과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

<<Assignment 3>>

**서강대학교 컴퓨터학과**

**[학번] 20151571**

**[이름] 유용혁**

목 차

1. **프로그램 개요** 6 **1.1 주소 지정 명령어** 6 **1.2 Linking Loader 명령어** 6 **1.3 프로그램 실행 명령어** 7  
   **1.4 debug 명령어** 7
2. **프로그램 설명** 8 **2.1 프로그램 흐름도** 8
3. **모듈 정의** 9 **3.1 모듈 이름 : Est\_init()** 9 **3.1.1 기능**  9 **3.1.2 사용 변수**  9 **3.2 모듈 이름 : Est\_key()** 9 **3.2.1 기능**  9 **3.2.2 사용 변수**  9 **3.3 모듈 이름 : Est\_insert()** 9 **3.3.1 기능**  9 **3.3.2 사용 변수**  9 **3.4 모듈 이름 : Est\_find()**  9 **3.4.1 기능**  9 **3.4.2 사용 변수**  9  
    **3.5 모듈 이름 : recode\_M()** 10 **3.5.1 기능** 10 **3.5.2 사용 변수** 10 **3.6 모듈 이름 : recode\_R()** 10 **3.6.1 기능** 10 **3.6.2 사용 변수** 10 **3.7 모듈 이름 : link\_pass1()** 10 **3.7.1 기능** 10 **3.7.2 사용 변수** 10 **3.8 모듈 이름 : link\_pass2()** 11 **3.8.1 기능** 11 **3.8.2 사용 변수** 11 **3.9 모듈 이름 : command\_progaddr()** 11 **3.9.1 기능** 11 **3.9.2 사용 변수** 11 **3.10 모듈 이름 : command\_loader()** 11 **3.10.1 기능** 11 **3.10.2 사용 변수** 11 **3.11 모듈 이름 : cmp()**  12 **3.11.1 기능** 12 **3.11.2 사용 변수** 12 **3.12 모듈 이름 : command\_bp()** 12 **3.12.1 기능** 12 **3.12.2 사용 변수** 12 **3.13 모듈 이름 : add\_bp()** 12 **3.13.1 기능** 12 **3.13.2 사용 변수** 12  
    **3.14 모듈 이름 : bp\_init()**  12 **3.14.1 기능** 12 **3.14.2 사용 변수** 12 **3.15 모듈 이름 : currbp\_init()**  13 **3.15.1 기능** 13 **3.15.2 사용 변수** 13 **3.16 모듈 이름 : get\_nextbp()** 13 **3.16.1 기능** 13 **3.16.2 사용 변수** 13
4. **구조체 정의** 14  
    **4.1 구조체 이름 : EXTSYM** 14 **4.2 구조체 이름 : CSADDR** 14 **4.3 구조체 이름 : ExtSymbol** 14 **4.4 구조체 이름 : ESTAB** 14
5. **전역 변수 정의** 15 **5.1 변수 명 : curr** 15 **5.2 변수 명 : end** 15 **5.3 변수 명 : bp** 15 **5.4 변수 명 : start\_address** 15 **5.5 변수 명 : ExtTable** 15 **5.6 변수 명 : breakpoint[]** 15 **5.7 변수 명 : currbp** 15  
    **5.8 변수 명 : length** 15
6. **코드** 16  
    **6.1 loader.h** 16 **6.2 loader.c** 17 **6.3 debug.h** 26  
    **6.4 debug.c** 27  
    **6.5 run.h** 29  
    **6.6 run.c** 31

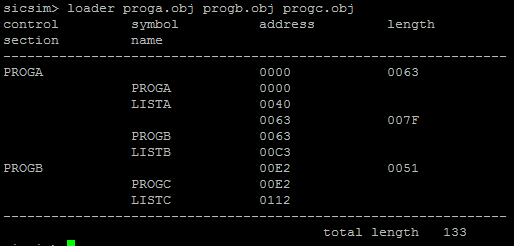
# 프로그램 개요 프로젝트 #1, #2 에서 구현한 셀(shell)에 linking과 loading 기능을 추가하는 프로그램입니다. 프로젝트 #2 에서 구현된 assemble 명령을 통해서 생성된 object 파일을 link시켜 메모리에 올리는 일을 수행하는 프로그램이다.

**1.1 주소 지정 명령어**

* **progaddr** [address]: loader 또는 run 명령어를 수행할 때 시작하는 주소를 지정합니다.

**1.2 Linking Loader 명령어**

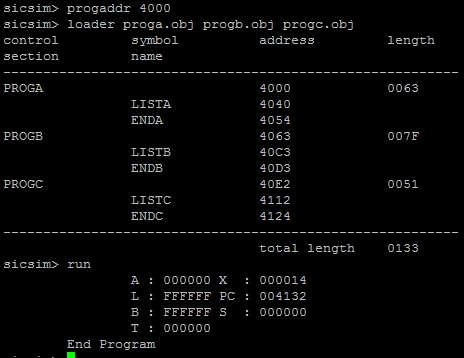
* **loader** [object filename1] [object filename2] [….] : filename1, filename2, … 에 해당하는 object 파일을 읽어서 linking 작업을 수행 후, 가상 메모리(1M)에 그 결과를 기록합니다. 파일 개수는 3개까지만 고려합니다.



위 사진 처럼 loader 실행이 성공적이라면 Load map을 화면에 출력한다.

**1.3 프로그램 실행 명령어**

* **run** : loader 명령어의 수행으로 메모리에 load된 프로그램을 실행한다. progaddr 명령어로 지정된 주소부터 시작한다.  
    
  프로그램을 수행하면 이렇게 register의 값을 화면에 출력한다.

