숭실대학교 Soongsil University

시스템프로그래밍 2021 보고서

보고서 제출서약서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
   1.1. 나는 동료의 보고서를 베끼지 않았습니다.
   1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히 인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)
3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.
4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

과목	시스템프로그래밍 2021
과제명	어셈블러 구현 (프로젝트#1C)
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20162449 김상현 (출석번호 105번)
제출일	2021년 05월 23일

차 례

1장　프로젝트 동기/목적

SIC/XE 소스를 C언어를 이용하여 Object Program Code로 변환하여 봄으로써 SIC/XE 머신 어셈블러의 동작을 이해한다. 이해한 기본 틀을 가지고 JAVA를 통해 프로젝트 1B에서 구현하였다. 이번 프로젝트 1C에서는 파이썬을 이용하여 어셈블러를 구현해 본다.

Python 특성상 JAVA로 구현했던 SIC/XE 머신 어셈블러와 크게 다르지 않았고 기존의 코드를 많이 참고했음을 밝힌다.

2장　설계/구현 아이디어

설계 구현 요약

시작 전 주어진 inst.data (appendix)를 instTable 에 저장한다.
또한 주어진 input.txt를 한 줄 씩 lineList 에 저장하였다.

pass1 에서는 저장했던 lineList에 있는 한 줄 씩 토큰 분석을 진행한다. 각 조건을 부여하여 SymbolTableList와 LiteraltableList를 구성하고, 명령어 줄을 파싱한 값을 TokenTableList에 저장한다.

만들어진 symboltabList 와 literaltabList를 각각 symboltabList.txt.와 literaltabList.txt 파일에 저장한다.

pass2 에서는 tokenTable을 가지고 nixbpe 값을 비트 연산을 통해 저장한다. 또한 주어진 opcode 와 nixbpe , 주소 계산을 통한 addr을 합하여 objectcode로 만들고 output에 작성할 codeList를 object code에 맞춰서 삽입힌다.

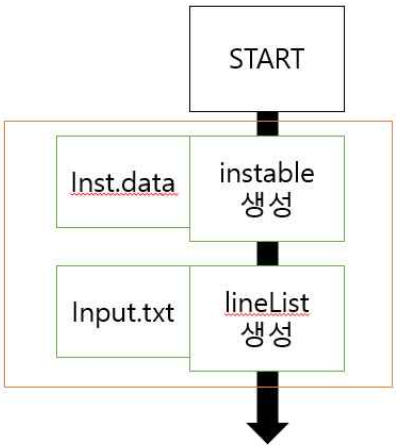
마지막으로 codeList를 기반으로 output file 에 들어갈 object code program을 작성한다.

Flow chat :

1. Pass_1 수행 전에 준비 과정이다.

● InstTable 생성 :
- 파이썬의 dictionary로 구현 ex) {key : value}

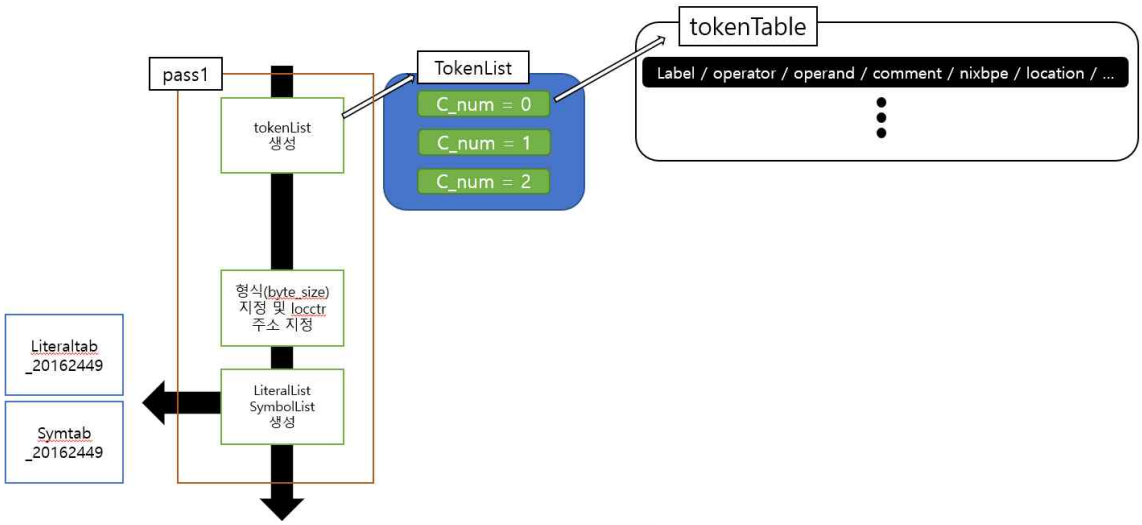
● LineList 생성 :
- List 형태로 구현
- make_line_list(filename) :
　　한 줄 씩 읽어들이고 '.'으로 시작하는 명령어줄을 삭제한다.　　현재 코드에서는 같은 코드가 두 번 들어간 것을 볼 수 있는데　　이는 이후에 4. 결론 및 아쉬운 점에 기술하겠다.

