과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 2>>

**서강대학교 컴퓨터공학과**

**[20151619]**

**[최준수]**

목 차

1. 프로그램 개요 4

2. 프로그램 설명 4

2.1 프로그램 흐름도 4-5

3. 모듈 정의 6

3.1 모듈 이름 : void cmd\_type(char\*) 6

3.1.1 기능

3.1.2 사용 변수

3.2 모듈 이름: int cmd\_assemble(char\*) 6

3.2.1 기능

3.2.2 사용 변수

3.3 모듈이름: int addressingMode(char\*) 8

3.3.1 기능

3.3.2 사용변수

3.4 모듈이름: int symfunction(char\*)

3.4.1 기능

3.4.2 사용변수

3.5 모듈이름: int symtab\_push(symtab\*\*, char\* int)

3.5.1 기능

3.5.2 사용변수

3.6 모듈이름: void symtab\_print(symptr) 9

3.6.1 기능

3.6.2 사용변수

3.7 모듈이름: void symtab\_printAll()

3.7.1 기능

3.7.2 사용변수

3.8 모듈이름: int symtab\_search(symtab\*, char\*)

3.8.1 기능

3.8.2 사용변수

3.9 모듈이름: void interm\_push(intermptr\*, intermptr)

3.9.1 기능

3.9.2 사용변수

3.10 모듈이름: int StrToInt(char\*) 10

3.10.1 기능

3.10.2 사용변수

3.11 모듈이름: int StrToHex(char\*)

3.11.1 기능

3.11.2 사용변수

3.12 모듈이름: char\* extractContent(char\*)

3.12.1 기능

3.12.2 사용변수

3.13 모듈이름: int isDec(char\*)

3.13.1 기능

3.13.2 사용변수

3.14 모듈이름: int isHex(char\*) 11

3.14.1 기능

3.14.2 사용변수

3.15 모듈이름: int inRange(int,int,int)

3.15.1 기능

3.15.2 사용변수

3.16 모듈이름: int insertHexAt(int,int,int)

3.16.1 기능

3.16.2 사용변수

3.17 모듈이름: int registerNum(char\*)

3.17.1 기능

3.17.2 사용변수

3.18 모듈이름: int asmSeparater(char\*, char\*, char\*, char\*, char\*) 12

3.18.1 기능

3.18.2 사용변수

4. 전역 변수 정의 12

4.1 symptr\* symboltable

4.2 int newsymtable

5. 코드 13

# 프로그램 개요

Assembly 코드로 이루어진 파일을 읽어와 강의시간에 학습한 PASS1, PASS2를 거쳐 Listing File과 Object File을 생성해주는 프로그램을 만든다.

# 프로그램 설명

## 프로그램 흐름도

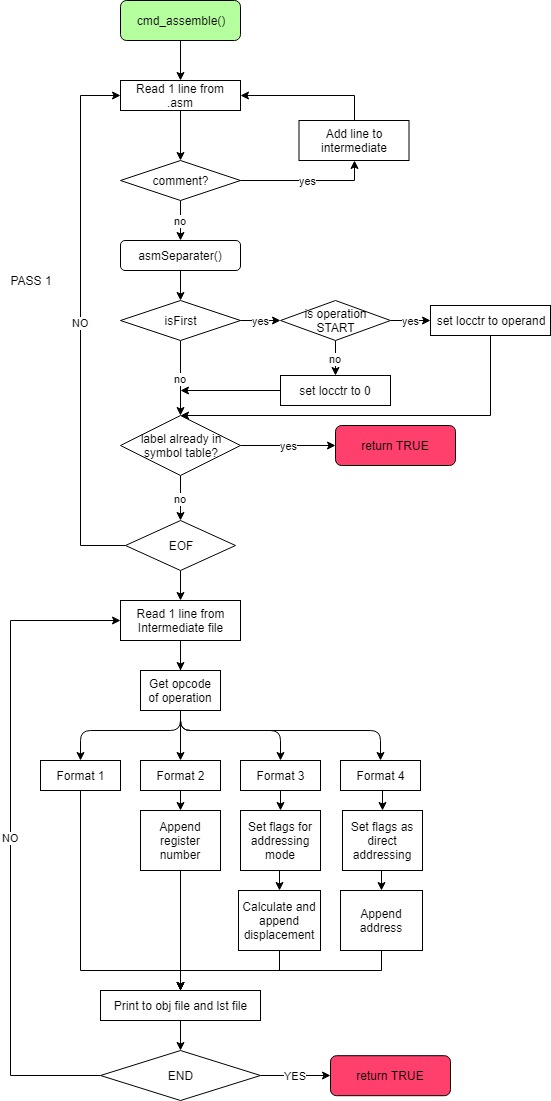
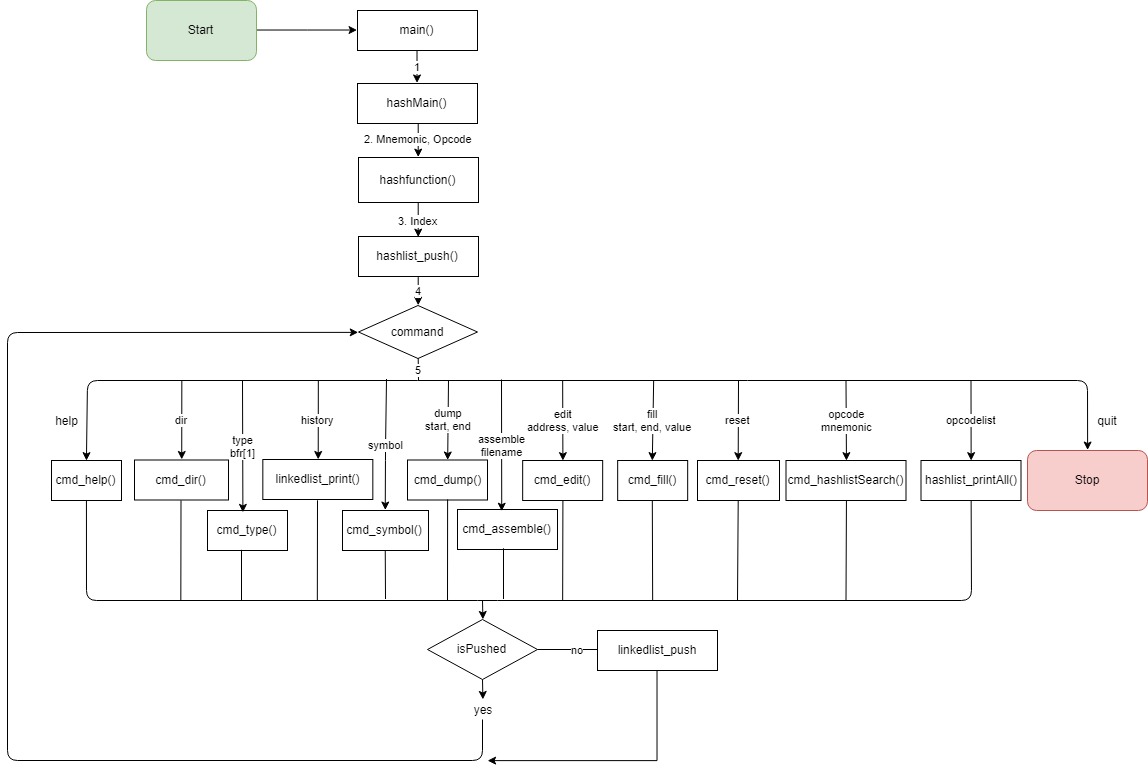


그림 1> 프로그램 흐름도

# 모듈 정의

## 모듈 이름 : void cmd\_type(char\* filename)

### 기능

전달받은 filename을 파일명으로 하는 파일을 한줄씩 출력해준다. 만약 해당 파일명이 존재하지 않을 경우 에러 메시지를 출력한다.

