과목명: 시스템프로그래밍

담당교수명 : 소정민

<<Assignment 3>>

서강대학교 컴퓨터공학과

이정원20171672

목차

1. **프로그램 개요3**
2. **프로그램 설명4**
   1. 프로그램 흐름도 4
3. 모듈 정의**6**
   1. main() 6
   2. loader() 7
   3. run() 8
4. **전역 변수 정의 9**
5. **코드10**
6. **프로그램 개요**

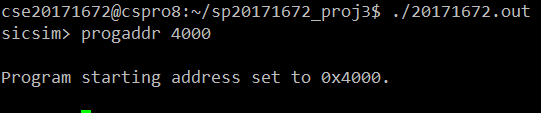
이 프로젝트에서는 오브젝트 프로그램을 로딩 및 링킹을 하여 메모리를 로드하고, 그것을 실행시키는 프로그램을 구현하는 것을 목적으로 한다.

사용자가 프로그램을 실행했을 경우, 다음과 같이 쉘 프롬프트가 나타난다.

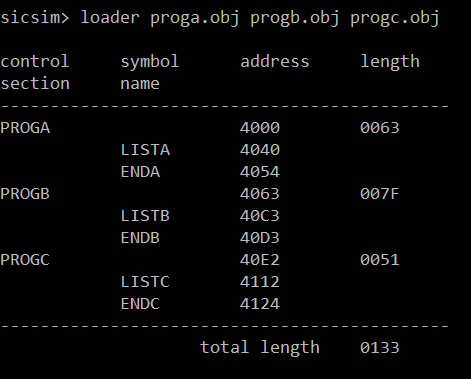


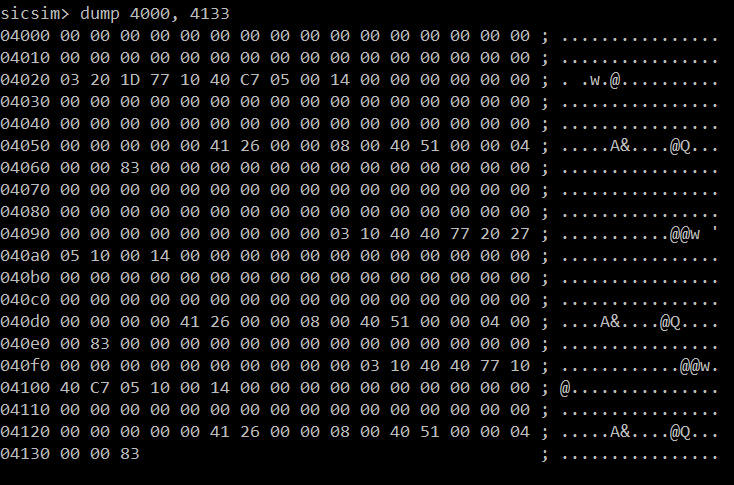
수행할 수 있는 기능들은 다음과 같다.

* **loader related commands : progaddr, load**
* progaddr [address] : 프로그램 시작 주소를 지정한다.



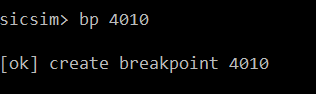
* load [filenames] : 오브젝트 파일을 읽어 링킹과 로딩을 한다.





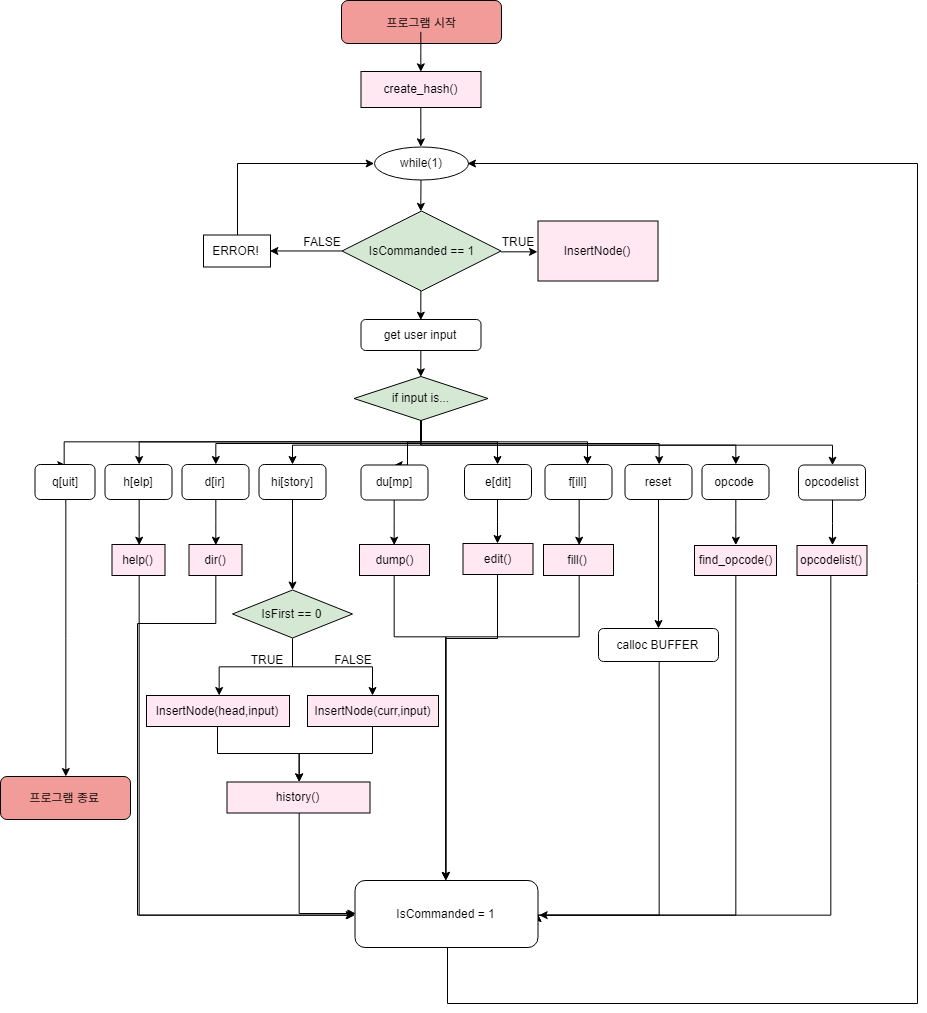
(dump를 통해 로딩이 되었는지 확인)

* **execution related commands : run, bp**
* run : 프로그램을 실행한다.
* bp : breakpoint를 지정한다.

