과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 2>>

**서강대학교 컴퓨터학과**

**20161622**

**이예은**

목 차

1. 프로그램 개요 4

2. 프로그램 설명 4

2.1 프로그램 흐름도 4

2.2 알고리즘 설명 5

3. 모듈 정의 5

3.1 모듈 이름 : InitSymbolTable() 5

3.1.1 기능 5

3.1.2 사용 변수 5

3.2 모듈 이름: InitAssemNode() 5

3.2.1 기능 5

3.2.2 사용 변수 5

3.3 모듈이름: Assemble(char \*file\_name) 5

3.3.1 기능 5

3.3.2 사용변수 5

3.4 모듈이름: IsAssemFile(char \*file\_name) 6

3.4.1 기능 6

3.4.2 사용변수 6

3.5 모듈이름: AssemPass1(char \*file\_name) 6

3.5.1 기능 6

3.5.2 사용변수 6

3.6 모듈이름: AssemPass2(char\* file\_name) 6

3.6.1 기능 6

3.6.2 사용변수 6

3.7 모듈이름: MakeAssemNode(char tkstr[][MAX\_LINESIZE]) 7

3.7.1 기능 7

3.7.2 사용변수 7

3.8 모듈이름: AssemToken(char \*asm\_line, char tk\_str[][MAX\_LINESIZE]) 7

3.8.1 기능 7

3.8.2 사용변수 7

3.9 모듈이름: GetType\_and\_SaveInst(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE]) 7

3.9.1 기능 7

3.9.2 사용변수 7

3.10 모듈이름: GetLoc(assem\_node \*new\_node) 7

3.10.1 기능 7

3.10.2 사용변수 7

3.11 모듈이름: GetObj(assem\_node \*cur\_node) 7

3.11.1 기능 7

3.11.2 사용변수 8

3.12 모듈이름: FindReg(char \*str) 8

3.12.1 기능 8

3.12.2 사용변수 8

3.13 모듈이름: FindPseudoInstr(char\* key) 8

3.13.1 기능 8

3.13.2 사용변수 8

3.14 모듈이름: Type(char \*file\_name) 8

3.14.1 기능 8

3.14.2 사용변수 8

3.15 모듈이름: GetOperand(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE]) 8

3.15.1 기능 8

3.15.2 사용변수 8

3.16 모듈이름: GetPseudoOperand(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE]) 9

3.16.1 기능 9

3.16.2 사용변수 9

3.17 모듈이름: Symbol() 9

3.17.1 기능 9

3.17.2 사용변수 9

3.18 모듈이름: SearchSymbol(char \*key) 9

3.18.1 기능 9

3.18.2 사용변수 9

3.19 모듈이름: MakeSymbolTable(assem\_node \*new\_node) 9

3.19.1 기능 9

3.19.2 사용변수 9

3.20 모듈이름: StrToDec(char\* str) 9

3.20.1 기능 9

3.20.2 사용변수 9

4. 전역 변수 정의 10

4.1 enum ASSEM\_TYPE 10

4.2 enum PSEUDO\_INSTR\_TYPE 10

4.3 enum ADDRESS\_MODE 10

4.4 int pc\_addr,base\_addr 10

4.5 assem\_node \*assem\_head, \*assem\_rear 10

4.6 symbol\_table\_node symbol\_table[26] 10

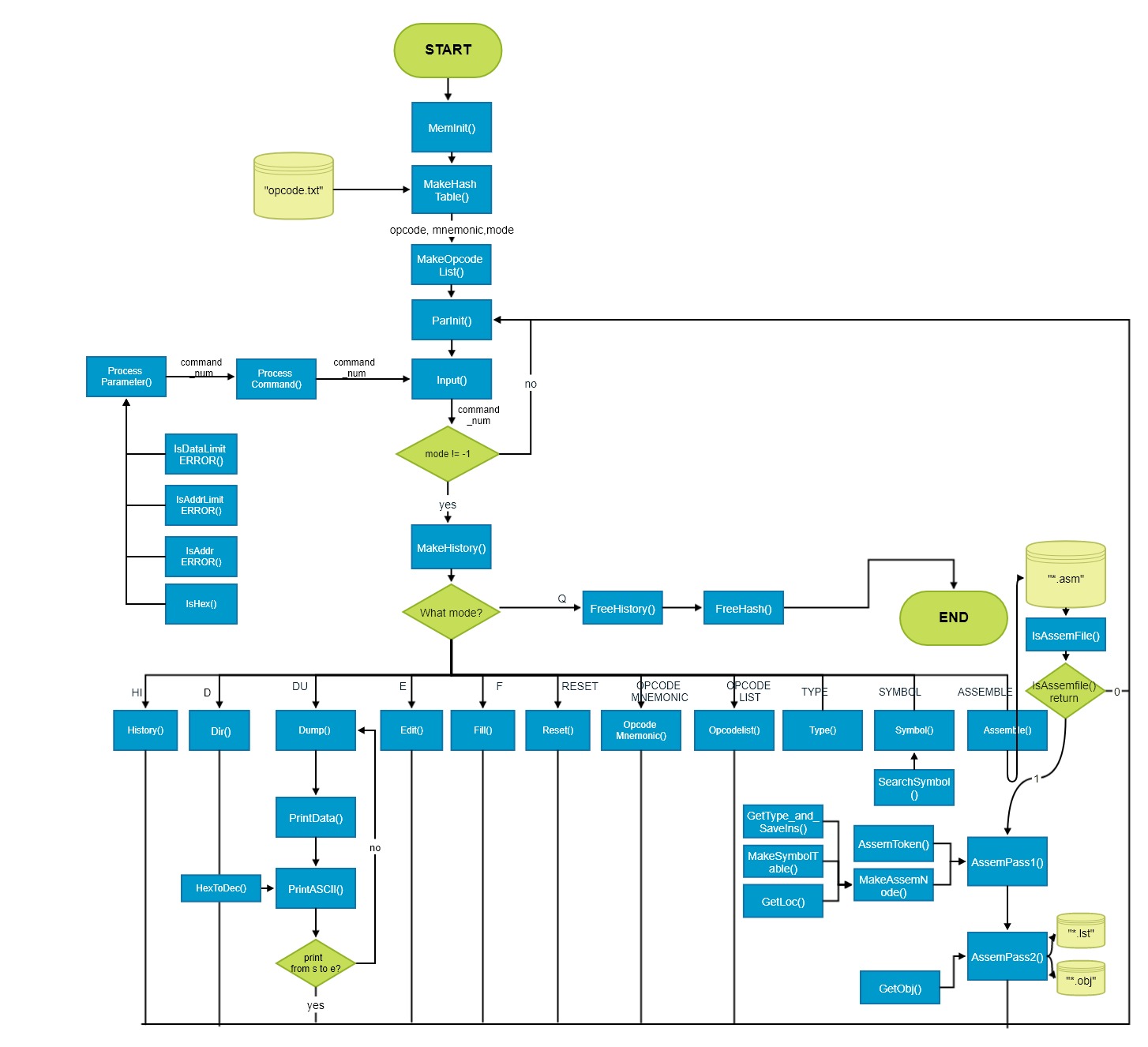
5. 코드 10

# 프로그램 개요

프로젝트 1에서 구현한 SIC/XE Shell에 assemble, symbol, type의 기능을 추가하는 프로젝트이다. Assemble은 SIC/XE machine의 ‘.asm’파일을 입력 받아 object파일과 list파일을 생성해야 한다. 그리고 Symbol 명령에서는 해당 ‘.asm’의 symbol을 오름차순으로 정렬해 출력하는 것을 구현한다. 또한 type에서는 입력 받은 파일의 내용을 출력한다.

# 프로그램 설명

## 프로그램 흐름도



## 알고리즘 설명

**<ASSEMBLE>**

올바른 파일 이름 형식이 들어오고 해당 파일이 존재할 경우,

**AssemPass1**에서 파일을 열어 한 줄씩 읽으면서, 어셈블리 코드를 strtoken을 이용해 분리시키고 분리된 스트링이 각각 symbol, instruction, operand으로 구분해 assem노드에 저장한다. 저장한 정보를 이용해 location을 만들어 assem노드 저장한다. 또한 symbol이 나올 경우 symbol table에 저장한다.

**AssemPass2** 에서는 리스트 파일 오브젝트파일 을 생성하고 GetObj()함수를 통해 object 코드를 만들어 assem노드에 추가한다. assem노드에 추가된 정보를 이용해 list파일을 완성하고, 다시 노드를 처음부터 읽어들여 obj파일을 생성한다.

**<Symbol>**

symbol\_table\_struct[26]을 만들어 symbol의 첫 알파벳 따른 테이블에 내림차순으로 노드를 추가하고

출력시에는 symbol\_table을 거꾸로(‘z’부터) 읽어들여 내림차순을 출력한다

**<TYPE>**

Fopen 으로 한 줄씩 파일을 읽어 해당 내용을 출력한다.

# 모듈 정의

## 모듈 이름 : InitSymbolTable()

### 기능

Symbol table에 symbol\_node들이 연결되어 있으면 연결된 노드를 모두 free시키고 symbol\_table의 모든 next포인터를 NULL 로 초기화하고 대표 알파벳(symbol\_table[i].alpha)을 설정한다..

### 사용 변수

**symbol\_node \*tmp\_p, \*del\_p:** 심볼 테이블에 연결된symbol\_node 를 free시키기 위해 위해 사용될 포인터

## 모듈 이름: InitAssemNode()

### 기능

연결되어 있는 모든 assem\_node를 free시키고 head, rear포인터를 NULL로 초기화 한다.

### 사용 변수

**assem\_node \*tmp\_p, \*del\_p** : free할 assem\_node 를 찾기 위해 사용될 변수

## 모듈이름: Assemble(char \*file\_name)

### 기능

‘assemble’ 명령을 수행하기 위한 함수이다. parameter로 넘어온 file\_name을 통해 해당 파일이 존재하며, asm파일이면 AssemPass1()함수와 AssemPass2()함수를 통해

### 사용변수

**FILE \*fp :** 파일 포인터

**char \*file\_name :** 입력받은 파일

## 모듈이름: IsAssemFile(char \*file\_name)

### 기능

char \*file\_name 을 확인해서 해당 파일이 “.asm”파일인지 확인한다. 만약 그렇지 않다면 0을 리턴하고’.asm’파일이면 1을 리턴한다

### 사용변수

**int check\_asm** : assemble파일인지 체크를 하기 위한 변수로 check\_asm = 4(‘.’ ‘a’ ‘s’ ‘m’)이면 1을 리턴한다.

## 모듈이름: AssemPass1(char \*file\_name)

### 기능

어셈블리 파일을 한줄씩 읽고 AssemToke()함수를통해 읽은 어셈블리 파일 코드 line을 쪼개어tk\_str[][]스트링에 옮겨 담는다.그리고 해당 스트링을 통해 MakeAssemNode()함수로 assem\_node를 채우고 END가 나오면 파일을 닫고 함수를 종료한다.

스트링을 옮겨 담다가 에러가 발생한 경우 ERROR(-1)을 리턴한다.

### 사용변수

**FILE \*fp** : 파일을 열어서 읽기 위한 파일 포인터

**char asm\_line[MAX\_LINESIZE]** : 어셈블리파일에서 읽어 들인 한 라인

**char tk\_str[MAX\_ASM\_TOKEN][MAX\_LINESIZE]** : asm\_line을 토큰화 시킨 스트링을 저장하는 스트링 배열

## 모듈이름: AssemPass2(char\* file\_name)

### 기능

.lst파일과 .obj파일을 생성하기 위한 함수이다. “START”가 존재 할 경우 첫 줄을 처리하고 list파일을 열어서 list파일을 채워나간다. “\*.asm”파일을 열어서 한줄씩 읽으며 GetObject함수에서 assem\_node에 담긴 data를 가지고 Object파일을 생성하고 형식에 맞춰 list파일을 만든다.

그러고 나서, list파일을 만들면서 assem\_node 저장해놓은 오브젝트 코드와 Tflag를 사용해 ‘.obj’ 파일을 만든다.

### 사용변수

**FILE \*fp** : 어셈블리 파일을 읽기 위한 파일 포인터

**FILE \*list** : 리스트 파일을 생성하기 위한 파일 포인터

**FILE \*object** : object파일을 생성하기 위한 파일 포인터

**char list\_name[MAX\_FILENAM**E] : list파일 이름

**char object\_name[MAX\_FILENAME] :** object파일 이름

**char asm\_line[MAX\_LINESIZE]** : ‘asm’파일에서 코드를 한 줄씩 읽어 들여 저장하기 위한 문자열

**int t\_total** : object파일을 생성하기 위해 byte수를 세기 위한 변수

**int t\_end\_flag** : object파일을 생성할 때 줄의 끝을 알리는 flag

**assem\_node\* tcnt** : 연결된 노드들을 읽어 들이기 위한 포인터

**assem\_node\* cur** : 현재 읽고 있는 assem\_node를 위한 포인터

## 모듈이름: MakeAssemNode(char tkstr[][MAX\_LINESIZE])

### 기능

assem\_node를 만들고, tkstr을 이용해서 어셈블리 파일로부터 들어온 문자열을 기능에 맞게(instruction, symbol, operand) 쪼개어 저장한다. 그리고 그 정보를 이용해 instruction 종류와 operand를 처리하고, GetLoc()함수를 통해 location을 구해 assem\_node에 저장한다.

