과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 3>>

**서강대학교 컴퓨터학과**

**[20151616]**

**[최승환]**

목 차

1. 프로그램 개요 3

2. 프로그램 설명 3

2.1 프로그램 흐름도 3

3. 모듈 정의 3

3.1 모듈 이름 : int Loader(char\* input) 3

3.1.1 기능 3

3.1.2 사용 변수 3

3.2 모듈 이름: void LoadMem(char\* fname) 3

3.2.1 기능 3

3.2.2 사용 변수 3

3.3 모듈이름: void ModMem(char\* fname) 4

3.3.1 기능 4

3.3.2 사용변수 4

3.4 모듈이름: void Run() 4

3.4.1 기능 4

3.4.2 사용변수 4

4. 전역 변수 정의 4

4.1 LSYM 4

4.2 LSYM\* symboltable 4

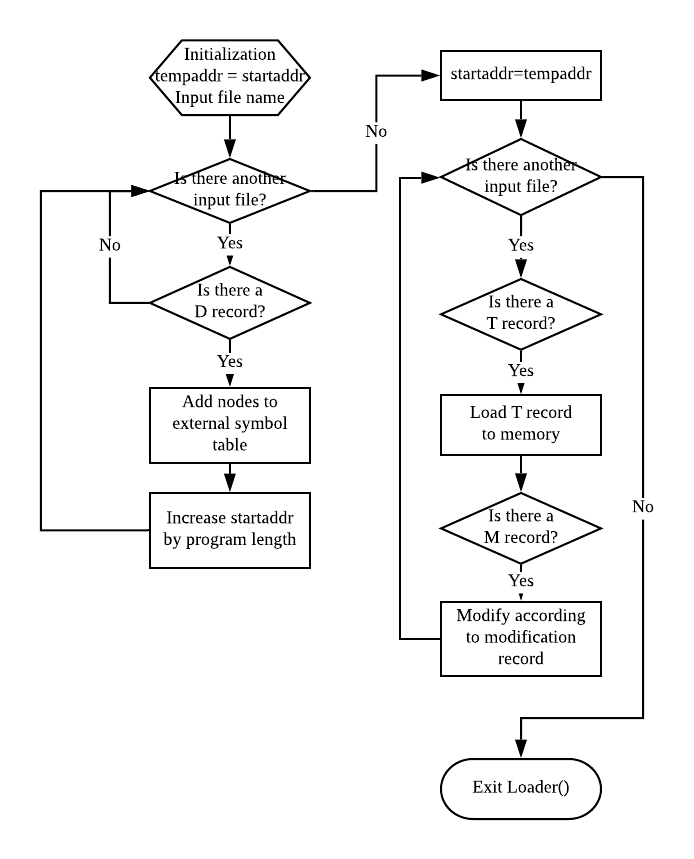
4.3 BP\* bphead, int startaddr, int totallength, int regarray[10] 4

# 프로그램 개요

LoadMem(),ModMem() 을 포함한 Loader() 와 SetBp() 를 통해 저장한 breakpoint 에 따라 디버깅을 할 수 있는 Run() 함수를 메인으로 하는 Linking Loader 및 디버깅이 가능한 프로그램이다. Loader 는 입력되는 프로그램을 읽어 External Symboltable 을 Linked list형태로 저장하며, 이를 참조해 Loader 가 메모리에 Load 한다. Run은 메모리에 저장되어 있는 값을 읽어 프로그램을 실행시키고, 저장되어 있는 breakpoint가 있으면 해당 라인에서 멈춘다. 멈추거나 프로그램이 종료되면 그 때마다 모든 레지스터의 값을 출력한다.

# 프로그램 설명

## 프로그램 흐름도



**그림 1> Loader() 프로그램 흐름도**

# 모듈 정의

## 모듈 이름 : int Loader(char\* input)

### 기능

입력한 한 줄의 input 을 받아 그 중에 있는 파일명들을 골라낸다. 파일명과 일치하는 파일들을 읽어내, Extsymtab() 을 실행해주고, Extsymtab() 으로 링크드 리스트가 완성된 뒤에는 각 파일을 다시 읽어 Load와 Modification 을 실행해준다.

