Санкт-Петербургский политехнический университет Петра Великого

Институт информационных технологий и управления

Кафедра «Информационные и управляющие системы»

**Курсовая работа**

**Разработка учебной системы программирования**

**Загрузчик, Эмулятор, Отладчик.**

по дисциплине «Системы программирования»

Вариант 9

Выполнили

студенты гр. 53504/3

Мамонтов Я. С.

Кузнецов Д. А.

Хутар Давуд Захи

Руководитель:

Расторгуев В. Я.

**«\_\_\_» \_\_\_\_\_\_\_\_\_\_** 2016 г.

Санкт-Петербург

2016

# Введение

Данная курсовая работа имеет своей целью получение практических навыков построения компилятора с языка высокого уровня (ЯВУ), являющегося одним из элементов системы программирования, образующих в совокупности технологический конвейер (см. рисунок 1).



Рис . Структура курсового проекта

При этом предполагается то, что данная система программирования работает на технологической ЭВМ (IBM PC) и является по существу кросс-системой для объектной ЭВМ (ЕС ЭВМ). В этой системе:

* в качестве языка высокого уровня (ЯВУ) выбран язык, образованный из подмножества языковых конструкций ПЛ1, а исходная программа готовится в виде текстового файла технологической ЭВМ с расширением \*.pli;
* язык АССЕМБЛЕРА сформирован из языковых конструкций АССЕМБЛЕРА ЕС ЭВМ, а ассемблеровский эквивалент исходной программы формируется в виде текстового файла технологической ЭВМ с расширением \*.ass;
* объектный эквивалент исходной программы готовится в формате объектных файлов операционной системы ОС ЕС ЭВМ и хранится в виде двоичного файла технологической ЭВМ с расширением \*.tex;
* загрузочный эквивалент исходной программы представляет собой машинный код ЕС ЭВМ, запоминаемый в области ОЗУ технологической ЭВМ, являющейся зоной загрузки для эмулятора объектной ЭВМ.

# Постановка задачи

В данной работе требуется доработать существующие элементы учебной системы программирования для обработки новых конструкций ассемблера. В данной части работы будет рассматриваться выполнение текста программы в машинном коде на абсолютном загрузчике (см. рисунок 2).

Исходный текст

на

ЯВУ

Эквивалент исходного текста на Ассемблере

Объектный эквивалент

исходного текста

Загрузочный эквивалент исходного текста

Кросс-компилятор с ЯВУ

Кросс-компилятор с Ассемблера

Загрузчик

Эмулятор

Отладчик

Рис 2: Рассматриваемая часть учебной системы

На вход загрузчику подается объектный эквивалент исходного текста для ЭВМ IBM 370 :

0245 5344 4040 4040 4040 0010 4040 0001

4558 3039 4040 4040 0000 0000 4000 0024

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0254 5854 40**00** **0000** 4040 **0002** 4040 0001

**05f0** 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0254 5854 40**00** **0002** 4040 **0004** 4040 0001

**5850 f018** 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0254 5854 40**00** **0006** 4040 **0004** 4040 0001

**8850 001d** 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0254 5854 40**00** **000a** 4040 **0004** 4040 0001

**4e50 f022** 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0254 5854 40**00** **000e** 4040 **0004** 4040 0001

**4140 f022** 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0254 5854 40**00** **0012** 4040 **0006** 4040 0001

**d203 f01c 4005** 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0254 5854 40**00** **0018** 4040 **0002** 4040 0001

**07fe** 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0254 5854 40**00** **001a** 4040 **0004** 4040 0001

**a000 0000** 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0254 5854 40**00** **001e** 4040 **0003** 4040 0001

**0000 0c**40 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0254 5854 40**00** **0024** 4040 **0008** 4040 0001

**0000 0000 0000 000c** 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

0245 4e44 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4040 4040 4040 4040

4040 4040 4040 4040 4558 3039 4040 4040

На выходе необходимо получить правильное пошаговое выполнение инструкций программы с отображением текущей команды, значением регистров и памяти программы (см. рисунок 3).

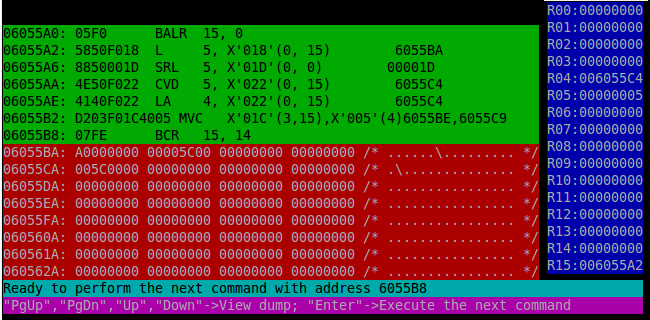


Рис 3. Выполнение программы в загрузчике

# Анализ поставленной задачи

Для реализации поставленной необходимо добавить обработку следующих команд: LA, SRL, CVD, MVC.

