과목명: 시스템프로그래밍

담당교수명 : 소정민

<<Assignment 1>>

서강대학교 컴퓨터공학과

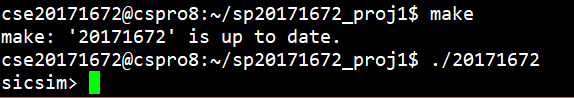
이정원20171672

목차

1. **프로그램 개요3**
2. **프로그램 설명6**
   1. 프로그램 흐름도 6
3. 모듈 정의**7**
   1. main() 7
   2. InsertNode() 8
   3. FreeNode() 8
   4. FreeHash() 8
   5. help() 8
   6. dir() 8
   7. history() 8
   8. dump() 9
   9. edit() 10
   10. fill() 10
   11. add\_hash() 10
   12. create\_hash() 10
   13. opcodelist() 11
   14. find\_opcode() 11
4. **전역 변수 정의12**
5. **코드12**
6. **프로그램 개요**

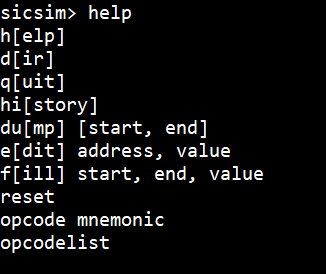
이 프로젝트에서는 SIC/XE machine을 구현하는 것을 목적으로 한다. 사용자의 입력을 받아 해당하는 명령을 수행하고, 메모리와 opcode 에 관련한 것들에 관여한다.

사용자가 프로그램을 실행했을 경우, 다음과 같이 쉘 프롬프트가 나타난다.

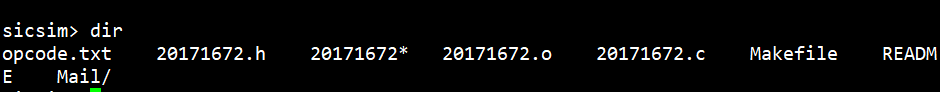


수행할 수 있는 기능들은 다음과 같다.

* **shell related commands : help, dir, quit, history**
* h[elp]: 사용자에게 어떠한 명령들이 있는지 보여 준다.



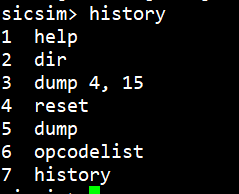
* d[ir]: 사용자의 현재 디렉토리에 있는 파일들과 상태를 보여 준다.



* q[uit]: 프로그램을 종료한다.

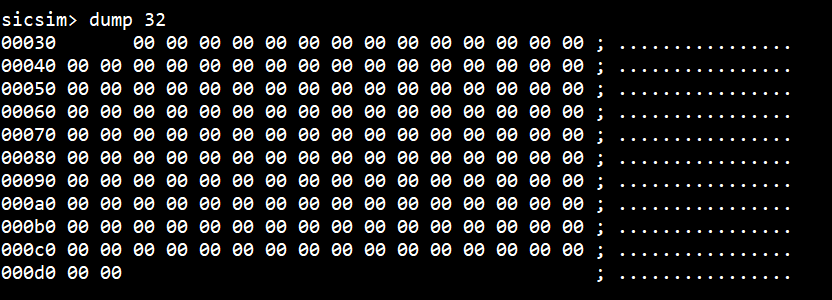


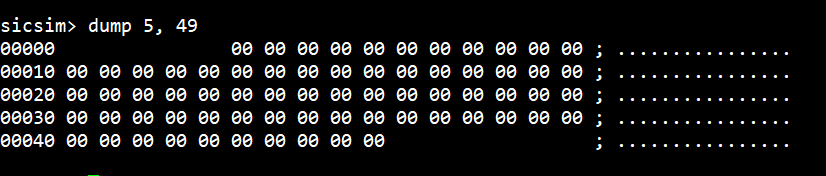
* hi[story] : 현재까지 실행했던 명령들을 나열하여 보여 준다.



