Linux ブートアップ ソースコードの解析

カーネルのセットアップを調べる ——head.Sの解読

芳之内弘

前回までは、Setup.SとSetup.Sにインクルードされているvideo.Sの解読をしました。今回は、Setup.Sからジャンプしてきたlinux/arch/i386/boot/compressed/head.Sから始めます。

今回調べるソースコードは、本誌2000年8月号付録CD-ROM中のv2.4/linux-2.4.0-test1.tar.bz2を展開して得られるlinux/arch/i386/boot/compressed/head.Sと同じディレクトリのmisc.cと、linux/arch/i386/kernel/head.Sです。head.Sが2種類あるので注意してください。内容は全く異なります。

なお今回は、シーケンスが単純なので、フローチャートは割愛しました。

misc.cは、C言語で書かれていて、比較的分かりやすいので解読は割愛します。内容は、圧縮されたカーネルの読み込まれ方に応じて解凍するアドレスを変えるようになっています。実際に解凍する関数は、linux/lib/inflate.cのgunzip関数です。

BOOT head.S

Head.Sの概要は、以下の通りです。

- A20(1Mbytes以上のメモリ)が有効かどうかをチェック。
- NTフラグとデータエリアをクリア。
- メモリに読み込まれているカーネルを解凍します。解凍は、kernel/head.Sがmisc.cのdecompress_kernel関数をコールし、このdecompress_kernel関数が、さらにlinux/lib/inflate.cのgunzip関数をコールして行います。
- 解凍したカーネルは、場合によっては0x100000にリロードされます。
- 0x100000にジャンプして、kernel/head.Sのstartup_32に制御が移ります。
- CPUレジスタとBSSをクリア。
- setup_idtをコールして割り込み記述子テーブルをセットアップ。
- A20をチェックし、EFLAGを初期化。
- Setup.Sがセットしたブートアップパラメータをempty_zero_pageにコピー。
- CPUとコプロセッサのタイプをチェック。
- ページテーブルをセット。
- CPUの割り込み記述子テーブルレジスタとグローバル記述子テーブルレジスタをセット。
- ローカル記述子テーブルレジスタをクリア。
- start_kernel関数をコールして制御を移す

BOOT GASの概要

今回からは、アセンブラがGNU Assembler(GAS)になり、AS86とは、以下に挙げるように、若干、書式が異なります。

- オペランドの方向
  最も大きな違いは、オペランドの方向です。GASでは、左から右の方向にデータが移動します。例えば、

```
movl  %eax,%ds
```

  は、EAXレジスタの内容をDSへコピーします。AS86の場合とは方向が逆です。レジスタは「%」記号を前置して区別します。

- ラベルの使い方
  「jmp 1b」の場合は、このコードより前にあるラベル1に移動します(backward)。「jmp 1f」の場合はこのコードよ

り後にあるラベル1に移動します(forward)。

・オペランドの長さ
オペランドの長さは、命令に「b」(byte)、「w」(word)、「l」(long)のサフィックスをつけて区別します。

・即値
即値(immediate operand)には「$」を前置します。例えば、

```
addl  $5 ,%eax
```

は、レジスタEAXにlong型の数値「5」を直接加算します。

・メモリアドレス
プレフィックスのないオペランドは、メモリアドレスを意味します。つまり、「movl $foo,%eax」は、値fooをEAXに書き込み、「movl foo,%eax」は、アドレスfooで示すメモリの値をEAXに書き込みます。

・間接参照
インデックスを使った間接参照の例を次に示します。

```
testb $0x80,17(%ebp)
```

これは、レジスタEBPで示すアドレスに「17」を加えたアドレスのメモリの値を0x80と比較します。

BOOT CPUID命令の概説とCR0、CR4の概説

head.Sを読み始める前に、知っておいた方が良いことを、ここでまとめておきます。

まず、CPUの情報を得る「CPUID」命令ですが、使い方は、表1のようになっています。

それから、CR0およびCR4レジスタ(i386やIntel 486にはCR4は実装されていません)の用途についても、表2、表3にまとめましたので、きちんと押さえておきましょう。

