4 Декабря 2013 года в 20:09

1. Введение.

LINKBK

Простейший линковщик объектных файлов ассемблера MACRO-11
(Версия 0.1)

Yellow Rabbit

Линковщик предназначен для получения исполняемых файлов из объектных файлов ассемблера MACRO-11[1]. Входными параметрами являются: перечень имен объектных файлов, имя выходного файла[2] и несколько управляющих ключей. На выходе создается исполняемый файл для БК11М.

Линковщик понимает следующие записи объектных файлов, генерируемых ассемблером MACRO-11:

- имя модуля, версия модуля, адрес запуска;
- программные секции, как абсолютные, так и перемещаемые;
- прямые и косвенные ссылки внутри модуля;
- прямые, прямые со смещением, косвенные и косвенные со смещением ссылки на глобальные символы;
- прямые, прямые со смещением, косвенные и косвенные со смещением ссылки на программные секции внутри модуля;
- запись предельных адресов генерируемого исполняемого файла;
- сложные ссылки.

Игнорируются:

- работа с библиотеками.

[1] Использовалась BSD-версия ассемблера Richard'a Krehbiel'a.

[2] На самом деле выходных файлов может быть много, но указывается только имя файла для основной/стартовой секции.

2. Примеры входных файлов.

3. В самом простом случае программа состоит из одного файла, в котором не используются именованные секции, и, следовательно, нет глобальных символов, и нет явных межсекционных ссылок.

Листинг простейшей программы.

```
;; vim: set expandtab ai textwidth=80:
                .TITLE  SIMLBL
                .IDENT  /V00.10/
;; Все секции начинаются с 0, поэтому нужно смещение
.=.+1000
Start:          jsr     PC,@140010      ; инициализируем монитор БК11М
mov     #30204,R0
                jsr     PC,@140132      ; дисплей в 80 символов в строке (и пр.)
                mov     #Welcome,R0
                jsr     PC,@140160      ; вывод строки
                jsr     PC,@140076      ; ожидание нажатия клавиши
                mov     #Bye,R0
                jsr     PC,@140160      ; еще вывод строки
                jsr     PC,@140076      ; еще ожидание нажатия клавиши
                jmp     @#140000        ; reset монитора
Welcome:        .ASCIZ  /Very simple MACRO-11 program./
Bye:            .ASCIZ  /Bye./
                .END    Start

```

Пусть файл называется `simple.asm`, и, после компиляции ассемблером, получается объектный файл `simple.o`. После запуска линковщика `linkerbk -v -o simple simple.o` получается отчет:

```
Module:SIMLBL
  Ident: V00.10
=Global Definitions:
=Sections:
. ABS., addr: 0x8006da000, len: 0, min addr: 37777777777, current start: 0
      , addr: 0x8006eb000, len: 1114, min addr: 1000, current start: 0
```

То есть, ассемблер все равно создал две секции: абсолютную секцию с именем `. ABS.`, которая не содержит зарезервированного места для данных (`len: 0`) и не содержит самих данных (`min addr: 37777777777`), и перемещаемую секцию с именем, состоящим из шести пробелов, которая резервирует место для 1114 байт и содержит данные, начиная со смещения 1000.

Линковщик создает только один выходной файл — `simple`.

4. Создаваемой по умолчанию абсолютной секцией можно воспользоваться, если применить директиву ассемблера `.ASECT` как в следующем примере:

```
;; vim: set expandtab ai textwidth=80:
                .TITLE  TSTABS
                .IDENT  /V00.10/
;; Все секции начинаются с 0, поэтому нужно смещение
.=.+1000
Start:          jsr     PC,@140010      ; инициализируем монитор БК11М
                mov     #30204,R0
                jsr     PC,@140132      ; дисплей в 80 символов в строке (и пр.)

                ;; Запоминаем старый обработчик клавиатуры
                mov     @#KeyboardVect,OldHandler
                mov     @#KeyboardPSW,OldHandler+2

                ;; «Тут установка нового обработчика и какая-то программа»

                jsr     PC,@140076      ; ожидание нажатия клавиши
                jmp     @#140000        ; reset монитора
OldHandler:     .BLKW   2               ; место под данные старого обработчика
                                        ; клавиатуры
;; Абсолютная секция без указания имени
                .ASECT
.=60             ; смещение, так как секция всегда начинаются с нуля
KeyboardVect:   .=.+2
KeyboardPSW:
                .END    Start

```

Лог линковки показывает, что абсолютная секция резервирует место для данных, но, поскольку сами данные в секцию не загружаются, то будет создан только один файл, который содержит неименованную перемещаемую секцию:

```
Module:TSTABS
  Ident: V00.10
=Global Definitions:
=Sections:
. ABS., addr: 0x8006da000, len: 62, min addr: 37777777777, current start: 0
      , addr: 0x8006eb000, len: 1044, min addr: 1000, current start: 0
```

Это только демонстрация того, что можно использовать неименованную абсолютную секцию.

5. На практике для таких вещей как адреса векторов прерываний или регистров устройств или еще чего-нибудь проще использовать прямое присваивание символов как в следующем примере[3].

```
;; vim: set expandtab ai textwidth=80:
                .TITLE  TSTAB2
                .IDENT  /V00.11/
                .INCLUDE lib/bk11m/bk11m.inc

                ;; Вектор прерывания от клавиатуры
KeyboardVect=60
KeyboardPSW=KeyboardVect+2

.=.+1000        ;; Все секции начинаются с 0, поэтому нужно смещение
Start:          .BINIT                  ; инициализируем монитор БК11М
                .BTSET  #30204          ; дисплей в 80 символов в строке (и пр.)

                .BPRIN   #HelloStr
                ;; Запоминаем старый обработчик клавиатуры
                mov     @#KeyboardVect,OldHandler
                mov     @#KeyboardPSW,OldHandler+2

                ;; «Тут установка нового обработчика и какая-то программа»

                .BTTIN                  ; ожидание нажатия клавиши
                .BEXIT                  ; выход
OldHandler:     .BLKW   2               ; место под данные старого обработчика
                                        ; клавиатуры
HelloStr:       .ASCIZ  /Hi, there! :)/
                .END    Start

```

[3] Это также пример того, как можно написать программу на одних макросах:) При компиляции нужно указать macro11 каталог с файлами макросов (опция `-p`) или определить переменную окружения `MCALL`. В конце этого документа приведены файлы макросов для монитора БК11М и MKDOS.

6. Использование именованной секции для размещения подпрограмм, подгружаемых во время выполнения. Пусть во время выполнения программы необходимо считать с диска и выполнить некую подпрограмму, тогда выделяем это подпрограмму в отдельную именованную программную секцию (SUBS) и указываем для нее начальный адрес, чтобы линковщик смог правильно скорректировать ссылки на эту секцию.

Линковщик пишет именованные нестартовые секции в отдельный файлы оверлеев, имена которых получаются из имени выходного исполняемого файла + "-" + имя секции + ".v". При необходимости полученное имя файла оверлея урезается до указанной длины[4].

Нужно заметить, что линковщик не накладывает ограничений на адреса именованных секций, и задача грузить их в нужное место остается за программистом.

```
1  ;; vim: set expandtab ai textwidth=80:
2                  .TITLE  OVRTST
3                  .IDENT  /V00.11/
4                  .INCLUDE lib/bk11m/bk11m.inc
5                  .INCLUDE lib/mkdos/mkdos.inc
6
7  ;; Адрес с которого располагается оверлеи
8  LoadAddr=40000
9
10 ;; Все секции начинаются с 0, поэтому нужно смещение
11 .=.+1000
12 Start:          AFTER$MKDOS             ; если планируем использовать функции
13                                         ; MKDOS
14                 .BINIT                  ; инициализируем монитор БК11М
15
16                 ; подготавливаем распределение памяти ‘‘под себя’’,
17                 ; с 40000 страница 1  -- здесь будут оверлеи
18                 ; с 100000 страница 2 -- просто так, чтобы там заведомо не было
19                 ; MKDOS.
20                 .BPAGE 1,0
21                 .BPAGE 2,1
22
23                 .BTSET  #30204
24                 .BPRIN  #Prompt
25                 .BTTIN
26
27                 ; загружаем оверлей
28                 MKDOS$TAPE #TapeParams
29
30                 ; вызываем функцию из оверлея
31                 jsr     PC,SayHi
32
33                 .BPRIN  #Loaded
34                 .BTTIN
35                 BACK$TO$MKDOS           ; назад в MKDOS
36
37 Prompt:         .ASCIZ  /Press any key to load overlay.../
38 Loaded:         .ASCIZ  /Overlay loaded./
39                 .EVEN
40 TapeParams:     .BYTE   3,0
```

[4] Ключ -l линковщика.

```
41                  .WORD   LoadAddr,0
42 1$:              .ASCII  /overlay-SUBS.v/        ; Имя файла оверлея
43                  .BLKB   ^D16-<.-1$>
44                  .BLKB   ^D16+4
45
46 ;; ============================================
47 ;;  Оверлей.
48 ;;  Содержит подпрограмму SayHi, которая
49 ;;  выводит на экран строчку.
50 ;; ============================================
51                  .PSECT  SUBS
52 .=.+LoadAddr
53 SayHi:           .BPRIN  #HiStr
54                  rts     PC
55 HiStr:           .ASCIZ  /I'm overlay!/
56                  .END    Start
57
```

Лог линковки:

```
1 Module:OVRTST
2   Ident: V00.11
3 =Global Definitions:
4 =Sections:
5 . ABS., addr: 0x8006da000, len: 0, min addr: 37777777777, current start: 0
6       , addr: 0x8006eb000, len: 1342, min addr: 1000, current start: 0
7 SUBS  , addr: 0x8006fc000, len: 40030, min addr: 40000, current start: 0
```

7. Линковщик объединяет именованные программные секции с одинаковым именем, поэтому программу можно разбить на много небольших файлов, которые ассемблируются по-отдельности, а затем линкуются.

Первый файл:

```
;; vim: set expandtab ai textwidth=80:
                .TITLE  OVR2
                .IDENT  /V01.00/
                .INCLUDE lib/bk11m/bk11m.inc
                .INCLUDE lib/mkdos/mkdos.inc

;; Адрес с которого располагается оверлеи
LoadAddr=40000

                .PSECT  MAIN
;; Все секции начинаются с 0, поэтому нужно смещение
.=.+1000
Start:          AFTER$MKDOS             ; если планируем использовать функции
                                        ; MKDOS
                .BINIT                  ; инициализируем монитор БК11М

                ; подготавливаем распределение памяти ‘‘под себя’’,
                ; с 40000 страница 1  -- здесь будут оверлеи
                ; с 100000 страница 2 -- просто так, чтобы там заведомо не было
                ; MKDOS.
                .BPAGE 1,0
                .BPAGE 2,1

                .BTSET  #30204
                .BPRIN  #Prompt
                .BTTIN

                ; загружаем оверлей
                MKDOS$TAPE #TapeParams

                ; вызываем функцию из оверлея
                jsr     PC,SayHi

                .BPRIN  #Loaded
                .BTTIN
                BACK$TO$MKDOS           ; назад в MKDOS

Prompt:         .ASCIZ  /Press any key to load overlay.../
Loaded:         .ASCIZ  /Overlay loaded./
                .EVEN
TapeParams:     .BYTE   3,0
                .WORD   LoadAddr,0
1$:             .ASCII  /ovr2-SUBS.v/        ; Имя файла оверлея
                .BLKB   ^D16-<.-1$>
                .BLKB   ^D16+4

;; Пустая секция, только задается начальное смещение
;; В других .asm файлах секции с именем SUBS будут дописываться
```

```
;; сюда
                .PSECT  SUBS
.=.+LoadAddr
                .END    Start

```

Второй файл:

```
;; vim: set expandtab ai textwidth=80:
                .TITLE  PART2
                .IDENT  /V01.00/
                .INCLUDE lib/bk11m/bk11m.inc

;; ============================================
;;  Оверлей.
;;  Содержит подпрограмму SayHi, которая
;;  выводит на экран строчку.
;; ============================================
                .PSECT  SUBS
SayHi::         .BPRIN  #HiStr
                rts     PC
HiStr:          .ASCIZ  /I'm overlay!/
                .END
```

