**과목명: 시스템프로그래밍**

**CSE4100-01**

**<<Project #1>>**

**서강대학교 컴퓨터공학부**

**20151523**

**김동현**

목 차

1. **프로그램 개요**
   1. 프로그램에 관하여
   2. 프로그램의 구성
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. 정의한 모듈에 대해 간략한 설명

3.1.1 main

3.1.2 help

3.1.3 dir

3.1.4 history

3.1.5 add\_node

3.1.6 dump

3.1.7 hextoAscii

3.1.8 edit

3.1.9 fill

3.1.10 reset

3.1.11 hash\_node

3.1.12 hash\_func

3.1.13 opcode

3.1.14 opcodelist

1. **전역 변수 정의**
   1. 라이브러리
   2. Define
   3. 변수
2. **코드 설명**
   1. 모듈

5.1.1 main

5.1.2 help

5.1.3 dir

5.1.4 history

5.1.5 add\_node

5.1.6 dump

5.1.7 hextoAscii

5.1.8 edit

5.1.9 fill

5.1.10 reset

5.1.11 hash\_node

5.1.12 hash\_func

5.1.13 opcode

5.1.14 opcodelist

**1.프로그램 개요**

**1.1 프로그램에 관하여**

본 프로그램은 앞으로 구현하게 될 SIC/XE 머신을 구현하기 위한 전 단계로서 어셈블러, 링크, 로더들을 실행하게 될 shell과 컴파일을 통해서 만들어진 object 코드가 적재 되고, 실행될 메모리 공간과 mnemonic을 opcode 값으로 변환하는 opcode table과 관련 명령어를 구현하는 프로그램이다.

**1.2 프로그램의 구성**

-shell (sicsim>)

-shell 관련 명령어들 (help, dir, quit, history)

-1MB의 메모리 공간

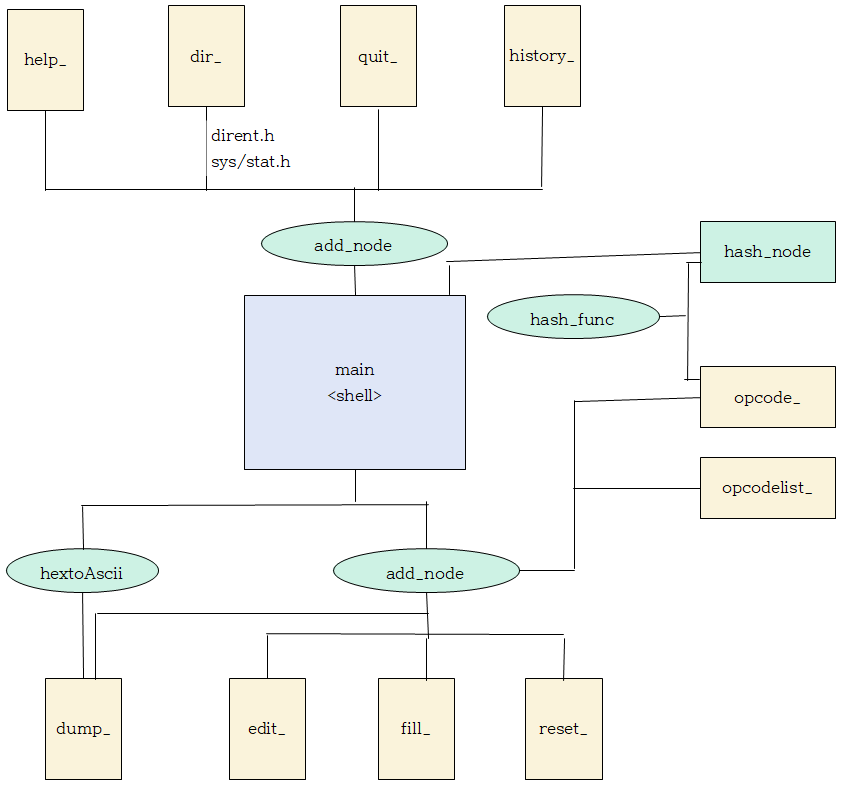
-메모리 공간 관련 명령어들 (dump, edit, fill, reset)

