Санкт-Петербургский политехнический университет Петра Великого
Институт информационных технологий и управления
Кафедра «Информационные и управляющие системы»

Курсовая работа
Разработка учебной системы программирования

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

Санкт-Петербургский политехнический университет Петра Великого
Институт информационных технологий и управления
Кафедра «Информационные и управляющие системы»

Курсовая работа
Разработка учебной системы программирования
Компилятор языка высокого уровня.

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).

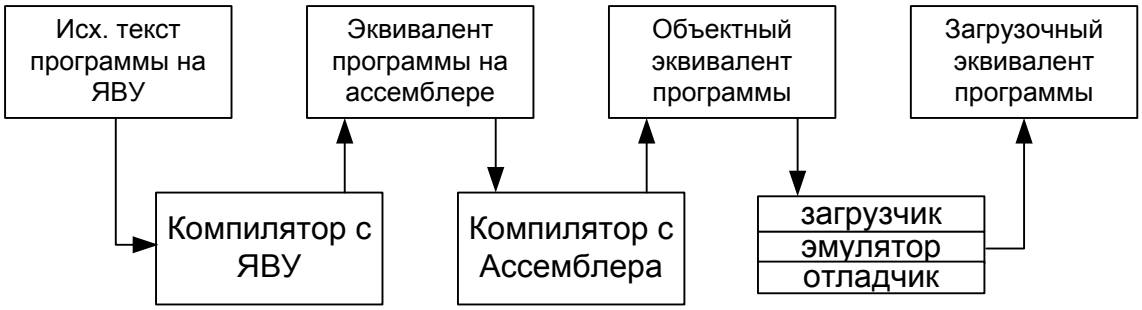


Рис 1. Структура курсового проекта

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 языка высокого уровня. В этой части работы будет рассматриваться преобразование исходного текста на языке высокого уровня в эквивалент исходного текста на Ассемблере (см. рисунок 2).

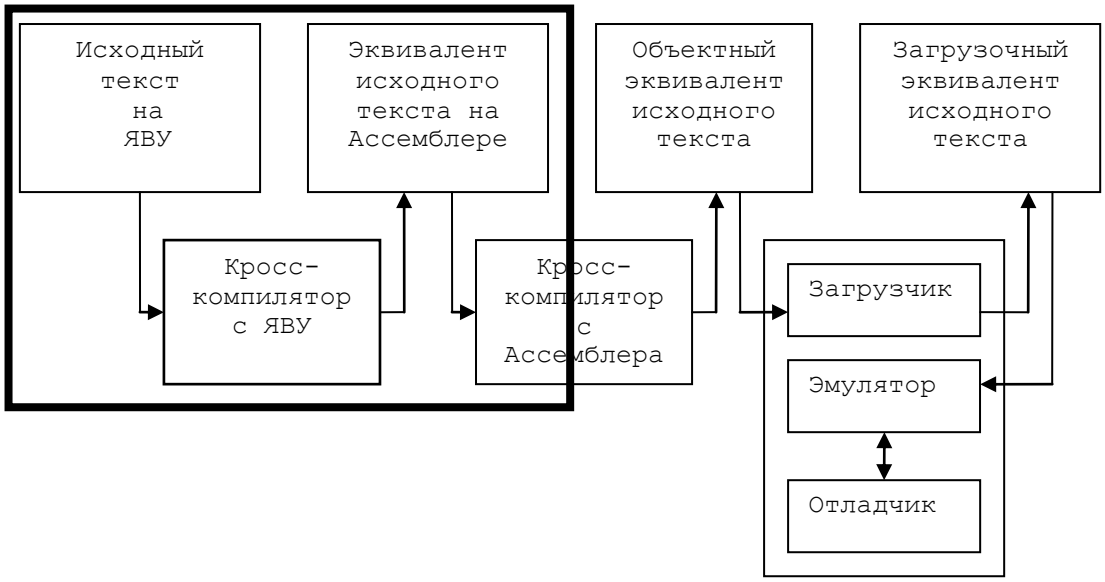


Рис 2. Рассматриваемая часть учебной системы

В качестве исходного текста на ЯВУ на вход кросс-компилятора подается текст программы, написанный на языке PL/1:

```
EX09:   PROC  OPTIONS  ( MAIN );
    DCL A   BIT  ( 5 )  INIT  ( '101'B );
    DCL B   DEC  FIXED;
    B = A;
END  EX09;
```

В результате работы компилятора должен получится эквивалент программы на языке Assembler архитектуры IBM 370:

```
EX09    START   0
        BALR    @RBASE, 0
        USING   *, @RBASE

        L       @RRAB, A
        SRL     @RRAB, 29
        CVD     @RRAB, @BUF
        LA      @RADD, @BUF
        MVC     B(3), 5(@RADD)
```

```
         BCR     15, 14

A        DC      BL4'101'
B        DC      PL3'0'
         DS      0F
@BUF     DC      PL8'0'
@RBASE   EQU     15
@RRAB    EQU     5
@RADD    EQU     4

         END
```

Анализ поставленной задачи

Основной задачей являлось преобразование переменной из битовой строки (BIT) в десятичное число (DEC). Для этого нужно осуществить цепочку преобразований BIT->BIN->DEC на языке Ассемблер.

После проведенного анализа было получено следующее эквивалентное преобразование:

Язык PL/I

```
B = A;
```

Язык Ассемблера

```
L    @RRAB, A       - Загрузка значения переменной в регистр
SRL  @RRAB, 29      - Арифметический сдвиг вправо
CVD  @RRAB, @BUF    - Смена типа
LA   @RADD, @BUF    - Загрузка адреса BUF в регистр RADD
MVC  B(3), 5(@RADD) - Перемещение
```

Входные ограничения

1. Объявление переменной как битовой строки согласно спецификации языка должно быть представлено в виде BL<кол-во байтов>.<кол-во битов>. Для решения нашей задачи мы в целях упрощения подменили объявление на BL<кол-во битов>.
2. Десятичные числа только положительные
3. Размер памяти, выделяемый по умолчанию под хранение десятичного числа: 3 байта. Данное ограничение было введено вследствие того, что в задании при объявлении десятичного числа не указывается размерность, следовательно, при разработке компилятора необходимо это учесть, и точно задать размер выделяемой памяти по умолчанию.
4. Проверка и обработка ситуаций выхода размера чисел за отведенный предел не производится.
5. Предполагается использовать для служебных целей метки, начинающиеся с символа «@». Таким образом пользователь не может использовать символ @ в начале идентификаторов.

