과목명 : 시스템 프로그래밍

담당 교수명 : 김지환

<<Assignment 2>>

서강대학교 컴퓨터공학과

학번: 20171690

이름: 정유석

**목차**

1. 프로그램 개요
2. 프로그램 설명
   1. 프로그램 흐름도
   2. SIC/XE Assembler 흐름도
   3. Symbol Table
3. 분할 c 파일
   1. 20171690.c
   2. shell.c
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : help()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : type()
            1. 기능
            2. 사용 변수
   3. memory.c
   4. opcode.c
   5. assembler.c
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : assemble()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : pass1()
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : pass2()
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : getInstructionSize()
            1. 기능
            2. 사용 변수
         5. 모듈 이름 : getObjCode()
            1. 기능
            2. 사용 변수
         6. 모듈 이름 : printObjCode()
            1. 기능
            2. 사용 변수
         7. 모듈 이름 : tokenizeAsmFile()
            1. 기능
            2. 사용 변수
         8. 모듈 이름 : removeSpaceAroundComma()
            1. 기능
            2. 사용 변수
   6. assembler\_functions.c
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : getRegNum()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : LDB()
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : printLineinLST()
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : isWhiteSpace()
            1. 기능
            2. 사용 변수
         5. 모듈 이름 : isBlankLine()
            1. 기능
            2. 사용 변수
         6. 모듈 이름 : isDirective()
            1. 기능
            2. 사용 변수
         7. 모듈 이름 : isComma()
            1. 기능
            2. 사용 변수
         8. 모듈 이름 : nameToListing()
            1. 기능
            2. 사용 변수
         9. 모듈 이름 : nameToObj()
            1. 기능
            2. 사용 변수
         10. 모듈 이름 : toUpper()
             1. 기능
             2. 사용 변수
         11. 모듈 이름 : toUpperCase()
             1. 기능
             2. 사용 변수
         12. 모듈 이름 : byteSize()
             1. 기능
             2. 사용 변수
         13. 모듈 이름 : wordSize()
             1. 기능
             2. 사용 변수
         14. 모듈 이름 : resbSize()
             1. 기능
             2. 사용 변수
         15. 모듈 이름 : reswSize()
             1. 기능
             2. 사용 변수
         16. 모듈 이름 : addNum()
             1. 기능
             2. 사용 변수
         17. 모듈 이름 : freeNums()
             1. 기능
             2. 사용 변수
         18. 모듈 이름 : printNums()
             1. 기능
             2. 사용 변수
         19. 모듈 이름 : enqueue()
             1. 기능
             2. 사용 변수
         20. 모듈 이름 : dequeue()
             1. 기능
             2. 사용 변수
         21. 모듈 이름 : freeQueue()
             1. 기능
             2. 사용 변수
   7. symbol.c
      1. 설명
      2. 모듈 설명
         1. 모듈 이름 : symbol()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : addSym()
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : freeSymTab()
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : symHashFunc()
            1. 기능
            2. 사용 변수
         5. 모듈 이름 : findSym()
            1. 기능
            2. 사용 변수
         6. 모듈 이름 : isStr()
            1. 기능
            2. 사용 변수
         7. 모듈 이름 : getBiggerStr()
            1. 기능
            2. 사용 변수
         8. 모듈 이름 : getMaxofFour()
            1. 기능
            2. 사용 변수
         9. 모듈 이름 : getMaxofThree()
            1. 기능
            2. 사용 변수
         10. 모듈 이름 : printSymbol()
             1. 기능
             2. 사용 변수
4. 구조체 정의
   1. HISTORY
   2. opNode
   3. symNode
   4. numNode
   5. tRecord
5. 전역 변수 정의
   1. HISTORY\* hisHead
   2. opNode\*\* opTable
   3. unsigned char\* MEMORY
   4. int END\_ADDR
   5. symNode\*\* SYMTAB
   6. numNode\* numHead
   7. tRecord\* tRHead
   8. tRecord\* tRTail
6. 코드

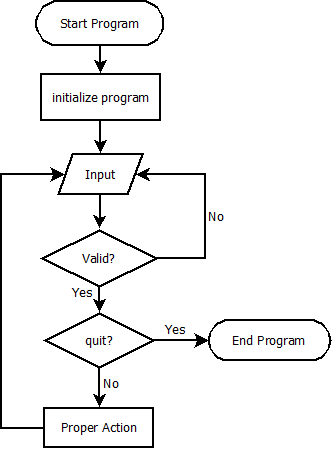
1. 프로그램 개요

프로젝트 1에서 Shell을 구현하여 메모리를 할당하고, opcode table 등을 생성하였다. 이번 프로젝트에서는 구현한 shell에서 실행할 SIC/XE Machine 어셈블러를 구현하였다. assembly program source 파일을 입력받아 object file과 listing file을 생성한다. 이 때 어셈블 과정에서 symbol table을 생성한다.

보고서에는 프로젝트 1에서 변경된 내용이나 새로 추가된 내용에 대해서만 기술하였다.

2. 프로그램 설명

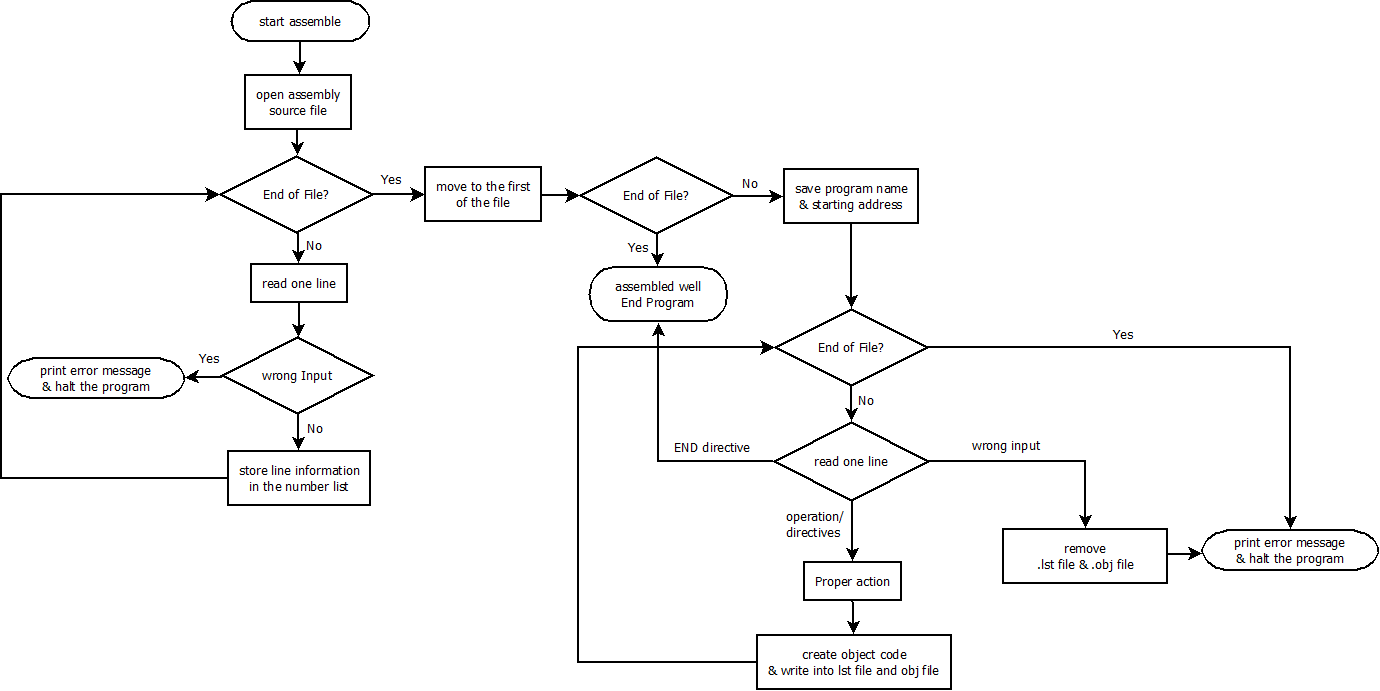
2.1 프로그램 흐름도



< SIC Shell Program Flow Chart >

전체적인 흐름도는 위와 같이 프로젝트 1과 동일하다. 한편 프로젝트 2는 주어진 assembly file에 대해 assembler를 구현하는 것이 주 목표이므로 assembler 및 symbol table에 초점을 맞춰 보고서를 작성한다.

2.2 SIC/XE Assembler 흐름도



< Assembler Flow Chart >

assembler의 전체적인 흐름도는 위와 같다. flow chart에서 ‘move to the first of the file’을 기준으로 좌측은 pass1, 우측은 pass2를 나타낸다. 이 때 pass1을 거치며 각 줄의 location counter를 부여하고, label과 location counter를 저장한 symbol table을 형성한다. 또한 pass2에서 활용할 중간단계 list인 number list를 형성한다.

pass1을 성공적으로 끝마쳤다면 pass2를 거치며 object code를 생성한다. 각 줄에 대해 적절한 action을 수행하고, listing file과 object file을 작성하며 전체 assemble 과정을 끝마친다.

한편 위의 flow chart에서 ‘halt the program’ 및 ‘End Program’은 실행 파일 자체를 중단시키는 것이 아닌, assemble 과정을 중단시킨단 뜻이다.

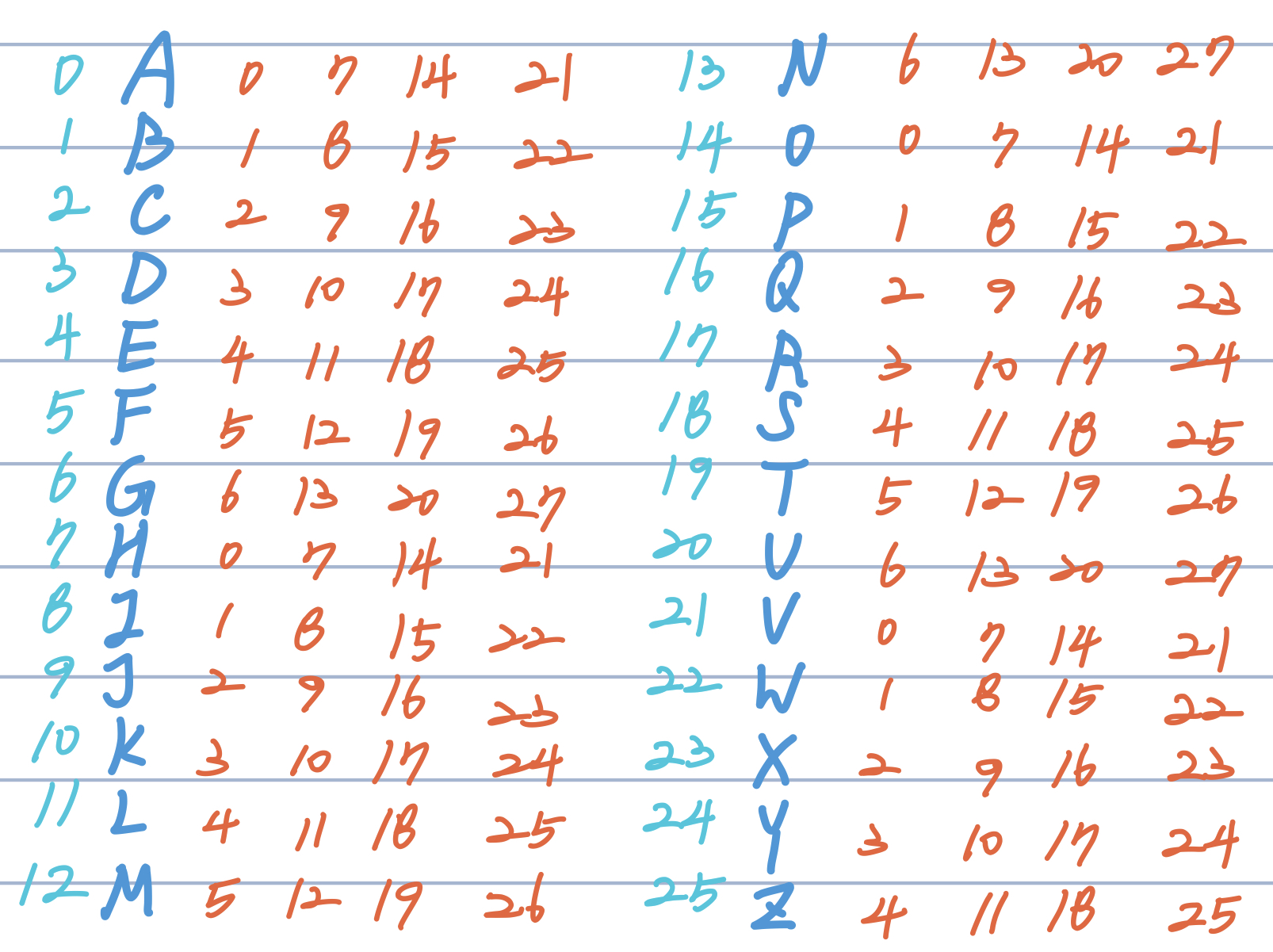
2.3 Symbol Table

assemble 과정에서 각 label과 location counter를 저장하는 symbol table을 생성한다. 이 때 symbol table을 형성하는 과정에서 고려한 사항은 아래와 같다.

