

---

과목 명: 시스템프로그래밍

담당 교수 명: 서 정 연

<<Assignment 5>>

서강대학교 컴퓨터공학과

20101631

모지수

---

## 내용

1. 프로그램 개요	6
2. 프로그램 설명	7
2.1 프로그램 흐름도	7
3. 모듈 정의	8
3.1 모듈 이름 : main	8
3.1.1 기능	8
3.1.2 사용 변수	8
3.2 모듈 이름: addq	9
3.2.1 기능	9
3.2.2 사용 변수	9
3.3 모듈이름: HexToDec	9
3.3.1 기능	9
3.3.2 사용변수	9
3.4 모듈이름: DecToHex	9
3.4.1 기능	9
3.4.2 사용변수	9
3.5 모듈이름: hashFunc_OP	9
3.5.1 기능	9
3.5.2 사용변수	9
3.6 모듈이름: hashFunc_MN	10
3.6.1 기능	10
3.6.2 사용변수	10
3.7 모듈이름: initHash	10
3.7.1 기능	10
3.7.2 사용변수	10
3.8 모듈이름: showTabl	10
3.8.1 기능	10
3.8.2 사용변수	10
3.9 모듈이름: isHex	11
3.9.1 기능	11
3.9.2 사용변수	11
3.10 모듈이름: getCmd	11
3.10.1 기능	11

---

3.10.2 사용변수	11
3.11 모듈이름: checkArgm	11
3.11.1 기능	11
3.11.2 사용변수	12
3.12 모듈이름: getArgm	12
3.12.1 기능	12
3.12.2 사용변수	13
3.13 모듈이름: showHelp	13
3.13.1 기능	13
3.13.2 사용변수	14
3.14 모듈이름: showDir	14
3.14.1 기능	14
3.14.2 사용변수	14
3.15 모듈이름: showHist	14
3.15.1 기능	14
3.15.2 사용변수	14
3.16 모듈이름: dumpLine	14
3.16.1 기능	14
3.16.2 사용변수	15
3.17 모듈이름: Dump	15
3.17.1 기능	15
3.17.2 사용변수	16
3.18 모듈이름: Edit	16
3.18.1 기능	16
3.18.2 사용변수	17
3.19 모듈이름: Fill	17
3.19.1 기능	17
3.19.2 사용변수	17
3.20 모듈이름: Reset	17
3.20.1 기능	17
3.20.2 사용변수	17
3.21 모듈이름: OPtoMN	17
3.21.1 기능	17
3.21.2 사용변수	17
3.22 모듈이름: MNtoOP	17
3.22.1 기능	17
3.22.2 사용변수	17

---

3.23	모듈이름: View	18
3.23.1	기능	18
3.23.2	사용변수	18
3.24	모듈이름: Assemble	21
3.24.1	기능	21
3.24.2	사용변수	21
3.25	모듈이름: initSYMTAB	22
3.25.1	기능	22
3.25.2	사용변수	22
3.26	모듈이름: writeSYMTAB	22
3.26.1	기능	22
3.26.2	사용변수	22
3.27	모듈이름: genErrorcode	22
3.27.1	기능	22
3.27.2	사용변수	23
3.28	모듈이름: Tokenize	23
3.28.1	기능	23
3.28.2	사용변수	23
3.29	모듈이름: indiCat	23
3.29.1	기능	23
3.29.2	사용변수	23
3.30	모듈이름: isReg	23
3.30.1	기능	23
3.30.2	사용변수	23
3.31	모듈이름: addLIT	25
3.31.1	기능	25
3.31.2	사용변수	25
3.32	모듈이름: addSYM	25
3.32.1	기능	25
3.32.2	사용변수	25
3.33	모듈이름: checkMN	25
3.33.1	기능	25
3.33.2	사용변수	25
3.34	모듈이름: ExpToVal	25
3.34.1	기능	25
3.34.2	사용변수	25
3.35	모듈이름: writeLstLn	26

---

3.35.1	기능	26
3.35.2	사용변수	26
3.36	모듈이름: dAssemble	26
3.36.1	기능	26
3.36.2	사용변수	26
3.37	모듈이름: ObjToken	26
3.37.1	기능	26
3.37.2	사용변수	27
3.38	모듈이름: writeDltLn	27
3.38.1	기능	27
3.38.2	사용변수	27
3.39	모듈이름: showBP()	27
3.39.1	기능	27
3.40	모듈이름: addBP(int loc)	27
3.40.1	기능	27
3.40.2	사용변수	27
3.41	모듈이름: delBP()	27
3.41.1	기능	27
3.41.2	사용변수	27
3.42	모듈이름: initBP()	28
3.42.1	기능	28
3.43	모듈이름: addES(char* str, int loc)	28
3.43.1	기능	28
3.43.2	사용변수	28
3.44	모듈이름: initESTAB()	28
3.44.1	기능	28
3.44.2	사용변수	28
3.45	모듈이름: Break()	28
3.45.1	기능	28
3.45.2	사용변수	28
3.46	모듈이름: dumpReg	28
3.46.1	기능	28
3.47	모듈이름: int getRegval(int reg)	29
3.47.1	기능	29
3.48	모듈이름: Break()	29
3.48.1	기능	29
3.48.2	사용변수	29

3.49	모듈이름: setADRS()	29
3.49.1	기능	29
3.50	모듈이름: Load()	29
3.50.1	기능	29
3.50.2	사용변수	29
3.51	모듈이름: int Read(int TA, int n, int i, int len)	31
3.51.1	기능	31
3.52	모듈이름: int Write(int VAL, int TA, int len)	31
3.52.1	기능	31
3.53	모듈이름: Run()	32
3.53.1	기능	32
3.53.2	사용변수	32
4.	전역 변수 정의	32
4.1	매크로	32
4.2	기본	33
4.3	dir 관련	33
4.4	dump 관련	33
4.5	history 관련	33
4.6	hash 관련	34
4.7	Assemble 관련	34
4.8	Loader 관련	35
5.	코드	오류! 책갈피가 정의되어 있지 않습니다.

## 1. 프로그램 개요

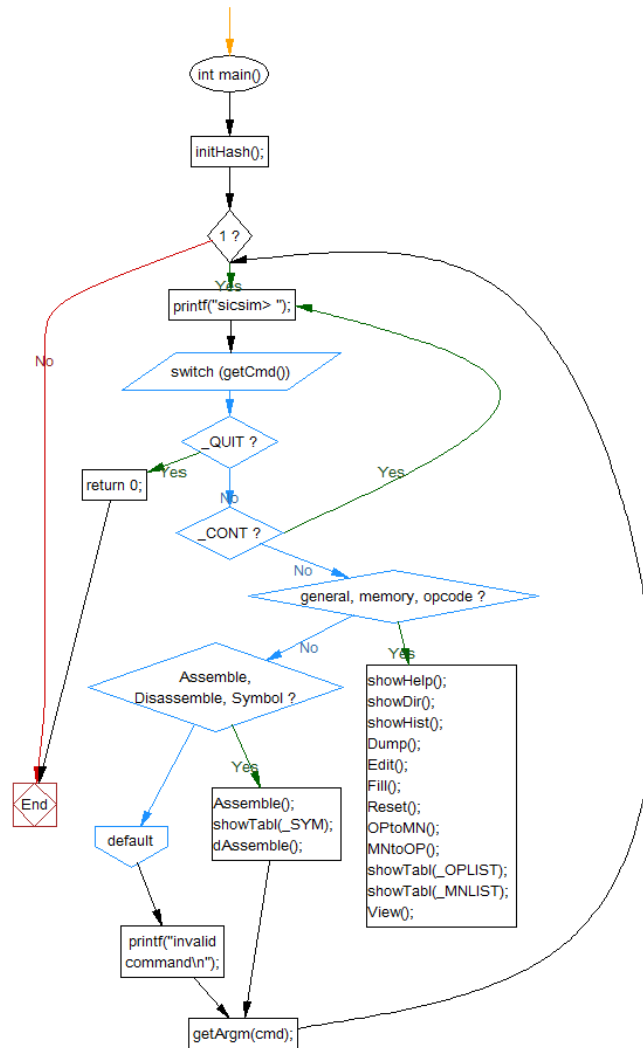
SIC/XE machine simulator로서, 셸(shell)을 구현한다. 컴파일을 통해서 만들어진 object 코드가 적재되고 실행될 메모리공간과 mnemonic(ADD, COMP, FLOAT, etc...)을 opcode 값으로 변환하는 OP CODE 테이블과 관련 명령어들(help, dir, quit, history, dump, edit, fill, reset, opcode, mnemonic, opcodelist, mnemoniclist)을 구현한다. 또한 SIC/XE machine의 assembly program source 파일을 입력 받아서 object 파일을 생성하고, SIC/S machine의 assembly object 파일을 다시 어셈블리 파일로 disassemble 하는 기능과, 어셈블리 과정 중 생성된 symbol table과 결과물인 object 파일을 볼 수 있도록 구현하였다.

