シリーズ ソースコードを読む

芳之内 弘

LILOブートアップ・ソースコードの解析④

LILOのソースコードを解析し、Linuxの起動の仕組みに迫ります。

LILOのソースコード解析の第4回目です。今月も、2000年5月号、2000年6月号に引続き、ブートアップ処理を行なっている「second.S」を説明していきます。second.Sについては、今回で終りです。例によって記事末には、処理の流れを分かりやすくするフローチャートを掲載しました。ソースコード解説と合わせて参照することで、理解を早められるでしょう。(編集部)

リスト1　※フローチャート1参照

```
パスワードのチェック
0595  dopw:   push    bx

(今回は省略)

ブート処理
0655
0656  doboot: mov     byte ptr prechr,#61          ! prechr="a"
0657          push    bx                           ! BX=ディスクリプタテーブル内の起動イメージ名のポインタを保存する
0658          mov     bx,#msg_l                    ! #msg_1は'Loding'のアドレス
0659          call    say                          ! Lodingを表示する
0660          pop     bx                           ! BX=ブートイメージ名のポインタ
0661          push    bx                           ! BX=を保存
0662          call    say                          ! ブートするイメージ名を表示する
0663          pop     si                           ! SI=ディスクリプタテーブルのブートイメージ名のポインタ
0664          add     si,#ADDR_OFFS                ! SI=SI+0x20  SIはディスクリプタテーブル内のRAMディスクサイズのポインタ
0665          cld                                  ! lodswでSIは2増加する(ワードのロード)

RAMディスクイメージのサイズのチェック
0667          mov     word ptr (rdmid),#0          ! (rdmid)=0  RAMディスクアドレスのクリア
0668          lodsw                                ! AX=RAMディスクサイズの下位ワード SI=SI+2
0669          mov     dx,ax                        ! DX=RAMディスクサイズ(Low)
0670          mov     rdszl,ax                     ! rdszl=RAMディスクサイズ(Low)
0671          lodsw                                ! AX=RAMディスクサイズの上ワード
0672          mov     rdszh,ax                     ! rdszh=RAMディスクサイズ(High)
0673          xchg    ax,dx                        ! DX=RAMディスクサイズ(High)、AX=RAMディスクサイズ(Low)
0674          add     ax,#4095                     ! 1ページ=4096バイト(ページ余り分を確保)
0675          adc     dx,#0                        ! 加算結果のキャリーをDXに加える
0676          mov     bx,#4096                     ! BX=4096  1ページ=4096バイト
0677          div     bx                           ! AX=(DX:AX)/BX  DX=剰余
0678          mov     bx,ax                        ! BX=AX  RAMディスクのページ数
0679          lodsw                                ! AX=RAMディスクマップテーブルのセクタアドレスのCX値
0680          mov     cx,ax                        ! CX=AX
0681          lodsw                                ! AX=RAMディスクマップテーブルのセクタアドレスのDX値
0682          mov     dx,ax                        ! DX=AX
0683          inc     si                           ! AL値をスキップする(AL値=01)
0684          or      bx,bx                        ! RAMディスクページ数=0か？  BX=0だとZF=1にセットされる
0685          jz      noramd                       ! Yesでnoramd(0731)へ

RAMディスクのマップテーブルのセクタアドレスをセットする
0686          push    si                           ! SI、ESとBX(RAMディスクページ数)を退避
```

リスト1 （続き）

```
0687            push    es
0688            push    bx
0689            mov     bx,#MAP                 ! BX=マップテーブルのバッファアドレス
0690            mov     al,#1                   ! 読み込むセクタ数=1
0691            call    sread                   ! RAMディスクマップテーブルを1セクタ、マップへ読み込む
0692            mov     moff,#0                 ! moff=0
0693    DEBUG
0694            mov bx,#stepa                   ! BX=" RAM disk, "のアドレス
0695            call say                        ! " RAM disk, "を表示する

0x100,000以上のメモリ(拡張メモリ)のサイズを得る
0697            mov     ah,#0x88                ! 拡張メモリのサイズを得る(1kBytes単位)
0698            int     0x15                    ! AX=メモリサイズ
0699            cmp     word ptr memlim,#0      ! メモリサイズリミットが宣言されているか？
0700            je      noulim                  ! Noでnoulim(0703)へ
0701            mov     ax,memlim               ! AX=memlim(制限値1kBytes単位)
0702            sub     ax,#1024                ! 1MBytes減じる(拡張メモリは1MBytes以上だから)
0703    noulim: cmp     ax,#0x3c00              ! 0x3c00+0x400=0x4000(これは16MBytes)
0704            jbe     no16mb                  ! メモリが16MBytes以下ならno16mb(0707)へ
0705            mov     ax,#0x3c00              ! BIOSコールでは16MBytes以下しかアクセスできない
0706                                            ! AX=0x3c00( 最大メモリサイズ)に設定する
0707    no16mb: shr     ax,1                    ! AX=AX/2  メモリサイズをページに変換する
```

