**과목명: 시스템프로그래밍**

**CSE4110-02**

**<<Project #3>>**

**서강대학교 [공학부 컴퓨터공학과]**

**[20171662]**

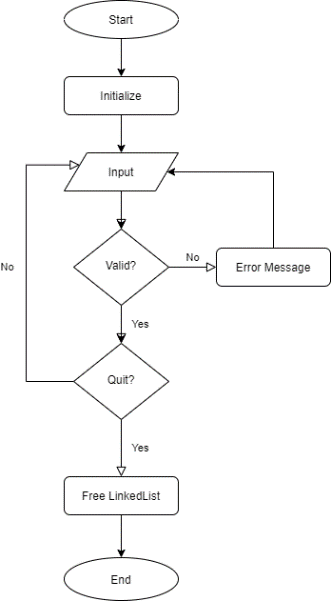
**[이나연]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
   2. linking/loader pass 1 & pass 2 흐름도
   3. run 구현 프로그램 흐름도
3. **모듈 정의**
   1. loader()
   2. linkload\_p1()
   3. linkload\_p2()
   4. insert\_essym()
   5. search\_extsym()
   6. search\_symname()
   7. search\_secname()
   8. show\_loadmap()
   9. show\_bp()
   10. save\_bp()
   11. clear\_bp()
   12. run()
   13. run\_format34()
   14. store\_mem()
   15. print\_reg()
   16. search\_bp()
4. **전역 변수 정의**
   1. int progaddr
   2. int CSADDR
   3. int CSLTH
   4. int EXECADDR
   5. ES\_TAB\* ESTAB
   6. bp\* bp\_head
5. **코드 설명**
   1. 20171662.h
   2. 20171662.c
   3. linkload.c
6. 프로그램 개요

해당 프로그램은 프로젝트 1과 2에서 구현한 셸(shell)에 linking과 loader 기능을 추가하는 프로그램이다. 프로젝트 2에서 구현된 assemble 명령을 통해서 생성된 object 파일을 link시켜 메모리에 올리는 일을 수행한다. linking 및 loader기능은 ‘loader’ 명령어로 실행되고, ‘run’ 명령어로 linking 및 loading된 프로그램을 직접 실행하는데, ‘bp’ 명령어로 설정된 breakpoint마다 프로그램이 멈추며 debugging의 기능을 제공한다. 이 때, 멈출 때 마다 A, X, L, PC, B, S, T의 각 레지스터에 들어가있는 값을 출력한다.

1. 프로그램 설명
   1. 프로그램 흐름도

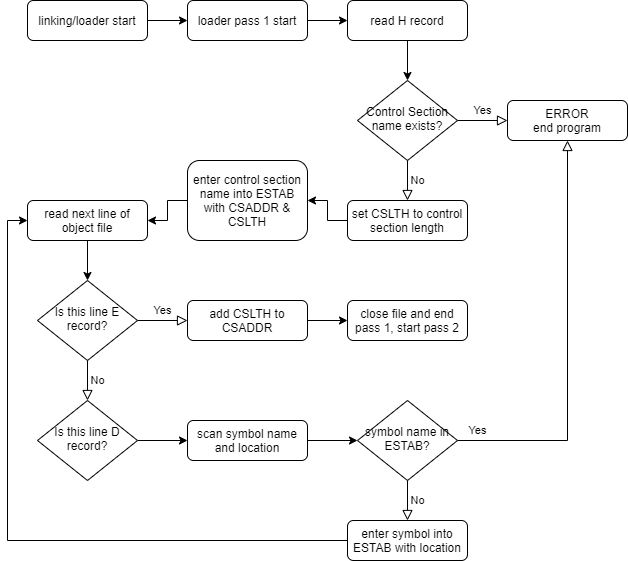


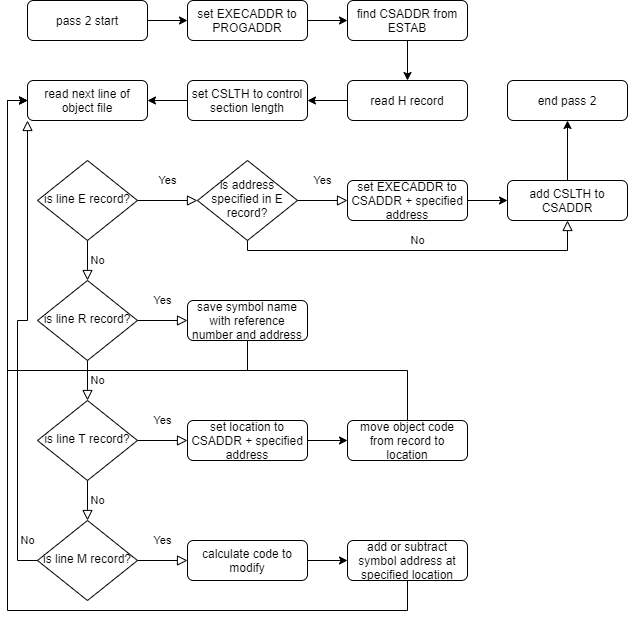
전체적인 프로그램의 흐름도는 다음과 같다. 먼저 프로그램이 실행되면, 추후 command들이 정상적으로 실행되기 위하여 필요한 여러 변수들에 메모리를 할당하여 주는 등의 Initialization을 실행한다. 그런 다음,

>>> sicsim> “input”

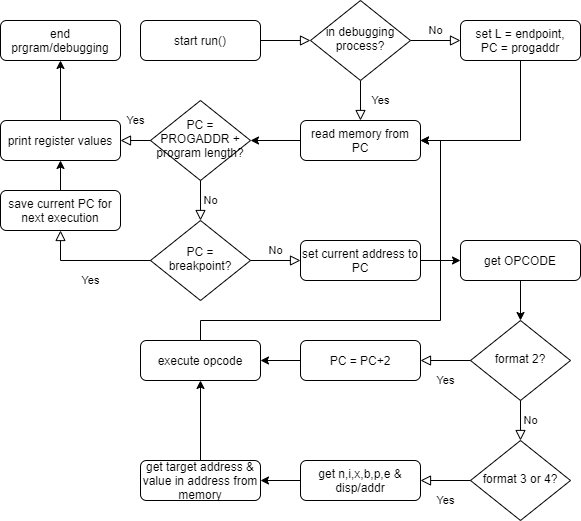
의 형태로 input을 받아 만약 해당 input이 valid하다면 해당 command의 기능을 수행하도록 하고, 만약 그렇지 않다면 적절한 에러 처리와 함께 다시 input을 새로 받는다. 만약 quit 이나 q에 해당하는 프로그램 종료에 관한 command가 들어오면, 할당된 메모리를 해제한 후 프로그램을 정상적으로 종료한다.

* 1. linking loader pass 1 & pass 2흐름도





* 1. run 구현 프로그램 흐름도



1. 모듈 정의
   1. loader(char\* objfile , int file\_num)
2. 기능

input으로 들어온 object file과 그 파일이 몇 번째 파일인지를 받아 파일이 오브젝트 파일이 맞는지를 확인한 후 linkload\_p1()함수를 수행한다.

1. 사용 변수

* char\* objfile: input으로 들어온 linking/loading 작업이 수행되어야 하는 오브젝트 파일명이다.
* int file\_num: 오브젝트 파일이 몇 번째로 들어왔는지를 나타내는데 사용된다.
* unsigned int i: for문에 loop인자로 사용된다. strlen()값과 비교 되어야 하기 때문에 unsigned int로 설정하였다.
  1. linkload\_p1(char\* objfile, int filenum)

1. 기능

linking loader의 pass 1 구현을 담당한다. 오브젝트 파일의 헤더 레코드에서 control section의 이름, 시작 주소, 길이를 읽어 들이고, D 레코드에서 섹션 내에 정의된 symbol들의 이름과 주소를 ESTAB에 저장한다.