**assembler 구현 범위 : 1 단계(기본 SIC/XE assemble) + 2 단계(Literal, EQU, Expression) + 추가구현(ORG)**

여기에 object file을 link시켜 메모리에 load하는 linking loader, 주소 지정 명령어(progaddr), 프로그램 실행 명령어(run), debug 명령어(bp, breakpoints)이 구현되었다.

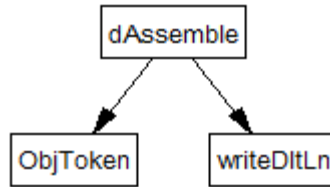
## 2. 프로그램 설명

### 2.1 프로그램 흐름도



Assemble()

- ▷ addLIT(char \* str, int loc)
- ▷ addSYM(char \* str, int loc)
- ▷ checkMN(char \* str)
- ▷ ExpToVal(char \* expr, int loc)
- ▷ genErrcode(int \* error)
- ▷ indiCat(char \* str, char \* indi, int flag)
- ▷ initSYMTAB()
- ▷ isReg(char \* str)
- ▷ StrToDec(char \* str)
- ▷ Tokenize(char \* str, char CODE[][TOKLEN])
- ▷ writeLstLn(FILE \* fp, int fmt, int LINE, int LOCCTR, char CODE[][TOKLEN], char \* objcode)
- ▷ writeSYMTAB()



```

  ▶ ⚙️ dumpReg()
  ▶ ⚙️ getRegval(int reg)
  ▶ ⚙️ hashFunc_OP(int op)
  ▶ ⚙️ printf(const char * _Format, ...)
  ▶ ⚙️ Read(int TA, int n, int i, int len)
  ▶ ⚙️ Write(int VAL, int TA, int len)
  ▶ ⚙️ Run()

```

```

  ▶ ⚙️ addBP(int loc)
  ▶ ⚙️ addq()
  ▶ ⚙️ delBP()
  ▶ ⚙️ HexToDec(char * hexArr)
  ▶ ⚙️ showBP()
  ▶ ⚙️ Break()

```

```

  ▶ ⚙️ addES(char * str, int loc)
  ▶ ⚙️ DecToHex(int len, int dec)
  ▶ ⚙️ HexToDec(char * hexArr)
  ▶ ⚙️ initESTAB()
  ▶ ⚙️ ObjToken(char * src, int s_col, int e_col, char * code)
  ▶ ⚙️ Reset()
  ▶ ⚙️ Load()

```

### 3. 모듈 정의

#### 3.1 모듈 이름 : main

##### 3.1.1 기능

SIC Simulator 의 기본 틀이다. UI 로서 입력 프롬프트를 제공하며 사용자의 입력을 getArgm 함수를 통해 입력받도록 한 후 subroutine 들 (showHelp( ), showDir( ), showHist( ), Dump( ), Edit( ), Fill( ), Reset( ), OPtoMN( ), MNtoOP( ), showTabl( ))을 호출한다.

##### 3.1.2 사용 변수

없음



---

## 3.2 모듈 이름: addq

### 3.2.1 기능

유효한 명령어 입력을 저장(history)하는 linked queue 에 노드를 삽입한다.

### 3.2.2 사용 변수

node\* temp : linked queue 탐색을 위한 node 포인터  
(char\* \_str , node\* front , node \*rear)

## 3.3 모듈이름: HexToDec

### 3.3.1 기능

16 진수 문자열 배열을 받아 integer 형으로 변환한다.

### 3.3.2 사용변수

char\* hexArr : 16 진수 문자열 배열  
int dec : 결과값  
int exp : 16 진수 지수부

## 3.4 모듈이름: DecToHex

### 3.4.1 기능

10 진수를 포맷에 맞게 16 진수 문자열 배열로 변환한다.

### 3.4.2 사용변수

int len : 16 진수 문자열 배열의 길이(포맷)  
int dec : 변환할 값  
char\* hexArr : 변환된 16 진수 문자열배열 포인터  
int e : 16 진수 지수부  
int den : 분모  
int quot : 몫

## 3.5 모듈이름: hashFunc\_OP

### 3.5.1 기능

opcode 의 hash key 값을 구하는 해시 함수.  
 $key = (opcode+3)/13;$

### 3.5.2 사용변수

int op : opcode 값

---

int key : key 값

### 3.6 모듈이름: hashFunc\_MN

#### 3.6.1 기능

mnemonic 의 hash key 값을 구하는 해시 함수.

```
int key=0, i;  
for(i=0;i<strlen(mn);i++)  
    key += mn[i]+'L';  
key = key%20;
```

#### 3.6.2 사용변수

char\* mn : mnemonic 문자열  
int key : key 값

### 3.7 모듈이름: initHash

#### 3.7.1 기능

opcode.txt로부터 opcode, mnemonic, priority 값을 입력 받아 해시 함수를 통해 얻은 key 값으로 해시 테이블을 생성

#### 3.7.2 사용변수

FILE\* fp : opcode.txt 를 가리키는 파일포인터  
hash \*ptr : 해시테이블 탐색을 위한 임시 해시포인터  
hash \*mntemp : mnemonic table 에 삽입될 새 노드  
hash \*optemp : opcode table 에 삽입될 새 노드  
char \*input : opcode.txt 한 줄을 입력 받을 문자열포인터  
char \*hex, \*str, \*ord : input 으로부터 추출될 opcode, mnemonic, priority  
int key : 해시 키 값

### 3.8 모듈이름: showTabl

#### 3.8.1 기능

opcode/mnemonic table 을 출력하는 함수

#### 3.8.2 사용변수

int cmd : opcode table 을 출력할지 mnemonic table 을 출력할지 결정  
hash \*ptr : hash table 탐색을 위한 임시 hash 포인터  
(hash \*opTabl, \*mnTabl)

---

### 3.9 모듈이름: isHex

#### 3.9.1 기능

넘겨받은 문자열이 Hexadecimal 표현이 맞는지 검사한다.

#### 3.9.2 사용변수

char\* hexArr : 검사할 문자열  
int maxlen : 문자열이 가질 수 있는 최대 길이  
int flag : Hexadecimal expression 인 경우 TRUE(1), 그렇지 않으면 FALSE(0)

### 3.10 모듈이름: getCmd

#### 3.10.1 기능

사용자로부터 명령을 입력 받고 문자열 비교 함수 strcmp 를 이용해 수행할 명령을 결정한다.

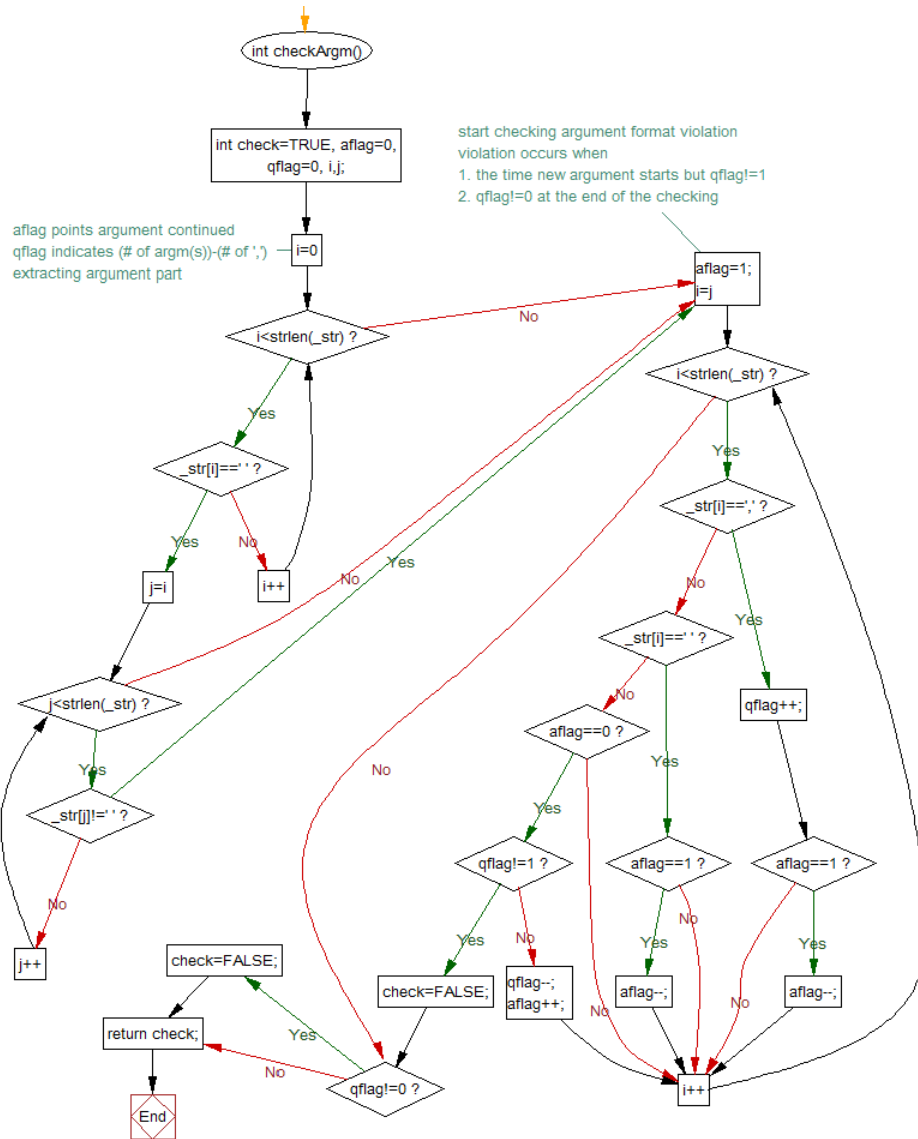
#### 3.10.2 사용변수

int cmd : 리턴 할 명령의 매크로 값을 가짐  
(char \*str , char \*\_str)

