과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 2>>

**서강대학교 컴퓨터학과**

**[20161601]**

**[송인아]**

목 차

1. 프로그램 개요 4

2. 프로그램 설명 4

2.1 프로그램 흐름도 4

3. 모듈 정의 8

\* structure 설명 8

3.1 모듈 이름 : int command\_file(char\* name) 9

3.1.1 기능 9

3.1.2 사용 변수 9

3.2 모듈 이름 : int command\_asm(char\* name) 9

3.2.1 기능 9

3.2.2 사용 변수 11

3.3 모듈 이름 : int store\_symbol(int adr,int size, char\*s, int line) 12

3.3.1 기능 12

3.3.2 사용 변수 13

3.4 모듈 이름 : int insert\_code(char\* inst, char\* operand, int line) 13

3.4.1 기능 13

3.4.2 사용 변수 15

3.5 모듈 이름 : int comp\_symbol(char\* name, char\* tmp) 15

3.5.1 기능 15

3.5.2 사용 변수 16

3.6 모듈 이름 : int insert\_symbol(char\* name, int adr) 16

3.6.1 기능 16

3.6.2 사용 변수 17

3.7 모듈 이름 : int free\_symbol() 17

3.7.1 기능 17

3.7.2 사용 변수 17

3.8 모듈 이름 : int read\_symbol(char\* name) 18

3.8.1 기능 18

3.8.2 사용 변수 18

3.9 모듈 이름 : int command\_sym() 18

3.9.1 기능 18

3.9.2 사용 변수 18

3.10 모듈 이름 : int create\_obj(FILE\* fp, char\* prog\_name) 19

3.10.1 기능 19

3.10.2 사용 변수 20

3.11 모듈 이름 : int create\_lst(FILE\* fp, char\* inst, int line) 21

3.11.1 기능 21

3.11.2 사용 변수 25

3.12 모듈 이름 : int read\_reg(char\* reg) 25

3.12.1 기능 25

3.12.2 사용 변수 25

3.13 모듈 이름 : char make\_char(int n) 25

3.13.1 기능 25

3.13.2 사용 변수 26

3.14 모듈 이름 : int make\_int(char\* s) 26

3.14.1 기능 26

3.14.2 사용 변수 26

4. 전역 변수 정의 27

4.1 sym\* symtable[26] 27

4.2 code code\_set[CODE\_MAX] 27

4.3 int lc[MEMORY\_ROW] 27

4.4 int a,t,f,l,sw,pc,base,x,s 27

4.5 int total\_size 27

4.6 int start\_check, end\_check 27

4.7 int code\_max 27

# 프로그램 개요

프로젝트 1에서 구현한 shell에 assemble 기능이 추가 된 프로그램으로, SIC/XE의 asm 파일을 입력 받으면, 그에 맞는 lst/obj 파일을 생성하거나 assemble 에러가 발생했음을 알려준다. 더불어 assemble 결과로 만들어진 symbol table을 출력하고, 현재 디렉토리에 있는 파일명을 입력받아 해당 파일 내용을 출력해주는 기능도 추가 구현한다. Assemble의 경우 교재 2.2까지 설명된 SIC/XE 어셈블러의 기능을 구현한다.

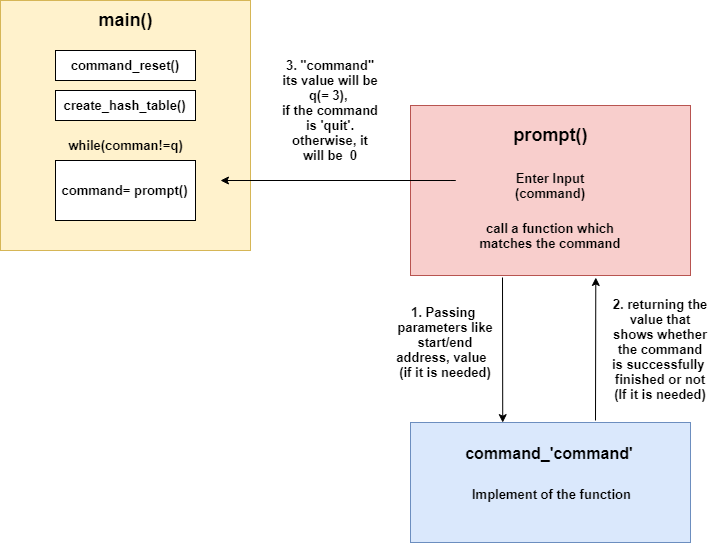
< 추가 구현 기능 >

1. help : type, assemble, symbol 명령어를 추가하여 출력한다.
2. type (filename) : 현재 디렉토리에서 filename 파일을 찾고, 존재하는 파일인 경우 내용을 출력한다.
3. assemble (filename.asm) : asm 파일을 입력하면, 그 파일을 assemble하여 lst/obj 파일을 생성해준다. 이 과정에서 asm 파일 코드에 오류가 있는 경우 lst/obj 파일을 생성하지 않고 에러가 발생한 라인 수를 출력해준다.
4. symbol : 마지막으로 assemble한 프로그램의 symbol table을 알파벳 내림차순으로 출력해주는 기능을 수행한다. 탭+Symbol+탭+주소값+\n 형식으로 출력하며, 맨 마지막 줄에는 엔터를 빼고 출력한다.

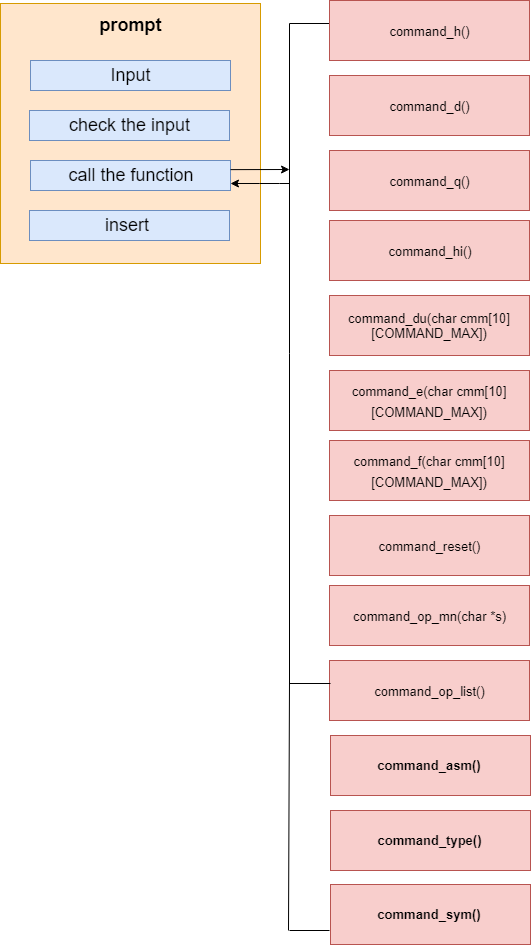
# 프로그램 설명

## 프로그램 흐름도 (편의를 위해 함수 이름만 표기하였음)

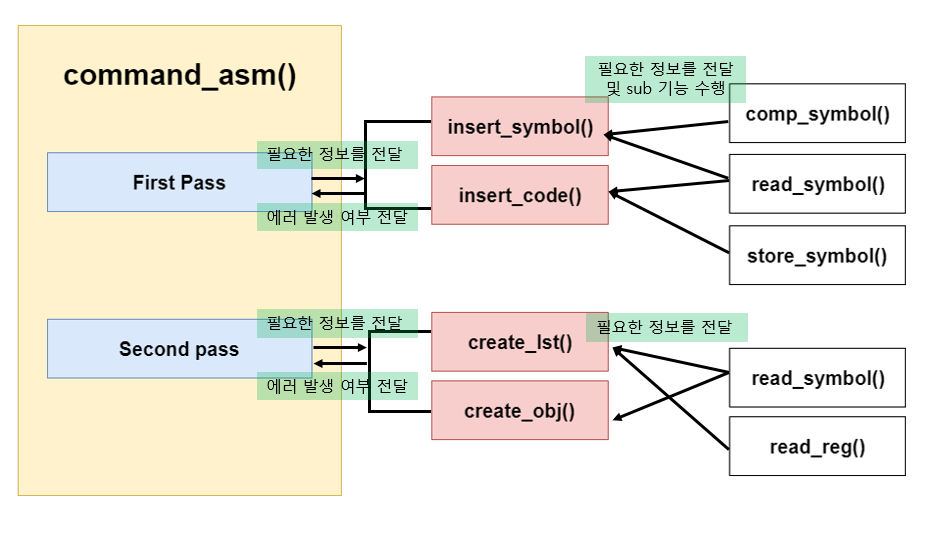
**< 전체 프로그램 흐름도 >**



**< prompt <-> command 함수들간의 흐름도 >**

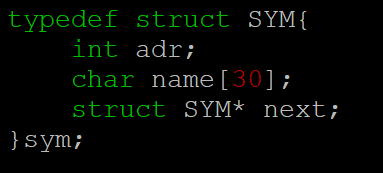


**< Assemble 과정 순서도>**



# 모듈 정의

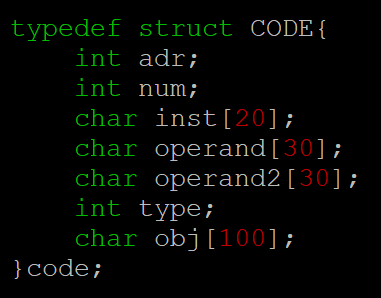
* 모듈 정의 편의를 위한 sructure 간략 설명



symbol table을 위한 structure

neme = symbol 이름

adr = symbol의 Loc



asm 코드 한줄마다의 정보를 저장하기 위한 structure

adr = 해당 code의 Loc

num = opcode

inst = 해당 operation 이름 (ex) LDA)

operand, operand2 = 매개변수들

type = instruction format 값

obj = 해당코드의 obj code

* ***프로젝트 1에서 설명한 함수는 이 보고서에 작성하지 않았음(help포함)***

## 모듈 이름 : int command\_file(char\* name)

### 기능

type 명령어 입력시 수행되는 함수로, 입력받은 이름을 가진 파일을 현재 디렉토리에서 검색하여 해당 파일이 있는 경우 내용을 출력해 주고, 존재하지 않는 경우에는 파일이 없음을 알려준다.

