**과목명: 시스템프로그래밍**

**2분반**

**<<Project #2>>**

**서강대학교 수학과**

**20171290**

**이승은**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. Int type\_filename\_func
   2. Void symbol\_func
   3. Void free\_symbol
   4. Int pass\_one
   5. Int pass\_two
   6. Symb\_node\* find\_symbol
   7. Void push\_symbol
   8. Void push\_modification
   9. Int write\_lst
   10. Int write\_obj
4. **전역 변수 정의**
5. **코드 설명**

**1. 프로그램 개요**

이 프로그램은 SIC/XE의 assembly program source file을 입력 받아서 listing file과 object file을 생성하고 file 생성 중 사용 된 symbol을 모아놓은 symbol table을 생성하는 프로그램이다. 지난 번에 구현 된 shell에 기능을 추가하여 프로그램을 만들고, 다음과 같은 명령어를 수행하도록 프로그램을 만든다.

1) shell 관련 명령어

(1) h[elp]

Shell에서 수행 가능 한 명령어를 모두 출력한다. 지난 프로젝트의 help의 출력 결과에

Assemble filename, type filename, symbol의 3개의 명령어를 추가한다.

(2) type filename

현재 directory에 있는 filename에 해당하는 file을 읽어 화면에 출력한다.

만약 현재 directory에 filename에 해당하는 file이 존재하지 않는다면 에러 메시지를

출력한다.

2) SIC/XE 어셈블러 명령어

(1) assemble filename

Filename에 해당하는 source file을 읽어서 object file과 listing file을 생성한다.

그리고 assembly source file에서 정의 된 symbol들을 LOC 값과 함께 symbol table에

저장한다. Listing file은 5의 배수로 assembly code의 line번호 또한 같이 저장한다.

이 때, listing file의 확장자는 .lst, object file의 확장자는 .obj이다.

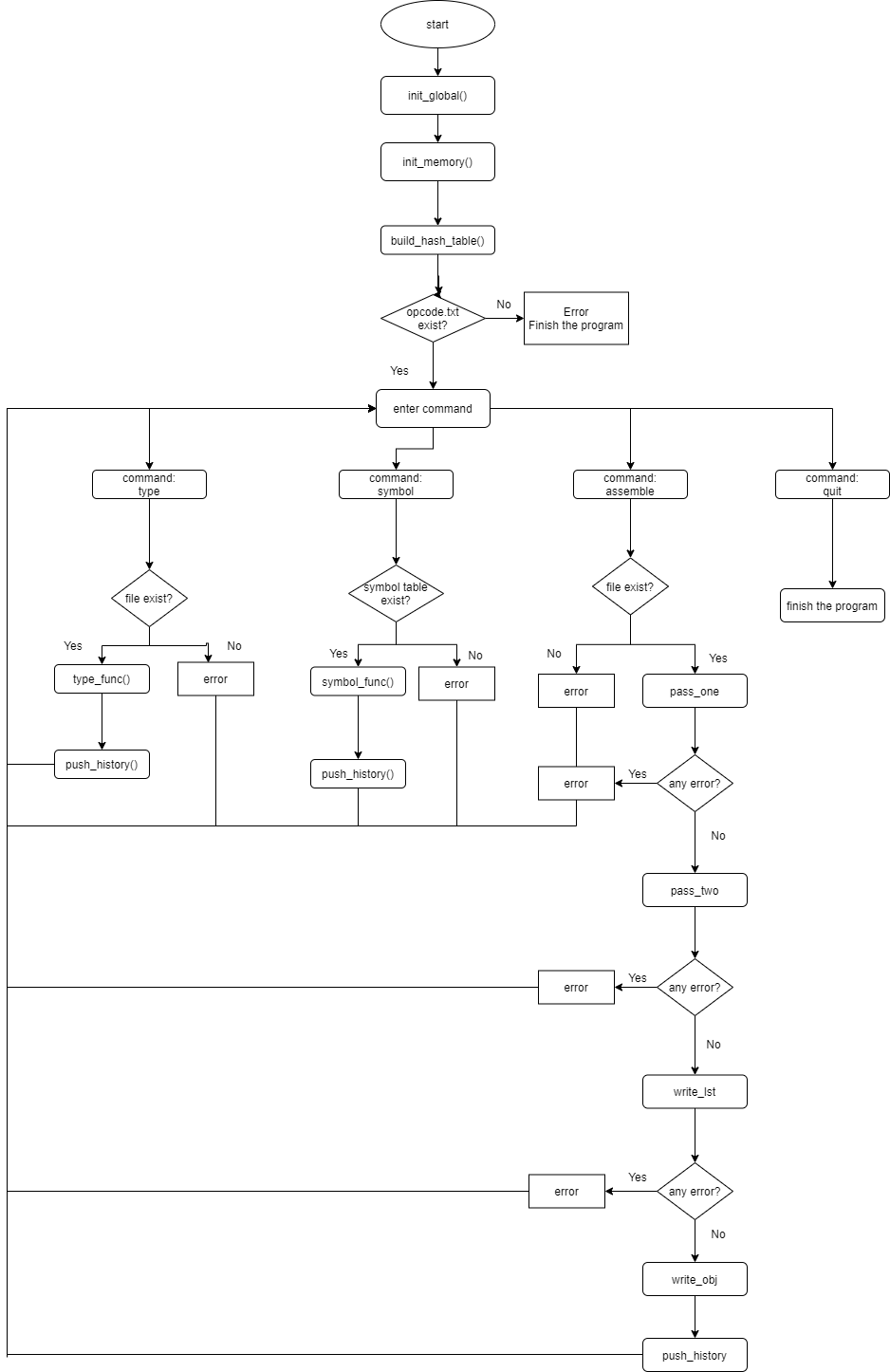
(2) symbol

가장 최근에 assemble된 file의 symbol table을 출력한다. 이 때, symbol들은 오름차순

으로 출력한다.

**2. 프로그램 설명**

2.1 프로그램의 흐름도



Quit 명령어를 받을 때까지 프로그램은 계속 수행되며, 2차 프로젝트에 추가 된 내용만 추가적으로 흐름도를 작성하였다.

**3. 모듈 정의**

3.1 Int type\_filename\_func(char\* filename)

입력: 출력하고자 하는 file명

출력: parameter로 들어 온 file명에 해당하는 file의 텍스트 내용

사용변수: FILE \*fp: parameter로 들어 온 파일의 메모리를 가리키는 포인터 변수

Char c: parameter로 들어온 filename에 해당하는 file의 글자 하나하나를 읽는

데 사용되는 변수

기능: parameter로 들어 온 file명에 해당하는 file이 현재 directory에 있는지 확인하고,

있다면, 해당 텍스트 파일의 내용을 화면에 출력한다.

3.2 void symbol\_func()

입력: NULL

출력: 가장 최근에 object file을 생성하면서 만들어 진 symbol table

사용변수: symb\_node\* symbol\_list: symbol table의 head pointer

Symb\_node\* walk: symbol table을 순회하면서 각 출력할 node를 가리키는 변수

기능: 가장 최근에 object file을 생성할 때 만들어진 symbol table을 순회하면서 오름차순

으로 각 symbol과 symbol에 해당하는 LOC의 값을 출력한다.

3.3 void free\_symb()

입력: NULL

출력: NULL

사용변수: symb\_node\* symbol\_list: symbol table의 head pointer

Symb\_node\* temp: 메모리 해제 할 symb\_node를 가리키는 포인터

기능: object file을 생성하기 전, 이전에 생성 된 symbol table이 있다면 해당 table에 할당

된 메모리를 모두 해제한다.