### 3.11 모듈이름: checkArgm

#### 3.11.1 기능

하나 이상의 argument 들을 사용하는 명령어들 (dump, edit, fill)의 argument 들이 제대로 입력이 되었는지(argument 와 쉼표와 공백문자의 사용이 올바른지) 검사한다.



### 3.11.2 사용변수

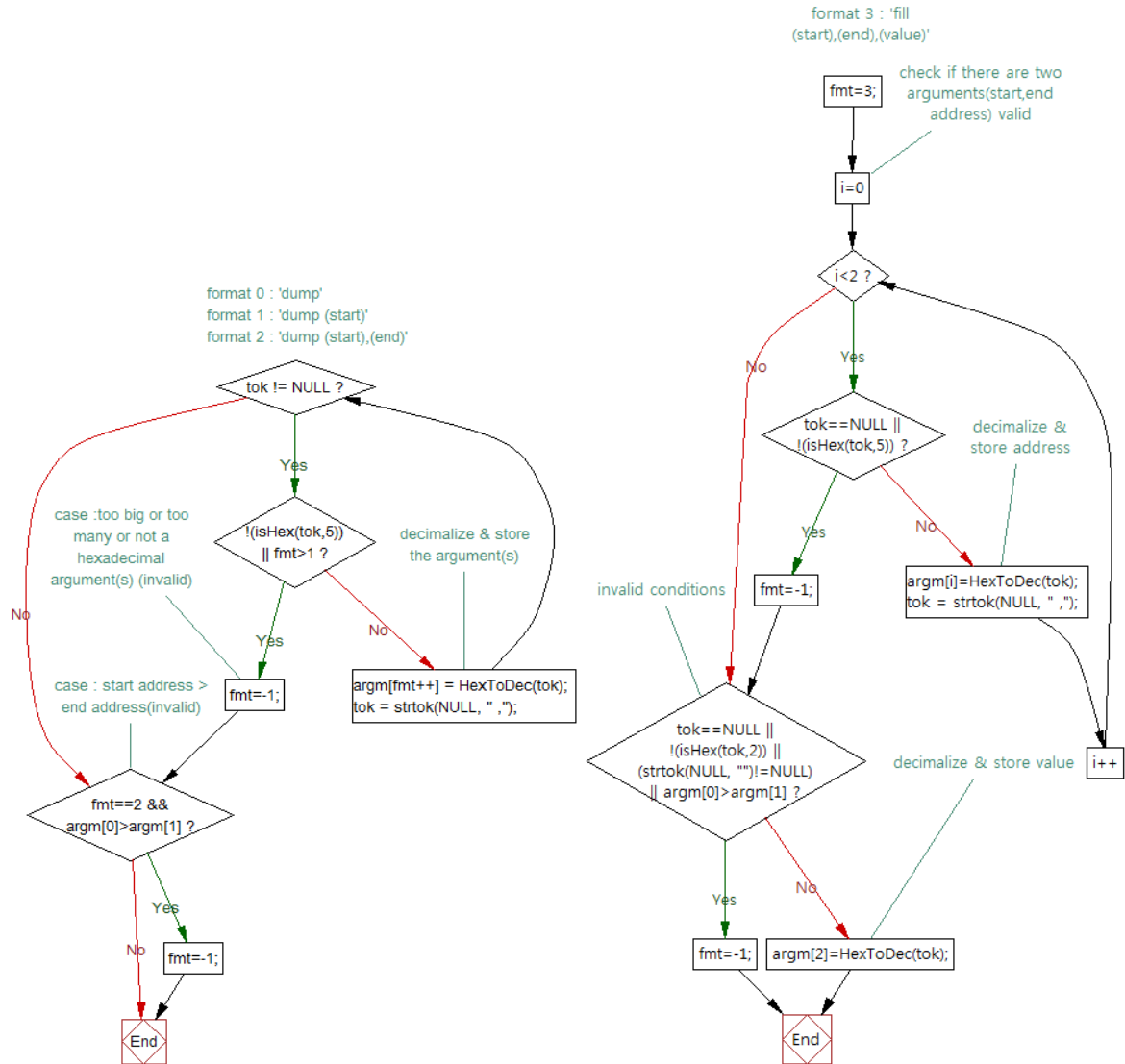
int check : 리턴값. 올바른 statement 일 경우 TRUE(1), 아닌 경우 FALSE(0)  
 int aflag : 유효문자(공백이나 침표가 아닌 문자)가 나타날 경우 1, 계속 유효문자가 이어지는 경우 2, 유효문자가 아닐 경우 0  
 int qflag : 침표가 나타나면 증가, 유효문자가 나타날 경우 감소  
 (char \*\_str)

## 3.12 모듈이름: getArgm

### 3.12.1 기능

명령의 종류와 포맷에 따른 passing argument 를 입력으로부터 추출, 가공하여 서브루틴에 넘겨준다.

순서도 : dump, fill



### 3.12.2 사용변수

int cmd : 명령의 종류 (dump, edit, fill, history, help, ...)  
int fmt : 명령의 형식 (argument 의 수에 따라 0,1,2,3 의 값을 가짐)  
(char \*tok) : 사용자 입력으로부터 추출된 argument 부분의 문자열 포인터  
(int argm[]) : 매개변수를 저장하는 배열

## 3.13 모듈이름: showHelp

### 3.13.1 기능

사용가능한 명령어들을 사용자에게 안내한다.

### 3.13.2 사용변수

없음

## 3.14 모듈이름: showDir

### 3.14.1 기능

현재 디렉터리 내의 모든 파일과 디렉터리(상위/현재 까지)를 출력한다. 디렉터리의 끝에는 '/'가, 실행파일의 끝에는 '\*' 표시가 붙는다.

### 3.14.2 사용변수

int cnt : 자동 줄바꿈 출력 포맷을 위한 카운트 변수  
(struct dirent \*entry) : 디렉터리 내의 파일/디렉터리 구조체 포인터  
(struct stat fileStat) : 파일/디렉터리의 속성을 저장하고 있는 구조체  
(DIR \*dir) : 디렉터리 포인터

## 3.15 모듈이름: showHist

### 3.15.1 기능

linked queue 에 저장된 유효 명령어들의 입력 legacy 를 출력한다.

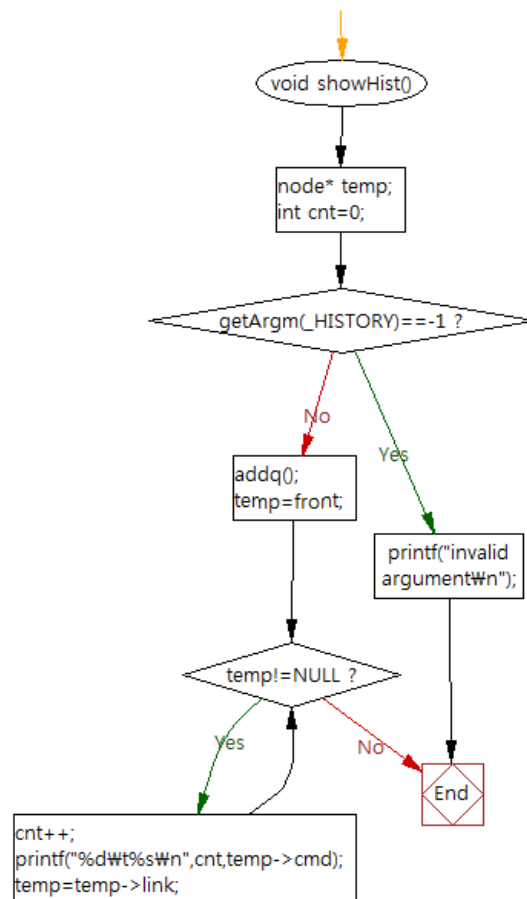
### 3.15.2 사용변수

node\* temp : linked queue 탐색에 필요한 임시 node 포인터  
int cnt : history 의 index 를 지정하기 위한 변수

## 3.16 모듈이름: dumpLine

### 3.16.1 기능

요청 받은 범위의 메모리에 저장된 값을 출력한다. 맨 왼쪽 칼럼에 메모리의 주소를, 가운데 칼럼엔 메모리에 저장된 실제 값을, 오른쪽 칼럼엔 값에 대응되는 ASCII 코드 문자를 출력한다.



