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②

LILOのソースコード解析の第2回目です。セカンダリブートローダがカーネルイメージを読み込むところから見ていきましょう。

セカンダリブートローダは、最初にプライマリブートローダの先頭部分のパラメータを使って、ディスクリプタテーブル、キーボード変換テーブル、そしてデフォルトコマンドラインを読み込み、この後、ディスクリプタテーブルの情報に従って、マップテーブルを読み込みます。カーネルやRAMディスク用のイメージファイルは、このマップテーブルが持っているセクタアドレス情報をもとにメモリに読み込まれます。

前号では、このシーケンスの概要とセカンダリブートローダが読み込まれて制御を移すところまでを説明しました。今回は、これ以降からカーネルイメージを読み込んでセットアッププログラムに制御を移すところまでを解説します。LILOがこのような複雑な過程を経てカーネルを読み込む理由は、LILOが持っている機能が複雑だからです。LILOのブートアップのシーケンスを理解することは、他のブートローダの仕組みを理解するためにも非常に役に立ちます。

今回は最初に、セカンダリブートローダのシーケンスを理解するために必要な周辺の説明から始めます。

セカンダリブートローダの解析

second.Sのソースはかなり長いので、主な部分を詳しく説明していきます。シーケンスも込み入ってきますので、フローチャートも使ってプログラムの流れを説明します。

LILO Ver.20 のリニアアドレスの変換ルーチンは、正常に動作しない場合があります。従って、ソースコードはLILO Ver.21のsecond.Sを使い、このコードの各行に頭から順に行番号を振り、この番号を使って各行の位置をはっきりさせてから説明します。ソースコードの行番号はLinuxのprコマンドを使って

```
$ pr -n second.S > second.S.n
```

のようにするとつけられます。オリジナルファイルの空行を削除したりすると、行番号はずれてしまいますので注意してください。もしずれている場合は、ずれを補正してから読んでください。

フローチャートのエントリ(入り口端子)に付属の4桁の番号は、ソースリストの行番号と同一のものです。このフローチャートはプログラムの大まかな流れをつかむために、主な部分のみを取りだしたものですので、正確で詳細なものではありません。

なぜこんなに複雑な動きをするのでしょう

標準のBIOSを使わないファームウェアならば自由度はありますが、普通のPC互換機では、必ず起動時にBIOSを使います。

DOS時代からの歴史的な理由だと思いますが、PC互換機では、BIOSを使ったブートアップは、最初のブートローダのファイルサイズを512バイトに制限します。BIOSは1個のブートセクタだけを読み込んでそれに制御を移すからです。実際はこのセクタには64Bytesのパーティションテーブルと、2Bytesのマジックナンバーが含まれますので、プログラムコードとして使えるのは446Bytesに制限されます。これだけではいくらアセンブラでもマルチブートローダは作れません。そこで主なブートローダの機能はsecond.Sにまかせて、first.Sはこのsecond.Sを読み込んでこれに制御を移すことだけに専念します。BIOSのファイルアクセク機能を使わないブートローダもありますが(NUNIがそうです)、ブートセクタ

の部分を512Bytesに収められていません。

LILOは、Linuxだけでなく他のOS(他のLinux、DOS、Windows 95、Windows 98、WindowsNTなど)を最大で19個まで選択してブートアップできるマルチブートローダです。従って、それらのOSにパラメータを渡したり、起動イメージを読み込んだりするために、さらに構造が複雑になります。また、起動するOSのファイルシステムは各々異なりますし、BIOSはこれらのファイルシステムをサポートしておりません。ディレクトリやファイル名をローダは理解できませんし、BIOSも理解できません。そこでLILOのローダは、BIOSにセクタアドレスを与えてファイルのアクセクをします。また長いファイルですとセクタアドレスを記述するだけでも数KBytesにもなります。これがマップテーブルになるわけです。このマップテーブルも一度で読み込めないために、リンクして読みこみます。これがブートローダのファイルアクセクがさらに複雑になる理由です。

BIOSの説明

セカンダリブートローダで新しく使われるBIOSコールは次の5つです。その他のBIOSコールは第1回目(前号)で説明しましたので、省きます。

①通信ポートの初期設定

```
int 0x14
```

設定値：

AH=0

AL=初期設定する値

DX=論理ASYNCポート(0,1,2,3)を指定

結果：

AL=モデム状況

AH=回線制御状況

表1　GDTの構造(ES:SIの値はこのGDTのポインタ)

オフセット	GDTの内容	説明
0x00(ES:SI)	00	16Bytes共に00
0x10	0xFFFF	コピー元のセグメント長(64KBytes)
0x12	0000	コピー元のアドレスの下位2Bytes
0x14	01	コピー元のアドレスの上位Bytes＝0x10000
0x15	0x93	コピー元のセグメントアクセス権
0x16	0000	1ワード00
0x18	0xFFFF	コピー先のセグメント長(64KBytes)
0x1A	(3Bytes)	コピー先のアドレス
0x1D	0x93	コピー先のセグメントアクセス権
0x1E	00	18Bytes共に00

②ブロックの移動

1MBytes以上の記憶域と、それ以下の記憶域の間でデータブロックを移動します。BIOSはGDT(Global Descripter Table)を用いてデータを移動させます。

```
int 0x15
```

設定値：

AH=0x87

CX=転送するブロックのワードカウント最大数=0x8000(64KBytes)

ES:SI=GDTへのポインタ(表1)

結果：

AH=0

(正常終了。そのときCF=0、ZF=0になる)

③アドレス0x100000以上のシステムメモリ量(1KBytes単位)。

```
int 0x15
```

設定値：

AH=x88

結果：

AX=アドレス0x100000以上の使用可能なメモリの、連続する1KBytes単位のブロック数

④文字の入力状況

キーボードより入力されたデータがバッファ内にあるかどうかを調べる。

```
int 0x16
```

設定値：

AH=1

結果：

ZF=0(読み取れるコードがバッファ内にある。このときは、AX=操作コードと文字コードになる。)

ZF=1(読み取れるコードはバッファ内にない。)

⑤キーのシフト状況

現在のキーボードのシフト状況を読み取ります。

```
int 0x16
```

設定値：

AH=02

結果：

AL=現在のシフト状況

bit7=1　挿入モード
bit6=1　CapsLockモード
bit5=1　NumLockモード
bit4=1　ScrollLockモード
bit3=1　Altキー押し下げ
bit2=1　Ctrlキー押し下げ
bit1=1　左シフトキー押し下げ
bit0=1　右シフトキー押し下げ