1. symbol을 저장할 때 각 label이 잘 배분될 수 있도록 하는 문제점.
2. symbol에 접근(탐색)할 때 접근의 효율성.
3. 내림차순(Z – A)으로 출력을 할 때 정렬의 효율성.

이러한 기준으로 symbol table은 아래와 같이 구성하였다.

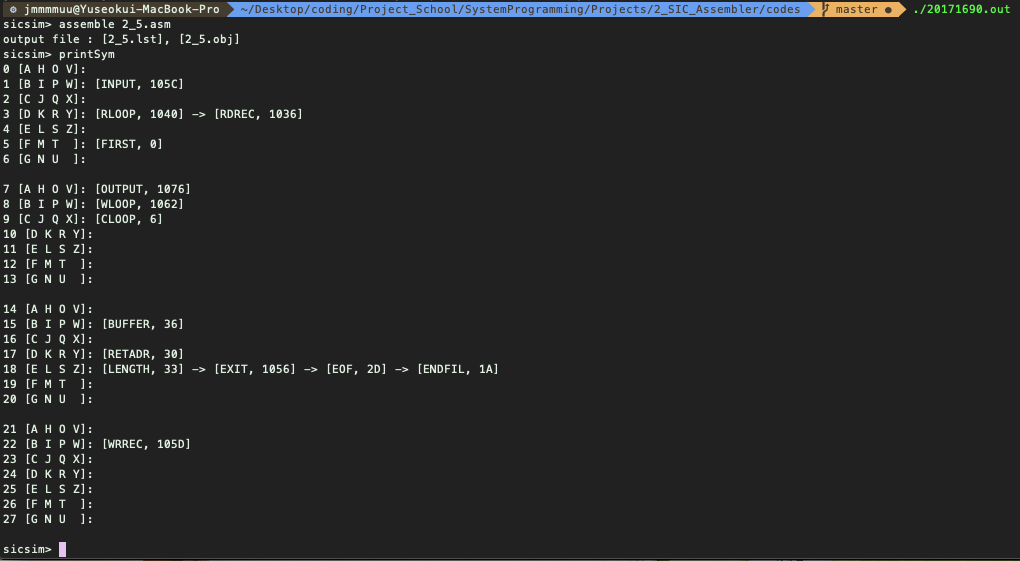
출력 시 정렬의 효율성을 고려하여 각 label의 첫 알파벳에 따라 각 알파벳은 총 4개의 head에 연결될 수 있다. 하지만 label의 첫 글자만을 이용해 index를 나눈다면 symbol을 저장할 때 각 label이 index에 따라 잘 배분되지 않을 것이므로 label의 모든 알파벳의 ASCII code를 더하여 4로 나눈 나머지를 이용해 4개의 head에 분배되도록 하였다. 이러한 과정을 총 26개 알파벳에 대하여 수행하였고, 따라서 각 index에는 최대 4개의 알파벳으로 시작하는 label이 저장된다. 때문에 Symbol table의 size는 알파벳 개수 26이상이며 26과 가장 근접한 4의 배수인 28이 된다. 이를 도식화하면 아래와 같다.



< alphabet number – alphabet – index >

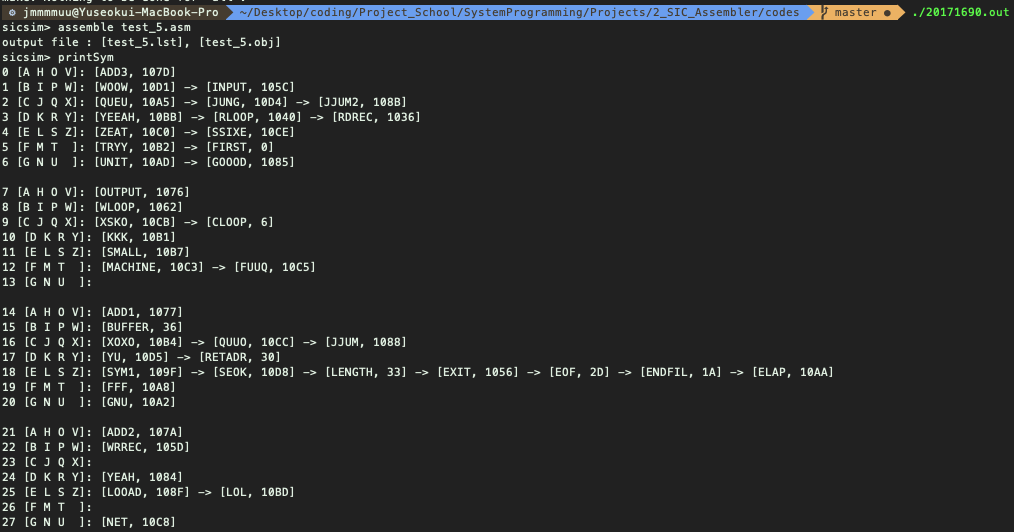
각 알파벳은 label의 시작 알파벳을 의미하며, 오른쪽 4개 숫자의 인덱스 중 하나에 해당 label이 저장된다. 따라서 한 알파벳은 총 4개의 head node 중 하나에 연결되는 것이다.

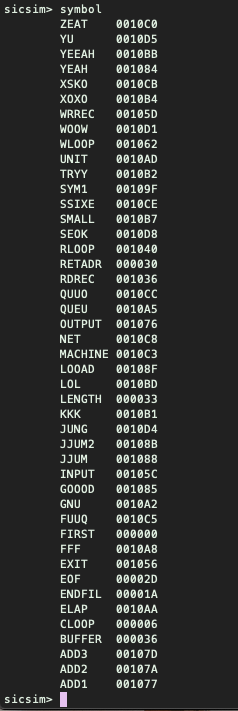
이는 또한 한 인덱스에 총 4개의 알파벳으로 시작하는 label이 연결된다는 것을 뜻한다. 한편 symbol table에 label을 추가할 때, 출력 시 정렬의 효율성을 높이기 위해 내림차순으로 저장하였다. 이를 도식화하여 symbol table을 출력하는 함수가 3.7.2.10의 printSymbol() 함수이다. SIC Shell에서 ‘printSym’을 입력하면 가장 최근에 assemble한 파일의 symbol table을 아래와 같이 출력한다.



< 배포된 2\_5.asm 파일에 대한 symbol table >

이를 직접 제작한 test file에 대해 실행시키면 결과는 아래와 같다.





Symbol table의 출력 또한 내림차순으로 출력되는 것을 확인할 수 있다.

3. 분할 c파일

프로젝트 1에서 수정된 c 파일은 shell.c 파일이다. help 명령어에서 새로운 명령어 목록을 업데이트하였고, type filename 기능을 추가하였다. 또한 이번 프로젝트의 목적인 어셈블러 기능을 구현한 assemble.c, 어셈블러 기능을 추가하기 위해 작성한 함수들을 모아 놓은 assembler\_functions.c, 그리고 assemble 과정에서 생성하는 symbol table과 관련된 기능을 구현한 symbol.c 파일이 새로 추가되었다.

3.1 20171690.c

**3.2 shell.c**

3.2.1 설명

Shell 관련 명령어들과 관련된 필요한 기능들을 구현한 파일이다. 이번 프로젝트에서 구현해야 할 shell 관련 명령어는 h[elp], d[ir], hi[story], q[uit] 으로 4개이다.

3.2.2 모듈 정의

3.2.2.1 모듈 이름 : **help()**

3.2.2.1.1 기능

Shell에서 실행 가능한 모든 명령어들의 리스트를 화면에 출력한다. 단순 printf를 활용하여 출력하였다. 이번 프로젝트에서 새로 구현한 명령어 (type, assemble, symbol) 목록을 추가하였다.

3.2.2.1.2 사용 변수

없음

3.2.2.2 모듈 이름 : **type(char\* filename)**

3.2.2.2.1 기능

parameter로 전달받은 filename에 해당하는 파일을 현재 디렉터리에서 읽어 화면에 출력한다. 이 때 파일이 없으면 에러 메시지를 출력한다.

3.2.2.2.2 사용 변수

**char\* filename** – 읽을 파일의 이름이다.

**FILE\* fp** – 읽어올 파일 포인터를 가리킨다.

3.3 memory.c

3.4 opcode.c

**3.5 assembler.c**

3.5.1 설명

assembler를 구현한 c파일이다. main함수에서 parameter로 받아온 파일을 pass1, pass2를 거쳐 어셈블하는 과정에 필요한 함수들을 구현하였다.

3.5.2 모듈 정의

3.5.2.1 모듈 이름 : **assemble(char\* filename)**

3.5.2.1.1 기능

main함수에서 파일이름을 parameter로 전달받는다. pass1과 pass2를 호출하여 전달받은 파일을 어셈블하며, 성공적으로 어셈블하면listing file과 object file을 생성한다. 그렇지 않다면 에러메시지를 출력하고 입력을 다시 받는다.

3.5.2.1.2 사용 변수

**char\* filename** – 읽을 파일의 이름이다.

**FILE\* asmFP** – 읽을 파일의 파일 포인터이다.

3.5.2.2 모듈 이름 : **pass1(FILE\* fp)**

3.5.2.2.1 기능

어셈블 과정에서 pass1에 해당하는 기능을 수행하는 함수이다. 파일을 읽으며 각 줄에 번호를 부여한다. 또한 instruction size를 계산하여 location Counter를 부여한다. 이 때 이러한 정보를 새로운 구조체를 선언하여 저장한다. 구조체는 line number, location counter 정보를 저장한다. 그리고 START directive와 END directive 및 comment, blank line을 구별하는 플래그를 포함한다. 이러한 정보를 담은 리스트를 생성하여 pass2에서 활용하였다.

또한 label이 있는 경우 해당 label과 location counter등의 정보를 저장하는 symbol table을 생성한다. 이 때 자세한 symbol table의 정보는 3.7에 기술하였다.

만약 중복된 label사용 및 잘못된 operation의 사용과 같이 오류가 있는 assembly file에 의해 에러가 발생하면 에러 메시지를 출력하고 어셈블 과정을 중단한다.

3.5.2.2.2 사용 변수

**FILE\* fp** – 함수의 parameter로 읽어올 파일의 파일 포인터이다.

**int LOCCTR** – 각 줄에 해당하는 location counter를 저장하는 변수이다.

**int startingAddr** – 시작 주소를 저장하는 변수이다.

**int i** – loop counter로 사용되었다.

**int lineNum** – 각 줄에 해당하는 line number를 저장하는 변수이다.

**int instructionSize** – 각 줄의 instruction size를 저장하는 변수이다.

**char\* line**  - string변수로 파일에서 한 줄을 읽어올 때 그 시작 주소를 가리킨다.

**char programName[10]** – START directive가 있는 경우에 프로그램 이름을 저장하는 변수이다.

**char\*\* token** – comment나 blank line이 아닐 경우에 파일의 한 줄을 label, operation, operand의 토큰으로 나눈다. 이 때 각 토큰을 저장하는 문자열 array의 시작주소를 가리킨다.

**int tokenNum** – 각 줄의 token의 개수를 저장하는 변수이다.

3.5.2.3 모듈 이름 : **pass2(FILE\* fp, char\* filename)**

3.5.2.3.1 기능

pass1에서 생성한 중간 정보 linked list와 symbol table을 활용하여 assembly file의 각 instruction에 대해 assemble을 끝마친다. 입력받은 line이 START directive가 포함되어 있다면 프로그램 이름과 시작 주소를 입력받는다. 입력받은 line에 skip flag가 설정되어 있다면 해당 라인은 object code를 생성하지 않고 지나친다.

