|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Мытищинский филиал**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет имени Н.Э. Баумана**  **(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Космический

КАФЕДРА «Прикладная математика, информатика и вычислительная техника

» К3-МФ

**Лабораторная работа**

*ПО ДИСЦИПЛИНЕ:*

***Системное Программное Обеспечение***

***НА ТЕМУ:***

Абсолютный ассемблер

Студент К3-43Б Ярославцев Егор Викторович

(Группа) (Подпись, дата) (И.О.Фамилия)

Преподаватель

\_ Чернышов Александр Викторович

(Подпись, дата) (И.О.Фамилия)

*2021 г*

Код программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "metTrans.h"

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

int adrBegin = 0;

int tekAdr = 0;

int main() {

//char strSKomandami[] = "begin end db dw rb rw add addx mul mov selin div sub suc int cmp je jne jg jge ldx jnl lda sta hlt call ret and or xor";

//zapolnitHashTabl(strSKomandami);

obnulSpisMetok();

FILE\* in;

FILE\* out;

in = fopen("asm.txt", "r");

out = fopen("lst.txt", "w");

char text[200];

if (in != NULL) {

while (fgets(text, sizeof(text), in) != NULL) {

parseStrForLst(text, out);

}

}

fclose(in);

fclose(out);

if (provNaOprMetok() < 0) {

deleteSpisMetok();

exit(6);

}

in = fopen("asm.txt", "r");

out = fopen("obj.txt", "w");

if (in != NULL) {

while (fgets(text, sizeof(text), in) != NULL) {

parseStrForObj(text, out);

}

}

fclose(in);

fclose(out);

deleteSpisMetok();

}

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Hash.h"

#include "metTrans.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//#define countStrok 89

extern const int countStrok = 89;

typedef struct Knot knot;

typedef struct Knot\* knotptr;

int elementovVMassive = 0;

union key {

char key[5];

int hashKey;

};

struct Knot {

char\* keyStr;

knotptr next;

};

int hashFunc(char\* key) {

int h = 0;

union key k1;

int j = 0;

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

k1.key[i] = ' ';

}

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

if (k1.key[j] == ' ') {

k1.key[j] = key[i];

}

else {

k1.key[j] = ((int)key[i] + (int)k1.key[j]) % 256;

}

j++;

if (j >= 4) j = 0;

}

h = k1.hashKey;

if (h < 0) h = -h;

h = h % countStrok;

return h;

}

void dobavitSlovoVMassiv(int hash, char\* key, knotptr spis) {

if (spis[hash].keyStr == NULL) {

char\* tempCh = (char\*)malloc(sizeof(char) \* (strlen(key) + 1));

strcpy(tempCh, key);

tempCh[strlen(key)] = '\0';

spis[hash].keyStr = tempCh;

spis[hash].next = NULL;

}

else {

struct Knot\* temp = (knotptr)malloc(sizeof(struct Knot) \* 1);

char\* tempCh = (char\*)malloc(sizeof(char) \* (strlen(key) + 1));

strcpy(tempCh, key);

tempCh[strlen(key)] = '\0';

temp->keyStr = tempCh;

temp->next = NULL;

if (spis[hash].next == NULL) {

spis[hash].next = temp;

}

else {

knotptr poisk = spis[hash].next;

while (poisk->next != NULL) {

poisk = poisk->next;

}

poisk->next = temp;

}

}

}

void zapolnenie(char\* vseKomandi, knotptr spis) {

char\* komanda = NULL;

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

int mem = i;

int len = 0;

while (vseKomandi[i] != ' ') {

if (i + 1 >= strlen(vseKomandi)) {

len++;

break;

}

i++;

len++;

}

i = mem;

komanda = (char\*)malloc(sizeof(char) \* (len + 1));

komanda[len] = '\0';

for (int j = 0; j < len; j++) {

komanda[j] = vseKomandi[i];

i++;

}

dobavitSlovoVMassiv(hashFunc(komanda), komanda, spis);

free(komanda);

}

}

void obnulMass(knotptr spis) {

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

spis[i].keyStr = NULL;

spis[i].next = NULL;

}

}

void printKnot(knotptr nap, int hash, FILE\* out) {

fprintf(out, "%s %i\n", nap->keyStr, hash);

}

void printTabl(knotptr spis) {

FILE\* out;

out = fopen("hashTabl.txt", "w");

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

if (spis[i].keyStr != NULL) {

printKnot(spis + i, i, out);

knotptr next = spis[i].next;

while (next != NULL) {

printKnot(next, i, out);

next = next->next;

}

}

}

fclose(out);

}

void deleteMass(knotptr mas) {

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

if (mas[i].next != NULL) {

while (mas[i].next != NULL) {

knotptr poisk = mas + i;

while (poisk->next->next != NULL) {

poisk = poisk->next;

