과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 3>>

**서강대학교 컴퓨터학과**

**[학번] 20171667**

**[이름] 이용욱**

목 차

1. 프로그램 개요 3

2. 프로그램 설명 3

2.1 프로그램 흐름도 3

3. 모듈 정의 3

3.1 모듈 이름 : linkingloader 3

3.1.1 기능 3

3.1.2 사용 변수 3

3.2 모듈 이름: run 3

3.2.1 기능 3

3.2.2 사용 변수 3

3.3 모듈이름: Vector 4

3.3.1 기능 4

3.3.2 사용변수 4

3.4 모듈이름: Breakpoints 5

3.4.1 기능 5

3.4.2 사용변수 5

4. 전역 변수 정의 5

4.1 extern int32\_t A, X, L, PC, B, S, T, SW; 5

4.2 extern unsigned int progaddr; 5

4.3 extern unsigned int execute\_addr; 5

## 4.4 extern int total\_length; 6

## 5 코드 7

5.1 breakpoints.h

5.2 breakpoints.c

5.3 linking\_loader.h

5.4 linking\_loader.c

5.5 run.h

5.6 run.c

5.7 vector.h

5.8 vector.c

# 프로그램 개요

프로젝트 1에서 생성한 sic/xe머신에서, 프로젝트 2에서 생성한 obj code를 받아 메모리에 linking load한다. 그리고 실행하는 기능과 break point를 넣는 기능도 구현한다.

# 프로그램 설명

## 프로그램 흐름도

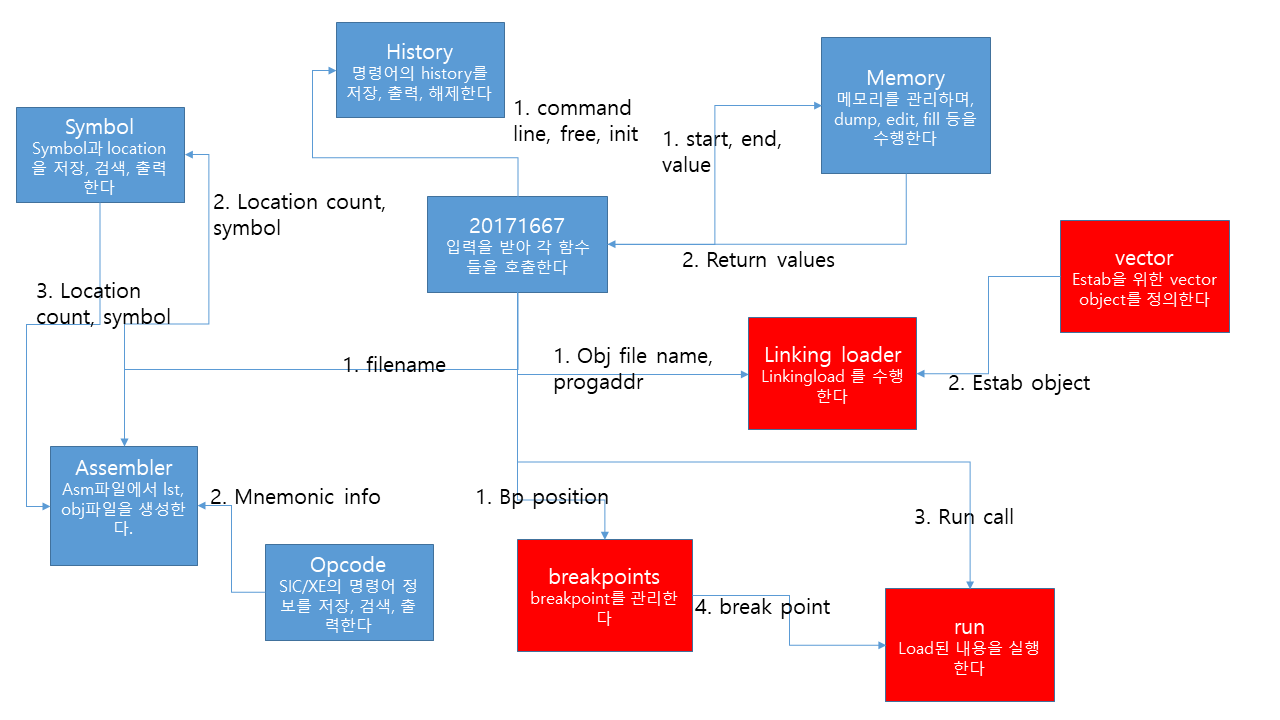


그림 1> 프로그램 흐름도 (붉은 색이 이번 프로젝트에서 추가)

우선 main함수가 있는 Main모듈이 호출된다. Main모듈이지만 소스파일의 이름은 명세에서 요구하는 대로 20171667.c, 20171667.h로 하였다. 여기서는 명령을 입력받고, 이를 해석하여 해당하는 함수를 출력한다.

이번 과제에서는 break points, run, linkingloader모듈을 호출한다.

break points 모듈은 break point를 관리한다. Bp를 set, reset하고, 모든 bp를 출력하거나, bp의 존재를 확인하는 일을 담당한다. 이는 main이나, run에서 호출될 것 이다.

linkingloader모듈에서는 강의자료와 같이 주어진 obj파일을 linkingloading 한다. 이 때에는 reference number을 용하므로 estab은 배열로 구현할 수 있다. Index로 원소를 접근하고, 뒤에 push하는 연산을 지원해야하기 때문에 vector모델에서 cpp의 vector와 같은 구조체를 구현하였다. Pass1에서는 H, D, E record에서 ESTAB을 완성하였다. Pass2에서는 ESTAB을 바탕으로 T record를 load하고, M record에서 이를 수정하였다.