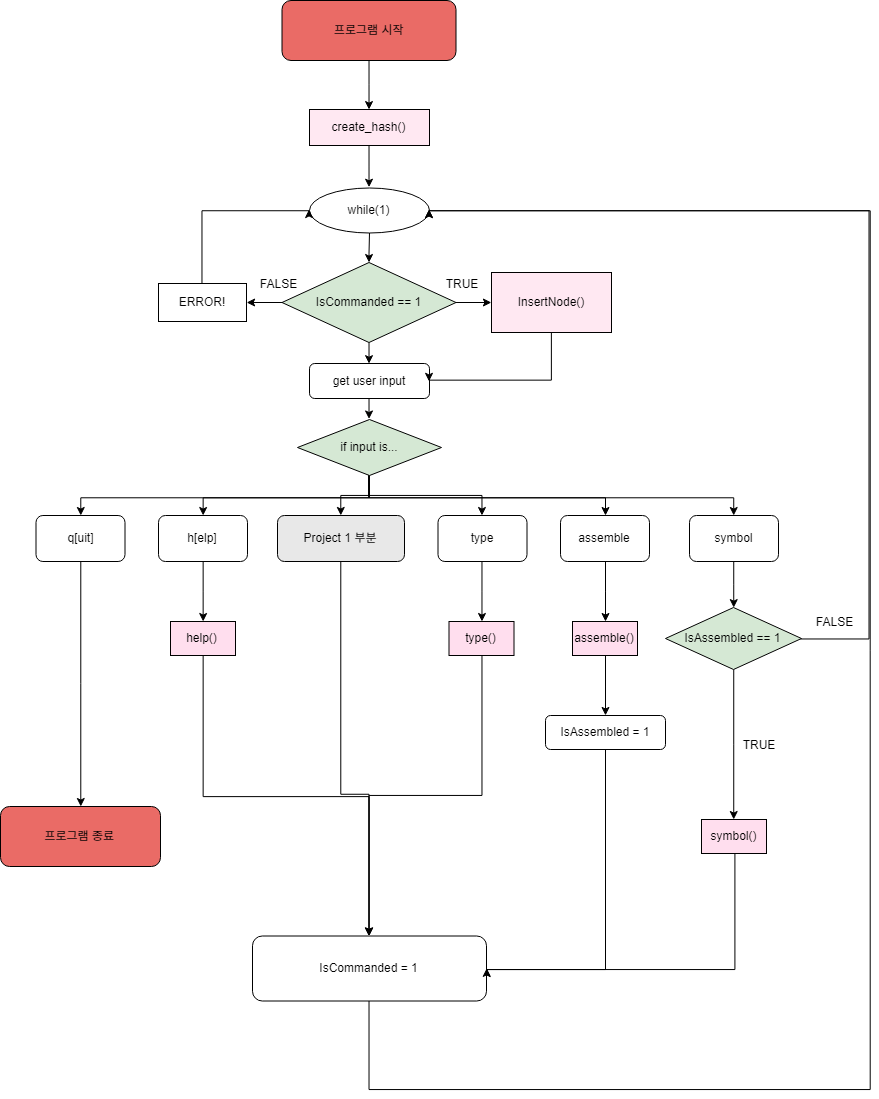


1. **프로그램 설명**
   1. 프로그램 흐름도

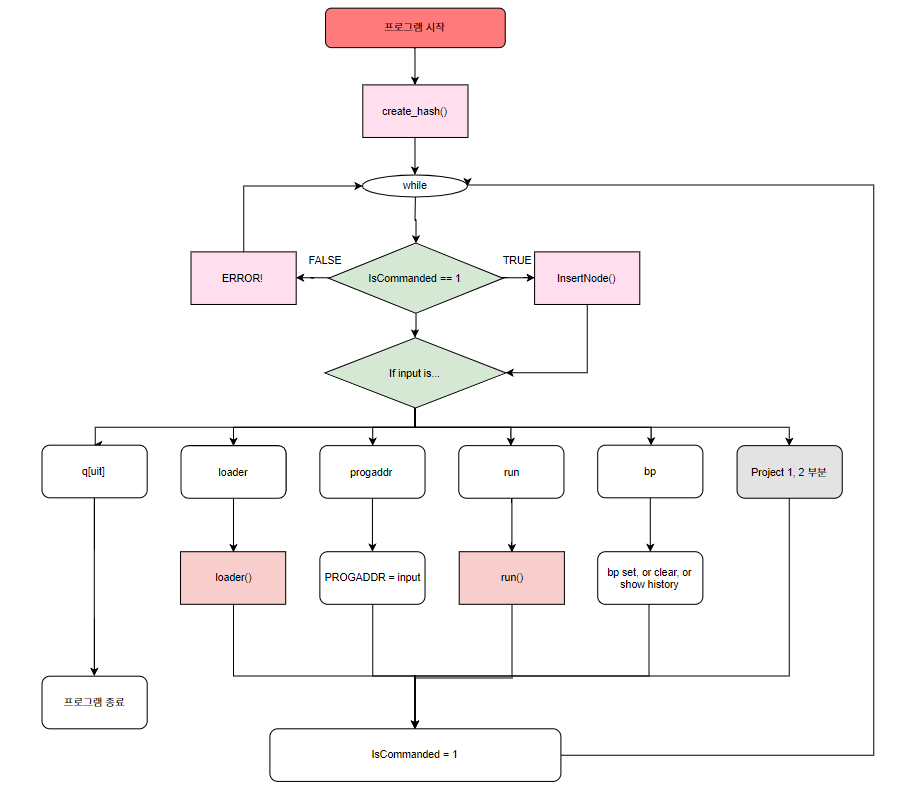
* Project 1 부분



* Project 2 부분



* Project 3 부분



1. **모듈 정의**
   1. **main()**

|  |  |
| --- | --- |
| 기능 | 사용자에게 입력받은 명령을 유효한지 아닌지 판별하여 해당 명령에 맞는 기능에 대응되는 함수들을 호출한다. 전체 while문은 입력받은 명령이 ‘quit’ 명령인 경우 break한다.  명령을 입력받을 때 공백이 들어올 수 있으므로 strtok을 이용하여 분리한 후 명령이 유효한지 판별한다. 명령이 유효하면 IsCommanded를 1로 설정해 주고, 그렇지 않으면 0으로 set한다. IsCommanded에 따라 에러 문구가 뜨게 구현하였다.  특히, dump, edit, fill 기능의 경우 입력받은 주소나 값이 지정 범위 내에 해당되지 않은 경우 에러 처리를 하였다.  quit 기능을 수행하는 경우에는 동적할당한 것들을 모두 free 해 주고, 연결 리스트의 경우 따로 함수를 만들어 호출하였다. (FreeNode, FreeHash) reset 기능을 수행하는 경우에는 calloc을 이용하여 BUFFER을 모두 0으로 초기화하였다.  특히 이번 프로젝트에서는 bp 와 progaddr 기능을 main에서 해결하였다.  progaddr 기능을 수행하였을 때 입력받은 주소를 16진수로 변환 시킨 다음, 전역으로 선언한 PROGADDR에 값을 저장한다.  bp 기능을 수행하였을 때 bp 만 입력받았을 경우 bp\_count만큼 여태까지 지정된 breakpoint를 출력한다. bp (주소)를 입력받았을 경우, 전역으로 선언한 BRKPN에 해당 값을 저장한다. bp clear을 입력받았을 경우, 입력받은 배열의 원소를 모두 0으로 초기화하고, bp\_count를 0으로 초기화한다. |
| 사용 변수 | **char\* input**  ⇒ 사용자에게 입력받은 명령을 저장할 배열  **char \*TEMP\_IN, \*TOK1, \*TOKEN, \*ORG\_IN, \*TOK2, \*ORG, \*O**  ⇒ 입력받은 명령을 공백이나 ‘,’등으로 구분하기 위해 저장할 토큰  **char \*AA, \*BB, \*ARR\_FILL, \*INPUT\_MN**  ⇒ 입력받은 명령을 토큰으로 분리하고 난 결과를 저장하기 위한 배열  **IsCommanded**  ⇒ 유효한 명령을 입력받고 수행했는지 판별하는 flag 변수 (0으로 초기화)  **IsFirst**  ⇒ 노드를 헤드에 이미 넣은 상태인지 판별하는 flag 변수 (0으로 초기화)  **IsHistory**  ⇒ history 기능을 수행했는지 판별하는 flag 변수 (0으로 초기화)/  **IsOneNum**  ⇒ ‘dump start’ 인지 ‘dump start end’인지 판별하는 flag 변수 (0으로 초기화)  **int LAST\_ADDRESS, LIMIT\_ADDRESS, EDIT\_ADDRESS, EDIT\_MEMORY, FILL\_START, FILL\_END, FILL\_MEMORY**  ⇒ dump, edit, fill에 관여하는 변수  **DIR \*dp, struct dirent \*file, struct stat buf**  ⇒ dir 기능에 관여하는 변수  **unsigned char \*BUFFER**  ⇒ dump, fill, edit 기능에 쓰일 메모리 배열  **struct NODE \*head**  ⇒ history 기능에 쓰일 노드의 head |

* 1. **loader()**

|  |  |
| --- | --- |
| 기능 | loader 기능을 수행하여 입력받은 파일 이름을 열어 오브젝트 파일을 읽어 링킹, 데이터에 로드하는 함수이다. 이 기능을 구현하기 위해 크게 pass 1 과 pass 2로 나누어 구현하였다.  CSADDR을 PROGADDR로 초기화한 다음, PASS 1 부분에서는 입력받은 파일의 수만큼 반복하는 큰 틀의 반복문이 존재한다. 파일을 열어 한 줄씩 읽어와 주소, 이름 등을 전역으로 선언한 ESTAB에 저장한다. H 레코드를 만났을 경우 컨트롤 섹션 (csect)에 저장하고, symbol에는 아무것도 저장하지 않는다. 그리고 해당하는 주소에 CSADDR을 더한 값을 테이블의 해당 주소와 프로그램 길이를 저장한다. 그리고 D 레코드를 만났을 경우, symbol에 이름을 저장하고, csect에는 아무것도 저장하지 않는다. 아까와 같이 테이블에 주소를 저장한다. 그리고 CSADDR에 프로그램 길이를 더한다.  EXECADDR 과 CSADDR을 PROGADDR로 초기화한 다음, PASS 2 부분에서는 PASS 1과 전체적인 틀을 같게 하되, T 레코드와 M 레코드를 본격적으로 읽어와 데이터를 로드한다. 헤더 레코드를 만났을 때, 해당 레퍼런스 숫자를 “01”로 지정한다. T 레코드를 만났을 경우, 주소와 길이를 읽어 반복문을 통해 오브젝트 코드를 데이터에 쭉 로드한다. R 레코드를 만났을 경우, ESTAB을 이용해 이름을 찾아 레퍼런스 숫자를 업데이트하고, 이름을 찾지 못했을 경우 에러 문구를 출력하고 강제종료한다. M 레코드를 만났을 경우 Modification이 필요한 주소와 형식을 읽어와 해당 주소의 데이터를 해당 레퍼런스 넘버(혹은 이름)의 주소를 ESTAB에서 찾아 더하거나 빼어 업데이트 한다. 링크할 프로그램이 없을 경우 해당 주소의 데이터에서 PROGADDR만 더한다. 성공적으로 PASS 2가 끝나면 파일을 닫고 반복문을 빠져나온다.  PASS 1과 PASS 2가 모두 끝났을 경우, ESTAB을 알맞은 형식에따라 출력하고 1을 반환한다. |
| 사용 변수 | **FILE \*fp**  ⇒ 입력받은 파일 이름을 열기 위한 포인터 변수  **char name[], symbol[], buffer[], temp\_address, temp\_length**  ⇒ 파일을 읽어와 해당 정보를 저장하기 위한 문자열 변수  **char first[]**  ⇒ 해당 줄의 첫 번째 문자를 저장하기 위한 변수  **int es\_count**  ⇒ ESTAB의  **int flag**  ⇒ ESTAB 에서 해당 이름이나 레퍼런스 넘버를 찾았는지 확인하기 위한 플래그 변수  **int loc, lent, specified add**  ⇒ 데이터 로드를 위해 필요한 정보를 저장하는 변수  **char temp\_refer, dummy\_memory, mod\_memory, fin\_memory**  **int temp\_memory**  **unsigned int dummy**  ⇒ 메모리 modification 할 때 관여하는 변수들 |

* 1. **run()**

|  |  |
| --- | --- |
| 기능 | 데이터를 읽어와 레지스터 값을 알아내기 위하여 오브젝트 코드를 차례대로 읽는 함수이다. breakpoint 가 없을 경우에는 전체 프로그램을 읽고, 그렇지 않으면 해당 breakpoint 주소에서 읽는 걸 멈춘다.  우선 시작 주소의 데이터를 읽어와 opcode 와 format 을 알아내는 과정을 거친다. 특히, & 연산을 이용하여 e\_flag를 구한 다음 형식이 4인지 3인지 저장한다. 그리고 switch 문을 이용하여 format을 구분하여 오브젝트 코드를 불러온 다음, disp, 플래그 변수들을 구한다. 그런 다음 opcode 명령어들에 따라 경우를 나누어 disp와 플래그 변수들을의 정보를 기반으로 레지스터 값을 변화시켜 준다. 그런 다음, 모든 프로그램을 읽었을 때 레지스터의 값들을 출력하고 1을 반환한다. |
| 사용 변수 | **int n\_flag, i\_flag, x\_flag, b\_flag, p\_flag, e\_flag**  ⇒ 플래그를 저장하기 위한 변수들  **int reg1, reg2**  ⇒ 해당 레지스터 넘버를 저장하기 위한 변수들  **int format, current, opcode**  ⇒ 데이터에서 불러온 정보를 저장하기 위한 변수들  **unsigned int loc**  ⇒ 데이터를 읽어오는 해당 데이터 주소를 저장하는 변수  **int disp**  ⇒ 해당 목적 주소를 저장하는 변수  **int obj\_full**  ⇒ 데이터에서 읽어온 오브젝트 코드를 저장하기 위한 변수 |

1. **전역 변수 정의**
   1. **typedef struct ESTAB**

⇒ loader 기능을 위해 이름, 주소, 길이, 레퍼런스 넘버를 저장하는 구조체.

* 1. **typedef struct runtable**

⇒ run 기능을 위해 명령어의 opcode와 형식을 저장하는 구조체.

* 1. **staticf runtable RUNTAB[]**

⇒ 이 프로젝트에서 필요한 opcode 와 형식을 저장한 구조체.

* 1. **struct ESTAB estab[HASHSIZE]**

⇒ 전역으로 선언한 loader 함수에서 쓰이는 ESTAB.

* 1. **unsinged int PROGADDR**

⇒ 시작 로딩 주소를 저장하기 위한 변수.

* 1. **unsigned int CSADDR**

⇒ control section의 시작 주소를 저장하기 위한 변수.

* 1. **unsigned int CSLTH**

⇒ 프로그램 길이를 저장하기 위한 변수.

* 1. **unsigned int EXECADDR**

⇒ pass 2 에서 주소를 변경할 때 필요한 변수.

* 1. **const char \*ARR\_LOAD[]**

⇒ 링킹할 파일 이름을 저장하기 위한 배열.

* 1. **unsigned int BRKPN[MAX\_INPUT\_SIZE]**

⇒ breakpoint를 저장하기 위한 배열.

* 1. **int bp\_count**

⇒ breakpoint 의 개수를 저장하기 위한 변수.

* 1. **static int REG[]**

⇒ run 기능을 위해 따로 선언한 레지스터 테이블. (각 인덱스가 레지스터 넘버에 해당한다.)

* 1. **int current\_bp**

⇒ breakpoint 를 현재 어디서 걸어야 하는지 저장하기 위한 변수

1. **코드**

|  |
| --- |
| **20171672.h** |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <dirent.h>  #include <sys/stat.h>  #include <time.h>  #include <math.h>  #define MAX\_INPUT\_SIZE 500  #define MEGA\_BYTE 1024\*1024  #define MEGA\_BYTE\_ROW 256\*256  #define MAX\_ARR\_SIZE 100  #define HASHSIZE 20  typedef struct NODE{  char command[100];  struct NODE\* link;  }NODE;  typedef struct OPTAB{  int id,count;  char code[10];  char name[10];  char format[10];  struct OPTAB\* next;  }OPTAB;  typedef struct TEMP\_OP{  char code[10];  char name[10];  char format[10];  }TEMP\_OP;  typedef struct SYMTAB{  char label[32];  int address;  }SYMTAB;  typedef struct HASH\_SYMTAB{  int id;  char label[32];  int address;  struct HASH\_SYMTAB\* next;  }HASH\_SYMTAB;  typedef struct ESTAB{  char csect[10];  char symbol[10];  unsigned int address;  unsigned int length;  char refer[10];  }ESTAB;  typedef struct runtable{  int code;  int format;  }runtable;  static runtable RUNTAB[]= {  {0x00,3},  {0x68,3},  {0x74,3},  {0x50, 3},  {0x0C, 3},  {0x10, 3},  {0x14, 3},  {0x54, 3},  {0x3C, 3},  {0x48, 3},  {0x38, 3},  {0x30, 3},  {0x4C, 3},  {0x28, 3},  {0xA0, 2},  {0xB4, 2},  {0xB8, 2},  {0xE0, 3},  {0xD8, 3},  {0xDC, 3}  };  typedef struct Intermediate{  unsigned short int line;  unsigned short int location;  unsigned int OBJ;  char label[32];  char mnmn[32];  char operand[32];  }Intermediate;  typedef struct Relocation{  int address;  int length;  }Relocation;  typedef struct Register{  char regname[32];  int number;  }Register;  static Register REGTAB[]={  { "A", 0 },  { "X", 1 },  { "L", 2 },  { "B", 3 },  { "S", 4 },  { "T", 5 },  { "F", 6 },  { "PC", 8 },  { "SW", 9 }  };  struct OPTAB\* hashtable[HASHSIZE];  struct HASH\_SYMTAB\* s\_hashtable[HASHSIZE];  struct SYMTAB symboltb[HASHSIZE][HASHSIZE];  struct TEMP\_OP optab;  struct ESTAB estab[HASHSIZE];  int BUFF\_length;  int REGidx;  int INDX;  int SYMidx;  int LOCCTR[100];  int LOCCTR\_counter = 0;  int SYMTAB\_counter[10] = { 0, };  int start\_address[10] = { 0, };  int program\_length[10] = { 0, };  int list\_index[10] = { 0, };  int relocation\_counter[10] = { 0, };  int prefix;  int IsAssembled = 0;  char BUFF[200];  char LABEL[32];  char MNMN[32];  char OPND[32];  char end\_operand[32];  char fileobj[32];  char filelst[32];  unsigned int PROGADDR = 0;  unsigned int CSADDR = 0;  unsigned int CSLTH = 0;  unsigned int EXECADDR = 0;  char ORG\_LOAD[200] = { 0, };  const char \*ARR\_LOAD[5];  unsigned int BRKPN[MAX\_INPUT\_SIZE] = { 0, };  int bp\_count = 0;  int current\_bp = 0;  static int REG[11] = { 0, };  Intermediate\* list[10][100];  Relocation Rlist[10][20]; |

