과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

[Assignment 3]

**서강대학교 컴퓨터공학과**

**20161665**

**황세현**

목 차

1. 프로그램 개요
2. 프로그램 설명
   1. 프로그램 흐름도
3. 모듈 정의
   1. 모듈 이름 : main()
      1. 기능
      2. 추가된 변수
   2. 모듈 이름 : int loader\_pass1(int num,char \*file1, char \* file2, char \*file 3)
      1. 기능
      2. 추가된 변수
   3. 모듈 이름 : int loader\_pass2(char memory[][16],int num,char \*file1, char \* file2, char \*file 3)
      1. 기능
      2. 추가된 변수
   4. 모듈 이름 : void add\_node\_to\_estable(estab\_node\*,char \*,int address)
      1. 기능
      2. 추가된 변수
   5. 모듈 이름 : int find\_node\_in\_estab(char \*,int ,int ,int )
      1. 기능
      2. 추가된 변수
   6. 모듈 이름 : int get\_string\_len6(char \* origin, char \*tmp, int \* mark, int Hbyte,int type)
      1. 기능
      2. 추가된 변수
   7. 모듈 이름 : void delete\_ESTAB()
      1. 기능
      2. 추가된 변수
   8. 모듈 이름 : void twos\_complement(char \*)
      1. 기능
      2. 추가된 변수
   9. 모듈 이름 : int func\_bp(bp\_head\* bp,int a)
      1. 기능
      2. 추가된 변수
   10. 모듈 이름 : int delete\_bp\_list(bp\_head\* bp)
       1. 기능
       2. 추가된 변수
   11. 모듈 이름 : int func\_run(char \*str,int \*reg\_list)
       1. 기능
       2. 추가된 변수
   12. 모듈 이름 : void init\_reg\_list(int \*a)
       1. 기능
       2. 추가된 변수
   13. 모듈 이름 : int find\_opcode(int opcode, char \*)
       1. 기능
       2. 추가된 변수
4. 전역변수의 정의
   1. int load\_success
   2. int execute\_start\_addr
   3. int prog\_len[3]
   4. int RnodeNum[3]
   5. unsigned int progaddr
   6. int main\_program
5. 코드
   1. loader\_pass1
   2. loader\_pass2
   3. func\_run
6. 기타
   1. 추가구현(예외처리)

# 프로그램 개요

기존에 구현한 어셈블러에 이어서 어셈블러에 의해 작성된 여러 개의 오브젝트 코드를 link하여 가상의 메모리 상에 load하는 2pass linking loader을 구현한다. 또한 완성된 오브젝트 코드를 disassemble 하여 프로그램을 실행하는 과정 또한 구현해야 한다.

# 프로그램 설명

## 프로그램 흐름도

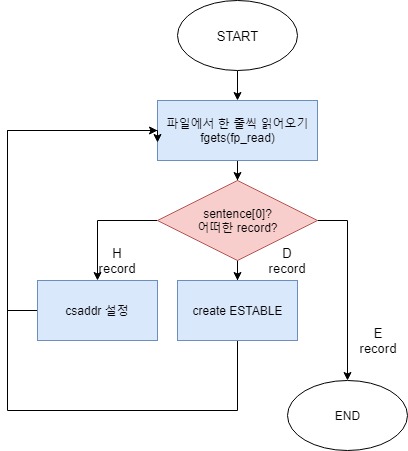


Figure 1 ) linking loader pass1

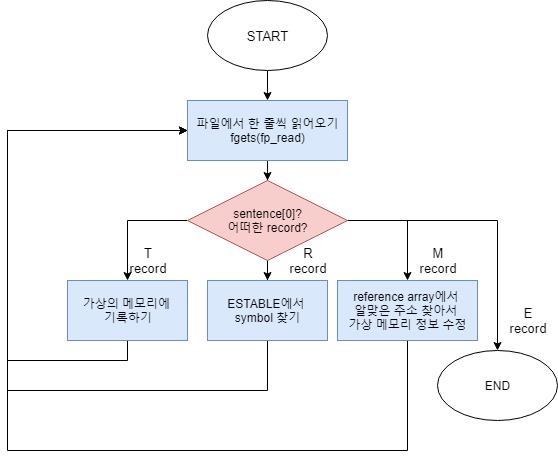


Figure 2) linking loader pass2

# 모듈 정의 [project3에 추가되거나 변경된 함수]

## 모듈 이름 : main()

### 기능

linking loader와 프로그램 실행에 대한 명령을 처리할 수 있도록 추가적인 명령어에 대한 처리를 실행하였다.

