

30 Ноября 2013 года в 20:58

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-11¹. Входными параметрами являются: перечень имен объектных файлов, имя выходного файла² 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.

```

1 ;; vim: set expandtab ai textwidth=80:
2             .TITLE  SIMLBL
3             .IDENT  /V00.10/
4 ;; Все секции начинаются с 0, поэтому нужно смещение
5 .=.+1000
6 Start:      jsr     PC,@140010      ; инициализируем монитор БК11М
7 mov         #30204,R0
8             jsr     PC,@140132      ; дисплей в 80 символов в строке (и пр.)
9             mov     #Welcome,R0
10            jsr     PC,@140160      ; вывод строки
11            jsr     PC,@140076      ; ожидание нажатия клавиши
12            mov     #Bye,R0
13            jsr     PC,@140160      ; еще вывод строки
14            jsr     PC,@140076      ; еще ожидание нажатия клавиши
15            jmp     @#140000        ; reset монитора
16 Welcome:   .ASCIZ  /Very simple MACRO-11 program./
17 Bye:       .ASCIZ  /Bye./
18            .END     Start
19

```

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:

```

1 Module:SIMLBL
2   Ident: V00.10
3 =Global Definitions:
4 =Sections:
5 . ABS., addr: 0x8006da000, len: 0, min addr: 37777777777, current start: 0
6   , 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:

```

1 ;; vim: set expandtab ai textwidth=80:
2         .TITLE  TSTABS
3         .IDENT  /V00.10/
4 ;; Все секции начинаются с 0, поэтому нужно смещение
5 .+=1000
6 Start:   jsr     PC,@140010      ; инициализируем монитор БК11М
7         mov     #30204,R0
8         jsr     PC,@140132      ; дисплей в 80 символов в строке (и пр.)
9
10        ;; Запоминаем старый обработчик клавиатуры
11        mov     @#KeyboardVect,OldHandler
12        mov     @#KeyboardPSW,OldHandler+2
13
14        ;; «Тут установка нового обработчика и какая-то программа»
15
16        jsr     PC,@140076      ; ожидание нажатия клавиши
17        jmp     @#140000        ; reset монитора
18 OldHandler: .BLKW  2           ; место под данные старого обработчика
19                                     ; клавиатуры
20 ;; Абсолютная секция без указания имени
21        .ASECT
22 .+=60      ; смещение, так как секция всегда начинаются с нуля
23 KeyboardVect: .+=+2
24 KeyboardPSW:
25        .END      Start
26
```

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:

```

1 Module:TSTABS
2   Ident: V00.10
3 =Global Definitions:
4 =Sections:
5 . ABS., addr: 0x8006da000, len: 62, min addr: 3777777777, current start: 0
6   , 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 примере³.

```

1 ;; vim: set expandtab ai textwidth=80:
2         .TITLE  TSTAB2
3         .IDENT  /V00.11/
4         .INCLUDE lib/bk11m/bk11m.inc
5
6         ;; Вектор прерывания от клавиатуры
7 KeyboardVect=60
8 KeyboardPSW=KeyboardVect+2
9
10 .+.1000      ;; Все секции начинаются с 0, поэтому нужно смещение
11 Start:      .BINIT          ; инициализируем монитор БК11М
12             .BTSET  #30204    ; дисплей в 80 символов в строке (и пр.)
13
14             .BPRIN  #HelloStr
15             ;; Запоминаем старый обработчик клавиатуры
16             mov     @#KeyboardVect,OldHandler
17             mov     @#KeyboardPSW,OldHandler+2
18
19             ;; «Тут установка нового обработчика и какая-то программа»
20
21             .BTIN           ; ожидание нажатия клавиши
22             .BEXIT         ; выход
23 OldHandler:  .BLKW  2       ; место под данные старого обработчика
24             ; клавиатуры
25 HelloStr:   .ASCIZ  /Hi, there! :)/
26             .END    Start
27

```

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 отдельные файлы оверлеев, имена которых получаются из имени выходного исполняемого файла + “-” + имя секции + “.v”. При необходимости полученное имя файла оверлея урезается до указанной длины⁴.

