Лабораторная работа №8

Генерация объектного кода для символьного ассемблерного файла

7.1 Пример программы, для команд которой генерируются символьные объектные коды (директивы db, dw, команды mov, mul, pop)

```
ten          db     0ah
hun      dw 100
ohun     db 101
thous dw        0ff6h
hund  db 78h
str      db 1ah

mul ten
mul hun
mul bx
mul cx
mul ax
pop      dx
pop      ds
pop hun
mov thous,ax
mov      al,ten
mov       ax,hun
mov      thous,cs
mov      hun,es
mov     ds,hun
mov     thous,es
```

7.3 Результат генерации объектного кода

№ строки программы	Адрес	Символический код директивы или команды	Строка программы
1	0000	0A	ten db 0ah
2	0001	0064	hun dw 100
3	0003	65	ohun db 101
4	0004	0FF6	thous dw 0ff6h
5	0006	78	hund db 78h
6	0007	1A	str db 1ah
8	0008	F6 26 0000	mul ten
9	000C	F7 26 0001	mul hun
10	0010	F7 E3	mul bx
11	0012	F7 E1	mul cx
12	0014	F7 E0	mul ax
13	0016	5A	pop dx
14	0017	1F	pop ds
15	0018	8F 06 0001	pop hun
16	001C	A3 0004	mov thous,ax
17	001F	A2 0000	mov ten,al
18	0022	A0 0000	mov al,ten
19	0025	A1 0001	mov ax,hun
20	0028	8C 0E 0004	mov thous,cs
21	002C	8C 06 0001	mov hun,es
22	0030	8E 1E 0001	mov ds,hun
23	0034	8C 06 0004	mov thous,es

7.3 Символические коды команд

Многие специфические команды имеют однобайтовые машинные коды, например:

№	Объектный код	Символическая команда
1	40	INC AX ;увеличение AX на 1
2	50	PUSH AX ;запись AX в стек
3	C3	RET (short) ;короткий возврат из процедуры

7.3.1 Обозначение регистров

Команды, использующие регистр, могут содержать три бита, указывающих на конкретный регистр, и один бит «w», определяющий размер регистра: байт или слово. Кроме того, лишь некоторые команды обеспечивают доступ к сегментным регистрам. Ниже показана полная идентификация регистров.

Основные, базовые и индексные регистры:

Биты	w=0	w=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Сегментные регистры:

Биты	Сегментный регистр
00	ES
01	CS
10	SS
11	DS

Рассмотрим команду MOV с однобайтовым непосредственным операндом:

```
MOV AH,00        10110 100 00000000
                       |   |
                       w  reg = AH
```

В данном случае первый байт машинного кода указывает на однобайтовый размер (w = 0) и на регистр AH (100). Следующая команда MOV содержит непосредственный двухбайтовый операнд:

```
MOV AX,00        10111 000 00000000
                      |   |
                     w  reg = AX
```

Первый байт машинного кода указывает на размер в одно слово (w=1) и на регистр AX (000). Не следует обобщать приведенные примеры, так как указание регистра и бита w может быть в различных позициях кода.

7.3.2 Байт способа адресации

Байт способа адресации, если он присутствует, занимает второй байт машинного кода и состоит из следующих трех элементов:

1) mod – двухбитового кода, имеющего значения 11 для ссылки на регистр и 00, 01 и 10 для ссылки на память;

2) reg – трехбитового указателя регистра;

3) r/m – трехбитового указателя регистра или памяти (r – регистр, m – адрес памяти).

Кроме того, первый байт машинного кода может содержать бит «d», который указывает направление потока между операндом 1 и операндом 2.

Рассмотрим пример сложения содержимого регистра AX с содержимым регистра BX:

```
ADD BX, AX 00000011    11    011  000
                      dw  mod   reg  r/m
```

В примере d=1 означает, что mod (11) и reg (011) описывают операнд 1, а r/m (000) описывает операнд 2. Так как бит w=1, то размер равен одному слову. Таким образом, команда должна прибавить AX (000) к BX (011).