3.4 int pass\_one(char\* filename)

입력: assembly source file명

출력: assemble 과정에서 오류가 생긴다면 해당 오류가 생긴 line 번호와 오류의 원인

사용변수: FILE \*fp: assemble 할 source file의 내용을 가리키는 포인터

Symbol\_list: assembly source file에 정의 된 symbol의 정보를 저장 할 linked

list의 head pointer.

Char temp[200]: assembly source file에서 code를 한 줄 읽어와 해당 code를

저장하는 임시 변수

Int line: 해당 assembly code의 line 번호를 저장해놓는 변수

Int address: 해당 assembly label이나 instruction에 해당하는 LOC 값을

저장하는 변수

Int number: 해당 assembly code line에 있는 argument의 수를 저장하는 변수

Int index: 해당 assembly code line을 space에 따라 parsing하기 위해서

사용되는 indexing 변수

Int size: 해당 assembly code line의 string 크기를 저장하는 변수

Int ret: pass\_one이 성공적으로 수행되었는지 여부를 return하는 변수

List\_node\*: 각 assembly code에 해당하는 LOC와 instructions를 저장하는 변수

Op\_node\* found: instruction에 대한 opcode 정보를 저장하는 변수

기능: 파라미터로 들어온 파일 명에 해당하는 assembly source file을 읽어 들인 후, 각

Assembly code의 instruction 또는 symbol에 해당하는 line number, LOC, assembly

Code를 저장한다. 이 때, 정의되지 않은 instruction이 들어온 경우 오류를

출력한다. 그리고 symbol이 주어지면, 해당 symbol을 symbol table에 저장한다.

3.5 int pass\_two()

입력: NULL

출력: 성공적으로 assemble이 되었는지 여부

사용변수: int ret: pass\_two가 성공적으로 수행됐는지 여부를 반환해주는 함수

Int size: instruction 또는 자료 형의 string 크기를 저장하기 위해 사용되는 변수

List\_node\* walk: object를 생성 할 node를 가리키는 포인터

Boolean ch: argument에 character가 있는지 여부를 나타내는 변수

Symb\_node\* f: variable 명에 해당하는 symbol의 정보를 저장하는 변수

Int base\_reg: base register가 사용될 경우, base 주소 값을 저장하는 변수

Op\_node\* found: instruction에 해당하는 opcode의 정보를 저장하는 변수

Int opni: object 코드의 생성을 위해 object코드의 두 번째 half byte에 해당하는

값을 저장하는 변수

Int n,i,x,b,p,e: SIC/XE machine의 format 4에 들어가는 nixbpe의 각 bit값을 저장

하는 변수

Int pc: 해당 assembly instruction이 pc relative여부 확인을 위해 사용되는 pc 값

을 저장하는 변수

Int xbpe: object 코드의 생성을 위해 object 코드의 세 번째 half byte에 해당하는

값을 저장하는 변수

기능: LOC와 instruction을 저장하고 있는 list\_node를 읽으면서 해당 assembly code에

Object code를 형성하여 해당 list\_node에 다시 저장한다. 이 때, 선언되지 않은

변수나 instruction의 사용, 또는 중복 된 symbol을 사용한 경우 오류를 출력한다.

3.6 symb\_node\* find\_symbol(char\* symbol)

입력: 찾고자 하는 symbol

출력: NULL

사용변수: symb\_node\* walk: linked list로 만들어진 symbol table을 순회하기 위한 변수

기능: symbol table을 순회하면서 파라미터로 들어온 symbol에 해당하는 node가 있으면

해당 node를 return한다.

3.7 void push\_symbol(char\* symbol, int address)

입력: 추가할 symbol 이름과 주소

출력: NULL

사용변수: symb\_node\* new: 파라미터로 들어온 symbol을 저장 할 node

Int flag: 새로운 symbol이 추가 되었는지 여부를 저장하는 변수

기능: 파라미터로 들어온 symbol을 파라미터로 들어 온 address값과 함께 symbol table에

오름차순으로 추가한다.

3.8 void push\_modification(list\_node\* list)

입력: relocation이 필요 한 list\_node

출력: NULL

사용변수: mod\_node\* new: 파라미터로 들어온 list의 정보를 저장 할 node

기능: parameter로 들어 온 relocation이 필요한 list\_node의 정보를 이용하여 object code

에서 modification part 작성을 위한 node를 생성한다.

3.9 int write\_lst(char\* filename)

입력: listing file을 작성하고자 하는 assembly source file의 이름

출력: assembly source file을 assemble하고 난 뒤 생기는 listing file

사용변수: FILE \*fp: 작성 할 file의 메모리를 가리키는 포인터

List\_node\* walk: listing file 작성을 위해 listing file의 정보를 가진 node를

순회하는 포인터

기능: listing file에 작성 할 내용을 저장하고 있는 linked list를 순회하면서 listing file

을 작성한다. 작성한 파일 명은 filename.lst로 한다.

3.10 int write\_obj(char\* filename)

입력: object file을 작성하고자 하는 assembly source file의 이름

출력: assembly source file을 assemble 하고 난 뒤 생기는 object file

사용변수: FILE \*fp: 작성 할 file의 메모리를 가리키는 포인터

List\_node\* start: object의 text part에서 한 text line의 시작부분을

가리키는 포인터

List\_node\* walk: object text part에서 한 text line의 끝 부분을

가리키는 포인터

Mod\_node\* walk2: object modification part를 작성하기 위해 modification

List를 순회하는 포인터

기능: assemble source file을 assemble 하고 난 뒤 생성되는 object file을 작성하는

함수. 작성한 파일 명은 filename.obj로 한다.

**4. 전역 변수 정의**

(프로젝트 1이후에 추가 된 전역변수)

Typedef struct symb\_node{

Char symbol[10];

Int addr;

Struct symb\_node\* ptr;

}

-symbol table linked list를 구성하는 node이다. Symbol 변수에는 symbol 명이, addr에는 해당

Symbol의 LOC값이 저장된다.

Typedef struct list\_node{

Int argnum;

Int line;

Int loc;

Char inst[50];

Char assembly[5][30];

Boolean obj\_flag;

Char objcode[10];

Struct list\_node \*ptr;

}

-listing file과 object file을 생성하기 위한 정보를 가지고 있는 node이다. Argnum은 assembly instruction의 argument의 수를, line은 listing file에 작성 될 assembly code의 line 번호를, loc는 각 assembly symbol 또는 instruction의 location값을, inst[50]은 assembly code 한 줄을 저장한다. 그리고 assembly[5][30]은 각 argument에 해당하는 값을, obj\_flag는 object code 생성 여부를, objcode는 해당 assembly code의 object 코드 값을 저장한다.

Typedef struct mod\_node{

Int address;

Int num;

Struct mod\_node\* ptr;

}

-object file의 modification part 작성을 위한 정보를 가지고 있는 node이다. address에는 modification이 필요한 부분의 시작 주소를, num은 시작 주소로부터 읽어야 할 half byte의 수를 저장하고 있다.

Symb\_node \*symbol\_list

-symbol table의 head pointer이다.

List\_node \*list\_head;

-listing file과 object file을 생성하기 위한 정보를 가지고 있는 list의 head pointer이다.

List\_node \*list\_tail;

-listing file과 object file을 생성하기 위한 정보를 가지고 있는 list의 tail pointer이다.

Char program\_title[6];

-프로그램 명을 저장하고 있는 변수

Int assemble\_error;

-assemble 과정에서 생긴 오류의 정보를 저장하는 변수다.