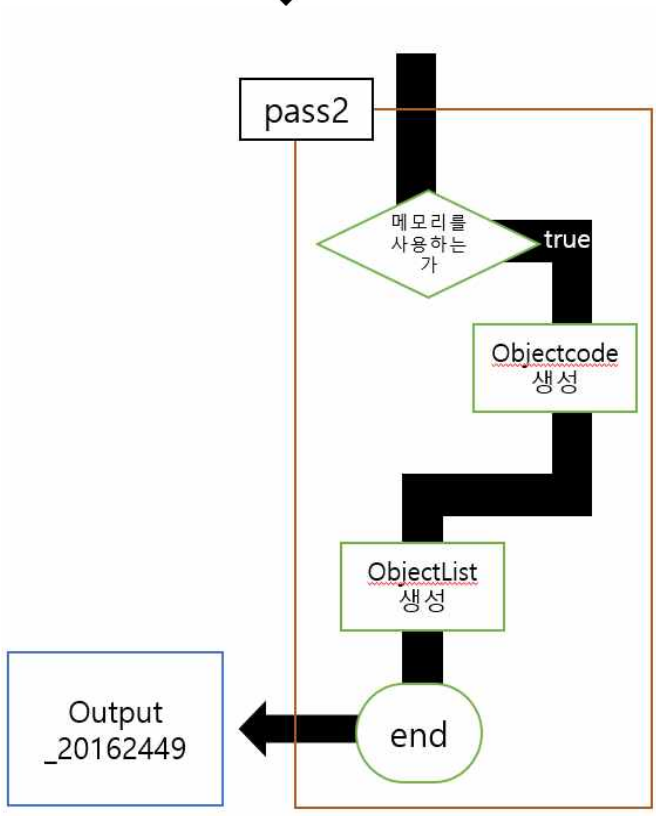


2. PASS 1 :

- 파이썬의 dictionary 형태로 구현
- Token 은 label / operator / operand / comment / locctr / byteSize / ObjectCode / nixbpe로 구성
- extTab 생성 및 저장 (extref , extdef
- 각 토큰마다의 byteSize를 저장하고 locctr을 저장하여 SymbolTab과 LiteralTab에 저장 한다.

3. PASS 2 :





● Objectcode 생성 .

각 token의 opcode를 받고 nixbpe를 설정한다. opcode를 완성한다.

● codeList 생성

만들어진 objectcode를 가지고 주어진 포맷에 맞추어 codeList를 만든다.

각 명령어줄에는 objectcode를 가지는데 예외로 메모리 할당이 필요없는 명령어 줄에서는 objectcode를 만들지 않는다.

만약 다음 줄에 objectcode 가 빈 명령어 줄이 중간에 나온다면 output 파일에서는 개행이 필요하다.

상세 구현 내용 (전체 설계 내용 / 중요 함수 및 모듈 설명)

기존의 JAVA언어로 구현한 project1B와 달라진 점은 기존의 instMap대신 딕셔너리로 instDic을 생성한다. 또한, SymbolTable, LiteralTable을 딕셔너리로, TokenTable은 리스트로 선언하여 control section별로 관리한다. 추가적으로 extdef, extref 리스트를 포함하는 ExtTable을 선언하여 사용한다.

pass1에서는 어셈블리어 코드를 한 줄 씩 토큰으로 분리하고, 이를 통해 각 control section 별로 사용된 symbol과 literal에 주소를 할당하여 저장한다. 또한 각 명령어 줄에 주소를 할당하고 nixbpe 값을 비트 연산으로 설정하며 생성될 objectcode의 크기를 기록한다.

Pass2에서는 pass1에서 할당된 주소를 통해 object code를 생성한다. operator에 따라서 명령어의 형식을 구분하고, nixbpe 비트 값, 피연산자의 주소 등을 기반으로 object code를 생성한다. 또한, 이후 ouput파일에 작성할 codeList를 object code에 맞춰서 작성한다.
모든 pass 종료 후 codeList 내용을 기반으로 output파일에 object code program을 작성한다.

def pass1()

구현하고자하는 two pass 어셈블러에서의 pass1은 symbol들과 literal들에 주소를 할당하고 지시어들을 처리한다. 명령어 각 한줄 씩을 매개변수로 하여 TokenTable의 매소드인 putToken을 호출한다. 각 TokenTable은 symbolTable과 literalTable을 section number를 기준으로 포함한다. 따라서 symboltabList 와 literaltabList를 통해 symbolTable 과 literaltable을 생성하고 연결한다.
이후 putToken을 호출하는데 이때 리턴 값이 음수라면 (control section 변경 의미) 새로운 TokenTable을 만들고 위의 과정을 반복한다.

def pass2()

TokenTable에 저장된 각 명령어의 주소와 SymbolTable, LiteralTable, ExtTable를 통해 object code를 만들어 codeList에 저장한다. (makeObjectCode() 사용)
순서대로 섹션(프로그램)의 이름과 시작주소 총 크기를 포함하는 'H' Header 와 extTable안에 저장되어 있는 extdef 와 extref로 'D' define , 'R' reference를 codeList에 저장한다.
이제 'T' text 부분이다. 시작주소 , 현재 줄 바이트 수 , objectcode 형식으로 각 라인이 이루어지는 Text record 는 한 줄에 최대 들어갈 수 있는 바이트 수는 32이고 명령어 objectcode가 이어지지 않게 되면 개행이 이루어진다. TokenTable 의 makeObjectCode 메소드는 주어진 tokenList를 사용하여 nixbpe(n,i는 opcode + x,b,p,e 는 addrflag로 구현) 를 설정하고 instTable이 get_opcode 메소드를 사용하여 opcode를 가져오고 각 symbolTable과 literalTable의 정보를 사용하여 타겟주소의 주소를 정하고 program counter(다음 명령어의 주소)를 구하여 pc realitive 계산하여 주소를 정한다. 완성된 objectcode 는 위의 조건에 벗어나지 않을 때 까지 str 에 임시로 쌓고 위의 조건에 부합하면 위 str을 포함하는 Text recode 한 줄을 저장하고 str을 초기화하고 위의 과정을 반복한다.
이후 modificationList를 확인하여 추가해야할 부분이 있으면 'M' modification을 추가하여 저장한다

