시스템 프로그래밍

담당 교수: 김지환

< Project 1 >

Shell 환경 구현

서강대학교 컴퓨터공학과

학번: 20161577

이름: 김인호

제출 마감:  
2018년 03월 26일 (월)

**목차**

[1. 프로그램 개요 5](#_Toc509855634)

[2. 프로그램 설명 5](#_Toc509855635)

[2.2. 프로그램 흐름도 5](#_Toc509855636)

[3. 모듈 정의 6](#_Toc509855637)

[3.1. 모듈 이름: main() 6](#_Toc509855638)

[3.1.1. 기능 6](#_Toc509855639)

[3.1.2. 사용 변수 6](#_Toc509855640)

[3.2. 모듈 이름: findCMD(char\*) 6](#_Toc509855641)

[3.2.1. 기능 6](#_Toc509855642)

[3.2.2. 사용 변수 6](#_Toc509855643)

[3.3. 모듈 이름: testValidInput( INPUT\_CMD, COMMAND ) 7](#_Toc509855644)

[3.3.1. 기능 7](#_Toc509855645)

[3.3.2. 사용 변수 7](#_Toc509855646)

[3.4. 모듈 이름: hexToDec(char\*) 7](#_Toc509855647)

[3.4.1. 기능 7](#_Toc509855648)

[3.4.2. 사용 변수 7](#_Toc509855649)

[3.5. 모듈 이름: invFormat() 7](#_Toc509855650)

[3.5.1. 기능 7](#_Toc509855651)

[3.5.2. 사용 변수 7](#_Toc509855652)

[3.6. 모듈 이름: invHexCMD() 8](#_Toc509855653)

[3.6.1. 기능 8](#_Toc509855654)

[3.6.2. 사용 변수 8](#_Toc509855655)

[3.7. 모듈 이름: invValCMD() 8](#_Toc509855656)

[3.7.1. 기능 8](#_Toc509855657)

[3.7.2. 사용 변수 8](#_Toc509855658)

[3.8. 모듈 이름: helpCMD() 8](#_Toc509855659)

[3.8.1. 기능 8](#_Toc509855660)

[3.8.2. 사용 변수 8](#_Toc509855661)

[3.9. 모듈 이름: dirCMD() 8](#_Toc509855662)

[3.9.1. 기능 8](#_Toc509855663)

[3.9.2. 사용 변수 9](#_Toc509855664)

[3.10. 모듈 이름: quitCMD() 9](#_Toc509855665)

[3.10.1. 기능 9](#_Toc509855666)

[3.10.2. 사용 변수 9](#_Toc509855667)

[3.11. 모듈 이름: histCMD() 9](#_Toc509855668)

[3.11.1. 기능 9](#_Toc509855669)

[3.11.2. 사용 변수 9](#_Toc509855670)

[3.12. 모듈 이름: histAdd(char\*) 9](#_Toc509855671)

[3.12.1. 기능 9](#_Toc509855672)

[3.12.2. 사용 변수 10](#_Toc509855673)

[3.13. 모듈 이름: histFree() 10](#_Toc509855674)

[3.13.1. 기능 10](#_Toc509855675)

[3.13.2. 사용 변수 10](#_Toc509855676)

[3.14. 모듈 이름: dumpCMD( INPUT\_CMD ) 10](#_Toc509855677)

[3.14.1. 기능 10](#_Toc509855678)

[3.14.2. 사용 변수 10](#_Toc509855679)

[3.15. 모듈 이름: editCMD( INPUT\_CMD ) 11](#_Toc509855680)

[3.15.1. 기능 11](#_Toc509855681)

[3.15.2. 사용 변수 11](#_Toc509855682)

[3.16. 모듈 이름: fillCMD( INPUT\_CMD ) 11](#_Toc509855683)

[3.16.1. 기능 11](#_Toc509855684)

[3.16.2. 사용 변수 11](#_Toc509855685)

[3.17. 모듈 이름: resetCMD() 11](#_Toc509855686)

[3.17.1. 기능 11](#_Toc509855687)

[3.17.2. 사용 변수 11](#_Toc509855688)

[3.18. 모듈 이름: opCMD( INPUT\_CMD ) 12](#_Toc509855689)

[3.18.1. 기능 12](#_Toc509855690)

[3.18.2. 사용 변수 12](#_Toc509855691)

[3.19. 모듈 이름: oplistCMD() 12](#_Toc509855692)

[3.19.1. 기능 12](#_Toc509855693)

[3.19.2. 사용 변수 12](#_Toc509855694)

[3.20. 모듈 이름: hashCreate() 12](#_Toc509855695)

[3.20.1. 기능 12](#_Toc509855696)

[3.20.2. 사용 변수 12](#_Toc509855697)

[3.21. 모듈 이름: hashFunction(char\*) 13](#_Toc509855698)