다른 경우에 대해서 symbol table과 operation code table 및 directive 정보 등을 활용해 object code를 생성한다. 성공적으로 어셈블을 마쳤을 경우 listing file과 object file을 생성한다. 그렇지 않을 경우 에러 메시지를 출력하고 어셈블 과정을 중단한다.

한편 신뢰도 있는 프로그램을 작성하기 위해 다양한 케이스들에 대해 예외처리를 하고 error message 및 warning message를 출력하도록 구현하였다. 아래에 error message와 warning message가 출력되는 경우들을 정리했다.

* Error message가 출력되는 경우
  + 파일이 empty file인 경우.
  + 입력받은 시작 주소가 16진수 수가 아닌 경우.
  + 정의되지 않은 label이 operand로 사용되었을 경우.
  + operand의 주소가 20 bit(4 형식)으로 나타낼 수 없는 경우.
  + BYTE directive 사용 시 operand로 잘못된 syntax가 사용된 경우.
  + 각종 syntax 에러들과 incorrect address의 사용 등의 경우.
* Warning message가 출력되는 경우
  + 어셈블하는 파일에 주석과 blank line을 제외하고 아무 내용이 없는 경우. 이 때 프로그램은 warning message를 출력하고 listing file과 object file을 생성하지 않는다.
  + BASE directive가 사용되지 않은(B register가 초기화되지 않은) 상태에서 B register에 접근하려는 경우. 이 때 프로그램은 해당 instruction에 대해 4형식 4byte object code를 생성한다. 이 때 바로 아래의 ‘+’ sign이 없다는 warning message가 항상 함께 출력된다.
  + 이번 프로젝트에서 4형식의 4byte object code를 생성하는 경우에 operation 앞에 ‘+’ sign을 붙였다. 만약 ‘+’ sign이 없지만 3형식으로 나타낼 수 없는 경우에 warning message를 출력하고 4형식 object code를 생성하였다.
  + BYTE directive나 WORD directive와 같은 상수에 label이 없는 경우, object code가 생성이 되고 location counter는 증가되지만 해당 상수에 접근할 수 있는 방법이 없다. 따라서 이러한 경우에 warning message를 출력한다.

3.5.2.3.2 사용 변수

**FILE\* fp** – open할 file pointer 변수.

**char\* filename** – open할 file의 이름이다.

**int i** – loop counter로 사용되었다.

**int tokenNum** – pass1과 마찬가지로 token의 개수를 저장한다.

**int startingAddr** – pass1과 마찬가지로 시작 주소를 저장한다.

**int endAddr** – 파일의 마지막 주소, 즉 END directive가 포함된 줄의 location counter를 저장한다.

**int size** – 각 instruction line들을 분석할 때 계산한 instruction size를 의미한다.

**numNode\* pCurrent** – pass1에서 생성한 중간 단계의 정보를 담은 linked list에서 현재 줄의 node를 가리키는 변수이다.

**char\* line** – pass1에서와 같이 현재 줄의 string 시작주소를 가리킨다.

**char\*\* token** – pass1과 마찬가지로 각 줄에서 각 정보들의 token 배열의 시작주소이다.

**FILE\* LF** – 생성할 listing file의 파일 포인터.

**FILE\* OF** – 생성할 object file의 파일 포인터.

**int format** – 각 라인에서 operation이 존재할 경우 지정 형식을 저장하는 변수이다.

**int directiveNum** – 만약 현재 line이 directive를 포함한다면 해당 directive의 번호를 저장하는 변수.

**int objCode** – 현재 line에 대해 계산한 object code를 저장하는 변수.

3.5.2.4 모듈 이름 : **getInstructionSize(char\*\* token, int lineNum, int isLabel)**

3.5.2.4.1 기능

instruction size를 계산하여 반환하는 함수이다. operation의 경우 ‘+’ sign의 유무 및 operation code table을 활용하여 각 포맷을 return 한다.

BYTE, WORD, RESB, RESW 등 변수 및 상수의 size는 계산 과정을 거쳐 그 size를 반환한다.

3.5.2.4.2 사용 변수

**char\*\* token** – 파일의 한 줄에서 입력받은 instruction을 token으로 나눈 배열의 시작 주소.

**int lineNum** – 현재 line의 번호를 저장한다. error 발생 시 error message를 출력하기 위해 parameter로 전달받았다.

**int isLabel** – 파일의 현재 line에서 label의 유무를 parameter로 전달받았다.

**int opIdx** – operand가 포함된 token의 인덱스를 저장하는 변수.

**int size** – 계산한 size를 저장하는 변수.

3.5.2.5 모듈 이름 : **getObjCode(char\*\* token, int\* format, int type, numNode\* pCurrent)**

3.5.2.5.1 기능

pass2 진행 중 object code를 생성해야하는 경우에 함수를 호출하여 object code를 계산하는 함수이다. operation 및 operand의 정보를 저장하는 token과 instruction size를 의미하는 format, operation 및 상수 등의 정보를 저장하는 type과 현재 위치의 node pCurrent를 parameter로 전달받아 object code를 구한다.

3.5.2.5.2 사용 변수

**char\*\* token** – pass2에서 operation의 위치를 parameter로 넘겨 operation 및 operand를 token으로 저장하는 변수이다.

**int\* format** – operation의 경우 format을, directive의 경우 필요한 size를 저장한다.

**int type** – operation, BYTE directive, WORD directive의 경우에 각각 0, 1, 2를 저장한다.

**numNode\* pCurrent** – 현재 line의 정보를 담고있는 node로서 error message를 출력할 때 사용하기 위해 parameter로 설정하였다.

**unsigned char n, i, x, b, p, e** – operation format 3, 4의 경우에 object code를 계산하기 위해 각각 n, i, x, b, p, e의 정보를 저장하는 변수이다.

**unsigned char r1, r2** – format 2의 경우에 register를 operand로 사용하는데, 이 때 각 register의 번호를 저장할 변수이다.

**int reg** – 위의 r1, r2를 이용해 계산한, object code의 일부로 들어갈 reg변수이다.

**int b1, b2, b3, b4** – operation의 경우에 1byte의 1형식부터 4byte의 4형식까지, 각 1byte씩의 정보를 저장하는 변수 4개이다.

**int disp, addr** – 각각 3형식과 4형식에서 displacement, address정보를 저장하는 변수이다.

**int opCode** – operation code를 저장하는 변수이다.

**int operand** – operand가 16진수 수일 때, 해당 수를 저장하는 변수이다.

**int objCode** – 계산하여 return할 object code를 저장하는 변수이다.

**int lc** – loop counter로 사용되었다.

**int target** – operand로 사용된 symbol의 LOCCTR을 저장하기 위한 변수이다.

**char\* operandStr[5]** – 만약 operand에 comma(‘,’)가 포함되어있을 때, 각 comma를 기준으로 token으로 나눈 문자열을 저장하는 변수이다.

**char\* sym** – operand에서 symbol에 해당하는 부분을 가리키는 문자열 변수이다.

3.5.2.6 모듈 이름 : **printObjCode(int size, int objCode, FILE\* fp)**

3.5.2.6.1 기능

parameter로 입력받은 object code를 1byte 단위로 출력하는 함수이다. 이 때 출력은 기본적으로 file 출력이며, 출력해야하는 파일 포인터는 parameter로 전달받은 fp이다. 출력해야하는 object code의 크기는 size에 저장되어있다.

3.5.2.6.2 사용 변수

**int size** – 출력해야할 object code의 크기이다.

**int objCode** – 출력해야할 object code이다.

**FILE\* fp** – 출력할 file의 파일 포인터이다.

**int hex** – 출력할 object code를 각 바이트 단위로 자른 수를 저장하는 변수이다.

**int i** – loop counter로 사용되었다.

3.5.2.7 모듈 이름 : **tokenizeAsmFile(char\*\*\* token, char\* input)**

3.5.2.7.1 기능

파일에서 입력받은 한 line을 label, operation, operand의 token으로 자르는 함수이다. token으로 자른 문자열은 문자열의 배열인 token에 저장된다. 그리고 token의 개수를 return한다.

3.5.2.7.2 사용 변수

**char\*\*\* token** – 입력받은 line을 token으로 잘라 저장할 문자열 배열 변수이다.

**char\* input** – 입력받은 한 줄의 시작 주소를 나타낸다.

**int cnt** – token의 개수를 저장하는 변수이다.

**char\* tmp** – 잘린 문자열의 주소를 임시로 저장하는 변수이다.

3.5.2.8 모듈 이름 : **removeSpaceAroundComma(char\* input)**

3.5.2.8.1 기능

입력받은 line에서 comma가 포함되어있을 경우, 해당 comma를 기준으로 좌우의 white space를 지우는 함수이다. 라인을 ‘,’ 기준으로 tokenize하기 쉽게 만들기 위해 해당 함수를 작성하였다.

3.5.2.8.2 사용 변수

**char\* input** – 입력받은 한 line의 시작 주소를 나타낸다.

**int isComma** – 받은 line에서 comma가 있으면 1을, 없으면 0을 저장하여 해당 값을 return한다.

**int cnt** – comma를 기준으로 white space를 지우고 난 뒤 변화된 index를 구하기 위해 사용된 변수이다.

**int i, j** – loop counter로 사용되었다.

**3.6 assembler\_functions.c**

3.6.1 설명

Assembler를 구현하기 위해 필요한 다양한 함수들을 작성한 c 파일이다.

3.6.2 모듈 정의

3.6.2.1 모듈 이름 : **getRegNum(char\* reg)**

3.6.2.1.1 기능

format2의 경우 register간의 연산을 수행한다. 이 때 object code를 생성하기 위해 각 register에 대한 register number를 반환해주는 함수이다. 만약 해당 register에 대한 정보가 존재하지 않는다면 0xFF를 반환한다.

3.6.2.1.2 사용 변수

**char\* reg** – 번호를 구할 register를 나타낸다.

3.6.2.2 모듈 이름 : **LDB(char\*\* token, int lineNum)**

3.6.2.2.1 기능

LDB operation이 사용되었을 때 B register에 값을 load하는 함수이다.

이 때 immediate addressing에 대해서만 구현이 된 상태이다. indirect addressing의 경우 프로젝트 1에서 할당한 memory를 직접 참조하여 값을 가져와야하므로 에러 메시지를 출력하고 어셈블 과정을 중단하도록 작성하였다.

기타 잘못된 입력에 대해서도 에러 메시지를 출력하고 어셈블 과정을 중단한다.

3.6.2.2.2 사용 변수

**char\*\* token** – operation 및 operand의 정보를 담은 token 변수.

**int lineNum** – 에러 메시지를 출력할 때 사용하기 위해 전달받은 현재 줄의 번호.

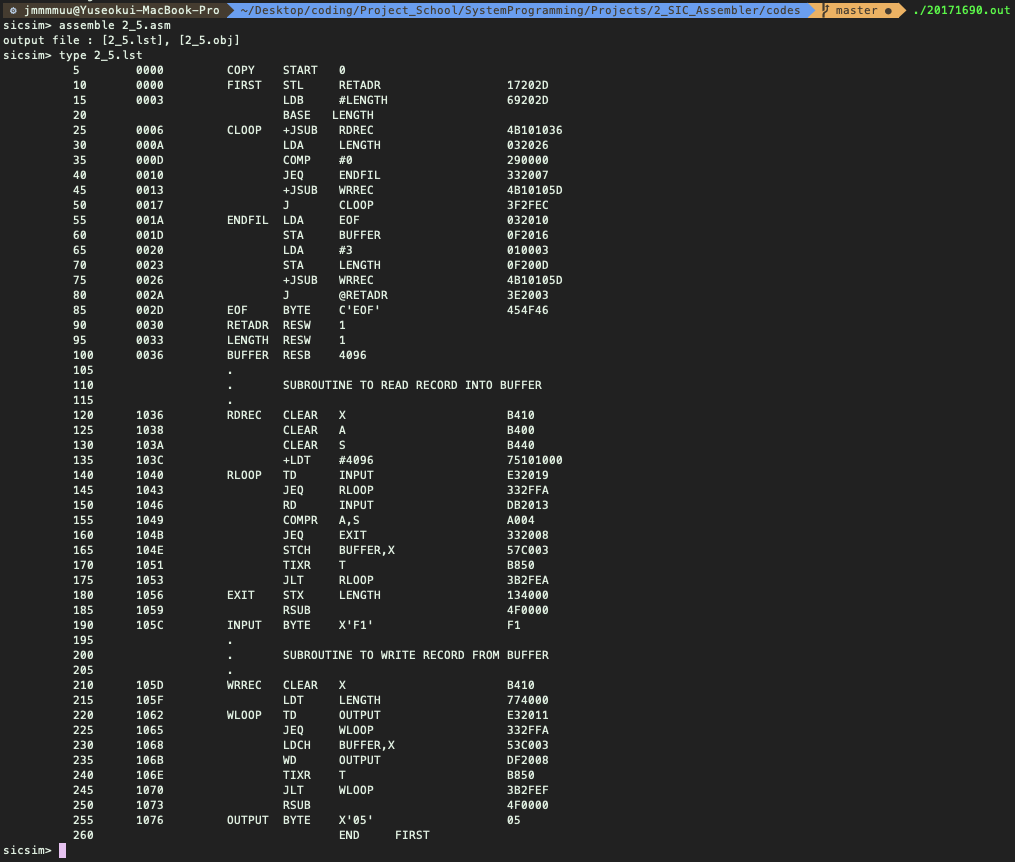
**int tmp** – indirect addressing을 구현할 때 사용하기 위해 선언하였으며, symbol의 LOCCTR을 임시로 저장하는 변수이다.

