

МИНОБРНАУКИ РОССИИ
ФГБОУ ВО «Ижевский государственный технический университет имени
М. Т. Калашникова»
Кафедра «Программное обеспечение»

Отчет по лабораторной работе №2
по дисциплине «Машинно-зависимые языки программирования»
на тему «Трансляция потока команд – построение листинга»

Выполнил:
студент группы Б04-191-3

Гумметов Р.А.

Принял:
к.т.н., профессор

Тарасов В.Г.

Ижевск 2020

1. Цель работы

Изучение структур данных и алгоритмов, применяемых при проектировании транслятора с языка символического кодирования в машинный код.

2. Задание

Исходные данные содержат команды вида:

МНЕМА r,r
 МНЕМА r,m
 МНЕМА m,r
 ...

Здесь МНЕМА = [sub, cmp].

Операнд в памяти (m) может быть задан одним из следующих способов: [BX], [SI], [DI], [BX+SI], [BX+DI].

Требуется сформировать листинг для этого потока команд для случая 16-разрядных операндов.

3. Шаблоны машинных команд

add:

001010dw	mod(2) reg(3) r/m(3)
----------	----------------------

cmp (рег/память с регистром):

0011101w	mod(2) reg(3) r/m(3)
----------	----------------------

cmp (регистр с рег/памятью):

0011100w	mod(2) reg(3) r/m(3)
----------	----------------------

По условию задачи на вход не подаются команды, содержащие непосредственный операнд, поэтому другие шаблоны машинных команд не используются.

В случае операндов reg, reg поле mod = 11, поле r/m используется для регистра. В случае reg, mem или mem, reg поле mod = 00 (так как DISP не обрабатывается) и поле r/m используется для операнда в памяти.

4. Описание программы

- 1) Команды считываются из файла.
- 2) Каждая команда разделяется на строку с названием команды (sub, cmp) и на строки, содержащие ее операнды. Заранее предполагается, что между названием и операндами и между операндом и операндом всегда один пробел.
- 3) Далее проводится анализ операндов.

- a. В соответствии с именем операнда определяется местонахождение данного операнда – в памяти или в регистре.
 - b. Для операнда в памяти определяется способ его задания: [BX], [SI], [DI], [BX+SI] или [BX+DI]
 - c. В случае с операндом в регистре необходимо запомнить номер регистра в таблице регистров `regt` и определить значение бита `w` в соответствии с данным номером.
- 4) В соответствии с проведенным анализом синтезируется машинная команда.
 - a. `mod = 11`, если операнды в регистрах; `mod = 00` – в памяти.
 - b. Поле `reg` определяется по номеру регистра в таблице.
 - c. Поле `r/m` определяется либо по номеру второго регистра, либо, если операнд в памяти, по номеру алгоритма, с помощью которого вычисляется способ задания операнда в памяти.
- 5) Машинная команда выводится на экран в шестнадцатеричном виде.

5. Текст программы

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <fstream>
#include <iomanip>
#define MEM_T 5
#define OPCODE 2
using namespace std;

bool reg[2], mem[2], bx[2], bx_si[2], bx_di[2], si[2], di[2], w, d;
int regs[2];

void clear()
{
    for (size_t i = 0; i < 2; i++)
    {
        reg[i] = false; mem[i] = false; bx[i] = false; bx_si[i] = false;
        bx_di[i] = false; si[i] = false; di[i] = false;
    }
    w = false;
    d = false;
}

char t[4] = "000";
void i2b(char x)
{
    char m = 1;
    for (size_t i = 0; i < 3; i++)
    {
        if (x & m) t[2 - i] = '1';
        else t[2 - i] = '0';
        m = m << 1;
    }
}
```

```

char regt[16][3] = { "al", "cl", "dl", "bl", "ah", "ch", "dh", "bh", "ax", "cx", "dx",
"bx", "sp", "bp", "si", "di" };

struct opcode_t
{
    string name;
    string code;
};

opcode_t op[OPCODE] =
{
    {"sub", "001010"},
    {"cmp", "001110"}
};

struct mem_t
{
    bool bx;
    bool bx_si;
    bool bx_di;
    bool si;
    bool di;
    string r_m;
};