1은 undefined variable error, 2는 same variable name error, 3은 undefined instruction

Error를 의미한다.

Int program\_length;

-program의 길이 정보를 저장한다.

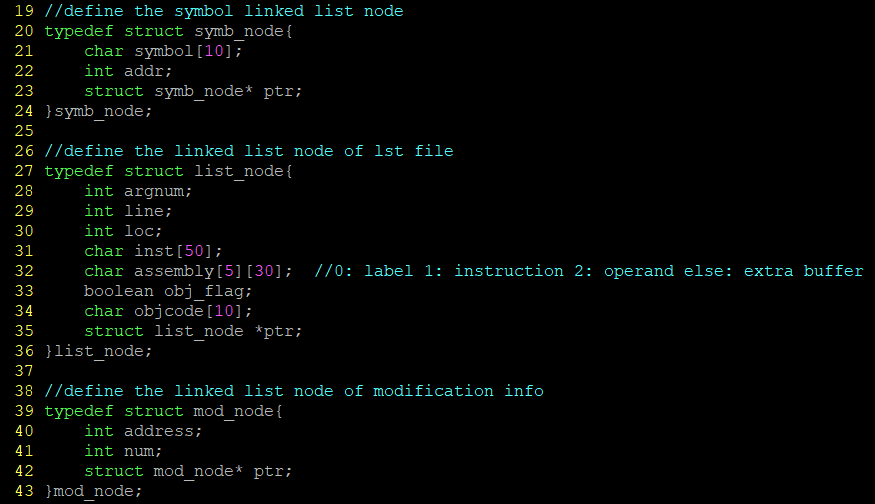
Int base\_reg;

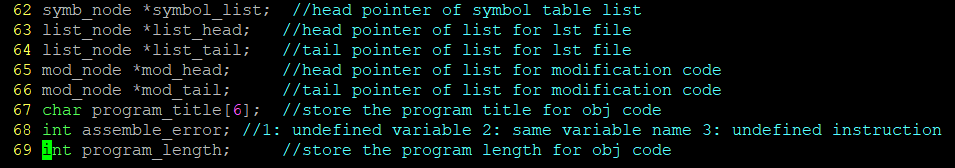
-base register가 사용 된 경우에 base 주소를 저장하는 변수다.

**5. 코드설명**

20171290.h : 프로그램 내에서 사용하는 전역변수들을 저장 해 놓은 header file

2차 프로젝트에서 필요한 전역변수를 추가로 선언하였다.





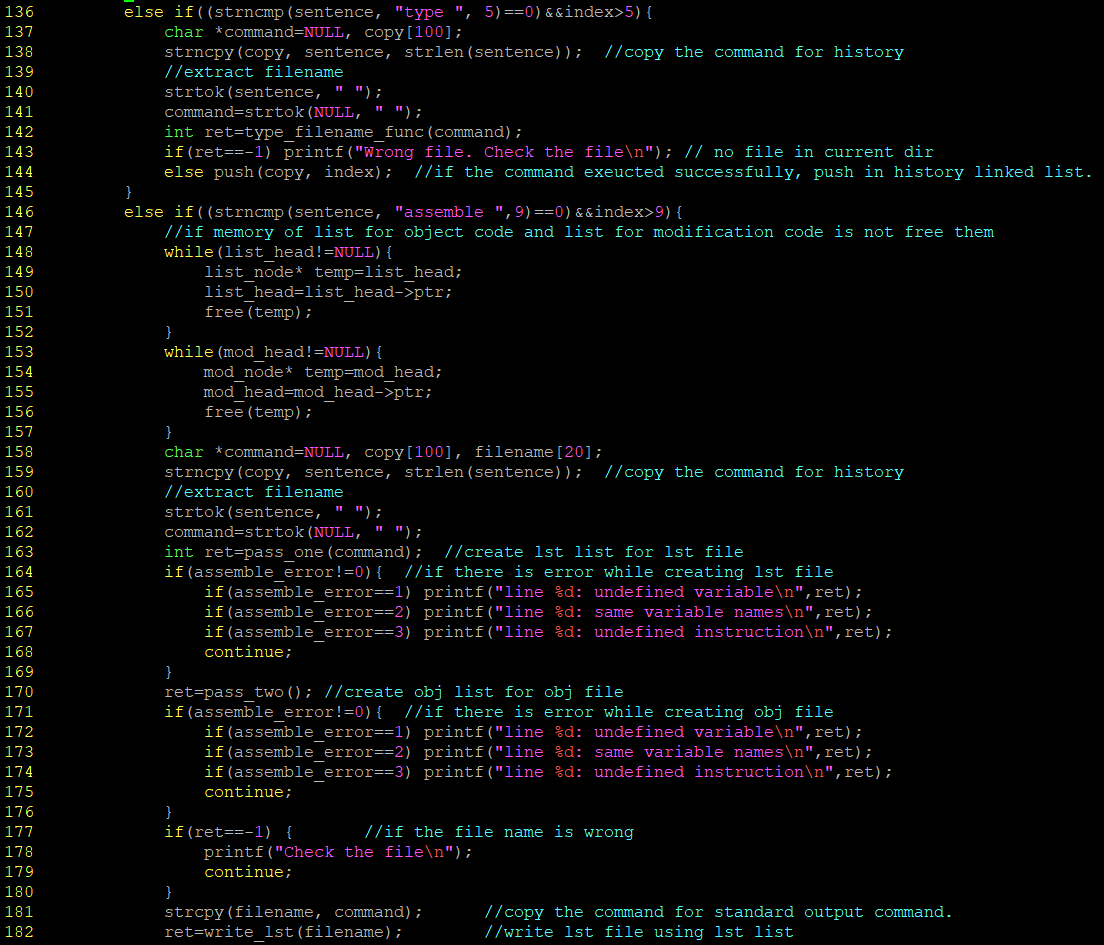
20171290.c: command로 들어 온 명령어를 읽어 알맞은 함수를 수행하도록 하는 코드

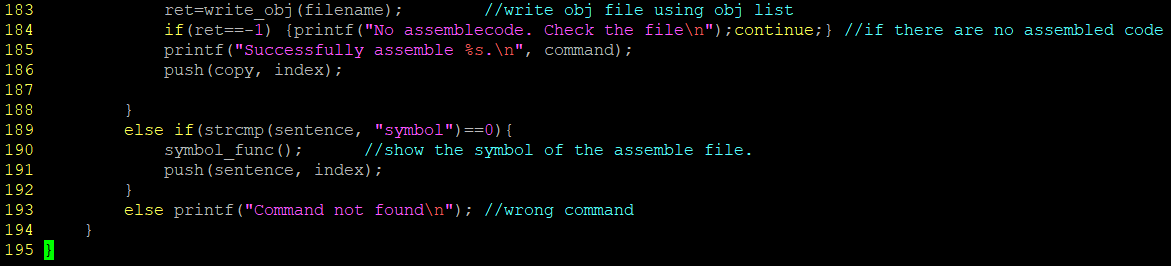
2차 프로젝트에서 필요한 코드를 추가하였다. main함수는 1주차 프로젝트에서

구현 된 명령어 외에 type, assemble, symbol 명령어를 추가적으로 인식하며,

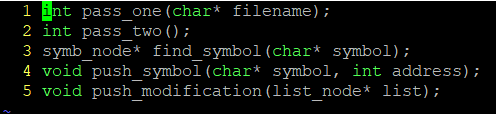
Assemble 명령어 입력 시에는 해제되지 않은 메모리를 해제하는 역할도 추가적으로

수행한다.