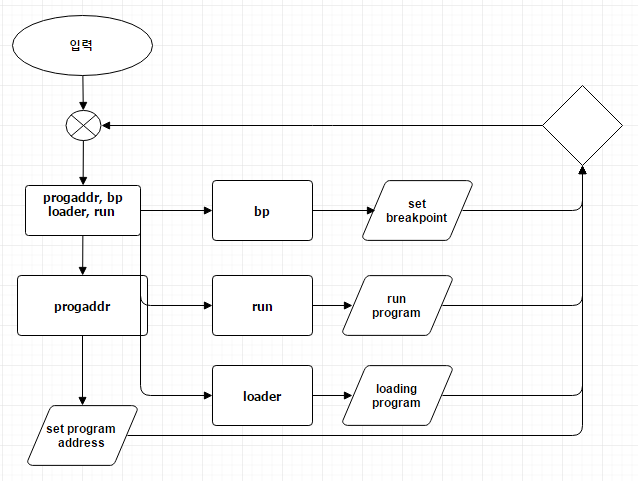


**1.4 debug 명령어**

* **bp [address]** : sicsim에서 run을 할 때 breakpoint를 지정한다. 프로그램이 실행되면 breakpoint까지 실행되고 정지한다.
* **bp clear** : 모든 breakpoint를 제거한다.
* **bp** : 모든 breakpoint를 오름차순으로 출력한다.

# 프로그램 설명

## 프로그램 흐름도



**3. 모듈 정의  
3.1 모듈 이름 : Est\_init()  
 3.1.1 기능**ESTAB을 초기화 하는 함수이다**.  
 3.1.2 사용 변수**extsptr ptr; // 임시 변수 **3.2 모듈 이름 : Est\_key()  
 3.2.1 기능**ESTAB에 넣을 때 key 값을 구해주는 함수이다**.  
 3.2.2 사용 변수**int key; // key value

size\_len // string length **3.3 모듈 이름 : Est\_insert()  
 3.3.1 기능**ESTAB에 해당하는 EXTERN SYMBOL을 넣어주는 함수이다**.  
 3.3.2 사용 변수**int key // key value **3.4 모듈 이름 : Est\_find()  
 3.4.1 기능**해당하는 extern symbol이 ESTAB에 있는지 확인해주는 함수이다. 만약에 있으면 해당하는 extern symbol의 주소를 반환한다.  
 **3.4.2 사용 변수**int key // key value **3.5 모듈 이름 : recode\_M()  
 3.5.1 기능**object code를 읽을 때 처음 recode가 M인 경우 처리해주는 함수이다. 이 부분에서 relocative addressing이 적용된다.  **3.5.2 사용 변수**int value // value 저장 변수  
int mask // for bit operator mask

int addr, tmp // value 저장 변수

int signb // signb을 저장하는 변수  
 **3.6 모듈 이름 : recode\_R()  
 3.6.1 기능**object code를 읽을 때 처음 recode가 R인 경우 처리해주는 함수이다.  **3.6.2 사용 변수**int len // string length  
int idx // index of reference number  
char name[7] // name of corresponding reference number  
 **3.7 모듈 이름 : link\_pass1()  
 3.7.1 기능**linking loader pass1을 구현하는 함수이다. ESTAB에 해당하는 프로그램 이름과 D recode에 있는 symbol들을 ESTAB에 넣어준다. **3.7.2 사용 변수**char buffer[256], str[256] // 파일 읽은 string 저장  
char name[7]; // 해당하는 extern symbol name 저장

char recode; // recode 저장

int length, len ,address, diff, flag = 1 // 프로그램 길이, 잡다한 길이, for progaddr, error\_flag

CSADDR \*ConSec // corresponding control section  
EXTSYM \*ExtSym // extsymbol table of control section **3.8 모듈 이름 : link\_pass2()  
 3.8.1 기능**linking loader pass2를 구현하는 함수이다. Pass1에서 저장한 ESTAB를 이용하여 T와 M record 부분을 처리하여 실질적으로 memory에 loading하는 과정이다. **3.8.2 사용 변수**//pass1에서 사용한 변수들은 제외하고int sign, curr, idx, value, size

// sign을 저장, 현재 파일에서 어느 부분을 읽어야하는지 저장, index, value값 저장, half byte size저장  
EXTSYM EXTARR[50] // reference number에 해당하는 extern symbol 정보 저장 **3.9 모듈 이름 : command\_progaddr()  
 3.9.1 기능**progaddr 기능을 수행하는 함수이다. 프로그램 load 할 때의 시작 주소를 정해준다.  
 **3.9.2 사용 변수**int str\_address // 입력 받은 시작 주소 저장

char \*token // char pointer for using tokenize

Char \*error // char pointer for using check error point **3.10 모듈 이름 : command\_loader()  
 3.10.1 기능**  
loader 기능을 수행하는 함수이다. Linking loader pass1, pass2를 수행하고 메모리에 해당하는 obj코드를 적재한다. **3.10.2 사용 변수**char copy[256] // copy string  
char \*token , \*filename[3] // tokenize pointer, filename string pointer  
FILE \*fp[3] // filepointer array  
int len, total\_length, error\_flag  
CSADDR \*csaddr // control section pointer **3.11 모듈 이름 : cmp()  
 3.11.1 기능**이 함수는 qsort에서 int 비교를 위해서 작성한 함수이다. **3.11.2 사용 변수**int \*a, int \*b  
 **3.12 모듈 이름 : command\_bp()  
 3.12.1 기능**이 함수는 bp 명령어를 구현하는 함수이다. bp 명령어에서는 3가지 경우가 있는데 bp address 인 경우는 breakpoint에 해당하는 address를 추가한다. bp clear인 경우에는 breakpoint를 모두 clear해버린다. bp만 입력한 경우에는 모든 breakpoint를 오름차순으로 출력한다. **3.12.2 사용 변수**char \*token , \*error // tokenize pointer, error\_check pointer

int bp, len // breakpoint  
 **3.13 모듈 이름 : add\_bp()  
 3.13.1 기능**입력 받은 breakpoint를 추가하는 함수이다. **3.13.2 사용 변수**없음 **3.14 모듈 이름 : bp\_init()  
 3.14.1 기능**breakpoint를 초기화하는 함수이다. **3.14.2 사용 변수**없음 **3.15 모듈 이름 : currbp\_init()  
 3.15.1 기능**현재 breakpoint를 초기화하는 함수이다. **3.15.2 사용 변수**

