과목명 : 시스템 프로그래밍

담당 교수명 : 김지환

<<Assignment 1>>

서강대학교 컴퓨터공학과

학번: 20171690

이름: 정유석

**목차**

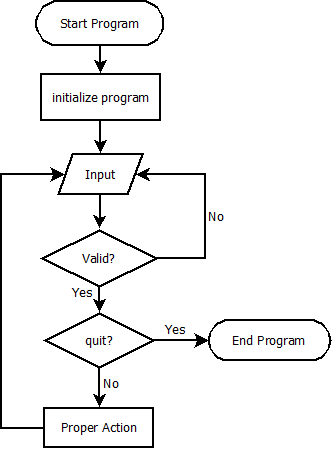
1. 프로그램 개요
2. 프로그램 설명
   1. 프로그램 흐름도
3. 분할 c 파일
   1. 20171690.c
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : main()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : init()
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : findCmd()
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : removeSpace()
            1. 기능
            2. 사용 변수
   2. shell.c
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : help()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : directory()
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : addHistory()
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : history()
            1. 기능
            2. 사용 변수
         5. 모듈 이름 : quit()
            1. 기능
            2. 사용 변수
   3. memory.c
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : dump()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : edit()
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : fill()
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : reset()
            1. 기능
            2. 사용 변수
         5. 모듈 이름 : validAddr()
            1. 기능
            2. 사용 변수
         6. 모듈 이름 : validAddrRange()
            1. 기능
            2. 사용 변수
         7. 모듈 이름 : strToHex()
            1. 기능
            2. 사용 변수
         8. 모듈 이름 : isHex()
            1. 기능
            2. 사용 변수
   4. opcode.c
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : opcode()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : opcodeList()
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : readOpTable()
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : hashFunction()
            1. 기능
            2. 사용 변수
4. 구조체 정의
   1. HISTORY
   2. opNode
5. 전역 변수 정의
   1. HISTORY\* hisHead
   2. opNode\*\* opTable
   3. unsigned char\* MEMORY
   4. int END\_ADDR
6. 코드

1. 프로그램 개요

SIC/XE Machine을 구현하기 전, 어셈블러, 링크, 로더들을 실행할 Shell을 구현, object 코드가 적재되고 실행될 메모리 공간 할당, mnemonic과 그에 상응하는 opcode를 저장하는 OPCODE table 생성 들을 수행하는 프로그램이다. 프로그램은 입력 프롬프트 상태로 시작되며 Shell, memory, opcode 관련 명령어들을 수행한다.

2. 프로그램 설명

2.1 프로그램 흐름도



< SIC Shell Program Flow Chart >

전체적인 프로그램 흐름도는 위와 같다. 프로그램을 실행하면 필요한 메모리를 할당하고 변수를 초기화한다. 입력을 받아 valid한 input이면 적절한 동작을 수행하고, invalid한 input이면 다시 입력을 받는다. 만약 프로그램 종료에 해당하는 input을 받으면 메모리를 해제하고 프로그램을 종료한다.

3. 분할 c파일

위에서 기술한 것과 같이 프로그램이 수행하는 명령어는 Shell 관련, 메모리 관련, opcode 관련 명령어로 크게 세 가지로 나눌 수 있다. 이러한 기준으로 c 파일을 분할하였다. main 함수가 포함된 20171690.c 파일과 각각 관련 명령어 및 관련 기능을 구현한 shell.c, memory.c, opcode.c 파일이 존재한다.

**3.1 20171690.c**

3.1.1 설명

main함수가 포함된 c 파일이다. 프로그램 시작 후 전역변수를 초기화한다. 그 후 shell 형태의 명령 프롬프트에서 입력을 받아 유효한 입력인지 판단하고, parsing하여 적절한 기능을 수행한다.

3.1.2 모듈 정의

3.1.2.1 모듈 이름 : **main()**

3.1.2.1.1 기능

명령 프롬프트에서 입력을 받는다. 입력받은 문자열에서 command와 parameter들을 적절히 parsing하여 해당하는 명령어에 대한 동작을 수행(함수 호출)한다. 이 때 invalid한 입력에 대해 error message를 출력하고 입력을 다시 받는다.

한편 프로그램이 수행되며 input 변수에 dummy 값이 잔존하여 오류가 발생하는 상황을 방지하기 위해 매 순간 메모리를 새로 할당하고, 해제한다.

3.1.1.1.2 사용 변수

**char\* input** – 명령 프롬프트에서 입력받은 문자열. 이 때 입력받는 문자열의 크기는 20171690.h 파일에 MACRO로 100으로 할당되었다.

**char\* input\_formed** – 입력받은 Input에서 앞, 뒤 white space를 제외한 문자열이다.

**char\* cmd** – input\_formed에서 strtok을 사용하여 tokenize한 후에 command를 가리키는 포인터이다.

**char \*params, \*start, \*end, \*addr, \*val** – cmd와 같이 input\_formed에서 각각 parameters, start 주소, end 주소, 주소, 값을 가리키는 포인터이다. 이 때 각각 변수는 주어진 명령어에 상응하는, 필요한 parameter에 대한 값만 가리키며 사용되지 않을 수 있고 잘못된 입력의 경우 원하는 값을 저장하지 못할 수도 있다.

