SIC 어셈블러 프로젝트

2019년 2학기 시스템 소프트웨어

시스템 소프트웨어 월678

-팀원 소개

1. 구성원

2. 역할

역할			

3. SIC Assembler

선택 : (1) SIC 2-pass 어셈블러

- 명령어 Instruction table : 구조체로 선언

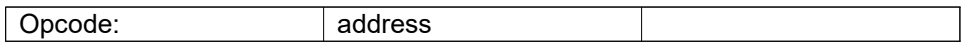
```
struct opcode_table {
        char mnemonic[7];
        int opcode;//16진수
};
struct opcode_table opt[] = {
        {"ADD",0x18}, {"AND",0x40},{"COMP",0x28},{"DIV",0x24},{"J",0x3c},{"JEQ",0x30},
        {"JGT",0x34}, {"JLT",0x38},{"JSUB",0x48},{"LDA",0x00},{"LDL",0x08},{"MUL",0x20},
        {"OR",0x44}, {"RSUB",0x4c},{"STA",0x0c},{"STL",0x14},{"STX",0x10},{"TD",0xe0},
        {"TIX",0x2c},{"LDX",0x04},{"STCH",0x54},{"WD",0xdc},{"RD",0xd8},{"LDCH",0x50}
};
```

- 한번에 처리할 수 있는 최대 용량(MAX_LINE) : 100, 구조체의 크기[100]

- Starting address

input파일의 첫 번째 라인의 operand부분 참조, 만약 null이면(지정하지않으면) 0x1000 에서 시작

- Instruction format(SIC와 동일)



0 8 16 23 -(3byte)

- Symbol table : 구조체로 선언

label값과 location을 저장

```
struct symbol_table{
        int location;
        char symbol[7];
};
```

- 어셈블러 아키텍처 결정

: Two pass 어셈블러

- 검색 알고리즘

: 반복문을 통한 선형 검색

- 데이터 구조

: 구조체 배열 구조, txt파일의 tab기준 7글자로 나뉘기 때문에 각 no, label, opcode, operand배열의 크기는 7

```
struct statement {
        char no[7];
    char label[7];    //MAX_length = 6, [6]은 null문자
        char opcode[7];
    char operand[7]; //대문자
    int location;
};
```

- Addressing mode 표기방법

sic와 동일, symbol테이블에 기록된 location을 찾아쓴다.

40 Null J CLOOP

Label	Location
CLOOP	0x1003

생성된 object code : 3c1003

- 지시자 명령어

파일의 시작에는 START가 끝에는 END를 사용해야한다.(sic와 동일)

BYTE명령어의 경우 X'05' 또는 C'EOF'와 같이 사용가능하다.

X는 hex C는 character형을 의미하며, object코드로는 X'05' : ^05^, X'EOF' : ^454f46^ 과 같이 나타난다.

```
struct opcode_table directive[] = {
        {"START"},{"END"},{"BYTE"},{"WORD"},{"RESB"},{"RESW"}
};
```

- Objects Codes 구조

2019 SS Lecture Slides – 3 – Assemblers.pdf 의 object program format을 참고

- The Object Program Format
  - Header record:
    - COL. 1 H
    - COL. 2-7 Program Name
    - COL. 8-13 Starting address of object program (hexadecimal)
    - COL. 14-19 Length of object program in bytes (hexadecimal)
  - Text record:
    - COL. 1 T
    - COL. 2-7 Starting address for object code in this record (hexadecimal)
    - COL. 8-9 Length of object code in this record in bytes (hexadecimal)
    - COL. 10-69 Object code, represented in hexadecimal (2 columns per byte of object code)
  - End record:
    - COL. 1 E
    - COL. 2-7 Address of first executable instruction in object program (hexadecimal)