BOOT compressed/head.Sの解読

以下のリストの左側の数字は、Linuxのprコマンドを使って、compressed/head.Sのファイルに付けた行番号です。これは、

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

とすることで作成することができます。

表1　CPUID命令の使い方

EAX=0で実行
EAX　CPUIDの最大入力値(Penium Proでは「2」)
EBX　"Genu"
ECX　"inel"
EDX　"ntel"
EAX=1で実行
EAX　タイプ(bit_13、bit_12)
ファミリ(bit_11~bit_8)
モデル(bit_7~bit_4)
ステッピングID(bit_3~bit_0)
EBX　予約
ECX　予約
EDX　機能情報
bit_0 ：FTP(オンチップ浮動小数点ユニット内蔵)
bit_1 ：VME(仮想8086モード強化)
bit_2 ：DE(デバッグ拡張)
bit_3 ：PSE(ページサイズ拡張)
bit_4 ：TSC(タイムスタンプカウンタサポート)
bit_5 ：MSR(モデル固有レジスタサポート)
bit_6 ：PAE(物理アドレス拡張サポート)
bit_7 ：MCE(マシンチェック例外)
bit_8 ：CX8(CMPXCHG8B命令サポート)
bit_9 ：APIC(APICを所有しエネーブル)
bit_12：MTTR(メモリタイプレンジレジスタサポート)
bit_13：プロセッサによるPGE-PTEグローバルフラグイネーブル
bit_14：MCA(マシーンチェックアーキテクチャ)
bit_15：CMOV(条件付き移動および比較命令サポート)

表2
CR0レジスタの主なビット

ビット	役割
bit_31	PG：ページングを有効にする。
bit_18	AM：自動アライメントチェクを有効にする。
bit_16	WP：スーパバイザレベルでユーザーレベルの読み出し専用ページへの書き込みを禁止する。
bit_5	NE：FPUエラーの報告を有効にする。
bit_4	ET：386や486でET=1のときは387DXがサポート。P6ファミリでは常に1。
bit_3	TS：タスクスイッチ。
bit_2	EM：EM=1のときはFPUがないことを意味する。

