과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 1>>

**서강대학교 컴퓨터학과**

**[20161601]**

**[송인아]**

목 차

1. 프로그램 개요 4

2. 프로그램 설명 5

2.1 프로그램 흐름도 5

3. 모듈 정의 7

3.1 모듈 이름 : prompt() 7

3.1.1 기능 7

3.1.2 사용 변수 8

3.2 모듈 이름: print\_error(int error) 8

3.2.1 기능 8

3.2.2 사용 변수 9

3.3 모듈이름: insert(char cmm[COMMAND\_MAX]) 9

3.3.1 기능 9

3.3.2 사용변수 9

3.4 모듈이름: remove\_space(char\* s) 9

3.4.1 기능 9

3.4.2 사용변수 10

3.5 모듈이름: one\_digit\_to\_int(char c) 10

3.5.1 기능 10

3.5.2 사용변수 10

3.6 모듈 이름 : make\_hex(char\* s) 10

3.6.1 기능 10

3.6.2 사용 변수 11

3.7 모듈 이름: hash\_key(char\* s) 11

3.7.1 기능 11

3.7.2 사용 변수 11

3.8 모듈이름: create\_hash\_table() 11

3.8.1 기능 11

3.8.2 사용변수 12

3.9 모듈이름: command\_h() 12

3.9.1 기능 12

3.9.2 사용변수 12

3.10 모듈이름: command\_d() 12

3.10.1 기능 12

3.10.2 사용변수 12

3.11 모듈이름: command\_q() 13

3.11.1 기능 13

3.11.2 사용변수 13

3.12 모듈이름: command\_hi() 13

3.12.1 기능 13

3.12.2 사용변수 13

3.13 모듈이름: command\_du(char cmm[10][COMMAND\_MAX]) 14

3.13.1 기능 14

3.13.2 사용변수 15

3.14 모듈이름: command\_e(char cmm[10][COMMAND\_MAX]) 15

3.14.1 기능 15

3.14.2 사용변수 16

3.15 모듈이름: command\_f(char cmm[10][COMMAND\_MAX]) 16

3.15.1 기능 16

3.15.2 사용변수 17

3.16 모듈이름: command\_reset() 17

3.16.1 기능 17

3.16.2 사용변수 17

3.17 모듈이름: command\_op\_mn(char\* s) 18

3.17.1 기능 18

3.17.2 사용변수 18

3.18 모듈이름: command\_op\_list() 18

3.18.1 기능 18

3.18.2 사용변수 18

4. 전역 변수 정의 19

4.1 op\* oplist[HASH\_MAX] 19

4.2 ll\* hlist 19

4.3 char memory[MEMORY\_ROW][MEMORY\_COL] 20

4.4 int cur\_adr 20

# 프로그램 개요

앞으로 구현할 SIC/XE 머신을 만들기 위해 필요한 전단계로, 어셈블러와 링크, 로더들을 실행하게 될 Shell을 구현하고, 여기서 제공하는 명령어들의 기능이 제대로 수행될 수 있도록 프로그램을 작성한다. 또한 컴파일을 통해서 만들어질 object 코드가 저장되고 실행될 메모리 공간과 문자열로 mnmonic을 입력하면 이를 opcode 값으로 변환하는 opcode table을 구현한다.

구현해야 할 기능은 다음과 같다.

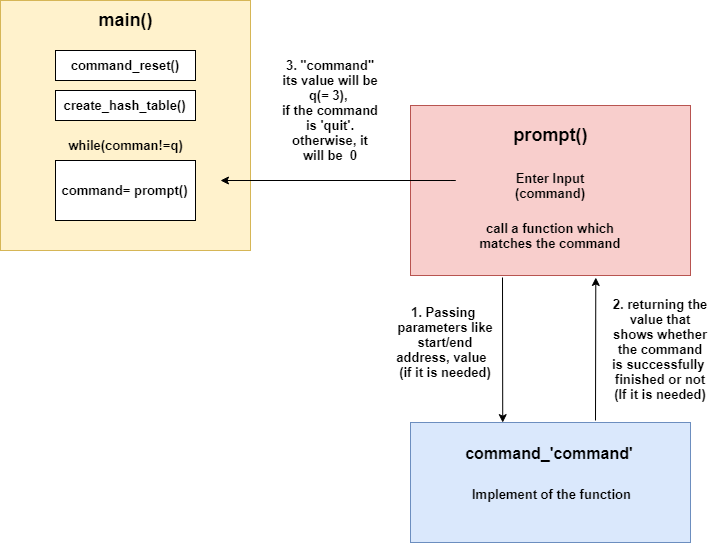
1. 셸 ( sicsim> )
2. 셸 관련 명령어들 (help, dir, quit, history)
3. 메모리공간 (1MB의 메모리를 할당해서 사용)
4. 메모리공간 관련 명령어들 (dump, edit, fill, reset)
5. opcode 테이블 (HashTable로 만들어야 함)
6. opcode 관련 명령어들 (opcode, opcodelist)

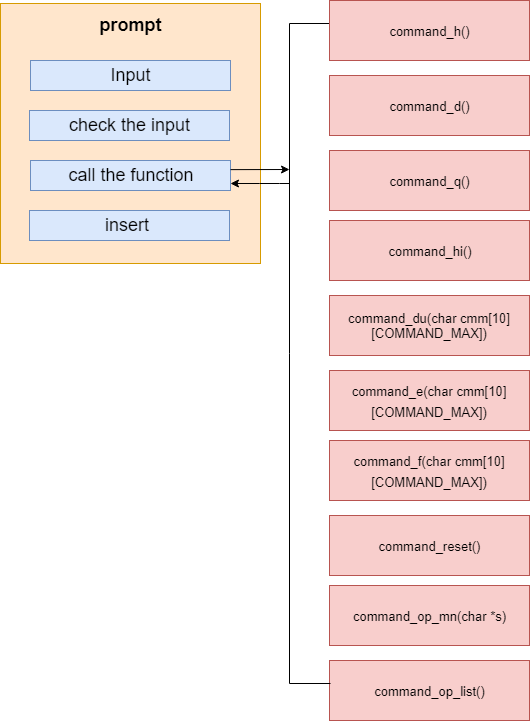
* h[elp], d[ir] ,q[uit], hi[story], du[mp] [start, end] ,e[dit] address, value , f[ill] start, end, value, reset ,opcode mnemonic, opcodelist 의 자세한 기능은 모듈 설명 차례에서 ‘ command\_ 명령어’ 부분을 통해 설명하도록 하겠다.