}

free(poisk->next->keyStr);

free(poisk->next);

poisk->next = NULL;

}

}

if (mas[i].keyStr != NULL) {

free(mas[i].keyStr);

}

}

}

int provVMas(const char\* key) {

FILE\* in;

in = fopen("hashTabl.txt", "r");

char buf[200];

int hash = 0;

if (in == NULL) return -2;

while (!feof(in)) {

fscanf(in, "%s", buf);

fscanf(in, "%i", &hash);

if (strcmp(buf, key) == 0) {

fclose(in);

return hash;

}

if (feof(in)) break;

}

fclose(in);

return -1;

}

void zapolnitHashTabl(char\* keys)

{

knot spis[89];

obnulMass(spis);

zapolnenie(keys, spis);

printTabl(spis);

deleteMass(spis);

}

#pragma once

#include <stdio.h>

typedef struct Knot knot;

typedef struct Knot\* knotptr;

union key;

struct Knot;

int hashFunc(char\* key); //функция нахождения хеша

void dobavitSlovoVMassiv(int hash, char\* key, knotptr spis); //добавления слова в массив (выявление коллизий)

void zapolnenie(char\* vseKomandi, knotptr spis); //разбиение на слова

void obnulMass(knotptr spis); //очищение массива

void printKnot(knotptr nap, int hash, FILE\* out); //печать одного элемента массива в файл

void printTabl(knotptr spis); //печать всего массива в файл

void deleteMass(knotptr mas); //удаление массива

int provVMas(const char\* key); //поиск по таблице хешей

void zapolnitHashTabl(char\* keys); //заполнение хеш-таблицы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Hash.h"

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <string.h>

#include <stdlib.h>

//#define countStrok 89;

extern const int countStrok;

int adr = 0;

int adrNach = 0;

int lenProgr = 0;

char strObj[3][7];

int iterForStrObj = 0;

struct tablMetok {

char\* name;

int opr;

int adr;

};

struct tablMetok spisMetok[89];

void obnulSpisMetok() {

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

spisMetok[i].name = NULL;

spisMetok[i].opr = 0;

spisMetok[i].adr = 0;

}

}

void deleteSpisMetok() {

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

if (spisMetok[i].name != NULL) {

free(spisMetok[i].name);

spisMetok[i].name = NULL;

}

}

}

int podchetBit() {

int ret = 0;

for (int i = 0; i < 3; i++) {

int it = 0;

while (it < 7) {

if (strObj[i][it] == '\0') break;

else{

if (strObj[i][it] == ' ') it++;

else {

ret++;

it += 2;

}

}

}

}

return ret;

}

char\* skompMasKod(int hash, int otnAdr, char\* slovo) {

char ret[7];

ret[6] = '\0';

char\* B[16] = { "0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F" };

int num1, num2;

num1 = hash / 16;

num2 = hash - 16 \* num1;

for (int i = 0; i < 16; i++) {

if (i == num1) {

sprintf(ret, "%s", B[i]);

}

if (i == num2) {

sprintf(ret + 1, "%s", B[i]);

}

}

if (otnAdr == 0) {

sprintf(ret + 2, "%x", adr);

}

else {

int temp = adr + 32768;

sprintf(ret + 2, "%x", temp);

}

return ret;

}

void addMet(char\* slovo, int adr) {

if (adr < 0) {

int i = 0;

while (i <= countStrok) {

if (spisMetok[i].name != NULL) {

if (strcmp(slovo, spisMetok[i].name) == 0) return;

i++;

}

else break;

}

char\* temp = (char\*)malloc(sizeof(char) \* strlen(slovo) + 1);

strcpy(temp, slovo);

temp[strlen(slovo)] = '\0';

spisMetok[i].name = temp;

spisMetok[i].adr = -1;

spisMetok[i].opr = 0;

}

else {

int i = 0;

while (i <= countStrok) {

if (spisMetok[i].name != NULL) {

if (strcmp(slovo, spisMetok[i].name) == 0) {

if (spisMetok[i].opr == 1) exit(12);

else {

spisMetok[i].opr = 1;

spisMetok[i].adr = adr;

return;

}

}

i++;

}

else break;

}

if (i <= countStrok) {

char\* temp = (char\*)malloc(sizeof(char) \* strlen(slovo) + 1);

strcpy(temp, slovo);

temp[strlen(slovo)] = '\0';

spisMetok[i].name = temp;

spisMetok[i].adr = adr;

spisMetok[i].opr = 1;

}

else exit(13);

}

}

void printObjStr(FILE\* out, char\* tip) {

if (!strcmp(tip, "00")) {

int pods = podchetBit();

if(pods < 16) fprintf(out, "0%x %s", pods, tip);

else fprintf(out, "%x %s", pods, tip);

for (int i = 0; i < 3; i++) {

if (strObj[i][0] == '\0') break;

fprintf(out, " %s", strObj[i]);

