第 2 刷・第 1 刷訂正情報

本書（第 2 刷および第 1 刷）の掲載内容に下記の誤りがございました。
ご迷惑をおかけしましたことをお詫び申し上げます。

（2020 年 1 月 17 日更新）

P.225　1 番目の表

誤：

書式	第 1 オペランド	第 2 オペランド	OF	SF	ZF	CF
CLD			–	0	–	–
STD			–	1	–	–
CLI			–	–	0	–
STI			–	–	1	–
CLC			–	–	–	0
STC			–	–	–	1

正：

書式	第 1 オペランド	第 2 オペランド	OF	DF	IF	CF
CLD			–	0	–	–
STD			–	1	–	–
CLI			–	–	0	–
STI			–	–	1	–
CLC			–	–	–	0
STC			–	–	–	1

P.243　図 8-35 中の右上

誤：

割り込みディスクリプタ

正：

割り込みゲートディスクリプタ

P.243　上から 2 段落目

誤：

ゲートディスクリプタには、セグメントディスクリプタも記載されているので、メモリ保護機能を有効にしたままセグメント内のオフセットに処理を移行することができます。

正：

ゲートディスクリプタには、セグメントセレクタも記載されているので、メモリ保護機能を有効にしたままセグメント内のオフセットに処理を移行することができます。

P.264　下部のコード

誤：

```
%assign i 3
%rep 3
        %if i <= 5
                %exitrep
        %endif
        db  i
        %assign i i+3
%endrep
```

正：

```
%assign i 3
%rep 3
        %if i >= 5
                %exitrep
        %else
                db i
        %endif
        %assign i i+3
%endrep
```

P.264　下から 1 番目のコードの直後に追記

本来であれば、マクロ内で使用しているデータの定義「db i」を「%else」に含める必要はありません。しかしながら、NASM の一部のバージョンでは「%exitrep」を実行してもすぐに繰り返し処理を終了しないので、すべてのバージョンで同じ出力結果を得るために、このように記載しています。このソースファイルをアセンブルすると、次のリスティングファイルが生成されます。

P.265　上部のコード

誤：

```
     1                                  %assign i 3
     2                                  %rep 3
     3                                          %if i <= 5
     4                                                  %exitrep
     5                                          %endif
     6                                          db  i
     7                                          %assign i i+3
     8                                  %endrep
     9                             <1>  %if i <= 5
    10                             <1>  %exitrep
    11                             <1>  %endif
    12 00000000 03                 <1>  db i
    13                             <1>  %assign i i+3
```

正：

```
     1                                  %assign i 3
     2                                  %rep 3
     3                                          %if i >= 5
     4                                                  %exitrep
     5                                          %else
     6                                                  db i
     7                                          %endif
     8                                          %assign i i+3
     9                                  %endrep
     9                             <1>  %if i >= 5
     9                             <1>  %exitrep
     9                             <1>  %else
     9 00000000 03                 <1>  db i
     9                             <1>  %endif
     9                             <1>  %assign i i+3
     9                             <1>  %if i >= 5
     9                             <1>  %exitrep
     9                             <1>  %else
     9                             <1>  db i
     9                             <1>  %endif
     9                             <1>  %assign i i+3
```

P.269　2 番目のコード

誤：

```
        push    ax                              ;           // 右端の引数をスタックにプッシュ
        call    putc, ax                        ; pubc(AX); // 1文字表示
        add     sp, 2                           ;           // スタックを破棄
```

正：

```
        push    ax                              ;           // 右端の引数をスタックにプッシュ
        call    putc                            ; pubc(AX); // 1文字表示
        add     sp, 2                           ;           // スタックを破棄
```

P.346　本文上から 6 行目

誤：

環境設定などを保存する「evn」

正：

環境設定などを保存する「env」

P.434　1 番目の段落

誤：

　プログラムの実行結果は、次のとおりです。画面表示から、最大セクタ数が 20（0x14）ヘッド数が 2 そしてシリンダ数が 16（0x10）であることが分かります。この値は、Bochs 起動時のパラメータに設定した値となっています。

正：

　プログラムの実行結果は、次のとおりです。画面表示から、最大シリンダ数が 20（0x14）ヘッド数が 2 そしてセクタ数が 16（0x10）であることが分かります。この値は、Bochs 起動時のパラメータに設定した値となっています。

P.439　2 番目のコードの直後に追記

　この定義を使ってメモリを確保する例を次に示します。

prog/src/modules/real/get_mem_info.s

```
ALIGN 4, db 0
.b0:    times E820_RECORD_SIZE db 0
```

P.449　3 番目のコード内（上から 8 行目）

誤：

loopnz .10L　; } while (--CX && !ZF);」

正：

loopz .10L　; } while (--CX && ZF);

P.460　下部のコード

誤：

prog/src/modules/real/read_lba.s

```
lba_chs:
        ...                             ; スタックフレームの作成とレジスタの保存

        mov     al,  [si + drive.head]  ; AL = 最大ヘッド数;
        mul     byte [si + drive.sect]  ; AX = 最大ヘッド数 * 最大セクタ数;
        mov     bx, ax                  ; BX = シリンダあたりのセクタ数;
```

