**과목명: 시스템프로그래밍**

**1분반**

**<<Project #1>>**

**서강대학교 [국제인문학부]**

**[20150038]**

**[김재원]**

목 차

1. **프로그램 개요**

앞으로 구현하게 될 SIC|XE 머신을 구현하기 위한 전 단계로서 어셈블러, 링크, 로더들을 실행하게 될 Shell과 컴파일을 통해서 만들어진 Object 코드가 적재되고 실행될 Memory Space, mnemonic을 opcode로 변환하는 Opcode Table과 관련 명령어들을 구현하는 프로그램입니다.

프로젝트 1에서는 Shell, Memory, Opcode 관련 코드를 구현할 예정입니다.

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

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

**20150038.C**

|  |  |
| --- | --- |
| **Int main(void)** | |
| **Input** | **X** |
| **Func** | SICSIM이 실행될 때 초기화되어야 할 일을 수행하고 명령어를 받습니다. 받은 명령어를 적절하게 처리해서 해당 명령어를 수행하는 함수에 넘겨줍니다. |
| **Output** | **0** |

|  |  |
| --- | --- |
| **Void operation** | |
| **Input** | **Char [MAX\_STR], int** |
| **Func** | String 형식의 명령어와 매개변수를 잘 처리할 수 있도록 분절하고 그 결과를 해당 함수에 넘겨줌. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Bool is\_range** | |
| **Input** | **char** |
| **Func** | 매개변수로 들어온 문자가 유효한 문자인지 체크하고, 유효하지 않다면 False를 return 한다. |
| **Output** | **True/false** |

|  |  |
| --- | --- |
| **Int first\_number\_extractor** | |
| **Input** | **Char [MAX\_STR], int \*** |
| **Func** | 매개변수가 있는 명령어일 경우 첫번째 매개변수를 return 한다. |
| **Output** | **입력 명령어의 첫번째 매개변수** |

|  |  |
| --- | --- |
| **Int middle\_number\_extractor** | |
| **Input** | **Char [MAX\_STR], int \*** |
| **Func** | 매개변수가 있는 명령어일 경우  중간 매개변수를 return 한다. |
| **Output** | **입력 명령어의 중간 매개변수** |

|  |  |
| --- | --- |
| **Int last\_number\_extractor** | |
| **Input** | **Char [MAX\_STR], int \*** |
| **Func** | 매개변수가 있는 명령어일 경우 중간 매개변수를 return 한다. |
| **Output** | **입력 명령어의 마지막 매개변수** |

|  |  |
| --- | --- |
| **Int rest\_and\_blank** | |
| **Input** | **Char [MAX\_STR], int \*** |
| **Func** | 매개변수 사이에 있는 공백 및 반점 등을 제거한다. |
| **Output** | **실패시 ERROR 성공시 1** |

|  |  |
| --- | --- |
| **Void argument\_extractor** | |
| **Input** | **Char [MAX\_STR],**  **Char [MAX\_STR], int \*** |
| **Func** | 입력된 명령어 String 중 ‘명령어’ 부분만 추출한다. |
| **Output** | **X** |

**=============================**

**BASIC.C**

|  |  |
| --- | --- |
| **Void init** | |
| **Input** | **X** |
| **Func** | 가능한 명령어 List를 만든다. |
| **Output** | **NULL** |

|  |  |
| --- | --- |
| **Int in** | |
| **Input** | **Char [MAX\_STR]** |
| **Func** | 명령어를 받아들이고 그 명령어 사이에 있는 빈 공간의 개수를 센다. |
| **Output** | **Blank 개수** |

|  |  |
| --- | --- |
| **Void list\_make** | |
| **Input** | **X** |
| **Func** | Help 명렁어를 수행할 때 출력할 list를 만든다. |
| **Output** | **NULL** |

|  |  |
| --- | --- |
| **Void dir** | |
| **Input** | **X** |
| **Func** | 현재 위치의 파일(폴더 포함) 목록을 출력한다. 실행파일은 \*를 붙이고 폴더는 /를 붙인다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void quit** | |
| **Input** | **X** |
| **Func** | 프로그램을 종료한다 |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void history** | |
| **Input** | **X** |
| **Func** | 지금까지 명령어를 실행할 때마다 list에 추가했는데 이제 그 list를 순회하며 출력한다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void help** | |
| **Input** | **X** |
| **Func** | List\_make()를 통해 만든 명령어 List를 출력한다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void list\_push** | |
| **Input** | **Char [MAX\_STR]** |
| **Func** | 명령어가 입력될 때마다 list에 해당 명령어를 추가한다. 그 list는 Linked List 형태이다. |
| **Output** | **X** |