Лог линковки:

```
Module:OVR2
  Ident: V01.00
Module:PART2
  Ident: V01.00
=Global Definitions:
SAYHI : SUBS  /40000
=Sections:
. ABS., addr: 0x8006da000, len: 0, min addr: 37777777777, current start: 0
      , addr: 0x8006eb000, len: 0, min addr: 37777777777, current start: 0
MAIN  , addr: 0x8006fc000, len: 1342, min addr: 1000, current start: 0
SUBS  , addr: 0x80070d000, len: 40027, min addr: 40000, current start: 40000
```

8. Общая схема программы.

```
⟨ Включение заголовочных файлов 126 ⟩
⟨ Директивы препроцессора ⟩
⟨ Константы 119 ⟩
⟨ Собственные типы данных 16 ⟩
⟨ Глобальные переменные 9 ⟩
int main(int argc, char *argv[])
{
  ⟨ Данные программы 10 ⟩
  const char *objname;
  int i, j, not_resolved;
  ⟨ Разобрать командную строку 125 ⟩
  ⟨ Инициализация каталога секций 42 ⟩
  ⟨ Инициализация таблицы глобальных символов 28 ⟩
  ⟨ Инициализация списка ссылок без констант 56 ⟩
  ⟨ Инициализация списка сложных выражений 99 ⟩
  ⟨ Инициализация списка пределов 70 ⟩
    /* Поочередно обрабатываем все заданные объектные файлы */
  cur_input = 0;
  not_resolved = 1;
  num_start_addresses = 0;
  while ((objname = config.objnames[cur_input]) ≠ ~) {
    ⟨ Открыть объектный файл 11 ⟩
    handleOneFile(fobj);      /* Разрешаем глобальные ссылки */
    not_resolved = resolveGlobals();      /* Разрешаем сложные ссылки */
    not_resolved += resolveComplex();
    fclose(fobj);
    ++cur_input;
    if (num_start_addresses ≥ 2) {
      PRINTERR("Too␣many␣start␣addresses.\n");
      return (1);
    }
  }
  if (not_resolved ≡ 0) {
    ⟨ Вывод таблицы глобальных символов 33 ⟩
    ⟨ Заполнить пределы секций 72 ⟩
    ⟨ Создачм файл результата 13 ⟩
  }
  else {
    ⟨ Вывод неразрешенных ссылок 12 ⟩
  }
  ⟨ Очистка каталога секций 44 ⟩
  ⟨ Освободить список сложных выражений 100 ⟩
  ⟨ Освободить список ссылок 57 ⟩
  ⟨ Освободить список пределов 71 ⟩
  return (0);
}
```

9. Номер текущего обрабатываемого объектного файла.

```
⟨ Глобальные переменные 9 ⟩ ≡
  static int cur_input;
  static int num_start_addresses;
```

Смотри также секции 18, 27, 31, 35, 37, 47, 52, 58, 65, 68, 92, 97, 102, 108, 114, 117, 120, 121, 123, и 127.

Этот код используется в секции 8.

10.

```
⟨ Данные программы 10 ⟩ ≡
  FILE *fobj, *fresult;
  char ovrname[200];
```

Смотри также секции 32 и 43.

Этот код используется в секции 8.

11.

```
⟨ Открыть объектный файл 11 ⟩ ≡
  fobj = fopen(objname, "r");
  if (fobj ≡ ˜) {
    PRINTERR("Can't␣open␣%s\n", objname);
    return (ERR_CANTOPEN);
  }
```

Этот код используется в секции 8.

12.

```
⟨ Вывод неразрешенных ссылок 12 ⟩ ≡
  if (¬simpleRefIsEmpty()) {
    printf("Unresolved␣simple␣refs:\n");
    for (i = SRefList.pool[0].link; i ≠ 0; i = SRefList.pool[i].link) {
      fromRadix50(SRefList.pool[i].name[0], name);
      fromRadix50(SRefList.pool[i].name[1], name + 3);
      fromRadix50(SectDir[SRefList.pool[i].sect].name[0], sect_name);
      fromRadix50(SectDir[SRefList.pool[i].sect].name[1], sect_name + 3);
      printf("i:␣%4d,␣name:␣%s,␣disp:␣%s/%o,␣file:␣%s\n", i, name, sect_name, SRefList.pool[i].disp,
          config.objnames[SRefList.pool[i].obj_file]);
    }
  }
  if (¬complexRefIsEmpty()) {
    printf("Unresolved␣complex␣refs:\n");
    for (i = CExprList.pool[0].link; i ≠ 0; i = CExprList.pool[i].link) {
      for (j = 0; j < CExprList.pool[i].NumTerms; ++j) {
        if (CExprList.pool[i].terms[j].code ≡ CREL_OP_FETCH_GLOBAL) {
          fromRadix50(CExprList.pool[i].terms[j].un.name[0], name);
          fromRadix50(CExprList.pool[i].terms[j].un.name[1], name + 3);
          printf("i:␣%4d,␣j:␣%2d,␣name:␣%s,␣file:""␣%s\n", i, j, name,
              config.objnames[CExprList.pool[i].obj_file]);
        }
      }
    }
  }
```

Этот код используется в секции 8.

13. По заданному в командной строке имени создаҷтся файл с программной секцией, для которой указан адрес запуска. Остальные секции ненулевой длины планируется писать в дополнительные файлы (оверлеи).

```
⟨ Создаҷм файл результата 13 ⟩ ≡
  for (i = 0; i < NumSections; ++i) {
    if (SectDir[i].transfer_addr ≠ 1 ∧ SectDir[i].len ≠ 0) {      /* Основной файл */
      fresult = fopen(config.output_filename, "w");
      if (fresult ≡ ~) {
        PRINTERR("Can't␣create␣%s\n", config.output_filename);
        return (ERR_CANTCREATE);
      }
      fwrite(SectDir[i].text + SectDir[i].min_addr, SectDir[i].len − SectDir[i].min_addr, 1, fresult);
      fclose(fresult);
      continue;
    }
    if (SectDir[i].len ≠ 0 ∧ SectDir[i].min_addr ≠ −1) {
      fromRadix50(SectDir[i].name[0], sect_name);
      fromRadix50(SectDir[i].name[1], sect_name + 3);      /* Оверлеи */
      for (j = 5; j ≥ 0; −−j) {
        if (sect_name[j] ≠ '␣') {
          sect_name[j + 1] = 0;
          break;
        }
      }
      strncpy(ovrname, config.output_filename, config.max_filename_len − strlen(sect_name) − 3);
      ovrname[config.max_filename_len − strlen(sect_name) − 3] = '\0';
      strcat(ovrname, "-");
      strcat(ovrname, sect_name);
      strcat(ovrname, ".v");
      fresult = fopen(ovrname, "w");
      if (fresult ≡ ~) {
        PRINTERR("Can't␣create␣%s\n", ovrname);
        return (ERR_CANTCREATE);
      }
      fwrite(SectDir[i].text + SectDir[i].min_addr, SectDir[i].len − SectDir[i].min_addr, 1, fresult);
      fclose(fresult);
    }
  }
```

Этот код используется в секции 8.

14. Обработка объектного файла.

15. Структура объектного файла. Объектный файл состоит из блоков, которые начинаются заголовком *BinaryBlock*, собственно данных длиной $len - 4$ и байта контрольной суммы (0 - сумма всех байт). Между блоками может быть произвольное количество нулевых байт.

16. ⟨ Собственные типы данных 16 ⟩ ≡

```
typedef struct _BinaryBlock {
  uint8_t one;      /* must be 1 */
  uint8_t zero;     /* must be 0 */
  uint16_t len;     /* length of block */
} BinaryBlock;
```

Смотри также секции 22, 26, 34, 51, 67, 76, 91, 96, и 122.

Этот код используется в секции 8.

17. Обработать один объектный файл.

```
static void handleOneFile(FILE *fobj)
{
  BinaryBlock obj_header;
  int first_byte, i;
  unsigned int block_len;
  char name[7];
  ⟨ Сбросить перекодировку секций 93 ⟩
  while (¬feof(fobj)) {      /* Ищем начало блока */
    do {
      first_byte = fgetc(fobj);
      if (first_byte ≡ EOF) goto end;
    } while (first_byte ≠ 1);      /* Читаем заголовок */
    ungetc(first_byte, fobj);
    if (fread(&obj_header, sizeof(BinaryBlock), 1, fobj) ≠ 1) {
      PRINTERR("IO␣error:␣%s\n", config.objnames[cur_input]);
      break;
    }
    if (obj_header.zero ≠ 0) continue;
    block_len = obj_header.len − 4;
    PRINTVERB(2, "Binary␣block␣found.␣Length:%o\n", block_len);
      /* Читаем тело блока с контрольной суммой */
    if (fread(block_body, block_len + 1, 1, fobj) ≠ 1) {
      PRINTERR("IO␣error:␣%s\n", config.objnames[cur_input]);
      break;
    }
    ⟨ Обработать блок 19 ⟩
  }
end: ;
}
```

18. Буффер для тела блока.

```
⟨ Глобальные переменные 9 ⟩ +≡
  static uint8_t block_body[65536 + 1];
```

19. Обработка одного бинарного блока. По первому байту блока выясняем его тип.

```
⟨ Обработать блок 19 ⟩ ≡
  PRINTVERB(2, "␣␣Block␣type:␣%o,␣", block_body[0]);
  switch (block_body[0]) {
  case 1: PRINTVERB(2, "GSD\n");
    ⟨ Разобрать GSD 21 ⟩
    break;
  case 2: PRINTVERB(2, "ENDGSD\n");
    ⟨ Вывести перекодировку секций 95 ⟩
    break;
  case 3: PRINTVERB(2, "TXT\n");
    ⟨ Обработать секцию TXT 112 ⟩
    break;
  case 4: PRINTVERB(2, "RLD\n");
    ⟨ Обработать секцию перемещений 113 ⟩
    break;
  case 5: PRINTVERB(2, "ISD\n");
    break;
  case 6: PRINTVERB(2, "ENDMOD\n");
    break;
  case 7: PRINTVERB(2, "Librarian␣header\n");
    break;
  case 8: PRINTVERB(2, "Librarian␣end\n");
    break;
  default: PRINTERR("Bad␣block␣type:␣%o␣:␣%s\n", block_body[0], config.objnames[cur_input]);
  }
```

Этот код используется в секции 17.

20. GSD.

21. Разбор блока GSD — Global Symbol Directory (каталог глобальных символов). Он содержит всю информацию, необходимую линковщику для присваивания адресов глобальным символам и выделения памяти. Каталог состоит из 8-ми байтовых записей следующих типов:

```
#define GSD_MODULE_NAME 0
#define GSD_CSECT_NAME 1
#define GSD_INTERNAL_SYMBOL_NAME 2
#define GSD_TRANFER_ADDRESS 3
#define GSD_GLOBAL_SYMBOL_NAME 4
#define GSD_PSECT_NAME 5
#define GDS_IDENT 6
#define GSD_MAPPED_ARRAY 7
```

⟨ Разобрать GSD 21 ⟩ ≡

```
handleGSD(block_len);
```

Этот код используется в секции 19.