Assemble.h: assemble.c에서 선언 된 함수들의 prototype을 모아놓은 헤더파일



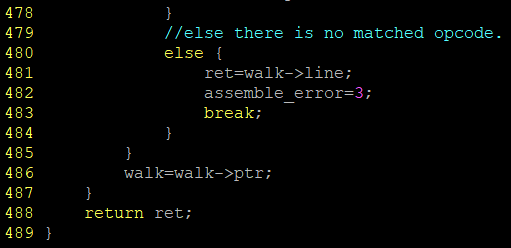
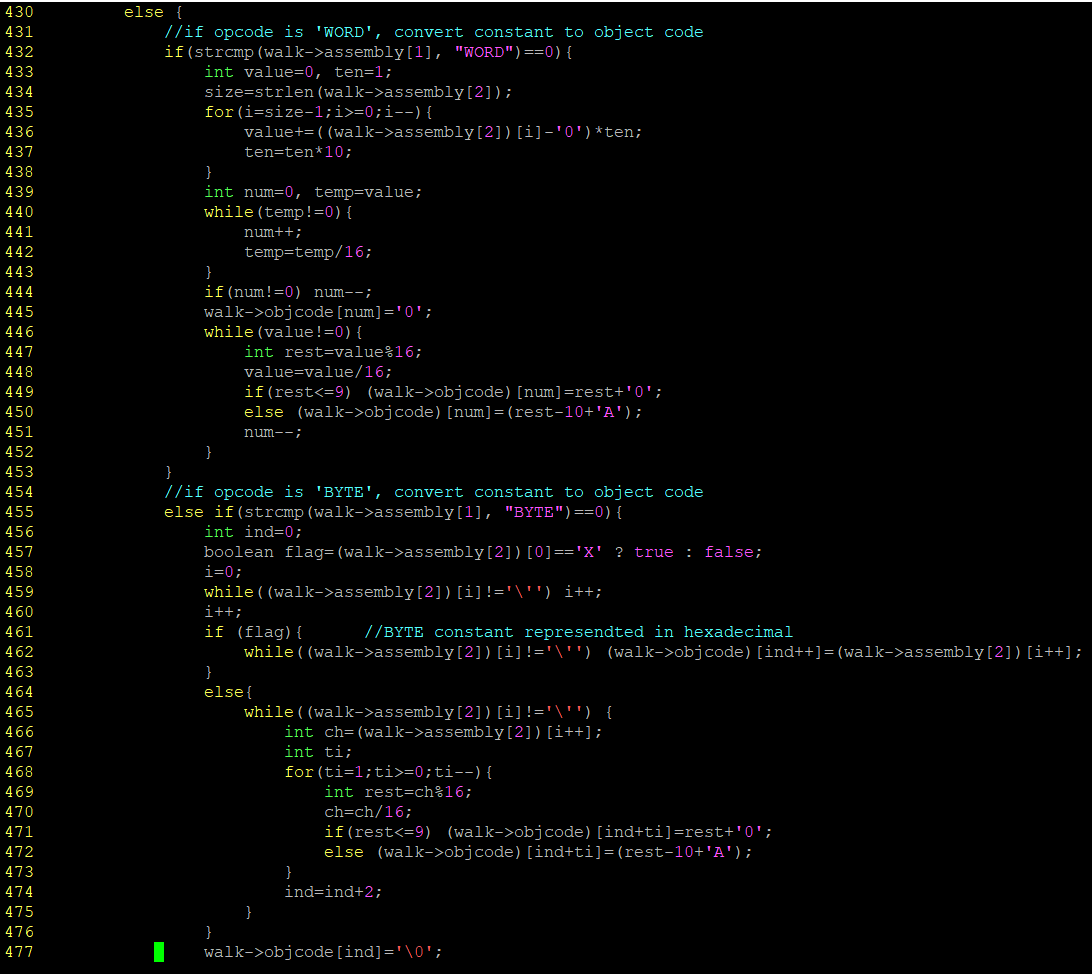
