과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

<<Project 1>>

**서강대학교 컴퓨터학과**

**[20161634]**

**[장수길]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**

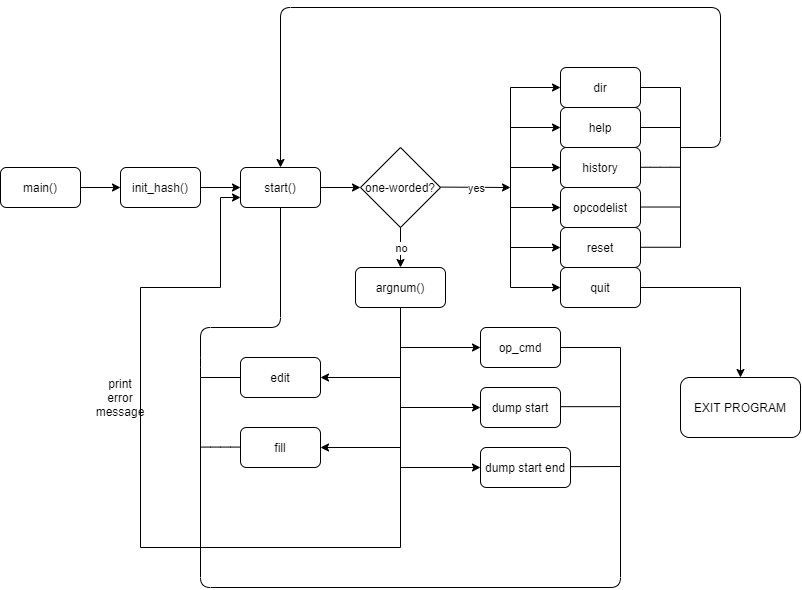
**2.1** 프로그램 흐름도

1. **구조체 및 헤더 정의**
2. **모듈 정의**
   1. 모듈 이름: int main(void)
      1. 기능
      2. 사용 변수
   2. 모듈 이름: void start()
      1. 기능
      2. 사용 변수
   3. 모듈 이름: int argnum(char\*cmd)
      1. 기능
      2. 사용 변수
   4. 모듈 이름: int hex(const char \*num)
      1. 기능
      2. 사용 변수
   5. 모듈 이름: void edit(unsigned char mem[], int address, int value)
      1. 기능
      2. 사용 변수
   6. 모듈 이름: void reset(unsigned char mem[])
      1. 기능
      2. 사용 변수
   7. 모듈 이름: void fill(unsigned char mem[], int start, int end, int value)
      1. 기능
      2. 사용 변수
   8. 모듈 이름: void print(unsigned char mem[], int start, int end)
      1. 기능
      2. 사용 변수
   9. 모듈 이름: void dump(unsigned char mem[], int args, ...)
      1. 기능
      2. 사용 변수
   10. 모듈 이름: char \*trim(char \*cmd)
       1. 기능
       2. 사용 변수
   11. 모듈 이름: bool checkDir(const char\* name)
       1. 기능
       2. 사용 변수
   12. 모듈 이름: void shell\_help(void)
       1. 기능
       2. 사용 변수
   13. 모듈 이름: void shell\_dir(void)
       1. 기능
       2. 사용 변수
   14. 모듈 이름: void history\_add(char\* command, struct NODE \*\*head)
       1. 기능
       2. 사용 변수
   15. 모듈 이름: void history\_print(struct NODE \*\*head)
       1. 기능
       2. 사용 변수
   16. 모듈 이름: int calc\_key(const char \*cmd)
       1. 기능
       2. 사용 변수
   17. 모듈 이름: void addHash(struct NODE \*node, struct NODE \*hash[])
       1. 기능
       2. 사용 변수
   18. 모듈 이름: init\_hash(struct NODE \*hash[])
       1. 기능
       2. 사용 변수
   19. 모듈 이름: op\_cmd(char \*mnemonic, struct NODE\*hash[], struct NODE \*\*head)
       1. 기능
       2. 사용 변수
   20. 모듈 이름: void op\_list(struct NODE \*hash[])
       1. 기능
       2. 사용 변수
3. **전역 변수 정의**
4. **코드**
   1. **defines.h**
   2. **20161634.h**
   3. **20161634.c**
   4. **mem\_cmd.h**
   5. **mem\_cmd.c**
   6. **shell\_cmd.h**
   7. **shell\_cmd.c**
   8. **opcode.h**
   9. **opcode.c**
5. **프로그램 개요**

SIC/XE 머신 구현을 위한 첫번째 단계이다.