Преобразование грамматики

В грамматику языка были внесены следующие изменения (выделены жирным):

```
1.  <PRO> ::= <OPR><TEL><OEN>
2.  <OPR> ::= <IPR>:PROC_OPTIONS(MAIN);
3.  <IPR> ::= <IDE>
4.  <IDE> ::= <BUK> | <IDE><BUK> | <IDE><CIF>
5.  <BUK> ::= A | B | C | D | E | M | P | X
6.  <CIF> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
7.  <TEL> ::= <ODC> | <TEL><ODC> | <TEL><OPA>
8.  <ODC> ::= DCL_<IPE>_BIN_FIXED(<RZR>);|
              DCL_<IPE>_BIN_FIXED(<RZR>)INIT(<LIT>) ;|
              DCL_<IPE>_BIT(<RZR>)INIT(<LIT>);|
               DCL_<IPE>_DEC_FIXED;
9.  <IPE> ::= <IDE>
10. <RZR> ::= <CIF> | <RZR><CIF>
11. <LIT> ::= <MAN>B
12. <MAN> ::= 1 | <MAN>0 | <MAN>1
13. <OPA> ::= <IPE>=<AVI>;
14. <AVI> ::= <LIT> | <IPE> | <AVI><ZNK><LIT> |
               <AVI><ZNK><IPE>
15. <ZNK> ::= + | -
16. <OEN> ::= END_<IPR>
```

Здесь использованы следующие метасимволы и символы:

"<" и ">" - левый и правый ограничители нетерминального символа;
"::=" - метасимвол со смыслом "равно по определению";
"|" - метасимвол альтернативного определения "или";
"_" - терминальный символ со смыслом "пробел";
"<PRO>" - нетерминал "программа";
"<OPR>" - нетерминал "оператор пролога программы";
"<IPR>" - нетерминал "имя программы";
"<IDE>" - нетерминал "идентификатор";
"<BUK>" - нетерминал "буква";
"<CIF>" - нетерминал "цифра";
"<TEL>" - нетерминал "тело программы";
"<ODC>" - нетерминал "оператор declare";
"<IPE>" - нетерминал "имя переменной";
"<RZR>" - нетерминал "разрядность";
"<LIT>" - нетерминал "литерал";
"<MAN>" - нетерминал "мантисса";
"<OPA>" - нетерминал "оператор присваивания арифметический";
"<AVI>" - нетерминал "арифметическое выражение";
"<ZNK>" - нетерминал "знак";

"<OEN>" - нетерминал "оператор эпилога программы".

Далее рассмотрим часть продукций, которые были изменены в ходе работы:

```
D-><BUK>
DCL_<IPE>_BIN_FIXED(<RZR>);-> <ODC>
DCL_<IPE>_BIN_FIXED(<RZR>)INIT(<LIT>); -> <ODC>
DCL_<IPE>_BIT(<RZR>)INIT(<LIT>); -> <ODC>
DCL_<IPE>_DEC_FIXED; -> <ODC>
```

Сгруппируем получившиеся продукции в «кусты»:

```
D-><BUK>
    |
    ->CL_<IPE>_->DEC_FIXED;-><ODC>
                 |
                 ->BI->T(<RZR>)INIT(<LIT>);-><ODC>
                       |
                       ->N_FIXED(<RZR>)->;-><ODC>
                                          |
                                            ->INIT(<LIT>);-><ODC>
```

Модификация базы данных исходного макета

Нами были добавлены дополнительные определения операций в синтаксис. Полученные ранее «кусты» были представлены в табличном виде и записаны в таблицу синтаксических правил:

```
 struct
{
    int  POSL;
    int  PRED;
    char DER[4];
    int  ALT;
} SINT[NSINT] =
/*
      |  NN      :    посл : пред  :  дер   : альт |
      |__________:_________:_______:_______:______|
*/
{
...
      {/*.  201   .*/  202 ,    51 , "T  " ,    0 },
      {/*.  202   .*/  203 ,   201 , "(  " ,    0 },
      {/*.  203   .*/  204 ,   202 , "RZR" ,    0 },
      {/*.  204   .*/  205 ,   203 , ")  " ,    0 },
      {/*.  205   .*/  206 ,   204 , ";  " ,   65 },
      {/*.  206   .*/  207 ,   205 , "ODC" ,    0 },
      {/*.  207   .*/    0 ,   206 , "*  " ,    0 },

      {/*.  208   .*/  209 ,    49 , "D  " ,    0 },
      {/*.  209   .*/  210 ,   208 , "E  " ,    0 },
      {/*.  210   .*/  211 ,   209 , "C  " ,    0 },
      {/*.  211   .*/  212 ,   210 , "   " ,    0 },
      {/*.  212   .*/  213 ,   211 , "F  " ,    0 },
      {/*.  213   .*/  214 ,   212 , "I  " ,    0 },
      {/*.  214   .*/  215 ,   213 , "X  " ,    0 },
      {/*.  215   .*/  216 ,   214 , "E  " ,    0 },
      {/*.  216   .*/  217 ,   215 , "D  " ,    0 },
      {/*.  217   .*/  218 ,   216 , ";  " ,   65 },
      {/*.  218   .*/  219 ,   217 , "ODC" ,    0 },
      {/*.  219   .*/  0   ,   218 , "*  " ,    0 }
};
```

Остальные таблицы базы данных компилятора менять не пришлось. Модифицировать таблицу входов в «кусты» не понадобилось, т.к. при добавлении новых синтаксических правил, новых входов в «кусты» не появилось. Таблица матрицы смежности также осталась без изменений, т.к. добавленные новые правила не привнесли новых символов в алфавит.