**int before, after** – 해당 변수는 “dump” 명령어에서만 사용된다. comma(,)에 의한 잘못된 입력이 들어왔을 경우 comma를 기준으로 문자열을 token으로 잘라 자르기 전의 문자열 길이(before)와 자른 후의 문자열 길이(after)를 비교한다. 둘의 길이가 다른 경우는 “dump start, ”와 같이 잘못된 입력을 의미하므로 에러 메시지를 출력한다.

3.1.2.2 모듈 이름 : **init()**

3.1.2.2.1 기능

프로그램을 처음 실행시켰을 때 전역변수를 초기화하고 필요한 메모리를 할당한다. opcode에 대한 hash table도 이 때 생성한다.

3.1.2.2.2 사용 변수

없음

3.1.2.3 모듈 이름 : **findCmd(char\* cmd)**

3.1.2.3.1 기능

main 함수에서 입력받은 input을 토큰 단위(해당 프로그램에서는 공백)로 자른다. 이 때 input에서 command에 해당하는 문자열을 parameter로 입력받아 원하는 command에 대한 코드를 반환한다. 상응하는 command가 없는 경우 main에서 에러 메시지를 출력한다.

3.1.2.3.2 사용 변수

**char\* cmd** – input에서 parsing한 명령어의 시작 주소를 가리킨다.

3.1.2.4 모듈 이름 : **removeSpace(char\* input)**

3.1.2.4.1 기능

parameter로 전달받은 문자열에 대해 시작과 끝부분에 있는 white space(‘ ‘, ‘\t’, ‘\n’)를 제거한다. 이 때 문자열의 시작 주소를 white space가 아닌 character가 처음 나오는 위치로 지정하고, 끝부분에는 white space가 아닌 가장 마지막 character 다음 위치에 null 문자(‘\0’)를 저장하는 방식이다. return 형은 char\*이며 새로 지정한 문자열의 시작 주소를 반환한다.

문자열의 시작 주소가 null인 경우, 바로 시작주소(null)를 반환한다.

3.1.2.4.2 사용 변수

**char\* input** – 앞과 뒷부분의 white space를 삭제할 문자열의 시작 주소이다.

**3.2 shell.c**

3.2.1 설명

Shell 관련 명령어들과 관련된 필요한 기능들을 구현한 파일이다. 이번 프로젝트에서 구현해야 할 shell 관련 명령어는 h[elp], d[ir], hi[story], q[uit] 으로 4개이다.

3.2.2 모듈 정의

3.2.2.1 모듈 이름 : **help()**

3.2.2.1.1 기능

Shell에서 실행 가능한 모든 명령어들의 리스트를 화면에 출력한다. 단순 printf를 활용하여 출력하였다.

3.2.2.1.2 사용 변수

없음

3.2.2.2 모듈 이름 : **directory()**

3.2.2.2.1 기능

현재 디렉터리에 있는 모든 파일을 출력한다. 이 때 디렉터리는 ‘/’ 표시를, 실행파일은 ‘\*’ 표시를 한다.

3.2.2.2.2 사용 변수

**struct dirent\* dirEntry** – 현재 디렉터리에 속하는 파일들의 정보를 저장하는 dirent 구조체 포인터이다. 구조체의 정보 중 파일의 이름을 출력하기 위해 d\_name 변수를 사용한다.

**struct stat fileInfo** – 파일의 상태를 나타내는 stat 구조체 변수이다. 읽어온 파일이 디렉터리이거나 실행파일인 경우, 특별한 표시를 하기 위해 사용하였다.

**DIR\* dr** – 디렉터리 검색을 위한 DIR 구조체 포인터 변수이다. opendir(), readdir, closedir() 함수와 함께 사용한다.

3.2.2.3 모듈 이름 : **addHistory()**

3.2.2.3.1 기능

main 함수에서 입력을 받을 때, valid한 input이라면 해당 input을 원형 그대로 history에 저장한다. HISTORY 구조체와 전역 변수에 대한 자세한 설명은 각각 4.1, 5.1에 기술하였다.

3.2.2.3.2 사용 변수

**HISTORY\* pNew** – 새로운 history 구조체를 저장할 노드의 포인터이다. 새로운 노드는 linked list의 끝에 연결되며, head는 제일 먼저 들어온 node를 가리킨다.

3.2.2.4 모듈 이름 : **history()**

3.2.2.4.1 기능

shell 명령어 hi[story]를 입력받았을 때 실행된다. 현재까지 사용한 모든 명령어들은 addHistory 함수를 통해 순서대로 번호와 함께 HISTORY linked list에 저장된다. 이 list에 있는 모든 정보를 순차적으로 출력한다.