# 프로그램 설명

## 프로그램 흐름도

그림 1> 전체 프로그램 흐름도





* prompt와 command 함수들

프로그램을 구성하는 모듈의 구조를 도식하고, 각 모듈간의 상호작용에 사용되는 데이터에 대해서 기술한다. 각각의 상호작용의 흐름은 번호를 부여하여 도식한다.

# 모듈 정의

## 모듈 이름 : int prompt()

### 기능

사용자로부터 명령어를 입력받아 그에 맞는 함수를 호출하여 기능을 수행하고, 입력 형식에 맞지 않은 경우에는 적절한 예외 처리를 해주는 함수이다. main 함수 안에서 사용자로부터 q[uit]을 입력받을 때까지 반복적으로 호출되는 함수로 quit을 입력받은 경우에는 return q (3)을, 나머지 모든 경우에는 return 0을 해준다.

prompt 함수는 크게 1) Input 입력 받아 필요한 부분만 저장, 2) 입력 형식에 맞는지 확인, 3) 입력받은 명령어와 매개변수에 맞는 함수 호출 세부분으로 나눌 수 있다.

1. Input 입력받아 필요한 부분만 저장

-1. Input 한 줄 받아 오기: while문으로 사용자로부터 ‘\n’을 입력 받을 때까지 문자를 입력받아 command(char command[COMMAND\_MAX])에 저장한다. 이때 input 최대 글자수인 COMMAND\_MAX개보다 많은 문자를 입력할 경우에는 print\_error 함수를 통해 INPUT\_SIZE\_OVER 에러 메시지를 출력하고, 함수를 종료한다(return 0).

-2. 받아온 Input을 필요한 부분들로 쪼개서 다시 저장: 먼저 Input 중 명령어만 얻어오기 위해 command를 strtok 함수를 사용하여 ‘ ‘ (space, tap등의 모든 공백) 단위로 쪼깨고 이를command\_words[0](char command\_words[10][COMMAND\_MAX])에 저장한다. 이후 남은 문자열을 ‘,’ 단위로 쪼개서 command\_words[1]부터 차근 차근 한 문자열씩 저장한다(input 문자열이 끝날 때까지 쪼개기 반복). 이때 사용자가 앞뒤로 공백을 포함하여 명령어나 매개변수를 입력했을 경우가 있기 때문에, ‘,’ 단위로 쪼갠 문자열을 command\_words[i]로 복사하기 전에 미리 remove\_space함수로 앞뒤 space를 제거해준다.

1. 입력 형식에 맞는지 확인: (명령어+ 매개변수)에 필요한 최대 문자열 개수(command\_words의 max index+1)는 ‘f[ill]’ 명령어일 때 4개(fill, start, end, value)이므로 이를 넘어가는 부분은(프로그램에서는 5번째 문자열까지 쪼깨서 저장) 더이상 쪼개지 않고 에러메시지를 출력한 뒤 함수를 끝낸다. 더불어 매개변수 개수가 맞지 않는 경우에도 에러메시지를 출력한 뒤 함수를 끝낸다. ( edit의 경우 e[dit] ard, val -> 3개의 문자열 필요, fill은 f[ill] start, end, val -> 4개의 문자열 필요, opcode mnemonic의 경우 2개의 문자열 필요 )
2. 입력받은 명령어와 매개변수에 맞는 함수 호출 : 명령어에 해당하는 함수를 호출하여 해당 기능을 수행시킨다. 호출한 함수가 끝난 뒤에는 해당 함수가 올바르게 수행되었는지를 확인하여 올바르게 수행된 경우에만 history에 저장될 수 있도록 해준다. 또한 지원하지 않는 명령어를 입력받은 경우에는 에러메시지를 출력하고 함수를 끝낸다.

### 사용 변수

* int flag : Input max size를 초과한 경우를 표시하기 위한 변수
* char command\_words[10][COMMAND\_MAX] : 명령어와 매개변수를 저장하는 문자열 변수 (command\_words[0]: 명령어, command\_words[1] 부터 매개변수)
* char command[COMMAND\_MAX] : 사용자로부터 입력받은 실제 INPUT (history 저장용)
* char cmm[COMMAND\_MAX], \*tmp : Input을 다듬기 위해(공백 제거, 구분자 단위로 쪼개기 등) 사용하는 변수
* char ch : 한글자씩 input 받을 때 사용하는 변수

## 모듈 이름 : void print\_error (int error)

### 기능

error type에 맞는 에러 메시지를 출력해 주는 함수이다. error의 종류로는

1. WRONG\_ADR : 범위에 맞지 않는 잘못된 주소값
2. WRONG\_VAL : 올바르지 않은 입력값
3. WRONG\_PARAMETER : 형식에 맞지 않는 매개변수 값
4. WRONG\_CMM : 제공하지 않는 명령어 입력 시
5. INPUT\_SIZE\_OVER : 최대 INPUT 사이즈를 초과하여 입력한 경우

### 사용 변수

없음

## 모듈 이름 : void insert(char cmm[COMMAND\_MAX])

### 기능

history 명령 수행을 위해 그 동안 받았던 유효한 명령어들을 Linked List인 ‘hlist’에 삽입하는 함수이다. (global로 선언된 ll\* hlist가 리스트 head pointer이다.) 새로운 노드를 생성하며 매개변수로 입력받은 문자열 cmm을 노드의 command 성분에 복사해오고, 해당 노드의 next 성분을 NULL로 설정해준뒤, hlist의 맨 뒤에 연결해준다. (hlist는 queue 형식이다. )

hlist에 처음으로 노드를 추가하는 경우에는 hlist부터 malloc으로 새로 생성을 해주고, front와 rear를 모두 새로 생성한 노드를 가리키게 해준 뒤 cnt를 하나 증가시킨다.