없음

**3.16 모듈 이름 : get\_nextbp()  
 3.16.1 기능**다음 breakpoint의 주소를 가져오는 함수이다. **3.16.2 사용 변수**  
없다.

**4. 구조체 정의  
  
 4.1 구조체 이름 : EXTSYM**typedef struct{

char name[7]; // external symbol name

int address; // external symbol address(relocative)

}EXTSYM;

**4.2 구조체 이름 : CSADDR**typedef struct{

char name[30]; // control section name

int address; // control section start address

int length; // control section length

int Dnum; // control section definition counter

EXTSYM ExtSym[50]; // control section external symbol

}CSADDR;

**4.3 구조체 이름 : ExtSymbol**

typedef struct EXTERNAL\_SYMBOL{

char symbol[30]; // External symbol symbol name

int address; // external symbol address

int cs\_flag; // control section flag

extsptr next; // next pointer of linked list

}ExtSymbol;

**4.4 구조체 이름 : ESTAB**

typedef struct{

const int size; // 37

ExtSymbol \*table[37]; // ESTAB hash table

CSADDR ConSec[4]; // control section array

}ESTAB;

# 전역 변수 정의 5.1 변수 명 : curr 현재 주소를 저장하고 있는 변수 5.2 변수 명 : end memory 끝의 주소를 저장하고 있는 변수 5.3 변수 명 : bp run.c에서 현재 breakpoint의 주소를 저장하는 변수 5.4 변수 명 : start\_address loader.c에서 program 시작 주소를 저장하는 전역 변수 5.5 변수 명 : ExtTable loader.c에서 ESTAB을 저장하는 변수 5.6 변수 명 : breakpoint[] debug.c에서 breakpoint들을 저장하고 있는 변수 5.7 변수 명 : currbp debug.c 에서 현재 breakpoint를 저장하고 있는 변수 5.8 변수 명 : length debug.c에서 현재 breakpoint의 개수를 저장하고 있는 변수

# 코드 6.1 loader.h

#ifndef \_\_LOADERH\_\_

#define \_\_LOADERH\_\_

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include "structure.h"

void Est\_init();

int Est\_key(char \*);

int Est\_insert( char \*, int , int );

extsptr Est\_find(char \*);

void recode\_M( EXTSYM \*, char , int , int, int , int );

int recode\_R(EXTSYM \*, char \*name);

int link\_pass1(FILE \*, int secnum, int \*);

int link\_pass2(FILE \*, int secnum, int );

int command\_progaddr(char \*buffer);

int command\_loader(char \*command);

#endif

# 6.2 loader.c

#ifndef \_\_LOADERH\_\_

#define \_\_LOADERH\_\_

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include "structure.h"

void Est\_init();

int Est\_key(char \*);

int Est\_insert( char \*, int , int );

extsptr Est\_find(char \*);

void recode\_M( EXTSYM \*, char , int , int, int , int );

int recode\_R(EXTSYM \*, char \*name);

int link\_pass1(FILE \*, int secnum, int \*);

int link\_pass2(FILE \*, int secnum, int );

int command\_progaddr(char \*buffer);

int command\_loader(char \*command);

#endif

cse20151571@cspro6:~/sp20151571\_proj3$ ^C

cse20151571@cspro6:~/sp20151571\_proj3$ ^C

cse20151571@cspro6:~/sp20151571\_proj3$ cat loader.c

#include "structure.h"

#include "loader.h"

#include "memory.h"

#include "string\_mani.h"

int start\_address = 0;

ESTAB ExtTable = { 37 };

/\*

\* EXTERN SYMBOL TABLE init function

\*/

void Est\_init(){

extsptr nptr;

for ( int i = 0 ; i < ExtTable.size; ++i){

for ( extsptr ptr = ExtTable.table[i]; ptr != NULL; ){

nptr = ptr->next;

free(ptr);

ptr = nptr;

}

ExtTable.table[i] = NULL;

}

for ( int i = 0; i < 4; ++i )

ExtTable.ConSec[i].Dnum = 0;

}

/\*

\* external symbol table key 값을 구하는함수

\*/

int Est\_key(char \*string){

int key;

size\_t len = strlen(string);

for ( size\_t i = 0; i < len; ++i )

key += string[i];

return key % ExtTable.size;

}

/\*

\* external symbol Table insert function

\* Table에 존재하지 않는 경우 0

\* 있는 경우 1을 return

\*/

int Est\_insert(char \*string, int address, int flag){

int key = Est\_key(string);

extsptr nptr;

if ( Est\_find(string) != NULL ){

fprintf(stderr, "Already exists EXTERNAL SYMBOL NAME %s\n", string);

return 0;

}

nptr = malloc(sizeof(ExtSymbol));

strncpy(nptr->symbol, string, sizeof(nptr->symbol));

nptr->address = address;

nptr->next = ExtTable.table[key];

nptr->cs\_flag = flag;

ExtTable.table[key] = nptr;

return 1;

}

/\*

\* ESTAB에 있는지 확인하는 함수

\*/

extsptr Est\_find(char \*string){

int key = Est\_key(string);

for ( extsptr ptr = ExtTable.table[key]; ptr != NULL; ptr = ptr->next){

if ( strcmp ( string, ptr->symbol ) == 0 )

return ptr;

}

return NULL;

}

/\*

\* Pass1 for linking loader

\* H, D record만 관련해서 수행한다.