### 추가된 변수

* bp\_head breakpoint : breakpoint 저장을 담당하는 변수
* int tmpAddr, tmpLen : 프로그램에서 임시 시작주소와 코드 길이를 저장
* int restart : 파일을 처음부터 다시 읽어야 하는지를 나타내는 flag 변수
* int run1, run2 : run 기능에 이용되는 부가적인 연산을 진행하는 변수

## 모듈 이름 : loader\_pass1(int num,char \*file1,char \*file2, char\* file3)

### 기능

linking loader 기능을 구현하기 위해서는 2pass 알고리즘이 필요하다. loader\_pass1함수는 그 중 pass1 기능을 담당하며 H ,D record 에 관련한 기능을 처리한다. H record를 읽어서 프로그램 실행과정에서 쓰이게 될 csaddr와 cslen을 설정한다. 또한 D record를 읽어서 define 된 symbol와 progaddr에 상대적인 address를 담고있는 external reference table 을 생성한다.. ESTAB구조는 pass2의 modification record처리 시 유용하게 쓰인다. pass1에서 비정상적으로 종료될 경우 pass2가 실행되지 않는다.

### 추가된 변수

* FILE\* fp\_read : 인자로 입력받은 파일을 여는 포인터
* int csaddr : control section의 시작주소를 저장하는 변수
* char sentence[100] : file을 한 줄씩 읽어올 때 문자열 정보를 담을 변수
* estab\_node\* tmpNode : ESTBLE을 탐색할 때 필요한 임시 estab\_node형 포인터
* ERROR error : 에러코드에 관한 정보를 담을 변수

## 모듈 이름 : loader\_pass2(char memory[][16], int num,char \*file1,char \*file2, char\* file3)

### 기능

linking loader의 두번째 pass를 담당하는 함수이다. pass1에서 이미 처리된 H record와 D record를 제외하고 R record를 읽어서 reference number와 주소를 함꼐 저장하는 array를 생성하고 , T record의 정보를 가상의 메모리에 load하는 역할을 한다. 또한 M record를 읽어서 가상 메모리에 저장된 정보를 proaddr에 맞게 수정하는 기능또한 담당한다. 이 과정에서 pass1에서 생성된 ESTAB에 대한 탐색이 이뤄진다. 만약 프로그램 중간에 에러가 발생하여 비정상적으로 종료될 시에는 에러 메시지를 출력하고 동적으로 생성된 ESTAB을 free (메모리 해지)한다.

### 추가된 변수

* FILE\* fp\_read : 인자로 입력받은 파일을 여는 포인터
* char sentence[100] : .obj file code를 1줄 전체를 담는 변수
* int mark : 문장 분리 시 시작 위치를 기억할 변수
* int csaddr : control section 의 start address
* int row,col : 가상의 메모리에서 문자 접근 시 사용됨
* int plus : modification record에서 덧셈연산, 뺄셈연산을 구분하는 변수
* R\_node\* Rhead : reference number array에 대한 정보를 담는 자료구조
* ERROR error : 에러코드에 관한 정보를 담을 변수
* estab\_node\* tmpNode : estab 조사에 사용되는 노트 포인터

## 모듈 이름 : add\_node\_to\_estable(estab\_node \*,char \*,int address)

### 기능

pass1에서 생성되는 ESTAB에 노드를 추가해주는 함수. pass1에서 D record 기능 처리 중간에 자주 호출된다.

### 추가된 변수

* estab\_node \* newNode : estab\_node의 동적할당된 포인터를 담을 변수
* estab\_node\* tmp : 노드 생성에 부수적인 역할을 할 temp pointer.

## 모듈 이름 : find\_node\_in\_estab(char\*, int ,int ,int)

### 기능

estable에 symbol이 존재하는지 아닌지를 판단하여 그에 따른 값을 반환하는 함수. 만약 symbol이 존재한다면 -1을 return하고 존재하지 않는다면 1을 return 한다. 이 함수는 linking loader 기능 구현에 있어서 여러 번 호출되는데 필요에 따라 약간씩 다른 값을 return할 필요가 있어서 flag를 추가 인자로 설정하였다. flag=1이라는 D record, flag=2 라면 H record, 그리고 flag=3 이라면 R record 에 관한 처리를 담당한다.

### 추가된 변수

* estab\_node\* head : estable search 에 이용될 head 변수

## 모듈 이름 : get\_string\_len6(char \*origin,char \*tmp,int \*mark, int Hbyte, int type)

### 기능

.obj file에서 정해진 글자 수만큼 읽어오는 함수. 비정상적인 문자를 만났을 경우에(즉, 예상치 못한 종료를 하게될 경우) -1을 return, 정상종료일 때는 1을 return. 문장의 끝을 알릴 때는 0을 return한다. 또한 type==0일때는 숫자에 관한 처리를 , type==1일 때는 문자열에 관한 처리를 담당하는 함수이다.

