МИНОБРНАУКИ РФ

ФГБОУ ВО «ИжГТУ имени М.Т. Калашникова»

Кафедра «Программное обеспечение»

Отчет

по лабораторной работе № 3 по дисциплине

на тему "Проектирование Ассемблера с обработкой именованных директив определения данных"

"Машинно-зависимые языки программирования"

Вариант 5

Выполнил студент группы Б04-191-3: Р.А. Гумметов

Принял: В.Г. Тарасов

Ижевск 2020

1. Постановка задачи

Задания связаны с проектированием ассемблеров. Исходными данными являются потоки ассемблерных команд и директив, записанные в соответствии с правилами языка Ассемблера. Результат обработки должен максимально совпадать с теми данными, которые формирует стандартный ассемблер (tasm, masm или ассемблер для ОС Linux). В текстах заданий для указания операндов используются следующие обозначения:

r – операнд находится в регистре общего назначения (РОН);

m – операнд находится в основной памяти;

imm – непосредственный операнд.

Исходные данные содержат команды вида:

МНЕМА r,r

МНЕМА r,m

МНЕМУ метка

…

ret

Здесь МНЕМА = [sub, cmp], МНЕМУ = [jne, je], “метка” - символическое обозначение какой-нибудь команды.

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

Требуется разработать ассемблер для формирования выполняемого файла .сом формата. Представить скриншоты проверочных запусков полученной программы под управлением отладчика (td.exe).

2. Ассемблер

Ассемблер - транслятор исходного текста программы, написанной на языке ассемблера, в программу на машинном языке.

Типы ассемблеров

По способу формирования кодов адресов операции:

* Типизированные (с каждым операндом ассоциируется тип (байт, слово и т.д.)- IBM PC. Тип операндов влияет на машинный код операции: AL - 1 байт, AX - 2 байта);
* Нетипизированные (машинный код операции определяется ассемблерной мнемоникой, а операнды не влияют на код, т.к. сами зависят от кода (IBM 370): movB - переслать байт, movW - переслать слово).

По количеству просмотров:

* однопросмотровые:
  + с записью в ОП и выполнением (схема 1A / ОП);
  + с записью результата на магнитный диск - создается объектный файл (схема 1A / МД);
* двухпросмотровые;
* многопросмотровые.

Функции ассемблера

1. Распознавание машинных команд и перевод их с языка программиста на язык машины (1-й просмотр).
2. Распознавание директив ассемблера:

* директивы управления ассемблированием (1-й просмотр);
* директивы описания данных (2-й просмотр).

1. Распределение памяти (минимально возможный объём машинной команды) (1-й просмотр).
2. Контроль синтаксиса (1-й просмотр).
3. Контроль семантики (1-й просмотр).

Требуется разработать ассемблер для формирования выполняемого файла .сом формата.

3. Файлы .com

Структура COM - файла проста. COM-файл — простой тип исполняемого файла, при выполнении которого данные, код и стек находятся в одном и том же 16-битном сегменте. В файлах данного типа, обычно не имеющими даже заголовка файла, содержатся только машинный код и данные программы.

Размер COM - файла ограничен 64 кб, т.е. размером одного сегмента памяти. COM файлы пишут в основном на языке Ассемблера, но они постепенно устаревают и на смену им приходят огромные по своим размерам и сложные по своей структуре EXE файлы.

.COM — один из простейших форматов исполняемых файлов для процессоров семейства x86. Программа, загруженная в память для исполнения, является точной копией файла на диске.

Запуск COM-программы в MS-DOS происходит следующим образом:

1. Система выделяет свободный сегмент памяти и заносит его адрес во все сегментные регистры (CS, DS, ES и SS).

2. В первые 256 байт этого сегмента записывается PSP

3. Непосредственно за ним загружается содержимое COM-файла без изменений

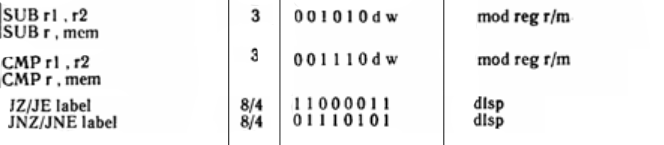
4. Указатель стека (регистр SP) устанавливается на конец сегмента

5. В стек записывается 0000h (адрес возврата для команды ret)

6. Управление передаётся по адресу CS:0100h, где находится первый байт исполняемого файла

7. Модель памяти, используемую COM-программами, когда код программы, все её данные, PSP и стек расположены в одном сегменте, компиляторы высокоуровневых языков называют TINY

4. Системы команд микропроцессора Intel 8086



Команда SUB для моего варианта имеет только один варинт трансляции. В этом случаи первый операнд является регистром, а второй может быть как регистр, так и адресов.

Команда CMP имеет аналогичную структура трансляции, как и у команды SUB.

Команда JE и JNE преобразуются в машинный код следующим образом, первый байт – это сам код команда, а далее следует смещение относительно этой команду, куда нужно переместиться.

