**과목명: 시스템프로그래밍**

**1반**

**<<Project #3>>**

**서강대학교 컴퓨터공학과**

**20161563**

**고성빈**

목 차

1 프로그램 개요

2 프로그램 설명

2.1 프로그램 흐름도(종합)

2.2 프로그램 흐름도(프로젝트 3 상세)

3 모듈 및 전역 변수 정의[[1]](#footnote-1)

3.1 메인 함수 변경점

3.1.1 int main()

3.2 help 명령어 변경점

3.3.1 int function\_help()

3.3 progaddr 명령어 구현

3.3.1 전역 변수

3.3.2 int function\_progaddr()

3.3.3 void set\_progaddr()

3.4 loader 명령어 구현

3.4.1 구조체 및 전역 변수

3.4.2 int function\_loader()

3.4.3 void loader\_part1()

3.4.4 void loader\_part2()

3.4.5 int find\_addr()

3.4.6 void print\_estab()

3.4.7 void add\_mod(int, int)

3.5 bp 명령어 구현

3.5.1 전역 변수

3.5.2 int function\_bp()

3.5.3 void set\_bp()

3.6 run 명령어 구현

3.6.1 구조체 및 전역 변수

3.6.2 int function\_run()

3.6.3 void esec\_opcode(int)

3.6.4 int find\_format(int, int)

3.6.5 void format1(int)

3.6.6 void format2(int, int)

3.6.7 void format34(int, int)

4 코드 설명

4.1 20161563.h 추가 사항

4.2 20161563.c 추가 사항

1 프로그램 개요

이 프로그램은 프로젝트 1, 2를 통해 구현한 셸(Shell)에 linkingr과 loading 기능을 추가하는 프로그램이다. 프로젝트 2에서 구현한 assemble 명령을 통해 생성된 object 파일을 link시켜 메모리에 올리고 이를 직접 실행하는 기능을 지원한다.

2 프로그램 설명

프로그램은 실행되고 나서 초기화 단계를 거친 후, 종료되기 전까지 다음과 같은 단계를 반복한다.

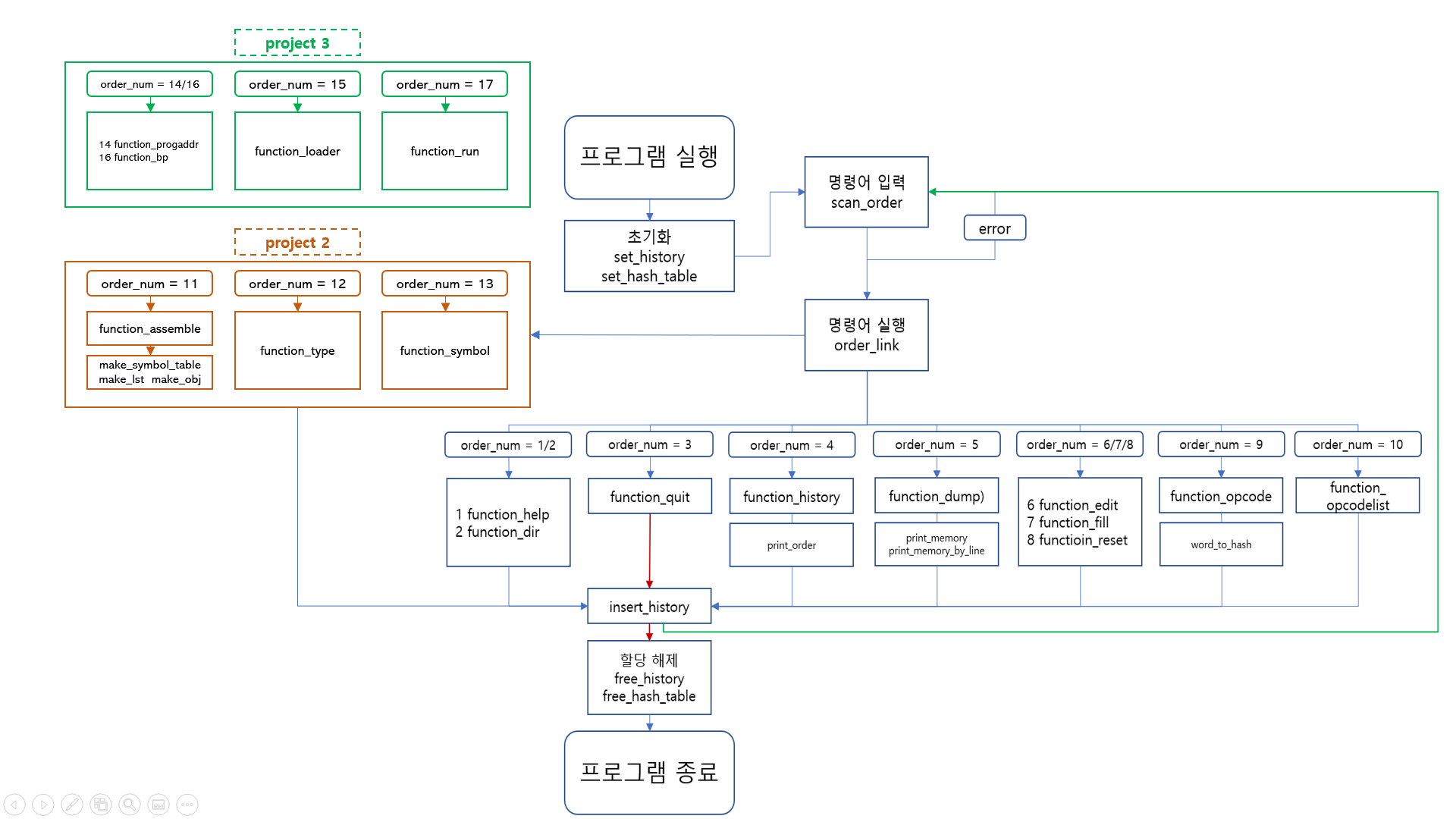
scan\_order로 명령어를 입력받고, order\_link를 통해 해당 명령어의 함수를 실행한다. 해당 함수가 종료된 후 insert\_history 함수를 통해 입력 내용을 기록하고 다시 명령어 입력 단계로 돌아온다. quit 명령어가 입력된 경우에만 반복문을 종료하고, 할당된 메모리를 해제한 뒤 프로그램을 종료한다.

