LINKBK

Простейший линковщик объектных файлов ассемблера MACRO-11
(Версия 0.6)

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 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.

Особенности:

- если основная/стартовая секция “размазана” по нескольким объектным файлам, то файл с точкой запуска программы должен быть указан в командной строке первым;
- если секция “размазана” по нескольким объектным файлам, то файл, в котором указано начальное смещение секции, должен быть первым в командной строке.

[1] Использовалась BSD-версия ассемблера Richard’a Krehbiel’a.
GIT-репозитарий ассемблера, линковщика и утилит https://github.com/yrabbit

19 Января 2014 года в 15:08

1. Примеры несложных программ. 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: 0x61ed10, len: 0, min addr: 37777777777, current start: 0
      , addr: 0x62ed20, 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`.

2. 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.

3. 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[2].

```
;; 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

```

[2] 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.

4. 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[3].

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

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

;; Адрес с которого располагается оверлеи
LoadAddr=40000

;; 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
```

[3] 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: 1340, min addr: 1000, current start: 0
7 SUBS  , addr: 0x8006fc000, len: 40030, min addr: 40000, current start: 0
```

5. 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
```

```
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: 0x61ed10, len: 0, min addr: 37777777777, current start: 0
9        , addr: 0x62ed20, len: 0, min addr: 37777777777, current start: 0
10 MAIN  , addr: 0x63ed30, len: 1340, min addr: 1000, current start: 0
11 SUBS  , addr: 0x64ed40, len: 40027, min addr: 40000, current start: 40000
```

6. Примеры с перемещаемыми секциями (атрибут SAV).

Допустим необходимо хранить все секции в одном исполняемом файле. Программист сам решает как разместить секции по нужным адресам на этопе выполнения программы.

Можно указать линковщику не делать файлы оверлеев, а помещать секции в исполняемый файл с помощью атрибута `SAV` директивы `.PSECT`. Такие программные секции помещаются в исполняемый файл сразу после стартовой программной секции.

Возникает вопрос определения размера и расположения перемещаемой секции в исполняемом файле (в случае с файлами оверлеев такой проблемы нет: считать и разместить файл оверлея является делом ОС). Расположение первой из перемещаемых секций — сразу после стартовой секции. Размер же ее можно вычислить как разницу между метками, поставленными в начале и конце перемещаемой секции.

Другой способ состоит в том, чтобы пожертвовать двумя словами в перемещаемой секции и использовать директиву `.LIMIT`, которая записывает в эти два слова начальный и конечный адреса секции после линковки, что демонстрируется в следующем примере.

Основной файл.

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

                .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 будет пересылаемая секция
                ; с 100000 страница 2 -- просто так, чтобы там заведомо не было
                ; MKDOS.
                .BPAGE 1,0
                .BPAGE 2,1

                .BTSET  #30204
                .BPRIN  #Prompt
                .BTTIN

                ; пересылаем секцию
                mov     #EndOfMain,R1   ; секция должна начинаться с .LIMIT
                mov     (R1),R2         ; R2 -- начальный адрес для размещения
                                        ; секции
                mov     2(R1),R0        ; R0 -- длина секции
                sub     R2,R0
                ror     R0
                adc     R0
1$:             mov    (R1)+,(R2)+
                sob     R0,1$
                ; вызываем функцию из перемещенной секции
                jsr     PC,SayHi

                .BPRIN  #Loaded
```

```
39                  .BTTIN
40                  BACK$TO$MKDOS           ; назад в MKDOS
41
42  Prompt:         .ASCIZ  /Press any key to load overlay.../
43  Loaded:         .ASCIZ  /Overlay loaded./
44                  .EVEN
45  EndOfMain:      ; Это конец основной секции и начало перемешаемой секции
46                  .END    Start
47
```

Файл, где описана перемещаемая секция.

```
1  ;; vim: set expandtab ai textwidth=80:
2                   .TITLE  BUILT2
3                   .IDENT  /V01.00/
4                   .INCLUDE lib/bk11m/bk11m.inc
5
6  ;; Адрес с которого располагается перемещаемая секция
7  LoadAddr==40000
8  ;; ============================================
9  ;;  Содержит подпрограмму SayHi, которая
10 ;;  выводит на экран строчку.
11 ;;  Установлен атрибут SAV.
12 ;; ============================================
13                  .PSECT  SUBS,SAV
14 .=.+LoadAddr
15                  .LIMIT          ; для автоматического вычисления размеров
16                                  ; секции
17 SayHi::          .BPRIN  #HiStr
18                  rts     PC
19 HiStr:           .ASCIZ  /I'm overlay!/
20                  .END
```

Лог линковки.

```
1  Module:BUILT
2    Ident: V01.00
3  Module:BUILT2
4    Ident: V01.00
5  =Global Definitions:
6  LOADAD: . ABS./40000
7  SAYHI : SUBS  /40004
8  =Sections:
9  . ABS., addr: 0x8006da000, len: 0, min addr: 37777777777, current start: 0
10       , addr: 0x8006eb000, len: 0, min addr: 37777777777, current start: 0
11 MAIN  , addr: 0x8006fc000, len: 1230, min addr: 1000, current start: 0
12 SUBS  , addr: 0x80070d000, len: 40034, min addr: 40000, current start: 0
```

7. Дополнительная память (RAM-BIOS).

При использовании контроллеров “АльтПро” с дополнительной памятью 64 – 512 КиБ и версией ПЗУ не ниже 2.0 возможно размещать исполняемый код и данные в этой дополнительной памяти. Работа с дополнительной памятью осуществляется через программные запросы к специальному диспетчеру[4], часть которого должна быть загружена в память.

В подкаталоге tests имеются файлы instBIOS и MEMORY, первый как раз и служит для загрузки части RAM-BIOS в память. Программа MEMORY предназначена для интерактивного просмотра/изменения состояния дополнительной памяти.

Чтобы не полагаться на наличие в памяти диспетчера можно встроить загрузчик RAM-BIOS в свою программу, оформив его как оверлей, работающий с адреса 40000.

В следующем примере диспетчер считается уже загруженным. Программа подготавливает заголовки для двух областей памяти (модулей в терминологии RAM-BIOS), запрашивает память и пересылает подпрограмму в обе выделенные области . Фрагмент с чтением слова — просто демонстрация. Затем подпрограммы вызываются для выполнения прямо в дополнительной памяти, результатом работы подпрограмм является “картинка”, заполненная переданным шаблоном. После чего эти “картинки” копируются в экранную область памяти для отображения.

Поскольку адреса выделенных кусочков дополнительной памяти “плавают”, то размещаемые в ней подпрограммы нужно писать в перемещаемом стиле, как показано в примере.