올바른 형식의 명령어의 경우 파일이 있는 경우, 없는 경우 모두 return 1이다

fopen 함수를 사용하여 입력 받은 파일이 존재하는지 확인하고, 있는 경우 내용을 출력한다.

### 사용 변수

* FILE\* fp : 파일에 접근하기 위한 파일 포인터
* char c[LIEN\_MAX] : 해당 파일이 존재하는 경우, fgets로 문자열을 한줄 씩 받아오는데, 이때 받아온 문자열을 저장하는 변수

## 모듈 이름: int command\_asm(char\* name)

### 기능

assemble 명령어 입력시 수행되는 함수로, 입력한 asm 파일에 대응하는 .lst / .obj 파일을 생성해준다. 이 때, asm 파일에 오류가 있는 경우, 해당 오류가 발생한 라인 수를 출력하며 함수가 종료된다(return 0). 이러한 경우에는 lst/obj 파일이 생성되지 않고 history에도 기록되지 않는다. 이 함수는 크게 0) asm 파일 체크 1) fisrt pass, 2) 파일 생성, 3) second pass 부분으로 나눌 수 있다.

1. asm 파일 체크: name에는 .asm 이 제외된 파일 이름 부분만 저장되어 있기 때문에 strcat으로 “name”.asm 형식으로 만들어주고, fopen 함수를 통해 해당 파일이 존재하는지 확인한다. 존재하지 않을 경우 에러메시지를 출력하고 함수를 종료한다. 기존에 symbol table이 채워져 있었을 수 있으므로 free\_symbol 함수로 symbol table을 비워준다.
2. first pass : 파일을 처음부터 끝까지 한 줄씩 읽으며, Location count값을 계산하고 symbol table을 생성하는 과정이다. 우선 lc[0] = 0으로 초기화 해둔다.

* lc[line] = asm 파일의 line번째 코드의 Location count 값
* fgets로 c에 최대 LINE\_MAX만큼 파일이 끝날 때까지 한 줄씩 읽어온다.
* c[0]이 ‘.’인 경우 주석, c[0]이 ‘ ‘(공백)인 경우 symbol이 없는 코드, 그 외의 경우는 symbol이 있는 코드로 구분한다.
* 주석인 경우 lc와 line수만 증가시키고 다음 코드로 넘어간다. 그 외의 경우 lst 파일에서 형식에 맞게 출력하기위해, 매번 c의 길이와 code\_max를 비교하여 해당 프로그램 내의 코드 최대 글자수를 구한다
* symbol이 없는 경우 tnum=2, 그렇지 않은 경우는 tnum=3으로 설정한다. tnum은 해당 line 코드를 몇 등분으로 나눌 것인지를 알려주는 값으로, tnum이 2이면 (operation) (operand) 두부분으로 나뉜다는 것이고, 3이면 (symbol) (operation) (operand) 세부분으로 나눈다는 것이다. strtok\_r 함수를 통해 tnum-1 번 쪼개어 필요한 부분들을 얻어낸다. (문자열 배열인 inst에 각각이 저장된다)
* tnum이 3이고, line이 0이면, 해당 symbol은 프로그램 이름을 나타내므로 prog\_name에 6글자만큼(strncpy) 받아와서 저장한다.
* tnum이 3인 경우, inst[0]에는 symbol의 이름이 저장되어 있으므로 insert\_symbol함수에 inst[0]과 lc[line]을 넘겨 해당 symbol정보를 symbol table에 저장시킨다. 그리고 insert\_code 함수에 operation(inst[1]에 저장됨)과 operand(마지막으로 짜르고 남은 문자열 = tmp2), 현재 line 수를 넘겨 해당 코드에 대한 정보를 code\_set[line]에 저장한다.
* tnum이 2인 경우, symbol이 없으므로 inst[0](operation)과 tmp2(operand), line을 insert\_code에 넘겨 코드 정보를 저장시킨다.
* 각각의 경우 모두 insert\_code나 insert\_symbol 함수의 return 값이 0인 경우 오류 메시지를 출력하고 함수를 종료시킨다. (return 0)
* 이러한 과정을 파일이 끝날 때까지 수행한다.

1. 파일 생성: name.lst와 name.obj 파일을 생성한다. 그리고 global 변수인last\_line(해당프로그램 마지막 라인 수), total\_size = 프로그램 코드 총 사이즈 합 (lc[last\_line]-lc[0])를 구한다.
2. Second Pass: name.asm 파일을 새로 열어 마찬가지로 한 줄씩 읽어온다. 이 과정에서는 line마다 create\_lst 함수를 호출하여 한 줄 한 줄씩 lst 파일을 작성한다. ( 주석이 아닌 경우만 함수를 호출하고, 주석인 경우 다음 줄로 넘어간다) 만약 create\_lst 함수의 return 값이 0인 경우 에러가 발생했다는 뜻이므로 에러메시지를 출력하고 함수를 종료한다. 그리고 이 경우에는 remove 함수로 만들었던 name.lst/name.obj 파일을 삭제한다. 이러한 과정을 파일끝까지 수행한 뒤, create\_obj 함수를 호출하여 obj 파일을 작성한다.

모든 파일의 작성이 끝난 뒤에는 asm/lst/obj 파일을 닫아준다. 이처럼 올바르게 assemble이 끝난 경우에는 assemble이 끝났음을 알려주는 안내 메시지를 출력하고 return 1 한다.

1. 다음 assemble을 위한 단계: start\_check=0, end\_check=0으로 초기화하고, 열었던 파일을 모두 닫아준다.

### 사용 변수

* FILE\* fp, \*fp\_lst, \*fp\_obj = 각각 .asm/.lst/.obj 파일을 읽고 쓰기 위해 사용하는 파일 포인터
* char asmfile[LINE\_MAX], \*lstfile, objfile[LINE\_MAX] : 각각 .asm/.lst/.obj 파일 이름을 저장하기 위한 문자열
* char c[LINE\_MAX] : 파일에서 fgets로 코드 한 줄 한 줄씩 받아올 때 쓰는 문자열
* char \*tmp, \*tmp2 : 코드 한 줄을 형식에 맞게 쪼개기 위해 사용하는 임시 문자열
* char prog\_name[7] : 프로그램 이름을 저장하기 위한 문자열로, “INAINA”로 초기화 되어있음
* char inst[5][LINE\_MAX] : 코드를 형식에 맞게 부분 부분 쪼갠 문자열들이 저장되는 문자열 배열
* int i: loop counter
* int line =-1 : 현재 보고있는 코드 line 수를 의미한다.
* int tnum = 코드를 몇 등분할지 나타내는 변수

## 모듈 이름: int store\_symbol(int adr, int size, char\* s, int line)

### 기능

BYTE, WORD로 상수를 받은 경우, 입력받은 상수값이 형식에 맞는지 체크하고, 올바른 값이라면 code\_set[line]에 해당 정보를 저장해주는 함수이다. 크게 문자형 상수와 정수형 상수로 나누어 해당 기능을 수행한다. ( C’string’, X’hex’ 이런 식으로 ‘ 앞 쪽의 문자가 상수 타입을 알려주고 있기 때문에, strtok 함수로 ‘ 앞쪽의 단어를 얻어오고 그에 따라 구분하여 저장한다. )

문자형 상수인 경우, WORD type으로 선언한 경우return 0으로 에러처리를 해준다. 이후strtok 함수로 ‘ 를 기준으로 다시 문자열을 쪼개고, 남은 문자열로 strtok 함수를 공백 기준으로 한 번 더 수행하였는데도 결과가 NULL이 아니라면 올바른 형식이 아니므로 return 0으로 에러처리를 해준다. 이후 ‘ ‘ 사이의 문자열의 길이를 strlen으로 받아오고, 그 값이 30보다 크다면 최대 길이수를 넘어간 것이므로 return 0을 해준다. 이러한 오류가 없는 올바른 형식의 문자열은 code\_set[line].obj 에 각각의 아스키코드 값으로 저장을 해주고, 해당 문자열 길이만큼(1byte\* 문자열 길이= 문자열길이) lc[line]에 더한 값을 lc[line+1]로 설정해주고 종료한다.