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$MKDOS           ; если планируем использовать функции
13                                     ; MKDOS
14         .BINIT                     ; инициализируем монитор БК11М
15
16         ; подготавливаем распределение памяти “под себя”,
17         ; с 40000 страница 1 -- здесь будут оверлеи
18         ; с 100000 страница 2 -- просто так, чтобы там заведомо не было
19         ; MKDOS.
20         .BPAGE 1,0
21         .BPAGE 2,1
22
23         .BTSET  #30204
24         .BPRIN  #Prompt
25         .BTIN
26
27         ; загружаем оверлей
28         MKDOS$TAPE #TapeParams
29
30         ; вызываем функцию из оверлея
31         jsr     PC,SayHi
32
33         .BPRIN  #Loaded
34         .BTIN
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: 3777777777, 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:

```

1 ;; vim: set expandtab ai textwidth=80:
2         .TITLE   OVR2
3         .IDENT   /V01.00/
4         .INCLUDE lib/bk11m/bk11m.inc
5         .INCLUDE lib/mkdos/mkdos.inc
6
7 ;; Адрес с которого располагается оверлеи
8 LoadAddr=40000
9
10        .PSECT   MAIN
11 ;; Все секции начинаются с 0, поэтому нужно смещение
12 .+=+1000
13 Start:      AFTER$MKDOS           ; если планируем использовать функции
14                                     ; MKDOS
15        .BINIT          ; инициализируем монитор BK11M
16
17        ; подготавливаем распределение памяти ‘‘под себя’’,
18        ; с 40000 страница 1 -- здесь будут оверлеи
19        ; с 100000 страница 2 -- просто так, чтобы там заведомо не было
20        ; MKDOS.
21        .BPAGE 1,0
22        .BPAGE 2,1
23
24        .BTSET   #30204
25        .BPRIN   #Prompt
26        .BTIN
27
28        ; загружаем оверлей
29        MKDOS$TAPE #TapeParams
30
31        ; вызываем функцию из оверлея
32        jsr      PC,SayHi
33
34        .BPRIN   #Loaded
35        .BTIN
36        BACK$TO$MKDOS           ; назад в MKDOS
37
38 Prompt:     .ASCIZ  /Press any key to load overlay.../
39 Loaded:     .ASCIZ  /Overlay loaded./
40        .EVEN
41 TapeParams: .BYTE   3,0
42        .WORD   LoadAddr,0
43 1$:         .ASCII  /ovr2-SUBS.v/           ; Имя файла оверлея
44        .BLKB   ^D16-<.-1$>
45        .BLKB   ^D16+4
46
47 ;; Пустая секция, только задается начальное смещение
48 ;; В других .asm файлах секции с именем SUBS будут дописываться

```

```

49 ;; сюда
50             .PSECT  SUBS
51 .=. +LoadAddr
52             .END    Start
53

```

Второй файл:

```

1 ;; vim: set expandtab ai textwidth=80:
2             .TITLE  PART2
3             .IDENT  /V01.00/
4             .INCLUDE lib/bk11m/bk11m.inc
5
6 ;; =====
7 ;; Оверлей.
8 ;; Содержит подпрограмму SayHi, которая
9 ;; выводит на экран строчку.
10 ;; =====
11             .PSECT  SUBS
12 SayHi::      .BPRIN  #HiStr
13             rts      PC
14 HiStr:        .ASCIZ  /I'm overlay!/
15             .END

```

Лог линковки:

```

1 Module:OVR2
2   Ident: V01.00
3 Module:PART2
4   Ident: V01.00
5 =Global Definitions:
6 SAYHI : SUBS  /40000
7 =Sections:
8 . ABS., addr: 0x8006da000, len: 0, min addr: 3777777777, current start: 0
9       , addr: 0x8006eb000, len: 0, min addr: 3777777777, current start: 0
10 MAIN  , addr: 0x8006fc000, len: 1342, min addr: 1000, current start: 0
11 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 ⟩
        ⟨ Создать 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. \langle Вывод неразрешенных ссылок 12 $\rangle \equiv$

```

if ( $\neg$ simpleRefsIsEmpty()) {
    printf("Unresolved_simple_refs:\n");
    for ( $i = SRefList.pool[0].link; i \neq 0; i = SRefList.pool[i].link$ ) {
        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 ( $\neg$ complexRefsIsEmpty()) {
    printf("Unresolved_complex_refs:\n");
    for ( $i = CExprList.pool[0].link; i \neq 0; i = CExprList.pool[i].link$ ) {
        for ( $j = 0; j < CExprList.pool[i].NumTerms; ++j$ ) {
            if (CExprList.pool[i].terms[j].code  $\equiv$  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]);
            }
        }
    }
}
#if 0
    fromRadix50(SectDir[CExprList.pool[i].sect].name[0], sect_name);
    fromRadix50(SectDir[CExprList.pool[i].sect].name[1], sect_name + 3);
    printf("i: %4d, disp: %s/%o, file: %s\n", i, sect_name, CExprList.pool[i].disp,
        config.objnames[CExprList.pool[i].obj_file]);
#endif
}
}

```

Этот код используется в секции 8.

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

```

< Создаем 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); /* Оверлей */
        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. \langle Собственные типы данных 16 $\rangle \equiv$

```
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];
     $\langle$  Сбросить перекодировку секций 93  $\rangle$ 
    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;
        }
    }
     $\langle$  Обработать блок 19  $\rangle$ 
}
end: ;
}
```

18. Буффер для тела блока.