### 사용 변수

|  |  |
| --- | --- |
| char\* filename | 내용을 출력하고자 하는 파일의 이름 |

## 모듈 이름: int cmd\_assemble(char\* filename)

### 기능

이 모듈은 크게 두 부분으로 나뉘는데 첫 부분이 pass1, 뒷 부분이 pass2이다. 간단하게 정리하자면 pass1에서는 pass2에서 사용할 symbol들의 값(locctr)이 저장된 symbol table을 생성해주는 것이 주목적이고, pass2에서는 이 symbol table과 프로젝트1에서 만들었던 optable 등을 사용하여 실제 object code와 이를 사용한 listing file과 object file을 만드는 것이 목적이다.

pass1의 주목적은 앞에서 언급했다시피 symbol table을 만드는 것이다. 다만 이를 위해선 .asm파일에서 줄단위로 받아와 이를 parsing하여 label, directive, operand로 분리하는 과정이 필요하다. 이를 위해 뒤에서 설명될 asmSeperator()라는 함수를 사용하였다. 각각 분리가 된 후에는 label이 symbol table에 이미 들어있는지 확인하고 없을 경우 삽입한다. 이미 존재할 경우 errorFlag를 세팅하여 Assemble을 종료한다. pass1이 끝났을 때의 locctr를 totalLength라는 변수에 저장하여 pass2에서 header record를 작성할 때 사용할 수 있도록 한다. pass1은 온전히 pass2를 위한 사전작업으로서 문자열 parsing과 symbol table 생성을 위해 기능한다.

pass2에서는 pass1에서 parsing된 값들이 저장되어 있는 intermediate file을 input으로 받아 object code를 생성한다. directive의 opcode와 operand에 있는 상수값은 Hexadecimal로 변환하고, symbol이 왔을 경우 symbol table을 검색하여 해당 값을 찾아준다. 그리고 operand에 @ 혹은 #이 붙었는지에따라 indirect addressing과 immediate addressing에 맞는 flag를 세팅해준다. 또한 operand, X 형식으로 들어왓을 경우 indexed addressing에 해당하는 e 플래그를 세팅해주었다. 물론 올바르지 않은 입력에 대한 예외처리 또한 해주었다.

pass2에서 만들어지는 object file에서 Header record는 ‘H’로 시작하여 START의 symbol을 프로그램이름으로, 그리고 총 프로그램 길이를 저장한다. 다음줄에는 Text record들이 오는데, 다음 줄의 주소값, 그리고 해당 줄의 총 길이를 입력한뒤 START 다음의 assembly 코드들이 최대 69개의 Hexadecimal숫자들이 온다. 마지막으로 End record에는 start 주소값이 위치하여 프로그램이 종료된 뒤 사용될 수 있도록 한다. 만약 중간에 format4 형식이 들어왔다면 그 줄의 주소를 modification 배열에 저장해뒀다가 End record 입력전에 다 입력해준다.

### 사용변수

|  |  |
| --- | --- |
| char \*filename  FILE \*fp  char \*line  char \*line2  char \*label  char \*operand  char \*operation  int argCount  int base  int hashindex  int totalLength  int isFirst  int errorFlag  int opcode  int format  int locctr  int linecount  \*modification  int obj12  int obj3  int obj4  char \*strcontent | 내용을 출력하고자 하는 파일의 이름  Assemble할 filename을 파일명으로 가진 .asm파일의 포인터  PASS2에서 object코드에 기록할 내용  PASS2에서 line의 최대 입력범위를 넘어간 내용을 임시로 담고 있는 배열  현재 라인에서 parsing한 label. 없다면 NULL  현재 라인에서 parsing한 operand. 없다면 NULL  현재 라인에서 parsing한 directive.  현재의 라인을 label, operand, operation으로 분리시켰을 때 3개중 몇 개가 입력되었는지를 저장  LDB directive가 입력되었을 때 base의 값을 설정해주어 base relative 연산에 사용된다.  hashtable에서 몇번째 index에 저장되어있는지/저장해야되는지 판별을 위해 hashfunction이 리턴해주는 index를 저장하고 있다.  Object파일의 Header record끝에 프로그램 사이즈를 출력해주기 위해 저장하는 값  첫 라인을 읽을 때 FALSE로 세팅된다. 첫 라인일 때 START가 입력되었다면 START의 operand로 locctr를 세팅해주고, 없을 경우 0으로 세팅한다.  PASS1과 PASS2를 진행하면서 정상적이지 않은 입력을 마주치면 TRUE로 세팅되어 Assemble을 종료한다.  추출한 operation(디렉티브)의 opcode를 저장  추출한 operation(디렉티브)의 opcode를 저장  각 라인들의 format에 따라 증가하며 label의 값으로서 symbol table에 삽입된다.  Assemble하다가 error가 발생했을 경우 몇번째 라인에서 문제가 발생했는지 출력하기 위해 현재 읽고 있는 파일의 줄을 저장  Modification Record에 추가되어야하는 format4들을 저장  format3/4에서 opcode + ni플래그를 저장(즉, 2개의 Hexadecimal)  format3/4에서 xbpe 플래그를 저장(즉, 1개의 Hexadecimal)  format3/4에서 displacement를 저장(format3에서는 3개, format4에서는 5개의 Hexadecimal을 저장)  operand의 처음에 #이나 @같이 기호가 붙어있는경우 제외하고 실제 값만을 저장 |

## 모듈이름: int addressingMode(char\* str)

### 기능

전달받은 str의 제일 앞자리를 확인하여 #(immediate addressing)일경우 01(=1), @(indirect addressing)일 경우 10(=2), 아무것도 없을 경우 11(=3)을 반환한다.

### 사용변수

|  |  |
| --- | --- |
| char\* str | 한줄의 assembly 코드에서 operand를 넘겨받아 저장하고 있는 값 |

## 모듈이름: int symfunction(char\* str)

### 기능

전달받은 str의 제일 첫글자를 검사하여 A~Z를 0~25로 normalize시킨 값을 반환한다.

### 사용변수

|  |  |
| --- | --- |
| char\* str | symbol table에서의 index를 알고싶은 symbol |

## 모듈이름: int symtab\_push(symtab\*\* head, char\* label, int addr)

### 기능

head는 symbol table의 한 index이다. label과 그것의 주소값인 addr를 head부터 시작되는 linked list에 삽입해주는 함수이다. 나중에 symbol table을 출력할 때 내림차순으로 정렬하여 출력하여야 되기 때문에 이 함수로 push할때마다 올바른 자리에 들어가도록 한다. 노드를 하나씩 이동하며 같은 이름을 가진 symbol이 있는지, 없다면 이 label이 현재의 자리에 들어가면 되는지 체크한다.