|  |
| --- |
| **20171672.c** |
| **#include "20171672.h"**  **/\*------------------------------------------------\*/**  **/\*Function : InsertNode()\*/**  **/\*Purpose : Put user's input into linked list for 'History' command.\*/**  **/\*No return value.\*/**  **/\*------------------------------------------------\*/**  **void InsertNode(NODE \*head, char input[]){**    **struct NODE \*new = malloc(sizeof(struct NODE));**  **strcpy(new->command, input);**  **new->link = head->link;**  **head->link = new;**  **}**  **/\*------------------------------------------------\*/**  **/\*Function : FreeNode()\*/**  **/\*Purpose : Free the node.\*/**  **/\*No return value.\*/**  **/\*------------------------------------------------\*/**  **void FreeNode(NODE \*head){**  **while(head->link != NULL){**  **NODE \*tmp = head;**  **head = head->link;**  **free(tmp);**  **}**  **}**  **/\*------------------------------------------------\*/**  **/\*Function : FreeHash()\*/**  **/\*Purpose : Free the hashtable.\*/**  **/\*No return value.\*/**  **/\*------------------------------------------------\*/**  **void FreeHash(){**  **for(int i = 0; i < HASHSIZE; i++){**  **while(hashtable[i] != NULL){**  **OPTAB \*tmp = hashtable[i];**  **hashtable[i] = hashtable[i]->next;**  **free(tmp);**  **}**  **}**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : FreeSHash()**  **Purpose : Free the symbol table**  **Return value : 1 or 0**  **\*/**  **/\*------------------------------------------------\*/**  **void FreeSHash(){**  **for(int i = 0; i < HASHSIZE; i++){**  **while(s\_hashtable[i] != NULL){**  **HASH\_SYMTAB \*tmp = s\_hashtable[i];**  **s\_hashtable[i] = s\_hashtable[i]->next;**  **free(tmp);**  **}**  **}**  **}**  **/\*------------------------------------------------\*/**  **/\*Function : help()\*/**  **/\*Purpose : Function 'help' command.\*/**  **/\*No return value.\*/**  **/\*------------------------------------------------\*/**  **int help(){**  **printf("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");**  **}**  **/\*------------------------------------------------\*/**  **/\*Fucntion : dir()\*/**  **/\*Purpose : Function 'dir' command.**  **Print current directory's file name and state. \*/**  **/\*No return value\*/**  **/\*------------------------------------------------\*/**    **void dir(DIR \*dp, struct dirent \*file, struct stat buf){**  **dp = opendir(".");**  **if(dp == NULL){**  **printf("There is no directory.\n");**  **return;**  **}**  **while(1){**  **file = readdir(dp);**  **if(!file) break;**    **if(strcmp(file->d\_name, ".") == 0) continue;**  **if(strcmp(file->d\_name, "..") == 0) continue;**  **stat(file->d\_name, &buf);**  **if(buf.st\_mode & S\_IFDIR){**  **printf("%s/ ", file->d\_name);**  **}**  **else if(buf.st\_mode & S\_IXUSR){**  **printf("%s\* ", file->d\_name);**  **}**  **else{**  **printf("%s ", file->d\_name);**  **}**  **}**    **closedir(dp);**  **printf("\n");**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : history()**  **Purpose : Function 'history' command.**  **Print all the previous valid commands in the linked list.**  **No return value.**  **\*/**  **/\*------------------------------------------------\*/**  **int history(NODE \*current){**  **int count=1;**  **while(current != NULL){**  **printf("%d %s\n", count, current->command);**  **current = current->link;**  **count++;**  **}**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : dump()**  **Purpose : Function 'dump', 'dump start', and 'dump start, end' command.**  **Print the memory address, memory content, and content in ASCII**  **code.**  **No return value.**  **\*/**  **/\*------------------------------------------------\*/**  **void dump(unsigned char \*BUFFER, int \*START\_ADDRESS, int \*END\_ADDRESS){**    **int i=0, j = 0, k = 0;**  **int MEMORY\_NUM = 0, LIMIT\_MEMORY = 160;**  **int START\_X=(\*START\_ADDRESS)/16, START\_Y= (\*START\_ADDRESS)%16;**  **int Hex\_Num = START\_X, END\_SIGN=0, IsCompleteLine = 0;**  **unsigned char \*\*NEW\_BUFFER;**  **NEW\_BUFFER = (unsigned char\*\*)malloc(sizeof(unsigned char\*)\*MEGA\_BYTE\_ROW);**  **for(i=0; i < MEGA\_BYTE\_ROW; i++){**  **NEW\_BUFFER[i] = (unsigned char \*)malloc(sizeof(unsigned char) \* 16);**  **}**  **if(\*END\_ADDRESS != 0){**  **LIMIT\_MEMORY = \*END\_ADDRESS - \*START\_ADDRESS; // LIMIT SETTING**  **}**  **for(i=START\_X; i < 11+START\_X+LIMIT\_MEMORY; i++){**  **for(k = 0; k < 16; k++){**  **NEW\_BUFFER[i][k] = BUFFER[j + \*START\_ADDRESS];**  **j++;**  **}**  **}**  **i=START\_X;**  **while(1){**  **for(k=0; k < 16; k++){**  **if(i\*16 + k > 0xfffff){**  **printf(" ");**  **END\_SIGN = 1;**  **}**  **if(MEMORY\_NUM == LIMIT\_MEMORY){**  **END\_SIGN = 1;**  **}**  **if(END\_SIGN == 1){**  **if(k == 0){**  **IsCompleteLine = 1;**  **break;**  **}**  **else printf(" ");**  **}**  **else{**  **if(k == 0) printf("%05x", Hex\_Num\*16);**  **if(i == START\_X && k < START\_Y){**  **printf(" ");**  **}**  **else{**  **printf(" %02X", NEW\_BUFFER[i][k]);**  **MEMORY\_NUM++;**  **}**  **}**  **}**  **if(IsCompleteLine == 1) break;**  **printf(" ; ");**  **for(k=0; k < 16; k++){**  **if(i == START\_X && k < START\_Y){**  **printf(".");**  **}**  **else{**  **if(END\_SIGN != 1){**  **if(NEW\_BUFFER[i][k] >= 0x20 && NEW\_BUFFER[i][k] <= 0x7E){**  **printf("%C", NEW\_BUFFER[i][k]);**  **}**  **else printf(".");**  **}**  **else printf(".");**  **}**  **}**    **printf("\n");**  **if(END\_SIGN == 1) break;**  **Hex\_Num++;**  **i++;**    **}**  **printf("\n");**  **for(i=0; i < MEGA\_BYTE\_ROW; i++){**  **free(NEW\_BUFFER[i]);**  **}**  **free(NEW\_BUFFER);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : edit()**  **Purpose : Function 'edit address value' command.**  **No return value.**  **\*/**  **/\*------------------------------------------------\*/**  **void edit(unsigned char \*BUFFER, int \*USER\_ADDRESS, int \*EDIT\_VALUE){**    **BUFFER[(unsigned int)(\*USER\_ADDRESS)] = (unsigned char)(\*EDIT\_VALUE);**  **printf("USER ADDRESS is %X\n", \*USER\_ADDRESS);**  **printf("BUFFER is %X\n", BUFFER[(unsigned int)(\*USER\_ADDRESS)]);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : fill()**  **Purpose : Function 'fill' command.**  **No return value.**  **\*/**  **/\*------------------------------------------------\*/**  **void fill(unsigned char \*BUFFER, int \*START\_ADDRESS, int \*END\_ADDRESS, int \*FILL\_VALUE){**  **int i;**  **for(i = \*START\_ADDRESS; i < \*END\_ADDRESS; i++){**  **BUFFER[i] = (unsigned char)(\*FILL\_VALUE);**  **}**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : add\_hash()**  **Purpose : Insert a node into hashtable of specific index.**  **No return value.**  **\*/**  **/\*------------------------------------------------\*/**  **void add\_hash(int key, OPTAB\* node){**  **int index = key % HASHSIZE;**  **OPTAB\* cur = (OPTAB\*)malloc(sizeof(OPTAB));**  **if(hashtable[index] == NULL){**  **hashtable[index] = node;**  **}**  **else{**  **cur = hashtable[index];**  **while(cur->next != NULL){**  **cur = cur->next;**  **}**  **cur->next = node;**  **}**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : create\_hash()**  **Purpose : Create hashtable by reading from the file.**  **No return value.**  **\*/**  **/\*------------------------------------------------\*/**  **void create\_hash(){**  **FILE \*fp1 = fopen("opcode.txt", "r");**  **char code[100], name[100], format[100];**  **srand(time(NULL));**  **if(fp1 == NULL){**  **printf("File Open Error!\n");**  **return;**  **}**  **while(!feof(fp1)){**  **fscanf(fp1, "%s%\*[ \t]%s%\*[ \t]%s\n", code, name, format);**  **OPTAB\* node = (OPTAB\*)malloc(sizeof(OPTAB));**  **node->id = rand() % HASHSIZE;**  **strcpy(node->code, code);**  **strcpy(node->name, name);**  **strcpy(node->format, format);**  **node->next = NULL;**  **add\_hash(node->id, node);**  **}**  **fclose(fp1);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : opcodelist()**  **Purposse : Function 'opcodelist' command.**  **No return value.**  **\*/**  **/\*------------------------------------------------\*/**  **void opcodelist(){**  **for(int i = 0; i < HASHSIZE; i++){**  **printf("%d : ", i);**  **if(hashtable[i] != NULL){**  **OPTAB\* node = hashtable[i];**  **while(node->next != NULL){**  **printf("[%s, %s] -> ", node->name, node->code);**  **node = node->next;**  **}**  **printf("[%s, %s]", node->name, node->code);**  **}**  **printf("\n");**  **}**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : find\_opcode()**  **Purpose : Function 'opcode menomonic' command.**  **Return value : IsCommanded 0 or 1**  **\*/**  **/\*------------------------------------------------\*/**  **int find\_opcode(char \*INPUT\_MN){**  **FILE \*fp2 = fopen("opcode.txt", "r");**  **char code[100], name[100];**  **int MAX\_SIZE = 100;**  **if(fp2 == NULL){**  **printf("File Open Error!\n");**  **return 0 ;**  **}**  **while(!feof(fp2)){**  **fscanf(fp2, "%s%\*[ \t]%s%\*[ \t]%\*s\n", code, name);**  **if(strcmp(name, INPUT\_MN) == 0){**  **printf("opcode is %s.\n", code);**  **fclose(fp2);**  **return 1;**  **}**  **}**  **fclose(fp2);**  **return 0;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : type()**  **Purpose : Function 'type' command.**  **Return value : IsCommanded 0 or 1**  **\*/**  **/\*------------------------------------------------\*/**  **int type(DIR \*dp, struct dirent \*file, struct stat buf, char \*AA){**  **FILE \*fp3;**  **char ch;**  **dp = opendir(".");**  **if(dp == NULL){**  **printf("There is no directory.\n");**  **closedir(dp);**  **return 0;**  **}**  **printf("\n");**  **while(1){**  **file = readdir(dp);**  **if(!file) {**  **printf("There is no file here!\n");**  **break;**  **}**  **if(strcmp(file->d\_name, AA) == 0){**  **if(buf.st\_mode & S\_IFDIR) continue;**  **fp3 = fopen(AA, "r");**  **if(fp3 == NULL){**  **closedir(dp);**  **return 0;**  **}**  **while(1){**  **ch = fgetc(fp3);**  **if(ch == EOF) break;**  **putchar(ch);**  **}**  **printf("\n");**  **fclose(fp3);**  **closedir(dp);**  **return 1;**  **}**  **}**  **closedir(dp);**  **return 0;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : readline()**  **Purpose : Read label from asm file and skip spaces.