Unix shell과 1MB의 가상 메모리, 그리고 opcode table을 구현하고 관련 명령어들에 대한 적절한 출력을 하는 프로그램이다.

1. **프로그램 설명**
   1. **프로그램 흐름도**



1. **구조체 및 헤더 정의**

모든 헤더 및 구조체는 defines.h 파일에 포함하였다.

|  |  |
| --- | --- |
| 헤더 | |
| #include <stdio.h> | 표준 입출력을 위한 헤더이다. |
| #include <string.h> | 문자열 관련 라이브러리 함수를 위한 헤더이다. |
| #include <stdbool.h> | Bool 자료형을 사용하기 위한 헤더이다. |
| #include <dirent.h> | 디렉토리 판별을 위한 헤더이다. |
| #include <sys/stat.h> | 디렉토리 판별을 위한 헤더이다. |
| #include <stdarg.h> | 가변 인자 함수 dump를 위한 헤더이다. |
| #include <ctype.h> | 공백 확인 함수인 isspace 함수를 사용하기 위한 헤더이다. |
| #include <stdlib.h> | 동적할당 및 memset 등 라이브러리 함수를 위한 헤더이다. |
| 구조체 | |
| struct NODE | 해시 테이블과 History 명령어 구현에 필요한 연결 리스트를 위한 구조체로, 문자열 하나와 정수형 변수 하나, 그리고 다음 노드를 가리키는 포인터를 저장한다. |

1. **모듈 정의**
   1. **모듈 이름 : int main(void)**
      1. *기능*가상 메모리와 해시 테이블을 구성하고, 무한루프를 통해 start 함수를 계속해서 호출하면서 사용자의 입력에 따라서 전용 함수들을 호출한다.
      2. *사용 변수*없음
   2. **모듈 이름: void start(void)**
      1. *기능*사용자의 입력을 받아 전처리하고, 한단어 입력의 경우 전용 함수를 호출한다.  
         fgets를 이용해 표준 입력버퍼의 한줄을 읽어오고, trim함수를 이용해 앞뒤 공백들을 제거해준다. 만약 앞뒤 공백을 제거한 결과가 dump, du, reset 등 한단어 명령어와 일치한다면 해당하는 함수들을 부른 뒤 종료한다.  
         해당하는 명령어가 없다면 여러 단어로 이루어진 명령어로 판단하고, argnum함수를 호출한다.
      2. *사용 변수*

|  |  |
| --- | --- |
| char buffer[MAX\_INPUT] | 사용자의 입력을 저장해주는 버퍼 |
| *char \*cmd* | 앞뒤 공백을 제거한 사용자의 입력을 저장해준다. |

* 1. **모듈 이름: int argnum(char \*cmd)**
     1. *기능*여러 개의 단어로 이루어진 입력을 처리하여 올바른 입력인지 확인하며, 적절한 입력일 경우 해당하는 함수를 호출해준다.  
        먼저 sscanf를 통해 opcode 관련 명령인지 확인한다. 두 단어로 이루어진 opcode 관련 명령일 경우, 뒤 mnemonic을 인자로 넘겨주면서 op\_cmd 함수를 호출한다. 해당하지 않을 경우 앞부분의 연속된 공백을 모두 제거해준 문자열을 추출하고, 가장 첫번째 단어를 확인해주는 것을 통해 명령 flag를 설정해준다.  
        이후 쉼표를 기준으로 문자열을 분리시켜주는데, 분리한 문자열들 또한 trim 함수를 통해 앞뒤 공백을 제거해준다. 제거해준 문자열을 hex함수를 이용하여 명령어에 따른 범위에 해당하는지 확인해주고, 적절한 값일 경우 인자 카운터를 올려주면서 콤마의 개수와 비례하는지 확인해준다.  
        최종적으로는 판별한 명령어와 인자의 개수에 따라서 해당하는 함수를 호출시켜주고, 앞서 언급한 과정에 해당하지 않는 케이스들은 적절하지 않은 입력이므로 -1을 반환시켜 경고문을 출력하도록 유도한다.
     2. *사용 변수*