3.6.2.3 모듈 이름 : **printLineinLST(numNode\* pCurrent, char\*\* token, int size, int tokenNum, int objCode, FILE\* LF, int isLabel)**

3.6.2.3.1 기능

listing file에 정보를 출력하는 함수이다. 이 때 각 line에서 token 의 개수 및 각 token 문자열의 길이, object code 및 location counter 출력 유무에 따라 출력 형식이 달라지므로 함수로서 해당 과정을 일원화시켰다.

한편 출력을 하는 과정에서 tab size는 8을 기준으로 코드를 작성하였다. 사용자 컴퓨터의 설정에 따라 출력이 다르게 나올 수 있으며, 기준으로 작성한 tab size = 8에 대한 프로그램 실행 예시는 아래와 같다.



3.6.2.3.2 사용 변수

**numNode\* pCurrent** – 현재 위치의 line 정보를 담고있다.

**char\*\* token** – 현재 line의 token 문자열을 저장한다.

**int size** – 출력해야할 object code의 size를 저장한다.

**int tokenNum** – token의 개수를 저장한다.

**int objCode** – 현재 위치의 instruction에 대한 object code를 저장한다.

**FILE\* LF** – listing file의 파일 포인터를 가리킨다.

**int isLabel** – label의 유무를 의미한다.

**int i** – loop counter로 사용되었다.

3.6.2.4 모듈 이름 : **isWhiteSpace(char ch)**

3.6.2.4.1 기능

parameter로 전달받은 character가 white space인지 아닌지를 판단한다. 맞다면 1을, 아니면 0을 반환한다.

3.6.2.4.2 사용 변수

**char ch** – 확인할 문자를 나타낸다.

3.6.2.5 모듈 이름 : **isBlankLine(char\* input)**

3.6.2.5.1 기능

parameter로 전달받은 문자열이 white space로만 이루어져있다면 1을, 아니면 0을 return한다.

3.6.2.5.2 사용 변수

**char\* input** – 확인할 문자열의 시작주소를 나타낸다.

3.6.2.6 모듈 이름 : **isDirective(char\*token)**

3.6.2.6.1 기능

입력받은 문자열이 directive인지 판단한다. 맞다면 해당 directive에 임의로 부여한 번호를, 아니면 0을 return한다.

3.6.2.6.2 사용 변수

**char\* token** - 확인할 문자열의 시작주소를 나타낸다.

3.6.2.7 모듈 이름 : **isComma(char\* input)**

3.6.2.7.1 기능

현재 입력받은 문자열에 comma(‘,’)가 포함되었다면 1을, 아니면 0을 return한다.

3.6.2.7.2 사용 변수

**char\* input** – 확인할 문자열의 시작주소를 나타낸다.

**int I** - loop counter로 사용되었다.

3.6.2.8 모듈 이름 : **nameToListing(char\* filename)**

3.6.2.8.1 기능

filename에 해당하는 이름의 listing file 이름을 생성한다. filename에서 확장자를 지우고 ‘.lst’를 붙인다.

3.6.2.8.2 사용 변수

**char\* filename** – 변경할 파일의 이름을 저장한다.

**char\* lstName** – 변경될 파일의 이름을 저장한다.

**int i** – loop counter로 사용되었다.

3.6.2.9 모듈 이름 : **nameToObj(char\* filename)**

3.6.2.9.1 기능

filename에 해당하는 이름의 object file 이름을 생성한다. filename에서 확장자를 지우고 ‘.obj’를 붙인다.

3.6.2.9.2 사용 변수

**char\* filename** – 변경할 파일의 이름을 저장한다.

**char\* lstName** – 변경될 파일의 이름을 저장한다.

**int i** – loop counter로 사용되었다.

3.6.2.10 모듈 이름 : **toUpper(char ch)**

3.6.2.10.1 기능

입력받은 character가 소문자 알파벳이라면 upper case로 바꿔 return한다.

3.6.2.10.2 사용 변수

**char ch** – 변경한 문자를 저장한다.

3.6.2.11 모듈 이름 : **toUpperCase(char\* input)**

3.6.2.11.1 기능

입력받은 문자열에서 모든 문자를 대문자 문자로 바꾼다.

3.6.2.11.2 사용 변수

**char\* input** – 변경할 문자열의 시작주소를 저장한다.

**int i** – loop counter로 사용되었다.

3.6.2.12 모듈 이름 : **byteSize(char\* input)**

3.6.2.12.1 기능

상수 BYTE의 size를 계산하고 return한다. 만약 옳바른 문법이 아니라면 -1을 return한다.

3.6.2.12.2 사용 변수

**char\* input** – Constant Byte에 대한 정보가 저장된 operand 문자열의 시작주소이다.

**int size** – 계산한 상수의 크기를 저장한다.

3.6.2.13 모듈 이름 : **wordSize(char\* input)**

3.6.2.13.1 기능

상수 WORD의 size를 계산하고 return한다. 이 때 WORD SIZE는 SIC/XE machine에서 3byte를 의미한다.

3.6.2.13.2 사용 변수

**char\* input** – Constant Word에 대한 정보가 저장된 문자열의 시작주소이다.

**int size** – 계산한 상수의 크기를 저장한다.

3.6.2.14 모듈 이름 : **resbSize(char\* input)**

3.6.2.14.1 기능

변수 바이트의 개수를 return한다.

3.6.2.14.2 사용 변수

**char\* input** – operand에 해당하는 문자열의 시작주소이다.

**int size** – 계산한 변수의 크기를 저장한다.

3.6.2.15 모듈 이름 : **reswSize(char\* input)**

3.6.2.15.1 기능

변수 WORD의 개수를 구하고, 해당 변수의 총 사이즈를 반환한다.

3.6.2.15.2 사용 변수

**char\* input** – operand에 해당하는 문자열의 시작주소이다.

**int size** – 계산한 변수의 크기를 저장한다.

3.6.2.16 모듈 이름 : **addNum(int lineNum, int LOC, numNode\* pLast, int flagType)**

3.6.2.16.1 기능

pass1에서 중간 단계 정보들을 저장하는 node를 추가하는 함수이다. 새로운 node의 공간을 할당하여 정보를 저장하고, list의 가장 마지막 노드에 추가한다.

3.6.2.16.2 사용 변수

**int lineNum** -

**int LOC** -

**numNode\* pLast** – node를 추가할 때 시간을 단축하기 위해 parameter로 전달받았으며, list의 마지막 node를 가리킨다.

**int flag\_type** – 각종 flag의 종류를 의미하며, text record인 경우, START directive 및 END directive가 나온 경우, 주석 및 blank line이 나온 경우를 구분한다.

3.6.2.17 모듈 이름 : **freeNums()**

3.6.2.17.1 기능

pass1을 거치며 만든 중간 정보 list인 numNode를 free하는 함수이다. HEAP 영역에 할당된 모든 공간을 해제한다.

3.6.2.17.2 사용 변수

**numNode\* pFree** – 해제할 노드를 가리킨다.

3.6.2.18 모듈 이름 : **printNums()**

3.6.2.18.1 기능

디버깅을 위해 임시로 작성한 함수로써, numHead에 저장된 모든 list의 정보를 출력한다.

3.6.2.18.2 사용 변수

**numNode\* pMove** – 출력할 현재 위치의 node를 의미한다.

3.6.2.19 모듈 이름 : **enqueue(int objCode, int size, int LOC, FILE\* OF)**

3.6.2.19.1 기능

object file의 출력 과정은 QUEUE를 활용해 구현하였다. 각 instruction line을 assemble할 때마다 QUEUE에 관련 정보를 저장하고, 저장된 정보의 크기가 MAX-SIZE에 도달했거나, 혹은 변수가 사용되었을 때마다 object file의 한 줄을 출력해주어야 한다. 따라서 FIFO 구조를 따르는 QUEUE를 활용해 구현하였으며, 그 중 queue에 정보를 입력하는 enqueue() 함수에 해당한다.

3.6.2.19.2 사용 변수

**int objCode** – 저장할 object code를 나타낸다.

**int size** – object code의 size를 저장한다.

**int LOC** – 현재 line의 LOCCTR을 저장한다.

**FILE\* OF** – object file의 파일 포인터를 가리킨다.

**tRecord\* pNew** – enqueue할 QUEUE의 새로운 노드를 가리킨다.

3.6.2.20 모듈 이름 : **dequeue(FILE\* OF)**

3.6.2.21.1 기능

QUEUE에 저장된 정보의 크기가 MAX-SIZE(0xFF)에 도달한 경우, 혹은 현재 PC가 변수를 가리키는 경우에 object file에서 줄을 바꿔주어야 한다. 이 때 dequeue를 활용해 저장된 정보를 모두 출력한다.

3.6.2.21.2 사용 변수

**FILE\* OF** – 출력할 object file의 파일 포인터를 가리킨다.

**int len** – text record 한 줄의 길이를 저장한다.

**tRecord\* pFree** – 출력하고 해제시킬 현재 노드를 가리킨다.

3.6.2.22 모듈 이름 : **freeQueue()**

3.6.2.22.1 기능

QUEUE에 저장된 모든 정보를 해제한다.

3.6.2.22.2 사용 변수

**tRecord\* pFree** – 해제할 현재 노드를 가리킨다.

**3.7 symbol.c**

3.7.1 설명

3.7.2 모듈 정의

3.7.2.1 모듈 이름 : symbol()

3.7.2.1.1 기능

sic shell에서 symbol 명령어를 입력했을 때 호출되는 함수로서 가장 최근에 어셈블한 파일의 symbol table을 출력한다. 만약 symbol table이 없다면 에러메시지를 출력한다.

출력하는 형식은 내림차순 순으로 Z to A 순서로 출력한다. symbol table에 대한 자세한 설명은 2.3에 기술하였다.

3.7.2.1.2 사용 변수

**int idx[4]** – 현재 검사할 알파벳이 포함된 4개 인덱스를 저장하는 배열이다.

**int ptIdx** – 현재 출력할 node의 인덱스를 가리킨다.

**int cnt** – 4개의 인덱스 중 원하는 알파벳으로 시작하는 symbol이 포함되지 않은 인덱스의 개수이다.

**int sign** – 4개의 인덱스 중 원하는 알파벳으로 시작하는 symbol이 없는 경우 각 인덱스 번호마다 logical left shift 연산을 통해 sign을 계산하여 그 값을 이용해 인덱스를 옮기는데 사용하였다.

**char ch** – symbol table을 출력할 때 각 symbol의 시작 character의 알파벳 ‘Z’ – ‘A’를 의미한다.

**int i** – loop counter로 사용되었다.

3.7.2.2 모듈 이름 : addSym(char\* label, int LOC)

3.7.2.2.1 기능

새로운 label과 location counter를 추가하는 함수이다. 이 때 symbol table은 hash table이며, 계산한 인덱스를 활용해 해당 헤드에 노드를 추가하였다.