リスト2 ※フローチャート2参照

```
0707    no16mb: shr     ax,1                    ! AX=AX/2  メモリサイズをページに変換する
0708            shr     ax,1                    ! AX=AX/2=0x0F00  (1ページ=4096Bytes)
0709            pop     bx                      ! BX=RAMディスクページ数(688行目でpush)
0710            sub     ax,bx                   ! AX=AX-BX
0711            jnc     rdokay                  ! AXがマイナスでないときはrdokay(0717)へ
0712            cmp     ax,#128                 ! AX=128  カーネルには128ページ以上必要
0713            ja      rdokay                  ! 残りが128ページ以上あればrdokay(0717)へ
0714            mov     bx,#msg_rd              ! 128ページ以下は問題あり(BX=#msg_rd)
0715            call    say                     !"not enough memory for RAM disk "を表示する
0716            jmpi    restrt,SECONDSEG        ! restrt(0142)へ  セカンドローダのrestart

GDTの転送先のアドレスを設定する(転送元アドレス=0x10000ですでに設定済み)
0717    rdokay: add     ax,#256                 ! AX=AX+256(256ページ=1MBytes)
0718            shl     ax,1                    ! 2倍
0719            shl     ax,1                    ! 2×2=4倍
0720            shl     ax,1                    ! 2×2×2=8倍
0721            shl     ax,1                    ! 2×2×2×2=16倍
0722            mov     (gdt+0x1b),ax           ! GDT転送先アドレス上位ワード=AX
0723            mov     byte ptr (gdt+0x1a),#0  ! 下位バイト=0(256×16=4096)
0724            mov     (rdmid),ax              ! rdmid(RAMディスク中位アドレス)=AX=転送先アドレス上位ワード
0725            xor     ax,ax                   ! AX=0
0726            mov     es,ax                   ! ES=0(GDTを使い1MBytes以上にコピーするときの暗号)
0727            mov     bx,#gdt                 ! BX=GDTのアドレス
0728            call    lfile                   ! 転送先へコピーする
                                                ! RAMディスクのマップテーブルに従ってRAMディスクイメージを0x10000に読み込み、その都
                                                ! 度、GDTテーブルの転送先(1MBytes以上のメモリにあるRAMディスク)へコピーする。複数ペー
                                                ! ジのマップテーブルは1ページごと読み込まれる。そして最後のセクタアドレスはNULLでター
                                                ! ミネトされ、読み込みを終了する。
0729            pop     es                      ! ESとSIを戻す(687行目でpush)
0730            pop     si                      ! 686行目でpush

カーネルノマップテーブルのセクタアドレスをセットする(RAMディスクイメージがない場合は直接ここへ来る)
0731    noramd: lodsw                           ! AX=カーネルのマップテーブルのCX値
0732            mov     cx,ax                   ! CX=AX
0733            lodsw                           ! AX=マップテーブルのDX値
0734            mov     dx,ax                   ! DX=AX
0735            inc     si                      ! AL値をスキップ  AL=01
0736            mov     word ptr (gdt+0x1b),#0  ! GDTの転送先アドレス上位ワード=0
                                                ! SI=SI+2(SI=ディスクリプタテーブルのフラグをポイントする)
0737            lodsw                           ! 転送先のスタートページ
0738            or      ax,ax                   ! AL=0か？  AL=0でZF=1になる
0739            jz      nohigh                  ! Yesでnohigh(0746)へ
                                                ! カーネルは0x100,000以下に読み込む
```

リスト2 （続き）

```
0740            shl     ax,1                    ! AXは2倍
0741            shl     ax,1                    ! さらに2倍(最初の4倍)
0742            shl     ax,1                    ! さらに2倍(最初の8倍)
0743            shl     ax,1                    ! さらに2倍(最初の16倍)
0744            mov     (gdt+0x1b),ax           ! GDTの転送先アドレスの上位ワード=AX
0745            mov     byte ptr (gdt+0x1a),#0  ! GDTの転送先アドレスの下位バイト=0
                                                ! GDTの転送先アドレスは3Bytes
                                                ! 256×16=4096(1ページ=4096Bytes)

0739から直接ここへジャンプすると1MBytes以下にカーネルを読み込む
0746    nohigh:
0747            push    si                      ! SI(ディスクリプタテーブルのポインタ)を退避(フラグをポイントしている)
0748    #ifdef DEBUG
0749            mov bx,#step0
0750            call say
0751    #endif
0752            mov     bx,#MAP                 ! BX=カーネルイメージのマップテーブルのバッファアドレス
0753            mov     al,#1                   ! AL=1(読み込むセクタ数)
0754            call    sread                   ! カーネルイメージのマップテーブルを1セクタ読み込む
0755            mov     moff,#0                 ! moff(マップテーブルオフセット)=0
0756    #ifdef DEBUG                            ! ここはデバッグ専用
0757            mov bx,#step0b
0758            call say
0759    #endif

デフォルトコマンドラインの読み込み
0760            mov     bx,#DFLCMD              ! BX=デフォルトコマンドラインのバッファアドレス
0761            seg     ss                      ! SEGMENT=SS=0x9000
0762            mov     cx,DFCMD_OFF+SSDIFF     ! CX=デフォルトコマンドラインのセクタアドレスのCX値
0763            seg     ss
0764            mov     dx,DFCMD_OFF+2+SSDIFF   ! DX=デフォルトコマンドラインのセクタアドレスのDX値
0765            mov     al,#1                   ! AL=1(読み込むセクタ数)
0766            call    cread                   ! デフォルトコマンドラインを1セクタ読み込む(1365)
0767            push    word ptr (DFLCMD)       ! デフォルトコマンドラインの最初のワードを退避

フォールバックセクタを読み込む
0768            mov     bx,#DFLCMD              ! BX=デフォルトコマンドラインのバアッファアドレス
0769            call    load1                   !(0951)フォールバックセクタを読み込む
                                                ! カーネルのマップテーブルの最初にはフォールバックセクタアドレスがある
0770            pop     ax                      ! AX=デフォルトコマンドラインの最初のワード
0771    #ifndef LCF_READONLY
0772            cmp     ax,#DC_MAGIC            ! AX=DC_MAGIG(0xf4f2)か？
0773            je      dclok                   ! Yesでdclock(0776)へ　書き戻し可
0774            cmp     ax,#DC_MGOFF            ! AX=DC_MGOFF(0x6b6d)か？
0775            jne     nofbck                  ! Noでnofbck(0785)へ　書き戻し禁止

デフォルトコマンドラインにフォールバックセクタを書き戻す
0776    dclok:  mov     bx,#DFLCMD              ! BX=デフォルトコマンドラインのバッファアドレス
0777            cmp     word ptr (bx),#DC_MAGIC ! 最初のワード=DC_MAGIC？
0778            jne     nofbck                  ! Noでnofbck(0785)へ
0779            seg     ss                      ! SEGMENT=SS
0780            mov     cx,DFCMD_OFF+SSDIFF     ! CX=デフォルトコマンドラインセクタアドレスのCX値
0781            seg     ss
0782            mov     dx,DFCMD_OFF+2+SSDIFF   ! DX=デフォルトコマンドラインセクタアドレスのDX値
0783            mov     al,#1                   ! AL=1(書き込むセクタ数)
0784            call    cwrite                  ! デフォルトコマンドラインへフォールバックセクタの内容を書き戻す
0785    nofbck:                                 ! 書き戻さないときは直接ここへジャンプ
```