**Memory.c**

|  |  |
| --- | --- |
| **Void dump** | |
| **Input** | **Int, int, int** |
| **Func** | Dump 명령어를 수행한다. 만약 세 번째 매개변수가 2라면 dump2를 실행한다. 아니라면 매개변수에 알맞게 메모리 상태를 출력한다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void dump2** | |
| **Input** | **Int, int** |
| **Func** | Start와 End가 모두 입력된 dump 명령어. 첫번째 매개변수는 start이고 두 번째 매개변수는 end 이다. Start 부터 end 주소까지 메모리 상태를 출력한다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void edit** | |
| **Input** | **Int, int** |
| **Func** | 첫번째 매개변수로 들어온 주소에 두 번째 매개변수로 들어온 Value를 넣는다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void fill** | |
| **Input** | **Int, int, int** |
| **Func** | 첫번째 매개변수로 들어온 값에해당하는 메모리 부터, 두 번째 매개변수로 들어온 값에 해당하는 메모리까지, 세 번째 매개변수를 assign 한다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void reset** | |
| **Input** | **X** |
| **Func** | 모든 메모리 영역을 0으로 만든다. |
| **Output** | **X** |

**Opcode.C**

|  |  |
| --- | --- |
| **Void opcode\_make** | |
| **Input** | **X** |
| **Func** | 해쉬 테이블 구조로 opcode 테이블을 만듭니다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void hash\_init** | |
| **Input** | **X** |
| **Func** | 전역변수로 선언된 해쉬 테이블을 초기화합니다. |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Int hash\_function** | |
| **Input** | **Const char \*** |
| **Func** | 해쉬 함수를 이용해서 문자열이 저장될 해쉬 주소를 계산한다. |
| **Output** | **해쉬 주소** |

|  |  |
| --- | --- |
| **Int hash\_find** | |
| **Input** | **Const char \*** |
| **Func** | 매개변수로 들어온 문자열의 해쉬 주소를 알아내서 해당 Memory Address의 값을 return 한다. |
| **Output** | **해당 해쉬 주소의 Memory Address의 값** |

|  |  |
| --- | --- |
| **Int hash\_add** | |
| **Input** | **Const char \*, int** |
| **Func** | 명령어와 그 명령어의 value를 매개변수로 받는다. 명령어의 해쉬 주소에 value를 저장한다. |
| **Output** | **에러 발생시 -1, 정상 종료 1** |

|  |  |
| --- | --- |
| **Void opcode\_mnemonic** | |
| **Input** | **Const char \*** |
| **Func** | Hash\_find 함수를 이용해서 해당 명령어의 value를 출력한다. 만약 존재하지 않는 명령어라면 에러 메시지를 출력한다 |
| **Output** | **X** |

|  |  |
| --- | --- |
| **Void opcodelist** | |
| **Input** | **X** |
| **Func** | Opcode Table에 있는 모든 명령어를 출력한다. |
| **Output** | **X** |

**4. 전역 변수 정의**

**20150038**

#define ERR -987654321

#define MAX\_STR 255

**Basic**

#define DIRECTORY 8

#define MAX\_STR 255

typedef struct \_linked{

    int cnt;

    char \*inst;

    struct \_linked \*next;

} linked\_list;

char \*\*inst\_list;

**Memory**

char memory[1 << 20];

int address = 0; // 주소의 시작은 0입니닷

**Opcode**

#define MAX\_INST 10

#define MAX\_HASH 20

typedef struct \_node{

    char key[MAX\_INST];

    int value;

    struct \_node \*next;

} Node;

Node \*hash\_table[MAX\_HASH];

**5. 코드 설명**

20150038.h

#ifndef \_\_20150038\_H\_\_

#define \_\_20150038\_H\_\_

#define ERR -987654321

#define MAX\_STR 255

void operation(char inst\_input[MAX\_STR], int blank);

bool is\_range(char a);

int first\_number\_extractor(char inst\_input[MAX\_STR], int \*i);

int middle\_number\_extractor(char inst\_input[MAX\_STR], int\* i);

int last\_number\_extractor(char inst\_input[MAX\_STR], int \*i);

int rest\_and\_blank(char inst\_input[MAX\_STR], int \*i);

void argument\_extractor(char src[MAX\_STR], char des[MAX\_STR], int \*i);

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* JaeWon KIM's Project 1       \*\*\*

 \* Plz Don't Copy me            \*\*\*

 \* I'm Junior of Junior         \*\*\*

 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

#include "Basic.h"

#include "Memory.h"

#include "Opcode.h"

#include "20150038.h"

int main(void)

{

    init();     // 초기화 작업을 합니다.

    opcode\_make();  // 프로그램이 시작될 때 opcode list를 만들어줍니다.

    while (1) // quit을 받을 때까지 계속 명령어를 받습니다.