사용 변수

* + - 1. char fname[MAX], int index, int tempaddr, int maxindex

fname 은 현재 읽고 있는 파일명을 저장하는 행렬이다.

index 는 입력받은 input 의 몇번째 인덱스를 참조하고 있는지 저장해 둔 변수이다. 이를 이용해 어떤 부분이 파일명인지 판별해낼 수 있다.

tempaddr 는 startaddr 의 초기값을 저장하기 위한 변수이다. external symboltable 을 linked list로 저장할 때, startaddr의 값이 바뀌기 때문에 Pass2 를 실행하기 전에 시작주솟값을 초기화하기 위함이다.

maxindex는 input 의 길이를 저장하고 있는 변수이다. 이를 이용해 input이 끝났는지를 확인할 수 있다.

코드부

int Loader(char\* input)

{

char fname[MAX];

int index=7;

int tempaddr=0;

int maxindex=strlen(input)-1;

tempaddr=startaddr;

while(index<maxindex)

{

if(input[index] == ' ')

{

++index;

continue;

}

sscanf(input+index,"%s",fname);

index+=strlen(fname);

//function to open specific file and read length from first line

if(!Extsymtab(fname))

{

return 0;

}

//

++index;

}

index=7;

startaddr = tempaddr;

while(index<maxindex)

{

if(input[index] == ' ')

{

++index;

continue;

}

sscanf(input+index,"%s",fname);

index+=strlen(fname);

//function to open specific file and read length from first line

LoadMem(fname);

//loading it on the memory

ModMem(fname);

//

++index;

}//completed making external symbol table

return 1;

}

## 모듈 이름: void LoadMem(char\* fname)

### 기능

프로그램의 시작주소를 저장하는 paddress를 이용해, 해당하는 메모리 주소로부터 T record 에 기록되어 있는 값을 저장시킨다. 이전에 작성했던 EditMem() 을 사용했다.

### 사용 변수

* + - 1. FILE\* fp, char buf[MAX], char pname[MAX]

fp 는 입력받은 프로그램명에 해당하는 파일을 읽어 그 파일 포인터를 저장하는 변수이다. buf 는 오브젝트 파일을 한 줄씩 읽은 값을 저장하는 행렬이다. pname 은 Csect 의 이름을 저장하고 있는 변수이다. 이를 이용해 실제로 저장할 주소를 정할 수 있다.

* + - 1. unsigned int index, maxindex, paddress, taddress

index 는 buf의 몇번째 index를 참조하고 있는지를 저장하는 인덱스이다. maxindex는 T record 의 길이를 저장하는 변수이다. paddress는 참조하고 있는 Csect의 주소를 저장하고 있는 변수이다. taddress는 각 T record 의 시작주소를 저장한다.

void LoadMem(char\* fname)

//fname contains the name of the obj file ex) "proga.obj"

{

FILE\* fp;

char buf[MAX];

char pname[MAX];

unsigned int index=0, maxindex=0, paddress=0, taddress=0;

int hb=0;

memset(buf,0,sizeof(char)\*MAX);

memset(pname,0,sizeof(char)\*MAX);

fp = fopen(fname,"r");

while(fgets(buf,MAX,fp))

{

index=0;

if((buf[0] == '.'))

{

continue;

}

else if(buf[0] == 'H')

{

sscanf(buf+1,"%s",pname);

paddress = GetCsect(pname);

}

else if(buf[0] == 'E')

{

break;

}

else if(buf[0] == 'T')

{

sscanf(buf+1,"%06X%02X",&taddress,&maxindex);

index+=4;

while(sscanf(buf+1+(index\*2),"%02X",&hb) != -1)

{

EditMem(paddress+taddress+index-4,hb);

index++;

}

AddTrec(taddress+paddress, maxindex);

}

}

return;

}

## 모듈이름: void ModMem(char\* fname)

### 기능

### Modification record 를 참조해 메모리에 Load된 값들을 시작 주소나, External symboltable 에 맞게 바꿔준다. 해당 M record 가 4형식인지 아닌지를 구분해, 각각 다른 종류의 처리를 해줘야 한다.

### 사용변수

* + - 1. char buf[MAX], pname[MAX], refname[10]

buf는 파일에서 읽은 한 줄씩의 내용을 저장하는 행렬이다. pname은 프로그램의 이름을 저장하는 행렬이다. refname 은 M record에서 External reference 의 이름을 쓸 때, 이를 저장하는 변수이다.