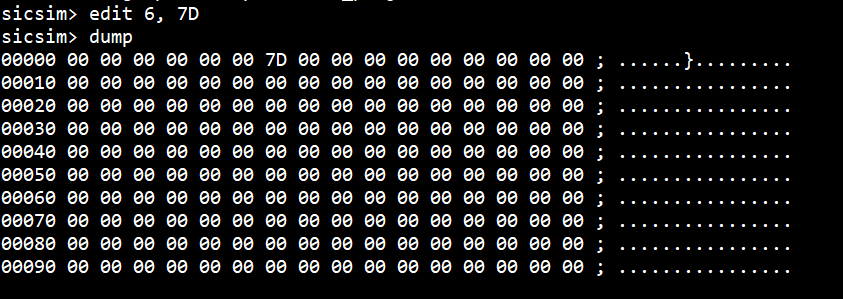
* **memory related commands : dump, edit, fill, reset**
* du[mp] [start,end] : 해당 주소 메모리 공간의 내용을 출력한다.



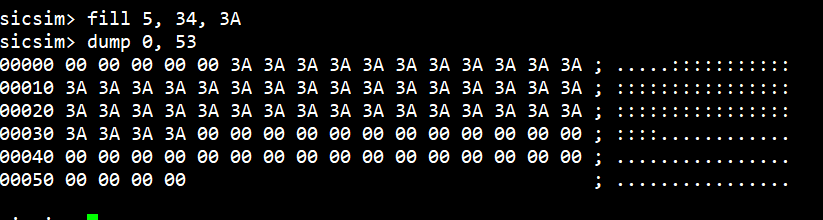




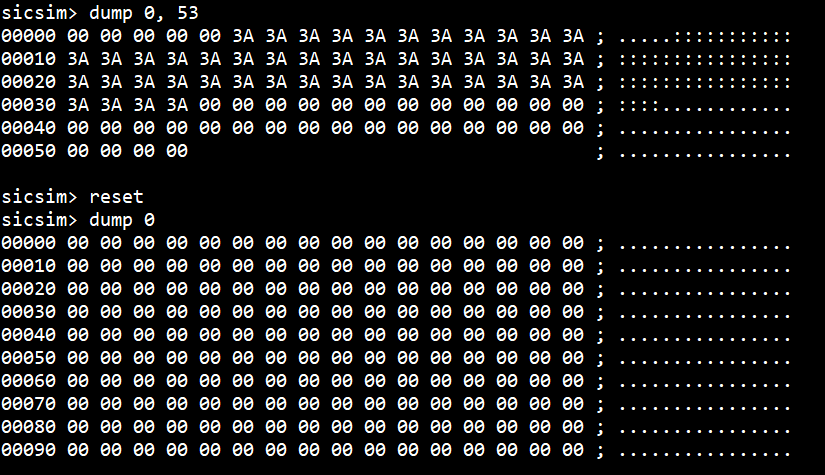
* e[dit] address, value : 입력받은 주소에 입력받은 값을 메모리에 넣는다.



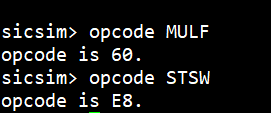
* f[ill] start, end, value : 입력받은 시작 주소와 끝 주소에 입력받은 값으로 채운다.



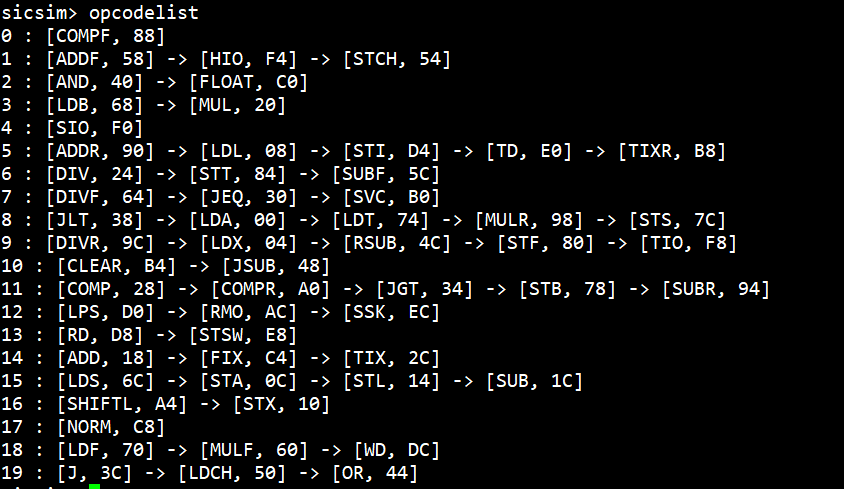
* reset : 메모리를 모두 0으로 초기화한다.