\* 만약에 에러가 있으면 return 0, else return 1

\*/

int link\_pass1(FILE \*fp, int csnum, int \*csaddr){

char buffer[256];

char str[256];

char name[7];

char recode;

int length;

int len;

int address;

int diff;

int flag = 1;

CSADDR \*ConSec = &(ExtTable.ConSec[csnum]);

EXTSYM \*ExtSym = ConSec->ExtSym;

while( fgets( buffer, sizeof(buffer), fp) != NULL ) {

if ( !flag )

return 0;

len = strlen(buffer);

if ( ! ( len > 0 ) )

break;

buffer[ len - 1 ] =

buffer[ len - 1] == '\n' ? '\0' : buffer[ len - 1];

if ( !flag )

return 0;

memset(str, 0 , sizeof(str));

sscanf(buffer, "%c%[^\n]", &recode, str);

if ( recode == 'H'){

sscanf(str, "%6s%6X%6X", name, &address, &length);

delete\_whitespace(name);

str\_replace(name, " ", "");

diff = \*csaddr - address;

flag = Est\_insert( name, address + diff, 1);

if ( !flag )

return 0;

strcpy( ConSec[csnum].name, name);

ConSec->address = address + diff;

ConSec->length = length;

ExtSym[0].address = address + diff;

strcpy( ExtSym[0].name, name);

// for rellocation

}

else if ( recode == 'D'){

len = (int) strlen(str);

for ( int i = 0; i < len; i += 12){

sscanf( str + i, "%6s%6X", name, &address);

delete\_whitespace(name);

str\_replace(name, " ", "");

flag = Est\_insert(name, address + diff, 0);

if ( !flag )

return 0;

strcpy(ExtSym[ i / 12 + 1 ].name , name);

ExtSym[ i / 12 + 1].address = address + diff;

// for rellocation

ConSec->Dnum++;

}

}

else if ( recode == 'E' ) {

sscanf( str, "%6X", &address);

}

}

\*csaddr += length;

rewind(fp);

return 1;

}

/\*

\* linking loader pass2 algorithm function

\* In this fucntion, process R and M, T recode.

\* So loading this objfile to memory in this function.

\* if there is error, then return 0, else return 1

\*/

int link\_pass2(FILE \*fp, int csnum, int start\_address){

char buffer[256];

char str[256];

char name[7];

char recode;

char sign;

int length;

int len;

int address;

int flag = 1;

int curr, idx;

int diff;

int value;

int size;

EXTSYM EXTARR[50];

extsptr ptr;

while( fgets( buffer, sizeof(buffer), fp) != NULL ) {

len = strlen(buffer);

if ( ! ( len > 0 ) )

break;

buffer[ len - 1 ] =

buffer[ len - 1] == '\n' ? '\0' : buffer[ len - 1];

memset(str, 0 , sizeof(str));

if ( !flag )

return 0;

sscanf(buffer, "%c%[^\n]", &recode, str);

if ( recode == 'H'){

sscanf(str, "%6s%6X%6X", name, &address, &length);

delete\_whitespace(name);

str\_replace(name, " ", "");

ptr = Est\_find(name);

if ( ptr == NULL ){

fprintf(stderr, "NO [%s] EXTSYM ERROR!\n", name);

return 0;

}

diff = start\_address - address;

EXTARR[0].address = ptr->address;

strcpy(EXTARR[0].name, name);

}

if ( recode == 'T'){

sscanf(str, "%6X%2X", &address, &length);

curr = 8; idx = 0;

while ( idx < length ){

sscanf(str + curr, "%2X", &value);

set\_memory(address + diff + idx, value);

idx++;

curr += 2;

}

}

else if ( recode == 'R' ){

if ( ! recode\_R(EXTARR, str ) )

return 0;

}

else if ( recode == 'M'){

sscanf(str, "%6X%2X%c%2X", &address, &size, &sign, &idx);

recode\_M(EXTARR, sign, address + diff, size, csnum, idx - 1);

}

}

rewind(fp);

return 1;

}

/\*

\* progadder 명령어를 수행하는 함수

\* loading 할 때 시작 주소를 정해준다.

\* error가 있으면 return 0

\*/

int command\_progaddr(char \*buffer){

char \*token = NULL;

char sep[] = " \t";

char \*error;

int str\_address;

int len = 0;

token = strtok(buffer, sep);

if ( token == NULL ) {

fprintf(stderr, "PLEASE INPUT ADDRESS PLEASE\n");

return 0;

}

while ( token != NULL ) {

token = strtok(NULL, sep);

if ( token == NULL )

break;

len++;

if ( len > 1 ){

fprintf(stderr, "PLEASE INPUT ONLY ADDRESS PLEASE\n");

return 0;

}

str\_address = ( int ) strtol(token, &error, 16);

if ( \*error != '\0'){

fprintf(stderr, "PLEASE INPUT ADDRESS BY HEX NUMBER\n");

return 0;

}

}

if ( len < 1 )

return 0;

start\_address = str\_address;

return 1;

}

/\*

\* loader 명령어를 수행하는 함수.

\* 주어진 시작 주소로 부터 obj 파일을 memory에 적재한다.

\* 에러가 있으면 return 0, else return 1이다.