    {

        int blank;

        char inst\_input[MAX\_STR];

        printf("sicsim> "); // 계속 출력하는 것

        blank = in(inst\_input);    // inst\_input으로 명령 받고 명령어에 빈칸 몇 갠지 return 받는다.

        operation(inst\_input, blank); // 받은 명령어를 통해 명령을 수행한다.

    }

    return 0;

}

/\* 명령어 들어온 걸로 실제 그 함수를 실행한다.

실행할 때마다 history list에 저장한다.\*/

// list\_push는 history에 추가하려고 넣었다.

void operation(char inst\_input[MAX\_STR], int blank){

    if(blank == 0){ // help 처럼 빈칸이 하나도 없는 명령어!

        if(!strcmp(inst\_input, "dir") || !strcmp(inst\_input, "d")){

            list\_push(inst\_input);

            dir();

        }

        else if (!strcmp(inst\_input, "dump") || !strcmp(inst\_input, "du")){

            list\_push(inst\_input);

            dump(-1, -1, 0);

        }

        else if(!strcmp(inst\_input, "help") || !strcmp(inst\_input, "h")){

            list\_push(inst\_input);

            help();

        }

        else if(!strcmp(inst\_input, "history") || !strcmp(inst\_input, "hi")){ // history

            list\_push(inst\_input);

            history();

        }

        else if(!strcmp(inst\_input, "opcodelist")){

            list\_push(inst\_input);

            opcodelist();

        }

        else if(!strcmp(inst\_input, "quit") || !strcmp(inst\_input, "q")){  // quit

            quit();

        }

        else if(!strcmp(inst\_input, "reset")){

            list\_push(inst\_input);

            reset();

        }

    }

    else if (blank == 1){ //dump start처럼 띄어쓰기가 1개!

        char blank1[MAX\_STR];

        int i = 0;

        argument\_extractor(inst\_input, blank1, &i); // 띄어쓰기가 있으면 Argument가 있당

        // 걔를 추출해야 써먹지 우선 이 명령어만 추출하자.

        // 여기있는 argument\_extractor 명령어를 추출한다.

        if (!strcmp(blank1, "dump") || !strcmp(blank1, "du"))

        {

            list\_push(inst\_input);

            int start\_point = last\_number\_extractor(inst\_input, &i);

            // 첫번쨰 매개변수 추출하기~

            if(start\_point == ERR){

                printf("DUMP 'START' ERROR\n");

                return;

            }

            dump(start\_point, -1, 1);

        }

        else if(!strcmp(blank1, "opcode")){

            list\_push(inst\_input);

            int j = 0;

            char inkey[MAX\_STR];

            while(inst\_input[i] == ' ')

                    i++;

            while(j < MAX\_STR && i < MAX\_STR && inst\_input[i] != ' '){

                inkey[j++] = inst\_input[i++];

            }

            // 이 함수는 매개변수가 문자열이다.

            inkey[j] = '\0';

            opcode\_mnemonic(inkey);

            return;

        }

    }

    else if(blank == 2){ // edit address, value 처럼 빈 칸이 두개!

        char blank1[MAX\_STR];

        int i = 0;

        argument\_extractor(inst\_input, blank1, &i); // 명령어 부터 추출해보자.

        if (!strcmp(blank1, "edit") || !strcmp(blank1, "e"))

        {

            int address, value;

            address = first\_number\_extractor(inst\_input, &i); // 첫번째 매개변수 추출

            if(address == ERR){

                printf("EDIT 'ADDRESS' ERROR\n");

                return;

            }

            if(rest\_and\_blank(inst\_input, &i) == ERR){   // 공백 제거

                printf("EDIT ERROR ADDRESS <---> VALUE\n");

                return;

            }

            value = last\_number\_extractor(inst\_input, &i);  // 두 번째 매개변수 추출

            if(value == ERR){

                printf("EDIT ERROR VALUE \n");

                return;