リスト3 ※フローチャート3参照

```
書き戻さないときはここへ戻る
0786    #endif
0787    #ifdef DEBUG
0788            mov     bx,#step1
0789            call    say
0790    #endif

オプショナルセクタを読み込み、セットする
0791            mov     bx,#DFLCMD              ! BX=デフォルトコマンドラインのバッファアドレス
```

リスト3 （続き）

```
0792            call    load1                           ! オプショナルセクタを読み込む
                                                        ! カーネルのマップテーブルの2番目のセクタアドレスはオプショナルセクタ
0793            mov     si,cmdbeg                       ! SI=cmdbeg(コマンドのオプション以外をコピーする)
0794            mov     di,#PARMLINE                    ! DI=#PARLMINE=0x2A00(パラメータラインのアドレス)
0795    cpnocl: cmp     si,options                      ! SI=optionsか？(オプションの最初か？)
0796            je      cpnodn                          ! Yesでcpnodn(0799)
0797            movsb                                   ! AL=[DS:SI]  SI=SI+1
0798            jmp     cpnocl                          ! SI=optionsまで繰り返す
0799    cpnodn: mov     si,#DFLCMD                      ! SI=デフォルトコマンドラインアドレス
0800            cmp     byte ptr (si),#0                ! 最初のバイト=0か？
0801            je      nocopt                          ! Noでnocopt(0817)へ
0802            mov     al,#32                          ! AL=AL+0x32(スペース)
0803            stosb                                   ! (DI)=AL  この文字をパラメータラインに書き込む
                                                        ! DI=DI+1
0804    cpcodsp:lodsb                                   ! AL=cmdbegの次のバイト
0805            cmp     al,#32                          ! AL=0x32(スペース)か？
0806            je      cpcodsp                         ! Yesでcpcodsp(0804)スペースを捨て次のバイトを
0807    cpcolp: or      al,al                           ! AL=0でZF=1になる
0808            jz      cpcodn                          ! Yes(AL=0)でcpcodn(0814)へ
0809            stosb                                   ! ALをパラメータラインに書き込む(DI=DI+1)
0810            cmp     al,#32                          ! SL=0x32(スペース)か？
0811            je      cpcodsp                         ! Yesでcpcodsp(0804)スペースを捨て次のバイト
0812            lodsb                                   ! AL=cmdbegの次のバイト
0813            jmp     cpcolp                          ! cpcolp(0807)から繰り返す
0814    cpcodn: cmp     byte ptr (di-1),#32             ! 最後のバイトはスペースか？
                                                        ! SIはデーターが0のアドレスを指している
0815            jne     nocopt                          ! Noでnocopt(0817)へ
0816            dec     di                              ! DI=DI-1(コマンドラインのスペースを捨てる)
0817    nocopt: mov     si,options                      ! SI=options(オプションのポインタ)オプションの追加
0818    cpvalp: lodsb                                   ! AL=オプションの1文字(SI=SI+1)
0819            stosb                                   ! ALをパラメータラインに追加(DI=DI+1)
0820            or      al,al                           ! AL=0か？  AL=0でZF=1
0821            jnz     cpvalp                          ! Noでcpvalp(0818)から繰り返す
0822    #ifdef DEBUG
0823            mov     bx,#step2
0824            call    say
0825    #endif
```