Run 모듈에서는 로드된 instruction 을 실행하였다. PC 레지스터의 값으로 시작하는 4바이트를 가져와 첫 바이트를 읽고 어떤 형식의 instruction인지 파악하였다. 그 후 해당하는 메모리 접근 방식, addressing 방식으로 TA(target address)와 TV(target value)를 얻었다. 이를 SIC/XE instruction table에 나온 바와 같이 실행하였다.

이 때 TD, WD, RD 등 I/O instruction에는 실제 디바이스가 없으므로 적당한 입력을 주었다. TD의 경우에는 항상 device가 ready되어 있는 것으로 정했으며, 입력은 “IN”이라는 두 문자를 입력하였다. 그리고 TD, WD, RD의 경우에는 어떻게 이 명령을 처리하였는지 출력하였다. H record에서는 name of program, cs\_length을 찾아 ESTAB에 넣어준다. D record에서는 ESTAB에 Symbol의 이름과 주소를 넣었다.

# 모듈 정의

## 모듈 이름 : linkingloader

### 기능

Linkingload 를 담당한다. main모듈에서 파일명을 받아온다. 최대 3개 받아온다. 이를 pass1과 pass2에 넘긴다. Pass1에서는 obj파일을 하나씩, 한 줄씩 읽는다. 첫 문자로 record를 분류하고, 맞는 처리를 한다. H record에서는 프로그램 이름과 시작 주소, 프로그램 길이를 얻는다. D record에서는 extdef를 ESTAB에 push한다. Pass2 에서는, 각 Control Section별로 local estab이 존재한다. R record에서 참조하고자 하는 symbol을 local estab에 저장하여 나중에 reference number로 참조할 수 있도록 한다. T record에서는 메모리에 값을 저장한다. M record에서는 이를 parse하여 필요한 값을 더해준다.

### 사용 변수

unsigned int progaddr = 0;

// 프로그램의 시작 주소

unsigned int execute\_addr;

// 실행을 시작할 주소

int total\_length;

// 프로그램의 총 길리

struct vector estab;

// ESTAB

unsigned int csaddr = 0;

// CSADDR

char record;

// record letter of current line

char name[6+1];

// section name

unsigned int start\_addr, cs\_length;

// start address, control section length

char line[100];

// one line

## 모듈 이름: run

### 기능

Load 된 instruction을 run 한다. from\_bp에서 break point에서부터의 실행인지 알고, 아니라면 PC 레지스터를 proaddr + total length로 초기화한다. PC 레지스터의 값으로 시작하는 4바이트를 가져와 첫 바이트를 읽고 첫 6 비트로부터 어떤 형식의 instruction인지 파악하였다. 그 후 해당하는 명령을 처리한다. 레지스터의 경우 uint32\_t 변수로 선언되어 있다. 비교 연산의 경우에는 좌항에서 우항을 뺀 값을 SW레지스터에 저장하였다. 3,4 형식의 경우에는 첫 두 바이트에서 메모리 접근 방식, addressing 방식을 파악하여 TA(target address)와 TV(target value)를 얻었다. 이를 SIC/XE instruction table에 나온 바와 같이 실행하였다.

이 때 TD, WD, RD 등 I/O instruction에는 실제 디바이스가 없으므로 적당한 입력을 주었다. TD의 경우에는 항상 device가 ready되어 있으므로 항상 SW = -1; 하는 것으로 정했으며, 입력은 uint8\_t device\_input[] = {'I', 'N', '\0', '\0', }; 이라는 배열에서 순서대로 문자를 fetch하였다. 그리고 TD, WD, RD의 경우에는 어떻게 이 명령을 처리하였는지 출력하였다. “[TD] : success”, “[WD] : ‘I’” 와 같은 형식으로 출력하였다.

PC레지스터의 값에 Break point가 있는 경우 run()을 종료하였다. 이때에 레지스터 변수가 static 메모리에 저장되어있으므로 다시 run()했을 때 이어서 실행이 가능하다.

프로그램의 범위를 벗어난 경우, run()을 종료하였다.

### 사용 변수

*const int POS\_N = 6;*

*const int POS\_I = 7;*

*const int POS\_X = 0;*

*const int POS\_B = 1;*

*const int POS\_P = 2;*

*const int POS\_E = 3;*

*// 각 flag의 위치*

typedef struct word{

union{

struct{

uint8\_t b3;

uint8\_t b2;

uint8\_t b1;

uint8\_t b0;

};

uint32\_t all;

};

}word;

// big endian 형식인 4바이트 정수

struct Float{

u\_int64\_t frac : 36;

u\_int64\_t exp : 11;

u\_int64\_t sign : 1;

};

// float를 저장하는 구조체

int32\_t A, X, L, PC, B, S, T, SW;

// 레지스터 변수

int from\_bp = 0;

// breakpoint로 부터 실행인지, 첫 실행인지 저장

const int is\_input\_test\_device\_ready = 1;

// 디바이스가 준비되었는지 저장

uint8\_t device\_input[] = {'I', 'N', '\0', '\0', '\0', '\0', '\0', '\0', '\0'};

int cnt\_device\_input = 0;

// 디바이스의 입력을 저장

int format;

// 현재 inst의 foramt을 저장

uint32\_t TA;

// target address

uint32\_t TV;

// target value

## 모듈이름: Vector

### 기능

C++ STL의 vector처럼 Random access 가 가능하면서 가변길이인 구조체가 필요해서 만들었다. C에서 class를 구현하는 방식을 배워 만들었다. External\_symbol의 데이터를 받아 추가하는 emplace\_back, 해당 인덱스에의 원소를 반환하는 get(), 이름으로 원소를 찾는 find, 그리고 출력을 하는 print가 있다. 그리고 이런 함수들을 멤버함수로 바인딩하는 new 함수가 있다.