-opcode table

-opcode 관련 명령어들 (opcode, opcodelist)

**2.프로그램 설명**

**2.1 프로그램 흐름도**



기본적으로 main함수에서 shell mode로 진입하여, 각각의 명령어에 대해 독립적으로 실행하는 구조이다. 사용 기록(history)을 저장하기 위한 add\_note, 16진수를 ascii 문자로 변환하는 hextoAscii, opcode table을 생성하기 위한 hash\_node, hash\_func의 추가적인 모듈을 통해 각 기능들을 수행할 수 있도록 설계하였다.

3. 모듈 정의

**3.1 정의한 모듈에 관한 간략한 설명**

**3.1.1 int main()**

Main 모듈에서는 주로 변수, 메모리의 할당과 stdin으로 받은 명령어에 대해 해당하는 모듈을 호출하는 기능을 한다. 참고로 메모리는 1byte \* 16 \* 65536 크기(1MB)의 1차원 배열 형식으로 동적할당 하였다. 이는 2차원 배열 형식보다 메모리접근에 더 용이하다고 판단하였다. 그리고 명령어의 인자의 유무를 조건문으로 분기하여 모듈을 호출하도록 하였다. While(1)을 통해 quit 명령이 들어오지 않는 이상 반복하여 명령을 수행할 수 있도록 하였다.

앉아있는이(가) 표시된 사진

자동 생성된 설명

<변수와 메모리를 할당한 모습>

노트북, 화면, 전화, 검은색이(가) 표시된 사진

자동 생성된 설명

<argument가 필요 없는 경우>

모니터, 앉아있는, 텔레비전, 화면이(가) 표시된 사진

자동 생성된 설명

<argument가 필요한 명령에 해당하는 모듈 호출 시>

그리고 추후에 history나 opcode table 구현에 필요한 linked-list 구조체의 할당, 초기 설정과 opcode.txt를 읽기위한 file pointer 설정 또한 main 모듈안에 구현하였다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

**3.1.2 void help\_()**

단순히 인자나 반환 값 없이 호출되면 모듈 내부에서 사용자가 사용가능한 명령어들을 차례로 출력하여 준다.

텍스트, 검은색, 앉아있는, 녹색이(가) 표시된 사진

자동 생성된 설명

<실행 모습>

**3.1.3 void dir\_()**

해당 프로그램의 디렉토리 내부에 존재하는 파일, 디렉토리를 나타낸다.

그리기이(가) 표시된 사진

자동 생성된 설명

<실행 모습>

**3.1.4 void history\_(node \*curr)**

해당 shell에서 사용자가 입력한 유효한 명령들의 기록을 출력해주는 기능이다. 단순히 history 모듈을 통해서는 add\_node에서 생성된 linked-list를 인자로 받아서 NULL이 나올때까지 출력하게만 설계하였다.

검은색, 표지판이(가) 표시된 사진

자동 생성된 설명

**3.1.5 node\* add\_node(node\* target, char\* cmd)**

History 명령의 구현에 필요한 linked-list를 구현하는 모듈이다. 인자로 main에서 선언된 node 포인터와 사용자가 입력한 명령어를 인자로 받아와서, 단순히 새로운 node를 할당하여 node->data에 명령어를 저장한 뒤, tail node에 새로운 node를 붙여주는 방식으로 설계하였다.

앉아있는, 모니터, 화면, 검은색이(가) 표시된 사진

자동 생성된 설명

<history\_와 add\_node의 설계>

**3.1.6 void dump\_(int \*addr, unsigned char \*mem, char \*disp, char \*arg1, char \*arg2, int \*cp, int arg1\_i, int arg2\_i)**

메모리 내부의 값을 출력해주는 기능이다. 사용자가 입력한 인자의 개수에 따라 dump, dump start, dump start end의 3가지 기능을 수행할 수 있게 하였다.