### 사용변수

**int type** : instruction type을 저장한다

**assem\_node \*new\_node** : 새로 생성할 assem\_node

## 모듈이름: AssemToken(char \*asm\_line, char tk\_str[][MAX\_LINESIZE])

### 기능

사용자에게 계속 진행할 것인지를 묻는다. 0이면 yes(계속 진행), 1이면 no(끝)을 리턴 한다.

만약 0, 1외의 수가 들어오면 사용자에게 적절한 입력이 아님을 알린 후 0,1이 입력될 때까지 입력을 받는다.

### 사용변수

int ask - 사용자의 입력을 받는 저장공간

## 모듈이름: GetType\_and\_SaveInst(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE])

### 기능

instruction type을 구별하고 instruction과 symbol을 저장하기 위한 함수이다. symbol과 instruction 을 분리하고comment와 format4를 따로 처리해 assem\_node에 저장하고, instruction의 opcode를 통해 format도 저장한다.

### 사용변수

**int opcode[3]** : tk\_str[0~3]이 instruction이면 opcode값을 아니면 -1값을 가지고 있는 문자열 배열

**int pseudo[3]** :tk\_str[0~3]이 pseudo code이면 pseudo code값을 아니면 -1을 가지고 있는 문자열 배열

## 모듈이름: GetLoc(assem\_node \*new\_node)

### 기능

해당 노드(new\_node)에 담겨있는 instruction을 이용해 location을 구해 저장하기 위한 변수이다. START가 있으면 pc\_addr을 설정하고, 이전 노드의 location과 이전 노드의 form과 operand를 사용해 location을 구한다.

### 사용변수

**int bef\_loc** : 이전 location을 담고있는 변수

## 모듈이름: GetObj(assem\_node \*cur\_node)

### 기능

object code를 구하기 위한 함수이다. cur\_node에 저장되어있는 instruction type과 format그리고 addr\_mode, base\_addr, pc\_addr(전역변수)를 사용해 object code를 구해 cur\_node->obj에 저장한다.

### 사용변수

**int sym\_addr** : operand에 있는 symbol의 address를 가지고 있는 변수

## 모듈이름: FindReg(char \*str)

### 기능

str과 일치하는register을 찾아 return 하는 함수이다. 만약 찾지 못했을 경우 -1을 리턴한다

### 사용변수

char reg[9][4] = {"A","X","L","B","S","T","F","PC","SW"} : 레지스터의 종류를 담고있는 변수

## 모듈이름: FindPseudoInstr(char\* key)

### 기능

key값과 일치하는 Pseudo Instruction이 있을 경우 해당 pseudo instruction 값(start = 0 …)을 반환한다 .

### 사용변수

## 모듈이름: Type(char \*file\_name)

### 기능

“Type” command를 구현하기 위한 함수이다. Type은 “file\_name” 함수를 읽어서, 해당 파일의 내용을 출력하는 함수이다. 만약 해당 파일이 존재하지 않을 경우 -1을 리턴한다

### 사용변수

**FILE \*fp** : 파일을 읽어 들이기 위한 파일 포인터

**char in** : 파일에서 읽어 들여 출력할 char

## 모듈이름: GetOperand(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE])

### 기능

new\_node에 instruction type이 instruction일 경우 저장되어 있는 format과 instruction에 따라 operand가 형식에 맞춰 입력이 들어왔는지 확인하고 처리해서 new\_node->operand[][]에 저장한다. 만약 에러가 있을 경우 -1을 리턴하고, 그렇지 않을 경우 0을 리턴한다

### 사용변수

**int is\_sym** : symbol이 존재하는 지의 여부를 0(존재x)과 1(존재 o)로 저장하는 함수

**char oper2\_except[4][3][10] = {** :format 2 instruction에서 예외적으로 처리할 instruction

**{"CLEAR","TIXR"},**

**{"CVC","CVC"},**

**{"SHIFTL", "SHIFTR"}**

**}**

## 모듈이름: GetPseudoOperand(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE])

### 기능

new\_node에 instruction type이pseudo instruction일 경우 저장되어 있는 instruction에 따라 operand가 형식에 맞춰 입력이 들어왔는지 확인하고 operand type(‘C’,’X’ .. )에 따라 operand[][]에 저장한다. 만약 에러가 있을 경우 -1을 리턴하고, 그렇지 않을 경우 0 을 리턴한다.

### 사용변수

**int is\_sym** : symbol이 존재하는지 여부를 0과 1로 저장해 놓기 위한 변수

## 모듈이름: Symbol()

### 기능

symbol\_table[0~25]에 연결되어 있는 노드들을 모두 출력하는 함수이다. 내림차순으로 정렬하기 위해 25(‘Z’)번째 테이블에 연결된 노드들부터 출력한다.

### 사용변수

**symbol\_node\* cur** : 현재 출력할 symbol\_node를 가리키는 함수이다.

## 모듈이름: SearchSymbol(char \*key)

### 기능

symbol테이블에 저장되어있는 symbol들 중에 key값과 일치하는 symbol이 있는지 확인하고 있으면 해당 symbol의 address주소를 출력한다.

### 사용변수

**symbol\_node \*cur** : 현재 비교할 symbol 노드를 가리키기 위한 변수이다.

## 모듈이름: MakeSymbolTable(assem\_node \*new\_node)

### 기능

assem노드를 통해 symbol table을 만들고 symbol의 맨 앞 문자의 알파벳에 해당하는 테이블에 symbol노드를 연결한다. 이때, 내림차순으로 정렬하기위해 다음 심볼보다 현재 심볼의 알파벳이 더 큰지 비교하면서 연결할 위치를 찾는다.

### 사용변수

**symbol\_node \*cur,\*bef** :이전노드와 현재 노드를 가리키는 포인터이다

**symbol\_node \*new\_sym** : 새로 만들 symbol node

**int start\_flag** : 맨 앞에 들어가게 되는지 아닌지를 확인하는 flag

## 모듈이름: StrToDec(char\* str)

### 기능

str문자열을 decmal값으로 바꿔 return해주는 함수이다. 만약 형식에 맞지 않으면 -1을 리턴한다

### 사용변수

**int dec** : str을 decmal로 바꾼 결과값을 저장하는 변수

# 전역 변수 정의

## enum ASSEM\_TYPE

ERROR = -1, COMMENT, PSEUDO\_INT, INST = 2

로 각각 assembly code line에 해당하는 type값

## enum PSEUDO\_INSTR\_TYPE

START = 0, END, BASE, BYTE, WORD, RESB, RESW

paseudo instruction의 종류를 구별하기 위한 value

## enum ADDRESS\_MODE

SIC = 0, IMMED = 1, INDIR, SIMPLE

Address mode에 따른 n,i bit를 10진수로 계산 값을 가지고있다.

.

## int pc\_addr,base\_addr

pc, base address를 저장하는 변수

## assem\_node \*assem\_head, \*assem\_rear

assem\_node의 처음과 끝을 가리키는 포인터이다.

## symbol\_table\_node symbol\_table[26]

symbol\_node가 연결될 symbol\_table이다.