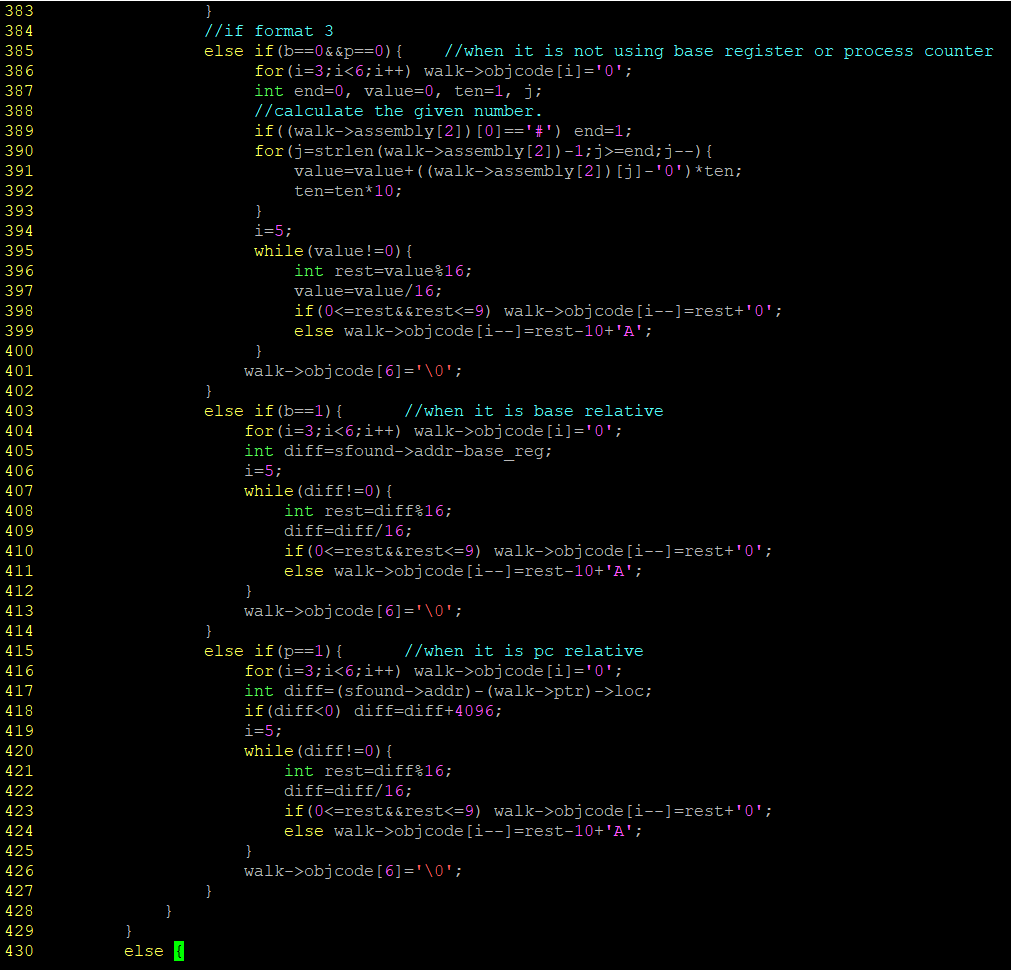
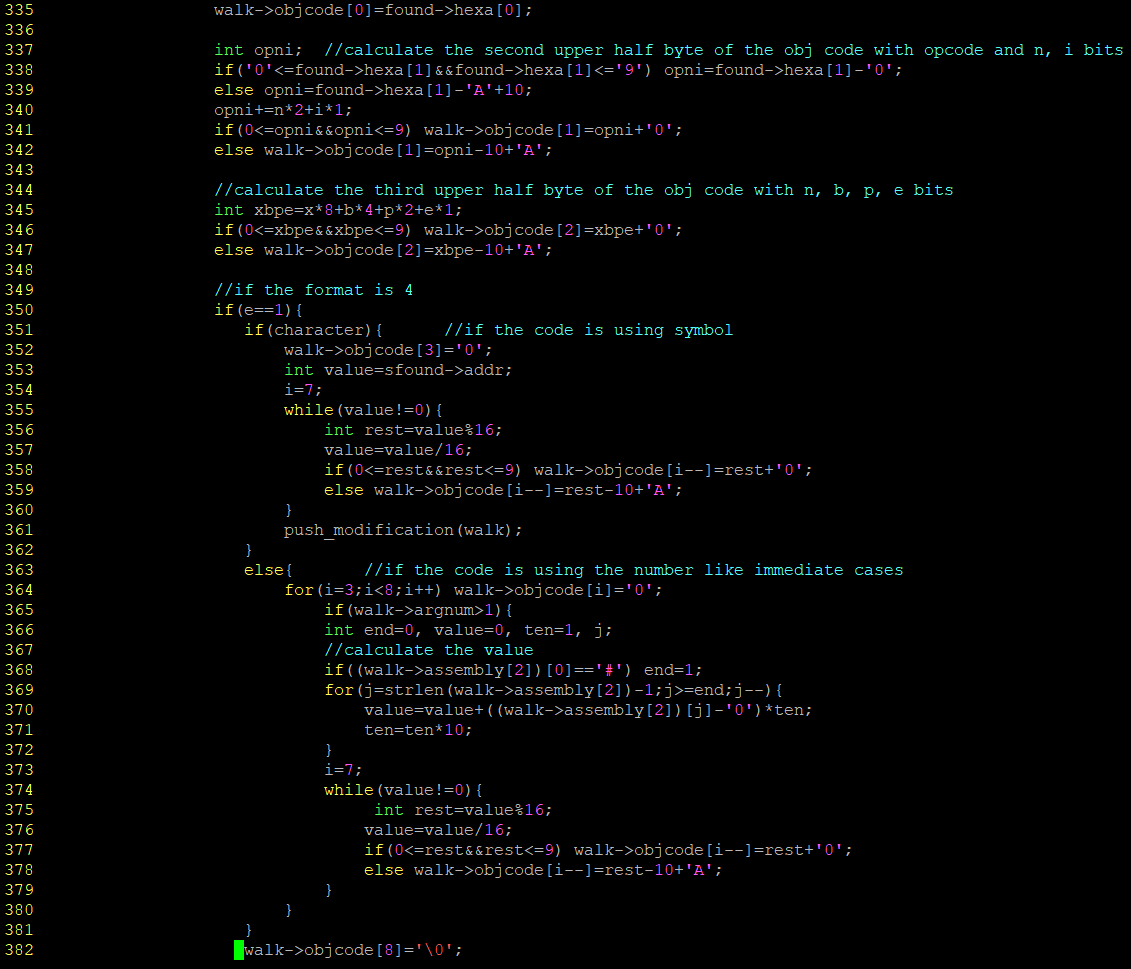
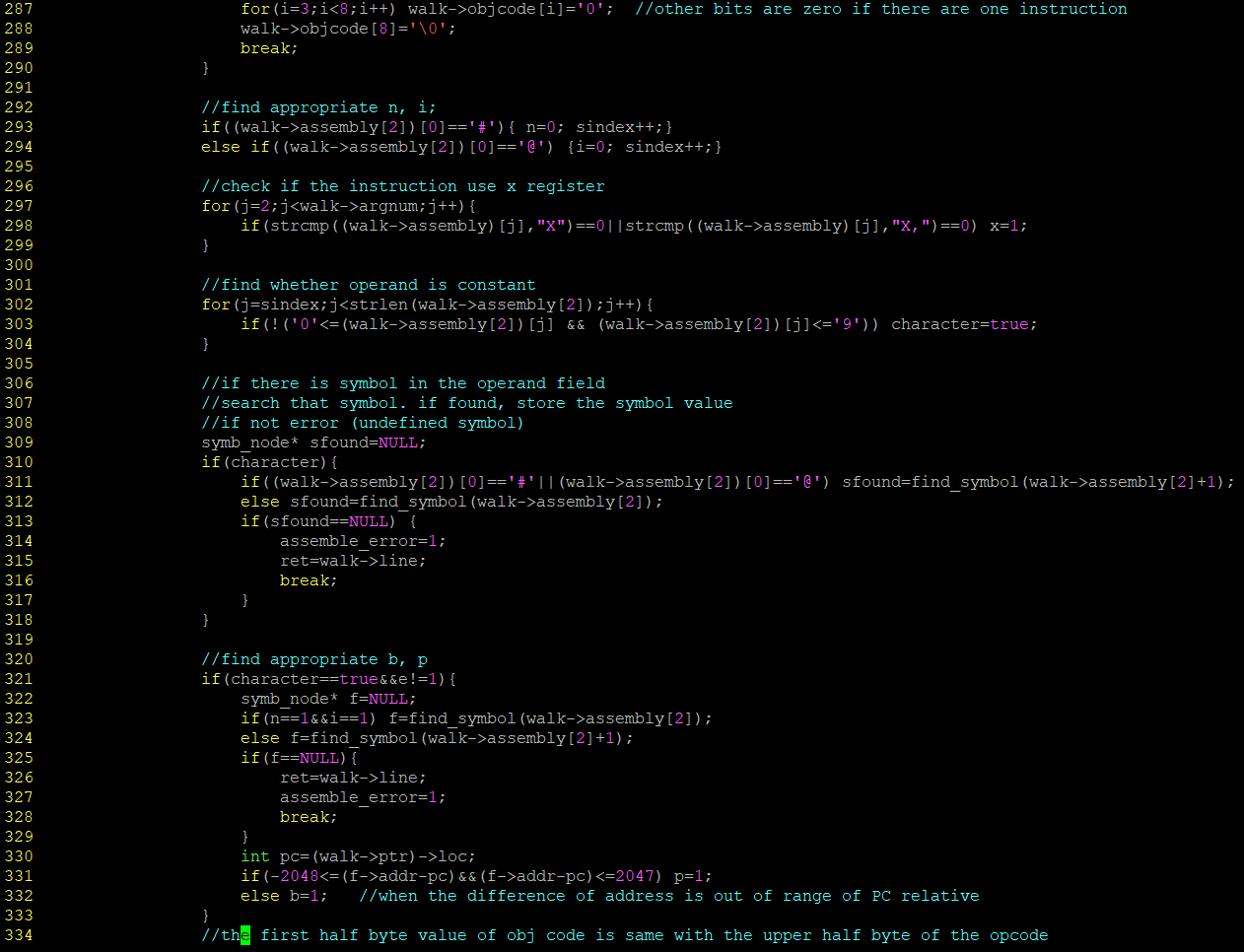