正：

prog/src/modules/real/lba_chs.s

```
lba_chs:
        ...                             ; スタックフレームの作成とレジスタの保存

        mov     si, [bp + 4]            ; SI  = driveバッファ;
        mov     di, [bp + 6]            ; DI  = drv_chsバッファ;

        mov     al,  [si + drive.head]  ; AL = 最大ヘッド数;
        mul     byte [si + drive.sect]  ; AX = 最大ヘッド数 * 最大セクタ数;
        mov     bx, ax                  ; BX = シリンダあたりのセクタ数;
```

P.461　上から 1 番目のコードのファイル名

誤：

prog/src/modules/real/read_lba.s

正：

prog/src/modules/real/lba_chs.s

P.461　上から 2 番目のコードのファイル名

誤：

prog/src/modules/real/read_lba.s

正：

prog/src/modules/real/lba_chs.s

P.473　上から 2 番目のコードのファイル名

誤：

prog/src/16_protect_mode/boot.s

正：prog/src/include/define.s

P.481　上から 1 番目のコード内

誤：

mov [FONT], eax

正：

mov [FONT_ADR], eax

P.487　図 16-8 内の 2 か所

誤：

前景色

正：

背景色

P.493　上部のコード

誤：

draw_char(row, col, color, ch);	
戻り値	なし
row	列（0～79）
col	行（0～29）
color	描画色
ch	文字

正：

draw_char(col, row, color, ch);	
戻り値	なし
col	列（0～79）
row	行（0～29）
color	描画色
ch	文字

P.497　上部のコード

誤：

draw_font(row, col);	
戻り値	なし
row	列
col	行

正：

draw_font(col, row);	
戻り値	なし
col	列
row	行

P.499　1 番目の表内

誤：

draw_str(row, col, color, p);	
戻り値	なし
row	列
col	行
color	描画色
p	文字列のアドレス

正：

draw_str(col, row, color, p);	
戻り値	なし
col	列
row	行
color	描画色
p	文字列のアドレス

P.502　1 番目の表内

誤：

draw_color_bar(row, col);	
戻り値	なし
row	列
col	行

正：

draw_color_bar(col,row);	
戻り値	なし
col	列
row	行

P.502　1 番目のコード

誤：

prog/src/modules/protect/draw_color_bar.s

```
draw_color_bar:
        ...
                                                ;
        ;---------------------------------------
        ; カラーバーを表示
        ;---------------------------------------
        mov     ecx, 0                          ; for (ECX = 0;
.10L:   cmp     ecx, 16                         ;      ECX < 16;
        jae     .10E                            ;
                                                ;      ECX++)
                                                ; {
        ...                                     ;   // カラーバー表示処理
                                                ;
        inc     ecx                             ;   // for (... ECX++)
        jmp     .10L                            ;
.10E:                                           ; }
```

正：

prog/src/modules/protect/draw_color_bar.s

```
draw_color_bar:
        ...
                                                ;
        mov     esi, [ebp + 8]                  ; ESI = X (列)
        mov     edi, [ebp +12]                  ; EDI = Y (行)

        ;---------------------------------------
        ; カラーバーを表示
        ;---------------------------------------
        mov     ecx, 0                          ; for (ECX = 0;
.10L:   cmp     ecx, 16                         ;      ECX < 16;
        jae     .10E                            ;
                                                ;      ECX++)
                                                ; {
        ...                                     ;   // カラーバー表示処理
                                                ;
        inc     ecx                             ;   // for (... ECX++)
        jmp     .10L                            ;
.10E:                                           ; }
```

P.523　1 つ目の表内

誤：

draw_time(row, col, color, time);	
戻り値	なし
row	列
col	行
color	描画色
time	時刻データ

正：

draw_time(col, row, color, time);	
戻り値	なし
col	列
row	行
color	描画色
time	時刻データ

P.534　上から 2 番目のコードのファイル名

誤：

prog/src/27_int_div_zero/modules/interrupt.s

正：

prog/src/include/define.s

P.536　本文 2 段落目より

誤：

　作成した割り込み処理の登録は、割り込みゲートディスクリプタの初期化が終了した後で、対応するベクタ番号のディスクリプタにアドレスを設定するだけです。この処理は、マクロとして作成します。このマクロは、2 つの引数を取り、最初の引数で示されるベクタ番号に 2 番目の引数の割り込み処理アドレスを設定します。

prog/src/include/macro.s

```
%macro set_vect 1-*
        push    eax
        push    edi

        mov     edi, VECT_BASE + (%1 * 8)       ; ベクタアドレス;
        mov     eax, %2

        mov     [edi + 0], ax                   ; 例外アドレス[15: 0]
        shr     eax, 16                         ;
        mov     [edi + 6], ax                   ; 例外アドレス[31:16]

        pop     edi
        pop     eax
%endmacro
```