Файл ram-bios.asm.

```
1  ;; vim: set expandtab ai textwidth=80:
2                  .TITLE  RAMBS
3                  .IDENT  /V00.10/
4                  .INCLUDE lib/bk11m/bk11m.inc
5                  .INCLUDE lib/ram-bios/ram-bios.inc
6                  .INCLUDE lib/mkdos/mkdos.inc
7
8                  .PSECT  MAINPS
9  .=.+1000        ;; Все секции начинаются с 0, поэтому нужно смещение
10 Start:          AFTER$MKDOS
11                 .BINIT                  ; инициализируем монитор БК11М
12                 .BTSET  #30204          ; дисплей в 80 символов в строке (и пр.)
13
14                 mov     #Exit+2,R0
15                 clr     R1
16                 CHW$G   DOS$,SYSPAG$
17                 mov     R0,R1
18                 jsr     PC,ToStr
19
20                 .BPRIN   #HelloStr
21                 .BTTIN                  ; ожидание нажатия клавиши
22
23                 ; Запрашиваем и выводим на экран некоторые
24                 ; параметры RAM-BIOS
25                 CAT$    #GetVersion     ; запросить версию RAM-BIOS
26                 mov     R1,-(SP)
27                 mov     R2,-(SP)
28                 mov     #NumberBuf,R1
29                 mov     R1,R4
30                 jsr     PC,ToStr
31                 .BSTR   #VersionStr
```

[4] Вызовы диспетчера подробно описаны http://forum.pk-fpga.ru/viewtopic.php?f=39&t=5410

```
32              .BPRIN  R4
33              mov     R4,R1
34              mov     (SP)+,R0
35              jsr     PC,ToStr
36              .BSTR   #NumRecordsStr
37              .BPRIN  R4
38              mov     R4,R1
39              mov     (SP)+,R0
40              jsr     PC,ToStr
41              .BSTR   #CatalogAddrStr
42              .BPRIN  R4
43              ; Добавляем первый программный модуль
44              mov     #Module,R1
45              clr     R2
46              mov     #"CY,Cat.Name(R1)
47              mov     #<YCodeEnd-YCode>/2+1,Cat.Len(R1)
48              mov     #<Cat.Flags.NoSplit!Cat.Flags.Run!Cat.Flags.Del>,Cat.Flags(R1)
49              CAT$    #CreateModule
50              ; Читаем смещение в сегменте добавленного модуля
51              CAT$    #ReadWriteCatRecord
52              .WORD   "CY,Cat.Off
53              mov     R1,R0
54              mov     R4,R1
55              jsr     PC,ToStr
56              .BSTR   #SegmentOffStr
57              .BPRIN  R4
58
59              ; Добавляем второй программный модуль
60              mov     #Module,R1
61              clr     R2
62              mov     #"DY,Cat.Name(R1)
63              mov     #<YCodeEnd-YCode>/2+1,Cat.Len(R1)
64              mov     #<Cat.Flags.NoSplit!Cat.Flags.Run!Cat.Flags.Del>,Cat.Flags(R1)
65              CAT$    #CreateModule
66              ; Читаем смещение в сегменте добавленного модуля
67              CAT$    #ReadWriteCatRecord
68              .WORD   "DY,Cat.Off
69              mov     R1,R0
70              mov     R4,R1
71              jsr     PC,ToStr
72              .BSTR   #SegmentOffStr
73              .BPRIN  R4
74
75              ; Пересылка подпрограммы в оба модуля
76              MOV$G   NASEG.Current,#YCode,"CY,0,<#YCodeEnd-YCode>/2+1
77              MOV$G   NASEG.Current,#YCode,"DY,0,<#YCodeEnd-YCode>/2+1
78
79              ; Прочитать первое слово из второго модуля
80              clr     R0
81              mov     PC,R1
82              CHW$G   0,"DY
83              mov     R4,R1
```

```
84                  jsr     PC,ToStr
85                  .BSTR   #WordStr
86                  .BPRIN  R4
87
88                  ; Выполняем обе подпрограммы
89                  mov     #123,R3
90                  CAL$G   0,"CY
91                  mov     #346,R3
92                  CAL$G   0,"DY
93
94                  ; Забираем результаты
95                  MOV$G   "CY,#Pic-YCode,NASEG.Current,#50000,#PicSize/2
96                  MOV$G   "DY,#Pic-YCode,NASEG.Current,#PicSize+50000,#PicSize/2
97
98                  ; Удаляем модули
99                  mov     #"CY,R1
100                 CAT$    #DeleteModule
101                 mov     #"DY,R1
102                 CAT$    #DeleteModule
103
104 Exit:           .EXIT$
105                 .BACK$TO$MKDOS
106                 .BEXIT                  ; выход
107
108 Module:         .BLKB   20              ; данные добавляемого модуля
109 HelloStr:       .ASCIZ  /Ask RAM-BIOS.../
110 VersionStr:     .ASCIZ  /Version: /
111 NumRecordsStr: .ASCIZ  /Number of catalog records: /
112 CatalogAddrStr:.ASCIZ  /Address of catalog: /
113 SegmentOffStr: .ASCIZ  /Offset in the segment: /
114 LogSegStr:     .ASCIZ  /Logical segment number: /
115 WordStr:       .ASCIZ  /First word of module: /
116 NumberBuf:     .BLKB   7
117                 .EVEN
118 ; Перевод числа из R0 с строку восьмеричных цифр в буфере R1
119 ToStr:          mov     #5,R2
120                 add     R2,R1
121                 inc     R1
122                 clrb    (R1)
123 1$:
124                 movb    R0,-(R1)
125                 bicb    #370,(R1)
126                 bisb    #'0,(R1)
127                 ror     R0
128                 ror     R0
129                 ror     R0
130                 sob     R2,1$
131                 bicb    #376,R0
132                 bisb    #'0,R0
133                 movb    R0,-(R1)
134                 rts     PC
135
```

```
136 ; Подпрограмма, помещаемая в память SMK и выполняющаяся прямо в этой памяти.
137 ;
138 PicSize=2400*2  ; память, занятная картинкой
139 YCode:          mov     PC,R1
140                 add     #Pic - .,R1
141                 mov     #PicSize,R0
142 1$:             movb    R3,(R1)+
143                 sob     R0,1$
144                 rts     PC
145 Pic:            .BLKB   PicSize
146 YCodeEnd:
147                 .END    Start
148
```

8. Общая схема программы.

```
⟨ Включение заголовочных файлов 117 ⟩
⟨ Директивы препроцессора ⟩
⟨ Константы 110 ⟩
⟨ Собственные типы данных 15 ⟩
⟨ Глобальные переменные 9 ⟩
int main(int argc, char *argv[])
{
  ⟨ Данные программы 10 ⟩
  const char *objname;
  int i, j, not_resolved;
  ⟨ Разобрать командную строку 116 ⟩
  ⟨ Инициализация каталога секций 39 ⟩
  ⟨ Инициализация таблицы глобальных символов 26 ⟩
  ⟨ Инициализация списка ссылок без констант 52 ⟩
  ⟨ Инициализация списка сложных выражений 91 ⟩
  ⟨ Инициализация списка пределов 64 ⟩
    /* 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 31 ⟩
    ⟨ Заполнить пределы секций 66 ⟩
    ⟨ Создаем файл результата 13 ⟩
  }
  else {
    ⟨ Вывод неразрешенных ссылок 12 ⟩
  }
  ⟨ Очистка каталога секций 41 ⟩
  ⟨ Освободить список сложных выражений 92 ⟩
  ⟨ Освободить список ссылок 53 ⟩
  ⟨ Освободить список пределов 65 ⟩
  return (0);
}
```

9. Номер текущего обрабатываемого объектного файла.

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

Смотри также секции 17, 25, 29, 33, 35, 44, 48, 54, 60, 62, 84, 89, 94, 100, 106, 108, 111, 112, 114, и 118.

Этот код используется в секции 8.

10.

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

Смотри также секции 30 и 40.