TokenTable: int putToken(String line)

lineList에서 저장된 어셈블리어 코드 한줄을 받아 parsing 한다. 또 CSECT 와 같은 section을 분리하는 operator 가 확인되면 -1을 반환하여 새로운 TokenTable을 pass1에서 생성할 수 있게 한다.

section에 따라 분리되는 TokenTable 은 각 instTable, SymbolTable 과 LiteralTable, ExtTable(extref[],extdef[]),modificationList, locctr 등 필요한 정보들을 포함한다.

putToken() 이 실행되면 Token객체로 주어진 line을 파싱한 값을 저장하는데 이때 Token객체는 operatort, operand, comment 로 각각 line을 분리한다.

각 instrction의 byteSize를 저장하고 해당 크기만큼 locctr을 증가시킨다. 각 line 별 locctr을 그 instruction의 location으로 정의한다.

Label이 존재하는 명령어 라인에 대해서는 label과 그 주소(location)을 symbolTable 에 key와 value로 (딕셔너리 형태이다) 저장한다. 또 operand 부분에 '=' 이 사용된 Literal가 존재하면 이를 literalTable에 operand 값과 주소를 key와 value 로 저장한다.

TokenTable: void makeObjectCode(int index)

각 TokenTable의 저장된 정보 (위에 putToken을 통해 만들어진)를 가지고 objectcode를 만든다. biteSize를 확인하여 2형식 3형식 4형식 혹은 기타 지시어를 파악한다.

operator를 기준으로 보면 일반적인 2/3/4형식 operator 가 아닌 'Word/Byte' 등 인 경우 메모리를 사용하지만 opcode가 존재하지 않는다. 이 때는 해당 Token의 operand 값을 object code로 생성한다.

2형식 operator 는 각 레지스터의 고유번호를 지정하고 해당 operator의 opcode를 불러온뒤 ("%X%X%X",%(opcode,reg1,reg2)) 형식으로 object 코드로 만든다.

3형식 operator 는 먼저 nixbpe를 먼저 설정한다. n = 1, I =1을 기본으로 가지고 X가 operand[1]로 나올때는 x = 1 로 설정하고 pc realtive를 성립한다면 p=1로 설정한다.
@(indirect) 나 #(immediate) 가 operand에 나온다면 각각 조건에 맞게 n과 I, p를 설정한다.

4형식 operator 또한 nixbpe를 설정하는데 n과 I는 모두 1이고 e가 1이 된다.

이제 완성된 각 nixbpe를 opcode 부분과 addflag로 분리하여 생각한다. 왜냐하면 ni 부분은 opcode 와 합쳐져야하기 때문이다.

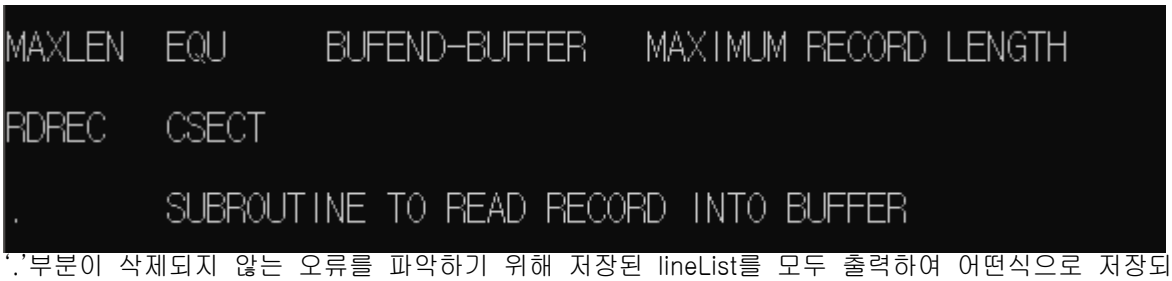
마지막으로 pc realative 계산을 하여 Target 주소와 program countr 의 차 연산을 하여 addr 부분을 저장한다. 이 때 addr 이 음수라면 보수를 통해 처리한다.

완성된 opcode(opcode + n, I) 와 addflag(x,b,p,e) , addr을 "%02X%X%03X",%(opcode,addflag,addr) 형식의 objectcode 로 저장한다. 물론 4형식은 "%02X%X%05X",%(opcode,addflag,addr) 가 될 것이다.
(코드에서는 10진수로 처리하다가 16진수로 변환하는 과정에서 addr이 string 형식이 되었고 이에따라 %05X, %03X가 아닌 addr.zfill(5) , addr.zfill(3) ㄹ 처리하였다.)