22. ⟨ Собственные типы данных 16 ⟩ +≡

```
typedef struct _GSD_Entry {
  uint16_t name[2];
  uint8_t flags;
  uint8_t type;
  uint16_t value;
} GSD_Entry;
```

23. **static void** *handleGSD*(**int** *len*)

```
{
  int i, sect;
  GSD_Entry *entry;
  char name[7];
  for (i = 2; i < len; i += 8) {
    entry = (GSD_Entry *)(block_body + i);
    ⟨ Распаковать имя 24 ⟩
    PRINTVERB(2, "␣␣␣␣Entry␣name:␣'%s',␣type:␣%o␣---␣", name, entry→type);
    switch (entry→type) {
    case GSD_MODULE_NAME:      /* Просто имя модуля. */
      PRINTVERB(2, "ModuleName.\n");
      PRINTVERB(1, "Module:%s\n", name);
      break;
    case GSD_CSECT_NAME:      /* Имя управляющей секции */
      PRINTVERB(2, "CSectName,␣flags:%o,␣length:%o.\n", entry→flags, entry→value);
      break;
    case GSD_INTERNAL_SYMBOL_NAME:      /* Имя внутреннего символа */
      PRINTVERB(2, "InternalSymbolName\n");
      break;
    case GSD_TRANFER_ADDRESS:      /* Адрес запуска программы */
      PRINTVERB(2, "TransferAddress,␣offset:%o.\n", entry→value);
      ⟨ Установить адрес запуска 40 ⟩
      break;
    case GSD_GLOBAL_SYMBOL_NAME:      /* Определение/ссылка на глобальный адрес */
      PRINTVERB(2, "GlobalSymbolName,␣flags:%o,␣value:%o.\n", entry→flags, entry→value);
      ⟨ Обработать глобальные символы и ссылки 110 ⟩
      break;
    case GSD_PSECT_NAME:      /* Имя программной секции */
      PRINTVERB(2, "PSectName,␣flags:%o,␣max␣length:%o.\n", entry→flags, entry→value);
      ⟨ Обработать программную секцию 111 ⟩
      break;
    case GDS_IDENT:      /* Версия модуля */
      PRINTVERB(2, "Ident.\n");
      PRINTVERB(1, "␣␣Ident:␣%s\n", name);
      break;
    case GSD_MAPPED_ARRAY:      /* Массив */
      PRINTVERB(2, "MappedArray,␣length:%o.\n", entry→value);
      break;
    default: PRINTERR("Bad␣entry␣type:␣%o␣:␣%s\n", entry→type, config.objnames[cur_input]);
    }
  }
}
```

24. ⟨ Распаковать имя 24 ⟩ ≡

```
fromRadix50(entry→name[0], name);
fromRadix50(entry→name[1], name + 3);
```

Этот код используется в секции 23.

25. Разбор определения/ссылки на глобальный символ.

26. Таблица глобальных символов. *addr* содержит уже смещенный адрес относительно 0.

```
#define MAX_GLOBALS 1024
⟨ Собственные типы данных 16 ⟩ +≡
  typedef struct _GSymDefEntry {
    uint16_t name[2];
    uint8_t flags;
    uint8_t sect;      /* Номер секции, в которой определен глобальный символ */
    uint16_t addr;     /* Адрес символа в секции */
    uint8_t obj_file;  /* Файл, где определен символ */
  } GSymDefEntry;
```

27.
```
⟨ Глобальные переменные 9 ⟩ +≡
  static GSymDefEntry GSymDef[MAX_GLOBALS];
  static int NumGlobalDefs;
```

28.
```
⟨ Инициализация таблицы глобальных символов 28 ⟩ ≡
  NumGlobalDefs = 0;
```

Этот код используется в секции 8.

29.

```
#define GLOBAL_WEAK_MASK °01      /* 00000001b */
#define GLOBAL_DEFINITION_MASK °10      /* 00001000b */
#define GLOBAL_RELOCATION_MASK °40      /* 00100000b */
  static void handleGlobalSymbol(GSD_Entry *entry)
  {
    char name[7];
    int found_sym;
    if (entry→flags & GLOBAL_DEFINITION_MASK) {
      /* Повторное определение глобального символа */
      if ((found_sym = findGlobalSym(entry→name)) ≠ -1) {
        fromRadix50(entry→name[0], name);
        fromRadix50(entry→name[1], name + 3);
        PRINTERR("Global␣definition␣conflict:␣%s␣in␣%s""␣conflicts␣with␣%s.\n", name,
            config.objnames[cur_input], config.objnames[found_sym]);
        exit(EXIT_FAILURE);
      }
      GSymDef[NumGlobalDefs].name[0] = entry→name[0];
      GSymDef[NumGlobalDefs].name[1] = entry→name[1];
      GSymDef[NumGlobalDefs].flags = entry→flags;
      GSymDef[NumGlobalDefs].sect = CurSect;
      GSymDef[NumGlobalDefs].addr = SectDir[CurSect].start + entry→value;
      GSymDef[NumGlobalDefs].obj_file = cur_input;
      ++NumGlobalDefs;
    }
    if (config.verbosity ≥ 2) {
      PRINTVERB(2, "␣␣␣␣␣␣␣␣Flags:␣");
      if (entry→flags & GLOBAL_WEAK_MASK) {
        PRINTVERB(2, "Weak,");
      }
      else {
        PRINTVERB(2, "Strong,");
      }
      if (entry→flags & GLOBAL_DEFINITION_MASK) {
        PRINTVERB(2, "Definition,");
      }
      else {
        PRINTVERB(2, "Reference,");
      }
      if (entry→flags & GLOBAL_WEAK_MASK) {
        PRINTVERB(2, "Relative.\n");
      }
      else {
        PRINTVERB(2, "Absolute.\n");
      }
    }
  }
```

30. Найти символ в таблице. -1 — символ не найден.

```
static int findGlobalSym(uint16_t * name)
{
  int found, i;
  found = -1;
  for (i = 0; i < NumGlobalDefs; ++i) {
    if (name[0] ≡ GSymDef[i].name[0] ∧ name[1] ≡ GSymDef[i].name[1]) {
      found = i;
      break;
    }
  }
  return (found);
}
```

31. ⟨ Глобальные переменные 9 ⟩ +≡

```
static int findGlobalSym ( uint16_t * ) ;
```

32. ⟨ Данные программы 10 ⟩ +≡

```
char name[7];
```

33. ⟨ Вывод таблицы глобальных символов 33 ⟩ ≡

```
if (config.verbosity ≥ 1) {
  PRINTVERB(1, "=Global␣Definitions:\n");
  for (i = 0; i < NumGlobalDefs; ++i) {
    fromRadix50(GSymDef[i].name[0], name);
    fromRadix50(GSymDef[i].name[1], name + 3);
    fromRadix50(SectDir[GSymDef[i].sect].name[0], sect_name);
    fromRadix50(SectDir[GSymDef[i].sect].name[1], sect_name + 3);
    PRINTVERB(1, "%s:␣%s/%o\n", name, sect_name, GSymDef[i].addr);
  }
}
```

Этот код используется в секции 8.

34. Разбор программной секции. Данные о секциях хранятся в каталоге секций.

```
#define MAX_PROG_SECTIONS 254
```

⟨ Собственные типы данных 16 ⟩ +≡

```
typedef struct _SectionDirEntry {
  uint16_t name[2];      /* Имя в Radix50 */
  uint8_t flags;      /* Флаги секции */
  uint16_t start;      /* Смещение секции для текущего модуля */
  int32_t min_addr;      /* Минимальный адрес, с которого расположены данные */
  uint16_t len;      /* Длина секции */
  uint16_t transfer_addr;      /* Адрес старта (1 — секция не стартовая) */
  uint16_t last_load_addr;      /* Адрес последнего загруженного блока TEXT */
  uint8_t * text;      /* Адрес блока памяти для текста секции */
} SectionDirEntry;
```

35. ⟨ Глобальные переменные 9 ⟩ +≡

```
static SectionDirEntry SectDir[MAX_PROG_SECTIONS];
static int NumSections;
```

36.

```
#define PSECT_SAVE_MASK °001      /* 00000001b */
#define PSECT_ALLOCATION_MASK °004      /* 00000100b */
#define PSECT_ACCESS_MASK °020      /* 00010000b */
#define PSECT_RELOCATION_MASK °040      /* 00100000b */
#define PSECT_SCOPE_MASK °100      /* 01000000b */
#define PSECT_TYPE_MASK °200      /* 10000000b */
  static void handleProgramSection(GSD_Entry *entry)
  {
    ⟨ Вывести отладочную информацию по секциям 48 ⟩
    CurSect = findSection(entry→name);
    if (CurSect ≡ −1) {
      ⟨ Добавить программную секцию 46 ⟩
    }
    else {      /* Изменить смещение секции в модуле */
      SectDir[CurSect].start += SectDir[CurSect].len;
      SectDir[CurSect].len += entry→value;
    }
    ⟨ Добавить перекодировку секции 94 ⟩
  }
```

37. ⟨ Глобальные переменные 9 ⟩ +≡

```
  static int CurSect;
```

38. Переключается текущая секция.

```
⟨ Установка текущей секции и позиции 38 ⟩ ≡
  const_entry = ( RLD_Const_Entry * ) entry;
  fromRadix50(entry→value[0], gname);
  fromRadix50(entry→value[1], gname + 3);
  PRINTVERB(2, "␣␣␣␣␣␣Name:␣%s,␣+Const:␣%o.\n", gname, const_entry→constant);
  CurSect = findSection(entry→value);
  if (SectDir[CurSect].min_addr ≡ −1 ∨ SectDir[CurSect].min_addr >
        (const_entry→constant + SectDir[CurSect].start)) {
    SectDir[CurSect].min_addr = const_entry→constant + SectDir[CurSect].start;
  }
  RLD_i += 8;
```

Этот код используется в секции 77.

39. Адрес запуска, равный единице игнорируется.

40. ⟨ Установить адрес запуска 40 ⟩ ≡

```
  sect = findSection(entry→name);
  SectDir[sect].transfer_addr = entry→value;
  if (entry→value ≠ 1) ++num_start_addresses;
```

Этот код используется в секции 23.

41. Обработать секцию `TEXT`. Содержимое добавляется к текущей секции *CurSect*. Поскольку секции `TEXT` могут следовать друг за другом, и лишь к последней из них применяется секция настройки адресов, то запоминаем адрес, с которого была загружена последняя секция `TEXT`.

```
static void handleTextSection(uint8_t * block, unsigned int len)
{
  uint16_t addr;
  addr = block[2] + block[3] * 256;
  PRINTVERB(2, "␣␣Load␣address:␣%o,␣Current␣section:␣%d.\n", addr, CurSect);
  memcpy(SectDir[CurSect].text + SectDir[CurSect].start + addr, block + 4, len - 4);
  SectDir[CurSect].last_load_addr = SectDir[CurSect].start + addr;
}
```

42.

⟨ Инициализация каталога секций 42 ⟩ ≡

```
NumSections = 0;
memset(SectDir, 0, sizeof (SectDir));
```

Этот код используется в секции 8.

43. ⟨ Данные программы 10 ⟩ +≡

```
char sect_name[7];
```

44. ⟨ Очистка каталога секций 44 ⟩ ≡

```
PRINTVERB(1, "=Sections:\n");
for (i = 0; i < NumSections; ++i) {
  fromRadix50(SectDir[i].name[0], sect_name);
  fromRadix50(SectDir[i].name[1], sect_name + 3);
  PRINTVERB(1, "%s,␣addr:␣%p,␣len:␣%o,␣min␣addr:␣%o,""␣current␣start:␣%o\n", sect_name,
      SectDir[i].text, SectDir[i].len, SectDir[i].min_addr, SectDir[i].start);
  if (SectDir[i].text ≠ ~) free(SectDir[i].text);
}
```

Этот код используется в секции 8.

45. Найти программную секцию по имени.

```
static int findSection(uint16_t * name)
{
  int found, i;
  found = −1;
  for (i = 0; i < NumSections; ++i) {
    if (SectDir[i].name[0] ≡ name[0] ∧ SectDir[i].name[1] ≡ name[1]) {
      found = i;
      break;
    }
  }
  return (found);
}
```

46. Память выделяется под все секции, даже те, которые имеют нулевую длину.

```
#define DEFAULT_SECTION_LEN 65536
```

⟨ Добавить программную секцию 46 ⟩ ≡

```
SectDir[NumSections].name[0] = entry¬name[0];
SectDir[NumSections].name[1] = entry¬name[1];
SectDir[NumSections].flags = entry¬flags;
SectDir[NumSections].len = entry¬value;
  /* Если секции при слиянии выравниваются на слово, то изменить длину */
if (¬(entry¬flags & PSECT_TYPE_MASK)) {
  if (SectDir[NumSections].len & 1) ++SectDir[NumSections].len;
}
SectDir[NumSections].min_addr = −1;
SectDir[NumSections].transfer_addr = 1; SectDir[NumSections].text = ( uint8_t * )
    calloc(1, DEFAULT_SECTION_LEN);
CurSect = NumSections;
++NumSections;
```

Этот код используется в секции 36.