Модификация алгоритма исходного макета

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

```
int ODC1();
int AVI2();
int OPA2();
int OEN2();
```

Также для повышения функциональности и улучшения внешнего вида команды создания временно переменной `BUFF` были вынесены в отдельную функцию `AddTemp8BytesVariable`.

Измененные отрывки функций представлены ниже:

```
/* Функция создания вспомогательной 8 байтовой переменной для CVD */
void AddTemp8BytesVariable( char* name )
{
    strcpy ( SYM [ISYM].NAME, name );    //Пишем имя, например TEMPDEC
    strcpy ( SYM [ISYM].RAZR, "8" );     //Разрядность 8 байт
    strcpy ( SYM [ISYM].INIT, "0B" );    //Значение 0
    SYM [ISYM].TYPE = 'D';               //Тип DEC
    ISYM++;                             //Увеличиваем количество символов на 1
}

/* Программа семантического вычисления нетерминала ODC на первом проходе */
int ODC1()
{
  ...
  /* Случай типа BIT */
    else if ( !strcmp( FORMT[2], "BIT" ) )
    {
        SYM[ISYM].TYPE = 'L';     /* Устанавливаем тип переменной */

        strcpy( SYM[ISYM].RAZR, FORMT[3] );  /* Устанавливаем разрядность */

        if( !strcmp(FORMT[4], "INIT") )       /* Инициализация */
            strcpy(SYM[ISYM].INIT, FORMT[5]);
        else
            strcpy(SYM[ISYM].INIT, "0B");
    }
  /* Случай типа DEC FIXED */
    else if ( !strcmp( FORMT[2], "DEC" ) && !strcmp( FORMT[3], "FIXED" ) )
    {
        SYM[ISYM].TYPE = 'D';       /* Устанавливаем тип переменной */

        strcpy( SYM[ISYM].RAZR, FORMT[3] );
        SYM[ISYM].RAZR[0] = '3';             /* Устанавливаем разрядность */
        SYM[ISYM].RAZR[1] = '\0';
        strcpy(SYM[ISYM].INIT, "0B");        /* Инициализация */
    }
  ...
 }

/* Программа семантического вычисления нетерминала AVI на втором проходе */
int AVI2()
```

```
{
   ...
   /* Случай типа BIT */
   else if( SYM[i].TYPE == 'L' )
   {
            /* Установка операции */
      memcpy(ASS_CARD._BUFCARD.OPERAC, "LH", 2);
            /* Установка регистра */
      strcpy( ASS_CARD._BUFCARD.OPERAND, "RRAB," );
            /* Установка адреса ячейки памяти */
      strcat( ASS_CARD._BUFCARD.OPERAND, FORMT[0]);
            /* вставляем разделитель  */
      ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = ' ';
            /* и построчный коментарий*/
      memcpy( ASS_CARD._BUFCARD.COMM, "Load the variable to the register", 33
);
            /*Запоминаем операцию ассемблера*/
      ZKARD();
      return 0;
   }
   ...
}

/* Программа семантического вычисления нетерминала OPA на втором проходе */
int OPA2()
{
   ...
   /* Случай типа DEC FIXED */
   else if( SYM[i].TYPE == 'D' )
   {
      int j;
      for( j = 0; j < ISYM; j++ )
      {
        if( !strcmp(SYM[j].NAME, FORMT[1]) && strlen(SYM[j].NAME) ==
strlen(FORMT[1]) )
        {
          /* Случай типа BIT */
          if( SYM[j].TYPE == 'L' )
          {
            /*    Arithmetic shift    */
            /* Установка операции */
            memcpy(ASS_CARD._BUFCARD.OPERAC, "SRL", 3);
            /* Установка регистра */
            strcpy( ASS_CARD._BUFCARD.OPERAND, "RRAB," );
            /* Вычисление смещения */
            int int_offset = 16 - strlen(SYM[j].INIT) + 1;
            char ch_offset[2];
            sprintf(ch_offset, "%d", int_offset);
            strcat( ASS_CARD._BUFCARD.OPERAND, ch_offset);
            /* вставляем разделитель  */
            ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = '
';
            /* и построчный коментарий*/
            memcpy( ASS_CARD._BUFCARD.COMM, "Arithmetic shift", 16 );
                  /*Запоминаем операцию ассемблера*/
            ZKARD();

            /*    Change type    */
            memcpy(ASS_CARD._BUFCARD.OPERAC, "CVD", 3);
            strcpy( ASS_CARD._BUFCARD.OPERAND, "RRAB," );
            strcat( ASS_CARD._BUFCARD.OPERAND, "BUF");
                                        /* вставляем разделитель  */
```

```
            ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = '
';
                                                /* и построчный коментарий*/
            memcpy( ASS_CARD._BUFCARD.COMM, "Change type", 11 );

            ZKARD();

            /*    Load addres to register    */
            AddTemp8BytesVariable("BUF");
            memcpy(ASS_CARD._BUFCARD.OPERAC, "LA", 2);
            strcpy( ASS_CARD._BUFCARD.OPERAND, "RADD," );
            strcat( ASS_CARD._BUFCARD.OPERAND, "BUF");
            /* вставляем разделитель  */
            ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = '
';
            /* и построчный коментарий*/
            memcpy( ASS_CARD._BUFCARD.COMM, "Load addres to register", 23 );

            ZKARD();

            /*    Moving to needed variable    */
            memcpy(ASS_CARD._BUFCARD.OPERAC, "MVC", 3);
            strcpy( ASS_CARD._BUFCARD.OPERAND, FORMT[1] );
            strcat( ASS_CARD._BUFCARD.OPERAND, "(3),5(RADD)");
            /* вставляем разделитель  */
            ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = '
';
            /* и построчный коментарий*/
            memcpy( ASS_CARD._BUFCARD.COMM, "Moving to needed variable", 25 );

            ZKARD();

            return 0;
          }
          return 3;
        }
      }
   }
   ...
}

/* Программа семантического вычисления нетерминала OEN на втором проходе */
int OEN2()
{
   ...
   /* Случай типа BIT */
    else if( SYM[i].TYPE == 'L' )
    {
     /* Установка имени переменной */
        strcpy( ASS_CARD._BUFCARD.METKA,SYM[i].NAME );
        ASS_CARD._BUFCARD.METKA[strlen(ASS_CARD._BUFCARD.METKA)] = ' ';
     /* Установка операции */
        memcpy( ASS_CARD._BUFCARD.OPERAC, "DC", 2 );
     /* Установка размера и начального значения */
        strcpy( ASS_CARD._BUFCARD.OPERAND,"BL" );
        strcat( ASS_CARD._BUFCARD.OPERAND, "16" );
        strcat( ASS_CARD._BUFCARD.OPERAND, "\'" );
        SYM[i].INIT[strlen(SYM[i].INIT) - 1] = '\0';
        strcat( ASS_CARD._BUFCARD.OPERAND, SYM[i].INIT );
     /* замыкающий апостроф    */
        ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = '\'';
        /* построчнsq комментарий*/
        memcpy(ASS_CARD._BUFCARD.COMM,"Definition of variable", 22);
```

```
            ZKARD();
        }
    /* Случай типа DEC FIXED */
        else if( SYM[i].TYPE == 'D' )
        {
         /* Добавляем выравнивание */
            if (!strcmp(SYM [i].NAME, "BUF"))
            {
                memcpy ( ASS_CARD._BUFCARD.OPERAC, "DS", 2 );
           /* Выравнивание 4 байта */
                memcpy ( ASS_CARD._BUFCARD.OPERAND, "0F", 2 );
                memcpy ( ASS_CARD._BUFCARD.COMM, "Aligment", 8 );
                ZKARD ();
            }

            strcpy( ASS_CARD._BUFCARD.METKA,SYM[i].NAME );
            ASS_CARD._BUFCARD.METKA[strlen(ASS_CARD._BUFCARD.METKA)] = ' ';
            memcpy( ASS_CARD._BUFCARD.OPERAC, "DC", 2 );
            strcpy( ASS_CARD._BUFCARD.OPERAND,"PL" );
            strcat( ASS_CARD._BUFCARD.OPERAND, SYM[i].RAZR );
            strcat( ASS_CARD._BUFCARD.OPERAND, "\'" );
            SYM[i].INIT[strlen(SYM[i].INIT) - 1] = '\0';
            strcat( ASS_CARD._BUFCARD.OPERAND, SYM[i].INIT );
            ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = '\'';
            memcpy(ASS_CARD._BUFCARD.COMM,"Definition of variable", 22);

            ZKARD();
        }
    ...
}
```