* + - 1. unsigned int index, modlength, paddress, taddress, sindex, disp

index는 buf의 참조하는 인덱스를 저장한다. modlength 는 M record 에서 몇 글자를 바꾸고 싶은지를 저장하는 변수이다. 4형식의 경우 이 변수에 5가 저장되므로, 이를 이용해, 4형식을 구별해낸다. paddress는 프로그램의 시작주소이다. taddress는 바꾸려고 하는 T record의 주소를 저장하고 있다. disp 는 바꾸고 싶은 주소값을 저장하고 있다. 입력이 모두 들어온 후에는 더하거나 빼야 하는 값을 disp 와 합쳐 계산한 후 그 값을 다시 메모리에 1Byte 씩 나눠 저장한다.

void ModMem(char\* fname)

{

FILE\* fp;

char buf[MAX];

char pname[MAX];

char refname[10];

unsigned int index=0, modlength=0, paddress=0, taddress=0, sindex=1, disp=0;

int hb=0;

memset(buf,0,sizeof(char)\*MAX);

memset(pname,0,sizeof(char)\*MAX);

memset(refname,0,sizeof(char)\*10);

fp = fopen(fname,"r");

while(fgets(buf,MAX,fp))

{

index=0;

if((buf[0] == '.'))

{

continue;

}

else if(buf[0] == 'H')

{

sscanf(buf+1,"%s",pname);

paddress = GetCsect(pname);

AddLsymtab(sindex,paddress);

//00 local symtable node becomes the head node

}

else if(buf[0] == 'D')

{

while(sscanf(buf+1+(index\*12),"%s %06X",refname,&taddress)!=-1)

{

index++;

AddDsymtab(refname,taddress);

//move to link index-1 times to add local symtable.

}

}

else if(buf[0] == 'R')

{

index=0;

//when input was came in from alphabet first

if(buf[1] == '0')

{

while(sscanf(buf+1+(index\*8),"%02X%s",&sindex,refname)!=-1)

{

index++;

AddLsymtab(index+1,GetEsymtab(refname));

//move to link index-1 times to add local symtable.

}

}

else

{

while(sscanf(buf+1+(index\*6),"%s",refname)!=-1)

{

index++;

AddLsymtab(index+1,GetEsymtab(refname));

}

}

}

else if(buf[0] == 'M')

{

sscanf(buf+1,"%06X%02X",&taddress,&modlength);

taddress += GetCsect(pname);

if(modlength == 5)

//format4 modification

{

disp = (mem[taddress]%16)\*65536+mem[taddress+1]\*256+mem[taddress+2];

}

else

//format3 modification

{

disp = mem[taddress]\*65536+mem[taddress+1]\*256+mem[taddress+2];

}

if(sscanf(buf+10,"%02X",&sindex) == 0)

{

continue;

}

if(buf[9] == '+')

{

disp += GetLsymtab(sindex);

}

else

{

disp -= GetLsymtab(sindex);

}

if(modlength == 5)

{

mem[taddress]/16\*16;

mem[taddress] += disp%1048576/65536;

mem[taddress+1] = disp%65536/256;

mem[taddress+2] = disp%256;

}

else

{

mem[taddress] = disp%16777216/65536;

mem[taddress+1] = disp%65536/256;

mem[taddress+2] = disp%256;

}

}

}

return;

}

## 모듈이름: void Run()

### 기능

메모리에 Load 된 값들을 이용해 프로그램을 실행시킨다. 실행을 시키며, 변하는 레지스터의 값을 행렬을 이용해 저장하고, 저장된 breakpoint 가 있으면, 이에 일치하는 주소마다 멈추고, 모든 레지스터의 값을 출력한다. 프로그램이 종료된 때에도 마찬가지이다..

### 사용변수

### unsigned int curraddr, endaddr, bpaddr, dispaddr curraddr 는 현재 참조하고 있는 메모리의 인덱스를 저장하고 있다. PC 레지스터의 값과 혼동할 수 있지만, 다르다. PC 레지스터는 instruction 이 완료되기 전에 이미 다음 주소를 가리키고 있기 때문에, curraddr는 과거의 PC 값이라고 할 수도 있다. endaddr 는 프로그램이 종료되는 주소를 저장한다. bpaddr 는 breakpoint 가 저장되어 있는 주소이다. dispaddr 는 disp를 가지고 있는 주소이다. J JEQ 등 해당 주소로 점프할 때 disp 를 사용하면, 다른 주소로 점프할 수 있기 때문에 dispaddr를 사용한다.

