APPENDIX E

The libepc Library

The functions provided in libepc are intended to make it easy to start writing embedded applications. The library provides processor and PC hardware initialization (including the Interrupt Descriptor Table), console I/O, timer access, sound, heap management, and more.

Unlike the functions in libc, those in libepc never use the ROM BIOS or expect an operating system. For example, something as simple as the putchar library function of libc requires support from an operating system like MS/DOS or MS/Windows, which in turn relies on the ROM BIOS. Since neither is likely to be available in most embedded systems, the display output routines found in libepc write directly to the display buffer.

In some cases, functions in libepc duplicate services found in the standard C run-time library (libc). The corresponding libc fountains assume a desktop (rather than embedded) application environment, and reference objects in other libc modules that cause the linker to include a large amount of unnecessary or inappropriate code. The alternative implementations of those functions provided by libepc are self contained and eliminate this problem.[1]

The following data types are defined in libepc.h and appear in the function descriptions given in this appendix:

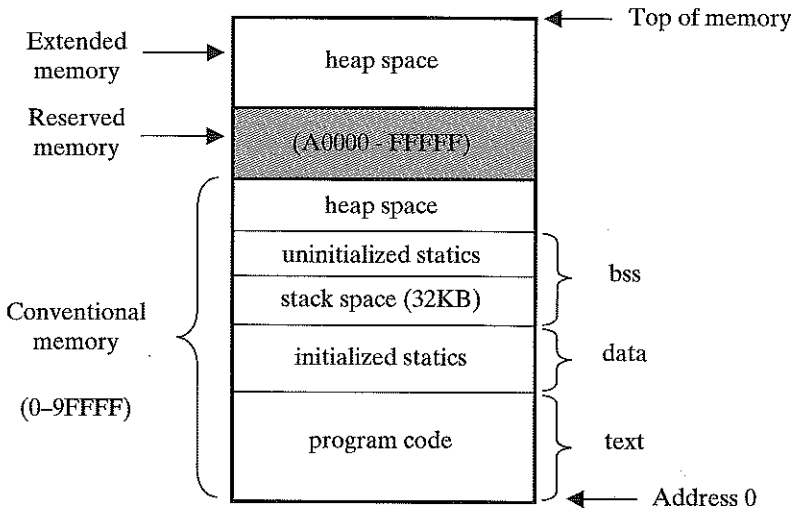
```
typedef int                         BOOL ;
typedef unsigned char               BYTE8 ;
typedef unsigned short int          WORD16 ;
typedef unsigned long int           DWORD32 ;
typedef unsigned long long int      QWORD64 ;
typedef signed long int             FIXED32 ;
typedef signed long long int        FIXED64 ;
typedef void                        (*ISR)(void) ;
```

Source code for all functions in libepc is provided on the CD that comes with this text.

MEMORY LAYOUT AND INITIALIZATION

The IBM-PC partitions the address space into three regions: "conventional memory" (0–640KB), "reserved" memory (640KB to 1MB), and "extended" memory (above 1MB).

[1] Note that to cause the linker to use the libepc versions instead of those in libc, you must list libepc.a on the linker command line *before* libc.a.



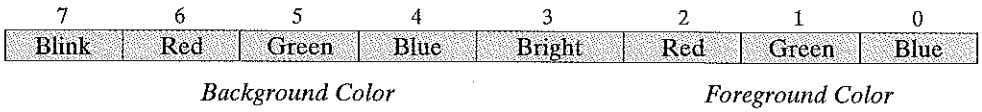
Conventional memory is subdivided into three major areas: the code space (known as 'text'), initialized 'data,' and uninitialized data (called 'bss'). The bss contains all uninitialized static objects and a program stack of 32KB. Any remaining conventional memory and all of extended memory are used for the heap.

Execution of the embedded application begins at address zero. Interrupts remain disabled throughout the initialization process, which ends with a call to function `main`. Initialization performs the following tasks:

1. Puts the processor into protected mode, establishes a flat memory model with all segments starting at address zero, and sets all segment sizes to 4GB. (`init-cpu.asm`)
2. Sets all uninitialized statics within the bss to zeroes, and (optionally) copies the contents of any ROM data into RAM. (`init-crt.c`)
3. Initializes the 8259 Programmable Interrupt Controller (`init8259.c`)
4. Initializes the 8253 Programmable Timer to provide DRAM refresh and a 100-tick-per-second interrupt. (`init8253.c`)
5. Creates and initializes an Interrupt Descriptor Table (`init-idt.c`)

DISPLAY FUNCTIONS (DISPLAY.C)

The contents of the display is a two-dimensional array of cells organized into rows and columns. Rows are numbered from 0 (top of screen) to 24 (bottom of screen); columns are numbered from 0 (left side of screen) to 79 (right side of screen). Each cell contains two bytes; the first holds the ASCII representation of the displayed character, and the second holds the display attributes for that character cell:



The display functions in libepc write directly to the display buffer. These functions make no call to the ROM BIOS, nor do they rely on the existence of an operating system.