### 추가된 변수

* int flag : 문자열 추출에 부수적인 도움을 주는 flag변수이다.
* int index : 다음 문자열 추출을 시작할 위치이다.

## 모듈 이름 : delete\_ESTAB()

### 기능

동적으로 할당된 external reference table을 삭제하는 기능을 담당한다. 메모리를 해지한다. (free)

### 추가된 변수

* estab\_node \* del, pre : estab\_node 삭제과정에 관여하는 변수

## 모듈 이름 : twos\_complement(char \*)

### 기능

6자리 signed로 표현된 16진수 문자열을 받아 보수 문자열 (음 🡪 양) 로 전환해주는 함수이다.

### 추가된 변수

* int len : 인자로 받은 문자열의 길이를 저장할 변수
* int i, res, tmp : 기타 연산에 부수적인 도움을 주는 변수

## 모듈 이름 : func\_bp(bp\_head\* bp,int a)

### 기능

breakpoint기능을 담당하는 함수. bp 추가, 출력, 삭제 등의 기능이 있다. 정상적으로 연산이 이뤄졌을 경우 1을 반환하고, 에러가 발생할 경우에는 -1을 return한다.

### 추가된 변수

* bp\_node\*tmp Node, \* newNode : node 생성 및 삭제 연산에 부수적인 역할을 하는 변수

## 모듈 이름 : delete\_bp\_list(bp\_head\* bp)

### 기능

동적으로 할당된 breakpoint 목록을 삭제해주는 함수. 만약 breakpoint list에 아무것도 없다면 경고 메시지를 띄우고 종료한다. 정상적으로 잘 지워졌다면 1을 반환한다.

### 추가된 변수

* bp\_node\* del : bp\_node 삭제 연산을 도울 변수

## 모듈 이름 : func\_run(char \*start,int \*reg\_list)

### 기능

object code를 한 줄씩 인자로 넘겨받아 해당하는 기능을 수행한 뒤,그 값을 인자로 받은 레지스터 배열에 저장하여 넘긴다. 레지스터 값에 대한 출력은 main 함수에서 담당한다.

### 추가된 변수

* int mark : 문장을 나누기 위해 사용되는 변수
* int format : mnemonic의 format을 저장하는 변수
* int ni,xbpe : format 3,4에서 ni와 xbpe값을 담고 있는 변수
* int reg1,reg2 : 몇 번째 레지스터인지에 대한 정보를 담고 있는 변수
* char opcode[5], mnemo[10], addr[10] : 각각 opcode, mnemonic, address에 대한 문자열

## 모듈 이름 : void init\_reg\_list(int \*a)

### 기능

인자로 전달받은 레지스터 목록들을 초기화한다.

### 추가된 변수

없음

## 모듈 이름 : int find\_opcode(int opcode,char \*)

### 기능

opcode를 받아서 hashTable에 존재하는 opcode인지를 확인한다. 만약 hashTable에 opcode가 존재한다면 opcode의 format을 반환하고 존재하지 않는다면 -1을 반환한다. 인자로 전달되는 char \*에는 opcode의 mnemonic이 문자열의 형태로 저장된다.

### 추가된 변수

* int format : opcode의 format을 저장하는 변수
* hashNode\* tmp : hashTable을 조사하기 위한 노드 포인터

# 전역 변수 정의

## int load\_success

load 명령어가 정상적으로 실행된 적이 있는지는 체크하는 bool형 변수

## int execute\_start\_addr

run 명령시 프로그램이 시작될 위치

## int prog\_len[3]

.obj file 각각의 길이를 기록

## int RnodeNum[3]

R record 때 extref symobl의 개수를 저장

## unsigned int progaddr;

linking loader에서 쓰일 변수(프로그램이 load될 위치를 설정)

## int main\_program

linking loader기능에서 몇 번째 입력 file이 main program인지를 파악

# 코드 (새로 추가된 함수 위주)

/\* linking loader의 pass1 기능을 담당하는 함수이다. symbol들을 조사하여

estable을 생성한다. symbol과 관련하여 에러 발생시 return -1, 정상종료시 return 1\*/

int loader\_pass1(int valid,char \*file1,char \*file2,char \*file3){

FILE \*fp\_read;

int csaddr=progaddr; //section의 start 주소를 담을 변수

int tmpStart,tmpLength,mark=0; //mark는 어디까지 읽었는지 기록하는 변수

char sentence[100];//.obj file code1줄 전체를 담는 변수

estab\_node \*tmpNode;

int sentence\_len=0;

int i,res,flag;

char tmp[30],tmp2[30];

ERROR error=notError; //에러코드를 담을 변수

/\*ESTAB initialization\*/

for(i=0;i<3;i++){

ESTAB[i].next=NULL;

ESTAB[i].address=-1; //아직 node의 개수가 없으므로

prog\_len[i]=0;