### 3.16.2 사용변수

int s\_row : 시작 주소의 행

int e\_row : 끝 주소의 행

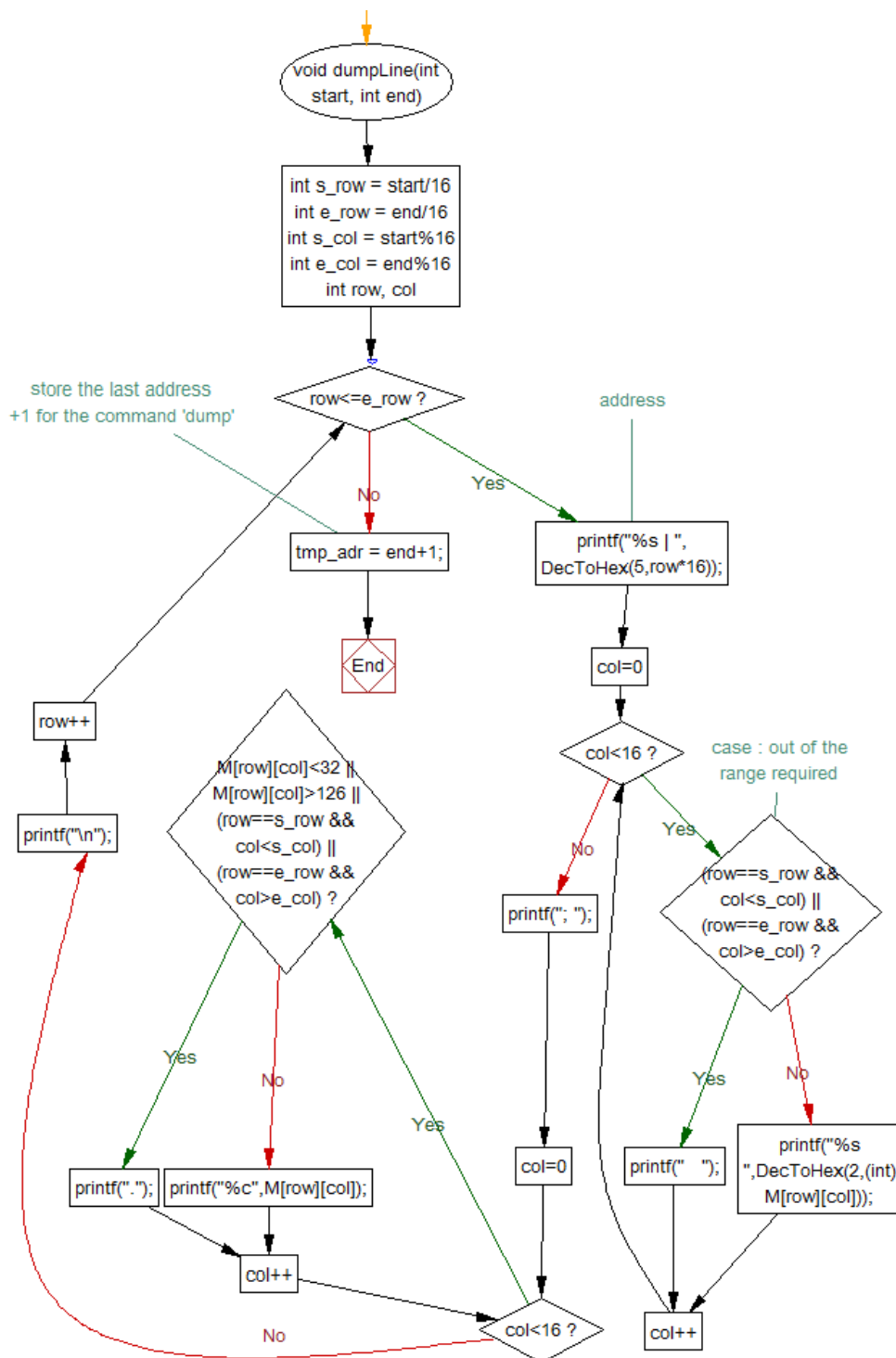
int s\_col :  
시작 주소의 열

int e\_col :  
끝 주소의 열

int row :  
행 탐색을 위한  
인덱스 변수

int col :  
열 탐색을 위한  
인덱스 변수

(int tmp\_adr) :  
'dump' 명령을  
위해 다음에  
출력을 시작할  
주소, 즉 마지막  
주소+1 을  
저장한다.



## 3.17 모듈이름: Dump

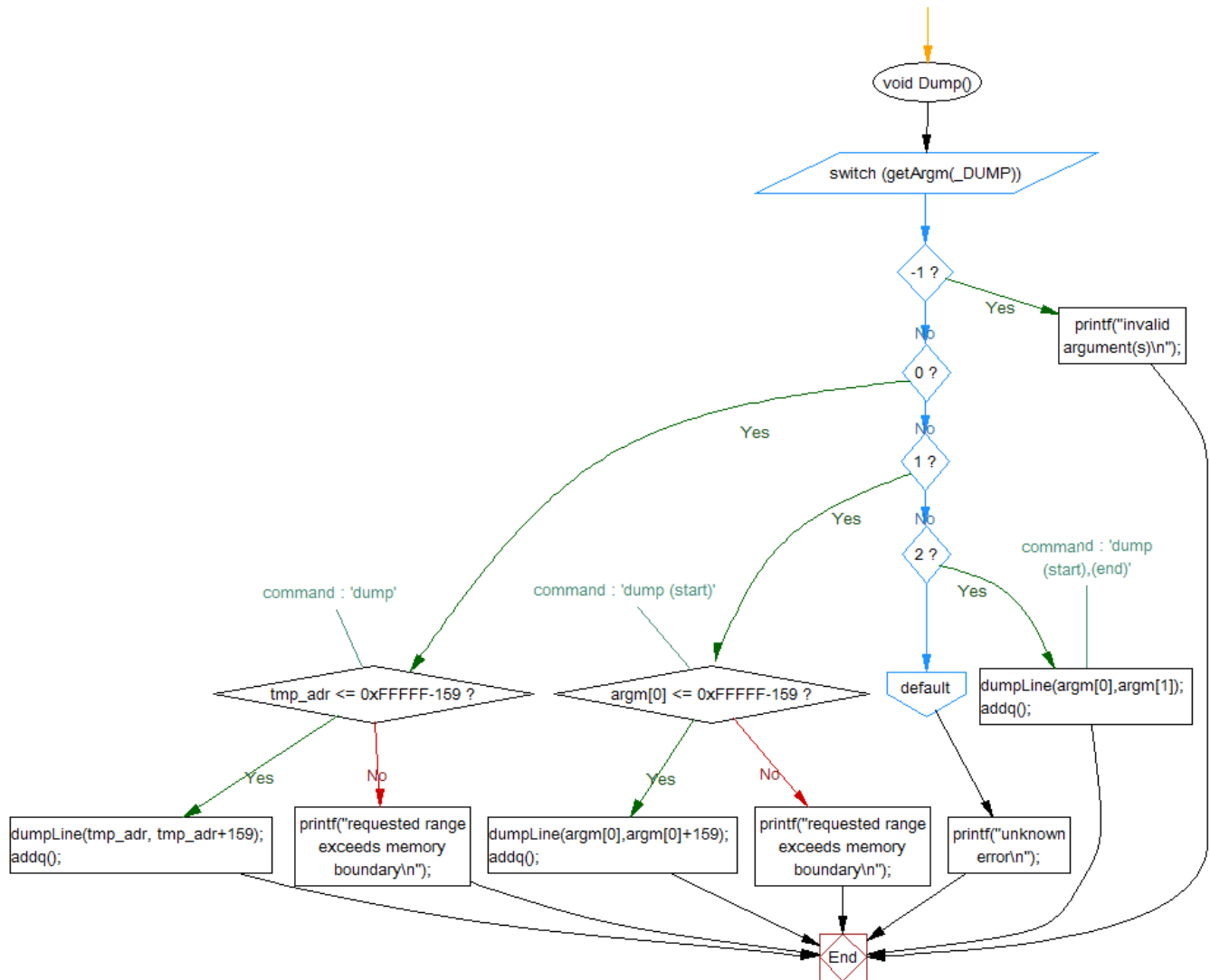
### 3.17.1 기능

getArgm 에서  
전처리된 argumen  
t 들을 명령 형식에  
따라 가공하여

dumpLine 으로 넘긴다.