リスト1　ブートセクタの最初の部分（LILO Ver.21で作成）

```
000000 fa eb 6c 00 00 00 4c 49 4c 4f 01 00 14 00 10 01
000010 00 00 00 00 4b 93 80 82 a6 ea 80 fd 01 a7 ea 80
000020 fd 01 a5 ea 80 fd 01 01 00 00 00 00 00 00 00 a9
000030 ea 80 fd 01 a8 e7 80 9c 01 a9 e7 80 9c 01 aa e7
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 b8
```

リスト2　ブートセクタの最初の部分（lilo.confでlinearを設定した場合）

```
000010 00 00 00 00 4b 93 80 82 62 1a c0 b7 01 63 1a c0
000020 b7 01 61 1a c0 b7 01 01 00 00 00 00 00 00 00 65
000030 1a c0 b7 01 b2 48 c0 b6 01 b3 48 c0 b6 01 b4 48
```

ブートセクタのパラメータ

前回説明したように、ブートセクタの最初の部分にはブートローダへ渡すパラメータが書かれています。ここでもう一度セカンダリローダに関係する部分について説明します。

リスト1、リスト2は、LILO Ver.21で作成したブートセクタの最初の部分です。前回の内容とは異なりますので注意してください。セクタアドレスはHCSの場合（リスト1）とリニア（リスト2）の場合を示します。

この2つでセクタアドレスの値を比べると表2のようになります。ただし、同じオフセットのアドレスが2つとも同一のアドレスを示しているとは限りません。liloを実行してマップファイルを作成し、そのアドレスをこのブートセクタに書き込みますので、そのたびに異なる場合があります。lilo.confを変更するとマップファイルも変更されます。同じマップファイルを使う場合はセクタアドレスは等しくなります。

この表のリニアアドレスのデータで、“c0”はHDD番号“0”を意味します。また、bit7が“1”ならハードディスク、bit6が“1”ならリニアアドレスを意味します。データ01は、セクタ数が“1”を意味します.

表3に、HCSアドレスの場合を分かりやすく、必要な部分のみ書き直します。

表2　HCSの場合とリニアの場合の、ブートセクタの比較

オフセット	HCSアドレス	リニヤアドレス	説明
0x000022	a5 ea 80 fd 01	61 1a c0 b7 01	0xb71a61（c0：drive番号＝0x00）
0x000018	a6 ea 80 fd 01	62 1a c0 b7 01	0xb71a62（bit6＝1ならリニアアドレス）
0x00001d	a7 ea 80 fd 01	63 1a c0 b7 01	0xb71a63（01：セクタ数）

表3　HCSアドレスの場合のブートセクタ

ソースのラベル	オフセット Hex	10進	データ	説明
最初の部分は省略				
	0B	11	14	VERSION=20（LILOのバージョンではない）
	0E	15	10 01	タイムアウト
	10	16	00 00	ディレイ
	12	18	00	シリアルポート
	13	19	00	シリアルポートパラメータ
	14	20	4b 93 80 82	タイムスタンプ
DSC_OFF	18	24	a6 ea 80 fd 01	ディスクリプタテーブルの第1セクタアドレス
DSC_OFF2	1D	29	a7 ea 80 fd 01	ディスクリプタテーブルの第2セクタアドレス
DFCMD_OFF	22	34	a5 ea 80 fd 1	デフォルトコマンドラインのセクタアドレス
	27	39	01	プロンプトオプション=1
MSG_OFF	28	40	00 00	グリーティングメッセージの長さ
	2A	42	00 00 00 00 00	イニシャルグリーティングメッセージファイルのマップセクタアドレス
	2F	47	a9 ea 80 fd 01	キートランスレーションテーブルセクタアドレス
途中省略（セカンダリローダのセクタアドレスはboot.bに含まれる）				
EXT_OFF	68	104	00 00	ext_si
	6A	106	00 00	ext_es
	6C	108	00 00	ext_bx
	6E	110	00	ext_dl

ブートセクタに書き込まれるパラメータの内容はcommon.hの「BOOT_PARAM_1」で示す構造体のデータです。前回のテーブルと大きく異なる点は、オフセットアドレスが「+4」されていることです。これはバージョンの違いによるものです。

ブートセクタは「9A00:000」にも読み込まれているので、上のオフセットを使って、このパラメータ値を読み出します。このパラメータは、boot.bとマップファイル内の必要な情報に対するポインタを持っています。この内、外部パラメータは、他のブートプログラム(System Commanderなど)がLILOのブートアップルーチンを呼び出して起動するときに使用します。通常のブートシーケンスでは使用しません。

ブート時に必要なファイルの関係

LILOでLinuxをブートアップする時に必要なファイルは、

①プライマリブートローダ

これはブートセクタ、あるいはパーテションテーブルのブートセクタにあります。

②カーネルのイメージ(通常はvmlinuzまたはvzImage)

このカーネルイメージの最初の512Bytesは、カーネルのオリジナルのブートローダです。また続いてカーネルのセットアップコードがあります。最後にカーネル本体のコードが続いています。

③マップファイル：map

デフォルトコマンドライン、ディスクリプタテーブル、RAMディスクイメージとカーネルイメージ、そしてイニシャルグリーティングメッセージのセクタ情報、ディスクリプタテーブル、キー変換テーブル等の情報が入っています。

④LILOブートローダ：boot.b

このファイルはfirst.Sとsecond.Sのバイナリコードです。

⑤RAMディスクイメージファイル(通常はinitrd)

これは必ずしも必要ではありません。SCSIドライブではこれが必要になるものがあります。例えば、SCSIドライバモジュールをRAMディスクイメージに読み込んで、SCSI上の実際のルートファイルシステムを読み込む場合です。別の言い方をすると、このRAMディスクはルートファイルシステムであり、実際のルートファイルシステムのロードの前に実行するプログラムが入っています。通常、このプログラムはシステム環境を調べたり、ユーザーに起動オプションを問い合わせたりします。このinitrdの終了後、カーネルは実際のルートイメージをロードして起動を続行します。

/bootディレクトリには(LILOで/bootを指定した場合)、boot.b、map、vmlinuzそしてinitrdがあります。ブートセクタのプライマリブートローダはboot.bの中のプライマリローダにパラメータを追加して、LILOが作成したものです(liloを実行したとき)。このパラメータを使ってセカンダリローダをboot.bからメモリに読み込みます。このパラメータが持っている他の情報を使って、セカンダリローダがブートアップに必要なマップテーブルをメモリに読み込み、このマップテーブルを使ってイメージを読み込みます。