\langle Глобальные переменные 9 $\rangle + \equiv$

```
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_TRANSFER_ADDRESS 3
#define GSD_GLOBAL_SYMBOL_NAME 4
#define GSD_PSECT_NAME 5
#define GSD_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_TRANSFER_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 */  
} 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)
{
    if (entry->flags & GLOBAL_DEFINITION_MASK) {
        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;
        ++NumGlobalDefs;
    }
    if (config.verbosity ≥ 2) {
        PRINTVERB(2, "UUUUUUUUFlags:");
        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, "UUUUUUName:_%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. \langle Вывести отладочную информацию по секциям 48 $\rangle \equiv$

```

if (config.verbosity  $\geq$  2) {
    PRINTVERB(2, "UUUUUUUUFlags:");
    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 удаляемого 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);
}
```


59. Разрешение ссылок на глобальные символы.

60. Пробегает 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 ⟩ \equiv
 $dest_addr = (uint16_t *) (SectDir[SRefList.pool[ref].sect].text + SRefList.pool[ref].disp);$
 $*dest_addr = GSymDef[global].addr;$

Этот код используется в секции 60.

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

⟨ Разрешить смещенную прямую ссылку 62 ⟩ \equiv
 $dest_addr = (uint16_t *) (SectDir[SRefList.pool[ref].sect].text + SRefList.pool[ref].disp);$
 $*dest_addr = GSymDef[global].addr + SRefList.pool[ref].constant;$

Этот код используется в секции 60.

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

⟨ Разрешить косвенную ссылку 63 ⟩ \equiv
 $dest_addr = (uint16_t *) (SectDir[SRefList.pool[ref].sect].text + SRefList.pool[ref].disp);$
 $*dest_addr = GSymDef[global].addr - (SRefList.pool[ref].disp + 2);$

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 ⟩ \equiv
 $dest_addr = (uint16_t *) (SectDir[SRefList.pool[ref].sect].text + SRefList.pool[ref].disp);$
 $*dest_addr = GSymDef[global].addr - (SRefList.pool[ref].disp + 2) + SRefList.pool[ref].constant;$

Этот код используется в секции 60.

65. ⟨ Глобальные переменные 9 ⟩ \equiv
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. \langle Освободить список пределов 71 $\rangle \equiv$

```
if (config.verbosity  $\geq$  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.

\langle Заполнить пределы секций 72 $\rangle \equiv$

```
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 должен иметь хотя бы один блок **RLD**, который расположен впереди всех блоков **TEXT**, его задача — описать текущую **PSECT** и её размещение.

Каталог перемещений состоит из записей:

76. $\langle \text{Собственные типы данных 16} \rangle + \equiv$

```
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: %04x", 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, "Displacement: %o, Constant: %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, "Displacement: %o, Constant: %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, "UUUUUUDisp:_%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, "UUUUUUDisp:_%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, "UUUUUUName:_%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, "UUUUUUName:_%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-11¹.

```
#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. \langle Инициализация списка сложных выражений 99 $\rangle \equiv$

```
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. \langle Освободить список сложных выражений 100 $\rangle \equiv$

```
if (config.verbosity ≥ 2) {
    PRINTVERB(2, "=ComplexRefs:\n\tavail: \td, \tpoolmin: \td\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: \td, \tdisp: \ts/\to, \tfile: \ts\n", i, sect_name, CExprList.pool[i].disp,
            config.objnames[CExprList.pool[i].obj_file]);
    }
}
```

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 удаляемого 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 (!complexRefsIsEmpty()); }
```

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_SUBTRACTION: *(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. \langle Глобальные переменные 9 $\rangle + \equiv$
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_SUBTRACTION °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, "UUUUUUDisp: %o.\nUUUUUUUU", 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_SUBTRACTION: PRINTVERB(2, "-_");
    addComplexTerm(CREL_OP_SUBTRACTION, ~, 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. \langle Обработать глобальные символы и ссылки 110 $\rangle \equiv$
handleGlobalSymbol(entry);

Этот код используется в секции 23.

111. \langle Обработать программную секцию 111 $\rangle \equiv$
handleProgramSection(entry);

Этот код используется в секции 23.

112. \langle Обработать секцию TXT 112 $\rangle \equiv$
handleTextSection(block_body, block_len);

Этот код используется в секции 19.

113. \langle Обработать секцию перемещений 113 $\rangle \equiv$
handleRelocationDirectory(block_body, block_len);

Этот код используется в секции 19.

114. \langle Глобальные переменные 9 $\rangle + \equiv$
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. \langle Константы 119 $\rangle \equiv$

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

Этот код используется в секции 8.

120. \langle Глобальные переменные 9 $\rangle + \equiv$

```
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.