[3.21.1. 기능 13](#_Toc509855699)

[3.21.2. 사용 변수 13](#_Toc509855700)

[3.22. 모듈 이름: hashAddBucket(int, HASH\_ENTRY\*) 13](#_Toc509855701)

[3.22.1. 기능 13](#_Toc509855702)

[3.22.2. 사용 변수 13](#_Toc509855703)

[3.23. 모듈 이름: hashFree() 13](#_Toc509855704)

[3.23.1. 기능 13](#_Toc509855705)

[3.23.2. 사용 변수 13](#_Toc509855706)

[4. 전역 변수 정의 14](#_Toc509855707)

[4.1. COMMAND cmdList[CMD\_CNT] 14](#_Toc509855708)

[4.2. HIST\_NODE\* histHead 14](#_Toc509855709)

[4.3. short mem[MEM\_SIZE] 14](#_Toc509855710)

[4.4. HASH\_ENTRY\* hashTable[HASH\_SIZE] 14](#_Toc509855711)

[5. 코드 15](#_Toc509855712)

[5.1. 20161577.h 15](#_Toc509855713)

[5.2. 20161577.c 16](#_Toc509855714)

[5.3. cmdProc.h 18](#_Toc509855715)

[5.4. cmdProc.c 19](#_Toc509855716)

[5.5. shell.h 22](#_Toc509855717)

[5.6. shell.c 23](#_Toc509855718)

[5.7. memory.h 25](#_Toc509855719)

[5.8. memory.c 25](#_Toc509855720)

[5.9. hash.h 27](#_Toc509855721)

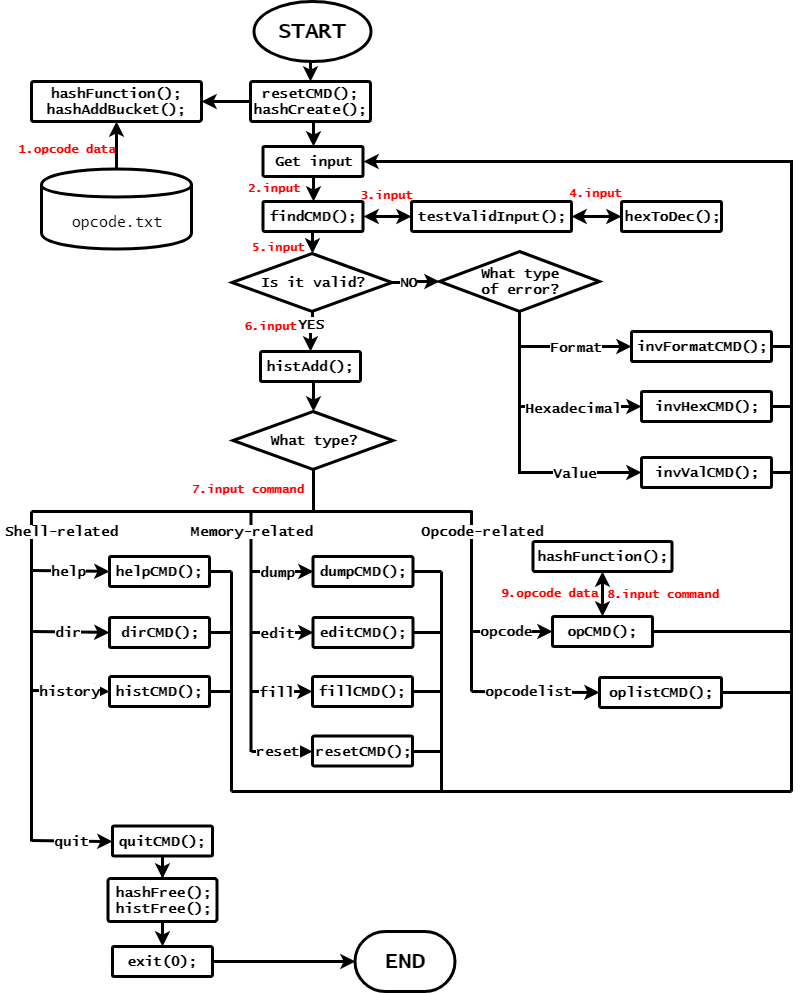
[5.10. hash.c 27](#_Toc509855722)

# 프로그램 개요

SIC/XE Machine 구현을 위한 전 단계로 assembler, linker, loader 등을 수행할 shell 환경과 object code가 놓일 가상 메모리 공간과 opcode의 mnemonic을 변환해주는 opcode table을 구현하는 프로그램이다.

# 프로그램 설명

## 프로그램 흐름도



# 모듈 정의

## 모듈 이름: main()

### 기능

가상 메모리 초기화와 opcode.txt로부터 hash table 생성 등 프로그램 실행을 위한 초기화 작업을 진행한다. 그 후 shell prompt를 출력하고 사용자 입력을 받고 이를 처리하여 history 에 추가하는 함수와 해당 기능을 수행해주는 함수를 호출하는 메인 루프를 실행한다.