表3
CR4レジスタの主なビット

ビット	役割
bit_4	PSE：4Mbytesのページサイズを有効にする。
bit_5	PAE：36bitの物理アドレスを有効にする。

リスト1

```
■レジスタの初期設定
 31 startup_32:
 32         cld                                     ; アドレスが大きくなる方向にカウントする
 33         cli                                     ; 割り込みを禁止する
 34         movl $(__KERNEL_DS),%eax                ; EAX=0x10(setup.Sで設定済み)
 35         movl %eax,%ds                           ; DS=ES=FS=GS=0x10にセットする
 36         movl %eax,%es                           ; セレクタ0x10のオフセット=0x000000
 37         movl %eax,%fs                           ; FS=EAX=0x10
 38         movl %eax,%gs                           ; GS=EAX=0x10
 39
 40         lss SYMBOL_NAME(stack_start),%esp       ; スタックポインタの設定
 41         xorl %eax,%eax                          ; EAX=0

 42 1:      incl %eax                               ; EAXに1を加える
 43         movl %eax,0x000000                      ; EAXの内容をメモリ0x000000に書き込む
 44         cmpl %eax,0x100000                      ; EAXの内容とメモリ0x100000の内容を比較する
 45         je 1b                                   ; 同じ場合は42行目の「1:」へジャンプする
                                                    ; A20が有効だと0x000000と0x100000の内容は異なる
■EFLAGをクリアし、BSS領域をクリアする
 51         pushl $0                                ; スタックに0をプッシュする
 52         popfl                                   ; EFLAGSにスタックの0をポップする
 56         xorl %eax,%eax                          ; EAX=0
 57         movl $ SYMBOL_NAME(_edata),%edi         ; EDI=(BSSの開始アドレス)
 58         movl $ SYMBOL_NAME(_end),%ecx           ; ECX=(BSSの終了アドレス)
 59         subl %edi,%ecx                          ; ECX=(BSS領域のバイト数)
 60         cld                                     ; アドレスが増加する方向にセット
 61         rep                                     ; 次の命令をECXの値だけ繰り返す
 62         stosb                                   ; EDIで示すメモリをクリアする
■カーネルを解凍する
 66         subl $16,%esp                           ; ESP=ESP-16
 67         pushl %esp                              ; 第1引数の構造体のアドレスをプシュする
 68         call SYMBOL_NAME(decompress_kernel)     ; misc.cの関数をコール
 69         orl  %eax,%eax                          ; 関数のリターン値をチェック
 70         jnz  3f                                 ; リターン値が0でないときは80へジャンプ
                                                    ; (リターン値が0のときは、カーネルは0x100000から解凍されている)

 71         xorl %ebx,%ebx                          ; EBX=0
 72         ljmp $(__KERNEL_CS), $0x100000          ; 0x100000(kernel/head.Sのstartup_32)へジャンプする。
■0x100000以下を使用して解凍しているので、0x100000へリロードする
 80 3:                                              ; 70よりこのラベルへジャンプしてくる

 81         movl $move_routine_start,%esi           ; カーネルのリロードルーチンの開始アドレス
 82         movl $0x1000,%edi                       ; リロードルーチンを読み込む最初のアドレス
 83         movl $move_routine_end,%ecx             ; ECX=最後のアドレス
 84         subl %esi,%ecx                          ; ECX=ECX-ESI(リロードルーチンの長さ)
 85         cld                                     ; アドレスが増加する方向にセット
 86         rep                                     ; 次の命令をECXの値の回数繰り返す
 87         movsb                                   ; ESIが示すアドレスの内容をEDIが示すアドレスへコピーする
                                                    ; (その後、ESIとEDIには1が加えられる)
■スタックより構造体のデータを取り出す(misc.cのddecompress_kernelで設定した値)
■C言語の関数の引数はスタックで渡されるので、スタックよりデータを取り出す
 89         popl %esi                               ; スタックの内容を4bytesバイト捨てる
 90         popl %esi                               ; ESI=low_buffer_start(低位のバッファの開始アドレス)
 91         popl %ecx                               ; ECX=lcount(低位のバッファのバイト数)
 92         popl %edx                               ; EDX=high_buffer_start(高位のバッファの開始アドレス)
 93         popl %eax                               ; EAX= hcount(高位のバッファのバイト数)
 94         movl $0x100000,%edi                     ; EDI=0x100000(移動先のアドレス)
```

リスト1 （つづき）

```
 95        cli                                   ; 割り込みを禁止する
 96        ljmp $(__KERNEL_CS), $0x1000          ; カーネルを0x100000へリロードするルーチンへジャンプ
```

■解凍したカーネルを0x100000へリロードするルーチン

```
102  move_routine_start:
103        rep                                   ; 最初にlow_buffer_startから
104        movsb                                 ; lcountのバイト数0x100000へコピーする
105        movl %edx,%esi                        ; ESI=EDX=high_buffer_start
106        movl %eax,%ecx                        ; ECX=EAX=hcount
107        rep                                   ; 続いてhigh_buffer_startからのコードを
108        movsb                                 ; hcountバイト数だけコピーする
109        xorl %ebx,%ebx                        ; EBX=0(EBXをクリア)
115        mov  $0x90000,%esp                    ; ESP=0x90000
116        ljmp $(__KERNEL_CS), $0x100000        ; 0x100000へジャンプする。
                                                 ; (__KERNEL_CS=0x10で、このセレクタのオフセットアドレスは0なので、
                                                 ; 0x100000へジャンプする。このアドレスは次のstartup_32になる)
117  move_routine_end:
```