}

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

if(i==0){

/\* .obj 각각의 파일 열기 \*/

if(!(fp\_read=fopen(file1,"r"))){

printf("[ERROR] No files having such a name [%s]\n",file1);return -1;

}

}

else if(i==1){

if(!(fp\_read=fopen(file2,"r"))){

printf("[ERROR] No files having such a name [%s]\n",file2);return -1;

}

}

else{

if(!(fp\_read=fopen(file3,"r"))){

printf("[ERROR] No files having such a name [%s]\n",file3);return -1;

}

}

sentence[0]=0;//첫번째 문자 초기화

/\* 각 .obj file 별 H record 와 D record 읽고 처리 \*/

while(fgets(sentence,sizeof(sentence),fp\_read)){//.obj file에서 한줄씩 읽어옴

mark=1;//시작 인덱스 초기화

/\* 공백이 허용되지않는다다고 가정하고 .obj 파일 읽기\*/

if(strlen(sentence)>1)

sentence[strlen(sentence)-1]='\0';

sentence\_len=strlen(sentence);

switch (sentence[0]) {

case 'H' : //1. H record 처리

if(get\_string\_len6(sentence,tmp,&mark,6,1)){ //prog name

get\_string\_len6(sentence,tmp2,&mark,6,0); //prog start

if((tmpStart=string\_to\_number(tmp2,2))==-1){ //start주소 오류

error=boundary; break;

}

else{

if(find\_node\_in\_estab(tmp,tmpStart,i,2)==-1){//H 노드 삽입

error=estab; break;

}

}

get\_string\_len6(sentence,tmp,&mark,6,0); //prog length

if((tmpLength=string\_to\_number(tmp,2))==-1){

error=boundary; break;

}

else

prog\_len[i]=tmpLength; //length 저장

}

if(strlen(sentence)==mark);

else if((get\_string\_len6(sentence,tmp,&mark,strlen(sentence)-mark+1,0))==0);

else { error=tail;break;} //비정상 종료시

break;

case 'D' : /\* 문자열의 끝에 도달할때까지 (symbol,address)쌍 읽기 \*/

flag=0; //symbol,address 짝이 맞는지 검사하는 flag

while((res=get\_string\_len6(sentence,tmp,&mark,6,1))!=0 && res!=-1){

////////////////////////위에 조건 mark<=sentence\_len으로 변경해야할수도

// symbol address get

if((res=get\_string\_len6(sentence,tmp2,&mark,6,0))==0 || res==-1){

error=get; break;

} //symbol,address 짝이 맞지 않을 때 오류 발생(거의 없음)

if((tmpStart=string\_to\_number(tmp2,2))==-1){

error=boundary; break;

}

else{

if(find\_node\_in\_estab(tmp,tmpStart,i,1)==-1){

error=estab; break;

} //노드 삽입 or 에러발생

}

}

break;

case 'E' :

if(get\_string\_len6(sentence,tmp,&mark,6,0)==1){

if(string\_to\_number(tmp,2)) //E000000과 같이 주소가 있을때

main\_program=i; //main\_program으로 설

}

break;

case 'R' :

RnodeNum[i]=0;;

while(mark<=sentence\_len){

if((res=get\_string\_len6(sentence,tmp,&mark,2,0))==0 || res==-1){

error=get;break;

}

if((res=get\_string\_len6(sentence,tmp2,&mark,6,1))==0 || res==-1){

error=get;break;

}

RnodeNum[i]++;

}

RnodeNum[i]+=1;

break;

case 'T' :

case 'M' :

case '.' : break;

default :

error=obj;break;

}

}

/\*에러코드가 존재하면 메세지 띄우고 종료(break)\*/

if(error!=notError){

printf("[ERROR] ");

switch (error){

case get : printf("Invalid object code format\n"); break;

case tail : printf("Too many arguments in object code\n"); break;

case boundary : printf("Number boundary error\n"); break;

case estab : printf("Symbol already exists in ESTAB\n"); break;

case obj : printf("Invalid record in object code\n");

break;

default : break;

}

delete\_ESTAB(); //생성된 estab삭제

fclose(fp\_read); //종료하기전에 file 닫기

return -1;

}

fclose(fp\_read);

}

/\* 생성된 establ의 address값 조정해주기 \*/

if(main\_program==-1) main\_program=0; //main program이 없다면 그냥 0으로 설정

tmpNode=&ESTAB[main\_program];

tmpNode->address += csaddr;

tmpNode=tmpNode->next;

while(tmpNode!=NULL){

tmpNode->address += csaddr;

tmpNode=tmpNode->next;

}

csaddr += prog\_len[main\_program];

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

if(i==main\_program) continue;

tmpNode=&ESTAB[i];

tmpNode->address += csaddr;

tmpNode=tmpNode->next;

while(tmpNode!=NULL){

tmpNode->address += csaddr;

tmpNode=tmpNode->next;