한편 노드를 출력할 때는 Z-A 순서로 symbold을 출력하여야 한다. 따라서 출력 시 시간 복잡도를 줄이기 위해 노드를 추가할 때 내림차순으로 저장하였다.

만약 label에 해당하는 node가 이미 hash table에 존재한다면 0을 return 한다. 입력된 label의 첫 문자가 대문자 알파벳이 아닌 경우에도 0을 return한다. 성공적으로 table에 node를 추가하였다면 1을 return한다.

3.7.2.2.2 사용 변수

**char\* label** – symbol table에 추가할 label을 의미한다.

**int LOC** – symbol table에 추가할 location counter를 의미한다.

**int idx** – 새로운 노드를 추가할 index를 저장한다.

3.7.2.3 모듈 이름 : freeSymTab()

3.7.2.3.1 기능

현재 symbol table에 있는 모든 노드를 해제한다.

3.7.2.3.2 사용 변수

**int i** – loop counter로 사용되었다.

**symNode\* pFree, \*ptmp** – table을 탐색하며 노드를 해제할 때 사용된 변수들이다.

3.7.2.4 모듈 이름 : symHashFunc(char\* label)

3.7.2.4.1 기능

현재 노드를 저장할 index를 계산하는 hash function이다.

입력되는 symbol을 assembly file에서 사용된 label로서, 프로그래머에 따라 LOOPA, LOOPB 와 같이 비슷한 이름의 label을 많이 사용한다는 특성을 고려하여 hash function을 구성하였다.

기본적으로 label의 시작 알파벳에 따라 한 알파벳은 총 4개의 head 중 하나에 연결된다. 이 때 4개의 head는 label에 포함된 모든 알파벳의 ASCII 코드를 더한 값을 4로 나눈 나머지에 의해 결정된다. 총 26개의 알파벳에 대해 이러한 계산이 이루어지고, 결론적으로 하나의 인덱스에는 총 4개의 알파벳으로 시작하는 label이 뒤따르게 된다. 이러한 index를 구하는 함수이다.

적절한 label이 parameter로 전달되었다고 가정하며, 계산한 index를 return한다.

3.7.2.4.2 사용 변수

**int idx** – 계산한 index를 저장한다.

**int rmd** – label의 시작 알파벳 번호를 7로 나눈 나머지를 저장한다.

**int i** – loop counter로 사용되었다.

**int quo** – label 의 모든 알파벳의 ASCII code를 더한 값을 4로 나눈 나머지를 저장한다.

3.7.2.5 모듈 이름 : findSym(char\* label)

3.7.2.5.1 기능

label을 parameter로 입력받아 해당 symbol이 symbol table에 있는지 찾는다. 만약 있다면 symbol의 location counter 를 return하고, 없다면 -1을 return한다.

3.7.2.5.2 사용 변수

**char\* label**  - symbol table에 존재하는지 확인할 label 이름 문자열이다.

3.7.2.6 모듈 이름 : isStr(char\* Str)

3.7.2.6.1 기능

입력받은 문자열이 null 문자열인지 아닌지 판단하는 함수이다. readability를 위해 구현하였다.

3.7.2.6.2 사용 변수

**char\* Str** – 문자열인지 아닌지 확인할 변수.

3.7.2.7 모듈 이름 : getBiggetStr(char\* str1, char\* str2)

3.7.2.7.1 기능

d

3.7.2.7.2 사용 변수

d

3.7.2.8 모듈 이름 : getMaxofFour(char\* str1, char\* str2, char\* str3, char\* str4)

3.7.2.8.1 기능

symbol table에서 label의 첫번째 알파벳은 각 알파벳당 최대 4개의 Head에 연결될 수 있다. 이 때 4개의 head에 검사해야할 이름의 label이 모두 포함되어있는 경우, 4개의 label중 가장 큰(Z에 가까운) 문자열의 번호를 반환한다.

3.7.2.8.2 사용 변수

**char\* str1, \*str2, \*str3, \*str4** – 비교할 4개의 문자열 변수이다.

3.7.2.9 모듈 이름 : getMaxofThree(char\* str1, char\* str2, chat\* str3)

3.7.2.9.1 기능

3.7.2.8의 getMaxofFour 함수와 같은 기능을 하며, 3개의 문자열을 비교해 가장 큰 label의 번호를 반환한다.

3.7.2.9.2 사용 변수

**char\* str1, \*str2, \*str3** – 비교할 3개의 문자열 변수이다.

3.7.2.10 모듈 이름 : printSymbol()

3.7.2.10.1 기능

Debugging을 위해 작성한 함수로, symbol table의 크기만큼 각 인덱스와 해당 인덱스에 연결된 node정보들을 출력하는 함수이다.

3.7.2.10.2 사용 변수

**char ch[4]** – 한 인덱스에 저장될 수 있는 4개의 알파벳을 저장하는 문자 배열 변수이다.

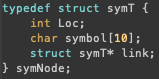
**int i, j** – loop counter로 사용되었다.

4. 구조체 정의

4.1 HISTORY

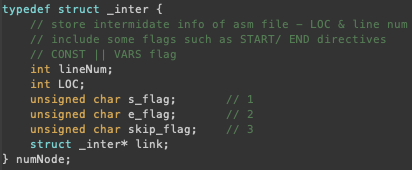
4.2 opNode

**4.3 symNode**

****

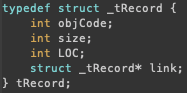
Symbol table을 구성하는 구조체 정보이다. 각 node들은 label과 Location counter를 저장한다.

**4.4 numNode**



pass1을 거치며 pass2를 진행하기 위한 중간 단계 파일을 생성하지 않고, linked list로 필요한 정보들을 저장하였다. 이 때 각 line number와 location counter를 저장하였으며, 추가로 START directive, END directive의 정보를 저장하기 위한 flag로 s\_flag, e\_flag를 사용하였다. 또한 현재 line이 comment이거나 blank line인 경우 추가적인 행동을 넘어가도 되므로 skip\_flag에 해당 정보를 나타낼 수 있도록 변수를 선언했다.

**4.5 tRecord**



text record를 저장하는 과정은 queue로 구현하였다. 이 때 queue의 각 node들이 갖는 정도는 위와 같다. 출력할 object code, object code의 크기인 size, 그리고 그 object code의 위치인 location counter를 저장한다.

5. 전역 변수 정의

5.1 HISTORY\* hisHead

5.2 opNode\*\* opTable

5.3 unsigned char\* MEMORY

5.4 END\_ADDR

**5.5 symNode\*\* SYMTAB**

Hash Table로 구성된 Symbol table의 시작 주소를 나타낸다.

**5.6 numNode\* numHead**

pass1을 거치며 pass2에서 사용할 필요한 정보를 저장하는 list의 시작주소를 나타낸다.

**5.7 tRecord\* tRHead**

object file의 text record부분을 출력하기 위해 queue에 정보를 저장하였다. 이 때 queue의 시작주소를 나타낸다.

**5.8 tRecord\* tRTail**

위의 queue의 마지막 노드를 가리킨다.

**6. 코드**

코드는 assembler.c, assembler\_functions.c, symbol.c 파일의 코드만 첨부하였다.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* assember.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ~ 190408 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "20171690.h"

int assemble(char\* filename) {

FILE\* asmFP = fopen(filename, "r");

if (!asmFP) {

// No input file

printf("%s: No such file in the directory\n", filename);

return 0;

}

// initialize Registers

A = 0; X = 0; L = 0; PC = 0;

SW = 0; B = -1; S = 0; T = 0; F = 0;

tRHead = NULL;

tRTail = NULL;

if (pass1(asmFP)) {

/\*\* printSymbol(); \*/

/\*\* printNums(); \*/

if (!pass2(asmFP, filename)) {

printf("Program halted in Pass 2\n");

remove(nameToListing(filename));

remove(nameToObj(filename));

return 0;

}

printf("output file : [%s], [%s]\n", nameToListing(filename), nameToObj(filename));

}

else {

printf("program halted in Pass 1\n");

return 0;

}

fclose(asmFP);

return 1;

}

int pass1(FILE\* fp) {

// Assign addresses to all statments in the program

// Save the values (addrs) assigned to all labels for use in Pass 2

// Perform some processing of assembler assignment, such as determining the

// length of data areas defined by BYTE, RESW, etc

// make symtab, assign LOC

int LOCCTR, startingAddr = 0, i;

int lineNum = 0;

int flag = 0;

int instructionSize;

char\* line = (char\*)malloc(MAX\_ASM\_LINE \* sizeof(char));

char programName[10];

fseek(fp, 0, SEEK\_SET); // move to the first

if ( !isStr(fgets(line, MAX\_ASM\_LINE, fp)) ) {

// file with no content

printf("No content in the file\n");

return 0;

}

lineNum++;

// initialize global vars

if (numHead) freeNums();

if (SYMTAB) freeSymTab();

numNode\* pLast = numHead;

while (line[0] == '.' || isBlankLine(line)) {

// skip comment lines & blank lines

memset(line, '\0', (int)sizeof(line));

if ( !isStr(fgets(line, MAX\_ASM\_LINE, fp)) ) {

// .asm file includes only comments

// do not create any symbol table

return 1;

}

pLast = addNum(lineNum, -1, pLast, 3);

lineNum++;

}

// First line except comments & blank line

char\*\* token = (char\*\*)malloc(MAX\_TOKEN\_NUM \* sizeof(char\*));

line = toUpperCase(line);

int tokenNum = tokenizeAsmFile(&token, line);

for (i = 0; i < tokenNum; i++) {

if (isDirective(token[i]) == 1) {

// if token[i] == "START"

if (i+1 >= tokenNum || tokenNum == 2) {

printf("Error occured at [%d] line: No name or starting address\n", lineNum);

return 0;

}

if ( (startingAddr = strToHex(token[i+1], 0)) == -1 ) {

printf("Error occured at [%d] line: Wrong Starting Address\n", lineNum);

return 0;

}

flag = 1;

break;

}

}

if (flag) {

// if there's START directive, save program name

// and get next line

strcpy(programName, token[0]);

pLast = addNum(lineNum, startingAddr, pLast, 1);

memset(line, '\0', (int)sizeof(line));

for (i = 0; i < MAX\_TOKEN\_NUM; i++) token[i] = NULL;

if ( !isStr(fgets(line, MAX\_ASM\_LINE, fp)) ) return 1;

lineNum++;

}

else strcpy(programName, "\0");

LOCCTR = startingAddr;

SYMTAB = (symNode\*\*)malloc(SYMTAB\_SIZE \* sizeof(symNode\*));

for (i = 0; i < SYMTAB\_SIZE; i++) SYMTAB[i] = NULL;

flag = 0;

while ( !feof(fp) ) {

// while not end of file

if (line[0] == '.' || isBlankLine(line)) {

// input is comment line or blank line

instructionSize = 0;

pLast = addNum(lineNum, -1, pLast, 3);

}

else {

line = toUpperCase(line);

tokenNum = tokenizeAsmFile(&token, line);

if (isDirective(token[0]) == 2) {

// if END

flag = 1;

break;

}

if (line[0] == ' ' || line[0] == '\t') {

// no label

instructionSize = getInstructionSize(token, lineNum, 0);

if (instructionSize == -1) return 0;

}

else {

// contains label, instruction, notes, ...