3.2.2.4.2 사용 변수

**HISTORY\* pMove** – 노드를 순차적으로 탐색하며 정보를 출력한다. 이 때 순차적으로 노드를 가리키는 구조체 포인터이다.

3.2.2.5 모듈 이름 : **quit()**

3.2.2.5.1 기능

shell 명령여 q[uit]를 입력받았을 때 실행된다. 프로그램을 종료하는 명령어에 해당되며, 따라서 할당하였던 모든 메모리를 해제한다. 이 때 해제하는 메모리는 init() 함수에서 생성했던 opcode table, 1024 byte가 할당된 MEMORY, 그리고 history가 저장된 history linked list이다.

3.2.2.5.2 사용 변수

**HISTORY\* pFree**

**opNode\* opFree, \*optmp**

세 변수 모두 linked list를 탐색하며 해당 노드를 free하기 위해 임시로 선언한 변수들이다.

**3.3 memory.c**

3.3.1 설명

assembler 및 linker와 loader를 통해 만들어진 object 파일을 올려 실행하게 될 1MB의 memory를 할당하였다. 이 때 이 메모리 공간에 대한 명령어를 구현한다. 프로젝트에서 구현해야하는 명령어는 du[mp], e[dit], f[ill], 그리고 reset의 4가지이다.

3.3.2 모듈 정의

3.3.2.1 모듈 이름 : **dump()**

3.3.2.1.1 기능

메모리 공간에 할당된 값과 그 주소를 출력한다. 이 때 가능한 parameter로는 start 주소, end 주소 두 가지가 있다. du[mp]만 입력했을 때 이전에 출력했던 마지막 주소값의 다음값부터 160 byte의 값을 출력한다. start 주소 하나를 입력받았을 때 입력받은 start 주소로부터 160 byte를 출력한다. start 주소와 end 주소를 모두 입력받았을 때 두 주소 사이의 값을 출력한다. 이 때 모든 경우에서 출력된 마지막 주소 값 + 1을 전역변수로 저장하여 다음 번 dump 명령어가 입력되었을 때 사용한다. 만약 마지막 주소(FFFFF)까지 출력을 하였다면 전역 변수 값은 0이 된다.

만약 입력받은 주소가 접근 불가능한 주소이거나, 두 주소의 범위가 올바르지 않다면 error message를 출력하고 0을 return한다. 입력받은 주소가 invalid 한 경우도 error message를 출력하고 0을 return한다. 성공적으로 기능을 수행했다면 1을 return한다.

3.3.2.1.2 사용 변수

**char\* start, \*end** – 함수 parameter로, 각각 16진수로 나타난 주소의 시작 값과 끝 값을 의미한다.

**int type** – input의 parameter 조합에 따라 구분한 type이다. parameter가 없는 경우 0, 1개인 경우 1, 2개인 경우 2로 지정한다.

**int s, e** – main 함수에서 입력받은 input을 parsing하여 전달한 parameter는 문자열이다. 따라서 문자열로 나타난 16진수를 정수로 저장하기 위한 변수이다. 입력받는 parameter의 조합에 따라 사용되지 않을 수도 있다.

**int tmp** – 전역 변수로 저장된 END\_ADDR의 값을 임시로 저장하는 변수이다. 16진수 문자열 parameter가 잘못된 주소를 가리키거나 타당하지 않은 주소일 때 이전의 END\_ADDR을 복원하기 위해 사용하였다.

**int i, j** – loop counter로 사용되었다.

**int startLineAddr, endLineAddr** – memory의 값을 출력할 때 각 line의 시작 주소를 0x10 단위로 표시해야한다. 이 때 출력될 line의 시작 주소와 끝 주소를 저장하는 변수이다.

**int line** – 총 출력할 line의 수를 저장한다.

**int addr** – 값을 출력할 때, 출력할 현재 위치의 주소값을 int로 저장한 값이다.

**char ASCIIcode[17]** – 한 line의 memory에 저장된 16개 값의 ASCII 값을 저장하는 문자열이다. 이 때 ASCII code로 출력해야할 범위 밖의 값은 ‘.’으로 저장한다.

3.3.2.2 모듈 이름 : **edit()**

3.3.2.2.1 기능

메모리의 address 번지에 저장된 값을 value에 지정된 값으로 변경한다. parameter로 변경할 값의 주소를 나타내는 address와 변경할 값인 value를 전달받는다. 접근할 수 없는 메모리 주소이거나, invalid한 주소일 경우 error message를 출력하고 0을 return한다. value 값을 1byte로 표현할 수 없는 경우 error message를 출력하고 0을 return한다. 오류가 없는 경우 입력받은 value 값을 해당 위치의 memory에 저장하고 1을 return한다.