BOOT kernel/head.Sの解読

次のコードは、linux/arch/i386/kernel/head.Sにライン番号を付加したものです。

リスト2

```
 20  #define CL_MAGIC_ADDR      0x90020            ; マジックコードのアドレス
 21  #define CL_MAGIC    0xA33F                    ; マジックコード
 22  #define CL_BASE_ADDR0x90000                   ; ブートパラメータのベースアドレス
 23  #define CL_OFFSET   0x90022                   ; コマンドラインのアドレスが書き込まれている
```

■boot_cpu_data構造体のメンバ(CPU情報を書き込む構造体)

```
 29  #define CPU_PARAMS  SYMBOL_NAME(boot_cpu_data)
 30  #define X86         CPU_PARAMS+0
 31  #define X86_VENDOR  CPU_PARAMS+1
 32  #define X86_MODEL   CPU_PARAMS+2
 33  #define X86_MASK    CPU_PARAMS+3
 34  #define X86_HARD_MATH      CPU_PARAMS+6
 35  #define X86_CPUID   CPU_PARAMS+8
 36  #define X86_CAPABILITY     CPU_PARAMS+12
 37  #define X86_VENDOR_ID      CPU_PARAMS+16
 44  startup_32:
```

■セグメントレジスタの初期化

```
 48        cld
 49        movl $(__KERNEL_DS),%eax              ; EAX=__KERNEL_DS=0x18
 50        movl %eax,%ds                         ; DS=EAX
 51        movl %eax,%es                         ; ES=EAX
 52        movl %eax,%fs                         ; FS=EAX
 53        movl %eax,%gs                         ; GS=EAX
 54  #ifdef CONFIG_SMP
 55        orw %bx,%bx                           ; BXのチェック
 56        jz 1f                                 ; BX=0で78へジャンプする


 71  #define cr4_bits mmu_cr4_features-__PAGE_OFFSET
 72        cmpl $0,cr4_bits                      ; cr4_bitsをチェック
 73        je 3f                                 ; cr4_bits=0で94へジャンプする
```

リスト2 （つづき）

```
 74          movl %cr4,%eax                          ; EAX=CR4 PSE,PAEを有効にする
 75          orl cr4_bits,%eax                       ; EAX=(EAX | cr4_bits)
 76          movl %eax,%cr4                          ; CR4のページングを有効にする(PSE、PAE)
 77          jmp 3f                                  ; 94へジャンプ
 78  1:
 79  #endif
 80  /*
```

■ページテーブルの初期化

```
 83          movl $pg0-__PAGE_OFFSET,%edi            ; EDI=ページゼロのエントリテーブル(PEI)アドレス
 84          movl $007,%eax                          ; EAX=007
 86  2:      stosl                                   ; PEIに007を書き込む。EDI=EDI+1
                                                     ; bit0=PRESENT  bit1=RW  bit2=SUPER-USER-MODE
 87          add $0x1000,%eax                        ; EAX=EAX+0x1000(テーブルアドレスのセット)
 88          cmp $empty_zero_page-__PAGE_OFFSET,%edi ; empty_zero_pageの手前まで書き込む
 89          jne 2b
 94  3:
 95          movl $swapper_pg_dir-__PAGE_OFFSET,%eax ; EAX=swapper_pg_dirのアドレス
 96          movl %eax,%cr3                          ; CR3=swapper_pg_dirのアドレス
 97          movl %cr0,%eax                          ; CR0=EAX
 98          orl $0x80000000,%eax                    ; EAXのbit31(PGビット)=1にセット
 99          movl %eax,%cr0                          ; CR0のPGビットをセット(ページングを有効にする)
100          jmp 1f                                  ; プリフェッチキューをクリアするため101へジャンプする
101 1:
102          movl $1f,%eax                           ; EAX=104のアドレス
103          jmp *%eax                               ; 104へジャンプする
104 1:


106          lss stack_start,%esp                    ; ESP=stack_start
108 #ifdef CONFIG_SMP
109          orw  %bx,%bx                            ; bx=0?
110          jz  1f                                  ; bx=0で114へ
111          pushl $0                                ; スタックに0をプッシュ
112          popfl                                   ; スタックからEFLAGSに0を書き戻す
113          jmp checkCPUtype                        ; 163へジャンプ
114 1:
115          #endif CONFIG_SMP
```