(저장 방법 - 1) i번째 문자를 (int)로 캐스팅하여 ASCII코드 값을 정수로 얻어온다. - 2) 그 값을 16으로 나눈 몫 => 왼쪽 half byte, 16으로 나눈 나머지 => 오른쪽 half byte 가 된다. 이 각각의 값들을 make\_char 함수를 사용하여 char 타입으로 바꾸고, 이 char 문자 2개를 각각 (code\_set[line].obj)[2\*i], (code\_set[line].obj)[2\*i+1]에 저장한다. 그리고 이러한 과정을i=0~문자열 길이-1까지 수행하고, 마지막으로 obj 배열의 마지막(2\*문자열 길이) 인덱스에 ‘\0’을 넣어주면 된다. )

16 진수의 경우, 위와 마찬가지로 우선 strtok 함수로 ‘ 를 기준으로 다시 문자열을 쪼개고, 남은 문자열로 strtok 함수를 공백 기준으로 다시 수행하였는데도 결과가 NULL이 아니라면 올바른 형식이 아니므로 return 0으로 에러처리를 해준다. 올바르게 받은 문자열(‘ ‘ 사이의 문자열)을 make\_hex 함수를 통해 16진수 정수값으로 받아오고, 형식에 맞지 않거나(make\_hex 함수의 return 값이 -1인 경우), 최대 범위를 넘어간 경우(size =1 인데 0xff보다 큰 경우, size = 3인데 0xffffff보다 큰 경우)는 return 0을 해준다. 오류가 없는 경우 code\_set[i].obj에 문자열을 문자열 길이만큼 그대로 저장해주고, 해당 상수의 사이즈 만큼 lc 값을 증가시켜준다. (만약 5를 저장하려 한다면, X’05’이렇게 0을 채워서 입력받았야 함)

상수 입력은 문자열 or 16진수만 가능하다.

### 사용 변수

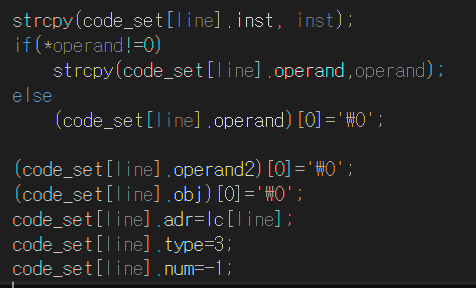
* int i: loop counter
* int len: 입력받은 상수 문자열의(정수형 상수이든, 문자형이든 모두 문자열로 넘겨받음) 글자수
* int tmp2= 문자형 상수 저장할 때, 각 digit 마다의 아스키코드 값(정수형)을 임시로 저장하기 위해 사용하는 변수
* char\* tmp = ‘ ‘사이의 문자열을 저장하기 위해 사용하는 변수

## 모듈 이름: int insert\_code(char\* inst, char\* operand, int line)

### 기능

line번째 코드 한 줄의 정보를 code\_set[line]에 저장해주는 함수이다. 해당 코드에 맞게 code\_set[line]의 inst, operand, operand2, obj, adr, type, num을 채워준다. 해당 코드의 inst가 directive인 경우(start, end, byte, word, resw, resb, base)와 일반 operation인 경우로 나누어 저장하며, 해당inst가 opcode.txt에 존재하지 않는 경우에는 type을 COMMENT로 해주고 return 1을 해준다.

1. Directive인 경우



이렇게 초기화를 해준다.

* START/END : type을 DIREC로 바꾸고, start의 경우 location count를 operand 로 바꿔준다. ( 즉, lc배열의 0번째 인덱스의 값을 make\_hex(operand)로 설정한다)
* 상수의 경우: type을 CONST로 바꿔주고, lc 값을 해당 상수 크기만큼 증가시킨다(lc[line+1] = lc[line] + 상수 사이즈) store\_symbol함수를 사용하여 obj code를 operand에 맞게 설정해준다. 잘못된 형식의 operand(ex) 사이즈 초과)인 경우 return 0을 해준다. obj code 생성 방법을 store\_symbol 함수에서 설명.
* 변수의 경우: type을 VAR로 바꿔주고, 해당 변수 사이즈\* 변수 개수(operand)만큼 다음 lc 값을 증가시킨다. (lc[line+1] = lc[line] + 변수 사이즈\* 개수)
* 모든 Directive들은 위의 연산이 끝나면 return 1을 하고 함수를 종료한다.

1. 일반 operation인 경우

* inst[0]==’+’인지 확인하여 그런 경우 flag로 체크해두고, inst++하여 ‘+’ 을 제거한다.
* hash\_table(oplist)의 inst에 해당하는 key번째 list를 돌며, inst와 동일한 operation이 있는지 strcmp로 확인한다. 동일한 이름의 operation이 없는 경우는 return 0을 한다.
* 해당 operation이 존재하는 경우:

1. flag가 set되었는데 type이 1이거나 2인 경우 return 0을 해준다.
2. operand가 있으면(\*operand!=0) strtok함수로 operand 문자열을 ‘.’ 단위로 쪼개고, ‘.’ 이전의 문자열을 para1, 이후의 문자열을 para2에 저장하고, code\_set[line]의 operand, operand2에 저장한다 (모두 앞뒤 공백 제거된 문자열).
3. strtok 함수 한번 수행 이후에도 남은 문자열이 NULL이 아니라면 2개 초과의 매개변수를 입력한 것이므로 return 0을 해준다.
4. 해당 inst의 type에 해당하는 opcode(num)와 type을 저장하고, 다음 lc[line+1] = lc[line] + code\_set[line].type으로 바꿔준다.

### 사용 변수

* int key : hash\_key(inst) 값이 저장된다.
* int flag: 해당 inst에 ‘+’ 가 있는지 없는지 확인하기 위해 사용하는 변수(1이면 ‘+’가 붙은 inst)
* op\* tmp : opcode hash table 의 각 노드들에 접근하기 위한 변수
* char para1[LINE\_MAX], \*para2 : operand를 ‘,’ 단위로 쪼갠 문자열이 저장된다. (para1 =’.’ 이전 , para2 = ‘.’ 이후)

## 모듈 이름: int comp\_symbol(char\* name, char\* tmp)

### 기능

name과 tmp 문자열의 알파벳 내림차순 순서를 알려주는 함수이다. name이 tmp보다 먼저인 경우 return 0, 그 반대인 경우 return 1을 해준다.

name[i]>tmp[i]인 경우 return 0을 해주고, 그 반대인 경우 return 1을 해주는 행위를 각 문자열의 길이 중 더 짧은 길이만큼 수행한다.

만약 더 짧은 문자열 길이만큼 수행했는데도, 함수가 끝나지 않은 경우에는(ab, ababb이런 경우) name의 문자열이 더 길면 return 0, 그렇지 않은 경우에는 return 1을 해준다.

### 사용 변수

* int i: loop counter
* len : name과 tmp중 더 짧은 문자열 길이를 저장하는 변수

## 모듈 이름: int insert\_symbol(char\* name, int adr)

### 기능

symbol table에 name과adr을 담고있는 node를 알파벳 내림차순으로 정렬하여 list에 삽입하는 함수이다. symbol table의 hash key 값은 name 첫글자의 ASCII 값 –‘A’이다. 따라서 hash size는 26이다.

우선 symbol 이름이 숫자로 시작하면 안되므로, name[0]이 ‘0’~’9’ 사이이면 return 0을 하여 함수를 종료한다. 이후read\_symbol(이후 설명될 함수) 함수를 통해 입력받은 name에 해당하는 symbol이 이미 존재하는지 확인 하고, 존재하는 경우 return 0을 해준다. (symbol 중복 선언)

그렇지 않은 경우, name과 adr의 정보를 담고있는 노드를 새로 할당한다. 대응되는 key번째 list가 비어있는 경우, 첫 번째에 이 노드를 연결하고, 그렇지 않은 경우는 정렬을 거친 뒤, 자신보다 알파벳 차수가 낮은 이름을 가진 노드의 앞쪽에 삽입한다.

for문을 통해 key 번째 list의 노드들을 하나씩 접근하고(tmp), 그 노드의 이전 노드(tmp2)를 저장해둔다. 해당 노드(tmp)와 name의 순서를 comp\_symbol함수를 통해 확인하고 comp\_symbol값이 0인 경우 name의 순서가 해당 노드보다 앞쪽인 것이므로, 루프를 중단하고 name노드를 tmp 앞쪽에 연결해준다. 이 경우 3가지로 연결 방법이 나뉜다. 성공적으로 노드를 연결한 경우 return 1을 해준다.

1. 맨 뒤에 삽입하는 경우(tmp==NULL)

name노드의 next 를 NULL로 해주고, tmp 의 next를 name node로 해준다.

1. 맨 처음 삽입하는 경우(tmp2==NULL)

name 노드의 next를 tmp로 해주고, symtable[key]를 name 노드로 해준다.

1. list 중간에 삽입하는 경우

tmp2의 next를 name 노드로, name 노드의 next를 tmp로 해준다.

### 사용 변수

* int key : name에 해당하는 hasy key 값을 저장한다. (name[0]-‘A’)
* sym\* node : name, adr의 정보를 담고있는 새로 할당한 노드
* sym \*tmp, \*tmp2 : name 노드 정렬 및 삽입을 위한 변수로, symtable\_tmp[key] 리스트의 각 노드들에 접근하기 위해 사용한다. (tmp2는 tmp 이전 노드이다(즉, tmp2->next==tmp)

## 모듈 이름: void free\_symbol()

### 기능

symtable의 내용을 모두 free 해주는 함수이다. i=0~25까지 총 26개의 list를 각각 노드들에 접근하여 하나씩 free시킨 뒤, 각 list의 head pointer인 symtable[i]를 NULL로 만들어 준다.