Второй байт команды в объектном коде указывает большинство способов адресации памяти.

7.3.3 Биты MOD

Два бита mod определяют адресацию регистра или памяти. Ниже поясняется их назначение:

00 – биты r/m дают абсолютный адрес, байт смещения (относительный адрес) отсутствует;

01 – биты r/m дают абсолютный адрес памяти и имеется один байт смещения;

10 – биты r/m дают абсолютный адрес и имеется два байта смещения;

11 – биты r/m определяют регистр. Бит w (в байте кода операции) определяет ссылку на восьми- или шестнадцатибитовый регистр.

7.3.4 Биты REG

Три бита reg (вместе с битом w) определяют конкретный восьми- или шестнадцатибитовый регистр.

7.3.5 Биты R/M

Три бита r/m (регистр/память) совместно с битами mod определяют способ адресации, как показано ниже

Биты R/M	**mod=00**	**mod=01**	**mod=10**	**mod=11 w=0**	**mod=11 w=0**
000	BX + SI	BX + SI + disp8	BX + SI + disp16	AL	AX
001	BX + DI	BX + DI + disp8	BX + DI + disp16	CL	CX
010	BP + SI	BX + DI + disp8	BX + DI + disp16	DL	DX
011	BP + DI	BP + DI + disp8	BP + DI + disp16	BL	BX
100	SI	SI + disp8	SI + disp16	AH	SP
101	DI	DI + disp8	DI + disp16	CH	BP
110	Direct	BP + disp8	BP + disp16	DH	SI
111	BX	BX + disp8	BX + disp16	BH	DI

Ниже приведены объектные коды команд MOV, MUL и POP.

Объектный код команды MOV (семь форматов):

Регистр/память в/из регистр:

|100010dw|modregr/m|

Непосредственное значение в регистр/память:

|1100011w|mod000r/m|--data--|data если w=1|

Непосредственное значение в регистр:

|1011wreg|--data--|data если w=1|

Память в регистр AX (AL):

|1010000w|addr-low|addr-high|

Регистр AX (AL) в память:

|1010001w|addr-low|addr-high|

Регистр/память в сегментный регистр:

|10001110|mod0sgr/m| (sg – сегментный регистр)

Сегментный регистр в регистр/память:

|10001100|mod0sgr/m| (sg – сегментный регистр)

Объектный код команды MUL:

|1111011w|mod100r/m|

Объектный код команды POP (три формата):

Регистр: |01011reg|

Сегментный регистр: |000sg111| (sg – сегментный регистр)

Регистр/память: |10001111|mod000r/m|

Объектные коды всех команд ассемблера можно посмотреть:

1. Абель П. Язык Ассемблера для IBM PC и программирования / П. Абель. – М.: Высшая школа, 1992. – 447 с.

2. Глава 25. Справочник по командам языка Ассемблер URL: http://www.studfiles.ru/preview/6324282/page:25/

7.4 Генерация символических кодов команд

Листинг 1 – основная программа Laba8.pas

{ Генерация объектного кода: Команды: POP, MUL, MOV.

Директивы: DB, DW }

```
program generate_code;
uses  lab7_u,crt;  { используется модуль lab7_u из предыдущей лаб. работы}
type    obj_addr = record
                    lex:integer;    { номер лексемы }
                    addr:integer;   { ее адрес  }
                   end;
var     cur_str:integer;
        addreses:array [1..100] of obj_addr;
        curr_obj:byte;
        strg,tmp_str:string;
        f,s_f:text;
        i,j:integer;
        ch:char;
        curr_addr:integer;
        str_index:integer;
        str_file:string;
{*****************************************************}
```