그렇지 않은 경에는 hlist의 rear의 next가 새로 생성한 노드를 가리키게 하고, hlist의 rear를 새로 생성한 노드로 바꿔준뒤 cnt를 하나 증가시킨다.

### 사용 변수

* node\* newNode : 새로 생성하는 노드

## 모듈 이름 : char\* remove\_space(char\* s)

### 기능

매개변수로 넘겨받은 문자열 s의 앞 뒤로 포함된 공백을 제거하여, 그렇게 제거한 문자열을 return 하는 함수이다. s가 NULL 문자인 경우에는 그냥 s를 return 해준다.

s[0]부터 s[strlen(s)-1]까지 앞에서 부터 검사하여, 처음으로 문자(공백,탭,엔터 아닌경우)가 나오는 index를 찾은 뒤, s+= 해당 index 를 해주어 앞쪽 공백을 제거한다.

s[strlen(s)-1]부터 s[0]까지 뒤에서 부터 검사하여 처음으로 문자(공백,탭,엔터 아닌 경우)가 나올 때까지 s[i]=’\0’로 채원준다. 이렇게 하여 뒤쪽 공백도 제거한 뒤 return s를 해준다.

### 사용 변수

int i: loop counter

## 모듈 이름 : int one\_digit\_to\_int(char c)

### 기능

매개변수로 받아온 문자 c를 16진수(int형)로 변환해주는 함수이다. c 가 ‘A’ ~ ‘F’인 경우에는 result = c-‘A’+10 ( 16진수에서 A=10, B=11 ... 이므로 ‘A’를 빼고+10)을 해주고, c가 ‘a’~’f’인 경우에는 result=c-‘a’+10을 해준다. 마지막으로 c가 ‘0’~’9’ 인 경우에는 result=c-‘0’을 해준다.

이 세가지 경우에 해당하지 않는 경우, c가 16진수 형태에 맞지 않는 것이므로 -1을 반환하고, 그렇지 않은 경우에는 result를 반환한다.

### 사용 변수

* int result : 16진수 변환 값을 저장하는 변수

## 모듈 이름 : int make\_hex(char\* s)

### 기능

매개변수로 입력 받은 문자열 s를 16진수(int) 형태로 반환해 주는 함수이다. 우선 앞쪽에 ‘ 0x ‘나 ‘0X’ 가 붙어있는 경우 s+=2를 해주어 이 부분을 제거해준다.

그리고 s[0]부터 s[strlen(s)-1]까지 loop를 돌며, 한 digit씩(s[i] 한 문자씩) 16진수로 변환하여 result에 계속 더해준다. 16진수로 변환하는 방법은 위의 one\_digit\_to\_int 함수에서 수행한 것과 동일하되, result에 s[i] 변환 값을 더해주시 전에 매번 result \*=16을 미리 해주어 자릿수에 맞는 값으로 만들어준다. 16진수로의 변환이 끝난 경우에는 return result를 해주며, loop안에서 한 글자씩 변환하는 도중 16진수 형식에 맞지 않는 경우가 나타날 경우 return -1을 하여 더 이상 변환하지 않고 함수를 끝낸다.

### 사용 변수

* int i : loop counter
* int result=0 : 변환 결과를 저장하는 변수
* char c : loop안에서 s문자열의 성분을 한글자씩 복사하여 사용하는 변수 (c=s[i])

## 모듈 이름 : int hash\_key(char\* s)

### 기능

넘겨받은 문자열 s에 맞는 hash key 값을 생성해주는 함수이다. s 각각의 문자들에 해당하는 아스키코드 값을 모두 다 더한 것에 %20 해준 것이 hash key 값이다.

### 사용 변수

* int i: loop counter
* int key=0 : 해당 문자열의 hash key 값을 계산을 위해 사용하는 변수

## 모듈 이름 : void create\_hash\_table()

### 기능

“opcode.txt” 파일을 읽어와서 연산명령어들을 hash table에 하나씩 저장해주는 함수이다. opcode.txt 파일이 없는 경우 파일이 존재하지 않음을 출력해주고 함수를 끝낸다.

그렇지 않은 경우, fscanf로 파일에서 한 줄씩 읽어와서 ( op\_num: opcode 값 , op\_name : operation 이름, op\_type: operation type 순으로 읽어온다) 이러한 정보를 가지는 노드를 하나 생성하고, op\_name를 넘겨 계산한hash key값에 해당하는 linked list 위치에 이 노드를 연결해준다.

global로 선언된 op\* oplist[20]를 바탕으로 hash table을 생성한다. oplist[key]에 노드를 하나 연결할 때, oplist[key]가 NULL인 경우에는 oplist[key]가 새로 생성한 노드를 가리키게 하고, 그렇지 않은 경우에는 새로 생성한 노드의 next가 oplist[key]를 가리키게 한 뒤, oplist[key]가 다시 새로 생성한 노드를 가리키게 하는 식으로 연결해준다.

### 사용 변수

* int key : 각 op\_name 에 해당하는 hash key 값을 저장하는 변수
* int op\_num : opcode 값을 얻어오기 위해 사용하는 변수
* char op\_name[20], op\_type[10] : operation name과type을 얻어오기 위해 사용하는 변수
* FILE\* fp : opcode.txt를 읽어오기 위해 사용하는 변수
* op\* opcode = NULL : hash table에 연결할 노드

## 모듈 이름 : command\_h()

### 기능

help 명령어 입력시 수행되는 함수로, 명령어들을 출력해준다.