 \langle Глобальные переменные 9 $\rangle + \equiv$

```
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. **\langle Собственные типы данных 16 $\rangle + \equiv$**

```
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. \langle Глобальные переменные 9 $\rangle + \equiv$

```
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-SECTION.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 (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

```

1 ;; vim: set expandtab ai textwidth=80:
2             ; Общие
3             .MCALL .BINIT,.BEXIT
4
5             ; Драйвер экрана
6             .MCALL .BTSET
7             .MCALL .BSTR,.BPRIN
8
9             ; Драйвер клавиатуры
10            .MCALL .BTIN
11
12            ; Управление памятью
13            .MCALL .BPAGE,.BMEM,.BJSR,.BWORK
14            .MCALL .BJMP
15
16
```

Файл lib/bk11m/.BINIT.MAC

```

1 ;; vim: set expandtab ai textwidth=80:
2 ;; Инициализация монитора
3             .MACRO .BINIT
4             jsr    PC,@140010
5             .ENDM

```

Файл lib/bk11m/.BEXIT.MAC

```

1 ;; vim: set expandtab ai textwidth=80:
2 ;; Выход в монитор
3             .MACRO .BEXIT
4             jsr    PC,@140012
5             .ENDM

```

Файл lib/bk11m/.BJSR.MAC

```

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

```

Файл lib/bk11m/.BMEM.MAC

```

1 ;; vim: set expandtab ai textwidth=80:
2 ;; Прочитать подключение страниц
3 ;; R0 - младший байт -- страница с 40000
4 ;;      старший байт -- страница с 100000
5             .MACRO .BMEM
6             jsr    PC,@140030
7             .ENDM

```

Файл lib/bk11m/.BPAGE.MAC

```

1 ;; vim: set expandtab ai textwidth=80:
2 ;; Подключить страницу
3 ;; ADDR <> 0 - подключить PAGE с 100000
4 ;;      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

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

```
4
5 ;; Адрес входа для магнитофонного интерфейса
6 MKDOS$MAG=120002
```

```
7
8 ;; Страница, в которой находится MKDOS
9 MKDOS$PAGE=4
```

Файл lib/mkdos/AFTER\$MKDOS.MAC

```
1 ;; vim: set expandtab ai textwidth=80:
2 ;; При запуске программы из-под MKDOS память находится в состоянии:
3 ;; с 40000 подключена страница 5 (первый экран),
4 ;; с 100000 подключена страница 4, в которой находится монитор БК0010
5 ;; и MKDOS.
6 ;; Чтобы использовать возможности монитора 11М нужно дать ему знать
7 ;; какие страницы куда подключены.
8 ;; По адресу 114 хранится копия регистра управления памятью по записи.
9 ;; Имитируем последнюю запись в регистр.
```

```
10             .MACRO  AFTER$MKDOS
11             mov     #16200,@#114
12             .ENDM
```

```
13
```

Файл lib/mkdos/MKDOS\$TAPE.MAC

```
1 ;; vim: set expandtab ai textwidth=80:
2 ;; === Обращение через магнитофонный интерфейс.
3             .MACRO  MKDOS$TAPE,ТАПЕСМД
4             mov     R0,-(SP)
5             .BWORK  #100000          ; нужно сохранить текущую страницу для
6             swab    R0              ; вызова подпрограмм
7             bic     #177400,R0
8             mov     R0,-(SP)
9
10            .BWORK  #200+MKDOS$PAGE ; страница MKDOS для вызовов
11
12            mov     ТАПЕСМД,R1      ; вызов интерфейса
13            .BJSR   #MKDOS$MAG
14
15            mov     (SP)+,R0
16            add     #200,R0
17            .BWORK  R0
18
19            mov     (SP)+,R0
20            .ENDM
```

```
21
```

Файл lib/mkdos/BACK\$TO\$MKDOS.MAC

```
1 ;; vim: set expandtab ai textwidth=80:
2 ;; Подключаем 5-ю страницу с 40000
3 ;; 4-ю с 10000 и передаем управление
4 ;; монитору БК0010
```

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

130. Индекс.