47. ⟨ Глобальные переменные 9 ⟩ +≡

```
static int findSection ( uint16_t * ) ;
```

48. ⟨ Вывести отладочную информацию по секциям 48 ⟩ ≡

```
if (config.verbosity ≥ 2) {
  PRINTVERB(2, "␣␣␣␣␣␣␣␣Flags:␣");
  if (entry→flags & PSECT_SAVE_MASK) {
    PRINTVERB(2, "RootScope,");
  }
  else {
    PRINTVERB(2, "NonRootScope,");
  }
  if (entry→flags & PSECT_ALLOCATION_MASK) {
    PRINTVERB(2, "Overlay,");
  }
  else {
    PRINTVERB(2, "Concatenate,");
  }
  if (entry→flags & PSECT_ACCESS_MASK) {
    PRINTVERB(2, "ReadOnly,");
  }
  else {
    PRINTVERB(2, "ReadWrite,");
  }
  if (entry→flags & PSECT_RELOCATION_MASK) {
    PRINTVERB(2, "Relocable,");
  }
  else {
    PRINTVERB(2, "Absolute,");
  }
  if (entry→flags & PSECT_SCOPE_MASK) {
    PRINTVERB(2, "Global,");
  }
  else {
    PRINTVERB(2, "Local,");
  }
  if (entry→flags & PSECT_TYPE_MASK) {
    PRINTVERB(2, "Dref.\n");
  }
  else {
    PRINTVERB(2, "Iref.\n");
  }
}
```

Этот код используется в секции 36.

49. Списки ссылок на глобальные символы.

50. Есть три вида ссылок на глобальные символы: без добавления константы, с добавлением константы и сложная ссылка. Первые два вида имеют фиксированный (хотя и разный) размер, а третья — произвольный размер.

Кусочки маленькие, поэтому попробуем не дергать операционку для выделения памяти, а используем линейные списки с хранением в массиве.

51. Структура элемента списка для хранения ссылок.

```
#define INITIAL_SIMPLE_REF_LIST_SIZE 100
⟨ Собственные типы данных 16 ⟩ +≡
  typedef struct _SimpleRefEntry {
    uint16_t link;      /* Поле связи */
    uint8_t type;
    uint8_t sect;      /* Номер секции */
    uint16_t disp;      /* Смещение в секции уже учитывающее адрес самой секции */
    uint16_t constant;
    uint16_t name[2];
    uint8_t obj_file;      /* Номер входного файла */
  } SimpleRefEntry;
  typedef struct _SimpleRefList {
    uint16_t avail;      /* Начало списка свободных блоков */
    uint16_t poolmin;      /* Номер элемента — нижней границы пула */
    SimpleRefEntry *pool;      /* Массив для хранения списка */
    int num_allocations;      /* Счетчик выделений памяти при нехватке начального пула*/
  } SimpleRefList;
```

52. ⟨ Глобальные переменные 9 ⟩ +≡

```
  static SimpleRefList SRefList;
  static int simpleRefIsEmpty(void);
```

53.

```
  static int simpleRefIsEmpty(void)
  {
    return (SRefList.pool[0].link ≡ 0);
  }
```

54. Добавляем новую ссылку в список

```
static void addSimpleRef(RLD_Entry * ref){
        SimpleRefEntry *new_entry;
    SimpleRefEntry *new_memory;
    uint16_t new_index;      /* Если не хватило начального размера пула */
    if (SRefList.poolmin ≡ INITIAL_SIMPLE_REF_LIST_SIZE * SRefList.num_allocations) {
      ++SRefList.num_allocations;
      new_memory = (SimpleRefEntry *) realloc(SRefList.pool,
          sizeof(SimpleRefEntry) * INITIAL_SIMPLE_REF_LIST_SIZE * SRefList.num_allocations);
      if (new_memory ≡ ~) {
        PRINTERR("No memory for simple ref list");
        abort( );
      }
      PRINTVERB(2, "Done SRefList allocation:%d\n", SRefList.num_allocations);
      SRefList.pool = new_memory;
    }     /* Если есть свободные блоки */
    if (SRefList.avail ≠ 0) {
      new_index = SRefList.avail;
      SRefList.avail = SRefList.pool[SRefList.avail].link;
    }
    else {      /* Свободных блоков нет, используем пул */
      new_index = SRefList.poolmin;
      ++SRefList.poolmin;
    }
    new_entry = SRefList.pool + new_index;
    new_entry→link = SRefList.pool[0].link;
    SRefList.pool[0].link = new_index;      /* Собственно данные ссылки */
    new_entry→obj_file = cur_input;
    new_entry→name[0] = ref→value[0];
    new_entry→name[1] = ref→value[1];
    new_entry→disp = ref→disp − 4 + SectDir[CurSect].last_load_addr;
    new_entry→sect = CurSect;
    new_entry→type = ref→cmd.type; if (new_entry→type ≡
        RLD_CMD_GLOBAL_ADDITIVE_DISPLACED_RELOCATION ∨ new_entry→type ≡
        RLD_CMD_GLOBAL_ADDITIVE_RELOCATION) { new_entry→constant = ( ( RLD_Const_Entry * ) ref
        ) → constant; } }
```

55. Удаляем ссылку из списка. Возвращает поле связи удалямого элемента. Задача вызывающей функции: записать это значение в поле связи предыдущего элемента.

```
static uint16_t delSimpleRef(uint16_t ref_i)
{
  uint16_t link;
  link = SRefList.pool[ref_i].link;
  SRefList.pool[ref_i].link = SRefList.avail;
  SRefList.avail = ref_i;
  return (link);
}
```

56. *poolmin* устанавливаем равным 1, так как для данной системы хранения ссылок нулевой элемент пула не используется, а его номер считается чем-то вроде NULL.

```
⟨ Инициализация списка ссылок без констант 56 ⟩ ≡
  SRefList.pool = (SimpleRefEntry *) malloc(sizeof(SimpleRefEntry) *
      INITIAL_SIMPLE_REF_LIST_SIZE);
  SRefList.num_allocations = 1;
  SRefList.pool[0].link = 0;
  SRefList.avail = 0;
  SRefList.poolmin = 1;
```

Этот код используется в секции 8.

57.

```
⟨ Освободить список ссылок 57 ⟩ ≡
  if (config.verbosity ≥ 2) {
    PRINTVERB(2, "=Simple␣Refs:\n␣avail:␣%d,␣poolmin:␣%d\n", SRefList.avail, SRefList.poolmin);
    for (i = SRefList.pool[0].link; i ≠ 0; i = SRefList.pool[i].link) {
      fromRadix50(SRefList.pool[i].name[0], name);
      fromRadix50(SRefList.pool[i].name[1], name + 3);
      fromRadix50(SectDir[SRefList.pool[i].sect].name[0], sect_name);
      fromRadix50(SectDir[SRefList.pool[i].sect].name[1], sect_name + 3);
      PRINTVERB(2, "i:␣%4d,␣name:␣%s,␣disp:␣%s/%o,␣file:␣%s\n", i, name, sect_name,
          SRefList.pool[i].disp, config.objnames[SRefList.pool[i].obj_file]);
    }
  }
  free(SRefList.pool);
```

Этот код используется в секции 8.

58.

```
⟨ Глобальные переменные 9 ⟩ +≡
  static void addSimpleRef ( RLD_Entry * ) ;
  static uint16_t delSimpleRef(uint16_t);
```

59. Разрешение ссылок на глобальные символы.

60. Пробегаем набранные списки ссылок на глобальные символы и смотрим нет ли уже возможности разрешить ссылки. Возвращает 0, если неразрешенных ссылок нет.

```
static int resolveGlobals(void)
{
  uint16_t ref, prev_ref, *dest_addr;
  int global;
  prev_ref = 0;

  if (¬simpleRefIsEmpty()) {

    for (ref = SRefList.pool[0].link; ref ≠ 0; prev_ref = ref, ref = SRefList.pool[ref].link) {
      global = findGlobalSym(SRefList.pool[ref].name);
      if (global ≡ −1) {
        continue;
      }
      if (SRefList.pool[ref].type ≡ RLD_CMD_GLOBAL_RELOCATION) {
                /* Прямая ссылка */
      ⟨ Разрешить прямую ссылку 61 ⟩
          /* При удалении ref стоит вернуться на шаг назад */
        SRefList.pool[prev_ref].link = delSimpleRef(ref);
        ref = prev_ref;
        continue;
      }
      if (SRefList.pool[ref].type ≡ RLD_CMD_GLOBAL_DISPLACED_RELOCATION) {
                /* Косвенная ссылка */
      ⟨ Разрешить косвенную ссылку 63 ⟩
        SRefList.pool[prev_ref].link = delSimpleRef(ref);
          /* При удалении ref стоит вернуться на шаг назад */
        ref = prev_ref;
        continue;
      }
      if (SRefList.pool[ref].type ≡ RLD_CMD_GLOBAL_ADDITIVE_RELOCATION) {
                /* Прямая ссылка со смещением */
      ⟨ Разрешить смещенную прямую ссылку 62 ⟩
        SRefList.pool[prev_ref].link = delSimpleRef(ref);
          /* При удалении ref стоит вернуться на шаг назад */
        ref = prev_ref;
        continue;
      }
      if (SRefList.pool[ref].type ≡ RLD_CMD_GLOBAL_ADDITIVE_DISPLACED_RELOCATION) {
                /* Косвенная ссылка со смещением */
      ⟨ Разрешить смещенную косвенную ссылку 64 ⟩
        SRefList.pool[prev_ref].link = delSimpleRef(ref);
          /* При удалении ref стоит вернуться на шаг назад */
        ref = prev_ref;
        continue;
      }
    }
  }
  return (¬simpleRefIsEmpty());
}
```

61. Для разрешения прямой ссылки записываем адрес ссылки поле операнда.

⟨ Разрешить прямую ссылку 61 ⟩ ≡
```
dest_addr = ( uint16_t * ) (SectDir[SRefList.pool[ref].sect].text + SRefList.pool[ref].disp);
*dest_addr = GSymDef[global].addr;
```
Этот код используется в секции 60.

62. Для разрешения прямой ссылки со смещение записываем адрес ссылки + константа в поле операнда.

⟨ Разрешить смещенную прямую ссылку 62 ⟩ ≡
```
dest_addr = ( uint16_t * ) (SectDir[SRefList.pool[ref].sect].text + SRefList.pool[ref].disp);
*dest_addr = GSymDef[global].addr + SRefList.pool[ref].constant;
```
Этот код используется в секции 60.

63. Для разрешения косвенной ссылки записываем смещение от поля операнда в поле операнда.

⟨ Разрешить косвенную ссылку 63 ⟩ ≡
```
dest_addr = ( uint16_t * ) (SectDir[SRefList.pool[ref].sect].text + SRefList.pool[ref].disp);
*dest_addr = GSymDef[global].addr - (SRefList.pool[ref].disp + 2);
```
Этот код используется в секции 60.

64. Для разрешения косвенной ссылки со смещением записываем смещение от поля операнда + константа в поле операнда.

⟨ Разрешить смещенную косвенную ссылку 64 ⟩ ≡
```
dest_addr = ( uint16_t * ) (SectDir[SRefList.pool[ref].sect].text + SRefList.pool[ref].disp);
*dest_addr = GSymDef[global].addr - (SRefList.pool[ref].disp + 2) + SRefList.pool[ref].constant;
```
Этот код используется в секции 60.

65. ⟨ Глобальные переменные 9 ⟩ +≡
```
static int resolveGlobals(void);
```

66. Обработка пределов (. LIMIT) для секций.