1. 사용 변수

* char\* objfile: input으로 들어온 linking/loading 작업이 수행되어야 하는 오브젝트 파일명이다.
* int filenum: 오브젝트 파일이 몇 번째로 들어왔는지를 나타내는데 사용된다.
* char rec\_type: 현재 읽어들이는 줄의 record type을 저장한다.
* char sec\_name[7]: 현재 control section의 이름을 저장한다.
* int sec\_start\_addr: 현재 control section의 시작 주소를 저장한다.
* int sec\_length: 현재 control section의 길이를 저장한다.
* unsigned int d\_index: D 레코드를 읽어 들일 때 해당 줄에서 읽어 들이는 부분의 시작 인덱스를 저장한다.
* char d\_sym[7]: D 레코드에 적혀 있는 symbol의 이름을 저장한다.
* int d\_addr: D 레코드에 적혀 있는 symbol의 주소를 저장한다.
* char\* obj\_line: 오브젝트 파일의 한 줄을 저장한다.
* FILE\* fp = 오브젝트 파일의 파일 포인터를 저장한다.
  1. linkload\_p2(char\* objfile, int filenum, int total\_file\_num)

1. 기능

linker loader의 pass 2의 구현을 담당한다. pass 1 에서 생성된 ESTAB을 토대로 외부 섹션에서 정의된 심볼들의 주소 등을 토대로 메인 메모리에 object code들을 올린다. 만약 modification이 필요하다면 object file에 나와있는 대로 해당 주소에 들어있는 code에 외부 symbol의 주소를 더하거나 빼준다.

1. 사용 변수

* char\* objfile: input으로 들어온 linking/loading 작업이 수행되어야 하는 오브젝트 파일명이다.
* int filenum: 오브젝트 파일이 몇 번째로 들어왔는지를 나타내는데 사용된다.
* int total\_file\_num: 오브젝트 파일이 총 몇 개가 링킹 되었는지를 저장한다.
* char obj\_line[80]: 오브젝트 파일의 한 줄을 저장한다.
* char rec\_type: 현재 읽어들이는 줄의 record type을 저장한다.
* char sec\_name[7]: 현재 control section의 이름을 저장한다
* int sec\_start\_addr: 현재 control section의 시작 주소를 저장한다.
* int sec\_length: 현재 control section의 길이를 저장한다.
* int t\_len: T record의 한 줄의 길이를 저장한다.
* int t\_start\_addr: T record에 언급된 해당 줄의 메모리에서의 시작 주소를 저장한다.
* int t\_index: T record를 읽어 들일 때 읽어 들이고 있는 값의 시작 인덱스를 저장한다.
* int t\_addr: T record를 읽어 들일 때 시작 주소에 CSADDR의 값을 더해 해당 progaddr에 맞게끔 조정된 시작 주소를 저장한다.
* int r\_num: R record에서의 reference number를 저장한다.
* int r\_addr: R record에서 해당 symbol의 주소를 계산하여 저장한다.
* unsigned int r\_index: R record를 읽어 들일 때 읽어 들이고 있는 값의 시작 인덱스를 저장한다.
* char r\_sym[7]: R record에서의 symbol의 이름을 저장한다.
* int obj\_byte: 메인 메모리에서의 1 바이트를 읽어 들인다.
* int m\_addr: M record에서 modify되어야 하는 주소를 계산하여 저장한다.
* int m\_len: M record에서 modify되어야 하는 half byte수를 저장한다.
* int m\_ref: M record에서 modify되는 데에 사용되는 symbol의 reference number를 저장한다.
* int m\_code: M record에서 modify가 완료된 값을 저장한다.
* int m\_tmp: modify실행 중 object code를 계산할 때에 필요하지 않은 바이트 값들을 임시적으로 저장한다.
* int save\_value: format 4인 object code를 modify과정에서 업데이트할 때에, 누락될 수 있는 ‘xbpe’값들을 저장하는 half byte의 값을 저장한다.
* char m\_code\_txt[7]: modify과정에서 편의를 위해 object code를 문자열로 변환한 값을 저장한다.
* char operator: M record에서 modify를 할 때에 symbol의 주소를 더해야하는지 빼야하는지를 나타내는 +/-를 저장한다.
* ref ref\_list[6]: ref는 symbol의 이름과 그 주소를 담고 있는 구조체로, ref\_list는 R record에 나와 있는 외부 참조해야하는 symbol들의 이름과 주소를 저장한다.
* FILE\* fp: object file의 파일 포인터를 저장한다.
  1. insert\_essym(int filenum, char\* sym\_name, int sym\_addr)

1. 기능

ESTAB에 symbol의 이름과 주소를 집어넣는다. 연결 리스트의 형태로 이루어진 ESTAB의 symbol 저장소에 list의 제일 뒤에 새로운 노드를 집어넣는 형식을 취했다.

1. 사용 변수

* int filenum: 저장할 symbol이 정의되어 있는 파일이 몇 번째로 들어왔는지를 저장한다. 이를 토대로 ESTAB의 몇 번째 인덱스에 symbol을 저장할 지 결정된다.
* char\* sym\_name: 저장할 symbol의 이름이 저장되어있다.
* int sym\_addr: 저장할 symbol의 주소가 저장되어있다.
* ESTAB\_sym\* node: ESTAB\_sym은 ESTAB속 symbol을 저장하기 위하여 형성된 구조체이다. symbol의 이름과 주소를 담을 ESTAB\_sym의 포인터 변수를 선언하였다.
* ESTAB\_sym\* cur: ESTAB에서 symbol을 저장하는 linked list의 head->next 주소를 저장하는 포인터 변수이다. 리스트의 맨 끝 노드를 찾기 위하여 사용된다.
  1. search\_extsym(char\* sym\_name, int file\_num)

1. 기능

ESTAB에서 주어진 symbol이 이미 존재하는지를 찾는 함수이다. 존재하면 주소를, 존재하지 않으면 -1을 반환한다.

1. 사용 변수

* char\* sym\_name: 찾고자 하는 symbol의 이름이 저장되어있다.
* int file\_num: linking/loading이 되는 파일들의 수가 저장되어있다.
* int i: for문에 사용되는 loop인자이다.
* ESTAB\_sym\* cur: ESTAB에서 symbol을 저장하는 ESTAB\_sym\*형 변수의 head->next주소를 저장한 후 리스트의 각 노드를 가리키며 주어진 symbol의 이름과 노드 안에 들어있는 symbol의 이름이 같은지를 찾는 데에 쓰인다.
  1. search\_symname(int filenum, char\* sym\_name)

1. 기능

현재 pass 1을 진행중인 파일에서 현재 저장하려 하는 symbol과 이름이 같은 다른 symbol이 이미 ESTAB에 저장 되어있는지를 확인해준다. 만약 존재한다면 1, 탐색을 완료한 후에도 찾지 못했다면 0을 반환한다.

1. 사용 변수

* int filenum: 현재 pass1을 진행중인 파일의 번호 (순서)를 저장한다.
* char\* sym\_name: 찾고자 하는 symbol의 이름을 저장한다.
* ESTAB\_sym\* cur: 현재 control section에서 symbol을 저장하는 ESTAB\_sym\*형 변수의 head->next주소를 저장한 후 리스트의 각 노드를 가리키며 주어진 symbol의 이름과 노드 안에 들어있는 symbol의 이름이 같은지를 찾는 데에 쓰인다.
  1. search\_secname(int filenum, char\* sec\_name)

1. 기능

pass 1을 진행 완료한 파일들을 기준으로 현재 저장하려 하는 contol section과 이름이 같은 다른 control section이 이미 ESTAB에 저장 되어있는지를 확인해준다. 만약 존재한다면 1, 탐색을 완료한 후에도 찾지 못했다면 0을 반환한다.

1. 사용 변수

* int filenum: 현재 pass 1을 진행중인 파일의 번호 (순서)를 저장한다.
* char\* sec\_name: 찾고자 하는 control section의 이름을 저장한다.
* int i: for문에 필요한 loop 인자를 위해 선언되었다. 0부터 filenum-1까지 바뀌며 pass 1이 완료된 파일들에서의 control section name이 저장된 ESTAB의 인덱스로 쓰인다.
  1. show\_loadmap(int file\_num)

1. 기능

주어진 파일들에 대한 pass 1이 다 끝난 후 ESTAB에 저장된 section들의 symbol 정보를 load map으로 출력한다.

1. 사용 변수

* int file\_num: 주어진 파일들의 수를 저장한다.
* int i: for문에 필요한 loop 인자를 위해 선언되었다. 0부터 filenum-1까지 바뀌며 pass 1이 완료된 파일들에서의 control section의 symbol의 정보들이 저장된 ESTAB의 인덱스로 쓰인다
* int total\_len: 주어진 control section들의 length를 다 더한 값을 저장한다. 프로그램 전체의 길이를 나타낸다.
* ES\_TAB cur: 현재 정보를 열람하고 있는 control section의 이름, 길이, 정의된 symbol의 list 포인터를 담고 있는 구조체인 ES\_TAB 구조체의 변수로, 열람하고자 하는 control section의 구조체를 가리킨다.
* ESTAB\_sym\* cur\_sym: 현재 control section에서 symbol을 저장하는 ESTAB\_sym\*형 변수의 head->next주소를 저장한 후 리스트의 각 노드를 가리킨다. 노드 안의 symbol name과 address를 access하는 데에 사용된다.
  1. show\_bp()

1. 기능

사용자가 현 시점까지 등록한 break point를 출력 형식에 맞게 출력해주는 역할을 한다.