Prototype: `WORD16 *Cell(int row, int col) ;`
Description: Returns a pointer to a position in the display buffer specified by 'row' and 'col'.
Interrupts: Unaffected.

Prototype: `void ClearScreen(BYTE8 attb) ;`
Description: Erases the display and sets the attribute bytes of every character position to the value specified by 'attb'. Does not affect the current cursor position.
Interrupts: Unaffected.

Prototype: `int GetCursorCol(void) ;`
Description: Returns the current column position of the cursor.
Interrupts: Unaffected.

Prototype: `int GetCursorRow(void) ;`
Description: Returns the current row position of the cursor.
Interrupts: Unaffected.

Prototype: `void PutAttb(BYTE8 attb, int cells) ;`
Description: Changes the attribute bytes of a number of character cells starting at the current cursor position. Does not affect the current cursor position. The new attribute value is specified by 'attb', and the number of cells is specified by 'cells'.
Interrupts: Unaffected.

Prototype: `void PutChar(char ch) ;`
Description: Displays a single ASCII character (ch) at the current cursor position. Automatically advances the current cursor position and properly interprets CR and LF. Display attributes of the text are not affected by this function (see function `PutAttb`).
Interrupts: Unaffected.

Prototype: `void PutCharAt(char ch, int row, int col) ;`
Description: Displays a single ASCII character (ch) at a specific display position (row, col) using the current display attribute previously stored at that position. Does not affect the current cursor position.
Interrupts: Unaffected.

Prototype: `void PutString(char *string) ;`
Description: Displays a NUL-terminated string of characters starting at the current cursor position. Automatically advances the current cursor position and properly interprets CR and LF. Display attributes of the text are not affected by this function (see function `PutAttb`).
Interrupts: Unaffected.

Prototype: `void PutUnsigned(unsigned n, int b, int w) ;`
Description: Displays the unsigned value 'n' as a base 'b' number right justified and zero filled in a field of at least 'w' characters at the current cursor position.
Interrupts: Unaffected.

Prototype: `void SetCursorPosition(int row, int col) ;`
Description: Sets the current row and column position of the cursor.
Interrupts: Disabled on entry; restored to previous state on return.

Prototype: `void SetCursorVisible(BOOL visible) ;`
Description: Controls visibility of the IBM-PC cursor. If the argument is TRUE (non-zero), the cursor will be made visible; if the argument is FALSE (zero), the cursor is made invisible.
Interrupts: Disabled on entry; restored to previous state on return.

WINDOW FUNCTIONS (WINDOW.C)

The following functions may be used to manage multiple text windows in separate areas of the physical display. Each window maintains its own logical cursor. Overlapping windows are not supported.

Prototype: `WINDOW *WindowCreate(char *title, int row_first, int row_last, int col_first, int col_last) ;`
Description: Paints an empty window on the display. Creates a window control block and returns a pointer to it; this pointer must be used in any other window function. Multiple windows may be defined. If "title" is NULL, no border is drawn, and the writeable area is given by the other parameters; otherwise, a titled border is drawn with the other parameters defining the position of the border.
Interrupts: Unaffected.

Prototype: `void WindowErase(WINDOW *w) ;`
Description: Erases the contents of the specified window.
Interrupts: Unaffected.

Prototype: `void WindowSelect(WINDOW *w) ;`
Description: Positions the display's physical cursor at the window's logical cursor position.
Interrupts: Unaffected.

Prototype: `void WindowSetCursor(WINDOW *w, int row, int col) ;`
Description: Positions the window's logical cursor; does not affect the physical cursor of the display.
Interrupts: Unaffected.

Prototype: `void WindowPutChar(WINDOW *w, char ch) ;`
Description: Displays a single character at the window's logical cursor position and advances the cursor.
Interrupts: Unaffected.

Prototype: `void WindowPutString(WINDOW *w, char *str) ;`
Description: Displays a character string at the window's logical cursor position and advances the cursor.
Interrupts: Unaffected.