### 사용 변수

|  |  |
| --- | --- |
| char inp[] | 사용자의 입력을 저장 변수 |
| char tmp[] | 입력 string 을 분석하기 위해 쓰이는 임시 공간 |
| INPUT\_CMD input | parsing된 입력을 저장하는 변수 |

## 모듈 이름: findCMD(char\*)

### 기능

입력 string을 parsing하여 해당 명령어를 찾는 함수이다. 입력 string을 띄어쓰기와 탭 등 빈 칸 기준으로 나눈 첫 단어와 매칭되는 명령어를 찾고 그 이후에 나온 단어들을 명령어 인자로 parsing하여 ipcmd 변수에 저장한다. Parsing 과정에 잘못된 입력을 발견 시 이를 감지하고 에러 종류를 찾는다. 모든 과정 후에는 ipcmd를 return한다.

### 사용 변수

|  |  |
| --- | --- |
| char\* str | 함수 인자, 사용자의 입력 string |
| char delim[] | 입력 string을 단어로 나누는 기준 character |
| char inp[] | parsing 과정을 위해 입력 string을 복사 해놓는 변수 |
| char\* tok | 입력 string tokenize를 위한 변수 |
| INPUT\_CMD ipcmd | parsing된 입력을 저장하고 return 되는 변수 |

## 모듈 이름: testValidInput( INPUT\_CMD, COMMAND )

### 기능

구현 중인 shell의 명시된 명령어의 형식에 따라 인자의 형식, 값 그리고 범위가 맞는지 확인하는 함수이다. 에러 코드를 return해준다.

### 사용 변수

|  |  |
| --- | --- |
| INPUT\_CMD ipcmd | 함수 인자, parsing된 입력 |
| COMMAND format | 함수 인자, system 명령으로 정의된 명령어의 형식 |
| int arg[] | 명령어의 각 인자를 저장하는 변수 |
| ERROR\_CODE code | return 되는 변수로 입력에 에러가 있으면 해당 에러코드 값을 갖고 없으면 SAFE 이다 |

## 모듈 이름: hexToDec(char\*)

### 기능

인자로 받는 16진수 string을 10진수 값으로 return 해준다. 만약에 올바른 16진수 형식이 아니라면 -1을 return 한다.

### 사용 변수

|  |  |
| --- | --- |
| char\* hex | 함수 인자, 10진수로 변환될 16진수 string |
| int dec | return 된 10진수 값 |
| int multiplier | 10진수 변환 작업에 필요한 16의 제곱들 저장 |

## 모듈 이름: invFormat()

### 기능

입력된 명령어의 형식이 잘못되었을 때 에러 메시지를 출력해준다.

### 사용 변수

사용 변수 없음.

## 모듈 이름: invHexCMD()

### 기능

입력된 인자의 16진수 값이 잘못되었을 때 에러 메시지를 출력해준다.

### 사용 변수

사용 변수 없음.

## 모듈 이름: invValCMD()

### 기능

입력된 인자의 값이 잘못되었을 때 에러 메시지를 출력해준다.

### 사용 변수

사용 변수 없음.

## 모듈 이름: helpCMD()

### 기능

help 명령어 입력 시 해당 기능을 수행한다. 이 system이 이해하는 명령어와 형식을 출력한다.

### 사용 변수

사용 변수 없음.

## 모듈 이름: dirCMD()

### 기능

dir 명령어 입력 시 해당 기능을 수행한다. 현재 directory의 내용을 출력한다. 현재 directory 내의 directory는 ‘/’ 그리고 실행파일은 ‘\*’ 으로 구분한다.

### 사용 변수

|  |  |
| --- | --- |
| DIR\* dir | 현재 directory 의 포인터 |
| char\* entStr | 현재 directory 내용의 파일명 |
| char path[] | 현재 directory path |
| ENTRY\* buf | 각 파일을 가리킬 포인터 |
| STBUF buf | directory 와 실행파일을 구분하기 위한 용도 |

## 모듈 이름: quitCMD()

### 기능

quit 명령어 입력 시 해당 기능을 수행한다. System 종료 메시지를 출력하고 history linked list와 opcode hash table의 메모리 할당을 해제한 후 프로그램을 종료한다.

### 사용 변수

사용 변수 없음.

## 모듈 이름: histCMD()

### 기능

history 명령어 입력 시 해당 기능을 수행한다. 지금까지 수행한 명령어의 기록을 출력한다.

### 사용 변수

|  |  |
| --- | --- |
| HIST\_NODE\* cur | linked list 탐색을 위한 변수 |
| cnt | 기록 개수 저장 변수 |

## 모듈 이름: histAdd(char\*)

### 기능

에러가 아닌 명령어들을 linked list 형태인 기록에 추가한다.

