|  |  |
| --- | --- |
|  | **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 Assembler\_\_\_\_­­­\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент \_\_К3-43Б\_\_  **\_\_\_\_\_\_\_\_\_\_** Цветков Юрий Алексеевич

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

#### Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_ Чернышов Александр Викторович

(Подпись, дата) (И.О.Фамилия)

*2021 г.*

**Задание**

Написать программу абсолютного ассемблера для гипотетической вычислительной машины.

**Код программы**

**Pars\_str.h**

#ifndef **SPO\_5\_PARSING\_STR\_H**#define **SPO\_5\_PARSING\_STR\_H**#include **<stdio.h>**#include **<stdlib.h>**#include **<string.h>**#include **<ctype.h>**#define **MAXLEN** 256  
  
**const char**\* dir[] = {**"BEGIN"**, **"END"**, **"DB"**, **"DW"**, **"RB"**, **"RW"**, **"LDX"**, **"LDA"**, **"ADD"**, **"STA"**, **"HLT"**};  
  
**struct** Str  
{  
 **char** marker[**MAXLEN**/4];  
 **char operator**[**MAXLEN**/4];  
 **char** operand[**MAXLEN**/4];  
 **char** comment[**MAXLEN**/4];  
};  
**int** file\_line(**char**\* argv)  
{  
 FILE\* in = fopen(argv, **"r"**);  
  
 **int** i = 1;  
 **while** (!feof(in))  
 **if**(fgetc(in) == **'\n'**)  
 i++;  
  
 fclose(in);  
 **return** i;  
}  
  
**void** clear\_str(**struct** Str \*res, **int** i)  
{  
 **for**(**int** j = 0; j < i; j++)  
 {  
 memset(res[j].marker, 0, strlen(res[j].marker));  
 memset(res[j].operand, 0, strlen(res[j].operand));  
 memset(res[j].**operator**, 0, strlen(res[j].**operator**));  
 memset(res[j].comment, 0, strlen(res[j].comment));  
 }  
}  
  
**char**\* correction(**char**\* str)  
{  
 **char**\* temp = malloc(strlen(str));  
 memset(temp, 0, strlen(temp));  
  
 **for**(**int** i = 0; \*str != **'\n'** && \*str != **'\0'**; str++, i++)  
 {  
 **if** (\*str == **' '** || \*str == **'\t'**)  
 **continue**;  
 temp[i] = \*str;  
 }  
 **return** temp;  
}  
  
**void** ParseStr(**char** \*str, **struct** Str \*res, **int** s)  
{  
 **if**(strchr(str, **'-'**) == str)  
 {  
 **char** temp[strlen(str)];  
 memset(temp, 0, strlen(temp));  
 str++;  
  
 **for**(**int** i = 0; \*str != **' '** && \*str != **'\t'** && \*str != **'\0'** ; str++, i++)  
 temp[i] = \*str;  
 strcpy(res[s].marker, correction(temp));  
 }  
 **while**(\*str ==**' '** || \*str == **'\t'**) str++;  
 **if**(strchr(str, **';'**) != **NULL**)  
 {  
 strcpy(res[s].comment, correction(strchr(str, **';'**) + 1));  
 memset(strchr(str, **';'**), 0, strlen(strchr(str, **';'**)));  
 }  
 **while**(\*str == **' '** || \*str == **'\t'**) str++;  
 **for**(**int** i = 0; i < 11; i++)  
 {  
 **if** (strstr(str, dir[i]) != **NULL**)  
 {  
 strcpy(res[s].**operator**, dir[i]);  
 str += strlen(dir[i]) + 1;  
 **break**;  
 }  
 }  
 **while**(\*str == **' '** || \*str == **'\t'**)str++;  
 strcpy(res[s].operand, correction(str));  
}  
  
**void** filling\_str(**char**\* argv, **struct** Str \*res)  
{  
 FILE\* in = fopen(argv, **"r"**);  
 **char** str[**MAXLEN**];  
  
 **int** i = 0;  
 **while** (fgets(str, **MAXLEN**, in) != **NULL**)  
 {  
 ParseStr(str, res, i);  
*// printf("Marker:%8s Operator:%8s Operand:%8s Comment:%8s\n",  
 // res[i].marker, res[i].operator, res[i].operand, res[i].comment);* i++;  
 }  
 fclose(in);  
}  
  