正：

　作成した割り込み処理の登録は、割り込みゲートディスクリプタの初期化が終了した後で、対応するベクタ番号のディスクリプタにアドレスを設定するだけです。この処理は、マクロとして作成します。このマクロは、2 つまたは 3 つの引数を取り、最初の引数で示されるベクタ番号に 2 番目の引数の割り込み処理アドレスを設定します。もし 3 番目の引数が指定されたときはゲートディスクリプタの属性として設定します。

prog/src/include/macro.s

```
%macro set_vect 1-*
        push    eax
        push    edi

        mov     edi, VECT_BASE + (%1 * 8)       ; ベクタアドレス;
        mov     eax, %2

    %if 3 == %0
        mov     [edi + 4], %3                   ; フラグ
    %endif

        mov     [edi + 0], ax                   ; 例外アドレス[15: 0]
        shr     eax, 16                         ;
        mov     [edi + 6], ax                   ; 例外アドレス[31:16]

        pop     edi
        pop     eax
%endmacro
```

P.551　下から１番目のコード

誤：

```
%define     RING_ITEM_SIZE      (1 << 4)
%define     RING_INDEX_MASK     (RING_ITEM_SIZE - 1)
```

正：

prog/src/include/macro.s

```
%define     RING_ITEM_SIZE      (1 << 4)
%define     RING_INDEX_MASK     (RING_ITEM_SIZE - 1)
```

P.552　上から１番目のコード

誤：

```
struc ring_buff
        .rp             resd    1               ; RP:書き込み位置
        .wp             resd    1               ; WP:読み込み位置
        .item           resb    RING_ITEM_SIZE  ; バッファ
endstruc
```

正：

prog/src/include/macro.s

```
struc ring_buff
        .rp             resd    1               ; RP:書き込み位置
        .wp             resd    1               ; WP:読み込み位置
        .item           resb    RING_ITEM_SIZE  ; バッファ
endstruc
```

P.553　上から番目のコード

誤：

prog/src/modules/protect/ring_buff.s

```
ring_wr:
        ...
        ;---------------------------------------
        ; 書き込み位置を確認
        ;---------------------------------------
        mov     eax, 0                          ; EAX  = 0;       // 失敗
        mov     ebx, [esi + ring_buff.wp]       ; EBX  = wp;      // 書き込み位置
        mov     ecx, ebx                        ; ECX  = EBX;
        inc     ecx                             ; ECX++;          // 次の書き込み位置
        and     ecx, RING_INDEX_MASK            ; ECX &= 0x0F     // サイズの制限
                                                ;
        cmp     ecx, [esi + ring_buff.rp]       ; if (ECX != rp)  // 読み込み位置と異なる
        je      .10E                            ; {
                                                ;
        mov     al, [ebp +12]                   ;   AL = データ;
                                                ;
        mov     [esi + ring_buff.item + ebx], al ;   BUFF[wp] = AL; // キーコードを保存
        mov     [esi + ring_buff.wp], ecx       ;   wp = ECX;      // 書き込み位置を保存
        mov     eax, 1                          ;   EAX = 1;       // 成功
.10E:                                           ; }
```

正：

prog/src/modules/protect/ring_buff.s

```
ring_wr:
        ...
        mov     esi, [ebp + 8]                  ; ESI = リングバッファ;

        ;---------------------------------------
        ; 書き込み位置を確認
        ;---------------------------------------
        mov     eax, 0                          ; EAX  = 0;       // 失敗
        mov     ebx, [esi + ring_buff.wp]       ; EBX  = wp;      // 書き込み位置
        mov     ecx, ebx                        ; ECX  = EBX;
        inc     ecx                             ; ECX++;          // 次の書き込み位置
        and     ecx, RING_INDEX_MASK            ; ECX &= 0x0F     // サイズの制限
                                                ;
        cmp     ecx, [esi + ring_buff.rp]       ; if (ECX != rp)  // 読み込み位置と異なる
        je      .10E                            ; {
                                                ;
        mov     al, [ebp +12]                   ;   AL = データ;
                                                ;
        mov     [esi + ring_buff.item + ebx], al ;   BUFF[wp] = AL; // キーコードを保存
        mov     [esi + ring_buff.wp], ecx       ;   wp = ECX;      // 書き込み位置を保存
        mov     eax, 1                          ;   EAX = 1;       // 成功
.10E:                                           ; }
```

P.555　下から２番目のコードにファイル名を追記

誤：

```
        outp    0x21, 0b_1111_1001              ; // 割り込み有効：スレーブPIC/KBC
        outp    0xA1, 0b_1111_1110              ; // 割り込み有効：RTC
```

正：

```
                                                                prog/src/29_int_keyboard/kernel.s
        outp    0x21, 0b_1111_1001              ; // 割り込み有効：スレーブPIC/KBC
        outp    0xA1, 0b_1111_1110              ; // 割り込み有効：RTC
```