### 사용 변수

|  |  |
| --- | --- |
| char\* str | 함수 인자, history에 기록될 입력 string |
| HIST\_NODE\* cur | linked list 탐색 용 변수 |
| HIST\_NODE\* newHist | 새 HIST\_NODE 가 할당될 포인터 |

## 모듈 이름: histFree()

### 기능

History linked list에 할당된 메모리를 해제한다.

### 사용 변수

|  |  |
| --- | --- |
| HIST\_NODE\* cur | linked list 탐색 및 free 용도 변수 |
| HIST\_NODE\* nex | linked list 탐색용 변수 |

## 모듈 이름: dumpCMD( INPUT\_CMD )

### 기능

dump 명령어 입력 시 해당 기능을 수행한다. 다음과 같이 가상 메모리 dump의 역할을 한다:

dump -> 마지막 dump 주소부터 기본 160개의 가상 메모리 칸을 출력

dump A -> A 주소부터 기본 160 개의 가상 메모리 칸을 출력

dump A, B -> A 주소부터 B 주소까지 가상 메모리 칸을 출력

해당 칸 (가상 1 Byte) 의 내용을 16진수와 ASCII로 해석한 결과를 출력한다.

### 사용 변수

|  |  |
| --- | --- |
| INPUT\_CMD ipcmd | 함수 인자, parsing 된 입력 |
| static int start | 마지막 dump 주소 + 1을 저장, 초기값은 0이다 |
| int end | dump의 끝 주소 |

## 모듈 이름: editCMD( INPUT\_CMD )

### 기능

edit 명령어 입력 시 해당 기능을 수행한다. 첫 인자로 들어온 메모리 주소의 값을 두번째 인자로 들어온 값으로 바꾼다.

### 사용 변수

|  |  |
| --- | --- |
| INPU\_CMD ipcmd | 함수 인자, parsing 된 입력 |
| int add | 수정될 메모리의 주소 |
| int val | 해당 주소에 쓰여질 새로운 값 |

## 모듈 이름: fillCMD( INPUT\_CMD )

### 기능

fill 명령어 입력 시 해당 기능을 수행한다. 첫 인자로 들어온 메모리 주소부터 두번째 인자 주소까지 세번째 인자의 값으로 전체 바꾼다.

### 사용 변수

|  |  |
| --- | --- |
| INPU\_CMD ipcmd | 함수 인자, parsing 된 입력 |
| int start | fill 수행할 시작 주소 |
| int end | end 수행할 끝 주소 |
| int val | fill 할 값 |

## 모듈 이름: resetCMD()

### 기능

reset 명령어 입력 시 해당 기능을 수행한다. 가상 메모리 전체를 0x00 값으로 설정한다.

### 사용 변수

사용 변수 없음.

## 모듈 이름: opCMD( INPUT\_CMD )

### 기능

opcode 명령어 입력 시 해당 기능을 수행한다. 사전에 만들어진 opcode hash table에서 첫 인자로 입력된 opcode mnemonic을 찾아 해당 코드 값을 출력하고 못 찾을 경우 에러 메시지를 출력한다.

### 사용 변수

|  |  |
| --- | --- |
| INPU\_CMD ipcmd | 함수 인자, parsing 된 입력 |
| HASH\_ENTRY\* bucket | hash table의 각 element를 가리킬 변수 |

## 모듈 이름: oplistCMD()

### 기능

opcodelist 명령어 입력 시 해당 기능을 수행한다. 사전에 만들어진 opcode hash table 의 내용을 출력해준다.

### 사용 변수

|  |  |
| --- | --- |
| HASH\_ENTRY\* bucket | hash table의 각 element를 가리킬 변수 |

## 모듈 이름: hashCreate()

### 기능

shell 환경 수행 전 opcode.txt로부터 opcode hash table을 생성한다.

### 사용 변수

|  |  |
| --- | --- |
| FILE\* fp | 파일 포인터, opcode.txt를 가리킨다 |
| char cd[] | opcode.txt의 code 부분을 읽는다 |
| char ins[] | opcode.txt의 instruction 부분을 읽는다 |
| char md[] | opcode.txt의 mode 부분을 읽는다 |

## 모듈 이름: hashFunction(char\*)

### 기능

인자로 받은 instruction string으로부터 고유 hash 값을 계산하여 return 해준다.

### 사용 변수

|  |  |
| --- | --- |
| char\* inst | 함수 인자, opcode instruction string |

## 모듈 이름: hashAddBucket(int, HASH\_ENTRY\*)

### 기능

opcode.txt로부터 읽은 opcode를 hash table에 추가하는 함수이다. 크기 20인 hash table각 인덱스에 Linked list 형식으로 저장한다.