* + - 1. int format, opcode, niflag, xbpeflag, disp

format 은 현재 참조하고 있는 memory에 저장된 instruction이 몇형식인지를 저장한다. 이는, 해당 메모리의 어디까지가 instruction인지를 의미하기도 한다. opcode 는 instruction의 opcode를 저장한다. 이는 format 이 몇형식인지를 판별하기에도 필수적이다. niflag 는 isntruction이 참조하고 있는 주소에 있는 값들을 어떻게 처리할지를 판별한다. xbpeflag는 어떤 레지스터를 더해줄지, e flag의 경우 4형식임을 의미한다. 이를 참조해서 disp의 값을 정한다. disp 는 현재 참조하고 있는 주소값의 값 또는 niflag 에 따라 그 주소 자체나, 주소값에 해당하는 메모리가 가지는 주소에 있는 값을 참조할 수도 있다.

# 전역 변수 정의

## LSYM

localsymboltable 한 노드를 의미하는 구조체이다. local symboltable 은 현재 프로그램에 있는 definition 과 externalreference의 주소를 모두 저장하고 있는 linked list이다.

4.3 BP\* bphead, int startaddr, int totallength, int regarray[10]

## LSYM\* lsymboltable

lsymboltable의 헤드 역할을 하는 변수이다. AddLsymtab(int index, unsigned int address)을 호출하면 이 변수를 참조해 노드를 추가해준다. GetLsymtab(int index) 를 호출하면 해당 external reference나 definition의 주소값을 리턴을 통해 알 수 있다.

## BP\* bphead, int startaddr, int totallength, int regarray[10]

bphead 는 linked list로 관리되는 breakpoint의 head를 저장한다. SetBp(char\* input) 를 통해 Bp를 추가하고, GetBp(int address) 를 호출해, address 에서 가장 가까운 breakpoint 를 알 수 있다. startaddr 는 프로그램이 처음 시작하는 주소를 저장하고 있다. Control section 이 다른 위치에 있을 때는 이 값도 변할 수 있다. 이 때문에 startaddr가 변하는 함수들을 호출한 뒤에는 startaddr를 초기값으로 돌려준다. totallength는 프로그램의 총 길이를 의미한다. 프로그램의 종료 여부를 아는데 중요한 역할을 한다. regarray는 레지스터의 값을 저장하고 있는 행렬이다. Proj2에서 사용한 enum들을 바탕으로 각 인덱스에 레지스터의 값이 저장되어 있다.

# 코드

(길이가 길어서, c 소스파일 중 추가된 부분만 올리겠습니다.)

int Progaddr(char\* input)

{

char buf1[MAX];

int tempnum=0;

int wnum=0;

wnum=sscanf(input, "%s %X",buf1,&tempnum);

if(wnum != 2)

{

return 0;

}

else

{

startaddr = tempnum;

return 1;

}

}

int Loader(char\* input)

{

char fname[MAX];

int index=7;

int tempaddr=0;

int maxindex=strlen(input)-1;

tempaddr=startaddr;

while(index<maxindex)

{

if(input[index] == ' ')

{

++index;

continue;

}

sscanf(input+index,"%s",fname);

index+=strlen(fname);

//function to open specific file and read length from first line

if(!Extsymtab(fname))

{

return 0;

}

//

++index;

}

index=7;

startaddr = tempaddr;

while(index<maxindex)

{

if(input[index] == ' ')

{

++index;

continue;

}

sscanf(input+index,"%s",fname);

index+=strlen(fname);

//function to open specific file and read length from first line

LoadMem(fname);

//loading it on the memory

ModMem(fname);

//

++index;

}//completed making external symbol table

return 1;

}

int Extsymtab(char\* filename)

//creates a external symbol table nodes for a single object file

{

char buf[MAX];

char csectname[10];

char symname[10];

int addrtemp=0;

int length=0;

int index=1;

int wnum=0;

FILE\* fp;

memset(buf,0,sizeof(char)\*MAX);

if(!(fp = fopen(filename,"r")))

{

return 0;

}

if(!fgets(buf,MAX,fp))

//read first line for recognizing csect name and head record

{

printf("file head record not in format!\n");

}

sscanf(buf+1,"%s %06X%06X",csectname,&addrtemp,&length);

AddCsect(csectname,addrtemp,length);

totallength += length;

while(fgets(buf,MAX,fp))

{

if(buf[0] == 'D')

//found definition record

{

break;

}

}

if(buf[0] != 'D')

{

printf("no D record found!\n");

return 1;

}

while(sscanf(buf+index,"%s %06X",symname,&addrtemp)!= -1)

{

if(!AddEsym(symname, addrtemp))

{

return 0;

}

index += 12;

}

startaddr += length;

//at the end of the external symboltable creation of the obj file, increase the starting address