### 사용변수

typedef struct node\_external\_symbol{

char name[7];

unsigned int addr;

int is\_control\_section;

int length;

}external\_symbol;

// node for external symbol

struct vector {

int size;

external\_symbol\* data;

int (\*emplace\_back)(struct vector\* this, char name[7], unsigned int addr, int is\_control\_section, int length);

external\_symbol\* (\*get) (struct vector\* this, int idx);

external\_symbol\* (\*find) (struct vector\* this, char name[7]);

void (\*print)(struct vector\* this);

};

// vector object

## 모듈이름: Breakpoints

### 기능

Breakpoints 를 관리한다. Bp의 존재 여부를 저장하는 배열을 선언하고 이를 조작한다. Clear, print, set, exist 을 수행한다.

### 사용변수

int bp[MEMORY\_SIZE];

// bp 의 존재 여부 저장

# 전역 변수 정의

## extern int32\_t A, X, L, PC, B, S, T, SW;

레지스터를 저장한다. 레지스터는 run()함수의 종료시에도 값을 보존해야 하므로 전역변수이다.

## extern unsigned int progaddr;

마지막으로 load한 프로그램의 PROGADDR 이다

## extern unsigned int execute\_addr;

마지막으로 load한 프로그램의 시작 주소이다

## extern int total\_length;

마지막으로 load한 프로그램의 총 길이이다

.

# 코드 (프로젝트 1에서 변경된 파일만 작성)

## breakpoints.h

#ifndef PROJECT\_BREAKPOINTS\_H  
#define PROJECT\_BREAKPOINTS\_H  
  
**void** clear\_bp();  
  
**void** print\_bp();  
  
**void** set\_bp(**int** idx);  
  
**int** exist\_bp(**int** idx);  
  
#endif //PROJECT\_BREAKPOINTS\_H

## breakpoints.c

#include "20171667.h"  
#include "breakpoints.h"  
  
**int** bp[MEMORY\_SIZE];  
// bp 의 존재 여부 저장  
  
**void** clear\_bp(){  
 // bp 모두 클리어  
 printf("[ok] clear all breakpoints\n");  
 memset(bp, 0, MEMORY\_SIZE);  
}  
  
**void** print\_bp(){  
 // bp 출력  
 printf("breakpoint\n");  
 printf("----------\n");  
 **for** (**int** i = 0; i < MEMORY\_SIZE; ++i) {  
 **if**(bp[i]){  
 printf("%X\n", i);  
 }  
 }  
}  
  
**void** set\_bp(**int** idx){  
 // bp 를 만든다  
 bp[idx] = 1;  
 printf("[ok] create breakpoint %X\n", idx);  
}  
  
  
**int** exist\_bp(**int** idx){  
 // bp의 존재 확인  
 **return** bp[idx];  
}

## linkingloader.h

#ifndef PROJECT\_LINKING\_LOADER\_H  
#define PROJECT\_LINKING\_LOADER\_H  
  
**int** linker\_pass1(**int** num\_files, **char** \*\*files);  
**int** linker\_pass2(**int** num\_files, **char** \*\*files);  
**int** linking\_loader(**int** num\_files, **char**\* file1, **char**\* file2, **char**\* file3);  
  
  
**extern unsigned int** progaddr;  
**extern unsigned int** execute\_addr;  
**extern int** total\_length;  
#endif //PROJECT\_LINKING\_LOADER\_H

## linkingloader.c

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#include "linking\_loader.h"  
#include "vector.h"  
#include "memory.h"  
#include "20171667.h"  
  
**unsigned int** progaddr = 0;  
// 프로그램의 시작 주소  
**unsigned int** execute\_addr;  
// 실행을 시작할 주소  
**int** total\_length;  
// 프로그램의 총 길리  
**struct** vector estab;  
// ESTAB  
  
**unsigned int** hex\_from\_substring(**char**\* str, **int** begin, **int** length){  
 **char** temp[100];  
 strncpy(temp, str+begin, length);  
 temp[length] = '\0';  
 **return** strtol(temp, NULL, 16);  
}  
// extract hex number from substring  
  
**int** linking\_loader(**int** num\_files, **char**\* file1, **char**\* file2, **char**\* file3){  
 total\_length = 0;  
 estab = vector.new();  
 **char**\* files[3] = {file1, file2, file3};  
 linker\_pass1(num\_files, files);  
 linker\_pass2(num\_files, files);  
}  
// main function of linking\_loader  
  
**int** linker\_pass1(**int** num\_files, **char** \*\*files){  
 **unsigned int** csaddr = 0;  
 // CSADDR  
 **for** (**int** file\_idx = 0; file\_idx < num\_files; file\_idx++){  
  
 FILE\* fp = fopen(files[file\_idx], "r");  
 **if** (fp == NULL){  
 printf("cannot open %s!\n", files[file\_idx]);  
 }  
  
 **char** record;  
 // record letter of current line  
 **char** name[6+1];  
 // section name  
 **unsigned int** start\_addr, cs\_length;  
 // start address, control section length  
 **char** line[100];  
 // one line  
 line[0] = '0';  
 **while** (fgets(line, 100, fp) != NULL){  
 **char** defs[6][7];  
 **unsigned int** addr\_defs[6];  
 **int** num\_defs, num\_end\_args, end\_addr;  
 **int** len;  
  
 record = line[0];  
  
 **switch** (record){  
  
 **case** 'H':  
 // in H record, set name of program, cs\_length, insert name to estab  
 strncpy(name, line+1, 6);  
 name[6] = '\0';  
 start\_addr = hex\_from\_substring(line, 1+6, 6);  
 cs\_length = hex\_from\_substring(line, 1+6+6, 6);  
 estab.emplace\_back(&estab, name, progaddr+csaddr, 1, cs\_length);  
 **break**;  
 **case** 'D':  
 // in D record, insert symbol to estab  
 len = strlen(line);  
 num\_defs = (strlen(line)-1-1+11)/12;  
  