### 사용 변수

|  |  |
| --- | --- |
| int hash | 함수 인자, 해당 bucket의 hash 값 |
| HASH\_ENTRY\* bucket | 함수 인자, hash table에 추가할 bucket |
| HASH\_ENTRY\* cur | hash table 탐색용 포인터 |

## 모듈 이름: hashFree()

### 기능

Opcode hash table에 할당된 메모리를 해제한다.

### 사용 변수

|  |  |
| --- | --- |
| HASH\_ENTRY\* cur | hash table 탐색 및 free 용도 포인터 |
| HASH\_ENTRY\* nex | hash table 탐색용 포인터 |

# 전역 변수 정의

## COMMAND cmdList[CMD\_CNT]

이 shell 환경에서 인식할 수 있는 명령어와 각 명령어의 형식 등을 저장한 배열.

## HIST\_NODE\* histHead

입력된 명령어들의 기록이 저장될 history linked list의 첫 노드를 가리키는 주소.

## short mem[MEM\_SIZE]

시스템의 가상 메모리 (1 MB)로 활용될 공간.

## HASH\_ENTRY\* hashTable[HASH\_SIZE]

opcode.txt로부터 읽은 opcode가 저장될 hash table이다. 중복된 hash value가 있음으로 linked list 형태로 저장한다.

# 코드

## 20161577.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [1] SIC/XE Machine - The Basics \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: 20161577.h \*

\* File description: Main header file for the project. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#include <math.h>

#include <dirent.h>

#include <sys/types.h>

#include <sys/stat.h>

#define CMD\_LEN 257 // maximum length of input string

#define CMD\_CNT 11 // list of command formats

#define ARG\_MAX 3 // maximum argument count

#define MEM\_SIZE 1048576 // 2^20 = 1MB

#define HASH\_SIZE 20 // hash table size

typedef struct dirent ENTRY; // for dir command

typedef struct stat STBUF; // for dir command

typedef enum { false, true } bool; // workaround for bool type in C

// error codes, for better readability

typedef enum {

SAFE, FORMAT, HEX, VALUE

} ERROR\_CODE;

// command types, for better readability

typedef enum {

shell, memory, opcode, invalid

} CMD\_TYPE;

// command functions, for better readablitiy

typedef enum {

help, dir, quit, hist, dump, edit, fill, reset, op, oplist, invFormat, invHex, invVal

} CMD\_FUNC;

// command format structure

typedef struct {

char str[CMD\_LEN];

char abb[CMD\_LEN];

CMD\_TYPE type;

CMD\_FUNC func;

bool arg;

} COMMAND;

// user input command parsed structure

typedef struct {

CMD\_FUNC cmd;

short argCnt;

char arg[ARG\_MAX][10];

} INPUT\_CMD;

// history node structure

typedef struct HIST\_STRUCT {

char str[CMD\_LEN];

struct HIST\_STRUCT\* next;

} HIST\_NODE;

// hash table bucket structure

typedef struct HASH\_STRUCT {

int codeVal;

char code[3];

char inst[CMD\_LEN];

enum { m1, m2, m34 } mode;

struct HASH\_STRUCT\* next;

} HASH\_ENTRY;

int hexToDec(char\*); // function to check for vaild hex value and return converted decimal value

void hashFree(); // function to free hash table memory allocation

## 20161577.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [1] SIC/XE Machine - The Basics \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: 20161577.c \*

\* File description: Main file for the project. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "cmdProc.h"

#include "shell.h"

#include "memory.h"

#include "hash.h"

int main() {

char inp[CMD\_LEN]; // input string

char tmp[CMD\_LEN]; // temporary string to copy input

int i, j;

INPUT\_CMD input; // storage for parsed input

resetCMD(); // initialize memory

hashCreate(); // create hash table of opcodes

while(1) {

printf("sicsim> ");

fgets(inp, CMD\_LEN, stdin); // get input string

inp[strlen(inp) - 1] = '\0'; // replace \n with null character

// copy input string to tmp but place one space before and after comma ','

j = 0;

for(i = 0; inp[i]; i++) {

if(inp[i] == ',') {

strcpy(tmp + j, " , "); // place space around commma ','

j += 3;

}

else

tmp[j++] = inp[i];

}

tmp[j] = '\0';

input = findCMD(tmp); // find the command format from input string

if(input.cmd < invFormat)

histAdd(inp); // if command is not invalid add to history

// call function for each command

switch(input.cmd) {

case help:

helpCMD();

break;

case dir:

dirCMD();

break;

case quit:

quitCMD();

break;

case hist:

histCMD();

break;

case dump:

dumpCMD(input);

break;

case edit:

editCMD(input);

break;

case fill:

fillCMD(input);

break;

case reset:

resetCMD();

break;

case op:

opCMD(input);

break;

case oplist:

oplistCMD();

break;

case invFormat:

invFormatCMD();

break;

case invHex:

invHexCMD();

break;

case invVal:

invValCMD();

break;