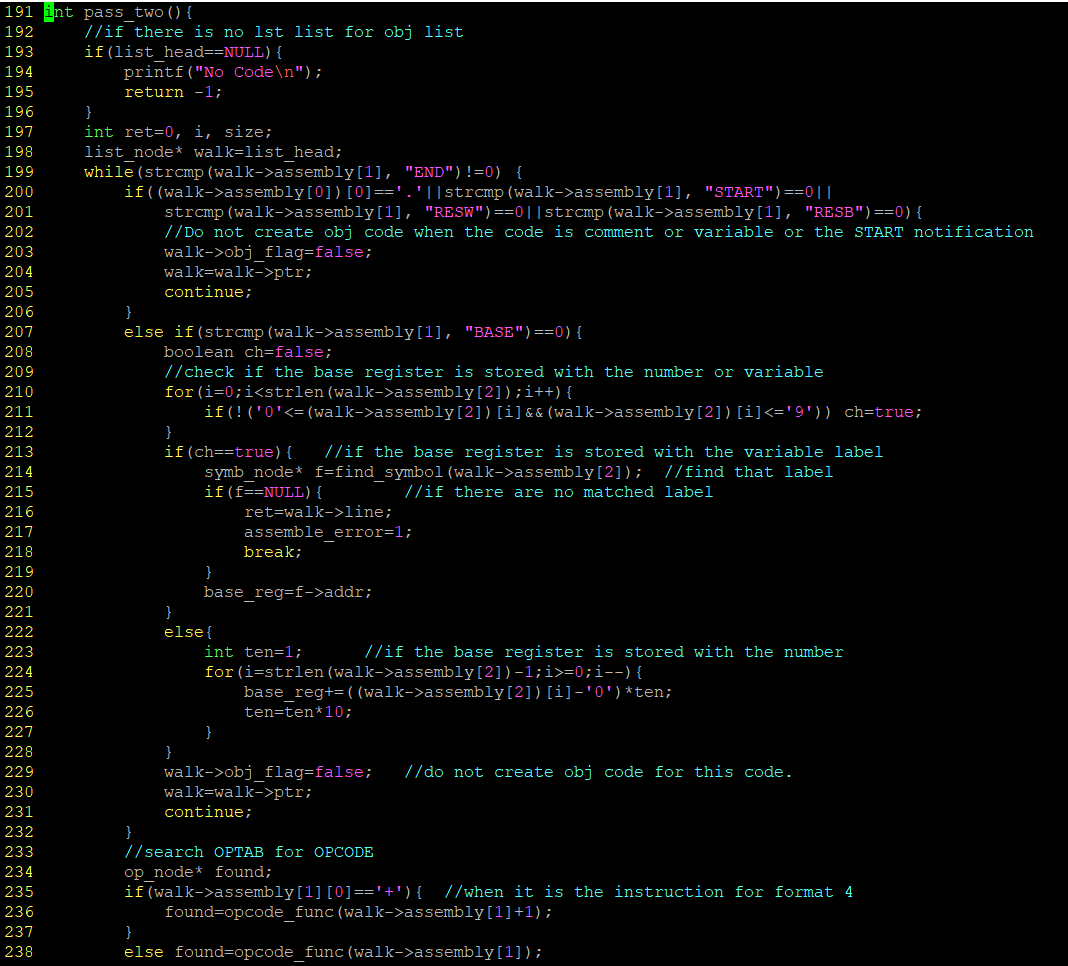
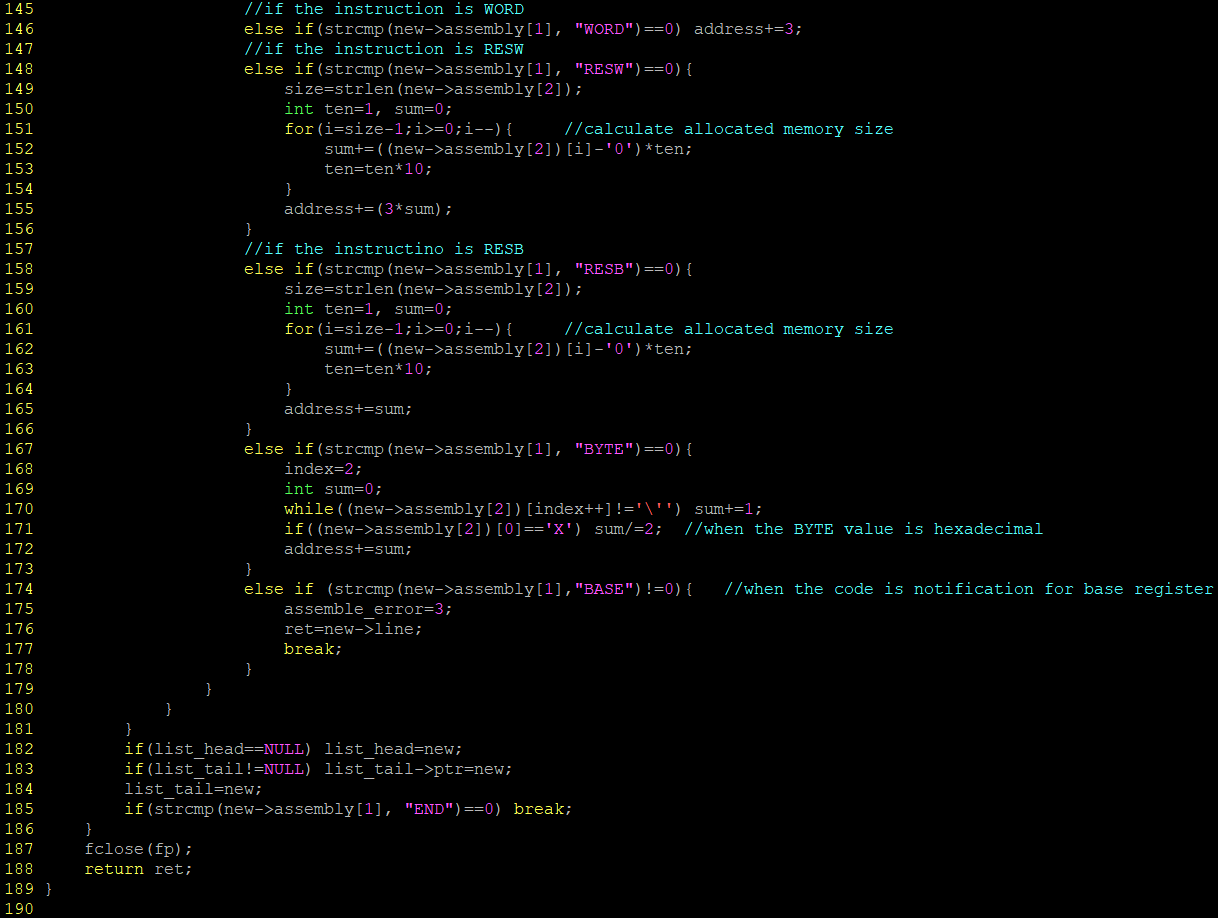
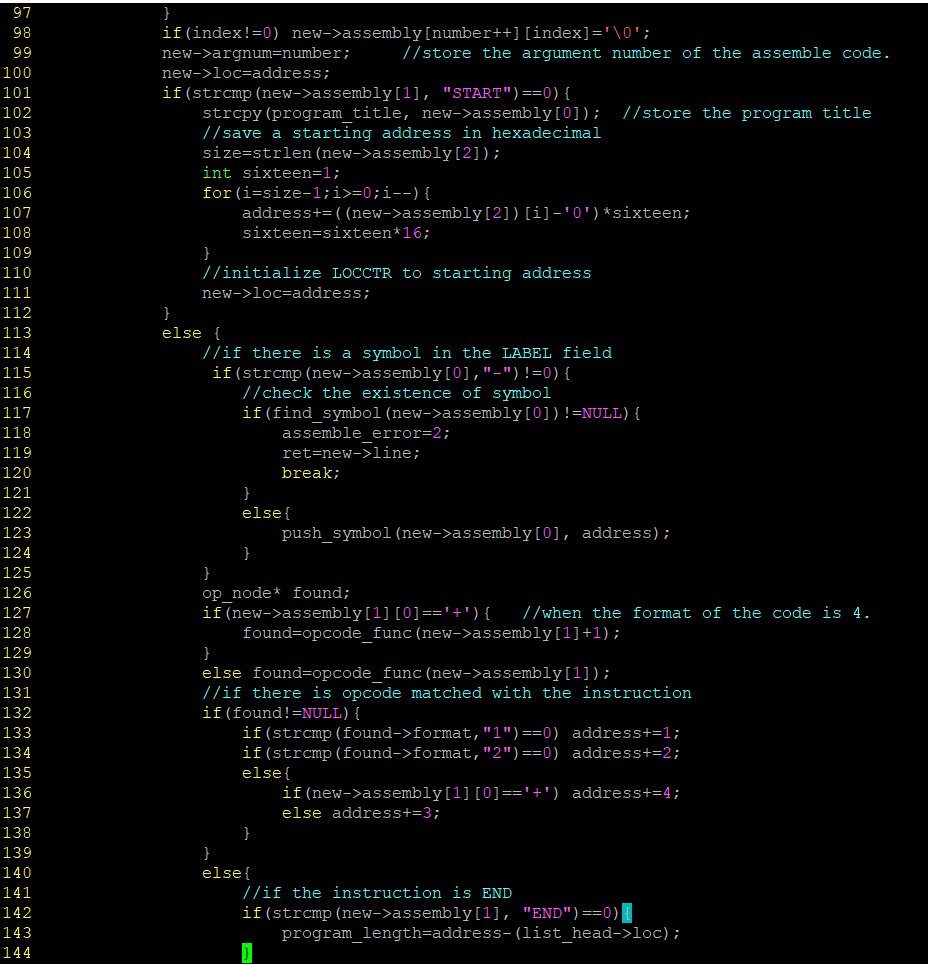