 **for** (**int** i = 0; i < num\_defs; ++i) {  
 strncpy(defs[i], line+1+12\*i, 6);  
 defs[i][6] = '\0';  
 addr\_defs[i] = hex\_from\_substring(line, 1+12\*i+6, 6);  
 estab.emplace\_back(&estab, defs[i], progaddr+csaddr+addr\_defs[i], 0, cs\_length);  
 }  
 **break**;  
 **case** 'R':  
 **break**;  
 **case** 'T':  
 **break**;  
 **case** 'M':  
 **break**;  
 **case** 'E':  
 // in E record, set execute addr and increase csaddr  
 **if** (strlen(line) >= 1+6+1){  
 execute\_addr = hex\_from\_substring(line, 1, 6);  
 }  
 csaddr += cs\_length;  
 **break**;  
 **case** '.':  
 **break**;  
 **default**:  
 printf("wrong odj file!\n");  
 **return** 1;  
 }  
 }  
 }  
 estab.print(&estab);  
 // print estab  
 total\_length = csaddr;  
 // total length of program is the last csaddr value  
 **return** 0;  
}  
  
**int** linker\_pass2(**int** num\_files, **char** \*\*files){  
 // linker\_pass2  
 **unsigned int** csaddr = 0;  
 **for** (**int** file\_idx = 0; file\_idx < num\_files; file\_idx++){  
  
 **struct** vector local\_estab = vector.new();  
 // for each control section, we have local estab for reference  
  
 FILE\* fp = fopen(files[file\_idx], "r");  
 **if** (fp == NULL){  
 printf("cannot open %s!\n", files[file\_idx]);  
 }  
  
 **char** record;  
 **char** name[6+1];  
 **unsigned int** start\_addr, length, cs\_length;  
 **char** line[100];  
 line[0] = '0';  
 **while** (fgets(line, 100, fp) != NULL){  
 **char** defs[6][7];  
 **unsigned int** addr\_defs[6];  
 **int** num\_defs, num\_end\_args, end\_addr;  
 **int** len;  
 **int** reference\_number, diff;  
 **int** to\_edit, sign;  
  
  
 record = line[0];  
  
 **switch** (record){  
  
 **case** 'H':  
 // in H record, set name of program, cs\_length, insert name to local estab  
 strncpy(name, line+1, 6);  
 name[6] = '\0';  
 start\_addr = hex\_from\_substring(line, 1+6, 6);  
 cs\_length = hex\_from\_substring(line, 1+6+6, 6);  
  
 local\_estab.emplace\_back(&local\_estab, name,  
 estab.find(&estab, name)->addr, 1, 0);  
 **break**;  
 **case** 'D':  
 **break**;  
 **case** 'R':  
 // push extref from estab to local estab  
 **for** (**int** i = 0; 1+8\*i+2 < strlen(line); ++i) {  
 strncpy(name, line+1+8\*i+2, 6);  
 name[6] = '\0';  
  
 local\_estab.emplace\_back(&local\_estab, name,  
 estab.find(&estab, name)->addr, 0, 0);  
 }  
   
 **break**;  
 **case** 'T':  
 // load text  
 start\_addr = hex\_from\_substring(line, 1, 6) + csaddr + progaddr;  
 length = hex\_from\_substring(line, 1+6, 2);  
 **for** (**int** i = 0; i < length; ++i) {  
 memory[start\_addr+i] = hex\_from\_substring(line+1+6+2, 2\*i, 2);  
 }  
 **break**;  
 **case** 'M':  
 // modify symbol form local estab  
 fflush(stdout);  
 start\_addr = hex\_from\_substring(line, 1, 6) + csaddr + progaddr;  
 length = 6;  
  
 reference\_number = atoi(line+10);  
 **if** (local\_estab.get(&local\_estab, reference\_number) != NULL)  
 diff = local\_estab.get(&local\_estab, reference\_number)->addr;  
 **else** printf("!!!!!!!!!!!!!!!!!!!");  
  
 to\_edit = ((**int**)memory[start\_addr])\*(1<<16)  
 + ((**int**)memory[start\_addr+1])\*(1<<8)  
 + memory[start\_addr+2];  
 **if**(line[9] == '+')  
 to\_edit += diff;  
 **else if** (line[9] == '-')  
 to\_edit -= diff;  
  
  
 memory[start\_addr+2] = (**unsigned char**)to\_edit;  
 to\_edit >>= 8;  
 memory[start\_addr+1] = (**unsigned char**)to\_edit;  
 to\_edit >>= 8;  
 memory[start\_addr] = (**unsigned char**)to\_edit;  
  
 **break**;  
 **case** 'E':  
 //local\_estab.print(&local\_estab);  
  
 **if** (strlen(line) >= 1+6+1){  
 execute\_addr = hex\_from\_substring(line, 1, 6);  
 }  
 csaddr += cs\_length;  
 **break**;  
 **case** '.':  
 **break**;  
 **default**:  
 printf("wrong odj file!\n");  
 **return** 1;  
 }  
 }  
 }  
 **return** 0;  
}

## run.h

#ifndef PROJECT\_RUN\_H  
#define PROJECT\_RUN\_H  
**extern** int32\_t A, X, L, PC, B, S, T, SW;  
  
**int** run();  
#endif //PROJECT\_RUN\_H

## run.c

#include <stdint.h>  
#include "run.h"  
#include "memory.h"  
#include "linking\_loader.h"  
#include "20171667.h"  
#include "opcode.h"  
#include "assembler.h"  
#include "breakpoints.h"  
  
#define r1 \*addr\_to\_reg(extract\_uint8(inst.b1, 0, 4))  
#define r2 \*addr\_to\_reg(extract\_uint8(inst.b1, 4, 8))  
#define WORD0 (word){.all = 0}  
//#define DEBUG  
  