{ Передаем номер строки исходного файла и флаг, возвращаем строку

из 4-х шестнадцатиричных цифр}

```
procedure to_binary(n:integer; var s:string; flag:byte);
const hex:array[0..15] of char = ('0','1','2','3','4','5','6','7','8','9','A',
                                  'B','C','D','E','F');
var  mask:longint;
     i,j,BT:byte;
```

```
    k:integer;
begin
    s:='';
  {  BT:=8;
    if flag = 1 then mask:=$80
    else       begin     mask:=$8000;  BT:=BT+8;  end;
    for i:=1 to BT do begin
       if (mask and n)=mask then s:=s+'1' else s:=s+'0';
       mask:=mask shr 1;
    end; }
    if flag=2 then begin
        mask:=$F000; BT:=4;  { количество шестн. цифр }
    end
    else begin
        mask:=$F0; BT:=2;     { количество шестн. цифр }
    end;
    for i:=1 to BT do begin
       k:= mask and n;
       for j:=i to BT-1 do
         k:=k shr 4;
       s:=s+hex[k];
       mask:=mask shr 4;
    end;
end;
{*****************************************************}
{ Получение кода регистра }
function reg_code(i:integer; code:integer):integer;
var      k:integer;
begin
```

```
   for k:=1 to 8 do
      if reg1[k]=lex_table[i].Name then begin
                                 reg_code:=code+k-1;
                                 exit;
                                 end;
   for k:=1 to 8 do
      if reg2[k]=lex_table[i].Name then begin
                                 reg_code:=code+k-1;
                                 exit;
                                 end;
end;
{*******************************************************}
```
{ Получение кода сегментного регистра }
```
function sreg_code(i:integer):integer;
begin
   if lex_table[i].Name='ES' then sreg_code:=$07        {000 00 111}
   else if lex_table[i].Name='SS' then sreg_code:=$17   {000 10 111}
   else sreg_code:=$1F;                                 {000 11 111}
end;
{*******************************************************}
```
{ Преобразование шестнадцатеричного числа в целое }
```
function hex_to_int(s:string):integer;
const    arr:array [0..15] of char=('0','1','2','3','4','5','6','7','8','9',
                    'A','B','C','D','E','F');
var i,j:byte;
   res,sixt:integer;
begin
   sixt:=16*16*16;
   res:=0;
```

```
   for i:=length(s) to 3 do s:='0'+s;
   for i:=1 to 4 do begin
      for j:=0 to 15 do if arr[j]=s[i] then break;
      res:=res+j*sixt;
      sixt:=sixt div 16;
   end;
   hex_to_int:=res;
end;
{*******************************************************}
```
{ Преобразование целого числа в строку шестнадцатеричных цифр }

```
procedure str_to_hex(i:integer; var s:string);
var     n,code:integer;
begin
   if Lex_table[i].Name[length(lex_table[i].Name)]='H' then begin
     s:=lex_table[i].Name;
     delete(s,length(s),1);
     n:=hex_to_int(s);
   end
   else
     val(lex_table[i].Name,n,code);
     if lex_table[i-1].lex_N=3 then to_binary(n,s,2)
                              else to_binary(n,s,1);
end;
{*******************************************************}
```
{ Функция получения адреса }

```
function get_addr(int:integer):integer;
var       name:string[7];
         j,k:integer;
begin
```

```
        name:=lex_table[int].Name;
        for j:=1 to int-1 do
           if lex_table[j].Name=name then break;
        for k:=1 to curr_obj-1 do
           if addreses[k].lex=j then begin
                                get_addr:=addreses[k].addr;
                                exit;
                                end;
end;
```
{ Генерация i-той строки файла листинга }
```
procedure generate(i:integer);
var code:integer;
   s:string;
begin
    { формируем i-тую строку для записи в файл }
    { Добавляем номер }
    str(lex_table[i].str_N,s);
    { Добавляем табуляцию }
    strg:=s+#9;