# 코드

**<20161622.c>**

|  |
| --- |
| /\* |
|  | +--------------------------------------------------------------+ |
|  | | System Programming Project 2 | |
|  | | ( SIC/XE Assembler ) | |
|  | | | |
|  | | File : 20161622.c | |
|  | | Created : Mar 18 ~ Apr 9, 2018 | |
|  | | Author : Ye-eun Lee | |
|  | +--------------------------------------------------------------+ |
|  | \*/ |
|  | #include "main.h" |
|  | #include "20161622.h" |
|  | #include "assembler.h" |
|  | #include <stdio.h> |
|  | #include <string.h> |
|  | #include <dirent.h> |
|  | #include <sys/stat.h> |
|  | #include <stdlib.h> |
|  |  |
|  |  |
|  | //enum about valid command |
|  | enum COMMAND\_TYPE { |
|  | H = 0, |
|  | D, |
|  | Q, |
|  | HI, |
|  | RESET, |
|  | OPCODELIST, |
|  | DU, |
|  | E, |
|  | F, |
|  | OPCODEMNEMONIC, |
|  | ASSEMBLE, |
|  | TYPE, |
|  | SYMBOL |
|  | }; |
|  |  |
|  | char mem[ MAX\_MEMORY ][3]; //memory to save data |
|  | char par[MAX\_PARAMETER][COMMANDSIZE]; //parameter from input |
|  | char command[ COMMANDSIZE ]; //command to save in history node |
|  | int last\_addr=-1; //save the last address in dump |
|  |  |
|  |  |
|  | /\*command list\*/ |
|  | char command\_list[13][2][15]={ |
|  | {"h","help"}, |
|  | {"d","dir"}, |
|  | {"q", "quit"}, |
|  | {"hi","history"}, |
|  | {"reset","reset"}, |
|  | {"opcodelist","opcodelist"}, |
|  | {"du","dump"}, |
|  | {"e","edit"}, |
|  | {"f", "fill"}, |
|  | {"opcode","opcode"}, |
|  | {"assemble", "assemble"}, |
|  | {"type", "type"}, |
|  | {"symbol","symbol"} |
|  | }; |
|  | /\*\*node to save History\*\*/ |
|  | his\_node\* his\_head = NULL; |
|  | his\_node\* his\_rear = NULL; |
|  |  |
|  |  |
|  |  |
|  | /\*Node to save OpcodeList using hash table\*/ |
|  | hash\_table hash[20]; |
|  |  |
|  |  |
|  | int main(){ |
|  | int mode; //execute mode |
|  |  |
|  | MemInit(); //Initialize memory array |
|  | MakeHashTable(); //Make hash table |
|  |  |
|  | while(1){ |
|  | ParInit(); //Initialize parameter(par) |
|  |  |
|  | //get input |
|  | printf("sicsim> "); |
|  | mode = Input(); |
|  | // execute the command according to mode |
|  | if(mode!=-1 && mode!=ASSEMBLE) AddHistory(); // If input is valid, add command to history |
|  |  |
|  | switch(mode){ |
|  | case -1: // If input is invalid, get input again |
|  | continue; |
|  | break; |
|  | case H: // execute Help() |
|  | Help(); |
|  | break; |
|  | case D: // execute Dir() |
|  | Dir(); |
|  | break; |
|  | case Q: // memory free and exit the program |
|  | FreeHistory(); |
|  | FreeHash(); |
|  | InitAssemNode(); |
|  | InitSymbolTable(); |
|  | return 0; |
|  | break; |
|  | case HI: // execute History() |
|  | History(); |
|  | break; |
|  | case DU: // execute Dump() |
|  | Dump(); |
|  | break; |
|  | case E: //execute Edit() |
|  | Edit(); |
|  | break; |
|  | case F: //execute Fill() |
|  | Fill(); |
|  | break; |
|  | case RESET: //execute Reset() |
|  | Reset(); |
|  | break; |
|  | case OPCODELIST: //execute Opcodelist() |
|  | Opcodelist(); |
|  | break; |
|  | case OPCODEMNEMONIC: //execute OpcodeMnemonic |
|  | OpcodeMnemonic(); |
|  | break; |
|  | case ASSEMBLE: //excute Assemble |
|  | if(Assemble(par[0]) != -1) |
|  | AddHistory(); |
|  | break; |
|  | case TYPE: //excute Type |
|  | Type(par[0]); |
|  | break; |
|  | case SYMBOL: //excute Symbol |
|  | Symbol(); |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Initialization \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | void ParInit(){ |
|  | //Initialize parameter to '\0' |
|  |  |
|  | int i; |
|  |  |
|  | for( i=0;i<MAX\_PARAMETER ; i++){ |
|  | par[i][0] = '\0'; // make string(par[i]) '\0' |
|  | } |
|  |  |
|  | } |
|  |  |
|  | void MemInit(){ |
|  | //Initialize memory to "00" |
|  |  |
|  | int i=0; |
|  | char tmp\_str[]="00"; //string to copy |
|  |  |
|  | for(i=0; i<MAX\_MEMORY ; i++) |
|  | strcpy(mem[i],tmp\_str); //make mem[i] "00" |
|  | } |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\* Process and check Input \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | int Input(){ |
|  | //Get input to process |
|  | //And if the input match the format, return command value to exeicute |
|  | //If not, return -1 |
|  |  |
|  | char ch\_in; //input char |
|  | int i=0,j=0; //i:index for cmd, j:index for command |
|  | char cmd[COMMANDSIZE]; //cmd is string to process command efficiently |
|  | //command is global variable to be saved in history |
|  |  |
|  | //get input |
|  | while(1){ |
|  | scanf("%c",&ch\_in); //get input |
|  | if( ch\_in == '\n') break; |
|  |  |
|  | //To process ',', add space on both side and save in cmd |
|  | if( ch\_in == ','){ |
|  | cmd[i++] = ' '; |
|  | cmd[i++] = ','; |
|  | cmd[i++] = ' '; |
|  | command[j++] = ch\_in; //command doesn't not space |
|  | } |
|  |  |
|  | //save charater |
|  | else{ |
|  | cmd[i++] = ch\_in; |
|  | command[j++] = ch\_in; |
|  | } |
|  |  |
|  | //if input is too long, then print error massage and return -1 |
|  | if(i>=COMMANDSIZE-1){ |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | } |
|  | } |
|  |  |
|  | //add '\0' to end of input |
|  | cmd[i] = '\0'; |
|  | command[j] = '\0'; |
|  |  |
|  | //return processed Command value |
|  | return ProcessCommand(cmd); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | int ProcessCommand(char\* cmd){ |
|  | //Tokenize cmd and separate the command and parameter |
|  | //If command is valied, return command value |
|  | //If not, return -1 |
|  |  |
|  | int i, j; |
|  | char \*tk; //string token |
|  | char tmp[COMMANDSIZE]; |
|  |  |
|  | //Input Command Token |
|  | //and classify command and parameter |
|  | i=-1; |
|  | strcpy(tmp, cmd); |
|  |  |
|  | tk = strtok(tmp, " \t"); |
|  | if(!tk) return -1; |
|  | do{ |
|  | //classify command and parameter |
|  | if(i==-1) strcpy(cmd, tk); |
|  | else strcpy(par[i],tk); |
|  | i++; |
|  |  |
|  | //If paramater is too much, print error message return -1 |
|  | if(i>=MAX\_PARAMETER){ |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | } |
|  | }while( (tk = strtok(NULL, " \t") ) ); |
|  |  |
|  | //If there is no input, return -1 |
|  | if(cmd[0] == '\0') return -1; |
|  |  |
|  | //Check command {H = 0, D = 1, q = 2, ... ,TYPE = 12 } |
|  | //If it matches format, return the enum value of the cmd. If not, return -1; |
|  | for(i=0; i<13 ;i++) |
|  | for(j=0 ; j<2 ; j++) |
|  | if(!strcmp(cmd, command\_list[i][j])) |
|  | return CheckParameter(i); |
|  |  |
|  | //If commad is invalid, return -1 |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | } |
|  |  |
|  | int CheckParameter(int cmd\_num){ |
|  | //Check that parameter according to com\_num is valid |
|  | //If parameter is valid, return cmd\_num |
|  | //else, return -1 |
|  |  |
|  | switch(cmd\_num){ |
|  |  |
|  | //command without parameter |
|  | case H: |
|  | case D: |
|  | case Q: |
|  | case HI: |
|  | case RESET: |
|  | case OPCODELIST: |
|  | case SYMBOL: |
|  | if(par[0][0]=='\0') |
|  | return cmd\_num; |
|  |  |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | break; |
|  |  |
|  |  |
|  | //Check parameter of DUMP |
|  | case DU: |
|  | //If there is no parameter, return cmd\_num |
|  | if(par[0][0] == '\0') |
|  | return cmd\_num; |
|  |  |
|  | //If there is parameter, check them |
|  | else if( IsHex(par[0]) ){ //check first parameter is hexadecimal |
|  | //Check the range exceed, return -1 |
|  | if( IsAddrLimitERROR( HexToDec(par[0]) ) ) |
|  | return -1; |
|  |  |
|  | //If there is valid one parameter, return comd\_num |
|  | if(par[1][0]=='\0') |
|  | return cmd\_num; |
|  |  |
|  | //Check the parameter valid if there is more than 1 |
|  | else if( par[1][0] == ',' && IsHex(par[2]) && par[3][0]=='\0'){ |
|  | if( IsAddrLimitERROR( HexToDec(par[2]) ) ) return -1; //check limit error |
|  | if( IsAddrERROR( HexToDec(par[0]) ,HexToDec(par[2]) ) ) return -1; //check address error |
|  | return cmd\_num; |
|  | } |
|  | } |
|  |  |
|  | //Print error message and return -1 if parameter is invalid. |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | break; |
|  |  |
|  |  |
|  | //Check parameter of EDIT |
|  | case E: |
|  |  |
|  | //Check parameter format |
|  | if(IsHex(par[0]) && par[1][0] ==',' && IsHex(par[2]) && par[3][0] == '\0'){ |
|  | if(IsAddrLimitERROR( HexToDec(par[0]) ))return -1; //check limit error |
|  | if( IsDataLimitERROR( HexToDec(par[2]) ) )return -1; //check limit error |
|  | return E; |
|  | } |
|  |  |
|  | //Print error message and return -1 if parameter is invalid |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | break; |
|  |  |
|  |  |
|  | //Check parameter of FILL |
|  | case F: |
|  | if(IsHex(par[0]) && par[1][0] ==',' && IsHex(par[2]) |
|  | && par[3][0] ==',' && IsHex(par[4]) && par[5][0]=='\0'){ |
|  |  |
|  | if(IsAddrLimitERROR( HexToDec(par[0]) ))return -1; |
|  | if( IsAddrLimitERROR( HexToDec(par[2]) ) )return -1; |
|  | if( IsAddrERROR( HexToDec(par[0]), HexToDec(par[2]) ) )return -1; |
|  | if( IsDataLimitERROR( HexToDec(par[4]) ) ) return -1; |
|  | return F; |
|  | } |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | break; |
|  |  |
|  | //Check parameter of OPCODEMNEMONIC |
|  | case OPCODEMNEMONIC: |
|  |  |
|  | //return -1 if no input |
|  | if(par[0][0]=='\0'){ |
|  | printf("No input mnemonic\n"); |
|  | return -1; |
|  | } |
|  |  |
|  | //Check the parameter format |
|  | else if(par[1][0]=='\0') |
|  | return OPCODEMNEMONIC; |
|  |  |
|  | //Print error message and return -1 if parameter is invlaid |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | break; |
|  |  |
|  | case ASSEMBLE: |
|  | if(par[0][0]=='\0'){ |
|  | printf("No input file name\n"); |
|  | return -1; |
|  | } |
|  | else if(par[1][0]=='\0') |
|  | return ASSEMBLE; |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | break; |
|  |  |
|  | case TYPE: |
|  | if(par[0][0]=='\0'){ |
|  | printf("No input file name\n"); |
|  | return -1; |
|  | } |
|  | else if(par[1][0]=='\0') |
|  | return TYPE; |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | break; |
|  |  |
|  |  |
|  | //Print error message and return -1 if parameter is invlaid |
|  | default: |
|  | PrintCmdERROR(); |
|  | return -1; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Make Hash Table to save OPCODE \*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | int MakeHashTable(){ |
|  | //Function to make Hash table |
|  | //It read "opcode.txt" and save information to opcode list |
|  |  |
|  | FILE \*fp=fopen("opcode.txt","r"); //file pointer |
|  | char in2[10],in3[5]; //input from "opcode.txt" |
|  | int in1; |
|  | // If file pointer is NULL, print error message and return -1 |
|  | if(!fp) { |
|  | printf("No \"opcode.txt\"\n"); |
|  | return -1; |
|  | } |
|  |  |
|  | // initialize hash table |
|  | for(int i=0; i<HASH\_SIZE ; i++){ |
|  | hash[i].size = 0; |
|  | hash[i].head = NULL; |
|  | hash[i].rear = NULL; |
|  | } |
|  |  |
|  | // Opcode nodes store input from "opcode.txt" |
|  | while( fscanf(fp,"%X %s %s",&in1,in2,in3)!=EOF ){ |
|  | MakeOpcodeList(in1,in2,in3); |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  | void MakeOpcodeList(int opcode, char\*mnemonic, char\* mode){ |
|  | // Make opcode node and Save opcode information using hashing |
|  | // Hash value : add up the ASCII code of mnemonic characters and modular 20 |
|  |  |
|  | int i; |
|  | int hash\_val=0; |
|  | opcode\_node\* new = (opcode\_node\*)malloc(sizeof(opcode\_node)); //Allocate memory for opcode node |
|  |  |
|  | //get hash value; |
|  | for(i=0 ; i<(int)strlen(mnemonic) ; i++) |
|  | hash\_val += mnemonic[i]; // add up the ASCII code of mnemonic |
|  | hash\_val%=HASH\_MOD; // modular 20 |
|  |  |
|  | //save the information of the opcode |
|  | new->opcode = opcode; |
|  | strcpy( (new->mnemonic), mnemonic); |
|  | strcpy( new->form , mode); |
|  | new->next = NULL; |
|  |  |
|  | //link the node to hash table according to hash\_val |
|  | hash[hash\_val].size++; |
|  | if(hash[hash\_val].head==NULL){ |
|  | hash[hash\_val].head = new; |
|  | hash[hash\_val].rear = new; |
|  | } |
|  | else{ |
|  | hash[hash\_val].rear->next = new; |
|  | hash[hash\_val].rear = new; |
|  | } |
|  | } |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Save COMMAND History \*\*\*\*\*\*\*\*\*\*\*/ |
|  | void AddHistory(){ |
|  | //Save the command hisotry to history node |
|  |  |
|  | his\_node \*new = (his\_node\*)malloc(sizeof(his\_node)); //allocate memory for his\_node |
|  |  |
|  | //save command to history node |
|  | strcpy(new->data, command); |
|  | new->next = NULL; |
|  |  |
|  | //link the node |
|  | if(his\_head == NULL){ |
|  | his\_head = new; |
|  | his\_rear = new; |
|  | } |
|  | else{ |
|  | his\_rear->next = new; |
|  | his\_rear = new; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Memeory Free \*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | void FreeHistory(){ |
|  | //Memory free related to history node |
|  |  |
|  | his\_node \*tmp\_p,\*del\_p; |
|  |  |
|  | if(his\_head==NULL) return; |
|  | else{ |
|  | tmp\_p = his\_head; //find head node |
|  | while(1){ //free his\_node |
|  | if(tmp\_p==NULL) return; |
|  | del\_p = tmp\_p; |
|  | tmp\_p = tmp\_p->next; |
|  | free(del\_p); |
|  | } |
|  | } |
|  | } |
|  | void FreeHash(){ |
|  | //Memroy free related to opcode node |
|  |  |
|  | opcode\_node \*tmp\_p, \*del\_p; |
|  |  |
|  | for(int i=0 ;i<HASH\_SIZE ; i++){ |
|  | tmp\_p = hash[i].head; //find head opcode\_node linked to hash table |
|  | while(1){ //free opcode\_node |
|  | if(tmp\_p == NULL) return; |
|  | del\_p = tmp\_p; |
|  | tmp\_p = tmp\_p->next; |
|  | free(del\_p); |
|  | } |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Help \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | void Help(){ |
|  | //Print all valid command |
|  |  |
|  | printf("h[elp]\n" |
|  | "d[ir]\n" |
|  | "q[uit]\n" |
|  | "hi[story]\n" |
|  | "du[mp] [start, end]\n" |
|  | "e[dit] address, value\n" |
|  | "f[ill] start, end, value\n" |
|  | "reset\n" |
|  | "opcode mnemonic\n" |
|  | "opcodelist\n" |
|  | "assemble filename\n" |
|  | "type filename\n" |
|  | "symbol\n" |
|  | ); |
|  | } |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Dir \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | int Dir(){ |
|  | //Function to print file information in current directory |
|  |  |
|  | struct dirent \*dir\_ent; //directory entry pointer |
|  | struct stat dir\_stat; //directory |
|  | DIR\* dir\_p = opendir("."); //directory pointer |
|  |  |
|  | if(dir\_p== NULL) return -1; |
|  |  |
|  | //read file information |
|  | while( ( dir\_ent = readdir(dir\_p) ) ){ |
|  | stat(dir\_ent->d\_name, &dir\_stat); |
|  |  |
|  | //classify and print file |
|  | if(S\_ISDIR(dir\_stat.st\_mode)) |
|  | printf("%s/\n",dir\_ent->d\_name); |
|  | else if(dir\_stat.st\_mode & S\_IXUSR) |
|  | printf("%s\*\n",dir\_ent->d\_name); |
|  | else |
|  | printf("%s\n",dir\_ent->d\_name); |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  | void History(){ |
|  | //Function to print history |
|  |  |
|  | his\_node\* hp; //his\_node pointer |
|  | int cnt=0; //count the line number to print |
|  |  |
|  | //no history |
|  | if(his\_head==NULL) return; |
|  |  |
|  | //print history |
|  | for(hp = his\_head ; hp!= NULL ;hp = hp->next){ |
|  | printf("%-3d %s\n",++cnt,hp->data); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Dump \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | void Dump(){ |
|  | //Function to process e and s |
|  | //and print data in memory form s to e |
|  |  |
|  | int s=0, e=0; //decimal value of start and end |
|  | char start[10], end[10]; //hex value of start and end |
|  |  |
|  | //move the parameter to start and end |
|  | strcpy(start, par[0]); |
|  | strcpy(end, par[2]); |
|  |  |
|  | //If no input about start, set s and e toprint 160 from last address |
|  | if(start[0] == '\0'){ |
|  |  |
|  | //Is last address is MEX\_MEMORY-1, set s is |
|  | if(last\_addr == MAX\_MEMORY-1) s = 0; |
|  |  |
|  | else s = last\_addr+1; |
|  | e = s+159; |
|  | } |
|  |  |
|  | //If exist input about start |
|  | else{ |
|  | s = HexToDec(start); //get decimal value s from start |
|  |  |
|  | //If only parameter about start, e is s+159 |
|  | if(end[0]=='\0') |
|  | e = s+159; |
|  |  |
|  | else e = HexToDec(end); //get decimal value e from end |
|  | } |
|  |  |
|  | //If e is excced the MAX\_MEMROY, reset e |
|  | if(e>= MAX\_MEMORY){ |
|  | e = MAX\_MEMORY-1; |
|  | } |
|  |  |
|  | //Print data in memory form s to e |
|  | PrintData(s,e); |
|  |  |
|  | //save last address |
|  | last\_addr = e; |
|  | } |
|  |  |
|  | void PrintData(int s,int e){ |
|  | //Function to print according to format the data in memory(mem) at address(addr) |
|  |  |
|  | int i,j; |
|  |  |
|  | //print from s to e |
|  | for(i=s ;i<=e; i++){ |
|  |  |
|  | //if i is start index print data |
|  | if(i==s){ |
|  | printf("%05X ",i); |
|  | if(i%16!=0){ |
|  | for(j=0 ;j<i%16 ; j++) |
|  | printf(" "); |
|  | } |
|  |  |
|  | printf("%02X ",HexToDec(mem[i])); |
|  |  |
|  | if(i%16==15) PrintASCII(s,e,i-i%16); |
|  | continue; |
|  | } |
|  |  |
|  | //print data |
|  | if(i%16==0) printf("%05X ",i); |
|  | printf("%02X ",HexToDec(mem[i])); |
|  |  |
|  | if(i%16==15) PrintASCII(s,e,i-i%16); |
|  | } |
|  |  |
|  | //print rest data to according to format |
|  | if(e%16!=15){ |
|  | for(; i<e-e%16+16; i++) |
|  | printf(" "); |
|  | PrintASCII(s,e,e-e%16); |
|  | } |
|  | } |
|  | void PrintASCII(int s, int e, int addr ){ |
|  | // Function outputs ASCII data of the line within addr. |
|  | // consider s and e to the fit format. |
|  |  |
|  | int i; |
|  | int dec; //decimal value of the data |
|  |  |
|  |  |
|  | printf("; "); |
|  |  |
|  | for( i=addr; i < addr+16 ; i++){ |
|  | dec = HexToDec(mem[i]); //get decimal value |
|  |  |
|  | //if 'dec: 20~7E' and 'i : s~e' print ASCII |
|  | if( (32<=dec && dec<=126) && (s<=i&&i<=e) ) |
|  | printf("%c",dec); |
|  | //if not print "." |
|  | else printf("."); |
|  | } |
|  | puts(""); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Edit \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | void Edit(){ |
|  | // Function to modify data in the memory |
|  | char val[8], addr[8]; |
|  |  |
|  | //get addr information and val information from par |
|  | strcpy(addr, par[0]); |
|  | strcpy(val, par[2]); |
|  |  |
|  | //modify data in mem at addr to val |
|  | strcpy(mem[HexToDec(addr)],val); |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Fill \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | void Fill(){ |
|  | //Function to fill from s(start) ~ e(end) |
|  |  |
|  | int i; |
|  | int s,e; //decimal number of start and end |
|  | char start[8],end[8],value[8]; |
|  |  |
|  | //get start, end, value from parameter |
|  | strcpy(start, par[0]); |
|  | strcpy(end, par[2]); |
|  | strcpy(value, par[4]); |
|  |  |
|  | //convert to decimal number |
|  | s = HexToDec(start); |
|  | e = HexToDec(end); |
|  |  |
|  | //Change data to value from s to e |
|  | for(i=s; i<=e ; i++){ |
|  | strcpy(mem[i],value); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Reset \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | void Reset(){ |
|  | //Function to fill all memory "00" |
|  |  |
|  | int i=0; |
|  | char tmpstr[] = "00"; //string to copy |
|  |  |
|  | //Make all data in memory "00" |
|  | for(i=0 ; i<MAX\_MEMORY ; i++){ |
|  | strcpy(mem[i],tmpstr); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* OPCODE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | void Opcodelist(){ |
|  | //Function to print opcode list |
|  |  |
|  | int i=0,flag=0; //i:index , flag: if flag is 1, print '->' |
|  | opcode\_node\* tmp; //opcode\_list pointer for searching |
|  |  |
|  |  |
|  | for(i=0 ; i<HASH\_MOD ; i++){ |
|  | printf("%d : ",i); //print hash number in hash table |
|  | flag = 0; //initialize flag to be 0 |
|  |  |
|  | //Check the hash is empty |
|  | if(hash[i].head == NULL) { |
|  | printf("empty\n"); |
|  | continue; |
|  | } |
|  |  |
|  | //Print data in opcode\_node connected to the hash |
|  | for(tmp = hash[i].head; tmp!=NULL ; tmp=tmp->next){ |
|  | if(flag) printf(" -> "); |
|  | else flag = 1; |
|  | printf("[%s,%X]",tmp->mnemonic,tmp->opcode); |
|  | } |
|  | puts(""); |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  | int OpcodeMnemonic(){ |
|  | //Function to find opcode. |
|  | //if you find the opcode print and return 1. |
|  | //if the opcode is not exist, print error message and return 0. |
|  |  |
|  | opcode\_node\* tmp; // opcode\_list pointer for searching |
|  | char key[10]; // mnemonic of the opcode you want to obtain |
|  | int hash\_val=0,i; // idx: hash table index, i:index |
|  |  |
|  | strcpy(key, par[0]); //key get mnemonic to find |
|  |  |
|  | //get hash value |
|  | for( i=0; i<(int)strlen(key) ; i++) |
|  | hash\_val += key[i]; |
|  | hash\_val %= HASH\_MOD; |
|  |  |
|  | //find opcode using hash value |
|  | for(tmp = hash[hash\_val].head; tmp!=NULL; tmp = tmp->next){ |
|  | if(!strcmp( key, tmp->mnemonic)) { |
|  | printf("opcode is %X\n",tmp->opcode); |
|  | return 1; |
|  | } |
|  | } |
|  |  |
|  | //If you can't find the opcode, return -1 |
|  | printf("OPCODE not found\n"); |
|  | return -1; |
|  | } |
|  |  |
|  | int FindOpcode(char\* key){ |
|  | //Function to find Opcode |
|  | //If same mnemonic as key exists, return opcode |
|  | //Else, return -1 |
|  |  |
|  | opcode\_node\* tmp; // opcode\_list pointer for searching |
|  | int hash\_val=0,i; // idx: hash table index, i:index |
|  | //get hash value |
|  | for( i=0; i<(int)strlen(key) ; i++) |
|  | hash\_val += key[i]; |
|  | hash\_val %= HASH\_MOD; |
|  |  |
|  | for(tmp = hash[hash\_val].head; tmp!=NULL; tmp = tmp->next){ |
|  | if(!strcmp( key, tmp->mnemonic)) { |
|  | return tmp->opcode; |
|  | } |
|  | } |
|  |  |
|  | //If you can't find the opcode, return -1 |
|  | return -1; |
|  | } |
|  |  |
|  | char\* FindForm(char\* key){ |
|  | //Function to find format |
|  | //If same mnemonic as key exists, return format |
|  | //Else, return -1 |
|  |  |
|  | opcode\_node\* tmp; // opcode\_list pointer for searching |
|  | int hash\_val=0,i; // idx: hash table index, i:index |
|  |  |
|  | //get hash value |
|  | for( i=0; i<(int)strlen(key) ; i++) |
|  | hash\_val += key[i]; |
|  | hash\_val %= HASH\_MOD; |
|  |  |
|  | for(tmp = hash[hash\_val].head; tmp!=NULL; tmp = tmp->next){ |
|  | if(!strcmp( key, tmp->mnemonic)) { |
|  | return tmp->form; |
|  | } |
|  | } |
|  |  |
|  | //If you can't find the opcode, return -1 |
|  | return NULL; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Processing Hexadecimal \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | int HexToDec(char\* hex){ |
|  | //Function to convert Hexadecimal to Decimal |
|  |  |
|  | int i; |
|  | int tmp, dec=0; |
|  |  |
|  | for(i= 0 ; i< (int)strlen(hex); i++){ |
|  | if(hex[i]=='\0') break; //if the char is '\0', then break |
|  |  |
|  | tmp=hex[i]; //get char in hex[i] |
|  |  |
|  | //save number to tmp using ASCII value |
|  | if('a'<=tmp && tmp<='f') tmp = tmp - 'a' + 10; |
|  | else if('A'<=tmp && tmp<='F') tmp = tmp - 'A' +10; |
|  | else if('0'<=tmp && tmp<= '9') tmp -= '0'; |
|  |  |
|  | //calculate hexadecimal vale to decimal value |
|  | dec \*= 16; |
|  | dec += tmp; |
|  | } |
|  |  |
|  | //return decimal value |
|  | return dec; |
|  | } |
|  |  |
|  | int IsHex( char \*ckstr ){ |
|  | //Check that the string(ckstr) is Hexadecimal |
|  | //if ckstr is Hexadecimal, return 1 |
|  | //if not, return 0 |
|  |  |
|  | int i; |
|  |  |
|  | //Check that ckstr is empty |
|  | if(ckstr[0] == '\0') return 0; |
|  |  |
|  | for(i=0; i<(int)strlen(ckstr); i++){ |
|  |  |
|  | //check the character in ckstr[i] |
|  | if(ckstr[i]<'0' || (ckstr[i] > '9' && ckstr[i] <'A') || (ckstr[i]>'F'&&ckstr[i]<'a') || ckstr[i]>'f') |
|  | return 0; |
|  | } |
|  |  |
|  | return 1; |
|  | } |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Check ERROR / Print ERROR MESSAGE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | void PrintCmdERROR(){ |
|  | //Function to print error message about invalid command |
|  |  |
|  | printf("Unknown Command!\n"); |
|  | } |
|  |  |
|  |  |
|  | int IsDataLimitERROR(int data){ |
|  | //function to check data limit |
|  | //if data exceed limit(0x00 ~ 0xff), then return 1 |
|  | //if not, return 0 |
|  |  |
|  | if(data<MAX\_DATA) |
|  | return 0; |
|  | else{ |
|  | printf("Exceede limit!\n"); |
|  | return 1; |
|  | } |
|  | } |
|  |  |
|  | int IsAddrERROR(int s, int e){ |
|  | //Function to check start address and end address |
|  | //If s is bigger than e, then return 1 |
|  | //If not, return 0 |
|  |  |
|  | if( e < s){ |
|  | printf("End address shouldn't be less than Start address!\n"); |
|  | return 1; |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  | int IsAddrLimitERROR(int addr){ |
|  | //function to check adrress limit |
|  | //if adddr exceed limit(0x00000 ~ 0xfffff), then return 1 |
|  | //if not, return 0 |
|  |  |
|  | if(addr < MAX\_MEMORY) |
|  | return 0; |
|  | else{ |
|  | printf("Exceeded limit!\n"); |
|  | return 1; |
|  | } |
|  | } |