1. 사용 변수

* bp\* cur: breakpoint의 위치를 저장하고 있는 bp 구조체를 하나의 노드로 가지고 있는 bp\_head 전역 변수의 next를 가리키게 될 bp\* 변수이다. 각 노드를 가리키며 존재하는 모든 break point의 위치를 출력한다.
  1. save\_bp(int bp\_loc)

1. 기능

사용자가 입력한 위치로 break point를 설정할 때, 전역 변수 bp\_head에 새로운 break point를 리스트에 추가하는 역할을 한다.

1. 사용 변수

* int bp\_loc: 사용자가 break point의 위치로 지정한 값을 저장한다.
* bp\* cur: bp 구조체 포인터 변수로 bp\_head의 제일 마지막 노드를 찾는 데에 쓰인다.
* bp\* node: bp 구조체 포인터 변수로 bp\_head의 새로운 노드의 역할을 한다. 사용자가 입력한 위치로 지정한 값을 포함한다.
  1. clear\_bp()

1. 기능

사용자가 입력한 모든 breakpoint를 삭제하는 역할을 한다.

1. 사용 변수

* bp\* cur: bp 구조체 포인터 변수로 bp\_head의 삭제할 각 노드를 가리키는 데에 사용된다.
* bp\* next: bp 구조체 포인터 변수로 삭제할 노드의 다음 노드를 가리키는 데에 사용된다.
  1. run()

1. 기능

현재 progaddr로 설정된 주소에 저장되어있는 object code를 토대로 프로그램을 실행하고, 입력된 break point에 PC가 다다르면 디버깅의 기능으로 프로그램 실행을 중단하고, 다음 run 명령어가 들어오면 이전의 중단점부터 프로그램을 실행하여 더 이상 break point가 존재하지 않는다면 프로그램의 끝까지 실행한다. 프로그램이 중단될 때 마다, 그리고 프로그램이 끝나면 A, X, L, PC, B, S, T 레지스터 안에 들어있는 값을 출력한다.

1. 사용 변수

* static int cur\_addr: 프로그램의 시작 주소를 저장한다. 프로그램이 중단 되어 해당 함수가 종료된 이후 다시 함수가 실행되어도 그 이전의 cur\_addr에 접근이 가능해야 하기 때문에 static int로 선언하였다.
* int run\_len: 실행할 프로그램의 총 길이를 저장한다. 이는 ESTAB에 저장되어 있는 프로그램의 전체 길이를 이용하여 저장하였다.
* static int reg[10]: 각각의 레지스터들의 값을 저장한다. run 함수 종료 이후에도 계속 레지스터의 값은 남아 있어야 하기 때문에 static int 배열로 선언하였다. A=0, X=1, L=2, B-3, S=4, T=5, PC=8, SW=9의 레지스터 번호를 가지고 있기 때문에, 해당 레지스터 번호들을 인덱스로 하게 하였다.
* int n,i,x,b,p: object code상에서의 n,i,x,b,p의 값을 저장한다. e를 포함하지 않는 이유는 만약 format 4라면 e는 1, 아니라면 e는 0일 것이기 때문에 따로 값을 저장하지 않았다.
* int code\_len: object code의 크기를 byte단위로 저장한다. 만약 format 2이면 2, format 3이면 3, format 4이면 4의 값을 저장한다.
* int hbyte: object code의 첫 half byte를 입력 받아 format 2인지 format 3/4인지를 확인하는 용도로 쓰인다. 만약 9~B이면 format 2이고, 아니라면 format 3/4이기 때문이다. (format 1은 해당 예제에서 쓰이지 않으므로 고려하지 않았다.)
* int objcode: object code를 저장한다.
* int disp: format 3에서의 disp 값을 저장한다. 12 bit의 크기를 가진다.
* int addr: format 4에서의 addr 값을 저장한다. 20 bit의 크기를 가진다.
* int reg1, reg2: format 2에서의 레지스터 관련 명령어에서 object code상에 명시되어 있는 레지스터 번호를 저장한다.
* int target\_addr: disp또는 addr의 값을 기준으로 addressing mode를 고려하여 최종 target address를 계산하여 저장한다.
* int m\_value: 메모리 상에서 target address가 가리키는 실제 값을 저장한다.
* int opcode: object code에서 opcode를 추출하여 저장한다.
* static int last\_bp: 이전에 run이 중단된 break point의 주소 값을 저장한다. run 함수가 끝난 후에도, 다시 접근하였을 때에도 그 값이 유지되어야 하므로 static int형으로 선언하였다.
  1. run\_format34(int opcode, inti m\_value, int code\_len, int target\_addr, int\* reg, int x)

1. 기능

format 3/4에서 object code의 opcode의 기능을 실행한다.

1. 사용 변수

* int opcode: object code에서의 opcode를 저장한다.
* int code\_len: format 3인지 4인지를 나타낸다. object code의 길이를 byte단위로 저장한다.
* int target\_addr: disp또는 addr의 값을 기준으로 addressing mode를 고려하여 최종 target address를 계산하여 저장된 값을 가리킨다.
* int\* reg: 각각의 레지스터들의 값을 저장한다. run 함수에서 설정된 reg를 그대로 가져온다.
* int x: x값을 저장한다. X레지스터의 사용 여부를 결정한다. 만약 0이라면 사용하지 않고, 그렇지 않으면 사용한다.
  1. store\_mem(int value, int addr, int halfbyte)

1. 기능

format 3/4에서 object code의 opcode의 기능을 실행할 때에 store의 기능을 수행해야 할 때 사용된다. 주어진 주소 값에 주어진 value를 저장한다.

1. 사용 변수

* int value: 주어진 주소에 저장할 값을 저장한다.
* int addr: 값을 저장할 주소를 저장한다.
* int halfbyte: 저장할 값이 half byte단위로 몇 half byte인지를 저장한다.
  1. print\_reg(int\* reg)

1. 기능

프로그램이 중단되거나 종료될 때 A, X, L, PC, B, S, T 레지스터에 저장되어있는 값을 출력한다.

1. 사용 변수

* int\* reg: 각각의 레지스터들의 값을 저장한다. run 함수에서 설정된 reg를 그대로 가져온다.
  1. search\_bp(int bp\_loc)

1. 기능

현재 PC가 저장되어있는 중단점 중 하나인지를 찾는 역할을 한다.

1. 사용 변수

* int bp\_loc: 현재 PC의 값을 저장한다. 각 bp\_head의 노드에 들어있는 break point의 주소와 함께 비교하여 같은지 확인한다.

1. 전역변수 정의
   1. int progaddr

사용자가 progaddr [address]로 입력한 주소를 저장하는 정수형 변수이다. linking/loading에서의 pass 1 & 2에서도 프로그램 상의 상대적 주소를 사용자가 원하는 주소에 맞추어 변환할 때에도, run에서 프로그램의 시작 지점을 설정할 때에도 사용된다.

* 1. int CSADDR

현재 사용하고 있는 control section의 시작 주소를 저장하는 데에 사용된다.

* 1. int CSLTH

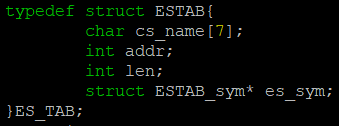
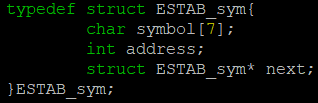
현재 사용하고 있는 control section의 길이를 저장하는 데에 사용된다. CSADDR에 더해져 그 다음 control section의 시작 주소를 계산하는 데에도 사용된다.

* 1. int EXECADDR

control section의 실행 시작 주소를 저장하는 데에 사용된다.

* 1. ES\_TAB\* ESTAB

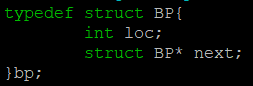
각 control section의 이름과 시작 주소, 그리고 그 안에 정의된 external symbol들의 이름과 주소를 저장하는 구조체 변수이다. control section의 개수만큼의 크기를 가지는 배열로 사용할 것이기 때문에 포이터 변수로 선언하였다. 아래는 해당 구조체의 형식이다.