\*/

int command\_loader(char \*command){

char copy[256];

char \*token = NULL;

int address = start\_address;

char \*filename[3];

int len = 0;

FILE \*fp[3];

char sep[] = " \t";

int total\_length = 0;

int error\_flag = 1; // if there is error -> 1

CSADDR \*csaddr;

Est\_init();

command\_reset();

strncpy(copy, command, sizeof(copy));

token = strtok( copy, sep );

while( token != NULL ) {

token = strtok( NULL, " \t");

if ( token == NULL)

break;

filename[len++] = token;

}

if ( len < 1 || len > 3 ) {

fprintf(stderr, "PLEASE INPUT COMMAND CORRECTLY\n");

return 0;

}

for ( int i = 0; i < len; ++i ){

fp[i] = fopen (filename[i], "r");

if ( fp[i] == NULL ){

fprintf(stderr, "NO FILE\n");

return 0;

}

}

for ( int i = 0; i < len; ++i ){

error\_flag = link\_pass1(fp[i], i, &address);

if ( !error\_flag )

break;

}

for ( int i = 0; i < len; ++i ){

if ( ! error\_flag )

break;

error\_flag = link\_pass2(fp[i], i, ExtTable.ConSec[i].address);

if( !error\_flag )

break;

}

for ( int i = 0; i < len; ++i )

fclose(fp[i]);

if ( ! error\_flag){

Est\_init();

return 0;

}

printf("control\t\tsymbol\t\taddress\t\tlength\n");

printf("section\t\tname\n");

printf("---------------------------------------------------------------\n");

for ( int i = 0; i < len; ++i ){

csaddr = &(ExtTable.ConSec[i]);

printf("%-6s\t\t\t\t%04X\t\t%04X\n", csaddr->name, csaddr->address, csaddr->length);

total\_length += csaddr->length;

for ( int j = 0; j < csaddr->Dnum; ++j )

printf("\t\t%-6s\t\t%04X\t\t\n", csaddr->ExtSym[j].name, csaddr->ExtSym[j].address);

}

printf("---------------------------------------------------------------\n");

printf("\t\t\t\t\ttotal length %4X\n", total\_length);

start\_address = 0;

return 1;

}

/\*

\* 이 함수는 recode가 M인 경우 처리해주는 함수이다.

\* 여기서 relocative addressing을 적용한다.

\* EXTARR에 reference number에 대한 저장된 정보를 가지고서 relocative addressing을 한다.

\*/

void recode\_M(EXTSYM \*EXTARR, char sign, int address, int size, int csnum, int idx){

int value = 0;

int mask = 0;

int addr, tmp;

int signb;

signb = ( sign == '-' ) ? -1 : 1;

tmp = size / 2 + size % 2;

addr = address;

while( tmp ){

value <<= 8;

value += get\_memory(addr);

tmp--;

addr++;

}

tmp = size ;

while ( tmp ) {

mask <<= 4;

mask += 0xF;

tmp--;

}

tmp = mask & value;

tmp += EXTARR[idx].address \* signb;

tmp = mask & tmp;

value = value & (~mask);

value += tmp;

addr = address;

tmp = size / 2 + size % 2;

while( tmp ){

set\_memory(addr, ((value >> ( ( tmp - 1 ) \* 8 ) ) & 0xFF));

addr++;

tmp--;

}

}

/\*

\* 이 함수는 recode가 R인 경우 처리해주는 함수이다.

\* 이 함수에서 reference number 처리를 해준다.

\*/

int recode\_R(EXTSYM \*EXTARR, char \*str){

int len;

int idx;

char name[7];

extsptr ptr;

len = (int) strlen(str);

for ( int i = 0; i < len; i+= 8 ){

sscanf(str + i, "%2X%6s", &idx, name);

delete\_whitespace(name);

str\_replace(name, " ", "");

ptr = Est\_find(name);

if ( ptr == NULL ){

fprintf(stderr, "NO [%s] EXTSYM ERROR!\n", name);

return 0;

}

EXTARR[idx - 1].address = ptr->address;

strcpy( EXTARR[ idx - 1 ].name, ptr->symbol);

}

return 1;

}

# 6.3 debug.h

#ifndef \_\_DEBUG\_\_

#define \_\_DEBUG\_\_

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int cmp(const void \*, const void \*);

int command\_bp(char \*buffer);

void print\_bp();

void add\_bp(int address);

void bp\_init(int option);

void currbp\_init();

int get\_nextbp(int curr);

#endif

# 6.4 debug.c

#include "debug.h"

#include "structure.h"

//typedef BP\_NODE \* node;

int breakpoint[200];

int currbp = 0;

int length;

/\*

\* command breakpoint 명령어를 수행하는 함수

\*/

void bp\_init(int option){

for ( int i = 0; i < length; ++i)

breakpoint[i] = -1;

if ( option == 1)

printf("\t[ok] clear all breakpoints\n");

}

void currbp\_init(){

currbp = -1;

}

int command\_bp(char \*buffer){

char \*token;

char \*error;

int bp = -1;

int len = 0;

token = strtok(buffer, " \t");

while(token != NULL ){

token = strtok(NULL, " \t");

if ( token == NULL)

break;

len++;

if ( strcmp(token, "clear") == 0)

bp\_init(1);

bp = strtol(token, &error, 16);

if ( \*error != '\0' )

return 0;

}

if ( len == 0 )

print\_bp();

else if ( len > 1)

return 0;

if ( bp != -1 ){

add\_bp(bp);

printf("\t[ok] create breakpoint %04X\n", bp);

}

return 0;

}

int cmp(const void \*a, const void \*b){

int \*A = (int \*)a, \*B = (int \*)b;

if ( \*A < \*B )

return -1;

else if ( \*A == \*B )

return 0;

return 1;