    { Добавляем адрес команды или директивы к строке }
    to_binary(curr_addr,s,2);
    strg:=strg+s+#9;
    { Если директивы db или dw}
    if lex_table[i+1].lex_N in [2,3] then begin
       { Номер лексемы = i}
       addreses[curr_obj].lex:=i;
       { Адрес лексемы = curr_addr}
       addreses[curr_obj].addr:=curr_addr;
       { Увеличиваем номер объекта }
```

```
      inc(curr_obj);
      { Преобразуем в шестнадцатиричный вид}
      str_to_hex(i+2,s);
      { Добавляем к строке }
      strg:=strg+s;
      { Если db, то адрес наращиваем на 1, если dw, то на 2 }
      if lex_table[i+1].lex_N=2 then curr_addr:=curr_addr+1
                              else curr_addr:=curr_addr+2;
      end;
    { Если команда mul }
    if lex_table[i].lex_N=6 then begin
      { $F6 - ячейка памяти-байт или регистр-байт }
      { $F7 - ячейка памяти-слово или регистр-слово }
      if (check_already_db(i+1)) or (lex_table[i+1].lex_N=7) then code:=$F6
                                                    else code:=$F7;
      { Преобразуем в шестнадцатеричную строку и добавляем к текущей
строке }
      to_binary(code,s,1);
      strg:=strg+s;
      { Если ячейка памяти, то добавляем 2 к адресу }
      if lex_table[i+1].lex_N=12 then begin
                              curr_addr:=curr_addr+2;
                              code:=$26;
                              end
          { Получение кодов регистров байтовых и словных }
      else if lex_table[i+1].lex_N in [7,8] then code:=reg_code(i+1,$E0);
      { Преобразуем в шестнадцатеричную строку и добавляем к текущей
строке }
      to_binary(code,s,1);
```

```
  strg:=strg+' '+s;
  if lex_TABLE[i+1].lex_N=12 then begin
        to_binary(get_addr(i+1),s,2);
        strg:=strg+' '+s;
  end;
  curr_addr:=curr_addr+2;
end;
{ Если команда Pop}
if lex_table[i].lex_N=5 then begin
      {ячейка памяти-слово}
   if check_already_dw(i+1) then begin code:=$8F;
         to_binary(code,s,1);
         strg:=strg+s;
         curr_addr:=curr_addr+3;
   end;
   { После идет ячейка-памяти}
   if lex_table[i+1].lex_N=12 then code:=$06
   { регистры размером слово }
   else if lex_table[i+1].lex_N=8 then code:=reg_code(i+1,$58)
   { регистр CS }
   else if lex_table[i+1].lex_N=10 then code:=$0F
   { остальные сегментные регистры }
   else if lex_table[i+1].lex_N=9 then code:=sreg_code(i+1);
   to_binary(code,s,1);
   strg:=strg+' '+s;
   { изменение адреса }
   if lex_table[i+1].lex_N=12 then begin
                        to_binary(get_addr(i+1),s,2);
                        strg:=strg+' '+s;
```

```
                                end;
    curr_addr:=curr_addr+1;
end;
if lex_table[i].lex_N=4 then begin
    if lex_table[i+1].lex_N in [7,8] then
      if (lex_table[i+1].Name='AX') or (lex_table[i+1].Name='AL') then
        if (lex_table[i+3].lex_N=12) then
        begin
        if check_already_dw(i+3) then code:=$A1
                                else code:=$A0;
        to_binary(code,s,1);
        strg:=strg+' '+s;
        to_binary(get_addr(i+3),s,2);
        strg:=strg+' '+s;
        end;
    if lex_table[i+3].lex_N in [7,8] then
      if (lex_table[i+3].Name='AX') or (lex_table[i+3].Name='AL') then
        if (lex_table[i+1].lex_N=12) then
        begin
        if check_already_dw(i+1) then code:=$A3
                                else code:=$A2;
        to_binary(code,s,1);
        strg:=strg+' '+s;
        to_binary(get_addr(i+1),s,2);
        strg:=strg+' '+s;
        end;
    if lex_table[i+1].lex_N = 9 then
      if (lex_table[i+3].lex_N=12) then
        if check_already_dw(i+3) then begin
```

```
                code:=$8E;
                to_binary(code,s,1);
                strg:=strg+' '+s;
                code:=sreg_code(i+3)-1;
                to_binary(code,s,1);