//Таблица для определения способа задания операнда в памяти
mem_t memt[MEM_T] =
{
    {true, false, false, false, false, "111"},
    {true, true, false, false, false, "000"},
    {true, false, true, false, false, "001"},
    {false, false, true, false, false, "100"},
    {false, false, false, false, true, "101"}
};

void both_reg(string& bin)
{
    bin += "11";           //Значение поля mod
    i2b(regs[0]);
    bin += t;              //reg
    i2b(regs[1]);
    bin += t;              //r_m
}

void reg_mem(string& bin)
{
    bin += "00";           //Значение поля mod
    size_t m, r;
    if (d)
    {
        m = 1;
        r = 0;
    }
    else
    {
        m = 0;
        r = 1;
    }
    i2b(regs[r]);
    bin += t;              //Поле reg

    for (size_t i = 0; i < MEM_T; i++)
    {
        if (memt[i].bx == bx[m] && memt[i].bx_di == bx_di[m] && memt[i].bx_si ==
bx_si[m]

```

```

        && memt[i].di == di[m] && memt[i].si == si[m])
    {
        bin += memt[i].r_m;          //Поле r_m
    }
}

void operand(int n, string& operand)
{
    n--;
    if (operand.find('[') != -1)      //Операнд в памяти
    {
        mem[n] = 1;
        if (operand.find("bx") != -1)
        {
            bx[n] = 1;
            if (operand.find("bx + si") != -1)
                bx_si[n] = 1;
            else if (operand.find("bx + di") != -1)
                bx_di[n] = 1;
        }
        else if (operand.find("si") != -1)
            si[n] = 1;
        else if (operand.find("di") != -1)
            di[n] = 1;
    }
    else                               //Операнд в регистре
    {
        reg[n] = 1;
        for (size_t i = 0; i < 16; i++)
            if (operand == regt[i])
                regs[n] = i;
        w = (regs[n] >> 3);
    }
}

void get_command(string &bin, string &name)
{
    for (size_t i = 0; i < OPCODE; i++)      //Определение кода операции
    {
        if (op[i].name == name)
            bin = op[i].code;
    }
    if (reg[0] && reg[1] || reg[0] && mem[1])
        d = true;
    else
        d = false;
    bin += to_string((int)d);
    bin += to_string((int)w);

    if (reg[0] && reg[1])
        both_reg(bin);                      //Оба операнда в регистрах
    else if (reg[0] && mem[1] || reg[1] && mem[0])
        reg_mem(bin);                       //Один в регистре, другой в памяти
}

int main()
{
    setlocale(LC_ALL, "Russian");
    ifstream rf("c.txt");
    string cmd;          //Команда на языке ассемблера
    string bin_str;      //Машинная команда
    long int longint;
    cout << setbase(16);

```

```

while (getline(rf, cmd))
{
    string name;           //Имя команды
    string operand_name;   //Имя операнда
    int j = 0, start_operand;
    for (; j < cmd.size(); j++)
    {
        if (cmd[j] == ' ')
        {
            name = cmd.substr(0, j);
            j++;
            start_operand = j;
            break;
        }
    }

    clear();
    for (; j < cmd.size(); j++)
    {
        if (cmd[j] == ',')
        {
            operand_name = cmd.substr(start_operand, j - start_operand);
            j+=2;
            start_operand = j;
            operand(1, operand_name); //Анализ операнда 1
        }
        else if (j == cmd.size() - 1)
        {
            operand_name = cmd.substr(start_operand, j - start_operand +
1);
            operand(2, operand_name); //Анализ операнда 2
        }
    }

    cout << "Исходная команда: " << cmd << endl;