}

void print\_bp(){

printf("\tbreakpoint\n");

printf("\t----------\n");

for ( int i = 0; i < length; ++i)

printf("\t%04X\n", breakpoint[i]);

}

void add\_bp( int address ){

for ( int i = 0; i < length; ++i)

if( breakpoint[i] == address) {

fprintf(stderr, "ALREADY EXIST breakpoint.\n");

break;

}

breakpoint[length++] = address;

qsort(breakpoint, length, sizeof(int), cmp);

}

int get\_nextbp(int curr){

if ( currbp == -1 )

return breakpoint[++currbp];

if ( breakpoint[currbp] != -1)

return breakpoint[++currbp];

else

return -1;

}

# 6.5 run.h

#ifndef \_\_RUN\_\_

#define \_\_RUN\_\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "structure.h"

typedef struct{

int A; int X; int L; int B; int S; int T; int SW; int PC;

}REGISTER;

typedef enum{ LOAD, STORE }REG\_OPTION;

int compare(int , int );

void reg\_init();

void set\_end\_addr(int address);

void set\_start\_addr(int address);

void print\_register();

int immediate(int xbpe, int data);

int indirect(int xbpe, int data, int byte, int option, int reg\_value);

int simple(int xbpe, int data, int byte, int option, int reg\_value);

void execute\_opcode(int opcode, int ni, int xbpe, int data);

void ADD(int ni, int xbpe, int data);

void ADDR(int ni, int xbpe, int data);

void STA(int ni, int xbpe, int data);

void STB(int ni, int xbpe, int data);

void STT(int ni, int xbpe, int data);;

void STS(int ni, int xbpe, int data);;

void STL(int ni, int xbpe, int data);

void STCH(int ni, int xbpe, int data);

void STX(int ni, int xbpe, int data);

void LDA(int ni, int xbpe, int data);

void LDB(int ni, int xbpe, int data);

void LDT(int ni, int xbpe, int data);

void LDS(int ni, int xbpe, int data);

void LDX(int ni, int xbpe, int data);

void LDCH(int ni, int xbpe, int data);

void JSUB(int ni, int xbpe, int data);

void JEQ(int ni, int xbpe, int data);

void JGT();

void MUL();

void MULR();

void J(int ni, int xbpe, int data);

void JLT(int ni, int xbpe, int data);

void RSUB();

void RMO();

void COMP(int ni, int xbpe, int data);

void COMPR(int data);

void TD();

void RD();

void WD();

void CLEAR(int );

void TIXR(int );

void SUB();

void SUBR();

int command\_run();

#endif

# 6.6 run.c

#include "run.h"

#include "debug.h"

#include "memory.h"

#include "opcode.h"

REGISTER reg;

int curr = 0xFFFFFF;

int end = 0xFFFFFF;

int bp;

int compare(int a, int b){

if ( a > b)

return 1;

else if ( a == b)

return 0;

return -1;

}

void reg\_init(){

reg.A = reg.X = reg.L = reg.B =

reg.S = reg.T = reg.SW = reg.PC = 0;

}

int command\_run(){

int value;

int ni;

int xbpe;

int ext\_flag;

Hnode opcode;

reg.PC = curr;

bp = get\_nextbp(reg.PC);

printf("bp : %06X\n", bp);

if(bp < reg.PC)

bp = -1;

// end of program 또는 end of memory까지

while(reg.PC < end && reg.PC <=0xFFFFF){

ext\_flag = 0;

xbpe = 0;

ni = 0;

value = 0;

if(bp != -1 && reg.PC >= bp){

printf("hahaha bp : %06X\n", bp);

print\_register();

printf("\tStop at checkpoint[%04X]\n",bp);

return 0;

}

value = get\_memory(reg.PC);

reg.PC++;

opcode = nopcode\_find(value & 0xFC); // 6bit 비교

if(opcode->format >= 1){

if(opcode->format == 3){ // format 3

ni = value & 0x3; // 2bit 비교

if(ni != 0){

value = get\_memory(reg.PC);

xbpe = value >> 4; // nixbpe중xbpe만가지고 옴.

if(xbpe & 0x01) // e = 1

ext\_flag = 1;

value &= 0x0F;

reg.PC++;

value <<= 8;

value += get\_memory(reg.PC);

reg.PC++;

if ( ext\_flag ){ // format 4

value <<= 8; // 3 bit 추가

value += get\_memory(reg.PC);

reg.PC++;

}

}

}

/\*\*\*\* format 2 \*\*\*\*/

else if(opcode->format == 2){

value = get\_memory(reg.PC);

reg.PC++;

}

execute\_opcode(opcode->n\_opcode, ni, xbpe, value);

}

curr = reg.PC;

}

print\_register();

printf("\tEnd Program\n");

currbp\_init();

return 0;

}

int immediate(int xbpe, int data){

int value = 0;

if(!(xbpe & 0x01) && (data & 0x800)) // e = 0 and

data |= 0xFFFFF000;

if ( xbpe == 0 ) // value = disp

value = data;

else if ( xbpe & 0x01 ) // e = 1, extend and value = addr

value = data;

else if ( xbpe & 0x02 ) // p = 1, pc relocative

value = (data + reg.PC);

else if ( xbpe & 0x04 ) // b = 1, base relocative

value = ( data + reg.B);

return value;

}

int indirect(int xbpe, int data, int byte, int option, int reg\_value){

int value;

int address;

if( xbpe & 0x01 ) // e = 1, extend and value = addr

address = simple(xbpe, data, 5, LOAD, 0);

else // value = disp

address = simple(xbpe, data, byte, LOAD, 0);

if(byte == 1){

if(option == LOAD){

if(address <= 0xFFFFF)

value = get\_memory(address);

return value;