컴퓨터, 키보드이(가) 표시된 사진

자동 생성된 설명

키보드, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

키보드, 전자기기, 실내, 앉아있는이(가) 표시된 사진

자동 생성된 설명

하얀색, 검은색, 키보드, 사진이(가) 표시된 사진

자동 생성된 설명

<실행 모습>

**3.1.7 void hextoAscii(char \*mem, char \*disp)**

메모리 배열과 ascii display 배열을 인자로 받아와서 20~7E의 범위 내의 값을 가진 것만 그 값 그대로 disp 배열에 저장하고 범위에 해당하지 않는 값은 .(46)으로 저장하게 하였다,

**3.1.8 void edit\_(unsigned char \*mem, int addr, int val)**

메모리 배열을 unsigned로 호출하여 해당 addr 번지에 int형 value 값을 저장할 수 있게 하였다.

컴퓨터, 키보드이(가) 표시된 사진

자동 생성된 설명

**3.1.9 void fill\_(unsigned char \*mem, int start, int end, int val)**

Edit과 비슷하게 메모리 배열을 unsigned로 호출하여 반복문을 통해 해당 범위(start ~ end)에 int형 value값을 저장할 수 있게 하였다.

컴퓨터, 키보드이(가) 표시된 사진

자동 생성된 설명

**3.1.10 void reset\_(char \*mem)**

단순하게 메모리 배열만 인자로 받아 반복문을 통해 모든 번지에 0값을 저장하였다.

**전자기기, 키보드, 앉아있는, 하얀색이(가) 표시된 사진

자동 생성된 설명**

**3.1.11 void hash\_node(hash \*target[20], FILE \*oplist)**

해시 배열과 파일 포인터를 인자로 받아서 해시 테이블을 생성하는 모듈이다. 파일에서 문자열을 입력 받는 과정도 포함되고, 계속하여 추가적인 node를 생성하는 과정을 포함하여 hash\_func의 연산을 통한 key값을 토대로 테이블이 디자인되어진다.

**3.1.12 int hash\_func(unsigned char \*arg1)**

해시 함수를 설정해 놓은 모듈이다. 문자열 인자를 이용하여 연산한 값을 key값으로 반환하는 형태로 설계하였다.

**3.1.13 int opcode\_(char \*arg1, hash\* table[20])**

해시 테이블에서 해당 mnemonic을 가진 opcode를 출력하는 모듈이다. 만약 찾지 못한다면 에러 코드를 출력한다.

병, 사진, 검은색, 전화이(가) 표시된 사진

자동 생성된 설명

**3.1.14 void opcodelist\_(hash \*table[20])**

Opcode table의 내용을 출력하여 주는 모듈이다.

노트북, 컴퓨터, 앉아있는, 검은색이(가) 표시된 사진

자동 생성된 설명

**4. 전역 변수 정의**

**4.1 라이브러리**

- stdio.h

- stdlib.h

- string.h

- dirent.h

- sys/stat.h

**4.2 define**

MEM\_height = 65536

MEM\_width = 16

메모리의 포맷을 쉽게 변화시킬 수 있도록 define을 통해 설정하였다.

**4.3 변수**

**텍스트, 테이블, 전화, 휴대폰이(가) 표시된 사진

자동 생성된 설명**

Linked-list의 생성에 필요한 두가지 구조체를 새롭게 선언하여 전역 변수로 사용하였다.