デフォルトコマンドライン

マップファイルの最初のセクタは、デフォルトコマンドラインになります。ブート時にユーザーがシフトモードキーを押さなかった場合、あるいは外部コマンドラインがない場合には、このマップファイルのデフォルトコマンドラインはキーボードを押したのと同様に扱われます。このコマンドラインはマジックナンバー(最初の2Bytes)がオフ(DC_MGOFF)のとき、あるいは3Bytes目がNULLのときは無効になります。

表4　ディスクパラメータテーブルの構造

オフセット	大きさ	内容
0x00	byte	b7-b4：ステップレート、b3-b0：ヘッドアンロード時間
0x01	byte	b7-b1：ヘッドロード時間、b0：DMAを使わない
0x02	byte	モーター回転停止までの待ち時間(Tick単位)
0x03	byte	セクタ長(00：128Bytes、01：256Bytes、02：512Bytes、03：1024Bytes)
0x04	byte	トラック当たりのセクタ長
0x05	byte	セクタ間のギャップ長
0x06	byte	データ長
0x07	byte	フォーマット時のギャップ長
0x08	byte	フォーマットデータ
0x09	byte	ヘッドセトリング時間(ミリ秒単位)
0x0A	byte	モーター回転開始時間(1/8秒単位)

ディスクパラメータ

「0x000:0x0078」には、フロッピーディスクのディスクパラメータテーブルのエントリが書き込まれています(BIOSによって)。このディスクパラメータテーブルの構造は表4のようになっています。このパラメータについて詳しく書かれた文献はあまりありませんので、少し詳しく説明します。

ディスクリプタテーブル

ブートセクタの情報(セクタアドレス)より、マップファイルから読み込まれるディスクリプタテーブルは、リスト3のような構造をしています。このデータはディスクリプタテーブルのセクタアドレス情報を元に読み込んだものです。ここにはDOSとKondara MUN/Linuxのイメージのテーブルのみを示しています。このテーブルはlilo.confに記入した起動イメージの数だけ作られます。テーブルの構造はcommon.hの「DESCR_SECTORS」で定義され、さらにこの内部に「IMAGE_DESCR」で定義される構造で、起動すべきイメージのデータが作られます。このテーブルは「0x9B000:0x2200(0x9D200)」に読み込まれます。各テーブルの間隔は52Bytesです。最初の2Bytesはチェックサムで、後は52(0x34)Bytesごとに各イメージのテーブルになります。

上のデータを必要な部分のみ、分かり易く書き換えたのが表5です。

lilo.confでシングルキーを設定すると表5のサンプルのように、1つのイメージに対して2つのテーブルが作成されます。

リスト3　ブートセクタ(DOS+Kondara MNU/Linuxの場合)

```
000002 64 00 00 00 00 00 05 08 d8 11 00 00 00 10    d
000010 00 00 00 00 00 00 70 a2 9f 38 2e f9 07 40 b8 6b
000020 11 40 00 00 00 00 44 fd ff bf 00 af ea 80 fd 01
000030 00 00 20 00 05 08

000036 5f 64 6f 73 36 2e 32 00 d8 11                    _dos6.2
----これ以下省略--------------------

00006a 6b 00 00 00 00 00                                K
000070 05 08 d8 11 00 00 00 10 00 00 00 00 00 00 70 a2
000080 9f 38 2e f9 07 40 b8 6b 11 40 de a4 04 00 83 ea
000090 80 fe 01 b2 ea 80 fd 01 00 01 38 00 05 08

00009e 5f 4b 6f 6e 64 61 72 61 00                       _kondara
-----これ以下省略--------------
```

表5　ディスクリプタテーブルの構造

オフセット	データ	意味
0x02(2+0x00)	64 00	"d": DOSのシングルキー
0x12(2+0x10)	00	パスワード
0x2b(2+0x29)	af ea 80 fd 01	ブートイメージマップテーブルのセクタアドレス
0x32(2+0x30)	20 00	フラグ(シングルキーフラグ:bit5=1)
0x34(2+0x32)	05 08	vga_mode
0x36 (2+0x34)	5f 64 6f 73 36 2e 32 00	"_dos6.2"イメージのラベル
これ以下は上のテーブルと同じ		
0x6a(0x36+0x34)	6b	"K": Kondara MNU/Linuxのシングルキー
0x7a(0x6a+0x10)	00	パスワード
0x8a(0x6a+0x20)	de a4 04 00	RAMディスクイメージサイズ=0x04a4de
0x8d(0x6a+0x24)	83 ea 80 fe 01	RAMディスクイメージマップテーブルのセクタアドレス
0x93(0x6a+0x29)	b2 ea 80 fd 01	カーネルイメージマップテーブルのセクタアドレス
0x98(0x6a+0x2a)	00 01	カーネルを転送する最初のページ
0x9a(0x6a+0x30)	38 00	フラグ(シングルキーフラグ:bit5=1)
0x9c(0x6a+0x32)	05 08	vga_mode
0x9e(0x6a+0x34)	5f 4b 6f 6e 64 61 72 61 00	"_kondara"イメージのラベル
これ以下は上のテーブルと同じ		

リスト4　マップファイルの例

```
000000      6d 6b 00 00 00 00 00 00 00 00 00 00 00 00 00 00      デフォルトコマンドライン
000010      00 00 00 00 00 00 00 00 00 00 00 00 00 00 0000
       ------以下省略-------------
000200      9b ab 64 00 00 00 00 00 05 08 d8 11 00 00 00 10      ディスクリプタテーブル
       ------以下省略-------------
000800      00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f      キーボード変換テーブル
000810      10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
       --------以下省略-------------
001a00      b0 ea 80 fd 01 b1 ea 80 fd 01 ae e6 80 92 01 af      マップテーブル(カーネルイメージ)
001a10      e6 80 92 01 b0 e6 80 92 01 b1 e6 80 92 01 b2 e6      1ページ目
       --------以下省略-------------
001be0      8e e6 80 94 01 8f e6 80 94 01 98 e6 80 94 01 99      ページの最後のセクタアドレスは次ページのファイル
                                                                のアドレス
001bf0      e6 80 94 01 9a e6 80 94 01 b3 ea 80 fd 01 00 00      "b3 ea 80 fd 01"=次のページセクタアドレス
       --------以下省略-------------
003950      e6 80 ab 01 bc e6 80 ab 01 bd e6 80 ab 01 00 00      マップテーブルの最後
003960      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00      EOF(CX=0、DX=0になる)
       --------以下省略-------------
003a00      b8 ea 80 b8 01 b9 ea 80 b8 01 ba ea 80 b8 01 bb      マップテーブル(RAMディスク)
003a10      ea 80 b8 01 bc ea 80 b8 01 bd ea 80 b8 01 be ea
       ---------以下省略-------------
```