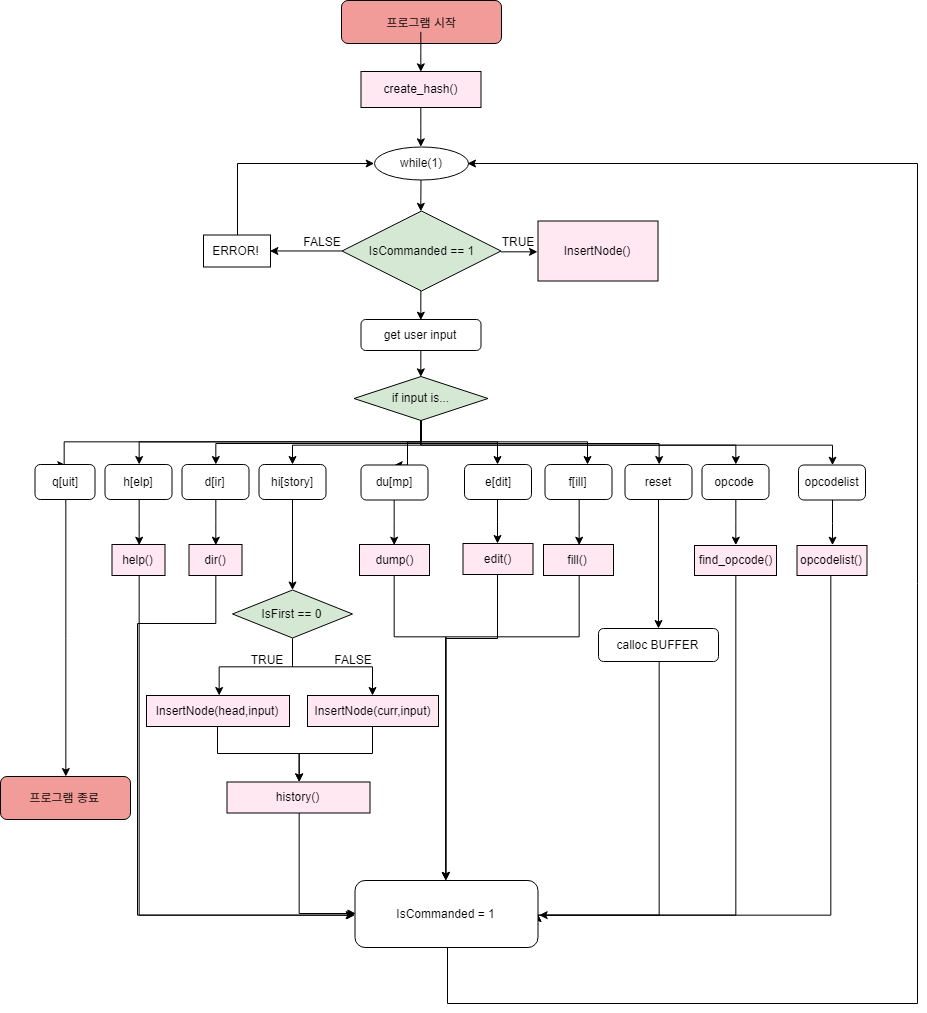
* **opcode related commands : opcode, opcodelist**
* opcode mnemonic : 입력받은 mnemonic 에 상응하는 opcode를 보여 준다.



* opcodelist : 해시 테이블을 이용하여 opcodelist를 보여 준다.



1. **프로그램 설명**
   1. 프로그램 흐름도



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. |
| 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. **InsertNode()**

|  |  |
| --- | --- |
| 기능 | 사용자에게 입력받은 유효한 명령을 연결 리스트에 저장하는 역할을 한다. |
| 사용 변수 | **struct NODE \*new**  ⇒ 사용자의 새 유효한 입력에 대한 정보를 저장하고 전체 노드에 삽입된다. |

* 1. **FreeNode()**

|  |  |
| --- | --- |
| 기능 | NODE를 해제한다. |
| 사용 변수 | **struct NODE \*tmp**  ⇒ 현재 노드를 해제하는 역할을 한다. |

* 1. **FreeHash()**

|  |  |
| --- | --- |
| 기능 | HASH를 해제한다. |
| 사용 변수 | **struct NODE \*tmp**  ⇒ 현재 노드를 해제하는 역할을 한다. |

* 1. **help()**

|  |  |
| --- | --- |
| 기능 | ‘help’ 또는 ‘h’ 명령을 수행하여 명령들을 모두 출력한다. |
| 사용 변수 | 없음. |