### 사용 변수

* sym\* tmp : 각각의 node에 접근하기 위한 변수
* sym\* tmp2 : tmp를 가리키는 변수이다. tmp2=tmp로 설정한 이후 tmp= tmp->next가 되고 free(tmp2)를 수행하면 노드 한 개를 free 시키게 된다.
* int i : loop counter (0~25)

## 모듈 이름: int read\_symbol(char\* name)

### 기능

name에 해당하는 symbol이 현재 어셈블하고 있는 파일에 선언이 되었는지 확인하고, 그런 경우에는 그 symbol의 Loc(node->adr성분)값을 반환해주는 함수이다. 반대의 경우에는 -1을 반환한다.

symble table key값을 계산하여(name[0]-‘A’) 해당 key번째 list안에 매개변수로 받은 name과 일치하는 symbol이 있는 경우 해당 adr(Loc)을, 만약 List끝까지 존재하지 않는 경우에는 -1을 return해준다.

### 사용 변수

* sym\* tmp = symbol table 노드들에 접근하기 위한 변수
* int key = name의

## 모듈 이름: int command\_sym()

### 기능

symbol 명령어 입력시 수행되는 함수로, symbol table의 내용을 내림차순으로 출력해주는 함수이다. 각 symbol들은 symtable이라는hash table에 저장되어 있다. symtable 사이즈는 26(key = symbol첫글자의 ASCII값) 이며, 각 symbol들은 insert\_symbol함수를 통해 리스트에 넣을 때부터 정렬이 되어 저장되므로 table key = 25부터 0까지 순서대로 출력해주면 내림차순으로 출력이 된다. symbol table이 비어있는 경우에는 return 0을 한다.

### 사용 변수

* int i: loop counter
* sym\* tmp : symtable의 각각의 노드들에 접근하기 위한 변수이다.

## 모듈 이름: int create\_obj(FILE\* fp, char\* prog\_name)

### 기능

fp가 가리키는 파일에 obj 파일 내용을 작성해주는 함수이다. 해당 함수를 수행할 시기에는 프로그램에 대한 정보가( objcode, 에러여부) 이미 code\_set에 모두 저장되어 있기때문에 형식에 맞게 code\_set에서 한줄 한줄 obj code를 얻어와서 파일에 작성해주면 된다. 크게, H record, T record, E record 부분으로 나눌 수 있다.

1. H record

* <H프로그램이름(6글자)시작주소(6-digit)총사이즈(6-digit)>를 파일에 작성한다.
* 프로그램 이름 : prog\_name을 6글자수 만큼 출력한다. 기본으로 설정된 prog\_name은 “INAINA”이다. 만약 글자수가 6개보다 많다면, 뒤의 이름은 무시되고 6글자만 출력해준다. 그리고 6글자보다 적다면, 남은 부분은 ‘ ‘공백으로 채워준다.
* 시작주소: lc[0] , 총사이즈: global 변수 total\_size 에 저장되어 있다.

1. T record

* <T해당라인의 첫 obj코드의lc 값(6)해당라인의코드사이즈합(2)obj코드들>

을 파일에 작성한다.

* i = 1~ last\_line까지 code\_set배열을 살펴본다. i는 해당 T record 라인의 첫 instruction이 코드 상에 위치한 line을 의미한다. obj코드가 없는 instruction의 경우 넘어가면서 (type이 directive이면서 (type>4 인 경우이다) CONST type이 아닌 경우) 시작 line 을 결정한다.
* 시작 라인이 결정되었으면, 코드 사이즈의 총 합이 0x1E를 넘지 않게 해주는 최대 코드 개수를 구한다. 또한 변수가 나오면, 거기서 해당 T record 한줄을 끝내고, 다음 obj 코드가 있는 곳을 시작으로하여 T record를 작성한다.
* 중간에 상수가 있는 경우, 상수사이즈+지금까지의 코드 사이즈 합이 30을 넘는 경우와, 그렇지 않은 경우로 나누어 생각한다.

-2) 30이 넘지 않는 경우, size(현재까지의 코드 사이즈 합) 에 해당 변수 사이즈를 더해주고 다시 반복문(최대 코드 개수 구하는 반복문)을 진행한다.

-1) 30이 넘는 경우, end = 30- size(현재까지의 코드 사이즈 총합), size=30으로 설정한 뒤, 반복문을 나온다. 반복문을 나온 뒤에 기본적으로 작성되는 T record 한줄 이후 해당 상수의 obj 코드를 end사이즈만큼 출력해주고, start를 end로 설정한다. 이후 새로운 T record를 작성할 때, size = (해당 상수 사이즈 – end )으로 설정하고, 최대 코드 개수 길이를 구하는 반복문을 수행한다. 그럼 해당 상수의 남은 obj 코드와 같은 줄에 작성될 코드들의 개수가 구해진다. 이후 최대 코드 개수가 구해져 반복문이 종료되면, 해당 상수를 start부터 상수 사이즈까지 출력하고 반복문을 통해 구한 최대 코드 개수만큼 다른 obj 코드를 출력해준다.

이렇게 상수를 중간에 짤라서 파일에 작성한 경우는, start와 end 의 값으로 구별한다. (기본 start, end 값은 -1)

\* start 가 -1이 아닌 경우, T record의 시작주소/사이즈총합 이후 해당 문자열을 start부터 남은 상수만큼 파일에 작성하고, 이후 남은 size 만큼 obj 코드를 작성한다. 그리고 strat= -1로 초기화 시킨다. \* end가 -1이 아닌 경우, T record의 시작주소/사이즈총합/objcode들 출력이 끝나면, 0~end-1사이즈만큼 해당 상수를 출력하고 start를 end로 설정한다.

1. M record : <M(format 4의 immediate mode가 아닌instruction의 Loc+1)(6)05>을 파일에 작성한다.
2. E record: <E처음 obj코드 나온 instruction의 Loc(6)> 을 파일에 작성한다.

### 사용 변수

* int i,j = loop counter ( i: T record 한 줄 시작 instruction의 코드 line 값, j = i부터 시작해서 한 줄에 최대로 작성할 수 있는 코드의 개수)
* int k =loop counter
* int size =i 부터 i+j 번째 코드까지의 총 사이즈 합이 저장되어 있다.
* int start, end = 상수를 중간에 짤라서 출력한 경우를 구별하기 위한 변수로, 기본 값은 -1이다. 만약 end 가 -1 이 아니라면, 해당 상수를 0~end-1 사이즈 만큼 출력하는 것이고, start가 -1이 아니면, 해당 상수를 start~상수 사이즈 만큼 출력하는 것이다.

## 모듈 이름: int create\_lst(FILE\* fp, char\* inst, int line)

### 기능

*FILE\* fp = 작성할 파일을 가리키는 파일 포인터*

*char\* inst = 프로그램의 line 번째 코드 ( 파일에 있는 한 줄 그대로)*

*int line = 현재 obj code를 만들 line*

fp 가 가리키는 파일에 lst 파일을 작성해주는 함수이다. command\_asm 함수에서 second pass를 수행할 때 실행되는 함수로, first pass에서 계산된 lc배열과 (Location count) code\_set배열의 정보를 바탕으로 해당 line번째 코드의obj code를 생성하여 파일에 작성해주며, 에러 발생 여부를 알려주는 역할을 수행한다. 이 함수는 크게 1) 라인 수 및 코드 출력, 2) obj 코드 생성 전 예외처리, 3) format 1,2 obj 코드 생성, 4) format 3,4 obj 코드 생성 (4.1) immediate mode, 4.2) indirect mode, 4.3) simple addressing, 4.4) offset 계산, 4.6) 최종 offset 체크 및 obj code 생성/작성 )

* 이 함수 안에서 code\_set[line]의 멤버 변수들을 통해, 해당 코드의 opcode number, type, instruction(operation), operand1,2, obj code에 대한 정보를 얻어오고, 수정할 수 있다.

1. 라인 수 및 코드 출력: lst 파일 형식에 맞게 줄번호와 Loc, 코드 한줄을 파일에 작성한다. 모든 코드들이 잘 줄맞춤되어 있어야 하기 떄문에, 모든 inst마다 strlen(inst)번 째 index부터 code\_max까지 ‘ ‘로 채워넣어 준다. 줄번호는 (line+1)\*5 이고, Loc는 lc[line]을 해주면 된다(이 때, inst가 주석문/BASE 바꿔주는 경우/마지막줄일 경우에는 Loc를 출력하지 않고 공백으로 채워둔다.) 그리고 마짐가으로 가공한 inst를 출력시켜주면 1)에서의 모든 기능은 끝난다.
2. obj 코드 생성전 예외처리: 모든 inst들이 obj 코드로 생성되는 것이 아니기 때문에, obj 코드를 본격적으로 계산하기 전에 이러한 예외적인 경우들을 처리해 주는 부분이다. 예외로 처리해주는 경우는 < Base를 바꿔주는 경우/code\_set[line].num==-1인 경우(directive) > 두가지가 있다.