■BSSエリアをクリアする

```
119          xorl %eax,%eax                          ; EAX=0
120          movl $ SYMBOL_NAME(__bss_start),%edi    ; EDI=__bss_start
121          movl $ SYMBOL_NAME(_end),%ecx           ; ECX=_end
122          subl %edi,%ecx                          ; ECX=ECX-EDI(領域のバイト数)
123          cld                                     ; アドレスの増加する方向
124          rep                                     ; __bss_startから_endまで
125          stosb                                   ; 0を書き込む
```

■IDTをセットし、EFLAGをクリアする

```
130          call setup_idt                          ; IDTをセットする
136          pushl $0                                ; 0をプッシュする
137          popfl                                   ; EFLAGS=0
```

■empty_zero_pageにブートパラメータ(2Kbytes)をコピーする

```
143          movl $0x90000,%esi                      ; ESI=0x90000
144          movl $ SYMBOL_NAME(empty_zero_page),%edi ; EDI=empty_zero_pageアドレス
145          movl $512,%ecx                          ; ECX=512(バイト数)
146          cld                                     ; 方向設定
147          rep                                     ; ESIで示すアドレスからEDIで示すアドレスへ
148          movsl                                   ; 4bytesごとコピーする(計2Kbytes)
```

■コマンドラインのクリア

```
149          xorl %eax,%eax                          ; EAX=0
```

リスト2 （つづき）

```
 74          movl %cr4,%eax                          ; EAX=CR4 PSE,PAEを有効にする
 75          orl cr4_bits,%eax                       ; EAX=(EAX | cr4_bits)
 76          movl %eax,%cr4                          ; CR4のページングを有効にする(PSE、PAE)
 77          jmp 3f                                  ; 94へジャンプ
 78  1:
 79  #endif
 80  /*
```

■ページテーブルの初期化

```
 83          movl $pg0-__PAGE_OFFSET,%edi            ; EDI=ページゼロのエントリテーブル(PEI)アドレス
 84          movl $007,%eax                          ; EAX=007
 86  2:      stosl                                   ; PEIに007を書き込む。EDI=EDI+1
                                                     ; bit0=PRESENT  bit1=RW  bit2=SUPER-USER-MODE
 87          add $0x1000,%eax                        ; EAX=EAX+0x1000(テーブルアドレスのセット)
 88          cmp $empty_zero_page-__PAGE_OFFSET,%edi ; empty_zero_pageの手前まで書き込む
 89          jne 2b
 94  3:
 95          movl $swapper_pg_dir-__PAGE_OFFSET,%eax ; EAX=swapper_pg_dirのアドレス
 96          movl %eax,%cr3                          ; CR3=swapper_pg_dirのアドレス
 97          movl %cr0,%eax                          ; CR0=EAX
 98          orl $0x80000000,%eax                    ; EAXのbit31(PGビット)=1にセット
 99          movl %eax,%cr0                          ; CR0のPGビットをセット(ページングを有効にする)
100          jmp 1f                                  ; プリフェッチキューをクリアするため101へジャンプする
101 1:
102          movl $1f,%eax                           ; EAX=104のアドレス
103          jmp *%eax                               ; 104へジャンプする
104 1:


106          lss stack_start,%esp                    ; ESP=stack_start
108 #ifdef CONFIG_SMP
109          orw  %bx,%bx                            ; bx=0?
110          jz  1f                                  ; bx=0で114へ
111          pushl $0                                ; スタックに0をプッシュ
112          popfl                                   ; スタックからEFLAGSに0を書き戻す
113          jmp checkCPUtype                        ; 163へジャンプ
114 1:
115          #endif CONFIG_SMP
```