KEYBOARD FUNCTIONS (KEYBOARD.C)

Every key on the IBM-PC keyboard issues two interrupts: one when the key is pressed, and a second when the key is released. Each action is represented by a "scan code" that is read from the keyboard data buffer by an interrupt service routine (ISR) and is placed in a queue. The two scan codes for the press and release of the same key differ only by the value of bit 7; bit 7 is clear (0) on a press, and set (1) on a release.

The keyboard functions in libepc access the keyboard hardware directly. These functions make no call to the ROM BIOS, nor do they rely on the existence of an operating system.

Prototype: `BYTE8 GetScanCode(void) ;`
Description: Returns scan code from the queue.
Interrupts: Disabled on entry; restored to previous state on return.

Prototype: `BOOL ScanCodeRdy(void) ;`
Description: Returns TRUE (1) if a scan code is available in the queue; returns FALSE (0) otherwise.
Interrupts: Disabled on entry; restored to previous state on return.

Prototype: `WORD16 ScanCode2Ascii(BYTE8 code) ;`
Description: Converts a scan code into a byte pair, with the LSByte containing the ASCII representation of the key and the MSByte containing the scan code. Keeps track of keyboard shift states (Shift, CapsLock, Ctrl, Alt, and NumLock).
Interrupts: Unaffected.

Prototype: `BOOL SetsKybdState(BYTE8 code) ;`
Description: Uses a scan code to update keyboard shift states (Shift, CapsLock, Ctrl, Alt, and NumLock). Returns TRUE (1) if the scan code affects the shift states; otherwise, has no effect and returns FALSE (0).
Interrupts: Unaffected.

SPEAKER FUNCTIONS (SPEAKER.C)

Prototype: `void Sound(int hertz) ;`
Description: Produces tones on the PC speaker. If 'hertz' is greater than zero, it turns on the speaker and starts generating a tone whose frequency is specified by 'hertz'; a zero or negative value turns off the speaker.
Interrupts: Disabled while programming timer port to produce the desired frequency; interrupts are restored to their previous state before the function returns.

TIMER FUNCTIONS (TIMER.C, CYCLES.ASM)

Prototype: `QWORD64 CPU_Clock_Cycles(void) ;`
Description: Returns a 64-bit count of CPU clock cycles since the processor was reset. This function uses the RDTSC (Read Time-Stamp Counter) instruction of the Intel Pentium. It will *not* work on a 486 or 386, and may also not work on any nonIntel processor.
Interrupts: Unaffected.

Prototype: `DWORD32 Milliseconds(void) ;`
Description: Returns the 32-bit unsigned count of milliseconds that have elapsed since program execution began.
Interrupts: Enables interrupts.

Prototype: `DWORD32 Now_Plus(int seconds) ;`
Description: Calculates and returns the 32-bit unsigned count that will be in the system timer some number of seconds in the future, specified by 'seconds'. The timer runs at 1000 ticks per second.
Interrupts: Enables interrupts.

INTERRUPT VECTOR ACCESS FUNCTIONS (INIT-IDT.C)

An Interrupt Descriptor Table (IDT) is built during initialization and is filled with links to a set of default Interrupt Service Routines (ISRs) for interrupt vectors 0–255. The functions described in the following table provide support for installing custom replacements for these ISRs (note that interrupt vectors 0–31 are reserved by Intel for processor traps and exceptions; their default ISRs simply display appropriate error messages and halt):

INT #	Intel Usage	INT #	Intel Usage
0	Divide by zero	11	Segment not present
1	Debug exception	12	Stack fault
2	NMI	13	General protection
3	One byte interrupt	14	Page fault
4	Interrupt on overflow	15	Intel Reserved
5	Array bounds error	16	Math error
6	Invalid opcode	17	Alignment check
7	Math not available	18	Machine check
8	Double fault	19	SIMD floating-point except
9	Math segment overflow	20–31	Intel reserved
10	Invalid TSS		

Another 16 vectors are used for the IBM-PC hardware interrupt request (IRQ) lines; their default ISRs simply clear the interrupt request and return. The initialization code initialized the 8259 Programmable Interrupt Controller so that it assigns these hardware IRQ lines to the last 16 interrupt vectors in the table (240–255):

IRQ #	INT #	IBM-PC Usage	IRQ #	INT #	IBM-PC Usage
0	240	System timer	8	248	Real-time clock
1	241	Keyboard	9	249	Redirected IRQ2
2	242	Redirected	10	250	Reserved
3	243	COM2/COM4	11	251	Reserved
4	244	COM1/COM3	12	252	PS/2 Mouse
5	245	Reserved	13	253	Math coprocessor
6	246	Floppy disk	14	254	Hard disk
7	247	Parallel port	15	255	Reserved

The remaining interrupt vectors initially point to an ISR that merely displays "Unassigned Interrupt" and halts.