프로젝트 2를 통해 기존의 프로그램에 assemble 기능이 추가되어 어셈블리 파일을 object 파일로 수정할 수 있게 되었다.

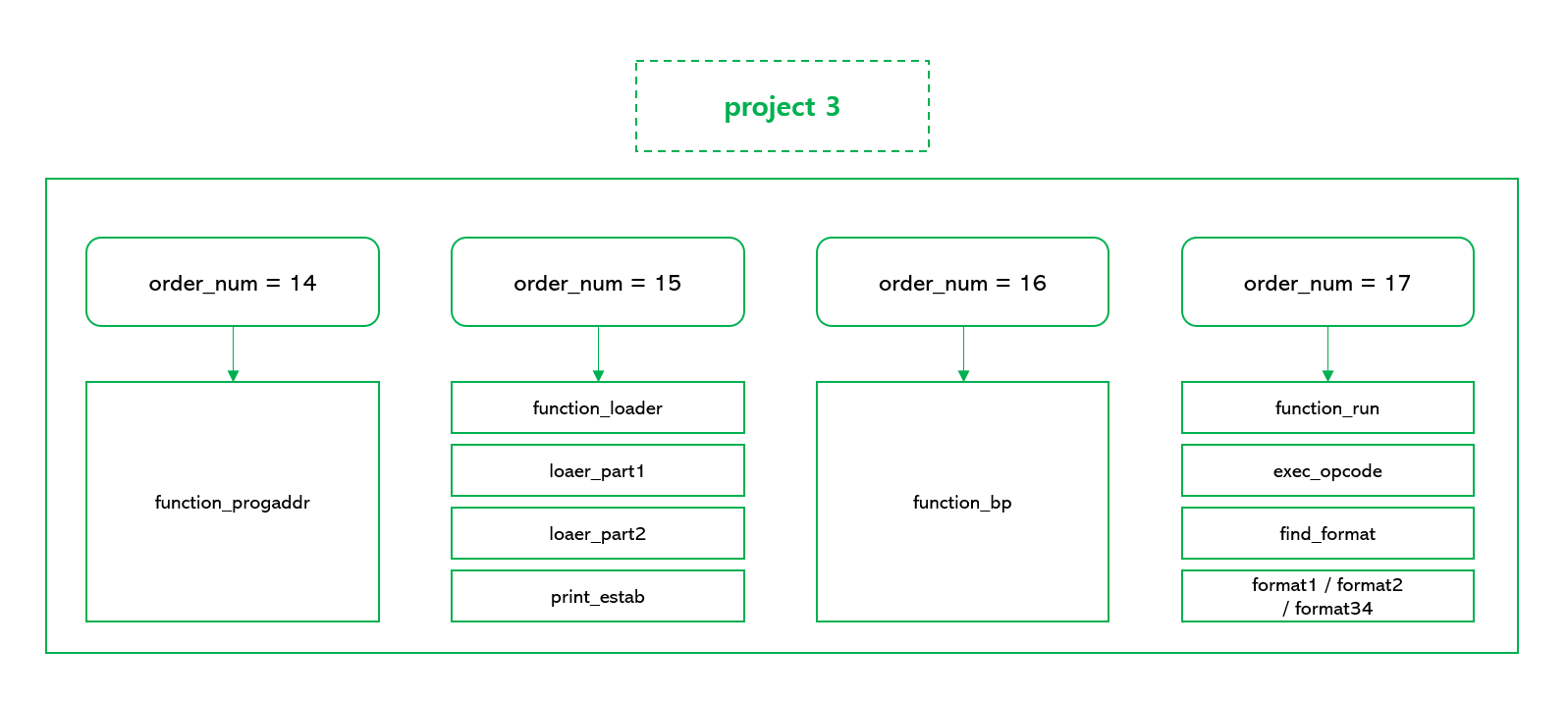
프로젝트 3을 통해 기존의 프로그램에 loader, run 기능이 추가되어 object 파일을 메모리에 저장하고 수행할 수 있게 되었다.