    get_command(bin_str, name);           //Синтез машинной команды

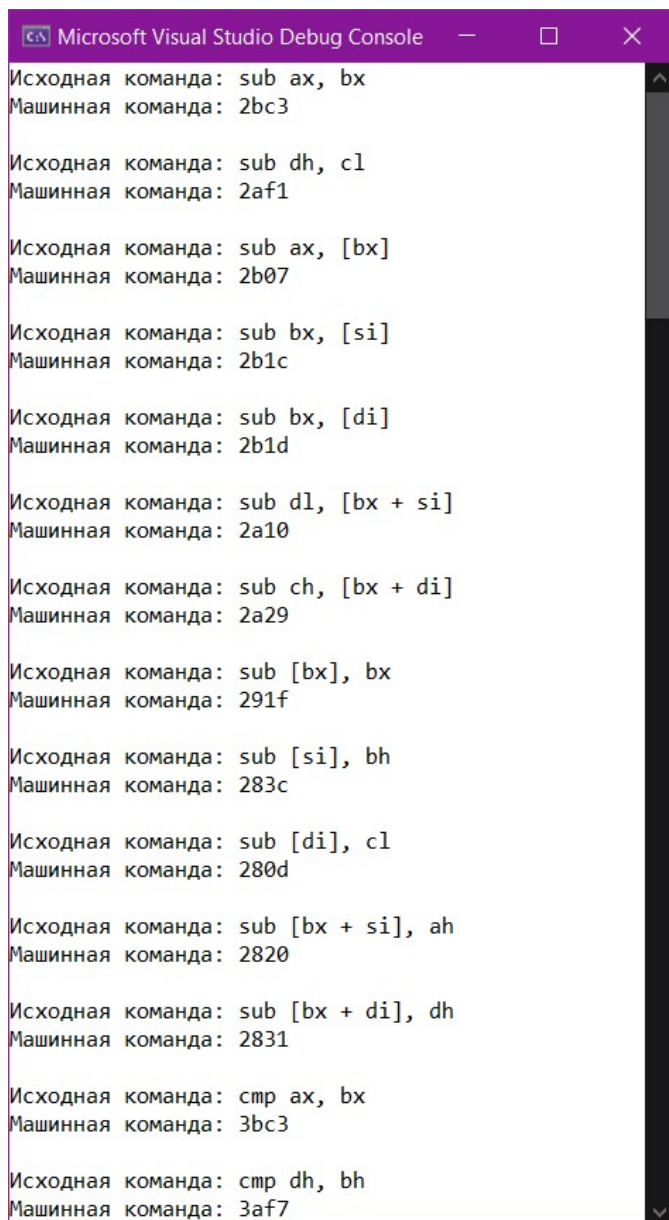
    longint = 0; //Перевод строки в шестнадцатеричное представление
    int len = bin_str.size();
    for (int i = 0; i < len; i++)
        longint += (bin_str[len - i - 1] - 48) * pow(2, i);

    cout << "Машинная команда: " << longint << endl << endl;
}

rf.close();
return 0;
}

```

6. Результаты работы программы



```

Microsoft Visual Studio Debug Console
Исходная команда: sub ax, bx
Машинная команда: 2bc3

Исходная команда: sub dh, cl
Машинная команда: 2af1

Исходная команда: sub ax, [bx]
Машинная команда: 2b07

Исходная команда: sub bx, [si]
Машинная команда: 2b1c

Исходная команда: sub bx, [di]
Машинная команда: 2b1d

Исходная команда: sub dl, [bx + si]
Машинная команда: 2a10

Исходная команда: sub ch, [bx + di]
Машинная команда: 2a29

Исходная команда: sub [bx], bx
Машинная команда: 291f

Исходная команда: sub [si], bh
Машинная команда: 283c

Исходная команда: sub [di], cl
Машинная команда: 280d

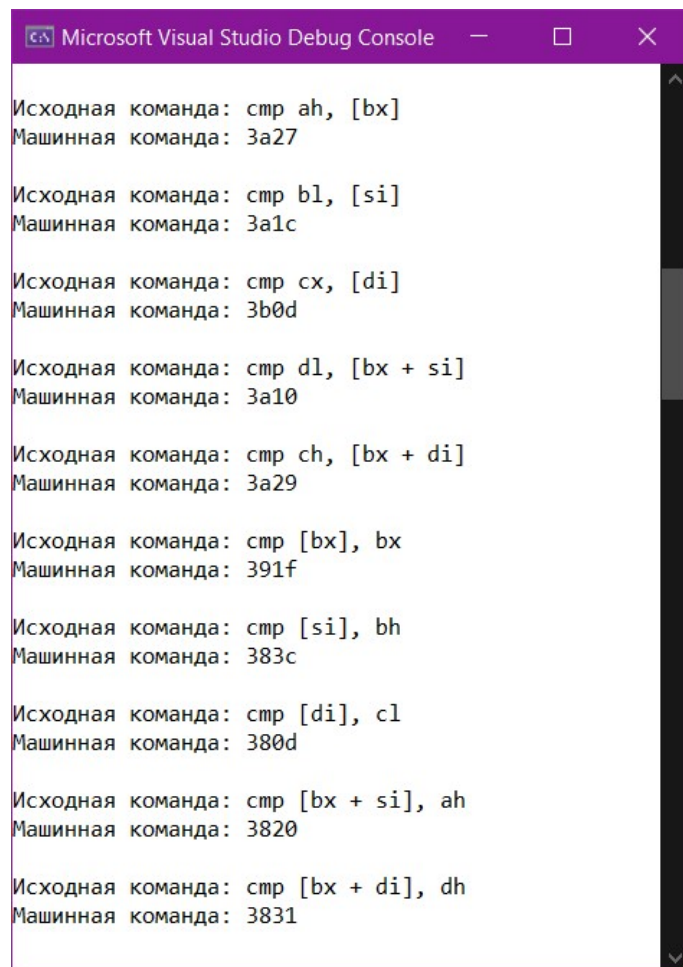
Исходная команда: sub [bx + si], ah
Машинная команда: 2820