Prototype: `ISR GetISR(int int_numb) ;`
Description: Returns the entry-point address of an ISR from the Interrupt Descriptor Table (IDT). The interrupt vector number specified by the 'int_numb' selects the IDT entry that provides the address.
Interrupts: Unaffected.

Prototype: `int IRQ2INT(int irq_numb) ;`
Description: Returns the interrupt vector number that is assigned to the corresponding hardware IRQ line (0-15) specified by 'irq_numb'.
Interrupts: Unaffected.
Note: Use of this function is preferred to hard-coding a constant for the interrupt vector number of a hardware IRQ line; using this function protects you against a possible reassignment of the hardware vectors as the result of some future modification of the file INIT–IDT.C.

Prototype: `void SetISR(int int_numb, ISR isr) ;`
Description: Stores a pointer to an ISR into the Interrupt Descriptor Table (IDT). The interrupt vector number is specified by 'int_numb', and a pointer to the ISR is given by 'isr'.
Interrupts: Unaffected.

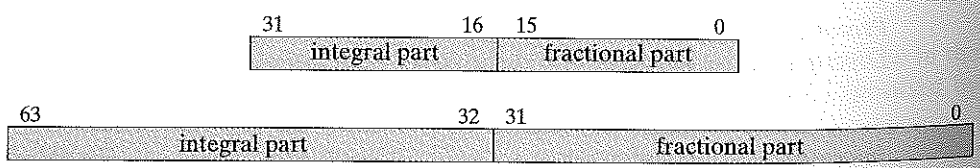
DYAMIC MEMORY ALLOCATION FUNCTIONS (HEAP.C)

Prototype: `void *malloc(unsigned long int bytes) ;`
Description: Identical to the traditional C library function of the same name, except that this implementation is self contained; that is, it makes no reference to objects or functions defined in other modules.
Interrupts: Temporarily disabled during certain critical sections within the function; original state of interrupt system is restored at end of critical sections.

Prototype: `void free(void *block) ;`
Description: Identical to the traditional C library function of the same name, except that this implementation is self contained; that is, it makes no reference to objects or functions defined in other modules.
Interrupts: Temporarily disabled during certain critical sections within the function; original state of interrupt system is restored at end of critical sections.

FIXED POINT (FIXEDPT.ASM)

These functions provide limited support for fixed-point math using 32-bit and 64-bit operands, with an equal number of bits allocated to the integral and fractional parts:



The smaller 16.16 format provides a significant speed improvement over that of the 32.32 format, but the latter offers much greater range and precision.

Prototype: `FIXED32 Inverse32(FIXED32 denominator) ;`
Description: Returns the 32-bit inverse of a 32-bit fixed-point operand. No error checking is performed.
Interrupts: Unaffected.

Prototype:
```
FIXED32 Product32
(FIXED32 multiplier, FIXED32 multiplicand) ;
```
Description: Returns the 32-bit product of two 32-bit fixed-point operands. No error checking is performed.
Interrupts: Unaffected.

Prototype:
```
FIXED64 Product64
(FIXED64 multiplier, FIXED64 multiplicand) ;
```
Description: Returns the 64-bit product of two 64-bit fixed-point operands. No error checking is performed.
Interrupts: Unaffected.

Prototype:
```
FIXED32 Quotient32
(FIXED32 dividend, FIXED32 divisor) ;
```
Description: Returns the 32-bit quotient of two 32-bit fixed-point operands. No error checking is performed.
Interrupts: Unaffected.

Prototype: `FIXED32 Sqrt32(FIXED32 radical) ;`
Description: Returns the 32-bit square root of a 32-bit fixed-point operand. No error checking is performed.
Interrupts: Unaffected.

INTERFUNCTION JUMPS (SETJMP.ASM)

Prototype: `int setjmp(jmp_buf) ;`
Description: Identical to the traditional C library function of the same name, except that this implementation is self contained; that is, it makes no reference to objects or functions defined in other modules.
Interrupts: Unaffected.

Prototype: `void longjmp(jmp_buf, int) ;`
Description: Identical to the traditional C library function of the same name, except that this implementation is self contained; that is, it makes no reference to objects or functions defined in other modules.
Interrupts: Unaffected.

MISCELLANEOUS FUNCTIONS (INIT-CRT.C)

Prototype: `void *LastMemoryAddress(void) ;`
Description: Returns a pointer to the last byte of physical memory.
Interrupts: Unaffected.