2.1 프로그램 흐름도(종합)

프로그램 흐름도는 다음과 같다.



2.2 프로그램 흐름도(프로젝트 3 상세)

위 프로그램에서 project 3 부분을 더 자세하게 표현하면 다음과 같다.

3 모듈 정의

3.1 메인 함수 변경점

프로그램에서 지원하는 명령어의 종류가 늘어남에 따라 기존의 메인 함수에서 실행하던 initialize 기능이 추가되었다. 프로그램 시작 시의 program load address(변수 int progaddr)을 0으로 초기화해주고, break point(bp)가 하나도 설정되지 않은 배열을 만든다.

3.1.1 int main()

- main 함수는 set\_history와 set\_hash\_table 함수를 통한 linked list 생성과 반복문 내에서 명령어

입력(scan\_order)과 실행(order\_link) 함수 호출, 이후 프로그램 종료 전 free\_history와

free\_hash\_table을 통해 linked list 할당 해제하는 기능을 한다.

- 추가 함수 :

void set\_progaddr() : 프로그램 로드를 위한 progaddr을 0으로 초기화한다.

void set\_bp() : break point를 확인하기 위한 배열을 0으로 초기화한다.

3.2 help 명령어 변경점

기존 13개의 명령어를 출력하던 help 명령어는 프로젝트 2에서 추가된 4개의 명령어 progaddr, loader, bp, run을 추가로 출력해준다.

3.2.1 int function\_help()

- function\_help 함수는 rest\_exist를 확인해 추가적인 인자가 있는지 보고, 없을 경우 명령어 목록을

출력한다. 인자가 있을 경우에는 에러처리한다.

3.3 progaddr 명령어 구현

progaddr 명령어는 1개의 인자를 받아 program load address를 해당 값으로 바꾸어준다.

입력받은 값이 16진수인지 확인해주고, 해당 값이 memory address 범위 내에 있는지 확인한다.

3.3.1 구조체 및 전역 변수

int progaddr : program load address를 저장한다.

3.3.2 int function\_progaddr()

- function\_progaddr 함수는 16진수로 주소값을 입력받아 program loade address를 해당 주소값으로설정해 준다. 해당 범위가 memory address 범위 내에 있는지 확인해 아닐 경우 program load address를 수정하지 않고 에러 메시지를 출력 후 -1을 반환한다.

- 사용 변수

int temp : 입력받은 값을 문자열에서 16진수로 변화하는 과정에서 사용한다.

3.3.3 void set\_progaddr()

- set\_progaddr 함수는 프로그램의 시작시 initialize 과정에서 호출되어 전역변수 progaddr을 0으로 초기화한다.

3.4 loader 명령어 구현

loader 명령어는 최대 3개까지의 파일명을 인자로 받아 해당 파일들을 memory에 load해준다. 그 과정에서 파일에 적힌 주소에 바로 저장하는 대신 linking 과정을 통해 program load address로부터 파일을 로드한다.

3.4.1 구조체 및 전역 변수

struct exsym : load 과정에서 필요한 external symbol의 이름과 주소를 저장한다.

exsym estab[3][100] : external symbol을 저장한다.

int progst[3], proglen[3], esnum[3] : 프로그램의 linking 후 시작 주소, 길이, external symbol의

갯수를 저장한다.

int execst[3] : 각 프로그램의 첫 실행 가능 코드가 적힌 주소값을 저장한다.