Выводы

В ходе выполнения данной части курсовой работы в базу данных и алгоритм компилятора с PL/1 были внесены необходимые изменения, позволяющие использовать в коде программы языковые конструкции, представленные в задании. Общая архитектура данной части системы позволяет достаточно легко вносить необходимые изменения в алгоритм работы и расширять набор поддерживаемых конструкций ЯВУ. При выполнении работы столкнулись с проблемой использования функции CVD, у которой на выходе 8 байтовая переменная, вместо необходимой 3 байтовой и с проблемой выравнивания переменных в памяти.

Полученный в ходе работы компилятор был успешно скомпилирован. После его запуска с поданным на вход заданным вариантом задания № 9 был получен исходный код на языке Ассемблера, который полностью соответствует разработанному вручную варианту, описанному в главе «Постановка задачи».

Санкт-Петербургский политехнический университет Петра Великого
Институт информационных технологий и управления
Кафедра «Информационные и управляющие системы»

Курсовая работа

Разработка учебной системы программирования

Компилятор с языка Assembler.

по дисциплине «Системы программирования»

Вариант 9

Выполнили
студенты гр. 53504/3

Мамонтов Я. С.
Кузнецов Д. А.
Хутар Давуд Захи

Руководитель:

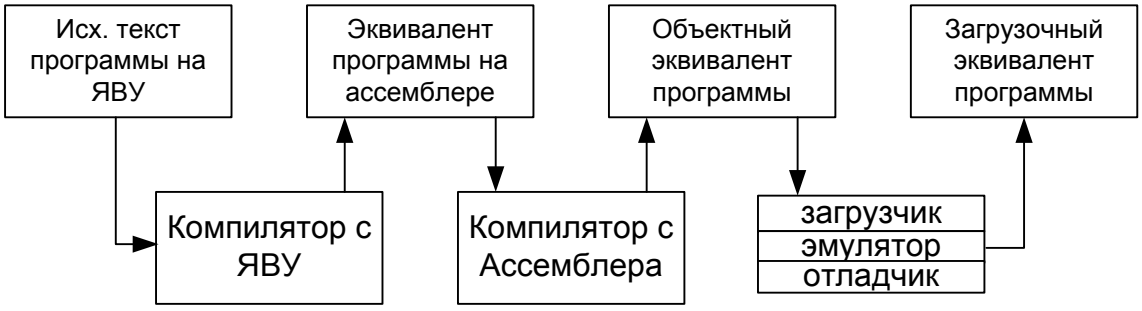
Расторгуев В. Я.

«___» _____ 2016 г.