1. LA r1, s2

*Описание*: загрузить адрес s2 в регистр r1

*Тип*: RX

*Код*: 0x41

1. SRL r1, d2

*Описание*: логический сдвиг битов в регистре r1 на d2 позиций влево

*Тип*: RS

*Код*: 0x88

1. CVD r1, s2(x2)

*Описание*: преобразование бинарного числа из регистра r1в десятичное по адресу s2

*Тип*: RX

*Код*: 0x4E

1. MVC s1(l), s2

*Описание*: пересылка символов из памяти s2 в память s1в размере l байт

*Тип*: SS

*Код*: 0xD2

# Входные ограничения

1. Десятичные числа только положительные
2. Проверка и обработка ситуаций выхода размера чисел за отведенный предел не производится.

# Модификация базы данных исходного макета

После модификации таблица машинных операций имеет вид (новые фрагменты отмечены жирным шрифтом):

struct TMOP /\*структ.стр.табл.маш.опер\*/

{

unsigned char MNCOP [5]; /\*мнемокод операции \*/

unsigned char CODOP ; /\*машинный код операции \*/

unsigned char DLOP ; /\*длина операции в байтах \*/

int (\*BXPROG)() ; /\*указатель на подпр.обраб\*/

} T\_MOP [NOP] = /\*об'явление табл.маш.опер\*/

{

{{'B' , 'A' , 'L' , 'R' , ' '} , '\x05', 2 , FRR}, /\*инициализация \*/

{{'B' , 'C' , 'R' , ' ' , ' '} , '\x07', 2 , FRR}, /\*строк \*/

{{'S' , 'T' , ' ' , ' ' , ' '} , '\x50', 4 , FRX}, /\*таблицы \*/

{{'L' , ' ' , ' ' , ' ' , ' '} , '\x58', 4 , FRX}, /\*машинных \*/

{{'A' , ' ' , ' ' , ' ' , ' '} , '\x5A', 4 , FRX}, /\*операций \*/

{{'S' , ' ' , ' ' , ' ' , ' '} , '\x5B', 4 , FRX}, /\* \*/

//Дополнительно внедренные машинные операции

**{{'S','R','L',' ',' '} , '\x88' , 4 , FRS}, /\* Логические сдвиг вправо \*/**

**{{'C','V','D',' ',' '} , '\x4E' , 4 , FRX}, /\* Перевод в 10ное число \*/**

**{{'M','V','C',' ',' '} , '\xD2' , 6 , FSS}, /\* Пересылка символов \*/**

**{{'L','A',' ',' ',' '} , '\x41' , 4 , FRX} ,/\* Загрузка адреса \*/**

};

# Модификация алгоритма исходного макета

С целью расширения функциональности языка в функции компилятора были внесены изменения представленные ниже:

...

int FRS(); //Прототип обращения к подпрограмме обработки операндов RS-форм

int FSS(); //Прототип обращения к подпрограмме обработки операндов SS-форм

...

int B2, //Номер второго базового регистра в формате SS

D2; //Второе смещение в формате SS

…

//..........................................................................

//Программа реализации команды SRL

int P\_SRL()

{

VR[R1] >>= D; //Сдвиг регистра на велечину, записанную в смещении

return 0;

}

//..........................................................................

//Программа реализации команды CVD

int P\_CVD()

{

int sm, i; /\*рабочие переменная \*/

unsigned char t;

/\* Вычисление абсолютного адреса операнда и смещения \*/

ADDR = VR[B] + VR[X] + D;

sm = ( int ) ( ADDR - I );

/\* Преобразование в десятичное представление \*/

int value = VR[R1]; //Записыем значение регистра в переменную

for (k = 7; k>=0; k--)

{

int i = (value % 10) << 4; //Записываем младший разряд во вторую половину байта

value = value / 10; //Уменьшаем число на разряд

i += value % 10; //Прибавляем более младший разряд

value = value / 10; //Уменьшаем число на разряд

OBLZ[BAS\_IND + CUR\_IND + sm + k] = i; //Записываем результат в соответсвующий байт

}

OBLZ[BAS\_IND + CUR\_IND + sm + 7] &= 0xF0; //Обнуляем половину младшего байта

OBLZ[BAS\_IND + CUR\_IND + sm + 7] += 0xC; //Обозначаем знак числа (+).

return 0;

}

//..........................................................................