マップファイル

LILOは、lilo.confの情報によりマップファイルを作成します。このマップファイルは、デフォルトコマンドライン、ディスクリプタテーブル、キーボード変換テーブル、そしてカーネルイメージとRAMディスクイメージのセクタアドレステーブルを持っています。ディスクリプタテーブルはこのマップファイル内のマップテーブルの先頭セクタアドレス情報を教えます。リスト4にマップファイルの例を示します。

マップテーブルは、5Bytesのセクタアドレスで書かれています。このテーブルが長いときは複数のページ(1ページは512Bytes)で構成されます。ブートローダはこのマップテーブルを1ページごとに読み込みます。ページの最後のセクタアドレスは、次ページの先頭を示すセクタアドレスになっています。ブートローダはこのセクタアドレスで次ページのテーブルを読み込み、次々とリンクして最後のEOF(NULL)まで読み込みます。

表6　カーネルに引き渡されるパラメータ

オフセット 16進	オフセット 10進	内容
0x20	32	マジック番号(0xA33F)
0x22	34	パラメータラインの「0x90000」に対するオフセット
	497	カーネルセットアップコードのセクタ数
	498-499	ルートファイルシステムがリードオンリーかリード／ライトかのフラグ
	500-501	パラグラフ(16Bytes)単位でのカーネルのサイズ
	504-505	RAMディスクサイズ(KBytes単位)
	506-507	テキストモードVGAのタイプ 0xFFFD：ブート時にユーザーはVGAタイプを要求される 0xFFFE：80x50(拡張)モード 0xFFFF：80x25(普通)モード
	508	ルートとしてマウントされるべきデバイスのマイナー番号
	509	ルートとしてマウントされるべきデバイスのメジャー番号

カーネルのオリジナルブートローダ

カーネルイメージ(vmlinuz)の最初のセクタは、カーネルオリジナルのブートローダです。このコードはLILOのセカンダリブートローダによって「0x90000」に読み込まれます。このコードはフロッピーディスクドライブで起動する場合のブートアップローダですが、LILOで起動するときは、このプログラムコードは使用されません。しかし、このセクタに書き込まれているパラメータは、カーネルのセットアップで必要とされますので、LILOのブートローダはこのパラメータをカーネルへ引き渡します。このパラメータの内容を表6で説明します。

メモリの配置

LILO によるブートアップ時に読み込まれるファイルの、メモリ上の配置は、表7のようになります。

CPUスタックの説明

ブートローダでは、データを一時退避させるためにスタックを使います。例えば次のコードを見てください。

```
0040 push   dx     ! STACK1=DX
0041 int    0x14
0042 xor    ax,ax
0043 mov    ds,ax
0044 pop    bx     ! BX=STACK1=(0040)でのDXの値
```

0040では、最初のDXの内容がスタックに退避されます。そして、0044で、このDXの値はBXレジスタにスタックから戻されます。このように、スタックを使用したデータの退避と読み込みは、頻繁に行なわれます。

次に、サブルーチンコールをした場合のスタックの操作について説明します。まず、リスト6のようなコードを仮定します。

このコードで、0021の後、プログラムはどこへ戻るでしょうか？　正解は「0002」です。なぜなら、0001でsub1をコールしたとき、スタックには0002のアドレスがプッシュされます。そして0010でsub2をコールすると、さらにスタックには0011のアドレスが追加されてプッシュされます。

ところが、0020にpop命令がありますので、この0011のアドレスはaxレジスタにポップされて、スタックからは捨てられます。そして、次にポップされるスタックの内容は0002のアドレスになります。0021でret命令を実行すると、リターンアドレスとしてスタックからポップされるのは0002のアドレスです。従って、0021の後は0002へ戻ります。0010からサブルーチンコールした後の、本来の戻り番地である0011には戻れません。この手法は何度かsecond.Sで使われています。

リトルエンディアンとビッグエンディアン

ディスクリプタテーブルのセクタアドレスをオフセットの昇順と降順に分けて、下に示してみます。このアドレスをDX、CX、ALにロードします。

昇順：

```
0x18―0x1c  a6 ea  80 fd  01
           CX     DX     AL
```

降順：

```
0x1c―0x18  01     fd 80  ea a6
           AL     DX     CX
```

表7　ブートアップ時に読み込まれるファイルのメモリ配置

アドレス	説明
0x7C00-0x7DFF	プライマリブートローダ
――省略――	
0x10000-0x8FFFF	カーネルイメージ
0x90000-0x901FF	カーネルオリジナルブートローダ
0x90200-0x90A00	カーネルセットアッププログラム(LILOがロード)
0x90A01-0x99FF	カーネルセットアッププログラム(残り)
0x9A000-9A1FF	プライマリブートローダ(0x7C00のものと同じ)
0x9A200-0x9AFFF	スタックエリア
0x9B000-0x9BFFF	セカンダリブートローダ
――省略――	
0x9D000-0x9D1FF	マップテーブルエリア
0x9D200-0x9D5FF	ディスクリプタテーブル
0x9D600-0x9D7FF	デフォルトコマンドラインほか
0x9D800-0x9D9FF	キーボード変換テーブル
0x9DA00-0x9BFF	パラメータラインエリア

リスト6　サブルーチンコールでのスタック操作

```
0001        call   sub1   ! スタック1＝0002のアドレス
0002        mov    ax,bx
-----略-----
0010 sub1:  call   sub2   ! スタック2＝0011のアドレス
0011        xor    ax,ax
-----略-----
0020 sub2:  pop    ax     ! ax＝スタック2のアドレス
0021        ret           ! リターンアドレスはスタック1の内容即ち0002のアドレスになる。
```

するとAL=01、DX=fd80、CX=eaa6となります。セクタアドレスを降順に書き直すと、このレジスタへのデータの読み込まれるバイトの順が良く理解できます。すなわち、x86系ではリトルエンディアンなので降順にバイトデータを読み込みます。モトローラの68x系ではビッグエンディアンなので、昇順のバイトデータをそのまま読み込みます。通常はメモリのデータをアドレスの昇順に書きますので、x86系ではレジスタの読み込みに戸惑いを感じることがあるのです。

