Hostess *186*™ 4
Hostess *186*™ 8

Programmer's Reference

Hostess 186 Programmer's Reference

First Edition, February 1992
Revised, April 1992

If you have any questions or comments about this manual or any COMTROL product, please send your inquiries to:

COMTROL CORPORATION
2675 Patton Road
P.O. Box 64750
St. Paul, Minnesota 55164
USA

COMTROL Europe Ltd.
The Courtyard Studio, Grange Farm
Station Road
Launton, Bicester
Oxfordshire, OX6-0EE England

Telephone: 1-800-926-6876, (612) 631-7654 (US), or (44) 869-323-220 (UK).
FAX (612) 631-8117 (US), (44) 869-323-211 (UK)

Trademarks

Hostess 186 Internal I/O Addresses

The table that follows shows the internal I/O addresses for the Hostess 186's devices.

Table 13. Hostess 186 internal I/O addresses.

Device	I/O Address:	Device	I/O Address:
Interrupt control register	0FF28h		
Timer 0 count register	FF50h	Timer 1 count register	FF58h
SCC port0 (Hostess 186 port1) command register	0004h	SCC port4 (Hostess 186 port5) command register	0104h
SCC port0 (Hostess 186 port1) data register	0006h	SCC port4 (Hostess 186 port5) data register	0106h
SCC port1 (Hostess 186 port2) command register	0000h	SCC port5 (Hostess 186 port6) command register	0100h
SCC port1 (Hostess 186 port2) data register	0002h	SCC port5 (Hostess 186 port6) data register	0102h
SCC port2 (Hostess 186 port3) command register	0084h	SCC port6 (Hostess 186 port7) command register	0184h
SCC port2 (Hostess 186 port3) data register	0086h	SCC port6 (Hostess 186 port7) data register	0186h
SCC port3 (Hostess 186 port4) command register	0080h	SCC port7 (Hostess 186 port8) command register	0180h
SCC port3 (Hostess 186 port4) data register	0082h	SCC port7 (Hostess 186 port8) data register	0182h
Modem status register	0200h		

CHAPTER 6 – Hostess 186 Dual-port Memory

Setting the System Computer's Dual-port Memory Addresses

Writing to the control registers sets the dual-port memory addresses in the system's address space. The controller reserves a 16, 32, 64, or 128 Kbyte block of addresses starting with the address set by the control registers. Addresses are in the range of 13 to 16 Mbytes (D00000 to FE0000h), and under 1 megabyte (080000 to 0F0000h). Table 14 shows the popular memory addresses found under one megabyte of memory.

Table 14. Under one megabyte memory addresses.

Controller Memory Address and Offset	Value for Control Register # 2 Bits 0 to 7 (SD7 to SD0)	Value for Control Register # 1 Bits 6 and 7 (SD7 to SD6)	Valid System Window Sizes (set with Control Register #1)
8000:0000	08h	00h	16K, 32K, 64K
8000:4000	08h	01h	16K
8000:8000	08h	02h	16K, 32K
8000:C000	08h	03h	16K
9000:0000	09h	00h	16K, 32K, 64K
9000:4000	09h	01h	16K
9000:8000	09h	02h	16K, 32K
9000:C000	09h	03h	16K
A000:0000	0Ah	00h	16K, 32K, 64K
A000:4000	0Ah	01h	16K
A000:8000	0Ah	02h	16K, 32K
A000:C000	0Ah	03h	16K
B000:0000	0Bh	00h	16K, 32K, 64K
B000:4000	0Bh	01h	16K
B000:8000	0Bh	02h	16K, 32K
B000:C000	0Bh	03h	16K
C000:0000	0Ch	00h	16K, 32K, 64K
C000:4000	0Ch	01h	16K
C000:8000	0Ch	02h	16K, 32K
C000:C000	0Ch	03h	16K
D000:0000	0Dh	00h	16K, 32K, 64K
D000:4000	0Dh	01h	16K
D000:8000	0Dh	02h	16K, 32K
D000:C000	0Dh	03h	16K
E000:0000	0Eh	00h	16K, 32K, 64K

Table 15. Above one megabyte memory addresses.