### 사용변수

|  |  |
| --- | --- |
| symtab\*\* head  char\* label  int addr | label과 해당 주소값을 삽입하고자 하는 symbol table의 한 index의 linkedlist의 첫 노드  symbol table에 저장하고자 하는 label  symbol table에 저장하고자 하는 label의 주소값 |

## 모듈이름: void symtab\_print(symptr \*head)

### 기능

symbol table의 한 index에 위치한 노드의 값을 받아 그 노드부터 시작되는 linked list를 전부 출력한다.

### 사용변수

|  |  |
| --- | --- |
| symptr \*head | symbol table중 한 index에 위치한 노드 |

## 모듈이름: void symtab\_printAll()

### 기능

symbol table의 배열의 사이즈 만큼 순서대로 symtab\_print()를 호출하여 symbol table 전체를 출력해주는 함수이다.

## 모듈이름: int symtab\_search(symtab \*head, char \*label)

### 기능

head부터 시작해서 head에 연결된 linked list를 검색하여 label과 같은 값을 가진 노드가 있는지 비교한다. 없을 경우 -1을 반환한다.

### 사용변수

|  |  |
| --- | --- |
| symtab \*head  char \*label | symbol table중 한 index에 위치한 노드  값을 찾고자 하는 symbol의 이름 |

## 모듈이름: void interm\_push(intermptr\* head,intermptr newNode)

### 기능

head부터 시작하는 linkedlist의 제일 끝에 newNode를 추가한다.

### 사용변수

|  |  |
| --- | --- |
| intermptr\* head  intermptr NewNode | linkedlist로 구성된 immediate file의 시작 노드  새로운 줄의 데이터가 들어있는 노드 |

## 모듈이름: int StrToInt(char \*str)

### 기능

넘겨받은 char형인 str배열의 내용물을 10진수 integer형으로 변환하여 반환한다.

### 사용변수

|  |  |
| --- | --- |
| char \*str | integer형으로 바꾸고 싶은 값이 char형으로 들어있는 배열 |

## 모듈이름: int StrToHex(char \*str)

### 기능

넘겨받은 char형인 str배열의 내용물을 Hexadecimal로 변환하여 반환한다.

### 사용변수

|  |  |
| --- | --- |
| char \*str | Hexadecimal형으로 바꾸고 싶은 값이 char형으로 들어있는 배열 |

## 모듈이름:char\* extractContent(char \*str)

### 기능

BYTE뒤의 값은 C’EOF’나 X’F1’처럼 C또는 X그리고 따옴표로 싸여져있다. 이 함수는 이런 것들을 제외하고 내용물만을 반환하여 준다.

### 사용변수

|  |  |
| --- | --- |
| char \*str | 내용물을 추출하고자 하는 문자열 |

## 모듈이름: int isDec(char \*str)

### 기능

str에 저장된값이 10진수로 변환이 가능한지 확인해주는 함수이다. 가능하면 TRUE, 불가능하면 FALSE를 반환한다.

### 사용변수

|  |  |
| --- | --- |
| char \*str | 변환이 가능한지 검사하고 싶은 문자열 |

## 모듈이름: int isHex(char\*)

### 기능

str에 저장된값이 16진수로 변환이 가능한지 확인해주는 함수이다. 가능하면 TRUE, 불가능하면 FALSE를 반환한다.

### 사용변수

|  |  |
| --- | --- |
| char \*str | 변환이 가능한지 검사하고 싶은 문자열 |

## 모듈이름: int inRange(int min, int max, int value)

### 기능

value가 min과 max의 사이에 위치하는지 확인한다. 위치할 경우 TRUE, 아닐 경우 FALSE를 반환한다.

### 사용변수

|  |  |
| --- | --- |
| int min  int max  int value | 검사하고자 하는 범위의 최소값  검사하고자 하는 범위의 최대값  범위에 위치하는지 확인하고자 하는 값. |

## 모듈이름: int insertHexAt(int orig, int value, int index)

### 기능

value를 orig의 index번째에 Hexadecimal로 삽입한다.

### 사용변수

|  |  |
| --- | --- |
| int orig  int value  int index | value를 저장하고자 하는 destination  orig에 삽입하고자 하는 값  value를 orig에 삽입하고자 하는 위치 |

## 모듈이름: int registerNum(char \*reg)

### 기능

전달받은 reg에 들어있는 문자열이 register의 이름인지를 확인하고 맞다면 그 레지스터에 해당하는 숫자값을 반환하여 준다.

### 사용변수

|  |  |
| --- | --- |
| char \*reg | 저장하고 있는 값이 register인지를 확인하고자 하는 문자열 |

## 모듈이름: int asmSeparater(char \*str, char \*label, char \*operation, char \*operand, char \*operand2)

### 기능

str은 .asm에서 한줄의 코드를 저장하고 있다. 이 한줄의 코드를 pasing해주는 함수로써, 이 한줄에 총 몇 개의 단어가 들어왔는지를 확인하여 1개가 들어왔다면 그 단어를 operation에, 2개가 들어왔다면 operation과 operand에, 3개가 들어왔다면 label, operation, operand에 4개가 들어왔다면 label, operation, operand, operand2에 저장하여 준다.

### 사용변수

|  |  |
| --- | --- |
| char\* str  char\* label  char\* operation  char\* operand  char\* operand2 | .asm의 한줄 코드를 가지고 있는 문자열  str에 label이 존재한다면 저장  str의 directive를 저장  str에 operand가 존재한다면 저장  operand뒤에 ,X 처럼 한 개 더 들어왔다면 저장 |

# 전역 변수 정의

## symptr \*symboltable

가장 최근에 assemble에 성공한 .asm파일의 Symbol Table을 가지고 있다.

## int newsymtable

symbol table이 세팅된 적이 있는지 없는지를 저장하고 있다. 혹시 세팅이 되지 않았는데 symbol커맨드가 나오면 error메시지를 출력해준다.

# 코드

-----------------

20151619.c

-----------------

#include "20151619.h"

#include "commands.h"

#include "util.h"

//DIR Executable

int main()

{

int isPushed = FALSE, opcode;

int argCount=0, bfrCount=0, comCount=0;

char command[30],fullCmd[200];

char bfr[7][30];

unsigned int arg1,arg2,arg3;

int nextAdr = 0,cmaFlag=1;

//Initialization

lptr history = NULL;

newsymtable = TRUE;

optable = (hptr\*)malloc(sizeof(hptr)\*HASH\_SIZE);

for (int i=0; i<HASH\_SIZE; i++)

optable[i] = NULL;

memory=(char\*)malloc(sizeof(char\*)\*MEM\_SIZE);

cmd\_reset();

for(int i=0; i<7; i++)

memset(bfr[i],0,sizeof(bfr[i]));

hashMain("opcode.txt");

while(TRUE)

{

printf("sicsim>");

//Get Input

fgets(fullCmd, sizeof fullCmd,stdin);

//save with the correct format

argCount = sscanf(fullCmd,"%s%x ,%x ,%x",command, &arg1, &arg2, &arg3);

//save everything as string for checking

bfrCount = sscanf(fullCmd,"%s%s%s%s%s%s%s",bfr[0],bfr[1],bfr[2],bfr[3],bfr[4],bfr[5],bfr[6]);

//For input with input value (space) , value (space) , value

if(!strcmp(",",bfr[2])){

for(int i=3; i<7; i++)

strcpy(bfr[i-1],bfr[i]);

comCount++;

}

if(!strcmp(",",bfr[3])){

for(int i=4; i<7; i++)

strcpy(bfr[i-1],bfr[i]);

comCount++;