- _linux_*: 126.
- _Arguments*: 122.
- _BinaryBlock*: 16.
- _ComplexExprEntry*: 96.
- _ComplexExpressionList*: 96.
- _ComplexTerm*: 96.
- _CurSectEntry*: 91.
- _GSD_Entry*: 22.
- _GSymDefEntry*: 26.
- _LimList*: 67.
- _LimListEntry*: 67.
- _RLD_Const_Entry*: 76.
- _RLD_Entry*: 76.
- _SectionDirEntry*: 34.
- _SimpleRefEntry*: 51.
- _SimpleRefList*: 51.
- a*: 127.
- abort*: 54, 69, 101.
- addComplexExpr*: 97, 101, 109.
- addComplexTerm*: 102, 103, 109.
- addLimit*: 68, 69, 85.
- addr*: 26, 29, 33, 41, 61, 62, 63, 64, 106.
- addSimpleRef*: 54, 58, 80, 81, 82, 83.
- arg*: 124.
- argc*: 8, 124, 125.
- argp*: 121, 125.
- ARGP_ERR_UNKNOWN: 124.
- ARGP_KEY_ARG: 124.
- argp-option*: 121.
- argp-parse*: 125.
- argp-program_bug_address*: 119.
- argp-program_doc*: 120, 121.
- argp-program_version*: 119.
- argp-state*: 121, 124.
- Arguments**: 122, 123, 124.
- arguments*: 124.
- argv*: 8, 124, 125.
- atoi*: 124.
- avail*: 51, 54, 55, 56, 57, 96, 99, 100, 101, 104.
- BinaryBlock**: 15, 16, 17.
- block*: 41, 77, 109.
- block_body*: 17, 18, 19, 23, 112, 113.
- block_len*: 17, 21, 112, 113.
- calcTerms*: 105, 107, 108.
- calloc*: 46.
- CExprList*: 12, 97, 98, 99, 100, 101, 103, 104, 105.
- cmd*: 54, 76, 77.
- code*: 12, 96, 103, 106, 107.
- ComplexExprEntry**: 96, 99, 101, 105, 106, 107, 108.
- ComplexExpressionList**: 96, 97.
- complexRefsEmpty*: 12, 97, 98, 105.
- ComplexTerm**: 96, 103, 107.
- config*: 8, 12, 13, 17, 19, 23, 29, 33, 48, 57, 71, 77, 100, 123, 125, 127.
- const_entry*: 38, 77, 82, 83, 88, 89.
- constant*: 38, 51, 54, 62, 64, 76, 82, 83, 88, 89, 96, 103, 106, 107.
- CREL_OP_ADDITION: 107, 109.
- CREL_OP_AND: 107, 109.
- CREL_OP_COM: 107, 109.
- CREL_OP_DIVISION: 107, 109.
- CREL_OP_FETCH_CONSTANT: 103, 106, 107, 109.
- CREL_OP_FETCH_GLOBAL: 12, 103, 106, 109.
- CREL_OP_FETCH_RELOCABLE: 103, 106, 109.
- CREL_OP_MULTIPLICATION: 107, 109.
- CREL_OP_NEG: 107, 109.
- CREL_OP_NONE: 107, 109.
- CREL_OP_OR: 107, 109.
- CREL_OP_STORE_RESULT: 105, 107, 109.
- CREL_OP_STORE_RESULT_DISP: 107, 109.
- CREL_OP_SUBTRACTION: 107, 109.
- CREL_OP_XOR: 107, 109.
- cur_input*: 8, 9, 17, 19, 23, 54, 77, 101.
- CurComplexExpr*: 101, 102, 103.
- CurSect*: 29, 36, 37, 38, 41, 46, 54, 69, 78, 79, 86, 87, 88, 89, 94, 101.
- CurSectEntry**: 91, 92.
- curSections*: 92, 94, 95, 106.
- DEFAULT_SECTION_LEN: 46.
- delComplexExpr*: 97, 104, 105.
- delSimpleRef*: 55, 58, 60.
- dest_addr*: 60, 61, 62, 63, 64, 77, 78, 79, 86, 87, 88, 89, 105.
- dest_dir*: 73.
- disp*: 12, 51, 54, 57, 61, 62, 63, 64, 67, 69, 71, 73, 76, 78, 79, 80, 81, 82, 83, 85, 86, 87, 88, 89, 96, 100, 101, 103, 105, 106, 109.
- end*: 17.
- ent*: 76.
- entry*: 23, 24, 29, 36, 38, 40, 46, 48, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 105, 106, 107, 109, 110, 111.
- EOF: 17.
- ERR_CANTCREATE: 13, 125.
- ERR_CANTOPEN: 11, 125.
- ERR_SYNTAX: 125.
- error_t*: 121, 124.
- fclose*: 8, 13.
- feof*: 17.
- fgetc*: 17.
- FILENAME_MAX: 122, 124.

