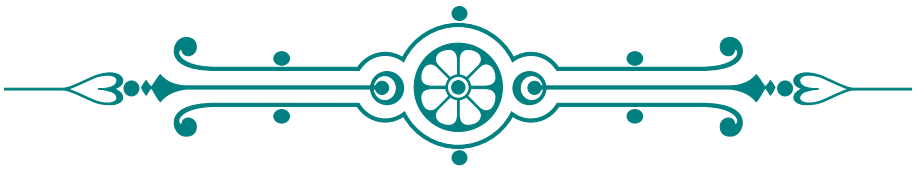

시스템 S/W 실습5

학번 : 2016110056

학과 : 불교학부

이름 : 박승원

날짜 : 2016년 10월 20일

1. 다음에 주어진 프로그램 참고해서, srcfile(입력파일)을 읽고 각 줄(line)을 LABEL, OPCODE, OPERAND로 분리해서 intfile(임시출력파일)에 저장하는 프로그램을 작성해서 test5.c로 저장하고, test5.c를 Compile 하여 test run하시오.
   단 C Program compile 명령은 다음과 같고, test의 입력파일 srcfile(SIC 어셈블리어 프로그램 파일)은 각자 준비한다.

   $gcc -0 test5 test5.c
   $./test5 srcfile intfile

test5.c

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<errno.h>
int main(int argc, char *argv[]) {
    char cline[67];
    char label[9];
    char others[58];
    char *cptr = NULL;
    FILE *rfd, *wfd; // read & write file descriptor
    if(argc != 3) {
        printf("Usage: %s srcfile \n", argv[0]);
        exit(0);
    }
    if((rfd = fopen(argv[1], "r")) == NULL) { // srcfile open
        fprintf(stderr, "%s %s: cannot open for reading: %s\n", argv[0], argv[1], strerror(
            errno));
        exit(0);
    }
    if((wfd = fopen(argv[2], "w")) == NULL) {
        fprintf(stderr, "%s %s: cannot open for reading: %s\n", argv[0], argv[1], strerror(
            errno));
    }
    while(fgets(cline, 67, rfd) > 0) { // get a line from srcfile
        cptr = cline;
        if(*cptr != ' ' && *cptr != '\'' && *cptr != '\t') { // if label exist
            int c = 0;
            while(*cptr != '\n') {
                if(*cptr == ' ' && c == 0) {
                    fprintf(wfd, ",");
                    c++;
                }
                fprintf(wfd, "%c", *cptr++);
```

```
                }
                fprintf (wfd, "\n");
            }
        }
        fclose (rfd); // srcfile close
        fclose (wfd);
        return 0;
}
```

srcfile

```
        start   1000
first   lda     seven
        sta  alpha
        lda  two
        add  incr
        sta  beta
        lda  gamma
        sub  two
        sta  delta
        ldch  charx
        stch  cha
        jsub  2
seven  word 7
two word 2
alpha  resw 1
beta  resw 1
gamma word 10
delta  resw 1
incr  word 3
cha  resb  1
charx  byte  78
        end  first
```

2. 1의 내용을 확장하여 기호표 SYMTAB[]을 생성하는 프로그램을 작성하고 test하시오.

   이미 지난 번에 낸 리포트에서 symtab을 컴파일러를 구현한 부분은 제출했으므로, 이번 시간에 배운 링크 부분을 매우 조악하게나마 구현한 것을 올립니다. 현재 구현한 링커에서는 다음과 같은 제한 사항이 존재합니다.

   - 목적파일의 이름이 반드시 숫자.o의 형식이어야 한다.
   - 프로그램 시작 번지가 위의 파일의 숫자보다 커야 한다.
   - 한 파일에서 서브루틴을 부를 때에 위의 숫자를 오퍼란드로 해야 한다.
   - 링커의 명령어 라인 변수 중 첫번째가 메인 함수 진입점이 된다.

   지난 시간까지 기존에 만들어 둔 소스를 최대한 적게 고치기 위해 임기응변으로 한 것입니다. 학습용으로 구현한 것이기에 사용자 편의성은 크게 신경쓰지 못함을 이해해 주시기 바랍니다.

헤더파일

```
#pragma once
#include<iostream>
#include<vector>
#include<map>

class Module
{
public:
    friend class Linker;
    Module(int module_num);

protected:
    Module() {}
    short module_num, start, data, end;
    std::vector<std::pair<short, std::string>> lines;
};
```

```
class Linker : public Module
{
public :
    void operator+=(Module m);
    void link () ;

protected :
    std :: map<short, short> module_address;

private :
    void write_file () ;
    short offset = 0;
};
```

구현부

```
#include"link.h"
#include<fstream>
#include<iomanip>
#include<string>
#include<sstream>
using namespace std;

void Linker :: operator+=(Module m)
{
    if ( lines .empty()) {
        module_num = m.module_num;
        start = m. start ;
        data = m.data;
        end = m.end;
    } else {
        offset = end + 1 − m.start ;
        module_address[m.module_num] = end + 1;
        end += 1 + m.end − m.start ;
    }
    for(auto& a : m.lines ) {
        if (a. first == m.data) break;
        stringstream ss ;
        ss << a.second;
        char c [2];
        ss >> c[0] >> c[1];
        short operand;
        ss >> hex >> operand;
```