P.556　上から１番目のコード

誤：

```
                                                                prog/src/modules/protect/ring_buff.s
draw_key:
        ...
        ;---------------------------------------
        ; リングバッファの情報を取得
        ;---------------------------------------
        mov     ebx, [esi + ring_buff.rp]       ; EBX = wp;             // 書き込み位置
        lea     esi, [esi + ring_buff.item]     ; ESI = &KEY_BUFF[EBX];
        mov     ecx, RING_ITEM_SIZE             ; ECX = RING_ITEM_SIZE; // 要素数
                                                ;
.10L:                                           ; do
                                                ; {
        ...                                     ;   【バッファ内要素の表示】
        loop    .10L                            ;
.10E:                                           ; } while (ECX--);
```

正：

```
                                                                prog/src/modules/protect/ring_buff.s
draw_key:
        ...
        mov     edx, [ebp + 8]                  ; EDX = X（列）;
        mov     edi, [ebp +12]                  ; EDI = Y（行）;
        mov     esi, [ebp +16]                  ; ESI = リングバッファ;

        ;---------------------------------------
        ; リングバッファの情報を取得
        ;---------------------------------------
        mov     ebx, [esi + ring_buff.rp]       ; EBX = wp;             // 書き込み位置
        lea     esi, [esi + ring_buff.item]     ; ESI = &KEY_BUFF[EBX];
        mov     ecx, RING_ITEM_SIZE             ; ECX = RING_ITEM_SIZE; // 要素数
                                                ;
.10L:                                           ; do
                                                ; {
        ...                                     ;   【バッファ内要素の表示】
        loop    .10L                            ;
.10E:                                           ; } while (ECX--);
```

P.556　下から１番目のコード（囲み内を薄い字で追記）

誤：

prog/src/29_int_keyboard/kernel.s

```
        cdecl   ring_rd, _KEY_BUFF, .int_key    ;   EAX = ring_rd(buff, &int_key);
        cmp     eax, 0                          ;   if (EAX == 0)
        je      .10E                            ;   {
                                                ;
        cdecl   draw_key, 2, 29, _KEY_BUFF      ;     ring_show(key_buff); // 全要素を表示
.10E:                                           ;   }

.int_key:   dd  0
```

正：

prog/src/29_int_keyboard/kernel.s

```
        cdecl   ring_rd, _KEY_BUFF, .int_key    ;   EAX = ring_rd(buff, &int_key);
        cmp     eax, 0                          ;   if (EAX == 0)
        je      .10E                            ;   {
                                                ;
        cdecl   draw_key, 2, 29, _KEY_BUFF      ;     ring_show(key_buff); // 全要素を表示
.10E:                                           ;   }

ALIGN 4, db 0
.int_key:   dd  0
```

P.560　下から１番目のコード（囲み内は薄い字で追記）

誤：

prog/src/30_int_timer/kernel.s

```
    outp    0x21, 0b_1111_1000              ; // 割り込み有効：スレーブPIC/KBC/タイマー
    outp    0xA1, 0b_1111_1110              ; // 割り込み有効：RTC
```

正：

prog/src/30_int_timer/kernel.s

```
    cdecl   rtc_int_en, 0x10                ; rtc_int_en(UIE); // 更新サイクル終了割り込み許可
    cdecl   int_en_timer0                   ; // タイマー（カウンタ0）割り込み許可

    outp    0x21, 0b_1111_1000              ; // 割り込み有効：スレーブPIC/KBC/タイマー
    outp    0xA1, 0b_1111_1110              ; // 割り込み有効：RTC
```

P.578･579　コードのファイル名（計４か所）

誤：

32_task_non_pre/task_1.s

正：

32_task_non_pre/tasks/task_1.s

P.585　下から１番目のコード（囲み内は薄い字で修正）

誤：

prog/src/34_call_gate/descriptor.s

```
GDT:            dq  0x0000000000000000          ; NULL
.cs_kernel:     dq  0x00CF9A000000FFFF          ; コードセグメント
.ds_kernel:     dq  0x00CF92000000FFFF          ; データセグメント
.ldt_0          dq  0x0000820000000000          ; LDT_0ディスクリプタ
.ldt_1          dq  0x0000820000000000          ; LDT_1ディスクリプタ
.tss_0:         dq  0x0000890000000067          ; TSS_0ディスクリプタ
.tss_1:         dq  0x0000890000000067          ; TSS_1ディスクリプタ
.call_gate:     dq  0x0000EC0400080000          ; 386コールゲート(DPL=3, count=4, SEL=8)
.end:

CS_KERNEL       equ .cs_kernel  - GDT
DS_KERNEL       equ .ds_kernel  - GDT
SS_LDT_0        equ .ldt_0      - GDT
SS_LDT_1        equ .ldt_1      - GDT
SS_TSS_0        equ .tss_0      - GDT
SS_TSS_1        equ .tss_1      - GDT
SS_GATE_0       equ .call_gate  - GDT
```