Исходная команда: sub [bx + di], dh
Машинная команда: 2831

Исходная команда: cmp ax, bx
Машинная команда: 3bc3

Исходная команда: cmp dh, bh
Машинная команда: 3af7
  
```

Рис. 1. Результаты работы программы.



```

Microsoft Visual Studio Debug Console
Исходная команда: cmp ah, [bx]
Машинная команда: 3a27

Исходная команда: cmp bl, [si]
Машинная команда: 3a1c

Исходная команда: cmp cx, [di]
Машинная команда: 3b0d

Исходная команда: cmp dl, [bx + si]
Машинная команда: 3a10

Исходная команда: cmp ch, [bx + di]
Машинная команда: 3a29

Исходная команда: cmp [bx], bx
Машинная команда: 391f

Исходная команда: cmp [si], bh
Машинная команда: 383c

Исходная команда: cmp [di], cl
Машинная команда: 380d

Исходная команда: cmp [bx + si], ah
Машинная команда: 3820

Исходная команда: cmp [bx + di], dh
Машинная команда: 3831
  
```

Рис. 2. Результаты работы программы (2)

Для сравнения на рис. 3 приведен листинг стандартного ассемблера (опущено несколько команд, в которых регистры заполняются начальными значениями).

```

Turbo Assembler      Version 3.2      06/19/20 13:36:28      Page 1
a.asm
1      0000                      cseg segment
2                                assume cs:cseg
3      0000                      start:
...
8
9      000A  2B C3                sub ax, bx
10     000C  2A F1                sub dh, cl
11
  
```

```

12      000E  2B 07      sub ax, [bx]
13      0010  2B 1C      sub bx, [si]
14      0012  2B 1D      sub bx, [di]
15      0014  2A 10      sub dl, [bx + si]
16      0016  2A 29      sub ch, [bx + di]
17
18      0018  29 1F      sub [bx], bx
19      001A  28 3C      sub [si], bh
20      001C  28 0D      sub [di], cl
21      001E  28 20      sub [bx + si], ah
22      0020  28 31      sub [bx + di], dh
23
24      0022  3B C3      cmp ax, bx
25      0024  3A F7      cmp dh, bh
26
27      0026  3A 27      cmp ah, [bx]
28      0028  3A 1C      cmp bl, [si]
29      002A  3B 0D      cmp cx, [di]
30      002C  3A 10      cmp dl, [bx + si]
31      002E  3A 29      cmp ch, [bx + di]
32
33      0030  39 1F      cmp [bx], bx
34      0032  38 3C      cmp [si], bh
35      0034  38 0D      cmp [di], cl
36      0036  38 20      cmp [bx + si], ah
37      0038  38 31      cmp [bx + di], dh
38
39      003A  B8 4C00     mov ax, 04C00h
40      003D  CD 21      int 21h
41      003F                      cseg ends
42                                end start

```

Рис. 3. Листинг кода в tasm

7. Вывод

В данной лабораторной работе были изучены основные структуры данных и алгоритмы, применяемые при проектировании транслятора с языка символического кодирования в машинный код. На вход подавался набор команд на языке ассемблера, ограниченный определенными условиями: команды `sub`, `cmp` с операндами, находящимися в регистрах и в оперативной памяти.