}

if (compareString(command, "opcode", NULL) && bfrCount == 2) {

//correct format for OPCODE [instruction] inserted

opcode = hashSearch\_opcode(bfr[1]);

if (opcode != -1){

printf("opcode is %X\n\n", opcode);

linkedlist\_push(&history,fullCmd);

}

//invalid [instruction]

else {

printf("Invalid Mnemonic!\n\n");

}

continue;

}

if (argCount != bfrCount - comCount) {

//cases such as dump 4, hello

cmaFlag = FALSE;

isPushed = TRUE;

}

// cases such as dump 1 , 10 ,

else if (comCount != 0 && argCount - 2 < comCount){

cmaFlag = FALSE;

isPushed = TRUE;

}

if (compareString(command, "h", "help") && argCount == 1){

cmd\_help();

isPushed = FALSE;

}

else if(compareString(command,"type",NULL) && argCount == 2){

cmd\_type(bfr[1]);

isPushed = FALSE;

}

else if(compareString(command,"symbol",NULL) && argCount == 1){

if(!newsymtable)

symtab\_printAll();

else {

printf("Not assembled yet!\n");

isPushed = TRUE;

}

}

else if(compareString(command,"assemble",NULL) && argCount == 2){

isPushed = cmd\_assemble(bfr[1]);

}

else if (compareString(command, "d", "dir") && argCount == 1){

cmd\_dir();

isPushed = FALSE;

}

else if (compareString(command, "q", "quit") && argCount == 1)

break;

else if (compareString(command, "hi", "history") && argCount == 1) {

linkedlist\_push(&history, fullCmd);

linkedlist\_print(history);

isPushed = TRUE;

}

//Maximum number of input is 3

else if (compareString(command, "du", "dump") && argCount <= 3 && cmaFlag) {

switch (argCount) {

//when only dump is inserted

case 1:

arg1 = INT\_MIN;

arg2 = INT\_MIN;

break;

//when dump start is inserted

case 2:

arg2 = INT\_MIN;

isPushed = checkHex(bfr[1]) && checkComma(bfr[1]);

isPushed = !isPushed;

break;

//when dump start, end is inserted

case 3:

isPushed = checkHex(bfr[1]) && checkHex(bfr[2]) && checkComma(bfr[2]);

isPushed = !isPushed;

break;

}

if (!isPushed)

isPushed = !(cmd\_dump(arg1, arg2, &nextAdr));

//Comma at the end

else

printf("Invalid command!");

}

else if (compareString(command, "e", "edit") && argCount == 3 && cmaFlag){

isPushed = checkComma(bfr[2]) && checkHex(bfr[1]) && checkHex(bfr[2]);

isPushed = !isPushed;

if (!isPushed)

isPushed = !(cmd\_edit(arg1, arg2));

else

printf("Invalid command!");

}

else if (compareString(command, "f", "fill") && argCount == 4 && cmaFlag){

//check if the command finished with a comma or if input has non-hex

isPushed = checkComma(bfr[3]) && checkHex(bfr[1]) && checkHex(bfr[2]) && checkHex(bfr[3]);

isPushed = !isPushed;

if (!isPushed)

isPushed = !(cmd\_fill(arg1,arg2, arg3));

else

printf("Invalid command!");

}

else if (compareString(command, "reset", NULL) && argCount == 1){

cmd\_reset();

isPushed = FALSE;

}

else if (compareString(command, "opcodelist", NULL) && argCount == 1){

hashlist\_printAll(optable);

isPushed = FALSE;

}

else {

isPushed = TRUE;

printf("Invalid Command!\n");

}

if (isPushed == TRUE)

isPushed = FALSE;

else

linkedlist\_push(&history,fullCmd);

//RESET

for(int i=0; i<7; i++)

memset(bfr[i],0,sizeof(bfr[i]));

argCount = bfrCount = comCount = 0;

arg1 = arg2 = arg3 = INT\_MIN;

cmaFlag = TRUE;

printf("\n");

}

}

-----------------

20151619.h

-----------------

#ifndef \_20151619\_H\_

#define \_20151619\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include <limits.h>

#include "commands.h"

#define TRUE 1

#define FALSE 0

#define MEM\_SIZE 65536 \* 16

#define WORD 3

#define BYTE 1

#define MAX\_TEXT\_LINE 69

#include "ds.h"

unsigned char \*memory;

hashlist \*\*optable;

symptr \*symboltable;

int newsymtable;

#endif

-----------------

commands.c

-----------------

#include "commands.h"

#include "util.h"

#include "20151619.h"

#include <math.h>

#include "ds.h"

//Prints Help

void cmd\_help() {

printf("h[elp]\nd[ir]\nq[uit]\nhi[story]\ndu[mp] [start,end]\n");

printf("e[dit] address, value\nf[ill] start, end, value\nreset\n");

printf("opcode mnemonic\nopcodelist\nassemble filename\ntype filename\nsymbol\n");

return;

}

void cmd\_type(char\* filename){

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

char line[100]={0,};

if (fp == NULL){

printf("File doesn't exist!\n");

return;

}

//prints 1 line at a time until FILE ends

while(fgets(line,100,fp) != NULL)

printf("%s",line);

fclose(fp);

return;

}

int cmd\_assemble(char\* filename){

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

char line[200]={0,}, line2[200]={0,};

char label[30], operand[30], operand2[30], operation[30], \*tempFormat;

char\* tempContent, \*fname, fname1[30], fname2[30];

int argCount, num, numCount, base, hashindex, totalLength;

int isFirst = TRUE, errorFlag = FALSE;

int opcode,format=-1, locctr=0,linecount;

intermptr intermediate=NULL;

intermptr newinterm;

symptr\* tempsymtab;

//PASS 1

if (fp == NULL){

printf("File doesn't exist!\n");

return TRUE;

}

tempsymtab = (symptr\*)calloc(SYM\_SIZE,sizeof(symptr)\*SYM\_SIZE);

linecount = 0;

//Get 1 line at a time from file until NULL

while(fgets(line,200,fp) != NULL && errorFlag == FALSE){

int i=0;

//count line for error

linecount++;

//empty line

if(!strcmp(line,"") || !strcmp(line,"\n"))

continue;

//ignore comment line

i=0;

while(line[i] == ' ' || line[i] =='\t')

i++;

//if it's a comment line

if(line[i] == '.'){

newinterm = (intermptr)calloc(1,sizeof(interm));

strcpy(newinterm->line,line);

newinterm->addr = -1;

interm\_push(&intermediate,newinterm);

continue;

}

//seperates a line by label, directive and operands

argCount = asmSeparater(line,label,operation,operand,operand2);

//checks if the first line is START

if(isFirst){

//sets initial location counter with the operand

if(!strcmp(operation,"START"))

locctr = StrToHex(operand);

//if start address wasn't given, set to 0

else

locctr = 0;

isFirst = FALSE;

}

//if there is a symbol

if(strcmp(label,"")){

//get the index in the symbol table

int index = symfunction(label);

//insert into that index of symbol table

if(symtab\_push(&(tempsymtab[index]), label, locctr) == TRUE){

printf("ERROR at line %d\n",linecount);

errorFlag = TRUE;

}

}

//make new node for a line of intermediate file

newinterm = (intermptr)calloc(1,sizeof(interm));

newinterm->addr = locctr;

newinterm->next = NULL;

newinterm->argCount = argCount;