### 3.17.2 사용변수

없음



## 3.18 모듈이름: Edit

### 3.18.1 기능

특정 주소의 값을 변경한다.



---

### 3.18.2 사용변수

없음

## 3.19 모듈이름: Fill

### 3.19.1 기능

특정 주소의 범위에 값을 변경한다.

### 3.19.2 사용변수

없음

## 3.20 모듈이름: Reset

### 3.20.1 기능

메모리의 모든 값을 0 으로 초기화한다.

### 3.20.2 사용변수

없음

## 3.21 모듈이름: OPtoMN

### 3.21.1 기능

opcode 를 입력 받아 hash table 에서 대응되는 mnemonic 을 찾아 출력한다. 입력받은 opcode 의 key 값을 해시함수를 이용해 구하여 opcode table 의 인덱스에 찾아 들어가 link 를 타고 넘어가며 찾는다.

### 3.21.2 사용변수

hash \*ptr : opcode table 탐색을 위한 임시 hash 포인터

hash \*opTable : opcode table

int key : 해시 키 값

int dec : opcode 값

int flag : 무효한 argument 처리를 위한 flag

## 3.22 모듈이름: MNtoOP

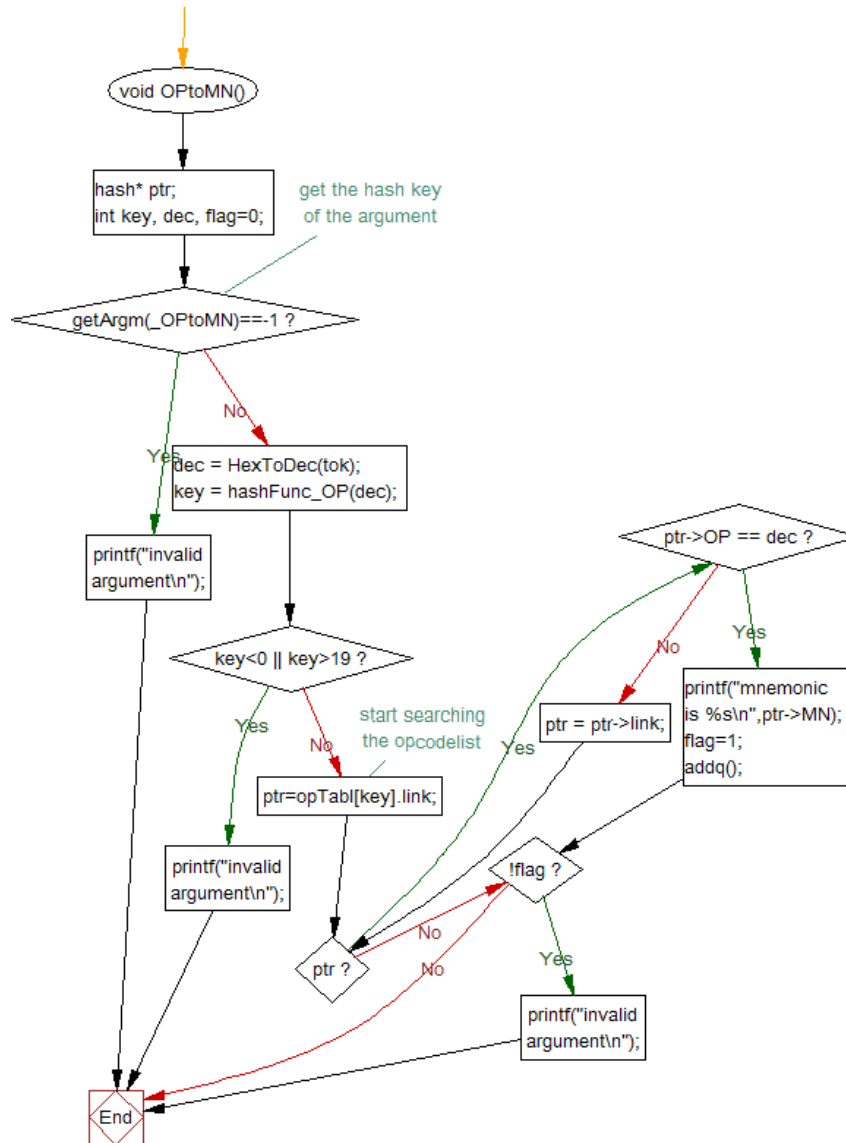
### 3.22.1 기능

mnemonic 을 입력 받아 hash table 에서 대응되는 opcode 의 key 값을 해시함수를 이용해 구하여 mnemonic table 의 인덱스에 찾아 들어가 link 를 타고 넘어가며 찾는다.

### 3.22.2 사용변수

hash \*ptr : mnemonic table 탐색을 위한 임시 hash 포인터

hash \*mnTable : mnemonic table  
 int key : 해시 키 값  
 int flag : 무효한 argument 처리를 위한 flag



### 3.23 모듈이름: View

#### 3.23.1 기능

디렉터리 내의 원하는 텍스트 파일을 로드 하여 화면에 출력한다. UNIX Shell 의 cat 명령어 기본옵션 명령과 같다.

#### 3.23.2 사용변수

FILE \*fp : 텍스트파일 포인터

---

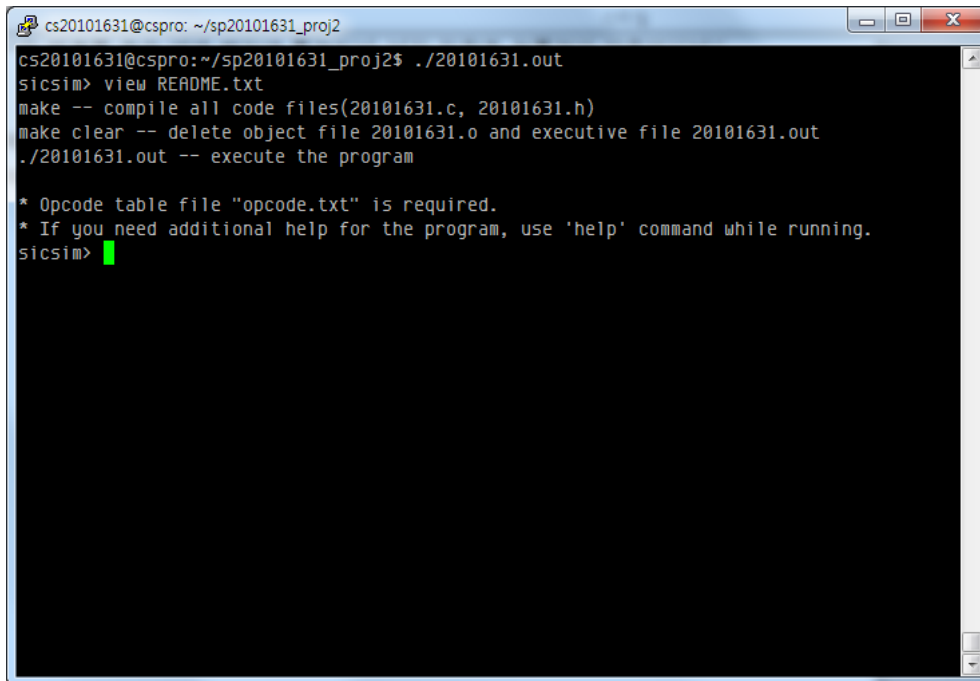
char line[TOKEN] : 텍스트파일을 라인 단위로 입력 받을 때 저장되는 문자열 변수