- 에러 메시지

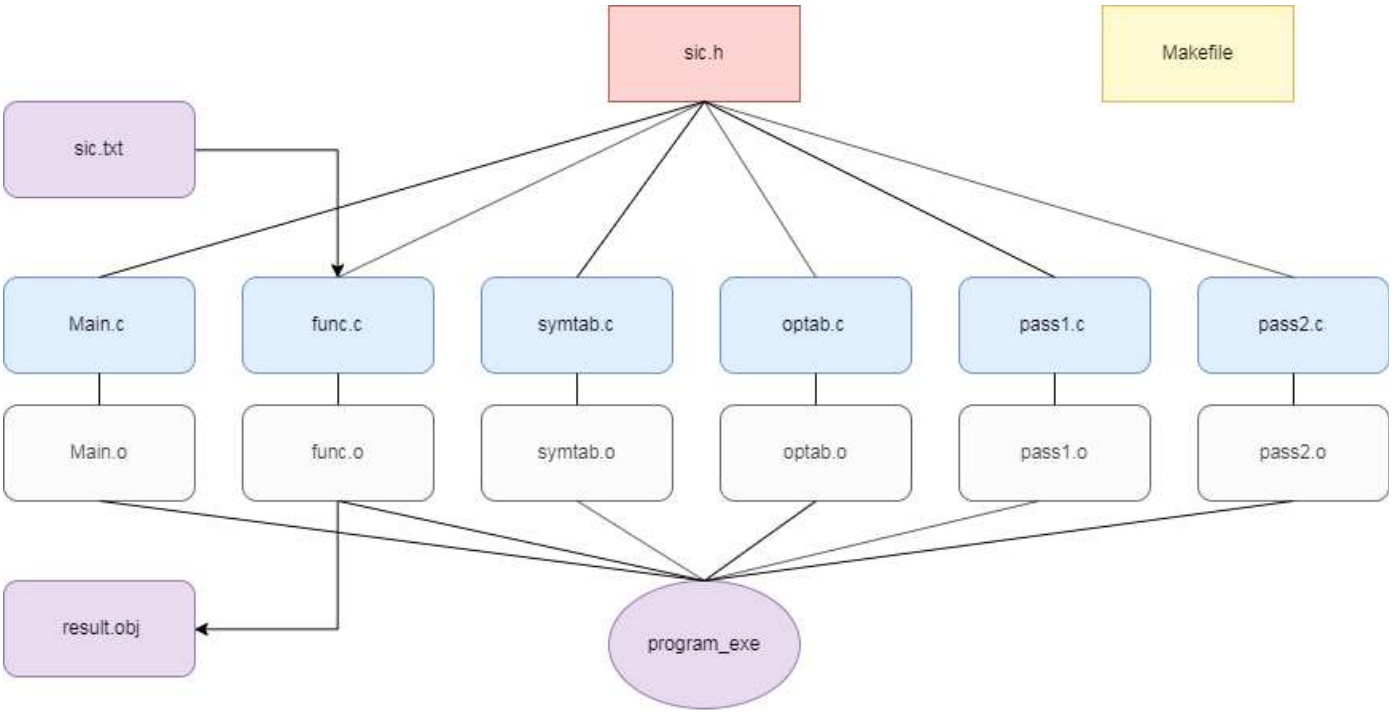
C언어의 전처리기 지시자를 통한 에러구문 처리

```
#define ERROR(error) printf("ERROR – %s(%d)\n",error, i)
```

에러 목록 예시.

```
        ERROR("CAN'T NOT FOUND FILE");
        ERROR("FILE START ERROR ...");
        ERROR("SYMTAB OVERLAP");
        ERROR("NOT FOUND OPCODE");
        ERROR("Undefined Symbol");
        ERROR("You must use type(CorX)");
```

4. 파일 구조



Program_exe :make명령어를 통해 생성된 실행파일

sic.txt : 입력파일

result.obj : pass2가 끝나고 objectcode가 기록되는 파일(최종 목적 파일)

Makefile : linux의 make명령어를 위한 파일

Main.c : read, print, 각 pass1, pass2를 수행하는 파일

func.c : 어셈블러에서 pass1, pass2를 실행시키기 위해 요구되는 함수들,
문자열을 10진수 또는 16진수로 변환해주는 함수, 파일읽기함수, byte처리함수 등
read함수 : 입력txt파일을 읽어와 statement구조체에 저장, 입력 파일이 몇줄인지 반환한다.
todec : 문자열에 대해 10진수로 변환, int(10) 으로 반환한다.
tohex : 문자열에 대해 16진수로 변환, int(10) 으로 반환한다.
byte_conversion : pass2에서 object code를 생성하기위해 byte지시자의 operand를 처리해준다.
(X'05', C'EOF')등..

pass1.c : sic어셈블러의 pass1을 수행하는 함수, 시작주소와 파일이름을 기록한다.
또한 각 statement마다location을 지정한다. symbol_table을 작성한다.

pass2.c : sic어셈블러의 pass2을 수행하는 함수, statement를 한줄 씩 다시 읽으면서 object code를
생성한다. 최종 결과물로 result.obj라는 파일을 생성한다.