            }

            list\_push(inst\_input);

            edit(address, value);   //함수 실행

        }

        else if (!strcmp(blank1, "dump") || !strcmp(blank1, "du"))

        {

            int b1 = 0, b2 = 0;

            b1 = first\_number\_extractor(inst\_input, &i);    // 첫번째 매개변수

            if(b1 == ERR){

                printf("DUMP FIRST ARGUMNET ERROR\n");

                return;

            }

            if (rest\_and\_blank(inst\_input, &i) == ERR){ // 공백 제거

                printf("DUMP ERROR BETWEEN FIRST <----> SECOND ARGU\n");

                return;

            }

            b2 = last\_number\_extractor(inst\_input, &i); //두번째 매개변수

            if(b2 == ERR){

                printf("DUMP ERROR AT LAST ARGUMENT\n");

                return;

            }

            list\_push(inst\_input);

            dump(b1, b2, 2);    // b1 ~ b2 인쇄하기!

        }

    }

    else if(blank == 3){ // fill start, end, value 처럼 빈칸이 3개!

        char blank1[MAX\_STR];

        int i = 0;

        argument\_extractor(inst\_input, blank1, &i); // 명령어는 무엇이냐

        if(!strcmp(blank1, "fill") || !strcmp(blank1, "f")){

            int start, end, value;

            start = first\_number\_extractor(inst\_input, &i);  // 첫번쨰 매개변수!

            if(start == ERR){

                printf("FILL ERROR AT START ARGU\n");

                return;

            }

            if(rest\_and\_blank(inst\_input, &i) == ERR){  // 매개변수 사이의 빈칸 제거

                printf("FILL ERROR START <---> END\n");

                return;

            }

            end = middle\_number\_extractor(inst\_input, &i); // 두 번쨰 매개변수

            if(end == ERR){

                printf("FILL ERROR AT END ARGU\n");

                return;

            }

            if(rest\_and\_blank(inst\_input, &i) == ERR){ // 두 번째와 세번째 매개변수 사이 공백

                printf("FILL ERROR END <---> VALUE\n");

                return;

            }

            value = last\_number\_extractor(inst\_input, &i); // 세 번째 매개변수 무엇이냐

            if(value == ERR){

                printf("FILL ERROR AT VALUE ARGU\n");

                return;

            }

            list\_push(inst\_input);

            fill(start, end, value); // fill 함수~

        }

            return;

    }

}

/\* 매개변수가 정상적인지 체크한다. \*/

bool is\_range(char a){

    if(!((a >= '0' && a <= '9') ||(a >= 'a' && a <= 'f')|| (a >= 'A' && a <= 'F'))){

        return false;

    }

    return true;

}

/\* 첫번째 매개변수 추출하기 \*/

int first\_number\_extractor(char inst\_input[MAX\_STR], int \*i){

    int first = 0;

    while (inst\_input[\*i] != ' ' && inst\_input[\*i] != ',')

    {

        char a = inst\_input[(\*i)++];

        if (!is\_range(a))

        {

            return ERR;

        }

        first \*= 16;

        if(a >= '0' && a <='9'){

            first += (a - '0');

        }

        else if(a >= 'a' && a <= 'f'){

            first+= (a - 'a') + 10;

        }

        else if(a >= 'A' && a <= 'F'){

            first += (a - 'A') + 10;

        }

    }

    return first;

}

/\*두 번째 매개변수 추출하기 \*/

int middle\_number\_extractor(char inst\_input[MAX\_STR], int \*i){

    int middle = 0;

    while (inst\_input[\*i] != ' ' && inst\_input[\*i] != ',')

    {

        char a = inst\_input[(\*i)++];

        if(!is\_range(a)){

            return ERR;

        }

        middle \*= 16;

        if(a >= '0' && a <='9'){

            middle += (a - '0');

        }

        else if(a >= 'a' && a <= 'f'){

            middle += (a - 'a') + 10;

        }

        else if(a >= 'A' && a <= 'F'){

            middle += (a - 'A') + 10;

        }

    }

    return middle;

}

/\* 마지막 매개변수 추출하기 \*/

int last\_number\_extractor(char inst\_input[MAX\_STR], int \*i){

    int last = 0;

    while (inst\_input[\*i] != '\0')

    {

        char a = inst\_input[(\*i)++];

        if(!is\_range(a)){

            return ERR;

        }

        last \*= 16; // 16진수라서 곱해줍니당

        if(a >= '0' && a <='9')

            last += (a - '0');

        else if(a >= 'a' && a <= 'f')

            last += (a - 'a') + 10;

        else if(a >= 'A' && a <= 'F')

            last += (a - 'A') + 10;

    }

    return last;

}

/\* 매개변수 사이에 있는 빈칸 제거하기 \*/

int rest\_and\_blank(char inst\_input[MAX\_STR], int\* i){

    if(inst\_input[(\*i)++] != ','){

        return ERR;

    }

    while (inst\_input[++(\*i)] == ' ');

    return 1;

}

/\* 명령어 뽑아내기

inst\_input에서 뽑아내서 des에 넣을 것이다.