}

csaddr += prog\_len[i];

}

return 1;

}

/\* loader pass2. obj file을 읽어서 가상메모리에 기록한다. pass1가 정상적으로

종료되었을 경우에만 pass2가 실행된다. 정상종료시 1반환, 에러발생시 -1반환 \*/

int loader\_pass2(char memory[][16],int valid,char\* file1,char \*file2,char \*file3){

FILE\* fp\_read;

char sentence[100]; //.obj file code 1줄 전체를 담는 변수

char tmp[30],tmp2[30];

int mark; //문장 분리에 쓰일 변수(start index 저장)

int csaddr,code\_length=0;

int address\_in\_memory=0; //메모리에서의 위치를 저장하는 변수

int i,j,tmpAddr,tmpLen,res;

int row,col,sentence\_len=0,add\_addr=0;//add\_addr은 M record 에서 더해질 주소값

int plus=True; //M record에서 덧셈연산 진

int total\_len=0;

R\_node\* Rhead=NULL; //'R' record를 통해 형성될 index table

ERROR error=notError; //에러 코드를 저장하는 enum형 변수

estab\_node\* tmpNode;

char\* haha;

/\* load될 메모리 공간을 초기화 \*/

tmpLen=prog\_len[0]+prog\_len[1]+prog\_len[2];

for(i=progaddr;i<progaddr+tmpLen;i++){

row=i/16;col=i%16;

memory[row][col]=-1;

}

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

/\* .obj 각각의 파일 열기 \*/

if(i==0)

fp\_read=fopen(file1,"r");

else if(i==1)

fp\_read=fopen(file2,"r");

else

fp\_read=fopen(file3,"r");

csaddr=ESTAB[i].address; //control section의 시작주소 불러오기

/\* .obj file 한 줄씩 읽어서 record별로 명령 수행 \*/

while(fgets(sentence,sizeof(sentence),fp\_read)){

mark=1;

if(strlen(sentence)>1)

sentence[strlen(sentence)-1]='\0';

sentence\_len=strlen(sentence); //문자열 길이 저장

switch(sentence[0]){

case 'H' : break; //pass1에서 처리완료

case 'D' : break; //pass1에서 처리완료

case '.' : break; //해줄거 없음

case 'T' :

get\_string\_len6(sentence,tmp,&mark,6,0); //주소가져오기

if((tmpAddr=string\_to\_number(tmp,2))==-1){

error=boundary; break;

}

get\_string\_len6(sentence,tmp,&mark,2,0); //code len 가져오기

if((tmpLen=string\_to\_number(tmp,0))==-1){

error=boundary; break;

}

address\_in\_memory=csaddr+tmpAddr; //메모리상의 주소값

row=address\_in\_memory/16;col=address\_in\_memory%16;

/\* 가상 메모리에 obj code를 load \*/

for(j=0;j<tmpLen;j++,mark+=2){

tmp2[0]=sentence[mark];tmp2[1]=sentence[mark+1];tmp2[2]='\0';

res=string\_to\_number(tmp2,0);

memory[row][col]=res;

address\_in\_memory++;

row=address\_in\_memory/16;col=address\_in\_memory%16;

}

/\* .obj code 검사 \*/ //뒤에 뭔가 남아있다면 error

if(get\_string\_len6(sentence,tmp,&mark,10,0)==-1){

error=tail; break;

}

break;

case 'R' : //index와 matching 되는 linkedlist 를 생성

// while((res=get\_string\_len6(sentence,tmp,&mark,2,0))!=0 && res!=-1){

/\* R record node저장을 위한 배열 동적할당 \*/

Rhead=(R\_node\*)malloc(sizeof(R\_node)\*RnodeNum[i]);

if((tmpAddr=find\_node\_in\_estab(ESTAB[i].symbol,0,0,3))==-1){

error=record;break;

}

Rhead[0].addr=tmpAddr;Rhead[0].index=0;

/\* R record 문장 계속 읽어들이기 \*/

while(mark<=sentence\_len){//////////////////////////조건문수정해야될수도

if((res=get\_string\_len6(sentence,tmp,&mark,2,0))==0 || res==-1){

error=get;break;

} //index get

if((res=get\_string\_len6(sentence,tmp2,&mark,6,1))==0 || res==-1){ //symbol get (문장을 읽던 중 끝에 도달하거나 비정상종료시에는 에러임)

error=get; break;

}

res=string\_to\_number(tmp,0); //index숫자로 변환

if(res>RnodeNum[i]){

error=idx;break;

