 3);

    }

    else if(strlen(binary) == 4){

        digit = 8;

        result = (char \*)malloc(sizeof(char) \* 2);

    }

    else {

        printf("strlen : %d\n", (int)strlen(binary));

        printf("s : %s\n", binary);

        return "ERROR AT BINARY\_TO\_DEX";

    }

    for (i = 0; i < (int)strlen(binary); i++, digit /= 2){

        if(binary[i] == '1') {

            temp += digit;

        }

    }

    if( temp >= 16){

        result[0] = temp / 16;

        temp -= (result[0] \* (temp / 16));

        result[1] = temp;

        if(result[0] >= 10){

            result[0] -= 10;

            result[0] += 'A';

        }

        else {

            result[0] += '0';

        }

        if(result[1] >= 10){

            result[1] -= 10;

            result[1] += 'A';

        }

        else{

            result[1] += '0';

        }

        result[2] = '\0';

    }

    else{

        if(temp >= 10){

            result[0] = temp - 10 + 'A';

        }

        else

            result[0] = temp + '0';

        result[1] = '\0';

    }

    return result;

}

/\* 10진수 정수를 16진수로 변환한다. \*/

char int\_to\_dex(int target){

    if(target < 10){

        target += '0';

    }

    else {

        target -= 10;

        target += 'A';

    }

    return target;

}

/\* lst 파일의 양식에 맞춰서 출력한다 \*/

void print\_assemble(FILE \* fp, int addr, int location, char \*symbol, char \*operation, char \*operand, char \*objcode){

    fprintf(fp, "%-5d %04X     %-6s   %-7s  %-10s  %-10s\n",

            addr, location, symbol, operation, operand, objcode);

}

/\* 이름은 마치 character 값을 변환하는 것 같지만, 실제로는 입력된 t를 두 자리 16진수 비트로 바꿈 \*/

void char\_to\_dex(int t, char \*temp, int \*index){

    temp[(\*index)++] = (int)t / 16;

    if(temp[(\*index)-1] >= 10){

        temp[(\*index) - 1] -= 10;

        temp[(\*index) - 1] += 'A';

    }

    else {

        temp[(\*index) - 1] += '0';

    }

    temp[(\*index)++] = (int)t % 16;

    if(temp[(\*index)-1] >= 10){

        temp[(\*index) - 1] -= 10;

        temp[(\*index) - 1] += 'A';

    }

    else {

        temp[(\*index) - 1] += '0';

    }

    temp[(\*index)] = '\0';

    return;

}

char register\_to\_num(char target){

    char e;

    switch (target)  {

    case 'B':

        e = '3';

        break;

    case 'S':

        e = '4';

        break;

    case 'T':

        e = '5';

        break;

    case 'F':

        e = '6';

        break;

    case 'A':

        e = '0';

        break;

    case 'X':

        e = '1';

        break;

    case 'L':

        e = '2';

        break;

    }

    return e;

}

int instruction\_check(char \*inst){

    int i;

    for (i = 0; i < op\_index; i++){

        if(!strcmp(inst, opcode[i].symbol))

            return 1;

    }

    if(!strcmp(inst, "BASE") || !strcmp(inst, "BYTE") || !strcmp(inst, "RESW") || !strcmp(inst, "RESB") ||

        !strcmp(inst, "WORD") || !strcmp(inst, "START") || !strcmp(inst, "END"))

        return 1;

    return -1;

}

/\* 위에서 dest만 선언해서 넘겨주면 여기서 알아서 M을 만들어준다. \*/

void modify\_relocation(char \* dest, int addr)  {

    dest[0] = 'M';

    int i;

    double digit = 5.0;

    for(i = 1; i < 8; i++)

        dest[i] = '0';

    dest[8] = '5';

    dest[9] = '\n';

    dest[10] = '\0';

    addr++;

    i = 1;

    while (digit > 0)  {

        if  (addr >= (int)pow(16, digit)){

            dest[i] = int\_to\_dex(addr / (int)pow(16, digit));

            addr -= ((addr / (int)pow(16,digit)) \* (pow(16,digit)));

        }

        digit--;

        i++;

    }

    dest[i++] = int\_to\_dex(addr);

    return;

}

/\* 주어진 포맷에 맞춰서 digit은 자릿수, target은 16진수로 바꾸려는 integer \*/

void formatting(char \* format, int digit, int target) {

/\* Int Address Value => Char Address Value \*/

    int i = 0;

    while (digit > 0)  {

        if(target >= (int)pow(16, digit))  {

            format[i] = target / (int)pow(16, digit);

            target -= (format[i] \* pow(16, digit));

            format[i] = int\_to\_dex(format[i]);

        }

        i++;

        digit--;

    }

        format[i++] = int\_to\_dex(target);

        format[i] = '\0';

}

/\*format4 앞에 달려있는 +가 늘 골칫덩어리 이므로 이를 제거한다.

순수한 instruction만 얻어낸다. \*/

void format4\_plus\_delete(char \*dest, char \*src){

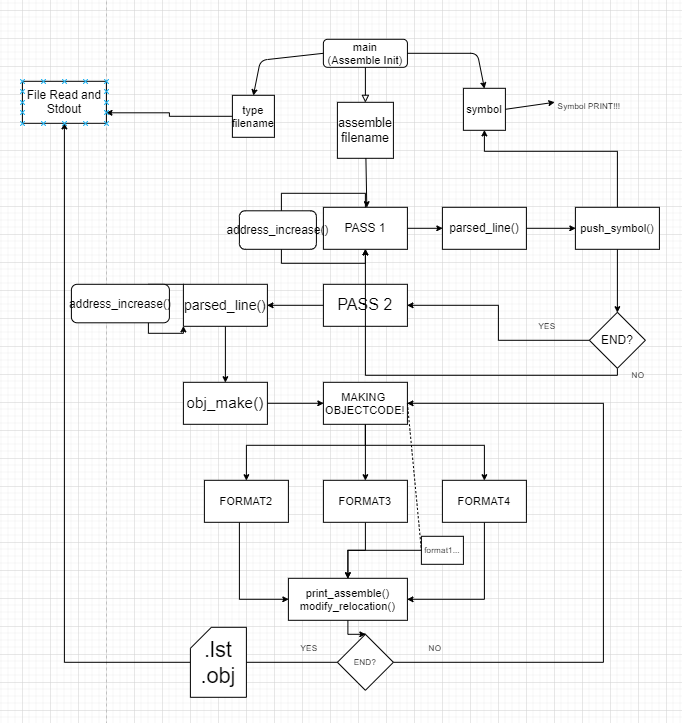
    int i;

    for (i = 1; i <strlen(src); i++)

        dest[i-1] = src[i];

    dest[i - 1] = '\0';

}



(별첨1) 흐름도