**int** len\_mark(**struct** Str \*pars, **int** s)  
{  
 **int** j = 0;  
 **for**(**int** i = 0; i < s; i++)  
 {  
 **if**(pars[i].marker[0] != **'\0'**)  
 j++;  
 }  
 **return** j;  
}  
  
**int** address(**struct** Str \*res)  
{  
 **if** (strcmp(res[0].**operator**, dir[0]) == 0)  
 {  
 **if**(res[0].operand[0] == **'\0'**)  
 **return** 0;  
 **else  
 return** strtol(res[0].operand, **NULL**, 16);  
 }  
 **else** exit(0);  
}  
#endif *//SPO\_5\_PARSING\_STR\_H*

Hash.h

#ifndef **SPO\_5\_HASH\_H**#define **SPO\_5\_HASH\_H**#include **<stdio.h>**#include **<stdlib.h>**#include **<string.h>**#include **<math.h>**#define **M** 29  
#define **A** 0.6180339887  
  
**struct** Hash;  
**typedef struct** Hash\* pHash;  
**typedef struct** Hash\*\* ppHash;  
  
**struct** Hash  
{  
 **int** mc;  
 **char**\* elem;  
 pHash next;  
};  
  
**int** HashMulti(**char**\* key)  
{  
 **unsigned int** K = 0, w = strlen(key);  
 **for**(**int** i = 0; i < w; i++)  
 K ^= (**unsigned int**) key[i];  
  
 **return** (**int**)(**M** \* fmod(**A** / w \* K, 1));  
}  
  
pHash find(ppHash list, **char**\* elem)  
{  
 **for**(pHash hash = list[HashMulti(elem)]; hash != **NULL**; hash = hash -> next)  
 **if**(strcmp(elem, hash -> elem) == 0)  
 **return** hash;  
  
 **return NULL**;  
}  
  
**void** add(ppHash list, **char**\* elem, **int** mc)  
{  
 **int** k = HashMulti(elem);  
 pHash hash = list[k];  
  
 **if**(hash == **NULL**)  
 {  
 list[k] = (pHash)malloc(**sizeof**(**struct** Hash));  
 list[k] -> elem = (**char**\*)malloc(strlen(elem));  
 list[k] -> next = **NULL**;  
 list[k] -> mc = mc;  
 strcpy(list[k] -> elem, elem);  
 }  
 **else** {  
 **while** (hash -> next != **NULL**) hash = hash -> next;  
 hash = hash -> next = (pHash)malloc(**sizeof**(**struct** Hash));  
 hash -> elem = (**char**\*)malloc(strlen(elem));  
 hash -> next = **NULL**;  
 hash -> mc = mc;  
 strcpy(hash -> elem, elem);  
 }  
}  
**void** clear\_hash(ppHash list, **int** s)  
{  
 **for**(**int** i = 0; i < s; i++)  
 list[i] = **NULL**;  
}  
**void** PrintToFile(ppHash list, FILE\* out)  
{  
 **for**(**int** i = 0; i < **M**; i++)  
 {  
 fprintf(out,**"%2i: "**, i);  
 **for**(pHash hash = list[i]; hash != **NULL**; hash = hash -> next)  
 fprintf(out, **"%10s -> "**, hash -> elem);  
  
 fprintf(out, **"%10s\n"**, **"NONE"**);  
 }  
}  
  
**void** Print(ppHash list)  
{  
 **for**(**int** i = 0; i < **M**; i++)  
 {  
 printf(**"%2i: "**, i);  
 **for**(pHash hash = list[i]; hash != **NULL**; hash = hash -> next)  
 printf( **"%10s -> "**, hash -> elem);  
  
 printf( **"%10s\n"**, **"NONE"**);  
 }  
}  
  
**void** Print\_mc(ppHash list)  
{  
 **for**(**int** i = 0; i < **M**; i++)  
 {  
 printf(**"%2i: "**, i);  
 **for**(pHash hash = list[i]; hash != **NULL**; hash = hash -> next)  
 printf( **"%10i -> "**, hash -> mc);  
  
 printf( **"%10s\n"**, **"NONE"**);  
 }  
}  
#endif *//SPO\_5\_HASH\_H*

Marker.h

#ifndef **SPO\_5\_MARKER\_H**#define **SPO\_5\_MARKER\_H**#define **STEP** 3  
#include **"hash.h"  
struct** marker\_table  
{  
 **int** sign;  
 **int** address;  
 **char** mark[**MAXLEN**/4];  
  