正：

prog/src/34_call_gate/descriptor.s

```
GDT:            dq  0x0000000000000000          ; NULL
.cs_kernel:     dq  0x00CF9A000000FFFF          ; コードセグメント
.ds_kernel:     dq  0x00CF92000000FFFF          ; データセグメント
.ldt            dq  0x0000820000000000          ; LDTディスクリプタ
.tss_0:         dq  0x0000890000000067          ; TSS_0ディスクリプタ
.tss_1:         dq  0x0000890000000067          ; TSS_1ディスクリプタ
.call_gate:     dq  0x0000EC0400080000          ; 386コールゲート(DPL=3, count=4, SEL=8)
.end:

CS_KERNEL       equ .cs_kernel  - GDT
DS_KERNEL       equ .ds_kernel  - GDT
SS_LDT          equ .ldt        - GDT
SS_TSS_0        equ .tss_0      - GDT
SS_TSS_1        equ .tss_1      - GDT
SS_GATE_0       equ .call_gate  - GDT
```

P.593　上から１番目のコードの直後（囲み内を薄い文字で追記）

同様の修正は、線の描画関数（draw_line）にも行います。線の描画関数は点の描画関数を繰り返し呼び出しているので、ループカウンタとして使用している ECX レジスタの値を変更しないように注意する必要があります。

prog/src/modules/protect/draw_line.s

```
draw_line:
        ...
%ifdef  USE_SYSTEM_CALL
        mov     eax, ecx                        ;   // 繰り返し回数を保存

        mov     ebx, [ebp +24]                  ;   EBX = 表示色;
        mov     ecx, [ebp - 8]                  ;   ECX = X座標;
        mov     edx, [ebp -20]                  ;   EDX = Y座標;
        int     0x82                            ;   sys_call(1, X, Y, 色, 文字); BX(C), CX(X),
DX(Y)

        mov     ecx, eax
%else
        cdecl   draw_pixel, dword [ebp - 8], ¥
                            dword [ebp -20], ¥
                            dword [ebp +24]     ;   // 点の描画
%endif
```

P.604　下から１番目のコード（囲み内を薄い字で追記）

誤：

prog/src/37_fpu/tasks/task_2.s

```
task_2:
        ...
                                ; ---------+---------+---------|---------|---------|---------|
                                ;       ST0|      ST1|      ST2|      ST3|      ST4|      ST5|
                                ; ---------+---------+---------|---------|---------|---------|
    fild    dword [.c1000]      ;     1000 |xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|
                                ; ---------+---------+---------|---------|---------|---------|
        ...
.c1000:     dd  1000
```

正：

prog/src/37_fpu/tasks/task_2.s

```
task_2:
        ...
                                ; ---------+---------+---------|---------|---------|---------|
                                ;       ST0|      ST1|      ST2|      ST3|      ST4|      ST5|
                                ; ---------+---------+---------|---------|---------|---------|
    fild    dword [.c1000]      ;     1000 |xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|
                                ; ---------+---------+---------|---------|---------|---------|
        ...
    ALIGN 4, db 0
.c1000:     dd  1000
```

P.605　上から２番目のコード（囲み内を薄い字で追記）

誤：

prog/src/37_fpu/tasks/task_2.s

```
task_2:
        ...
                                ; ---------+---------+---------|---------|---------|---------|
                                ;       ST0|      ST1|      ST2|      ST3|      ST4|      ST5|
                                ; ---------+---------+---------|---------|---------|---------|
    fild    dword [.c1000]      ;     1000 |xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|
    fldpi                       ;       pi |    1000 |xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|
    fidiv   dword [.c180]       ;   pi/180 |    1000 |xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|
                                ; ---------+---------+---------|---------|---------|---------|
        ...
.c180:      dd  180
```

正：

prog/src/37_fpu/tasks/task_2.s

```
task_2:
        ...
                                ; ---------+---------+---------|---------|---------|---------|
                                ;       ST0|      ST1|      ST2|      ST3|      ST4|      ST5|
                                ; ---------+---------+---------|---------|---------|---------|
    fild    dword [.c1000]      ;     1000 |xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|
    fldpi                       ;       pi |    1000 |xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|
    fidiv   dword [.c180]       ;   pi/180 |    1000 |xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|xxxxxxxxx|
                                ; ---------+---------+---------|---------|---------|---------|
        ...
ALIGN 4, db 0
.c1000:     dd  1000
.c180:      dd  180
```

P.608　上から２番目のコード（囲み内を薄い字で追記）

誤：

prog/src/37_fpu/tasks/task_2.s

```
task_2:
        ...
        mov     [.s2 + 0], bh               ;   // 1桁目
        mov     [.s3 + 0], ah               ;   // 小数1桁目
        mov     [.s3 + 1], bl               ;   // 小数2桁目
        mov     [.s3 + 2], al               ;   // 小数3桁目


        ...
.s0     db  "Task-2", 0
.s1:    db  "-"
.s2:    db  "0."
.s3:    db  "000", 0
```

正：

prog/src/37_fpu/tasks/task_2.s

```
task_2:
        cdecl   draw_str, 63, 1, 0x07, .s0      ; draw_str(.s0);
        ...
        mov     [.s2 + 0], bh                   ;   // 1桁目
        mov     [.s3 + 0], ah                   ;   // 小数1桁目
        mov     [.s3 + 1], bl                   ;   // 小数2桁目
        mov     [.s3 + 2], al                   ;   // 小数3桁目

        ...
.s0     db  "Task-2", 0
.s1:    db  "-"
.s2:    db  "0."
.s3:    db  "000", 0
```