Address:	Value for Control Register #2, Data bits SD7 to SD0:	Value for Control Register #1, Data bits SD7 to SD6:
F00000h	0F0h	30h
F20000h	0F2h	30h
F40000h	0F4h	30h
F60000h	0F6h	30h
F80000h	0F8h	30h
FA0000h	0FAh	30h
FC0000h	0FCh	30h
FE0000h	0FEh	30h
E00000h	0E0h	30h
E20000h	0E2h	30h
E40000h	0E4h	30h
E60000h	0E6h	30h
E80000h	0E8h	30h
EA0000h	0EAh	30h
EC0000h	0ECh	30h
EE0000h	0EEh	30h
D00000h	0D0h	30h
D20000h	0D2h	30h
D40000h	0D4h	30h
D60000h	0D6h	30h
D80000h	0D8h	30h
DA0000h	0DAh	30h
DC0000h	0DCh	30h
DE0000h	0DEh	30h

Dual-port Memory Map

Of the 128 Kbytes of dual-port memory on the Hostess 186, more than 124 Kbytes are available for control programs. Table 16 shows how dual-port memory is mapped out.

Table 16. Hostess 186 memory map.

System Computer Memory Address:	Hostess 186 Memory Address:	Description:	Length:
Base + 10080h	10080h	unused	FF80h bytes
Base + 10000h	10000h	firmware data area	80h bytes
Base + C00h	00c00h	unused	F400h bytes
Base + 400h	00400h	firmware work space	800h bytes
Base* + 0	00000h	interrupt vector table	400h bytes

*Base memory address in the system computer's memory space, using a 128K window.

The lower 400h bytes are reserved for the interrupt vector table. The firmware uses from 400h to C00h for miscellaneous work space.

The 80h bytes from 10000h to 10080h are called the firmware data area. The firmware stores information about the controller in this area. The rest of the unused memory is available for control programs to use.

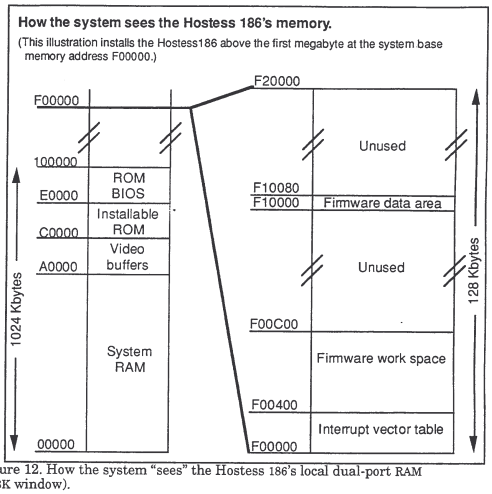


Figure 12. How the system "sees" the Hostess 186's local dual-port RAM (128K window).

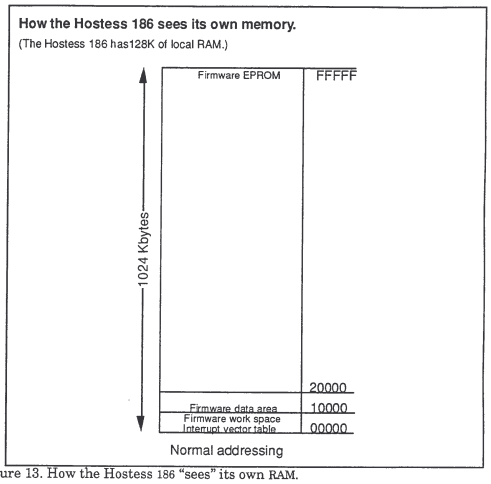


Figure 13. How the Hostess 186 "sees" its own RAM.

Firmware Data Area Map

The firmware data area located at controller memory address 10000h is 80h bytes long. As the firmware executes, the data area fills with the information listed in Table 17.

Table 17. Firmware data area map.