■BSSエリアをクリアする

```
119          xorl %eax,%eax                          ; EAX=0
120          movl $ SYMBOL_NAME(__bss_start),%edi    ; EDI=__bss_start
121          movl $ SYMBOL_NAME(_end),%ecx           ; ECX=_end
122          subl %edi,%ecx                          ; ECX=ECX-EDI(領域のバイト数)
123          cld                                     ; アドレスの増加する方向
124          rep                                     ; __bss_startから_endまで
125          stosb                                   ; 0を書き込む
```

■IDTをセットし、EFLAGをクリアする

```
130          call setup_idt                          ; IDTをセットする
136          pushl $0                                ; 0をプッシュする
137          popfl                                   ; EFLAGS=0
```

■empty_zero_pageにブートパラメータ(2Kbytes)をコピーする

```
143          movl $0x90000,%esi                      ; ESI=0x90000
144          movl $ SYMBOL_NAME(empty_zero_page),%edi ; EDI=empty_zero_pageアドレス
145          movl $512,%ecx                          ; ECX=512(バイト数)
146          cld                                     ; 方向設定
147          rep                                     ; ESIで示すアドレスからEDIで示すアドレスへ
148          movsl                                   ; 4bytesごとコピーする(計2Kbytes)
```

■コマンドラインのクリア

```
149          xorl %eax,%eax                          ; EAX=0
```

リスト2 （つづき）

```
212         movl $1,%eax                        ; EAX=1
213         cpuid                               ; EAXにCPUIDを得る
214         movb %al,%cl                        ; CL=AL(ALをCLに保存する)
215         andb $0x0f,%ah                      ; AHの下位4bitを取り出す(CPUファミリ名)
216         movb %ah,X86                        ; [X86]=CPUファミリ名
217         andb $0xf0,%al                      ; ALの上位4bit(モデル名)を取り出す
218         shrb $4,%al                         ; AL=CPUモデル名
219         movb %al,X86_MODEL                  ; [X86_MODEL]=CPUモデル名
220         andb $0x0f,%cl                      ; CLの下位4bit(ステッピングID)を取り出す
221         movb %cl,X86_MASK                   ; [X86_MASK]=ステッピングID
222         movl %edx,X86_CAPABILITY            ; [X86_CAPABILITY]=EDX
```

■CPUが486の場合ここへジャンプする

```
224 is486:
225         movl %cr0,%eax                      ; AEX=CR0
226         andl $0x80000011,%eax               ; CR0のPG、PE、ETビット以外をクリアする
227         orl $0x50022,%eax                   ; CR0のAM、WP、NE、MPビットをセットする
228         jmp 2f                              ; 235へジャンプしてCR0にEXAの値を書き込む

230 is386: pushl %ecx                           ; ECX(最初のEFLAGS)をプッシュ
231         popfl                               ; EFLAGSを戻す
232         movl %cr0,%eax                      ; EAX=CR0
233         andl $0x80000011,%eax               ; CR0のPG、PE、ETビット以外をクリアする
234         orl $2,%eax                         ; CR0のMPビットをセットする
235 2:      movl %eax,%cr0                      ; CR0にEAXの内容を書き込む
236         call check_x87                      ; check_x87(278)をコールしてコプロセッサをチェックする
237 #ifdef CONFIG_SMP
238         incb ready                          ; [ready]=[ready]+1
239 #endif
240         lgdt gdt_descr                      ; gdt_descr(368)のアドレスをGDTRにセットする
241         lidt idt_descr                      ; idt_descr(362)のアドレスをIDTRにセットする
242         ljmp $(__KERNEL_CS),$1f             ; 243へジャンプする
243 1:      movl $(__KERNEL_DS),%eax            ; EAX=0x18セグメントレジスタの再設定
244         movl %eax,%ds                       ; DS=EAX
245         movl %eax,%es                       ; ES=EAX
246         movl %eax,%fs                       ; FS=EAX
247         movl %eax,%gs                       ; GS=EAX
248 #ifdef CONFIG_SMP
249         movl $(__KERNEL_DS), %eax           ; EAX=0x18
250         movl %eax,%ss                       ; SS=EAX(SSを再設定する)
251 #else
252         lss  stack_start,%esp               ; ESP=stack_start(ESPの設定)
253 #endif
254         xorl %eax,%eax                      ; EAX=0
255         lldt %ax                            ; LDTR=0(ローカル記述子テーブル)
256         cld                                 ; ディレクションフラグをクリア(gccの標準)
257 #ifdef CONFIG_SMP
258         movb ready, %cl                     ; CL=[ready]
259         cmpb $1,%cl                         ; CL=1?
260         je 1f                               ; Yesなら1番目のCPUなのでstart_kernelをコール
261                                             ; 1番目以外のCPUならinitialize_secondaryをコール
262         call SYMBOL_NAME(initialize_secondary)
263         jmp L6                              ; 267へ
264 1:
265 #endif
266         call SYMBOL_NAME(start_kernel)      ; start_kernelへ制御を移す
267 L6:
268         jmp L6                              ; 無限ループ
271 #ifdef CONFIG_SMP
272 ready: .byte 0                              ; CPUのカウンター(SMPのときに使う)
273 #endif
```