//Handle format 4 exception

if(operation[0] == '+'){

format = 4;

//extract from index 1

tempFormat = hashSearch\_format(&(operation[1]));

opcode = hashSearch\_opcode(&(operation[1]));

strcpy(newinterm->operation,&(operation[1]));

if(tempFormat == NULL){

printf("ERROR at line %d\n",linecount);

errorFlag = TRUE;

continue;

}

}

else{

tempFormat = hashSearch\_format(operation);

opcode = hashSearch\_opcode(operation);

strcpy(newinterm->operation,operation);

}

//Set format for 1,2,3

if(tempFormat != NULL && format != 4){

if(!strcmp(tempFormat,"3/4"))

format = 3;

else

format = (int)tempFormat[0] - (int)'0';

}

//make new node for a line of intermediate file

newinterm->format = format;

strcpy(newinterm->operand,operand);

strcpy(newinterm->operand2,operand2);

strcpy(newinterm->line,line);

strcpy(newinterm->label,label);

interm\_push(&intermediate,newinterm);

//increase LOCCTR

if(format != -1){

locctr += format;

}//if

//directives without opcode such as RESB, RESW

else{

if(!strcmp(operation,"RESW")){

num = StrToInt(operand);

locctr += num \* 3;

}

else if(!strcmp(operation,"RESB")){

num = StrToInt(operand);

locctr += num;

}

else if(!strcmp(operation, "BYTE")){

//remove indicator X and C

tempContent = extractContent(operand);

//if the input is in Hexadecimal

if(operand[0] == 'X'){

num = (int)strlen(tempContent);

locctr += ceil((float)num/2);

}

//if the input is in Character

else if(operand[0] == 'C'){

locctr += strlen(tempContent);

}

//other than these 2, error

else{

printf("ERROR at line %d\n",linecount);

errorFlag = TRUE;

}

}

else if(!strcmp(operation, "WORD")){

num = StrToInt(operand);

locctr += 3;

}

else if(!strcmp(operation,"END")){

totalLength = locctr;

}

else if(compareString(operation,"START","BASE")){

opcode = 0;

}

//unknown directive

else{

printf("ERROR at line %d\n",linecount);

errorFlag = TRUE;

continue;

}

}//else

//reset

format = -1;

memset(operation, 0, sizeof operation);

memset(label, 0, sizeof label);

memset(operand, 0, sizeof operand);

memset(operand2, 0, sizeof operand2);

}//while

fclose(fp);

//stop if there was an error

if(errorFlag)

return TRUE;

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

//PASS 2

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

if (intermediate == NULL){

printf("No lines to assemble\n");

return TRUE;

}

else{

int offset, pc, hashindex, newlineadr;

int colcount=0,colcount2=0, linenum=0, newlineFlag=FALSE;

int \*modification = (int\*)malloc(sizeof(int)\*300);

int modfIdx=0,modfSize=300;

unsigned int obj12=0,obj3=0,obj4=0;

char strcontent[30];

FILE \*lstfile, \*objfile;

intermptr curline;

curline = intermediate;

//open list file and object file

fname = strtok(filename,".");

strcpy(fname1,fname);

strcpy(fname2,fname);

strcat(fname1,".lst");

strcat(fname2,".obj");

lstfile = fopen(fname1,"w+");

objfile = fopen(fname2,"w+");

//reset values;

colcount = 9;

colcount2 = 0;

newlineadr = -1;

memset(line,0,sizeof(line));

//if first line isn't Start

if(strcmp(curline->operation,"START"))

fprintf(objfile,"H 000000%06X\n",totalLength);

linecount = 0;

while(curline != NULL && errorFlag == FALSE){

char objstr[3][20];

linecount++;

//lines that don't produce opcodes

curline->line[strlen(curline->line)-1] = '\0';

//print line number

fprintf(lstfile,"%04d\t",linenum);

linenum +=5;

//if current line of intermediate file is a comment line

if(curline->addr == -1){

fprintf(lstfile,"\t%s\n",curline->line);

curline = curline->next;

continue;

}

//print location counter if it's not END

if(strcmp(curline->operation,"END"))

fprintf(lstfile,"%04X\t",curline->addr);

else

fprintf(lstfile," \t");

//print assembly code

fprintf(lstfile,"%-30s",curline->line);

//if current line is RESW or RESB, start new line

if(compareString(curline->operation,"RESW","RESB")){

fprintf(lstfile,"\n");

newlineFlag = TRUE;

curline = curline->next;

continue;

}

//if current line is BASE, move to the next line

else if(!strcmp(curline->operation,"BASE")){

fprintf(lstfile,"\n");

curline = curline->next;

continue;

}

//if current line is START

else if(!strcmp(curline->operation,"START")){

fprintf(lstfile,"\n");

//Start Header record

fprintf(objfile,"H");

fprintf(objfile,"%-6s",curline->label);

fprintf(objfile,"%06X",curline->addr);

fprintf(objfile,"%06X\n",totalLength-curline->addr);

//start a new Text Record

fprintf(objfile,"T%06X",curline->addr);

curline = curline->next;

continue;

}

else if(!strcmp(curline->operation,"END")){

fprintf(lstfile,"\n");

//print last line TEXT LINE of obj file

if(strcmp(line,""))

fprintf(objfile,"%02X%s",(int)(strlen(line))/2,line);

if(strcmp(line2,""))

fprintf(objfile,"\nT%06X%s",newlineadr,line2);

//modification record

for(int k=0;k<modfIdx;k++)

fprintf(objfile,"\nM%06X05",modification[k]);

//get the address of the symbol that END points

hashindex = symfunction(curline->operand);

offset = symtab\_search(tempsymtab[hashindex],curline->operand);

fprintf(objfile,"\nE%06X\n",offset);

curline = curline->next;

continue;

}

//initialize variables

obj12 = obj3 = obj4 = 0;

for(int i=0;i<3;i++)

memset(objstr[i],0,sizeof(objstr[i]));

//intialize new line

if(newlineFlag == TRUE){

//if there's anything in line

if(strcmp(line,"")){

fprintf(objfile,"%02X%s\n",(int)(strlen(line))/2,line);

memset(line,0,sizeof(line));

}

//line2 is empty

if(!strcmp(line2,"")){

fprintf(objfile,"T%06X",curline->addr);

}

//if line2 isn't empty

else{

fprintf(objfile,"T%06X",newlineadr);

strcpy(line,line2);

memset(line2,0,sizeof(line2));

}

//reset values for new line

colcount = 9 + colcount2;

colcount2 = 0;

newlineadr = -1;

newlineFlag = FALSE;

}

//first 2 digits of object code

obj12 = hashSearch\_opcode(curline->operation);

//format 2

if(curline->format == 2){

//if command is SVC

if(!strcmp(curline->operand,"SVC")){

int temp = StrToHex(curline->operand);

obj3 = insertHexAt(obj3,temp,1);

}

else{

int temp = registerNum(curline->operand);

//if operand isn't a register

if(temp == -1){

printf("ERROR at line %d\n",linecount);

errorFlag = TRUE;

continue;

}

obj3 = insertHexAt(obj3,temp,1);

//If second operand exists

if(curline->argCount < 0){

//if directive is SHIFTL or SHIFTR, 2nd operand is an integer

if(compareString(curline->operand2,"SHIFTL","SHIFTR")){

temp = StrToHex(curline->operand2);

obj3 = insertHexAt(obj3,temp,0);