<u>- 부가설명</u>

텍스트 레코드 기록시, 한 라인당 길이는 최대 69이고 넘어갈 경우 새로운 텍스트 레코드를 생성한다.
이를 위해 현재줄에서 위치를 기억하기 위해 line_len을 사용한다.
fp_len은 파일 포인터의 위치를 기억하는 변수로, 하나의 텍스트 레코드가 끝나면 해당 텍스트 레코드의
길이를 나타내는 부분으로 돌아가 길이를 기록해주기위해 사용한다.

optab.c : opcode_table저장, 검색, 출력 등을 담당하는 파일

struct opcode_table opt : mnemonic와 opcode가 들어있다. (24개의 미리정의한 sic opcode)

struct opcode_table directive : 지시자가 들어있다. (START, END, BYTE, WORD, RESB, RESW)

check_optab : pass1에서 optab에 해당 opcode가 존재하는지 검사, 있다면 1 없다면 0을 반환한다.

search_optab : pass2에서 해당 명령어의 opcode를 찾아서 반환해준다.

(LDA를 주면 0x00을 반환한다.)

symtab.c : symbol_table에 기록, 검색, 출력 등을 담당하는 파일

struct symbol_table st : location과 symbol이 들어있다. (location은 pass1수행시에 값이 들어간다.)

check_symtab : pass1에서 symtab에 해당 symbol이 존재하는지 검사한다.

있다면 1 없다면 0을 반환한다.

search_symtab : pass2에서 해당 symbol의 location을 찾아서 반환해준다.

add_symtab : symtab에 추가, pass1에서 해당 label에 symtab에 없다면, 이 함수를 실행한다.

sic.h : 함수와 구조체가 정의된 헤더파일

#define ERROR(error) printf("ERROR - %s(%d)\n",error, i) : 에러처리를 위한 전처리기

struct info info : 파일의 이름, 시작 주소, 파일 길이를 기록하는 구조체이다.

struct statement s : 파일을 읽어올 때, no, label, opcode, operand를 저장하는 구조체이다.

pass1수행시에 location값도 채워넣는다.

/* 사용하는 파일에 대한 함수 선언 */

5. 입력 파일(sic.txt)

2019 SS Lecture Slides - 3 - Assemblers.pdf 의 sic 입력 예시를 참고

[라인번호, 라벨, OPCODE, OPERAND]로 구성되며

.txt파일의 첫줄의 라벨에는 프로그램의 이름이, OPCODE에는 START가 OPERAND에는 시작주소가 들어간다.

또한 파일의 마지막줄의 OPCODE에는 END를 사용한다.

해당 데이터가 존재하지않을 경우에는 'null' 이 들어간다.

```
5       COPY    START   1000
10      FIRST    STL     RETADR
15      CLOOP   JSUB     RDREC
20      null     LDA     LENGTH
25      null     COMP    ZERO
30      null     JEQ     ENDFIL
35      null     JSUB     WRREC
40      null     J       CLOOP
45      ENDFIL   LDA     EOF
50      null     STA      BUFFER
55      null     LDA     THREE
60      null     STA      LENGTH
65      null     JSUB     WRREC
70      null     LDL     RETADR
75      null     RSUB     null
80      EOF      BYTE     C'EOF'
85      THREE    WORD    3
90      ZERO     WORD    0
95      RETADR   RESW     1
100     LENGTH   RESW     1
105     BUFFER    RESB     4096
125     RDREC    LDX     ZERO
```

```
130         null        LDA         ZERO
135         RLOOP     TD          INPUT
140         null        JEQ         RLOOP
145         null        RD          INPUT
150         null        COMP      ZERO
155         null        JEQ         EXIT
160         null        STCH       BUFFER
165         null        TIX         MAXLEN
170         null        JLT         RLOOP
175         EXIT        STX         LENGTH
180         null        RSUB       null
185         INPUT      BYTE       X'F1'
190         MAXLEN WORD     4096
210         WRREC    LDX        ZERO
215         WLOOP    TD         OUTPUT
220         null        JEQ         WLOOP
225         null        LDCH      BUFFER
230         null        WD         OUTPUT
235         null        TIX         LENGTH
240         null        JLT         WLOOP
245         null        RSUB       null
250         OUTPUT   BYTE       X'05'
255         null        END        FIRST
```

6. 출력 파일(result.txt)

2019 SS Lecture Slides – 3 – Assemblers.pdf 의 object program format을 참고

linux명령어 cat을 사용하여... ~$ cat result.txt

기록된 object_code를 확인할 수 있다.

```
H^COPY   ^001000^00107a
T^001000^1e^141033^482039^001036^281030^301015^482061^3c1003^00102a^0c1039^00102d^
T^00101e^1e^0c1036^482061^081033^4c0000^454f46^000003^000000^041030^001030^e0205d^
T^002042^1b^30203f^d8205d^281030^302057^541039^2c205e^38203f^101036^4c0000^F1^
T^00205e^1b^001000^041030^e02079^302064^501039^dc2079^2c1036^382064^4c0000^05^
E^001000
```