};  
**void** clear\_mark(**struct** marker\_table \*table, **int** i)  
{  
 **for**(**int** j = 0; j < i; j++)  
 {  
 table[j].sign = 0;  
 table[j].address = 0;  
 memset(table[j].mark, 0, strlen(table[j].mark));  
 }  
}  
  
**void** check\_table(**struct** marker\_table \*mark, **int** len\_m)  
{  
 **for**(**int** i = 0; i < len\_m; i++)  
 {  
 **if**(mark[i].sign == 0)  
 exit(0);  
 }  
}  
**int** find\_address(**struct** marker\_table\* table, **int** len\_m, **char**\* mark)  
{  
 **for**(**int** i = 0; i < len\_m; i++)  
 {  
 **if**(strcmp(table[i].mark, mark) == 0)  
 **return** i;  
 }  
 **return** -1;  
}  
#endif *//SPO\_5\_MARKER\_H*

Main.cpp

#include **"parsing\_str.h"**#include **"marker.h"**#include **"hash.h"  
char**\* direct[] = {**"BEGIN"**, **"END"**, **"DB"**, **"DW"**, **"RB"**, **"RW"**};  
**char**\* commands[] = {**"LDX"**, **"LDA"**, **"ADD"**, **"STA"**, **"HLT"**};  
  
**int** check\_num(**const char**\* str)  
{  
 **for** (**int** i = 0; str[i] != **'\0'**; i++)  
 **if** ((str[i] >= **'0'** && str[i] <= **'9'**) || str[i] == **'h'**)  
 **continue**;  
 **else return** -1;  
 **return** 1;  
}  
  
**char**\* make\_address(**int** mc, **int** adr)  
{  
 **char**\* res = (**char**\*)malloc(6);  
 sprintf(res, **"%2x%4x"**, mc, adr);  
 **for**(**int** i = 0; i < 6; i++) **if**(res[i] == **' '**) res[i] = **'0'**;  
  
 **return** res;  
}  
  
**void** first\_check(**struct** Str\* pars, **int** pars\_len, **struct** marker\_table \*table, **int** len\_m, **int** begin\_address, ppHash list)  
{  
 **int** j = 0;  
 **for**(**int** i = 0; i < pars\_len; i++)  
 {  
 **if** (pars[i].marker[0] != **'\0'**)  
 {  
 **if** (find\_address(table, len\_m, pars[i].marker) != -1)  
 exit(0);  
  
 strcpy(table[j].mark, pars[i].marker);  
 table[j].address = begin\_address + (i-1) \* **STEP**;  
 j++;  
 }  
 }  
  
 **for**(**int** i = 0; i < pars\_len; i++)  
 {  
 printf(**"%s %x\n"**, table[i].mark, table[i].address);  
 **if** (find(list, pars[i].operator) == **NULL**)  
 exit(0);  
  
 **if**(pars[i].operand[0] != **'\0'**)  
 **if** (check\_num(pars[i].operand) == -1)  
 {  
  
 **int** fa = find\_address(table, len\_m, pars[i].operand);  
 **if** (fa != -1)  
 table[fa].sign = 1;  
 **else** exit(0);  
 }  
 }  
}  
**void** create\_table(**char**\* argv1, **char**\* argv2, **struct** Str\* pars, **int** pars\_len, **struct** marker\_table\* table, **int** m\_len, ppHash list, **int** begin)  
{  
 FILE\* in = fopen(argv1, **"r"**);  
 FILE\* out = fopen(argv2, **"w"**);  
  