}

}

return 0;

}

## cmdProc.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [1] SIC/XE Machine - The Basics \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: cmdProc.h \*

\* File description: Header file for input processing tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

INPUT\_CMD findCMD(char\*); // function to find command type of input

ERROR\_CODE testValidInput(INPUT\_CMD, COMMAND); // function to check validity of input command

void invFormatCMD(); // called when invalid format input

void invHexCMD(); // called when invalid hexadecimal value

void invValCMD(); // called when invalid value

## cmdProc.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [1] SIC/XE Machine - The Basics \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: cmdProc.c \*

\* File description: Tasks related to parse input by user. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "cmdProc.h"

// store command format in structure array to compare

COMMAND cmdList[CMD\_CNT] = {

{ "help", "h", shell, help, false }, { "dir", "d", shell, dir, false },

{ "quit", "q", shell, quit, false }, { "history", "hi", shell, hist, false },

{ "dump", "du", memory, dump, true }, { "edit", "e", memory, edit, true },

{ "fill", "f", memory, fill, true }, { "reset", "reset", memory, reset, false },

{ "opcode", "opcode", opcode, op, true }, { "opcodelist", "opcodelist", opcode, oplist, false },

{ "invalid", "invalid", invalid, invFormat, true }

};

INPUT\_CMD findCMD(char\* str) {

int i, j;

char delim[] = " \t\n"; // characters used to tokenize

char inp[CMD\_LEN];

char\* tok;

INPUT\_CMD ipcmd;

strcpy(inp, str); // copy input string

// initialize as invalid

ipcmd.cmd = invFormat;

ipcmd.argCnt = 0;

if(!strlen(str))

return ipcmd; // if empty string, return as invalid

tok = strtok(inp, delim); // first word of input

if(!tok)

return ipcmd; // emtpy token

for(i = 0; i < CMD\_CNT - 1; i++)

if(!strcmp(tok, cmdList[i].str) || !strcmp(tok, cmdList[i].abb)) {

ipcmd.cmd = cmdList[i].func; // if input command matches one of hard coded commands

break;

}

if(ipcmd.cmd == invFormat) // invalid command

return ipcmd;

// get arguments

j = 0;

while(tok) {

tok = strtok(NULL, delim); // next token (NOT expected comma if valid command)

if(!j && !tok) // no argument for command

break;

if((j && !tok) || tok[0] == ',') { // there was a previous argument but empty token or comma found

ipcmd.cmd = invFormat; //invalid command

return ipcmd;

}

strcpy(ipcmd.arg[j++], tok); // copy argument to input command structure

tok = strtok(NULL, delim); // next token (expected a comma if valid command)

if(tok && tok[0] != ',') { // if token not empty, expected a comma

ipcmd.cmd = invFormat;

return ipcmd;

}

}

ipcmd.argCnt = j; // save argument count

// after input string parsed, do further check for validity

switch(testValidInput(ipcmd, cmdList[i])) { // get error type, if any

case FORMAT:

ipcmd.cmd = invFormat;

break;

case HEX:

ipcmd.cmd = invHex;

break;

case VALUE:

ipcmd.cmd = invVal;

break;

default: // no error found

break;

}

return ipcmd;

}

ERROR\_CODE testValidInput(INPUT\_CMD ipcmd, COMMAND format) {

int i;

int arg[3];

ERROR\_CODE code = SAFE; // initialize as correct command

if(ipcmd.cmd == invFormat)

return FORMAT;

// check argument count

switch(ipcmd.cmd) {

// strictly 0 arguments

case help:

case dir:

case quit:

case hist:

case reset:

case oplist:

if(ipcmd.argCnt)

code = FORMAT;

break;

// strictly 1 argument

case op:

if(ipcmd.argCnt != 1)

code = FORMAT;

break;

// strictly 2 arguments

case edit:

if(ipcmd.argCnt != 2)

code = FORMAT;

break;

// strictly 3 arguments

case fill:

if(ipcmd.argCnt != 3)

code = FORMAT;

break;

// need less than 3

case dump:

if(ipcmd.argCnt > 2)

code = FORMAT;

break;

default:

break;

}

if(code == FORMAT)

return code;

// check hexadecimal number if command is memory-related

if(format.type == memory) {

for(i = 0; i < ipcmd.argCnt; i++)

if((arg[i] = hexToDec(ipcmd.arg[i])) == -1)

code = HEX;

// check with each command's criteria

switch(ipcmd.cmd) {

case edit:

if(arg[0] >= MEM\_SIZE || arg[1] > 255)

code = VALUE;

break;

case fill:

if(arg[0] >= MEM\_SIZE || arg[1] >= MEM\_SIZE || arg[0] > arg[1] || arg[2] > 255)

code = VALUE;

break;

case dump:

switch(ipcmd.argCnt) {

case 2:

if(arg[1] >= MEM\_SIZE || arg[0] > arg[1])

code = VALUE;

case 1:

if(arg[0] >= MEM\_SIZE)

code = VALUE;

break;