\*/

void argument\_extractor(char inst\_input[MAX\_STR], char des[MAX\_STR], int \*i){

    while (inst\_input[\*i] != ' ')

        des[\*i] = inst\_input[(\*i)++];

    des[(\*i)++] = '\0';

}

Basic.h

#ifndef \_\_Basic\_H\_\_

#define \_\_Basic\_H\_\_

#define DIRECTORY 8

#define MAX\_STR 255

typedef struct \_linked{

    int cnt;

    char \*inst;

    struct \_linked \*next;

} linked\_list;

char \*\*inst\_list;

// Basic Function

void init();

void list\_make();

int in(char [MAX\_STR]);

// Linked List Operation

void list\_push(char [MAX\_STR]);

// Execution Function

void dir();

void quit();

void history();

void help();

#endif

Basic.C

#include "Basic.h"

#include <dirent.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

linked\_list \*start = NULL;

linked\_list \*last = NULL;

void init() {

    list\_make(); // help 같은 거 할떄 list를 미리 만들어둬야지.

}

/\* 명령어를 받아들이자. c++의 string이 그리워지는 밤입니다. \*/

int in(char inst\_input[MAX\_STR]) {

    int blank = 0, inst\_index = 0;

    char c;

    while ((c = getchar()) != '\n')

    {

        if (c == ' ')

            blank++;

        inst\_input[inst\_index++] = c;

    }

    inst\_input[inst\_index] = '\0';

    return blank;

}

/\* help 명령어를 위해 리스트를 만들어줘야지 후후 \*/

void list\_make(){

    inst\_list = (char \*\*)malloc(sizeof(char \*) \* 10);

    inst\_list[0] = "h[elp]";

    inst\_list[1] = "d[ir]";

    inst\_list[2] = "q[uit]";

    inst\_list[3] = "hi[story]";

    inst\_list[4] = "du[mp] [start, end]";

    inst\_list[5] = "e[dit] address, value";

    inst\_list[6] = "f[ill] start, end, value";

    inst\_list[7] = "reset";

    inst\_list[8] = "opcode mnemonic";

    inst\_list[9] = "opcodelist";

}

void dir(){

    DIR \*dirp;

    struct dirent \*now;

    struct stat buf;

    now = (struct dirent \*)malloc(sizeof(struct dirent));

    dirp = opendir(".");

    if(dirp == NULL){

        printf("dirp Error\n");

        exit(-1);

    }

    while((now = readdir(dirp)) != NULL) {

        lstat(now->d\_name, &buf);

        if (S\_ISDIR(buf.st\_mode)) // DIRECTORY 면 \\ << 얘 붙여줄건데...

            printf("%s/\t", now->d\_name);

            /\* 실행파일이면 \* 붙여줄거임 ㅎ\*/

        else if(strstr(now->d\_name, ".exe") != NULL)

            printf("%s\*\t", now->d\_name);

        else if(strstr(now->d\_name, ".out") != NULL)

            printf("%s\*\t", now->d\_name);

        else

        /\*나머지는 그냥~ 그냥~ 그냥!\*/

            printf("%s\t", now->d\_name);

    }

    closedir(dirp); // 열었으면 닫는건 필수

    printf("\n");

    return;

}

void quit(){ // 시스템 종료

    exit(0);

}

void history(){ // 지금까지 쌓아온 리스트를 그대로 출력하겠습니다.

    int cnt = 1;

    linked\_list \*i;

    for (i = start; i != NULL; i = i->next, cnt++)

    {

        if(i->inst == "NULL"){

            continue;

        }

        printf("%d\t %s\n", cnt, i->inst);

    }

    return;

}

void help(){ /\* inst\_list에 저장한 문자열을 출력합니다. \*/

    int i;

    for (i = 0; i < 10; i++){

        printf("%s\n", inst\_list[i]);

    }

    return;

}

void list\_push(char inst\_[MAX\_STR]) { // 명령어가 입력될 때마다 history를 위한 리스트에 넣을거여요!