}

/\* 받아온 인덱스와 문자열 정보를 갖고 배열 생성 \*/

tmpAddr=find\_node\_in\_estab(tmp2,0,0,3);

if(tmpAddr==-1){

error=record;break; //찾는 symbol이 없다면 error

}

else{ //reference array를 조사하는데 이미 존재하는 extref일때

for(j=0;j<res-1;j++){

if(Rhead[j].addr==tmpAddr){

error=already;break;

}

}

}

Rhead[res-1].addr=tmpAddr;Rhead[res-1].index=res-1; //index에 맞는 정보 주소와 함께 저장

}

break;

case 'M' :

plus=True; //plus 변수값 초기화

get\_string\_len6(sentence,tmp,&mark,6,0); //주소가져오기

if((tmpAddr=string\_to\_number(tmp,2))==-1){

error=boundary;break;

}

get\_string\_len6(sentence,tmp,&mark,2,0); //바꿔야할 len 가져오기

if((tmpLen=string\_to\_number(tmp,0))==-1){

error=boundary;break;

}

if(sentence[mark]=='-') plus=False; //modification + or -

tmp[0]=sentence[mark+1];tmp[1]=sentence[mark+2];tmp[2]='\0';

res=string\_to\_number(tmp,0); //reference index

if(res>RnodeNum[i] || res<0){ //유효하지 않은 index 에러

error=idx;break;

}

add\_addr=Rhead[res-1].addr; //reference number 참조해서 주소받기

address\_in\_memory=csaddr+tmpAddr; //메모리상의 주소값

tmp2[0]='\0'; //tmp2문자열 초기화(tmp2=바뀔 메모리 주소가 문자열형태로)

for(j=0;j<3;j++){ //가상메모리로부터 3byte 받아오기

row=address\_in\_memory/16;col=address\_in\_memory%16;

sprintf(tmp,"%02X",(unsigned char)memory[row][col]);

strcat(tmp2,tmp);

address\_in\_memory++;

}

//받아온 문자열 (3byte)는 6개의 char로 tmp2에 저장되어있음

if((tmpLen==5 && tmp2[1]=='F') || (tmpLen==6 && tmp2[0]=='F')) {//보수문자열로 전환

twos\_complement(tmp2);

sscanf(tmp2,"%X",&res);//문자열을 숫자로

res=(res\*-1)-1;

}

else

sscanf(tmp2,"%06X",&res);//문자열을 숫자로

if(plus)

res += add\_addr; //address modification(주소륿 보정해줌);

else

res -= add\_addr;

sprintf(tmp,"%06X",res); //update된 정수값을 16진수 문자열로 변경하기

if(strlen(tmp)==8) //만약 문자열로 바꿨는데 길이가 8이라면(음수라면)

{ memcpy(tmp,tmp+2,6);tmp[6]='\0'; }

//다시 가상의 메모리 공간에 저장

address\_in\_memory=csaddr+tmpAddr;

for(j=0;j<3;j++){

row=address\_in\_memory/16;col=address\_in\_memory%16;

tmp2[0]=tmp[j\*2];tmp2[1]=tmp[j\*2+1];tmp2[2]='\0';

res=string\_to\_number(tmp2,0);

memory[row][col]=res;

address\_in\_memory++;

}

break;

case 'E' : //program execution start addr 설정

if(sentence\_len>1){

get\_string\_len6(sentence,tmp,&mark,6,0);

if((tmpAddr=string\_to\_number(tmp,2))==-1){

error=boundary;break;

}

execute\_start\_addr=tmpAddr+progaddr;

}

break;

default :

error=obj;

break;

}

}

if(error!=notError){ //////////////////////pass1의 에러코드 위치 수정

printf("[ERROR] ");

switch(error){

case get : printf("Invalid object code format\n"); break;

case tail : printf("Too many arguments in object code\n"); break;

case boundary : printf("Number boundary error\n"); break;

case estab : printf("Symbol already exists in ESTABLE\n"); break;

case record : printf("No such a symbol in ESTABLE\n");break;

case obj : printf("Invalid record in object code\n"); break;

case already : printf("Already existing EXTREF\n");break;

case idx : printf("Invalid index in R record\n");break;

default : break;

}

/\*비정상 종료시 fclose, free,delete\_ESTAB & 메모리 초기화\*/

delete\_ESTAB();free(Rhead);fclose(fp\_read);

init\_memory(memory);prog\_length=0;

return -1;

}

/\* R record를 통해 동적할당 된 R head free 해주기 \*/

fclose(fp\_read); //만약 중간에 에러가 발생해서 종료할 경우 fclose꼭 해주기`

}

////////////////////////////////////////////////////////////////////////

/\* 파일에 에러가 없다면 ESTAB 출력. main program이 먼저 출력되도록! \*/

printf("\n");

printf("control symbol address length\n");

printf("section name \n");

printf("--------------------------------------------------------\n");