ソースコードの解析（メインルーチン）

前回はアセンブラのコードも詳しく説明しましたが、今回は少し簡単にします。second.Sは合計で1930行あり、すべてのコードを解析すると誌面が足りません。サブルーチンについては主なものだけを解説し、残りのサブルーチンについては別の機会に説明したいと思います。皆さんも練習のつもりでトライしてみてください。アセンブラのコードは少し複雑になると読みにくくなりますので、フローチャートを見ながら読んでください。サブルーチンのコードの解析は今回は大部分省略しましたが、構造が複雑なものはフローチャートを添付して説明をつけましたので、参考にしてください。間違いのないように充分に気をつけたつもりですが、浅学のせいで誤りがあるかもしれません。そのときは遠慮なくメールしてください。

なお、first.Sから「jmp start」を経て、「start:」へ来たときのレジスタの値は次のようになります。

```
CS=SECOND_SEG=0x9B00
IP=CODE_START_2=0x000E
DS=0x9A00
SS=0x9000
SP=0xB000
ES=0x9A00
```

編集部から

以下のページ（p.37〜p.38、p.165〜p.167）で「second.S」のソースコードを詳細なコメントやフローチャート図解とともに解説しますが、今回は誌面の都合により、解説部分の前半1/3だけの掲載となってしまいました。これはあくまでも編集上の都合によるものです。申し訳ありませんがご了承ください。残る2/3の部分については、次号で掲載させていただきます。

R E S O U R C E

参考文献
LILO、i386アセンブラ言語、BIOS関連の参考文献を紹介しておきます。

★

●i386アセンブラ関連

[1] 80386プログラミング・ブック
B・E・Sumisu 著、荒木 訳、啓学出版

[2] 80386ハンドブック
Pen Brumm 著、古性 訳、丸善出版

●BIOS関連

[3] OADGテクニカル・リファレンス
PCオープンアーキテクチャー推進協議会

[4] The Programer's PC SOURCEBOOK
Thom Hogan著、SE編集部 訳、Microsoft Press

[5] DOS/Vプログラミングガイド
最上 著、アスキー出版

[6] PC DOS/V J6.1 BIOS技術解説書
日本IBM

[7] DOS J5.0 BIOSインターフェース技術解説書
日本IBM

●LILO関連

[8] LILO generic boot loader for Linux, Ver.20: Ueser's Guide
Wener Almesberger

[9] LILO generic boot loader for Linux, Ver.20: Technical overview
Wener almesberger

[10] LILO：the Linux Loaderの動作について
佐野武俊 著
http://www.linux.or.jp/JF/JFdocs/

[11] コメントから読むLinux カーネル
佐野武俊 著
http://www.linulx.or.jp/JF/JFdoc/readkernel.html

[12] Linux ブートローダーLILO
大崎博之 著、UNIX MAGAZINE、1998年12月号、アスキー

second.Sの動作解析

```
0033  start:   seg     ss                          ! 次のパラメータをSSからのオフセットで読む
0034           mov     dx,DSC_OFF-6+SSDIFF         ! DSC_OFF=20=0x18、SSDIFF=0xA000
                                                   ! DX=[0x9B00:0xA012](シリアルポート)
0035           sub     dl,#1                       ! DL-1を実行
0036           jc      nocom                       ! DL=0のときはキャリービットがセットされnocom(0091)へジャンプする(シリアルポートなし)

～途中省略(シリアルポートの設定と出力)～

0091   nocom:
0092   #ifndef LCF_NODRAIN
```

キーバッファを空にする

```
0093           mov     cx,#32                      ! ループの回数を32回に設定
0094   drkbd:  mov     ah,#1                       ! AH=1はBIOSコールのファンクション番号
0095           int     0x16                        ! キーバッファの内容があるかを調べる
0096           jz      comcom                      ! ない場合はcomcom(0120)へジャンプする
0097           xor     ah,ah                       ! ある場合はその内容を取り出す。AH=0はファンクション番号
0098           int     0x16                        ! バッファの文字が1つ減る
0099           loop    drkbd                       ! drkbdから繰り返す。CXは1減じる。
0100   #endif
0101                                               ! ディスクパラメータを読んでチェックする
0102   comcom: mov     al,#0x4c                    ! 0x4C="L"。LILOの2番目のLを表示する。
0103           call    display                     ! ALの文字を1文字表示するルーチン
0104           xor     ax,ax                       ! AX=0
0105           mov     ds,ax                       ! DS=AX=0
0106           lds     si,0x78                     ! SI=0x78、DS=0、SI=0x78
                                                   ! 「00:0x78」はディスクパラメータテーブルの先頭番地
0107   #ifndef LCF_XL_SECS
0108           cmp     byte ptr (si+4),#9          ! パラメータテーブルの4バイト目を比較する
                                                   ! 4バイト目はトラック当たりのセクタ数。これが9より大きいかを比較する。
0109           ja      dskok                       ! 大きいときはOKなのでdskok(0129)へジャンプする
0110   #endif
```

ディスクパラメータの書き換え

```
0111           mov     ax,cs                       ! AX=CS=0xB000
0112           mov     es,ax                       ! ES=0xB000
0113           mov     di,#dskprm                  ! DI=dskprmのオフセット
0114           mov     cx,#6                       ! 繰り返し数を6回にセット
0115           rep                                 ! 次の命令を6回実行する(12Bytesコピーする)
0116           movsw                               ! [DS：SI]で示すメモリから[ES:DI]で示すメモリへコピーする。CXは1減算、SIとDIは2加算
                                                   ! (ワードなので)され、CX=0まで続ける。
0117           seg     es                          ! セグメント=ES=0xB000
0118   #ifndef LCF_XL_SECS                         ! LCF_XL_SECSが定義されていない場合は
0119           mov     byte ptr (di-8),#18         ! トラック当たりのセクタ数を18に設定する
0120   #else                                       ! 定義されている場合は
0121           mov     byte ptr (di-8),#LCF_XL_SECS ! セクタ数に定義された値を代入する
0122   #endif
0123           cli                                 ! 割り込み禁止
0124           xor     ax,ax                       ! AX=0
0125           mov     ds,ax                       ! DS=0
0126           mov     0x78,#dskprm                ! [0x78]=dskprmのオフセット
0127           mov     0x7a,es                     ! [0x7A]=ES、ディスクパラメータテーブルのエントリ=「ES：dskparamのセグメント値」に変更。
0128           sti                                 ! 割り込み許可
```