### 사용 변수

없음

## 모듈 이름: command\_d()

### 기능

dir 명령어 입력시 수행되는 함수로, 현재 디렉토리에 있는 파일들을 출력해 준다. reeddir 함수를 통해 디렉토리 element가 존재할 때까지 받아온뒤, 해당 파일의 종류에 따라 파일 이름을 출력해준다. ../. 의 유무는 출력내용에 영향을 주지 않으며, 실행파일은 이름 옆에 ‘\*’를, 티렉토리는 ‘/’를 표시해준다.

### 사용 변수

* struct diret\* file : 현재 디렉토리에 있는 파일들의 정보를 얻기 위한 변수 ( struct diret 는 dirent.h 헤더파일 안에 정의되어있는 구조체로, < d\_name(파일 이름) , d\_fileno(파일의 serial number), f\_namlen(파일이름 길이) , d\_type (파일 타입) > 를 field 로 가진다. )
* struct stat type : 파일의 타입(모드)을 받아오기 위한 변수
* DIR \*dir : 각각의 directory stream 을 저장하는 변수

## 모듈이름: command\_q()

### 기능

quit 명령어 입력시 수행되는 함수로, history와 opcode mnemonics 저장을 위해 할당했뒀던 공간을 free해준다.

### 사용변수

* int i : loop counter로 hash table head pointer 배열의 index용 변수
* node \*tmp, tmp2 : hlist(history 저장을 위한 Linked lsit) 를 모두free하기 위해 선언한 node\* 변수로, hlist에 연결된 노드들을 하나씩 접근한다.
* op \*t,\*t2 : oplist(opcode mnemonics 저장을 위한 Linked List) 를 모두 free 하기 위해 선언한 op\*변수로, oplist에 연결된 노드들을 하나씩 접근한다.

## 모듈이름: command\_hi()

### 기능

history 명령어 입력시 수행되는 함수로, 현재까지 입력 받았던 명령어들 중 올바르게 수행된 명령어들의 내용을 그대로 출력해준다. hlist가 NULL인 경우(list가 빈 경우) 아무것도 출력하지 않고 다시 프롬프트로 돌아가며, 그렇지 않은 경우 아니면node\* tmp로 리스트 노드에 하나씩 접근하여 각 node의 command를 출력해준다.

### 사용변수

* int i: loop counter로 hlist에 연결된 노드의 command를 출력하기 위해 사용된다.
* node\* tmp : hlist 에 연결된 노드들에 접근하기 위해 사용된다.

## 모듈이름: command\_du(char cmm[10][COMMAND\_MAX],int cnt)

### 기능

dump 명령어 수행을 위한 함수로, parameter의 종류에 따라 3가지 방식으로 메모리의 내용을 출력해준다. paramet에 따라 출력 시작 주소인 start와 마지막 주소인 end를 적절하게 대입하여 한가지 함수로 3가지의 출력 방식을 모두 수행 가능하도록 구현하였다. 올바른 형식의 parameter가 입력되지 않았을 경우에는 그에 맞는 경고 메시지를 띄어주고 다시 프롬프트로 돌아간다. 이런식으로 dump가 올바르게 수행되지 않은 경우에는 return 0을, 올바르게 수행된 경우에는 return 1을 한다.

cmm에는 총 cnt개의 문자열이 저장되어 있으며, cmm[0]은 d[ump], cnt>=2인경우cmm[1]에는start address, cnt=3인 경우cmm[2]에는 end address가 저장되어있다. 문자열 형태로 저장된 start address와 end address는 make\_hex(해당 문자열이 16진수 형식에 맞는지 확인 후, 맞으면 그에 맞는 16진수(int)를 반환하는 함수)함수를 통해 int 형으로 변환하여 사용한다.

왼쪽에는 5-digit의 16진수로 메모리 주소를, 가운데는 메모리에 저장된 값(1byte-아스키코드)들을, 오른쪽에는 해당 번지에 저장된 아스키 코드 값이 의미하는 문자(아스키코드 범위를 벗어난 값들에 대해서는 ‘.’을 출력)를 출력해준다

memory가 char형 2차원 배열로 구현되어 있기 때문에, 1byte를 저장하기 위해서는 2개의 배열 공간이 필요하다. 따라서 한 줄에 1byte 값을16개씩 출력하기 위해서 총 32개의 memory 배열 원소들을 출력한다. j (loop counter= column index)를 0부터 31까지 한 번에 2씩 증가시키며 memory[row][j], memory[row][j+1]을 출력해준다.

.

1. dump : 전역변수로 선언된 cur\_adr 를 출력 시작 주소로하여 160개의 (10줄) 메모리 내용을 출력해준다. 이후 cur\_adr은 마지막으로 출력한 메모리 번지+1이 된다.
2. dump start : start로 받은 값을 시작 주소로 하여 160개의 메모리 내용을 출력해준다. cur\_adr은 마지막으로 출력하 메모리 번지 +1 이 되며, start가 oxfffff를 넘어가는 경우에는 경고 메시지를 출력하고 프롬프트로 다시 돌아간다.
3. dump start,end : start부터 end 주소까지의 메모리 내용을 출력해준다. start>end인 경우, end>0xfffff 인 경우는 경고 메시지 출력 후 프롬프트로 돌아간다.