// add to symbol table

instructionSize = getInstructionSize(token, lineNum, 1);

if (instructionSize == -1) return 0;

if( !addSym(token[0], LOCCTR) ) {

printf("Error occured at [%d] line: use label already declared - %s\n", lineNum, token[0]);

return 0;

}

}

if (instructionSize == 0)

pLast = addNum(lineNum, -1, pLast, 3);

else

pLast = addNum(lineNum, LOCCTR, pLast, 0);

}

LOCCTR += instructionSize;

memset(line, '\0', (int)sizeof(line));

for (i = 0; i < MAX\_TOKEN\_NUM; i++) token[i] = NULL;

fgets(line, MAX\_ASM\_LINE, fp);

lineNum++;

}

free(line);

free(token);

if (!flag) {

printf("Error occured at [%d] line: No END Directive\n", lineNum);

return 0;

}

addNum(lineNum, LOCCTR, pLast, 2);

return 1;

}

int pass2(FILE\* fp, char\* filename) {

// Assemble instructions (translating opcodes and looking up addrs)

// Generate data values defined by BYTE, WORD, etc

// Perform processing of assembler directives not done during Pass 1

// Write obj program and assembly listing.

fseek(fp, 0, SEEK\_SET); // move to the start

int i, tokenNum, startingAddr, endAddr, size;

numNode\* pCurrent = numHead;

char\* line = (char\*)malloc(MAX\_ASM\_LINE \* sizeof(char));

char\*\* token = (char\*\*)malloc(MAX\_TOKEN\_NUM \* sizeof(char\*));

for (i = 0; i < MAX\_TOKEN\_NUM; i++) token[i] = NULL;

tRHead = NULL; tRTail = NULL;

FILE\* LF = fopen(nameToListing(filename), "w"); // listing file pointer

FILE\* OF = fopen(nameToObj(filename), "w"); // object file pointer

if (!pCurrent) {

printf("No content in the [%s] file", filename);

return 0;

}

while (pCurrent->skip\_flag) {

// while comment line or blank line

fgets(line, MAX\_ASM\_LINE, fp);

fprintf(LF, "\t %d\t\t\t%s", pCurrent->lineNum \* 5, line);

pCurrent = pCurrent->link;

if (!pCurrent) {

printf("Warning! - no content except comment/ blank lines\n");

return 1;

}

}

if (pCurrent->s\_flag) {

// START directive

fgets(line, MAX\_ASM\_LINE, fp);

tokenNum = tokenizeAsmFile(&token, line);

printLineinLST(pCurrent, token, 0, tokenNum, 0, LF, 1);

startingAddr = strToHex(token[2], 0);

if (startingAddr == -1) {

printf("Error occured at [%d] line: Wrong starting address - %s\n", pCurrent->lineNum, token[2]);

return 0;

}

numNode\* pMove;

for (pMove = numHead; pMove->link; pMove = pMove->link) ;

endAddr = pMove->LOC;

fprintf(OF, "H%-6s%06X%06X\n", token[0], startingAddr, endAddr - startingAddr);

pCurrent = pCurrent->link;

}

else {

// no START directive

// starting addr == 0. no program name

numNode\* pMove;

for (pMove = numHead; pMove->link; pMove = pMove->link) ;

startingAddr = 0;

endAddr = pMove->LOC;

fprintf(OF, "H 000000%06X\n", endAddr);

}

PC = startingAddr;

while (pCurrent) {

fflush(stdin);

for (i = 0; i < MAX\_TOKEN\_NUM; i++) token[i] = NULL;

memset(line, '\0', (int)sizeof(line));

fgets(line, MAX\_ASM\_LINE, fp);

if (pCurrent->skip\_flag) {

// if the line is comment or blank

if (line[0] == '.') fprintf(LF, "\t %d\t\t\t%s\n", pCurrent->lineNum \* 5, removeSpace(line));

else fprintf(LF, "\t %d\t\t\t\t%s\n", pCurrent->lineNum \* 5, removeSpace(line));

pCurrent = pCurrent->link;

continue;

}

if (pCurrent->e\_flag) {

// END directive

if (tRHead) dequeue(OF);

fprintf(OF, "E%06X\n", startingAddr);

fprintf(LF, "\t %d\t\t\t\t\t%s\n", pCurrent->lineNum \* 5, removeSpace(line));

break;

}

// TEXT RECORD

line = toUpperCase(line);

tokenNum = tokenizeAsmFile(&token, line);

int format, directiveNum;

int objCode;

// set Program Counter

numNode\* pMove = pCurrent->link;

while (pMove->link) {

if (pMove->LOC == -1) {

// comment line or blank line

pMove = pMove->link;

continue;

}

break;

}

PC = pMove->LOC;

if (line[0] == ' ' || line[0] == '\t') {

// no label

format = opcode(token[0], 3);

if ( (token[0])[0] == '+') {

format = 4;

token[0] += 1;

}

if ( format ) {

// There exist matching operation

objCode = getObjCode(token, &format, 0, pCurrent);

if (objCode == -1) return 0;

if (opcode(token[0], 4) == 0x68) {

// Load operand value in the B register

if ( !LDB(token, pCurrent->lineNum) ) return 0;

}

if (format == 4) token[0] -= 1;

printLineinLST(pCurrent, token, format, tokenNum, objCode, LF, 0);

enqueue(objCode, format, pCurrent->LOC, OF);

}

else {

// no matching operation

// it can be directives

// or wrong input

directiveNum = isDirective(token[0]);

if ( !directiveNum ) {

// wrong input

printf("Error occured at [%d] line: Wrong operation/ directive\n", pCurrent->lineNum);

return 0;

}

switch (directiveNum) {

case 3: // BYTE

objCode = getObjCode(&(token[0]), 0, 1, pCurrent);

if (objCode == -1) return 0;

size = (int)strlen(token[1]) - 3;

size = (token[1][0] == 'X') ? size / 2 : size;

printLineinLST(pCurrent, token, size, tokenNum, objCode, LF, 0);

enqueue(objCode, size, pCurrent->LOC, OF);

printf("Warning! - No label to point BYTE constant at [%d] line\n", pCurrent->lineNum);

break;

case 4: // WORD

objCode = getObjCode(&(token[0]), 0, 2, pCurrent);

if (objCode == -1) return 0;

printLineinLST(pCurrent, token, 3, tokenNum, objCode, LF, 0);

enqueue(objCode, 3, pCurrent->LOC, OF);

printf("Warning! - No label to point WORD constant at [%d] line\n", pCurrent->lineNum);

break;

case 5: // RESB

case 6: // RESW

printf("Warning! - No label to point variable at [%d] line\n", pCurrent->lineNum);

dequeue(OF);

case 7:

// No need to create opcode

printLineinLST(pCurrent, token, 0, tokenNum, 0, LF, 0);

pCurrent = pCurrent->link;

continue;

}

}

}

else {

// label!!

if (tokenNum == 1) {

printf("Error occured at [%d] line: no operation/ directive\n", pCurrent->lineNum);

return 0;

}

format = opcode(token[1], 3);

if ( (token[1])[0] == '+' ) {

format = 4;

token[1] += 1;

}

if ( format ) {

// exist matching operation

objCode = getObjCode(&(token[1]), &format, 0, pCurrent);

if (objCode == -1) return 0;

if (opcode(token[1], 4) == 0x68) {

// LOAD operand value to B register

if ( !LDB(token+1, pCurrent->lineNum) ) return 0;

}

if (format == 4) token[1] -= 1;

printLineinLST(pCurrent, token, format, tokenNum, objCode, LF, 1);

enqueue(objCode, format, pCurrent->LOC, OF);

}

else {

// no matching operation

// directives or wrong input

directiveNum = isDirective(token[1]);

if ( !directiveNum ) {

// wrong input

printf("Wrong!\n");

return 0;

}

// directive!

switch (directiveNum) {

case 3: // BYTE

objCode = getObjCode(&(token[1]), 0, 1, pCurrent);

if (objCode == -1) return 0;

size = (int)strlen(token[2]) - 3;

size = (token[2][0] == 'X') ? size / 2 : size;

printLineinLST(pCurrent, token, size, tokenNum, objCode, LF, 1);

enqueue(objCode, size, pCurrent->LOC, OF);

break;

case 4: // WORD

objCode = getObjCode(&(token[1]), 0, 2, pCurrent);

if (objCode == -1) return 0;

printLineinLST(pCurrent, token, 3, tokenNum, objCode, LF, 1);

enqueue(objCode, 3, pCurrent->LOC, OF);

break;

case 5: // RESB

case 6: // RESW

fprintf(LF, "\t %d\t %04X\t", pCurrent->lineNum \* 5, pCurrent->LOC);

for (i = 0; i < tokenNum; i++)

fprintf(LF, "\t%s", token[i]);

fprintf(LF, "\n");

dequeue(OF);

break;

case 7:

// No need to create opcode

fprintf(LF, "\t %d\t %04X\t", pCurrent->lineNum \* 5, pCurrent->LOC);

for (i = 0; i < tokenNum; i++)

fprintf(LF, "\t%s", token[i]);

fprintf(LF, "\n");

pCurrent = pCurrent->link;

continue;

}

}

}

pCurrent = pCurrent->link;

}

fclose(LF); fclose(OF);

free(line);

free(token);

return 1;

}

int getInstructionSize(char\*\* token, int lineNum, int isLabel) {

// return instructionSize

// if isLabel is on, it has label. else no

// if error occured, return 0

int opIdx = isLabel ? 1 : 0;

int size = 0;

if (!token[opIdx]) return -1;

if (token[opIdx][0] == '+')

// Foramt 4

return 4;

size = opcode(token[opIdx], 3);

if (size == 0) {

// token is no operation

// it can be directives

// or wrong input

switch (isDirective(token[opIdx])) {

//case 1: case 2:

case 3: // BYTE

size = byteSize(token[opIdx+1]);

break;

case 4: // WORD

size = wordSize(token[opIdx+1]);

break;

case 5: // RESB

size = resbSize(token[opIdx+1]);

break;

case 6: // RESW

size = reswSize(token[opIdx+1]);

break;

case 7: // directives without memory allocation

return 0;

default:

// wrong input

printf("Error occured at [%d] line: No matching operation code - %s\n", lineNum, token[opIdx]);

return -1;

}

if (size == -1) {

// wrong syntax.

// ex) no input || no hexa, ...

printf("Error occured at [%d] line: Wrong operand used\n", lineNum);

return -1;

}

}

return size;

}

int getObjCode(char\*\* token, int\* format, int type, numNode\* pCurrent) {

// format: operation code format / or size

// type: if 0, operation

// if 1, BYTE Const

// if 2, WORD Const

unsigned char n, i, x, b, p, e;

unsigned char r1, r2;

int reg;

int b1, b2, b3, b4;

int disp, addr;

int opCode, operand;

int objCode = 0;

int lc, target;

char\* operandStr[5];

char\* tmp;

char\* sym;

// initialize

for (lc = 0; lc < 5; lc++) operandStr[lc] = NULL;

if (type == 0) {

// find obj code for operation + operand

opCode = opcode(token[0], 4);

int commaFlag = isComma(token[1]); // if on, there exist comma in operand

if (commaFlag) {

tmp = (char\*)malloc(MAX\_ASM\_LINE \* sizeof(char));

strcpy(tmp, token[1]);

lc = 0;

operandStr[lc] = strtok(tmp, ",");

while (operandStr[lc++] && lc <= 5)

operandStr[lc] = strtok(NULL, ",");

if ( !operandStr[1] ) {

printf("Error occured at [%d] line: Wrong Syntax\n", pCurrent->lineNum);

return -1;

}

}

switch (\*format) {

case 1:

return opCode;

case 2:

// get register code

if (commaFlag) {

r1 = getRegNum(operandStr[0]);

r2 = getRegNum(operandStr[1]);

// if operand is not a register, then it's decimal n

// only for the case of SHIFTL/ SHIFTR

if (r1 == 0xFF) r1 = strToDecimal(operandStr[0]) - 1;

if (r2 == 0xFF) r2 = strToDecimal(operandStr[1]) - 1;

}

else {

r1 = getRegNum(token[1]);

if (r1 == 0xFF) r1 = strToDecimal(token[1]); // SVC operation

r2 = 0x00;

}

reg = (r1 << 4) + r2;

objCode = (opCode \* (1 << 8)) + reg;

return objCode;

case 3:

// operand

// get nixbpe

if (token[1]) {

sym = token[1];

// x bit

x = 0;

if (commaFlag) {

sym = operandStr[0];

lc = 1;

while (operandStr[lc]) {

if (strcmp(operandStr[lc], "X") == 0) {

x = 1;

break;

}

if (++lc == 5) break;

}

}

// n & i bits

if (token[1][0] == '#') {

// immediate

n = 0; i = 1;

sym += 1;

}

else if (token[1][0] == '@') {

// indirect

n = 1; i = 0;

sym += 1;

}

else {

// simple

n = 1; i = 1;

}

// b & p bits

target = findSym(sym);

if (target != -1) {

// operand point specific addr

if (target - PC >= -0xFFF && target - PC < 0x1000) {

// PC relative

disp = target - PC;

b = 0; p = 1;