Offset	Description	Length
0h	interaction flag, 55AAh = controller active	2h bytes
2h	boot flag, 0000h = hard boot FFFFh = soft boot	2h bytes
4h	old configuration map, 0000h = not used now	2h bytes
6h	firmware release number	8h bytes
Eh	open for control program release number	8h bytes
16h	unused	4h bytes
1Ah	local RAM map, one bit set per 64K.	2h bytes
1Ch	unused	
1Eh	SCC port map, one bit set per port found	4h bytes
22h	identification number, 00000h = Hostess 186	4h bytes
26h	invalid interrupt field	4h bytes
2Ah	reserved for future use	56h bytes

At offset 0h, the "interaction flag" equals 55AAh when the controller is functioning properly.

At offset 2h, the "boot flag" equals 0000h when the system powers up and changes to FFFFh when the system reboots without powering down.

At offset 4h, the two-byte "old config map," is not used, but space is allocated so the firmware data area is compatible with the firmware data area of other COMTROL controllers.

At offset 6h, the "firmware release number" is an ASCII string.

At offset Eh, eight bytes are open for an ASCII string that identifies the control program release.

At offset 22h, the "identification number" for the Hostess 186 is 00000h.

At offset 26h, the "invalid interrupt field" marks any spurious interrupts that come into the interrupt controller. The firmware recovers from spurious interrupts, so the control program does not have to handle it.

The remaining 56h bytes in the firmware data area are reserved for future use.

CHAPTER 7 – Interrupts

Interrupting the System Processor

The Hostess 186 controller can use IRQ 3, 4, 5, 9, 10, 11, 12, or 15. A system write to the control register #4 sets the IRQ that the Hostess 186 controller uses to interrupt the system processor.

The Hostess 186's control program interrupts the system processor on the IRQ line by writing 0008h to the I/O address EF60h. After a two-microsecond delay, you must clear the interrupt by writing 0000h to address EF60h. (The delay is simple; execute three consecutive `jmp short $+2` statements.)

Here is an example of setting and clearing an interrupt to the system processor:

```
mov   dx,ef60h    ; dx = interrupt address
mov   ax,0008h    ; ax = value to set interrupt low
out   dx,ax       ; set the interrupt

jmp   short $+2   ; delay
jmp   short $+2   ; delay
jmp   short $+2   ; delay

mov   ax,0000h    ; ax = value to clear interrupt high
out   dx,ax       ; clear the interrupt
```

Note that multiple Hostess 186 controllers may share the same IRQ line. To share an IRQ, the interrupt service routine (ISR) on the system computer must include code that identifies which controller generates the interrupt.

Interrupting the Controller

By writing to the `I/O base+2` address, the system processor interrupts the controller on the 80186's interrupt 3 line. This generates an interrupt vector type 0Dh. The control program's interrupt service routine (ISR) must clear the interrupt after processing it. The interrupt is cleared by writing 8000h to the "interrupt control register" at address FF22h.

Here is an example of the device driver setting an interrupt to the controller:

```
outp(0x218+2,0);  /* write anything to io_base+2 */
```

Here is an example of the control program clearing the interrupt:

```
mov   dx,0ff22h   ; dx = interrupt control register
mov   ax, 08000h  ; ax = 08000h to clear the interrupt
out   dx,ax       ; clear the interrupt
iret              ; return from interrupt
```

NOTE: The firmware uses this interrupt to invoke the control program at address 1000:80. Change the interrupt vector table entry 0Dh before you use this interrupt.

Writing an Internal Interrupt Service Routine

The control program must have interrupt service routines (ISRs) for all interrupts it uses. The interrupt vector table stores the address of the interrupt service routine, so when the interrupt comes in, execution immediately jumps to the correct interrupt service routine.

The interrupt service routine processes the interrupt, clears the interrupt, and executes the `iret` instruction to return from the interrupt. The processing of the interrupt is specific to the control program. To clear the interrupt, write 08000h to the "interrupt control register" at address 0FF22h.

Do not disable and enable other interrupts with the `cli` and `sti` instructions while in an interrupt service routine. Doing so can let another interrupt come in before the current interrupt is cleared.