}

else{

temp = registerNum(curline->operand2);

//if operand isn't a register

if(temp == -1){

printf("ERROR at line %d\n",linecount);

errorFlag = TRUE;

continue;

}

obj3 = insertHexAt(obj3,temp,0);

}

}

}//else

}

else if(curline->format >= 3){

//set e flag for format 4

if(curline->format == 4)

obj3++;

if(abs(curline->argCount) > 1){

//set flag for addressing mode

obj12 += addressingMode(curline->operand);

//Set flag for indexed addressing

if(!strcmp(curline->operand2,"X"))

obj3 += 8;

//remove first character

if(curline->operand[0] == '#' || curline->operand[0] == '@')

strcpy(strcontent,curline->operand +1);

else

strcpy(strcontent,curline->operand);

//Check if operation is LDB and set base variable

hashindex = symfunction(strcontent);

offset = symtab\_search(tempsymtab[hashindex],strcontent);

if(!strcmp(curline->operation,"LDB"))

base = offset;

//operand doesn't exist in the symbol table

if(offset == -1){

int temp = StrToInt(strcontent);

if (temp != -1)

obj4 += temp;

else{

printf("ERROR at line %d\n",linecount);

errorFlag = TRUE;

continue;

}

}

else{

//if format4, direct addressing

if(curline->format == 4){

//if it is not immediate addressing

if(curline->operand[0] != '#'){

//add to modification record

modification[modfIdx++] = curline->addr + 1;

if(modfIdx == modfSize - 1){

modfSize \*= 2;

modification = (int\*)realloc(modification,sizeof(int)\*modfSize);

}

}

obj4 = offset;

}

else{

//check for PC relative

pc = curline->next->addr;

if(offset - pc >= -2048 && offset - pc <= 2047){

//use pc relative

obj4 = offset - pc;

obj3 += 2;

}

else{

//use base relative

obj4 = offset - base;

obj3 += 4;

}

}

}

}//if argCount > 1

else{

obj12 += 3;

}

}

//When BYTE directive is inserted

else if(!strcmp(curline->operation, "BYTE")){

//Extract content, removing C and X

char \*tempContent = extractContent(curline->operand);

if(curline->operand[0] == 'X'){

//write content to file

fprintf(lstfile,"%s\n",tempContent);

//if this line can fit into the current line of object file

if(colcount + strlen(tempContent) <= MAX\_TEXT\_LINE){

colcount += strlen(tempContent);

strcat(line,tempContent);

}

//if this line needs to be on the next line of object file

else{

newlineFlag = TRUE;

colcount2 = strlen(tempContent);

newlineadr = curline->addr;

strcat(line2,tempContent);

}

}

//Character inserted

else if(curline->operand[0] == 'C'){

char convstr[10];

char tempstr[10];

//convert each character to corresponding ASCII in hex and write to file

for(int i=0; i<strlen(tempContent); i++){

fprintf(lstfile,"%2X",(int)(tempContent[i]));

sprintf(convstr,"%2X",(int)(tempContent[i]));

strcat(tempstr,convstr);

}

fprintf(lstfile,"\n");

//if this line can fit into the current line of object file

if(colcount + strlen(tempstr) <= MAX\_TEXT\_LINE){

colcount += strlen(tempstr);

strcat(line,tempstr);

}

//if this line needs to be on the next line of object file

else{

newlineFlag = TRUE;

colcount2 = strlen(tempstr);

newlineadr = curline->addr;

strcat(line2,tempstr);

}

}

else

fprintf(lstfile,"%s\n",curline->operand);

curline = curline->next;

continue;

}//format3,4 if

else if(!strcmp(curline->operation, "WORD")){

char tempstr[10];

int temp = StrToInt(curline->operand);

sprintf(tempstr,"%06X",temp);

fprintf(lstfile,"%06X\n",temp);

//if this line can fit into the current line of object file

if(colcount + strlen(tempstr) <= MAX\_TEXT\_LINE){

colcount += strlen(tempstr);

strcat(line,tempstr);

}

//if this line needs to be on the next line of object file

else{

newlineFlag = TRUE;

colcount2 = strlen(tempstr);

newlineadr = curline->addr;

strcat(line2,tempstr);

}

curline = curline->next;

continue;

}

//make first 2 Hexadecimal

sprintf(objstr[0],"%02X",obj12);

//make the rest of object codes

switch(curline->format){

//format 1

case 1:

fprintf(lstfile,"%02X\n",obj12);

break;

//format 2

case 2:

fprintf(lstfile,"%02X%02X\n",obj12,obj3);

sprintf(objstr[1],"%02X",obj3);

break;

//format 3

case 3:

fprintf(lstfile,"%02X%01X%03X\n",obj12,obj3,obj4%(1<<12));

sprintf(objstr[1],"%01X",obj3);

sprintf(objstr[2],"%03X",obj4%(1<<12));

break;

//format 4

case 4:

fprintf(lstfile,"%02X%01X%05X\n",obj12,obj3,obj4%(1<<20));

sprintf(objstr[1],"%01X",obj3);

sprintf(objstr[2],"%05X",obj4%(1<<20));

break;

}

//if this line can fit into the current line of object file

if(colcount + curline->format\*2 <= MAX\_TEXT\_LINE){

colcount += curline->format\*2;

for(int k=0;k<curline->format && k<3;k++)

strcat(line,objstr[k]);

}

//if this line needs to be on the next line of object file

else{

newlineFlag = TRUE;

colcount2 = curline->format\*2;

newlineadr = curline->addr;

for(int k=0;k<curline->format && k<4;k++)

strcat(line2,objstr[k]);

}

curline = curline->next;

}//while

fclose(lstfile);

fclose(objfile);

}//else

if(errorFlag){

remove(fname1);

remove(fname2);

return TRUE;

}

//if symbol table is alreay set, clear

if(!newsymtable){

for(int i = SYM\_SIZE; i>=0; i--)

free(symboltable[i]);

free(symboltable);

symboltable = NULL;

}

printf("\noutput file: [%s], [%s]\n",fname1,fname2);

//successfully assembled. Set global symbol table to current file's symbol table

symboltable = tempsymtab;

newsymtable = FALSE;

return FALSE;

}

//Prints files in current directory

void cmd\_dir() {

DIR \*dir;

struct dirent \*file;

struct stat info;

if ((dir = opendir("./")) != NULL) {

/\* print all the files and directories within directory \*/

while ((file = readdir(dir)) != NULL) {

stat(file->d\_name, &info);

//If file is a directory

if(S\_ISDIR(info.st\_mode))

printf("%s/\n",file->d\_name);

//If file is executable

else if(info.st\_mode & S\_IXUSR)

printf("%s\*\n", file->d\_name);

//other files

else

printf("%s\n",file->d\_name);

}

closedir(dir);

}

else {

/\* could not open directory \*/

perror("");

printf("Error opening directory");

}

return;

}

//Prints memory

int cmd\_dump(int start, int end, int\* nextAdr) {

int row, col, over = FALSE;

int rowStart, rowEnd, curAdr;

//if ending addr is not inserted

if (end == INT\_MIN) {

//if starting addr is not inserted

if (start == INT\_MIN)

start = \*nextAdr;

end = start + 159;