**const int** POS\_N = 6;  
**const int** POS\_I = 7;  
**const int** POS\_X = 0;  
**const int** POS\_B = 1;  
**const int** POS\_P = 2;  
**const int** POS\_E = 3;  
// 각 flag의 위치  
  
**typedef struct** word{  
 **union**{  
 **struct**{  
 uint8\_t b3;  
 uint8\_t b2;  
 uint8\_t b1;  
 uint8\_t b0;  
 };  
 uint32\_t all;  
 };  
}word;  
// big endian 형식인 4바이트 정수  
  
**struct** Float{  
 u\_int64\_t frac : 36;  
 u\_int64\_t exp : 11;  
 u\_int64\_t sign : 1;  
};  
// float를 저장하는 구조체  
  
**static** uint8\_t set\_uint8(uint8\_t src, **int** start, **int** end){  
  
 **for**(**int** i = start; i < end; i++){  
 src |= (0b10000000 >> i);  
 }  
 **return** src;  
}  
// uint8의 [start, end)를 1로 set  
  
**static** word set\_word(word src, **int** start, **int** end){  
  
 **for**(**int** i = start; i < end; i++){  
 **if**(i < 8){  
 src.b0 = set\_uint8(src.b0, i, i+1);  
 }  
 **else if** (i < 16){  
 src.b1 = set\_uint8(src.b1, i - 8, i+1 - 8);  
 }  
 **else if** (i < 24){  
 src.b2 = set\_uint8(src.b2, i - 16, i+1 - 16);  
 }  
 **else if** (i < 32){  
 src.b3 = set\_uint8(src.b3, i - 24, i+1 - 24);  
 }  
 }  
 **return** src;  
}  
// word의 [start, end)를 1로 set  
  
**static int** check\_uint8(uint8\_t src, **int** pos){  
 **return** (src & (0b10000000 >> pos)) != 0;  
}  
// uint8의 pos가 1인지 0인지 return  
  
**static int** check\_word(word src, **int** pos){  
 **if**(pos < 8){  
 **return** check\_uint8(src.b0, pos);  
 }  
 **else if** (pos < 16){  
 **return** check\_uint8(src.b1, pos - 8);  
 }  
 **else if** (pos < 24){  
 **return** check\_uint8(src.b2, pos - 16);  
 }  
 **else if** (pos < 32){  
 **return** check\_uint8(src.b3, pos - 24);  
 }  
}  
// word의 pos가 1인지 0인지 return  
  
**static** uint8\_t extract\_uint8(uint8\_t src, **int** start, **int** end){  
 src &= set\_uint8(0, start, end);  
 src >>= 8-end;  
 **return** src;  
}  
// uint8에서 [start, end)범위의 비트를 추출  
  
**static** word extract\_word(word src, **int** start, **int** end){  
  
 src.all &= set\_word( (word){.all= 0}, start, end).all;  
 src.all >>= 32 - end;  
 **return** src;  
}  
// word에서 [start, end)범위의 비트를 추출  
  
int32\_t A, X, L, PC, B, S, T, SW;  
// 레지스터 변수  
**int** from\_bp = 0;  
// breakpoint로 부터 실행인지, 첫 실행인지 저장  
  
int32\_t\* addr\_to\_reg(**int** addr){  
 // return pointer of register of given addr  
  
 **switch** (addr) {  
 **case** 0:  
 **return** &A;  
 **case** 1:  
 **return** &X;  
 **case** 2:  
 **return** &L;  
 **case** 3:  
 **return** &B;  
 **case** 4:  
 **return** &S;  
 **case** 5:  
 **return** &T;  
 **case** 8:  
 **return** &PC;  
 **case** 9:  
 **return** &SW;  
 **default**:  
 **return** NULL;  
 }  
}  
  
**static void** print\_reg(){  
 // print information of register  
 printf("A : %06X ", A);  
 printf("X : %06X \n", X);  
 printf("L : %06X ", L);  
 printf("PC: %06X \n", PC);  
 printf("B : %06X ", B);  
 printf("S : %06X \n", S);  
 printf("T : %06X \n", T);  
}  
  
**static** uint32\_t deref(uint32\_t src){  
 // derefernce src and fetch 3 byte  
 word w = {.all = src};  
 w.b0 = 0;  
  
 **return** (uint32\_t)memory[w.all] \* (1 << 16)  
 + (uint32\_t)memory[w.all+1] \* (1 << 8)  
 + (uint32\_t)memory[w.all+2];  
}  
  
**static void** store(uint32\_t TA, uint32\_t val){  
 // store val into TA(target addr)  
 word w = {.all = val};  
 memory[TA] = w.b1;  
 memory[TA+1] = w.b2;  
 memory[TA+2] = w.b3;  
}  
  
  
**int** run(){  
 **const int** is\_input\_test\_device\_ready = 1;  
 // 디바이스가 준비되었는지 저장  
  
 uint8\_t device\_input[] = {'I', 'N', '\0', '\0', '\0', '\0', '\0', '\0', '\0'};  
 **int** cnt\_device\_input = 0;  
 // 디바이스의 입력을 저장  
  
 **if**(from\_bp == 0) {  
 // if this is first run() call  
 PC = progaddr + execute\_addr;  
 L = progaddr + total\_length;  
 }  
  
 **while** (1){  
 **int** format;  
 // 현재 inst의 foramt을 저장  
  
 word inst = {.b0 = memory[PC], .b1 = memory[PC+1], .b2 = memory[PC+2], .b3 = memory[PC+3]};  
 // fetch (m .. m3) to parse  
 word disp12 = {.all = extract\_word(inst, 12, 24).all};  
 word disp20 = {.all = extract\_word(inst, 12, 32).all};  
 uint8\_t opcode = extract\_uint8(inst.b0, 0, 6) << 2;  
 format = get\_format\_from\_opcode(opcode);  
 // get format info from opcode.c  
  