カーネルのオリジナルブートセクタの読み込みとパラメータのセット

```
0826            mov     ax,#INITSEG                     ! AX=#INTSEG=0x9000
0827            mov     es,ax                           ! ES=0x9000
0828            xor     bx,bx                           ! BX=0  BX=バッファアドレス
0829            call    load1                           ! 0x90000にカーネルのオリジナルブートローダを読み込む
0830            seg     es                              ! ES=0x9000
0831            mov     CL_MAGIC_ADDR,#CL_MAGIC         ! CL_MAGIC_ADDR=0x20、CL_MAGIC=0xA33F
                                                        ! 0x90020にマジックナンバ(0xA3FF)をセット(これはカーネルへのパラメータを意味する)
0832            seg     es                              ! ES=0x9000
0833            mov     word ptr CL_OFFSET,#PARMLINE+SECOND_SS
                                                        ! パラメータラインアドレスセット
                                                        ! CL_OFFSET=0x22、PARMLIN=0x2A00、SECOND_SS=0x9B00
                                                        ! 0x90022に0x9000に対するパラメータラインのオフセット値=0xDA00をセットする
0834
0835            pop     si                              ! SI=ディスクリプタテーブルのポインタ(0747)でpush SI
0836            lodsw                                   ! AX=フラグ、SI=SI+2
0837            mov     bx,ax                           ! BX=フラグ
0838            lodsw                                   ! AX= VGAモード、SI=SI+2
0839            cmp     word ptr vgaovr,#VGA_NOCOVR     ! vgaovr=#VGA_NOCOVRか？
0840
0841            je      vganorm                         ! Yesでvganorm(0844)へ
0842            mov     ax,vgaovr                       ! NoでAX=vagovrにセット
0843            jmp     vgaset                          ! vgaset(0846)へ
0844    vganorm:test    bx,#FLAG_VGA                    ! FLAG_VGA=1(BXのbit0=1か？)
0845            jz      novga                           ! No(bit0=0)でnovga(0848)へ
0846    vgaset: seg     es                              ! ES=0x9000
0847            mov     506,ax                          ! [0x9000：506](VGAパラメタ)=AX(vgaovr)
0848    novga:  push    bx                              ! BX=フラグを待避
0749            test    bx,#FLAG_LOCK                   ! FLAG_LOCK=4(BXのbit2=1か？)
0850            jnz     lockit                          ! Yesでlockit(0853)へ
0851            cmp     byte ptr dolock,#0              ! dolock=0か？  新しいターゲットのlockを指定しているか？
0852            je      nolock                          ! Yesでnolock(0869)へ
```

FLAG_LOCKがONのとき、lkwbufの内容でデフォルトコマンドラインを書き戻す

```
0853    lockit:
```

リスト3 （続き）

```
0854  #ifndef LCF_READONLY
0855          mov     bx,#lkwbuf                  ! BX=#lkwbufコマンドラインを書き込む(0xF4F2+lkcbufで書き戻す)
0856          seg     ss                          ! SS=0x9000
0857          mov     cx,DFCMD_OFF+SSDIFF         ! CX=デフォルトコマンドラインのCX値
0858          seg     ss
0859          mov     dx,DFCMD_OFF+2+SSDIFF       ! DX=コマンドラインのDX値
0860          push    es                          ! ESを待避する
0861          mov     ax,ds                       ! AX=DS
0862          mov     es,ax                       ! ES=DS
0863          mov     ax,#1                       ! 書き込むセクタ数=1
0864          push    si                          ! SIを待避する
0865          call    cwrite                      ! デフォルトコマンドラインを書き込む
0866          pop     si                          ! SIを戻す
0867          pop     es                          ! ESを戻す
0868
0869  nolock:
```

カーネルのセットアップコードを読み込む

```
0871          mov     bx,#step3                   ! BX=メッセージ‘setup,’のアドレス
0872          call    say                         ! “setup’を表示する
0873          seg     es
0874
0875          mov     cx,#SETUPSECS               ! CX=4(読み込むセットアップセクタ数)
0876  #ifdef LCF_VARSETUP
0877          seg     es                          ! ES=0x9000
0878          cmp     byte ptr VSS_NUM,#0         ! VSS_NUM=497  [0x9000:497]=0か？
                                                  ! [0x9000:497]=セットアップコードのセクタ数
0879          je      lsetup                      ! Yes(0)でlsetup(0883)へ
0880          seg     es                          ! ES=0x9000
0881          mov     cl,VSS_NUM                  ! 0でないときはCL=その値をセット
0882  #endif
0883  lsetup: mov     ax,#SETUPSEG                ! AX=0x9020
0884          mov     es,ax                       ! ES=0x9020
0885          xor     bx,bx                       ! BX=0
0886  lsloop: push    cx                          ! CXを待避(CXはループカウンタ、初期値=4またはVSS_NUM)
0887          call    loadopt                     ! セットアップコードを1セクタ読み込む
0888          pop     cx                          ! CX=カウンタ
0889          loop    lsloop                      ! lsloopからCX=0まで繰り返す(CX=CX-1)
                                                  ! ここまででセットアップコードは0x90200に4セクタ分もしくはVSS_NUMセクタ読み込まれた
0890  #ifdef DEBUG
0891          mov     bx,#step4                   ! BX=メッセージ‘system,’のアドレス
0892          call    say                         ! “system,’を表示する
0893  #endif
0894          pop     bx                          ! BX=フラグ(0848でpush BXしたフラグを取り出す)
0895          test    bx,#FLAG_MODKRN             ! BX=#FLAG_MODKRN=8か？(モダンカーネルか？)
0896          jz      loadlow                     ! Noでloadlow(0940)へ(1MBytes以下へロード)
```