왼쪽은 메인 구조체이다. 각 control section의 이름, 시작 주소, 그리고 해당 control section내에 정의된 symbol의 정보를 담고 있는 연결 리스트가 포함되어있다. 오른쪽은 해당 연결 리스트의 하나의 노드의 구성이다. symbol의 이름, 주소, 그리고 다음 노드를 가리킬 포인터 변수가 내장 되어있다.

* 1. bp\* bp\_head

사용자가 입력한 break point들을 저장하고 있는 연결 리스트의 head node를 가리킨다. 아래는 각 노드의 구성을 보여준다.



각 노드에는 break point의 주소와 다음 노드를 가리키는 포인터 변수가 내장되어있다.

1. 코드 설명
   1. 20171662.h

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <dirent.h>

#define MEM\_SIZE 1<<20//2^20, 1megabyte memory

#define OPCODE "opcode.txt"

#define TABLE\_SIZE 20//hashTable size

//for du[mp]. to save the last address printed

int end\_index;

//for main input. gets the whole line

char\* input;

//to save tokenized input

char\* modified\_input[100];

//to check if input is hex.

int isHex(char\*);

//to change the tokenized input into a single line with correct input form

char\* right\_input\_form(int,char\*\*);

//=========================================

//===========FOR SHELL COMMANDS============

//=========================================

//\*\*\*when d[ir]\*\*\*

//-----------------------------------------

void printdir();

//-----------------------------------------

//\*\*\*when hi[story]\*\*\*

//-----------------------------------------

//Node for saving input

typedef struct Node{

char\* data;

struct Node\* next;

}Node;

//head of Linked List for hi[story]

Node\* head;

//to add new node at the end of Linked List

void insertNode(Node\*, char\*);

//to print all nodes in the history list

void ShowAll(Node\*);

//to free all memory allocated for history Linked List when q[uit]

void FreeAll(Node\*);

//-----------------------------------------

//\*\*\*when type filename\*\*\*

//-----------------------------------------

int type(char\*);

//=========================================

//===========FOR MEMORY COMMANDS===========

//=========================================

//\*\*\*when du[mp] [start,end]\*\*\*

//-----------------------------------------

//memory with size 2^20

int my\_mem[MEM\_SIZE];

//for du[mp]

void onlydump();

//for du[mp] start

void dumpstart(int);

//for du[mp] start, end

void dumpstartend(int,int);

//for printing out memory values of the given address range

void print\_mem(int, int);

//-----------------------------------------

//\*\*\*when e[dit] address, value\*\*\*

//-----------------------------------------

//for e[dit] address, value. changes the value of the given memory address

void editvalue(int,int);

//-----------------------------------------

//\*\*\*when f[ill] start, end, value\*\*\*

//-----------------------------------------

//for f[ill] start, end, value. changes the memory value of given range to new value

void fillvalue(int, int, int);

//-----------------------------------------

//\*\*\*when reset\*\*\*

//-----------------------------------------

//for reset. changes all memory values to 0.

void reset\_mem();

//-----------------------------------------

//=========================================

//===========FOR OPCODE COMMANDS===========

//=========================================

//\*\*\*when opcode mnemonic\*\*\*

//-----------------------------------------

//structure for Opcode hashTable

typedef struct Opcode{

int opcodenum;//opcode

char instruction\_name[10];//instruction name

char format[5];//format number

struct Opcode\* next;//points to the next node (Linked List)

}Opcode;

//hashTable for OPCODE TABLE.

Opcode\*\* hashTable;

//for allocating memory for each hashTable[i]

void init();

//for freeing memory allocated for hashTable when q[uit]

void destructor();

//for searching for opcode of the given instruction

int search(int, char\*);

//for searching for opcode format number

int searchformat(int, char\*);

//for adding node to the hashTable[key]

void add(Opcode\*, int);

//for opcodelist. prints out all contents of the hashTable

void show();

//-----------------------------------------

//=========================================

//========FOR ASSEMBLER COMMANDS===========

//=========================================

//\*\*\*when assemble filename\*\*\*

//-----------------------------------------

//SYMTAB - Linked List

typedef struct SYMTAB{

char symbol[10];

int loc;

struct SYMTAB\* next;

}SYMTAB;

SYMTAB\* symhead;

typedef struct ObjectCode{

char objectcode[30];

int locctr;

int mod\_flag;

struct ObjectCode\* next;

}ObjectCode;

ObjectCode\* obhead;

//program length

int prog\_len;

//program starting address;

int start\_addr;

//base address

int base\_addr;

//insert LABEL, LOCCTR into SYMTAB

void insertSymbol(char\*, int);

//search for symbol

int SearchSymbol(char\*);

//main assembler

int assemble(char\*);

//for Pass 1 of Assembler

int pass\_1(char\*);

//for Pass 2 of Assembler

int pass\_2(char\*);

//for object code

char\* findObj(char\*,char\*,int);

//for inserting node at ObjectCode Linked List

void insertObj(char\*, int, int);

//for Freeing Object Code Linked List

void FreeObj();

//\*\*\*when symbol\*\*\*

//----------------------------------------

//SYMTAB print

void ShowSymbol();

//Free SYMTAB

void FreeSymbol();

//========================================

//=====FOR LINKING/LOADER COMMANDS========

//========================================

//for setting starting address for loader or 'run' command

int progaddr;

//for CSADDR (Control Section Address)

int CSADDR;

//for CSLTH (Control Section Length)

int CSLTH;

//for EXECADDR (execution address)

int EXECADDR;

//for ESTAB (External Symbol Table)

//Stores the name and addresses of each external symbol in the set of control sections

//Indicates in which control section the symbol is defined.

//ESTAB is for different sections

//ESTAB\_sym is for different symbols in one section

typedef struct ESTAB\_sym{

char symbol[7];

int address;

struct ESTAB\_sym\* next;

}ESTAB\_sym;

typedef struct ESTAB{

char cs\_name[7];

int addr;

int len;

struct ESTAB\_sym\* es\_sym;

}ES\_TAB;

ES\_TAB\* ESTAB;

//for saving breakpoints

typedef struct BP{

int loc;

struct BP\* next;

}bp;

bp\* bp\_head;

//flag for debugging

int in\_debug;

//for loader [object filename1] [object filename2] [...]

int loader(char\*,int);

//for linker loader pass 1

int linkload\_p1(char\*,int);

//for linker loader pass 2

int linkload\_p2(char\*,int,int);

//for searching for existing sections

int search\_secname(int,char\*);

//for searching for existing symbols

int search\_symname(int, char\*);

//for inserting symbols to ESTAB

void insert\_essym(int, char\*, int);

//for printing load map

void show\_loadmap(int);

//for searching for symbol's address in ESTAB

int search\_extsym(char\*,int);

//for showing all breakpoints

void show\_bp();

//for saving breakpoint

void save\_bp(int);

//for clearing all breakpoints

void clear\_bp();

//for run

void run();

//for finding breakpoint

int search\_bp(int);

//for printing out register values

void print\_reg(int\*);

//for running format 3 & 4

void run\_form34(int, int, int, int, int\*, int);

//for storing operations STA, STX, STL, STCH

void store\_mem(int, int, int);

* 1. 20171662.c

#include "20171662.h"

//available commands, has to be printed when h[elp]

char helpprint[] = "h[elp]\nd[ir]\nq[uit]\nhi[story]\ndu[mp] [start, end]\ne[dit] address, value\nf[ill] start, end, value\nreset\nopcode mnemonic\nopcodelist\nassemble filename\ntype filename\nsymbol\n";

int main(){

int i;

//flag to check for too many arguments

int flag=0;

//size of command (no. of arguments, ex. dump 4, 2E: size=3)

int size = 1;

//for saving address from command dump

int startnum, endnum;

//for saving input itself for history (char\* input will be changed due to strtok)

char\* save\_input = (char\*)malloc(sizeof(char)\*100);

//for saving breakpoint

int bp\_loc;

//to start with 0 when dump is first called

end\_index = 0xFFFFF;

//initializing head node for history

head = (Node\*)malloc(sizeof(Node));

head->next = NULL;

//initializing hashTable for opcode

hashTable = (Opcode\*\*)malloc(sizeof(Opcode\*)\*TABLE\_SIZE);

init();

//initializing SYMTAB

symhead = (SYMTAB\*)malloc(sizeof(SYMTAB));

symhead->next = NULL;

//initializing Object Code Linked List

obhead = (ObjectCode\*)malloc(sizeof(ObjectCode));

obhead->next = NULL;

//setting all memory to 0

memset(my\_mem,0,MEM\_SIZE);

//set default progaddr as 0x00

progaddr = 0;