**<main.h>**

|  |
| --- |
| #define COMMANDSIZE 256 //max command size |
|  | #define MAX\_PARAMETER 8 //max number of parameter |
|  | #define MAX\_MEMORY 1048576 //max memory size (0x0000~0xffff) |
|  | #define MAX\_DATA 256 //max data size (0x00~0xff) |
|  | #define HASH\_SIZE 20 //hash size |
|  | #define HASH\_MOD 20 //modular to get modual value |
|  |  |
|  |  |
|  |  |
|  | #define MAX\_LINESIZE 120 |
|  | #define MAX\_FILENAME 55 |
|  | #define MAX\_ASM\_TOKEN 7 |
|  | #define PRINT\_ERROR( line , c ) printf("line %d ERROR:: %s\n",line,c) |
|  | #define OBJ\_LINE\_SIZE 30 |
|  |  |
|  | /\*\*node to save History\*\*/ |
|  | typedef struct HistoryNode{ |
|  | char data[COMMANDSIZE]; |
|  | struct HistoryNode \*next; |
|  | }his\_node; |
|  |  |
|  | /\*Node to save OpcodeList using hash table\*/ |
|  | typedef struct OpcodeNode{ |
|  | struct OpcodeNode\* next; |
|  | int opcode; |
|  | char mnemonic[10]; |
|  | char form[5]; |
|  | }opcode\_node; |
|  |  |
|  | typedef struct{ |
|  | int size; |
|  | opcode\_node \*head; |
|  | opcode\_node \*rear; |
|  | }hash\_table; |
|  |  |
|  |  |
|  | /\*\*symbol\*\*/ |
|  | typedef struct SymbolNode{ |
|  | char sym[12]; |
|  | int loc; |
|  | struct SymbolNode \*next; |
|  | } symbol\_node; |
|  |  |
|  |  |
|  | typedef struct SymbolTableNode{ |
|  | char alpha; |
|  | struct SymbolNode \*next; |
|  | } symbol\_table\_node; |
|  |  |
|  |  |
|  | typedef struct AssemNode{ |
|  | int t\_flag; |
|  |  |
|  | int line; |
|  | int loc; |
|  | int size; |
|  | int type; |
|  | int form; |
|  |  |
|  | int addr\_mode; |
|  |  |
|  | unsigned obj; |
|  | int opcode; |
|  | char comment[55]; |
|  | char sym[12]; |
|  | char inst[12]; |
|  | char operand[2][12]; |
|  |  |
|  | struct AssemNode \*next; |
|  | }assem\_node; |