3.3.2.2.2 사용 변수

**char\* address, \*value** – 함수 parameter로, 각각 16진수로 나타난 주소와 변경할 값을 의미한다.

**int addr** – parameter로 전달받은 address 주소값은 문자열이다. 이 문자열을 16진수 정수로 변환시켜 저장할 변수이다.

**int val** – addr과 마찬가지로 문자열로 저장된 value 값을 정수로 변환하여 저장한다.

**unsigned char realValue** – int로 저장한 value 값이 1 byte로 표현할 수 있는 타당한 16진수 값이라면, 그 값을 unsigned char type으로 저장한다.

3.3.2.3 모듈 이름 : **fill()**

3.3.2.3.1 기능

memory의 start 번지부터 end 번지까지의 값을 value로 변경한다. parameter는 변경할 memory 번지의 범위를 나타낼 start와 end, 그리고 변경할 값인 value를 전달받는다. 접근할 수 없는 memory 주소이거나 invalid한 주소, 혹은 범위일 경우 error message를 출력하고 0을 return한다. value 값이 1 byte로 표현할 수 없는 경우 error message를 출력하고 0을 return한다. 오류가 없는 경우 입력받은 value 값을 적절한 주소 번지 범위의 memory에 저장한다.

3.3.2.3.2 사용 변수

**char\* start, \*end, \*value** – 함수 parameter로, 각각 16진수로 나타난 주소 범위의 시작 값, 끝 값, 그리고 변경할 value 값이다.

**int i** – loop counter로 사용하였다.

**int s, e, v** – 문자열로 전달받은 parameter인 start, end, value 값을 각각 int로 저장한다.

**unsigned char realValue** – int로 저장한 value 값이 1byte로 표현할 수 있는 타당한 16진수 값이라면, 그 값을 unsigned char type으로 저장한다.

3.3.2.4 모듈 이름 : **reset()**

3.3.2.4.1 기능

memory 전체를 0으로 초기화한다.

3.3.2.4.2 사용 변수

**int i** – loop counter로 사용하였다.

3.3.2.5 모듈 이름 : **validAddr()**

3.3.2.5.1 기능

parameter로 전달받은 int addr이 타당한 memory 주소 범위 내에 있는지 확인한다. 0 ~ 0xFFFFF 사이의 값이라면 1을, 그렇지 않다면 0을 반환한다.

3.3.2.5.2 사용 변수

**int addr** – 함수 parameter로 확인할 주소를 나타낸다.

3.3.2.6 모듈 이름 : **validAddrRange()**

3.3.2.6.1 기능

parameter로 전달받은 두 개의 주소 int start와 int end가 타당한 주소 범위인지 확인한다. 두 주소가 모두 0 ~ 0xFFFFF 사이의 값이고, end >= start 라면 1을 반환한다. 그렇지 않다면 적절한 error message를 출력하고 0을 반환한다.

3.3.2.6.2 사용 변수

**int start, end** – 함수 parameter로 확인할 범위의 주소를 나타낸다.

3.3.2.7 모듈 이름 : **strToHex()**

3.3.2.7.1 기능

main 함수에서 입력받은 input은 문자열이다. 이 문자열을 parsing하여 전달받은 parameter들도 모두 문자열로 저장되어 있다. 따라서 해당 문자열을 16진수 정수로 변환하여 return하는 함수를 구현하였다.

만약 문자열에서 16진수(0 ~ F)가 아닌 문자가 포함되었을 경우 error message를 출력하고 -1을 반환한다. 16진수를 성공적으로 읽어오지 못해도 error message를 출력하고 -1을 반환한다. 그렇지 않을 경우 변환된 16진수 값을 반환한다.

3.3.2.7.2 사용 변수

**char\* param** – 함수 parameter로, 정수로 변환시킬 16진수 수가 나타난 문자열이다.

**int hex** – 문자열로 표현된 16진수를 저장한 정수형의 변수이다.

**int res** – sscanf로 param 문자열에서 16진수 수를 scan할 때, 성공적으로 받은 값의 개수를 의미한다. 1이 아닐 경우 error message를 출력한다.

3.3.2.8 모듈 이름 : **isHex()**

3.3.2.8.1 기능

parameter로 전달받은 character 하나가 16진수(0 ~ F)인지 아닌지 확인한다. 맞다면 1을, 아니면 0을 반환한다. 이 때 알파벳은 대문자와 소문자를 모두 인식한다.

3.3.2.8.2 사용 변수

**char ch** – 16진수인지 확인할 character이다.

**3.4 opcode.c**

3.4.1 설명

txt 파일로 제공된 Opcode와 mnemonic에 대한 작업을 수행한다. 프로젝트에서 구현한 명령어는 opcode, opcodelist 두 가지이다.

3.4.2 모듈 설명

3.4.2.1 모듈 이름 : **opcode()**

3.4.2.1.1 기능

명령어를 입력하면 해당하는 opcode를 출력한다. 이 때 mnemonic에 대한 opcode는 프로그램이 시작할 때 init() 함수(3.1.2.2 참고)에서hash table로 저장한다. 이 hash table을 탐색하여 적절한 opcode를 출력한다. 해당하는 mnemonic을 찾을 수 없다면 error message를 출력하고 0을 return한다. 성공적으로 탐색했다면 1을 return한다.