67. Структура элемента списка для хранения ссылок на пределы.

```
#define INITIAL_LIMIT_LIST_SIZE 5
⟨ Собственные типы данных 16 ⟩ +≡
  typedef struct _LimListEntry {
    uint16_t link;      /* Поле связи */
    uint8_t sect;      /* Номер секции */
    uint16_t disp;      /* Смещение в секции уже учитывающее адрес самой секции */
  } LimListEntry;
  typedef struct _LimList {
    LimListEntry *pool;      /* Массив для хранения списка */
    int num_limits;
    int num_allocations;      /* Счетчик выделений памяти при нехватке начального пула */
  } LimList;
```

68. ⟨ Глобальные переменные 9 ⟩ +≡

```
  static LimList LimitList; static void addLimit ( RLD_Entry * ) ;
  static void resolveLimit(void);
```

69. Добавляем новую ссылку на предел в список

```
  static void addLimit(RLD_Entry * ref)
  {

    LimListEntry *new_entry;
    LimListEntry *new_memory;      /* Если не хватило начального размера пула */
    if (LimitList.num_limits ≡ INITIAL_LIMIT_LIST_SIZE * LimitList.num_allocations) {
      ++LimitList.num_allocations;
      new_memory = (LimListEntry *) realloc(LimitList.pool,
          sizeof(LimListEntry) * INITIAL_LIMIT_LIST_SIZE * LimitList.num_allocations);
      if (new_memory ≡ ~) {
        PRINTERR("No␣memory␣for␣limit␣list");
        abort( );
      }
      PRINTVERB(2, "Done␣LimitList␣allocation:%d\n", LimitList.num_allocations);
      LimitList.pool = new_memory;
    }
    new_entry = LimitList.pool + LimitList.num_limits;      /* Собственно данные ссылки */
    new_entry→disp = ref→disp − 4 + SectDir[CurSect].last_load_addr;
    new_entry→sect = CurSect;
    ++LimitList.num_limits;
  }
```

70. ⟨ Инициализация списка пределов 70 ⟩ ≡

```
  LimitList.pool = (LimListEntry *) malloc(sizeof(LimListEntry) * INITIAL_LIMIT_LIST_SIZE);
  LimitList.num_allocations = 1;
  LimitList.num_limits = 0;
```

Этот код используется в секции 8.

71. ⟨ Освободить список пределов 71 ⟩ ≡

```
if (config.verbosity ≥ 2) {
  PRINTVERB(2, "=Limit␣Refs:\n␣num_limits:␣%d\n", LimitList.num_limits);
  for (i = 0; i < LimitList.num_limits; ++i) {
    fromRadix50(SectDir[LimitList.pool[i].sect].name[0], sect_name);
    fromRadix50(SectDir[LimitList.pool[i].sect].name[1], sect_name + 3);
    PRINTVERB(2, "i:␣%4d,␣disp:␣%s/%o\n", i, sect_name, LimitList.pool[i].disp);
  }
}
free(LimitList.pool);
```

Этот код используется в секции 8.

72.

⟨ Заполнить пределы секций 72 ⟩ ≡

```
resolveLimit( );
```

Этот код используется в секции 8.

73.

```
static void resolveLimit(void){ int i;
    uint16_t * dest_dir; for (i = 0; i < LimitList.num_limits; ++i) { dest_dir = ( uint16_t * )
        (SectDir[LimitList.pool[i].sect].text + LimitList.pool[i].disp);
    dest_dir[0] = SectDir[LimitList.pool[i].sect].min_addr;
    dest_dir[1] = SectDir[LimitList.pool[i].sect].len; } }
```

74. Каталоги перемещений.

75. Блоки каталогов перемещений содержат информацию, которая нужна линковщику для корректировки ссылок в предыдущем блоке `TEXT`. Каждый модуль должеен иметь хотя бы один блок RLD, который расположен впереди всех блоков `TEXT`, его задача — описать текущую PSECT и еч размещение.

Каталог перемещений состоит из записей:

76. ⟨ Собственные типы данных 16 ⟩ +≡

```
typedef struct _RLD_Entry {
  struct {
  uint8_t type: 7;
  uint8_t b: 1;
  } cmd;
  uint8_t disp;
  uint16_t value[2];
} RLD_Entry;
typedef struct _RLD_Const_Entry {
  RLD_Entry ent;
  uint16_t constant;
} RLD_Const_Entry;
```

77. Поле *cmd.type* указывает

```
#define RLD_CMD_INTERNAL_RELOCATION °1
#define RLD_CMD_GLOBAL_RELOCATION °2
#define RLD_CMD_INTERNAL_DISPLACED_RELOCATION °3
#define RLD_CMD_GLOBAL_DISPLACED_RELOCATION °4
#define RLD_CMD_GLOBAL_ADDITIVE_RELOCATION °5
#define RLD_CMD_GLOBAL_ADDITIVE_DISPLACED_RELOCATION °6
#define RLD_CMD_LOCATION_COUNTER_DEFINITION °7
#define RLD_CMD_LOCATION_COUNTER_MODIFICATION °10
#define RLD_CMD_PROGRAM_LIMITS °11
#define RLD_CMD_PSECT_RELOCATION °12
#define RLD_CMD_PSECT_DISPLACED_RELOCATION °14
#define RLD_CMD_PSECT_ADDITIVE_RELOCATION °15
#define RLD_CMD_PSECT_ADDITIVE_DISPLACED_RELOCATION °16
#define RLD_CMD_COMPLEX_RELOCATION °17
  static void handleRelocationDirectory(uint8_t * block, int len)
  {
    RLD_Entry *entry;
    RLD_Const_Entry *const_entry;
    char gname[7];
    uint16_t * value, *dest_addr;
    int RLD_i, sect;
    for (RLD_i = 2; RLD_i < len; ) {
      entry = (RLD_Entry *)(block + RLD_i);
      PRINTVERB(2, "␣␣␣␣cmd:␣%o␣---␣", entry→cmd.type);
      switch (entry→cmd.type) {
      case RLD_CMD_INTERNAL_RELOCATION: PRINTVERB(2, "Internal␣Relocation.\n");
        ⟨ Прямая ссылка на абсолютный адрес 78 ⟩
        break;
      case RLD_CMD_GLOBAL_RELOCATION: PRINTVERB(2, "Global␣Relocation.\n");
        ⟨ Прямая ссылка на глобальный символ 80 ⟩
        break;
      case RLD_CMD_INTERNAL_DISPLACED_RELOCATION:
        PRINTVERB(2, "Internal␣Displaced␣Relocation.\n");
        ⟨ Косвенная ссылка на абсолютный адрес 79 ⟩
        break;
      case RLD_CMD_GLOBAL_DISPLACED_RELOCATION:
        PRINTVERB(2, "Global␣Displaced␣Relocation.\n");
        ⟨ Косвенная ссылка на глобальный символ 81 ⟩
        break;
      case RLD_CMD_GLOBAL_ADDITIVE_RELOCATION:
        PRINTVERB(2, "Global␣Additive␣Relocation.\n");
        ⟨ Прямая ссылка на смещенный глобальный символ 82 ⟩
        break;
      case RLD_CMD_GLOBAL_ADDITIVE_DISPLACED_RELOCATION:
        PRINTVERB(2, "Global␣Additive␣Displaced␣Relocation.\n");
        ⟨ Косвенная ссылка на смещенный глобальный символ 83 ⟩
        break;
      case RLD_CMD_LOCATION_COUNTER_DEFINITION:
        PRINTVERB(2, "Location␣Counter␣Definition.\n");
        ⟨ Установка текущей секции и позиции 38 ⟩
```

```
        break;
      case RLD_CMD_LOCATION_COUNTER_MODIFICATION:
        PRINTVERB(2, "Location␣Counter␣Modification.\n");
        ⟨ Изменение текущей позиции 84 ⟩
        break;
      case RLD_CMD_PROGRAM_LIMITS: PRINTVERB(2, "Program␣Limits.\n");
        ⟨ Установка пределов 85 ⟩
        break;
      case RLD_CMD_PSECT_RELOCATION: PRINTVERB(2, "PSect␣Relocation.\n");
        ⟨ Прямая ссылка на секцию 86 ⟩
        break;
      case RLD_CMD_PSECT_DISPLACED_RELOCATION:
        PRINTVERB(2, "PSect␣Displaced␣Relocation.\n");
        ⟨ Косвенная ссылка на секцию 87 ⟩
        break;
      case RLD_CMD_PSECT_ADDITIVE_RELOCATION: PRINTVERB(2, "PSect␣Additive␣Relocation.\n");
        ⟨ Прямая смещенная ссылка на секцию 88 ⟩
        break;
      case RLD_CMD_PSECT_ADDITIVE_DISPLACED_RELOCATION:
        PRINTVERB(2, "PSect␣Additive␣Displaced␣Relocation.\n");
        ⟨ Косвенная смещенная ссылка на секцию 89 ⟩
        break;
      case RLD_CMD_COMPLEX_RELOCATION: PRINTVERB(2, "Complex␣Relocation.\n");
        ⟨ Сложная ссылка 109 ⟩
        break;
      default:
        PRINTERR("Bad␣RLD␣entry␣type:␣%o␣:␣%s\n", entry→cmd.type, config.objnames[cur_input]);
        return;
      }
    }
  }
```

78.

⟨ Прямая ссылка на абсолютный адрес 78 ⟩ ≡

```
PRINTVERB(2, "␣␣␣␣␣␣Disp:␣%o,␣+Const:␣%o.\n", entry→disp, entry→value[0]); dest_addr = ( uint16_t *
    ) (SectDir[CurSect].text + SectDir[CurSect].last_load_addr + entry→disp − 4);
*dest_addr = SectDir[CurSect].start + entry→value[0];
RLD_i += 4;
```

Этот код используется в секции 77.

79.

⟨ Косвенная ссылка на абсолютный адрес 79 ⟩ ≡

```
PRINTVERB(2, "␣␣␣␣␣␣Disp:␣%o,␣+Const:␣%o.\n", entry→disp, entry→value[0]); dest_addr = ( uint16_t *
    ) (SectDir[CurSect].text + SectDir[CurSect].last_load_addr + entry→disp − 4);
*dest_addr = entry→value[0] − SectDir[CurSect].last_load_addr − entry→disp + 4 − 2;
RLD_i += 4;
```

Этот код используется в секции 77.

80.

⟨ Прямая ссылка на глобальный символ 80 ⟩ ≡
```
fromRadix50(entry→value[0], gname);
fromRadix50(entry→value[1], gname + 3);
PRINTVERB(2, "␣␣␣␣␣␣Disp:␣%o,␣Name:␣%s.\n", entry→disp, gname);
addSimpleRef(entry);
RLD_i += 6;
```
Этот код используется в секции 77.

81.

⟨ Косвенная ссылка на глобальный символ 81 ⟩ ≡
```
fromRadix50(entry→value[0], gname);
fromRadix50(entry→value[1], gname + 3);
PRINTVERB(2, "␣␣␣␣␣␣Disp:␣%o,␣Name:␣%s.\n", entry→disp, gname);
addSimpleRef(entry);
RLD_i += 6;
```
Этот код используется в секции 77.

82.

⟨ Прямая ссылка на смещенный глобальный символ 82 ⟩ ≡
```
const_entry = (RLD_Const_Entry *) entry;
fromRadix50(entry→value[0], gname);
fromRadix50(entry→value[1], gname + 3);
PRINTVERB(2, "␣␣␣␣␣␣Disp:␣%o,␣Name:␣%s,␣+Const:␣%o.\n", entry→disp, gname, const_entry→constant);
addSimpleRef(entry);
RLD_i += 8;
```
Этот код используется в секции 77.

83.

⟨ Косвенная ссылка на смещенный глобальный символ 83 ⟩ ≡
```
const_entry = (RLD_Const_Entry *) entry;
fromRadix50(entry→value[0], gname);
fromRadix50(entry→value[1], gname + 3);
PRINTVERB(2, "␣␣␣␣␣␣Disp:␣%o,␣Name:␣%s,␣+Const:␣%o.\n", entry→disp, gname, const_entry→constant);
addSimpleRef(entry);
RLD_i += 8;
```
Этот код используется в секции 77.