**<20161622.h>**

|  |
| --- |
| /\* |
|  | +--------------------------------------------------------------+ |
|  | | System Programming Project 2 | |
|  | | ( SIC/XE Assembler ) | |
|  | | | |
|  | | File : 20161622.c | |
|  | | Created : Mar 18 ~ Apr 9, 2018 | |
|  | | Author : Ye-eun Lee | |
|  | +--------------------------------------------------------------+ |
|  | \*/ |
|  |  |
|  |  |
|  | /\* Function to execute Command \*/ |
|  | void Help(); |
|  | int Dir(); |
|  | void History(); |
|  | void Dump(); |
|  | void Edit(); |
|  | void Fill(); |
|  | void Reset(); |
|  | int Input(); |
|  | int OpcodeMnemonic(); |
|  | void Opcodelist(); |
|  |  |
|  | /\*Function to Initialize\*/ |
|  | void ParInit(); |
|  | void MemInit(); |
|  |  |
|  | /\*Function to Free\*/ |
|  | void FreeHistory(); |
|  | void FreeHash(); |
|  |  |
|  | /\*Function to save command history\*/ |
|  | void AddHistory(); |
|  |  |
|  | /\*Function to Process Hexadecimal \*/ |
|  | int IsHex(char \*ckstr ); |
|  | int HexToDec(char \*hex); |
|  |  |
|  |  |
|  | /\*Function to check and process command \*/ |
|  | int ProcessCommand(char\* cmd); |
|  | int CheckParameter(int cmd\_num); |
|  |  |
|  |  |
|  | /\*Function to Make Opcode list using hash table\*/ |
|  | int MakeHashTable(); |
|  | void MakeOpcodeList(int opcode, char\* mnemonic, char\* mode); |
|  |  |
|  |  |
|  | /\*Function to print according to format data in memory\*/ |
|  | void PrintData(int s, int e); |
|  | void PrintASCII(int s, int e, int addr); |
|  |  |
|  |  |
|  | /\*ERROR FUNCTION\*/ |
|  | int IsAddrLimitERROR(int addr); |
|  | int IsAddrERROR(int s, int e); |
|  | int IsDataLimitERROR(int data); |
|  | void PrintCmdERROR(); |