}

else {

if (B == -1) printf("Warning! - incorrect access to B register at [%d] line: B register uninitialized\n", pCurrent->lineNum);

else if (target - B >= 0 && target- B < 0x1000) {

// B relative

disp = target - B;

b = 1; p = 0;

if (B == -1) {

// Base register uninitialized

printf("Warning! - B register used at [%d] line without ininitializing.\n", pCurrent->lineNum);

}

}

if (B == -1 || !(target - B >= 0 && target - B < 0x1000)) {

// format 4

\*format = 4;

printf("Warning! - format 4 used without '+' sign at [%d] line\n", pCurrent->lineNum);

b = 0; p = 0; e = 1;

addr = target;

b1 = opCode + n \* 2 + i;

b2 = (x << 7) + (b << 6) + (p << 5) + (e << 4) + (addr >> 16);

b3 = (addr >> 8) - ((addr / (1 << 16)) << 8);

b4 = addr % (1 << 8);

objCode = (b1 << 24) + (b2 << 16) + (b3 << 8) + b4;

return objCode;

}

}

}

else {

// operand is just value of addr

operand = strToHex(sym, 0);

if (operand == -1) {

printf("Error occured at [%d] line: incorrect use of undeclared label - %s\n", pCurrent->lineNum, sym);

return -1;

}

b = 0; p = 0;

disp = operand;

}

// e bit

e = 0;

}

else {

// no operand

n = 1; i = 1; x = 0;

b = 0; p = 0; e = 0;

disp = 0;

}

if (disp < 0)

disp = disp & 0x0FFF;

b1 = opCode + n \* 2 + i;

b2 = (x << 7) + (b << 6) + (p << 5) + (e << 4) + (disp >> 8);

b3 = disp % (1 << 8);

objCode = (b1 << 16) + (b2 << 8) + b3;

if (commaFlag) free(tmp);

return objCode;

case 4:

// get nixbpe

if (token[1]) {

// operand exist

sym = token[1];

// x bit

x = 0;

if (commaFlag) {

sym = operandStr[0];

if (strcmp(operandStr[1], "X") == 0)

x = 1;

}

// n & i bits

if (token[1][0] == '#') {

// immdeditae

n = 0; i = 1;

sym += 1;

}

else if (token[1][0] == '@') {

// indirect

n = 1; i = 0;

sym += 1;

}

else {

// simple

n = 1; i = 1;

}

// b & p bits

b = 0;

p = 0;

// e bits

e = 1;

operand = strToDecimal(sym);

if (operand == -1) {

// operand point specific addr

// find symbol

target = findSym(sym);

if (target == -1) {

printf("Error occured at [%d] line: incorrect use of undeclared label - %s\n", pCurrent->lineNum, sym);

return -1;

}

if (target < 0 || target > 0xFFFFFF) {

printf("Error occured at [%d] line: incorrect addr value\n", pCurrent->lineNum);

return -1;

}

addr = target;

}

else // operand is just value of addr

addr = operand;

b1 = opCode + n \* 2 + i;

b2 = (x << 7) + (b << 6) + (p << 5) + (e << 4) + (addr >> 16);

b3 = (addr >> 8) - ((addr / (1 << 16)) << 8);

b4 = addr % (1 << 8);

objCode = (b1 << 24) + (b2 << 16) + (b3 << 8) + b4;

}

if (commaFlag) free(tmp);

break;

}

}

else if (type == 1) {

// BYTE CONST

int size = 0;

int hex;

char\* strHex = (char\*)malloc(3 \* sizeof(char));

if (token[1][0] == 'C') {

size = (int)strlen(token[1]) - 3;

for (lc = 0; lc < size; lc++)

objCode += (((int)token[1][lc+2]) << (8 \* (size - lc - 1)));

}

else if (token[1][0] == 'X') {

size = ((int)strlen(token[1]) - 3) / 2;

for (lc = 0; lc < size; lc++) {

strHex[0] = token[1][lc\*2 + 2];

strHex[1] = token[1][lc\*2 + 3];

strHex[2] = '\0';

hex = strToHex(strHex, 1);

objCode += (hex << (8 \* (size - lc - 1)));

}

}

else {

printf("Error occured at [%d] line: Wrong Syntax for BYTE Const\n", pCurrent->lineNum);

return -1;

}

}

else if (type == 2) {

// WORD CONST

operand = strToHex(token[1], 1);

b1 = operand / (1 << 16);

b2 = operand / (1 << 8) - (b1 << 8);

b3 = operand % (1 << 8);

objCode = (b1 << 16) + (b2 << 8) + b3;

}

return objCode;

}

void printObjCode(int size, int objCode, FILE\* fp) {

if (size == 0) return ;

int hex;

for (int i = size-1; i >= 1; i--) {

hex = (objCode >> (8 \* i)) - ((objCode >> (8 \* i)) << (8 \* i));

hex &= ONE\_BYTE;

fprintf(fp, "%02X", hex);

}

hex = objCode % (1 << 8);

fprintf(fp, "%02X", hex);

}

int tokenizeAsmFile(char\*\*\* token, char\* input) {

// tokenize all the possible options into token

// return number of strings

int cnt = 0;

char\* tmp;

removeSpaceAroundComma(input);

input = removeSpace(input);

(\*token)[cnt] = strtok(input, " \t");

while (cnt < MAX\_TOKEN\_NUM && (\*token)[cnt]) {

tmp = strtok(NULL, "\0");

if (!tmp) break;

(\*token)[++cnt] = strtok(removeSpace(tmp), " \t");

}

cnt++;

return cnt;

}

int removeSpaceAroundComma(char\* input) {

// if comma included, return 1, else 0

int isComma = 0;

int cnt;

for (int i = 1; input[i]; i++) {

if (input[i] == ',') {

isComma = 1;

if ( isWhiteSpace(input[i-1]) ) {

cnt = 0; // num of spaces

while ( i - 1 - cnt >= 0 && isWhiteSpace(input[i - 1 - cnt]) ) cnt++;

for (int j = 0; j < (int)strlen(input) - i; j++)

input[i - cnt + j] = input[i + j];

input[(int)strlen(input) - cnt] = '\0';

i -= cnt;

}

if ( isWhiteSpace(input[i+1]) ) {

cnt = 0;

while ( input[i + 1 + cnt] && isWhiteSpace(input[i + 1 + cnt]) ) cnt++;

for (int j = 0; j < (int)strlen(input) - i - 1 - cnt; j++)

input[i + 1 + j] = input[i + 1 + cnt + j];

input[(int)strlen(input) - cnt] = '\0';

i -= cnt;

}

}

}

return isComma;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\* assember\_functions.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ~ 190408 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "20171690.h"

int getRegNum(char\* reg) {

if (!reg) return 0xFF;

if (strcmp(reg, "A") == 0) return 0x00;

if (strcmp(reg, "X") == 0) return 0x01;

if (strcmp(reg, "L") == 0) return 0x02;

if (strcmp(reg, "B") == 0) return 0x03;

if (strcmp(reg, "S") == 0) return 0x04;

if (strcmp(reg, "T") == 0) return 0x05;

if (strcmp(reg, "F") == 0) return 0x06;

if (strcmp(reg, "PC") == 0) return 0x08;

if (strcmp(reg, "SW") == 0) return 0x09;

return 0xFF;

}

int LDB(char\*\* token, int lineNum) {

// Load address to B register

// if success, return 1. else return 0

int tmp;

if ( !token[1] ) {

printf("Error occured at [%d] line: Can't load value to B register\n", lineNum);

return 0;

}

if (token[1][0] == '#') // immediate Addressing

B = findSym(token[1]+1);

else if (token[1][0] == '@') {

// indirect Addressing

tmp = findSym(token[1]+1);

numNode\* pMove = numHead;

while (pMove) {

if (pMove->LOC == tmp) break;

pMove = pMove->link;

if (!pMove) {

printf("Error occured at [%d] line: Can't access value at %s Address\n", lineNum, token[1]);

return 0;

}

}

printf("Cannot load BASE register: memory not initialized at %s Address\n", token[1]);

}

else if ( (tmp = strToHex(token[1], 0)) != -1)

B = tmp;

if (B == -1) {

if ( (tmp = strToHex(token[1]+1, 0)) != -1)

B = tmp;

else {

printf("Error occured at [%d] line: incorrect use of undeclared label - %s\n", lineNum, token[1]+1);

}

}

return 1;

}

void printLineinLST(numNode\* pCurrent, char\*\* token, int size, int tokenNum, int objCode, FILE\* LF, int isLabel) {

fprintf(LF, "\t %d", pCurrent->lineNum \* 5);

//if (pCurrent->lineNum <= 1) fprintf(LF, "\t");

fprintf(LF, "\t ");

if (pCurrent->LOC == -1) fprintf(LF, "\t\t");

else fprintf(LF, "%04X\t", pCurrent->LOC);

if (!isLabel)

fprintf(LF, "\t");

int i;

for (i = 0; i < tokenNum; i++) {

fprintf(LF, "\t%s", token[i]);

//if ( (int)strlen(token[i]) < 4 ) fprintf(LF, "\t");

}

if (tokenNum > 0)

if ( (int)strlen(token[--i]) < 8 ) fprintf(LF, "\t");

if (tokenNum == 1)

fprintf(LF, "\t");

fprintf(LF, "\t\t");

printObjCode(size, objCode, LF);

fprintf(LF, "\n");

}

int isWhiteSpace(char ch) {

return (ch == ' ' || ch == '\n' || ch == '\t') ? 1 : 0;

}

int isBlankLine(char\* input) {

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

if (!isWhiteSpace(input[i])) return 0;

return 1;

}

int isDirective(char\* token){

// check if the token is a directive

// if it is, return corresponding number. else 0

// START, END, BYTE, WORD, RESB, RESW

// BASE

if (!token) return 0;

if (strcmp(token, "START") == 0) return 1;

if (strcmp(token, "END") == 0) return 2;

if (strcmp(token, "BYTE") == 0) return 3;

if (strcmp(token, "WORD") == 0) return 4;

if (strcmp(token, "RESB") == 0) return 5;

if (strcmp(token, "RESW") == 0) return 6;

if (strcmp(token, "BASE") == 0) return 7;

return 0;

}

int isComma(char\* input) {

if (!input) return 0;

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

if (input[i] == ',') return 1;

return 0;

}

char\* nameToListing(char\* filename) {

char\* lstName = (char\*)malloc(50 \* sizeof(char));

for (int i = 0; i < (int)strlen(filename); i++) {

if (filename[i] != '.')

lstName[i] = filename[i];

else break;

}

strcat(lstName, ".lst");

return lstName;

}

char\* nameToObj(char\* filename) {

char\* objName = (char\*)malloc(50 \* sizeof(char));

for (int i = 0; i < (int)strlen(filename); i++) {

if (filename[i] != '.')

objName[i] = filename[i];

else break;

}

strcat(objName, ".obj");

return objName;

}

char toUpper(char ch){

if (ch >= 'a' && ch <= 'z') ch -= 'a' - 'A';

return ch;

}

char\* toUpperCase(char\* input) {

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

if (input[i] >= 'a' && input[i] <= 'z') input[i] -= 'a' - 'A';

return input;

}

int byteSize(char\* input) {

if (!input) return -1;

int size;

if (input[0] == 'C') {

if (input[1] == input[(int)strlen(input)-1] && input[1] == '\'') {

size = (int)strlen(input) - 3;

}

else return -1;

}

if (input[0] == 'X') {

if (input[1] == input[(int)strlen(input)-1] && input[1] == '\'') {

size = (int)strlen(input) - 3;

size = (size % 2 == 0) ? size / 2 : size / 2 + 1;

}

else return -1;

}

return size;

}

int wordSize(char\* input) {

if (!input) return -1;

int size = WORD\_SIZE;

return size;

}

int resbSize(char\* input) {

if (!input) return -1;

int size;

if ( (size = strToDecimal(input)) == -1 ) return -1;

return size;

}

int reswSize(char\* input) {

if (!input) return -1;

int size;

if ( (size = strToHex(input, 0)) == -1 ) return -1;

return size \* WORD\_SIZE;