- findGlobalSym*: [30](#), [31](#), [60](#), [106](#).
findSection: [36](#), [38](#), [40](#), [45](#), [47](#), [86](#), [87](#), [88](#), [89](#).
first_byte: [17](#).
flags: [22](#), [23](#), [26](#), [29](#), [34](#), [46](#), [48](#).
fmt: [127](#).
fobj: [8](#), [10](#), [11](#), [17](#).
fopen: [11](#), [13](#).
found: [30](#), [45](#).
fprintf: [127](#).
fread: [17](#).
free: [44](#), [57](#), [71](#), [100](#).
fresult: [10](#), [13](#).
fromRadix50: [12](#), [13](#), [24](#), [33](#), [38](#), [44](#), [57](#), [71](#), [80](#), [81](#),
[82](#), [83](#), [86](#), [87](#), [88](#), [89](#), [95](#), [100](#), [109](#), [116](#), [117](#).
fwrite: [13](#).
GDS_IDENT: [21](#), [23](#).
global: [60](#), [61](#), [62](#), [63](#), [64](#), [106](#).
GLOBAL_DEFINITION_MASK: [29](#).
GLOBAL_RELOCATION_MASK: [29](#).
global_sect: [91](#), [94](#), [95](#), [106](#).
GLOBAL_WEAK_MASK: [29](#).
gname: [38](#), [77](#), [80](#), [81](#), [82](#), [83](#), [86](#), [87](#), [88](#), [89](#), [109](#).
GSD_CSECT_NAME: [21](#), [23](#).
GSD_Entry: [22](#), [23](#), [29](#), [36](#), [114](#).
GSD_GLOBAL_SYMBOL_NAME: [21](#), [23](#).
GSD_INTERNAL_SYMBOL_NAME: [21](#), [23](#).
GSD_MAPPED_ARRAY: [21](#), [23](#).
GSD_MODULE_NAME: [21](#), [23](#).
GSD_PSECT_NAME: [21](#), [23](#).
GSD_TRANSFER_ADDRESS: [21](#), [23](#).
GSymDef: [27](#), [29](#), [30](#), [33](#), [61](#), [62](#), [63](#), [64](#), [106](#).
GSymDefEntry: [26](#), [27](#).
handleGlobalSymbol: [29](#), [110](#), [114](#).
handleGSD: [21](#), [23](#), [117](#).
handleOneFile: [8](#), [17](#), [117](#).
handleProgramSection: [36](#), [111](#), [114](#).
handleRelocationDirectory: [77](#), [113](#), [114](#).
handleTextSection: [41](#), [112](#), [114](#).
i: [8](#), [17](#), [23](#), [30](#), [45](#), [73](#), [105](#), [106](#), [107](#), [116](#).
INITIAL_COMPLEX_EXPR_LIST_SIZE: [98](#), [99](#).
INITIAL_LIMIT_LIST_SIZE: [67](#), [69](#), [70](#).
INITIAL_SIMPLE_REF_LIST_SIZE: [51](#), [54](#), [56](#), [101](#).
input: [124](#).
inter: [96](#), [103](#), [106](#).
int32_t: [34](#).
j: [8](#).
key: [124](#).
last_load_addr: [34](#), [41](#), [54](#), [69](#), [78](#), [79](#), [86](#), [87](#),
[88](#), [89](#), [101](#).
len: [13](#), [15](#), [16](#), [17](#), [23](#), [34](#), [36](#), [41](#), [44](#), [46](#), [73](#), [77](#).
level: [127](#).
LIMIT: [66](#).
LimitList: [68](#), [69](#), [70](#), [71](#), [73](#).
LimList: [67](#), [68](#).
LimListEntry: [67](#), [69](#), [70](#).