タイマ割り込みを設定する

```
0129   dskok:  seg     cs                          ! CS=0x9B00
0130           mov     byte ptr break,#0           ! break=0にクリアする(ブレーク受付フラグ)
0131           call    instto                      ! タイマ割り込みのベクタを設定する
0132           jmp     restrt                      ! rstst(0142)へジャンプする
0133
0134                                               ! Die
0135                                               !ローダのエラーで無限ループするルーチン
0136           crshbrn:mov     al,#63              ! 63="?"
0137           call    display                     !"?"を表示する(1188)
0138   zzz:    jmp     zzz                         ! 無限にループする
0139
0140
0141                                               !これより本格的にスタートする
```

まずレジスターを再設定する

```
0142  restrt: mov     ax,cs                  ! AX=CS=0x9B00
0143          mov     ds,ax                  ! DS=CS=0x9B00
0144          mov     es,ax                  ! ES=CS=0x9B00
0145          cli                            ! SPを操作する間割り込みを禁止する
0146          mov     sp,#STACK              ! SP=STACK=0xB000
0147          sti                            ! 割り込みを許可する
0148          cmp     sig,#0x494c            ! ローダが正しく読みこまれているかをチェックする。
0149                                         ! sig:≒"LILO"なのでこの4Bytesを検査する。"LI"≒0x494c
0150          jne     crshbrn                ! 違うときはcrshbrn(0136)へジャンプする
0151          cmp     sig+2,#0x4f4c          ! 次の2Bytesを検査する"LO"≒0x4f4c
0152          jne     crshbrn                ! 違うときはcrshbrn(0136)へジャンプする
0153          cmp     stage,#STAGE_SECOND    ! stateフラグをSTAGE_SECOND=2と比較する
0154          jne     crshbrn                ! 違うときはcrshbrn(0136)の無限ループへジャンプする
0155          cmp     version,#VERSION       ! バージョンフラグをVERSIONと比較する
0156          jne     crshbrn                ! 違うときはcrshbrn(0139)へジャンプする
```

ディスクリプタテーブルを読み込む

```
0157          mov     cmdbeg,#acmdbeg        !cmdbeg=#acmdbeg（メッセージ"auto"のアドレス）
0158  ldsc:   mov     al,#1                  ! AL=1は読み込みセクタ数=1のこと
0159          mov     bx,#DESCR              ! 読み込むバッファアドレス=#DESCR=0x2200
0160          seg     ss                     ! 次の命令で読みこむデータのSEGMENTはSS=0x9000
0161          mov     cx,DSC_OFF+SSDIFF      ! ディスクリプタテーブル第1セクタのCX値
                                             ! SSDIFF=0xA000  DSC_OFF=0x18
0162          seg     ss
0163          mov     dx,DSC_OFF+2+SSDIFF    ! ディスクリプタテーブル第一セクタのDX値
0164          call    cread                  ! テーブルを1セクタ読みこむ(1365)
0165          jc      fdnok                  !エラーだとキャリーがセットされfdnok(0214)へジャンプする
0166          mov     al,#1                  ! 読みこみセクタ数=1
0167          mov     bx,#DESCR+512          ! 読み込むアドレスを512Bytes増やす
0168          seg     ss
0169          mov     cx,DSC_OFF2+SSDIFF     ! テーブルの2番目のセクタのCX値
                                             ! DSC_OFF2=0x1D
0170          seg     ss
0171          mov     dx,DSC_OFF2+2+SSDIFF   ! 2番目のDX値
0172          call    cread                  ! テーブルの2番目のセクタを読みこむ(1365)
0173          jc      fdnok                  ! エラーのときはfdnok(0214)へジャンプする
```

読み込んだテーブルのチェックサムを計算する

```
0174          mov     si,#DESCR              ! SI=テーブルの先頭アドレス
0175          mov     bx,#INIT_CKS           ! BX=チェクサムの正しい値
0176          cld                            ! 繰り返しでSI値は加算されるようにセット
0177          mov     cx,#0x200              ! CX=繰り返しの回数=0x200(２セクタ分)
0178  csloop: lodsw                          ! AX=DS:SIで示すメモリの内容(SI=S+2)
0179          xor     bx,ax                  ! AX=(INIT_CKS EXOR AX)を計算する
0180          loop    csloop                 ! csloopからCX=0になるまで繰り返す
                                             ! CXは繰り返しの最後で1減じられる
0181          or      bx,bx                  ! BXが0ならZFがセットされる
0182          jnz     chkerr                 ! ZF=0だとエラーなのでchkerr(0221)へジャンプする
```

キーボード変換テーブルを読み込む

```
0183          seg     ss                     ! SEG=SSで次のデータを読み込む
0184          mov     cx,MSG_OFF+SSDIFF+7    ! CX=キーボード変換テーブルのCX値
0185          seg     ss                     ! MSG_OFF=0x28
0186          mov     dx,MSG_OFF+SSDIFF+9    ! DX=キーボード変換テーブルのDX値
0187          mov     bx,#KEYTABLE           ! BXはバッファアの先頭アドレス=0x2800
0188          mov     al,#1                  ! 読み込むセクタ数=1
0189          call    cread                  ! キーボード変換テーブルを読み込む(1365)
0190          jc      fdnok                  ! エラーがある場合はfdnok(0214)へジャンプする
```

デフォルトコマンドラインを読む

```
0191          mov     al,#1                  ! 読み込みセクタ数=1
0192          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=0x2600
0193          seg     ss
0194          mov     cx,DFCMD_OFF+SSDIFF    ! CX=デフォルトコマンドラインのCX値
0195          seg     ss                     ! DFCMD_OFF=0x22
0196          mov     dx,DFCMD_OFF+2+SSDIFF  ! DX=デフォルトコマンドラインのDX値
0197          call    cread                  ! デフォルトコマンドラインを読み込む(1365)
0198          jc      fdnok                  ! エラーの時はfdnok(0214)へジャンプ
0199          mov     bx,#DFLCMD             ! 頭の2Bytesをチェックする
0200          cmp     word ptr (bx),#DC_MAGIC ! 頭の2BytesをDC_MAGICと比べる
                                             ! DC_MAGIC=0xf4f2
```