* 1. **dir()**

|  |  |
| --- | --- |
| 기능 | ‘directory’ 또는 ‘dir’ 명령을 수행하 현재 디렉토리에 있는 파일들의 이름은 dirent 구조체를 이용하여 출력하는 함수이다. dirent.h에 저장되어 있는 함수인opendir(”.”)과 readdir()를 통해 현재 디렉토리를 열고 읽을 수 있다. 이름이 ‘.’ 이거나 ‘..’ 이면 출력하지 않고, 파일의 종류가 디렉토리라면 이름 옆에 ‘/’를 출력하고, 실행 파일이라면 옆에 ‘\*’를 붙인다. 함수가 끝나기 전에는 cloesdir()를 통해 디렉토리를 닫는다. |
| 사용 변수 | **DIR \*dp**  ⇒ 현 디렉토리 스트림을 열고 그것에 대해 반환한 포인터 변수  **struct dirent \*file**  ⇒ 파일의 이름에 접근하기 위한 구조체이다.  **struct stat buf**  ⇒ 파일의 상태에 접근하기 위한 구조체이다. |

* 1. **history()**

|  |  |
| --- | --- |
| 기능 | ‘history’ 혹은 ‘his’ 기능을 수행하여 유효한 명령들이 저장된 연결 리스트의 내용을 모두 출력하는 함수이다. |
| 사용 변수 | **int count**  ⇒ 내용을 출력할 때 몇 번째 인덱스인지 알려 준다. |

* 1. **dump()**

|  |  |
| --- | --- |
| 기능 | Argument가 없는 ‘dump’ 기능과 Argument가 하나인 ‘dump start’ 기능과 argument 가 두 개인 ‘dump start, end’ 기능을 수행한다.  main에서 할당해 준 배열을 2차원 배열에 저장하기 위해 NEW\_BUFFER을 1mbyte 만큼 동적 할당해 준 다음 시작 주소를 고려하여 BUFFER의 내용을 NEW\_BUFFER 에 넣어 준다. argument가 없거나 하나면 끝 주소가 0이므로 0이면 160byte만을 보여 주고, 그게 아니라면 끝 주소에서 시작 주소를 빼고 1을 더하여 LIMIT\_MEMORY에 저장한다.  while문이 돌아가는 동안 메모리의 주소와 메모리의 내용물들을 출력한다. 캐릭터로 출력할 때는 0x20 이내거나 0x7E를 넘어가면 ‘.’을 출력한다.  현재 (i,k) 주소가 0xfffff 넘어가거나, LIMIT\_MEMORY에 도달하면 끝내야 하는 신호인 END\_SIGN을 1로 저장한다. 특히, 끝나는 좌표가 k가 0이라면, 다음줄로 넘어가지 않기 위해 while문을 아예 break한다. (IsCompleteLine으로 구분)  while 문이 끝나면 이 함수에서 쓰인 2중 배열 NEW\_BUFFER를 free 해 주고 끝낸다. |
| 사용 변수 | **int MEMORY\_NUM**  ⇒ 출력하는 메모리 주소를 세기 위한 변수이다. (0으로 초기화)  **int LIMIT\_MEMORY**  ⇒ 출력하는 메모리의 수를 160으로 제한할 건지, 끝과 시작 주소를 통하여 어떤 특정한 수로 지정할 건지 저장하는 변수이다. (160으로 초기화)  **int START\_X**  ⇒ 입력받은 주소(또는 시작 주소 또는 이전 실행의 마지막 시작 주소)를 2차원 배열에 저장하기 위해 사용되는 x 좌표 변수이다. (시작 주소를 16으로 나눈 몫으로 지정)  **int START\_Y**  ⇒ 입력받은 주소(또는 시작 주소, 또는 이전 실행의 마지막 시작 주소)를 2차원 배열에 저장하기 위해 사용되는 y 좌표 변수이다. (시작 주소를 16으로 나눈 나머지로 지정)  **int Hex\_Num**  ⇒ 메모리 주소를 출력하기 위한 변수이다. (START\_X로 지정)  **int END\_SIGN**  ⇒ 그만 출력할 건지 0과 1로 구분하는 flag변수이다. (0으로 초기화)  **int IsCompleteLine**  ⇒ 출력을 멈추는 좌표가 줄의 시작이라면 다음 줄로 넘어가지 않게 하는 flag 변수이다. (0으로 초기화)  **unsigned char \*\*NEW\_BUFFER**  ⇒ main에서 할당해 준 배열 BUFFER의 메모리 내용을 저장하기 위한 2차원 배열이다.  **unsigned char \*BUFFER**  ⇒ main에서 할당해 준 메모리 배열  **int \*START\_ADDRESS**  ⇒ 사용자에게서 입력받은 시작 주소값  **int \*END\_ADDRESS**  ⇒ 사용자에게서 입력받은 끝 주소값 |