 fprintf(out, **"%3s %6s %6s %10s\n"**, **"num"**, **"adr"**, **"code"**, **"str"**);  
 **for**(**int** i = 0; i < pars\_len; i++)  
 {  
 **char** str[**MAXLEN**];  
 fgets((**char** \*) str, **MAXLEN**, in);  
 pHash hash = find(list, pars[i].operator);  
 **int** s = find\_address(table, m\_len, pars[i].operand);  
 **if**(i>0)  
 fprintf(out, **"%3i %6x %s %s"**, i+1, begin + (i-1) \* **STEP** ,make\_address(hash->mc, table[s].address), str);  
 **else** fprintf(out, **"%3i %6x %s %s"**, i+1, 0, make\_address(hash->mc, table[s].address), str);  
 }  
 fclose(out);  
}  
**void** create\_obj\_file(**struct** Str\* pars, **int** pars\_len, **struct** marker\_table\* table, **int** m\_len, ppHash list, **int** begin)  
{  
 pHash hash = find(list, pars[0].operator);  
 **int** s = find\_address(table, m\_len, pars[0].operand);  
 printf(**"H\_%s\_0000%x\nT\_"**, make\_address(hash->mc, table[s].address), (pars\_len-2) \* **STEP**);  
 **for**(**int** i = 1; i < pars\_len - 1; i++)  
 {  
 hash = find(list, pars[i].operator);  
 s = find\_address(table, m\_len, pars[i].operand);  
 printf(**"%s "**, make\_address(hash->mc, table[s].address));  
 }  
 hash = find(list, pars[pars\_len-1].operator);  
 s = find\_address(table, m\_len, pars[pars\_len-1].operand);  
 printf(**"\nE\_%s"**, make\_address(hash->mc, table[s].address));  
}  
  
**int** main(**int** argc, **char**\* argv[])  
{  
 *//====================================================pars str====================================================//* **int** len\_asm = file\_line(argv[1]);  
 **struct** Str pars[len\_asm];  
 clear\_str(pars, len\_asm);  
 filling\_str(argv[1], pars);  
 *//==================================================init hash\_table===============================================//* pHash list[**M**];  
 clear\_hash(list, **M**);  
  
 **char** \*temp = direct[0];  
 **for**(**int** i = 0; temp != **NULL**; i++, temp = direct[i])  
 add(list, temp, 0);  
  
 temp = commands[0];  
 **for**(**int** i = 0; temp != **NULL**; i++, temp = commands[i])  
 add(list, temp, rand() % **MAXLEN**);  
  
 *//====================================================first pass==================================================//* **int** len\_m = len\_mark(pars, len\_asm);  
 **struct** marker\_table table[len\_m];  
 clear\_mark(table, len\_m);  
  
 **int** begin\_address = address(pars);  
  
 first\_check(pars, len\_asm, table, len\_m, begin\_address, list);  
 check\_table(table, len\_m);  
  
 *//===================================================second pass==================================================//* create\_table(argv[1], argv[2], pars, len\_asm, table, len\_m, list, begin\_address);  
 create\_obj\_file(pars, len\_asm, table, len\_m, list, begin\_address);  
 **return** 0;  
}

**Исходная код на Assembler**

BEGIN 1000h  
-start LDX zero  
 LDA lel  
 ADD wsz  
 STA rez  
 HLT  
-zero DW 0  
-wsz DW 3  
-lel DW 7  
-rez RW 1  
 END start

**Результат**

num adr code str

1 0 001000 BEGIN 1000h

2 1000 29100f -start LDX zero

3 1003 231015 LDA lel

4 1006 be1012 ADD wsz

5 1009 841018 STA rez

6 100c e10000 HLT

7 100f 001012 -zero DW 0

8 1012 001015 -wsz DW 3

9 1015 001018 -lel DW 7

10 1018 000001 -rez RW 1

11 101b 001000 END start

**Объектный файл**

H\_001000\_00001b

T\_29100f\_231015\_be1012\_841018\_e10000\_001012\_001015\_001018\_000001

E\_001000

**Примечание**

Формирование машинного кода для директив RB, RW, DB, DW происходит по следующему алгоритму:

Для DB, DW в старшем байте записываются нули, в двух других адрес следующей строки.

Для RB, RW во всех байтах записываются нули.