**  **Return value : LABEL**  **\*/**  **/\*------------------------------------------------\*/**  **char\* readline(){**  **int JNDX = 0;**  **LABEL[0] = '\0';**  **while(BUFF[INDX] != ' ' && BUFF[INDX] != '\t' && BUFF[INDX] != '\n'){**  **LABEL[JNDX++] = BUFF[INDX++];**  **}**  **LABEL[JNDX] = '\0';**  **return(LABEL);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : skip()**  **Purpose : To skip space and tab.**  **Return value : None.**  **\*/**  **/\*------------------------------------------------\*/**  **void skip(){**  **while(BUFF[INDX] == ' ' || BUFF[INDX] == '\t') INDX++;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : readmnmn()**  **Purpose : Read mnemonic from the file.**  **Return value : MNMN**  **\*/**  **/\*------------------------------------------------\*/**  **char\* readmnmn(){**  **int JNDX = 0;**  **MNMN[0] = '\0';**  **while(BUFF[INDX] != ' ' && BUFF[INDX] != '\t' && BUFF[INDX] != '\n'){**  **MNMN[JNDX++] = BUFF[INDX++];**  **}**  **MNMN[JNDX] = '\0';**  **return(MNMN);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : readopnd()**  **Purpose : Read operand from the file.**  **Return value : OPND**  **\*/**  **/\*------------------------------------------------\*/**  **char\* readopnd(){**  **int temp = 0, flag = 0;**  **int JNDX = 0;**  **OPND[0] = '\0';**  **temp = INDX;**  **while(1){**  **if(BUFF[temp] == '\0') break;**  **if(BUFF[temp] == ',') flag = 1;**  **temp++;**  **}**  **if(flag == 0){**  **while((BUFF[INDX] != '\n') && (BUFF[INDX] != '\r') && (BUFF[INDX] != ' ') && (BUFF[INDX] != '\t') && (INDX < BUFF\_length)){**  **OPND[JNDX] = BUFF[INDX];**  **JNDX++;**  **INDX++;**  **}**  **OPND[JNDX] = '\0';**  **}**  **else{**  **while(BUFF[INDX-1] != ','){**  **OPND[JNDX] = BUFF[INDX];**  **JNDX++;**  **INDX++;**  **}**  **skip();**  **while((BUFF[INDX] != '\n') && (BUFF[INDX] != '\r') && (BUFF[INDX] != ' ') && (BUFF[INDX] != '\t') && (INDX < BUFF\_length)){**  **OPND[JNDX] = BUFF[INDX];**  **JNDX++;**  **INDX++;**  **}**  **OPND[JNDX] = '\0';**  **}**  **return(OPND);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : readprefix()**  **Purpose : Read mnemonic and determine what the prefix is.**  **Return value : prefix**  **\*/**  **/\*------------------------------------------------\*/**  **int readprefix(char \*MNMN){**  **prefix = 0;**  **switch (MNMN[0]){**  **case '+':**  **prefix = 1;**  **break;**  **case '#':**  **prefix = 2;**  **break;**  **case '@':**  **prefix = 3;**  **break;**  **default:**  **prefix = 0;**  **}**  **return prefix;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : searchSYMTAB()**  **Purpose : Search label from the symbol table.**  **Return value : 1 or 0**  **\*/**  **/\*------------------------------------------------\*/**  **int searchSYMTAB(char\* label, int idx){**  **if(readprefix(label)){**  **label = label + 1;**  **}**  **for(int i = 0; i <= SYMTAB\_counter[idx]; i++){**  **if(!strcmp(symboltb[idx][i].label, label)){**  **SYMidx = i;**  **return 1;**  **}**  **}**  **return 0;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : searchREGTAB()**  **Purpose : Search register from the register table.**  **Return value : 1 or 0**  **\*/**  **/\*------------------------------------------------\*/**  **int searchREGTAB(char \*REG){**  **int size = sizeof(REGTAB) / sizeof(Register);**  **for(int i = 0; i < size; i++){**  **if(!strcmp(REG, REGTAB[i].regname)){**  **REGidx = i;**  **return 1;**  **}**  **}**  **return 0;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : searchOPTAB()**  **Purpose : Search mnemonic from the opcode table.**  **Return value : 1 or 0**  **\*/**  **/\*------------------------------------------------\*/**  **int searchOPTAB(char \*MNMN){**  **if(readprefix(MNMN)){**  **MNMN = MNMN + 1;**  **}**    **for(int i = 0; i < HASHSIZE; i++){**  **if(hashtable[i] != NULL){**  **OPTAB\* node = hashtable[i];**  **while(node != NULL){**  **if(!strcmp(node->name, MNMN)){**  **strcpy(optab.name, node->name);**  **strcpy(optab.code, node->code);**  **strcpy(optab.format, node->format);**  **return 1;**  **}**  **else node = node->next;**  **}**  **}**  **}**  **return 0;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : insertSYMTAB()**  **Purpose : Record symbol into symbol table.**  **Return value : None**  **\*/**  **/\*------------------------------------------------\*/**  **void insertSYMTAB(char\* label){**  **if(readprefix(label)){**  **label = label + 1;**  **}**    **strcpy(symboltb[0][SYMTAB\_counter[0]].label, label);**  **symboltb[0][SYMTAB\_counter[0]].address = LOCCTR[LOCCTR\_counter - 1];**  **SYMTAB\_counter[0]++;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : IsNum()**  **Purpose : To determine whether string is number or not.**  **Return value : 1 or 0**  **\*/**  **/\*------------------------------------------------\*/**  **int IsNum(char \*c){**  **if(readprefix(c)) c += 1;**  **for(int i = 0; i < strlen(c); i++){**  **if('0' > c[i] || '9' < c[i]){**  **if(c[i] == '-') continue;**  **return 0;**  **}**  **}**  **return 1;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : IsFloatNum()**  **Purpose : To determine whether string is float number or not.**  **Return value : 1 or 0**  **\*/**  **/\*------------------------------------------------\*/**  **int IsFloatNum(char \*str){**  **int f = 0;**  **if(readprefix(str)) str += 1;**  **for(int i = 0; i < strlen(str); i++){**  **if('0' > str[i] || '9' < str[i]){**  **if(str[i] == '.' && f == 0){**  **f = 1;**  **continue;**  **}**  **if(str[i] == '-') continue;**  **return 0;**  **}**  **}**  **return (f != 0) ? 1 : 0;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : TwelveBit()**  **Purpose : To do 2's complement when disp is negative.**  **Return value : disp**  **\*/**  **/\*------------------------------------------------\*/**  **unsigned long TwelveBit(int disp, int length){**  **if(disp >= 0) return disp;**  **if(length == 5) disp ^= 0xFF00000;**  **else disp ^= 0xFFFFF000;**  **return disp;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : StrToDec**  **Purpose : To make string into decimal number**  **Return value : dec\_num**  **\*/**  **/\*------------------------------------------------\*/**  **int StrToDec(char \*c){**  **if(readprefix(c)) c+= 1;**  **int dec\_num = 0;**  **char temp[10];**  **strcpy(temp, c);**  **for(int i = strlen(c) - 1, l = 1; i >= 0; i--){**  **if(temp[0] == '-') continue;**  **dec\_num = dec\_num + (int)(temp[i] - '0')\*l;**  **l = l\*10;**  **}**  **return (temp[0] == '-') ? (-dec\_num) : (dec\_num);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : StrToFloat()**  **Purpose : To make string into float number**  **Return value : number**  **\*/**  **/\*------------------------------------------------\*/**  **int StrToFloat(char \*c){**  **double number = 0;**  **for(int i = strlen(c) - 1; i >= 0; i--){**  **number /= 10.0;**  **number += (c[i] - '0')/10.0;**  **}**  **return number;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : StrToHex()**  **Purpose : To make string into hexadecimal number**  **Return value : hex**  **\*/**  **/\*------------------------------------------------\*/**  **int StrToHex(char \*c){**  **int hex = 0;**  **int i,j;**  **char temp[10];**  **strcpy(temp, c);**  **for(i = strlen(temp) - 1, j = 1; i >= 0; i--){**  **if(temp[i] >= '0' && temp[i] <= '9') hex = hex + (int)(temp[i] - '0')\*j;**  **else if(temp[i] >= 'A' && temp[i] <= 'F') hex = hex + (int)(temp[i] - 'A' + 10)\*j;**  **else if(temp[i] >= 'a' && temp[i] <= 'f') hex = hex + (int)(temp[i] - 'a' + 10)\*j;**  **j = j\*16;**  **}**  **return (hex);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : findlength()**  **Purpose : To compute the length of the string**  **Return value : b**  **\*/**  **/\*------------------------------------------------\*/**  **int findlength(char \*c){**  **unsigned int b;**  **char len[32];**  **strcpy(len, c);**  **if(len[0] == 'C' || len[0] == 'c' && len[1] == '\''){**  **for(b = 2; b <= strlen(len); b++){**  **if(len[b] == '\''){**  **b -= 2;**  **break;**  **}**  **}**  **}**  **if(len[0] == 'X' || len[0] == 'x' && len[1] == '\'') b = 1;**  **return (b);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : add\_SYMTAB()**  **Purpose : To add symbol into symbol table**  **Return value : none**  **\*/**  **/\*------------------------------------------------\*/**  **void add\_SYMTAB(int key, HASH\_SYMTAB\* node){**  **int index = key % HASHSIZE;**  **HASH\_SYMTAB\* cur = (HASH\_SYMTAB\*)malloc(sizeof(HASH\_SYMTAB));**  **if(s\_hashtable[index] == NULL) s\_hashtable[index] = node;**  **else{**  **cur = s\_hashtable[index];**  **while(cur->next != NULL) cur = cur->next;**  **cur->next = node;**  **}**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : create\_SYMTAB()**  **Purpose : To create hash table of symbol table**  **Return value : none**  **\*/**  **/\*------------------------------------------------\*/**  **void create\_SYMTAB(){**  **int j;**  **srand(time(NULL));**  **for(j = 0; j < SYMTAB\_counter[0]; j++){**  **HASH\_SYMTAB\* node = (HASH\_SYMTAB\*)malloc(sizeof(HASH\_SYMTAB));**  **node->id = rand() % HASHSIZE;**  **strcpy(node->label, symboltb[0][j].label);**  **node->address = symboltb[0][j].address;**  **node->next = NULL;**  **add\_SYMTAB(node->id, node);**    **}**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : symbol()**  **Purpose : Function command 'symbol'**  **Return value : none**  **\*/**  **/\*------------------------------------------------\*/**  **void symbol(){**  **int i, j, k, t;**  **char temp[32];**  **char lab1[32], lab2[32];**  **for(i = 0; i < SYMTAB\_counter[0] -1; i++){**  **for(j = 0; j < SYMTAB\_counter[0] - (i+1); j++){ // bubble sort to rearrange in descending order**  **strcpy(lab1, symboltb[0][j].label);**  **strcpy(lab2, symboltb[0][j+1].label);**  **if(strcmp(lab1, lab2) < 0){**  **strcpy(temp, symboltb[0][j+1].label);**  **t = symboltb[0][j+1].address;**  **strcpy(symboltb[0][j+1].label, symboltb[0][j].label);**  **symboltb[0][j+1].address = symboltb[0][j].address;**  **strcpy(symboltb[0][j].label, temp);**  **symboltb[0][j].address = t;**  **}**  **}**  **}**  **for(j = 0; j < SYMTAB\_counter[0]; j++){**  **printf("\t%s\t%04X\n", symboltb[0][j].label, symboltb[0][j].address);**  **}**    **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : createLST()**  **Purpose : To create listing file**  **Return value : none**  **\*/**  **/\*------------------------------------------------\*/**  **void createLST(char \*AA){**  **int j, length;**  **FILE \*fp\_lst, \*f;**  **char \*T = NULL, copy[10];**    **strcpy(copy, AA);**  **T = strtok(AA, ".");**  **strcat(T, ".lst");**  **strcpy(filelst, T);**  **fp\_lst = fopen(filelst, "w");**  **f = fopen(copy, "r");**  **for(j = 0; j < list\_index[0]; j++){**  **length = 0;**  **while(1){**  **fgets(BUFF, 200, f);**  **if(BUFF[0] == '.') fprintf(fp\_lst, "%s", BUFF);**  **else break;**  **}**  **if(strlen(list[0][j]->operand) <= 0 && !strcmp(list[0][j]->mnmn, "END")) continue;**  **if(!strcmp(list[0][j]->mnmn, "BASE") || !strcmp(list[0][j]->mnmn, "END")){**  **fprintf(fp\_lst, " \t%-10s%-10s%-10s\t", list[0][j]->label, list[0][j]->mnmn, list[0][j]->operand);**  **}**  **else fprintf(fp\_lst, "%04X\t%-10s%-10s%-10s\t", list[0][j]->location, list[0][j]->label, list[0][j]->mnmn, list[0][j]->operand);**  **if(!strcmp(list[0][j]->mnmn, "START") || !strcmp(list[0][j]->mnmn, "RESW") || !strcmp(list[0][j]->mnmn, "RESB") || !strcmp(list[0][j]->mnmn, "BASE") || !strcmp(list[0][j]->mnmn, "NOBASE") || !strcmp(list[0][j]->mnmn, "END")) {**  **fprintf(fp\_lst, " ");**  **}**  **else if(searchOPTAB(list[0][j]->mnmn)){**  **if(!strcmp(optab.format, "3/4")){**  **if(readprefix(list[0][j]->mnmn)) fprintf(fp\_lst, "%08X", list[0][j]->OBJ);**  **else fprintf(fp\_lst, "%06X", list[0][j]->OBJ);**  **}**  **else if(!strcmp(optab.format, "2")) fprintf(fp\_lst, "%04X", list[0][j]->OBJ);**    **else if(!strcmp(optab.format,"1")) fprintf(fp\_lst, "%02X", list[0][j]->OBJ);**  **}**  **else{**  **if(IsFloatNum(list[0][j]->operand)) fprintf(fp\_lst, "%012X", list[0][j]->OBJ);**  **else{**  **length = findlength(list[0][j]->operand);**  **if(length == 1) fprintf(fp\_lst, "%02X", list[0][j]->OBJ);**  **else if(length == 2) fprintf(fp\_lst, "%04X", list[0][j]->OBJ);**  **else fprintf(fp\_lst, "%06X", list[0][j]->OBJ);**  **}**  **}**  **fprintf(fp\_lst, "\n");**  **}**  **fclose(f);**  **fclose(fp\_lst);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : createOBJ()**  **Purpose : To create object file**  **Return value : none**  **\*/**  **/\*------------------------------------------------\*/**  **void createOBJ(char \*AA){**    **int first\_address, last\_address, temp\_address, temp\_format;**  **int first\_index, last\_index;**  **int x, y;**  **char \*T = NULL;**  **FILE \*fp\_obj;**  **int j;**    **char temp\_operator[12][10];**  **char temp\_operand[12][10];**  **unsigned int temp\_obj[30];**  **T = strtok(AA, ".");**  **strcat(T, ".obj");**  **strcpy(fileobj, T);**  **fp\_obj = fopen(T, "w");**  **j = 0;**  **if(!strcmp(list[0][j]->mnmn, "START")){**  **fprintf(fp\_obj, "H%-6s%06X%06X\n", list[0][j]->label, start\_address[0], program\_length[0]);**  **j++;**  **}**  **while(1){**  **first\_address = list[0][j]->location;**  **last\_address = list[0][j]->location + 29;**  **first\_index = j;**  **for(x = 0, temp\_address = first\_address; temp\_address <= last\_address; j++){**  **if(!strcmp(list[0][j]->mnmn, "END")) break;**  **else if(strcmp(list[0][j]->mnmn, "RESB") && strcmp(list[0][j]->mnmn, "RESW") && strcmp(list[0][j]->mnmn, "BASE")){**  **temp\_obj[x] = list[0][j]->OBJ;**  **strcpy(temp\_operator[x], list[0][j]->mnmn);**  **strcpy(temp\_operand[x], list[0][j]->operand);**  **last\_index = j + 1;**  **x++;**  **}**    **temp\_address = list[0][j+1]->location;**  **if(searchOPTAB(list[0][j+1]->mnmn)){**  **if(readprefix(list[0][j+1]->mnmn)) temp\_address++;**  **if(!strcmp(optab.format, "1")) temp\_format = 1;**  **if(!strcmp(optab.format, "2")) temp\_format = 2;**  **if(!strcmp(optab.format, "3/4")) temp\_format = 3;**  **temp\_address += temp\_format;**  **}**  **else{**  **if(!strcmp(list[0][j+1]->mnmn, "WORD") || !strcmp(list[0][j+1]->mnmn, "BYTE")){**  **if(IsFloatNum(list[0][j+1]->mnmn)) temp\_address += 6;**  **else if(!strcmp(list[0][j+1]->mnmn, "BYTE")) temp\_address += findlength(list[0][j+1]->mnmn);**  **else if(!strcmp(list[0][j+1]->mnmn, "WORD")) temp\_address += 3;**  **}**  **}**  **}**  **if((list[0][last\_index]->location - list[0][first\_index]->location) == 0){**  **if(!strcmp(list[0][j]->mnmn, "END")) break;**  **else continue;**  **}**  **fprintf(fp\_obj, "T%06X%02X", first\_address, (list[0][last\_index]->location - list[0][first\_index]->location));**  **for(y = 0; y < x; y++){**  **if(strcmp(temp\_operator[y], "BYTE") == 0) {**  **if(temp\_operand[y][0] == 'X') fprintf(fp\_obj, "%02X", temp\_obj[y]);**  **else if(IsFloatNum(temp\_operand[y])) fprintf(fp\_obj, "%012X", temp\_obj[y]);**  **else fprintf(fp\_obj, "%06X", temp\_obj[y]);**  **}**  **else{**  **if(searchOPTAB(temp\_operator[y])){**  **if(!strcmp(optab.format, "3/4")){**  **if(readprefix(temp\_operator[y])) fprintf(fp\_obj, "%08X", temp\_obj[y]);**  **else fprintf(fp\_obj, "%06X", temp\_obj[y]);**  **}**  **else if(!strcmp(optab.format, "2")) fprintf(fp\_obj, "%04X", temp\_obj[y]);**  **else if(!strcmp(optab.format, "1")) fprintf(fp\_obj, "%02X", temp\_obj[y]);**  **}**  **else{**  **if(IsFloatNum(temp\_operand[y])) fprintf(fp\_obj, "%012X", temp\_obj[y]);**  **else fprintf(fp\_obj, "%06X", temp\_obj[y]);**  **}**  **}**  **}**  **fprintf(fp\_obj, "\n");**  **if(!strcmp(list[0][j]->mnmn, "END")) break;**  **}**  **// Modification**  **for(j = 0; j < relocation\_counter[0]; j++) fprintf(fp\_obj, "M%06X%02X\n", Rlist[0][j].address, Rlist[0][j].length);**  **fprintf(fp\_obj, "E");**  **if(searchSYMTAB(end\_operand, 0)) fprintf(fp\_obj, "%06X\n", symboltb[0][SYMidx].address);**  **else fprintf(fp\_obj, "\n");**  **fclose(fp\_obj);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : relocation()**  **Purpose : To record the relocation**  **Return value : none**  **\*/**  **/\*------------------------------------------------\*/**  **void relocation(char\* mnmn, int location, int index){**    **Rlist[index][relocation\_counter[index]].address = location + 1;**  **Rlist[index][relocation\_counter[index]].length = 3;**  **if(readprefix(mnmn)) Rlist[index][relocation\_counter[index]].length += 2; // format 4**  **relocation\_counter[index]++;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : assemble()**  **Purpose : To create listing file and object file from reading asm file**  **Return value : 1 or 0**  **\*/**  **/\*------------------------------------------------\*/**  **int assemble(char \*AA){**  **FILE \*fp4;**  **char label[32], opcode[32], operand[32];**  **int IsComment = 0, IsEmpty = 0, IsStart = 0, LINE\_count = 0;**  **int loc = 0, temp\_format = 0;**    **fp4 = fopen(AA, "r");**  **if(fp4 == NULL) {**  **printf("File open error!\n");**  **return 0;**  **}**  **if(IsAssembled == 1){**  **for(int i = 0; i < SYMTAB\_counter[0]; i++){**  **symboltb[0][i].label[0] = '\0';**  **symboltb[0][i].address = 0;**  **}**  **SYMTAB\_counter[0] = 0;**  **}**  **/\* Initialize \*/**  **INDX = 0;**  **SYMidx = 0;**  **for(int i = 0; i < 100; i ++){**  **LOCCTR[i] = 0;**  **}**  **LOCCTR\_counter = 0;**  **for(int i = 0; i < 10; i++){**  **SYMTAB\_counter[i] = 0;**  **start\_address[i] = 0;**  **program\_length[i] = 0;**  **list\_index[i] = 0;**  **relocation\_counter[i] = 0;**  **}**  **// PASS 1**  **while(fgets(BUFF, 200, fp4) != NULL){**  **list[0][list\_index[0]] = (Intermediate\*)malloc(sizeof(Intermediate));**    **IsEmpty = strlen(BUFF);**  **BUFF\_length = strlen(BUFF);**  **INDX = 0;**  **strcpy(label, readline());**  **if(label[0] == '.') IsComment = 1;**  **else IsComment = 0;**  **if(IsEmpty > 1 && IsComment == 0){ // this is not a comment line then**  **INDX = 0;**  **list[0][list\_index[0]]->line = \*list\_index;**  **strcpy(label, readline());**  **strcpy(list[0][list\_index[0]]->label, label);**  **skip();**  **if(LINE\_count == IsStart){ // this is a first line**  **strcpy(opcode, readmnmn());**  **strcpy(list[0][list\_index[0]]->mnmn, opcode);**  **if(!strcmp(opcode, "START")){**  **skip();**  **strcpy(operand, readopnd());**  **strcpy(list[0][list\_index[0]]->operand, operand);**  **LOCCTR[LOCCTR\_counter] = StrToHex(operand);**  **start\_address[0] = LOCCTR[LOCCTR\_counter];**  **}**  **else{**  **LOCCTR[LOCCTR\_counter] = 0;**  **start\_address[0] = LOCCTR[LOCCTR\_counter];**  **}**  **}**  **else{ // not a first line**  **strcpy(opcode, readmnmn());**  **strcpy(list[0][list\_index[0]]->mnmn, opcode);**  **skip();**  **strcpy(operand, readopnd());**    **strcpy(list[0][list\_index[0]]->operand, operand);**  **if(strcmp(opcode, "END")){ // this is not END**  **if(label[0] != '\0'){**  **if(searchSYMTAB(label, 0)){ // duplicate symbol**  **fclose(fp4);**  **printf("Debugging Error on line %d\n", 5\*(LINE\_count));**  **SYMidx = 0;**  **return 0;**  **}**  **else{**  **insertSYMTAB(label);**  **} // insert (LABEL, LOCCTR) into SYMTAB**  **}**  **if(searchOPTAB(opcode)){**    **if(strcmp(optab.format, "3/4") == 0) temp\_format = 3;**  **if(strcmp(optab.format, "2") == 0) temp\_format = 2;**  **if(strcmp(optab.format, "1") == 0) temp\_format = 1;**  **LOCCTR[LOCCTR\_counter] = loc + temp\_format;**  **if(readprefix(opcode)) LOCCTR[LOCCTR\_counter] += 1;**  **}**  **else if(!strcmp(opcode, "WORD")){**  **if(IsFloatNum(operand)) LOCCTR[LOCCTR\_counter] = loc + 6;**  **else LOCCTR[LOCCTR\_counter] = loc + 3;**  **}**  **else if(!strcmp(opcode, "RESW")){**  **LOCCTR[LOCCTR\_counter] = loc + 3 \* StrToDec(operand);**  **}**  **else if(!strcmp(opcode, "RESB")){**  **LOCCTR[LOCCTR\_counter] = loc + StrToDec(operand);**  **}**  **else if(!strcmp(opcode, "BYTE")){**  **if(IsFloatNum(operand)) LOCCTR[LOCCTR\_counter] = loc + 6;**  **else LOCCTR[LOCCTR\_counter] = loc + findlength(operand);**  **}**  **else if(!strcmp(opcode, "BASE") || !strcmp(opcode, "NOBASE")) LOCCTR[LOCCTR\_counter] = loc;**  **else{**  **fclose(fp4);**  **printf("Invalid Operation Code at line %d\n", 5\*(LINE\_count));**  **return 0;**  **}**  **}**  **else