//main program 출력

tmpNode=&ESTAB[main\_program];

printf("%s %04X %04X\n",tmpNode->symbol,tmpNode->address,prog\_len[main\_program]);

tmpNode=tmpNode->next;

while(tmpNode!=NULL){

printf(" %s %04X\n",tmpNode->symbol,tmpNode->address);

tmpNode=tmpNode->next;

}

total\_len += prog\_len[main\_program];

//sub routine 출력

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

if(i==main\_program) continue;

tmpNode=&ESTAB[i];

printf("%s %04X %04X\n",tmpNode->symbol,tmpNode->address,prog\_len[i]);

tmpNode=tmpNode->next;

while(tmpNode!=NULL){

printf(" %s %04X\n",tmpNode->symbol,tmpNode->address);

tmpNode=tmpNode->next;

}

total\_len+=prog\_len[i];

}

printf("--------------------------------------------------------\n");

printf(" total length %04X\n",total\_len);

////////////////////////////////////////////////////////////////////////

prog\_length=total\_len;

delete\_ESTAB();

free(Rhead);

return 1;

}

/\*start주소와 end 주소에 해당하는 만큼의 코드를 메모리 공간에서 읽어서

동작을 수행시킨다. \*/

int func\_run(char \*objcode,int \*reg\_list){

int mark=0; //문장 끊어읽는 시작위치 기록

int len=strlen(objcode);

int res,format;

int ni=0,xbpe=0;

int reg1,reg2,flag;

char opcode[5],mnemo[10],addr[10],mode;

Register r;

ERROR error=notError;

for(mark=0;mark<len;mark+=(format\*2)){

opcode[0]=objcode[mark];opcode[1]=objcode[mark+1];opcode[2]='\0';//opcode받기

sscanf(opcode,"%X",&res); //opcode가 hexa형태로 변형된후 저장

ni=res%4; //1,2 번째 bit (opcode+ni)

if(ni==0 || !strcmp(opcode,"FF\0")){

format=3;continue;

}

opcode[0]=objcode[mark+2];opcode[1]='\0';

sscanf(opcode,"%X",&xbpe); //3번째 bit(xbpe)

if((format=find\_opcode(res-ni,mnemo))==-1){

error=mnemonic; break;

}

printf(" [mnemonic : %s]\n",mnemo);

if(format==3 && xbpe==1) //extended format(4)

format=4;

reg\_list[8] += format; //PC register 갱신

/\* format별로 처리해주기 (target Address 설정)\*/

if(format==3 || format==4){

if(format==3){

memcpy(addr,objcode+mark+3,3);

addr[3]='\0';

}

else if(format==4){

memcpy(addr,objcode+mark+3,5);

addr[5]='\0';

}

sscanf(addr,"%X",&res); //res=target address(보정 전)

if(xbpe==2)

res += reg\_list[8]; //PC relative mode(보정 후)

else if(xbpe==4)

res += reg\_list[3]; //BASE relative mode(보정 후)

else if(xbpe==10)

res += (reg\_list[8]+reg\_list[1]); //index+PC relative

else if(xbpe==12)

res += (reg\_list[3]+reg\_list[1]); //index+Base relative

}

else if(format==2){

reg1=objcode[mark+2]-'0';

reg2=objcode[mark+3]-'0';

}

/\* 명령어 별로 기능 처리하기 \*/

mode=mnemo[0];

if(mode=='L'){ //1. Load 명령어

flag=True;

switch(mnemo[2]){ //LDA,LDB,LDCH,LDL 등등

case 'A' : r=A; break;

case 'B' : r=B; break;

case 'C' : r=A; break;

case 'L' : r=L; break;

case 'S' : r=S; break;

case 'T' : r=T; break;

case 'X' : r=X; break;

default : flag=False; break;

}

if(flag)

reg\_list[r]=res; //register에 address load

}

else if(mode=='T'){ //2. TIX, TIXR 명령어

r=X;

reg\_list[r]++;

}

else if(mode=='R'){ //3. RMO, RSUB 명령어

switch(mnemo[1]){

case 'M' : reg\_list[reg2]=reg\_list[reg1]; break;

case 'S' : r=L;reg\_list[8]=reg\_list[r]; break; //PC <- L

}

}

else if(mode=='O'){ //4. OR 명렁어

r=A;

reg\_list[r] = (reg\_list[r] | res);

}

else if(!strcmp(mnemo,"CLEAR")) //5. CLEAR 명령어

reg\_list[reg1]=0;

else if(mode=='J'){ //6. J,JEQ,JGT,JSUB 명령어

if(mnemo[1]=='S'){ //JSUB의 경우

r=L;

reg\_list[r]=reg\_list[8]; //L<-PC

reg\_list[8]=res; //PC<-memory

}

else //J,JEQ,JGT의 경우

reg\_list[8]=res;