5. Контрольный пример

metka2:

subax, bx

sub cx, di

metka3:

sub al, bl

sub cx, [BX]

sub di, [SI]

sub dl, [SI]

sub bx, [DI]

metka1:

sub bx, [BX + DI]

cmp ax, bx

jne metka1

cmp cx, [BX]

je metka2

cmp bx, [BX + DI]

je metka3

cmp bl, [BX + SI]

ret

Также создадим контрольный пример для трансляции и компоновки его с помощью tasm.exe и tlink.exe.

Файл test.asm

.MODEL tiny

.CODE

.386

ORG 0100h

start:

metka2:

sub ax, bx

sub cx, di

metka3:

sub al, bl

sub cx, [BX]

sub di, [SI]

sub dl, [SI]

sub bx, [DI]

metka1:

sub bx, [BX + DI]

cmp ax, bx

jne metka1

cmp cx, [BX]

je metka2

cmp bx, [BX + DI]

je metka3

cmp bl, [BX + SI]

ret

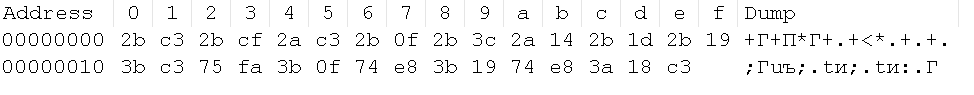
END start

Создадим файл test.com с помощью команд

*tasm test.asm/l*

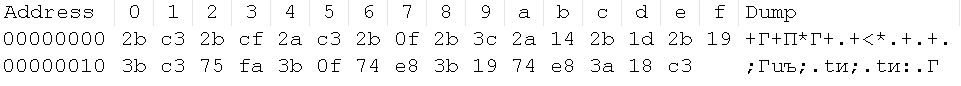
*tlink test.obj /t*

Дамп файла test.com

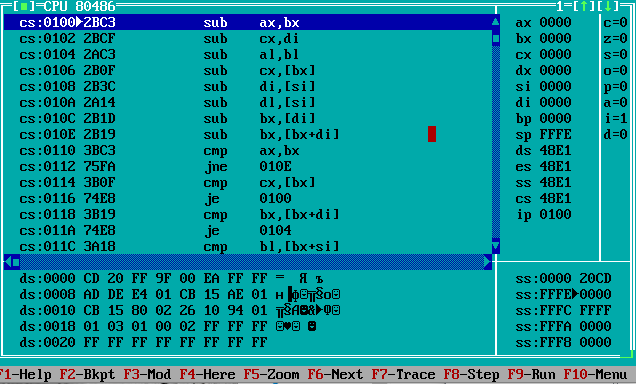


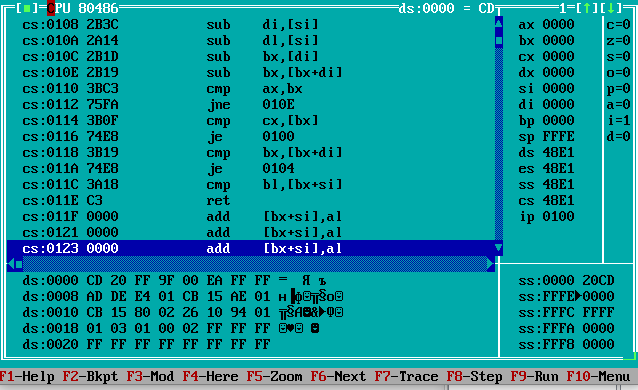
6. Результат работы програмы

Дамп файла, полученного при использовании моей программы

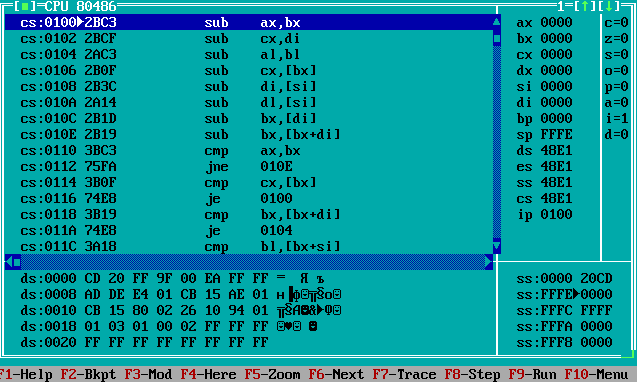
Сравниим результаты в отладчике td.exe

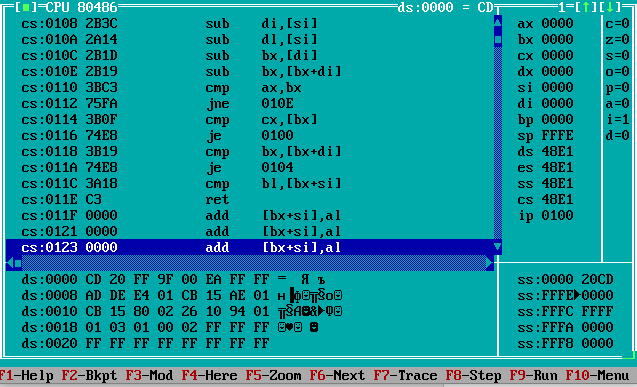
Моя программа





Программа, полученная с помощью tasm и tlink:





Сравним дампы двух программ.