3.4.2 int\_function\_loader()

- function\_loader 함수는 최대 3개의 파일명을 입력받아 해당 함수를 프로그램 memory에 할당해준다. loader\_part1, loader\_part2, print\_estab 함수를 차례로 호출하며 pass1, pass2 과정을 수행하고 결과를 출력한다.

- 사용 변수

int files : 입력받은 파일의 개수를 저장한다.

- 호출 함수

loader\_part1(filename, files) : pass1 (external symbol table 작성) 과정을 수행한다.

loader\_part2(filename, files) : pass2 (파일을 memory에 저장) 과정을 수행한다.

print\_estab(files) : 파일 수행 후 external symbol값들을 출력해준다.

3.4.3 void loader\_part1(char filename[3][20], int files)

- loader의 pass1 과정을 수행해준다. 입력받은 파일들을 차례로 읽으며 H 레코드의 값과 D 레코드의 symbol을 estab에 저장한다. E레코드를 마주치면 종료한다.

3.4.4 void loader\_part2(char filename[3][20], int files)

- loader의 pass2 과정을 수행해준다. 입력받은 파일들을 차례로 읽으며 R 레코드를 만나면 해당 external symbol의 index와 address를 미리 확인한다. T 레코드를 만나면 메모리의 해당 address에 코드를 저장한다. M 레코드를 만나면 해당 address로 가 값을 수정해준다. E 레코드를 만나면 해당 파일을 읽는 것을 종료한다.

- 호출 함수

find\_addr(name) : external symbol의 주소값을 확인하는데 사용한다.

add\_modi(modiaddr, adaddr) : 수정할 주소로 이동해 주소값 수치만큼 더해준다.

3.4.5 int find\_addr(char refs[20])

- symbol name을 입력받아 estab에서 해당 주소를 찾아 반환한다.

3.4.6 void add\_modi(int modiaddr, int adaddr)

- 메모리의 주소로 이동해 해당 address~address+2의 범위에 adaddr 값만큼을 더해준다.

3.4.7 void print\_estab(int files)

- 입력받은 file 개수만큼 external symbol table을 읽은 다음, 출력 양식에 맞추어 control section/symbol name, address, length를 출력해준다.

3.5 bp 명령어 구현

bp 명령어는 breakpoint를 지정해주거나, 전부 초기화하거나, 설정된 모든 breakpoint를 확인한다.

3.5.1 전역 변수

char bp[65536][16] : 해당 memory에 bp가 설정되어 있는지 확인한다.

int bpnum : bp의 개수를 저장한다.

3.5.2 function\_bp()

- 뒤에 받는 인자에 따라 실행하는 결과가 달라진다. 인자가 없을 경우 전체 bp를 출력하고, 인자로 clear가 올 경우 모든 bp를 제거한다. 인자로 16진수의 수가 올 경우 memory range 내부일 경우 해당 위치에 breakpoint를 설정해준다.

- 사용 변수

char oder[20] : 입력받은 인자를 저장한다. clear나 16진수의 숫자가 들어온다.

int newbp : 새 breakpoint를 설정할 메모리 주소를 저장한다.

- 호출 함수

void set\_bp() : bp 초기화 시에 호출해 모든 bp를 제거한다.

3.5.3 void set\_bp()

- 프로그램 시작 시, 그리고 bp clear 명령어가 들어올 때 호출되어 모든 bp 배열에 저장된 값을0으로초기화한다.

3.6 run 명령어 구현

run 명령어는 인자를 가지지 않고 execaddr로부터 프로그램을 시작해 프로그램 범위를 벗어날 때까지 프로그램을 실행한다.

3.5.1 전역 변수

int regis[10] : 0~9번 레지스터 값을 저장한다.

3.5.2 int function\_run()

- PC register와 종료 주소, 그리고 breakpoint를 확인하며 프로그램의 수행을 반복문을 통해 반복한다. breakpoint에 도달하거나 end address를 지날 경우 프로그램을 종료한다.

- 사용 변수

int curaddr : 현재 PC register값을 저장한다.

int nextaddr : 다음 PC register 값을 저장해 실행 정지/종료 조건을 확인한다.

- 호출 함수

exec\_opcode(curaddr) : 현재 주소에서 명령을 수행한다.

3.5.3 void exec\_opcode(int curaddr)

- current address를 통해 첫 주소의 값을 확인한다. 이후 find\_format함수를 통해 포맷을 확인하고 format1/2/34를 호출해 명령을 수행한다.

- 사용 변수

int bit1 : memory의 curaddr에 저장된 값을 가진다.

- 호출 함수

find\_format(curaddr, bit1) : 포맷을 확인하고 해당 포맷에 해당하는 실행 함수를 호출한다.