(시험 화면)

view README.txt

view opcode.txt

view 20101631.c



```
cs20101631@cspro: ~/sp20101631_proj2
cs20101631@cspro:~/sp20101631_proj2$ ./20101631.out
sicsim> view README.txt
make -- compile all code files(20101631.c, 20101631.h)
make clear -- delete object file 20101631.o and executive file 20101631.out
./20101631.out -- execute the program

* Opcode table file "opcode.txt" is required.
* If you need additional help for the program, use 'help' command while running.
sicsim> 
```

```
cs20101631@cspro: ~/sp20101631_proj2
sicsim> view opcode.txt
18 ADD 3/4
58 ADDF 3/4
90 ADDR 2
40 AND 3/4
B4 CLEAR 2
28 COMP 3/4
88 COMPF 3/4
A0 COMPR 2
24 DIV 3/4
64 DIVF 3/4
9C DIVR 2
C4 FIX 1
C0 FLOAT 1
F4 HIO 1
3C J 3/4
30 JEQ 3/4
34 JGT 3/4
38 JLT 3/4
48 JSUB 3/4
00 LDA 3/4
68 LDB 3/4
50 LDCH 3/4
70 LDF 3/4
08 LDL 3/4
6C LDS 3/4
74 LDT 3/4
04 LDX 3/4
D0 LPS 3/4
20 MUL 3/4
60 MULF 3/4
98 MULR 2
C8 NORM 1
44 OR 3/4
D8 RD 3/4
AC RMO 2
4C RSUB 3/4
A4 SHIFTL 2
F0 SIO 1
EC SSK 3/4
0C STA 3/4
78 STB 3/4
54 STCH 3/4
80 STF 3/4
D4 STI 3/4
14 STL 3/4
7C STS 3/4
E8 STSW 3/4
84 STT 3/4
10 STX 3/4
1C SUB 3/4
5C SUBF 3/4
94 SUBR 2
B0 SVC 2
E0 TD 3/4
F8 TIO 1
2C TIX 3/4
B8 TIXR 2
DC WD 3/4
sicsim>
```

```
cs20101631@cspro: ~/sp20101631_proj2
cs20101631@cspro:~/sp20101631_proj2$ !.
./20101631.out
sicsim> view 20101631.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <dirent.h>
#include <math.h>
#include "20101631.h"

int main(){

    initHash();
    while(1){
        printf("sicsim> ");
        switch(getCmd()){

            case _QUIT :
                return 0;
            case _CONT :
                break;
            case _HELP :
                showHelp();
                break;
            case _DIR :
                showDir();
                break;
            case _HISTORY :
                showHist();
                break;
            case _DUMP :
                Dump();
                break;
            case _EDIT :
                Edit();
                break;
            case _FILL :
                Fill();
                break;
            case _RESET:
                Reset();
                break;
            case _OPtoMN :
                OPtoMN();
                break;
            case _MNtoOP :
                MNtoOP();
                break;
            case _OPLIST :
                showTabl(_OPLIST);
                break;
            case _MNLIST :
                showTabl(_MNLIST);
                break;
            case _VIEW :
                View();
                break;
            default :
                printf("invalid command\n");
                fflush(stdin);

        }
    }
}
sicsim>
```

---

## 3.24 모듈이름: Assemble

### 3.24.1 기능

Filename 을 argument 로 받아 해당하는 SIC/XE machine assembly 소스 파일을 읽어서 object 파일과 listing 파일을 만든다. 소스파일에 에러가 존재할 경우 object 파일과 listing 파일을 생성하지 않고 에러내용을 화면에 출력한다.

2 pass assemble algorithm 을 구현하였다.

Relative addressing 의 구현은 pass2 에서 간단한 case 분류와 계산을 통해 구현할 수 있었고, LORG 와 ORG 는 pass1 에서 내부 loop 를 통해 구현하였다.

### 3.24.2 사용변수

FILE \*fpASM, \*fpINT, \*fpLST, \*fpOBJ; : .asm .lst .obj 파일 그리고 intermediate 파일의 읽기/쓰기를 위한 파일 포인터

SYMTAB \*ptr; : SYMTAB 탐색을 위한 포인터

hash \*\_ptr; : OPTAB 탐색을 위한 포인터

LITTAB \*lptr : LITTAB 탐색을 위한 포인터

char filename[TOKENLEN]; : filename 추출을 위한 string 저장공간

char lstFilename[TOKENLEN]; : listing file name

char objFilename[TOKENLEN]; : object file name

char stm[TOKENLEN], \_stm[TOKENLEN]; : .asm 파일의 입력 한 줄과 그 백업

char \*tmp; : string 처리를 위한 문자열 포인터

char CODE[4][TOKENLEN]; : 어셈블리 파일의 한 line 을 tokenize 하여 token 들을 label, opcode, operand 칼럼에 맞추어 저장하기 위한 문자열의 배열

int strtFlag=-1, i, fmt, key; : START 명령 처리를 위한 flag, statement 의 format(label, opcode, operand 의 유무), OPTAB 탐색을 위한 key 값

int litFlag : flag - operand 가 literal 인 경우 1

int ltoFlag : flag - literal pool 처리 중일 경우 1

int constFlag : flag - operand 가 상수인 경우 1

int orgFlag : flag - ORG 명령어 처리 중일 경우 1

int baseFlag : flag - Base relative addressing 을 사용하는 경우 1

int error[4]={FALSE}, ERR; : error table 과 에러코드

int \_LOCCTR=0, LOCCTR=0, LINE=0, DSP, LENGTH; : location counter 와 그 백업, line, displacement, length

int ORGLOC : ORG 명령어 이전의 location counter 값을 백업

int OPfmt : opcode 의 명령어 format (1,2,3,4)를 가리킴

int BASE : Base register

---

---

int objCnt=0;	: 다음 Text record 로 넘어가기 위한 조건을 따지기 위해 object code 의 길이를 저장
int newlnFlag=FALSE;	: RESB, RESW 명령어 처리 시 다음 Text record 로 넘어가기 위한 flag
char tmpObj[OBJMAX];	: object code 를 저장하는 임시 문자열
char *mRec	: Modification Record 가 catenation 되는 문자열
char* objcode;	: 하나의 object code 를 저장하는 문자열
int n, i, x, b, p, e	: instruction assemble 에 사용되는 n, i, x, b, p, e Flag

### 3.25 모듈이름: initSYMTAB

#### 3.25.1 기능

SYMTAB 과 LITTAB 을 초기화하거나 free 시킨다.

#### 3.25.2 사용변수

SYMTAB \*ptr, \*tmp : SYMTAB 탐색을 위한 포인터와 free 를 위해 현재 포인터를 저장하는 임시 포인터

LITTAB \*lptr, \*ltmp : LITTAB 탐색을 위한 포인터와 free 를 위해 현재 포인터를 저장하는 임시 포인터

### 3.26 모듈이름: writeSYMTAB

#### 3.26.1 기능

성공적으로 assemble 이 된 경우 SYMTAB 을 symtab.txt 에 저장한다. 이는 SYMTAB 출력 시 실패한 어셈블 후에도 가장 최근에 성공한 어셈블의 SYMTAB 을 출력하기 위함이다.

#### 3.26.2 사용변수

SYMTAB \*ptr : SYMTAB 탐색을 위한 포인터

### 3.27 모듈이름: genErrorcode

#### 3.27.1 기능

Error table 을 파라미터로 받아 자연수 에러코드를 생성한다. Duplicated symbol 인 경우 1 을 더하고, invalid opcode 인 경우 2, undefined symbol 인 경우 4, out of relative range 의 경우 8 을 더하는 방식이다. Decode 시 역순으로, 나뉘본 다음 몫이 1 일경우 해당 에러가 존재하는 것이고 나눈 값을 빼서 다음 에러를 검사하는 알고리즘이다.

---

### 3.27.2 사용변수

Int code : code+=error[**dupSYM**]+2\*error[**invOPC**]+4\*error[**undSYM**]+8\*error[**outRNG**]  
error 는 error table 로, TRUE/FASLE 의 값을 갖는 int 형 array 이다. 해당 에러에 define 된 매크로를 이용해 error table 의 index 에 접근한다.

## 3.28 모듈이름: Tokenize

### 3.28.1 기능

한 줄의 assembly 명령을 label, opcode, operand 로 분류하여 저장한다. 없을 경우 NULL 값을 저장한다. 문자열의 배열 CODE[4]에서, 각각 0,1,2,3 으로 define 된 매크로 LABEL, OPCODE, OPRND1, OPRND2 로 접근한다. (OPRND2 는 ' ,X ' 에 접근하기 위함). 명령의 format (int fmt) 를 리턴한다.

공백문자로 tokenize 한 문자열이 opcode 인지 확인한 후 맞을 경우 두번째 token 이 opcode 인지 확인하여 맞으면 에러처리로 넘어가고, 아닐 경우 label 을 비운다. 또한 OPRND2 부분이 "X"가 아닐 경우 operand 부분에 string 이 있고 그 안에 공백문자나 seperator 가 있는 경우이므로 백업된 명령 문자열을 C' 로 tokenize 하여 string 을 추출한다.

### 3.28.2 사용변수

int fmt : 명령의 format (label, opcode, operand, index addressing 여부를 나타냄)  
char \*tmp, \*\_tmp : string 처리를 위한 임시 문자열

## 3.29 모듈이름: indiCat

### 3.29.1 기능

Extended format 을 가리키는 '+'나 addressing 방식을 가리키는 '@', '#' indicator 를 문자열에서부터 빼거나, 다시 복원한다.

### 3.29.2 사용변수

Char \*str : 수정될 문자열  
Char\* indi : 더하거나 뺄 indicator 의 종류(+,@,#)  
Int flag : -1 을 넘겨받을 경우 문자열에서 빼고, +1 을 넘겨받을 경우 복원한다.