 PC += format;  
 **if**(format == 3 && check\_uint8(inst.b1, POS\_E)){  
 PC += 1;  
 }  
  
#ifdef DEBUG  
 printf("-----------------------------\n");  
 printf("pc : %04X\n", PC);  
 printf("inst : %X\n", inst.all);  
 printf("disp12 : %03X\n", disp12.all);  
 printf("disp20 : %05X\n", disp20.all);  
 printf("n : %d\ti : %d\tx : %d\tb : %d\tp : %d\te : %d\t\n"  
 , check\_uint8(inst.b0, POS\_N)  
 , check\_uint8(inst.b0, POS\_I)  
 , check\_uint8(inst.b1, POS\_X)  
 , check\_uint8(inst.b1, POS\_B)  
 , check\_uint8(inst.b1, POS\_P)  
 , check\_uint8(inst.b1, POS\_E)  
 );  
 fflush(stdout);  
#endif  
  
 **int** n;  
 **int** bit;  
 uint32\_t TA;  
 // target address  
 uint32\_t TV;  
 // target value  
  
 **switch** (format) {  
 **case** 1:  
 **switch** (opcode) {  
 **case** 0xC4:  
 // float  
 **case** 0xC0:  
 // float  
 **case** 0xF4:  
 // Halt I/O channel number (A)  
 **case** 0xC8:  
 // float  
 **case** 0xF0:  
 // Start I/O channel number (A); P X  
 // address of channel program  
 // is given by (S)  
 **case** 0xF8:  
 // Test I/O channel number (A)  
 **break**;  
 **default**:  
 **break**;  
 }  
 **break**;  
 **case** 2:  
 **switch** (opcode) {  
 **case** 0x90:  
 //ADDR r1,r2 2 90 r2 <-- (r2) + (r1)  
 r2 = r2 + r1;  
 **break**;  
 **case** 0xB4:  
 // CLEAR r1 2 B4 r1 <-- 0  
 r1 = 0;  
 **break**;  
 **case** 0xA0:  
 // COMPR r1,r2 2 A0 (r1) : (r2)  
 SW = r1 - r2;  
 **break**;  
 **case** 0x9C:  
 // DIVR r1,r2 2 9C (r2) <-- (r2) / (r1)  
 r2 = r2 / r1;  
 **break**;  
 **case** 0x98:  
 // MULR r1,r2 2 98 r2 <-- (r2) \* (r1)  
 r2 = r2 \* r1;  
 **break**;  
 **case** 0xAC:  
 // RMO r1,r2 2 AC r2 <-- (r1)  
 r2 = r1;  
 **break**;  
 **case** 0xA4:  
 // SHIFTL r1,n 2 A4 r1 <-- (r1); left circular X  
 // shift n bits. {In assembled  
 // instruction, r2=n-1}  
 bit = ((1 << 23) & r1) != 0;  
 n = r2 + 1;  
 r1 <<= n;  
 r1 |= bit;  
 **break**;  
 **case** 0x94:  
 // SUBR r1,r2 2 94 r2 <-- (r2) - (r1)  
 r2 = r2 - r1;  
 **break**;  
 **case** 0xB0:  
 // SVC n 2 B0 Generate SVC interrupt. {In X  
 // assembled instruction, r1=n}  
 //  
 **break**;  
 **case** 0xB8:  
 // X <-- (X) + 1; (X) : (r1)  
 X++;  
 SW = X - r1;  
 **break**;  
 **default**:  
 **break**;  
 }  
 **break**;  
 **case** 3:  
 **if** (check\_uint8(inst.b1, POS\_E)) {  
 TA = extract\_word(inst, 12, 32).all;  
 format = 4;  
 } **else if** (!check\_uint8(inst.b0, POS\_N) && !check\_uint8(inst.b0, POS\_I)) {  
 // SIC opcode  
 TA = extract\_word(inst, 8 + 1, 24).all;  
 } **else if** (check\_uint8(inst.b1, POS\_B) && !check\_uint8(inst.b1, POS\_P)) {  
 // base relative  
 TA = B + disp12.all;  
 } **else if** (!check\_uint8(inst.b1, POS\_B) && check\_uint8(inst.b1, POS\_P)) {  
 // pc relative  
 **if**(check\_word(disp12, 20)){  
 disp12.all -= 1;  
 disp12.all ^= set\_word(WORD0, 20, 32).all;  
 TA = PC - disp12.all;  
 }  
 **else**{  
 TA = PC + disp12.all;  
 }  
 } **else if** (opcode == 0x4C) {  
 // RSUB  
 // pass!  
 }  
 **else if** (!check\_uint8(inst.b0, POS\_N)  
 && check\_uint8(inst.b0, POS\_I)  
 && !check\_uint8(inst.b1, POS\_X)  
 && !check\_uint8(inst.b1, POS\_B)  
 && !check\_uint8(inst.b1, POS\_P)  
 && !check\_uint8(inst.b1, POS\_E)  
 ){  
 // imm constant  
 // pass!  
 TA = extract\_word(inst, 12, 24).all;  
 }  
 **else** {  
 printf("not supported addressing!\n");  
 }  
  
 **if** (check\_uint8(inst.b1, POS\_X)) {  
 TA += X;  
 }  
  
#ifdef DEBUG  
 printf("target addr is %x\n", TA);  
 fflush(stdout);  
#endif  
  
 **switch** (extract\_uint8(inst.b0, 6, 8)) {  
 **case** 0b00000001:  
 // imm addressing  
 TV = TA;  
 **break**;  
 **case** 0b00000000:  
 **case** 0b00000011:  
 // simple addressing  
 TV = deref(TA);  
 **break**;  
 **case** 0b00000010:  
 // indirect addressing  
 TA = deref(TA);  
 TV = deref(TA);  
 **break**;  
 **default**:  
 // no case  
 **break**;  
 }  
  