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 создается файл с программной секцией, для которой указан адрес запуска. Остальные секции ненулевой длины планируется писать в дополнительные файлы (оверлеи). Исключение составляют секции с атрибутом `SAV` — такие секции дописываются в исполняемый файл.

```
⟨ Создаем файл результата 13 ⟩ ≡
  for (i = 0; i < NumSections; ++i) {
    if (SectDir[i].len ≠ 0 ∧ SectDir[i].min_addr ≠ −1 ∧ SectDir[i].transfer_addr ≡
          1 ∧ ¬(SectDir[i].flags & PSECT_SAVE_MASK)) {
      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);
      continue;
    }
    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;
    }
  }     /* Дописываем в главный модуль секции с флагом SAVE */
  for (i = 0; i < NumSections; ++i) {
    if (SectDir[i].flags & PSECT_SAVE_MASK) {
      fresult = fopen(config.output_filename, "a");
      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);
    }
  }
```

Этот код используется в секции 8.

14. Обработка объектного файла.

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.

15. ⟨ Собственные типы данных 15 ⟩ ≡

```
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 20, 24, 32, 47, 61, 69, 83, 88, и 113.

Этот код используется в секции 8.

16. Обработать один объектный файл.

```
static void handleOneFile(FILE *fobj)
{
  BinaryBlock obj_header;
  int first_byte, i;
  int crc;
  unsigned int block_len;
  char name[7];
  ⟨ Сбросить перекодировку секций 85 ⟩
  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 while read header: %s\n", config.objnames[cur_input]);
      exit(EXIT_FAILURE);
    }
    if (obj_header.zero ≠ 0) continue;
    block_len = obj_header.len − 4;
    PRINTVERB(2, "Binary block found. Length:%o\n", block_len);
    if (obj_header.len ≡ 0) {
      PRINTERR("Block len = 0: %s\n", config.objnames[cur_input]);
      exit(EXIT_FAILURE);
    }      /* Читаем тело блока с контрольной суммой */
    if (fread(block_body, block_len + 1, 1, fobj) ≠ 1) {
      PRINTERR("IO error while read block: %s\n", config.objnames[cur_input]);
      exit(EXIT_FAILURE);
    }      /* Подсчет контрольной суммы */
    crc = −obj_header.one − obj_header.zero − obj_header.len % 256 − obj_header.len/256;
    for (i = 0; (uint16_t)i < block_len; ++i) {
      crc −= block_body[i];
    }
    crc &= #ff;
    if (crc ≠ block_body[block_len]) {
      PRINTERR("Bad block checksum: %s\n", config.objnames[cur_input]);
      exit(EXIT_FAILURE);
    }
    ⟨ Обработать блок 18 ⟩
  }
end: ;
}
```

17. Буффер для тела блока.

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

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

```
⟨ Обработать блок 18 ⟩ ≡
  PRINTVERB(2, "␣␣Block␣type:␣%o,␣", block_body[0]);
  switch (block_body[0]) {
  case 1: PRINTVERB(2, "GSD\n");
    ⟨ Разобрать GSD 19 ⟩
    break;
  case 2: PRINTVERB(2, "ENDGSD\n");
    ⟨ Вывести перекодировку секций 87 ⟩
    break;
  case 3: PRINTVERB(2, "TXT\n");
    ⟨ Обработать секцию TXT 104 ⟩
    break;
  case 4: PRINTVERB(2, "RLD\n");
    ⟨ Обработать секцию перемещений 105 ⟩
    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 16.

19. GSD.

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 19 ⟩ ≡

```
handleGSD(block_len);
```

Этот код используется в секции 18.

20. ⟨ Собственные типы данных 15 ⟩ +≡

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

21. **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 22 ⟩
    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 37 ⟩
      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 102 ⟩
      break;
    case GSD_PSECT_NAME:      /* Имя программной секции */
      PRINTVERB(2, "PSectName,␣flags:%o,␣max␣length:%o.\n", entry→flags, entry→value);
      ⟨ Обработать программную секцию 103 ⟩
      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]);
    }
  }
}
```

22. ⟨ Распаковать имя 22 ⟩ ≡

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

Этот код используется в секции 21.

23. Разбор определения/ссылки на глобальный символ.

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

```
#define MAX_GLOBALS 1024
⟨ Собственные типы данных 15 ⟩ +≡
  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;
```

25.
```
⟨ Глобальные переменные 9 ⟩ +≡
  static GSymDefEntry GSymDef[MAX_GLOBALS];
  static int NumGlobalDefs;
```

26.
```
⟨ Инициализация таблицы глобальных символов 26 ⟩ ≡
  NumGlobalDefs = 0;
```

Этот код используется в секции 8.

27.

```
#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");
      }
    }
  }
```

28. 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);
}
```

29. ⟨ Глобальные переменные 9 ⟩ +≡

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

30. ⟨ Данные программы 10 ⟩ +≡

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

31. ⟨ Вывод таблицы глобальных символов 31 ⟩ ≡

```
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.

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

```
#define MAX_PROG_SECTIONS 254
```

⟨ Собственные типы данных 15 ⟩ +≡

```
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;
```

33. ⟨ Глобальные переменные 9 ⟩ +≡

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

34.

```
#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)
  {
    char name[7];
    ⟨ Вывести отладочную информацию по секциям 45 ⟩
    CurSect = findSection(entry→name);
    if (CurSect ≡ −1) {
      ⟨ Добавить программную секцию 43 ⟩
    }
    else {      /* Проверяем не изменились ли флаги секции */
      if (SectDir[CurSect].flags ≠ entry→flags) {
        fromRadix50(SectDir[CurSect].name[0], name);
        fromRadix50(SectDir[CurSect].name[1], name + 3);
        PRINTERR("Section␣%s␣flags␣conflict.␣Old␣flags:␣%x,␣new""␣flags:␣%x.␣File:␣%s\n",
            name, SectDir[CurSect].flags, entry→flags, config.objnames[cur_input]);
        exit(EXIT_FAILURE);
      }      /* Изменить смещение секции в модуле */
      SectDir[CurSect].start = SectDir[CurSect].len;
      SectDir[CurSect].len += entry→value;
    }
    ⟨ Добавить перекодировку секции 86 ⟩
  }
```

35. ⟨ Глобальные переменные 9 ⟩ +≡

```
  static int CurSect;
```

36. Переключается текущая секция.

```
⟨ Установка текущей секции и позиции 36 ⟩ ≡
  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 70.

37. Адрес запуска, равный единице игнорируется.

```
⟨ Установить адрес запуска 37 ⟩ ≡
  sect = findSection(entry→name);
  SectDir[sect].transfer_addr = entry→value;
  if (entry→value ≠ 1) ++num_start_addresses;
```

Этот код используется в секции 21.

38. 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;
  }
```

39.

```
⟨ Инициализация каталога секций 39 ⟩ ≡
  NumSections = 0;
  memset(SectDir, 0, sizeof (SectDir));
```

Этот код используется в секции 8.

40.

```
⟨ Данные программы 10 ⟩ +≡
  char sect_name[7];
```

41.

```
⟨ Очистка каталога секций 41 ⟩ ≡
  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.

42. 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);
}
```

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

```
#define DEFAULT_SECTION_LEN 65536
```

⟨ Добавить программную секцию 43 ⟩ ≡

```
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 34.

44. ⟨ Глобальные переменные 9 ⟩ +≡

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

45. ⟨ Вывести отладочную информацию по секциям 45 ⟩ ≡

```
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 34.

46. Списки ссылок на глобальные символы.

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.

47. Структура элемента списка для хранения ссылок.