**5. 코드 설명**

**5.1.1 int main()**

**모니터, 스크린샷, 앉아있는, 테이블이(가) 표시된 사진

자동 생성된 설명**

먼저 필요한 변수들(stdin으로 받을 문자열, strtok로 구분할 문자열 변수와 argument 문자열 변수들)과 메모리, 메모리 주소, ascii 저장 배열을 할당하였다. 그리고 메모리 주소는 16씩 더해가며 초기화하였고, 이를 출력시에는 %.5X를 통하여 16진수로 나타나게했다. 메모리의 값들은 0으로 초기화 하였다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

그리고 history와 hash table에 사용할 링크드 리스트의 초기 설정을 했다. Head node를 할당하고 next를 NULL로 가게끔 해주었다. Argv#\_dec는 후에 사용할 변수로 char\* 형 문자열을 int형 16진수로 바꿔 저장할 변수들이다. Checkpoint는 dump시 마지막 address+1을 저장할 포인터이다. 그리고 opcode.txt를 읽기전용으로 읽어서 \*fp 파일 포인터에 지정해 두었고, 만일 opcode.txt를 읽어오지 못할 시에 에러문을 출력하게 했다. Hash\_node를 통해 바로 opcode table을 생성하도록 하였다.

앉아있는, 전화, 쥐고있는이(가) 표시된 사진

자동 생성된 설명

이러한 형태로 기능 선택을 하도록 하였다. While(1)을 통해 반복하여 쉘의 기능을 사용할 수 있게 하였고 main에 내장된 quit의 기능으로만 탈출할 수 있다. 그리고 각 명령어의 축약어도 인식되게 끔 조건문을 설정하였다. 각 기능의 조건문에 해당하면 head\_tmp node에 새로이 history node를 연결하게끔 해서 잘못된 명령이 들어왔을 경우에는 저장되지 않게 설계하였다. Dump는 메모리 값을 출력함과 동시에 ascii값 또한 출력해야하므로 hextoAscii까지 같이 실행되게 하였다.

스크린샷, 앉아있는, 전화, 제어이(가) 표시된 사진

자동 생성된 설명

이 부분은 argument가 필요한 기능들을 호출하는 파트이다. 먼저 입력받은 명령어를 strtok을 이용해 ,를 단위로 분절하여 argv# 변수들에게 저장해주었다. 그리고 argv#\_dec에는 16진수로 변환한 int형으로 저장하게 하였다. 이를 통해 각 argument들의 범위를 비교하여 범위 밖의 인자가 들어올 시 조건에 해당되지 않고 에러문을 출력하게 하였다. Argv2 == NULL 조건은 인자가 1개이므로 dump start의 조건이고, argv3 == NULL은 인자가 2개이므로 dump start end의 조건이다.

스크린샷, 앉아있는, 전화, 휴대폰이(가) 표시된 사진

자동 생성된 설명

마찬가지로 edit과 fill 또한 argv#\_dec의 범위 비교를 통해 유효한 값들만 받아서 실행되게끔 하였고fill은 기본적으로 3개의 인자만 받기 때문에 argv4까지가서 추가적인 인자가 있는지 검사하였다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

그리고 마지막으로 opcode\_는 h\_flag를 반환하므로 main함수에서 history list에 올릴지 말지를 검사하지 않고 opcode\_함수 내에서 이 명령이 유효한 명령인지에 대한 flag를 반환하도록 하였다. Shell이 종료될 시 할당되었던 메모리들을 모두 free로 해제하여주었다.

**5.1.2 void help\_()**

**테이블, 앉아있는, 검은색, 화면이(가) 표시된 사진

자동 생성된 설명**

단순히 printf를 하여 명령어 list를 출력하게 하였다.

**5.1.3 void dir\_()**

**화면, 노트북, 앉아있는, 모니터이(가) 표시된 사진

자동 생성된 설명**

Dirent.h 라이브러리를 참조하여 디렉토리 포인터를 통해 디렉터리에 접근하는 방식이다. 실행하기 위하여 파일의 이름과 파일의 type을 찾아야 했는데, 파일의 이름은 dirent 구조체의 d\_name을 통해 접근하여 출력하였고, 파일의 type은 sys/stat.h 라이브러리에 포함된 stat 구조체의 st\_mode를 S\_ISDIR이라는 디렉토리 구분 함수를 통해 그 파일이 디렉토리인지, 아니면 실행 파일인지 구분하여 디렉토리 파일의 이름의 뒤에는 /를, 실행파일의 이름의 뒤에는 \*를 추가하여 구분할 수 있도록 하였다. 그리고 cnt 변수를 활용하여 파일이 4개씩 출력 될때마다 개행을 하도록 하여 심미성을 높였다.