セットアップコードにパラメータを書き込む

```
0897          seg     es                          ! SEGMENT=ES=0x9020
0898          mov     byte ptr (16),#LOADER_VERSION ! ローダバージョン=1にセット
0899          seg     es
0900          cmp     word ptr (6),#NEW_HDR_VERSION ! ヘッダフォマットのバージョン番号を0x200と比較
0901          jbe     noheap                      ! 0x200より小でnoheap(0906)へ(ヒープをパッチしない)
0902          seg     es                          ! 0x200以上の場合
0903          mov     word ptr (36),#SLA_SIZE     ! セットアップコードサイズのアドレス
0904          seg     es
0905          or      byte ptr (17),#LFLAG_USE_HEAP ! FLAG_USE_HEAP=0x80
                                                  ! 後でセットアップコードを移動させるサイズ=0x80(0x80×0x100=0x8000バイト)
0906  noheap: cmp     word ptr (rdmid),#0         ! RAMディスクはあるか？ (0724)でセット
0907          je      nordpt                      ! ないときはnordpt(0922)へ
0908          xor     al,al                       ! RAMディスクのアドレスをセットする
0909          mov     ah,(rdmid)                  ! AH=rdmid(RAMディスク中位アドレスバイト)、AL=0
0910          seg     es                          ! AX=rdmid×256=RAMディスク下位アドレス
0911          mov     (24),ax                     ! RAMディスク下位アドレスをセット(4Bytesアドレス)
0912          xchg    al,ah                       ! AH=0
0913          mov     al,(rdmid+1)                ! AH=0、AL=(rmdim+1)
0914          seg     es
0915          mov     (26),ax                     ! RAMディスク上位アドレスセット(4Bytesアドレス)
                                                  ! メモリ(24 - 27)の内容＝ワード(rmdim)の256倍
0916          mov     ax,rdszl                    ! AX=rdszl=RAMディスクサイズLow(0670でセット)
0918          mov     (28),ax                     ! RAMディスクサイズLowをセット
0919          mov     ax,rdszh                    ! AX=rdszh=RAMディスクサイズHigh(0672でセット)
```

リスト3 （続き）

```
0920            seg     es
0921            mov     (30),ax                 ! RAMディスクサイズHighをセット
0922  nordpt:   cmp   word ptr (gdt+0x1b),#0     ! カーネル転送先アドレスの上位ワード=0か？
0923            je      loadlow                 ! Yes(0)ならloadlow(0940)へ(1MBytes以下へロードする)
0924  #if 0 /* not necessary! */
```

（必要ないので省略）

```
0934  #endif
0935            xor     ax,ax                   ! AX=0
0936            mov     es,ax                   ! ES=0 （これは1MBytes以上へロードする時の暗号）
0937            mov     bx,#gdt                 ! BX=GDTのアドレス
0938            call    lfile                   ! 1MBytes以上にカーネルをロードする
0939            jmp     launch                  ! launch(0995)へ(カーネルセットアップへ移行する)
0940  loadlow:call    loadfile                ! loadfile(0943)へ  1MBytes以下にカーネルをロードする
0941            jmp     launch                  ! launch(0995)へ
0942
0943  loadfile:mov    ax,#SYSSEG              ! AX=0x1000
0944            mov     es,ax                   ! ES=AX=0x1000
0945            xor     bx,bx                   ! BX=0
0946  lfile:    call    load                    ! カーネルを0x10000から読み込む
0947            jmp     lfile                   ! マップテーブルのセクタアアドレスがNULLになるまで繰り返す
0948
```

1セクタの読み込み(カーネルオリジナルブートセクタ、フォールバックセクタ、オプショナルセクタの読み込みに使う)

```
0951  load1:   call    loadit                  ! 1セクタ読み込む
                                                ! 正常に読み込みが実行されたときは「call load1」(0769、0792、0829)の後に戻り、
                                                ! 次行のコードは実行しない。
                                                !「call load」でセクタアドレスがEOF(Null)のときここに来る
0952            mov     bx,#msg_eof             ! BX=メッセージ‘ Unexpected EOF ’のアドレス
0953            call    say                     ! 表示する
0954            jmpi    restrt,SECONDSEG        ! restrt(0142)から再スタートする

0955  loadit:  call    load                    ! 1セクタ読み込む
0956            pop     ax                      ! load1(951)の「call loadit」の後に戻らないようにリターンアドレスを捨てる
0957            ret                             ! 「call load1」の後に戻る
0958
0959
0960
0961  loadopt:   call   loadit                  ! 1セクタ読み込み、正常終了した場合は次行の「jmp launch」は実行せず、
                                                ! 「call loadopt」の次に戻る
0962            jmp     launch                  ! 0995のカーネルセットアップへ移行する
0963
```

1セクタ読み込む。マップテーブルのセクタアドレスがまだあるときはリターンアドレスが変わる(コールしたもう1つ外側のルーチンへ)

```
0966  load:    push    es                      ! ESを退避
0967            push    bx                      ! BXを退避
0968  lfetch:  mov     si,moff                 ! SI=moff=マップテーブルポインタ
0969            mov     cx,MAP(si)              ! CX=セクタアドレスのCX値
0970            mov     dx,MAP+2(si)            ! DX=セクタアドレスのDX値
0971            mov     al,MAP+4(si)            ! AL=読み込むセクタ数
0972            or      cx,cx                   ! CX=0でZF=1にセット
0973            jnz     noteof                  ! No(0ではない)でnoteof(0980)へ
0974            or      dx,dx                   ! DX=0でZF=1にセット
0975            jnz     noteof                  ! No(0でない)でnoteof(0980)へ
```

EOFの場合

```
0976            pop     bx                      ! BXを戻す
0977            pop     es                      ! ESを戻す
0978            pop     ax                      ! リターンアドレスを捨てる。
0979            ret                             ! 「call load」のもう1つ外側のルーチンに戻る
```

EOFでない場合

```
0980  noteof:  add     si,#5                   ! SI=SI+5  ポインタを次のアドレスへ
0981            mov     moff,si                 ! moff=SI  ポインタをmoffへストア
0982            cmp     si,#508                 ! マップテーブル1ページ=1セクタ=512バイト、
                                                ! 最後のセクタアドレスはオフセット503-507になる
0983            jb      doload                  ! SI<508なら同じマップテーブルのページなのでdoload(1013)へ
0984            mov     moff,#0                 ! ページ変更なので、ポインタmoff=0
0985            push    cs                      ! ESを変更するため
0986            pop     es                      ! ES=C=0x9B00
0987            mov     bx,#MAP                 ! BX=マップテーブルバッファアドレス
```