link: [12](#), [51](#), [53](#), [54](#), [55](#), [56](#), [57](#), [60](#), [67](#), [96](#), [98](#),
[99](#), [100](#), [101](#), [104](#), [105](#).
main: [8](#).
malloc: [56](#), [70](#), [99](#).
MAX_COMPLEX_TERMS: [96](#), [107](#).
max_filename_len: [13](#), [122](#), [124](#).
MAX_GLOBALS: [26](#), [27](#).
MAX_PROG_SECTIONS: [34](#), [35](#), [92](#).
memcpy: [41](#).
memset: [42](#).
min_addr: [13](#), [34](#), [38](#), [44](#), [46](#), [73](#).
n: [116](#).
name: [12](#), [13](#), [17](#), [22](#), [23](#), [24](#), [26](#), [29](#), [30](#), [32](#), [33](#),
[34](#), [36](#), [40](#), [44](#), [45](#), [46](#), [51](#), [54](#), [57](#), [60](#), [71](#), [91](#),
[94](#), [95](#), [96](#), [100](#), [103](#), [106](#), [116](#).
new_entry: [54](#), [69](#), [101](#).
new_index: [54](#), [101](#).
new_memory: [54](#), [69](#), [101](#).
next: [124](#).
not_resolved: [8](#), [106](#).
num_allocations: [51](#), [54](#), [56](#), [67](#), [69](#), [70](#), [96](#), [99](#), [101](#).
num_limits: [67](#), [69](#), [70](#), [71](#), [73](#).
num_start_addresses: [8](#), [9](#), [40](#).
NumCurSections: [92](#), [93](#), [94](#), [95](#).
NumGlobalDefs: [27](#), [28](#), [29](#), [30](#), [33](#).
NumSections: [13](#), [35](#), [42](#), [44](#), [45](#), [46](#).
NumTerms: [12](#), [96](#), [103](#), [106](#), [107](#).
obj_file: [12](#), [51](#), [54](#), [57](#), [96](#), [100](#), [101](#).
obj_header: [17](#).
objname: [8](#), [11](#).
objnames: [8](#), [12](#), [17](#), [19](#), [23](#), [57](#), [77](#), [100](#), [122](#),
[124](#), [125](#).
one: [16](#).
options: [121](#).
output_filename: [13](#), [122](#), [124](#), [125](#).
ovrname: [10](#), [13](#).
parse_opt: [121](#), [124](#).
pool: [12](#), [51](#), [53](#), [54](#), [55](#), [56](#), [57](#), [60](#), [61](#), [62](#), [63](#),
[64](#), [67](#), [69](#), [70](#), [71](#), [73](#), [96](#), [98](#), [99](#), [100](#), [101](#),
[103](#), [104](#), [105](#).
poolmin: [51](#), [54](#), [56](#), [57](#), [96](#), [99](#), [100](#), [101](#).
prev: [105](#).
prev_ref: [60](#).
PRINTERR: [8](#), [11](#), [13](#), [17](#), [19](#), [23](#), [54](#), [69](#), [77](#), [101](#),
[107](#), [109](#), [125](#), [127](#).
printf: [12](#), [127](#).
PRINTVERB: [17](#), [19](#), [23](#), [29](#), [33](#), [38](#), [41](#), [44](#), [48](#), [54](#),
[57](#), [69](#), [71](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#),
[86](#), [87](#), [88](#), [89](#), [95](#), [100](#), [101](#), [109](#), [127](#).