//Программа реализации команды MVC

int P\_MVC()

{

int l = LENGTH; //Длина первого операнда

int sm1 = ADDR - I; //Смещение первого операнда

int sm2 = ADDR2 - I; //Смещение второго операнда

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

{

//Побайтово копируем из второго операнда в первый

OBLZ[BAS\_IND + CUR\_IND + sm1 + i] = OBLZ[BAS\_IND + CUR\_IND + sm2 + i];

}

printf("sm1:%d, sm2:%d\n",sm1,sm2);

return 0;

}

//..........................................................................

//Программа реализации команды SRL

int P\_LA()

{

VR[R1] = VR[B] + VR[X] + D;

return 0;

}

...

//...........................................................................

//Подпрограмма обработки операндов RS-форм

int FRS()

{

int i, j;

/\* Цикл по всем возможным командам \*/

for (i = 0; i < NOP; i++)

{

if (INST[0] == T\_MOP[i].CODOP) /\* КОП найден \*/

{

/\* Вывод мнемоники команды \*/

waddstr(wgreen, " ");

for (j = 0; j < 5; j++)

waddch(wgreen, T\_MOP[i].MNCOP[j]);

waddstr(wgreen, " ");

//Вывод регистра

j = INST[1] >> 4;

R1 = j;

wprintw(wgreen, "%.1d, ", j);

//Вывод смещения

j = INST[2] % 16;

j = j \* 256 + INST[3];

D = j;

wprintw(wgreen, "X'%.3X'(", j);

//Вывод индекса

j = INST[1] % 16;

X = j;

wprintw(wgreen, "%1d, ", j);

//Вывод базы

j = INST[2] >> 4;

B = j;

wprintw(wgreen, "%1d)", j);

//Вывод абсолютного адреса

ADDR = VR[B] + VR[X] + D;

wprintw(wgreen," %.06lX \n", ADDR);

break;

}

}

return 0;

}

//...........................................................................

//Подпрограмма обработки операндов SS-форм

int FSS()

{

int i,j,k;

/\* Цикл по всем возможным командам \*/

for ( i=0; i < NOP; i++)

{

if ( INST [0] == T\_MOP[i].CODOP ) /\* КОП найден \*/

{

/\* Вывод мнемоники команды \*/

wprintw(wgreen," ");

for ( j=0; j < 5; j++)

wprintw(wgreen,"%c",T\_MOP[i].MNCOP[j]) ;

wprintw(wgreen," ");

/\* Вывод первого операнда \*/

j = INST[2] % 16;

j = j \* 256 + INST[3];

D = j; // Вывод смещения

wprintw(wgreen,"X'%.3X'(",j);

j = LENGTH = INST[1];

wprintw(wgreen,"%1d,", j);

j = INST[2] >> 4;

B = j; //Вывод базы

wprintw(wgreen,"%1d),",j);

ADDR = VR[B] + D; //Вычисление абсолютного адреса

/\* Вывод второго операнда \*/

j = INST[4] % 16;

j = j \* 256 + INST[5];

D2 = j; //Вывод смещения

wprintw(wgreen,"X'%.3X'(",j);

j = INST[4] >> 4;

B2 = j; //Вывод базы

wprintw(wgreen,"%1d)",j);

ADDR2 = VR[B2] + D2; //Запись абсолютного адреса

/\* Вывод абсолютных адресов операндов \*/

wprintw(wgreen,"%.06lX,%.06lX\n",ADDR, ADDR2);

break;

}

}

return 0;

}

...

//---------------------------------------------------------------------------

//программа покомандной интерпретпции(отладки)

// загруженной программы

int sys(void)

{

...

SKIP:

switch (T\_MOP[k].CODOP) //согласно коду команды,

{ //селектируемой сч.адреса

... //выбрать подпрогр.интер-

case 0x88:

P\_SRL();

break;

case '\x4E':

P\_CVD();

break;

case 0xD2:

P\_MVC();

break;

case '\x41':

P\_LA();

break;

}

...

}

...

# Выводы

В ходе выполнения данной части курсовой работы в базу данных и алгоритм объектного загрузчика были внесены изменения, позволяющие использовать в коде обрабатываемой программы языковые конструкции, представленные в задании.

Загрузчик успешно выполняет объектный код, представленный в главе постановка задачи и выдает ожидаемые данные в регистрах и памяти. Получаемые с помощью эмулятора данные полностью совпадают с ожидаемыми в предыдущих частях курсовой работы данными. С помощью абсолютного загрузчика и эмулятора машины была подтверждена корректность полученного во второй части курсовой работы объектного файла. Поставленная задача была успешно выполнена.