```
#define INITIAL_SIMPLE_REF_LIST_SIZE 100
⟨ Собственные типы данных 15 ⟩ +≡
  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;
```

48.
```
⟨ Глобальные переменные 9 ⟩ +≡
  static SimpleRefList SRefList;
  static int simpleRefIsEmpty(void);
```

49.
```
  static int simpleRefIsEmpty(void)
  {
    return (SRefList.pool[0].link ≡ 0);
  }
```

50. 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; } }
```

51. 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);
}
```

52. *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 52 ⟩ ≡
  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.

53.

```
⟨ Освободить список ссылок 53 ⟩ ≡
  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.

54.

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

55. Разрешение ссылок на глобальные символы.

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 56 ⟩      /* 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 58 ⟩
        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 57 ⟩
        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 59 ⟩
        SRefList.pool[prev_ref].link = delSimpleRef(ref);
          /* При удалении ref стоит вернуться на шаг назад */
        ref = prev_ref;
        continue;
      }
    }
  }
  return (¬simpleRefIsEmpty());
}
```

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

⟨ Разрешить прямую ссылку 56 ⟩ ≡
```
dest_addr = ( uint16_t * ) (SectDir[SRefList.pool[ref].sect].text + SRefList.pool[ref].disp);
*dest_addr = GSymDef[global].addr;
```
Этот код используется в секции 55.

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

⟨ Разрешить смещенную прямую ссылку 57 ⟩ ≡
```
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 55.

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

⟨ Разрешить косвенную ссылку 58 ⟩ ≡
```
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 55.

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

⟨ Разрешить смещенную косвенную ссылку 59 ⟩ ≡
```
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 55.

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

61. Обработка пределов (. LIMIT) для секций.
Структура элемента списка для хранения ссылок на пределы.

```
#define INITIAL_LIMIT_LIST_SIZE 5
⟨ Собственные типы данных 15 ⟩ +≡
  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;
```

62. ⟨ Глобальные переменные 9 ⟩ +≡

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

63. 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;
  }
```

64. ⟨ Инициализация списка пределов 64 ⟩ ≡

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

Этот код используется в секции 8.

65. ⟨ Освободить список пределов 65 ⟩ ≡

```
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.

66.

⟨ Заполнить пределы секций 66 ⟩ ≡

```
resolveLimit( );
```

Этот код используется в секции 8.

67.

```
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; } }
```

68. Каталоги перемещений.

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 ее размещение.

Каталог перемещений состоит из записей:

69. ⟨ Собственные типы данных 15 ⟩ +≡

```
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;
```

70. Поле *cmd.type* указывает на тип ссылки.

```
#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 71 ⟩
        break;
      case RLD_CMD_GLOBAL_RELOCATION: PRINTVERB(2, "Global␣Relocation.\n");
        ⟨ Прямая ссылка на глобальный символ 73 ⟩
        break;
      case RLD_CMD_INTERNAL_DISPLACED_RELOCATION:
        PRINTVERB(2, "Internal␣Displaced␣Relocation.\n");
        ⟨ Косвенная ссылка на абсолютный адрес 72 ⟩
        break;
      case RLD_CMD_GLOBAL_DISPLACED_RELOCATION:
        PRINTVERB(2, "Global␣Displaced␣Relocation.\n");
        ⟨ Косвенная ссылка на глобальный символ 74 ⟩
        break;
      case RLD_CMD_GLOBAL_ADDITIVE_RELOCATION:
        PRINTVERB(2, "Global␣Additive␣Relocation.\n");
        ⟨ Прямая ссылка на смещенный глобальный символ 75 ⟩
        break;
      case RLD_CMD_GLOBAL_ADDITIVE_DISPLACED_RELOCATION:
        PRINTVERB(2, "Global␣Additive␣Displaced␣Relocation.\n");
        ⟨ Косвенная ссылка на смещенный глобальный символ 76 ⟩
        break;
      case RLD_CMD_LOCATION_COUNTER_DEFINITION:
        PRINTVERB(2, "Location␣Counter␣Definition.\n");
        ⟨ Установка текущей секции и позиции 36 ⟩
```

```
        break;
      case RLD_CMD_LOCATION_COUNTER_MODIFICATION:
        PRINTVERB(2, "Location␣Counter␣Modification.\n");
        ⟨ Изменение текущей позиции 77 ⟩
        break;
      case RLD_CMD_PROGRAM_LIMITS: PRINTVERB(2, "Program␣Limits.\n");
        ⟨ Установка пределов 78 ⟩
        break;
      case RLD_CMD_PSECT_RELOCATION: PRINTVERB(2, "PSect␣Relocation.\n");
        ⟨ Прямая ссылка на секцию 79 ⟩
        break;
      case RLD_CMD_PSECT_DISPLACED_RELOCATION:
        PRINTVERB(2, "PSect␣Displaced␣Relocation.\n");
        ⟨ Косвенная ссылка на секцию 80 ⟩
        break;
      case RLD_CMD_PSECT_ADDITIVE_RELOCATION: PRINTVERB(2, "PSect␣Additive␣Relocation.\n");
        ⟨ Прямая смещенная ссылка на секцию 81 ⟩
        break;
      case RLD_CMD_PSECT_ADDITIVE_DISPLACED_RELOCATION:
        PRINTVERB(2, "PSect␣Additive␣Displaced␣Relocation.\n");
        ⟨ Косвенная смещенная ссылка на секцию 82 ⟩
        break;
      case RLD_CMD_COMPLEX_RELOCATION: PRINTVERB(2, "Complex␣Relocation.\n");
        ⟨ Сложная ссылка 101 ⟩
        break;
      default:
        PRINTERR("Bad␣RLD␣entry␣type:␣%o␣:␣%s\n", entry→cmd.type, config.objnames[cur_input]);
        return;
      }
    }
  }
```

71.

⟨ Прямая ссылка на абсолютный адрес 71 ⟩ ≡

```
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 70.

72.

⟨ Косвенная ссылка на абсолютный адрес 72 ⟩ ≡

```
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 70.

73.

⟨ Прямая ссылка на глобальный символ 73 ⟩ ≡
```
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 70.

74.

⟨ Косвенная ссылка на глобальный символ 74 ⟩ ≡
```
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 70.

75.

⟨ Прямая ссылка на смещенный глобальный символ 75 ⟩ ≡
```
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 70.

76.

⟨ Косвенная ссылка на смещенный глобальный символ 76 ⟩ ≡
```
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 70.

77. Не используется.

⟨ Изменение текущей позиции 77 ⟩ ≡
```
PRINTVERB(2, "␣␣␣␣␣␣+Const:␣%o.\n", entry→value[0]);
RLD_i += 4;
```
Этот код используется в секции 70.

78.

⟨ Установка пределов 78 ⟩ ≡
```
PRINTVERB(2, "␣␣␣␣␣␣Disp:␣%o.\n", entry→disp);
addLimit(entry);
RLD_i += 2;
```
Этот код используется в секции 70.

79.

⟨ Прямая ссылка на секцию 79 ⟩ ≡
```
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 70.

80.

⟨ Косвенная ссылка на секцию 80 ⟩ ≡
```
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 70.

81.

⟨ Прямая смещенная ссылка на секцию 81 ⟩ ≡
```
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 70.

82.

⟨ Косвенная смещенная ссылка на секцию 82 ⟩ ≡
```
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 70.

83. Обработка сложных ссылок.
Для сложных ссылок нужно знать номера секций в текущем модуле.

```
⟨ Собственные типы данных 15 ⟩ +≡
  typedef struct _CurSectEntry {
    uint16_t name[2];
    uint8_t global_sect;
  } CurSectEntry;
```

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

85. ⟨ Сбросить перекодировку секций 85 ⟩ ≡
```
  NumCurSections = 0;
```
Этот код используется в секции 16.