}

for (int i = 0; i < 3; i++) {

strObj[i][0] = '\0';

}

fprintf(out, " XX\n");

iterForStrObj = 0;

}

if (!strcmp(tip, "01")) {

for (int i = 1; i < 3; i++) {

strObj[i][0] = '\0';

}

int pods = podchetBit();

if (pods < 16) fprintf(out, "0%x %s %s", pods, tip, strObj[0]);

else fprintf(out, "%x %s %s", pods, tip, strObj[0]);

for (int i = 0; i < 3; i++) {

strObj[i][0] = '\0';}

fprintf(out, " XX\n");

iterForStrObj = 0;}

if (!strcmp(tip, "02")) {

char temp[7];

strcpy(temp, strObj[2]);

strObj[2][0] = '\0';

int pods = podchetBit();

//if (pods < 16) fprintf(out, "0%x %s %s", pods, "00", strObj[0]);

//else fprintf(out, "%x %s %s", pods, "00", strObj[0]);

if (pods < 16) fprintf(out, "0%x %s", pods, "00");

else fprintf(out, "%x %s", pods, "00");

for (int i = 0; i < 3; i++) {

if (strObj[i][0] == '\0') break;

fprintf(out, " %s", strObj[i]);

}

//fprintf(out, " %c", '\0');

fprintf(out, " XX\n");

for (int i = 0; i < 3; i++) {

strObj[i][0] = '\0';

}

strcpy(strObj[2], temp);

pods = podchetBit();

if (pods < 16) fprintf(out, "0%x %s %s", pods, tip, strObj[2]);

else fprintf(out, "%x %s %s", pods, tip, strObj[2]);

//fprintf(out, " %c", '\0');

fprintf(out, " XX\n");

iterForStrObj = 0;

}

}

int getInt(char\* slovo) {

char\* slovon = slovo;

int lenSlovo = 0;

int ret = 0;

while ((\*slovon != '\0') && (\*slovon != '\n')) {

lenSlovo++;

slovon++;

}

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

slovon = slovo;

slovon += lenSlovo - i - 1;

int step = pow(10, i);

ret += step \* ((\*slovon) - 48);

}

return ret;

}

int proverkaNaOperator(char\* slovo, int len) {

if (len > 3 || len < 2) return 0;

if (len == 2) {

if (slovo[0] == 'd' && slovo[1] == 'w') return 1;

if (slovo[0] == 'd' && slovo[1] == 'b') return 1;

if (slovo[0] == 'r' && slovo[1] == 'w') return 1;

if (slovo[0] == 'r' && slovo[1] == 'b') return 1;

}

else {

if (len == 3) {

if (slovo[0] == 'e' && slovo[1] == 'n' && slovo[2] == 'd') return 1;

}

else if(len == 5){

if (slovo[0] == 'b' && slovo[1] == 'e' && slovo[2] == 'g' && slovo[3] == 'i' && slovo[4] == 'n') return 1;

}

}

return 0;

}

void parseStrForLst(char\* str, FILE\* out) {

char slovo[20];

char\* sohr = str;

int lenSlova = 0;

int flagForHash = 0;

int hash = 0;

while (sscanf(str, "%s", slovo) != 0) {

if (str[0] == '\0' || str[0] == '\n') break;

lenSlova = strlen(slovo);

str += strlen(slovo);

if (str[0] == ' ') str++;

if (slovo[lenSlova - 1] == ',') slovo[lenSlova - 1] = '\0';

if (!strcmp(slovo, "begin")) {

sscanf(str, "%x", &adr);

return;

}

if (!strcmp(slovo, "end")) {

return;

}

if (slovo[lenSlova - 1] == ':') {

slovo[lenSlova - 1] = '\0';

addMet(slovo, adr);

continue;

}

if (slovo[0] == ';') {

break;

}

if (lenSlova == 1) continue;

if ((int)slovo[0] < 58 && (int)slovo[0] > 47) continue;

if (proverkaNaOperator(slovo, strlen(slovo))) {

continue;

}

if (provVMas(slovo) != -1) {

hash = provVMas(slovo);

flagForHash = 1;

continue;

}

addMet(slovo, -1);

}

if (flagForHash == 1) {

if(hash < 16) fprintf(out, "%x 0%x %s", adr, hash, sohr);

else fprintf(out, "%x %x %s", adr, hash, sohr);

flagForHash = 0;

}

else fprintf(out, "%x %s", adr, sohr);

lenProgr += 3;

adr += 3;

}

int parseStrForObj(char\* str, FILE\* out) {

char slovo[20];

int lenSlova = 0;

while (sscanf(str, "%s", slovo) != 0) {

if (str[0] == '\0' || str[0] == '\n') break;

lenSlova = strlen(slovo);

str += strlen(slovo);

if (str[0] == ' ') str++;

if (!strcmp(slovo, "begin")) {

sscanf(str, "%x", &adr);

adrNach = adr;

sprintf(strObj[0], "%x %x", adr, lenProgr);

printObjStr(out, "01");

return;