* Base 바꿔주는 부분(LDB 연산인 경우): inst가 LDB인 경우, 전역변수로 선언된 base 의 값을 operand로 받은 상수(10진수)나, symbol의 주소 값으로 바꿔주면 된다. 먼저 read\_symbol 함수를 사용하여 해당 operand를 이름하는 symbol이 있으면 그 주소값으로 base를 수정하고, 함수 return 값이 -1인 경우 make\_int 함수로 해당 operand가 올바른 10진수 형식인지를 체크한다. 만약 operand가 올바른 숫자상수값이면 그 값을 base로 해주고, 그렇지 않은 경우는return 0을 하여 함수를 종료시킨다. 본 코드에서는 base 값은 Immediate mode로만 수정이 가능하다
* Directive: inst가 CONST(상수)일 때를 제외하면 obj 코드가 생성되지 않으므로 파일에 ‘\n’을 작성해주고 끝내면 된다. CONST 인 경우 create\_lst 함수가 실행되기 전에 store\_symbol 함수에서 해당 상수에 대한 obj 코드가 이미 생성되었으므로 code\_set[line].obj를 파일에 작성해주고 함수를 끝낸다. ( return 1)

1. format 1,2 obj 코드 생성: inst type이 1이나 2인 경우 그에 해당하는 obj code를 생성하여 파일에 작성해주는 역할을 수행한다. format 1,2는 공통적으로 앞의 1byte가 opcode 값으로 채워지므로 (code\_set[line].obj)[0], (code\_set[line].obj)[1]을 계산하여 수정해준다. format 1의 경우 operand를 가지는 연산이 없기 때문에, operand[0]!=’\0’인 경우에는 return 0을 하여 함수를 종료한다. format 2의 경우 “CLEAR/TIRX/SVC” 연산은 operand가 한개, 나머지 연산은 모두 operand가 2개여야 한다. 이러한 형식이 맞는지 체크한 후, 맞지 않다면 return 0을 해준다. SHIFTL/SHIFTR 연산을 제오하고는 올바른 형식이라면 각각 (code\_set[line].obj)[2], (code\_set[line].obj)[3]에 read\_reg로 입력받은 레지스터 number를 char 형태로 (read\_reg값 + ‘0’) 설정해준다. SHIFTL/SHIFTR 연산의 경우 (code\_set[line].obj)[3] 에 operand2[0]을 저장한다. (저장 전에 operand2에 담긴 값이 0xf 범위 안에 들어오는지, 올바른 형식인지 체크하고, 에러가 발생한 경우 return 0한다)

(operand가 1개인 연산의 경우 (code\_set[line].obj)[3]의 값을 ‘0’으로 설정해준다.) 모든 obj 코드 계산이 끝났으면, obj의마지막에 ‘\0’를 저장하고 파일에 해당 obj 코드를 작성하고 함수를 종료한다(return 1)

* 이후에도 함수가 종료되지 않은 경우는 inst type이 3이나 4이다. 이러한 type에서 operand를 받지 않는 연산은 “RSUB” 밖에 없으므로 operand[0]==’\0’(operand가 없는 경우)인데 연산이 RSUB이 아닌 경우는 return 0을 해주고, RSUB인 경우는 obj 코드를 계산하여 파일에 작성하고 함수를 종료한다. (return 1)

1. format 3,4 obj 코드 생성: format 3,4d의 경우 우선 ni 비트를 결정하고, offset(displacement)을 계산하여 xbpe 비트를 결정하는 식으로 obj 코드를 생성할 수 있다

4.1) Immediate mode

operand[0]이 #인 경우에 해당한다. #뒤에는 숫자(상수)나, symbol이 올 수 있는데, symbol이 오는 경우는 offset이 다른 일반 경우와 동일한 방법으로 계산되므로 이 단계에서는 숫자인 경우에만 offset을 바꿔준다.

우선 ni비트를 01로 설정하고, operand++하여 #을 제거하고, read\_symbol(operand)의 return 값이 -1인 경우와 그렇지 않은 경우로 나눈다. -1이 아닌 경우 #symbol 형식의 code이므로 따로 offset을 설정해주지 않고 넘어가며, -1인 경우에는 make\_int 함수를 통해 해당 operand문자열에 담긴 값이 알맞은 10진수 형식인지를 확인하고, 잘못된 형식인 경우 return 0을 한다. 올바른 경우 offset에 해당 값을 저장하고 이 값이 범위를 넘지 않는지 체크한다. (type이 3인 경우 offset> 0xfff, type이 4인 경우 offset>0xfffff 인 경우 범위를 넘어간 것으로, return 0한다.) # 뒤의 숫자에 음수가 오는 것은 오류로 처리하였다.

4.2) Indirect mode

operand[0]이 @인 경우에 해당한다. 이 경우 operand++하여 @을 제거하고 ni비트를 10으로 설정한다. 이후 symbol table에 operand가 있는지 확인하고(read\_symbol함수), 존재하지 않는다면 return 0을 해준다. indirect mode의 경우 offset이 일반적인 경우와 동일한 방식으로 계산되므로 별로로 offset을 수정하지 않고 symbol table 존재 여부만 확인 한 뒤 넘어가면 된다.

@상수 형태의 입력은 오류 처리하였다. @symbol 만 올바른 형태이다.

4.3) Simple addressing

4.1)과 4.2)에 해당하지 않는 경우는 모두 simple addressing이 된다. ni를 11로 설정해주고, 해당 instruction의 type이 4이면 e비트를 1로 설정하고 offset에 operand(symbol)의 주소값을 그대로 저장한다. operand의 주소값은 read\_symbol 함수로 얻어올 수 있으며, 해당 symbol이 존재하지 않는 경우는 return 0을 해준다.

4.5) offset 계산

offset의 값은 123456789로 초기화 되어있다. 만약 4.5)단계에서 offset이 123456789인 경우는 아직 offset이 계산되지 않았다는 의미이므로 offset을 계산해주어야 한다. (123456789가 아닌 경우는 #숫자 or format 4인 경우에 해당한다.)

pc=lc[line+1]로 설정하면, offset= operand(symbol)의 Loc – PC 가 된다. 만약 이 값이 -0xfff ~0xfff 사이라면 PC relative 방식으로 addressing 할 수 있으므로 bp 비트를 01로 설정해준다. 그리고 offset이 음수인 경우 0x1000을 더하여 2의 보수 형태로 나타낸다.

만약 offset의 값이 범위를 벗어난 경우, Base relative 방식을 사용해야 한다. 따라서 bp를 10으로 설정하고, offset = operand(symbol) Loc – base(base register, 전역변수로 선언됨) 로 수정해준다. 이 경우 offset이 음수가 되면 return 0을 해준다.

이후 해당 instruction이 index 모드인지를 확인하여(operand2==”X”인 경우) 인덱스 모드인 경우 offset에 x(x register, 전역변수로 선언됨) 값을 더해주고, x 비트를 1로 설정한다.

4.6) 최종 offset 체크 및 obj 코드 생성/ 작성

최종적으로 구한 offset의 범위를 체크한다. (type=4 ->offset > 0xfffff / type=3 -> offset > 0xfff인 경우 에러이다. 음수인 경우는 Pc relative 방식 offset 계산 할 때 체크하였으므로 할 필요없다.)

체크를 끝마친 경우, 해당 obj 코드를 code\_set[line].obj에 형식을 맞춰 저장시킨 뒤 파일에 작성해주고 함수를 종료한다.(return 1)

### 사용 변수

* int opcode : 해당 line번째 코드 operation의 opcode 값
* int nixbpe[6] : nixbit 비트를 저장하는 배열
* int offset = 123456789 : obj 코드 생성시 계산되는 offset을 저장하는 변수
* int obj[8] : obj 코드를 계산하는 과정에서 필요한 배열로, obj[i] = obj 코드의 i번째 half byte(digit)의 정수값을 의미한다. ( 0< obj[i] <16 )
* char\* operand : 해당 line 번째 코드의 operand를 저장하는 문자열

## 모듈 이름: int read\_reg(char\* reg)

### 기능

reg에 해당하는 레지스터 number을 return 해주는 함수이다. strcmp로 reg와 각 레지스터 이름을 비교하여(A,B,S,T,F,L,PC,X,SW) 각 레지스터에 맞는 숫자를 반환하고, 잘못된 레지스터를 입력한 경우 return -1을 한다.

### 사용 변수

없음

## 모듈이름: char make\_char(int n)

### 기능

정수형 변수 n(0~15) 에 해당하는 문자형 타입을 return 하는 함수이다. n<0 거나 n>=16 인 경우에는 ‘\0’을 반환하고, 그렇지 않은 경우 n이 0~9 인 경우 ‘0’+n을, n이 10~15인 경우에는 (n-10)+’A’를 반환한다.

### 사용변수

* char c: int n의 char 변환 값을 저장하는 변수

## 모듈이름: int make\_int(char\* s)

### 기능

매개변수로 받은 문자열 s를 int로 변환한 값을 return 하는 함수이다. s의 시작부터 끝까지 읽으며, 그 값이 ‘0’~’9’ 사이에 있지 않으면 return -1을 하여 잘못된 형식임을 알리고, 그렇지 않으면 result 에 result\*10과 해당 문자의 int 값( s[i]-‘0’) 을 더해주며 result를 계산한다.

(ex) make\_int(“321” ) = 321)

### 사용변수

* int i : loop counter
* int result: 넘겨받은 문자열 s를 int로 변환한 값이 저장된다.