### 사용변수

* int i,j : loop counter
* int tmp : 메모리 값을 임시로 저장하는 변수 ( 20~7E 사이의 값인지 체크하기 위해 사용)
* int s, end : s= start address, end=end address로 입력을 받은 경우엔 그 입력값으로 저장되고, 그렇지 않은 경우(start,end를 입력 받지 않았거나, end를 입력받지 않은 경우)는 적절하게 계산한 값이 저장된다.
* int srow, scol ,erow,ecol : 형식에 맞게( 가로 한줄에 16개 씩 start 부터 end까지 ) 출력하기 위해 사용되는 변수로, memory 2차원 배열의 출력 시작 row,col index와 출력 마지막 row,col index의 값이 저장된다.
* srow=s/16
* scol = (s%16)\*2 -> char 배열로 memory를 구현하였기 때문에 1byte에 2개의 배열 공간 필요
* erow=end/16
* ecol =(end%16)\*2

## 모듈 이름 : command\_e(char cmm[10][COMMAND\_MAX])

### 기능

edit 명령어 입력시 수행되는 함수로, 메모리 주소와 수정 값을 입력받아 해당 주소에 수정값으로 채워주는 역할을 한다. 입력 받은 주소가 0~0xfffff 사이의 값이 아닌 경우와 수정값이 1byte를 넘어간 경우(0xff 보다 큰 경우) 그에 맞는 에러 메시지를 출력하고 프롬프트로 돌아간다.

cmm[0]에는 e[dit], cmm[1]에는 주소, cmm[2]에는 값이 저장되어 있으며, 모두 char\*형이다. 주소와 값은 make\_hex (해당 문자열이 16진수 형식에 맞는지 확인 후, 맞으면 그에 맞는 16진수(int)를 반환하는 함수) 함수를 통해 int로 변환하여 확인한다.

memory[adr/16][(adr%16)\*2]=cmm[2][0];

memory[adr/16][(adr%16)\*2+1]=cmm[2][1];

이렇게 16진수 1 digit(4 –bit)씩 메모리 배열을 수정하여 edit 기능을 수행하며, 함수가 올바르게(주소, 값 범위에 오류가 없음) 수행된 경우엔 return 1, 그렇지 못한 경우에는 return 0을 한다.

### 사용 변수

* int adr : 입력받은 주소값을 저장하는 변수이다. cmm[1]을 int로 변환한 값이 저장된다.
* int val : 입력받은 수정값을 저장하는 변수이다. cmm[2]을 int로 변환한 값이 저장된다.

## 모듈 이름 : command\_f()

### 기능

fill 명령어 입력시 수행되는 함수로, 입력 받은 시작 주소부터 끝 주소까지의 메모리 공간에 입력받은 값을 채워넣는 기능을 수행한다. cmm[0]에는 f[ill], cmm[1]에는 시작 주소, cmm[2]에는 끝 주소, cmm[3]에는 수정값이 저장되어 있으며, 이러한 문자열 타입의 값들을 make\_hex(해당 문자열이 16진수 형식에 맞는지 확인 후, 맞으면 그에 맞는 16진수(int)를 반환하는 함수)를 통해 int로 변환하여 사용한다.

각각의 값들이 올바른 16진수 형태가 아니거나, end address의 범위가 0xfffff를 벗어난 경우, start address> end address인 경우, value의 값이 1byte를 넘어간 경우(0xff보다 큰 값)에는 경고 메시지를 출력하고 프롬프트로 돌아간다. 이 경우 해당 함수는 return 0을 하여 올바르게 수행되지 않았음을 알린다.

시작 주소와 끝 주소에 맞게 start row, start column, end row, end column을 게산하여 memory배열 [start row][start column] 부터 [end row][end column] 사이의 공간에 수정값을 입력한다. 함수가 올바르게 수핸된 경우 return 1을 한다

memory가 char형 2차원 배열로 구현되어 있기 때문에, 1byte를 저장하기 위해서는 2개의 배열 공간이 필요하다. edit함수와 마찬가지로

memory[i][j] = cmm[3][0];

memory[i][j+1]=cmm[3][1];

이런식으로 16진수 1 digit씩 메모리를 수정해주었다.

### 사용 변수

* int i,j : loop counter (i = row, j= col)
* int s, end, val : 각각 시작 주소, 끝 주소, 수정값을 저장하는 변수로, cmm배열에 문자열로 저장된 각각의 값들을 make\_hex 함수를 통해 int로 변환한 값들이 저장된다.
* int srow, scol, erow, ecol : 시작 주소와 끝 주소에 대응하는memory 2차원 배열의 시작 row,col index와 마지막 row,col index의 값이 저장된다.
* srow=s/16
* scol = (s%16)\*2 -> char 배열로 memory를 구현하였기 때문에 1byte에 2개의 배열 공간 필요
* erow=end/16
* ecol =(end%16)\*2

## 모듈 이름 : command\_reset()

### 기능

reset 명령어 입력시 수행되는 함수로, 메모리를 모두 0으로 초기화 해준다. i=0~MEMORY\_ROW(0xfffff) -1, j= 0~31까지 memory[i][j]=’0’을 하여reset기능을 수행한다.

### 사용 변수

int i,j : loop counter ( i= row, j= col)

## 모듈 이름 : int command\_op\_mn(char\* s)

### 기능

opcode “mnemonic” 명령어 입력시 수행되는 함수로, 입력받은 mnemonic이 hash\_table에 있는지 확인하여 있는 경우에는 opcode 값을 출력하고, 그렇지 않은 경우에는 해당 mnemonic이 존재하지 않음을 출력해주는 기능을 수행한다. (hash\_table에 있는 경우에만 history에 해당 명령어 저장 됨)

hash\_key(char\* s) 함수를 통해 문자열 s의 hash table key 값을 얻어온 뒤, 그 key의 hash table 원소들의 op\_name과 s를 하나씩 비교하여, s와 동일한 명령어가 있는지 체크하는 방식으로 해당 기능을 수행한다.

올바른 mnemonic을 입력받아 opcode를 출력한 경우에는 return 1, 그렇지 않은 경우에는 return 0을 한다.

### 사용 변수

* int key : s의 hash table key 값을 저장하는 변수
* int flag : 입력받은 명령어가 존재하는지 확인하기 위한 변수(flag=-1 이면 opcode 존재하지 않음, flag=1이면 opcode 존재)
* op\* tmp : oplist 원소들을 하나씩 접근하기 위해 사용하는 변수

## 모듈 이름 : void command\_op\_list()

### 기능

odcodelist 명령어 입력시 수행되는 함수로, opcode hash table의 내용들을 출력해준다.