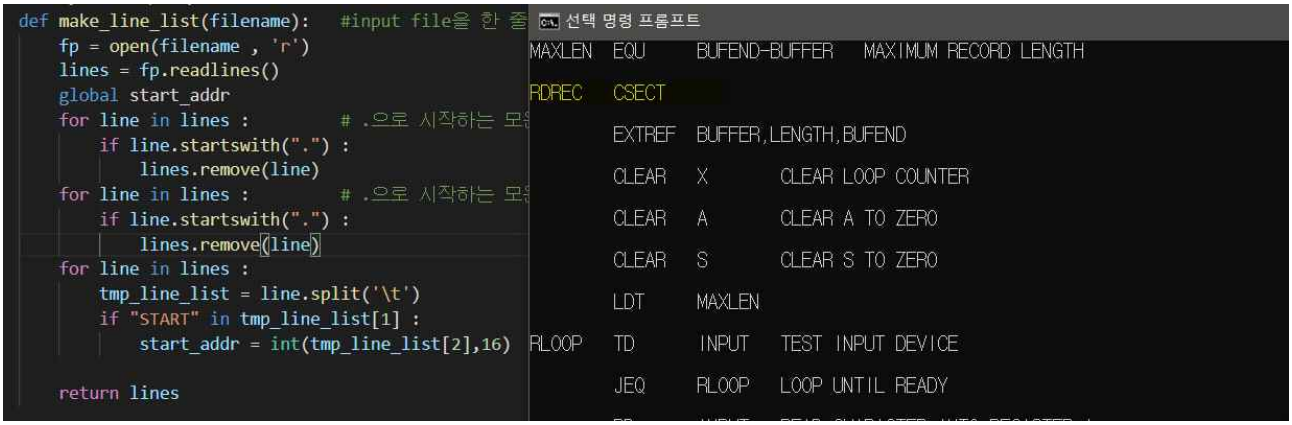
objectcode를 만드는 과정에서 extref 로 선언된 symbol이 사용된 경우 modificationList에 값을 변경해야 하는 주소, 변경할 값의 개수, 변경 시 수행할 연산자, 변경할 피연산자 이름 정보를 저장한다.

3장 수행결과(구현 화면 포함)

디버깅 화면

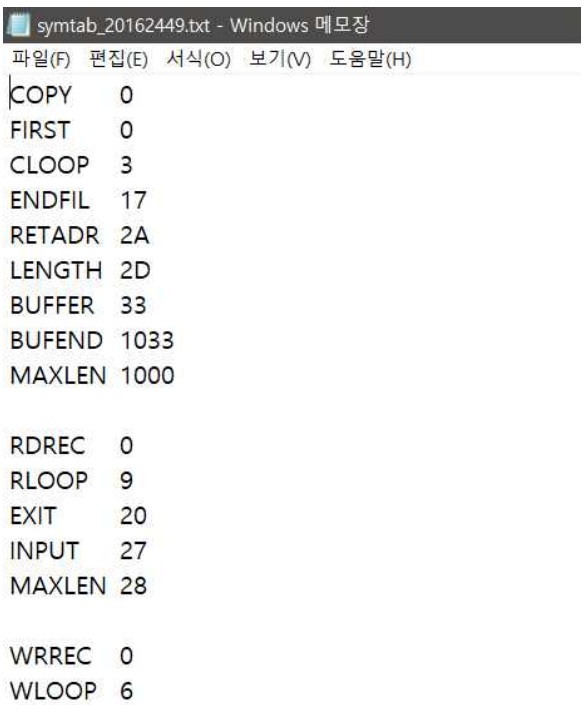


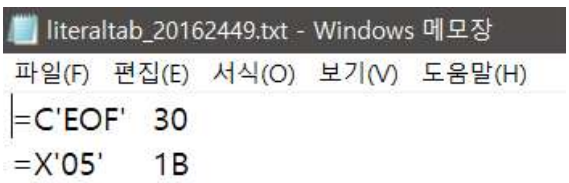
'.'부분이 삭제되지 않는 오류를 파악하기 위해 저장된 lineList를 모두 출력하여 어떤식으로 저장되는지 print를 통한 디버깅