# 전역 변수 정의

## sym\* symtable[26]

symbol table 을 위한변수로 최종적으로 assemble이 성공한 프로그램의 symbol table이 저장된다. (symtable[0] =’A’로 시작하는 symbol ~ symtable[25]=’Z’로 시작하는 symbol)

## code code\_set[CODE\_MAX]

한 프로그램의 코드 한줄 한줄에 대한 정보를 저장하기 위한 CODE structure array이다. ( CODE\_MAX = 0xFFFFF)

## int lc[MEMORY\_ROW]

한 프로그램의 각 line 마다의 Location Count 값을 저장하는 배열이다.

## int a,t,f,l,sw,pc,base,x,s.nobase

SIX/XE 레지스터 변수와 nobase flag이다 . nobase=1,나머지는 모두 0으로 초기화 되어있다.

## total\_size=0

한 프로그램의 총 사이즈를 나타내는 변수이다.

## int start\_check, end\_check

프로그램 안에 START, END가 있는지를 알려주는 변수로 초기값은 0이다. 각각 START나 END가 존재할 경우 1로 세팅된다.

## int code\_max

한 프로그램의 각각 코드 길이(strlen) 중 제일 긴 코드 길이를 저장한다. lst 파일 작성시 줄을 맞추는 용도로 사용한다.

# 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

#define LINE\_MAX 256

#define CODE\_MAX 0xfffff

//예외처리

#define WRONG\_ADR 1

#define WRONG\_VAL 2

#define WRONG\_PARAMETER 3

#define WRONG\_CMM 4

#define INPUT\_SIZE\_OVER 5

#define WRONG\_FILE 6

#define ASSEMBLE\_FAIL 7

#define NO\_SYMBOL 8

//명령어 num

#define q 3

#define VAR 9

#define CONST 10

#define BASE 11

#define DIREC 12

#define COMMENT 13

#define A 0

#define X 1

#define L 2

#define B 3

#define S 4

#define T 5

#define F 6

#define PC 8

#define SW 9

//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;

typedef struct SYM{

int adr;

char name[30];

struct SYM\* next;

}sym;

typedef struct CODE{

int adr;

int num;

char inst[20];

char operand[30];

char operand2[30];

int type;

char obj[100];

}code;

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 hash\_key(char\* s);

void create\_hash\_table();

int command\_file(char\* name);

int command\_asm(char\* name);

int command\_sym();

int create\_lst(FILE\* fp,char\* inst, int line);

int create\_obj(FILE\* fp,char\* prog\_name);

int insert\_code(char\* inst, char\* operand, int line);

int store\_symbol(int adr, int size, char\* s, int line);

int comp\_symbol(char \*name, char\* tmp);

int insert\_symbol(char\* name, int adr);

int read\_symbol(char\* name);

void free\_symbol();

int read\_reg(char\* s);

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);

char make\_char(int n);

int make\_int(char\* s);

**<20161601.c>**

#include "20161601.h"

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

ll\* hlist=NULL; //history 저장 linked list

sym\* symtable[26];

code code\_set[CODE\_MAX];

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

int lc[MEMORY\_ROW];

int cur\_adr=0; // dump 수행시 시작 메모리 번지

int start\_check=0, end\_check=0;

int a=0,t=0,f=0,l=0,sw=0,pc=0,base=0,x=0,s=0,nobase=1; // 레지스터

int last\_line=0,total\_size=0, code\_max=0;

//코드 라인 개수, 총 코드 사이즈, 코드 한 줄 최대 길이

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();

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

//파일 이름 잘못 입력한 경우

if(i!=2){

print\_error(WRONG\_PARAMETER);

return 0;

}

//해당 이름의 파일이 없는 경우 history에는 저장

command\_file(command\_words[1]);

}

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

//파일 이름 잘못 입력한 경우

if(i!=2){

print\_error(WRONG\_FILE);

return 0;

}

//파일 이름 확인하기 (.asm 파일인지/ 올바른 형식인지)

tmp= strtok(command\_words[1], ".");

if( tmp ==NULL){

print\_error(WRONG\_FILE);

return 0;

}

//파일이름 ('.' 앞 문자열) 저장하고 tmp에 확장자명 저장

strcpy(command\_words[1],tmp);

tmp = strtok(NULL," \r\n\t");

//확장자명이 asm 이 아니거나 파일 입력을 잘못한 경우

if(strcmp(tmp, "asm")!=0){

print\_error(WRONG\_FILE);

return 0;

}

//어셈블이 실패한 경우 (history 에는 저장x)

if(!command\_asm(command\_words[1])){

free\_symbol();

return 0;

}

}

else if(strcmp(command, "symbol")==0){

if(!command\_sym()){

print\_error(NO\_SYMBOL);

}

}

//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");

else if(error==WRONG\_FILE)

printf("Wrong file name\n");

else if(error==ASSEMBLE\_FAIL)

printf("Assemble is failed\n");

else if(error==NO\_SYMBOL)

printf("Symbol table does not exist\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;

}

int make\_int(char\* s){

int i, result=0;

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

result\*=10;

if(s[i]<'0' || s[i]>'9')

return -1;

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

}

return result;

}

char make\_char(int n) {

char c;

if (n<0 || n >= 16)

return '\0';

if (n<10)

c = n + '0';

else

c = (n - 10) + 'A';

return c;

}

//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");

puts("assemble filename");

puts("type filename");

puts("symbol");

}

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;) {

t2 = t;

t=t->next;

free(t2);

}

}

free\_symbol();

}

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");

}

}

int command\_file(char\* name){

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

char c[LINE\_MAX];

//해당 이름의 파일이 현재 디렉토리에 없는 경우 에러메시지

if(fp==NULL){

print\_error(WRONG\_FILE);

return 1;

}

//파일이 존재하면 내용 출력

while(fgets(c,LINE\_MAX,fp)!=NULL)

printf("%s",c);

fclose(fp);

return 1;

}

int command\_asm(char\* name){

FILE \*fp;

FILE \*fp\_lst, \*fp\_obj;

char asmfile[LINE\_MAX],\*lstfile,objfile[LINE\_MAX];

char c[LINE\_MAX],\*tmp, \*tmp2,prog\_name[7]={"INAINA"};

char inst[5][LINE\_MAX];

int i,line=-1,tnum;

//name.asm 파일 열기

strcpy(asmfile, name);

strcat(asmfile, ".asm");

fp=fopen(asmfile, "r");

if(fp==NULL){

print\_error(WRONG\_FILE);

return 0;

}

free\_symbol();

nobase=1;

lc[0]=0;

//first pass

while(fgets(c, LINE\_MAX,fp)!=NULL){

line++;

tnum=3;

lc[line+1]=lc[line];

if(c[0]=='.'){

code\_set[line].type=COMMENT;

continue;

}

//symbol 아닌경우, 코드 한줄을 2부분으로(tnum=2) 나눈다(inst parameter)

if(c[0]==' '||c[0]=='\t' || c[0]=='\n')

tnum=2;

if( code\_max < (int)strlen(c))

code\_max= (int)strlen(c);

//tmp2에는 tmun-1번 문자열을 짜르고 난 뒤의 남은 문자열을 저장할 것

tmp2=c;

for(i=0; i<tnum-1; i++){

tmp=strtok\_r(tmp2," \r\t\n",&tmp2);

if(tmp==NULL)

break;

strcpy(inst[i],remove\_space(tmp));

}

tmp2=remove\_space(tmp2);

//symbol인 경우

if(tnum==3){

if(!insert\_code(inst[1],tmp2, line)){ // 에러발생시 종료

printf("Error! Line %d\n", (line+1)\*5);

return 0;

}

//프로그램 이름이면

if(line==0 && strcmp(inst[1],"START")==0)

strncpy(prog\_name,inst[0],6);

else if(!insert\_symbol(inst[0],lc[line])){

printf("Error! Line %d\n", (line+1)\*5);

return 0;

}

}

//그외

else if(tnum==2){

if(!insert\_code(inst[0],tmp2,line)){

printf("Error! Line %d\n", (line+1)\*5);

return 0;

}

}

}

//END 없는 경우

if( end\_check==0){

printf("There dose not exist END directive\n");

return 0;

}

//lst, obj 파일 생성

lstfile=name;

strcpy(objfile, name);

strcat(lstfile, ".lst");

fp\_lst=fopen(lstfile,"wt");

strcat(objfile,".obj");

fp\_obj=fopen(objfile, "wt");

//파일 생성 오류시 에러 메시지 출력 후 종료

if(fp\_lst==NULL || fp\_obj ==NULL){

printf("File Error\n");

return 0;

}

//2nd pass

last\_line=line-1;

total\_size= lc[last\_line]-lc[0];

code\_max= code\_max + 4;

fclose(fp);

fp=fopen(asmfile,"r");

//lst 파일 생성

for(i=0; i<=last\_line; i++){

fgets(c, LINE\_MAX, fp);

if(c[0]!='.'){

if(!create\_lst(fp\_lst, c, i)){

printf("Error! Line %d\n", (i+1)\*5);

printf("Error line => %s\n", c);

fclose(fp\_lst);

fclose(fp\_obj);

fclose(fp);

//파일 삭제

remove(lstfile);

remove(objfile);

return 0;

}

else

fprintf(fp\_lst, "\n");

}

else

fprintf(fp\_lst,"%d\t%s",(i+1)\*5, c);