86. ⟨ Добавить перекодировку секции 86 ⟩ ≡
```
  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 34.

87. ⟨ Вывести перекодировку секций 87 ⟩ ≡
```
  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 18.

88. Список сложных ссылок. Каждая ссылка содержит простой массив термов сложного выражения. Максимальное количество термов в выражении ограничено согласно документации MACRO-11[5].

```
#define MAX_COMPLEX_TERMS 20
⟨ Собственные типы данных 15 ⟩ +≡
  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;
```

89.
```
⟨ Глобальные переменные 9 ⟩ +≡
  static ComplexExpressionList CExprList;
  static int complexRefIsEmpty(void);
  static void addComplexExpr(RLD_Entry *);
  static uint16_t delComplexExpr(uint16_t);
```

90.
```
#define INITIAL_COMPLEX_EXPR_LIST_SIZE 10
  static int complexRefIsEmpty(void)
  {
    return (CExprList.pool[0].link ≡ 0);
  }
```

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

91. ⟨ Инициализация списка сложных выражений 91 ⟩ ≡

```
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.

92. ⟨ Освободить список сложных выражений 92 ⟩ ≡

```
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.

93. 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;
}
```

94. ⟨ Глобальные переменные 9 ⟩ +≡

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

95. 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: ;
  }
}
```

96. 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);
}
```

97. 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( ));
    }
```

98. 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);
}
```

99. 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);
}
```

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

101.

```
#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 101 ⟩ ≡

```
  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 70.

102. ⟨ Обработать глобальные символы и ссылки 102 ⟩ ≡

```
  handleGlobalSymbol(entry);
```

Этот код используется в секции 21.

103. ⟨ Обработать программную секцию 103 ⟩ ≡

```
  handleProgramSection(entry);
```

Этот код используется в секции 21.

104. ⟨ Обработать секцию TXT 104 ⟩ ≡

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

Этот код используется в секции 18.

105. ⟨ Обработать секцию перемещений 105 ⟩ ≡

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

Этот код используется в секции 18.

106. ⟨ 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 ) ;
```

107. Вспомогательные функции.
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';
}
```

108. ⟨ Глобальные переменные 9 ⟩ +≡

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

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

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

110. ⟨ Константы 110 ⟩ ≡

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

Этот код используется в секции 8.

111. ⟨ Глобальные переменные 9 ⟩ +≡

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

112. 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, args_doc, argp_program_doc};
```

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

⟨ Собственные типы данных 15 ⟩ +≡

```
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;
```

114. ⟨ Глобальные переменные 9 ⟩ +≡

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

115. 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);
}
```

116.

```
#define ERR_SYNTAX 1
#define ERR_CANTOPEN 2
#define ERR_CANTCREATE 3
⟨ Разобрать командную строку 116 ⟩ ≡
  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.

117. ⟨ Включение заголовочных файлов 117 ⟩ ≡

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

Этот код используется в секции 8.

118.

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

119. Макросы для монитора БК11М.

Файл lib/bk11m/bk11m.inc

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

                ; Драйвер экрана
                .MCALL  .BTSET,.BPAL
                .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
```

120. 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
```

121. Макросы для RAM-BIOS.

Файл lib/ram-bios/ram-bios.inc

```
;; vim: set expandtab ai textwidth=80:
;;
;; Специальные значение имени сегментов:
NASEG.Dos=377          ; DOS режим
NASEG.Current=376      ; текущий режим

                ; Выход в ДОС
                .MCALL  EXIT$
SYSPAG$=2160
DOS$=120020     ; системная ячейка, содержит адрес TOP.
                ; Формат Области TOP
                ; TOP:   <код режима (DOS)> -- код исходного режима, того,
                ; в котором работает ДОС с оболочкой; изначально это Std10
                ; или Std11, т.е. стандартный режим БК10 или 11М, коды 60
                ; или 140; если программа перемещает ДОС в другой режим
                ; (например, Halt10 или Halt11), то она должна корректировать
                ; и эту ячейку
                ; TOP-2:  < old PC > -- псевдо-адрес возврата -- 100000;
                ; TOP-4:  <PS & @#6> -- здесь -- 0;
                ; TOP-6:  < @#4 > - копия ячейки 4 для ее восстановления -- 100442;
                ; TOP-10: <код текущ режима>     - при запуске и EXIT$
                ; устанавливается равным коду режима DOS, указатель WRK$
                ; устанавливается = TOP-10 (таким образом, WRK$ указывает
                ; на ячейку с кодом текущего режима).

                ; Вызов подпрограммы
                .MCALL  CAL$G

                ; Чтение или обмен слова
                .MCALL  CHW$G

                ; Копирование или обмен массивов
                .MCALL  MOV$G

                ; Работа с каталогом
                .MCALL  CAT$

;; Структура записи каталога модулей
Cat.Name=0              ; Имя модуля (состоит из двух байтов:
                        ;            1 байт - идентификатор автора,
                        ;            1 байт - идентификатор модуля)
                        ; Авторы: "$" - системная
                        ;         "R" - все Reiter'ы
                        ; Лучше называть модули печатаемыми символами
Cat.Len=2               ; длина модуля в словах
Cat.Flags=4             ; флаги модуля.
Cat.ParName=6           ; Имя родительского модуля
Cat.ExNameOff=10        ; Смещение до расширенного имени модуля (ASCIZ)
Cat.Off=12              ; Смещение от начала сегмента до модуля.
Cat.Seg=14              ; Логический номер сегмента начала модуля (один байт).
```

```
51                          ; Резерв (1 байт).
52
53
54 ;; Флаги модуля в записи каталога:
55 Cat.Flags.NoMove=2       ; Перемещаемость модуля (0 - да). При исчерпании памяти
56                          ; модуль может быть перемещен в другое место. Если
57                          ; планируется обращаться к модулю на среднем или низком
58                          ; уровне нужно устанавливать в 1.
59 Cat.Flags.NoSplit=4      ; Разделение модуля на два подключаемых подряд сегмента
60                          ; (0 - да). Максимальная длина 10000 слов.
61 Cat.Flags.NoDelOnExit=10  ; Удалять модуль по запросу EXIT$ или перезапуску (0 -
62                            ; да)
63 Cat.Flags.Run=20         ; Возможность запуска модуля (1 - да, актуально для
64                          ; модулей, содержащих программы). Установка этого флага
65                          ; приводит к размещению по адресам от 120000 до 140000
66 Cat.Flags.Del=40         ; Подлежит удалению, если родительский модуль в каталоге
67                          ; отсутствует. (1 - да).
68 Cat.Flags.RunOnDel=100   ; Перед удалением модуля запускать END-2 (по CALL)?
69 Cat.Flags.ExName=200     ; Есть расширенное имя модуля.
70
71 ;; Команды работы с каталогом модулей
72 CreateModule=0
73 DeleteModule=2
74 ReadWriteCatRecord=10
75 GetVersion=12
76
77
```

Файл lib/ram-bios/MOV$G.MAC

```
1 ;; vim: set expandtab ai textwidth=80:
2 ;; Пересылается из 0 в 1. Отрицательная длина означает обмен.
3                 .MACRO  MOV$G NASEG0,ADRSME0,NASEG1,ADRSME1,LEN
4                 .IF DIF ADRSME0 R0
5                 mov     ADRSME0,R0
6                 .ENDC
7                 .IF DIF ADRSME1 R1
8                 mov     ADRSME1,R1
9                 .ENDC
10                .IF DIF LEN R2
11                mov     LEN,R2
12                .ENDC
13                JSR     R5,@#167520
14                .WORD   NASEG0,NASEG1
15                .ENDM
16
```

Файл lib/ram-bios/CHW$G.MAC