* 1. **edit()**

|  |  |
| --- | --- |
| 기능 | ‘edit’ 혹은 ‘e’ 기능을 수행하여 버퍼의 해당 주소에 입력받은 value를 넣는 함수이다. |
| 사용 변수 | **unsigned char \*BUFFER**  ⇒ main에서 할당해 준 메모리 배열  **int \*USER\_ADDRESS**  ⇒ 사용자에게서 입력받은 주소값  **int \*EDIT\_VALUE**  ⇒ 사용자에게서 입력받은 값 |

* 1. **fill()**

|  |  |
| --- | --- |
| 기능 | ‘fill’ 혹은 ‘f’ 기능을 수행하여 버퍼의 특정 범위의 주소에 입력받은 값으로 채우는 함수이다. |
| 사용 변수 | **unsigned char \*BUFFER**  ⇒ main에서 할당해 준 메모리 배열  **int \*START\_ADDRESS**  ⇒ 사용자에게서 입력받은 시작 주소값  **int \*END ADDRESS**  ⇒ 사용자에게서 입력받은 끝 주소값  **int \*FILL\_VALUE**  ⇒ 사용자에게서 입력받은 값 |

* 1. **add\_hash**

|  |  |
| --- | --- |
| 기능 | create\_hash() 함수 실행시 해당 새 노드를 원래 있던 특정 인덱스의 hashtable에 삽입하는 함수이다. 해당 인덱스의 hashtable에 아무것도 없다면 그냥 넣고, 그렇지 않으면 꼬리까지 따라가 해당 노드를 삽입한다. |
| 사용 변수 | **int key**  ⇒create\_hash()에서 지정된 인덱스 값  **HASH \* node**  ⇒ 삽입할 새 노드  **int index**  ⇒ key에서 HASHSIZE를 나눈 나머지 |

* 1. **create\_hash()**

|  |  |
| --- | --- |
| 기능 | “opcode.txt” 파일을 읽어와 해당 줄의 정보들을 해시 테이블에 저장하는 함수이다. 파일을 한 줄씩 읽으며 임시로 code와 name에 저장하고, HASH 노드를 새로 생성한 후 그 노드의 정보에 저장한 다음 add\_hash()를 호출한다. 해당 해시 테이블의 아이디는 rand()함수를 이용해 임의로 0부터 19의 수를 지정해준다.  함수를 끝내기 전에 동적할당한code와 name 배열을 free해 주고 fclose()를 이용해 파일을 닫는다. |
| 사용 변수 | **FILE \*fp**  ⇒ 불러올 파일의 포인터 변수  **char \*code**  ⇒ 파일의 opcode를 저장할 배열  **char \*name**  ⇒ 파일의 해당 코드의 mnemonic을 저장할 배열  **int MAX\_SIZE**  ⇒ 배열의 크기를 100으로 제한한다. |

* 1. **opcodelist()**

|  |  |
| --- | --- |
| 기능 | ‘opcodelist’의 기능을 수행하여 해당 인덱스의 해시 테이블에 저장된 내용들을 나열식으로 출력하는 함수이다.  해당 인덱스를 출력한 뒤, 해당 인덱스의 해시 테이블을 새로 선언한 노드에 넣은 다음, 노드의 link가 끝에 도달할 때까지 노드를 출력한다. |
| 사용 변수 | **HASH \*node**  ⇒ 해당 인덱스의 해시 테이블을 임시로 저장할 HASH 노드 |

* 1. **find\_opcode()**

|  |  |
| --- | --- |
| 기능 | ‘opcode mnemonic’ 기능을 수행하여 파일을 읽어와 해당 mnemonic에 대응하는 opcode를 찾아 출력하는 함수이다.  파일을 한 줄씩 읽을 때 입력받은 mnemonic과 읽어온 name이 같을 때 code를 출력하고, code와 name 을 free해 주고 파일을 닫은 다음 1을 반환한다. (성공적으로 찾았다는 뜻) 파일을 끝까지 읽고 못 찾았을 경우에 0을 반환한다. (찾지 못했다는 뜻) |
| 사용 변수 | **FILE \*fp**  ⇒ 파일을 읽을 때 필요한 포인터 변수  **char \*code**  ⇒ 파일의 opcode를 저장할 배열  **char \*name**  ⇒ 파일의 mnemonic을 저장할 배열  **int MAX\_SIZE**  ⇒ 배열의 크기를 100으로 제한한다 |

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

⇒ history 기능을 위해 사용자의 유효한 명령을 저장하는 연결 리스트이다.