リスト2 （つづき）

```
■コプロセッサのチェック
■i80386やIntel486では、CR0のbit_4(ETフラグ)が1のときは、コプロセッサがi80287もしくはIntel387DXのいずれかであることを示します。
278 check_x87:
279         movb $0,X86_HARD_MATH                   ; [X86_HARD_MATH]=0
280         clts                                    ; CR0のTSフラグをクリア
281         fninit                                  ; 浮動小数点ユニット(FPU)を初期化する
282         fstsw %ax                               ; AX=(FPUのステータスワード)
283         cmpb $0,%al                             ; AL=0 ?
284         je 1f


285         movl %cr0,%eax                          ; EAX=CR0(コプロセッサがないときはエミュレーションモード)
286         xorl $4,%eax                            ; bit_2(EMビット)を反転する
287         movl %eax,%cr0                          ; CR0=EAX(EMビットセット)
288         ret
289         ALIGN
■コプロセッサがある場合
290 1:      movb $1,X86_HARD_MATH                   ; [X86_HARD_MATH]=1
291         .byte 0xDB,0xE4                         ; 287に対してfsetpmを実行する387では無視される
292         ret
```

<これ以降は省略します>

BOOT 最後に

Linuxのブートアップの仕組みを9回に渡って調べてきました。PCをリセットしてから、start_kernelまでは、ハードウェアに依存する部分が多く、CPUに依存するアセンブラでかかれていました。従って、ソースコードの追跡も面倒で、取りつきにくく、詳しく解説されたものがあまりありませんでした。

しかし、次の段階のstart_kernelからは、そのほとんどがC言語で書かれています。また、本誌2000年11月号より連載が開始された「図解Linuxカーネル2.4の設計と実装」では、カーネルの起動についての解説が予告されております。私はOSの専門家でもなく、ソフトの専門家でもありません。Linuxを初めて手にして、その起動の仕組みに興味を持ちましたが、詳しく書かれたものがなく、非力を省みず、解析にとりかかりました。しかし、ここにきて、専門家による解説が予告されましたので、これ以降の解説はその方にお願いしたいと思います。私はまた別の機会に、カーネルが立ち上がった次の段階の、init以降の解析をしたいと思っております。

また、組み込みではないLinuxのFAの分野での応用例についても、私の経験から具体的に書いて紹介していきたいと思っております。みなさんのご支援をお願いいたします。長い間のご支援ありがとうございました。

R E S O U R C E

[1] x86 CPUのプロテクトモードに関する参考文献
「初めて読む486」(簡池輝尚著／アスキー出版)
「80486の使い方」(W.B.スルヤント著／オーム社)
「Inter Architecture Software Developer's Manual Volume 3」(Intel)

[2] CPUの資料
Pentium Proファミリデベロッパーズマニュアル(Intel)