Санкт-Петербург
2016

Введение

Данная курсовая работа имеет своей целью получение практических навыков построения компилятора с языка Ассемблер, являющегося одним из элементов системы программирования, образующих в совокупности следующий технологический конвейер:

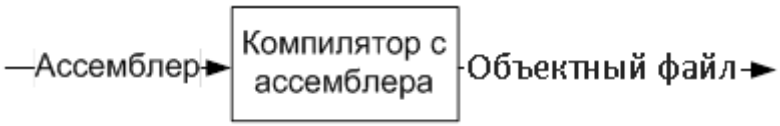


При этом предполагается то, что данная система программирования работает на технологической ЭВМ (IBM PC) и является по существу кросс-системой для объектной ЭВМ (ЕС ЭВМ). В этой системе:

- 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

Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать “новые” для макета конструкции языка высокого уровня, примененные в соответствующем варианте:



На входе имеется исходный код программы на ассемблере ЭВМ IBM 370:

```
EX09     START 0             Start of the programm
         BALR  @RBASE,0      Load the register of the base
         USING *,@RBASE      Set register as the base
         L     @RRAB,A       Load the variable to the register
         SRL   @RRAB,29      Logic shift
         CVD   @RRAB,@BUF    Change type
         LA    @RADD,@BUF    Load addres to register
         MVC   B(3),5(@RADD) Moving to needed variable
         BCR   15,14         Exit from the programm
A        DC    BL4'101'      Definition of variable
B        DC    PL3'0'        Definition of variable
         DS    0F            Aligment
@BUF     DC    PL8'0'        Definition of variable
@RBASE   EQU   15
@RRAB    EQU   5
@RADD    EQU   4

         END                 End of the programm
```

В результате работы компилятора должен получится эквивалент исходной программы в виде объектного файла для IBM 370:

```
0245 5344 4040 4040 4040 0010 4040 0001
4558 3039 4040 4040 0000 0000 4000 002c
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 4000 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 4000 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 4000 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 4000 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 4000 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 4000 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 4000 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 4000 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 4000 001e 4040 0003 4040 0001
0000 0040 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 4000 0024 4040 0008 4040 0001
0000 0000 0000 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 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

Анализ поставленной задачи

Команды ассемблера кодируются следующими кодами:

SRL @RRAB, 13	**88 50 00 0d**
CVD @RRAB, @BUF	**4e 50 f0 46**
LA @RADD, @BUF	**41 40 f0 46**
MVC B(3), 5(@RADD)	**d2 03 f0 1c 40 05**

Входные ограничения

Битовая строка должна быть не более 1 байта.

Модификация базы данных исходного макета

Таблица машинных операций имеет вид:

```
// Кодирование дополнительных команд для нашего
варианта
{{'L','A',' ',' ',' '} , '\x41' , 4 , FRX} ,
{{'C','V','D',' ',' '} , '\x4E' , 4 , FRX} ,
{{'S','R','L',' ',' '} , '\x88' , 4 , FRS} ,
{{'M','V','C',' ',' '} , '\xD2' , 6 , FSS}
```

Также в начале второго просмотра были установлены указатели на программные обработчики новых команд:

```
// Указатели на подпрограммы-обработчики команд АССЕМБЛЕРА
// при втором просмотре
T_MOP[6].BXPROG = SRX;
T_MOP[7].BXPROG = SRX;
T_MOP[8].BXPROG = SRS;
T_MOP[9].BXPROG = SSS;
```

Модификация алгоритма исходного макета

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

//БЛОК объявлений подпрограмм, используемых при 1-ом проходе

```
int FDC()
{
    int size;
    if ( PRNMET == 'Y' )
    {
        if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0]=='F')
```

```
        {
            T_SYM[ITSYM].DLSYM = 4;
            T_SYM[ITSYM].PRPER = 'R';
            if ( CHADR % 4 )
            {
                CHADR = (CHADR /4 + 1) * 4;
                T_SYM[ITSYM].ZNSYM = CHADR;
            }
            PRNMET = 'N';
        }
        else if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0]=='B')
        {
            if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1]=='L')
            {
                size = atoi(&TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[2]);
                T_SYM[ITSYM].DLSYM = size;
                T_SYM[ITSYM].PRPER = 'R';
                T_SYM[ITSYM].ZNSYM = CHADR;
                PRNMET = 'N';
            }
        }
        else if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0]=='P')
        {
            if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1]=='L')
            {
                size = atoi(&TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[2]);
                T_SYM[ITSYM].DLSYM = size;
                T_SYM[ITSYM].PRPER = 'R';
                T_SYM[ITSYM].ZNSYM = CHADR;
                PRNMET = 'N';
            }
        }
        else
        return (1);                                    /* иначе выход по ошибке  */
    }
    else                                              /*если же псевдооп.непомеч*/
        if ( CHADR % 4 )                                /*и CHADR не кратен 4,то: */
            CHADR = (CHADR /4 + 1) * 4;                    /* установ.CHADR на гр.сл.*/

    PRNMET = 'N';
    CHADR = CHADR + size;                             /*увелич.CHADR на 4 и     */
    return (0);                                       /*успешно завершить подпр.*/
}
```

```
int FDS()
{
    if ( PRNMET == 'Y' )
    {
        if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1]=='F')



        {
            T_SYM[ITSYM].DLSYM = 4;
            T_SYM[ITSYM].PRPER = 'R';
            if ( CHADR % 4 )
            {
                CHADR = (CHADR /4 + 1) * 4;
            }
            else
```

```
            {
              CHADR = CHADR + 4;
            }
            PRNMET = 'N';
        }
        else
        {
          return (1);
        }
    }
    else
    {
        T_SYM[ITSYM].DLSYM = 4;
        T_SYM[ITSYM].PRPER = 'R';
        if ( CHADR % 4 )
            CHADR = (CHADR /4 + 1) * 4;
        else
          CHADR = CHADR + 4;
    }
    T_SYM[ITSYM].ZNSYM = CHADR;
    return (0);
}
```

```
//Обработка RS комманд на первом проходе
int FRS()
{
   CHADR = CHADR + 4;
   if ( PRNMET == 'Y' )
   {
      T_SYM[ITSYM].DLSYM = 4;
      T_SYM[ITSYM].PRPER = 'R';
   }
   return(0);
}
```

```
//Обработка SS комманд на первом проходе
int FSS()
{
    CHADR = CHADR + 6;
    if ( PRNMET == 'Y' )
    {
        T_SYM[ITSYM].DLSYM = 6;
        T_SYM[ITSYM].PRPER = 'R';
    }
    return 0;
}
```

// Второй проход