P. 165に続く →

→ P. 36の続き

```
0201            jne     bdcmag                  ! 違う場合はbdcmag(0212)へジャンプする
                                                ! エラーの無い場合は通常は(0211)へそして(0261)へジャンプする
0202    #ifndef LCF_READONLY                    ! デフォルトではLCF_READONLY=1

デフォルトコマンドラインを書き戻す
0203            mov     word ptr (bx),#DC_MGOFF ! 先頭ワード=DC_MGOFF=0x6b6d
                                                ! マジックナンバーを消す
0204            mov     al,#1                   ! AL=1(書き込むセクタ数)
0205            seg     ss                      ! SEGMENT=SS
0206            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
0207            seg     ss
0208            mov     dx,DFCMD_OFF+2+SSDIFF   ! DX=セクタアドレスのDX値
0209            call    cwrite                  ! デフォルトコマンドラインを1セクタ書き戻す
0210    endif
0211            jmp     dokay                   ! dokay(0261)へ

デフォルトコマンドラインがDC_MAGICでないとき、200からここへ来る
0212    bdcmag: mov     byte ptr (bx+2),#0      ! デフォルトコマンドライン を無効にする
                                                ! デフォルトコマンドラインテーブルの3バイト目=0
0213            jmp     dokay                   ! dokay(0261)へジャンプする

キーおよびディスクリプタテーブル読み込み時のエラー処理ルーチン
0214    fdnok:  xor     ax,ax                   ! reset FDCここは「mv ax,0x80」がよい
0215            mov     dl,al                   ! drive no=0x00これはFDCになる
0216            int     0x13                    ! ディスクシステムのリセット
0217            br      ldsc                    ! ldsc(0158)にジャンプしてリトライする
0218
0219                                            ! チェックサムエラー
0220                                            ! チェクサムエラーの時の処理ルーチン
0221    chkerr: mov     al,#45                  ! 45=" - "(マイナス)
0222            call    display                 ! " - "を表示する
0223    sixfeet: jmp    sixfeet                 ! 無限ループになる
0224
0225                                            ! ブート可能なOSを表示するルーチン
0226                                            ! 1行あたり4イメージ名を1イメージは15文字で表示
0227    list:   mov     byte ptr (bx),#0        ! BX=cmdlineのポインタ(EOL をセットする)
0228            mov     bx,#crlf                ! BX=改行メッセージのアドレス
0229            call    say                     ! 改行する
0230            mov     si,#DESCR+2             ! SI=ディスクリプタテーブル内イメージ名の最初のアドレス
0231            mov     cx,#IMAGES              ! CX=#IMAGES=19(起動イメージの最大数)
0232            xor     dl,dl                   ! DL(表示イメージのカウンタ)=0
0233    lloop:  testb   (si),#0xff              ! ディスクリプタテーブルの文字=0xFF
0234            jz      ldone                   ! " Yes "でldone(0253)へ
0235            mov     bx,si                   ! BX(表示ストリングのポインタ)=SI
                                                ! SIはディスクリプタテーブルの起動イメージ名を指し示す
0236            call    say                     ! イメージ名をNULLまで表示する
0237            add     si,#MAX_IMAGE_NAME      ! SI=SI+15(イメージ名領域の最後をポイントする)
0238            inc     dl                      ! DL=DL+1  表示した起動イメージをカウント
0239            test    dl,#3                   ! DL=3(1行4イメージまで表示する)
0240            jnz     fill                    ! カウンドが3以下なのでfill(0244)へ。表示したイメージ名の後をスペースで埋める
0241            mov     bx,#crlf                ! BX=メッセージ改行のアドレス
0242            call    say                     ! 改行する(カウント=3なので)
0243            jmp     imgdne                  ! imgdne(0251)へ。次のイメージ名を得る。

イメージ名が最大文字数以下の部分はスペースを表示する
BX(表示ストリングのポインタ)を退避
0245            mov     al,#0x20                ! AL=0x20(スペース)
0246            call    display                 ! ALの文字(スペース)を表示する
0247            pop     bx                      ! BX(ストリングポインタ)を復帰
0248            inc     bx                      ! BX=BX+1(次の文字)
0249            cmp     bx,si                   ! BX=SI
0250            jbe     fill                    ! 「BX < SI」なのでfill(0244)から繰り返す
0251    imgdne: add     si,#DESCR_SIZE-MAX_IMAGE_NAME ! SI=SI+(52-15)次のイメージ名をポイントする(DESCR_SEZE=52、MAX_IMAGE_NAM=15)
0252            loop    lloop                   ! CX=CX-1 CX>0のときはlloop(0233)へ。次のイメージ名を得る。
253     ldone:  test    dl,#3                   ! DL=3(改行直後かどうか?)。  DL=3,7,0xB,0xFでもZF=1になる
0254            jz      atbol                   ! Yesで改行せずiloopへ戻る
0255            mov     bx,#crlf                ! ZF=1の場合はBX=メッセージ改行のアドレス
0256            call    say                     ! 改行する
0257    atbol:  br      iloop                   ! iloop(0314)へ戻る
0258
0259
0260
```