|  |  |
| --- | --- |
| char what[100] | 사용자의 명령어 종류를 저장한다 (dump, fill, edit…) |
| char \*cmd | 앞뒤 공백을 제거한 사용자의 입력을 저장해준다 |
| int s | dump 명령어의 시작 주소를 저장한다 |
| int e | dump 명령어의 끝 주소를 저장한다 |
| int adr | edit 명령어의 주소를 저장한다 |
| int value | fill 및 edit 명령어의 변경할 값을 저장한다 |
| int space | 명령어 앞부분의 연속된 공백 개수를 저장한다 |
| int comma | 명령어의 쉼표 개수를 저장한다 |
| int args | 명령어의 인자 개수를 저장한다 |
| bool editflag, dumpflag, fillflag | 명령어의 종류를 나타낸다 |
| char history\_string[100] | history에 저장할 문자열을 저장한다 |
| int words | opcode명령을 확인하기 위해 단어개수를 체크해준다 |
| char \*tokenize | 앞 공백을 제거한 문자열의 포인터를 저장한다 |
| char \*ptr | 쉼표 기준으로 문자열을 나누어줄 포인터를 저장한다. |
| char hexStart[10] | 사용자가 입력한 형태의 시작주소 16진수를 저장한다. |
| char hexEnd[10] | 사용자가 입력한 형태의 끝주소 16진수를 저장한다. |
| char hexAddress[10] | 사용자가 입력한 형태의 주소16진수를 저장한다. |
| char hexValue[10] | 사용자가 입력한 형태의 16진수를 저장한다. |

* 1. **모듈 이름: int hex(const char \*num)**
     1. *기능*인자로 주어진 문자열이 올바른 범위내의 16진수인지 확인해주고, 해당하는 16진수값을 반환한다.
     2. *사용 변수*

|  |  |
| --- | --- |
| int ret | 16진수 값을 저장한다. |

* 1. **모듈 이름: void edit(unsigned char mem[], int address, int value)**
     1. *기능*주어진 주소의 메모리 값을 입력값으로 변경해준다.
     2. *사용 변수*없음
  2. **모듈 이름: void reset(unsigned char mem[])**
     1. *기능*메모리 전체를 0으로 초기화시켜준다.
     2. *사용 변수*없음
  3. **모듈 이름: void fill(unsigned char mem[], int start, int end, int value)**
     1. *기능*주어진 범위의 메모리를 입력값으로 변경해준다.
     2. *사용 변수*없음
  4. **모듈 이름: void print(unsigned char mem[], int start, int end)**
     1. *기능*주어진 범위의 메모리를 정해진 포맷으로 출력해준다.
     2. *사용 변수*

|  |  |
| --- | --- |
| int s | 포맷 형식에 맞게 시작범위를 재설정한 값 |
| int e | 포맷 형식에 맞게 종료범위를 재설정한 값 |

* 1. **모듈 이름: void dump(unsigned char mem[], int args, …)**
     1. *기능*가변 인자 함수로, dump의 인자 개수에 따라서 각각 다른 서브루틴을 수행시킨다. 두번째 인자는 인자의 개수를 의미하는데, 이 값에 따라서 뒤 인자들을 받을지 결정하고, 최종적으로 호출할 함수의 범위를 결정해준다.  
        올바르지 않은 범위를 인자로 받을 경우 에러 메시지를 호출하고 종료한다.
     2. *사용 변수*

|  |  |
| --- | --- |
| int num | 인자의 개수를 |
| int start | print함수에 전달할 출력 범위를 저장한다 |
| int end | print함수에 전달할 출력 범위를 저장한다. |

* 1. **모듈 이름: char\* trim(char \*cmd)**
     1. *기능*주어진 문자열의 앞부분과 뒷부분에 있는 연속된 공백 문자들 (‘\t’, ‘ ‘, ‘\n)  
        을 제거해준 문자열을 반환해준다. Ctypes.h 헤더에 내장되어있는 isspace 함수를 사용한다.   
        뒤에서부터 가장 먼저 나오는 공백이 아닌 문자의 바로 뒤 인덱스에 널문자를 삽입해주고, 앞에서부터 가장 먼저 나오는 공백이 아닌 문자의 인덱스를 포인터로 반환해준다.
     2. *사용 변수*

|  |  |
| --- | --- |
| int end | 문자열의 전체 길이를 저장한다. |

* 1. **모듈 이름: bool checkDir(const char\* name)**
     1. *기능*파일의 이름을 인자로 받아 해당하는 파일이 디렉토리인지 판별해준다.
     2. *사용 변수*

|  |  |
| --- | --- |
| struct stat status | 파일의 종류를 저장한다 |

* 1. **모듈 이름: void shell\_help(void)**
     1. *기능*가능한 명령어들을 나열해준다.
     2. *사용 변수*없음
  2. **모듈 이름: void shell\_dir(void)**
     1. *기능*

현재 디렉토리를 열고, 존재하는 모든 파일들을 한 번씩 확인한다. 확인하는 과정에서 파일의 이름을 checkDir함수에 넣어 디렉토리인지 확인하여 디렉토리일 경우 뒤에 ‘/’ 문자를 추가해주고, 마지막 4글자를 확인하여 실행 파일인지 확인해준다. 실행 파일일 경우, 마지막에 ‘\*’ 문자를 추가해준다.  
작업이 끝난 파일 이름을 출력해준다.