2BC3 2BCF 2AC3 2B 0F 2B 3C 2A 14 2B 1D 2B 19 3BC3 75 FA 3B 0F 74 E8 3B 19 74 E8 3A 18 C3

2B C3 2B CF 2A C3 2B 0F 2B 3C 2A 14 2B 1D 2B 19 3B C3 75 FA 3B 0F 74 E8 3B 19 74 E8 3A 18 C3

Машинный код, полученный с помощью tasm и tlink и код, полученный с помощью моей программы, получились идентичные. Моя программа и tasm использовали один и тот же вариант трансляции данных команд.

7. Описание работы программы

Для данного задания достаточно было однопроходного ассемблера, так как нет ссылок вперед и получить машинный код можно за один проход.

Программа считывает каждую строчку и приводит её к стандартному виду (в нижнем регистре и без пробелов). После чего, в строчке ищутся ключевые слова, относящиеся к заданному списку команд. Так как в моем случае количество команд небольшое, то достаточно было сгруппировать некоторые команды и обрабатывать только группы.

8. Код программы

Файл main.cpp

#include<iostream>

#include<fstream>

#include<string>

#include<map>

#include<vector>

usingnamespace std;

map<string, int> METKA;

map<string, unsignedchar> MEMS = {

{"[bx]", 7},

{"[si]", 4},

{"[di]", 5},

{"[bx+si]", 0},

{"[bx+di]", 1},

};

map<string, unsignedchar> REGISTERS = {

{"ax", 0},

{"cx", 1},

{"dx", 2},

{"bx", 3},

{"sp", 4},

{"bp", 5},

{"si", 6},

{"di", 7},

{"al", 8},

{"cl", 9},

{"dl", 10},

{"bl", 11},

{"ah", 12},

{"ch", 13},

{"dh", 14},

{"bh", 15}

};

void noSpaces(string&str) {

string res = "";

for (auto s : str)

if (s != ' '&& s != '\n'&& s != '\t'&& s != '\r')

res += tolower(s);

str= res;

}

unsignedchar\* processString(string&line, int&size, int&address) {

int findIndex = 0;

bool MNEMA = false;

bool MNEMY = false;

string metka;

unsignedchar\* bytes = newunsignedchar[4];

for (int i = 0; i < 4; i++) bytes[i] = 0;

if (line.find("sub") != string::npos)

{

bytes[0] = 0b10101 << 1;

MNEMA = true;

}

if (line.find("cmp") != string::npos)

{

bytes[0] = 0b11101 << 1;

MNEMA = true;

}

if (line.find("jne") != string::npos) {

metka =line.substr(3);

bytes[0] = 0b01110101;

MNEMY = true;

}

if (line.find("je") != string::npos) {

metka =line.substr(2);

bytes[0] = 0b01110100;

MNEMY = true;

}

if (MNEMY) {

size = 2;

for (auto met : METKA)

if (met.first == metka) {

char disp = met.second - address - 2;

metka = to\_string((unsignedchar)disp);

break;

}

bytes[1] = stoi(metka);

}

if (MNEMA) {

size = 2;

string f\_reg = line.substr(3, 2);

string s\_opp = line.substr(6);

if (REGISTERS[f\_reg]< 8)

bytes[0] |= 1;

bool find = false;

for (auto regs : REGISTERS)

if (regs.first == s\_opp)

{

find = true;

break;

}

if (find) // r, r

{

bytes[1] |= 0b11 << 6;

bytes[1] |= (REGISTERS[f\_reg]& 0b111) << 3;

bytes[1] |= REGISTERS[s\_opp]& 0b111;

}

else// r, m

{

bytes[1] |= 0b00 << 6;

bytes[1] |= (REGISTERS[f\_reg]& 0b111) << 3;

bytes[1] |= MEMS[s\_opp]& 0b111;

}

}

findIndex = line.find(":");

if (findIndex != string::npos)

METKA[line.substr(0, findIndex)] = address;

address += size;

return bytes;

}

int main() {

ifstream cin;

cin.open("input.txt");

string line;

string oldline;

int i = 0;

int address = 0;

getline(cin, line);

vector<unsignedchar> bytes;

while (true)

{

if (line.find("ret") != string::npos)

break;

oldline = line;

noSpaces(line);

int size = 0;

unsignedchar\* com = processString(line, size, address);

printf("%2d %04X ", i, address - size);

for (int i = 0; i < size; i++)

printf("%02X ",com[i]);

printf(" %s\n", oldline.c\_str());

getline(cin, line);

delete [] com;

i++;

}

return 0;

}

9. Вывод

Я создал программу для транслирования определенного набора команд ассемблера в машинный код. Я разобрался со структурой .com файла. Получившийся .com файл, полученный с помощью моей программы, оказался иднетичным .com файлу, полученным с помощью tasm и tlink.