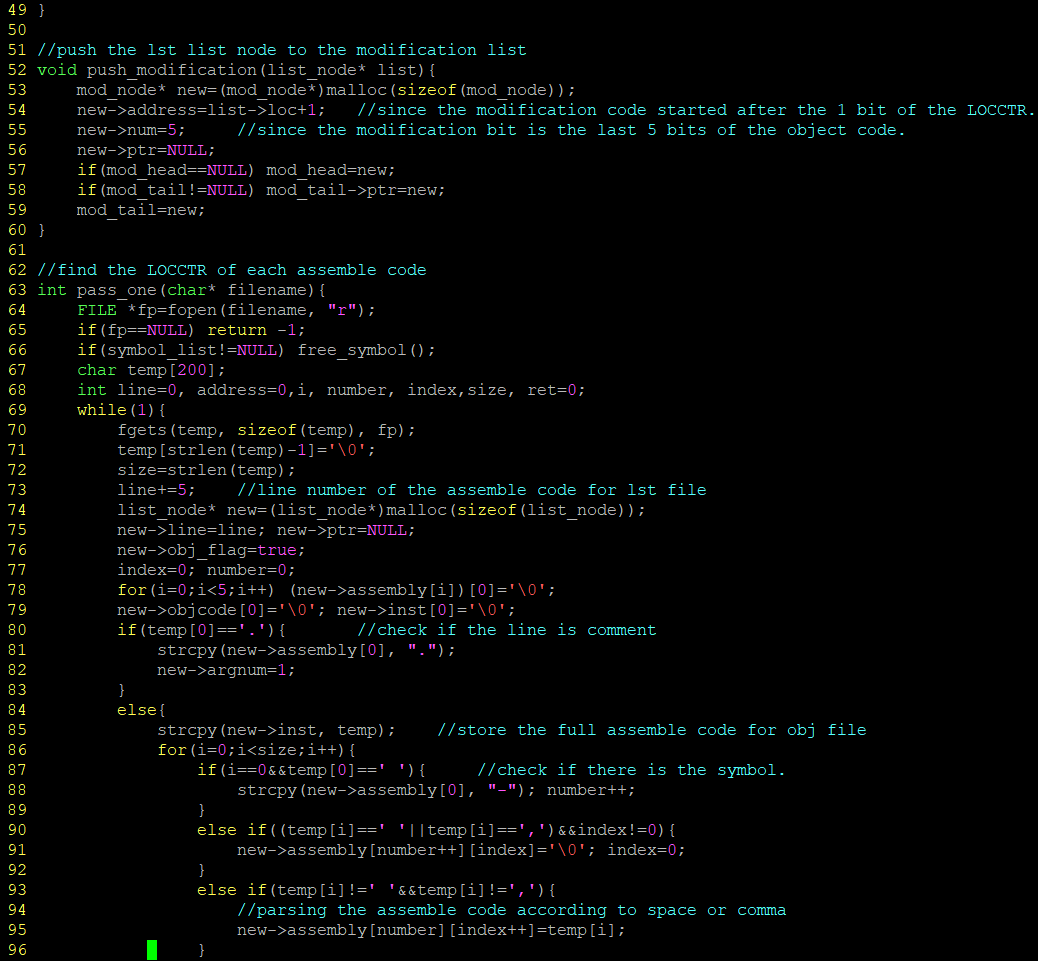
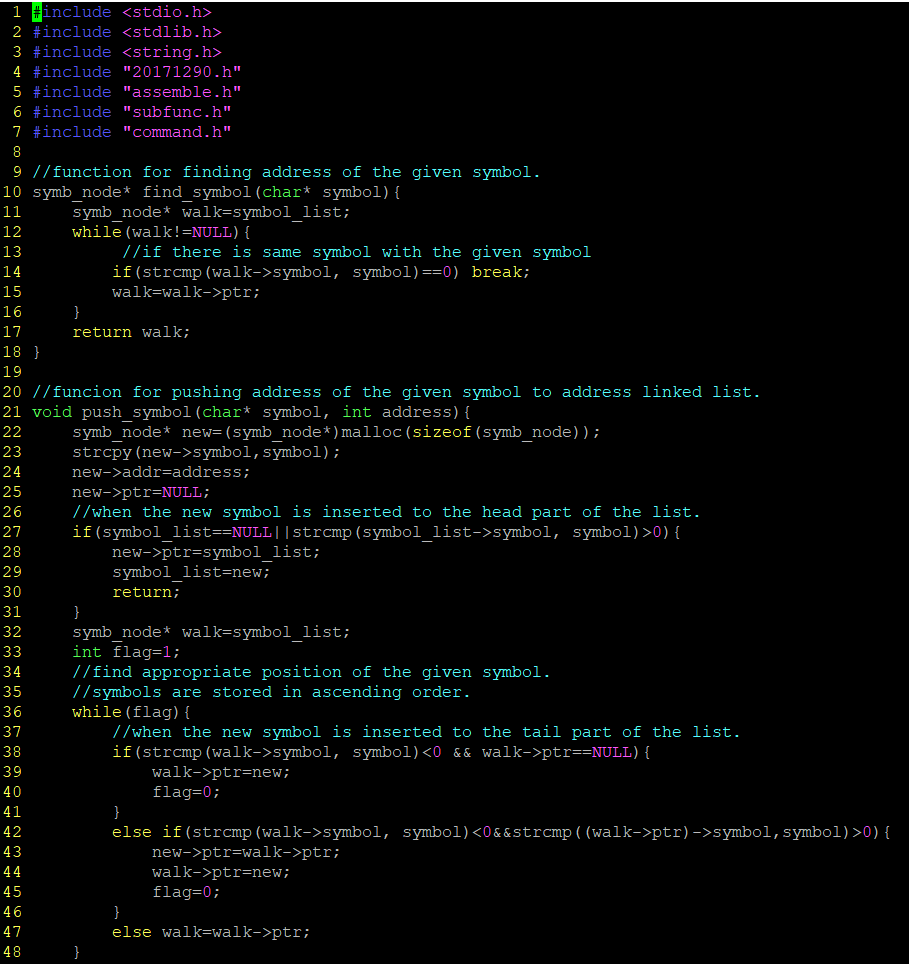
Assemble.c: assemble하는 과정에서 일어나는 일들을 모두 가지고 있는 소스코드