}

//obj 파일 생성

create\_obj(fp\_obj,prog\_name);

start\_check=0;

end\_check=0;

fclose(fp\_lst);

fclose(fp\_obj);

fclose(fp);

printf("output file : [%s], [%s]\n",lstfile,objfile);

return 1;

}

int command\_sym() {

int i,flag=1;

sym\* tmp;

for (i = 25; i >= 0; i--) {

if(symtable[i]!=NULL)

flag=0;

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

//마지막 라인의 경우 엔터 제거

if (i == 0 && tmp->next == NULL)

printf("\t%s\t%04X", tmp->name, tmp->adr);

else

printf("\t%s\t%04X\n", tmp->name, tmp->adr);

}

}

//sybol table 이 비어있는 경우

if (flag==1)

return 0;

return 1;

}

int create\_obj(FILE\* fp, char\* prog\_name) {

int i, j = 0, k, size = 0, start = -1, end = -1;

fprintf(fp, "H%s", prog\_name);

for (i = (int)strlen(prog\_name); i<6; i++)

fprintf(fp, " ");

fprintf(fp, "%06X%06X\n", lc[0], total\_size);

for (i = 1; i <= last\_line;) {

//obj code 없는 문장

if (code\_set[i].type>4 && code\_set[i].type != CONST) {

i++;

continue;

}

//상수 중간에 짤라서 출력한 경우, 남은 obj 코드 출력

if (end != -1) {

size = (int)strlen(code\_set[i].obj) / 2 - end;

end = -1;

j = 1;

}

while (i + j <= last\_line) {

//obj code가 없는 경우

if (code\_set[i + j].type == DIREC || code\_set[i + j].type == BASE || code\_set[i + j].type == COMMENT) {

j++;

continue;

}

//변수를 선언한 경우 변수 선언이 끝나고 난 뒤부터 시작

if (code\_set[i + j].type == VAR) {

while (code\_set[i + j].type == VAR) {

if (i + j<last\_line)

j++;

}

break;

}

//상수인 경우

if (code\_set[i + j].type == CONST) {

//0x1E 넘어가면 나눠서 출력

if ((int)strlen(code\_set[i + j].obj) / 2 + size >30) {

end = 30 - size;

size = 30;

break;

}

else {

size += (int)strlen(code\_set[i + j].obj) / 2;

j++;

}

continue;

}

if (size + code\_set[i + j].type>30)

break;

size += code\_set[i + j].type;

j++;

}

//사이즈 만큼 objcode 출력

fprintf(fp, "T%06X%02X", code\_set[i].adr, size);

if (start != -1) {

for (k = start; k<(int)(strlen(code\_set[i].obj) / 2); k++)

fprintf(fp, "%c%c", (code\_set[i].obj)[2 \* k], (code\_set[i].obj)[2 \* k + 1]);

start = -1;

i++;

}

for (k = i; k<i + j; k++) {

if ((code\_set[k].obj)[0] != '\0')

fprintf(fp, "%s", code\_set[k].obj);

}

//상수가 중간에 짤려서 출력된 경우

if (end != -1) {

for (k = 0; k<end; k++) {

fprintf(fp, "%c%c", (code\_set[i + j].obj)[2 \* k], (code\_set[i + j].obj)[2 \* k + 1]);

}

start = k;

}

fprintf(fp, "\n");

i += j;

size = 0;

j = 0;

}

for(i=0; i<=last\_line; i++){

if(code\_set[i].type==4){

//immediate mode 아닌 경우

if((code\_set[i].operand)[0]!='#'){

fprintf(fp,"M%06X05\n", code\_set[i].adr+1);

}

}

}

//End Record obj 코드 생성된 첫 inst의 주소 출력

for (i = 0; i<last\_line; i++) {

if (code\_set[i].num != -1) {

fprintf(fp, "E%06X\n", code\_set[i].adr);

break;

}

}

return 1;

}

int create\_lst(FILE\* fp, char\* inst, int line) {

int opcode, i, offset = 123456789;

int nixbpe[6];

int obj[8]; //object code

char\* operand;

if (inst == NULL)

return 0;

//엔터 제거

if (inst[strlen(inst) - 1] == '\n') {

//줄간격 맞추기

for (i = (int)strlen(inst) - 1; i <= code\_max; i++)

inst[i] = ' ';

inst[i] = '\0';

}

//line 출력

fprintf(fp, "%d\t", (line + 1) \* 5);

//lc 출력 안해도 되는 경우

if (inst[0] == '.' || code\_set[line].type == COMMENT || line == last\_line)

fprintf(fp, " \t");

//LoC

else if (lc[line]>0xffff)

fprintf(fp, "%05X ", lc[line]);

else

fprintf(fp, "%04X ", lc[line]);

fprintf(fp, "%s\t", inst); //Inst

opcode = code\_set[line].num;

operand = code\_set[line].operand;

pc = code\_set[line + 1].adr; //PC는 다음 inst adr

//COMMENT인 경우

if (code\_set[line].type == COMMENT) {

return 1;

}

//object code 생성할 필요 없는 경우

if (opcode == -1) {

if (code\_set[line].type == CONST) {

fprintf(fp, "%s", code\_set[line].obj);

}

return 1;

}

if(strcmp(code\_set[line].inst,"LDB")==0){

base=0;

//LDB 연산은 오직 immediate mode 에서만 가능 #

if(operand[0]=='#'){

//symbol이면 Loc 그대로 입력

if( read\_symbol(operand+1)!=-1)

base = read\_symbol(operand+1);

else if(make\_int(operand+1)!=-1)

base = make\_int(operand+1);

else

return 0;

}

nobase=0;

}

//obj code 생성!

//nixbpe 모두 0으로 초기화

for (i = 0; i<6; i++)

nixbpe[i] = 0;

//format 1 or 2

if (code\_set[line].type == 1 || code\_set[line].type == 2) {

//format 1의 겨우 operand 있으면 오류

if (code\_set[line].type == 1 && operand[0] != '\0')

return 0;

(code\_set[line].obj)[0] = make\_char(opcode / 16);

(code\_set[line].obj)[1] = make\_char(opcode % 16);

(code\_set[line].obj)[2] = '\0';

//format 2인 경우

if (code\_set[line].type == 2) {

//parameter 1개인 inst

if (!strcmp(code\_set[line].inst, "CLEAR") || !strcmp(code\_set[line].inst, "TIXR") || !strcmp(code\_set[line].inst, "SVC")) {

if ((code\_set[line].operand2)[0] != '\0')

return 0;

(code\_set[line].obj)[2] = '0' + read\_reg(code\_set[line].operand);

(code\_set[line].obj)[3] = '0';

}

//parameter 2개

else {

if ((code\_set[line].operand2)[0] == '\0')

return 0;

//reg, n 형식 연산

if (!strcmp(code\_set[line].inst, "SHIFTL") || !strcmp(code\_set[line].inst, "SHIFTR")) {

if (make\_int(code\_set[line].operand2) == -1 || make\_int(code\_set[line].operand2)>0xf)

return 0;

(code\_set[line].obj)[2] = '0' + read\_reg(code\_set[line].operand);

(code\_set[line].obj)[3] = (code\_set[line].operand2)[0];

}

else {

(code\_set[line].obj)[2] = '0' + read\_reg(code\_set[line].operand);

(code\_set[line].obj)[3] = '0' + read\_reg(code\_set[line].operand2);

}

}

(code\_set[line].obj)[4] = '\0';

}

fprintf(fp, "%s", code\_set[line].obj);

return 1;

}

//operand 없는 경우( RSUB 아닌 경우 에러)

if (operand[0] == '\0') {

if (strcmp(code\_set[line].inst, "RSUB") != 0)

return 0;

(code\_set[line].obj)[0] = make\_char((opcode + 3) / 16);

(code\_set[line].obj)[1] = make\_char((opcode + 3) % 16);

strcpy(code\_set[line].obj + 2, "0000");

fprintf(fp, "%X0000", opcode + 3);

return 1;

}

//immediate mode

if (operand[0] == '#') {

nixbpe[1] = 1;

operand++; //#제거

//숫자인 경우

if (read\_symbol(operand) == -1) {

//decimal

if (make\_int(operand) != -1)

offset = make\_int(operand);

//잘못된 형식

else

return 0;

//range 넘어가는 경우

if (code\_set[line].type == 3 && offset>0xFFF)

return 0;

if (code\_set[line].type == 4 && offset>0xFFFFF)

return 0;

}

}

//indirect

else if (operand[0] == '@') {

operand++;

nixbpe[0] = 1;

//해당 symbol없는 경우

if (read\_symbol(operand) == -1)

return 0;

}

//simple addressing

else {

nixbpe[0] = 1;

nixbpe[1] = 1;

//format 4인 경우 offset 에 read adr 저장

if (code\_set[line].type == 4) {

nixbpe[5] = 1;

offset = read\_symbol(operand);

}

}

//offset 계산해야 하는 경우(immediate mode 중상수 대입을 제외하곤 모두 offset 계산필요)

if (offset == 123456789) {

pc = lc[line + 1];

offset = read\_symbol(operand) - pc;

//base relative

if (offset>0xFFF || offset\*(-1)> 0xFFF) {

//base 값이 없는 경우

if(nobase==1)

return 0;

nixbpe[3] = 1;

offset = read\_symbol(operand) - base;

//offset 음수인 경우 에러

if (offset<0)

return 0;

}

//PC

else {

nixbpe[4] = 1;