```
1 ;; vim: set expandtab ai textwidth=80:
2 ;; Чтение или обмен одного слова
3 ;; R0 -- обмен/чтение.
4 ;; R1 -- 0 - обмен, иначе чтение.
5 ;; Если при возврате R1 = 0 после возврата, то ошибка.
6                 .MACRO  CHW$G ADRSME, NASEG
7                 JSR     R5,@#167517
```

```
8                 .WORD   ADRSME, NASEG
9                 .ENDM
10
```

Файл lib/ram-bios/CAT$.MAC

```
1  ;; vim: set expandtab ai textwidth=80:
2  ;; CreateModule
3  ;; R0=0 -- создание модуля (создание записи о модуле в каталоге).
4  ;;              R1 -- адрес записи, которую нужно занести в каталог (ниже
5  ;;                      100000);
6  ;;              R2 -- код требуемого сегмента, если нужен определенный сегмент,
7  ;;                      или 0 -- любой. В процессе обработки R2 очищается по
8  ;;                      маске #74000;
9  ;;              Если установлен бит #4 во флагах, то учитываются:
10 ;;              - флаг #20;
11 ;;              - код сегмента (т.е. R2 неравный 0);
12 ;;              - требуемое смещение от начала сегмента.
13 ;;
14 ;; GetVersion
15 ;; R0=12 -- возвращает информацию о RAM-BIOS.
16 ;;              R0 -- номер версии.
17 ;;              R1 -- физический адрес начала каталога модулей
18 ;;                 (он находится по данному адресу в системной странице в
19 ;;                  SYS-режиме, код включения #2160)
20 ;;              R2 -- количество записей в каталоге.
21                .MACRO  CAT$ NUMBER
22                .IF DIF R0 NUMBER
23                mov     NUMBER,R0
24                JSR     R5,@#167521
25                .ENDC
26                .ENDM
27
```

Файл lib/ram-bios/CAL$G.MAC

```
1  ;; vim: set expandtab ai textwidth=80:
2  ;;
3                 .MACRO  CAL$G ADRSME,NASEG
4                 JSR     R5,@#167516
5                 .WORD   ADRSME, NASEG
6                 .ENDM
7
```

Файл lib/ram-bios/EXIT$.MAC