3.5.4 int find\_format(int curaddr, int bit1)

- opcode 값의 앞 4bit를 확인해 해당 값으로 포맷을 확인한다. 포맷 값에 따라 PC 레지스터의 값을 증가시킨 이후 각 포맷 타입을 실행하는 함수를 실행한다.

- 사용 변수

int bit1 : memory의 curaddr에 저장된 값을 가진다.

- 호출 함수

void format1 : 포맷 1 형식의 코드를 수행한다.

void format2 : 포맷 2 형식의 코드를 수행한다.

void format34 : 포맷 3 또는 4 형식의 코드를 수행한다.

3.5.5 void format1(int bit1)

- format 1에 해당하는 함수를 수행한다. 이번 프로젝트에서는 포맷 1에 대해서는 따로 처리해주지 않으므로 아무것도 수행하지 않고 반환한다.

- 사용 변수

int bit1 : memory의 curaddr에 저장된 값을 가진다.

3.5.6 void format2(int curaddr, int bit1)

- format 2에 해당하는 함수를 수행한다. curaddr과 curaddr+1번째의 메모리 값을 읽어 curaddr + 1번째 메모리는 레지스터로 인식한다. 해당 함수에는 COMPR, CLEAR, TIXR이 있다.

- 사용 변수

int r1, r2 : 다음 주소에 있는 값을 레지스터로 인식한다.

3.5.7 void format34(int curaddr, int bit1)

- format 3과 4에 해당하는 함수를 수행한다. 다음 byte의 e bit를 확인해 포맷을 확인한다. 이후 format에 해당하는 길이의 데이터를 읽고, xbp bit를 통해 확인할 address를 정한 뒤 ni bit를 통해 addressing mode를 정한다. 이후 해당 위치에서 함수를 수행한다.

- 사용 변수

int nformat : 현재 포맷을 저장한다. 기본으로 3이 저장되었다가 e bit를 확인해 포맷 4일경우 4로 증가한다.

int ta : 타겟 위치의 데이터값을 저장한다.

int tb : 타겟 위치의 주소값을 저장한다.

- 호출 함수

int reg(char\*) :

4 코드 설명

4.1 20161563.h 추가 사항

/\* structures \*/

~~~ line 50

typedef struct \_exsym {

char name[10];

int addr;

} exsym;

/\* functions \*/

~~~ line 137

// order progaddr

int function\_progaddr();

void set\_progaddr();

// order loader

int function\_loader();

void loader\_part1(char[3][20], int);

void loader\_part2(char[3][20], int);

int find\_addr(char[20]);

void print\_estab(int);

void add\_modi(int, int);

// order bp

int function\_bp();

void set\_bp();

// order run

int function\_run();

void exec\_opcode(int);

int find\_format(int, int);

void format1(int);

void format2(int, int);

void format34(int, int

/\* global variables \*/

~~~ line 190

// program address

int progaddr;

// external symbol table

exsym estab[3][100];

int progst[3], proglen[3], esnum[3];

int execst[3];

// breakpoint

char bp[65536][16];

int bpnum;

// run

int regis[10];

4.2 20161563.c 추가 사항

~~~ line 1866

/\* progaddr(14) : set program load address \*/

int function\_progaddr()

{

if(rest\_exist == 0)

{

printf("input valid order\n");

return -1;

}

char pa[10];

scanf("%[^\n]%\*c", pa);

int temp = 0;

for(int i = 0; i < strlen(pa); i++)

{

temp \*= 16;

if('A' <= pa[i] && pa[i] <= 'F')

temp += pa[i] - 'A' + 10;

else if('0' <= pa[i] && pa[i] <= '9')

temp += pa[i] - '0';

else

{

printf("address number error!\n");

return -1;

}

}

progaddr = temp;

insert\_history(14, pa);

return 0;

}

void set\_progaddr()

{

progaddr = 0;

return;

}

/\* loader(15) : link and load object files \*/

int function\_loader()

{

if(rest\_exist == 0)

{

printf("input valid order\n");

return -1;

}

int files = 0;

char filename[3][20];

FILE\* fp;

// read filenames (maximum 3)

for(files = 0; files < 3; files++)

{

char check;

scanf("%s", filename[files]);

scanf("%c", &check);

// check file exists

fp = fopen(filename[files], "r");

if(fp == NULL)

{

printf("file open error\n");

return -1;

}

fclose(fp);

if(check == '\n')

{

files++;

break;

}

}

loader\_part1(filename, files);

loader\_part2(filename, files);

print\_estab(files);

// set registers

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

regis[i] = 0;

regis[reg("PC")] = progaddr;

for(int i = 0; i < files; i++)

regis[reg("L")] += proglen[i];

char tot[100] = {0, };

for(int i = 0; i < files; i++)