//음수인 경우 2의 보수의 decimal 값으로 ( a+(-a) = 0x1000 -> -a= 0x1000-a)

if (offset<0)

offset = 0x1000 + offset;

}

}

//index 모드인 경우

if (code\_set[line].type>2 && strcmp(code\_set[line].operand2, "X") == 0) {

//simple addressing인 경우에만 index 모드 가능하다

if (!(nixbpe[0] == 1 && nixbpe[1] == 1))

return 0;

nixbpe[2] = 1;

offset += x;

}

//최종offset 체크

if ((code\_set[line].type == 4 && offset>0xFFFFF) || (code\_set[line].type == 3 && offset >0xFFF))

return 0;

//obj code 계산

obj[0] = opcode / 16;

obj[1] = opcode % 16 + nixbpe[0] \* 2 + nixbpe[1];

obj[2] = 0;

//format 3

if (code\_set[line].type == 3) {

obj[3] = offset / 0x100;

obj[4] = (offset % 0x100) / 0x10;

obj[5] = (offset % 0x100) % 0x10;

obj[6] = '\0';

for (i = 2; i<6; i++)

obj[2] = obj[2] \* 2 + nixbpe[i];

//obj code 저장

for (i = 0; i<6; i++)

(code\_set[line].obj)[i] = make\_char(obj[i]);

(code\_set[line].obj)[6] = '\0';

}

//format 4

else {

nixbpe[5]=1;

for (i = 2; i<6; i++)

obj[2] = obj[2] \* 2 + nixbpe[i];

obj[3] = offset / 0x10000;

obj[4] = (offset % 0x10000) / 0x1000;

obj[5] = ((offset % 0x10000) % 0x1000) / 0x100;

obj[6] = (((offset % 0x10000) % 0x1000) % 0x100) / 0x10;

obj[7] = (((offset % 0x10000) % 0x1000) % 0x100) % 0x10;

for (i = 0; i<8; i++)

(code\_set[line].obj)[i] = make\_char(obj[i]);

(code\_set[line].obj)[8] = '\0';

}

fprintf(fp, "%s", code\_set[line].obj);

return 1;

}

int store\_symbol(int adr , int size, char\* s,int line){

int i,len;

int tmp2;

char\* tmp;

tmp=strtok(s,"'");

//문자인 경우

if(!strcmp(tmp,"C")){

//문자인데 WORD로 선언한 경우 에러

if(size==3)

return 0;

tmp = strtok(NULL,"'");

//올바른 형식이 아닌 경우

if(strtok(NULL, " \r\t\n")!=NULL)

return 0;

len=(int)strlen(tmp);

if(len>30)

return 0;

//아스키 코드값 저장(최대상수문자열길이는30= 3byte)

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

tmp2=(int)(tmp[i]);

(code\_set[line].obj)[2\*i]=make\_char(tmp2/16);

(code\_set[line].obj)[2\*i+1]=make\_char(tmp2%16);

}

(code\_set[line].obj)[2\*len]='\0';

lc[line+1]=lc[line]+strlen(tmp);

return 1;

}

//16진수

else if(!strcmp(tmp, "X")|| !strcmp(tmp,"x")){

tmp = strtok(NULL,"'");

//올바른 입력 아닌 경우

if(tmp==NULL || strtok(NULL," \r\t\n")!=NULL)

return 0;

tmp2=make\_hex(tmp);

//올바른 16진수 형태가 아닌경우

if( tmp2==-1||( size==1 &&tmp2>0xff ) || (size==3 && tmp2>0xffffff ))

return 0;

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

(code\_set[line].obj)[i]= tmp[i];

}

(code\_set[line].obj)[i]='\0';

lc[line+1]=lc[line]+size;

}

//형식에 맞지 않음

else

return 0;

return 1;

}

int insert\_code(char\* inst, char\* operand, int line){

int key,flag=-1;

op\* tmp=NULL;

char para1[LINE\_MAX],\*para2=NULL;

strcpy(code\_set[line].inst, inst);

if(\*operand!=0)

strcpy(code\_set[line].operand,operand);

else

(code\_set[line].operand)[0]='\0';

(code\_set[line].operand2)[0]='\0';

(code\_set[line].obj)[0]='\0';

code\_set[line].adr=lc[line];

code\_set[line].type=3;

code\_set[line].num=-1;

//directives인 경우

if(strcmp(inst, "START")==0){

lc[0]=make\_int(operand);

lc[1]=lc[0];

code\_set[line].type = DIREC;

start\_check=1;

return 1;

}

if(strcmp(inst, "END")==0){

code\_set[line].type = DIREC;

end\_check=1;

return 1;

}

//상수인 경우

if(strcmp(inst, "WORD")==0){

code\_set[line].type = CONST;

lc[line+1]=lc[line]+3;

if(!store\_symbol(lc[line], 3, operand,line))

return 0;

return 1;

}

if(strcmp(inst, "BYTE")==0){

code\_set[line].type = CONST;

lc[line+1]=lc[line]+1;

if(!store\_symbol(lc[line],1,operand,line))

return 0;

return 1;

}

//변수

if(strcmp(inst, "RESW")==0){

code\_set[line].type = VAR;

lc[line+1]= lc[line]+ make\_int(operand)\*3;

return 1;

}

if(strcmp(inst, "RESB")==0){

code\_set[line].type = VAR;

lc[line+1]=lc[line]+make\_int(operand);

return 1;

}

/\*

if(strcmp(inst, "BASE")==0){

code\_set[line].type = BASE;

return 1;

}\*/

//format 4인 경우, 체크하고 +제거

if(inst[0]=='+'){

flag=1;

inst++;

}

key= hash\_key(inst);

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

// 입력받은 명령어를 찾은 경우

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

//+ 입력 했는데 format 1이나 2인 경우 에러

if(flag==1 && strlen(tmp->op\_type)==1)

return 0;

//operand가 있으면(0이 아니면) parameter ','단위로 나누기

if( \*operand!=0){

strcpy(para1,operand);

strcpy(code\_set[line].operand,remove\_space(strtok(para1,",")));

para2=strtok(NULL," ");

}

//parameter가 2개인 경우 operand2에 저장

if(para2!=NULL){

strcpy(code\_set[line].operand2,remove\_space(para2));

//잘못된 형식(매개변수 너무많음)

if(strtok(NULL," \r\t\n")!=NULL)

return 0;

}

code\_set[line].num= tmp->op\_num;

//해당 inst type 저장(op\_type은char배열 타입)

if(strlen(tmp->op\_type)==1)

code\_set[line].type= (int)(tmp->op\_type)[0] -'0';

else if( flag==1)

code\_set[line].type= 4;

else

code\_set[line].type=3;

//Locaton count 계산

code\_set[line].adr=lc[line];

lc[line+1]=lc[line]+code\_set[line].type;

return 1;

}

}

//명령어 없는 경우 주석

code\_set[line].type=COMMENT;

return 1;

}

int comp\_symbol(char \*name, char\* tmp){

int i,len= (int)strlen(name);

if( len>(int)strlen(tmp))

len = (int)strlen(tmp);

for(i=1; i<len; i++){

if( name[i] > tmp[i])

return 0;

if(name[i]< tmp[i])

return 1;

}

if( (int)strlen(name) > (int)strlen(tmp))

return 0;

return 1;

}

//Loc가 ard인 'name' symbol을 symbol table 에 저장

int insert\_symbol(char\* name, int adr){

int key=name[0]-'A';

sym \*node, \*tmp=NULL, \*tmp2=NULL;

//이름 첫글자가 숫자이면 에러

if( name[0]>='0' && name[0]<'9')

return 0;

//새로운 노드 생성

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

strcpy(node->name,name);

node->adr=adr;

//처음 삽입

if( symtable[key]==NULL){

node->next= symtable[key];

symtable[key] = node;

}

//정렬하여 삽입

else{

tmp2= NULL;

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

if( comp\_symbol(name, tmp->name)==0)

break;

tmp2=tmp;

}

//맨뒤에 삽입

if(tmp==NULL){

tmp2->next= node;

node->next=NULL;

}

//맨처음 삽입

else if(tmp2==NULL){

node->next= tmp;

symtable[key]= node;

}

//중간에 삽입

else{

tmp2->next = node;

node->next= tmp;

}

}

return 1;

}

void free\_symbol(){

sym\* tmp, \*tmp2;

int i;

for(i=0; i<26; i++){

if(symtable[i]==NULL)

continue;

for(tmp= symtable[i]; tmp!=NULL;){

tmp2= tmp;

tmp=tmp->next;

free(tmp2);

}

symtable[i]=NULL;

}

}

//name에 해당하는 sym의 adr return (없으면 -1 return )

int read\_symbol(char\* name){

int key= name[0]-'A';

sym\* tmp;

if(symtable[key]!=NULL){

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

if( strcmp(tmp->name, name)==0)

return tmp->adr;

}

}

return -1;

}

int read\_reg(char\* reg){

if(!strcmp(reg, "A"))

return A;

if(!strcmp(reg, "B"))

return B;

if(!strcmp(reg, "S"))

return S;

if(!strcmp(reg, "T"))

return T;

if(!strcmp(reg, "F"))

return F;

if(!strcmp(reg, "L"))

return L;

if(!strcmp(reg, "PC"))

return PC;

if(!strcmp(reg, "X"))

return X;

if(!strcmp(reg, "SW"))

return SW;

return -1;

}