}

else{

//if inserted range end is out of bound

if (checkRange(end) == FALSE || checkRange(start) == FALSE){

printf("Invalid Range\n");

return FALSE;

}

}

//if start addr is bigger than end addr

if (end < start) {

printf("Invalid Range\n");

return FALSE;

}

// if start is out of memory range

if (!checkRange(start)) {

printf("Invalid Range\n");

return FALSE;

}

rowStart = start / 16;

rowEnd = end / 16;

for (row = rowStart; row <= rowEnd; row++) {

// if row is out of range

if (row \* 16 > 0xFFFFF) {

over = TRUE;

break;

}

printf("%05X ", row \* 16);

//Content

for (col = 0; col < 16; col++) {

curAdr = row \* 16 + col;

if (curAdr > 0xFFFFF) {

over = TRUE;

break;

}

//Range before start

if (curAdr < start)

printf(" ");

//Range after end

else if (curAdr > end)

printf(" ");

else

printf("%02X ", memory[curAdr]);

}

printf("; ");

//Value

for (col = 0; col < 16; col++)

{

curAdr = row \* 16 + col;

if (curAdr > 0xFFFFF) {

over = TRUE;

break;

}

//Range before start

if (curAdr < start)

printf(". ");

//Range after end

else if (curAdr > end)

printf(". ");

else {

if ((int)memory[curAdr] >= 0x20 && (int)memory[curAdr] <= 0x7E)

printf("%c ", memory[curAdr]);

else

printf(". ");

}

}

printf("\n");

}

if (over)

\*nextAdr = 0;

else

\*nextAdr = end + 1;

return TRUE;

}

//change a memory's content

int cmd\_edit(int adr, int value) {

if (!checkRange(adr)) {

printf("Wrong Address!\n");

return FALSE;

}

memory[adr] = value;

return TRUE;

}

//changes memory to value from start adr to end adr

int cmd\_fill(int start, int end, int value) {

int rowStart, rowEnd, row, col, curAdr;

if (end < start) {

printf("Invalid Range!\n");

return FALSE;

}

if (!(checkRange(start) && checkRange(end))) {

printf("Invalid Range!\n");

return FALSE;

}

rowStart = start / 16;

rowEnd = end / 16;

for (row = rowStart; row <= rowEnd; row++) {

for (col = 0; col < 16; col++) {

curAdr = row \* 16 + col;

//change curAdr memory to value if curAdr is not out of bound

if (curAdr >= start && curAdr <= end)

memory[curAdr] = value;

}

}

return TRUE;

}

//resets entire memory

void cmd\_reset() {

memset(memory, 0, sizeof(char)\*MEM\_SIZE);

}

int hashSearch\_opcode(char\* mnem) {

//Convert lowercase to uppercase

LowerToUpper(mnem);

int index = hashfunction(mnem);

hptr temp = optable[index];

//search untill the end of one optable's index

while (temp != NULL) {

//if the mnemonic that i'm looking for is found

if (!strcmp(mnem, temp->mnem))

return temp->opcode;

//otherwise continue to the next node

temp = temp->next;

}

return -1;

}

char\* hashSearch\_format(char\* mnem) {

//Convert lowercase to uppercase

LowerToUpper(mnem);

int index = hashfunction(mnem);

hptr temp = optable[index];

//search untill the end of one optable's index

while (temp != NULL) {

//if the mnemonic that i'm looking for is found

if (!strcmp(mnem, temp->mnem))

return temp->format;

//otherwise continue to the next node

temp = temp->next;

}

return NULL;

}

int addressingMode(char\* str){

if(str[0] == '#')

return 1;

else if(str[0] == '@')

return 2;

else

return 3;

}

-----------------

commands.h

-----------------

#ifndef \_COMMANDS\_H\_

#define \_COMMANDS\_H\_

#define CHAR 0

#define HEX 1

#define DEC 2

void cmd\_dir();

void cmd\_help();

void cmd\_type(char\*);

int cmd\_assemble(char\*);

void cmd\_reset();

int cmd\_dump(int, int, int\*);

int cmd\_edit(int, int);

int cmd\_fill(int, int, int);

int hashSearch\_opcode(char\*);

char\* hashSearch\_format(char\*);

int addressingMode(char\*);

#endif

-----------------

ds.c

-----------------

#include "ds.h"

#include "20151619.h"

#include <string.h>

#include <stdlib.h>

//appends new node with command to the end of head linked list

void linkedlist\_push(lptr\* head,char\* command){

lptr temp = \*head;

lptr newNode = (lptr)malloc(sizeof(linkedlist));

newNode->next = NULL;

strcpy(newNode->command,command);

if(temp != NULL){

//goes to the last node

while(temp->next != NULL)

temp = temp->next;

temp->next = newNode;

}

else{

\*head = newNode;

}

return;

}

//prints all the nodes in the linked list my moving one node at a time

void linkedlist\_print(lptr head){

int count = 1;

lptr temp = head;

while(temp != NULL){

printf("%-4d %s",count++,temp->command);

temp = temp->next;

}

return;

}

//main function to create initial hashtable

void hashMain(char\* fname){

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

int opcode,index;

char mnem[30], format[30];

if (fp != NULL){

while(fscanf(fp,"%X %s %s",&opcode,mnem,format) != EOF){

//finds the index for mnemonic to go in the hashtable

index = hashfunction(mnem);

//appends that mnemonic to the table

hashlist\_push(&(optable[index]),mnem,opcode,format);

}

}

else{

printf("Cannot open File\n!");

}

}

//returns index number for the mnemonic to go into in the hashtable

int hashfunction(char\* str){

int i,sum=0;

for(i=0; i<(int)strlen(str); i++)

sum += (int)str[i];

return sum % HASH\_SIZE;

}

//creates a new node with mnemonic, opcode and format. Then inserts the new node the end of head list

void hashlist\_push(hptr \*head,char\* mnem,int opcode, char\* format){

hptr temp = \*head;

//make new node

hptr newNode = (hptr)malloc(sizeof(hashlist));

newNode->next = NULL;

strcpy(newNode->mnem,mnem);

strcpy(newNode->format,format);

newNode->opcode = opcode;

if(\*head != NULL){

//go to the end of head

while(temp->next != NULL)

temp = temp->next;

//insert to new node

temp->next = newNode;

}else{

\*head = newNode;

}

return;

}

//prints whole hashtable

void hashlist\_printAll(hptr \*head){

int i, isFirst;

hptr temp;

for(i=0; i<HASH\_SIZE; i++){

temp = head[i];

isFirst = TRUE;

//if an index is not empty

if(temp != NULL){

printf("%d : ",i);

//iterate through the list

while(temp != NULL){

if(!isFirst)

printf(" -> ");

printf("[%s, %X]",temp->mnem,temp->opcode);

temp = temp->next;

isFirst = FALSE;

}

printf("\n");

}

}

}

int symtab\_push(symtab \*\*head,char\* label,int addr){

symtab \*temp;

symtab \*prev;

//make new node

symtab\* newNode = (symtab\*)malloc(sizeof(symtab));

strcpy(newNode->label,label);

newNode->addr = addr;

newNode->next = NULL;

if(\*head != NULL){

//head에 삽입하는 경우

if (strcmp((\*head)->label, label) < 0){

newNode->next = \*head;

\*head = newNode;

return FALSE;

}

else{

temp = (\*head)->next;

prev = \*head;