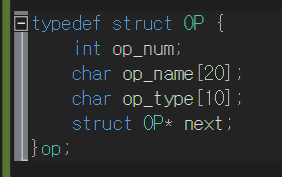
### 사용 변수

* int i: loop counter로 0~hash table max key값 만큼 증가
* int flag : 출력형식을 위해 사용하는 변수
* op\* tmp : hash table 원소들을 하나씩 접근하기 위해 사용하는 변수

# 전역 변수 정의

## op\* oplist[HASH\_MAX]

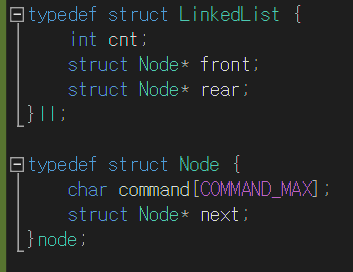
opcode mnemonic 을 저장하는 hash table을 만들기 위한 op\* 형 배열이다. HASH\_MAX는 20이며, 배열의 각 원소인 oplist[i] 는 hash key= i인 opcode mnemonics 들이 연결된 linked list의head pointer이다 .



struct OP 의 모습

## ll\* hlist

사용자로부터 입력받은 명령어(올바른 명령어)가 저장되는 linked list의 head pointer 이다. 이 linked list는 queue형식으로 되어있다. history 명령어 수행시 여기에 연결된 노드들의 command가 출력된다.



struct LinkedList의 모습

## char memory[MEMORY\_ROW][MEMORY\_COL]

memory 내용을 담고있는(메모리 역할을 하는) char 형 2차원 배열이다. MEMORY\_ROW = 0xfffff 이고, MEMORY\_COL = 50 (실제로 사용하는 column size는 32) 이다.

## int cur\_adr

dump (start, end 입력 없는 경우) 명령어 수행시 메모리 출력 시작 주소를 저장하는 변수이다. 초기값은 0이다.

# 코드

**<20161601.h>**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <dirent.h>

#include <sys/stat.h>

//명령어글자 최대 길이

#define COMMAND\_MAX 256

#define MEMORY\_ROW 0xfffff

#define MEMORY\_COL 35

#define HASH\_MAX 20

//예외처리

#define WRONG\_ADR 1

#define WRONG\_VAL 2

#define WRONG\_PARAMETER 3

#define WRONG\_CMM 4

#define INPUT\_SIZE\_OVER 5

//명령어 num

#define h 1

#define d 2

#define q 3

#define hi 4

#define du 5

#define e 6

#define f 7

#define reset 8

#define op\_mn 9

#define op\_list 10

//Linked List

typedef struct LinkedList {

int cnt;

struct Node\* front;

struct Node\* rear;

}ll;

typedef struct Node {

char command[COMMAND\_MAX];

struct Node\* next;

}node;

typedef struct OP {

int op\_num;

char op\_name[20];

char op\_type[10];

struct OP\* next;

}op;

void command\_h();

void command\_d();

void command\_q();

void command\_hi();

int command\_du(char cmm[10][COMMAND\_MAX],int cnt);

int command\_e(char cmm[10][COMMAND\_MAX]);

int command\_f(char cmm[10][COMMAND\_MAX]);

void command\_reset();

int command\_op\_mn(char\* s);

void command\_op\_list();

int prompt();

void print\_error(int error);

void insert(char cmm[COMMAND\_MAX]);

char\* remove\_space(char\* s);

int one\_digit\_to\_int(char c);

int make\_hex(char\* s);

int hash\_key(char\* s);

void create\_hash\_table();

**<20161601.c>**

#include "20161601.h"

op\* oplist[HASH\_MAX]; //opcode hash table을 위한 linked list

ll\* hlist; //history 저장 linked list

char memory[MEMORY\_ROW][MEMORY\_COL];

int cur\_adr=0; // dump 수행시 시작 메모리 번지

int main() {

int command = 0;// 명령어

command\_reset(); //memory 초기화

create\_hash\_table(); //hash table 생성

while (command != q) {

command = prompt(); //q 명령어 입력 전까지 prompt 수행

}

return 0;

}

//명령어 입력 받고, 해당 명령어에 해당하는 함수 호출quit 명령어를 받았을 때만 q(3) return 하고 , 나머지는 return 0

int prompt(){

int i=0,flag=0;

char command\_words[10][COMMAND\_MAX]; // (공백 제거된)명령어를 이루는 문자열 저장 배열

char command[COMMAND\_MAX]; // 입력받은 실제 명렁어

char cmm[COMMAND\_MAX];

char \*tmp =NULL,ch=-1;

printf("sicsim> ");

// 명령어 입력 받음(공백 포함)

while(1){

scanf("%c", &ch);

if(ch=='\n')

break;

command[i++]=ch;

//input 최대 글자수 초과인 경우

if(i==COMMAND\_MAX){

print\_error(INPUT\_SIZE\_OVER);

flag=1;

}

}

//input size 초과인 경우 종료

if(flag==1)

return 0;

command[i]='\0';

i=0; // 여기서부터 i = 명령어에 포함된 문자열 덩어리 개수

strcpy(cmm,command);

tmp= strtok(cmm," \r\n\t"); // ' ' 기준으로 자른 문자열

strcpy(command\_words[i++],tmp);

while(i<6){

tmp = strtok(NULL,","); // ',' 단위로 자른 문자열

if(tmp==NULL)

break;

tmp=remove\_space(tmp);

strcpy(command\_words[i++],tmp);

}

//잘못된 명령어(i 의 최대값 = 4)

if(i>4){

print\_error(WRONG\_CMM);

return 0;

}

//해당 명령어에 해당하는 함수 수행

if( strcmp(command, "q") ==0 || strcmp(command,"quit")==0){

command\_q();

return q;

}

if( strcmp(command, "h") ==0 || strcmp(command,"help")==0)

command\_h();

else if( strcmp(command,"d") ==0 || strcmp(command,"dir")==0)

command\_d();

else if( strcmp(command\_words[0],"du")==0 || strcmp(command\_words[0],"dump")==0){

if(i>3){ //너무 많은 parameter 받은 경우

print\_error(WRONG\_PARAMETER);

return 0;