* + 1. *사용 변수*

|  |  |
| --- | --- |
| DIR \*dir\_i | 디렉토리를 저장한다 |
| dirent \*dir\_e | 현재 파일을 저장한다 |

* 1. **모듈 이름: void history\_add(char\* command, struct NODE \*\*head)**
     1. *기능*정제된 명령어를 history 연결 리스트에 추가해준다.
     2. *사용 변수*

|  |  |
| --- | --- |
| NODE \*newNode | 새롭게 추가할 노드의 정보를 저장한다 |
| NODE \*cur | 리스트의 끝까지 순회하기 위한 노드 |

* 1. **모듈 이름: void history\_print(struct NODE \*\*head)**
     1. *기능*현재까지 올바르게 수행된 명령어들의 기록을 출력한다.
     2. *사용 변수*

|  |  |
| --- | --- |
| int cnt | 명령어 번호 |
| NODE \*cur | 리스트의 끝까지 순회하기 위한 노드 |

* 1. **모듈 이름: int calc\_key(const char \*cmd)**
     1. *기능*주어진 mnemonic에 대응하는 해시값을 반환한다.  
        해시 함수는 mnemonic의 아스키 코드를 모두 더한 값을 해시 테이블의 크기로 나눈 값을 사용하였다.
     2. *사용 변수*

|  |  |
| --- | --- |
| int key | 해시함수의 반환값을 저장한다 |

* 1. **모듈 이름: void addHash(struct NODE \*node, struct NODE \*hash[])**
     1. *기능*해시 테이블에 노드를 추가해준다. 노드의 해시값을 먼저 계산해주고, 해당하는 인덱스의 해시 테이블에 노드를 추가시켜준다.
     2. *사용 변수*

|  |  |
| --- | --- |
| int key | 해시함수의 반환값을 저장한다 |

* 1. **모듈 이름: init\_hash(struct NODE \*hash[])**
     1. *기능*opcode.txt파일을 읽어들여 mnemonic과 해당하는 opcode들을 모두 해시 테이블로 구성해준다.
     2. *사용 변수*

|  |  |
| --- | --- |
| FILE \*fp | opcode.txt의 파일 포인터 |
| int cmd\_num | opcode 값을 저장한다 |
| char op[10] | mnemonic을 저장한다 |
| char format[10] | format을 저장한다 |

* 1. **모듈 이름: op\_cmd(char \*mnemonic, struct NODE\*hash[], struct NODE \*\*head)**
     1. *기능*사용자가 입력한 명령어가 해시 테이블에 존재하는지 확인해준다. 해시값을 계산해준 뒤 해당하는 인덱스의 연결 리스트를 조사하여 동일한 이름의 mnemonic이 존재하는지 확인해준 후, 일치할 경우 opcode값을 출력한다.  
        일치하는 mnemonic이 존재하지 않을 경우 에러 메시지를 출력 후 반환한다.
     2. *사용 변수*

|  |  |
| --- | --- |
| int key | 해시함수의 반환값을 저장한다. |
| char history\_string[100] | history에 저장할 문자열을 저장한다 |
| NODE \*cur | 순회용 노드이다 |

* 1. **모듈 이름: void op\_list(struct NODE \*hash[])**
     1. *기능*해시 테이블의 내용물들을 모두 출력한다.
     2. *사용 변수*

|  |  |
| --- | --- |
| NODE \*cur | 순회용 노드이다 |

1. **전역변수 정의**
   1. struct NODE \*head

history 출력을 위한 구조체 포인터이다

* 1. struct NODE \*hash[MAX\_HASH] (#MAX\_HASH = 20)

해시 테이블 구성을 위한 구조체 포인터의 배열이다

* 1. unsigned char m[2<<20]