//initialize bp list

bp\_head = (bp\*)malloc(sizeof(bp));

bp\_head->next = NULL;

//set debug flag to 0

in\_debug=0;

//importing opcode.txt

FILE\* fp;

char opcode\_input[81];

char\* mod\_opinput[3];

fp = fopen(OPCODE,"r");

while(fgets(opcode\_input,80,fp)!=NULL){

//tokenize opcode

mod\_opinput[0]=strtok(opcode\_input," \n\t");

mod\_opinput[1]=strtok(NULL," \n\t");

mod\_opinput[2]=strtok(NULL," \n\t");

//generate new node & initialize

Opcode\* op\_mem = (Opcode\*)malloc(sizeof(Opcode));

op\_mem->opcodenum = strtol(mod\_opinput[0],NULL,16);

strcpy(op\_mem->instruction\_name,mod\_opinput[1]);

strcpy(op\_mem->format,mod\_opinput[2]);

//setting the index into mod 20

int index = (int)(op\_mem->instruction\_name[0])%20;

add(op\_mem,index);

}//end of file reading

fclose(fp);

while(1){

flag = 0;

//allocate memory for the input variable

input = (char\*)malloc(sizeof(char)\*100);

printf("sicsim> ");

fflush(stdin);

fgets(input,100,stdin);

//printf("%s",input);

if(strlen(input)>100) printf("ERROR: INPUT TOO LONG");

if(input[0]=='\n') continue;

//removing white space

size = 1;

for(i=0;;i++){

//returns the first token, adds '\0' at the end of the token at the string

if (i==0)modified\_input[i] = strtok(input," \n\t");

else modified\_input[i] = strtok(NULL," \n\t");

//if too many arguments

if (i>4) {

printf("ERROR: TOO MANY ARGUMENTS\n");

flag=1;

break;

}

//if end of input

if(!modified\_input[i]) break;

}

size = i;

if(flag) continue;

//for checking ','

if(size>=3 && strcmp("loader",modified\_input[0])!=0){

for(i=1;i<size-1;i++){

if(modified\_input[i][strlen(modified\_input[i])-1]!=','){

printf("ERROR: Invalid Input. See h[elp].\n");

flag=1;

break;

}

else modified\_input[i][strlen(modified\_input[i])-1]='\0';

}

if(flag) continue;

}//end of checking for ','

//saving the input in the right form

strcpy(save\_input,right\_input\_form(size,modified\_input));

if(size==1){

//if q[uit]

if(!strcmp("quit",modified\_input[0]) || !strcmp("q",modified\_input[0])) {

FreeAll(head);

destructor();

FreeSymbol();

FreeObj();

break;

}

//if h[elp]

else if(!strcmp("help",modified\_input[0]) || !strcmp("h",modified\_input[0])) {

printf("%s",helpprint);

insertNode(head, save\_input);

continue;

}

//if d[ir]

else if(!strcmp("dir",modified\_input[0]) || !strcmp("d",modified\_input[0])) {

printdir();

insertNode(head, save\_input);

continue;

}

//if hi[story]

else if(!strcmp("history",modified\_input[0]) || !strcmp("hi",modified\_input[0])){

insertNode(head, save\_input);

ShowAll(head);

continue;

}

}//end of size=1

//if du[mp] [start, end]

if(!strcmp("dump",modified\_input[0]) || !strcmp("du",modified\_input[0])){

switch(size){

case 1:

//boundary check

if(end\_index > 0xFFFFF) end\_index=0xFFFFF;

onlydump();

insertNode(head,save\_input);

break;

case 2:

//check if address is in hex form.

if(isHex(modified\_input[1])==-1){

printf("ERROR: address not in hex form.\n");

continue;

}

//change the starting address from char to hex

startnum = strtol(modified\_input[1],NULL,16);

//boundary check

if(startnum > 0xFFFFF || startnum < 0){

printf("ERROR: Start Address out of range.\n");

continue;

}

dumpstart(startnum);

insertNode(head,save\_input);

break;

case 3:

//check if address is in hex form.

if(isHex(modified\_input[1])==-1 || isHex(modified\_input[2])==-1){

printf("ERROR: address not in hex form.\n");

continue;

}

//change start&end address from char to hex

startnum = strtol(modified\_input[1],NULL,16);

endnum = strtol (modified\_input[2],NULL,16);

//boundary check

if(startnum > 0xFFFFF || endnum > 0xFFFFF || startnum < 0 || endnum < 0){

printf("ERROR: Address out of range.\n");

continue;

}

else if(startnum > endnum){

printf("ERROR: Range ERROR\n");

continue;

}

dumpstartend(startnum,endnum);

insertNode(head,save\_input);

break;

default:

printf("ERROR! TOO MANY ARGUMENTS\n");

continue;

//flag = 1;

continue;

}//end of switch

}//end of du[mp] [start, end]

//if e[dit] address, value

else if (!strcmp("edit",modified\_input[0]) || !strcmp("e",modified\_input[0])){

if (size == 3){

//check if address is in hex form.

if(isHex(modified\_input[1])==-1 || isHex(modified\_input[2])==-1){

printf("ERROR: Address or Value not in hex form.\n");

continue;

}

//change address and value from char to hex

int add = strtol(modified\_input[1],NULL,16);

int val = strtol(modified\_input[2],NULL,16);

//boundary check

if (add<0 || add>0xFFFFF) {

printf("ERROR: Address out of range.\n");

continue;

}

//check value (00~FF)

else if(val<0x00 || val>0xFF){

printf("ERROR: Value out of range.\n");

continue;

}

else {

editvalue(add,val);

insertNode(head,save\_input);

continue;

}

}

else{

printf("ERROR: Input has to be e[dit] address, value. See h[elp].\n");

continue;

}

}//end of e[dit] address, value

//if f[ill] start, end, value

else if(!strcmp("f",modified\_input[0]) || !strcmp("fill",modified\_input[0])){

if(size==4){

//check if address or value is in hex form.

if(isHex(modified\_input[1])==-1 || isHex(modified\_input[2])==-1 || isHex(modified\_input[3])==-1){

printf("ERROR: Address or Value not in hex form.\n");

continue;

}

//change start, end address and value from char to hex

int start = strtol(modified\_input[1],NULL,16);

int end = strtol(modified\_input[2],NULL,16);

int value = strtol(modified\_input[3],NULL,16);

//boundary check

if(start<0 || start>0xFFFFF || end<0 || end>0xFFFFF || start>end){

printf("ERROR: Address out of range.\n");

continue;

}

//value boundary check

else if(value<0x00 || value>0xFF){

printf("ERROR: Value out of range.\n");

continue;

}

else{

fillvalue(start,end,value);

insertNode(head,save\_input);

continue;

}

}//end of size==4

else{

printf("ERROR: 4 arguments needed. See h[elp].\n");

continue;

}

}//end of f[ill] start, end, value

//if reset

else if(!strcmp("reset",modified\_input[0]) && size==1){

reset\_mem();

insertNode(head,save\_input);

continue;

}

//if opcode mnemonic

else if(!strcmp("opcode",modified\_input[0]) && size == 2){

//setting key to mod 20 of the first alphabet of the opcode

int key = (int)(modified\_input[1][0])%20;

int result\_opcode = search(key,modified\_input[1]);

if(result\_opcode == -1){

printf("ERROR: CANNOT FIND OPCODE. See opcodelist.\n");

continue;

}

else {

printf("opcode is %X\n",search(key,modified\_input[1]));

insertNode(head, save\_input);

continue;

}

}//end of opcode mnemonic

//if opcodelist

else if(!strcmp("opcodelist",modified\_input[0]) && size == 1){

insertNode(head,save\_input);

show();

continue;

}//end of opcodelist

//if type filename

else if(!strcmp("type",modified\_input[0]) && size == 2){

if(type(modified\_input[1])) insertNode(head,save\_input);

continue;

}

//if assemble filename

else if(!strcmp("assemble",modified\_input[0]) && size == 2){

if(assemble(modified\_input[1])) insertNode(head,save\_input);

continue;

}

//if symbol

else if(!strcmp("symbol",modified\_input[0]) && size == 1){

ShowSymbol();

insertNode(head, save\_input);

continue;

}

//if progaddr [address]

else if(!strcmp("progaddr",modified\_input[0]) && size==2){

//check if given bp is hex

if(!isHex(modified\_input[1])){

printf("ERROR: breakpoint has to be in hex number.\n");

continue;

}

//update progaddr

progaddr = strtol(modified\_input[1],NULL,16);

insertNode(head,save\_input);

continue;