リスト3 （続き）

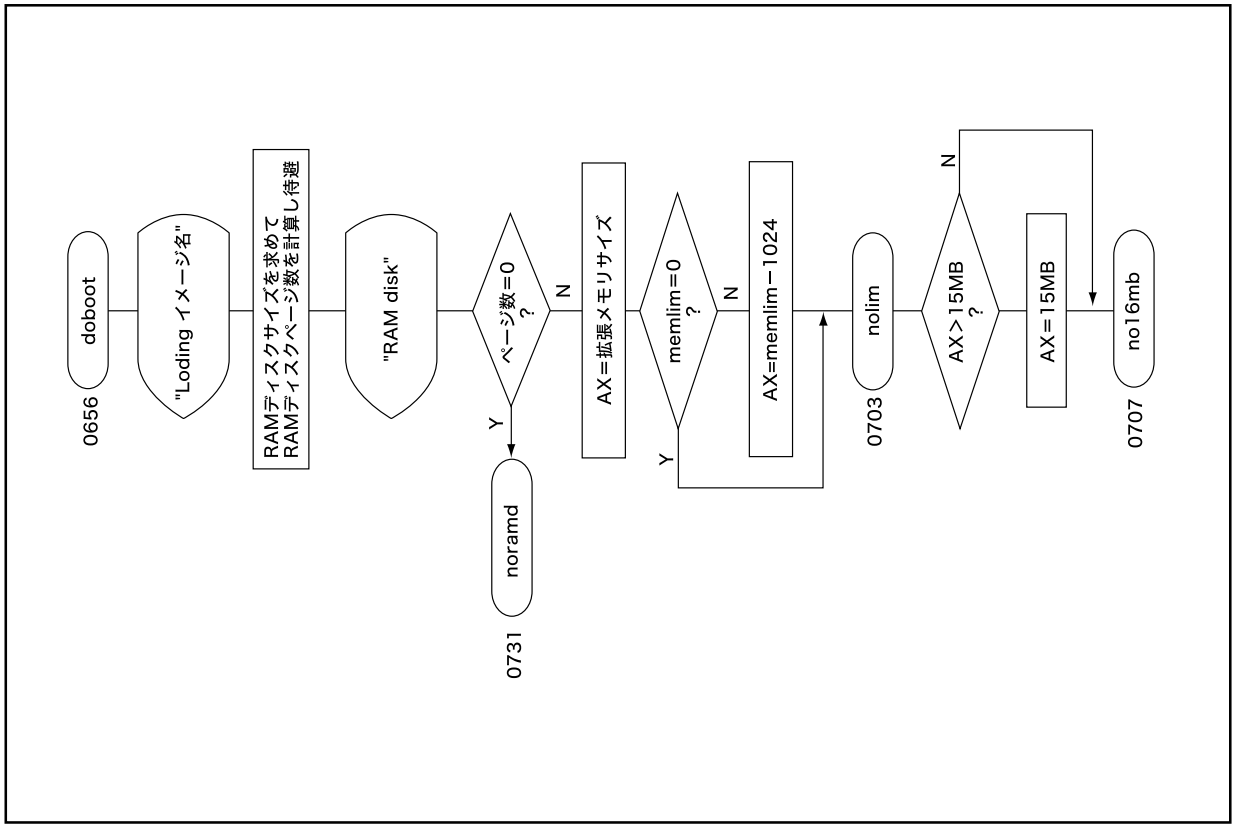
```
0988            call    sread                   ! 次のマップテーブルを1ページ読み込む
0989            mov     al,#0x2e                ! AL=0x2e=ドット
0990            call    display                 ! ドットを1文字表示
0991            jmp     lfetch                  ! lfetch(0968)へ戻る
0992
```