1MB의 가상 메모리를 위한 배열 할당

* 1. int dump\_adr

단순 dump 명령어에서 adr를 유지하기 위한 변수

1. **코드**
   1. **defines.h**
2. #ifndef \_\_CHECK\_\_
3. #define \_\_CHECK\_\_
4. #include <stdarg.h>
5. #include <stdio.h>
6. #include <string.h>
7. #include <stdbool.h>
8. #include <dirent.h>
9. #include <sys/stat.h>
10. #include <ctype.h>
11. #include <stdlib.h>
12. struct NODE{
13. struct NODE \*next;
14. char format[10];
15. char cmd[100];
16. int num;
17. };
18. #endif
    1. **20161634.h**
19. #pragma once
20. #include <stdio.h>
21. #include <stdlib.h>
22. #include <string.h>
23. #define MAX\_INPUT 100
24. void start(void);
25. int argnum(char\* cmd);
26. int hex(const char\* num);

**6.3 20161634.c**

#include "20161634.h"

#include "shell\_cmd.h"

#include "mem\_cmd.h"

#include "opcode.h"

#include "defines.h"

#define MAX\_HASH 20

//head of linked list for history cmd

struct NODE \*head=NULL;

//hash table pointer

struct NODE \*hash[MAX\_HASH];

//works as 1MB virtual memory 1byte \* 2^20

unsigned char m[2<<20];

int dump\_adr=0;

int main(void){

    //resets virtual memory with value

    memset(m, 0, sizeof(m));

    //reads from opcode.txt and initiates hash table

    init\_hash(hash);

    while(1){

        start();

    }

    return 0;

}

/\*---------------------------------------------------------------------\*/

/\*Function: hex\*/

/\*Purpose: reads a hexadecimal string and returns positive decimal value\*/

/\*Return value: if invalid -1 / if valid positive decimal value\*/

/\*---------------------------------------------------------------------\*/

int hex(const char\* num){

    if(strlen(num)==0) return -1;

    for(int i=0;i<strlen(num);i++){

        if(!((num[i]>='0' && num[i]<='9') || (num[i]>='a' && num[i]<='f') || (num[i]>='A' && num[i]<='F'))){

            return -1;

        }

    }

    int ret;

    sscanf(num, "%x", &ret);

    return ret;

}

/\*---------------------------------------------------------------------\*/

/\*Function: argnum\*/

/\*Purpose: parses multi-word commands and calls appropriate functions depending on input\*/

/\*Return value: if invalid -1 / if valid 0 \*/

/\*---------------------------------------------------------------------\*/

int argnum(char\* cmd){

    char what[100];

    //variables for start end value address

    int s,e,val, adr;

    //variable to check arguments

    int args=1;

    //checks validity of input

    int space=0, comma=0;

    //sets flag according to command

    bool editflag=false, dumpflag=false, fillflag=false;

    bool spaceflag=false;

    s=e=val=adr=-1;

    //string to add to history list

    char history\_string[100];

    char op[100], mne[100], trash[100]; trash[0]='\0';

    int words=sscanf(cmd, "%s %s %s", op, mne, trash);

    if(words==2 && trash[0]=='\0' && !strcmp(op, "opcode")){

        //ex: opcode add

        //special case : need to check whether correct opcode, so add history inside function

        op\_cmd(mne, hash, &head); return 0;

    }

    //checks comma numbers

    for(int i=0;i<strlen(cmd);i++){

        if(!spaceflag && isspace(cmd[i])){

            space=i; spaceflag=true;

        }

        if(cmd[i]==',') comma++;

    }

    sscanf(cmd, "%s", what);

    char \*tokenize=cmd;

    tokenize+=space;

    char \*ptr;

    //decides command flags

    if(!strcmp("e", what)||!strcmp("edit", what)) editflag=true;

    if(!strcmp("du", what)||!strcmp("dump", what)) dumpflag=true;

    if(!strcmp("f", what)||!strcmp("fill", what)) fillflag=true;

    char hexStart[10], hexEnd[10],hexAddress[10], hexValue[10];

    //splits command by comma, and does boundary check and argument checks

    while((ptr=strsep(&tokenize,","))){

        int tempVal=hex(trim(ptr));

        //if invalid hexadecimal error

        if(tempVal==-1) return -1;

        if(editflag){

            //edit : check second argument range 00~FF

            //edit : first argument should be 00~FFFFF

            if(args==1 && 0<=tempVal && tempVal<=0xFFFFF){

                adr=tempVal; args++;

                strcpy(hexAddress, trim(ptr));

            }

            else if(args==2 && 0<=tempVal && tempVal<=0xFF){

                val=tempVal; args++;

                strcpy(hexValue, trim(ptr));

            }

            else return -1;

        }

        else if(dumpflag){

            //dump : all arguments should be 00~FFFFF

            if(0<=tempVal && tempVal<=0xFFFFF){

                if(args==1){

                    s=tempVal; args++;

                    strcpy(hexStart, trim(ptr));