}

void AddCsect(char\* csectname, int addrtemp, int length)

{

ESYM\* newnode = (ESYM\*)malloc(sizeof(ESYM));

ESYM\* currnode = esymboltable;

strcpy(newnode->csect, csectname);

memset(newnode->symname,0,sizeof(char)\*10);

newnode->address = (startaddr+addrtemp);

newnode->length = length;

newnode->link = NULL;

//newnode initialization

if(currnode == NULL)

{

esymboltable = newnode;

return;

}

while(currnode->link != NULL)

{

currnode = currnode->link;

}

currnode->link = newnode;

return;

}

int AddEsym(char\* symname, int addrtemp)

{

ESYM\* newnode = (ESYM\*)malloc(sizeof(ESYM));

ESYM\* currnode=esymboltable;

memset(newnode->csect,0,sizeof(char)\*10);

strcpy(newnode->symname,symname);

newnode->address = (startaddr+addrtemp);

newnode->length = -1;

newnode->link = NULL;

if(currnode == NULL)

{

printf("objectfile has no preceding program label!\n");

return 0;

}

while(currnode->link !=NULL)

{

if(!strcmp(currnode->symname,symname))

{

printf("same node input\n");

return 0;

}

currnode = currnode -> link;

}

currnode->link = newnode;

return 1;

}

int AddLsymtab(int index, unsigned int address)

{

LSYM\* newnode = (LSYM\*)malloc(sizeof(LSYM));

LSYM\* currnode = lsymboltable;

int curridx=1;

newnode->laddress = address;

if(lsymboltable == NULL)

{

lsymboltable = newnode;

return 1;

}

while(++curridx < index)

{

if(currnode->link == NULL)

{

printf("error : node number not right!\n");

return 0;

}

currnode = currnode->link;

}

currnode->link = newnode;

return 1;

}

int AddDsymtab(char\* refname, int address)

{

DSYM\* newnode = (DSYM\*)malloc(sizeof(DSYM));

DSYM\* currnode = dsymboltable;

newnode->daddress = address;

newnode->link=NULL;

if(dsymboltable == NULL)

{

dsymboltable = newnode;

return 1;

}

while(currnode->link != NULL)

{

currnode = currnode->link;

}

currnode->link = newnode;

}

int AddTrec(unsigned int address, int length)

{

TREC\* newnode = (TREC\*)malloc(sizeof(TREC));

TREC\* currnode = trecordhead;

newnode->address = address;

newnode->length = length;

newnode->link=NULL;

if(trecordhead == NULL)

{

trecordhead = newnode;

return 1;

}

while(currnode->link != NULL)

{

currnode = currnode->link;

}

currnode->link = newnode;

}

void PrintExtsymtab()

{

ESYM\* currnode = esymboltable;

if(currnode == NULL)

{

printf("Empty external symbol node!\n");

return;

}

printf("\tcontrol\t\tsymbol\t\taddress\t\tlength\n");

printf("\tsection\t\tname\n");

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

while(currnode != NULL)

{

printf("\t%s\t\t%s\t\t%04X",currnode->csect,currnode->symname,currnode->address);

if(currnode->length != -1)

{

printf("\t\t%04X",currnode->length);

}

printf("\n");

currnode = currnode -> link;

}

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

printf("\t\t\t\t\ttotal length\t%04X\n",totallength);

return;

}

void PrintLsymtab()

{

LSYM\* currnode = lsymboltable;

if(currnode == NULL)

{

printf("Empty local symbol node!\n");

return;

}

printf("\tladdress\n");

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

while(currnode != NULL)

{

printf("\t%04X",currnode->laddress);

printf("\n");

currnode = currnode -> link;

}

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

return;

}

void ResetEsym()

{

ESYM\* deletenode = etemp;

while(etemp != NULL)

{

deletenode = etemp;

etemp = etemp->link;

deletenode->link = NULL;

free(deletenode);

}

return;

}

void SetBp(char\* input)

{

int address=0;

char buf[10];

BP\* newnode = (BP\*)malloc(sizeof(BP));

BP\* currnode = bphead;

BP\* tempnode=NULL;

newnode->link=NULL;

newnode->bp=-1;

if(sscanf(input+3,"%X %s",&address,buf)!=1)

{

printf("input not a decimal number!\n");

return;

}

//tokenize input and get address

newnode->bp = address;

if(currnode == NULL)

{

bphead = newnode;

return;

}

if(currnode->bp > address)

{

newnode->link = currnode;

bphead = newnode;

return;