}

//if loader [object filename1] [object filename2] [...]

else if(!strcmp("loader",modified\_input[0])){

//get PROGADDR from operating system

//set CSADDR to PROGADDR for first control section

CSADDR = progaddr;

//allocate memory for ESTAB by the number of files

ESTAB = (ES\_TAB\*)malloc(sizeof(ES\_TAB)\*(size-1));

for(i=0;i<size-1;i++) {

ESTAB[i].es\_sym=(ESTAB\_sym\*)malloc(sizeof(ESTAB\_sym));

ESTAB[i].es\_sym->next=NULL;

}

//link & load pass 1 for each object file

for(i=1;i<size;i++) {

//if linking/loading fails, return 0

if(!loader(modified\_input[i],i-1)) break;

}

//if pass 1 fails, continue

if(i!=size) continue;

//if pass 1 succeeds, print load map

show\_loadmap(size-1);

for(i=1;i<size;i++){

//execute pass 2

if(!linkload\_p2(modified\_input[i],i-1,size-1)) break;

}

//if loop ended successfully, add to history

if(i==size) insertNode(head,save\_input);

continue;

}

//if bp

else if(!strcmp("bp",modified\_input[0])){

//if bp, show breakpoint list

if(size==1){

show\_bp();

insertNode(head,save\_input);

continue;

}

if(size==2){

//if bp clear

if(!strcmp("clear",modified\_input[1])) clear\_bp();

//if bp [address], add to bp list

else if(isHex(modified\_input[1])==1) {

bp\_loc = strtol(modified\_input[1],NULL,16);

save\_bp(bp\_loc);

}

else{

printf("ERROR: only bp [address] or bp clear allowed.\n");

continue;

}

insertNode(head,save\_input);

continue;

}

}

//if run

else if(!strcmp("run",modified\_input[0]) && size==1){

run();

insertNode(head,save\_input);

continue;

}

//if invalid input

else {

printf("ERROR! Not an available input. see h[elp].\n");

continue;

}

}

return 0;

}

int isHex(char\* hex\_input){

unsigned int i;

//flag to check for non-hex num.

int hex\_flag=1;

for(i=0;i<strlen(hex\_input);i++){

//if each character is 0~9 or a~f or A~F

if((hex\_input[i]>='0' && hex\_input[i]<='9') || (hex\_input[i]>='a' && hex\_input[i]<='f') || (hex\_input[i]>='A' && hex\_input[i]<='F')) hex\_flag=1;

//not a hex num

else{

hex\_flag=-1;

return -1;

}

}

return hex\_flag;

}

//to save the correct form for each command

char\* right\_input\_form(int size, char\*\* m\_input){

char\* ret\_input = (char\*)malloc(sizeof(char)\*100);

int i;

ret\_input[0]='\0';

for(i=0;i<size;i++){

strcat(ret\_input,m\_input[i]);

if(i==0 && size>1) strcat(ret\_input," ");

if(i>=1 && size>i+1 && !strcmp(m\_input[0],"loader")) strcat(ret\_input," ");

else if(i>=1 && size>i+1){

strcat(ret\_input,", ");

}

}

return ret\_input;

}

* 1. linkload.c

#include "20171662.h"

int loader(char\* objfile, int file\_num){

unsigned int i;

//check file type

for(i=0;i<strlen(objfile);i++){

if(objfile[i]=='.') break;

}

if(strcmp(objfile+i,".obj")){

printf("ERROR: File is not .obj file.\n");

return 0;

}

//first execute pass 1, and if it succeeds,

if(linkload\_p1(objfile,file\_num)) return 1;

//if pass 1 fails, return 0

else return 0;

}

int linkload\_p1(char\* objfile,int filenum){

//for saving current line's record type

char rec\_type;

//for saving current program's name

char sec\_name[7];

//for saving current program's starting address and length

int sec\_start\_addr, sec\_length;

//for keeping track of each D record's symbol

unsigned int d\_index=1;

//for symbol name in D record

char d\_sym[7];

//for symbol address in D record

int d\_addr;

//object file's maximum column number is 73 (at the class material Chapter 2)

char\* obj\_line = (char\*)malloc(sizeof(char)\*80);

//open object file

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

//if file doesn't exist, return 0

if(!fp){

printf("ERROR: File does not exist.\n");

return 0;

}

//get the header record

fgets(obj\_line,80,fp);

//check for header record

if(obj\_line[0]!='H'){

printf("ERROR: There is no header record in this object file.\n");

fclose(fp);

return 0;

}

//save record type, control section name,

// control section starting address,

// control section length

//at header record,

//H^\_\_\_\_\_\_^\_\_\_\_\_\_^\_\_\_\_\_\_

// name staddr seclen

sscanf(obj\_line,"%c%6s%06X%06X",&rec\_type,sec\_name,&sec\_start\_addr,&sec\_length);

//search ESTAB for control section name

if(search\_secname(filenum,sec\_name)){

printf("ERROR: Control section with this name already exists.\n");

fclose(fp);

return 0;

}

//set CSLTH to control section length

CSLTH = sec\_length;

//enter control section name into ESTAB with value CSADDR and CSLTH

strcpy(ESTAB[filenum].cs\_name,sec\_name);

ESTAB[filenum].addr = CSADDR;

ESTAB[filenum].len = CSLTH;

while(fgets(obj\_line,80,fp)!=NULL){

//if record type = 'E', break

if(obj\_line[0]=='E') break;

//if record type = 'D'

if(obj\_line[0]=='D'){

d\_index = 1;

while(d\_index<strlen(obj\_line)-1){

//scan each symbols separately

sscanf(obj\_line+d\_index,"%6s%06X",d\_sym,&d\_addr);

//search ESTAB for symbol name

if(search\_symname(filenum,d\_sym)){

printf("ERROR: Symbol name already exists.\n");

fclose(fp);

return 0;

}

//enter symbol into ESTAB with value

insert\_essym(filenum,d\_sym,d\_addr+CSADDR);

//point to next symbol

d\_index+=12;

}//end of while 'symbol exists'

}//end of record type = 'D'

}//end of while !'E'

//add CSLTH to CSADDR (starting address for next control section)

CSADDR += CSLTH;

//at the end of pass 1, ESTAB contains all external symbols

//defined in the set of control sections together with the address

//assigned to each.

fclose(fp);

return 1;

}

int linkload\_p2(char\* objfile, int filenum, int total\_file\_num){

char obj\_line[80];

char rec\_type;

char sec\_name[7];

int sec\_start\_addr, sec\_length;

//for reading T record

int t\_len, t\_start\_addr, t\_index, t\_addr;

//for reading R record

int r\_num, r\_addr;

unsigned int r\_index=1;

char r\_sym[7];

//for saving 1 byte from main memory

int obj\_byte;

//for reading M record

int m\_addr, m\_len, m\_ref, m\_code, m\_tmp;

//when format 4, save half byte containing 'xbpe'

int save\_value;

//used for modifying int object code into text for convenience

char m\_code\_txt[7];

//for saving + or - in M record

char operator;

//structure for saving symbols and their address with reference numbers as index

typedef struct REF{

char sym\_name[7];

int sym\_addr;

}ref;

ref ref\_list[6];

EXECADDR = progaddr;

CSADDR = ESTAB[filenum].addr;

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

//get the header record

fgets(obj\_line,80,fp);

//for 'H' record

//save record type, control section name,

// control section starting address,

// control section length

sscanf(obj\_line,"%c%6s%06X%06X",&rec\_type,sec\_name,&sec\_start\_addr,&sec\_length);

//set CSLTH to control section length

CSLTH = sec\_length;

while(fgets(obj\_line,80,fp)!=NULL){

if(obj\_line[0]=='E'){

//if an address is specified in End record then

//set EXECADDR to (CSADDR+specified address)

if(isHex(obj\_line+1)) EXECADDR = CSADDR + strtol(obj\_line+1,NULL,16);

break;

}

//for 'R' record

else if(obj\_line[0]=='R'){

r\_index=1;

//reference number 1 is for control section's address

ref\_list[1].sym\_addr = CSADDR;

//save reference number and symbol name with its address

while(r\_index<strlen(obj\_line)-1){

sscanf(obj\_line+r\_index,"%02X%6s",&r\_num,r\_sym);

r\_addr = search\_extsym(r\_sym,total\_file\_num);

if(r\_addr==-1){

printf("ERROR: Symbol not found in ESTAB.\n");

fclose(fp);

return 0;