}

if (!strcmp(slovo, "end")) {

sprintf(strObj[2], "%x", adrNach);

printObjStr(out, "02");

return;

}

if (slovo[lenSlova - 1] == ':') {

continue;

}

if (slovo[0] == ';') {

break;

}

if (proverkaNaOperator(slovo, strlen(slovo))) {

if (!strcmp(slovo, "dw")) {

int temp = 0;

char temp2[7];

int sdvig = 0;

strObj[iterForStrObj][6] = '\n';

for (int i = 0; i < 6; i++) {

strObj[iterForStrObj][i] = '0';

}

strObj[iterForStrObj][6] = '\0';

for (int i = 0; i < 6; i++) {

temp2[i] = 'p';

}

temp2[6] = '\0';

temp = getInt(str);

sprintf(temp2, "%x", temp);

if (strlen(temp2) < 6) {

temp2[strlen(temp2)] = 'p';

}

for (int i = 0; i < 6; i++) {

if (temp2[i] == 'p') sdvig++;

}

int j = 0;

for (int i = 0; i < 6 - sdvig; i++) {

strObj[iterForStrObj][i + sdvig] = temp2[j];

j++;

}

iterForStrObj++;

if (iterForStrObj + 1 > 3) {

printObjStr(out, "00");

}

}

if (!strcmp(slovo, "rw")) {

int temp = 0;

char temp2[7];

int sdvig = 0;

int i = 5;

strObj[iterForStrObj][6] = '\n';

for (int i = 0; i < 6; i++) {

strObj[iterForStrObj][i] = '0';

}

strObj[iterForStrObj][6] = '\0';

//iterForStrObj++;///////////////////////////////////////

//break;

for (int i = 0; i < 6; i++) {

temp2[i] = 'p';

}

temp2[6] = '\0';

temp = getInt(str);

while (temp > 0) {

temp--;

strObj[iterForStrObj][i] = 'f';

strObj[iterForStrObj][--i] = 'f';

if (i == 0) {

iterForStrObj++;

if (iterForStrObj + 1 > 3) {

printObjStr(out, "00");

}

i = 5;

strObj[iterForStrObj][6] = '\n';

}

strObj[iterForStrObj][--i] = 'f';

strObj[iterForStrObj][--i] = 'f';

}

if (iterForStrObj + 1 > 3) {

printObjStr(out, "00");

}

}

continue;

}

if (provVMas(slovo) != -1) {

int hash;

int otnAdr = 0;

char slovo2[20];

slovo2[0] = '\0';

sscanf(str, "%s", slovo2);

if (slovo2[0] == '\0') {

goto label;

}

if (slovo2[strlen(slovo2) - 1] == ',') otnAdr = 1;

str += strlen(slovo2) + 1;

label:

hash = provVMas(slovo);

//sprintf(strObj[iterForStrObj], "%s", skompMasKod(hash, otnAdr, slovo));

strcpy(strObj[iterForStrObj], skompMasKod(hash, otnAdr, slovo));

iterForStrObj++;

if (iterForStrObj + 1 > 3) {

printObjStr(out, "00");

}

continue;

}

if (lenSlova == 1) break;

}

lenProgr += 3;

adr += 3;

}

int provNaOprMetok() {

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

if (spisMetok[i].name != NULL) {

if (spisMetok[i].opr == 0) {

return -1;

}

}

else return 1;

}

return 1;

}

#pragma once

#include <stdio.h>



struct tablMetok;

void obnulSpisMetok(); //обнуление массива меток

void deleteSpisMetok(); //удаление массива меток

void addMet(char\* slovo, int adr); //

int proverkaNaOperator(char\* slovo, int len); //

void parseStrForLst(char\* str, FILE\* out); //

int parseStrForObj(char\* str, FILE\* out); //

int provNaOprMetok(); //

**Hash Tabl**

**Assembler code**

Объектный код

03 01 1000 21 xx

09 00 271012 0f1018 071015 xx

09 00 0b101b 18101e 56100f xx

09 00 000000 000003 000007 xx

06 00 00000f 00ffff xx

02 02 1000 xx

03 – кол-во байт на строку

*01 – старт*

*00 – все остальные*

*02 – энд*

1000 – адрес начала программы

21 – сколько заняла сама программа в 16ричной системе без учитывания begin end

Когда мы 2 раз обращаемся по метке add, используем dw 15





Сложение 2ух элементов массива

После выполнения листинга фиксируется метки в массиве

Когда я делаю листинг каждой строке присваивается собственный адрес и заполняется массив меток, в котором запоминаются все метки и их адреса.