}

while(currnode->bp < address)

{

if(currnode->link == NULL)

{

currnode->link = newnode;

return;

}

tempnode = currnode;

currnode = currnode->link;

}

if(currnode->bp == address)

{

printf("error : same bp input!\n");

return;

}

//when address is between two nodes

newnode->link = tempnode->link;

tempnode->link = newnode;

return;

}

int GetBp(int address)

{

BP\* currnode = bphead;

if(currnode == NULL)

//when there is no bp stored in the list

{

return -1;

}

while(address>currnode->bp)

{

if(currnode->link == NULL)

//when there is no bp bigger than address

{

return -1;

}

currnode = currnode->link;

}

return currnode->bp;

}

void PrintBp()

{

BP\* currnode = bphead;

if(currnode == NULL)

{

printf("Empty breakpoint node!\n");

return;

}

printf("\t\tbreakpoint\n");

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

while(currnode != NULL)

{

printf("\t\t%04X\n",currnode->bp);

currnode = currnode -> link;

}

return;

}

void LoadMem(char\* fname)

//fname contains the name of the obj file ex) "proga.obj"

{

FILE\* fp;

char buf[MAX];

char pname[MAX];

unsigned int index=0, maxindex=0, paddress=0, taddress=0;

int hb=0;

memset(buf,0,sizeof(char)\*MAX);

memset(pname,0,sizeof(char)\*MAX);

fp = fopen(fname,"r");

while(fgets(buf,MAX,fp))

{

index=0;

if((buf[0] == '.'))

{

continue;

}

else if(buf[0] == 'H')

{

sscanf(buf+1,"%s",pname);

paddress = GetCsect(pname);

}

else if(buf[0] == 'E')

{

break;

}

else if(buf[0] == 'T')

{

sscanf(buf+1,"%06X%02X",&taddress,&maxindex);

index+=4;

while(sscanf(buf+1+(index\*2),"%02X",&hb) != -1)

{

EditMem(paddress+taddress+index-4,hb);

index++;

}

AddTrec(taddress+paddress, maxindex);

}

}

return;

}

int GetCsect(char\* csectname)

{

ESYM\* currnode = esymboltable;

while(currnode!=NULL)

{

if(!strcmp(currnode->csect,csectname))

{

return currnode->address;

}

if(currnode->link ==NULL)

{

printf("error : program name not found!\n");

return -1;

}

currnode = currnode->link;

}

return 0;

}

int GetEsymtab(char\* input)

{

ESYM\* currnode = esymboltable;

if(currnode == NULL)

{

printf("error : empty external symbol table!\n");

return -1;

}

while(currnode->link != NULL)

{

if(!strcmp(currnode->symname,input))

{

return currnode->address;

}

currnode = currnode->link;

}

if(!strcmp(currnode->symname,input))

{

return currnode->address;

}

else

{

printf("error : node not found!\n");

return -1;

}

}

int GetLsymtab(int index)

{

LSYM\* currnode = lsymboltable;

int curridx=1;

if(currnode == NULL)

{

printf("error : local symbol table empty!\n");

return -1;

}

while(curridx < index)

{

currnode = currnode->link;

++curridx;

}

return currnode->laddress;

}

void ModMem(char\* fname)

{

FILE\* fp;

char buf[MAX];

char pname[MAX];

char refname[10];

unsigned int index=0, modlength=0, paddress=0, taddress=0, sindex=1, disp=0;

int hb=0;

memset(buf,0,sizeof(char)\*MAX);

memset(pname,0,sizeof(char)\*MAX);

memset(refname,0,sizeof(char)\*10);

fp = fopen(fname,"r");

while(fgets(buf,MAX,fp))

{

index=0;

if((buf[0] == '.'))

{

continue;

}

else if(buf[0] == 'H')

{

sscanf(buf+1,"%s",pname);

paddress = GetCsect(pname);

AddLsymtab(sindex,paddress);

//00 local symtable node becomes the head node

}

else if(buf[0] == 'D')

{

while(sscanf(buf+1+(index\*12),"%s %06X",refname,&taddress)!=-1)

{

index++;

AddDsymtab(refname,taddress);

//move to link index-1 times to add local symtable.

}

}

else if(buf[0] == 'R')

{

index=0;

//when input was came in from alphabet first@@@@@@@@@@@@@@@@@@@@@@

if(buf[1] == '0')

{

while(sscanf(buf+1+(index\*8),"%02X%s",&sindex,refname)!=-1)

{

index++;

AddLsymtab(index+1,GetEsymtab(refname));

//move to link index-1 times to add local symtable.