                strg:=strg+' '+s;
                to_binary(get_addr(i+3),s,2);
                strg:=strg+' '+s;
                inc(curr_addr);
              end;
          if lex_table[i+3].lex_N in [9,10] then
            if (lex_table[i+1].lex_N=12) then
              if check_already_dw(i+1) then begin
                code:=$8C;
                to_binary(code,s,1);
                strg:=strg+' '+s;
                if lex_table[i+3].lex_N<>10 then
                    code:=sreg_code(i+3)-1
                else code:=$0E;
                to_binary(code,s,1);
                strg:=strg+' '+s;
                to_binary(get_addr(i+1),s,2);
                strg:=strg+' '+s;
                inc(curr_addr);
              end;
          curr_addr:=curr_addr+3;
      end;
  end;
```

```
begin   { очистка экрана}
        clrscr;
        { Поле строка в таблице лексем обнуляем }
        for i:=1 to 1000 do begin
            lex_table[i].str_N:=0;
        end;
        { Текущая лексема =1}
        curr_lex:=1;
        write('Введи имя файла: ');
        readln(str_file);
        assign(f,str_file);
        { Открываем файл на чтение }
        reset(f);
        str_index:=1;
        strg:='';
        { Читаем посимвольно из файла }
        while not eof(f) do begin
             read(f,ch);
             if ch=#13 then begin
                    { Если конец строки, то синтаксический анализ }
                            writeln(strg);
                            string_analize(strg,str_index);
                            inc(str_index);
                            strg:='';
                            read(f,ch);
                        end
             else begin
              { Если символ входит в алфавит, то добавляем его к строке }
                     ch:=upcase(ch);
```

```
                if (ch in [' ',',',#13,#10,#9,'A'..'Z','0'..'9','_','?','@','$','&'])
                then strg:=strg+ch
                else begin
                      writeln('Ошибка в строке ',str_index);
                      readln;
                      halt;
                     end;
            end;
        end;
        { Закрываем файл и анализируем последнюю строку }
        close(f);
        writeln(strg);
        string_analize(strg,str_index);
        { Создаем файл лексем и записываем в него лексемы }
        i:=1;
        assign(f,'results.txt');
        rewrite(f);
        while lex_table[i].str_N<>0 do begin
           writeln(f,lex_table[i].str_N:15,lex_table[i].lex_N:15,
                                  lex_table[i].Name:15);
           i:=i+1;
        end;
        close(f);
        { Получаем код ошибки синтаксического разбора }
        synth_analize(i);
        if i=0 then writeln('Ошибок нет')
              else writeln('Ошибка в строке № ',i);
{*******************************************************}
        assign(f,'listing.lst');
```

```
        assign(s_f,str_file);
        { открываем файл listing на запись }
        rewrite(f);
        { открываем исходный файл на чтение }
        reset(s_f);
        i:=1;
        cur_str:=1;     { текущая строка в исходном файле }
        curr_addr:=0;   { текущий адрес элемента в листинге}
        curr_obj:=1;    { текущий объект в листинге }
        { Перебираем все строки таблицы лексем }
        while lex_table[i].str_N<>0 do begin
            generate(i);
            readln(s_f,tmp_str);
            strg:=strg+#9#9+tmp_str;
            writeln(f,strg);
            i:=i+1;
            while lex_table[i].str_N = cur_str do i:=i+1;
            for j:=cur_str+1 to lex_table[i].str_N-1 do
               readln(s_f,tmp_str);
            cur_str:=lex_table[i].str_N;
        end;
        close(f);
        close(s_f);
        readln;
end.
```