```
            ss.clear();
            ss << c[0] << c[1];
            if(operand >= start) {
                operand += offset;
                ss << setfill('0') << setw(4) << hex << operand;
            } else {// 매우 제한적이지만 오브젝트파일은 숫자로 이름을
                지어야 한다.
                // 그리고 그 파일명이 스타트 번지보다 작아야 한다.
                // 링커에 첫번째 매개변수로 들어오는 파일이 메인이 된다.
                // 서브루틴은 오브젝트파일의 숫자를 오퍼란드로 한다.
                // 이 부분은 서브루틴이라고 마크하는 부분이다.
                ss << '<' << setfill('0') << setw(2) << hex << operand << '>';
            }
            ss >> a.second;
        }
        for(auto& a : m.lines) a.first += offset;
        lines.insert(lines.end(), m.lines.begin(), m.lines.end());
}

void Linker::link()
{
        for(auto& a : lines) {
            char c[3] {};
            if(a.second[2] == '<') {
                c[0] = a.second[3];
                c[1] = a.second[4];
                short jump_address = module_address[atoi(c)];
                stringstream ss;
                ss << setw(4) << hex << setfill('0') << jump_address;
                a.second.replace(2, 4, ss.str());
            }
        }
        write_file();
}

void Linker::write_file()
{
        ofstream f(to_string(module_num) + ".x");
        f << "start " << hex << start << endl << "data " << hex << data << endl << "end " <<
            hex << end << endl;
        for(auto& a : lines) f << hex << a.first << ' ' << a.second << endl;
}
```

```
Module::Module(int num)
{
    ifstream f( to_string (num) + ".o");
    string s;
    module_num = num;
    f >> s >> hex >> start >> s >> hex >> data >> s >> hex >> end;
    short addr;
    while(f >> hex >> addr) {
        f >> s;
        lines .push_back({addr, s});
    }
}
```

실행화일

```
#include"link.h"
using namespace std;

int main(int c, char** v)
{
    Linker lnk;
    for(int i=1; i<c; i++) lnk += Module(atoi(v[i]));
    lnk.link();
}
```

링커를 테스트 하기 위한 소스파일. jsub 2 이 부분이 2.o 서브루틴을 부른다.

소스파일 1.s

```
    start 1000
first lda seven
    sta alpha
    lda two
    add incr
    sta beta
    lda gamma
    sub two
    sta delta
    ldch charx
    stch cha
    jsub 2
seven word 7
```

```
two word 2
alpha  resw 1
beta  resw 1
gamma word 10
 delta  resw 1
incr  word 3
cha  resb  1
charx  byte  78
      end  first
```

목적파일 1.o

```
 start  1000
data  1021
end 1038
1000  001021
1003  0c1027
1006  001024
1009  181033
100c  0c102a
100f  00102d
1012  1c1024
1015  0c1030
1018  e41037
101b  e81036
101e  480002
1021  000007
1024  000002
1027  000000
102a  000000
102d  000010
1030  000000
1033  000003
1036  00
1037  78
1038  00
```

서브루틴이 되는 파일. 현재로서는 실제로 서브루틴의 역할을 하는 소스가 아니라 그냥 메모리에 링크해서 올리기 위한 소스입니다.

소스파일 2.s

```
      start      1000
 first      lda  data1
```

```
      add data2
      sta  sum
      lda  data1
      sub  data2
      sta   diff
data1  word 8
data2  word 7
sum resw 1
diff   resw 4
      end  first
```

목적파일 2.o

```
start   1000
data  1012
end 1027
1000  001012
1003  181015
1006  0c1018
1009  001012
100c  1c1015
100f  0c101b
1012  000008
1015  000007
1018  000000
101b  000000000000000000000000
1027  00
```

linker.x 1 2 를 실행한 결과물인 링크된 파일

1.o와 2.o를 입력받아 출력된 링크된 파일

```
start   1000
data  1021
end 1060
1000  001021
1003  0c1027
1006  001024
1009  181033
100c  0c102a
100f  00102d
1012  1c1024
1015  0c1030
1018  e41037
```

```
101b e81036
101e 481039
1021 000007
1024 000002
1027 000000
102a 000000
102d 000010
1030 000000
1033 000003
1036 00
1037 78
1038 00
1039 00104b
103c 18104e
103f 0c1051
1042 00104b
1045 1c104e
1048 0c1054
104b 000008
104e 000007
1051 000000
1054 000000000000000000000000
1060 00
```

두번째 목적파일이 첫번째 목적파일의 위에 0x1039부터 이어져 있고, 오퍼란드의 주소가 모두 오프셋에 맞게 변경되어 있다. 또한, 0x101e번지에서 서브루틴을 부르는 부분이 이 시작지점인 0x1039로 제대로 위치를 찾아주고 있다.

소감

이상하게도 실습의 진도와 수업의 진도가 묘하게 엇박자가 난다. 수업시간에 배운 것을 그 다음 실습 시간에 하는 것이 정상이고, 학생들의 실력향상에도 도움이 되지 않을까 하고 생각한다.