                }

                else if(args==2){

                    e=tempVal; args++;

                    strcpy(hexEnd, trim(ptr));

                }

            }

            //if else invalid input ex: du 123A

            else return -1;

        }

        else if(fillflag){

            //fill : first and second argument (00~FFFFF) third argument (00~FF)

            //printf("tempval:%d\n", tempVal);

            if(args==1 && 0<=tempVal && tempVal<=0xFFFFF){

                s=tempVal; args++;

                strcpy(hexStart, trim(ptr));

            }

            else if(args==2 && 0<=tempVal && tempVal<=0xFFFFF){

                e=tempVal; args++;

                strcpy(hexEnd, trim(ptr));

            }

            else if(args==3 && 0<=tempVal && tempVal<=0xFF){

                val=tempVal; args++;

                strcpy(hexValue, trim(ptr));

            }

            else return -1;

        }

    }

    //command should be checked

    //all s,e,val should never be -1

    //comma number and argument number should match

    if(comma+2!=args) return -1;

    //call functions based on command and append to history list

    if(args==2 && (s!=-1) && (!strcmp(what, "du")||!strcmp(what,"dump")) ){

        sprintf(history\_string, "%s %s", what, hexStart);

        history\_add(history\_string, &head);

        dump(m,1,s); return 0;

    }

    if(args==3 && (s!=-1 && e!=-1) &&  (!strcmp(what, "du")||!strcmp(what,"dump"))){

        sprintf(history\_string, "%s %s, %s", what, hexStart,hexEnd);

        history\_add(history\_string, &head);

        dump(m,2,s,e); return 0;

    }

    if(args==4 && (s!=-1 && e!=-1 && val!=-1) && (!strcmp(what, "f")||!strcmp(what,"fill"))){

        sprintf(history\_string, "%s %s, %s, %s", what, hexStart,hexEnd,hexValue);

        history\_add(history\_string, &head);

        fill(m,s,e,val); return 0;

    }

    if(editflag && args==3 && (adr!=-1 && val!=-1)){

        sprintf(history\_string, "%s %s, %s", what, hexAddress, hexValue);

        history\_add(history\_string, &head);

        edit(m,adr, val); return 0;

    }

    return -1;

}

/\*---------------------------------------------------------------------\*/

/\*Function: start\*/

/\*Purpose: reads user input and handles single word inputs\*/

/\*Return: void\*/

/\*---------------------------------------------------------------------\*/

void start(void){

    printf("sicsim>");

    char buffer[MAX\_INPUT];

    if(fgets(buffer, sizeof(buffer), stdin)){

        size\_t len=strlen(buffer);

        if(len>0 && buffer[len-1]=='\n') buffer[len-1]='\0';

    }

    char \*cmd=trim(buffer);

    if(strlen(cmd)==0) return;

    //one word inputs (trivial) - call appropriate functions

    if(!strcmp(cmd, "help") || !strcmp(cmd, "h")){

        shell\_help();

        history\_add(cmd, &head); return;

    }

    if(!strcmp(cmd, "dir") || !strcmp(cmd, "d")){

        shell\_dir();

        history\_add(cmd, &head); return;

    }

    if(!strcmp(cmd, "quit") || !strcmp(cmd, "q")){

        exit(0);

    }

    if(!strcmp(cmd, "history") || !strcmp(cmd, "hi")){

        history\_add(cmd, &head);

        history\_print(&head); return;

    }

    if(!strcmp(cmd, "du")||!strcmp(cmd,"dump")){

        dump(m,1, dump\_adr); dump\_adr+=160;

        history\_add(cmd, &head); return;

    }

    if(!strcmp(cmd, "opcodelist")){

        history\_add(cmd, &head);

        op\_list(hash); return;

    }

    if(!strcmp(cmd, "reset")){

        history\_add(cmd, &head);

        reset(m); return;

    }

    //memory commands with multi word inputs

    if(argnum(cmd)==-1){

        printf("invalid input\n"); return;

    }

}

**6.4 mem\_cmd.h**

#include "defines.h"

void dump(unsigned char mem[], int args, ...);

void print(unsigned char mem[], int start, int end);

char \*intToHex(int n, int len);

void fill(unsigned char mem[], int start, int end, int value);

void edit(unsigned char mem[], int address, int value);

void reset(unsigned char mem[]);

**6.5 mem\_cmd.c**

#include "mem\_cmd.h"