**5.1.4 void history\_(node \*curr)**

**개체, 시계, 앉아있는, 모니터이(가) 표시된 사진

자동 생성된 설명**

curr으로 history 의 head node를 받아와서 curr이 NULL이 나올 때 까지 해당 node의 index와 data를 출력해준다.

**5.1.5 node\* add\_node(node\* target, char\* cmd)**

History 명령의 구현에 필요한 linked-list를 구현하는 모듈이다. 인자로 main에서 선언된 node 포인터와 사용자가 입력한 명령어를 인자로 받아와서, 단순히 새로운 node를 할당하여 node->data에 strcpy를 통해 cmd의 문자열을 복사한 뒤, tail node에 새로운 node를 붙여주는 방식으로 설계하였다.

그리고 history 출력에 필요한 index number 또한 node 구조체 내에 멤버로 할당하여 추가적으로 저장하도록 하였다. 성공적으로 연결한 뒤 그 새로운 node에 대한 포인터를 반환함으로써 그 다음 수행 시 자동적으로 그 node 뒤에 새로운 node를 연결할 수 있도록 설계하였다.

앉아있는, 모니터, 화면, 검은색이(가) 표시된 사진

자동 생성된 설명

<history\_와 add\_node의 설계>

**5.1.6 void dump\_(int \*addr, unsigned char \*mem, char \*disp, char \*arg1, char \*arg2, int \*cp, int arg1\_i, int arg2\_i)**

우선 dump\_라는 이 한 개의 모듈을 통해 dump, dump start, dump start end의 명령을 수행할 수 있도록 많은 조건을 통한 분기를 만들었다. 그리고 각각의 경우 안에서도 초기에 blank가 출력되는 경우와 그렇지 않은 경우를 구분하였다. ‘

1. Dump (arg1 == NULL)

모니터, 스크린샷, 화면, 텔레비전이(가) 표시된 사진

자동 생성된 설명

우선 \*cp (체크포인트)를 통해 마지막에 실행된 dump의 주소 값을 참조한다. 그래서 \*cp % MEM\_width가 0인 경우, blank가 없는 경우이고, 0이 아니라면 blank가 필요한 경우이다. Blank가 없는 경우에는 단순히 2중 반복문으로 주소 출력 후, 메모리 내의 16개 값 출력 후, 그를 ascii로 변환한 값을 출력하도록 하였다. 메모리 참조 인덱스가 MEM\_width \* i + j + \*cp인데 이는 2차원 배열을 생각할 때에 i를 행, j를 열로 잡고 1차원 배열이므로 추가적으로 이전 \*cp만큼을 더해준 값이다. 그렇게 160개의 메모리 값을 출력 후 체크포인트 또한 160개 늘려주었다. 그리고 출력 후 접근 가능한 주소를 넘으면 에러 코드가 출력되게 하였다.

스크린샷, 앉아있는, 테이블, 은색이(가) 표시된 사진

자동 생성된 설명

Blank가 출력되어야 하는 경우에는 첫 줄, 중간 줄, 마지막줄 이렇게 3파트로 구분하였다. 어드레스값이 연속적으로 나오는게 아니라 어드레스 1개 후 메모리 값 출력 후, ascii값 출력하는 방식이라서 구분하는 방식으로 설계하였다. 첫 줄에서는 \*cp % MEM\_width 만큼 공백이 나오고, 메모리 값 출력, 그 다음 ascii 또한 .이 나오다가 메모리 값을 변환한 문자를 출력하게 하였다.