```
1  ;; vim: set expandtab ai :
2  ;; Инициализирует RAM-BIOS (в частности очищает стек возвратов из подпрограмм,
3  ;; вызванных по запросам CAL$G и CAL$N) и выходит в режим ‘‘DOS’’,
4  ;; обычно на 100000.
5  ;; Таким образом, обеспечивается корректный выход в ДОС из любого режима. Также
6  ;; очищает стек сохранений (используемый запросами SAVE$ и NEW$SP).
7  ;; Кроме того удаляются модули, у которых установлен признак удаления по EXIT$,
8  ;; а также все модули. у которых отсутствует родительский модуль и при
9  ;; этом их удалять не запрещено.
10                .MACRO  EXIT$
11                JSR     R5,@#167527
12                .ENDM
```

13

122. Индекс.

__*linux*__: 117.
_Arguments: 113.
_BinaryBlock: 15.
_ComplexExprEntry: 88.
_ComplexExpressionList: 88.
_ComplexTerm: 88.
_CurSectEntry: 83.
_GSD_Entry: 20.
_GSymDefEntry: 24.
_LimList: 61.
_LimListEntry: 61.
_RLD_Const_Entry: 69.
_RLD_Entry: 69.
_SectionDirEntry: 32.
_SimpleRefEntry: 47.
_SimpleRefList: 47.
a: 118.
abort: 50, 63, 93.
addComplexExpr: 89, 93, 101.
addComplexTerm: 94, 95, 101.
addLimit: 62, 63, 78.
addr: 24, 27, 31, 38, 56, 57, 58, 59, 98.
addSimpleRef: 50, 54, 73, 74, 75, 76.
arg: 115.
argc: 8, 115, 116.
argp: 112, 116.
ARGP_ERR_UNKNOWN: 115.
ARGP_KEY_ARG: 115.
argp_option: 112.
argp_parse: 116.
argp_program_bug_address: 110.
argp_program_doc: 111, 112.
argp_program_version: 110.
argp_state: 112, 115.
args_doc: 111, 112.
Arguments: 113, 114, 115.
arguments: 115.
argv: 8, 115, 116.
atoi: 115.
avail: 47, 50, 51, 52, 53, 88, 91, 92, 93, 96.
BinaryBlock: 14, 15, 16.
block: 38, 70, 101.
block_body: 16, 17, 18, 21, 104, 105.
block_len: 16, 19, 104, 105.
calcTerms: 97, 99, 100.
calloc: 43.
CExprList: 12, 89, 90, 91, 92, 93, 95, 96, 97.
cmd: 50, 69, 70.
code: 12, 88, 95, 98, 99.
ComplexExprEntry: 88, 91, 93, 97, 98, 99, 100.
ComplexExpressionList: 88, 89.
complexRefIsEmpty: 12, 89, 90, 97.
ComplexTerm: 88, 95, 99.
config: 8, 12, 13, 16, 18, 21, 27, 31, 34, 45, 53, 65, 70, 92, 114, 116, 118.
const_entry: 36, 70, 75, 76, 81, 82.
constant: 36, 47, 50, 57, 59, 69, 75, 76, 81, 82, 88, 95, 98, 99.
crc: 16.
CREL_OP_ADDITION: 99, 101.
CREL_OP_AND: 99, 101.
CREL_OP_COM: 99, 101.
CREL_OP_DIVISION: 99, 101.
CREL_OP_FETCH_CONSTANT: 95, 98, 99, 101.
CREL_OP_FETCH_GLOBAL: 12, 95, 98, 101.
CREL_OP_FETCH_RELOCABLE: 95, 98, 101.
CREL_OP_MULTIPLICATION: 99, 101.
CREL_OP_NEG: 99, 101.
CREL_OP_NONE: 99, 101.
CREL_OP_OR: 99, 101.
CREL_OP_STORE_RESULT: 97, 99, 101.
CREL_OP_STORE_RESULT_DISP: 99, 101.
CREL_OP_SUBSTRACTION: 99, 101.
CREL_OP_XOR: 99, 101.
cur_input: 8, 9, 16, 18, 21, 27, 34, 50, 70, 93.
CurComplexExpr: 93, 94, 95.
CurSect: 27, 34, 35, 36, 38, 43, 50, 63, 71, 72, 79, 80, 81, 82, 86, 93.
CurSectEntry: 83, 84.
curSections: 84, 86, 87, 98.
DEFAULT_SECTION_LEN: 43.
delComplexExpr: 89, 96, 97.
delSimpleRef: 51, 54, 55.
dest_addr: 55, 56, 57, 58, 59, 70, 71, 72, 79, 80, 81, 82, 97.
dest_dir: 67.
disp: 12, 47, 50, 53, 56, 57, 58, 59, 61, 63, 65, 67, 69, 71, 72, 73, 74, 75, 76, 78, 79, 80, 81, 82, 88, 92, 93, 95, 97, 98, 101.
end: 16.
ent: 69.
entry: 21, 22, 27, 34, 36, 37, 43, 45, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 97, 98, 99, 101, 102, 103.
EOF: 16.
ERR_CANTCREATE: 13, 116.
ERR_CANTOPEN: 11, 116.
ERR_SYNTAX: 116.
error_t: 112, 115.
exit: 16, 27, 34.
EXIT_FAILURE: 16, 27, 34.
fclose: 8, 13.

feof: 16.
fgetc: 16.
FILENAME_MAX: 113, 115.
findGlobalSym: 27, 28, 29, 55, 98.
findSection: 34, 36, 37, 42, 44, 79, 80, 81, 82.
first_byte: 16.
flags: 13, 20, 21, 24, 27, 32, 34, 43, 45.
fmt: 118.
fobj: 8, 10, 11, 16.
fopen: 11, 13.
found: 28, 42.
found_sym: 27.
fprintf: 118.
fread: 16.
free: 41, 53, 65, 92.
fresult: 10, 13.
fromRadix50: 12, 13, 22, 27, 31, 34, 36, 41, 53, 65, 73, 74, 75, 76, 79, 80, 81, 82, 87, 92, 101, 107, 108.
fwrite: 13.
GDS_IDENT: 19, 21.
global: 55, 56, 57, 58, 59, 98.
GLOBAL_DEFINITION_MASK: 27.
GLOBAL_RELOCATION_MASK: 27.
global_sect: 83, 86, 87, 98.
GLOBAL_WEAK_MASK: 27.
gname: 36, 70, 73, 74, 75, 76, 79, 80, 81, 82, 101.
GSD_CSECT_NAME: 19, 21.
GSD_Entry: 20, 21, 27, 34, 106.
GSD_GLOBAL_SYMBOL_NAME: 19, 21.
GSD_INTERNAL_SYMBOL_NAME: 19, 21.
GSD_MAPPED_ARRAY: 19, 21.
GSD_MODULE_NAME: 19, 21.
GSD_PSECT_NAME: 19, 21.
GSD_TRANFER_ADDRESS: 19, 21.
GSymDef: 25, 27, 28, 31, 56, 57, 58, 59, 98.
GSymDefEntry: 24, 25.
handleGlobalSymbol: 27, 102, 106.
handleGSD: 19, 21, 108.
handleOneFile: 8, 16, 108.
handleProgramSection: 34, 103, 106.
handleRelocationDirectory: 70, 105, 106.
handleTextSection: 38, 104, 106.
i: 8, 16, 21, 28, 42, 67, 97, 98, 99, 107.
INITIAL_COMPLEX_EXPR_LIST_SIZE: 90, 91.
INITIAL_LIMIT_LIST_SIZE: 61, 63, 64.
INITIAL_SIMPLE_REF_LIST_SIZE: 47, 50, 52, 93.
input: 115.
inter: 88, 95, 98.
int32_t: 32.
j: 8.
key: 115.
last_load_addr: 32, 38, 50, 63, 71, 72, 79, 80, 81, 82, 93.
len: 13, 14, 15, 16, 21, 32, 34, 38, 41, 43, 67, 70.
level: 118.
LIMIT: 61.
LimitList: 62, 63, 64, 65, 67.
LimList: 61, 62.
LimListEntry: 61, 63, 64.
link: 12, 47, 49, 50, 51, 52, 53, 55, 61, 88, 90, 91, 92, 93, 96, 97.
main: 8.
malloc: 52, 64, 91.
MAX_COMPLEX_TERMS: 88, 99.
max_filename_len: 13, 113, 115.
MAX_GLOBALS: 24, 25.
MAX_PROG_SECTIONS: 32, 33, 84.
memcpy: 38.
memset: 39.
min_addr: 13, 32, 36, 41, 43, 67.
n: 107.
name: 12, 13, 16, 20, 21, 22, 24, 27, 28, 30, 31, 32, 34, 37, 41, 42, 43, 47, 50, 53, 55, 65, 83, 86, 87, 88, 92, 95, 98, 107.
new_entry: 50, 63, 93.
new_index: 50, 93.
new_memory: 50, 63, 93.
next: 115.
not_resolved: 8, 98.
num_allocations: 47, 50, 52, 61, 63, 64, 88, 91, 93.
num_limits: 61, 63, 64, 65, 67.
num_start_addresses: 8, 9, 37.
NumCurSections: 84, 85, 86, 87.
NumGlobalDefs: 25, 26, 27, 28, 31.
NumSections: 13, 33, 39, 41, 42, 43.
NumTerms: 12, 88, 95, 98, 99.
obj_file: 12, 24, 27, 47, 50, 53, 88, 92, 93.
obj_header: 16.
objname: 8, 11.
objnames: 8, 12, 16, 18, 21, 27, 34, 53, 70, 92, 113, 115, 116.
one: 15, 16.
options: 112.
output_filename: 13, 113, 115, 116.
ovrname: 10, 13.
parse_opt: 112, 115.
pool: 12, 47, 49, 50, 51, 52, 53, 55, 56, 57, 58, 59, 61, 63, 64, 65, 67, 88, 90, 91, 92, 93, 95, 96, 97.
poolmin: 47, 50, 52, 53, 88, 91, 92, 93.
prev: 97.
prev_ref: 55.
PRINTERR: 8, 11, 13, 16, 18, 21, 27, 34, 50, 63, 70, 93, 99, 101, 116, 118.

printf: 12, 118.
PRINTVERB: 16, 18, 21, 27, 31, 36, 38, 41, 45, 50, 53, 63, 65, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 87, 92, 93, 101, 118.