}

ref\_list[r\_num].sym\_addr = r\_addr;

strcpy(ref\_list[r\_num].sym\_name,r\_sym);

r\_index+=8;

}

}

//for 'T' record

else if(obj\_line[0]=='T'){

sscanf(obj\_line,"%c%06X%02X",&rec\_type,&t\_start\_addr,&t\_len);

//original length was representing total bytes

t\_len \*= 2;

//include 'T'+start\_address

t\_len += 9;

//object code starts from index 9 in T record

t\_index = 9;

//set location to CSADDR + specified address

t\_addr = t\_start\_addr + CSADDR;

//move object code from record to location

while(t\_index<t\_len){

sscanf(obj\_line+t\_index,"%02X",&obj\_byte);

my\_mem[t\_addr] = obj\_byte;

t\_addr++;

t\_index+=2;

}

}//end of if 'T'

//for 'M' record

else if(obj\_line[0]=='M'){

sscanf(obj\_line,"%c%06X%02X%c%02X",&rec\_type,&m\_addr,&m\_len,&operator,&m\_ref);

m\_addr += CSADDR;

//calculate code to modify

sprintf(m\_code\_txt,"%02X%02X%02X",my\_mem[m\_addr],my\_mem[m\_addr+1],my\_mem[m\_addr+2]);

sscanf(m\_code\_txt,"%06X",&m\_code);

save\_value = m\_code/0x100000;

//add or subtract symbol value at location

if(operator=='+') m\_code += ref\_list[m\_ref].sym\_addr;

else if(operator=='-') m\_code -= ref\_list[m\_ref].sym\_addr;

else{

printf("ERROR: operator needs to be '+' or '-'.\n");

fclose(fp);

return 0;

}

sprintf(m\_code\_txt,"%08X",m\_code);

if(m\_len==0x05){

sscanf(m\_code\_txt,"%03X%1X%02X%02X",&m\_tmp,&my\_mem[m\_addr],&my\_mem[m\_addr+1],&my\_mem[m\_addr+2]);

my\_mem[m\_addr]+=save\_value\*0x10;

}

else sscanf(m\_code\_txt,"%02X%02X%02X%02X",&m\_tmp,&my\_mem[m\_addr],&my\_mem[m\_addr+1],&my\_mem[m\_addr+2]);

}//end of 'M'

}//end of while not EOF

CSADDR += CSLTH;

fclose(fp);

return 1;

}

void insert\_essym(int filenum, char\* sym\_name, int sym\_addr){

//create new node for ESTAB symbol

ESTAB\_sym\* node = (ESTAB\_sym\*)malloc(sizeof(ESTAB\_sym));

node->next = NULL;

node->address = sym\_addr;

strcpy(node->symbol,sym\_name);

//set cursor to current control section's node's es\_sym

ESTAB\_sym\* cur = ESTAB[filenum].es\_sym->next;

//if there is no symbol in ESTAB for this control section, insert node

if(cur==NULL) ESTAB[filenum].es\_sym->next = node;

//else find the end of the list and insert node

else{

while(cur->next!=NULL) cur = cur->next;

cur->next = node;

}

}

int search\_extsym(char\* sym\_name, int file\_num){

int i;

ESTAB\_sym\* cur;

//search for symbol and return symbol's address

for(i=0;i<file\_num;i++){

cur = ESTAB[i].es\_sym->next;

while(cur!=NULL){

if(!strcmp(cur->symbol,sym\_name)) return cur->address;

cur = cur->next;

}

}

//if symbol not found in ESTAB

return -1;

}

int search\_symname(int filenum, char\* sym\_name){

//symbol list of current control section

ESTAB\_sym\* cur = ESTAB[filenum].es\_sym->next;

while(cur!=NULL){

if(!strcmp(cur->symbol,sym\_name)) return 1;

cur = cur->next;

}

return 0;

}

int search\_secname(int filenum,char\* sec\_name){

int i;

for(i=0;i<filenum;i++){

if(!strcmp(ESTAB[i].cs\_name,sec\_name)) return 1;

}

return 0;

}

void show\_loadmap(int file\_num){

int i;

//for calculating total length of the program (sum of each section's length)

int total\_len=0;

ES\_TAB cur;

ESTAB\_sym\* cur\_sym;

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

printf("section\tname\n");

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

for(i=0;i<file\_num;i++){

cur = ESTAB[i];

//first print out current control section's data

printf("%s\t\t%04X\t%04X\n",cur.cs\_name,cur.addr,cur.len);

//add current control section's length

total\_len+=cur.len;

//set pointer to current contrl section's symbol list

cur\_sym=cur.es\_sym->next;

while(cur\_sym!=NULL){

//print out current control section's symbol list

printf("\t%6s\t%04X\n",cur\_sym->symbol,cur\_sym->address);

cur\_sym = cur\_sym->next;

}

}

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

//print out total length of the whole program

printf("\t total length %04X\n",total\_len);

}

void show\_bp(){

bp\* cur = bp\_head->next;

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

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

while(cur!=NULL){

printf("\t\t%X\n",cur->loc);

cur = cur->next;

}

}

void save\_bp(int bp\_loc){

bp\* cur = bp\_head->next;

//create new node

bp\* node = (bp\*)malloc(sizeof(bp));

node->loc = bp\_loc;

node->next = NULL;

//insert at the end of list

if(cur==NULL) bp\_head->next = node;

else{

while(cur->next!=NULL) cur = cur->next;

cur->next = node;

}

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

}

void clear\_bp(){

bp\* cur = bp\_head->next;

if(cur==NULL){

printf("\t\tno breakpoints existing!\n");

return;

}

//free nodes in bp list except for the head node

while(cur!=NULL){

bp\* next = cur->next;

free(cur);

cur = next;

}

bp\_head->next = NULL;

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

}

void run(){

//set start address

static int cur\_addr = 0;

//save total length

int run\_len = ESTAB[0].len;

//register values, A=0, X=1, L=2, B=3, S=4, T=5, PC=8, SW=9

static int reg[10];

//n,i,x,b,p

int n=0,i=0,x=0,b=0,p=0;

//code length

int code\_len;

//first half byte

int hbyte;

//one full object code

int objcode;

//displacement, 12 bit (for format 3)

int disp;

//address, 20 bit (for format 4)

int addr;

//for register instructions

int reg1, reg2;

//for target address

int target\_addr=0;

//for immediate addressing mode

int im\_value;

//set opcode

int opcode;

//when debugging, do not stop at the same bp as last time (it starts from there)

static int last\_bp = -1;

//if there was no bp before, set L = endpoint, PC = progaddr

if(!in\_debug){

reg[2] = progaddr + run\_len;

reg[8] = cur\_addr = EXECADDR;

}

//run through file until bp

while(1){

//if file reaches the last code

if(reg[8]==progaddr+run\_len){

//print register values

print\_reg(reg);

//initialize register values

for(i=0;i<10;i++) reg[i]=0;

printf("\t End Program\n");

//set debug flag to 0 to show no debugging is going on

in\_debug=0;

return;

}

//if file reaches bp

if(search\_bp(reg[8]) && last\_bp != reg[8]){

//print register values

print\_reg(reg);

printf("\t Stop at checkpoint[%X]\n",reg[8]);

last\_bp = reg[8];

//set debug flag to 1 to show debugging is going on

in\_debug=1;

return;

}

//set current address to PC

cur\_addr = reg[8];

//check first half byte of the code

hbyte = my\_mem[cur\_addr]/0x10;

//first half byte 9~B:format 2, else format 3/4 (no format 1 in this ex)

//if format 2, 2 bytes

if(hbyte>=0x9 && hbyte<=0xB) code\_len=2;

//if format 3 or 4, 3 or 4 bytes

else code\_len=3;

//set opcode, get rid of n & i

opcode = my\_mem[cur\_addr] & 0xFC;

//printf("opcode: %X\n",opcode);

//if format 2

if(code\_len==2){

//PC += 2

reg[8]+=2;

//if CLEAR, r1<-0

if(my\_mem[cur\_addr]==0xB4){

reg1 = my\_mem[cur\_addr+1]/0x10;

reg[reg1] = 0;

}

//if COMPR, (r1):(r2)

else if(my\_mem[cur\_addr]==0xA0){

//A0(r1)(r2)

reg1 = my\_mem[cur\_addr+1]/0x10;

reg2 = my\_mem[cur\_addr+1]%0x10;

if(reg1<0 || reg1>9 || reg2<0 || reg2>9){

printf("ERROR: register number is incorrect.\n");

return;