**<assembler.c>**

|  |
| --- |
| /\* |
|  | +--------------------------------------------------------------+ |
|  | | System Programming Project 2 | |
|  | | ( SIC/XE Assembler ) | |
|  | | | |
|  | | File : assembler.c | |
|  | | Created : Mar 18 ~ Apr 9, 2018 | |
|  | | Author : Ye-eun Lee | |
|  | +--------------------------------------------------------------+ |
|  | \*/ |
|  |  |
|  | #include "main.h" |
|  | #include "assembler.h" |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <string.h> |
|  |  |
|  | //enum to distinguish instruction type\*/ |
|  | enum ASSEM\_TYPE{ |
|  | ERROR = -1, |
|  | COMMENT = 0, |
|  | PSEUDO\_INST = 1, |
|  | INST= 2 |
|  |  |
|  | }; |
|  | //enum to distinguish pseudo instruction type |
|  | enum PSEUDO\_INSTR\_TYPE{ |
|  | START = 0, |
|  | END, |
|  | BASE, |
|  | BYTE, |
|  | WORD, |
|  | RESB, |
|  | RESW |
|  | }; |
|  |  |
|  | //enum about address\_mode and n,i bit |
|  | enum ADDRESS\_MODE{ |
|  | SIC = 0, |
|  | IMMED = 1, |
|  | INDIR, |
|  | SIMPLE |
|  | }; |
|  |  |
|  | //string about pseudo instruction |
|  | char pseudo\_instr[7][10]={ |
|  | "START","END","BASE","BYTE","WORD","RESB","RESW" |
|  | }; |
|  |  |
|  | //pc and base address |
|  | int pc\_addr=0,base\_addr=0; |
|  |  |
|  | //assemble node |
|  | assem\_node \*assem\_head = NULL; |
|  | assem\_node \*assem\_rear = NULL; |
|  |  |
|  | //symbol table |
|  | symbol\_table\_node symbol\_table[26]; |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*INITIALIZE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | void InitAssemNode(){ |
|  | //Initailize assem\_node |
|  | assem\_node \*tmp\_p, \*del\_p; |
|  |  |
|  | tmp\_p = assem\_head; |
|  | while(1){ |
|  | if(tmp\_p == NULL) return; |
|  | del\_p = tmp\_p; |
|  | tmp\_p = tmp\_p->next; |
|  | free(del\_p); |
|  | } |
|  |  |
|  | assem\_head = NULL; |
|  | assem\_rear = NULL; |
|  | } |
|  |  |
|  | void InitSymbolTable(){ |
|  | //Function to free and intialize symbol table. |
|  | int i=0; |
|  | symbol\_node \*tmp\_p, \*del\_p; |
|  |  |
|  | for(i=0 ; i<26; i++){ |
|  | //free |
|  | tmp\_p = symbol\_table[i].next; |
|  | while(1){ |
|  | if(tmp\_p == NULL) break; |
|  | del\_p = tmp\_p; |
|  | tmp\_p = tmp\_p->next; |
|  | free(del\_p); |
|  | } |
|  | //initialize |
|  | symbol\_table[i].alpha = 'a'+i; |
|  | symbol\_table[i].next = NULL; |
|  | } |
|  | } |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TYPE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | int Type(char \*file\_name){ |
|  | //excution TYPE command |
|  | //file open and print contents |
|  | FILE \*fp = fopen(file\_name,"r"); |
|  | char in; |
|  |  |
|  | if(!fp){ |
|  | printf("File not found\n"); |
|  | return -1; |
|  | } |
|  |  |
|  | while( fscanf(fp,"%c",&in)!=EOF ){ |
|  | printf("%c",in); |
|  | } |
|  |  |
|  | fclose(fp); |
|  | return 0; |
|  | } |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SYMBOL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | int Symbol(){ |
|  | //Excution symbol command |
|  | //print symbol table |
|  | int i; |
|  | symbol\_node\* cur; |
|  |  |
|  | if(assem\_head==NULL) { |
|  | printf("A file recently assembled doesn't exist!\n"); |
|  | return -1; |
|  | } |
|  | for( i='Z'-'A' ; i>=0; i--){ |
|  | for(cur = symbol\_table[i].next ; cur!=NULL; cur = cur->next){ |
|  | printf("\t%s\t%04X\n",cur->sym,cur->loc); |
|  | } |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | int SearchSymbol(char \*key){ |
|  | //Function to find symbol |
|  | //Find : return addr |
|  | //unfind : -1 |
|  | symbol\_node \*cur; |
|  |  |
|  | // If not format or table is empty, return -1 |
|  | if(key[0]<'A' || key[0]>'Z') |
|  | return -1; |
|  | if(symbol\_table[ key[0]-'A'].next == NULL) |
|  | return -1; |
|  |  |
|  |  |
|  | //Check symbol node linked with symbol table |
|  | for(cur = symbol\_table[ key[0]-'A' ].next; cur != NULL; ){ |
|  | //if(cur==NULL) break; |
|  | if(!strcmp(cur->sym, key)) return cur->loc; |
|  | cur = cur->next; |
|  | } |
|  | return -1; |
|  | } |
|  |  |
|  | int MakeSymbolTable(assem\_node \*new\_node){ |
|  | //Make symbol table using assem\_node |
|  | //symbol save in symbol\_node['A'=0 ~ 'F'15 ] |
|  | //If successfully end , return 0 |
|  | //It this code have error return -1 |
|  |  |
|  | symbol\_node \*cur,\*bef; |
|  | symbol\_node \*new\_sym; //node to save symbol |
|  | int start\_flag=1; |
|  | int i=1; |
|  |  |
|  | //if symbol is empty or start return 0 |
|  | if( !strcmp(new\_node->sym, "\0") ) return 0; |
|  | else if( !strcmp(new\_node->inst,"START") ) return 0; |
|  |  |
|  | //allocate symbol\_node and initial symbol\_node |
|  | new\_sym = (symbol\_node\*)malloc(sizeof(symbol\_node)); |
|  | strcpy( new\_sym -> sym , new\_node->sym); |
|  | new\_sym -> loc = new\_node -> loc; |
|  | new\_sym -> next = NULL; |
|  |  |
|  | //If symbol table[] is emtpy, link new symbol node and return 0 |
|  | if(symbol\_table[ (new\_node->sym[0]) - 'A' ].next == NULL ){ |
|  | symbol\_table[ ((new\_node->sym)[0]) - 'A' ].next = new\_sym; |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | //check overlap error! |
|  | if(SearchSymbol( new\_node->sym)!=-1){ |
|  | PRINT\_ERROR(new\_node->line , "Overlap symbols"); |
|  | return ERROR; |
|  | } |
|  |  |
|  | //find place to link |
|  | bef = symbol\_table[ (new\_node->sym)[0] - 'A' ].next; |
|  | for(cur = symbol\_table[ (new\_node->sym)[0] - 'A' ].next; cur!=NULL; ){ |
|  |  |
|  | //if new symbol is bigger than next symbol, |
|  | //link new symbol node before next symbol node |
|  | if( new\_sym->sym[i] > cur->sym[i]){ |
|  | if(start\_flag) { |
|  | new\_sym->next = bef; |
|  | symbol\_table[(new\_node->sym)[0]-'A'].next = new\_sym; |
|  | } |
|  | else { |
|  | bef->next = new\_sym; |
|  | new\_sym->next = cur; |
|  | } |
|  | break; |
|  | } |
|  |  |
|  | //if same alphabet check to next alphabet |
|  | else if( new\_sym->sym[i]== cur->sym[i] ){ |
|  | i++; |
|  | } |
|  | else{ |
|  | bef = cur; |
|  | cur = cur->next; |
|  | } |
|  | start\_flag = 1; |
|  | } |
|  |  |
|  | //linke to last |
|  | if(cur == NULL){ |
|  | new\_sym->next = NULL; |
|  | bef->next = new\_sym; |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ASSEMBLE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | int Assemble(char \*file\_name){ |
|  | //Excution Assemble command(assemble file and make '.lst''.obj') |
|  | //if .asm file have error return -1 |
|  |  |
|  | FILE \*fp=NULL; |
|  |  |
|  | //check file type and name |
|  | if(!IsAssemFile(file\_name)){ |
|  | printf("%s is not .asm file\n", file\_name); |
|  | return -1; |
|  | } |
|  |  |
|  | fp = fopen(file\_name, "r"); |
|  |  |
|  | if(fp==NULL){ |
|  | printf("%s not founded\n",file\_name); |
|  | return -1; |
|  | } |
|  |  |
|  | fclose(fp); |
|  |  |
|  | //initialization |
|  | pc\_addr=0,base\_addr=0; |
|  | assem\_head = NULL; |
|  | assem\_rear = NULL; |
|  | InitAssemNode(); |
|  | InitSymbolTable(); |
|  |  |
|  | //process string and symbol table |
|  | if(AssemPass1(file\_name)==-1){ |
|  | InitAssemNode(); |
|  | InitSymbolTable(); |
|  | return -1; |
|  | } |
|  |  |
|  | //make '.lst' '.obj' |
|  | if(AssemPass2(file\_name)==-1){ |
|  | InitAssemNode(); |
|  | InitSymbolTable(); |
|  | return -1; |
|  | } |
|  |  |
|  |  |
|  | return 0; |
|  |  |
|  | } |
|  |  |
|  |  |
|  | int IsAssemFile(char \*file\_name){ |
|  | // Check that file type is '.asm' |
|  | // If filename in par[0] is '.asm' file, return 1 |
|  | // If not, return 0 |
|  | int i; |
|  | int check\_asm=0; |
|  |  |
|  | for(i=0 ; i< (int)strlen(file\_name) ; i++){ |
|  | if(file\_name[i]=='.'){ |
|  | if(strlen(file\_name)-i == 4){ |
|  | if(file\_name[i+1]=='a' && file\_name[i+2]=='s' && file\_name[i+3]=='m') return 1; |
|  | else return 0; |
|  | } |
|  | else return 0; |
|  | } |
|  | } |
|  | if(check\_asm == 4) return 1; |
|  |  |
|  | return 0; |
|  |  |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ASSEMBLE PASS1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  |  |
|  | int AssemPass1(char\* file\_name){ |
|  | //Function to read '.asm' file |
|  | //and make assem\_ node that have data about assemble code. |
|  |  |
|  | FILE \*fp = fopen(file\_name, "r"); |
|  | char asm\_line[MAX\_LINESIZE]; |
|  | char tk\_str[MAX\_ASM\_TOKEN][MAX\_LINESIZE] = {'\0'}; |
|  |  |
|  | //read and save data |
|  | while(fgets(asm\_line,MAX\_LINESIZE,fp)!=NULL){ |
|  | //token one line string in .asm file |
|  | AssemToken(asm\_line, tk\_str); |
|  |  |
|  | //make assemble node. If this line have error return -1 |
|  | if (MakeAssemNode(tk\_str) ) return ERROR; |
|  |  |
|  | //if end, finish this loop |
|  | if(FindPseudoInstr(assem\_rear->inst) == END) |
|  | break; |
|  |  |
|  | } |
|  | fclose(fp); |
|  | return 0; |
|  | } |
|  |  |
|  | int AssemToken(char asm\_line[], char tk\_str[][MAX\_LINESIZE]){ |
|  | int i=0, j=0; |
|  | char \*tk; |
|  | char asm\_str[MAX\_LINESIZE] = {'\0'}; |
|  |  |
|  |  |
|  | for(i=0; i < (int)strlen(asm\_line) ;){ |
|  | if(asm\_line[i]==','){ |
|  | i++; |
|  | asm\_str[j++] = ' '; |
|  | asm\_str[j++] = ','; |
|  | asm\_str[j++] = ' '; |
|  | continue; |
|  | } |
|  | asm\_str[j++]=asm\_line[i++]; |
|  | } |
|  | asm\_str[j] = '\0'; |
|  | for(i=0; i<MAX\_ASM\_TOKEN; i++){ |
|  | tk\_str[i][0] = '\0'; |
|  | } |
|  |  |
|  | i=0; |
|  |  |
|  | tk = strtok(asm\_str, " \t\n"); |
|  | if(tk==NULL) return 0; |
|  | do{ |
|  | strcpy(tk\_str[i],tk); |
|  |  |
|  | //process comment |
|  | if(tk\_str[i][0]=='.'){ |
|  | strcpy(tk\_str[i],"."); |
|  | strcpy(tk\_str[i+1],asm\_line); |
|  | break; |
|  | } |
|  |  |
|  | i++; |
|  |  |
|  | }while( (tk = strtok(NULL, " \t\n")) ); |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | int MakeAssemNode(char tk\_str[][MAX\_LINESIZE]){ |
|  | //make node to save data in assemble file using tk\_str(token string) |
|  | int type; //instruction type |
|  | assem\_node \*new\_node; //node to create |
|  | new\_node = (assem\_node\*)malloc( sizeof(assem\_node) ); |
|  |  |
|  | //node initialization; |
|  | memset(new\_node->comment, '\0', 55); |
|  | memset(new\_node->sym, '\0', 12); |
|  | memset(new\_node->inst, '\0', 12); |
|  | memset(new\_node->operand[0], '\0', 12); |
|  | memset(new\_node->operand[1], '\0', 12); |
|  |  |
|  | new\_node->obj = 0; |
|  | new\_node->next = NULL; |
|  | new\_node->type = -1; |
|  | new\_node->t\_flag = 0; |
|  | new\_node->loc = 0; |
|  | new\_node->size = 0; |
|  | new\_node->form = 0; |
|  | new\_node->opcode = 0; |
|  | new\_node->addr\_mode = 0; |
|  |  |
|  | new\_node->line = (assem\_head==NULL) ? 5 : ( (assem\_rear->line) + 5 ); |
|  |  |
|  | type=GetType\_and\_SaveInst(new\_node,tk\_str); |
|  |  |
|  | switch(type){ |
|  | case INST: |
|  | if(GetOperand(new\_node,tk\_str) == ERROR) return ERROR; |
|  | break; |
|  | case PSEUDO\_INST: |
|  | if(GetPseudoOperand(new\_node,tk\_str)==ERROR) return ERROR; |
|  | break; |
|  | case COMMENT: |
|  | break; |
|  | default: |
|  | return ERROR; |
|  | break; |
|  | } //link node |
|  |  |
|  | GetLoc(new\_node); |
|  | if(MakeSymbolTable(new\_node)==ERROR) return ERROR; |
|  |  |
|  | if(assem\_head==NULL){ |
|  | assem\_rear = new\_node; |
|  | assem\_head = new\_node; |
|  | } |
|  | else{ |
|  | assem\_rear->next = new\_node; |
|  | assem\_rear = new\_node; |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | void GetLoc(assem\_node \*new\_node){ |
|  | //function to get location using data in new\_node |
|  | int bef\_loc; |
|  |  |
|  | //if new\_node is first |
|  | if(assem\_rear == NULL){ |
|  | new\_node->loc = pc\_addr; |
|  | } |
|  |  |
|  | else{ |
|  | //get before location |
|  | bef\_loc = assem\_rear -> loc; |
|  |  |
|  | //get loction according to instruction type |
|  | switch(assem\_rear->type){ |
|  | case COMMENT: |
|  | new\_node->loc = bef\_loc; |
|  | break; |
|  | case PSEUDO\_INST: |
|  | //get loctaion accrding to pseudo instruction. |
|  | switch(FindPseudoInstr(assem\_rear->inst)){ |
|  | case START: |
|  | case BASE: |
|  | new\_node->loc = bef\_loc; |
|  | break; |
|  | case BYTE: |
|  | //if hexcadecimal |
|  | if(assem\_rear->operand[0][0] == 'X') |
|  | new\_node->loc = bef\_loc + (HexToDec(assem\_rear->operand[1])/ 256 )+ 1; |
|  | //if charictor |
|  | else if(assem\_rear->operand[0][0] == 'C') |
|  | new\_node->loc = bef\_loc + strlen(assem\_rear->operand[1]); |
|  | break; |
|  | case WORD: |
|  | new\_node->loc = bef\_loc + 3; |
|  | break; |
|  | case RESB: |
|  | new\_node->loc = bef\_loc + StrToDec(assem\_rear->operand[0]); |
|  | break; |
|  | case RESW: |
|  | new\_node->loc = bef\_loc + 3\*StrToDec(assem\_rear->operand[0]); |
|  | break; |
|  | } |
|  | break; |
|  |  |
|  | //get loction if instruction type is instruction |
|  | case INST: |
|  | new\_node->loc = bef\_loc + assem\_rear -> form; |
|  | break; |
|  |  |
|  | } |
|  | } |
|  | } |
|  |  |
|  |  |
|  | int GetType\_and\_SaveInst(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE]){ |
|  | //distinguish instruction type and save instruction and symbol |
|  | //and processing error about insruction and symbol |
|  |  |
|  | int opcode[3]; |
|  | int pseudo[3]; |
|  | int i; |
|  |  |
|  | //comment |
|  | if(tk\_str[0][0]=='.'){ |
|  | new\_node->type = COMMENT; |
|  | strcpy(new\_node->comment, tk\_str[1]); |
|  | return COMMENT; |
|  | } |
|  |  |
|  | //format 4 |
|  | for(i=0; i<2; i++){ |
|  | if(tk\_str[i][0]=='+') { |
|  | opcode[3] = FindOpcode(&tk\_str[i][1]); |
|  | if(opcode[3] != -1){ |
|  | if(!strcmp(FindForm(&tk\_str[i][1]),"3/4")){ |
|  | new\_node->form = 4; |
|  | new\_node->opcode = opcode[3]; |
|  | strcpy(new\_node->inst , &tk\_str[i][1]); |
|  | if(i==1){ |
|  | strcpy(new\_node->sym, tk\_str[0]); |
|  | } |
|  | new\_node->type = INST; |
|  | return INST; |
|  | } |
|  | else{ |
|  | PRINT\_ERROR(new\_node->line, "invalid intstruction"); |
|  | return ERROR; |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | //Get symbol, instruction, opcode o.w |
|  |  |
|  | //If this instruction has opcode or pseudo code, find and save to opcode[i] and pseudo[i] |
|  | //If not, opcode[i] or pseudo[i] is -1 |
|  | for(int i=0; i<2; i++){ |
|  | opcode[i] = FindOpcode(tk\_str[i]); |
|  | pseudo[i] = FindPseudoInstr(tk\_str[i]); |
|  | } |
|  |  |
|  |  |
|  | //Process ERROR: 2/no instruction in a line |
|  | if(opcode[0]==-1 && pseudo[0]==-1 |
|  | && opcode[1]== -1 && pseudo[1]==-1){ |
|  | PRINT\_ERROR(new\_node->line, "invalid instruction"); |
|  | return ERROR; |
|  | } |
|  | else if((opcode[0]!=-1 || pseudo[0]!=-1) |
|  | && (opcode[1]!= -1 || pseudo[1]!=-1)){ |
|  | PRINT\_ERROR(new\_node->line, "Symbol name overlaps with instruction"); |
|  | return ERROR; |
|  | } |
|  |  |
|  | //Separate instruction type(instruction / pseudo instruction) |
|  | for(int i=0; i<2; i++){ |
|  |  |
|  | //If exist symbol, save it |
|  | if(i==1){ |
|  | strcpy(new\_node->sym,tk\_str[0]); |
|  | } |
|  |  |
|  | //instruction type is instruction |
|  | if(opcode[i]!=-1){ |
|  | strcpy(new\_node->inst, tk\_str[i]); |
|  | new\_node->opcode = opcode[i]; |
|  | new\_node->form = FindForm(tk\_str[i])[0] - '0'; |
|  | new\_node->type = INST; |
|  | return INST; |
|  | } |
|  |  |
|  | //instruction type is pseudo instruction |
|  | else if(pseudo[i]!=-1){ |
|  | strcpy(new\_node->inst, tk\_str[i]); |
|  | new\_node->type = PSEUDO\_INST; |
|  | return PSEUDO\_INST; |
|  | } |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | int GetPseudoOperand(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE]){ |
|  | //Precess operand in pseudo instruction |
|  | //if this line has error, return -1 |
|  | //else return 0 |
|  |  |
|  | int i; |
|  | int is\_sym = (!strcmp(new\_node->sym , "\0")) ? 0 : 1; |
|  |  |
|  | //process operand according to instruction |
|  | switch(FindPseudoInstr(new\_node->inst)){ |
|  | case START: |
|  | // if opernad is hex, then save data |
|  | if(IsHex(tk\_str[1+is\_sym])){ |
|  | pc\_addr = HexToDec(tk\_str[1+is\_sym]); |
|  | strcpy( new\_node -> operand[0] , tk\_str[1+is\_sym]); |
|  | return 0; |
|  | } |
|  | else{ |
|  | PRINT\_ERROR(new\_node->line, "Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | break; |
|  | case END: |
|  | if(!strcmp(tk\_str[2+is\_sym],"\0") ){ |
|  | strcpy(new\_node -> operand[0], tk\_str[1+is\_sym]); |
|  | return 0; |
|  | } |
|  | else { |
|  | PRINT\_ERROR(new\_node->line, "1Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | break; |
|  | case BASE: |
|  | if(!strcmp(tk\_str[2+is\_sym],"\0") ){ |
|  | strcpy(new\_node -> operand[0], tk\_str[1+is\_sym]); |
|  | return 0; |
|  | } |
|  | break; |
|  | case BYTE: |
|  | case WORD: |
|  | //Check the format and find type of operator |
|  | //and process operand accorcing to operand type |
|  | if((tk\_str[1+is\_sym][0]=='X' ||tk\_str[1+is\_sym][0]=='C') |
|  | && tk\_str[1+is\_sym][strlen(tk\_str[1+is\_sym]) - 1] == '\'' |
|  | && (tk\_str[1+is\_sym][1]=='\'')){ |
|  |  |
|  | new\_node -> operand[0][0] = tk\_str[1+is\_sym][0]; |
|  | tk\_str[1+is\_sym][strlen(tk\_str[1+is\_sym]) - 1] = '\0'; |
|  | strcpy((new\_node -> operand[1]), &tk\_str[1+is\_sym][2]); |
|  |  |
|  | if(tk\_str[1+is\_sym][0]=='X' && !IsHex(new\_node->operand[1])){ |
|  | PRINT\_ERROR(new\_node->line, "data type error!"); |
|  | return ERROR; |
|  | } |
|  | return 0; |
|  | } |
|  | else if((tk\_str[1+is\_sym])){ |
|  | for( i=0; i<(int)strlen(tk\_str[1+is\_sym]); i++){ |