* 1. **typedef struct HASH**

⇒ opcodelist 기능을 위해 “opcode.txt” 파일에서 읽은 내용을 저장하는 해시 테이블이다.

* 1. **struct HASH\* hashtable[HASHSIZE]**

⇒ opcodelist 기능을 위한 전체 해시 테이블이다. (사이즈는 20)

1. **코드**

|  |
| --- |
| **20171672.h** |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <dirent.h>  #include <sys/stat.h>  #include <time.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[10];  struct NODE\* link;  }NODE;  typedef struct HASH{  int id,count;  char code[10];  char name[10];  struct HASH\* next;  }HASH;  struct HASH\* hashtable[HASHSIZE]; |

|  |
| --- |
| **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){**  **HASH \*tmp = hashtable[i];**  **hashtable[i] = 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\n\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(char) \* 16);**  **}**  **for(i=START\_X; i < 11+START\_X; i++){**  **for(k = 0; k < 16; k++){**  **NEW\_BUFFER[i][k] = BUFFER[j + 16\*START\_X];**  **j++;**  **}**  **}**  **if(\*END\_ADDRESS != 0){**  **LIMIT\_MEMORY = \*END\_ADDRESS - \*START\_ADDRESS + 1; // LIMIT SETTING**  **}**  **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);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **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, HASH\* node){**  **int index = key % HASHSIZE;**  **HASH\* cur = (HASH\*)malloc(sizeof(HASH));**  **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 \*fp = fopen("opcode.txt", "r");**  **char \*code, \*name;**  **int MAX\_SIZE = 100;**  **srand(time(NULL));**  **if(fp == NULL){**  **printf("File Open Error!\n");**  **return;**  **}**  **while(!feof(fp)){**  **code = (char\*)malloc(sizeof(char)\*MAX\_SIZE);**  **name = (char\*)malloc(sizeof(char)\*MAX\_SIZE);**  **fscanf(fp, "%s%\*[ \t]%s%\*[ \t]%\*s\n", code, name);**  **HASH\* node = (HASH\*)malloc(sizeof(HASH));**  **node->id = rand()%20;**  **strcpy(node->code, code);**  **strcpy(node->name, name);**  **node->next = NULL;**  **add\_hash(node->id, node);**  **}**  **free(code);**  **free(name);**  **fclose(fp);**  **}**  **/\*------------------------------------------------\*/**  **/\***  **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){**  **HASH\* 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 \*fp = fopen("opcode.txt", "r");**  **char \*code, \*name;**  **int MAX\_SIZE = 100;**  **if(fp == NULL){**  **printf("File Open Error!\n");**  **return 0 ;**  **}**  **while(!feof(fp)){**  **code = (char\*)malloc(sizeof(char)\*MAX\_SIZE);**  **name = (char\*)malloc(sizeof(char)\*MAX\_SIZE);**  **fscanf(fp, "%s%\*[ \t]%s%\*[ \t]%\*s\n", code, name);**  **if(strcmp(name, INPUT\_MN) == 0){**  **printf("opcode is %s.\n", code);**  **free(code);**  **free(name);**  **fclose(fp);**  **return 1;**  **}**  **}**  **free(code);**  **free(name);**  **fclose(fp);**  **return 0;**  **}**  **/\* Program Start \*/**  **int main(){**  **char \*input = (char\*)malloc(sizeof(char)\*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;**  **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;**  **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;**  **}**  **}**  **printf("sicsim> ");**  **scanf("%[^\n]%\*c", input);**  **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(input);**  **free(BUFFER);**  **free(AA);**  **free(BB);**  **free(TEMP\_IN);**  **free(ORG\_IN);**  **free(INPUT\_MN);**  **free(O);**  **free(ORG);**  **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(IsCommanded == 0){**  **printf("There is no command or invalid input. Please input agian.\n");**  **continue;**  **}**    **}**  **return 0;**  **}** |