/\*---------------------------------------------------------------------\*/

/\*Function: edit\*/

/\*Purpose: edits memory based on address and value\*/

/\*Return: void\*/

/\*---------------------------------------------------------------------\*/

void edit(unsigned char mem[], int address, int value){

    mem[address]=(char)value;

}

/\*---------------------------------------------------------------------\*/

/\*Function: reset\*/

/\*Purpose: resets whole memory to 0\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void reset(unsigned char mem[]){

    for(int i=0;i<=0xFFFFF;i++){

        mem[i]=0;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: fill\*/

/\*Purpose: assigns value to memory inside given address range\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void fill(unsigned char mem[], int start, int end, int value){

    //range overflow exception

    if(start>end) {

        printf("invalid range\n");

        return;

    }

    for(int i=start;i<=end;i++){

        mem[i]=value;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: print\*/

/\*Purpose: prints formatted contents of virtual memory\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void print(unsigned char mem[], int start, int end){

    //ex: du 15 57 (decimal)  -->  0 ~ 59

    int s=(start/16)\*16;

    int e=(end/16+1)\*16-1;

    //range overflow exception

    if(e>0xFFFFF) e=0xFFFFF;

    for(int i=s;i<=e;i+=16){

        //address | memory contents | ASCII CODE

        printf("%05X ", i);

        for(int j=i;j<i+16;j++){

            if(j<start || j>end) printf("   ");

            else{

                printf("%02X ", (int)mem[j]);

            }

        }

        printf("; ");

        for(int j=i;j<i+16;j++){

            if(j<start || j>end) printf(".");

            else if(mem[j]<0x20 || mem[j]>0x7E) printf(".");

            else printf("%c", mem[j]);

        }

        printf("\n");

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: dump\*/

/\*Purpose: variadic function - checks argument size and calls corresponding function\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void dump(unsigned char mem[], int args, ...){

    va\_list ap;

    va\_start(ap, args);

    int num;

    if(args==1){

        //if only dump or dump start

        int start,end;

        for(int i=0;i<args;i++){

            num=va\_arg(ap,int);

            if(i==0) start=num;

        }

        end=start+159;

        va\_end(ap);

        print(mem, start, end);

    }

    else if(args==2){

        //dump start end

        int start, end;

        for(int i=0;i<args;i++){

            num=va\_arg(ap, int);

            if(i==0) start=num;

            if(i==1) end=num;

        }

        va\_end(ap);

        //range exception when dump 4,3

        if(start>end){

            printf("range error!\n"); return;

        }

        print(mem, start, end);

    }

}

**6.6 shell\_cmd.h**

#include "defines.h"

void history\_add(char command[], struct NODE \*\*head);

void history\_print(struct NODE \*\*head);

void shell\_dir(void);

void shell\_help(void);

char\* trim(char \*cmd);

**6.7 shell\_cmd.c**

#include "shell\_cmd.h"

/\*---------------------------------------------------------------------\*/

/\*Function: trim\*/

/\*Purpose: removes leading and trailing spaces in string\*/

/\*Return: char pointer of trimmed string\*/

/\*---------------------------------------------------------------------\*/

char\* trim(char \*cmd){

    //trims leading and trailing spaces

    while(isspace(\*cmd))cmd++;

    if(\*cmd==0) return cmd;

    int end=strlen(cmd)-1;

    while(isspace(cmd[end])){

        end--;

    }

    cmd[end+1]='\0';

    return cmd;

}

/\*---------------------------------------------------------------------\*/

/\*Function: checkDir\*/

/\*Purpose: checks if given file name is directory\*/

/\*Return: true if directory / false if not directory \*/

/\*---------------------------------------------------------------------\*/

bool checkDir(const char\* name){

    //checks if given file name is a directory

    struct stat status;

    stat(name,&status);

    return S\_ISDIR(status.st\_mode);

}

/\*---------------------------------------------------------------------\*/

/\*Function: shell\_help\*/

/\*Purpose: Print available commands\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void shell\_help(void){

    printf("h[elp]\n");

    printf("d[ir]\n");

    printf("q[uit]\n");

    printf("du[mp] [start, end]\n");

    printf("e[dit] address, value\n");

    printf("f[ill] start, end, value\n");

    printf("reset\n");

    printf("opcode mnemonic\n");

    printf("opcodelist\n");

}

/\*---------------------------------------------------------------------\*/

/\*Function: shell\_dir\*/

/\*Purpose: prints files/directories in current directory\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void shell\_dir(void){