3.4.2.1.2 사용 변수

**int idx** – hash table에서 해당 mnemonic을 매핑하는 리스트의 헤드 인덱스를 가리킨다. 구현한 hash Function을 이용해 계산한다.

**opNode\* pMove** – hash table을 탐색할 때 사용할 노드 포인터이다.

3.4.2.2 모듈 이름 : **opcodeList()**

3.4.2.2.1 기능

opcode hash table에 저장된 mnemonic과 opcode를 모두 출력한다. 구현한 hash function에 따라 저장된 값이다. 만약 파일이 잘못되었다면 0을 return한다. 성공적으로 출력하였다면 1을 return한다.

3.4.2.2.2 사용 변수

**opNode\* pMove** – hash table을 탐색하며 내용을 출력할 때 사용한 노드 포인터이다.

3.4.2.3 모듈 이름 : **readOpTable()**

3.4.2.3.1 기능

주어진 opcode.txt 파일을 읽어 해당 파일의 내용을 hash table로 저장한다. 프로그램이 시작할 때 최초 1회 실행된다. 파일을 읽을 수 없다면 0을 return한다. 성공적으로 hash table에 저장했다면 1을 return한다.

3.4.2.3.2 사용 변수

**FILE\* fp** – 읽어올 파일을 가리키는 파일 포인터이다.

**ind idx** – hash function으로 계산한, mnemonic을 저장할 hash table의 인덱스를 나타낸다.

**int opcode** – 파일에서 읽어온 opcode를 저장하는 변수이다.

**char operation[7]** – 파일에서 읽어온 mnemonic을 저장하는 문자열이다.

**char formatStr[4]** – format을 나타내는 코드를 저장하는 문자열이다. ‘3/4’를 처리하기 위해 정수가 아닌 문자열로 저장하였다.

**opNode\* pNew** – 새로운 operation과 opcode 및 format을 저장할 새로운 노드이다.

3.4.2.4 모듈 이름 : **hashFunction()**

3.4.2.4.1 기능

opcode를 hash table에 저장하기 위해 구현한 hash functiond이다. 이 때 operation의 각 character에 대응하는 ASCII 코드를 모두 더해 20으로 나눈 나머지를 사용한다.

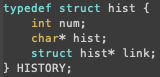
3.4.2.4.2 사용 변수

**int idx** – hash function으로 구현한 index를 저장하는 변수이다.

**int i** – loop counter로 사용되었다.

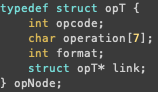
4. 구조체 정의

4.1 **HISTORY**



사용한 명령어를 저장하는 node로 구조는 위와 같다. 명령어와 그 명령어의 순서를 저장하는 변수인 num과 hist, 그리고 linked list로 구현하기 위한 노드 포인터 link로 구성되어있다.

4.2 **opNode**



opcode.txt 파일에서 읽어온 operation에 관한 정보를 저장하는 node이다. opcode와 format을 저장하는 int형 변수, operation 이름을 저장할 operation, 그리고 linked list로 구현하기 위한 node pointer인 link로 구성되어있다.

5. 전역 변수 정의

5.1 **HISTORY\* hisHead**

수행한 명령어들을 저장하는 linked list의 Head를 나타낸다. hisHead를 기준으로 명령어들이 순차적으로 저장되어있다.

5.2 **opNode\*\* opTable**

opcode의 hash table 시작주소를 나타낸다.

5.3 **unsigned char\* MEMORY**

구현할 1MB Memory의 시작주소를 나타낸다.

5.4 **END\_ADDR**

dump명령어를 사용하여 출력한 마지막 값의 주소를 나타낸다. dump 명령어에서 start parameter를 사용하지 않을 경우, END\_ADDR + 1번째 주소부터 값을 출력한다.END\_ADDR이 마지막 주소를 가리킬 경우, 0번지 값부터 출력한다.

**6. 코드**

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 20171690.h \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ~ 190325 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include <sys/types.h>

// MACROs

#define MEMORY\_SIZE 1<<20

#define COMMAND\_SIZE 100

#define OPCODE "opcode.txt"

#define ONE\_BYTE (unsigned char)0xFF

// STRUCTUREs

typedef struct hist {

int num;

char\* hist;

struct hist\* link;

} HISTORY;

typedef struct opT {

int opcode;

char operation[7];

int format;

struct opT\* link;

} opNode;

// GLOBAL VARIANTs

HISTORY\* hisHead;

opNode\*\* opTable;

unsigned char\* MEMORY;

int END\_ADDR;

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

void init();

int findCmd(char\*);

char\* removeSpace(char\*);

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SHELL COMMANDS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

void help(); // 0x00

void directory(); // 0x01

void history(); // 0x02

void quit(); // 0x03

void addHistory(char\*);

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MEMORY COMMANDS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

int dump(char\* start, char\* end, int type); // 0x10 ~ 0x12

int edit(char\* address, char \*value); // 0x13 ~ 0x14

int fill(char\* start, char\* end, char \*value); // 0x15

void reset(); // 0x16

int validAddr(int addr);

int validAddrRange(int start, int end);

int strToHex(char\* param);

int isHex(char ch);

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

\*\*\*\*\*\*\*\*\*\*\*\*\* OPCODE TABLE COMMANDS \*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

int opcode(char\* mnemonic); // 0x20

int opcodeList(); // 0x21

int readOpTable();

int hashFunction(char\* op);