```
int SDC()
{
    char *RAB;                                  /*рабочая переменная      */
    RX.OP_RX.OP   = 0;                          /*занулим два старших     */
    RX.OP_RX.R1X2 = 0;                          /*байта RX.OP_RX          */
    if (!memcmp(TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "F'", 2))
    {
        RAB = strtok((char*)TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND+2, "'");
        RX.OP_RX.B2D2 = atoi ( RAB );             /*перевод ASCII-> int     */
        RAB = (char *) &RX.OP_RX.B2D2;            /*приведение к соглашениям*/
        swab ( RAB , RAB , 2 );                   /* ЕС ЭВМ                 */
```

```
        STXT (4);                                          /*формирование TXT-карты  */
        return (0);                                        /*успешн.завершение подпр.*/
    }
    else if ( !memcmp( TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "BL", 2 ) )
    {
      RAB=strtok( (char*)TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND + 4, "'" );
      //Our awesome input
      int size = strlen(RAB);
      int value = strtol( RAB, NULL, 2 );
      char buffer[1];
      buffer[0] = value<<(8-size);
      memcpy(BL_BUFFER, buffer, 1);
      STXT (1);                                            /*формирование TXT-карты  */
      return (0);
    }
    else if ( !memcmp(TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "PL", 2) )
    {
      RAB=strtok( (char*)TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND+4, "'" );
      int size = TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[2]-'0';

      RX.OP_RX.B2D2 = atoi ( RAB );                    /*перевод ASCII-> int     */
      RAB = (char *) &RX.OP_RX.B2D2;                   /*приведение к соглашениям*/
        printf("1 -> %d\n", RX.OP_RX.B2D2);
        printf("1 -> %s\n", RAB);

      char buffer[8];
      memset ( buffer , 64 , 8 );
      memset ( buffer , 0 , size-1 );
      buffer[size-1] = RX.OP_RX.B2D2;
      if (RX.OP_RX.B2D2 >= 0)
      {
        buffer[0] = 0xc<<4;
      }
      if (size <= 4)
      {
        memcpy(RX.BUF_OP_RX, buffer, 4);
        printf("2 -> %s\n", RX.BUF_OP_RX);
        STXT (size);
      }
      else
      {
        memcpy(PL8_BUFFER, buffer, 8);
        STXT (size);
      }
      return (0);
    }
    else
        return (1);
}
```

```
int SDS()                                              /*подпр.обр.пс.опер.DS    */
 {

  RX.OP_RX.OP   = 0;                                   /*занулим два старших     */
  RX.OP_RX.R1X2 = 0;                                   /*байта RX.OP_RX          */

  if ( TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1]=='F')
  {                          /* то:                       */
      RX.OP_RX.B2D2 = 0;                                 /*занулим RX.OP_RX.B2D2   */
  }
  else                                                 /*иначе                   */
```

```
   return (1);                                        /*сообщение об ошибке     */

  if ( CHADR % 4 )
  {                                                   /*  если не кратен 4   */
    STXT2(4 - (CHADR % 4));                          /*   делаем выравнивание*/
  }
  else
  {
    STXT2(4);
  }

  return (0);
 }
```

```
int SRS()
{
   char *METKA;                                       /*набор                   */
   char *METKA1;                                      /*рабочих                 */
   char *METKA2;                                      /*переменных              */
   char *PTR;                                         /*                        */
   int  DELTA;                                        /*                        */
   int  ZNSYM;                                        /*                        */
   int  NBASRG;                                       /*                        */
   int J;                                             /*                        */
   int I;                                             /*                        */
   unsigned char R1X2;                                /*                        */
   int B2D2;
   int DLSYM;                                        /*                       */
   RX.OP_RX.OP = T_MOP[I3].CODOP;                     /*формирование кода операц*/
   METKA1 = strtok((char*) TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, ",");
   METKA2 = strtok(NULL, " ");

   if ( isalpha ( (int) *METKA1 ) || METKA1[0] == '@' )
   {
      for ( J=0; J<=ITSYM; J++ )
      {
         METKA = strtok((char*) T_SYM[J].IMSYM , " ");
         if(!strcmp (METKA, METKA1))
         {
            NBASRG = 0;
            DELTA  = 0xfff - 1;
            ZNSYM  = T_SYM[J].ZNSYM;
            DLSYM =  T_SYM[J].DLSYM;   /*   смещен.втор.операнда */
            R1X2 = T_SYM[J].ZNSYM << 4;

            // New code
            DELTA  = atoi(METKA2);
            B2D2 = NBASRG << 12;
            B2D2 = B2D2 + DELTA;
            PTR = (char *)&B2D2;
            swab ( PTR , PTR , 2 );
            RX.OP_RX.B2D2 = B2D2;
            RX.OP_RX.R1X2 = R1X2;
            STXT(4);
            return(0);
         }
      }
      return(2);
   }
   else
   {
```

```
     R1X2 = atoi ( METKA1 ) << 4;
   }
}
```

```
int SSS() {

  unsigned i, j;
    unsigned rbase, delta, offset;
    char *op1, *len1, *op2, *op3;
    char *tmp;

    op1 = strtok((char *) TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "(");
    len1 = strtok(NULL, ")");
    char * t = strtok(NULL, ",");
    op2 = strtok(t, "(");
    op3 = strtok(NULL, ")");


    SS.OP_SS.OP  = T_MOP[I3].CODOP;
    SS.OP_SS.L1 = atoi(len1);

    if (isalpha((int) *op1) || op1[0] == '@')
    {
        for (i = 0; i <= ITSYM; i++)
        {
            tmp = strtok((char *) T_SYM[i].IMSYM, " ");
            if (!strcmp(tmp, op1))
            {
                rbase = 0;
                delta = 0xfff - 1;
                offset = T_SYM[i].ZNSYM;

                for (j = 0; j < 15; j++)
                {
                    if (T_BASR[j].PRDOST == 'Y' &&
                        offset - T_BASR[j].SMESH >= 0 &&
                        offset - T_BASR[j].SMESH < delta)
                    {
                        rbase = j + 1;
                        delta = offset - T_BASR[j].SMESH;
                    }
                }
                if (rbase == 0 || delta > 0xfff)
                    return 5;
                else
                {
                    SS.OP_SS.B1D1 = rbase << 12;
                    SS.OP_SS.B1D1 = SS.OP_SS.B1D1 + delta;
                    tmp = (char *) &SS.OP_SS.B1D1;
                    swab(tmp, tmp, 2);
                    goto CNT1;
                }
            }
        }
        printf("FAIL 1\n");
        return 2;
    }
    else
        printf("FAIL 2\n");
        return 2;
```