strcpy(end\_operand, operand);**  **}**  **loc = LOCCTR[LOCCTR\_counter];**  **list[0][list\_index[0]]->location = LOCCTR[LOCCTR\_counter - 1];**  **LOCCTR\_counter++;**  **list\_index[0]++;**  **}**  **if(IsComment == 1) {**  **IsStart++;**  **}**    **LINE\_count++;**  **}**  **program\_length[0] = LOCCTR[LOCCTR\_counter - 2] - LOCCTR[0];**  **create\_SYMTAB();**    **// PASS 2**  **int disp = 0;**  **unsigned long obj, obj\_opcode, obj\_sign, obj\_relative, obj\_indexed, obj\_address, obj\_extended;**  **int obj\_format, reg\_index, IsBase, k;**  **unsigned short int temp;**  **char reg\_name[10];**  **char \*ch;**  **IsBase = -1;**  **disp = 0;**  **for(int j = 1; j < list\_index[0]; j++){**  **obj\_opcode = 0;**  **obj\_sign = 0;**  **obj\_relative = 0;**  **obj\_indexed = 0;**  **obj\_address = 0;**  **obj\_format = 0;**  **obj\_extended = 0;**  **reg\_name[0] = '\0';**  **strcpy(opcode, list[0][j]->mnmn);**  **if(searchOPTAB(opcode)){**  **if(!strcmp(optab.name, "RSUB")){**  **temp = strtoul(optab.code, &ch, 16);**  **list[0][j]->OBJ = ((temp+3) << 16);**  **continue;**  **}**  **obj\_opcode = strtoul(optab.code, &ch, 16);**  **if((strcmp(optab.format, "3/4") == 0)){**  **if(readprefix(opcode)){**  **obj\_format = 4;**  **obj\_extended = 0X00100000;**  **}**  **else obj\_format = 3;**  **}**  **if(strcmp(optab.format, "2") == 0) obj\_format = 2;**  **if(strcmp(optab.format, "1") == 0) obj\_format = 1;**  **obj\_opcode <<= (8\* (obj\_format - 1));**  **list[0][j]->OBJ = obj\_opcode;**  **strcpy(operand, list[0][j]->operand);**  **if(readprefix(operand)){**  **if(obj\_format <= 2){**  **fclose(fp4);**  **printf("Invalid Addressing mode at line %d\n", 5\*(list[0][j]->line));**  **return 0;**  **}**  **if(prefix == 2) obj\_sign = 0X010000;**  **else if(prefix == 3) obj\_sign = 0x020000;**  **obj\_sign <<= 8 \* (obj\_format - 3);**  **}**  **else if(obj\_format >= 3){**  **obj\_sign = 0x030000;**  **obj\_sign <<= 8 \* (obj\_format - 3);**  **}**  **if(obj\_format >= 3){**  **if(operand[strlen(operand) - 2] == ',' && operand[strlen(operand) - 1] == 'X'){ // indexed addressing**  **obj\_indexed = 0x008000;**  **obj\_indexed <<= 8 \* (obj\_format - 3);**  **operand[strlen(operand) - 2] = '\0';**  **}**  **if(searchSYMTAB(operand, 0)){**  **// search SYMTAB for operand**  **if(obj\_format == 4){**  **obj\_address = symboltb[0][SYMidx].address;**    **relocation(list[0][j]->mnmn, list[0][j]->location, 0);**  **}**  **else{ // relative : base or pc**  **disp = symboltb[0][SYMidx].address - list[0][j]->location - obj\_format;**  **if(disp >= -2048 && disp <= 2047){ // pc relatvie**  **obj\_address = 0x002000;**  **obj\_address += TwelveBit(disp, 3);**  **}**  **else{ // base relative**  **disp = symboltb[0][SYMidx].address - IsBase;**  **if(IsBase != -1 && disp >= 0 && disp <= 4095){**  **obj\_address = 0x004000;**  **obj\_address += disp;**  **}**  **else{// relatvie mode fail**  **fclose(fp4);**  **printf("Relative Mode Failed at line %d\n", 5\*(list[0][j]->line));**  **return 0;**  **}**  **}**  **}**  **}**  **else{ // symbol not found**  **readprefix(operand);**  **if(prefix == 2 && IsNum(operand)) obj\_address = TwelveBit(StrToDec(operand), (obj\_format == 4) ? 5 : 3);**  **else {**  **printf("Label doesn't exist at line %d\n", 5\*(list[0][j]->line));**  **fclose(fp4);**  **return 0;**  **}**  **}**  **}**  **else if(obj\_format == 2){ // format 2**  **k = 0;**  **reg\_index = 0;**  **do{**  **if(operand[k] == ',' || operand[k] =='\0'){**  **reg\_name[reg\_index] = '\0';**  **if(operand[k] == ',') obj\_address <<= 4;**    **if(searchREGTAB(reg\_name)) obj\_address += REGTAB[REGidx].number;**  **else{**  **if(!strcmp(optab.name, "SVC") || !strcmp(optab.name, "SHIFTL") || !strcmp(optab.name, "SHIFTR")){**  **if(IsNum(reg\_name)) obj\_address += StrToDec(reg\_name);**    **}**  **else{**  **fclose(fp4);**  **printf("Invalid Register at line %d\n", 5\*(list[0][j]->line));**  **return 0;**  **}**  **}**  **reg\_index = 0;**  **}**  **else{**  **reg\_name[reg\_index++] = operand[k];**  **}**  **} while (operand[k++] != '\0');**  **if(!strcmp(optab.name, "CLEAR") || !strcmp(optab.name, "TIXR") || !strcmp(optab.name, "SVC")) obj\_address <<= 4;**  **}**  **//Record of Object Code**  **obj = obj\_opcode + obj\_sign + obj\_indexed + obj\_relative + obj\_extended + obj\_address;**  **list[0][j]->OBJ = obj;**  **}**  **else if(!strcmp(opcode, "WORD")){ // convert constant to object code**  **strcpy(operand, list[0][j]->operand);**  **list[0][j]->OBJ = StrToHex(operand);**  **}**  **else if(!strcmp(opcode, "BYTE")){**  **strcpy(operand, list[0][j]->operand);**  **list[0][j]->OBJ = 0;**  **if(operand[0] == 'C' || operand[0] == 'c' && operand[1] == '\''){**  **for(int z = 2; z <= (int)(strlen(operand) - 2); z++){**  **list[0][j]->OBJ += (int)operand[z];**  **list[0][j]->OBJ <<= 8;**  **}**  **}**  **if(operand[0] == 'X' || operand[0] == 'x' && operand[1] == '\''){**  **char \*op\_ptr;**  **op\_ptr = &operand[2];**  **\*(op\_ptr + 2) = '\0';**  **for(int z = 2; z <= (int)(strlen(operand) - 2); z++){**  **list[0][j]->OBJ += StrToHex(op\_ptr);**  **list[0][j]->OBJ <<= 8;**  **}**  **}**  **list[0][j]->OBJ >>= 8;**  **}**  **else if (!strcmp(opcode, "BASE")){**  **strcpy(operand, list[0][j]->operand);**  **list[0][j]->OBJ = 0;**  **if(searchSYMTAB(operand, 0)) IsBase = symboltb[0][SYMidx].address;**  **else{**  **fclose(fp4);**  **printf("No Label is SYMTAB at line %d\n", 5\*(list[0][j]->line));**  **return 0;**  **}**  **}**  **else if (!strcmp(opcode, "NOBASE")){**  **list[0][j]->OBJ = 0;**  **IsBase = -1;**  **}**  **}**  **createLST(AA);**  **createOBJ(AA);**  **for(int j = 0; j < list\_index[0]; j++){**  **free(list[0][j]);**  **}**  **fclose(fp4);**  **printf("\noutput file: [%s], [%s]\n\n", filelst, fileobj);**    **return 1;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : loader()**  **Purposse : Function 'loader' command.**  **Return value : 0 or 1.**  **\*/**  **/\*------------------------------------------------\*/**  **int loader(unsigned char \* BUFFER, int file\_count){**  **FILE \*fp;**  **char name[10], symbol[10], buffer[200], first[2];**  **char temp\_address[10], temp\_length[10];**  **int es\_count = 0;**  **unsigned int add, len;**  **char \*cha;**  **int i = 0, index = 0, j=0, flag = 0;**    **// PASS 1**  **CSLTH = 0;**  **CSADDR = PROGADDR;**    **for(i = 0; i < file\_count; i++){**  **fp = fopen(ARR\_LOAD[i+1], "r");**  **buffer[0] = '\0';**    **if(fp == NULL){**  **printf("Error : There is no such file.\n");**  **return 0;**  **}**  **while(fgets(buffer, 200, fp) != NULL){**  **index = 0;**  **j = 0;**  **name[0] = '\0';**  **symbol[0] = '\0';**  **temp\_address[0] = '\0';**  **temp\_length[0] = '\0';**  **first[0] = buffer[0];**  **first[1] = '\0';**  **if(strcmp("H", first) == 0){**  **index++;**  **j = 0;**  **while(buffer[index] != ' ' && buffer[index] != '\t') name[j++] = buffer[index++];**  **name[j] = '\0';**  **strcpy(estab[es\_count].csect, name);**  **strcpy(estab[es\_count].symbol, " ");**  **while(buffer[index] == ' ' || buffer[index] == '\t') index++;**  **for(int x = 0; x < 6; x++){**  **temp\_address[x] = buffer[index];**  **index++;**  **temp\_address[x+1] = '\0';**  **}**  **add = strtoul(temp\_address, &cha, 16);**  **estab[es\_count].address = add + CSADDR;**  **for(int x = 0; x < 6; x++){**  **temp\_length[x] = buffer[index];**  **index++;**  **temp\_length[x+1] = '\0';**  **}**  **len = strtoul(temp\_length, &cha, 16);**  **estab[es\_count].length = len;**  **es\_count++;**  **}**  **else if(strcmp("D",first) == 0){**  **index++;**  **while(buffer[index] != '\n'){**  **symbol[0] = '\0';**  **temp\_address[0] = '\0';**  **j=0;**    **while(buffer[index] != ' ' && buffer[index] != '\t') symbol[j++] = buffer[index++];**  **symbol[j] = '\0';**  **strcpy(estab[es\_count].csect, " ");**  **strcpy(estab[es\_count].symbol, symbol);**  **while(buffer[index] == ' ' || buffer[index] == '\t') index++;**  **for(int x = 0; x < 6; x++){**  **temp\_address[x] = buffer[index];**  **index++;**  **temp\_address[x+1] = '\0';**  **}**  **add = strtoul(temp\_address, &cha, 16);**  **estab[es\_count].address = add + CSADDR;**  **estab[es\_count].length = 0;**  **es\_count++;**  **}**  **CSADDR = CSADDR + len;**  **}**  **}**  **fclose(fp);**  **}**    **// PASS 2**  **int loc, lent, l =0, k, specified\_add;**  **char temp\_refer[10], IsExtended[3], dummy\_memory[3], mod\_memory[10], fin\_memory[10];**  **unsigned int dummy;**  **int temp\_memory;**    **EXECADDR = PROGADDR;**  **CSADDR = PROGADDR;**  **for(i = 0; i < file\_count; i++){**  **fp = fopen(ARR\_LOAD[i+1], "r");**  **for(int x = 0; x < es\_count; x++) estab[x].refer[0] = '\0';**  **// initiallize**  **while(fgets(buffer, 200, fp) != NULL){**  **index = 0;**  **j = 0;**  **name[0] = '\0';**  **symbol[0] = '\0';**  **temp\_address[0] = '\0';**  **temp\_length[0] = '\0';**  **first[0] = buffer[0];**  **first[1] = '\0';**  **if(strcmp(first, "H") == 0){**  **index++;**  **while(buffer[index] != ' ' && buffer[index] != '\t') name[j++] = buffer[index++];**  **name[j] = '\0';**  **for(int x = 0; x < es\_count; x++){**  **if(strcmp(name, estab[x].csect) == 0){**  **loc = estab[x].address;**  **strcpy(estab[x].refer, "01");**  **break;**  **}**  **}**  **EXECADDR = loc;**  **}**  **else if(strcmp(first, "T") == 0){**    **index++;**  **for(int x = 0; x < 6; x++){**  **temp\_address[x] = buffer[index];**  **index++;**  **temp\_address[x+1] = '\0';**  **}**    **for(int x = 0; x < 2; x++){**  **temp\_length[x] = buffer[index];**  **index++;**  **temp\_length[x+1] = '\0';**  **}**  **dummy = strtoul(temp\_address, &cha, 16);**  **loc = EXECADDR + dummy;**  **dummy = strtoul(temp\_length, &cha, 16);**  **lent = dummy;**  **for(k = 0; k < (lent)\*2; k += 2){**  **for(int x = 0; x < 2; x++){**  **dummy\_memory[x] = buffer[index];**  **index++;**  **dummy\_memory[x+1] = '\0';**  **}**  **sscanf(dummy\_memory, "%02X", &temp\_memory);**  **BUFFER[(unsigned int)loc] = (unsigned char)(temp\_memory);**  **loc = loc + 1;**  **}**  **}**  **else if(strcmp(first, "R") == 0){**  **index++;**  **while(buffer[index] != '\n'){**  **j = 0;**  **for(int x = 0; x < 2; x++){**  **temp\_refer[j++] = buffer[index++];**  **temp\_refer[j+1] = '\0';**  **}**  **j = 0;**    **while(buffer[index] != ' ' && buffer[index] != '\t' && buffer[index] != '\n'){**  **symbol[j++] = buffer[index++];**  **symbol[j+1] = '\0';**  **}**  **while(buffer[index] == ' ' || buffer[index] == '\t') index++;**    **for(int x = 0; x < es\_count; x++){**  **if(strcmp(symbol, estab[x].symbol) == 0){**  **strcpy(estab[x].refer, temp\_refer);**  **flag = 1;**  **break;**  **}**  **}**  **if(flag == 0){**  **printf("Error : There is no corresponding symbol name.