P. 620　1 番目の表

誤：

fpu_rose_update(t, px, py);

戻り値	なし
t	角度
px	計算した X 座標を格納するアドレス
py	計算した Y 座標を格納するアドレス

正：

fpu_rose_update(px, py, t);

戻り値	なし
px	計算した X 座標を格納するアドレス
py	計算した Y 座標を格納するアドレス
t	角度

P.638　下から 1 番目のコード（囲み内を薄い字で追記）

誤：

prog/src/40_paging/kernel.s

```
        mov     eax, cr0                        ; // PGビットをセット
        or      eax, (1 << 31)                  ; CR0 |= PG;
        mov     cr0, eax                        ;
        jmp     $ + 2                           ; FLUSH();
```

正：

prog/src/40_paging/kernel.s

```
        mov     eax, CR3_BASE                   ;
        mov     cr3, eax                        ; // ページテーブルの登録

        mov     eax, cr0                        ; // PGビットをセット
        or      eax, (1 << 31)                  ; CR0 |= PG;
        mov     cr0, eax                        ;
        jmp     $ + 2                           ; FLUSH();

        ;-----------------------------------------
        ; CPUの割り込み許可
        ;-----------------------------------------
        sti                                     ; // 割り込み許可
```

P.646　上から１番目のコードの直後に追記

各タスクが使用するセグメントディスクリプタを LDT に登録し、データ用セグメントセレクタを定義します。

prog/src/42_rose_multi/descriptor.s

```
LDT:            dq  0x0000000000000000          ; NULL
～～～～
.cs_task_3:     dq  0x00CFFA000000FFFF          ; CODE 4G
.ds_task_3:     dq  0x00CFF2000000FFFF          ; DATA 4G
.ds_task_4:     dq  0x00CFF2000000FFFF          ; DATA 4G
.ds_task_5:     dq  0x00CFF2000000FFFF          ; DATA 4G
.ds_task_6:     dq  0x00CFF2000000FFFF          ; DATA 4G
.end:

～～～～
CS_TASK_3       equ (.cs_task_3 - LDT) | 4 | 3  ; タスク3用CSセレクタ
DS_TASK_3       equ (.ds_task_3 - LDT) | 4 | 3  ; タスク3用DSセレクタ
DS_TASK_4       equ (.ds_task_4 - LDT) | 4 | 3  ; タスク4用DSセレクタ
DS_TASK_5       equ (.ds_task_5 - LDT) | 4 | 3  ; タスク5用DSセレクタ
DS_TASK_6       equ (.ds_task_6 - LDT) | 4 | 3  ; タスク6用DSセレクタ
```

P.646　上から２番目のコード（囲み内は薄い字で修正）

誤：

prog/src/42_rose_multi/descriptor.s

```
GDT:            dq  0x0000000000000000          ; NULL
～～～～
.ldt_3:         dq  0x0000820000000000          ; LDT_3ディスクリプタ
.tss_3:         dq  0x0000890000000067          ; TSS_3ディスクリプタ
.tss_4:         dq  0x0000890000000067          ; TSS_4ディスクリプタ
.tss_5:         dq  0x0000890000000067          ; TSS_5ディスクリプタ
.tss_6:         dq  0x0000890000000067          ; TSS_6ディスクリプタ
.call_gate:     dq  0x0000EC0400080000          ; 386コールゲート(DPL=3, count=4, SEL=8)
.end:

～～～～
SS_LDT_3        equ .ldt_3      - GDT
SS_TSS_3        equ .tss_3      - GDT
SS_TSS_4        equ .tss_4      - GDT
SS_TSS_5        equ .tss_5      - GDT
SS_TSS_6        equ .tss_6      - GDT
SS_GATE_0       equ .call_gate  - GDT
```

正：

prog/src/42_rose_multi/descriptor.s

```
GDT:            dq  0x0000000000000000          ; NULL
～～～～
.tss_3:         dq  0x0000890000000067          ; TSS_3ディスクリプタ
.tss_4:         dq  0x0000890000000067          ; TSS_4ディスクリプタ
.tss_5:         dq  0x0000890000000067          ; TSS_5ディスクリプタ
.tss_6:         dq  0x0000890000000067          ; TSS_6ディスクリプタ
.call_gate:     dq  0x0000EC0400080000          ; 386コールゲート(DPL=3, count=4, SEL=8)
.end:

～～～～
SS_TASK_3       equ .tss_3      - GDT
SS_TASK_4       equ .tss_4      - GDT
SS_TASK_5       equ .tss_5      - GDT
SS_TASK_6       equ .tss_6      - GDT
SS_GATE_0       equ .call_gate  - GDT
```