#ifdef DEBUG  
 printf("Target value is %x\n", TV);  
 fflush(stdout);  
#endif  
  
 **switch** (extract\_uint8(inst.b0, 0, 6) << 2) {  
 **case** 0x18:  
 // ADD m 3/4 18 A <-- (A) + (m..m+2)  
 A = A + TV;  
 **break**;  
 **case** 0x58:  
 // ADDF m 3/4 58 F <-- (F) + (m..m+5)  
 // *TODO* **break**;  
 **case** 0x40:  
 // AND m 3/4 40 A <-- (A) & (m..m+2  
 A = A & TV;  
 **break**;  
 **case** 0x28:  
 // COMP m 3/4 28 A : (m..m+2)  
 SW = A - TV;  
 **break**;  
 **case** 0x88:  
 // COMPF m 3/4 88 F : (m..m+5)  
 // *TODO* **break**;  
 **case** 0x24:  
 //DIV m 3/4 24 A : (A) / (m..m+2)  
 A = A / TV;  
 **break**;  
 **case** 0x64:  
 // DIVF m 3/4 64 F : (F) / (m..m+5)  
 // *TODO* **break**;  
 **case** 0x3C:  
 //J m 3/4 3C PC <-- m  
 PC = TA;  
 **break**;  
 **case** 0x30:  
 //JEQ m 3/4 30 PC <-- m if CC set to =  
 **if** (SW == 0) {  
 PC = TA;  
 }  
 **break**;  
 **case** 0x34:  
 //JGT m 3/4 34 PC <-- m if CC set to >  
 **if** (SW > 0) {  
 PC = TA;  
 }  
 **break**;  
 **case** 0x38:  
 //JLT m 3/4 38 PC <-- m if CC set to <  
 **if** (SW < 0) {  
 PC = TA;  
 }  
 **break**;  
 **case** 0x48:  
 //JSUB m 3/4 48 L <-- (PC); PC <-- m  
 L = PC;  
 PC = TA;  
 **break**;  
 **case** 0x00:  
 //LDA m 3/4 00 A <-- (m..m+2)  
 A = TV;  
 **break**;  
 **case** 0x68:  
 //LDB m 3/4 68 B <-- (m..m+2) X  
 B = TV;  
 **break**;  
 **case** 0x50:  
 //LDCH m 3/4 50 A [rightmost byte] <-- (m)  
 ((word\*)(&A))->b3 = ((word\*)(&TV))->b1;  
 **break**;  
 **case** 0x70:  
 //LDF m 3/4 70 F <-- (m..m+5) X F  
 // *TODO* **break**;  
 **case** 0x08:  
 //LDL m 3/4 08 L <-- (m..m+2)  
 L = TV;  
 **break**;  
 **case** 0x6C:  
 //LDS m 3/4 6C S <-- (m..m+2) X  
 S = TV;  
 **break**;  
 **case** 0x74:  
 //LDT m 3/4 74 T <-- (m..m+2) X  
 T = TV;  
 **break**;  
 **case** 0x04:  
 //LDX m 3/4 04 X <-- (m..m+2)  
 X = TV;  
 **break**;  
 **case** 0xD0:  
 //LPS m 3/4 D0 Load processor status from  
 // information beginning at  
 // address m (see Section  
 // 6.2.1)  
 // *TODO* **break**;  
 **case** 0x20:  
 //MUL m 3/4 20 A <-- (A) \* (m..m+2)  
 A = A \* TV;  
 ((word\*)A)->b0 = 0;  
 **break**;  
 **case** 0x60:  
 //MULF m 3/4 60 F <-- (F) \* (m..m+5) X F  
 // *TODO* **break**;  
 **case** 0x44:  
 //OR m 3/4 44 A <-- (A) | (m..m+2)  
 A = A | TV;  
 **break**;  
 **case** 0xD8:  
 //RD m 3/4 D8 A [rightmost byte] <-- data P  
 // from device specified by (m)  
 printf("[RD] : %c\n", device\_input[cnt\_device\_input]);  
 A &= set\_word(WORD0, 0, 24).all;  
 A |= device\_input[cnt\_device\_input++];  
 **break**;  
 **case** 0x4C:  
 //RSUB 3/4 4C PC <-- (L)  
 PC = L;  
 **break**;  
 **case** 0xEC:  
 //SSK m 3/4 EC Protection key for address m P X  
 // <-- (A) (see Section 6.2.4)  
 // *TODO* **break**;  
 **case** 0x0C:  
 //STA m 3/4 0C m..m+2 <-- (A)  
 store(TA, A);  
 **break**;  
 **case** 0x78:  
 //STB m 3/4 78 m..m+2 <-- (B) X  
 store(TA, B);  
 **break**;  
 **case** 0x54:  
 //STCH m 3/4 54 m <-- (A) [rightmost byte]  
 memory[TA]  
 = (uint8\_t)A;  
 **break**;  
 **case** 0x80:  
 //STF m 3/4 80 m..m+5 <-- (F) X  
 // *TODO* **break**;  
 **case** 0xD4:  
 //STI m 3/4 D4 Interval timer value <-- P X  
 // (m..m+2) (see Section  
 // 6.2.1)  
 // we dont support it  
 **break**;  
 **case** 0x14:  
 //STL m 3/4 14 m..m+2 <-- (L)  
 store(TA, L);  
 **break**;  
 **case** 0x7C:  
 //STS m 3/4 7C m..m+2 <-- (S) X  
 store(TA, S);  
 **break**;  
 **case** 0xE8:  
 //STSW m 3/4 E8 m..m+2 <-- (SW) P  
 store(TA, SW);  
 **break**;  
 **case** 0x84:  
 //STT m 3/4 84 m..m+2 <-- (T) X  
 store(TA, T);  
 **break**;  
 **case** 0x10:  
 //STX m 3/4 10 m..m+2 <-- (X)  
 store(TA, X);  
 **break**;  
 **case** 0x1C:  
 //SUB m 3/4 1C A <-- (A) - (m..m+2)  
 A = A - TV;  
 **break**;  
 **case** 0x5C:  
 //SUBF m 3/4 5C F <-- (F) - (m..m+5) X F  
 // *TODO* **break**;  
 **case** 0xE0:  
 //TD m 3/4 E0 Test device specified by (m)  
 // IO  
 **if**(is\_input\_test\_device\_ready){  
 SW = 0 - 1;  
 printf("[TD] : success\n");  
 }  
 **else**{  
 SW = 0;  
 printf("[TD] : failed\n");  
 }  
 **break**;  
 **case** 0x2C:  
 //TIX m 3/4 2C X <-- (X) + 1; (X) : (m..m+2) C  
 X = X + 1;  
 SW = X - TV;  
 **break**;  
 **case** 0xDC:  
 //WD m 3/4 DC Device specified by (m) <-- (A)[RMB] P  
 // IO  
 printf("[WD] : %c\n", (uint8\_t)A);  
 **break**;  
 **default**:  
 printf("not supported opcode!\n");  
 **break**;  
  