{

strcat(tot, filename[i]);

strcat(tot, " ");

}

insert\_history(15, tot);

return 0;

}

void loader\_part1(char filename[3][20], int files)

{

FILE\* fp;

int curad = progaddr;

for(int i = 0; i < files; i++)

{

fp = fopen(filename[i], "r");

progst[i] = curad;

int index = 1;

int init = 0;

int len = 0;

// read file

while(1)

{

char st[300] = {0, };

fgets(st, 300, fp);

// check END

if(st[0] == 'E')

break;

// header record

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

{

sscanf(st + 1, "%6s", estab[i][0].name);

estab[i][0].addr = curad;

sscanf(st + 7, "%6X%6X", &init, &len);

}

// define record

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

for(int j = 1; j < strlen(st) - 1; j += 12)

{

sscanf(st + j, "%6s", estab[i][index].name);

sscanf(st + j + 6, "%6X", &estab[i][index].addr);

estab[i][index].addr += curad;

index++;

}

}

curad += len;

esnum[i] = index;

proglen[i] = len;

fclose(fp);

}

return;

}

void loader\_part2(char filename[3][20], int files)

{

for(int fnum = 0; fnum < files; fnum++)

{

int csaddr = progst[fnum];

FILE\* fp = fopen(filename[fnum], "r");

exsym reftab[100];

strcpy(reftab[0].name, estab[fnum][0].name);

reftab[0].addr = estab[fnum][0].addr;

int refnum = 1;

while(1)

{

char st[300];

fgets(st, 300, fp);

// reference record

if(st[0] == 'R')

{

for(refnum = 1; 8\*refnum - 5 < strlen(st) - 1; refnum++)

{

sscanf(st + 8\*refnum - 5, "%6s", reftab[refnum].name);

reftab[refnum].addr = find\_addr(reftab[refnum].name);

}

}

// text record

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

{

int naddr;

sscanf(st + 1, "%6X", &naddr);

naddr += csaddr;

int linelen;

sscanf(st + 7, "%2X", &linelen);

for(int i = 0; i < linelen; i++)

{

int posy = (naddr + i) / 16;

int posx = (naddr + i) % 16;

int num;

sscanf(st + 9 + 2\*i, "%2X", &num);

memory[posy][posx] = (unsigned char)num;

}

}

// modification record

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

{

int modiaddr;

sscanf(st + 1, "%6X", &modiaddr);

modiaddr += progst[fnum];

int modibit;

sscanf(st + 7, "%2X", &modibit);

char posneg;

sscanf(st + 9, "%c", &posneg);

int modiref;

sscanf(st + 10, "%2X", &modiref);

int adaddr = reftab[modiref-1].addr;

if(posneg == '-')

adaddr = (1<<24) - adaddr;

add\_modi(modiaddr, adaddr);

}

// end record

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

{

sscanf(st + 1, "%6X", &execst[files]);

execst[files] += progst[files];

break;

}

}

}

return;

}

int find\_addr(char refs[20])

{

for(int tnum = 0; tnum < 3; tnum++)

for(int i = 0; i < esnum[tnum]; i++)

if(strcmp(refs, estab[tnum][i].name) == 0)

return estab[tnum][i].addr;

printf("cannot find %s\n", refs);

return 0;

}

void add\_modi(int modiaddr, int adaddr)

{

unsigned char uc[3];

uc[0] = adaddr / (1<<16);

uc[1] = (adaddr / (1<<8)) % (1<<8);

uc[2] = adaddr % (1<<8);

int carry = 0;

for(int i = 2; i >= 0; i--)

{

int maddr = modiaddr + i;

int posy = maddr / 16;

int posx = maddr % 16;

if(memory[posy][posx] + uc[i] + carry >= 256)

{

memory[posy][posx] += uc[i] + carry - 256;

carry = 1;

}

else

{

memory[posy][posx] += uc[i] + carry;

carry = 0;

}

}

return;

}

void print\_estab(int files)

{

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

printf("section name\n");

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

for(int i = 0; i < files; i++)

{

printf("%-6s %6s %04X %04X\n", estab[i][0].name, "", estab[i][0].addr, proglen[i]);

for(int x = 1; x < esnum[i]; x++)

printf("%6s %6s %04X\n", "", estab[i][x].name, estab[i][x].addr);

}

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

int totlen = 0;

for(int i = 0; i < files; i++)

totlen += proglen[i];

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

return;

}

/\* bp(16) : set breakpoint \*/

int function\_bp()

{

// bp

if(rest\_exist == 0)

{

printf("\tbreakpoint\n");