    DIR \*dir\_i=opendir(".");

    struct dirent \*dir\_e;

    if(dir\_i!=NULL){

        while((dir\_e=readdir(dir\_i))){

            char filename[100];

            //copy file name

            strcpy(filename, dir\_e->d\_name);

            int len=strlen(filename);

            //check if directory

            if(checkDir(dir\_e->d\_name)){

                filename[len]='/';

                filename[len+1]='\0';

            }

            //check if execution file (.out)

            const char \*ext=&filename[len-4];

            if(!strcmp(".out", ext)){

                filename[len]='\*';

                filename[len+1]='\0';

            }

            printf("%s\n", filename);

        }

    }

    closedir(dir\_i);

}

/\*---------------------------------------------------------------------\*/

/\*Function: history\_add\*/

/\*Purpose: adds command to history linked list\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void history\_add(char\* command, struct NODE \*\*head){

    struct NODE \*newNode=(struct NODE\*)malloc(sizeof(struct NODE));

    strcpy(newNode->cmd, command);

    newNode->next=NULL;

    struct NODE \*cur;

    if(\*head==NULL){

        \*head=newNode; return;

    }

    else{

        for(cur=(\*head);cur->next!=NULL;cur=cur->next){}

            cur->next=newNode;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: history\_print\*/

/\*Purpose: prints contents of linked list \*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void history\_print(struct NODE \*\*head){

    if(\*head==NULL) return;

    //prints contents of linked list

    int cnt=1;

    for(struct NODE \*cur=(\*head);cur!=NULL;cur=cur->next, cnt++){

        printf("%d %s\n", cnt, cur->cmd);

    }

}

**6.8 opcode.h**

#include "defines.h"

void init\_hash(struct NODE\*hash[]);

void op\_cmd(char \*mnemonic, struct NODE\*hash[], struct NODE \*\*head);

void op\_list(struct NODE\*hash[]);

**6.9 opcode.c**

#include "opcode.h"

#define MAX\_HASH 20

void history\_add(char\* command, struct NODE \*\*head);

/\*---------------------------------------------------------------------\*/

/\*Function: calc\_key\*/

/\*Purpose: calculates hash function\*/

/\*Return: integer value of hash key\*/

/\*---------------------------------------------------------------------\*/

int calc\_key(const char\*cmd){

    int key=0;

    for(int i=0;i<strlen(cmd);i++){

        key+=(int)cmd[i];

    }

    return key%MAX\_HASH;

}

/\*---------------------------------------------------------------------\*/

/\*Function: addHash\*/

/\*Purpose: adds a node to hash table\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void addHash(struct NODE \*node, struct NODE \*hash[]){

    int key=calc\_key(node->cmd);

    if(hash[key]==NULL){

        hash[key]=node;

    }

    else{

        node->next=hash[key];

        hash[key]=node;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: init\_hash\*/

/\*Purpose: reads opcode.txt and makes hash table\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void init\_hash(struct NODE \*hash[]){

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

    int cmd\_num;

    char op[10];

    char format[10];

    while(EOF!=fscanf(fp,"%x %s %s", &cmd\_num, op, format)){

        struct NODE \*newNode=(struct NODE\*)malloc(sizeof(struct NODE));

        strcpy(newNode->cmd, op);

        strcpy(newNode->format, format);

        newNode->num=cmd\_num;

        newNode->next=NULL;

        addHash(newNode, hash);

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: op\_cmd\*/

/\*Purpose: takes mnemonic input and prints corresponding opcode\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void op\_cmd(char \*mnemonic, struct NODE \*hash[], struct NODE \*\*head){

    int key=calc\_key(mnemonic);

    char history\_string[100];

    struct NODE\*cur=hash[key];

    while(cur!=NULL){

        if(!strcmp(cur->cmd, mnemonic)){

            sprintf(history\_string,"opcode %s", mnemonic);

            history\_add(history\_string, head);

            printf("opcode is %X\n", cur->num);

            return;

        }

        cur=cur->next;

    }

    printf("no such opcode\n"); return;

}

/\*---------------------------------------------------------------------\*/

/\*Function: op\_list\*/

/\*Purpose: Prints all opcodes in format\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void op\_list(struct NODE \*hash[]){

    for(int i=0;i<MAX\_HASH;i++){

        printf("%2d : ", i);

        if(hash[i]!=NULL){

            struct NODE \*cur=hash[i];

            while(cur!=NULL){

                printf("[%s, %X] ", cur->cmd, cur->num);

                cur=cur->next;

                if(cur!=NULL){

                    printf(" -> ");

                }

            }

        }

        printf("\n");

    }

}