}

if(!command\_du(command\_words, i)) // 예외 발생 시 종료( hisory 기록 안됨)

return 0;

}

else if(strcmp(command\_words[0],"e")==0 || strcmp(command\_words[0],"edit")==0){

//잘못된 형식으로 입력한 경우 예외처리 후 함수 종료

if(i!=3){

print\_error(WRONG\_PARAMETER);

return 0;

}

if(!command\_e(command\_words))

return 0;

}

else if(strcmp(command\_words[0],"f")==0 || strcmp(command\_words[0],"fill")==0){

if(i!=4){

print\_error(WRONG\_PARAMETER);

return 0;

}

if(!command\_f(command\_words))

return 0;

}

else if( strcmp(command, "reset") ==0)

command\_reset();

else if( strcmp(command\_words[0], "opcode")==0){

if(i!=2){

print\_error(WRONG\_PARAMETER); //너무 많은 mnemonic을 받았거나, 띄어쓰기가 포함된 경우

return 0;

}

if(!command\_op\_mn(command\_words[1])) //해당op\_mn이 없는 경우

return 0;

}

else if( strcmp(command, "opcodelist") ==0)

command\_op\_list();

//입력 받은 명령어가 history도 아니면, 정의되지 않은 명령어 -> 예외처리

else if( strcmp(command, "hi") !=0 && strcmp(command,"history")!=0){

print\_error(WRONG\_CMM);

return 0;

}

//예외 없이 잘 수행한 명령어들은 hlist에 추가

insert(command);

if( strcmp(command, "hi") ==0 || strcmp(command,"history")==0)

command\_hi();

return 0;

}

char\* remove\_space(char\* s) {

int i;

if (s == NULL)

return s;

//앞쪽 공백 제거

for (i = 0; i < strlen(s); i++) {

if (s[i] != ' ' && s[i] != '\t' && s[i] != '\n')

break;

}

s += i;

//뒤쪽 공백 제거

for (i = strlen(s); i > 0; i--) {

if (s[i] != '\0' && s[i] != ' ' && s[i] != '\t' && s[i] != '\n')

break;

s[i] = '\0';

}

return s;

}

void print\_error(int error) {

if (error == WRONG\_ADR)

printf("Wrong address\n");

else if (error == WRONG\_VAL)

printf("Wrong Value\n");

else if (error == WRONG\_PARAMETER)

printf("Wrong parameter\n");

else if (error == WRONG\_CMM)

printf("Wrong Command\n");

else if (error == INPUT\_SIZE\_OVER)

printf("Input size is over\n");

}

void insert(char cmm[COMMAND\_MAX]) {

//새로운 노드 할당

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

newNode->next = NULL;

strcpy(newNode->command, cmm);

//처음으로 노드 추가하는 경우

if (hlist == NULL) {

hlist = (ll\*)malloc(sizeof(ll));

hlist->front = hlist->rear = newNode;

hlist->cnt = 0;

}

else {

(hlist->rear)->next = newNode;

hlist->rear = newNode;

}

(hlist->cnt)++;

}

//한 digit(char) -> int

int one\_digit\_to\_int(char c){

int result=0;

if(c>='A' && c<='F')

result= c-'A'+10;

else if(c>='a' && c<='f')

result=c-'a'+10;

else if(c>='0' && c<='9')

result=c-'0';

//16진수 형식에 맞지 않으면 -16 return

else

result=-1;

return result;

}

int make\_hex(char\* s) {

int i, result = 0;

char c;

//앞에 0x(0X) 붙여준 경우그부분 짤라줌

if (s[0] == '0' && (s[1] == 'x' || s[1] == 'X'))

s += 2;

//한 digit씩 체크

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

c = s[i];

result \*= 16;

if (c >= 'A' && c <= 'F')

result += c - 'A' + 10;

else if (c >= 'a' && c <= 'f')

result += c - 'a' + 10;

else if (c >= '0' && c <= '9')

result += c - '0';

//16진수 형식에 맞지 않으면 -1 return

else

return -1;

}

return result;

}

//hash key 계산 ( 문자열 아스키 코드 값의 합)

int hash\_key(char\* s) {

int i, key = 0;

for (i = 0; i<strlen(s); i++)

key += s[i] - '0';

return key%HASH\_MAX;

}

void create\_hash\_table() {

int key, op\_num;

char op\_name[20], op\_type[10];

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

op\* opcode = NULL;

//파일이 없는 경우

if (fp == NULL) {

printf("'opcode.txt' dose not exist\n");

return;

}

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

//새로운 op node 생성

opcode = (op\*)malloc(sizeof(op));

opcode->op\_num = op\_num;

strcpy(opcode->op\_name, op\_name);

strcpy(opcode->op\_type, op\_type);

opcode->next = NULL;

key = hash\_key(op\_name);

//해당 위치에 처음으로 추가하는 경우

if (oplist[key] == NULL) {

oplist[key] = opcode;

}

else {

opcode->next = oplist[key];

oplist[key] = opcode;

}

}

}

//help 명령어 수행 함수

void command\_h() {

puts("h[elp]");

puts("d[ir]");

puts("q[uit]");

puts("hi[story]");

puts("du[mp] [start, end]");

puts("e[dit] address, value");

puts("f[ill] start, end, value");

puts("reset");

puts("opcode mnemonic");

puts("opcodelist");

}

void command\_d() {

struct dirent\* file;

struct stat type;

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

while (1) {

file = readdir(dir);

if (file == NULL)

break;

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

if (S\_ISDIR(type.st\_mode)) {

//. & .. 은 무시하고 넘어감

if (strcmp(file->d\_name, ".") == 0 || strcmp(file->d\_name, "..") == 0)

continue;

//directory 일 경우 /

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

}

//실행파일의 경우 \*

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

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

else

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

}

closedir(dir);