}

else{

if(address <= 0xFFFFF)

set\_memory(address, reg\_value);

return 0;

}

}

/\* \*\*\* load from memory \*\*\*\*/

if(option == LOAD){

if(address <= 0xFFFFF){

value = get\_memory(address);

}

if(address + 1 <= 0xFFFFF){

value = value << 8;

value += get\_memory(address + 1);

}

if(address + 2 <= 0xFFFFF){

value = value << 8;

value += get\_memory(address + 2);

}

}

/\* \*\*\* reg\_value at memory \*\*\*\*/

else{

if(address <= 0xFFFFF)

set\_memory(address, ((reg\_value >> 16) & 0xFF));

if(address + 1 <= 0xFFFFF)

set\_memory(address+1, ((reg\_value >> 8) & 0xFF));

if(address + 2 <= 0xFFFFF)

set\_memory(address+2, (reg\_value & 0xFF));

return 0;

}

return value;

}

int simple(int xbpe, int data, int byte, int option, int reg\_value){

int value = 0;

int address = data;

if ( xbpe & 0x01 )

byte = 5;

if ( byte == 3)

if(data & 0x800) // if negative

data |= 0xFFFFF000;

if ( xbpe & 0x08 ) // x = 1, index

address = data + reg.X;

if ( xbpe & 0x02 ) // p = 1, pc relative

address += reg.PC;

else if( xbpe & 0x04 ) // b = 1, base relative

address += reg.B;

if(byte == 1) { // CH 인 경우

if(option == LOAD){

if(address <= 0xFFFFF)

value = get\_memory(address);

return value;

}

else{

if(address <= 0xFFFFF)

set\_memory(address, reg\_value);

return 0;

}

}

if(option == LOAD){

if(address <= 0xFFFFF)

value = get\_memory(address);

if(address + 1 <= 0xFFFFF){

value = value << 8;

value += get\_memory(address + 1);

}

if(address + 2 <= 0xFFFFF){

value = value << 8;

value += get\_memory(address + 2);

}

}

else{

if(address <= 0xFFFFF)

set\_memory(address, ((reg\_value >> 16) & 0xFF));

if(address + 1 <= 0xFFFFF)

set\_memory(address+1, ((reg\_value >> 8) & 0xFF));

if(address + 2 <= 0xFFFFF)

set\_memory(address+2, (reg\_value & 0xFF));

return 0;

}

return value;

}

void ADD(int ni, int xbpe, int data){

int value;

if ( ni == 2)

value = indirect(xbpe, data, 3, LOAD, reg.A);

else if ( ni == 3)

value = simple(xbpe, data, 3, LOAD, reg.A);

reg.A += value;

}

void ADDR(int ni, int xbpe, int data){

int \*value1, \*value2;

switch(data >> 4){

case 0:

value1 = &reg.A;

break;

case 2:

value1 = &reg.L;

break;

case 3:

value1 = &reg.B;

break;

case 4:

value1 = &reg.S;

break;

case 5:

value1 = &reg.T;

break;

default:

break;

}

switch(data & 0x0F){

case 0:

value2 = &reg.A;

break;

case 2:

value2 = &reg.L;

break;

case 3:

value2 = &reg.B;

break;

case 4:

value2 = &reg.S;

break;

case 5:

value2 = &reg.T;

break;

default:

break;

}

\*value2 += \*value1;

}

void STA(int ni, int xbpe, int data){

if ( ni == 2)

indirect(xbpe, data, 3, STORE, reg.A);

else if ( ni == 3 )

simple(xbpe, data, 3, STORE, reg.A);

}

void STB(int ni, int xbpe, int data){

if ( ni == 2)

indirect(xbpe, data, 3, STORE, reg.B);

else if ( ni == 3 )

simple(xbpe, data, 3, STORE, reg.B);

}

void STT(int ni, int xbpe, int data){

if ( ni == 2)

indirect(xbpe, data, 3, STORE, reg.T);

else if ( ni == 3 )

simple(xbpe, data, 3, STORE, reg.T);

}

void STS(int ni, int xbpe, int data){

if ( ni == 2)

indirect(xbpe, data, 3, STORE, reg.S);

else if ( ni == 3 )

simple(xbpe, data, 3, STORE, reg.S);

}

void STL(int ni, int xbpe, int data){

if ( ni == 2 )

indirect(xbpe, data, 3, STORE, reg.L);

else if ( ni == 3 )

simple(xbpe, data, 3, STORE, reg.L);

}

void STCH(int ni, int xbpe, int data){

if ( ni == 2 )

indirect(xbpe, data, 1, STORE, reg.A & 0xFF);

else if(ni == 3)

simple(xbpe, data, 1, STORE, reg.A & 0xFF);

}

void STX(int ni, int xbpe, int data){

if(ni == 2)

indirect(xbpe, data, 3, STORE, reg.X);

else if(ni == 3)

simple(xbpe, data, 3, STORE, reg.X);

}

void LDA(int ni, int xbpe, int data){

if(ni == 1)

reg.A = immediate(xbpe, data);

else if(ni == 2)

reg.A = indirect(xbpe, data, 3, LOAD, 0);

else if(ni == 3)

reg.A = simple(xbpe, data, 3, LOAD, 0);

}

void LDB(int ni, int xbpe, int data){

if(ni == 1)

reg.B = immediate(xbpe, data);

else if(ni == 2)

reg.B = indirect(xbpe, data, 3, LOAD, 0);

else if(ni == 3)

reg.B = simple(xbpe, data, 3, LOAD, 0);

}

void LDT(int ni, int xbpe, int data){

if(ni == 1)

reg.T = immediate(xbpe, data);

else if(ni == 2)

reg.T = indirect(xbpe, data, 3, LOAD, 0);

else if(ni == 3)

reg.T = simple(xbpe, data, 3, LOAD, 0);