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 20171690.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 190325 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

#include "20171690.h"

int main() {

init();

char\* input;

char\* input\_formed;

while (1) {

printf("sicsim> ");

// allocate memories

input = (char\*)malloc(COMMAND\_SIZE \* sizeof(char));

input\_formed = (char\*)malloc(COMMAND\_SIZE \* sizeof(char));

// get input

fflush(stdin);

fgets(input, COMMAND\_SIZE, stdin);

if ((int)strlen(input) > COMMAND\_SIZE) {

printf("Too long command!\n");

continue;

}

// remove white spaces

strcpy(input\_formed, removeSpace(input));

char\* cmd = strtok(input\_formed, " \t");

char\* params;

char\* start, \*end, \*addr, \*val;

int before, after;

// if input is white spaces, continue;

if (cmd) {

switch (findCmd(cmd)) {

case 0x00: // h[elp] done

if ( (params = strtok(NULL, "\0")) ) {

printf("h[elp]: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

addHistory(input);

help();

break;

case 0x01: // d[ir] done

if ( (params = strtok(NULL, "\0")) ) {

printf("d[ir]: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

directory();

addHistory(input);

break;

case 0x02: // hi[story] done

if ( (params = strtok(NULL, "\0")) ) {

printf("hi[story]: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

addHistory(input);

history();

break;

case 0x03: // q[uit] done

if ( (params = strtok(NULL, "\0")) ) {

printf("q[uit]: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

free(input);

free(input\_formed);

quit();

return 0;

case 0x10: case 0x11: case 0x12: // du[mp] [start, end]

params = strtok(NULL, "\0");

if (!params) {

// "dump"

addHistory(input);

dump(NULL, NULL, 0);

break;

}

// parameter(s) follow(s)

before = (int)strlen(params);

start = removeSpace(strtok(params, ","));

after = (int)strlen(params);

end = strtok(NULL, "\0");

if (!end) {

if (before != after) {

// "dump start, "

printf("Invalid Syntax. See 'h[elp]'\n");

break;

}

if (!dump(start, NULL, 1)) {

// inavalid addr

break;

}

// "dump start"

addHistory(input);

break;

}

// "dump start, end"

end = removeSpace(end);

if (!dump(start, end, 2)) {

// invalid addr

break;

}

addHistory(input);

break;

case 0x13: case 0x14: // e[dit] address, value

params = strtok(NULL, "\0");

if (!params) {

// no parameters

printf("Syntax Error. See 'h[elp]'\n");

break;

}

addr = removeSpace(strtok(params, ","));

val = strtok(NULL, "\0");

if (!val) {

// no value + invalid sytax

printf("Syntax Error. See 'h[elp]'\n");

break;

}

val = removeSpace(val);

if (!edit(addr, val)) {

// invalid addr & val

break;

}

addHistory(input);

break;

case 0x15: // f[ill] start, end, value

params = strtok(NULL, "\0");

if (!params) {

// No parameters at all

printf("Syntax Error. See 'h[elp]'\n");

break;

}

start = removeSpace(strtok(params, ","));

end = strtok(NULL, ",");

if (!end) {

// "fill start, "

printf("Syntax Error. See 'h[elp]'\n");

break;

}

end = removeSpace(end);

val = strtok(NULL, "\0");

if (!val) {

// "fill start, end, "

printf("Syntax Error. See 'h[elp]'\n");

break;

}

val = removeSpace(val);

if (!fill(start, end, val)) {

// invalid addr & val

break;

}

addHistory(input);

break;

case 0x16: // reset

if ( (params = strtok(NULL, "\0")) ) {

printf("reset: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

reset();

addHistory(input);

printf("Reset Successfully\n");

break;

case 0x20: // opcode mnemonic //done

params = strtok(NULL, "\0");

if (!params) {

// no mnemonic

printf("Invalid Syntax. See 'h[elp]'\n");

break;

}

params = removeSpace(params);

if (opcode(params))

addHistory(input);

else

printf("%s: No matching mnemonic\n", params);

break;

case 0x21: // opcodelist done

if ( (params = strtok(NULL, "\0")) ) {

printf("opcodelist: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

if (!opcodeList())

printf("There's no opcode.txt in the directory\n");

else

addHistory(input);

break;

case 0x30:

printf("command not found: %s\n", cmd);

break;

}

}

free(input);

free(input\_formed);

}

return 0;

}

void init() {

// allocate memories

hisHead = NULL;

opTable = (opNode\*\*)malloc(20 \* sizeof(opNode\*));

for (int i = 0; i < 20; i++)

opTable[i] = NULL;

MEMORY = (unsigned char\*)calloc(MEMORY\_SIZE, sizeof(unsigned char));

END\_ADDR = 0;

readOpTable();