```
CNT1:
    if (isalpha((int) *op3) || op3[0] == '@')
    {
        for (i = 0; i <= ITSYM; i++)
        {
            tmp = strtok((char *) T_SYM[i].IMSYM, " ");
            if (!strcmp(tmp, op3))
            {
                SS.OP_SS.X2 = atoi(op2);
                SS.OP_SS.L2 = T_SYM[i].ZNSYM << 4;
                goto CNT2;
            }
        }
    }

CNT2:
    STXT(6);

    return 0;
}
```

```
void STXT( int ARG )                                /*подпр.формир.TXT-карты  */
 {
  char *PTR;                                        /*рабоч.переменная-указат.*/

  PTR = (char *)&CHADR;                             /*формирование поля ADOP  */
  TXT.STR_TXT.ADOP[2]  = *PTR;                      /*TXT-карты в формате     */
  TXT.STR_TXT.ADOP[1]  = *(PTR+1);                  /*двоичного целого        */
  TXT.STR_TXT.ADOP[0]  = '\x00';                    /*в соглашениях ЕС ЭВМ    */

  if ( ARG == 1 )                                   /*формирование поля OPER  */
  {
    memset ( TXT.STR_TXT.OPER , 64 , 8 );
    memcpy ( TXT.STR_TXT.OPER,BL_BUFFER , 4 );
    TXT.STR_TXT.DLNOP [1] = 4;
    ARG = 4;
  }
  else if ( ARG == 2 )
   {
    memset ( TXT.STR_TXT.OPER , 64 , 8 );
    memcpy ( TXT.STR_TXT.OPER,RR.BUF_OP_RR , 2 ); /* для RR-формата         */
    TXT.STR_TXT.DLNOP [1] = 2;
   }
  else if (ARG == 4)
   {
    memcpy ( TXT.STR_TXT.OPER , RX.BUF_OP_RX , 4);/* для RX-формата         */
    TXT.STR_TXT.DLNOP [1] = 4;
   }
   else if (ARG == 6)
   {
    memcpy ( TXT.STR_TXT.OPER , SS.BUF_OP_SS , 6);/* для SS-формата         */
    TXT.STR_TXT.DLNOP [1] = 6;
   }
  else if (ARG ==8)
  {
    memset ( TXT.STR_TXT.OPER , 64 , 8 );
    memcpy ( TXT.STR_TXT.OPER , PL8_BUFFER , 8);     /* для PL8                */
    TXT.STR_TXT.DLNOP [1] = 8;
  }
  else
  {
```

```
    memset ( TXT.STR_TXT.OPER , 64 , 8 );
    memcpy ( TXT.STR_TXT.OPER , RX.BUF_OP_RX , ARG);/* для PL         */
    TXT.STR_TXT.DLNOP [1] = ARG;
  }

  memcpy (TXT.STR_TXT.POLE9,ESD.STR_ESD.POLE11,8);/*формиров.идентифик.поля */

  memcpy ( OBJTEXT[ITCARD] , TXT.BUF_TXT , 80 );  /*запись об'ектной карты  */
  ITCARD += 1;                                    /*коррекц.инд-са своб.к-ты*/
  CHADR = CHADR + ARG;                            /*коррекц.счетчика адреса */
  return;
 }



// Используется SDS для выравнивания адреса заолнением памяти до значения кратного 4
 void STXT2( int ARG )                             /*подпр.формир.TXT-карты  */
 {
  char *PTR;                                      /*рабоч.переменная-указат.*/