|  | if(tk\_str[1+is\_sym][i]<'0' || tk\_str[1+is\_sym][i]>'9'){ |
|  | PRINT\_ERROR(new\_node->line, "Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | } |
|  | strcpy(new\_node ->operand[0] , tk\_str[1+is\_sym]); |
|  | return 0; |
|  | } |
|  | else{ |
|  | PRINT\_ERROR(new\_node->line, "Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | case RESB: |
|  | case RESW: |
|  | for( i=0; i<(int)strlen(tk\_str[1+is\_sym]); i++){ |
|  | if(tk\_str[1+is\_sym][i]<'0' || tk\_str[1+is\_sym][i]>'9'){ |
|  | PRINT\_ERROR(new\_node->line, "Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | } |
|  | strcpy( new\_node->operand[0] ,tk\_str[1+is\_sym]); |
|  | break; |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | int GetOperand(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE]){ |
|  |  |
|  | int i; |
|  | int is\_sym = (!strcmp(new\_node->sym , "\0")) ? 0 : 1; //If this line has symbol add 1 |
|  | char oper2\_except[4][3][10] = { |
|  | {"CLEAR","TIXR"},//r1 |
|  | {"CVC","CVC"}, //n |
|  | {"SHIFTL", "SHIFTR"}//r1, n |
|  | }; //instruction to process exceptionlly in format 2 |
|  |  |
|  | switch(new\_node->form){ |
|  | //in format 1 |
|  | case 1: |
|  | //check error (has more instruction) |
|  | if(strcmp(tk\_str[1 + is\_sym],"\0")){ |
|  | PRINT\_ERROR(new\_node->line, "Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | return 0; |
|  | break; |
|  |  |
|  | //in format 2 |
|  | case 2: |
|  | //process command exceptionally in format 2 |
|  | for(i=0;i<=2;i++){ |
|  | if( ( !strcmp(oper2\_except[i][0],new\_node->inst) ) |
|  | || (!strcmp(oper2\_except[i][1],new\_node->inst)) ){ |
|  | switch(i){ |
|  | //"CLEAR","TIXR" - r1 |
|  | case 0: |
|  | if(IsReg(tk\_str[1+is\_sym])){ |
|  | strcpy(new\_node->operand[0] , tk\_str[1+is\_sym]); |
|  | return 0; |
|  | } |
|  | else{ |
|  | PRINT\_ERROR(new\_node->line, "Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | break; |
|  | //"CVC","CVC" - n |
|  | case 1: |
|  | if(IsHex(tk\_str[1+is\_sym])){ |
|  | strcpy(new\_node->operand[0], tk\_str[1+is\_sym]); |
|  | return 0; |
|  | } |
|  | else{ |
|  | PRINT\_ERROR(new\_node->line, "Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | break; |
|  | //"SHIFTL", "SHIFTR" - r1, n |
|  | case 2: |
|  | if(IsReg(tk\_str[1+is\_sym]) && IsHex(tk\_str[3+is\_sym])){ |
|  | strcpy(new\_node->operand[0], tk\_str[1+is\_sym]); |
|  | strcpy(new\_node->operand[1], tk\_str[3+is\_sym]); |
|  | return 0; |
|  | } |
|  | else{ |
|  | PRINT\_ERROR(new\_node->line, "Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | // r1, r2 |
|  | if(IsReg(tk\_str[1+is\_sym])&&IsReg(tk\_str[3+is\_sym])){ |
|  | strcpy(new\_node->operand[0], tk\_str[1+is\_sym]); |
|  | strcpy(new\_node->operand[1], tk\_str[3+is\_sym]); |
|  | return 0; |
|  | } |
|  | else{ |
|  | PRINT\_ERROR(new\_node->line, "Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | break; |
|  | case 3: |
|  | case 4: |
|  | //distinguish address mode |
|  | switch(tk\_str[1+is\_sym][0]){ |
|  | case '#': |
|  | strcpy(new\_node->operand[0], &tk\_str[1+is\_sym][1]); |
|  | new\_node -> addr\_mode = IMMED; |
|  | break; |
|  | case '@': |
|  | strcpy(new\_node->operand[0], &tk\_str[1+is\_sym][1]); |
|  | new\_node -> addr\_mode = INDIR; |
|  | break; |
|  | default: |
|  | strcpy(new\_node->operand[0], tk\_str[1+is\_sym]); |
|  | new\_node ->addr\_mode = SIMPLE; |
|  | break; |
|  | } |
|  |  |
|  |  |
|  | //no operand instruction |
|  | if( (!strcmp(new\_node->inst, "RSUB")) && (!strcmp(tk\_str[1+is\_sym],"\0"))) |
|  | return 0; |
|  | //if operand has x resgister |
|  | else if(!strcmp(tk\_str[3+is\_sym],"X")){ |
|  | strcpy(new\_node->operand[1],"X"); |
|  | return 0; |
|  | } |
|  | else if(!strcmp(tk\_str[2+is\_sym],"\0")){ |
|  | return 0; |
|  | } |
|  | else{ |
|  | PRINT\_ERROR(new\_node->line, "Incorrect format!"); |
|  | return ERROR; |
|  | } |
|  | break; |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ASSEMBLE PASS2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | int AssemPass2(char\* file\_name){ |
|  | //Make "\*.lst" file and "\*.obj" file |
|  |  |
|  | FILE \*fp = fopen(file\_name, "r"); //read .asm file |
|  | FILE \*list; //FILE POINTER for .lst file |
|  | FILE \*object; //FILE POINTER for .obj file |
|  | char list\_name[MAX\_FILENAME]; //string for list file name |
|  | char object\_name[MAX\_FILENAME]; //string for object file name |
|  | char asm\_line[MAX\_LINESIZE]; //string to get 1 line in .asm file |
|  | int i; |
|  | int t\_total=0,t\_end\_flag,t\_next\_loc; //t\_total : count byte size to print to object file |
|  | //t\_end\_flag : line feed sign to print to object file |
|  | assem\_node\* tcnt; |
|  | assem\_node\* cur = assem\_head; //Pointer to point current assem\_node. |
|  |  |
|  | //process file name to get list\_name, object\_name |
|  | for(i=0 ; i<(int)strlen(file\_name); i++){ |
|  | if(file\_name[i] == '.') |
|  | break; |
|  | list\_name[i] = file\_name[i]; |
|  | object\_name[i] = file\_name[i]; |
|  | } |
|  | strcat(list\_name, ".lst"); |
|  | strcat(object\_name, ".obj"); |
|  |  |
|  | //If '.asm' file is not emtpy or has "START" at first line, |
|  | //make list file and object file |
|  | //and write first line data to list and obj. |
|  | if(!strcmp(cur->inst,"START")||strcmp(cur->inst, "\0") ){ |
|  | list = fopen(list\_name,"w"); |
|  | object = fopen(object\_name,"w"); |
|  | if(fgets(asm\_line,MAX\_LINESIZE,fp) == NULL) return 0; |
|  | fprintf(list,"%d\t%04X\t%s",cur->line,cur->loc,asm\_line); |
|  | fprintf(object,"H%-6s%06X%06X",cur->sym,assem\_head->loc, (assem\_rear->loc - assem\_head->loc)); |
|  | cur=cur->next; |
|  | } |
|  |  |
|  | // Read data one line in '.asm' file at a time |
|  | // and write to '.lst' |
|  | while(fgets(asm\_line,MAX\_LINESIZE,fp) != NULL || cur==NULL ){ |
|  | if(asm\_line[(int)strlen(asm\_line)-1]=='\n') |
|  | asm\_line[(int)strlen(asm\_line)-1]='\0'; |
|  |  |
|  | //process and write comment to list file |
|  | if(cur->type == COMMENT) |
|  | fprintf(list,"%d\t%s\n",cur->line,asm\_line); |
|  |  |
|  | //If instruction is base, set base address and print to list file |
|  | else if(!strcmp(cur->inst,"BASE")){ |
|  | if(cur->operand[0][0]<='0' && cur->operand[0][0]>='9'){ |
|  | base\_addr = HexToDec(cur->operand[0]); |
|  | } |
|  | else{ |
|  | base\_addr = SearchSymbol(cur->operand[0]); |
|  | if(base\_addr == -1){ |
|  | remove(list\_name); |
|  | remove(object\_name); |
|  | PRINT\_ERROR(cur->line,"Referenced an undeclared symbol!"); |
|  | return -1; |
|  | } |
|  | } |
|  | fprintf(list,"%d\t\t%s\n",cur->line,asm\_line); |
|  | } |
|  |  |
|  |  |
|  | else{ |
|  | //If instruction is "END", print to list file and break this loop |
|  | if( !strcmp(cur->inst,"END")){ |
|  | cur->t\_flag=1; |
|  |  |
|  | fprintf(list,"%d\t\t%-33s\n",cur->line,asm\_line); |
|  | break; |
|  | } |
|  | fprintf(list,"%d\t%04X\t%-33s",cur->line,cur->loc,asm\_line); |
|  | cur->size = 0; |
|  | cur->t\_flag = 0; |
|  | //Get object code |
|  | //If it has error, return -1(ERROR) |
|  | if(GetObj(cur)==ERROR){ |
|  | remove(list\_name); |
|  | remove(object\_name); |
|  | return ERROR; |
|  | } |
|  | if(t\_total==0) { |
|  | cur->t\_flag = 1; |
|  | } |
|  | //If current line has print object code |
|  | if(cur->size!=0){ |
|  |  |
|  | //Add byte size to t\_total to make object file |
|  | //Print object code to list file according to size of object code |
|  | switch(cur->size){ |
|  | case 1: |
|  | t\_total+=1; |
|  | fprintf(list,"%02X",cur->obj); |
|  | break; |
|  | case 2: |
|  | t\_total+=2; |
|  | fprintf(list,"%04X",cur->obj); |
|  | break; |
|  | case 3: |
|  | t\_total+=3; |
|  | fprintf(list,"%06X",cur->obj); |
|  | break; |
|  | case 4: |
|  | t\_total+=4; |
|  | fprintf(list,"%08X",cur->obj); |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | // To match the object file format(consider linesize, constant ), |
|  | // check t\_flag at the beginning of the line |
|  | if(t\_end\_flag==1 || t\_total>OBJ\_LINE\_SIZE){ |
|  | //consider line size |
|  | cur->t\_flag = 1; |
|  | t\_end\_flag = 0; |
|  | t\_total = cur->size; |
|  | } |
|  | else if(!strcmp(cur->inst,"BYTE")||!strcmp(cur->inst,"WORD")){ |
|  | //consider constant |
|  | t\_end\_flag=1; |
|  | t\_total = 0; |
|  | } |
|  | fprintf(list,"\n"); |
|  | } |
|  |  |
|  | cur = cur->next; |
|  | } |
|  |  |
|  | //Make .obj file using data in assem\_node |
|  | for(cur = assem\_head->next; cur!=NULL ;cur = cur->next){ |
|  | //if end break this loop |
|  | if(!strcmp(cur->inst,"END")){ |
|  | break; |
|  | } |
|  | if(cur->size==0){ |
|  | continue; |
|  | } |
|  |  |
|  | if(cur->t\_flag){ |
|  | //get line size |
|  | for(tcnt=cur->next;tcnt!=NULL; tcnt=tcnt->next){ |
|  | if(tcnt->t\_flag==1){ |
|  | t\_next\_loc = tcnt->loc; |
|  | break; |
|  | } |
|  | } |
|  | //print 'T' and address |
|  | fprintf(object, "\nT%06X%02X",cur->loc,t\_next\_loc - (cur->loc) ); |
|  | } |
|  |  |
|  | //print object code to '.obj' according to size t |
|  | switch(cur->size){ |
|  | case 1: |
|  | fprintf(object,"%02X",cur->obj); |
|  | break; |
|  | case 2: |
|  | fprintf(object,"%04X",cur->obj); |
|  | break; |
|  | case 3: |
|  | fprintf(object,"%06X",cur->obj); |
|  | break; |
|  | case 4: |
|  | fprintf(object,"%08X",cur->obj); |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | //print modification to '.obj' |
|  | for(cur = assem\_head->next; cur!=NULL ;cur = cur->next){ |
|  | //if end, print 'X' and break this loop |
|  | if(!strcmp(cur->inst,"END")){ |
|  | fprintf(object, "\nE%06X\n",assem\_head->loc); |
|  | break; |
|  | } |
|  | //print 'M' and information about address |
|  | if(cur->form == 4 |
|  | && ('0'>cur->operand[0][0]||cur->operand[0][0]>'9')){ |
|  | fprintf(object, "\nM%06X%02X",(cur->loc)-(assem\_head->loc)+1,5); |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  | printf("output file : [%s], [%s]\n",list\_name,object\_name); |
|  |  |
|  | //file close |
|  | fclose(fp); |
|  | fclose(list); |
|  | fclose(object); |
|  | return 0; |
|  | } |
|  |  |
|  | int GetObj(assem\_node \*cur\_node){ |
|  | //Get obejct code in current node(cur\_node) |
|  | //If this node(cur\_node) has error, return -1 |
|  | //Else, return 0; |
|  | int sym\_addr; |
|  | int i=0; |
|  |  |
|  | //In pseudo instruction, set object code according to the instruction |
|  | if(cur\_node->type == PSEUDO\_INST){ |
|  | switch(FindPseudoInstr(cur\_node->inst)){ |
|  | case BYTE: |
|  | case WORD: |
|  | if(!strcmp(cur\_node->operand[0],"X")){ |
|  | cur\_node->size = (int)(strlen(cur\_node->operand[0])/2+strlen(cur\_node->operand[0])%2); |
|  | cur\_node->obj = HexToDec(cur\_node->operand[1]); |
|  | } |
|  | else if(!strcmp(cur\_node->operand[0],"C")){ |
|  | cur\_node->size = (int)strlen( cur\_node->operand[0] ); |
|  | for(i=0 ; i < strlen(cur\_node->operand[1]) ; i++){ |
|  | cur\_node->obj \*= (16\*16); |
|  | cur\_node->obj += cur\_node->operand[1][i]; |
|  | } |
|  | } |
|  | else{ |
|  | cur\_node->obj = StrToDec(cur\_node->operand[0]); |
|  | } |
|  | break; |
|  | default: |
|  | break; |
|  | } |
|  | } |
|  | //In instruction, Set object code accrding to the format |
|  | else if(cur\_node->type == INST){ |
|  | switch(cur\_node->form){ |
|  | //in format 1 |
|  | case 1: |
|  | cur\_node->size = 1; |
|  | cur\_node->obj = cur\_node->opcode; |
|  | break; |
|  | //in format 2 |
|  | case 2: |
|  | cur\_node->size = 2; |
|  | cur\_node->obj = (cur\_node->opcode)\*16\*16; |
|  | if(!strcmp(cur\_node->inst,"CLEAR")|| !strcmp(cur\_node->inst,"TIXR")){ |
|  | cur\_node->obj += FindReg(cur\_node->operand[0])\*16; |
|  | } |
|  | else if(!strcmp(cur\_node->inst,"CVC")){ |
|  | cur\_node->obj += HexToDec(cur\_node->operand[0])\*16; |
|  | } |
|  | else if(!strcmp(cur\_node->inst,"SHIFTL")|| !strcmp(cur\_node->inst,"SHIFTR")){ |
|  | cur\_node->obj += FindReg(cur\_node->operand[0])\*16; |
|  | cur\_node->obj += HexToDec(cur\_node->operand[1]); |
|  | } |
|  | else{ |
|  | cur\_node->obj += FindReg(cur\_node->operand[0])\*16; |
|  | cur\_node->obj += FindReg(cur\_node->operand[1]); |
|  | } |
|  | break; |
|  |  |
|  | //in format 3 |
|  | case 3: |
|  | pc\_addr = cur\_node->loc + 3; |
|  | cur\_node->size = 3; |
|  | cur\_node->obj = (cur\_node->opcode)\*16\*16\*16\*16; |
|  | if( !strcmp(cur\_node->operand[1], "X") ){ |
|  | cur\_node->obj += (2\*2\*2)\*(16\*16\*16); |
|  | } |
|  |  |
|  | //Set object code according to the address mode |
|  | switch(cur\_node->addr\_mode){ |
|  | //in immediate mode |
|  | case IMMED: |
|  | cur\_node->obj += (IMMED)\*16\*16\*16\*16; |
|  | if('0' <= cur\_node->operand[0][0] |
|  | && cur\_node->operand[0][0] <= '9') |
|  | cur\_node->obj += StrToDec(cur\_node->operand[0]); |
|  | else{ |
|  | sym\_addr = SearchSymbol(cur\_node->operand[0]); |
|  | if(sym\_addr==-1){ |
|  | PRINT\_ERROR(cur\_node->line,"Referenced an undeclared symbol!"); |
|  | return -1; |
|  | } |
|  | //pc relative |
|  | if( sym\_addr-pc\_addr >= -2048 && sym\_addr-pc\_addr <= 2047){ |
|  | if((sym\_addr-pc\_addr)<0) |
|  | cur\_node->obj += 16\*16\*16; |
|  | cur\_node->obj += sym\_addr-pc\_addr; |
|  | cur\_node->obj += 2\*16\*16\*16; |
|  | } |
|  | //base relative |
|  | else if( 0<=sym\_addr-base\_addr && sym\_addr-base\_addr <= 4095){ |
|  | cur\_node->obj += sym\_addr-base\_addr; |
|  | cur\_node->obj += 4\*16\*16\*16; |
|  | } |
|  | } |
|  | break; |
|  | //in indircet mode |
|  | case INDIR: |
|  | cur\_node->obj += (INDIR)\*16\*16\*16\*16; |
|  | sym\_addr = SearchSymbol(cur\_node->operand[0]); |
|  | if(sym\_addr==-1){ |
|  | PRINT\_ERROR(cur\_node->line,"Referenced an undeclared symbol!"); |
|  | return -1; |
|  | } |
|  | //pc relative |
|  | if( sym\_addr-pc\_addr >= -2048 && sym\_addr-pc\_addr <= 2047){ |
|  | if((sym\_addr-pc\_addr)<0) |
|  | cur\_node->obj += 16\*16\*16; |
|  | cur\_node->obj += sym\_addr-pc\_addr; |
|  | cur\_node->obj += 2\*16\*16\*16; |
|  | } |
|  | //base relative |
|  | else if( 0<=sym\_addr-base\_addr && sym\_addr-base\_addr <= 4095){ |
|  | cur\_node->obj += sym\_addr-base\_addr; |
|  | cur\_node->obj += 4\*16\*16\*16; |
|  | } |
|  | break; |
|  | //in simple addressing mode or SIC format |
|  | default: |
|  | if(!strcmp(cur\_node->operand[0],"\0")){ |
|  | cur\_node->obj += (SIMPLE)\*16\*16\*16\*16; |
|  | cur\_node->addr\_mode = SIMPLE; |
|  | } |
|  | else{ |
|  | sym\_addr = SearchSymbol(cur\_node->operand[0]); |
|  | if(sym\_addr==-1){ |
|  | PRINT\_ERROR(cur\_node->line,"Referenced an undeclared symbol!"); |
|  | return -1; |
|  | } |
|  | //pc relative |
|  | if( sym\_addr-pc\_addr >= -2048 && sym\_addr-pc\_addr <= 2047){ |
|  | cur\_node->obj += (SIMPLE)\*16\*16\*16\*16; |
|  | if((sym\_addr-pc\_addr)<0) |
|  | cur\_node->obj += 16\*16\*16; |
|  | cur\_node->obj += sym\_addr-pc\_addr; |
|  | cur\_node->obj += 2\*16\*16\*16; |
|  | } |
|  | //base relative |
|  | else if( 0<=sym\_addr-base\_addr && sym\_addr-base\_addr <= 4095){ |
|  | cur\_node->obj += (SIMPLE)\*16\*16\*16\*16; |
|  | cur\_node->obj += sym\_addr-base\_addr; |
|  | cur\_node->obj += 4\*16\*16\*16; |
|  | } |
|  | else{ |
|  | cur\_node->addr\_mode = SIC; |
|  | cur\_node->obj += sym\_addr; |
|  | } |
|  |  |
|  | } |
|  | break; |
|  | } |
|  |  |
|  |  |
|  | break; |
|  | case 4: |
|  | //format 4 |
|  | cur\_node->size = 4; |
|  | cur\_node->obj = ((cur\_node->opcode)+(cur\_node->addr\_mode))\*16\*16\*16\*16\*16\*16; |
|  | cur\_node->obj += 1\*16\*16\*16\*16\*16; |
|  | //simple address mode |
|  | if(cur\_node->addr\_mode == SIMPLE){ |
|  | if(SearchSymbol(cur\_node->operand[0])==-1){ |
|  | PRINT\_ERROR(cur\_node->line,"Referenced an undeclared symbol!"); |
|  | return -1; |
|  | } |
|  | cur\_node->obj += SearchSymbol(cur\_node->operand[0]); |
|  | } |
|  | //immediate address mode |
|  | else if(cur\_node->addr\_mode == IMMED){ |
|  | cur\_node->obj += StrToDec(cur\_node->operand[0]); |
|  | } |
|  | break; |
|  | } |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Process string/ decimal/hexadecimal \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | int StrToDec(char\* str){ |
|  | //change str to decimal number |
|  | //If str is emtpy, return -1 |
|  | int i=0; |
|  | int dec=0; |
|  | if(str[0] == '\0'){ |
|  | return -1; |
|  | } |
|  | for(i=0 ; i<(int)strlen(str); i++){ |
|  | dec \*= 10; |
|  | dec += str[i]-'0'; |
|  | } |
|  | return dec; |
|  | } |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Find data or check value \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | int FindReg(char \*str){ |
|  | //Find regisister and return resister number |
|  | //If str is not register, return -1 |
|  | int i; |
|  | char reg[9][4] = {"A","X","L","B","S","T","F","PC","SW"};//register terminate |
|  | for(i=0; i<9; i++){ |
|  | if(!strcmp(str,reg[i])) return i; |
|  | } |
|  | return -1; |
|  | } |
|  |  |
|  |  |
|  | int IsReg(char \*c){ |
|  | //If c is register, return 1 |
|  | //if c is not register, return 0 |
|  | int i; |
|  | char reg[10][3] = {"A","X","L","B","S","T","F","PC","SW"}; |
|  | for(i=0; i<9; i++){ |
|  | if(!strcmp(c, reg[i]) ){ |
|  | return 1; |
|  | } |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  | int FindPseudoInstr(char\* key){ |
|  | //If key is pseudo instruction, return pseudo number (START = 0,... ) |
|  | int i; |
|  | for(i=0; i < 7; i++){ |
|  | if(!strcmp(key, pseudo\_instr[i])) |
|  | return i; |
|  | } |
|  |  |
|  | return -1; |
|  | } |