중간 줄은 blank 없이 연속적인 9줄이므로 위의 방식과 같이 배열 index를 MEM\_width \* I + \*cp – (\*cp % MEM\_width)로 하였고, 마지막 줄은 첫 줄의 반대 순서로 수행하게 하였다.

1. dump start (arg2 == NULL)

모니터, 화면, 앉아있는, 검은색이(가) 표시된 사진

자동 생성된 설명

dump start는 기존의 dump와 크게 다르지 않았다. 단지 \*cp의 역할을 arg1(start 인자)가 대신하는 정도로 설계하였다. Arg1\_i라는 추가 인자를 받아서 사용하였는데 이는 main 모듈에서 인자를 받아서 strtoul 함수를 통해 int형으로 바꾼 것이다. 이를 이용하여 반복문이나 배열 인덱스 연산에 사용하였다. Dump와 마찬가지로 160개의 메모리 값을 출력하는 것이므로 코드는 거의 동일하지만 cp의 자리에 arg1\_i를 넣는 식으로 구성하였다. 하지만 마지막에 \*cp는 새로이 저장해야 그 다음 dump 실행 시 최신의 주소를 참조할 수 있으므로 \*cp = arg1\_i + 160으로 초기화 해주었다.

앉아있는, 검은색, 오렌지, 어두운이(가) 표시된 사진

자동 생성된 설명

<strtoul을 사용한 모습>

스크린샷, 앉아있는, 전화, 테이블이(가) 표시된 사진

자동 생성된 설명

blank가 필요한 경우도 마찬가지였다. Cp를 대신해 arg1을 시작 주소점으로 하여서 기존의 dump와 유사하게 설계하였다. 첫줄, 중간줄(9줄), 마지막 줄 3부분으로 구성하였고, 역시 마지막에 체크포인트 업데이트는 arg1을 기준으로 160개 더하여서 초기화 하였다.

1. dump start end

스크린샷, 앉아있는, 검은색, 화면이(가) 표시된 사진

자동 생성된 설명

위의 코드는 blank가 필요 없는 경우로써, cnt 변수를 활용하여 cnt가 arg2 – arg1보다 커질 경우 주어진 범위의 메모리 값은 모두 출력한 것으로 판단하고 뒷부분은 blank로 처리하였다. 그리고 체크포인트 초기화도 arg1 + (arg2 – arg1) + 1로 초기화 해주었다.

그리고 dump start end에서는 blank가 필요한 경우와 그렇지 않은 경우, blank가 필요한 경우는 다시 end – start <= MEM\_width - (arg1\_i % MEM\_width) 인 경우와 그렇지 않은 경우로 나누었다. 왜냐하면 blank이 필요한 경우는 출력을 해야하는 메모리 값 개수에 따라 상황이 달라졌기 때문이다.

테이블, 검은색, 앉아있는, 노트북이(가) 표시된 사진

자동 생성된 설명스크린샷, 앉아있는이(가) 표시된 사진

자동 생성된 설명

end – start <= MEM\_width - (arg1\_i % MEM\_width) 인 경우, 출력은 한 줄인데 그 양 옆 모두 blank가 생기게 된다. 그래서 그 반복문을 1중으로만 설계하여서 1줄만 출력하게끔 설계했다. 그렇지 않은 경우는 dump start의 blank가 필요한 경우와 비슷하게 되어있고, 마지막 줄 출력에만 blank가 나오게끔 예외처리를 해주었다. I가 (arg2- arg1) / MEM\_width일 때 마지막 줄이므로 그때 blank가 나오는 범위를 설정하여 따로따로 반복문을 실행해서 표현해주었다.

**5.1.7 void hextoAscii(char \*mem, char \*disp)**

**검은색, 화면, 방, 어두운이(가) 표시된 사진

자동 생성된 설명**

메모리와 디스플레이 배열을 인자로 받아와서 두 배열의 원소를 1대1 대응으로 하여 유효한 범위의 value만 변환시키고, 나머지는 .으로 변환하게끔 하였다.