}

void command\_q() {

int i;

node\* tmp = NULL;

node\* tmp2 = NULL;

op\* t = NULL;

op\* t2 = NULL;

//hlist free

if (hlist != NULL) {

for (tmp = hlist->front; tmp != NULL;) {

tmp2 = tmp;

tmp = tmp->next;

free(tmp2);

}

free(hlist);

}

//oplist(hash\_table) free

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

for (t = oplist[i]; t != NULL; t = t->next) {

t2 = t;

free(t2);

}

}

}

void command\_hi() {

int i;

node\* tmp;

if (hlist == NULL) // 빈 리스트의 경우

return;

tmp = hlist->front;

for (i = 0; i<hlist->cnt; i++) {

printf("%d %s\n", i + 1, tmp->command);

tmp = tmp->next;

}

}

int command\_du(char cmm[10][COMMAND\_MAX], int cnt){

int i,j,tmp;

int s=0,end=0; // s: start, e: end

int srow=-1,scol=-1, ecol=-1, erow=-1; // 출력 row/col 범위

if( cnt>=2)

s= make\_hex(cmm[1]); //start 입력 받음

if(cnt ==3)

end= make\_hex(cmm[2]);//end 입력 받음

//입력 주소 체스 ( 입력받은 start/end가16진수 형식이 아니거나 범위 넘어간 경우 or s>e

if(s==-1||end==-1||s>0xfffff || end>0xfffff||( cnt==3 && s>end))//start > end 인 경우

{

print\_error(WRONG\_ADR);

return 0;

}

//start 안받은 경우

if(cnt<2){

s=cur\_adr;

if(s>0xfffff)

s=0;

}

//end 안 받은 경우

if(cnt<3){

end= s+159;

if(end>0xfffff)

end=0xfffff;

}

cur\_adr=end+1;

srow=s/16;

scol=(s%16)\*2;

erow=end/16;

ecol=(end%16)\*2;

for(i=srow; i<=erow; i++){

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

for(j=0; j<32; j+=2){

if((i==srow && j<scol)||(i==erow && j>ecol)) //범위아닌 메모리 값은 출력 x

printf(" ");

else

printf("%02X ", one\_digit\_to\_int(memory[i][j])\*16+ one\_digit\_to\_int(memory[i][j+1]));

}

printf("; ");

for(j=0; j<32; j+=2){

if((i==srow && j<scol)||(i==erow && j>ecol)) //범위아닌 메모리 값은 출력 x

printf(".");

else{

tmp=one\_digit\_to\_int(memory[i][j])\*16 + one\_digit\_to\_int(memory[i][j+1]); // 메모리에 저장된 값

if( tmp>=0X20 && tmp <=0X7E) // 20~7E 사이 값이면 해당 값의 아스키코드 출력

printf("%c",tmp);

else

printf(".");

}

}

printf("\n");

}

return 1;

}

int command\_e(char cmm[10][COMMAND\_MAX]){

int adr,val;

adr= make\_hex(cmm[1]);

val= make\_hex(cmm[2]);

if(adr<0 || adr >0Xfffff){

print\_error(WRONG\_ADR);

return 0;

}

if(val>0xff){

print\_error(WRONG\_VAL);

return 0;

}

memory[adr/16][(adr%16)\*2]=cmm[2][0];

memory[adr/16][(adr%16)\*2+1]=cmm[2][1];

return 1;

}

int command\_f(char cmm[10][COMMAND\_MAX]){

int i,j,s,end,val;

int srow,scol,erow,ecol;

s=make\_hex(cmm[1]);

end=make\_hex(cmm[2]);

val=make\_hex(cmm[3]);

//잘못된 주소를 입력한 경우 (16진수 타입이 아님, end가 범위를 벗어남, start> end)

if(s==-1 || end==-1 ||end>0xfffff|| s>end ){

print\_error(WRONG\_ADR);

return 0;

}

//value로 1byte 벗어나는 값을 입력한 경우

if(val>0xff){

print\_error(WRONG\_VAL);

return 0;

}

srow=s/16;

scol=(s%16)\*2;

erow=end/16;

ecol=(end%16)\*2;

for(i=srow; i<=erow; i++){

for(j=0; j<32; j+=2){

if((i==srow && j<scol )||( i==erow && j>ecol))

continue;

memory[i][j] = cmm[3][0];

memory[i][j+1]=cmm[3][1];

}

}

return 1;

}

void command\_reset(){

int i,j;

for(i=0; i<MEMORY\_ROW; i++){

for(j=0; j<32; j++){

memory[i][j]='0';

}

}

}

int command\_op\_mn(char \*s){

int key,flag=-1;

op\* tmp=NULL;

key=hash\_key(s);

for(tmp=oplist[key]; tmp!=NULL; tmp=tmp->next){

// 입력받은 명령어를 찾은 경우

if(strcmp(tmp->op\_name,s)==0){

printf("opcode %s is %X\n", s, tmp->op\_num);

flag=1;

}

}

//해당 명령어가 존재하지 않는 경우

if(flag==-1){

print\_error(WRONG\_PARAMETER);

return 0;

}

return 1;

}

void command\_op\_list(){

int i,flag=-1;

op\* tmp;

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

printf("%02d : ",i);

if(oplist[i]==NULL) //비어있는 경우

printf("empty");

for(tmp=oplist[i]; tmp!=NULL; tmp=tmp->next)

{

if(flag==-1){

printf("[%s,%X]", tmp->op\_name,tmp->op\_num);

flag=1;

}

else

printf("-> [%s,%X]", tmp->op\_name,tmp->op\_num);

}

flag=-1;

printf("\n");

}

}

**< Makefile >**

CC=gcc

CFLAGS=-g -Wall

OBJS=20161601.o

TARGET=20161601.out

$(TARGET) : $(OBJS)

$(CC) -o $@ $(OBJS)

20161601.o : 20161601.c 20161601.h

$(CC) -c 20161601.c -Wall

clean :

rm 20161601.o

rm 20161601.out