84. Не используется.

⟨ Изменение текущей позиции 84 ⟩ ≡
```
PRINTVERB(2, "␣␣␣␣␣␣+Const:␣%o.\n", entry→value[0]);
RLD_i += 4;
```
Этот код используется в секции 77.

85.

⟨ Установка пределов 85 ⟩ ≡
```
PRINTVERB(2, "␣␣␣␣␣␣Disp:␣%o.\n", entry→disp);
addLimit(entry);
RLD_i += 2;
```
Этот код используется в секции 77.

86.

⟨ Прямая ссылка на секцию 86 ⟩ ≡
```
fromRadix50(entry→value[0], gname);
fromRadix50(entry→value[1], gname + 3);
PRINTVERB(2, "␣␣␣␣␣␣␣Disp:␣%o,␣Name:␣%s.\n", entry→disp, gname);
sect = findSection(entry→value); dest_addr = ( uint16_t * )
    (SectDir[CurSect].text + SectDir[CurSect].last_load_addr + entry→disp − 4);
*dest_addr = SectDir[sect].start;
RLD_i += 6;
```
Этот код используется в секции 77.

87.

⟨ Косвенная ссылка на секцию 87 ⟩ ≡
```
fromRadix50(entry→value[0], gname);
fromRadix50(entry→value[1], gname + 3);
PRINTVERB(2, "␣␣␣␣␣␣␣Disp:␣%o,␣Name:␣%s.\n", entry→disp, gname);
sect = findSection(entry→value); dest_addr = ( uint16_t * )
    (SectDir[CurSect].text + SectDir[CurSect].last_load_addr + entry→disp − 4);
*dest_addr = SectDir[sect].start − SectDir[CurSect].last_load_addr − entry→disp + 4 − 2;
RLD_i += 6;
```
Этот код используется в секции 77.

88.

⟨ Прямая смещенная ссылка на секцию 88 ⟩ ≡
```
const_entry = (RLD_Const_Entry *) entry;
fromRadix50(entry→value[0], gname);
fromRadix50(entry→value[1], gname + 3);
PRINTVERB(2, "␣␣␣␣␣␣␣Name:␣%s,␣+Const:␣%o.\n", gname, const_entry→constant);
sect = findSection(entry→value); dest_addr = ( uint16_t * )
    (SectDir[CurSect].text + SectDir[CurSect].last_load_addr + entry→disp − 4);
*dest_addr = SectDir[sect].start + const_entry→constant;
RLD_i += 8;
```
Этот код используется в секции 77.

89.

⟨ Косвенная смещенная ссылка на секцию 89 ⟩ ≡
```
const_entry = (RLD_Const_Entry *) entry;
fromRadix50(entry→value[0], gname);
fromRadix50(entry→value[1], gname + 3);
PRINTVERB(2, "␣␣␣␣␣␣␣Name:␣%s,␣+Const:␣%o.\n", gname, const_entry→constant);
sect = findSection(entry→value); dest_addr = ( uint16_t * )
    (SectDir[CurSect].text + SectDir[CurSect].last_load_addr + entry→disp − 4);
*dest_addr = SectDir[sect].start − SectDir[CurSect].last_load_addr − entry→disp + 4 − 2 +
    const_entry→constant;
RLD_i += 8;
```
Этот код используется в секции 77.

90. Обработка сложных ссылок.

91. Для сложных ссылок нужно знать номера секций в текущем модуле.

```
⟨ Собственные типы данных 16 ⟩ +≡
  typedef struct _CurSectEntry {
    uint16_t name[2];
    uint8_t global_sect;
  } CurSectEntry;
```

92.
```
⟨ Глобальные переменные 9 ⟩ +≡
  static CurSectEntry curSections[MAX_PROG_SECTIONS];
  static int NumCurSections;
```

93.
```
⟨ Сбросить перекодировку секций 93 ⟩ ≡
  NumCurSections = 0;
```
Этот код используется в секции 17.

94.
```
⟨ Добавить перекодировку секции 94 ⟩ ≡
  curSections[NumCurSections].name[0] = SectDir[CurSect].name[0];
  curSections[NumCurSections].name[1] = SectDir[CurSect].name[1];
  curSections[NumCurSections++].global_sect = CurSect;
```
Этот код используется в секции 36.

95.
```
⟨ Вывести перекодировку секций 95 ⟩ ≡
  PRINTVERB(2, "=Sections recoding.\n");
  for (i = 0; i < NumCurSections; ++i) {
    fromRadix50(curSections[i].name[0], name);
    fromRadix50(curSections[i].name[1], name + 3);
    PRINTVERB(2, "sect: %3d, %s, global sect: %d\n", i, name, curSections[i].global_sect);
  }
```
Этот код используется в секции 19.

96. Список сложных ссылок. Каждая ссылка содержит простой массив термов сложного выражения. Максимальное количество термов в выражении ограничено согласно документации MACRO-11[1].

```
#define MAX_COMPLEX_TERMS 20
⟨ Собственные типы данных 16 ⟩ +≡
  typedef struct _ComplexTerm {
    uint8_t code;
    union {
      uint16_t name[2];
      struct {
        uint8_t sect;
        uint16_t disp;
      } inter;
      uint16_t constant;
    } un;
  } ComplexTerm;
  typedef struct _ComplexExprEntry {
    uint16_t link;      /* Поле связи */
    uint16_t disp;
    uint8_t sect;
    uint8_t obj_file;
    uint8_t result_type;
    uint8_t NumTerms;      /* Количество термов в выражении */
    ComplexTerm terms[MAX_COMPLEX_TERMS];
  } ComplexExprEntry;
  typedef struct _ComplexExpressionList {
    uint16_t avail;
    uint16_t poolmin;
    uint16_t num_allocations;
    ComplexExprEntry *pool;
  } ComplexExpressionList;
```

97.
```
⟨ Глобальные переменные 9 ⟩ +≡
  static ComplexExpressionList CExprList;
  static int complexRefIsEmpty(void);
  static void addComplexExpr(RLD_Entry *);
  static uint16_t delComplexExpr(uint16_t);
```

98.
```
#define INITIAL_COMPLEX_EXPR_LIST_SIZE 10
  static int complexRefIsEmpty(void)
  {
    return (CExprList.pool[0].link ≡ 0);
  }
```

[1] AA-KX10A-TC_PDP-11_MACRO-11_Reference_Manual_May88

99. ⟨ Инициализация списка сложных выражений 99 ⟩ ≡

```
CExprList.pool = (ComplexExprEntry *) malloc(sizeof(ComplexExprEntry) *
    INITIAL_COMPLEX_EXPR_LIST_SIZE);
CExprList.num_allocations = 1;
CExprList.pool[0].link = 0;
CExprList.avail = 0;
CExprList.poolmin = 1;
```

Этот код используется в секции 8.

100. ⟨ Освободить список сложных выражений 100 ⟩ ≡

```
if (config.verbosity ≥ 2) {
  PRINTVERB(2, "=Complex␣Refs:\n␣avail:␣%d,␣poolmin:␣%d\n", CExprList.avail,
      CExprList.poolmin);
  for (i = CExprList.pool[0].link; i ≠ 0; i = CExprList.pool[i].link) {
    fromRadix50(SectDir[CExprList.pool[i].sect].name[0], sect_name);
    fromRadix50(SectDir[CExprList.pool[i].sect].name[1], sect_name + 3);
    PRINTVERB(2, "i:␣%4d,␣disp:␣%s/%o,␣file:␣%s\n", i, sect_name, CExprList.pool[i].disp,
        config.objnames[CExprList.pool[i].obj_file]);
  }
}
free(CExprList.pool);
```

Этот код используется в секции 8.

101. Добавляем новое сложное выражение в список

```
static void addComplexExpr(RLD_Entry *ref)
{

  ComplexExprEntry *new_entry;
  ComplexExprEntry *new_memory;
  uint16_t new_index;      /* Если не хватило начального размера пула */
  if (CExprList.poolmin ≡ INITIAL_SIMPLE_REF_LIST_SIZE * CExprList.num_allocations) {
    ++CExprList.num_allocations;
    new_memory = (ComplexExprEntry *) realloc(CExprList.pool,
        sizeof(ComplexExprEntry)*INITIAL_SIMPLE_REF_LIST_SIZE* CExprList.num_allocations);
    if (new_memory ≡ ~) {
      PRINTERR("No␣memory␣for␣complex␣ref␣list");
      abort();
    }
    PRINTVERB(2, "Done␣CExprList␣allocation:%d\n", CExprList.num_allocations);
    CExprList.pool = new_memory;
  }     /* Если есть свободные блоки */
  if (CExprList.avail ≠ 0) {
    new_index = CExprList.avail;
    CExprList.avail = CExprList.pool[CExprList.avail].link;
  }
  else {      /* Свободных блоков нет, используем пул */
    new_index = CExprList.poolmin;
    ++CExprList.poolmin;
  }
  new_entry = CExprList.pool + new_index;
  new_entry→link = CExprList.pool[0].link;
  CExprList.pool[0].link = new_index;      /* Собственно данные ссылки */
  new_entry→obj_file = cur_input;
  new_entry→disp = ref→disp − 4 + SectDir[CurSect].last_load_addr;
  new_entry→sect = CurSect;
  CurComplexExpr = new_index;
}
```

102. ⟨ Глобальные переменные 9 ⟩ +≡

```
static uint16_t CurComplexExpr; static void addComplexTerm (uint8_t, uint16_t *,
    uint8_t, uint16_t, uint16_t );
```

103. Добавляем терм в текущее сложное выражение.

```
static void addComplexTerm(uint8_t code, uint16_t * name, uint8_t sect, uint16_t disp, uint16_t constant)
{
  ComplexTerm *term;
  term = CExprList.pool[CurComplexExpr].terms + (CExprList.pool[CurComplexExpr].NumTerms++);
  term->code = code;
  switch (code) {
  case CREL_OP_FETCH_GLOBAL: term->un.name[0] = name[0];
    term->un.name[1] = name[1];
    break;
  case CREL_OP_FETCH_RELOCABLE: term->un.inter.sect = sect;
    term->un.inter.disp = disp;
    break;
  case CREL_OP_FETCH_CONSTANT: term->un.constant = constant;
  default: ;
  }
}
```

104. Удаляем ссылку из списка. Возвращает поле связи удалямого элемента. Задача вызывающей функции: записать это значение в поле связи предыдущего элемента.

```
static uint16_t delComplexExpr(uint16_t ref_i)
{
  uint16_t link;
  link = CExprList.pool[ref_i].link;
  CExprList.pool[ref_i].link = CExprList.avail;
  CExprList.avail = ref_i;
  return (link);
}
```

105. Разрешение комплексных ссылок. Список содержит только те выражения, которые содержат неразрешенные ссылки. Возможно за один раз не получится разрешить все ссылки в выражении, но можно модифицировать выражение, заменяя ссылки, которые удалось разрешить, на константы. В следующий раз уже не придется заниматься поиском по имени.

```
static int resolveComplex(void){ ComplexExprEntry *entry;
    int prev, i;
    uint16_t value, *dest_addr;
    prev = 0; for (i = CExprList.pool[0].link; i ≠ 0; prev = i, i = CExprList.pool[i].link) {
        entry = CExprList.pool + i;      /* Пытаемся разрешить все ссылки внутри выражения */
    if (¬resolveTerms(entry)) {      /* Удалось разрешить все ссылки */
    value = calcTerms(entry);      /* В зависимости от типа записываем результат */
    if (entry->result_type ≡ CREL_OP_STORE_RESULT) {      /* Прямое обращение */
    dest_addr = ( uint16_t * ) (SectDir[entry->sect].text + +entry->disp);
    *dest_addr = value; } else {      /* Косвенное обращение */
    dest_addr = ( uint16_t * ) (SectDir[entry->sect].text + +entry->disp);
    *dest_addr = value − 2 − entry->disp; } CExprList.pool[prev].link = delComplexExpr(i);
    i = prev; } } return (¬complexRefIsEmpty( )); }
```