**5.1.8 void edit\_(unsigned char \*mem, int addr, int val)**

**개체, 시계, 앉아있는, 공이(가) 표시된 사진

자동 생성된 설명**

메모리 배열과 주소,value를 받아와서 해당 addr의 val을 단순하게 초기화 해주는 식으로 설계하였다.

**5.1.9 void fill\_(unsigned char \*mem, int start, int end, int val)**

**시계, 어두운, 검은색, 사진이(가) 표시된 사진

자동 생성된 설명**

Edit과 거의 비슷하게 구성 되어있는데 반복문으로 한번에 정해진 범위만큼 초기화하도록 하였다.

**5.1.10 void reset\_(char \*mem)**

**어두운, 시계, 화면, 검은색이(가) 표시된 사진

자동 생성된 설명**

Fill과 거의 동일하고 reset은 범위가 전체이므로 인자도 메모리 배열만 받아서 정해진 0값을 초기화 시켜주었다.

**5.1.11 void hash\_node(hash \*target[20], FILE \*oplist)**

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

해시 테이블을 생성하는 함수이다. 우선 해시 배열과 파일 포인터를 인자로 받는다. 그리고 파일이 끝날 때 까지 (!feof(oplist)) 새로운 해시 노드를 생성하여 그 노드 안에 멤버안에 opcode(code), mnemonic(name)를 받아 저장한다. 그리고 idx에 hash\_func을 통한 key값을 주고, idx에 따라 opcode들을 target[idx]에 링크드 리스트로 달아주었다. 만약 해당 [idx]의 target에 처음 들어온 opcode라면 target[idx] == NULL 조건에 걸려 처음으로 등록되고 그 뒤로 오는 노드는 curr 노드의 탐색을 통해 next가 NULL인 노드 뒤에 붙이게 된다. Feop\_flag는 일부러 설정하였는데, !feof(oplist)라는 조건을 사용하니 opcode.txt의 마지막의 개행값이 자꾸 포함되어서 일부러 그런 것을 방지하기위해 flag로 세워 개행값을 차단하였다.

**5.1.12 int hash\_func(unsigned char \*arg1)**

**쥐고있는, 방, 화면, 앉아있는이(가) 표시된 사진

자동 생성된 설명**

해쉬 펑션의 식을 그냥 단순하게 mnemonic 문자열을 strtoul을 통해 16진수 int형으로 변환한 뒤, 3으로 나눈 몫의 20으로 나눈 나머지로 정하였다. 별다른 의미는 없고 생각보다 키값의 분포가 썩 좋지는 않다. 0에 몰린 형태로 나타났다.

**5.1.13 int opcode\_(char \*arg1, hash\* table[20])**

앉아있는이(가) 표시된 사진

자동 생성된 설명

우선 인자로 받은 mnemonic을 hash\_func으로 key값을 받아 i에 저장하고, curr node로 주어진 key값을 인덱스로하는 배열에서 링크드리스트 탐색하였다. 두 문자열을 비교하여 같은경우 code값을 출력하고 h\_flag를 올려주었는데, h\_flag는 history에 저장해야하는 경우 1, 아닌 경우 0를 가지는 변수로 설정하였다. 그렇게 탐색하다가 curr가 NULL에 다다르면 table에 해당 mnemonic이 없다는 의미이므로 찾을 수 없다는 메시지가 출력되도록 하였다.

**5.1.14 void opcodelist\_(hash \*table[20])**

화면, 앉아있는, 노트북, 검은색이(가) 표시된 사진

자동 생성된 설명

우선 index값을 먼저 출력하고 , 해당 인덱스의 링크드 리스트를 차례로 출력하게 하였다. Curr node가 NULL이 되면 해당 index의 배열의 링크드 리스트는 모두 탐색하였다는 의미이므로 개행을 한 뒤 그 다음 인덱스로 넘어가서 탐색하게 하였다.