キー入力を待つ

ここには「0211」あるいは「0213」からジャンプしてきます。

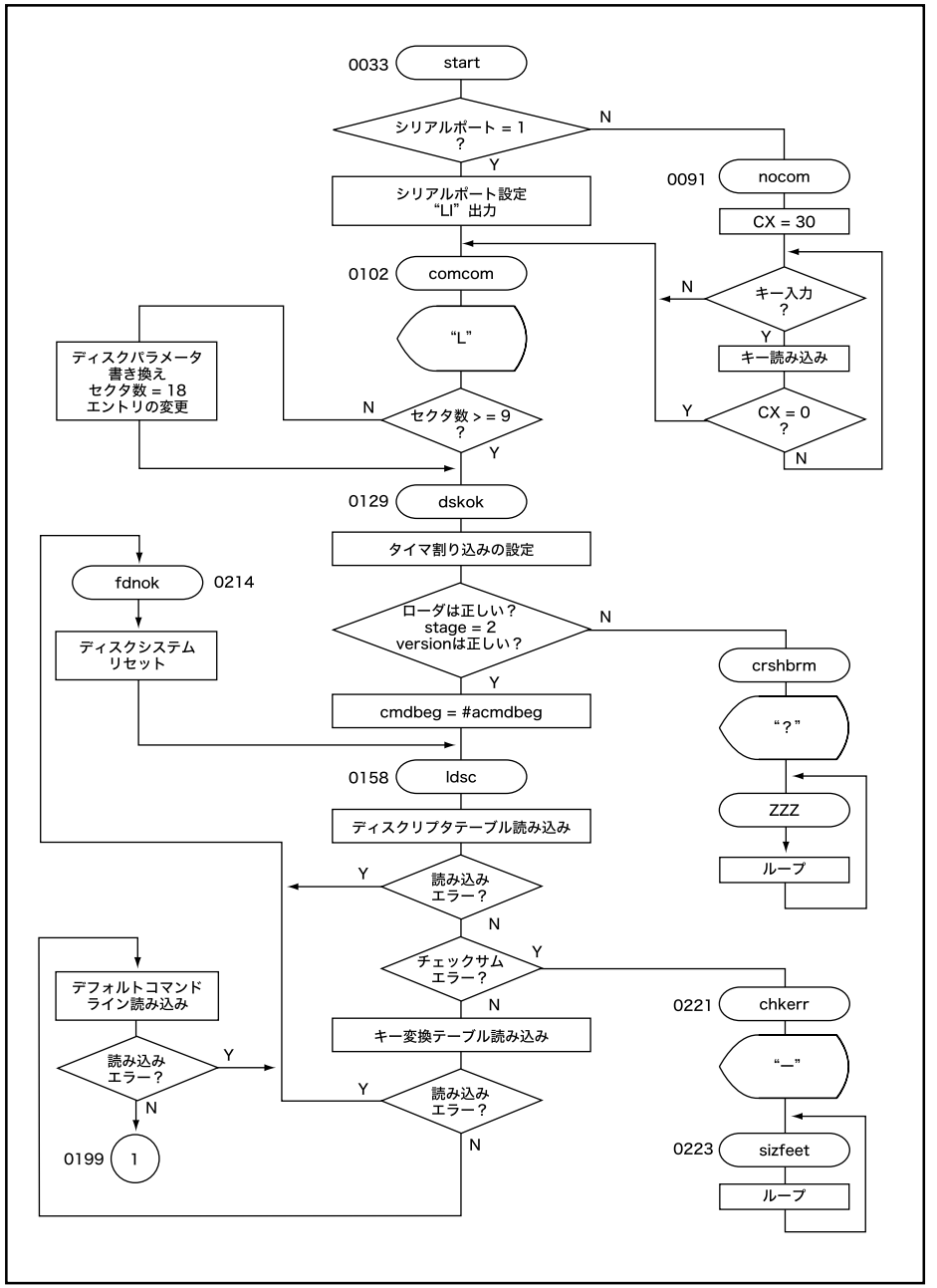
lilo.confにpromptの記述がない場合は、シフトモードキーの入力を待ちます。シフトモードキーが押されると対話モードに切り替わります。入力がないときはタイムアウト後に1番最初のイメージを起動します。対話モード(promptがあるとき)の場合は、キー入力に従って処理が開始されます。

```
0261  dokay:   mov     bx,#ospc                        ! ospc=“0”
0262           call    say                             ! LILOの最後の文字、“0”を表示する
0263           mov     ospc,#0                         ! ospc=0メッセージを無効にする
0264           mov     word ptr vgaovr,#VGA_NOCOVR     ! VGAオーバーライドを無効にする(vgaovr=0x8000)
0265           mov     word ptr memlim,#0              ! メモリ制限を無効にする(memlim=0)
0266           xor     ax,ax                           ! ブート時のキー入力待ち時間を得る
0267           seg     ss
0268           xchg    ax,DSC_OFF-8+SSDIFF             ! AX=タイムアウト値=[0x9B00:0x0F]
0269           or      old_del,ax                      ! old_del=AX=タイムアウト値
0270           mov     nodfl,#iloop                    ! nodfl(ジャンプ先アドレス)=#iloop
0271           seg     ss                              ! iloopは対話モード
0272           cmp     byte ptr DSC_OFF+15+SSDIFF,#0   ! プロンプトオプションが“0”かどうか
                                                       ! lilo.confにpromptオプションの記述がないときは0になる
0273           jne     extp                            ! 0でないときはextp(0280)へジャンプ
```

ブート時promptオプションがないとき

```
0274           mov     nodfl,#bfirst                   ! nodfl(ジャンプ先アドレス)=#bfirst。キー入力がなくタイムアウトのときは最初のイメージ
                                                       ! をロードする
0275           call    waitsh                          ! 次のキーの押下、あるいはタイムアウトまで待つ
                                                       ! 「Shift」か「Control」か「Alt」が押されるとCY=1
                                                       ! 「CapsLock」か「ScrllLock」がONだとCY=1
                                                       ! タイムアウトの場合はCY=0
0276           jc      iloop                           ! 上のキーが押された場合はCY=1なのでiloop(0314)へジャンプ(対話モード)
0277
0278                                                   !外部パラメをチェックして分岐する
0279                                                   !外部パラメータはext_si、ext_dx、ext_bx、ext_dlです
0280  extp:    seg     ss                              ! 次のデータはセグメントをSS=0x9000で読む
0281           cmp     byte ptr EX_OFF+6,#EX_DL_MAG    ! ext_dlは0xFEかどうか?
                                                       ! EX_OFF=CODE_START_1+SSDIFF=0x68+0xA000=0xA068
0282           jne     noex                            ! 違うときはnoex(0302)へ
0283           seg     ss
0284           mov     byte ptr EX_OFF+6,#0            ! ex_dl=0をクリアする
0285           seg     ss
0286           les     bx,EX_OFF                       ! BX=ext_esのオフセット値=0xA068
0287           seg     es                              !“LI”
0288           cmp     (bx),#EX_MAG_L                  ! ext_siは‘LI’か?
0289           jne     noex                            ! 違うときはnoex(0302)へ
0290           seg     es                              !“LO”か?
0291           cmp     (bx+2),#EX_MAG_H                ! ext_esは‘LO”?
0292           jne     noex                            ! 違うときはnoex(0302)へ
0293           seg     ss
0294           mov     si,EX_OFF+4                     ! SI=ext_bxのオフセット=0xA06B
0295           seg     es
0296           cmp     byte ptr (si),#0                ! emptyか?、ext_bxは0か?
0297           je      iloop                           ! 0のときはiloop(0314)へジャンプ(対話モード)
0298           jmp     niloop                          ! 0でないときはniloop(0342)へ(非対話モード)
0299
0300                                                   ! No external parameters after timeout -> boot first image
0301                                                   ! タイムアウト後、あるいは外部パラメータが無いときここへくる
0302  noex:    mov     ax,cs                           ! AX=CS
0303           mov     es,ax                           ! ES=AX=0x9B00(ESを再設定)
0304           mov     si,#DFLCMD+2                    ! SI=デフォルトコマンドラインの3バイト目のアドレス
0305           cmp     byte ptr (si),#0                ! デフォルトコマンドライン の3バイト目が0かどうか?
0306           jne     niloop                          ! 0でないときはniloop(0342)へ
0307           mov     ax,nodfl                        ! AX=nodfl
0308           jmp     ax                              ! nodfl=iloop(0314)またはbfirst(0562)へジャンプする
0309
0310  tolist:  br      list                            ! list(0227)へ
```

← フローチャート1



↓ フローチャート2