}

}

else

{

while(sscanf(buf+1+(index\*6),"%s",refname)!=-1)

{

index++;

AddLsymtab(index+1,GetEsymtab(refname));

}

}

}

else if(buf[0] == 'M')

{

sscanf(buf+1,"%06X%02X",&taddress,&modlength);

taddress += GetCsect(pname);

if(modlength == 5)

//format4 modification

{

disp = (mem[taddress]%16)\*65536+mem[taddress+1]\*256+mem[taddress+2];

}

else

//format3 modification

{

disp = mem[taddress]\*65536+mem[taddress+1]\*256+mem[taddress+2];

}

if(sscanf(buf+10,"%02X",&sindex) == 0)

{

continue;

}

if(buf[9] == '+')

{

disp += GetLsymtab(sindex);

}

else

{

disp -= GetLsymtab(sindex);

}

if(modlength == 5)

{

mem[taddress]/16\*16;

mem[taddress] += disp%1048576/65536;

mem[taddress+1] = disp%65536/256;

mem[taddress+2] = disp%256;

}

else

{

mem[taddress] = disp%16777216/65536;

mem[taddress+1] = disp%65536/256;

mem[taddress+2] = disp%256;

}

}

}

return;

}

void PrintReg()

{

int i=0;

printf("\t\tA : %06X X : %06X\n",regarray[RegA],regarray[RegX]);

printf("\t\tL : %06X PC: %06X\n",regarray[RegL],regarray[RegPC]);

printf("\t\tB : %06X S : %06X\n",regarray[RegB],regarray[RegS]);

printf("\t\tT : %06X\n",regarray[RegT]);

}

void Run()

{

unsigned int curraddr = 0;

unsigned int endaddr = totallength+startaddr;

unsigned int bpaddr=0;

unsigned int dispaddr=0;

int format=0;

int opcode=0;

int niflag = 0;

int xbpeflag = 0;

int disp=0;

regarray[RegPC] = startaddr;

regarray[RegL] = endaddr;

curraddr = regarray[RegPC];

bpaddr = GetBp(startaddr);

while(curraddr < endaddr)

{

format=3;

niflag = mem[curraddr]%4;

opcode = mem[curraddr]/4\*4;

xbpeflag = mem[curraddr+1]/16;

if((opcode == 160)||(opcode == 184)||(opcode == 180))

//compr, tixr, clear

{

format=2;

}

else

{

format += mem[curraddr+1]/16%2;

//add e flag to find what instruction is in format4

}

regarray[RegPC]+=format;

if(niflag == 1)

//direct addressing

{

dispaddr = mem[curraddr+1]%16\*256+mem[curraddr+2];

if(mem[curraddr+1]%16>7)

{

dispaddr = dispaddr|0xFFFFF000;

}

if(xbpeflag/8)

//xflag on

{

dispaddr += regarray[RegX];

}

if(xbpeflag%8/4)

//bflag on

{

dispaddr += regarray[RegB];

}

if(xbpeflag%4/2)

//pcflag on

{

dispaddr += regarray[RegPC];

}

disp = dispaddr;

}

else if(niflag == 2)

//indirect addressing

{

dispaddr = mem[curraddr+1]%16\*256+mem[curraddr+2];

if(mem[curraddr+1]%16>7)

{

dispaddr = dispaddr|0xFFFFF000;

}

if(xbpeflag/8)

//xflag on

{

dispaddr += regarray[RegX];

}

if(xbpeflag%8/4)

//bflag on

{

dispaddr += regarray[RegB];

}

if(xbpeflag%4/2)

//pcflag on

{

dispaddr += regarray[RegPC];

}

disp = mem[dispaddr]\*65536+mem[dispaddr+1]\*256+mem[dispaddr+2];

dispaddr = disp;

}

else if(niflag == 3)

//simple addressing

{

dispaddr = mem[curraddr+1]%16\*256+mem[curraddr+2];

if(mem[curraddr+1]%16>7)

{

dispaddr = dispaddr|0xFFFFF000;

}

if(xbpeflag/8)

//xflag on

{

dispaddr += regarray[RegX];

}

if(xbpeflag%8/4)

//bflag on

{

dispaddr += regarray[RegB];

}

if(xbpeflag%4/2)

//pcflag on

{

dispaddr += regarray[RegPC];