이 소스 코드에는 symbol을 table에 추가하는 과정, symbol을 찾는 과정,

Modification이 필요한 코드를 list에 추가하는 과정, LOC 계산하는 과정,

Object code 생성하는 과정이 구현되어 있다.

Assemble 하는 과정에 대해 알고리즘을 부가적으로 서술한다면, 우선 assemble file source를 한 줄씩 읽어온다. 그리고 해당 assemble code line의 정보를 저장 할 node를 새로 할당한다. 해당 assemble line의 line값을 저장한 후, 만약 해당 line이 주석이면, 주석임을 의미하는 정보를 추가하고 다음 줄을 읽는다. 만약 주석이 아니라면, space 또는 comma를 기준으로 instruction과 argument를 parsing한다. 그리고 object code 생성을 위해 argument의 수를 저장 해 놓는다. 만약 instruction이 START라면 해당 line의 symbol 값을 프로그램 이름의 값으로 저장한다. 만약 symbol이 있다면 이 symbol 이전에 동일한 이름의 symbol이 저장되어있는지 확인하고, 저장되어있다면 오류를 출력한다. 그리고 저장이 안되어있다면 symbol table에 해당 symbol을 추가한다. 그리고 나서는 instruction을 확인한다. 읽은 instruction이 정의되지 않은 instruction이면 오류를 출력하고 그렇지 않다면, instruction의 format에 맞추어 LOC의 값을 저장한다. 만약 instruction의 값이 END이면 program의 크기를 구한다. 만약 instruction의 값이 WORD이면 3byte 상수가 들어올 것 이므로, 이에 맞춰 LOC의 값을 추가 해 준다. 만약 RESW나 RESB라면, 할당 된 메모리의 수를 계산하여 LOC의 값을 추가 해 준다. 만약 instruction의 값이 BYTE라면, BYTE 상수에 할당 된 메모리를 계산한 후 LOC의 값을 추가 해 준다. 만약 instruction으로 base가 들어온다면 이는 instruction 미정의 오류로 처리하지 ㅏㅇㄶ는다. 그리고 이렇게 형성 된 node들을 linked list에 추가 해 준다. 이 과정이 pass\_one 함수가 수행하는 과정이다. 만약 pass\_one이 오류 없이 잘 형성되었다면, object code를 생성하는 pass\_two로 넘어간다. Pass\_one에서 만들어진 linked list를 순회하며 각 instruction에 대한 object code를 작성한다. 우선 instruction이나 변수인 경우는 object code를 생성하지 않는다. 그리고 instruction이 base인 경우에도, object code를 따로 생성하지 않고, base register에 들어가야 하는 주소의 값을 저장한다. 그리고 instruction의 format이 1인 경우는 instruction의 opcode의 값을, 2인 경우에는 opcode와 register number를 이용하여 object code를 생성한다. 그 외에 우선 opcode로만 이루어진 assembly code인 경우에, simple addressing을 적용하여 object code를 작성한다. 그리고 만약 format 3라면 simple addressing, immediate addressing, indirect addressing 여부를 파악한다. 그리고 만약 simple addressing이라면 base relative인지 pc relative인지 여부를 확인한다. 그리고 각 addressing 방법과 x register 사용 유무, format number를 확인하여 nixbpe bit를 확인한다. Format 3인 경우에는 추가적으로 각 relative에 따라 display값을 구하고, format 4는 direct addressing 기법을 사용하여 object code의 값을 구한다. 그리고 format 4의 경우relocation 과정이 필요하므로 modification list에 해당 instruction의 정보를 추가 해 준다. 만약 instruction이 WORD인 경우에는, 해당 상수의 값을 계산하여 그 값을 object code값으로 설정하고, BYTE인 경우에는 각 글자의 ASCII code의 값을 이용하여 object 값으로 설정한다. 이 과정이 pass\_two이다. Pass\_two가 성공적으로 수행이 되었다면, lst file과 obj file을 작성하며 assemble 과정을 마무리 한다.



Command.h: shell이 수행할 수 있는 명령어에 대한 수행 함수의 prototype을 모아놓은 헤더파일

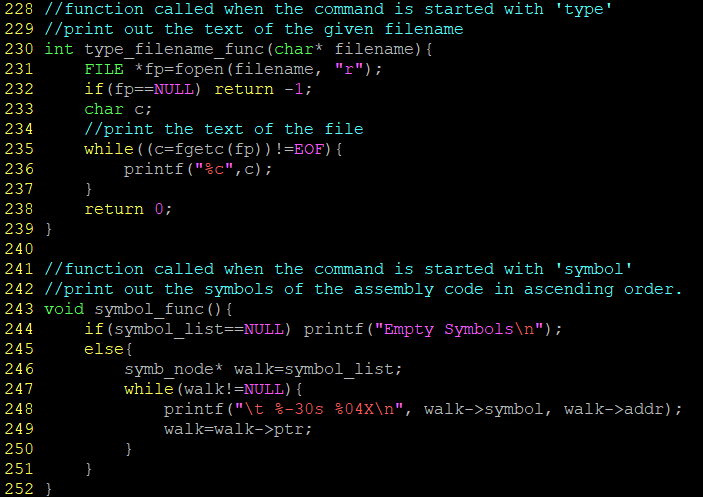
1차 프로젝트에서 작성한 command.h에 2차 프로젝트에 필요한 prototype을

추가했다.



Command.c: shell이 수행할 수 있는 명령어에 대한 수행 함수를 정의한 소스파일.

이 파일 역시 2차 프로젝트에 필요한 함수를 추가적으로 작성하였다.



Subfunc.h: 프로그램을 수행하는데 있어서 필요한 부가적인 함수의 prototype을 모아놓은

헤더파일. 1차 프로젝트에서 작성한 subfunc.h에 2차 프로젝트에 필요한 prototype을

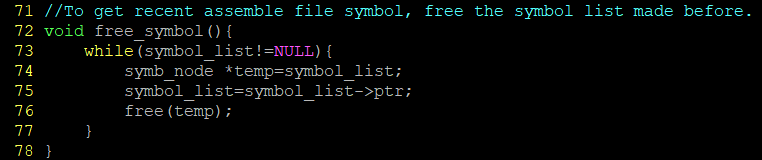
추가했다.



Subfunc.c: 프로그램을 수행하는데 있어서 필요한 부가적인 함수를 정의한 소스코드. 이 파일

역시 1차 프로젝트에서 작성한 subfunc.c에 2차 프로젝트에 필요한 함수를

추가적으로 정의하였다.



Write.h: listing file과 object file을 작성하기 위한 함수의 prototype을 모아놓은 헤더파일.



Write.c: 저장 된 list\_node list와 mod\_list node를 이용하여 listing file과 object file을 작성하는

함수들을 정의한 파일

(코드는 다음 장에 참조하였다.)