    linked\_list \*t = (linked\_list \*)malloc(sizeof(linked\_list));

    t->next = NULL;

    t->inst = (char \*)malloc(sizeof(char) \* 20);

    strcpy(t->inst, inst\_);

    if(start == NULL){

        start = t;

        last = t;

    }

    else{

        last->next = t;

        last = last->next;

    }

    return;

}

Memory.h

#ifndef \_\_Memory\_H\_\_

#define \_\_Memory\_H\_\_

char memory[1 << 20];

int address = 0; // 주소의 시작은 0입니닷

void dump(int , int, int );

void dump2(int, int);

void edit(int, int);

void fill(int, int, int);

void reset();

#endif

Memory.C

#include "Memory.h"

#include <stdio.h>

void dump(int dump\_start, int dump\_end, int flag){

    int i, j;

    if(flag == 2){

        // 주소값 잘못됐습니다.

        if(dump\_start > dump\_end){

            printf("FUNCTION 'DUMP' RANGE ERROR\n");

            return;

        }

            // 주소값 잘못됐다구요.

        if(dump\_start < 0 || dump\_start > 0xFFFFF ||

        dump\_end <0 || dump\_end > 0xFFFFF){

            printf("FUNCTION 'DUMP' RANGE ERROR\n");

            return;

        }

        //주솟값 잘못된 거 없으니 시작!

        dump2(dump\_start, dump\_end);

        return;

    }

    if (flag == 0) {

        // 기본 모드는 10줄을 출력하는 것이죠~

            dump\_start = address;

            dump\_end = address + 10;

    }

    else if(flag == 1){ // start 지점만 요청들어왔을 경우 어찌 할까요~

        // 주소가 이상해요~

        if(dump\_start <0 || dump\_start > 0xFFFFF){

            printf("FUNCTION 'DUMP' RANGE ERROR\n");

            return;

        }

        dump\_end = dump\_start + 10;

    }

    /\* 16진수 형식에 맞춰서 출력합니다 \*/

    for (i = 0 ; i < 10; i++){

        printf("%04X0 ", dump\_start / 16);

        for (j = 0; j < 16; j++) {

            if(dump\_start + i \* 16 + j < dump\_start ) {

                printf("   ");

                continue;

            }

            printf("%02X ", memory[i \* 16 + j]);

        }

        printf(" ; ");

        for(j = 0; j < 16; j++){

            if(memory[dump\_start + i \* 16 + j] < 0x20 ||

            memory[dump\_start + i \* 16 + j] > 0x7E){

                printf(".");

            }

            else {

                printf("%c", memory[i\*16 + j]);

            }

        }

        printf("\n");

        dump\_start += 16;

        if (dump\_start > 0xFFFFF){ // 만약 메모리가 끝까지 갔다면 0으로 초기화해주죠

            dump\_start = 0;

        }

    }

    if(flag == 0)

        address = dump\_start;

}

void dump2(int dump\_start, int dump\_end){

    int i, j, s\_f, e\_f;

    s\_f = 0;

    e\_f = 0;

    /\* start와 end를 지정해주는 경우에 대해 인쇄하기 \*/

    //16진수 형식에 맞춰서...

    while (dump\_start <= dump\_end)

    {

        printf("%04X0 ", (int)(dump\_start / 16));

        if (s\_f == 0)

        {

            for (i = 0; i < dump\_start % 16; i++)

                printf("   ");

            s\_f = 1;

        }

        for (i = dump\_start % 16; i < 16 && dump\_start <= dump\_end; i++){

            printf("%02X ", memory[dump\_start++]);

        }

        if(dump\_start > dump\_end){

            while (dump\_start % 16 != 0)

            {

                printf("   ");

                dump\_start++;

            }

        }

        printf(" ; ");

        dump\_start -= 16;

        for (j = 0; j < 16; j++)

        {

            if(dump\_start > dump\_end){

                printf(".");

                continue;

            }

            if(memory[dump\_start] < 0x20 || memory[dump\_start] > 0x7E)

            {

                printf(".");

            }

            else

            {

                printf("%c", memory[dump\_start]);

            }

            dump\_start++;

        }

        printf("\n");

    }

}

void edit(int address, int value){

    // 매개변수로 들어온 주소에 value 넣기~

    memory[address] = value;

}

/\* 매개변수 start, end 영역에 value 넣어주기 \*/

void fill(int start, int end, int value){

    int i;

    for (i = start; i <= end; i++)