P.649　下から１番目のコード（囲み内は薄い字で修正）

誤：

prog/src/42_rose_multi/modules/int_timer.s

```
        ;-------------------------------------
        ; タスクの切り替え
        ;-------------------------------------
        str     ax                          ; AX = TR; // 現在のタスクレジスタ
        cmp     ax, SS_TSS_0                ; case (AX)
        je      .11L                        ; {
        cmp     ax, SS_TSS_1                ;
        je      .12L                        ;
        cmp     ax, SS_TSS_2                ;
        je      .13L                        ;
        cmp     ax, SS_TSS_3                ;
        je      .14L                        ;
        cmp     ax, SS_TSS_4                ;
        je      .15L                        ;
        cmp     ax, SS_TSS_5                ;
        je      .16L                        ;
                                            ;   default:
        jmp     SS_TSS_0:0                  ;     // タスク0に切り替え
        jmp     .10E                        ;     break;
.11L:                                       ;   case SS_TASK_0:
        jmp     SS_TSS_1:0                  ;     // タスク2に切り替え
        jmp     .10E                        ;     break;
.12L:                                       ;   case SS_TASK_1:
        jmp     SS_TSS_2:0                  ;     // タスク2に切り替え
        jmp     .10E                        ;     break;
.13L:                                       ;   case SS_TASK_2:
        jmp     SS_TSS_3:0                  ;     // タスク3に切り替え
        jmp     .10E                        ;     break;
.14L:                                       ;   case SS_TASK_3:
        jmp     SS_TSS_4:0                  ;     // タスク4に切り替え
        jmp     .10E                        ;     break;
.15L:                                       ;   case SS_TASK_4:
        jmp     SS_TSS_5:0                  ;     // タスク5に切り替え
        jmp     .10E                        ;     break;
.16L:                                       ;   case SS_TASK_5:
        jmp     SS_TSS_6:0                  ;     // タスク6に切り替え
        jmp     .10E                        ;     break;
.10E:                                       ; }
```

正：

prog/src/42_rose_multi/modules/int_timer.s

```
        ;-------------------------------------
        ; タスクの切り替え
        ;-------------------------------------
        str     ax                      ; AX = TR; // 現在のタスクレジスタ
        cmp     ax, SS_TASK_0           ; case (AX)
        je      .11L                    ; {
        cmp     ax, SS_TASK_1           ;
        je      .12L                    ;
        cmp     ax, SS_TASK_2           ;
        je      .13L                    ;
        cmp     ax, SS_TASK_3           ;
        je      .14L                    ;
        cmp     ax, SS_TASK_4           ;
        je      .15L                    ;
        cmp     ax, SS_TASK_5           ;
        je      .16L                    ;
                                        ;   default:
        jmp     SS_TASK_0:0             ;     // タスク0に切り替え
        jmp     .10E                    ;     break;
.11L:                                   ;   case SS_TASK_0:
        jmp     SS_TASK_1:0             ;     // タスク2に切り替え
        jmp     .10E                    ;     break;
.12L:                                   ;   case SS_TASK_1:
        jmp     SS_TASK_2:0             ;     // タスク2に切り替え
        jmp     .10E                    ;     break;
.13L:                                   ;   case SS_TASK_2:
        jmp     SS_TASK_3:0             ;     // タスク3に切り替え
        jmp     .10E                    ;     break;
.14L:                                   ;   case SS_TASK_3:
        jmp     SS_TASK_4:0             ;     // タスク4に切り替え
        jmp     .10E                    ;     break;
.15L:                                   ;   case SS_TASK_4:
        jmp     SS_TASK_5:0             ;     // タスク5に切り替え
        jmp     .10E                    ;     break;
.16L:                                   ;   case SS_TASK_5:
        jmp     SS_TASK_6:0             ;     // タスク6に切り替え
        jmp     .10E                    ;     break;
.10E:                                   ; }
```

P.658　下から１番目のコードの直後に追記

ここで使用したファイル属性は、次のように定義しています。

prog/src/include/define.s

```
        ATTR_ARCHIVE        equ     0x20
        ATTR_VOLUME_ID      equ     0x08
```

P.659 上から１番目のコードの直後に追記

作成したファイルはカーネルの末尾にインクルードして使います。

prog/src/43_fat/kernel.s

```
;*************************************************************************
;   パディング
;*************************************************************************
        times KERNEL_SIZE - ($ - $$) db 0x00    ; パディング

;*************************************************************************
;   FAT
;*************************************************************************
%include    "fat.s"
```

FAT はカーネルの直後に配置されるので、1 番目の FAT 領域の開始位置はカーネルサイズと同じ値となります。2 番目の FAT 領域は BPB の「FAT 領域のセクタ数(0x16)」に 256、「セクタのバイト数(0x0B)」に 512 を設定したので、それよりも 0x02_0000（256 セクタ×512）バイト後方に配置します。

prog/src/include/define.s

```
        FAT_SIZE            equ     (1024 * 128)    ; FAT-1/2
        ROOT_SIZE           equ     (1024 *  16)    ; ルートディレクトリ領域

        FAT1_START          equ     (KERNEL_SIZE)
        FAT2_START          equ     (FAT1_START + FAT_SIZE)
        ROOT_START          equ     (FAT2_START + FAT_SIZE)
        FILE_START          equ     (ROOT_START + ROOT_SIZE)
```