}

//Search for the right place to go

while(temp != NULL){

//duplicate symbol

if(!strcmp(temp->label,label))

return TRUE;

if(strcmp(temp->label,label) < 0){

prev->next = newNode;

newNode ->next = temp;

return FALSE;

}

prev = temp;

temp = temp->next;

}

//insert to new node

prev->next = newNode;

}else{

\*head = newNode;

}

return FALSE;

}

int symtab\_search(symtab\* head,char\* label){

while(head != NULL){

if(!strcmp(head->label,label))

return head->addr;

head = head->next;

}

return -1;

}

void symtab\_printAll(){

for(int i=SYM\_SIZE-1;i>=0;i--)

symtab\_print(symboltable[i]);

}

void symtab\_print(symtab \*head){

while(head != NULL){

printf("\t%s %X\n",head->label,head->addr);

head = head->next;

}

}

int symfunction(char\* str){

return (int)str[0] - (int)'A';

}

void interm\_push(intermptr \*head,intermptr newNode){

intermptr temp = \*head;

if(\*head != NULL){

//go to the end of head

while(temp->next != NULL)

temp = temp->next;

//insert to new node

temp->next = newNode;

}else{

\*head = newNode;

}

return;

}

-----------------

ds.h

-----------------

#ifndef \_DS\_C\_

#define \_DS\_C\_

#define HASH\_SIZE 20

#define SYM\_SIZE 26

typedef struct interm\* intermptr;

typedef struct interm {

int addr;

int format;

int argCount;

char operation[30];

char operand[20];

char operand2[20];

char label[30];

char line[200];

intermptr next;

}interm;

typedef struct linkedlist\* lptr;

typedef struct linkedlist {

char command[100];

lptr next;

}linkedlist;

void linkedlist\_push(lptr\*, char\*);

void linkedlist\_print(lptr);

typedef struct symtable\* sptr;

typedef struct symtable {

char label[30];

unsigned int addr;

sptr next;

}symtable;

typedef struct hashlist\* hptr;

typedef struct hashlist {

char mnem[20];

int opcode;

char format[20];

hptr next;

}hashlist;

void hashMain(char\*);

int hashfunction(char\*);

void hashlist\_push(hptr\*, char\*, int, char\*);

void hashlist\_printAll(hptr\*);

typedef struct symtab\* symptr;

typedef struct symtab {

char label[30];

int addr;

symptr next;

}symtab;

int symfunction(char\*);

int symtab\_push(symtab\*\*,char\*,int);

void symtab\_print(symptr);

void symtab\_printAll();

int symtab\_search(symtab\*,char\*);

void interm\_push(intermptr\*, intermptr);

#endif

-----------------

util.c

-----------------

#include "util.h"

#include "20151619.h"

#include "math.h"

//compares command with shortcommand and longcommand and see if command matches with one of them

int compareString(char\* command, char\* shortcommand, char\* longcommand) {

if (!strcmp(command, shortcommand) || (longcommand != NULL && !strcmp(command, longcommand)))

return TRUE;

else

return FALSE;

}

//check if adr is within the valid range

int checkRange(int adr) {

if (adr < 0 || adr >= 0x100000)

return FALSE;

return TRUE;

}

//check if str has any non-hex char

int checkHex(char\* str){

for(int i=0; i<(int)strlen(str); i++){

if((str[i] >=48 && str[i] <=57) || (str[i] >= 65 && str[i] <=70) || (str[i] >= 97 && str[i] <= 102) || str[i] == ',')

continue ;

else if(str[i] == 'X' || str[i] == 'x'){

if(i != 0)

if(str[i-1] == '0')

continue;

}

else{

return FALSE;

}

}

return TRUE;

}

//if argCount < 0 , you need to check operand2

int asmSeparater(char\* str, char\* label, char\* operation, char\* operand, char\* operand2){

int argCount = sscanf(str,"%s %s %s %s",label,operation,operand,operand2);

//cases with , such as LDCH BUFFER, X

if(!checkComma(operation)){

argCount \*= -1;

//if not all 4 inputs come in, move back

if(argCount != -4){

operation[(int)strlen(operation)-1] = '\0';

strcpy(operand2,operand);

strcpy(operand,operation);

strcpy(operation,label);

memset(label,0,sizeof label);

}

}

else if(!checkComma(operand)){

argCount \*= -1;

operand[(int)strlen(operand)-1] = '\0';

}

//printf("[%d] ",argCount);

switch(argCount){

case 1:

strcpy(operation,label);

memset(label,0,sizeof label);

break;

case 2:

strcpy(operand,operation);

strcpy(operation,label);

memset(label,0,sizeof label);

break;

}

return argCount;

}

//changes word to uppercase

void LowerToUpper(char\* word) {

for (int i = 0; i < (int)strlen(word); i++)

if ((int)word[i] > 96 && (int)word[i] < 123)

word[i] -= 32;

}

//check if command finishes with a comma

int checkComma(char\* command){

int len = (int)strlen(command);

if(command[len-1] == ',')

return FALSE;

return TRUE;

}

int StrToHex(char\* str){

unsigned int num;

int cnt = sscanf(str,"%x",&num);

if(cnt == 0)

return -1;

return num;

}

int StrToInt(char\* str){

unsigned int num;

int cnt = sscanf(str,"%d",&num);

if(cnt == 0)

return -1;

return num;

}

char\* extractContent(char\* str){

char \*temp = (char\*)malloc(30);

strncpy(temp,&(str[2]),strlen(str)-3);

temp[strlen(str)-3] = 0;

return temp;

}

int isDec(char\* str){

int len = (int)strlen(str);

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

if(!inRange((int)'0',(int)'9',str[i]))

return FALSE;

return TRUE;

}

int isHex(char\* str){

int len = (int)strlen(str);

if(isDec(str) == TRUE)

return TRUE;

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

if(!inRange((int)'a',(int)'f',str[i]) || !inRange((int)'A',(int)'F',str[i]))

return FALSE;

return TRUE;

}

int inRange(int min,int max,int value){

if(value >= min && value <=max)

return TRUE;

return FALSE;

}

int HexBitCount(int num){

int digit = 0;

while(num >=16){

num /= 16;

digit++;

}

return digit;

}

int HexByteCount(int num){

return (int)(ceil((float)num/2));

}

int insertHexAt(int orig, int value, int index){

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

value \*= 16;

return orig + value;

}

int registerNum(char\* reg){

if(!strcmp(reg,"A"))

return 0;

else if(!strcmp(reg,"X"))

return 1;

else if(!strcmp(reg,"L"))

return 2;

else if(!strcmp(reg,"PC"))

return 8;

else if(!strcmp(reg,"SW"))

return 9;

else if(!strcmp(reg,"B"))

return 3;

else if(!strcmp(reg,"S"))

return 4;

else if(!strcmp(reg,"T"))

return 5;

else if(!strcmp(reg,"F"))

return 6;

else

return -1;

}

-----------------

util.h

-----------------

#ifndef \_UTIL\_H\_

#define \_UTIL\_H\_

int checkRange(int);

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

int checkComma(char\*);

int checkHex(char\*);

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

void LowerToUpper(char\*);

int StrToInt(char\*);

int StrToHex(char\*);

char\* extractContent(char\*);

int isDec(char\*);

int isHex(char\*);

int inRange(int,int,int);

int HexBitCount(int);

int insertHexAt(int,int,int);

int registerNum(char\*);

#endif