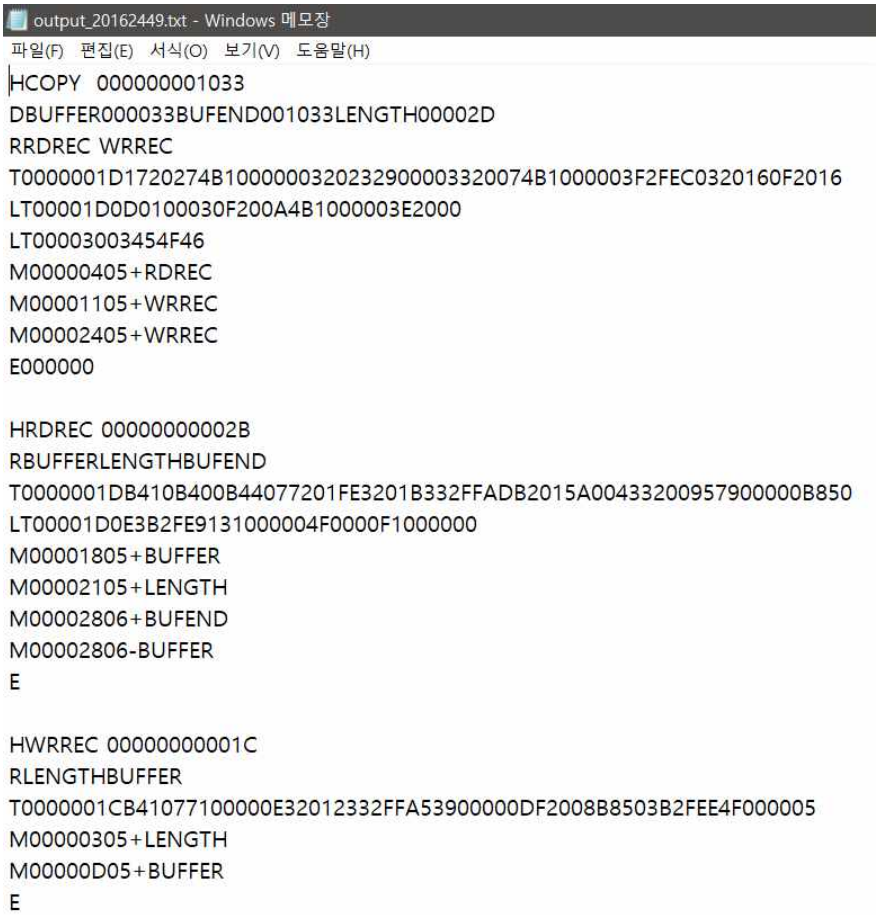


디버깅 이후 . 으로 시작하는 부분에 대한 삭제를 두 번 진행하는 걸로 수정하였음.

결과 화면







4장 결론 및 보충할 점

어셈블리어를 다루고 SIC/XE 머신을 총 3가지 버전으로 만들어 보며 전체 적인 흐름에 대해 완벽하게 이해했고 처음에는 하지 못했던 몇가지의 예외처리 또한 해냈다. 또 처음 C 언어로 구현했을 때는 완벽하게 구현하지 못하였고 전체적인 흐름과 이해에 문제가 있다고 생각하여 다시 공부를 진행하였고 다행히 JAVA로 구현했을 때는 오히려 JAVA가 개인적으로 더 어려운 언어 였음에도 구현하는데 성공하였다. 또 파이썬 특성상 JAVA의 코드를 기반으로 할 수 있었기에 조금더 수월하게 할 수 있었던거 같다.

마지막에 꼭 언급하고 싶었던 부분이 있어서 짚고 넘어가보려고 한다. 처음 LineList를 만들기 위해 readlines()를 하는 부분에서 '.'으로 시작하는 line은 저장하지 않게 코드를 추가하였는데 전체 코드를 완성한 후에 확인 하였을 때 .으로 시작하지만 뒤에 설명이 추가 되어있는 어셈블리어 line 에 대해서는 제대로 삭제가 되지 않는 모습이 보였다. 그리하여 output이 계속 바뀌었고 여러 디버깅을 통해(visual code로 작업하였기 때문에 디버깅이 원활하지 않아 사실 print()를 사용하여 노가다로 디버깅하였다.) '.' 으로 시작하고 comment 부분이 추가 되어있는 어셈블리 명령어의 경우 뒤의 comment 가 해당 토큰의 operator 로 저장되었고 이에 삭제가 되지 않는 다고 판단했다. 기존의 코드를 수정하지 않고 해당 오류를 하기 위해서 그저 '.' 으로 시작하는 line 삭제를 위한 코드르 2번 반복하였더니 해결되었다. 이 해결과정이 정확한 흐름을 파악하여 해결한 것이 아니라 운이 좋게 걸린 느낌으로 해결된 것 같아 이후에 수정이 필요하다고 결론지었다.