}

break;

default:

break;

}

}

return code;

}

void invFormatCMD() {

puts("ERROR: Invalid command.");

puts("Type \"help\" for list and formats of commands.");

}

void invHexCMD() {

puts("ERROR: Incorrect hexadecimal.");

}

void invValCMD() {

puts("ERROR: Invalid address.");

puts("Memory size:\t\t1MB [0x00000 ~ 0xFFFFF]");

puts("Edit/Fill value range:\t 1B [0x00 ~ 0xFF]");

}

int hexToDec(char\* hex) {

int i, dec = 0, multiplier = 1;

for(i = strlen(hex) - 1; i >= 0; i--) {

if(!isxdigit(hex[i])) // check if character if hexadecimal digit

return -1;

dec += multiplier \* (isdigit(hex[i]) ? (hex[i] - '0') : (toupper(hex[i]) - 'A' + 10));

multiplier \*= 16;

}

return dec;

}

## shell.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [1] SIC/XE Machine - The Basics \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: shell.h \*

\* File description: Header file for shell related tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

void helpCMD(); // COMMAND: help

void dirCMD(); // COMMAND: dir

void quitCMD(); // COMMAND: quit

void histCMD(); // COMMAND: history

void histAdd(char\*); // function to add input command into history

void histFree(); // function to free history linked list

## shell.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [1] SIC/XE Machine - The Basics \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: shell.c \*

\* File description: Tasks processed in shell environment. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "shell.h"

HIST\_NODE\* histHead = NULL; // head of history linked list

void helpCMD() {

printf("h[elp]\n"

"d[ir]\n"

"q[uit]\n"

"hi[story]\n"

"du[mp] [start, end]\n"

"e[dit] address, value\n"

"f[ill] start, end, value\n"

"reset\n"

"opcode mnemonic\n"

"opcodelist\n");

}

void dirCMD() {

DIR\* dir = opendir("."); // current directory

char\* entStr;

char path[258] = "./"; // entry path string

ENTRY\* ent; // entry

STBUF buf; // stat

if(!dir) {

puts("ERROR opening directory...");

return;

}

ent = readdir(dir); // read entry

while(ent) {

path[2] = '\0'; // clear path string

entStr = ent->d\_name; // entry name

stat(strcat(path, entStr), &buf);

printf("%-s", entStr); // print entry name

if(S\_ISDIR(buf.st\_mode)) // check for directory

printf("/");

else if(buf.st\_mode & S\_IXUSR) // check for exec file

printf("\*");

ent = readdir(dir); // read next entry

if(ent)

puts("");

}

closedir(dir);

puts("");

}

void quitCMD() {

puts("Exiting SIC...");

histFree(); // free history linked list

hashFree(); // free hash table

exit(0);

}

void histCMD() {

HIST\_NODE\* cur = histHead;

int cnt = 1;

while(cur) {

printf("%-3d ", cnt++);

puts(cur->str); // print command in history

cur = cur->next;

}

}

void histAdd(char\* str) {

HIST\_NODE\* cur = histHead;

HIST\_NODE\* newHist = malloc(sizeof(HIST\_NODE));

strcpy(newHist->str, str);

newHist->next = NULL;

if(!histHead) { // if history linked list is empty

histHead = newHist;

return;

}

while(cur->next)

cur = cur->next;

cur->next = newHist;

}

void histFree() {

HIST\_NODE\* cur = histHead;

HIST\_NODE\* nex;

while(cur) {

nex = cur->next;

free(cur);

cur = nex;

}

histHead = NULL;

}

## memory.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [1] SIC/XE Machine - The Basics \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: memory.h \*

\* File description: Header file for memory related tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

short mem[MEM\_SIZE]; // virtual memory (1MB)

void dumpCMD(INPUT\_CMD); // COMMAND: dump

void editCMD(INPUT\_CMD); // COMMAND: edit

void fillCMD(INPUT\_CMD); // COMMAND: fill

void resetCMD(); // COMMAND: reset

## memory.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [1] SIC/XE Machine - The Basics \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: memory.c \*

\* File description: Tasks related to virtual memory handing. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "memory.h"

void dumpCMD(INPUT\_CMD ipcmd) {

static int start = 0;

int end, i, j;

if(start >= MEM\_SIZE)

start = 0; // if start address exceeded memory limit, reset to 0x00000

end = start + 159; // set initial end value

if(ipcmd.argCnt) { // if there was argument entered in command

start = hexToDec(ipcmd.arg[0]);

end = start + 159;

if(ipcmd.argCnt == 2) // if there was 2 arguments

end = hexToDec(ipcmd.arg[1]);