 }  
 **break**;  
 **default**:  
 **break**;  
 }  
#ifdef DEBUG  
 print\_reg();  
 printf("after TA : %04X\n", TA);  
 printf("deref TA : %06X\n", deref(TA));  
#endif  
  
 **if**(PC < progaddr || progaddr + total\_length <= PC){  
 // out of program  
 print\_reg();  
 printf("End Program\n");  
 from\_bp = 0;  
 **return** 0;  
 }  
 **else if**(exist\_bp(PC)){  
 // break point hit  
 print\_reg();  
 printf("Stop at checkpoint[%X]\n", PC);  
 from\_bp = 1;  
 **return** 0;  
 }  
 }  
}

## vector.h

#include <stdio.h>  
#include <string.h>  
  
#include "vector.h"  
  
**static int** emplace\_back (**struct** vector\* this, **char** name[7], **unsigned int** addr, **int** is\_control\_section, **int** length){  
  
 this->size++;  
 this->data = realloc(this->data, this->size \* **sizeof**(external\_symbol));  
  
 **for** (**int** i = 0; i < 7; ++i) {  
 **if**(name[i] == ' ' || name[i] == '\n')  
 name[i] = '\0';  
 }  
 strncpy(this->data[this->size - 1].name, name, 7);  
 this->data[this->size - 1].addr = addr;  
 this->data[this->size - 1].is\_control\_section = is\_control\_section;  
 this->data[this->size - 1].length = length;  
 **return** 0;  
}  
// emplace\_back method for vector  
  
**static** external\_symbol\* get (**struct** vector\* this, **int** idx){  
  
 idx--;  
 **if**(idx >= this->size || idx < 0)  
 **return** NULL;  
  
 **return** &(this->data[idx]);  
}  
// get method for vector  
  
**static** external\_symbol\* find (**struct** vector\* this, **char** name[7]){  
  
 **for** (**int** i = 0; i < 7; ++i) {  
 **if**(name[i] == ' ' || name[i] == '\n')  
 name[i] = '\0';  
 }  
  
 **for** (**int** i = 0; i < this->size; ++i) {  
 **if**(!strcmp(this->data[i].name, name)){  
 **return** &(this->data[i]);  
 }  
 }  
 **return** NULL;  
}  
// find element with name  
  
**static void** print(**struct** vector\* this){  
 printf("control\t symbol\t address\t length\n");  
 printf("section\t name\n");  
  
 **for** (**int** i=0;i<40;i++)  
 printf("-");  
 printf("\n");  
  
  
 **unsigned int** total\_length = 0;  
  
 **for** (**int** i = 0; i < this->size; ++i) {  
 **if**(this->data[i].is\_control\_section){  
 printf("%6s\t \t %04X\t %04X\n", this->data[i].name, this->data[i].addr, this->data[i].length);  
 total\_length += this->data[i].length;  
 }  
 **else**{  
 printf(" \t %6s\t %04X\t \n", this->data[i].name, this->data[i].addr);  
 }  
 }  
  
 **for** (**int** i=0;i<40;i++)  
 printf("-");  
 printf("\n");  
 printf("total length : %X\n", total\_length);  
}  
// print all extab  
  
**static struct** vector new(){  
  
 **return** (**struct** vector){  
 .size = 0,  
 .data = malloc(0),  
 .emplace\_back = &emplace\_back,  
 .get = &get,  
 .print = &print,  
 .find = &find  
 };  
}  
// construct vector object  
  
**const struct** vector\_class vector = {.new = &new};

## vector.c

#ifndef PROJECT\_HASHTABLE\_H  
#define PROJECT\_HASHTABLE\_H  
  
#include <string.h>  
#include <stdlib.h>  
  
#define NOT\_FOUND NULL  
  
**typedef struct** node\_external\_symbol{  
 **char** name[7];  
 **unsigned int** addr;  
 **int** is\_control\_section;  
 **int** length;  
}external\_symbol;  
// node for external symbol  
  
  
**struct** vector {  
  
 **int** size;  
 external\_symbol\* data;  
  
 **int** (\*emplace\_back)(**struct** vector\* this, **char** name[7], **unsigned int** addr, **int** is\_control\_section, **int** length);  
 external\_symbol\* (\*get) (**struct** vector\* this, **int** idx);  
 external\_symbol\* (\*find) (**struct** vector\* this, **char** name[7]);  
 **void** (\*print)(**struct** vector\* this);  
};  
// vector object  
  
**extern const struct** vector\_class{  
 **struct** vector (\*new)(**void**);  
} vector;  
// to make vector object  
  
#endif //PROJECT\_HASHTABLE\_H