\n");**  **return 0;**  **}**  **flag = 0;**  **}**  **}**  **else if(strcmp(first, "M") == 0){**  **index++;**  **for(int x = 0; x < 6; x++){**  **temp\_address[x] = buffer[index];**  **index++;**  **temp\_address[x+1] = '\0';**  **}**  **IsExtended[0] = buffer[8];**  **IsExtended[1] = '\0';**  **if(strcmp(IsExtended, "5") == 0) lent = 8;**  **else lent = 6; // Format determination**  **index += 2;**  **dummy = strtoul(temp\_address, &cha, 16);**  **loc = EXECADDR + dummy;**  **if(buffer[index] == '\n'){ // if there's no link**  **sprintf(mod\_memory, "%02X%02X%02X%02X", BUFFER[(unsigned int)(loc-1)], BUFFER[(unsigned int)(loc)], BUFFER[(unsigned int)(loc+1)], BUFFER[(unsigned int)(loc+2)]);**  **dummy = strtoul(mod\_memory, &cha, 16);**  **k = dummy + PROGADDR;**  **sprintf(fin\_memory, "%08X", k);**  **if(strlen(fin\_memory) >= 9){**  **for(int x = 0; x < strlen(fin\_memory); x++){**  **fin\_memory[x] = fin\_memory[x+strlen(fin\_memory) - 8];**  **}**  **fin\_memory[8] = '\0';**  **}**  **j = 0;**  **for(k = 0; k < strlen(fin\_memory); k+=2){**  **for(int x = 0; x < 2; x++){**  **dummy\_memory[x] = fin\_memory[j];**  **j++;**  **dummy\_memory[x+1] = '\0';**  **}**  **sscanf(dummy\_memory, "%02X", &temp\_memory);**  **BUFFER[(unsigned int)(loc-1)] = (unsigned char)(temp\_memory);**  **}**  **}**  **else if(buffer[index] != '\n'){ // If there is link**  **while(buffer[index] != '\n') name[j++] = buffer[index++];**  **name[j] = '\0';**  **for(int x = 0; x < strlen(name); x++){**  **symbol[x] = name[x+1];**  **symbol[x+1] = '\0';**  **} // ex) name is +LISTC , symbol is LISTC or +02 and 02**  **for(int x = 0; x < es\_count; x++){**  **if((strcmp(symbol, estab[x].symbol) == 0) || (strcmp(symbol, estab[x].csect) == 0) || (strcmp(symbol, estab[x].refer) == 0)){**  **specified\_add = estab[x].address;**  **flag = 1;**  **break;**  **}**  **}**  **if(flag == 0){**  **printf("Error : Undefined external symbol.\n");**  **return 0;**  **}**  **flag = 0;**  **mod\_memory[0] = '\0';**  **if(name[0] == '+'){**    **if(lent == 6){**  **sprintf(mod\_memory, "%02X%02X%02X", BUFFER[(unsigned int)loc], BUFFER[(unsigned int)(loc+1)], BUFFER[(unsigned int)(loc+2)]);**  **dummy = strtoul(mod\_memory, &cha, 16);**  **k = dummy + specified\_add;**  **sprintf(fin\_memory, "%06X", k);**  **if(strlen(fin\_memory) >= 7){**  **for(int x = 0; x < strlen(fin\_memory); x++){**    **fin\_memory[x] = fin\_memory[x+strlen(fin\_memory)-6];**  **}**  **fin\_memory[6] = '\0';**  **}**  **}**  **else{**  **sprintf(mod\_memory, "%02X%02X%02X%02X", BUFFER[(unsigned int)(loc-1)], BUFFER[(unsigned int)(loc)], BUFFER[(unsigned int)(loc+1)], BUFFER[(unsigned int)(loc+2)]);**  **dummy = strtoul(mod\_memory, &cha, 16);**  **k = dummy + specified\_add;**  **sprintf(fin\_memory, "%08X", k);**  **if(strlen(fin\_memory) >= 9){**    **for(int x = 0; x < strlen(fin\_memory); x++){**  **fin\_memory[x] = fin\_memory[x+strlen(fin\_memory) - 8];**  **}**    **fin\_memory[8] = '\0';**  **}**    **}**  **j = 0;**  **for(k = 0; k < strlen(fin\_memory); k+=2){**  **for(int x = 0; x < 2; x++){**  **dummy\_memory[x] = fin\_memory[j];**  **j++;**  **dummy\_memory[x+1] = '\0';**  **}**  **sscanf(dummy\_memory, "%02X", &temp\_memory);**  **if(strlen(fin\_memory) != 6) BUFFER[(unsigned int)(loc-1)] = (unsigned char)(temp\_memory);**  **else BUFFER[(unsigned int)(loc)] = (unsigned char)(temp\_memory);**  **loc = loc + 1;**  **}**  **}**  **else if(name[0] == '-'){**  **if(lent == 6){**  **sprintf(mod\_memory, "%02X%02X%02X", BUFFER[(unsigned int)(loc)], BUFFER[(unsigned int)(loc+1)], BUFFER[(unsigned int)(loc+2)]);**  **dummy = strtoul(mod\_memory, &cha, 16);**  **k = dummy - specified\_add;**  **sprintf(fin\_memory, "%06X", k);**  **if(strlen(fin\_memory) >= 7){**    **for(int x = 0; x < strlen(fin\_memory); x++){**  **fin\_memory[x] = fin\_memory[x+strlen(fin\_memory) - 6];**  **}**  **fin\_memory[6] = '\0';**  **}**  **}**  **else{**  **sprintf(mod\_memory, "%02X%02X%02X%02X", BUFFER[(unsigned int)(loc-1)], BUFFER[(unsigned int)(loc)], BUFFER[(unsigned int)(loc+1)], BUFFER[(unsigned int)(loc+2)]);**  **dummy = strtoul(mod\_memory, &cha, 16);**  **k = dummy - specified\_add;**  **sprintf(fin\_memory, "%08X", k);**  **if(strlen(fin\_memory) >= 9){**  **for(int x = 0; x < strlen(fin\_memory); x++){**  **fin\_memory[x] = fin\_memory[x+strlen(fin\_memory)-8];**  **}**  **fin\_memory[8] = '\0';**  **}**  **}**  **j = 0;**  **for(k = 0; k < strlen(fin\_memory); k+=2){**  **for(int x = 0; x < 2; x++){**  **dummy\_memory[x] = fin\_memory[j];**  **j++;**  **dummy\_memory[x+1] = '\0';**  **}**  **sscanf(dummy\_memory, "%02X", &temp\_memory);**  **if(strlen(fin\_memory) != 6) BUFFER[(unsigned int)(loc-1)] = (unsigned char)(temp\_memory);**  **else BUFFER[(unsigned int)(loc)] = (unsigned char)(temp\_memory);**  **loc = loc + 1;**  **}**    **}**    **}**  **}**  **}**  **fclose(fp);**  **}**    **printf("\ncontrol symbol address length\n");**  **printf("section name\n");**  **printf("---------------------------------------------\n");**  **for(i = 0; i < es\_count; i++){**  **if(estab[i].length != 0){**  **printf("%-12s%-12s%04X %04X\n", estab[i].csect, estab[i].symbol, estab[i].address, estab[i].length);**  **}**  **else{**  **printf("%-12s%-12s%04X\n", estab[i].csect, estab[i].symbol, estab[i].address);**  **}**  **CSLTH += estab[i].length;**  **}**  **printf("---------------------------------------------\n");**  **printf(" total length %04X\n\n", CSLTH);**  **return 1;**  **}**  **/\*------------------------------------------------\*/**  **/\***  **Function : run()**  **Purposse : Function 'run' command.**  **Return value : 1 or 0**  **\*/**  **/\*------------------------------------------------\*/**  **int run(unsigned char\* BUFFER){**    **int n\_flag, i\_flag, x\_flag, b\_flag, p\_flag, e\_flag, reg1, reg2;**  **int format = 0, current =0, opcode = 0;**  **unsigned int loc = 0;**  **char temp\_memory[10];**  **int obj\_full, disp;**  **char \*ch;**  **char dummy[2];**  **int i, j, k;**    **REG[2] = PROGADDR + CSLTH;**  **loc = PROGADDR;**  **REG[9] = 0;**  **while(REG[8] != PROGADDR + CSLTH){**  **// opocde part!**  **// if(BRKPN[current\_bp] != '\0' && BRKPN[current\_bp] == loc) break;**    **current = BUFFER[loc];**  **opcode = current & 252;**  **for(i = 0; i < sizeof(RUNTAB); i++){**  **if(opcode == RUNTAB[i].code) format = RUNTAB[i].format;**  **} // format checking**  **if(format != 2){**  **loc++;**  **current = BUFFER[loc];**  **e\_flag = current & 0x001000;**  **if(e\_flag) format == 4; // format 4 checking**  **}**  **switch(format){**  **case 2 :**  **loc++;**  **sprintf(temp\_memory, "%02X", BUFFER[loc]);**  **dummy[0] = temp\_memory[0];**  **dummy[1] = '\0';**  **sscanf(dummy, "%d", &reg1);**  **dummy[0] = temp\_memory[1];**  **dummy[1] = '\0';**  **sscanf(dummy, "%d", &reg2);**    **loc++;**  **break;**  **case 3:**  **sprintf(temp\_memory, "%02X%02X%02X", BUFFER[loc-1], BUFFER[loc], BUFFER[loc+1]);**  **obj\_full = strtoul(temp\_memory, &ch, 16);**  **sprintf(temp\_memory, "%01X%02X", BUFFER[loc], BUFFER[loc+1]);**  **for(int x =0; x < 4; x++){**  **temp\_memory[x] = temp\_memory[x+1];**  **}**  **temp\_memory[4] = '\0';**  **disp = strtoul(temp\_memory, &ch, 16);**  **n\_flag = obj\_full & 0x020000;**  **i\_flag = obj\_full & 0x010000;**  **x\_flag = obj\_full & 0x008000;**  **b\_flag = obj\_full & 0x004000;**  **p\_flag = obj\_full & 0x002000; // flag checking**    **loc += 2;**    **break;**  **case 4:**  **sprintf(temp\_memory, "%02X%02X%02X%02X", BUFFER[loc-1], BUFFER[loc], BUFFER[loc+1], BUFFER[loc+2]);**  **obj\_full = strtoul(temp\_memory, &ch, 16);**  **sprintf(temp\_memory, "%02X%02X%02X", BUFFER[loc], BUFFER[loc+1], BUFFER[loc+2]);**  **for(int x = 0; x < 6; x++) temp\_memory[x] = temp\_memory[x+1];**  **temp\_memory[6] = '\0';**  **disp = strtoul(temp\_memory, &ch, 16);**  **n\_flag = obj\_full & 0x02000000;**  **i\_flag = obj\_full & 0x01000000;**  **x\_flag = obj\_full & 0x00800000;**  **b\_flag = obj\_full & 0x00400000;**  **p\_flag = obj\_full & 0x00200000;**  **loc += 4;**  **break;**  **}**  **REG[8] = loc;**  **if(opcode == 0x00){ // LDA**  **if(i\_flag) REG[0] = BUFFER[disp];**  **else REG[0] = BUFFER[BUFFER[disp]];**  **}**  **if(opcode == 0x68){ // LDB**  **if(i\_flag) REG[3] = BUFFER[disp];**  **else REG[3] = BUFFER[BUFFER[loc]];**  **}**  **if(opcode == 0x74){ // LDT**  **if(i\_flag) REG[5] = BUFFER[disp];**  **else REG[5] = BUFFER[BUFFER[loc]];**  **}**  **if(opcode == 0x0C){ // STA**  **disp = REG[0];**  **}**  **if(opcode == 0x14){ // STL**  **disp = REG[2];**  **}**  **if(opcode == 0x10){ // STX**  **disp = REG[1];**  **}**  **if(opcode == 0x3C){ // J**  **REG[8] = BUFFER[disp];**  **}**  **if(opcode == 0x48){ // JSUB**  **REG[2] = REG[8];**  **REG[8] = BUFFER[disp];**  **}**  **if(opcode == 0x30){ // JEQ**  **if(REG[9] ==0) REG[8] = BUFFER[disp];**  **}**  **if(opcode == 0x38){ // JLT**  **if(REG[9] < 0) REG[8] = BUFFER[disp];**  **}**    **if(opcode == 0x28){ // COMP**  **if(REG[reg1] > BUFFER[disp]) REG[9] = -REG[9];**  **if(REG[reg1] < BUFFER[disp]) REG[9] = -REG[9];**  **if(REG[reg1] == BUFFER[disp]) REG[9] = 0;**  **}**  **if(opcode == 0xB4) REG[reg1] = 0; //CLEAR**    **if(opcode == 0xA0){ // COMPR**  **if(REG[reg1] > REG[reg2]) REG[9] = -REG[9];**  **if(REG[reg1] < REG[reg2]) REG[9] = -REG[9];**  **if(REG[reg1] == REG[reg2]) REG[9] = 0;**  **}**  **if(opcode == 0x50){ // LDCH**  **REG[0] <<= 8 \* (BUFFER[disp]-1);**  **}**  **if(opcode == 0x54){ // STCH**  **disp <<= 8 \* (REG[0] -1);**  **}**  **if(opcode == 0xB8){ // TIXR**  **REG[1] = REG[1] + 1;**  **if(REG[1] < REG[reg1]) REG[9] = -REG[9];**  **else REG[9] = REG[9];**  **}**  **if(opcode == 0x4C){ // RSUB**  **REG[8] = REG[2];**  **}**  **if(opcode == 0xE0){ // TD**  **REG[9] = -1;**  **}**  **if(opcode == 0xD8){ // RD**  **REG[0] <<= 8 \* (0x00);**  **}**  **if(opcode == 0xDC){ // WD**  **}**  **}**  **REG[8] = PROGADDR + CSLTH;**  **printf("A : %06X X : %06X\nL : %06X PC: %06X\nB : %06X S : %06X\nT : %06X\nEnd Program\n\n", REG[0], REG[1], REG[2], REG[8], REG[3], REG[4], REG[5]);**  **if(BRKPN[current\_bp] != '\0'){**  **printf("Stop at checkpoint[%d]\n\n", BRKPN[current\_bp]);**  **current\_bp++;**  **}**  **return 1;**  **}**  **/\* Program Start \*/**  **int main(){**  **char input[MAX\_INPUT\_SIZE];**  **char \*TEMP\_IN = NULL, \*TOK1 = NULL, \*TOKEN=NULL, \*ORG\_IN=NULL, \*TOK2 = NULL, \*TOK3 = NULL, \*ORG = NULL;**  **char \*AA, \*BB, \*\*ARR\_FILL, \*INPUT\_MN, \*temp, \*O, \*ch;**  **int IsCommanded=0, IsFirst=0, IsHistory=0, IsOneNum=0;**  **int LAST\_ADDRESS = 0, LIMIT\_ADDRESS=0, EDIT\_ADDRESS = 0, EDIT\_MEMORY = 0, FILL\_START =0, FILL\_END =0, FILL\_MEMORY = 0;**  **int z;**  **DIR \*dp;**  **struct dirent \*file;**  **struct stat buf;**  **unsigned char\* BUFFER = (unsigned char\*)calloc(MEGA\_BYTE, sizeof(unsigned char));**  **dp = opendir(".");**  **struct NODE \*head = malloc(sizeof(struct NODE));**  **head->link = NULL;**  **NODE \*curr;**  **create\_hash();**  **while(1){**  **if(IsCommanded == 1){**  **if(IsFirst == 0){**  **InsertNode(head, input);**  **curr = head->link;**  **IsFirst = 1;**  **IsCommanded = 0;**  **}**  **else{**  **if(IsHistory == 0){**  **InsertNode(curr,input);**  **curr = curr->link;**  **curr->link = NULL;**  **}**  **IsCommanded = 0;**  **}**  **}**  **IsHistory = 0;**  **input[0] = '\0';**  **printf("sicsim> ");**  **scanf("%[^\n]%\*c", input);**  **input[strlen(input)] = '\0';**  **if(input[0] == 0){**  **getchar();**  **printf("You should input something. Please input again.\n");**  **continue;**  **}**  **AA = (char\*)malloc(sizeof(char)\*MAX\_ARR\_SIZE);**  **BB = (char\*)malloc(sizeof(char)\*MAX\_ARR\_SIZE);**  **ARR\_FILL = (char \*\*)malloc(sizeof(char\*)\*MAX\_ARR\_SIZE);**  **for(int i = 0; i < MAX\_ARR\_SIZE; i++){**  **ARR\_FILL[i] = (char\*)malloc(sizeof(char) \* strlen(input));**  **}**  **INPUT\_MN = (char\*)malloc(sizeof(char)\*MAX\_ARR\_SIZE);**  **ORG\_IN = (char\*)malloc(sizeof(char)\*strlen(input));**  **TEMP\_IN = (char\*)malloc(sizeof(char)\*strlen(input));**  **ORG = (char\*)malloc(sizeof(char)\*strlen(input));**  **O = (char\*)malloc(sizeof(char)\*MAX\_ARR\_SIZE);**  **temp = (char\*)malloc(sizeof(char)\*MAX\_ARR\_SIZE);**    **strcpy(TEMP\_IN, input);**  **strcpy(ORG\_IN, input);**  **strcpy(ORG, input);**  **strcpy(O, input);**  **TOKEN = strtok(TEMP\_IN, " \t");**    **if(strcmp(TOKEN, "q") == 0 || strcmp(TOKEN, "quit") == 0){**  **FreeNode(head);**  **FreeHash();**  **free(BUFFER);**  **free(AA);**  **free(BB);**  **free(TEMP\_IN);**  **free(ORG\_IN);**  **free(INPUT\_MN);**  **free(O);**  **free(ORG);**  **if(IsAssembled == 1){**  **FreeSHash();**  **}**  **break;**  **}**  **if(strcmp(TOKEN, "h") == 0 || strcmp(TOKEN, "help") == 0){**  **IsCommanded = 1;**  **help();**  **}**  **if(strcmp(TOKEN, "d") == 0 || strcmp(TOKEN, "dir") == 0){**  **IsCommanded = 1;**  **dir(dp,file,buf);**  **}**  **if(strcmp(TOKEN, "hi") == 0 || strcmp(TOKEN, "history") == 0){**  **if(IsFirst == 0){**  **InsertNode(head, input);**  **curr = head->link;**  **IsFirst = 1;**  **IsCommanded = 1;**  **IsHistory = 1;**  **history(head->link);**  **}**  **else{**  **InsertNode(curr, input);**  **curr = curr->link;**  **curr->link = NULL;**  **history(head->link);**  **IsHistory = 1;**  **IsCommanded = 1;**  **}**  **}**    **if(strcmp(TOKEN, "du") == 0 || strcmp(TOKEN, "dump") == 0){**  **IsOneNum = 0;**  **if(LAST\_ADDRESS > 0xfffff) LAST\_ADDRESS = 0;**  **TOK3 = strtok(ORG, " \t");**  **while(TOK3 != 0){**  **strcpy(temp, TOK3);**  **TOK3 = strtok(NULL, " \t");**  **}**  **if(strcmp(temp, "dump") == 0 || strcmp(temp, "du")==0 ){**  **dump(BUFFER, &LAST\_ADDRESS, &LIMIT\_ADDRESS);**  **LAST\_ADDRESS++;**  **IsCommanded = 1;**  **} // If it is just "dump " command.**  **else{**  **TOK1 = strtok(ORG\_IN, ",");**  **while(TOK1 != 0){**  **strcpy(BB, TOK1);**  **TOK1 = strtok(NULL,",");**  **}**  **if((strstr(BB, "dump") != NULL) || (strstr(BB, "du") != NULL)) IsOneNum = 1;**  **if(IsOneNum == 1){**  **TOK2 = strtok(BB, " \t");**  **while (TOK2 != 0){**  **strcpy(AA, TOK2);**  **TOK2 = strtok(NULL, " \t");**  **}**  **sscanf(AA, "%x", &LAST\_ADDRESS);**  **if(LAST\_ADDRESS > 0xfffff || LAST\_ADDRESS < 0x00000){**  **printf("Invalid Input!\n");**  **continue;**  **}**  **dump(BUFFER, &LAST\_ADDRESS, &LIMIT\_ADDRESS);**  **LAST\_ADDRESS++;**  **IsCommanded = 1;**  **} // If it is ' dump AA ' command.**  **else{**  **TOK2 = strtok(ORG\_IN, " \t");**  **while(TOK2 != 0){**  **strcpy(AA, TOK2);**  **TOK2 = strtok(NULL, " \t");**  **}**  **sscanf(AA, "%x", &LAST\_ADDRESS);**  **sscanf(BB, "%x", &LIMIT\_ADDRESS);**  **if((LAST\_ADDRESS > 0xfffff ) || (LAST\_ADDRESS < 0x00000 ) || (LIMIT\_ADDRESS > 0xfffff) || (LIMIT\_ADDRESS < 0x00000) || (LAST\_ADDRESS >= LIMIT\_ADDRESS)){**  **//Range Error**  **printf("Invalid Input!\n");**  **continue;**  **}**  **dump(BUFFER, &LAST\_ADDRESS, &LIMIT\_ADDRESS);**  **LAST\_ADDRESS++;**  **IsCommanded = 1;**  **LIMIT\_ADDRESS = 0;**  **} // If it is ' dump AA , BB ' command.**  **}**  **}**  **if(strcmp(TOKEN, "edit") == 0 || strcmp(TOKEN, "e") == 0){**  **if(strcmp(input, "edit") == 0 || strcmp(input, "e") == 0){**  **IsCommanded = 0;**  **printf("Invalid Input!\n");**  **continue;**  **}**  **else{**  **TOK1 = strtok(ORG\_IN, ",");**  **while(TOK1 != 0){**  **strcpy(BB, TOK1);**  **TOK1 = strtok(NULL, ",");**  **}**  **TOK2 = strtok(ORG\_IN, " \t");**  **while(TOK2 != 0){**  **strcpy(AA, TOK2);**  **TOK2 = strtok(NULL, " \t");**  **}**  **sscanf(AA, "%x", &EDIT\_ADDRESS);**  **sscanf(BB, "%x", &EDIT\_MEMORY);**  **if((EDIT\_ADDRESS > 0xfffff) || (EDIT\_ADDRESS < 0x00000) || (EDIT\_MEMORY < 0x00) || (EDIT\_MEMORY > 0xff)){**  **printf("Invalid Input!\n");**  **continue;**  **}**  **edit(BUFFER, &EDIT\_ADDRESS, &EDIT\_MEMORY);**  **IsCommanded = 1;**  **}**  **}**  **if(strcmp(TOKEN, "f") == 0 || strcmp(TOKEN, "fill") == 0){**  **int j = 0;**  **TOK1 = strtok(ORG\_IN, ",");**  **while(TOK1 != NULL){**  **strcpy(ARR\_FILL[j], TOK1);**  **TOK1 = strtok(NULL, ",");**  **j++;**  **}**  **TOK2 = strtok(ARR\_FILL[0], " \t");**  **while(TOK2 != 0){**  **strcpy(AA, TOK2);**  **TOK2 = strtok(NULL, " \t");**  **}**  **sscanf(AA, "%x", &FILL\_START);**  **sscanf(ARR\_FILL[1], "%x", &FILL\_END);**  **sscanf(ARR\_FILL[2], "%x", &FILL\_MEMORY);**  **if((FILL\_START >= FILL\_END) || (FILL\_START > 0xfffff) || (FILL\_START < 0x00000) || (FILL\_END > 0xfffff) || (FILL\_END < 0x00000) || (FILL\_MEMORY < 0x00) || (FILL\_MEMORY > 0xff)){**  **printf("Invalid Input!\n");**  **continue;**  **}**  **fill(BUFFER, &FILL\_START, &FILL\_END, &FILL\_MEMORY);**  **IsCommanded = 1;**  **}**  **if(strcmp(TOKEN, "reset") == 0){**    **BUFFER = (unsigned char\*)calloc(MEGA\_BYTE, sizeof(unsigned char));**  **IsCommanded = 1;**  **}**  **if(strcmp(TOKEN, "opcodelist") == 0){**  **opcodelist();**  **IsCommanded = 1;**  **}**    **if(strcmp(TOKEN, "opcode") == 0){**  **TOK1 = strtok(ORG\_IN, " \t");**  **while(TOK1 != 0){**  **strcpy(INPUT\_MN, TOK1);**  **TOK1 = strtok(NULL, " \t");**  **}**  **IsCommanded = find\_opcode(INPUT\_MN);**  **}**  **if(strcmp(TOKEN, "type") == 0){**    **TOK1 = strtok(ORG\_IN, " \t");**  **while(TOK1 != 0){**  **strcpy(AA, TOK1);**  **TOK1 = strtok(NULL, " \t");**  **}**  **IsCommanded = type(dp, file, buf, AA);**    **}**  **if(strcmp(TOKEN, "assemble") == 0){**  **TOK1 = strtok(ORG\_IN, " \t");**  **while(TOK1 != 0){**  **strcpy(AA, TOK1);**  **TOK1 = strtok(NULL, " \t");**  **}**  **IsCommanded = assemble(AA);**  **IsAssembled = IsCommanded;**  **}**    **if(strcmp(TOKEN, "symbol") == 0){**  **if(IsAssembled == 1){**  **symbol();**  **IsCommanded = 1;**  **}**  **else{**  **printf("There is no symbol table.\n");**  **IsCommanded = 0;**  **}**  **}**  **if(strcmp(TOKEN, "progaddr") == 0){**  **char \*ch;**  **TOK1 = strtok(ORG\_IN, " \t");**  **while(TOK1 != 0){**  **strcpy(AA, TOK1);**  **TOK1 = strtok(NULL, " \t");**  **}**  **PROGADDR = strtoul(AA, &ch, 16);**  **IsCommanded = 1;**  **printf("\nProgram starting address set to 0x%X.\n\n", PROGADDR);**  **}**  **if(strcmp(TOKEN, "loader") == 0){**  **z = 0;**  **for(int i = 0; i < 5; i++){**  **ARR\_LOAD[i] = (const char\*)malloc(sizeof(const char) \* MAX\_ARR\_SIZE);**  **}**  **TOK1 = strtok(ORG\_IN, " \t");**  **while(TOK1 != NULL){**  **ARR\_LOAD[z] = TOK1;**  **TOK1 = strtok(NULL, " \t");**  **z++;**  **}**  **z--;**    **IsCommanded = loader(BUFFER, z);**  **}**  **if(strcmp(TOKEN, "bp") == 0){**  **TOK3 = strtok(ORG, " \t");**  **while(TOK3 != 0){**  **strcpy(temp, TOK3);**  **TOK3 = strtok(NULL, " \t");**  **}**  **if(strcmp(temp, "bp") == 0){**  **if(bp\_count == 0) printf("\nno breakpoints set.\n\n");**  **else{**  **printf("\nbreakpoints\n----------\n");**  **for(int i = 0 ; i < bp\_count; i++) printf("%d\n", BRKPN[i]);**  **printf("\n");**  **}**  **IsCommanded = 1;**  **}**  **else{**  **TOK1 = strtok(ORG\_IN, " \t");**  **while(TOK1 != 0){**  **strcpy(BB, TOK1);**  **TOK1 = strtok(NULL," \t");**  **}**  **TOK2 = strtok(BB, " \t");**  **while (TOK2 != 0){**  **strcpy(AA, TOK2);**  **TOK2 = strtok(NULL, " \t");**  **}**  **if(strcmp(AA, "clear") == 0){**  **// bp clear**  **for(int i =0; i < MAX\_INPUT\_SIZE; i++)BRKPN[i] = 0;**  **bp\_count = 0;**  **printf("\n[ok] clear all breakpoints\n\n");**  **}**  **else{ // bp 4000**  **BRKPN[bp\_count] = strtoul(AA, &ch, 10);**  **bp\_count++;**  **printf("\n[ok] create breakpoint %d\n\n", BRKPN[bp\_count-1]);**  **}**  **IsCommanded = 1;**  **}**  **}**  **if(strcmp(TOKEN, "run") == 0){**  **IsCommanded = run(BUFFER);**  **}**  **if(IsCommanded == 0){**  **printf("There is no command or invalid input. Please input agian.\n");**  **continue;**  **}**    **}**  **return 0;**  **}** |