}

//disp = mem[dispaddr]\*65536+mem[dispaddr+1]\*256+mem[dispaddr+2];

disp = mem[dispaddr]\*65536+mem[dispaddr+1]\*256+mem[dispaddr+2];

}

if(niflag != 0)

{

if(mem[curraddr+1]/16%2)

{

dispaddr = mem[curraddr+1]%16\*65536+mem[curraddr+2]\*256+mem[curraddr+3];

disp = mem[dispaddr];

}

//add e flag to find what instruction is in format4

}

//calculate displacement

//@@@@@@@@@@@@@@@@@@@@@@

if(opcode == 0)

//instruction LDA

{

regarray[RegA] = disp;

}

else if(opcode == 104)

//instruction LDB

{

regarray[RegB] = disp;

}

else if(opcode == 116)

//instruction LDT

{

regarray[RegT] = disp;

}

else if(opcode == 80)

//instruction LDCH

{

regarray[RegA] = regarray[RegA]/256\*256;

regarray[RegA] = regarray[RegA]+disp/65536;

}

else if(opcode == 12)

//instruction STA

{

mem[dispaddr] = regarray[RegA]/65536;

mem[dispaddr+1] = regarray[RegA]%65536/256;

mem[dispaddr+2] = regarray[RegA]%256;

}

else if(opcode == 16)

//instruction STX

{

mem[dispaddr] = regarray[RegX]/65536;

mem[dispaddr+1] = regarray[RegX]%65536/256;

mem[dispaddr+2] = regarray[RegX]%256;

}

else if(opcode == 20)

//instruction STL

{

mem[dispaddr] = regarray[RegL]/65536;

mem[dispaddr+1] = regarray[RegL]%65536/256;

mem[dispaddr+2] = regarray[RegL]%256;

}

else if(opcode == 84)

//instruction STCH

{

mem[disp] = regarray[RegA]%256;

}

else if(opcode == 60)

//instruction J

{

regarray[RegPC] = dispaddr;

curraddr = regarray[RegPC];

continue;

}

else if(opcode == 72)

//instruction JSUB

{

regarray[RegL] = regarray[RegPC];

regarray[RegPC] = dispaddr;

curraddr = regarray[RegPC];

continue;

}

else if(opcode == 56)

//instruction JLT

{

if(regarray[RegSW] == -1)

//-1 for a less than b

{

regarray[RegPC] = dispaddr;

curraddr = regarray[RegPC];

continue;

}

}

else if(opcode == 48)

//instruction JEQ

{

if(regarray[RegSW] == 0)

{

regarray[RegPC] = dispaddr;

curraddr = regarray[RegPC];

continue;

}

}

else if(opcode == 76)

//instruction RSUB

{

regarray[RegPC] = regarray[RegL];

curraddr = regarray[RegPC];

continue;

}

else if(opcode == 40)

//instruction COMP

{

if(regarray[RegA] < disp)

{

regarray[RegSW] = -1;

}

else if(regarray[RegA] == disp)

{

regarray[RegSW] = 0;

}

else

{

regarray[RegSW] = 1;

}

}

else if(opcode == 160)

//instruction COMPR

{

disp = mem[curraddr+1];

if(regarray[disp/16] < regarray[disp%16])

{

regarray[RegSW] = -1;

}

else if(regarray[disp/16] == regarray[disp%16])

{

regarray[RegSW] = 0;

}

else

{

regarray[RegSW] = 1;

}

}

else if(opcode == 180)

//instruction CLEAR

{

disp = mem[curraddr+1];

regarray[disp/16] = 0;

}

else if(opcode == 184)

//instruction TIXR

{

disp = mem[curraddr+1];

regarray[RegX] ++;

if(regarray[RegX] < regarray[disp/16])

{

regarray[RegSW] = -1;

}

else if(regarray[RegX] == regarray[disp/16])

{

regarray[RegSW] = 0;

}

else

{

regarray[RegSW] = 1;

}

}

else if(opcode == 224)

//instruction TD

{

regarray[RegSW]=1;

}

else if(opcode == 216)

//instruction RD

{

regarray[RegA]/16\*16;

}

else if(opcode == 220)

//instruction WD

{

}

//@@@@@@@@@@@@@@@@@@@@@@

curraddr+=format;

if(curraddr == bpaddr)

{

PrintReg();

printf("Stop at checkpoint[%04X]\n",bpaddr);

bpaddr = GetBp(bpaddr);

regarray[RegPC] = curraddr+format;

return;

}

}

PrintReg();

printf("\tEnd program.\n");

regarray[RegPC] = startaddr;

return;

}