}

void LDX(int ni, int xbpe, int data){

if(ni == 1)

reg.X = immediate(xbpe, data);

else if(ni == 2)

reg.X = indirect(xbpe, data, 3, LOAD, 0);

else if(ni == 3)

reg.X = simple(xbpe, data, 3, LOAD, 0);

}

void LDS(int ni, int xbpe, int data){

if(ni == 1)

reg.S = immediate(xbpe, data);

else if(ni == 2)

reg.S = indirect(xbpe, data, 3, LOAD, 0);

else if(ni == 3)

reg.S = simple(xbpe, data, 3, LOAD, 0);

}

void LDCH(int ni, int xbpe, int data){

if(ni == 1)

reg.A = immediate(xbpe, data);

else if(ni == 2)

reg.A = indirect(xbpe, data, 1, LOAD, 0);

else if(ni == 3)

reg.A = simple(xbpe, data, 1, LOAD, 0);

}

void JSUB(int ni, int xbpe, int data){

int value;

if(ni == 2)

value = simple(xbpe, data, 3, LOAD, 0);

if(ni == 3)

value = immediate(xbpe, data);

reg.L = reg.PC;

reg.PC = value;

currbp\_init();

bp = get\_nextbp(reg.PC);

}

void JEQ(int ni, int xbpe, int data){

int value;

if(reg.SW == 0){

if(ni == 2)

value = simple(xbpe, data, 3, LOAD, 0);

if(ni == 3)

value = immediate(xbpe, data);

reg.PC = value;

currbp\_init();

bp = get\_nextbp(reg.PC);

}

}

void J(int ni, int xbpe, int data){

int value;

if(ni == 2)

value = simple(xbpe, data, 3, LOAD, 0);

if(ni == 3)

value = immediate(xbpe, data);

reg.PC = value;

currbp\_init();

bp = get\_nextbp(reg.PC);

}

void JLT(int ni, int xbpe, int data){

int value;

if(reg.SW < 0){

if(ni == 2)

value = simple(xbpe, data, 3, LOAD, 0);

if(ni == 3)

value = immediate(xbpe, data);

reg.PC = value;

currbp\_init();

bp = get\_nextbp(reg.PC);

}

}

void RSUB(){

reg.PC = reg.L;

currbp\_init();

bp = get\_nextbp(reg.PC);

}

void COMP(int ni, int xbpe, int data){

int value;

if(ni == 1)

value = immediate(xbpe, data);

else if(ni == 2)

value = indirect(xbpe, data, 3, LOAD, 0);

else if(ni == 3)

value = simple(xbpe, data, 3, LOAD, 0);

reg.SW = compare(reg.A, value);

}

void COMPR(int data){

int value1, value2;

switch(data >> 4){

case 0:

value1 = reg.A;

break;

case 2:

value1 = reg.L;

break;

case 3:

value1 = reg.B;

break;

case 4:

value1 = reg.S;

break;

case 5:

value1 = reg.T;

break;

default:

break;

}

switch(data & 0x0F){

case 0:

value2 = reg.A;

break;

case 2:

value2 = reg.L;

break;

case 3:

value2 = reg.B;

break;

case 4:

value2 = reg.S;

break;

case 5:

value2 = reg.T;

break;

default:

break;

}

reg.SW = compare(value1, value2);

}

void TD(){

reg.SW = 1;

}

void RD(){

}

void WD(){

reg.SW = -1;

}

void CLEAR(int data){

switch(data >> 4){

case 0:

reg.A = 0;

break;

case 1:

reg.X = 0;

break;

case 2:

reg.L = 0;

break;

case 3:

reg.B = 0;

break;

case 4:

reg.S = 0;

break;

case 5:

reg.T = 0;

break;

default:

break;

}

}

void TIXR(int data){

int value;

reg.X++;

switch(data >> 4){

case 0:

value = reg.A;

break;

case 2:

value = reg.L;

break;

case 3:

value = reg.B;

break;

case 4:

value = reg.S;

break;

case 5:

value = reg.T;

break;

default:

break;

}

reg.SW = compare(reg.X, value);

}

void print\_register(){

printf("\t\tA : %06X X : %06X\n\t\tL : %06X PC : %06X\n\t\tB : %06X S : %06X\n\t\tT : %06X\n",

reg.A, reg.X, reg.L, reg.PC, reg.B, reg.S, reg.T);

}

void execute\_opcode(int opcode, int ni, int xbpe, int data){

switch(opcode){

case 0x00:

LDA(ni, xbpe, data);

break;

case 0x68:

LDB(ni, xbpe, data);

break;

case 0x74:

LDT(ni, xbpe, data);

break;

case 0x04:

LDX(ni, xbpe, data);

break;

case 0x50:

LDCH(ni, xbpe, data);

break;

case 0x0C:

STA(ni, xbpe, data);

break;

case 0x14:

STL(ni, xbpe, data);

break;

case 0x10:

STX(ni, xbpe, data);

break;

case 0x54:

STCH(ni, xbpe, data);

break;

case 0x48:

JSUB(ni, xbpe, data);

break;

case 0x30:

JEQ(ni, xbpe, data);

break;

case 0x38:

JLT(ni, xbpe, data);

break;

case 0x3C:

J(ni, xbpe, data);

break;

case 0x28:

COMP(ni, xbpe, data);

break;

case 0xE0:

TD();

break;

case 0xD8:

RD();

break;

case 0x4C:

RSUB();

break;

case 0xDC:

WD();

break;

case 0xB4:

CLEAR(data);

break;

case 0xA0:

COMPR(data);

break;

case 0xB8:

TIXR(data);

break;

}

}