5장 소스코드(+주석)

```
def make_line_list(filename):    #input file을 한 줄 씩 읽고 저장하는 함수
    fp =open(filename , 'r')
    lines = fp.readlines()
    global start_addr
    for line in lines :          # .으로 시작하는 모든 line 제거
        if line.startswith(".") :
            lines.remove(line)
    for line in lines :          # .으로 시작하는 모든 line 제거
        if line.startswith(".") :
            lines.remove(line)
    for line in lines :
        tmp_line_list = line.split('₩t')
        if "START"in tmp_line_list[1] :
            start_addr =int(tmp_line_list[2],16)
    return lines
class InstTable:
    def __init__(self):
        self.instDic = {} #dictionary { inst name : line }

    def openFile(self, fileName):        ### inst.data file 오픈하여 파싱.
        f =open(fileName,'r')
        instructions = f.readlines()
        for line in instructions:
            instruction = line.split(' ') #' '을 기준으로 파싱
            self.instDic[instruction[0]] = instruction
    def get_format(self,op): #파라미터의 형식을 정수로 반환
        if op[0]=='+':
            op = op[1:]
            return 4
        if self.instDic.get(op)==None :
            return -1
        elif "3/4"in self.instDic[op][1]:
            return 3
        elif "2"in self.instDic[op][1]:
            return 2
    def get_opcode(self,label):  #label을 기준으로 opcode 반환
        if '+'in label:
            label = label[1:]
        if self.instDic.get(label) ==None:
            return -1
        else:
            return int(self.instDic[label][2],16) #op_code를 16진수 정수로 변환하여 반환

class SymbolTable:      #딕셔너리 구조로 symbolTable 구성
    def __init__(self):
        self.symbolDic = {}

    def putSymbol(self,symbol,location):  # symbol table에 주어진 symbol 을 삽입
        if(self.search(symbol)<0):          #이미 symbol table에 있을시
            self.symbolDic[symbol] = location #딕셔너리 key : symbol , value : location

    def search(self,symbol):       #symbolTable 검색
        if(self.symbolDic.get(symbol)==None):
            return -1
        else:
            return self.symbolDic[symbol]

    def size(self):
        return len(self.symbolDic)
class LiteralTable:      #딕셔너리 구조로 LiteralTable 구성
    def __init__(self):
        self.litDic = {}

    def putLiteral(self,literal,location): # literal table 삽입
            self.litDic[literal] = location # 딕셔너리 key : literal , value : location


    def search(self,literal):   #literal table 검색
        if(self.litDic.get(literal)==None):
            return -1
        else:
            return self.litDic[literal]

    def size(self):
        return len(self.litDic)
class ExtTable:    #두개의 리스트 구조로 ExtTable 구성
    def __init__(self):
        self.extdef = []      #각각 define 리스트, reference 리스트로 구성
        self.extref = []

    def addD(self,symList): #extdef 리스트에 삽입
        self.extdef = symList

    def addR(self,symList): #extref리스트에 삽입
        self.extref = symList

    def searchD(self,symbol): #extdef에서 탐색
        for s in self.extdef:
            if(s == symbol):
```

```
                return self.extdef.index(s)
        return -1

    def searchR(self,symbol): #extref에서 탐색
        for s in self.extref:
            if(s == symbol):
                return self.extref.index(s)
        return -1
class Token :
    def __init__(self,tokens) :
        #  {  label,  operator,  operand[],
comment, locctr, byteSize, Objectcode, nixbpe
}
        self.label = tokens[0]
        self.operator = tokens[1].rstrip('₩n')
        if len(tokens)>2  :
            self.operand                    =
tokens[2].rstrip('₩n').split(',')
        else :
            self.operand =None
        if len(tokens)>3 :
            self.commenet = tokens[3]
        else :
            self.comment =None

        self.locctr =0
        self.byteSize =0
        self.Objectcode =""
        self.nixbpe =[0,0,0,0,0,0]

class TokenTable:
    global start_addr
    def __init__(self,symtab,littab,insttab):  #필
요 테이블 링크
        self.symTab = symtab
        self.literalTab = littab
        self.instTab = insttab
        self.extTab = ExtTable()
        self.locctr = start_addr
        self.size =0
        self.modiLIst = []
        self.tokenList = []
        self.ltorg_flag =False
        self.name =""
        self.modificationList = []
    def putToken(self,line): # 일반 문자열을 받
아서 Token단위로 분리시켜 tokenList에 추가한
다.
        newToken = Token(line.split('₩t'))
        newToken.location =self.locctr
        if "START"in newToken.operator:

self.symTab.putSymbol(newToken.label,start_ad
dr) #symbolTable에 프로그램 이름 저장
            self.name = newToken.label
            return 1
        elif    "CSECT"in    newToken.operator:
#control section 분리
            if not self.ltorg_flag: #LTORG 호출
없을때
                #LTORG 가 없을 때 literal 이
나오게 되면 CSECT 의 operand 부분에 literal
을 임의로 넣음으로써 literal의 loccation을 조정
한다.
                for             lit             in
self.literalTab.litDic.keys():

self.literalTab.putLiteral(lit,self.locctr)
                    if lit[1] =='C' :
                        self.locctr          +=
(len(lit)-4) # ex) =C'EOF' 에서 '=','C',' ', ' '
' 총 4개 삭제
                        newToken.byteSize
=len(lit)-4
                    elif lit[1] =='X' :
                        self.locctr
+=int((len(lit)-4)/2)
                        newToken.byteSize
=int((len(lit)-4)/2)
                    newToken.operand  =  [lit]

                self.ltorg_flag =True
            self.size =self.locctr
            self.tokenList.append(newToken)
            return -1 #음수 리턴
        elif("END"in newToken.operator): #프로
그램 종료
            if not self.ltorg_flag: #LTORG 호출
없을 때
                #LTORG 가 없을 때 literal 이
나오게 되면 END 의 operand 부분에 literal 을
임의로 넣음으로써 literal의 loccation을 조정한
다.
                for             lit             in
self.literalTab.litDic.keys():

self.literalTab.putLiteral(lit,self.locctr)
                    if lit[1] =='C' :
                        self.locctr +=len(lit)-4
                        newToken.byteSize
=len(lit)-4
                    elif lit[1] =='X' :
                        self.locctr
+=int((len(lit)-4)/2)  # ex) =X'05' 에서 '=','X','
' ', ' ' ' 총 4개 삭제 후 16진수 두글자당 한 바
이트기 때문에 /2
                        newToken.byteSize
=int((len(lit)-4)/2)
                    newToken.operand = [lit]