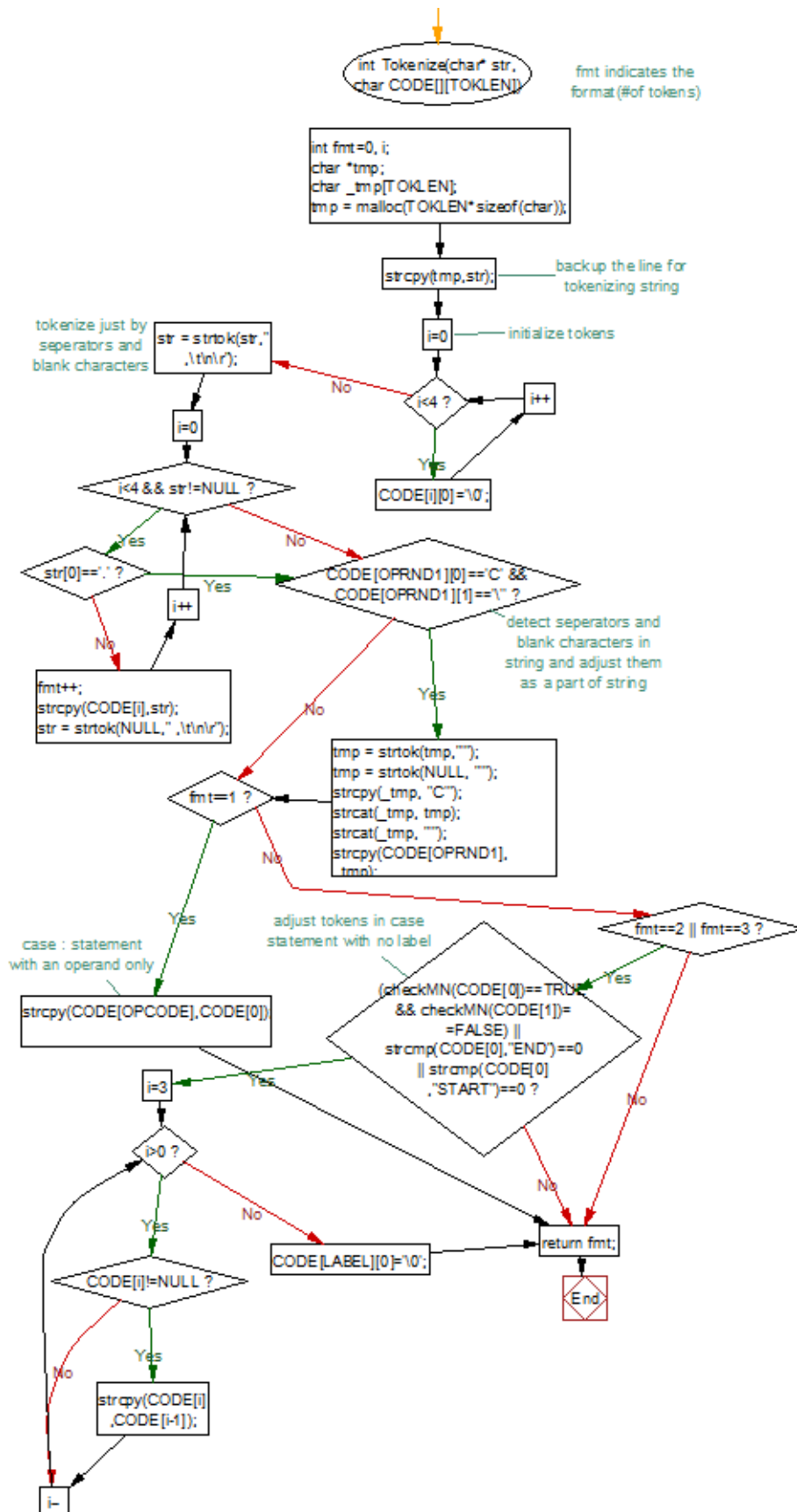
## 3.30 모듈이름: isReg

### 3.30.1 기능

레지스터 이름으로 추정되는 문자열을 파라미터로 받아서 레지스터일 경우 레지스터 번호를 , 아닐 경우 -1 을 반환

### 3.30.2 사용변수

Char\* str : 레지스터의 이름으로 추정되는 문자열





---

### 3.31 모듈이름: addLIT

#### 3.31.1 기능

pass 1 에서 발견되는 새로운 literal 들을 LITTAB 에 추가한다.

#### 3.31.2 사용변수

LITTAB \*New, \*ptr : 메모리가 할당될 new node, LITTAB 탐색을 위한 임시 포인터

### 3.32 모듈이름: addSYM

#### 3.32.1 기능

pass 1 에서 발견되는 새로운 symbol 들을 SYMTAB 에 추가한다.

#### 3.32.2 사용변수

SYMTAB \*New, \*ptr : 메모리가 할당될 new node, SYMTAB 탐색을 위한 임시 포인터

### 3.33 모듈이름: checkMN

#### 3.33.1 기능

넘겨받은 문자열이 opcode 인지 확인한다. 맞으면 TRUE(1) 아니면 FALSE(0)를 리턴한다.

#### 3.33.2 사용변수

hash \*ptr : OPTAB 탐색을 위한 임시 포인터

int key : hash key

int flag : 리턴값(TRUE or FALSE)

### 3.34 모듈이름: ExpToVal

#### 3.34.1 기능

Expression 문자열을 넘겨받아 값을 도출한다.

#### 3.34.2 사용변수

SYMTAB\* ptr : SYMTAB 탐색을 위한 임시 포인터

int constFlag : operand 가 상수인 경우 1 값을 갖는 flag

int val : 계산된 결과값

int sign : operand 에 곱해져서 더하기나 빼기 연산의 flag 로 쓰임

---

### 3.35 모듈이름: writeLstLn

#### 3.35.1 기능

토큰들의 배열 CODE 를 넘겨받아 출력 형식에 맞게 리스팅 파일에 한 줄을 출력한다.

#### 3.35.2 사용변수

int fmt , LINE, LOCCTR : fmt 가 0 인 경우 comment line 이므로 '.'을 출력. 라인과 location counter 값  
CODE[3][TOKEN] : 코드의 토큰들을 저장하고 있는 문자열 배열

### 3.36 모듈이름: dAssemble

#### 3.36.1 기능

object file 을 입력 받아 역 리스팅 파일을 만든다. object code 를 한번에 half byte 단위로 두 단위를 읽어 들여 opcode 일 경우 뒤에 4 half bytes 를 target address 로 하여 출력한다. opcode 가 아닐 경우 constant 이므로 2 half bytes 씩을 계속 읽어 3 회가 될 경우나 한 Text record 가 끝날 경우 출력을 한다.

#### 3.36.2 사용변수

FILE \*fpOBJ, \*fpDLT; : object file, disassemble listing file 포인터  
hash \*ptr; : OPTAB 탐색을 위한 포인터  
char filename[TOKEN] : file name  
char Recln[TOKEN]; : object code 에서 입력 받은 하나의 record line 을 저장  
char CODE[4][TOKEN]; : assemble 에서 쓰인 CODE 문자열 배열의 컨셉은 disassemble 프로시저에서 그 의미가 오버로딩 된다. 즉 인덱스 0,1,2,3 이 각각 location counter, opcode, target address, object code 로 매크로화 되어있다.  
char \*str, \*tmp, \*\_tmp; : 문자열 처리를 위한 문자열 포인터들  
int LOCCTR=0, col, i, len, key : location counter, column, index, record 의 length, opcode hash table 의 key 값  
int constFlag; : 상수 처리와 출력을 위한 flag

### 3.37 모듈이름: ObjToken

#### 3.37.1 기능

Text record 한 라인, start column, end column 을 파라미터로 넘겨받아 start col 부터 end col 까지의 코드를 추출한다.

---

### 3.37.2 사용변수

int i : column indexing 을 위한 변수  
char \*src : tokenize 될 타겟 string.  
int s\_col, e\_col : 자를 부분의 start column, end column  
char \*tmp : 추출된 코드가 저장될 임시 문자열

## 3.38 모듈이름: writeDltLn

### 3.38.1 기능

토큰들의 배열 CODE 를 넘겨받아 출력 형식에 맞게 역 리스팅 파일에 한 줄을 출력한다

### 3.38.2 사용변수

int i : CODE 문자열 배열 indexing 을 위한 변수  
FILE \*fp : .dlt 파일 포인터  
char CODE[4][TOKEN] : 토큰들의 배열(location counter, opcode, target address, object code)

## 3.39 모듈이름: showBP()

### 3.39.1 기능

모든 breakpoint 를 화면에 출력한다.

## 3.40 모듈이름: addBP(int loc)

### 3.40.1 기능

Breakpoint 를 추가한다.(linked list)

### 3.40.2 사용변수

int loc : break point 주소  
BP \*bptr, \*New : break point linked list 탐색을 위한 포인터와 새로 삽입될 포인터

## 3.41 모듈이름: delBP()

### 3.41.1 기능

Breakpoint 를 모두 제거한다.