}

numNode\* addNum(int lineNum, int LOC, numNode\* pLast, int flag\_type) {

// flag\_type: 0 - normal text record

// 1 - START directive

// 2 - END directive

// 3 - comment/ blank line

numNode\* pNew = (numNode\*)malloc(sizeof(numNode));

pNew->lineNum = lineNum; pNew->LOC = LOC;

switch (flag\_type) {

case 0:

pNew->s\_flag = 0; pNew->e\_flag = 0; pNew->skip\_flag = 0;

break;

case 1:

pNew->s\_flag = 1; pNew->e\_flag = 0; pNew->skip\_flag = 0;

break;

case 2:

pNew->s\_flag = 0; pNew->e\_flag = 1; pNew->skip\_flag = 0;

break;

case 3:

pNew->s\_flag = 0; pNew->e\_flag = 0; pNew->skip\_flag = 1;

break;

}

pNew->link = NULL;

if (!numHead) {

numHead = pNew;

return pNew;

}

pLast->link = pNew;

return pNew;

}

void freeNums() {

if (!numHead) return ;

numNode\* pFree = numHead;

while (numHead) {

pFree = numHead;

numHead = numHead->link;

free(pFree);

}

numHead = NULL;

}

void printNums() {

numNode\* pMove;

for (pMove = numHead; pMove; pMove = pMove->link)

printf("\t%d\t%06X\t%X %X %X\n", pMove->lineNum, pMove->LOC, pMove->s\_flag, pMove->e\_flag, pMove->skip\_flag);

}

void enqueue(int objCode, int size, int LOC, FILE\* OF) {

tRecord\* pNew = (tRecord\*)malloc(sizeof(tRecord));

pNew->objCode = objCode;

pNew->size = size;

pNew->LOC = LOC;

pNew->link = NULL;

if ( !tRHead ) {

tRHead = pNew;

tRTail = pNew;

return ;

}

int len = 0;

tRecord\* pMove;

for (pMove = tRHead; pMove; pMove = pMove->link)

len += pMove->size;

if (len + size > MAX\_OBJ\_TRECORD) {

dequeue(OF);

tRHead = pNew;

tRTail= pNew;

return ;

}

tRTail->link = pNew;

tRTail = pNew;

}

void dequeue(FILE\* OF) {

// pop node from the queue

// and print obj codes in the .obj file

if (!tRHead) return ;

// print T, starting LOC, and Length of the record

int len = 0;

tRecord\* pMove;

for (pMove = tRHead; pMove; pMove = pMove->link)

len += pMove->size;

fprintf(OF, "T%06X%02X", tRHead->LOC, len);

while (tRHead) {

printObjCode(tRHead->size, tRHead->objCode, OF);

tRecord\* pFree = tRHead;

tRHead = tRHead->link;

free(pFree);

}

fprintf(OF, "\n");

tRHead = NULL;

tRTail = NULL;

}

void freeQueue() {

if (tRHead) {

tRecord\* pFree = tRHead;

while (tRHead) {

pFree = tRHead;

tRHead = tRHead->link;

free(pFree);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* symbol.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ~ 190408 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "20171690.h"

int symbol() {

// print symbol table of recently assembled file

if (!SYMTAB) {

// no symbol table

printf("assemble file first!\n");

return 0;

}

int idx[4];

int ptIdx, cnt, sign;

symNode\* ptPtr[28];

for (int i = 0; i < 28; i++)

ptPtr[i] = SYMTAB[i];

for (char ch = 'Z'; ch >= 'A'; ch--) {

ptIdx = -1;

cnt = 0; // num of head which has no corresponding char out of 4

sign = 0;

for (int i = 0; i < 4; i++) {

idx[i] = (ch - 'A') % 7 + i \* 7;

if (!ptPtr[idx[i]]) {

cnt++;

sign += 1 << i;

}

else if ( (ptPtr[idx[i]]->symbol)[0] != ch ) {

cnt++;

sign += 1 << i;

}

}

if (cnt == 0) {

// all four heads have chars

while (ptPtr[idx[0]] && ptPtr[idx[1]] && ptPtr[idx[2]] && ptPtr[idx[3]]) {

if ( (ptIdx = getMaxofFour(ptPtr[idx[0]]->symbol, ptPtr[idx[1]]->symbol, ptPtr[idx[2]]->symbol, ptPtr[idx[3]]->symbol)) == -1) return 0;

printf("\t%s\t%06X\n", ptPtr[idx[ptIdx]]->symbol, ptPtr[idx[ptIdx]]->Loc);

ptPtr[idx[ptIdx]] = ptPtr[idx[ptIdx]]->link;

if (!ptPtr[idx[ptIdx]]) break;

if ( (ptPtr[idx[ptIdx]]->symbol)[0] != ch ) break;

}

}

if (cnt <= 1) {

if (ptIdx == -1) {

switch (sign) {

case 4: idx[2] = idx[3]; break;

case 2: idx[1] = idx[3]; break;

case 1: idx[0] = idx[3]; break;

}

}

else

idx[ptIdx] = idx[3];

while (ptPtr[idx[0]] && ptPtr[idx[1]] && ptPtr[idx[2]]) {

if ( (ptIdx = getMaxofThree(ptPtr[idx[0]]->symbol, ptPtr[idx[1]]->symbol, ptPtr[idx[2]]->symbol)) == -1) return 0;;

printf("\t%s\t%06X\n", ptPtr[idx[ptIdx]]->symbol, ptPtr[idx[ptIdx]]->Loc);

ptPtr[idx[ptIdx]] = ptPtr[idx[ptIdx]]->link;

if (!ptPtr[idx[ptIdx]]) break;

if ( (ptPtr[idx[ptIdx]]->symbol)[0] != ch ) break;

}

}

if (cnt <= 2) {

if (ptIdx == -1) {

switch (sign) {

case 12: break;

case 10: idx[1] = idx[2]; break;

case 6: idx[1] = idx[3]; break;

case 9: idx[0] = idx[2]; break;

case 5: idx[0] = idx[3]; break;

case 3: idx[0] = idx[2]; idx[1] = idx[3]; break;

}

}

else

idx[ptIdx] = idx[2];

while (ptPtr[idx[0]] && ptPtr[idx[1]]) {

ptIdx = getBiggerStr(ptPtr[idx[0]]->symbol, ptPtr[idx[1]]->symbol);

printf("\t%s\t%06X\n", ptPtr[idx[ptIdx]]->symbol, ptPtr[idx[ptIdx]]->Loc);

ptPtr[idx[ptIdx]] = ptPtr[idx[ptIdx]]->link;

if (!ptPtr[idx[ptIdx]]) break;

if ( (ptPtr[idx[ptIdx]]->symbol)[0] != ch ) break;

}

}

if (cnt <= 3) {

if (ptIdx == -1) {

switch (sign) {

case 7: idx[0] = idx[3]; break;

case 11: idx[0] = idx[2]; break;

case 13: idx[0] = idx[1]; break;

}

}

else

idx[ptIdx] = idx[1];

while (ptPtr[idx[0]]) {

if ( (ptPtr[idx[0]]->symbol)[0] != ch ) break;

if (!ptPtr[idx[0]]) break;

printf("\t%s\t%06X\n", ptPtr[idx[0]]->symbol, ptPtr[idx[0]]->Loc);

ptPtr[idx[0]] = ptPtr[idx[0]]->link;

}

}

}

return 1;

}

int addSym(char\* label, int LOC) {

// add symbol to the SYMTAB

// if label already exist, return 0

// if add successfully, return 1

if (findSym(label) != -1) {

// label exist

return 0;

}

if (label[0] < 'A' || label[0] > 'Z') return 0;

// create new node

symNode\* pNew = (symNode\*)malloc(sizeof(symNode));

pNew->Loc = LOC;

strcpy(pNew->symbol,label);

pNew->link = NULL;

int idx = symHashFunc(label);

if (!SYMTAB[idx]) {

// no node at the header

SYMTAB[idx] = pNew;

return 1;

}

symNode\* pMove = SYMTAB[idx];

if ( getBiggerStr(pNew->symbol, pMove->symbol) == 0 ) {

// header - pNew - pMove - ...

pNew->link = pMove;

SYMTAB[idx] = pNew;

return 1;

}

else {

// header - pMove - pNew - ...

pNew->link = pMove->link;

pMove->link = pNew;

if ( !(pNew->link) ) return 1;

}

// if the header has more than two nodes

symNode\* ptmp = pNew->link;

while (ptmp) {

// header - ... - pMove - pNew - ptmp - ...

if ( getBiggerStr(pNew->symbol, ptmp->symbol) == 0)

return 1;

pMove->link = ptmp;

pNew->link = ptmp->link;

ptmp->link = pNew;

// ... - pMove - ptmp - pNew - ...

pMove = pMove->link;

ptmp = pNew->link;

}

return 1;

}

void freeSymTab() {

symNode\* pFree, \*ptmp;

for (int i = 0; i < SYMTAB\_SIZE; i++) {

ptmp = SYMTAB[i];

while (ptmp) {

pFree = ptmp;

ptmp = ptmp->link;

free(pFree);

}

}

free(SYMTAB);

SYMTAB = NULL;

}

int symHashFunc(char\* label) {

// Assum proper label input

int idx = 0;

int rmd = (label[0] - 'A') % 7;

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

idx += label[i];

int quo = idx % 4;

idx = quo \* 7 + rmd;

return idx;

}

int findSym(char\* label) {

// if found, return Loc of the label. else -1

if (label[0] < 'A' || label[0] > 'Z') return -1;

if (!SYMTAB) {

// no Symbol Table

printf("No Symbol Table. Please assemble file first\n");

return -1;

}

if (!isStr(label)) {

printf("Wrong Label\n");

return -1;

}

int idx = symHashFunc(label);

symNode\* pMove;

for (pMove = SYMTAB[idx]; pMove; pMove = pMove->link) {

if (strcmp(pMove->symbol, label) == 0) {

return pMove->Loc;

}

}

return -1;

}

int isStr(char\* Str) {

// if str contains no character, return 0. else 1

return Str ? 1 : 0;

}

int getBiggerStr(char\* str1, char\* str2){

// Bigger means closer to Z

// return 0 : str1 > str2

// return 1 : str1 < str2

int len1 = (int)strlen(str1);

int len2 = (int)strlen(str2);

int shorterLen = (len1 > len2) ? len2 : len1;

// if flag set, str1 > str2

int i;

for (i = 0; i < shorterLen; i++) {

if (str1[i] > str2[i])

return 0;

else if (str1[i] < str2[i])

return 1;

}

// longer one is bigger

if (shorterLen == len1) return 1;

return 0;

}

int getMaxofFour(char\* str1, char\* str2, char\* str3, char\* str4) {

if ( getBiggerStr(str2, str1) \* getBiggerStr(str3, str1) \* getBiggerStr(str4, str1) ) return 0;

if ( getBiggerStr(str1, str2) \* getBiggerStr(str3, str2) \* getBiggerStr(str4, str2) ) return 1;

if ( getBiggerStr(str1, str3) \* getBiggerStr(str2, str3) \* getBiggerStr(str4, str3) ) return 2;

if ( getBiggerStr(str1, str4) \* getBiggerStr(str2, str4) \* getBiggerStr(str3, str4) ) return 3;

return -1;

}

int getMaxofThree(char\* str1, char\* str2, char\* str3) {

if ( getBiggerStr(str2, str1) \* getBiggerStr(str3, str2) ) return 0;

if ( getBiggerStr(str1, str2) \* getBiggerStr(str3, str2) ) return 1;

if ( getBiggerStr(str1, str3) \* getBiggerStr(str2, str3) ) return 2;

return -1;

}

void printSymbol() {

if (!SYMTAB) {

printf("No SYMTAB\n");

return ;

}

symNode\* pMove;

char ch[4];

for (int i = 0; i < SYMTAB\_SIZE; i++) {

if (i % 7 == 5 || i % 7 == 6) {

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

ch[j] = 'A' + i % 7 + j \* 7;

printf("%d [%c %c %c ]: ", i, ch[0], ch[1], ch[2]);

}

else {

for (int j = 0; j < 4; j++)

ch[j] = 'A' + i % 7 + j \* 7;

printf("%d [%c %c %c %c]: ", i, ch[0], ch[1], ch[2], ch[3]);

}

pMove = SYMTAB[i];

if (pMove) {

for ( ; pMove->link; pMove = pMove->link)

printf("[%s, %X] -> ", pMove->symbol, pMove->Loc);

printf("[%s, %X]", pMove->symbol, pMove->Loc);

}

if (i % 7 == 6) printf("\n");

printf("\n");

}

}