    {

        memory[i] = value;

    }

}

/\* 전부 0으로 만들어버릴겁니다 ! \*/

void reset(){

    int i;

    for (i = 0; i < (1 << 20); i++)

    {

        memory[i] = 0;

    }

}

Opcode.h

#ifndef \_\_Opcode\_H\_\_

#define \_\_Opcode\_H\_\_

#define MAX\_INST 10

#define MAX\_HASH 20

typedef struct \_node{

    char key[MAX\_INST];

    int value;

    struct \_node \*next;

} Node;

Node \*hash\_table[MAX\_HASH];

void opcode\_make();

void hash\_init();

int hash\_function(const char \*str);

int hash\_find(const char \*inkey);

int hash\_add(const char \*inkey, int value);

void opcode\_mnemonic(const char \* inkey);

void opcodelist();

#endif

Opcode.C

#include "Opcode.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void opcode\_make(){

    hash\_init();

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

    //opcode.txt를 읽기 모드로 엽니다.

    if (fp == NULL)

    {

        printf("FILE READ ERROR\n");

        return;

    }

    // 파일 끝까지 읽어가면서 OPCODE 정보를 value와 temp에 저장합니다.

    while(!feof(fp)){

        int i, value = 0;

        char \*str;

        char buffer[255], temp[255];

        for (i = 0; i < 255; i++){

            buffer[i] = temp[i] = 0;

        }

        str = fgets(buffer, sizeof(buffer), fp);

        for (i = 0; i < 2; i++)

        {

            if (buffer[i] >= '0' && buffer[i] <= '9')

            {

                value \*= 16;

                value += (buffer[i] - '0');

            }

            else if (buffer[i] >= 'A' && buffer[i] <= 'Z')

            {

                value \*= 16;

                value += (buffer[i] - 'A' + 10);

            }

            else

            {

                value \*= 16;

                value += (buffer[i] - 'a' + 10);

            }

        }

        while (!(buffer[i] >= 'A' && buffer[i] <= 'Z'))

            i++;

        int j = 0;

        while (buffer[i] >= 'A' && buffer[i] <= 'Z')

        {

            temp[j++] = buffer[i++];

        }

        temp[j] = '\0';

        hash\_add(temp, value); // 읽은 값을 바탕으로 해쉬에 추가합니다.

    }

    fclose(fp); // 역할을 다했으니 파일을 닫습니다.

}

/\* 해쉬 테이블을 초기화합니다. \*/

void hash\_init(){

    int i;

    for (i = 0; i < MAX\_HASH; i++){

        hash\_table[i] = NULL;

    }

}

/\*

명령어와 그 명령어의 value를 매개변수로 받는다.

명령어를 통해 해쉬 테이블에서 어디에 위치할 지 찾는다.(hash\_function 사용)

만약 그 위치에 다른 값이 이미 들어있다면 연결 리스트 특성을 이용해서

그 노드의 Next에 연결한다.

만약 그 위치에 다른 값이 없다면, 이 노드가 최초로 그곳에 있게되므로 그냥 assign 한다

\*/

int hash\_add(const char \*inkey, int value){

    /\* 새롭게 노드를 만들고 매개변수를 그 노드에 삽입하는 과정 \*/

    Node \*new\_node = (Node \*)malloc(sizeof(Node));

    if(new\_node == NULL){

        printf("HASH\_ADD MALLOC ERROR\n");

        return -1;

    }

    strcpy(new\_node->key, inkey);

    new\_node->value = value;

    new\_node->next = NULL;

    /\* 매개변수 inkey를 이용해서 해쉬테이블에 어디에 위치할 지 보자\*/

    int hash\_index = hash\_function(inkey);

    /\* 그 위치에 다른 애가 없다? 아싸 바로 넣어야지 \*/

    if(hash\_table[hash\_index] == NULL){

        hash\_table[hash\_index] = new\_node;

    }

    /\* 그 위치에 다른 애가 있네... 줄 제일 끝으로 가야겠다\*/

    else{

        Node \*current\_node = hash\_table[hash\_index];

        while(current\_node->next != NULL)

            current\_node = current\_node->next;

        current\_node->next = new\_node;

    }

    return 1;

}

/\*

inkey에 해당하는 value를 찾을 것이다. 늘 그랬듯이 \*/

int hash\_find(const char \*inkey){

    // 매개변수 inKey를 이용해서 hash\_table에서 이 string이

    // 어느 Bucket에 있는지 찾는다.

    int hash\_index = hash\_function(inkey);

    //그 찾은 위치의 제일 앞에 있는 node를 cur이라고 명명한다.