PSECT_ACCESS_MASK: 34, 45.
PSECT_ALLOCATION_MASK: 34, 45.
PSECT_RELOCATION_MASK: 34, 45.
PSECT_SAVE_MASK: 13, 34, 45.
PSECT_SCOPE_MASK: 34, 45.
PSECT_TYPE_MASK: 34, 43, 45.
realloc: 50, 63, 93.
ref: 50, 55, 56, 57, 58, 59, 63, 93.
ref_i: 51, 96.
resolveComplex: 8, 97, 100.
resolveGlobals: 8, 55, 60.
resolveLimit: 62, 66, 67.
resolveTerms: 97, 98, 100.
result_type: 88, 97, 99.
RLD_CMD_COMPLEX_RELOCATION: 70.
RLD_CMD_GLOBAL_ADDITIVE_DISPLACED_RELOCATION: 55, 70.
RLD_CMD_GLOBAL_ADDITIVE_RELOCATION: 50, 55, 70.
RLD_CMD_GLOBAL_DISPLACED_RELOCATION: 55, 70.
RLD_CMD_GLOBAL_RELOCATION: 55, 70.
RLD_CMD_INTERNAL_DISPLACED_RELOCATION: 70.
RLD_CMD_INTERNAL_RELOCATION: 70.
RLD_CMD_LOCATION_COUNTER_DEFINITION: 70.
RLD_CMD_LOCATION_COUNTER_MODIFICATION: 70.
RLD_CMD_PROGRAM_LIMITS: 70.
RLD_CMD_PSECT_ADDITIVE_DISPLACED_RELOCATION: 70.
RLD_CMD_PSECT_ADDITIVE_RELOCATION: 70.
RLD_CMD_PSECT_DISPLACED_RELOCATION: 70.
RLD_CMD_PSECT_RELOCATION: 70.
RLD_Const_Entry: 36, 50, 69, 70, 75, 76, 81, 82.
RLD_Entry: 50, 54, 62, 63, 69, 70, 89, 93.
RLD_i: 36, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 101.
sect: 12, 21, 24, 27, 31, 37, 47, 50, 53, 56, 57, 58, 59, 61, 63, 65, 67, 70, 79, 80, 81, 82, 88, 92, 93, 95, 97, 98.
sect_name: 12, 13, 31, 40, 41, 53, 65, 92.
SectDir: 12, 13, 27, 31, 33, 34, 36, 37, 38, 39, 41, 42, 43, 50, 53, 56, 57, 58, 59, 63, 65, 67, 71, 72, 79, 80, 81, 82, 86, 92, 93, 97, 98.
SectionDirEntry: 32, 33.
SimpleRefEntry: 47, 50, 52.
simpleRefIsEmpty: 12, 48, 49, 55.
SimpleRefList: 47, 48.
sp: 99.
SRefList: 12, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 59.
stack: 99.
start: 27, 32, 34, 36, 38, 41, 71, 79, 80, 81, 82, 98.
state: 115.
static: 112.
stderr: 118.
strcat: 13.
strlen: 13, 115, 116.
strncpy: 13, 115.
term: 95, 99.
terms: 12, 88, 95, 98, 99.
text: 13, 32, 38, 41, 43, 56, 57, 58, 59, 67, 71, 72, 79, 80, 81, 82, 97.
TEXT: 38, 68.
transfer_addr: 13, 32, 37, 43.
type: 20, 21, 47, 50, 55, 69, 70.
uint16_t: 15, 16, 20, 24, 28, 29, 32, 38, 42, 44, 47, 50, 51, 54, 55, 56, 57, 58, 59, 61, 67, 69, 70, 71, 72, 79, 80, 81, 82, 83, 88, 89, 93, 94, 95, 96, 97, 98, 99, 100, 101.
uint8_t: 15, 17, 20, 24, 32, 38, 43, 47, 61, 69, 70, 83, 88, 94, 95, 106.
un: 12, 88, 95, 98, 99.
ungetc: 16.
value: 20, 21, 27, 34, 36, 37, 43, 50, 69, 70, 71, 72, 73, 74, 75, 76, 77, 79, 80, 81, 82, 97, 101.
verbosity: 27, 31, 45, 53, 65, 92, 113, 115, 118.
VERSION: 109, 110.
x: 107.
zero: 15, 16.

⟨ Включение заголовочных файлов 117 ⟩ Используется в секции 8.
⟨ Вывести отладочную информацию по секциям 45 ⟩ Используется в секции 34.
⟨ Вывести перекодировку секций 87 ⟩ Используется в секции 18.
⟨ Вывод неразрешенных ссылок 12 ⟩ Используется в секции 8.
⟨ Вывод таблицы глобальных символов 31 ⟩ Используется в секции 8.
⟨ Глобальные переменные 9, 17, 25, 29, 33, 35, 44, 48, 54, 60, 62, 84, 89, 94, 100, 106, 108, 111, 112, 114, 118 ⟩
Используется в секции 8.
⟨ Данные программы 10, 30, 40 ⟩ Используется в секции 8.
⟨ Добавить перекодировку секции 86 ⟩ Используется в секции 34.
⟨ Добавить программную секцию 43 ⟩ Используется в секции 34.
⟨ Заполнить пределы секций 66 ⟩ Используется в секции 8.
⟨ Изменение текущей позиции 77 ⟩ Используется в секции 70.
⟨ Инициализация каталога секций 39 ⟩ Используется в секции 8.
⟨ Инициализация списка пределов 64 ⟩ Используется в секции 8.
⟨ Инициализация списка сложных выражений 91 ⟩ Используется в секции 8.
⟨ Инициализация списка ссылок без констант 52 ⟩ Используется в секции 8.
⟨ Инициализация таблицы глобальных символов 26 ⟩ Используется в секции 8.
⟨ Константы 110 ⟩ Используется в секции 8.
⟨ Косвенная смещенная ссылка на секцию 82 ⟩ Используется в секции 70.
⟨ Косвенная ссылка на абсолютный адрес 72 ⟩ Используется в секции 70.
⟨ Косвенная ссылка на глобальный символ 74 ⟩ Используется в секции 70.
⟨ Косвенная ссылка на секцию 80 ⟩ Используется в секции 70.
⟨ Косвенная ссылка на смещенный глобальный символ 76 ⟩ Используется в секции 70.
⟨ Обработать блок 18 ⟩ Используется в секции 16.
⟨ Обработать глобальные символы и ссылки 102 ⟩ Используется в секции 21.
⟨ Обработать программную секцию 103 ⟩ Используется в секции 21.
⟨ Обработать секцию перемещений 105 ⟩ Используется в секции 18.
⟨ Обработать секцию TXT 104 ⟩ Используется в секции 18.
⟨ Освободить список пределов 65 ⟩ Используется в секции 8.
⟨ Освободить список сложных выражений 92 ⟩ Используется в секции 8.
⟨ Освободить список ссылок 53 ⟩ Используется в секции 8.
⟨ Открыть объектный файл 11 ⟩ Используется в секции 8.
⟨ Очистка каталога секций 41 ⟩ Используется в секции 8.
⟨ Прямая смещенная ссылка на секцию 81 ⟩ Используется в секции 70.
⟨ Прямая ссылка на абсолютный адрес 71 ⟩ Используется в секции 70.
⟨ Прямая ссылка на глобальный символ 73 ⟩ Используется в секции 70.
⟨ Прямая ссылка на секцию 79 ⟩ Используется в секции 70.
⟨ Прямая ссылка на смещенный глобальный символ 75 ⟩ Используется в секции 70.
⟨ Разобрать командную строку 116 ⟩ Используется в секции 8.
⟨ Разобрать GSD 19 ⟩ Используется в секции 18.
⟨ Разрешить косвенную ссылку 58 ⟩ Используется в секции 55.
⟨ Разрешить прямую ссылку 56 ⟩ Используется в секции 55.
⟨ Разрешить смещенную косвенную ссылку 59 ⟩ Используется в секции 55.
⟨ Разрешить смещенную прямую ссылку 57 ⟩ Используется в секции 55.
⟨ Распаковать имя 22 ⟩ Используется в секции 21.
⟨ Сбросить перекодировку секций 85 ⟩ Используется в секции 16.
⟨ Сложная ссылка 101 ⟩ Используется в секции 70.
⟨ Собственные типы данных 15, 20, 24, 32, 47, 61, 69, 83, 88, 113 ⟩ Используется в секции 8.
⟨ Создаем файл результата 13 ⟩ Используется в секции 8.
⟨ Установить адрес запуска 37 ⟩ Используется в секции 21.
⟨ Установка пределов 78 ⟩ Используется в секции 70.
⟨ Установка текущей секции и позиции 36 ⟩ Используется в секции 70.

LINKER

	Секция	Страница
Примеры несложных программ	1	2
Примеры с перемещаемыми секциями (атрибут SAV)	6	9
Общая схема программы	8	15
Обработка объектного файла	14	19
GSD	19	22
Списки ссылок на глобальные символы	46	31
Разрешение ссылок на глобальные символы	55	34
Обработка пределов (. `LIMIT`) для секций	61	36
Каталоги перемещений	68	38
Обработка сложных ссылок	83	43
Вспомогательные функции	107	53
Разбор параметров командной строки	109	54
Макросы для монитора БК11М	119	57
Макросы для MKDOS	120	59
Макросы для RAM-BIOS	121	61
Индекс	122	65