カーネルをスタートさせる

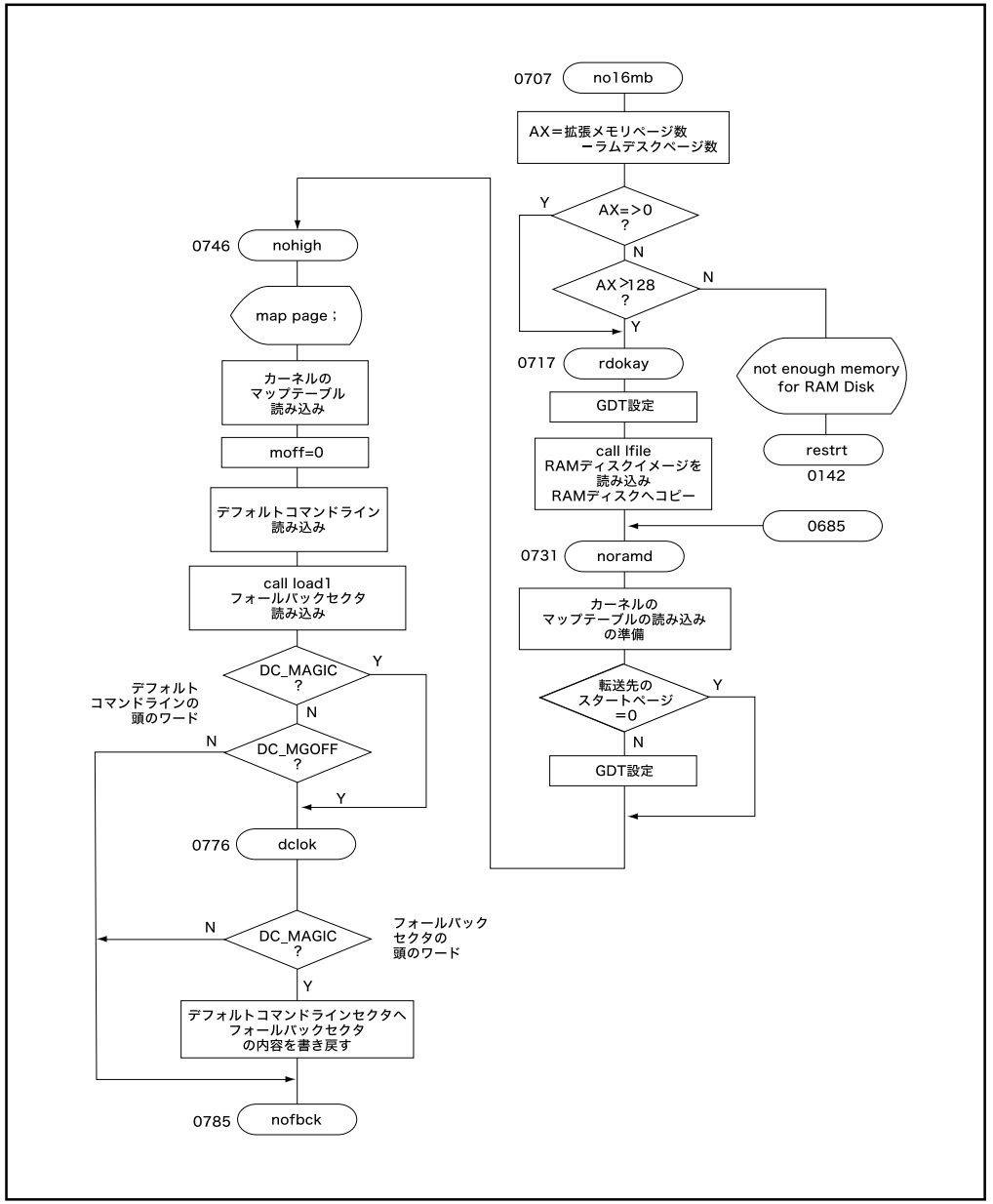
```
0994
0995    launch: push    es                      ! ESを退避
0996            push    bx                      ! BXを退避
0997            mov     bx,#crlf                ! BX=“crlf”のアドレス
0998            call    say                     ! 改行する
0999            mov     dx,#0x3f2               ! フロッピーディスクドライブのモータ停止
1000            xor     al,al                   ! AL=0
1001            outb                            ! I/O番地0x32Fに0を出力
1002            xor     ax,ax                   ! AX=0
1003            mov     dl,al                   ! DL=0
1004            int     0x13                    ! FDCのリセット
1005            call    remto                   ! free timer interrupt
1006            mov     ax,#INITSEG             ! AX=0x0x9000   レジスタの再設定
1007            mov     ds,ax                   ! DS=0x9000
1008            mov     es,ax                   ! ES=0x9000
1009            jmpi    0,SETUPSEG              ! CS=0x9020,IP=0  カーネルセットアップへ移行する
```

（これ以降のサブルーチンは省略します）

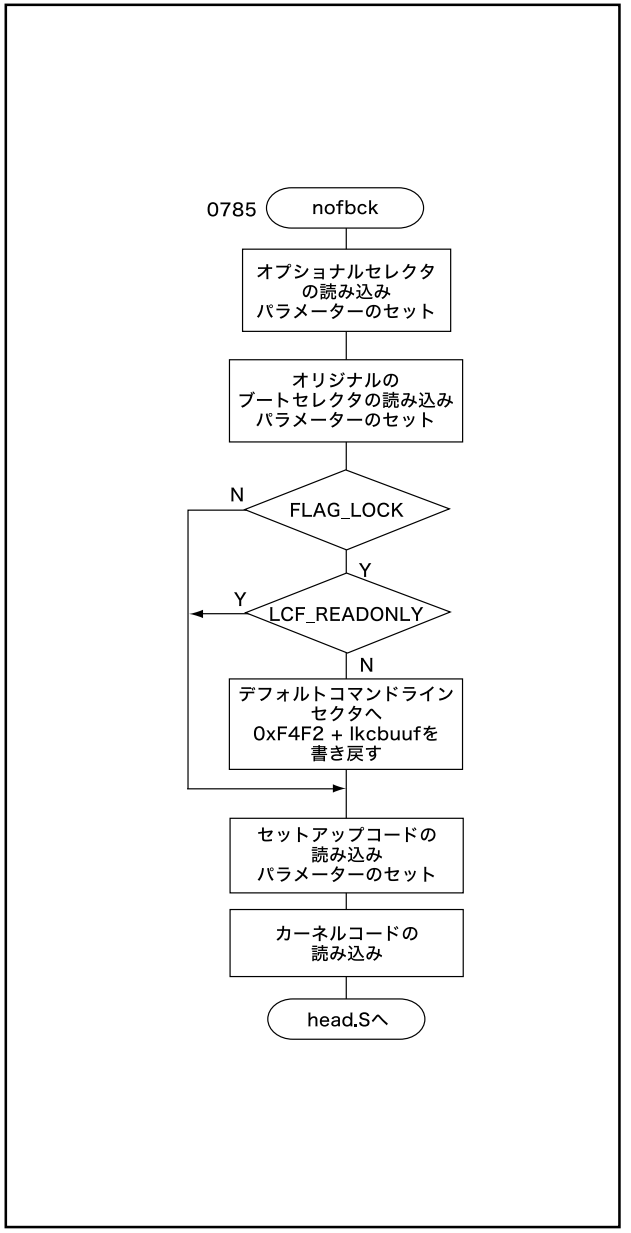
4回にわたり、LILOのブートアップ部分のソースコードを解析してきました。次回はいよいよカーネルのセットアップ部分である「setup.S」について解析します。ご期待ください。