  PTR = (char *)&CHADR;                           /*формирование поля ADOP  */
  TXT.STR_TXT.ADOP[2]  = *PTR;                    /*TXT-карты в формате     */
  TXT.STR_TXT.ADOP[1]  = *(PTR+1);                /*двоичного целого        */
  TXT.STR_TXT.ADOP[0]  = '\x00';                  /*в соглашениях ЕС ЭВМ    */
  memcpy ( TXT.STR_TXT.OPER , RX.BUF_OP_RX , 0);/* для RX-формата         */
  TXT.STR_TXT.DLNOP [1] = ARG;
  memcpy (TXT.STR_TXT.POLE9,ESD.STR_ESD.POLE11,8);/*формиров.идентифик.поля */
  memcpy ( OBJTEXT[ITCARD] , TXT.BUF_TXT , 80 );  /*запись об'ектной карты  */
  ITCARD += 1;                                    /*коррекц.инд-са своб.к-ты*/
  CHADR = CHADR + ARG;                            /*коррекц.счетчика адреса */
  return;
 }
```

Заключение

В ходе выполнения данной части курсовой работы в алгоритм компилятора с ассемблера были внесены изменения, позволяющие использовать в коде обрабатываемой им программы языковые конструкции, представленные в задании. В результате данной части курсовой работы был получен объектный модуль. Проверить его правильность можно с помощью абсолютного загрузчика и эмулятора машины.

Полученный в ходе работы компилятор был успешно скомпилирован. После его запуска с поданным на вход кодом ассемблера, полученным в результате работы компилятора с языка PL/1, примененного к коду варианта задания № 9, был получен объектный файл для IBM 370, который полностью соответствует разработанному вручную варианту, описанному в главе «Постановка задачи».

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).

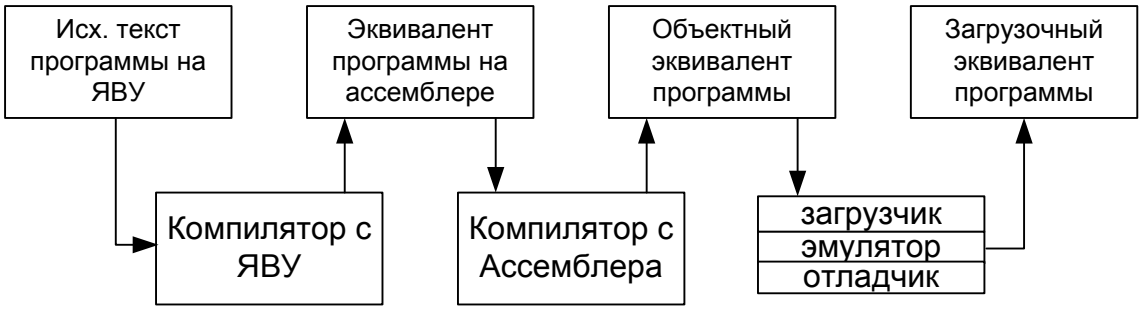


Рис 1. Структура курсового проекта

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).

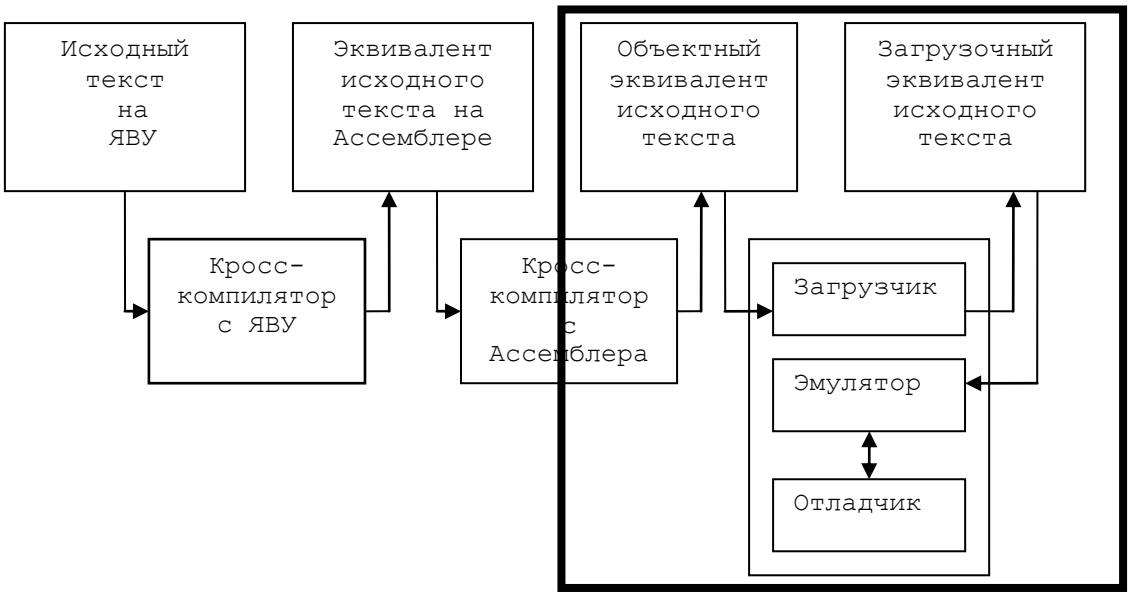


Рис 2: Рассматриваемая часть учебной системы

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 4000 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 4000 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 4000 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 0c40 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).

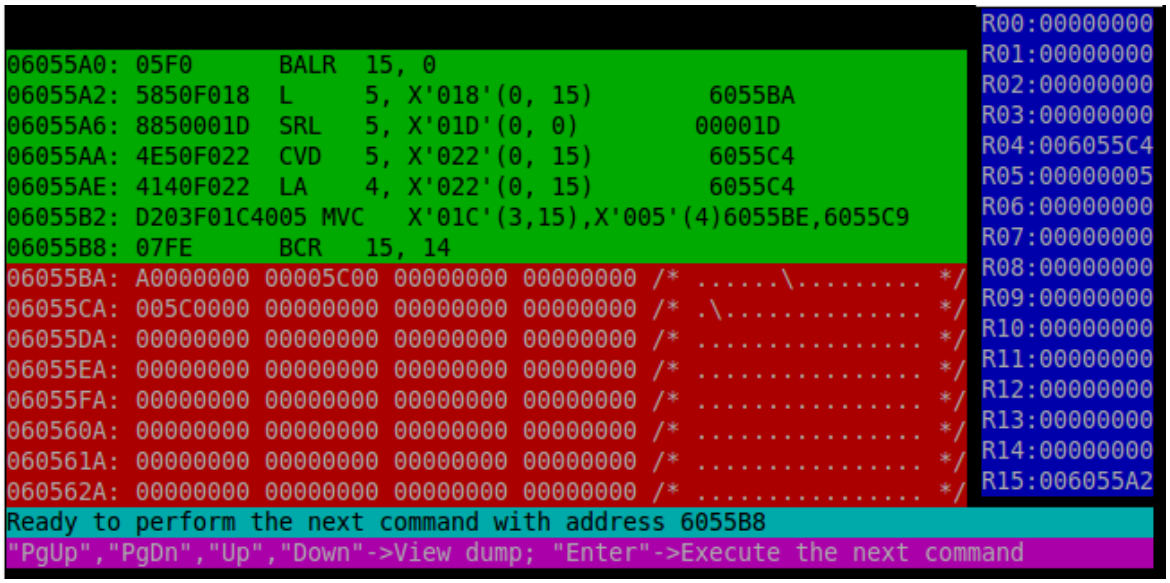


Рис 3. Выполнение программы в загрузчике

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

2) SRL r1, d2
   Описание: логический сдвиг битов в регистре r1 на d2 позиций влево
   Тип: RS
   Код: 0x88

3) CVD r1, s2(x2)
   Описание: преобразование бинарного числа из регистра r1в десятичное по адресу s2
   Тип: RX
   Код: 0x4E

4) 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.