}

else if(mode=='A'){ //7.ADD, ADDR명령어

if(strlen(mnemo)==4 && mnemo[3]=='R'){ //ADDR

reg\_list[reg2] += reg\_list[reg1];

}

else if(strlen(mnemo)==3){ //ADD,AND

if(mnemo[1]=='D'){

r=A;

reg\_list[r] += res;

}

else{

r=A;

reg\_list[r]= reg\_list[r] & res;

}

}

}

else if(mode=='D'){ //8.DIV , DIVR

if(strlen(mnemo)==3){

if(res!=0){

r=A;

reg\_list[r] /= res;

}

}

else if(strlen(mnemo)==4){

if(mnemo[3]=='R'){

reg\_list[reg2] /= reg\_list[reg1];

}

}

}

else if(mode=='M'){ //9.MUL,MULR

if(strlen(mnemo)==3){

r=A;

reg\_list[r] \*= res;

}

else if(strlen(mnemo)==4){

if(mnemo[3]=='R'){

reg\_list[reg2] \*= reg\_list[reg1];

}

}

}

else if(mode=='S'){ //10.SUB,SUBR

if(strlen(mnemo)==3){

r=A;

reg\_list[r] -= res;

}

else if(strlen(mnemo)==4){

if(mnemo[3]=='R'){

reg\_list[reg2] -= reg\_list[reg1];

}

}

}

//11.나머지 명령어는 처리X

/\* 명령어 기능처리 종료 \*/

}

/\*에러코드 정리\*/

if(error!=notError){

printf("[ERROR] ");

switch (error){

case obj : printf("Invalid object code\n"); break;

case mnemonic : printf("No such a mnemonic\n");break;

default : break;

}

return -1;

}

/\* register 값 갱신해주기 \*/

return 1;

}

void init\_reg\_list(int \*a){

int i;

for(i=0;i<10;i++)

a[i]=0;

}

# 기타

6.1. 추가구현

* linking loader ) main program이 첫번째 입력으로 들어오지 않았을 때

ex) loader progc.obj progb.obj proga.obj 🡪 main program을 메모리에 첫번째 순서로 load 나머지는 입력된 순서대로 . 즉, A-C-B순서대로 메모리상에 loade된다.

ex) loader progb.obj progc.obj proga.obj 🡪 A-B-C 순서대로 메모리 상에 load된다.

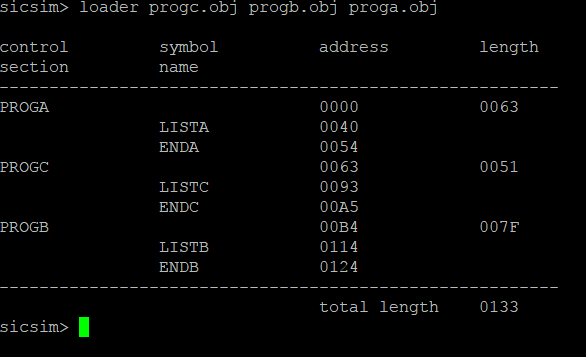


Figure 3 loader progc.obj progb.obj proga.obj

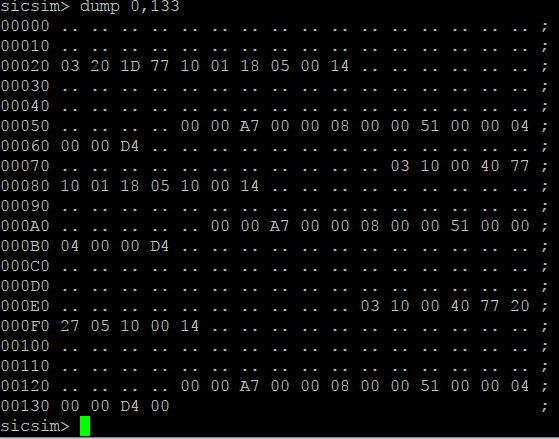


Figure 4 가상 메모리에 load된 정보

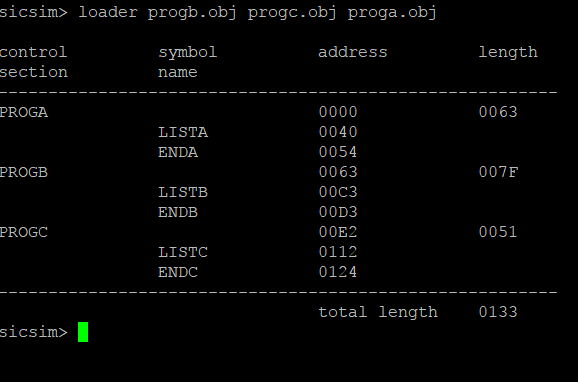


Figure 5 loader progb.obj progc.obj proga.obj