PSECT_ACCESS_MASK: [36](#), [48](#).
PSECT_ALLOCATION_MASK: [36](#), [48](#).
PSECT_RELOCATION_MASK: [36](#), [48](#).
PSECT_SAVE_MASK: [36](#), [48](#).
PSECT_SCOPE_MASK: [36](#), [48](#).
PSECT_TYPE_MASK: [36](#), [46](#), [48](#).
realloc: [54](#), [69](#), [101](#).
ref: [54](#), [60](#), [61](#), [62](#), [63](#), [64](#), [69](#), [101](#).
ref_i: [55](#), [104](#).
resolveComplex: [8](#), [105](#), [108](#).
resolveGlobals: [8](#), [60](#), [65](#).
resolveLimit: [68](#), [72](#), [73](#).
resolveTerms: [105](#), [106](#), [108](#).
result_type: [96](#), [105](#), [107](#).
RLD_CMD_COMPLEX_RELOCATION: [77](#).
RLD_CMD_GLOBAL_ADDITIVE_DISPLACED_RELOCATION: [60](#), [77](#).
RLD_CMD_GLOBAL_ADDITIVE_RELOCATION: [54](#), [60](#), [77](#).
RLD_CMD_GLOBAL_DISPLACED_RELOCATION: [60](#), [77](#).
RLD_CMD_GLOBAL_RELOCATION: [60](#), [77](#).
RLD_CMD_INTERNAL_DISPLACED_RELOCATION: [77](#).
RLD_CMD_INTERNAL_RELOCATION: [77](#).
RLD_CMD_LOCATION_COUNTER_DEFINITION: [77](#).
RLD_CMD_LOCATION_COUNTER_MODIFICATION: [77](#).
RLD_CMD_PROGRAM_LIMITS: [77](#).
RLD_CMD_PSECT_ADDITIVE_DISPLACED_RELOCATION: [77](#).
RLD_CMD_PSECT_ADDITIVE_RELOCATION: [77](#).
RLD_CMD_PSECT_DISPLACED_RELOCATION: [77](#).
RLD_CMD_PSECT_RELOCATION: [77](#).
RLD_Const_Entry: [38](#), [54](#), [76](#), [77](#), [82](#), [83](#), [88](#), [89](#).
RLD_Entry: [54](#), [58](#), [68](#), [69](#), [76](#), [77](#), [97](#), [101](#).
RLD_i: [38](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [109](#).
sect: [12](#), [23](#), [26](#), [29](#), [33](#), [40](#), [51](#), [54](#), [57](#), [61](#), [62](#), [63](#), [64](#), [67](#), [69](#), [71](#), [73](#), [77](#), [86](#), [87](#), [88](#), [89](#), [96](#), [100](#), [101](#), [103](#), [105](#), [106](#).
sect_name: [12](#), [13](#), [33](#), [43](#), [44](#), [57](#), [71](#), [100](#).
SectDir: [12](#), [13](#), [29](#), [33](#), [35](#), [36](#), [38](#), [40](#), [41](#), [42](#), [44](#), [45](#), [46](#), [54](#), [57](#), [61](#), [62](#), [63](#), [64](#), [69](#), [71](#), [73](#), [78](#), [79](#), [86](#), [87](#), [88](#), [89](#), [94](#), [100](#), [101](#), [105](#), [106](#).
SectionDirEntry: [34](#), [35](#).
SimpleRefEntry: [51](#), [54](#), [56](#).
simpleRefIsEmpty: [12](#), [52](#), [53](#), [60](#).
SimpleRefList: [51](#), [52](#).
sp: [107](#).
SRefList: [12](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [60](#), [61](#), [62](#), [63](#), [64](#).
stack: [107](#).
start: [29](#), [34](#), [36](#), [38](#), [41](#), [44](#), [78](#), [86](#), [87](#), [88](#), [89](#), [106](#).
state: [124](#).
static: [121](#).
stderr: [127](#).
strcat: [13](#).
strlen: [13](#), [124](#), [125](#).
strncpy: [13](#), [124](#).
term: [103](#), [107](#).
terms: [12](#), [96](#), [103](#), [106](#), [107](#).
text: [13](#), [34](#), [41](#), [44](#), [46](#), [61](#), [62](#), [63](#), [64](#), [73](#), [78](#), [79](#), [86](#), [87](#), [88](#), [89](#), [105](#).
TEXT: [41](#), [75](#).
transfer_addr: [13](#), [34](#), [40](#), [46](#).
type: [22](#), [23](#), [51](#), [54](#), [60](#), [76](#), [77](#).
uint16_t: [16](#), [22](#), [26](#), [30](#), [31](#), [34](#), [41](#), [45](#), [47](#), [51](#), [54](#), [55](#), [58](#), [60](#), [61](#), [62](#), [63](#), [64](#), [67](#), [73](#), [76](#), [77](#), [78](#), [79](#), [86](#), [87](#), [88](#), [89](#), [91](#), [96](#), [97](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#).
uint8_t: [16](#), [18](#), [22](#), [26](#), [34](#), [41](#), [46](#), [51](#), [67](#), [76](#), [77](#), [91](#), [96](#), [102](#), [103](#), [114](#).
un: [12](#), [96](#), [103](#), [106](#), [107](#).
ungetc: [17](#).
value: [22](#), [23](#), [29](#), [36](#), [38](#), [40](#), [46](#), [54](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [86](#), [87](#), [88](#), [89](#), [105](#), [109](#).
verbosity: [29](#), [33](#), [48](#), [57](#), [71](#), [100](#), [122](#), [124](#), [127](#).
VERSION: [118](#), [119](#).
x: [116](#).
zero: [16](#), [17](#).

- 〈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 ТХТ 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

	Секция	Страница
Введение	1	1
Примеры входных файлов	2	2
Общая схема программы	8	9
Обработка объектного файла	14	13
GSD	20	15
Списки ссылок на глобальные символы	49	24
Разрешение ссылок на глобальные символы	59	27
Обработка пределов (. LIMIT) для секций	66	30
Каталоги перемещений	74	32
Обработка сложных ссылок	90	37
Вспомогательные функции	115	47
Разбор параметров командной строки	118	48
Макросы для монитора БК11М	128	51
Макросы для MKDOS	129	53
Индекс	130	55