### 3.41.2 사용변수

BP \*bptr, \*btmp : break point linked list 탐색을 위한 포인터와 memory free 를 위한 포인터

---

### 3.42 모듈이름: initBP()

#### 3.42.1 기능

Breakpoint 의 linked list 형태 자료구조 메모리를 할당, 초기화한다.

### 3.43 모듈이름: addES(char\* str, int loc)

#### 3.43.1 기능

D 레코드의 External Definition 이름과 주소를 External symbol table 에 추가한다.

#### 3.43.2 사용변수

ESTAB \*New, \*ptr : 새로 삽입될 ESTAB 포인터와 ESTAB linked list 탐색을 위한 포인터

### 3.44 모듈이름: initESTAB()

#### 3.44.1 기능

Linked list 형태 ESTAB 을 초기화한다.

#### 3.44.2 사용변수

ESTAB \*eptr, \*etmp : 초기화와 탐색에 필요한 ESTAB 포인터

### 3.45 모듈이름: Break()

#### 3.45.1 기능

Break point 관련 명령(출력, 추가, 초기화)을 수행하기 위한 전처리 함수

#### 3.45.2 사용변수

Int loc : break point 를 추가할 경우 사용자로부터 입력받은 argument 를 저장

### 3.46 모듈이름: dumpReg

#### 3.46.1 기능

레지스터들의 값을 화면에 dump 한다.

---

### 3.47 모듈이름: int getRegval(int reg)

#### 3.47.1 기능

레지스터 번호를 넘겨받아 그 레지스터에 저장된 값을 리턴한다.

### 3.48 모듈이름: Break()

#### 3.48.1 기능

Break point 관련 명령(출력, 추가, 초기화)을 수행하기 위한 전처리 함수

#### 3.48.2 사용변수

Int loc : break point 를 추가할 경우 사용자로부터 입력받은 argument 를 저장

### 3.49 모듈이름: setADRS()

#### 3.49.1 기능

사용자로부터 입력받은 프로그램이 로드 될 시작 주소를 set 한다.

### 3.50 모듈이름: Load()

#### 3.50.1 기능

Object file 들 또는 control section 들을 입력 받아 이를 link 하여 load 하는 linking loader 프로시저.

#### 3.50.2 사용변수

FILE \*fpObj, \*fpMap : object file 포인터와 loadmap 을 임시로 저장할 파일 포인터

ESTAB \*eptr : ESTAB 탐색을 위한 포인터

Char \*Rec : 한 줄의 record 를 입력받아 저장한다.

Char \*objcode, \*tmp : 동작에 필요한 object code 의 일부분이 tokenize 되어 저장될 문자열

Int \*REFLOC : reference number 를 index 로 갖는 EXTREF 의 주소값 배열

Int CSADDR, CSLTH, RECADDR : Control section address, Control section length, Starting address of the record

Int error[4], ERR : error flags

---

---

Control section	Symbol name	Address	Length
PROGA		4000	0063
	LISTA	4040	
	ENDA	4054	
PROGB		4063	007F
	LISTB	40C3	
	ENDB	40D3	
PROGC		40E2	0051
	LISTC	4112	
	ENDC	4124	

---

Pass 1:

```

begin
  get PROGADDR from operating system
  set CSADDR to PROGADDR {for first control section}
  while not end of input do
    begin
      read next input record {Header record for control section}
      set CSLTH to control section length
      search ESTAB for control section name
      if found then
        set error flag {duplicate external symbol}
      else
        enter control section name into ESTAB with value CSADDR
      while record type ≠ 'E' do
        begin
          read next input record
          if record type = 'D' then
            for each symbol in the record do
              begin
                search ESTAB for symbol name
                if found then
                  set error flag {duplicate external symbol}
                else
                  enter symbol into ESTAB with value
                    (CSADDR + indicated address)
                end {for}
              end {while ≠ 'E'}
            add CSLTH to CSADDR {starting address for next control section}
          end {while not EOF}
        end {Pass 1}

```

---

Pass 2:

```
begin
  set CSADDR to PROGADDR
  set EXECADDR to PROGADDR
  while not end of input do
    begin
      read next input record {Header record}
      set CSLTH to control section length
      while record type ≠ 'E' do
        begin
          read next input record
          if record type = 'T' then
            begin
              {if object code is in character form, convert
               into internal representation}
              move object code from record to location
                (CSADDR + specified address)
            end {if 'T'}
          else if record type = 'M' then
            begin
              search ESTAB for modifying symbol name
              if found then
                add or subtract symbol value at location
                  (CSADDR + specified address)
              else
                set error flag (undefined external symbol)
              end {if 'M'}
            end {while ≠ 'E'}
          if an address is specified {in End record} then
            set EXECADDR to (CSADDR + specified address)
            add CSLTH to CSADDR
          end {while not EOF}
        jump to location given by EXECADDR {to start execution of loaded program}
      end {Pass 2}
```

### 3.51 모듈이름: int Read(int TA, int n, int i, int len)

#### 3.51.1 기능

Simple / immediate / indirect addressing 에 따라 메모리를 read 하여 실제 operand 로 쓰일 값을 리턴한다.

### 3.52 모듈이름: int Write(int VAL, int TA, int len)

#### 3.52.1 기능

메모리에 값 VAL 를 write 한다.

---

### 3.53 모듈이름: Run()

#### 3.53.1 기능

메모리에 로드된 프로그램을 실행한다.

#### 3.53.2 사용변수

BP *bptr	: breakpoint linked list 탐색을 위한 포인터
Hash *ptr	: OPTAB 탐색을 위한 포인터
Int n, i, x, b, p, e	: addressing 방식과 target address 계산에 사용되는 flags
Int TA, OPC	: Target address 와, instruction 에서 추출된 opcode 값
Int key	: opcode 해시 값

## 4. 전역 변수 정의

### 4.1 매크로

```
#define TRUE 1
#define FALSE 0
#define TOKEN 80
#define _CONT -1
#define _QUIT 0
#define _HELP 1
#define _DIR 2
#define _HISTORY 3
#define _DUMP 4
#define _EDIT 5
#define _FILL 6
#define _RESET 7
#define _OPtoMN 8
#define _MNtoOP 9
#define _OPLIST 10
#define _MNLIST 11
#define _TYPE 13
#define _ASM 12
#define _SYM 14
#define _dASM 15
```

- Assemble 관련

```
#define LABEL 0
```



---

```
#define OPCODE 1
#define OPRND1 2
#define OPRND2 3
#define OBJMAX 60
- error code 관련
#define dupSYM 0
#define invOPC 1
#define undSYM 2
#define outRNG 3
- disassemble 관련
#define _LOC 0
#define _OPC 1
#define _TA 2
#define _OBJ 3
- loader 관련
#define REFMAX 256
```

## 4.2 기본

```
unsigned char M[65536][16]={0};    //SICSIM virtual memory space
char str[TOKENLEN];                //raw user input
char _str[TOKENLEN];               //backup of str
char *tok;                          //string tokens of command, indicating arguments
int argm[3];                       //store argument(s)
```

## 4.3 dir 관련

```
DIR *dir;                          // 디렉터리 포인터
struct dirent *entry;              // 디렉터리 내의 파일/디렉터리 구조체 포인터
struct stat fileStat;              // 파일/디렉터리의 속성을 저장하고 있는 구조체
```

## 4.4 dump 관련

```
int tmp_adr;                       // store next default start address of dump
```

## 4.5 history 관련

```
typedef struct _node{
    char cmd[TOKENLEN];             //사용자가 입력한 유효 명령을 저장
    struct _node* link;
```

---

```
}node;
```

```
node* front;
```

```
node* rear;
```

## 4.6 hash 관련

```
typedef struct _hash{
    char MN[5];           //mnemonic string
    int OP;               //opcode
    int key;              //hash key
    char order[3];        //order priority
    struct _hash* link;    //next hash node
}hash;
```

```
hash mnTabl[20];        //array of index nodes(mnemonic table)
```

```
hash opTabl[20];        //array of index nodes(opcode table)
```

## 4.7 Assemble 관련

```
typedef struct _symtab{
    char label[TOKENLEN];
    int loc;
    int flag;
    struct _symtab *link;
}SYMTAB;
```

```
typedef struct _littab{
    char name[TOKENLEN];
    char hex[TOKENLEN];
    int len;
    int loc;
    struct _littab *link;
}LITTAB;
```

```
LITTAB* _LITTAB;
```

```
SYMTAB* _SYMTAB;
```

```
int asmcnt    //어셈블을 한 횟수. 1 이상일 경우 어셈블 수행 전 SYMTAB 을 초기화시킨다.
```

---

## 4.8 Loader 관련

```
int PROGADDR
int EXECADDR
int loadcnt = 0      //로드를 수행한 횟수. ESTAB 초기화에 사용된다.
int A,X,L,PC,B,S,T,SW; //registers
//break point linked list
typedef struct _bp{
    int loc;
    struct _bp *link;
}BP;
BP *_BP;
```