**<assembler.h>**

|  |
| --- |
| /\* |
|  | +--------------------------------------------------------------+ |
|  | | System Programming Project 2 | |
|  | | ( SIC/XE Assembler ) | |
|  | | | |
|  | | File : assembler.h | |
|  | | Created : Mar 18 ~ Apr 9, 2018 | |
|  | | Author : Ye-eun Lee | |
|  | +--------------------------------------------------------------+ |
|  | \*/ |
|  |  |
|  | /\*initialization\*/ |
|  | void InitSymbolTable(); |
|  | void InitAssemNode(); |
|  |  |
|  | /\*assemble\*/ |
|  | int Assemble(char \*file\_name); //execute assembler |
|  | int IsAssemFile(char \*file\_name); //check that input is '.asm' file |
|  | int AssemPass1(char \*file\_name); //make assem\_node and symbol table |
|  | int AssemPass2(char\* file\_name); //make '.lst' and '.obj' file |
|  |  |
|  | /\*Process '.asm' file and make assem\_node\*/ |
|  | int MakeAssemNode(char tkstr[][MAX\_LINESIZE]); //make assem\_node |
|  | int AssemToken(char \*asm\_line, char tk\_str[][MAX\_LINESIZE]); //token string in '.asm' file |
|  | int GetType\_and\_SaveInst(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE]);//get instruction type and save istruction |
|  | void GetLoc(assem\_node \*new\_node); //get location |
|  | int GetObj(assem\_node \*cur\_node); //get object code |
|  |  |
|  |  |
|  | /\*Function to find and get some value\*/ |
|  | int FindReg(char \*str); |
|  | char\* FindForm(char\* key); |
|  | int FindPseudoInstr(char\* key); |
|  | int FindOpcode(char\* key); |
|  | int IsReg(char \*c); |
|  | /\*Type\*/ |
|  | int Type(char \*file\_name); |
|  | int GetOperand(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE]); |
|  | int GetPseudoOperand(assem\_node \*new\_node, char tk\_str[][MAX\_LINESIZE]); |
|  |  |
|  | /\*Symbol\*/ |
|  | int Symbol(); |
|  | int SearchSymbol(char \*key); |
|  | int MakeSymbolTable(assem\_node \*new\_node); |
|  |  |
|  | /\*Function to Process Hexadecimal and Decimal number\*/ |
|  | int HexToDec(char\*hex); |
|  | int IsHex(char\*ckstr); |
|  | int StrToDec(char\* str); |