    Node \*cur = hash\_table[hash\_index];

    // 그 노드부터 차례로 쭉 가면서 내가 찾으려는 Instruction이랑 일치하는 애가 있는가?

    while(strcmp(cur->key, inkey) != 0){

        cur = cur->next;

        if(cur == NULL){

            // 일치하는 애가 없으면...

            // 없으면...!! -1을 return 하자. 이건 일종의 에러인데, 에러처리는 다른 함수에서

            // 해줄 것이다.

            return -1;

        }

    }

    // 일치하는 애를 return 해준다.

    return cur->value;

}

/\* 입력받은 문자열을 바탕으로 해쉬 값을 만들어서 return 한다. \*/

int hash\_function(const char \*str){

    int hash = 401;

    while (\*str != '\0'){

        hash = ((hash << 4) + (int)(\*str)) % MAX\_HASH;

        str++;

    }

    return hash % MAX\_HASH;

}

/\* 명령어를 입력하면 해당하는 opcode를 출력하는 함수

매개변수로 명령어를 받고, hash\_find 함수로 그 opcode를 출력하면 된다.

\*/

void opcode\_mnemonic(const char \*inkey){

    // 관상가 양반, 내가 opcode가 될 상인가!

    int opcode\_candidate = hash\_find(inkey);

    // 아니옵니다... 에러 상이옵니다.

    if(opcode\_candidate == -1){

        printf("'%s' Not in Hash\n", inkey);

        return;

    }

    // OPCODE가 될 상이옵니다.

    printf("opcode is %X\n", opcode\_candidate);

}

/\* 프로그램 시작할 떄 opcode list를 만드는데 그걸 잘 읽도록 합니다 \*/

void opcodelist(){

    int i;

    for (i = 0; i < 20; i++){

        Node \*cur = hash\_table[i];

        printf("%d : ", i);

        if(cur != NULL){

            printf("[%s,%d]", cur->key, cur->value);

            cur = cur->next;

        }

        while (cur != NULL)

        {

            printf(" -> [%s,%d]", cur->key, cur->value);

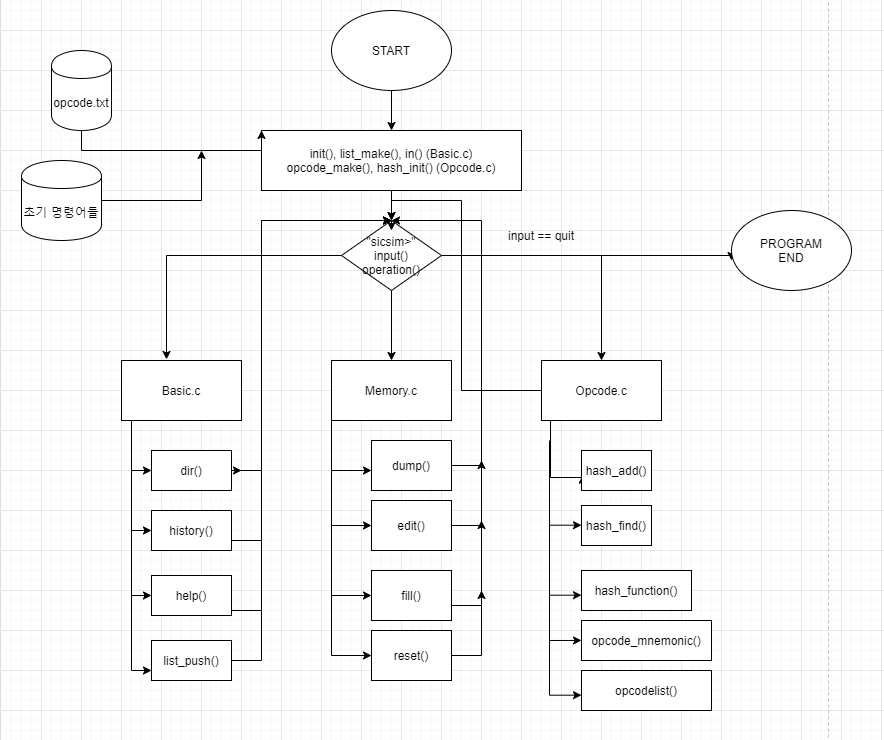
            cur = cur->next;

        }

        printf("\n");

    }

}



(별첨1) 흐름도