                self.ltorg_flag =True
            self.size =self.locctr
            self.tokenList.append(newToken)
            return 0
        elif   "EXTDEF"in   newToken.operator:
#ExtTable.extdef에 저장
```

```
self.extTab.addD(newToken.operand)
                return 1
            elif "EXTREF"in newToken.operator:
#ExtTable.extref에 저장

self.extTab.addR(newToken.operand)
                return 1
            elif "LTORG"in newToken.operator:
#literalTable에 주소 갱신
                if not self.ltorg_flag:
                    for lit in
self.literalTab.litDic.keys():

self.literalTab.putLiteral(lit,self.locctr)
                        self.locctr +=len(lit)-4
                        newToken.operand = [lit]
                        newToken.byteSize
=len(lit)-4
                    self.ltorg_flag =True
                self.tokenList.append(newToken)
                return 1
            if(newToken.label !=""): #lable을 알맞
은 주소로 symbolTable에 저장
                if("EQU"in newToken.operator):
                    if "*"in newToken.operand[0]:

self.symTab.putSymbol(newToken.label,self.locc
tr)
                        newToken.location =
start_addr
                        newToken.byteSize =0
                    elif (not
newToken.operand[0].isdigit()):
                        addr =0
                        # '-' 일때 minus '+'일 때
plus, 일반 적인 주소연산에서 +,- 를 제외하고
쓰이지 않는다고 가정한다.
                        if "-"in
newToken.operand[0] :
                            newToken.operand =
newToken.operand[0].split('-')
                            addr
=self.symTab.search(newToken.operand[0])-sel
f.symTab.search(newToken.operand[1])
                        elif "+"in
newToken.operand[0] :
                            newToken.operand =
newToken.operand[0].split('+')
                            addr
=self.symtTab.search(newToken.operand[0])+s
elf.symTab.search(newToken.operand[1])

self.symTab.putSymbol(newToken.label,addr)
                        newToken.location =
start_addr
                        newToken.byteSize=0
                else:

self.symTab.putSymbol(newToken.label,self.locc
tr)

            if(newToken.operand !=None):
                if("="in newToken.operand[0]):
#operand 에 literal 사용할 때

self.literalTab.putLiteral(newToken.operand[0],0
)


if(self.instTab.get_opcode(newToken.operator)<
0):   #instTable 에 해당 operator 가 없다면
                if "RESB"in newToken.operator:
                    newToken.byteSize
=int(newToken.operand[0])
                    self.locctr +=
newToken.byteSize
                elif "RESW"in newToken.operator:
                    newToken.byteSize
=int(newToken.operand[0])*3
                    self.locctr +=
newToken.byteSize
                elif "BYTE"in newToken.operator:
                    newToken.byteSize =1
                    self.locctr +=1
                elif "WORD"in newToken.operator:
                    newToken.byteSize =3
                    self.locctr +=3
                    if "-"in newToken.operand[0]
:
                        newToken.operand =
newToken.operand[0].split('-')
                        addr
=self.symTab.search(newToken.operand[0])-sel
f.symTab.search(newToken.operand[1])
                    elif "+"in
newToken.operand[0] :
                        newToken.operand =
newToken.operand[0].split('+')
                        addr
=self.symTab.search(newToken.operand[0])+sel
f.symTab.search(newToken.operand[1])
            else:
                a
=self.instTab.get_format(newToken.operator)
#format 값만큼 bytesize 저장
                newToken.byteSize = a
                self.locctr += a
            self.tokenList.append(newToken)
            return 1
        def makeObjectCode(self,index):
            token =self.tokenList[index]
            opcode
=self.instTab.get_opcode(token.operator)
            addflag =0
            #program countrer 은 다음 명령어의
주소를 가리킨다.
```

```
        if index ==len(self.tokenList)-1:
            pc    =self.tokenList[index].location
+ token.byteSize
        else:
            pc              =self.tokenList[index
+1].location
        addr =0
        token.nixbpe[0] =True
        token.nixbpe[1] =True

        if token.byteSize ==2: #2형식
            r = []
            for i in range(len(token.operand)):
#레지스터 번호 저장
                if "A"in token.operand[i]:
                    r.append(0)
                elif "X"in token.operand[i]:
                    r.append(1)
                elif "S"in token.operand[i]:
                    r.append(4)
                elif "T"in token.operand[i]:
                    r.append(5)
            if len(r) ==1:
                r.append(0)

            token.Objectcode
="%X%X%X"%(opcode,r[0],r[1])
            self.tokenList[index] = token
            return
        str = token.operand[0]
        if opcode <0: #operator 가 insttable
에 없을 때
            if "WORD"in token.operator or
"BYTE"in token.operator:
                if    self.extTab.searchR(str)<0:
# ex) X'F1'
                    str =str[2:-1]
                else:
                    str ="000000"

self.modificationList.append("%06X06+%s₩n"%(
token.location,token.operand[0]))

self.modificationList.append("%06X06-%s₩n"%(
token.location,token.operand[1]))
                token.Objectcode =str
            elif "="in str: #리터럴 ex) =C'EOF'
                if "C"in str:
                    str =str[3:-1]
                    token.Objectcode
="%X%X%X"%(ord(str[0]),ord(str[1]),ord(str[2])
)
                elif "X"in str:
                    str =str[3:-1]
                    token.Objectcode =str
        else: # operator 가 instTable에 정의
되어 있음
            if "+"in token.operator: #4형식
                token.nixbpe[-1] =True
            if not token.operand[0]: #operand
가 없는 operator
                addr =0
            elif self.symTab.search(token.operand[0])>=0:
#symbol 피연산자
                addr =self.symTab.search(token.operand[0])
                addr -=pc
                token.nixbpe[4] =True
            elif self.extTab.searchR(token.operand[0])>=0: #외
부에 정의된 피연산자

self.modificationList.append("%06X05+%s₩n"%(
token.location    +1,token.operand[0]))    #M
record에 저장
                addr =0
            elif self.literalTab.search(token.operand[0])>=0:
#literal 피연산자
                addr =self.literalTab.search(token.operand[0])
                addr -=pc
                token.nixbpe[4] =True
            elif    "#"in    token.operand[0]:
#immediate addressing
                token.nixbpe[0] =False
                addr =int(str[1:])
            elif    "@"in    token.operand[0]:
#indirect addressing
                token.nixbpe[1] =False
                addr =self.symTab.search((token.operand[0])[1:])
                addr -=pc
                token.nixbpe[4] =True
            if token.operand and
len(token.operand)>1 and "X"in
token.operand[1]: #indexing loop 사용
                token.nixbpe[2] =True

            #nixbpe비트 설정에 따른 값 저장
            if token.nixbpe[0]:
                opcode +=2
            if token.nixbpe[1]:
                opcode +=1
            if token.nixbpe[2]:
                addflag +=8
            if token.nixbpe[3]:
                addflag +=4
            if token.nixbpe[4]:
                addflag +=2
            if token.nixbpe[5]:
                addflag +=1

            if addr <0: #pc relative 계산 시 음
수일 경우 보수 처리
```

```
                str = hex((addr + (1 <<12)) %
(1 <<12))
                addr =str[2:]
            else:
                addr = hex(addr)
                addr = addr[2:]

            if      token.nixbpe[5]:      #4형식
objectcode
                token.Objectcode
="%02X%X%s"%(opcode,addflag,addr.zfill(5).up
per()) #4형식 objectcode 저장
            else: #3형식 objectcode
                token.Objectcode
="%02X%X%s"%(opcode,addflag,addr.zfill(3).up
per()) #3형식 objectcode 저장
        self.tokenList[index] = token
d                       e                       f
pass1(symboltabList,literalList,tokentabList,instT
able,lineList):
    tokenTab                                =
TokenTable(symboltabList[0],literalList[0],instTa
ble)
    sec_n =0
    global start_addr
    for line in lineList: #input의 한 줄씩 토큰
분석 진행
        if(tokenTab.putToken(line)<0): #control
section이 분리될때 -1 반환
            tokentabList.append(tokenTab)

symboltabList.append(SymbolTable())
            literalList.append(LiteralTable())
            sec_n +=1
            new = Token(line.split('₩t'))

symboltabList[sec_n].putSymbol(new.label,start
_addr) #새로운 control section의 이름 저장
            tokenTab                        =
TokenTable(symboltabList[sec_n],literalList[sec
_n],instTable)
            tokenTab.name = new.label #새로
운 control section의 이름 저장
    tokentabList.append(tokenTab)
def pass2(tokentabList,instTable):
    sec_n =0
    codeList = []
    global start_addr
    for tokentab in tokentabList:
        current_addr = start_addr
        length =0
        str =""

codeList.append("H%-6s%06X%06X₩n"%(token
tab.name,start_addr,tokentab.size)) # 'H'
        if tokentab.extTab.extdef: # 'D'
            codeList.append("D")
            for             symbol             in
tokentab.extTab.extdef:

codeList.append("%-6s%06X"%(symbol,tokenta
b.symTab.search(symbol)))
            codeList.append("₩n")
        if(tokentab.extTab.extref    !=None):    #
'R'
            codeList.append("R")
            for             symbol             in
tokentab.extTab.extref:

codeList.append("%-06s"%(symbol))
            codeList.append("₩n")
        i=0
        while i <len(tokentab.tokenList): # 'T'
            if      tokentab.tokenList[i].byteSize
!=0:  #메모리  크기를  필요로  하는  명령어만
objectcode 생성
                tokentab.makeObjectCode(i)
            if                              not
tokentab.tokenList[i].Objectcode:      #다음 명
령어의 objectcode 가 없어지만 지금까지의 str을
codeList에 저장

codeList.append("LT%06X%02X%s₩n"%(current
_addr,length,str))
                str =""
                length =0
                while i <len(tokentab.tokenList)
and not tokentab.tokenList[i].Objectcode:
                    i                       f
tokentab.tokenList[i].byteSize !=0:

tokentab.makeObjectCode(i)
                    i                       f
tokentab.tokenList[i].Objectcode:
                        break
                    i +=1
                if i <len(tokentab.tokenList):
                    current_addr                =
tokentab.tokenList[i].location
                    i -=1
            elif                            length
+tokentab.tokenList[i].byteSize >=32: # 한 줄
당 용량 : 32

codeList.append("T%06X%02X%s₩n"%(current_
addr,length,str))  # 한 줄에 들어갈 수 있는 용
량이 꽉 차면 지금까지의 만들어진 str 을 포멧에
맞추어 codeList에 추가
                current_addr += length
                str =""
                length =0
                i -=1
            else:
                str                             +=
tokentab.tokenList[i].Objectcode
                length                          +=
```

```
tokentab.tokenList[i].byteSize
                i+=1
            if str:

codeList.append("T%06X%02X%s₩n"%(current_
addr,length,str))

            if tokentab.modificationList: # 'M'
                for record in
tokentab.modificationList:

codeList.append("M%s"%(record))

            if sec_n ==0: # 'E'
                codeList.append("E000000₩n")
            else:
                codeList.append("E₩n")
            codeList.append("₩n")

            sec_n +=1
        return codeList
if __name__ =='__main__' :
        instTable = InstTable()
        instTable.openFile("inst.data")
        lineList = make_line_list("input.txt")
        for line in lineList :
            print(line)
        symboltabList = [SymbolTable()]
        literaltabList = [LiteralTable()]
        tokentabList = []

pass1(symboltabList,literaltabList,tokentabList,in
stTable,lineList)
        f =open("symtab_20162449.txt",'w')
        for symtab in symboltabList:
            for key,value in
symtab.symbolDic.items():
                f.write("%s₩t%X₩n"%(key,value))
            f.write("₩n")
        f.close()
        f =open("literaltab_20162449.txt",'w')
        for littab in literaltabList:
            for key,value in littab.litDic.items():
                f.write("%s₩t%X₩n"%(key,value))
        f.close()
        """pass2"""
        codeList = pass2(tokentabList,instTable)
        """objectcode가 저장된 codeList내용 파일에
저장"""
        f =open("output_20162449.txt",'w')
        for line in codeList:
            f.write(line)
        f.close()
```