}

int findCmd(char\* cmd) {

// get command as a parameter, return proper cmd

if (strcmp(cmd, "help") == 0 || strcmp(cmd, "h") == 0) return 0x00;

if (strcmp(cmd, "dir") == 0 || strcmp(cmd, "d") == 0) return 0x01;

if (strcmp(cmd, "history") == 0 || strcmp(cmd, "hi") == 0) return 0x02;

if (strcmp(cmd, "quit") == 0 || strcmp(cmd, "q") == 0) return 0x03;

if (strcmp(cmd, "dump") == 0 || strcmp(cmd, "du") == 0) return 0x10;

if (strcmp(cmd, "edit") == 0 || strcmp(cmd, "e") == 0) return 0x13;

if (strcmp(cmd, "fill") == 0 || strcmp(cmd, "f") == 0) return 0x15;

if (strcmp(cmd, "reset") == 0) return 0x16;

if (strcmp(cmd, "opcode") == 0) return 0x20;

if (strcmp(cmd, "opcodelist") == 0) return 0x21;

return 0x30;

}

char\* removeSpace(char\* input) {

// remove Spaces at front & at the end

if (!input) return input;

int i;

for (i = 0; input[i] == ' ' || input[i] == '\t' || input[i] == '\n'; i++) ;

input = input + i;

i = 0;

while (input[strlen(input) - 1 - i] == ' ' || input[strlen(input) - 1 - i] == '\n' || input[strlen(input) - 1 - i] == '\t') i++;

input[strlen(input) - i] = '\0';

return input;

}

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* shell.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 190325 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

#include "20171690.h"

void help() {

printf("\t\th[elp]\n\t\td[ir]\n\t\tq[uit]\n\t\thi[story]\n");

printf("\t\tdu[mp] [start, end]\n");

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

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

printf("\t\treset\n");

printf("\t\topcode mnemonic\n\t\topcodelist\n");

}

void directory() {

struct dirent\* dirEntry; // directory entry Pointer

struct stat fileInfo;

DIR\* dr = opendir(".");

while ( (dirEntry = readdir(dr)) != NULL) {

printf("\t\t%s", dirEntry->d\_name);

stat(dirEntry->d\_name, &fileInfo);

if (S\_ISDIR(fileInfo.st\_mode)) printf("/");

else if (fileInfo.st\_mode & S\_IXUSR) printf("\*");

printf("\n");

}

closedir(dr);

}

void addHistory(char\* input) {

HISTORY\* pNew = (HISTORY\*)malloc(sizeof(HISTORY));

pNew->hist = (char\*)malloc(COMMAND\_SIZE \* sizeof(char));

strcpy(pNew->hist, input);

pNew->link = NULL;

if (!hisHead) {

hisHead = pNew;

pNew->num = 1;

return ;

}

HISTORY\* pMove;

for (pMove = hisHead; pMove->link; pMove = pMove->link) ;

pMove->link = pNew;

pNew->num = pMove->num + 1;

}

void history() {

HISTORY\* pMove;

for (pMove = hisHead; pMove; pMove = pMove->link) {

printf("\t%d\t%s\n", pMove->num, pMove->hist);

}

}

void quit() {

// free memories

HISTORY\* pFree;

while (hisHead) {

pFree = hisHead;

hisHead = hisHead->link;

free(pFree);

}

opNode\* opFree, \*optmp;

for (int i = 0; i < 20; i++) {

optmp = opTable[i];

while (optmp) {

opFree = optmp;

optmp = optmp->link;

free(opFree);

}

}

free(MEMORY);

printf("exit shell\n");

}

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* memory.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 190325 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

#include "20171690.h"

int dump(char\* start, char\* end, int type) {

// du[mp] [start, end]

int s, e;

int tmp = END\_ADDR;

switch (type) {

case 0: // du[mp]

if (END\_ADDR > 0xFFFFF) END\_ADDR = 0;

s = END\_ADDR;

e = ((END\_ADDR + 0x9F) > 0xFFFFF) ? 0xFFFFF : END\_ADDR + 0x9F;

END\_ADDR = e + 0x1;

break;

case 1: // du[mp] start

s = strToHex(start);

e = ((s + 0x9F) > 0xFFFFF) ? 0xFFFFF : s + 0x9F;

END\_ADDR = e + 0x1;

break;

case 2: // du[mp] start, end

s = strToHex(start);

e = strToHex(end);

END\_ADDR = e + 0x1;

break;

}

// incorrect input or addr and val are not hex

if (s == -1 || e == -1) {

printf("Incorrect input. Please text again\n");

END\_ADDR = tmp;

return 0;

}

if (!validAddrRange(s, e)) {

END\_ADDR = tmp;

return 0;

}

int i, j;

int startLineAddr = s / 16 \* 16;

int endLineAddr = e / 16 \* 16;

int line = (endLineAddr - startLineAddr) / 16 + 1;

int addr;

char ASCIIcode[17];

ASCIIcode[16] = '\0';

for (i = 0; i < line; i++) {

printf("%05X ", startLineAddr + i \* 16);

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

addr = startLineAddr + i \* 16 + j;

if (addr < s) {

printf(" ");

ASCIIcode[j] = '.';

}

else if (addr > e) {

printf(" ");

ASCIIcode[j] = '.';