P.667 ３番目のコード

誤：

prog/src/44_to_real_mode/boot.s

```
TO_REAL_MODE:
        ;---------------------------------------
        ; 【スタックフレームの構築】
        ;---------------------------------------
                                            ; ------|--------
                                            ; EBP+ 8| row（列）
                                            ; EBP+12| col（行）
                                            ; EBP+16| color（色）
                                            ; EBP+20| *p（文字列へのアドレス）
                                            ; ---------------
        push    ebp                         ; EBP+ 4| EIP（戻り番地）
        mov     ebp, esp                    ; EBP+ 0| EBP（元の値）
                                            ; ---------------
```

正：

prog/src/44_to_real_mode/boot.s

```
TO_REAL_MODE:
        ;--------------------------------------
        ; 【スタックフレームの構築】
        ;--------------------------------------
                                            ; ------|--------
                                            ; EBP+ 8| col（列）
                                            ; EBP+12| row（行）
                                            ; EBP+16| color（色）
                                            ; EBP+20| *p（文字列へのアドレス）
                                            ; ---------------
        push    ebp                         ; EBP+ 4| EIP（戻り番地）
        mov     ebp, esp                    ; EBP+ 0| EBP（元の値）
                                            ; ---------------
```

P.680　下から２番目のコードのファイル名

誤：

prog/src/45_fat_bios/kernel.s

正：

prog/src/46_acpi/kernel.s

P.680　下から１番目のコードのファイル名

誤：

prog/src/45_fat_bios/boot.s

正：

prog/src/46_acpi/boot.s

P.683　上から 2 番目のコード

誤：

prog/src/modules/protect/power_off.s

```
power_off:
        ...
        ;-------------------------------------
        ; パッケージデータの取得
        ;-------------------------------------
        add     eax, 4                          ;   EAX  = 先頭の要素;
        cdecl   acpi_Package_value, eax         ;   EAX = パッケージデータ;
        mov     [S5_PACKAGE], eax               ;   S5_PACKAGE = EAX;
        ...

ALIGN 4, db 0
PM1a_CNT_BLK:   dd  0
PM1b_CNT_BLK:   dd  0
S5_PACKAGE:
.0:             db  0
.1:             db  0
.2:             db  0
.3:             db  0
```

正：

prog/src/modules/protect/power_off.s

```
power_off:
        ...
        ;-------------------------------------
        ; パッケージデータの取得
        ;-------------------------------------
        add     eax, 4                          ;   EAX  = 先頭の要素;
        cdecl   acpi_package_value, eax         ;   EAX = パッケージデータ;
        mov     [S5_PACKAGE], eax               ;   S5_PACKAGE = EAX;
        ...

ALIGN 4, db 0
PM1a_CNT_BLK:   dd  0
PM1b_CNT_BLK:   dd  0
S5_PACKAGE:
.0:             db  0
.1:             db  0
.2:             db  0
.3:             db  0
```

P.710　コマンドプロンプト（囲みで示した部分）

誤：

コマンドプロンプト

```
<bochs:50> x /8bx 0x7c00
[bochs]:
0x0000000000007c00 <bogus+         0>:      0xeb      0x3c      0x90      0x4f      0x450x4d
0x2d     0x4e
<bochs:51> x /8wx 0x7c00
[bochs]:
0x0000000000007c00 <bogus+         0>:      0x4f903ceb        0x4e2d4d45        0x00454d41
0x00200102
0x0000000000007c10 <bogus+        16>:      0xf0020002        0x0100f8ff        0x00020010
0x00000000
<bochs:52> x /8wd 0x7c00
[bochs]:
0x0000000000007c00 <bogus+         0>:     1334852843       1311591749       45417612097410
0x0000000000007c10 <bogus+        16>:     -268304382       16840959         1310880
<bochs:53> x /32bm 0x7c00
[bochs]:
0x0000000000007c00:EB 3C 90 4F 45 4D 2D 4E  41 4D 45 00 02 01 20 00
0x0000000000007c10:02 00 02 F0 FF F8 00 01  10 00 02 00 00 00 00 00
```

正：

コマンドプロンプト

```
<bochs:50> x /8bx 0x7c00
[bochs]:
0x0000000000007c00 <bogus+         0>:      0xeb      0x3c      0x90      0x4f      0x45
0x4d     0x2d     0x4e
<bochs:51> x /8wx 0x7c00
[bochs]:
0x0000000000007c00 <bogus+         0>:      0x4f903ceb        0x4e2d4d45        0x00454d41
0x00200102
0x0000000000007c10 <bogus+        16>:      0xf0020002        0x0100f8ff        0x00020010
0x00000000
<bochs:52> x /8wd 0x7c00
[bochs]:
0x0000000000007c00 <bogus+         0>:     1334852843       1311591749       45417612097410
0x0000000000007c10 <bogus+        16>:     -268304382       16840959         1310880
<bochs:53> x /32bm 0x7c00
[bochs]:
0x0000000000007c00:EB 3C 90 4F 45 4D 2D 4E  41 4D 45 00 02 01 20 00
0x0000000000007c10:02 00 02 F0 FF F8 00 01  10 00 02 00 00 00 00 00
```