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

for(int y = 0; y < 65536; y++)

for(int x = 0; x < 16; x++)

if(bp[y][x]) printf("\t%X\n", y\*16 + x);

insert\_history(16, "");

return 0;

}

char order[20] = {0, };

scanf("%[^\n]", order);

scanf("%\*c");

// too long

if(strlen(order) > 10)

{

printf("wrong address\n");

return -1;

}

// bp clear

if(strcmp(order, "clear") == 0)

{

printf("\t[ok] clear breakpoints\n");

set\_bp();

insert\_history(16, order);

return 0;

}

// bp address

int newbp = 0;

for(int i = 0; i < 20; i++)

{

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

{

newbp \*= 16;

newbp += order[i] - '0';

}

else if(order[i] >= 'A' && order[i] <= 'F')

{

newbp \*= 16;

newbp += order[i] - 'A' + 10;

}

else if(order[i] >= 'a' && order [i] <= 'f')

{

newbp \*= 16;

newbp += order[i] - 'a' + 10;

}

else if(order[i] == '\0')

break;

else

{

printf("wrong address\n");

return -1;

}

}

if(newbp >= 0 && newbp < memory\_end)

bp[newbp/16][newbp%16] = 1;

else

{

printf("address out of range\n");

return -1;

}

printf("\t[ok] create breakpoint %X\n", newbp);

bpnum++;

insert\_history(16, order);

return 0;

}

void set\_bp()

{

for(int y = 0; y < 65536; y++)

for(int x = 0; x < 16; x++)

bp[y][x] = 0;

bpnum = 0;

return;

}

/\* run(17) : run program \*/

int function\_run()

{

if(rest\_exist == 1)

{

dump\_rest();

return -1;

}

int endaddr = progst[0] + proglen[0];

while(1)

{

int curaddr = regis[reg("PC")];

// execute opcode

exec\_opcode(curaddr);

int nextaddr = regis[reg("PC")];

// check breakpoint

if(bp[nextaddr/16][nextaddr%16] == 1)

{

printf("A : %06X X : %06X\n", regis[reg("A")], regis[reg("X")]);

printf("L : %06X PC : %06X\n", regis[reg("L")], regis[reg("PC")]);

printf("B : %06X S : %06X\n", regis[reg("B")], regis[reg("S")]);

printf("T : %06X\n", regis[reg("T")]);

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

break;

}

// check program ends

if(nextaddr >= endaddr)

{

printf("A : %06X X : %06X\n", regis[reg("A")], regis[reg("X")]);

printf("L : %06X PC : %06X\n", regis[reg("L")], regis[reg("PC")]);

printf("B : %06X S : %06X\n", regis[reg("B")], regis[reg("S")]);

printf("T : %06X\n", regis[reg("T")]);

printf(" End Program\n");

break;

}

}

insert\_history(17, "");

return 0;

}

void exec\_opcode(int curaddr)

{

int posy = curaddr / 16;

int posx = curaddr % 16;

int bit1 = memory[posy][posx];

find\_format(curaddr, bit1);

return;

}

int find\_format(int curaddr, int bit1)

{

switch(bit1 / 16)

{

case 12:

case 15:

regis[reg("PC")] += 1;

format1(bit1);

break;

case 9:

case 10:

case 11:

regis[reg("PC")] += 2;

format2(curaddr, bit1);

break;

default:

regis[reg("PC")] += 3;

format34(curaddr, bit1);

}

return 0;

}

void format1(int bit1)

{

return;

}

void format2(int curaddr, int bit1)

{

int bit2 = memory[(curaddr+1)/16][(curaddr+1)%16];

int r1 = bit2 / 16;

int r2 = bit2 % 16;

if(bit1 == 0xA0) // COMPR

{

int regr1 = regis[r1];

int datar1 = memory[regr1/16][regr1%16];

int regr2 = regis[r2];

int datar2 = memory[regr2/16][regr2%16];

if(datar1 == datar2)

regis[reg("SW")] = '=';

else if(datar1 < datar2)

regis[reg("SW")] = '<';

else

regis[reg("SW")] = '>';

}

if(bit1 == 0xB4) // CLEAR

{

regis[r1] = 0;

}

if(bit1 == 0xB8) // TIXR

{

regis[reg("X")]++;

int regx = regis[reg("X")];

int datax = memory[regx/16][regx%16];

int regr = regis[r1];

int datar = memory[regr/16][regr%16];

if(datax == datar)

regis[reg("SW")] = '=';

else if(datax < datar)

regis[reg("SW")] = '<';

else

regis[reg("SW")] = '>';

}

return;

}

void format34(int curaddr, int bit1)