}

if(reg[reg1]<reg[reg2]) reg[9]=-1;

else if(reg[reg1]==reg[reg2]) reg[9]=0;

else reg[9]=1;

}

//if TIXR, X<-X+1, X:r1

else if(my\_mem[cur\_addr]==0xB8){

//X<-X+1

reg[1]++;

reg1 = my\_mem[cur\_addr+1]/0x10;

if(reg1<0||reg1>9){

printf("ERROR: register number is incorrect.\n");

return;

}

//X:r1

if(reg[1]<reg[reg1]) reg[9]=-1;

else if(reg[1]==reg[reg1]) reg[9]=0;

else reg[9]=1;

}

else{

printf("ERROR: invalid OPCODE in format 2.\n");

return;

}

continue;

}//end of format 2

//check if format 4

else{

hbyte = my\_mem[cur\_addr+1]/0x10;

//if 3rd half byte of object code is odd, e=1, format 4.

if(hbyte%2) code\_len = 4;

}

//if format 3

if(code\_len==3){

//PC += 3

reg[8]+=3;

//set object code

objcode = my\_mem[cur\_addr]\*0x10000+my\_mem[cur\_addr+1]\*0x100+my\_mem[cur\_addr+2];

//set nixbp (since format 3, e=0)

n = objcode & 0x020000;

i = objcode & 0x010000;

x = objcode & 0x008000;

b = objcode & 0x004000;

p = objcode & 0x002000;

disp = objcode & 0x000FFF;

//check sign bit(leftmost bit) of disp

//if minus, set left bits to F

if(disp & 0x000800) disp = disp | 0xFFFFF000;

//if simple addressing mode, n=1, i=1

if(n && i){

//if pc relative, disp + PC

if(p) target\_addr = reg[8]+disp;

//if base relative, disp + B

else if(b) target\_addr = disp+reg[3];

//if direct addressing mode

else if(!p && !b) target\_addr = disp+progaddr;

//if X

if(x) target\_addr += reg[1];

im\_value = my\_mem[target\_addr]\*0x10000+my\_mem[target\_addr+1]\*0x100+my\_mem[target\_addr+2];

}

//if indirect addressing mode, n=1, i=0

else if(n && !i){

if(p) disp += reg[8];

else if(b) disp += reg[3];

else if(!p && !b) disp+=progaddr;

if(x) disp += reg[1];

//since this is indirect addressing mode,

//target address = object code at updated disp

target\_addr = my\_mem[disp]\*0x10000+my\_mem[disp+1]\*0x100+my\_mem[disp+2];

im\_value = my\_mem[target\_addr]\*0x10000+my\_mem[target\_addr+1]\*0x100+my\_mem[target\_addr+2];

}

//if immediate addressing mode, n=0, i=1

else{

//no need to set target addr

im\_value = disp;

if(p) im\_value += reg[8];

else if(b) im\_value += reg[3];

}

run\_form34(opcode,im\_value,code\_len,target\_addr,reg,x);

}//end of format 3

//if format 4

else if(code\_len==4){

//for(int k=0;k<100;k++) printf("INFORM4");

reg[8]+=4;

//set object code

objcode = my\_mem[cur\_addr]\*0x1000000+my\_mem[cur\_addr+1]\*0x10000+my\_mem[cur\_addr+2]\*0x100+my\_mem[cur\_addr+3];

//set n,i,x,b,p,(e=1)

n = objcode & 0x02000000;

i = objcode & 0x01000000;

x = objcode & 0x00800000;

b = objcode & 0x00400000;

p = objcode & 0x00200000;

//set address to rightmost 5 half bytes

addr = objcode & 0x000FFFFF;

if(addr & 0x00080000) addr = addr | 0xFFF00000;

//if simple addressing mode, n=1, i=1

if(n&&i){

if(p) target\_addr = addr + reg[8];

else if(b) target\_addr = addr + reg[2];

else if(!p && !b) target\_addr = addr + progaddr;

if(x) target\_addr += reg[1];

im\_value = my\_mem[target\_addr]\*0x1000000+my\_mem[target\_addr+1]\*0x10000+my\_mem[target\_addr+2]\*0x100+my\_mem[target\_addr+3];

}

//if indirect mode, n = 1, i = 0

else if(n&&!i){

if(p) addr += reg[8];

else if(b) addr += reg[3];

if(x) addr += reg[1];

target\_addr = my\_mem[target\_addr]\*0x1000000+my\_mem[target\_addr+1]\*0x10000+my\_mem[target\_addr+2]\*0x100+my\_mem[target\_addr+3];

im\_value = my\_mem[target\_addr]\*0x1000000+my\_mem[target\_addr+1]\*0x10000+my\_mem[target\_addr+2]\*0x100+my\_mem[target\_addr+3];

}

//if immediate addressing mode, n=0, i=1

else{

im\_value = addr;

if(p) im\_value += reg[8];

else if(b) im\_value += reg[3];

}

run\_form34(opcode,im\_value,code\_len,target\_addr,reg,x);

}//end of format 4

//print\_reg(reg);

//return;

}//end of run before bp

return;

}

void run\_form34(int opcode, int im\_value, int code\_len, int target\_addr, int\* reg, int x){

//LDA A<-m..m+2

if(opcode==0x00) reg[0] = im\_value;

//LDB B<-m..m+2

else if(opcode==0x68) reg[3] = im\_value;

//LDT T<-m..m+2

else if(opcode==0x74) reg[5] = im\_value;

//LDCH A[rightmost byte] <- m

else if(opcode==0x50){

//erase rightmost byte

reg[0] = reg[0] & 0xFFFF00;

//put memory value in rightmost byte

//if x, A<-m[x]

if(x) reg[0] += im\_value / 0x10000;

else reg[0] += im\_value & 0x0000FF;

}

//STA m..m+2<-A

else if(opcode==0x0C) store\_mem(reg[0],target\_addr,code\_len\*2);

//STX m..m+2<-X

else if(opcode==0x10) store\_mem(reg[1],target\_addr,code\_len\*2);

//STL m..m+2<-L

else if(opcode==0x14) store\_mem(reg[2],target\_addr,code\_len\*2);

//STCH m<-A[rightmost byte]

else if(opcode==0x54) store\_mem(reg[0]&0x000000FF,target\_addr,code\_len\*2);

//J PC<-m

else if(opcode==0x3C) reg[8] = target\_addr;

//JSUB L<-PC; PC<-m

else if(opcode==0x48){

reg[2] = reg[8];

reg[8] = target\_addr;

//for(int i=0;i<100;i++) printf("%06X\n",target\_addr);

}

//JLT PC<-m if CC set to < (CC saved in SW reg)

else if(opcode==0x38){

if(reg[9]<0) reg[8] = target\_addr;

}

//JEQ PC<-m if CC set to =

else if(opcode==0x30){

if(reg[9]==0) reg[8] = target\_addr;

}

//RSUB PC<-L

else if(opcode==0x4C) reg[8] = reg[2];

//COMP A:m..m+2

else if(opcode==0x28){

if(reg[0]<im\_value) reg[9] = -1;

else if(reg[0]==im\_value) reg[9] = 0;

else reg[9] = 1;

}

//TD: continue after setting CC to <

else if(opcode==0xE0) reg[9] = -1;

//RD: continue after setting CC to =

else if(opcode==0xD8) reg[9] = 0;

//WD: skip

}

void store\_mem(int value, int addr, int halfbyte){

char txt\_value[10];

//if format 4, store 4 bytes starting from addr

if(halfbyte==8){

value = value & 0xFFFFFFFF;

sprintf(txt\_value,"%08X",value);

sscanf(txt\_value,"%02X%02X%02X%02X",&my\_mem[addr],&my\_mem[addr+1],&my\_mem[addr+2],&my\_mem[addr+3]);

}

//if format 3, store 3 bytes starting from addr

else{

//make sure there are only 3 bytes

value = value & 0x00FFFFFF;

sprintf(txt\_value,"%06X",value);

sscanf(txt\_value,"%02X%02X%02X",&my\_mem[addr],&my\_mem[addr+1],&my\_mem[addr+2]);

}

}

void print\_reg(int\* reg){

printf("A : %06X X : %06X\n",reg[0],reg[1]);

printf("L : %06X PC : %06X\n",reg[2],reg[8]);

printf("B : %06X S : %06X\n",reg[3],reg[4]);

printf("T : %06X\n",reg[5]);

}

int search\_bp(int bp\_loc){

bp\* cur = bp\_head->next;

while(cur!=NULL){

if(cur->loc == bp\_loc) return 1;

cur = cur->next;

}

return 0;

}