}

else {

printf("%02X ", \*(MEMORY + addr));

if (\*(MEMORY + addr) >= 0x20 && \*(MEMORY + addr) <= 0x7E) ASCIIcode[j] = \*(MEMORY + addr);

else ASCIIcode[j] = '.';

}

}

printf("; %s\n", ASCIIcode);

}

return 1;

}

int edit(char\* address, char\* value) {

// e[dit] address, value

int addr = strToHex(address);

int val = strToHex(value);

// incorrect input or addr and val are not hex

if (addr == -1 || val == -1) {

printf("Incorrect input. Please text again\n");

return 0;

}

if (val > 0xFF) {

printf("Invalid value. Too big!\n");

return 0;

}

// then edit memory

unsigned char realValue = val & ONE\_BYTE;

memcpy(MEMORY + addr, &realValue, 1);

return 1;

}

int fill(char\* start, char\* end, char\* value) {

// f[ill] start, end, value

int s, e, v;

s = strToHex(start);

e = strToHex(end);

v = strToHex(value);

// incorrect input or addr and val are not hex

if (s == -1 || e == -1 || v == -1) {

printf("Incorrect input. Please text again\n");

return 0;

}

if (!validAddrRange(s, e)) {

return 0;

}

if (v > 0xFF) {

printf("Invalid value. Too big!\n");

return 0;

}

unsigned char realValue = v & ONE\_BYTE;

for (int i = s; i <= e; i++)

memcpy(MEMORY + i, &realValue, 1);

return 1;

}

void reset() {

// reset

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

MEMORY[i] = 0x00;

}

int validAddr(int addr) {

// if given address is vaild, return 1, else return 0

if (addr >= 0 && addr <= 0xFFFFF)

return 1;

else

return 0;

}

int validAddrRange(int start, int end) {

if (!validAddr(start)) {

printf("Segmentation fault: Cannot access %05X\n", start);

return 0;

}

else if (!validAddr(end)) {

printf("Segmentation fault: Cannot access %05X\n", end);

return 0;

}

else if (end < start) {

printf("Incorrect Address Range. End address must be bigger than Start address.\n");

return 0;

}

return 1;

}

int strToHex(char\* param) {

// if parameter is not a hex, or incorrect parser return -1

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

if ( !isHex(param[i]) ) {

printf("Address is not a Hexadecimal!\n");

return -1;

}

}

// else, return proper hexa decimal number

int hex;

int res = sscanf(param, "%x", &hex);

if (res == 0) {

printf("error occured. Please type again\n");

return -1;

}

return hex;

}

int isHex(char ch) {

if ( (ch >= 'A' && ch <= 'F') || (ch >= 'a' && ch <= 'f') || (ch >= '0' && ch <= '9') )

return 1;

else

return 0;

}

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* opcode.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 190325 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

#include "20171690.h"

int opcode(char\* mnemonic) {

// if found correct mnemonic, return 1, else return 0

int idx = hashFunction(mnemonic);

opNode\* pMove;

for (pMove = opTable[idx]; pMove; pMove = pMove->link) {

if (strcmp(pMove->operation, mnemonic) == 0) {

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

return 1;

}

}

printf("Wrong mnemonic\n");

return 0;

}

int opcodeList() {

if (!opTable) return 0;

opNode\* pMove;

for (int i = 0; i < 20; i++) {

printf("%d : ", i);

pMove = opTable[i];

if (pMove) {

for ( ; pMove->link; pMove = pMove->link)

printf("[%s,%X] -> ", pMove->operation, pMove->opcode);

printf("[%s,%X]", pMove->operation, pMove->opcode);

}

printf("\n");

}

return 1;

}

int readOpTable() {

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

if (!fp) {

printf("wrong input file\n");

return 0;

}

int idx;

int opcode;

char operation[7];

char formatStr[4];

while (fscanf(fp, "%X %s %s\n", &opcode, operation, formatStr) != EOF) {

opNode\* pNew = (opNode\*)malloc(sizeof(opNode));

pNew->opcode = opcode;

strcpy(pNew->operation, operation);

if (strcmp(formatStr, "3/4") == 0) pNew->format = 3;

else if (strcmp(formatStr, "1") == 0) pNew->format = 1;

else if (strcmp(formatStr, "2") == 0) pNew->format = 2;

else pNew->format = 0;

pNew->link = NULL;

idx = hashFunction(operation);

if (!opTable[idx])

opTable[idx] = pNew;

else {

opNode\* pMove;

for (pMove = opTable[idx]; pMove->link; pMove = pMove->link) ;

pMove->link = pNew;

}

}

fclose(fp);

return 1;

}

int hashFunction(char\* op) {

int idx = 0;

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

idx += op[i];

return idx % 20;

}

# makefile

CC = gcc

CFLAGS = -Wall -W -g

OBJECTS = 20171690.o shell.o memory.o opcode.o

TARGET = 20171690.out

all: $(TARGET)

$(TARGET) : $(OBJECTS)

$(CC) $(CFLAGS) -o $(TARGET) $(OBJECTS)

clean:

rm -rf $(OBJECTS) $(TARGET) $(TARGET).dSYM