}

if(end >= MEM\_SIZE)

end = MEM\_SIZE - 1; // if end address exceed memory limit, set to 0xFFFFF

// start loop from the beginning of each 16 Bytes

// finish loop till end of each 16 Bytes

for(i = start / 16 \* 16; i < (end / 16 + 1) \* 16; i++) {

if(!(i % 16))

printf("%05X ", i); // print address for beginning of each line

if(i < start || i > end)

printf(" "); // do NOT dump memory outside print range

else

printf("%02X ", mem[i]); // dump memory in hexadecimal value

if(!((i + 1) % 16)) { // line finished

printf("; ");

// print content converted into ASCII character

for(j = i - 15; j <= i; j++)

printf("%c", ((j >= start && j <= end) && mem[j] >= 32 && mem[j] <= 126) ? mem[j] : '.');

puts("");

}

}

start = end + 1; // remember last print address

}

void editCMD(INPUT\_CMD ipcmd) {

int add, val;

add = hexToDec(ipcmd.arg[0]); // address to edit

val = hexToDec(ipcmd.arg[1]); // replace value

mem[add] = val;

}

void fillCMD(INPUT\_CMD ipcmd) {

int i, start, end, val;

start = hexToDec(ipcmd.arg[0]); // fill start address

end = hexToDec(ipcmd.arg[1]); // fill end address

val = hexToDec(ipcmd.arg[2]); // fill value

for(i = start; i <= end; i++)

mem[i] = val;

}

void resetCMD() {

int i;

for(i = 0; i < MEM\_SIZE; i++)

mem[i] = 0; // set memory to 0x00000

}

## hash.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [1] SIC/XE Machine - The Basics \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: hash.h \*

\* File description: Header file for hash table related tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

HASH\_ENTRY\* hashTable[HASH\_SIZE]; // hash table pointer array

void opCMD(INPUT\_CMD); // COMMAND: opcode

void oplistCMD(); // COMMAND: opcodelist

void hashCreate(); // function to create hash table

int hashFunction(char\*); // function that returns hash function

void hashAddBucket(int, HASH\_ENTRY\*); // function to add bucket to hash table

## hash.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [1] SIC/XE Machine - The Basics \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: hash.c \*

\* File description: Tasks handling opcode and hash table. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "hash.h"

void opCMD(INPUT\_CMD ipcmd) {

HASH\_ENTRY\* bucket;

bucket = hashTable[hashFunction(ipcmd.arg[0])]; // get front bucket from hash function

while(bucket && strcmp(bucket->inst, ipcmd.arg[0])) // search till match or end

bucket = bucket->next;

if(bucket) // target found!

printf("opcode is %s\n", bucket->code);

else { // target NOT found..

puts("ERROR: mnemonic not found.");

puts("Type \"opcodelist\" for list of available opcodes.");

}

}

void oplistCMD() {

HASH\_ENTRY\* bucket;

int i;

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

printf("%2d : ", i); // print table index

bucket = hashTable[i]; // get front bucket

while(bucket) {

printf("[%s,%s]", bucket->inst, bucket->code);

if((bucket = bucket->next)) // if there exists next bucket

printf(" -> ");

}

puts("");

}

}

void hashCreate() {

FILE\* fp = fopen("opcode.txt", "r"); // open file

char cd[3], ins[10], md[4];

HASH\_ENTRY\* bucket;

if(!fp) {

puts("ERROR: Unable to load \"opcode.txt\".");

return;

}

while(fscanf(fp, "%s %s %s", cd, ins, md) == 3) {

bucket = malloc(sizeof(HASH\_ENTRY));

strcpy(bucket->code, cd);

strcpy(bucket->inst, ins);

bucket->codeVal = hexToDec(cd);

bucket->mode = md[0] - '1';

bucket->next = NULL;

hashAddBucket(hashFunction(bucket->inst), bucket);

}

if(fclose(fp)) {

puts("WARNING: Error closing \"opcode.txt\".");

return;

}

}

int hashFunction(char\* inst) {

return abs( (int) inst[0] \* 2 + abs(inst[0] + inst[1] + inst[2]) ) % HASH\_SIZE;

}

void hashAddBucket(int hash, HASH\_ENTRY\* bucket) {

HASH\_ENTRY\* cur = hashTable[hash];

if(!cur) { // if empty hash table index

hashTable[hash] = bucket;

return;

}

while(cur->next)

cur = cur->next; // go to the end of list

cur->next = bucket;

}

void hashFree() {

HASH\_ENTRY\* cur;

HASH\_ENTRY\* nex;

int i;

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

cur = hashTable[i]; // front bucket in each hash table index

while(cur) {

nex = cur->next;

free(cur);

cur = nex;

}

hashTable[i] = NULL; // reset pointer to NULL

}

}