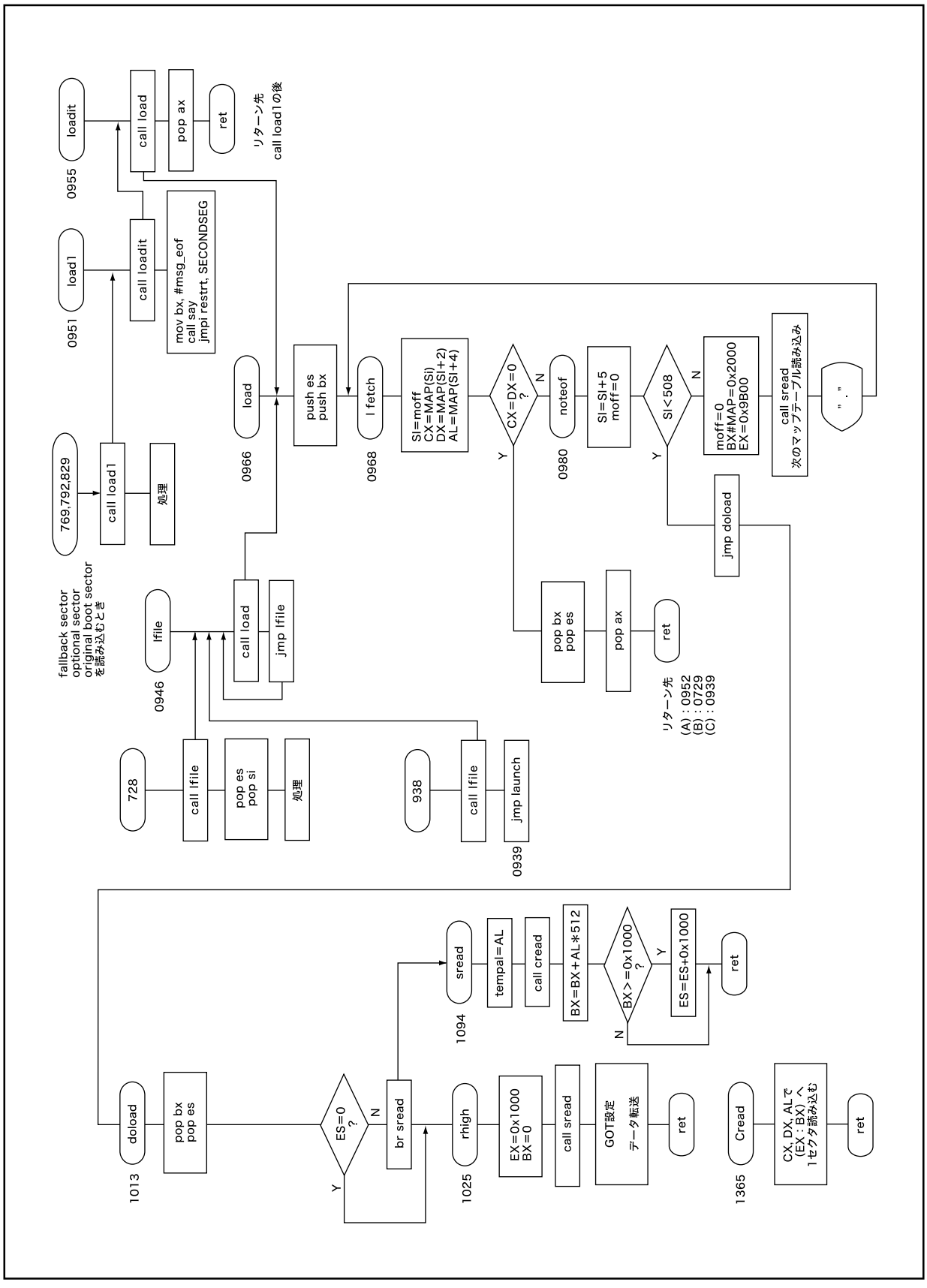
フローチャート1



フローチャート2



フローチャート3


doload
pop bx
pop es
ES=0 ?
Y
N
br sread
rhigh
EX=0x1000
BX=0
call sread
GOT設定
データ転送
ret
1013
1025
sread
templ=AL
call cread
BX=BX+AL*512
BX>=0x1000 ?
Y
N
ES=ES+0x1000
ret
1094
Cread
CX, DX, ALで
(EX : BX) へ
1セクタ読み込む
ret
1365
728
call lfile
pop es
pop si
処理
938
call lfile
jmp launch
0939
lfile
0946
call load
jmp lfile
fallback sector
optional sector
original boot sector
を読み込むとき
769,792,829
call load1
処理
load1
0951
call loadit
mov bx, #msg_eof
call say
jmpi restrt, SECONDSEG
loadit
0955
call load
pop ax
ret
リターン先
call load1の後
load
0966
push es
push bx
l fetch
0968
SI=moff
CX=MAP(SI)
DX=MAP(SI+2)
AL=MAP(SI+4)
CX=DX=0 ?
Y
N
noteof
0980
SI=SI+5
moff=0
SI<508
Y
N
moff=0
BX#MAP=0x2000
EX=0x9B00
call sread
次のマップテーブル読み込み
" . "
jmp doload
pop bx
pop es
pop ax
ret
リターン先
(A) : 0952
(B) : 0729
(C) : 0939

フローチャート4