106. Попытка разрешить символы внутри одного выражения.

```
static int resolveTerms(ComplexExprEntry *entry)
{
  int i, not_resolved, global;
  uint16_t addr;
  not_resolved = 0;
  for (i = 0; i < entry->NumTerms; ++i) {
    switch (entry->terms[i].code) {
    case CREL_OP_FETCH_GLOBAL: global = findGlobalSym(entry->terms[i].un.name);
      if (global == -1) {
        ++not_resolved;
        break;
      }     /* Делаем из терма константу */
      entry->terms[i].code = CREL_OP_FETCH_CONSTANT;
      entry->terms[i].un.constant = GSymDef[global].addr;
      break;
    case CREL_OP_FETCH_RELOCABLE:     /* Перекодируем номер секции и вычисляем адрес */
      global = curSections[entry->terms[i].un.inter.sect].global_sect;
      addr = SectDir[global].start + entry->terms[i].un.inter.disp;
      entry->terms[i].un.constant = addr;
      entry->terms[i].code = CREL_OP_FETCH_CONSTANT;
      break;
    default: ;
    }
  }
  return (not_resolved);
}
```

107. Вычисление сложного выражения. Уже ничего не осталось, кроме констант и операций над ними, так что заводим стек на 20 позиций и подсчитываем. В самом элементе запоминаем какая команда была последней — по документации возможно использование как прямого, так и смещенного результата вычислений.

```
static uint16_t calcTerms(ComplexExprEntry *entry)
{
  uint16_t stack[MAX_COMPLEX_TERMS];
  uint16_t * sp;
  ComplexTerm *term;
  int i;
  sp = stack;
  for (i = 0; i < entry->NumTerms; ++i) {
    term = entry->terms + i;
    switch (term->code) {
    case CREL_OP_NONE: break;
    case CREL_OP_ADDITION: *(sp - 1) = *sp + *(sp - 1);
      --sp;
      break;
    case CREL_OP_SUBSTRACTION: *(sp - 1) = *(sp - 1) - *sp;
      --sp;
      break;
    case CREL_OP_MULTIPLICATION: *(sp - 1) = *sp * *(sp - 1);
      --sp;
      break;
    case CREL_OP_DIVISION: *(sp - 1) = *(sp - 1)/*sp;
      --sp;
      break;
    case CREL_OP_AND: *(sp - 1) = *sp & *(sp - 1);
      --sp;
      break;
    case CREL_OP_OR: *(sp - 1) = *sp | *(sp - 1);
      --sp;
      break;
    case CREL_OP_XOR: *(sp - 1) = *sp ⊕ *(sp - 1);
      --sp;
      break;
    case CREL_OP_NEG: *sp = 0 - *sp;
      break;
    case CREL_OP_COM: *sp = ~*sp;
      break;
    case CREL_OP_STORE_RESULT: case CREL_OP_STORE_RESULT_DISP: entry->result_type = term->code;
      break;
    case CREL_OP_FETCH_CONSTANT: *(++sp) = term->un.constant;
      break;
    default: PRINTERR("Bad␣term␣code:␣%o\n", term->code);
    }
  }
  return (*sp);
}
```

108. ⟨ Глобальные переменные 9 ⟩ +≡
```
static int resolveComplex(void);
static int resolveTerms(ComplexExprEntry *);
static uint16_t calcTerms(ComplexExprEntry *);
```

109.

```
#define CREL_OP_NONE °00
#define CREL_OP_ADDITION °01
#define CREL_OP_SUBSTRACTION °02
#define CREL_OP_MULTIPLICATION °03
#define CREL_OP_DIVISION °04
#define CREL_OP_AND °05
#define CREL_OP_OR °06
#define CREL_OP_XOR °07
#define CREL_OP_NEG °10
#define CREL_OP_COM °11
#define CREL_OP_STORE_RESULT °12
#define CREL_OP_STORE_RESULT_DISP °13
#define CREL_OP_FETCH_GLOBAL °16
#define CREL_OP_FETCH_RELOCABLE °17
#define CREL_OP_FETCH_CONSTANT °20
```

⟨ Сложная ссылка 109 ⟩ ≡

```
  addComplexExpr(entry);
  PRINTVERB(2, "␣␣␣␣␣␣Disp:␣%o.\n␣␣␣␣␣␣␣␣", entry→disp); for (RLD_i += 2;
      block[RLD_i] ≠ CREL_OP_STORE_RESULT ∧ block[RLD_i] ≠ CREL_OP_STORE_RESULT_DISP; ++RLD_i)
      { switch (block[RLD_i]) {
case CREL_OP_NONE: addComplexTerm(CREL_OP_NONE, ~, 0, 0, 0);
  break;
case CREL_OP_ADDITION: PRINTVERB(2, "+␣");
  addComplexTerm(CREL_OP_ADDITION, ~, 0, 0, 0);
  break;
case CREL_OP_SUBSTRACTION: PRINTVERB(2, "-␣");
  addComplexTerm(CREL_OP_SUBSTRACTION, ~, 0, 0, 0);
  break;
case CREL_OP_MULTIPLICATION: PRINTVERB(2, "*␣");
  addComplexTerm(CREL_OP_MULTIPLICATION, ~, 0, 0, 0);
  break;
case CREL_OP_DIVISION: PRINTVERB(2, "/␣");
  addComplexTerm(CREL_OP_DIVISION, ~, 0, 0, 0);
  break;
case CREL_OP_AND: PRINTVERB(2, "and␣");
  addComplexTerm(CREL_OP_AND, ~, 0, 0, 0);
  break;
case CREL_OP_OR: PRINTVERB(2, "or␣");
  addComplexTerm(CREL_OP_OR, ~, 0, 0, 0);
  break;
case CREL_OP_XOR: PRINTVERB(2, "xor␣");
  addComplexTerm(CREL_OP_XOR, ~, 0, 0, 0);
  break;
case CREL_OP_NEG: PRINTVERB(2, "neg␣");
  addComplexTerm(CREL_OP_NEG, ~, 0, 0, 0);
  break;
case CREL_OP_COM: PRINTVERB(2, "com␣");
  addComplexTerm(CREL_OP_COM, ~, 0, 0, 0);
  break;
case CREL_OP_FETCH_GLOBAL: ++RLD_i; value = ( uint16_t * ) (block + RLD_i);
  fromRadix50(value[0], gname);
```

```
  fromRadix50(value[1], gname + 3);
  RLD_i += 3;
  PRINTVERB(2, "%s␣", gname);
  addComplexTerm(CREL_OP_FETCH_GLOBAL, value, 0, 0, 0);
  break; case CREL_OP_FETCH_RELOCABLE: value = ( uint16_t * ) (block + RLD_i + 2);
  PRINTVERB(2, "sect:%o/%o␣", block[RLD_i + 1], value[0]);
  addComplexTerm(CREL_OP_FETCH_RELOCABLE, ~, block[RLD_i + 1], value[0], 0);
  RLD_i += 3;
  break;
case CREL_OP_FETCH_CONSTANT: ++RLD_i; value = ( uint16_t * ) (block + RLD_i);
  ++RLD_i;
  PRINTVERB(2, "%o␣", *value);
  addComplexTerm(CREL_OP_FETCH_CONSTANT, ~, 0, 0, value[0]);
  break;
default: PRINTERR("Bad␣complex␣relocation␣opcode:␣%d.\n", block[RLD_i]);
  return; } } addComplexTerm(block[RLD_i], ~, 0, 0, 0);
  ++RLD_i;
  PRINTVERB(2, "\n");
```

Этот код используется в секции 77.

110. ⟨ Обработать глобальные символы и ссылки 110 ⟩ ≡

```
  handleGlobalSymbol(entry);
```

Этот код используется в секции 23.

111. ⟨ Обработать программную секцию 111 ⟩ ≡

```
  handleProgramSection(entry);
```

Этот код используется в секции 23.

112. ⟨ Обработать секцию TXT 112 ⟩ ≡

```
  handleTextSection(block_body, block_len);
```

Этот код используется в секции 19.

113. ⟨ Обработать секцию перемещений 113 ⟩ ≡

```
  handleRelocationDirectory(block_body, block_len);
```

Этот код используется в секции 19.

114. ⟨ Глобальные переменные 9 ⟩ +≡

```
  static void handleGlobalSymbol(GSD_Entry *);
  static void handleProgramSection(GSD_Entry *); static void handleTextSection ( uint8_t * ,
      unsigned int ) ; static void handleRelocationDirectory ( uint8_t * , int ) ;
```

115. Вспомогательные функции.

116. Перевод двух байт из RADIX-50 в строку.

```
static void fromRadix50(int n, char *name)
{
  int i, x;
  for (i = 2; i ≥ 0; --i) {
    x = n % °50;
    n /= °50;
    if (x ≤ °32 ∧ x ≠ 0) {
      name[i] = x + 'A' − 1;
      continue;
    }
    if (x ≥ °36) {
      name[i] = x + '0' − °36;
      continue;
    }
    switch (x) {
    case °33: name[i] = '$';
      break;
    case °34: name[i] = '.';
      break;
    case °35: name[i] = '%';
      break;
    case °00: name[i] = '␣';
      break;
    }
  }
  name[3] = '\0';
}
```

117. ⟨ Глобальные переменные 9 ⟩ +≡

```
static void handleOneFile(FILE *);
static void handleGSD(int);
static void fromRadix50(int, char *);
```

118. Разбор параметров командной строки.
Для этой цели используется достаточно удобная свободная библиотека *argp*.

```
#define VERSION "0.6"
```

119. ⟨ Константы 119 ⟩ ≡

```
const char *argp_program_version = "linkbk, "VERSION;
const char *argp_program_bug_address = "<yellowrabbit@bk.ru>";
```

Этот код используется в секции 8.

120. ⟨ Глобальные переменные 9 ⟩ +≡

```
static char argp_program_doc[] = "Link MACRO-11 object files";
```

121. Распознаются следующие опции:

`-o` — имя выходного файла.
`-v` — вывод дополнительной информации (возможно указание дважды);
`-l NUM` — создаваемые файлы оверлеев имеют имена длиной не более NUM символов.

⟨ Глобальные переменные 9 ⟩ +≡

```
static struct argp_option options[] = {
    {"output", 'o', "FILENAME", 0, "Output filename"},
    {"verbose", 'v', ~, 0, "Verbose output"}, {"length", 'l', "LENGTH", 0,
    "Max overlay file name length"}, {0}
};
static error_t parse_opt(int, char *, struct argp_state *);
static struct argp argp = {options, parse_opt, ~, argp_program_doc};
```

122. Эта структура используется для получения результатов разбора параметров командной строки.

⟨ Собственные типы данных 16 ⟩ +≡

```
typedef struct _Arguments {
  int verbosity;
  char output_filename[FILENAME_MAX];      /* Имя файла с текстом */
  int max_filename_len;
    /* Максимальная длина имени выходных файлов. По умолчанию для MKDOS равна 14 */
  char **objnames;      /* Имена объектных файлов objnames[?] == NULL -> конец имен */
} Arguments;
```

123. ⟨ Глобальные переменные 9 ⟩ +≡

```
static Arguments config = {0, {0}, 14, ~, };
```

124. Задачей данного простого парсера является заполнение структуры **Arguments** из указанных параметров командной строки.

```
static error_t parse_opt(int key, char *arg, struct argp_state *state)
{
  Arguments *arguments;
  arguments = (Arguments *) state->input;
  switch (key) {
  case 'l': arguments->max_filename_len = atoi(arg);      /* не меньше x-SECTIO.v */
    if (arguments->max_filename_len < 1 + 6 + 2) arguments->max_filename_len = 1 + 1 + 6 + 2;
    break;
  case 'v': ++arguments->verbosity;
    break;
  case 'o':
    if (strlen(arg) ≡ 0) return (ARGP_ERR_UNKNOWN);
    strncpy(arguments->output_filename, arg, FILENAME_MAX - 1);
    break;
  case ARGP_KEY_ARG:      /* Имена объектных файлов */
    arguments->objnames = &state->argv[state->next - 1];      /* Останавливаем разбор параметров */
    state->next = state->argc;
    break;
  default: break;
    return (ARGP_ERR_UNKNOWN);
  }
  return (0);
}
```

125.

```
#define ERR_SYNTAX 1
#define ERR_CANTOPEN 2
#define ERR_CANTCREATE 3
```

⟨ Разобрать командную строку 125 ⟩ ≡

```
argp_parse(&argp, argc, argv, 0, 0, &config);
    /* Проверка параметров */
if (strlen(config.output_filename) ≡ 0) {
  PRINTERR("No output filename specified\n");
  return (ERR_SYNTAX);
}
if (config.objnames ≡ ~) {
  PRINTERR("No input filenames specified\n");
  return (ERR_SYNTAX);
}
```

Этот код используется в секции 8.