{

int op = (bit1 / 4) \* 4;

int nformat = 3;

int bit2 = memory[(curaddr+1)/16][(curaddr+1)%16];

int bit3 = memory[(curaddr+2)/16][(curaddr+2)%16];

int bit4 = memory[(curaddr+3)/16][(curaddr+3)%16];

// check e bit(format 4)

if(bit2 & (1<<4))

{

nformat = 4;

regis[reg("PC")]++;

}

int ta;

if(nformat == 3)

{

ta = (bit2 % 16)<<8;

ta += bit3;

}

if(nformat == 4)

{

ta = (bit2 % 16)<<16;

ta += bit3<<8;

ta += bit4;

}

if(bit2 & (1<<6)) // b bit

ta += regis[reg("B")];

if(bit2 & (1<<5)) // p bit

{

if(ta >= 2048)

ta -= 4096;

ta += regis[reg("PC")];

}

if(bit2 & (1<<7)) // x bit

ta += regis[reg("X")];

//check ni bit

int tb = 0;

// immediate addressing. ni = 01

if(bit1 % 4 == 1)

tb = ta;

// simple addressing. ni = 11

if(bit1 % 4 == 3)

{

tb = ta;

int d1 = memory[ta/16][ta%16];

int d2 = memory[(ta+1)/16][(ta+1)%16];

int d3 = memory[(ta+2)/16][(ta+2)%16];

ta = (d1<<16) + (d2<<8) + d3;

}

// indirect addressing. ni = 10

if(bit1 % 4 == 2)

{

int d1 = memory[ta/16][ta%16];

int d2 = memory[(ta+1)/16][(ta+1)%16];

int d3 = memory[(ta+2)/16][(ta+2)%16];

ta = (d1<<16) + (d2<<8) + d3;

tb = ta;

d1 = memory[ta/16][ta%16];

d2 = memory[(ta+1)/16][(ta+1)%16];

d3 = memory[(ta+2)/16][(ta+2)%16];

ta = (d1<<16) + (d2<<8) + d3;

}

if(op == 0x00) // LDA

regis[reg("A")] = ta;

if(op == 0x0C) // STA

{

int data = regis[reg("A")];

memory[tb/16][tb%16] = data / (1<<16);

memory[(tb+1)/16][(tb+1)%16] = data / (1<<8) % (1<<8);

memory[(tb+2)/16][(tb+2)%16] = data % (1<<8);

}

if(op == 0x10) // STX

{

int data = regis[reg("X")];

memory[tb/16][ta%16] = data / (1<<16);

memory[(tb+1)/16][(tb+1)%16] = data / (1<<8) % (1<<8);

memory[(tb+2)/16][(tb+2)%16] = data % (1<<8);

}

if(op == 0x14) // STL

{

int data = regis[reg("L")];

memory[tb/16][tb%16] = data / (1<<16);

memory[(tb+1)/16][(tb+1)%16] = data / (1<<8) % (1<<8);

memory[(tb+2)/16][(tb+2)%16] = data % (1<<8);

}

if(op == 0x28) // COMP

{

if(regis[reg("A")] == ta)

regis[reg("SW")] = '=';

if(regis[reg("A")] < ta)

regis[reg("SW")] = '<';

if(regis[reg("A")] > ta)

regis[reg("SW")] = '>';

}

if(op == 0x30) // JEQ

{

if(regis[reg("SW")] == '=')

regis[reg("PC")] = tb;

}

if(op == 0x38) // JLT

{

if(regis[reg("SW")] == '<')

regis[reg("PC")] = tb;

}

if(op == 0x3C) // J

{

regis[reg("PC")] = tb;

}

if(op == 0x48) // JSUB

{

regis[reg("L")] = regis[reg("PC")];

regis[reg("PC")] = tb;

}

if(op == 0x4C) // RSUB

{

regis[reg("PC")] = regis[reg("L")];

}

if(op == 0x50) // LDCH

{

regis[reg("A")] = ((regis[reg("A")]>>8)<<8) + ta / (1<<16);

}

if(op == 0x54) // STCH

{

memory[tb/16][tb%16] = regis[reg("A")] % (1<<8);

}

if(op == 0x68) // LDB

regis[reg("B")] = ta;

if(op == 0x74) // LDT

regis[reg("T")] = ta;

if(op == 0xD8) // RD

{

regis[reg("A")] = 0;

}

if(op == 0xDC) // WD

{

// do nothing

}

if(op == 0xE0) // TD

regis[reg("SW")] = '<';

return;

}

1. 원활한 설명을 위해 모듈 정의 단계와 전역 변수 정의 단계를 합쳤습니다. [↑](#footnote-ref-1)