126. ⟨ Включение заголовочных файлов 126 ⟩ ≡

```
#include <string.h>
#include <stdlib.h>
#ifdef __linux__
#include <stdint.h>
#endif
#include <argp.h>
```

Этот код используется в секции 8.

127.

⟨ Глобальные переменные 9 ⟩ +≡
#**define** PRINTVERB (*level*, *fmt*, *a* ...) (((*config*.*verbosity*) ≥ *level*) ? *printf* ((*fmt*), ##*a*) : 0)
#**define** PRINTERR (*fmt*, *a* ...) *fprintf* (*stderr*, (*fmt*), ##*a*)

128. Макросы для монитора БК11М.

Файл lib/bk11m/bk11m.inc

```
;; vim: set expandtab ai textwidth=80:
                ; Общие
                .MCALL  .BINIT,.BEXIT

                ; Драйвер экрана
                .MCALL  .BTSET
                .MCALL  .BSTR,.BPRIN

                ; Драйвер клавиатуры
                .MCALL  .BTTIN

                ; Управление памятью
                .MCALL  .BPAGE,.BMEM,.BJSR,.BWORK
                .MCALL  .BJMP


```

Файл lib/bk11m/.BINIT.MAC

```
;; vim: set expandtab ai textwidth=80:
;; Инициализация монитора
                .MACRO  .BINIT
                jsr     PC,@140010
                .ENDM
```

Файл lib/bk11m/.BEXIT.MAC

```
;; vim: set expandtab ai textwidth=80:
;; Выход в монитор
                .MACRO  .BEXIT
                jsr     PC,@140012
                .ENDM

```

Файл lib/bk11m/.BJSR.MAC

```
;; vim: set expandtab ai textwidth=80:
;; Вызвать подпрограмму из рабочей страницы
                .MACRO  .BJSR ADDR
                mov     ADDR,-(SP)
                jsr     PC,@140054
                .ENDM

```

Файл lib/bk11m/.BMEM.MAC

```
;; vim: set expandtab ai textwidth=80:
;; Прочитать подключение страниц
;; R0 - младший байт -- страница с 40000
;;      старший байт -- страница с 100000
                .MACRO  .BMEM
                jsr     PC,@140030
                .ENDM
```

Файл lib/bk11m/.BPAGE.MAC

```
;; vim: set expandtab ai textwidth=80:
;; Подключить страницу
;; ADDR <> 0 - подключить PAGE с 100000
;;      иначе подключить PAGE с 40000
```

```
5                 .MACRO  .BPAGE PAGE,ADDR
6                 mov     #<ADDR*400>+PAGE,R0
7                 jsr     PC,@140034
8                 .ENDM
```

Файл lib/bk11m/.BWORK.MAC

```
1 ;; vim: set expandtab ai textwidth=80:
2 ;; Назначить/получить рабочую страницу
3 ;; если 15-ый бит = 0 -- установить страницу
4 ;;                 = 1 -- получить страницы (R0 ст.байт для вызовов,
5 ;;                                           мл.байт для чтения/записи
6 ;; если 7-ой бит = 1 -- установить для вызова подпрограмм
7 ;;                  0 -- установить для чтения/записи
8                 .MACRO  .BWORK ARG
9                 .IF DIF R0 <ARG>
10                mov     ARG,R0
11                .ENDC
12                jsr     PC,@140036
13                .ENDM
14
```

Файл lib/bk11m/.BSTR.MAC

```
1 ;; vim: set expandtab ai textwidth=80:
2 ;; Вывод строки
3                 .MACRO  .BSTR ADDR
4                 .IF DIF R0 <ADDR>
5                 mov     ADDR,R0
6                 .ENDC
7                 jsr     PC,@140162
8                 .ENDM
9
```

Файл lib/bk11m/.BTSET.MAC

```
1 ;; vim: set expandtab ai textwidth=80:
2 ;; Установка режима
3                 .MACRO  .BTSET MODE
4                 .IF DIF R0 <MODE>
5                 mov     MODE,R0
6                 .ENDC
7                 jsr     PC,@140132
8                 .ENDM
9
10
```

Файл lib/bk11m/.BTTIN.MAC

```
1 ;; vim: set expandtab ai textwidth=80:
2 ;; === Драйвер клавиатуры
3                 .MACRO  .BTTIN
4                 jsr     PC,@140076
5                 .ENDM
```

129. Макросы для MKDOS.

Файл lib/mkdos/mkdos.inc

```
;; vim: set expandtab ai textwidth=80:
                .MCALL  AFTER$MKDOS,MKDOS$TAPE
                .MCALL  BACK$TO$MKDOS

;; Адрес входа для магнитофонного интерфейса
MKDOS$MAG=120002

;; Страница, в которой находится MKDOS
MKDOS$PAGE=4
```

Файл lib/mkdos/AFTER$MKDOS.MAC

```
;; vim: set expandtab ai textwidth=80:
;; При запуске программы из-под MKDOS память находится в состоянии:
;; с 40000 подключена страница 5 (первый экран),
;; с 100000 подключена страница 4, в которой находится монитор БК0010
;; и MKDOS.
;; Чтобы использовать возможности монитора 11M нужно дать ему знать
;; какие страницы куда подключены.
;; По адресу 114 хранится копия регистра управления памятью по записи.
;; Имитируем последнюю запись в регистр.
                .MACRO  AFTER$MKDOS
                mov     #16200,@#114
                .ENDM

```

Файл lib/mkdos/MKDOS$TAPE.MAC

```
;; vim: set expandtab ai textwidth=80:
;; === Обращение через магнитофонный интерфейс.
                .MACRO  MKDOS$TAPE,TAPECMD
                mov     R0,-(SP)
                .BWORK  #100000         ; нужно сохранить текущую страницу для
                swab    R0              ; вызова подпрограмм
                bic     #177400,R0
                mov     R0,-(SP)

                .BWORK  #200+MKDOS$PAGE ; страница MKDOS для вызовов

                mov     TAPECMD,R1      ; вызов интерфейса
                .BJSR   #MKDOS$MAG

                mov     (SP)+,R0
                add     #200,R0
                .BWORK  R0

                mov     (SP)+,R0
                .ENDM

```

Файл lib/mkdos/BACKTOMKDOS.MAC

```
;; vim: set expandtab ai textwidth=80:
;; Подключаем 5-ю страницу с 40000
;; 4-ю с 10000 и передаем управление
;; монитору БК0010
```

```
5                .MACRO  BACK$TO$MKDOS
6                .BPAGE  5,0
7                .BPAGE  MKDOS$PAGE,1
8                jmp     @#100000
9                .ENDM
```

130. Индекс.

⟨ Включение заголовочных файлов 126 ⟩ Используется в секции 8.
⟨ Вывести отладочную информацию по секциям 48 ⟩ Используется в секции 36.
⟨ Вывести перекодировку секций 95 ⟩ Используется в секции 19.
⟨ Вывод неразрешенных ссылок 12 ⟩ Используется в секции 8.
⟨ Вывод таблицы глобальных символов 33 ⟩ Используется в секции 8.
⟨ Глобальные переменные 9, 18, 27, 31, 35, 37, 47, 52, 58, 65, 68, 92, 97, 102, 108, 114, 117, 120, 121, 123, 127 ⟩
Используется в секции 8.
⟨ Данные программы 10, 32, 43 ⟩ Используется в секции 8.
⟨ Добавить перекодировку секции 94 ⟩ Используется в секции 36.
⟨ Добавить программную секцию 46 ⟩ Используется в секции 36.
⟨ Заполнить пределы секций 72 ⟩ Используется в секции 8.
⟨ Изменение текущей позиции 84 ⟩ Используется в секции 77.
⟨ Инициализация каталога секций 42 ⟩ Используется в секции 8.
⟨ Инициализация списка пределов 70 ⟩ Используется в секции 8.
⟨ Инициализация списка сложных выражений 99 ⟩ Используется в секции 8.
⟨ Инициализация списка ссылок без констант 56 ⟩ Используется в секции 8.
⟨ Инициализация таблицы глобальных символов 28 ⟩ Используется в секции 8.
⟨ Константы 119 ⟩ Используется в секции 8.
⟨ Косвенная смещенная ссылка на секцию 89 ⟩ Используется в секции 77.
⟨ Косвенная ссылка на абсолютный адрес 79 ⟩ Используется в секции 77.
⟨ Косвенная ссылка на глобальный символ 81 ⟩ Используется в секции 77.
⟨ Косвенная ссылка на секцию 87 ⟩ Используется в секции 77.
⟨ Косвенная ссылка на смещенный глобальный символ 83 ⟩ Используется в секции 77.
⟨ Обработать блок 19 ⟩ Используется в секции 17.
⟨ Обработать глобальные символы и ссылки 110 ⟩ Используется в секции 23.
⟨ Обработать программную секцию 111 ⟩ Используется в секции 23.
⟨ Обработать секцию перемещений 113 ⟩ Используется в секции 19.
⟨ Обработать секцию TXT 112 ⟩ Используется в секции 19.
⟨ Освободить список пределов 71 ⟩ Используется в секции 8.
⟨ Освободить список сложных выражений 100 ⟩ Используется в секции 8.
⟨ Освободить список ссылок 57 ⟩ Используется в секции 8.
⟨ Открыть объектный файл 11 ⟩ Используется в секции 8.
⟨ Очистка каталога секций 44 ⟩ Используется в секции 8.
⟨ Прямая смещенная ссылка на секцию 88 ⟩ Используется в секции 77.
⟨ Прямая ссылка на абсолютный адрес 78 ⟩ Используется в секции 77.
⟨ Прямая ссылка на глобальный символ 80 ⟩ Используется в секции 77.
⟨ Прямая ссылка на секцию 86 ⟩ Используется в секции 77.
⟨ Прямая ссылка на смещенный глобальный символ 82 ⟩ Используется в секции 77.
⟨ Разобрать командную строку 125 ⟩ Используется в секции 8.
⟨ Разобрать GSD 21 ⟩ Используется в секции 19.
⟨ Разрешить косвенную ссылку 63 ⟩ Используется в секции 60.
⟨ Разрешить прямую ссылку 61 ⟩ Используется в секции 60.
⟨ Разрешить смещенную косвенную ссылку 64 ⟩ Используется в секции 60.
⟨ Разрешить смещенную прямую ссылку 62 ⟩ Используется в секции 60.
⟨ Распаковать имя 24 ⟩ Используется в секции 23.
⟨ Сбросить перекодировку секций 93 ⟩ Используется в секции 17.
⟨ Сложная ссылка 109 ⟩ Используется в секции 77.
⟨ Собственные типы данных 16, 22, 26, 34, 51, 67, 76, 91, 96, 122 ⟩ Используется в секции 8.
⟨ Создачм файл результата 13 ⟩ Используется в секции 8.
⟨ Установить адрес запуска 40 ⟩ Используется в секции 23.
⟨ Установка пределов 85 ⟩ Используется в секции 77.
⟨ Установка текущей секции и позиции 38 ⟩ Используется в секции 77.

LINKER