Here is an example of a timer interrupt service routine:

```
timer_isr   proc
      push  ax                ; save registers
      push  dx

      ; do internal processing

      mov   dx,0ff22h         ; dx = interrupt control register
      mov   ax, 08000h        ; ax = value to clear interrupt
      out   dx,ax             ; clear the interrupt

      pop   dx                ; restore registers
      pop   ax
      iret                    ; return from the interrupt routine
timer_isr   endp
```


Initializing Hostess 186 Interrupt Vectors

Each interrupt has a vector type number. The address of the interrupt service routine requires four bytes and is placed in the interrupt vector table at ("vector_type" * 4). The two-byte offset address is stored first in the interrupt vector table, followed by the two-byte segment address.

Here is an example of storing the SYSTEM interrupt service routine address in the interrupt vector table:

```
xor   ax,ax                   ; zero ax
mov   es,ax                   ; segment of vector table = 0
mov   bx,0dh*4                ; get vector table location
mov   ax,offset system_isr    ; offset of isr
mov   es:[bx],ax              ; store in vector table
mov   ax,cs                   ; segment of isr routine
mov   es:[bx+2],ax            ; store in vector table
```

Table 18 lists the interrupts the Hostess 186 can use, their vector types, interrupt vector table locations, the possibility that the control program can modify the vector, and whether hardware or software generates the interrupt.

Table 18. Hostess 186 interrupt vector types and locations.

Hostess 186 Interrupt:	Vector type number:	Vector table location:	Control program modifiable:	Hardware/ Software generated:	Comments:
NMI	2h	8h	no	H/W	
DEBUGGER	20h	80h	no	S/W	
RAM QUERY	21h	84h	no	S/W	
DEBUG PORT	22h	88h	no	S/W	
CONFIG QUERY	23h	8Ch	no	S/W	
TURBO_DEBUGGER REMOTE	27h	9Ch	no	S/W	
8530 bank 1	0Ch	*	no	H/W	
SYSTEM (I/O + 2 write)	0Dh	34h	yes	H/W	Generated by the system to the Hostess 186.
TIMER 0	8h	20h	yes	H/W	
TIMER 1	12h	48h	yes	H/W	
IRQ7 (Catches invalid interrupts)	37h	C8h	no	H/W	
SCC base	80h	200h	yes	H/W	

Hostess 186 Interrupt Vectors Defined

The firmware sets up twelve interrupt vectors, eight of which should not be changed and four that can be modified by the control program.

The external NMI (non-maskable interrupt, type 2h) occurs only on a "software development" controller; one equipped with the reset and debug switches. The debug switch triggers an NMI, which invokes the debugger.

The software DEBUGGER interrupt (type 20h) invokes the firmware debugger.

The software RAM_QUERY interrupt (type 21h) returns the first segment that is open for control program use in the AX register. This may be used to determine where to load the control program.

The software DEBUG_PORT interrupt (type 22h) changes the firmware's debugging port (the first serial port) to the one specified in the AL register.

The software CONFIG_QUERY interrupt (type 23h) returns information in the firmware data area about the number of ports and amount of dual-ported RAM on the controller. This depends on the value of the AL register on entry:

If the AL register = 0 on entry, the "old config map" is returned in the AX register.

If the AL register = 1 on entry, the "dual-ported RAM map" is returned. The low word is in the AX register and the high word is in the BX register.

If the AL register = 2 on entry, the "SCC port map" is returned.
The low word is in the AX register and the high word is in the BX register.

The TURBO_DEBUGGER_REMOTE interrupt (type 27h) is used to invoke the remote Borland® Turbo Debugger™ kernel on the Hostess 186 controller.

The 8530 interrupts (type 0Ch) are cascaded from the SCC interrupt. These interrupts should not be used nor modified. Furthermore, these interrupts do not use the vector table entries of the Hostess 186. For more information on SCC interrupt type, the section "Finding the SCC Interrupt Vector Types" in this chapter.

When the system processor writes to the "I/0 base+2" address to interrupt the controller, this generates the SYSTEM interrupt (type 0Dh). This vector should be replaced with the control program's vector to process system interrupts.

The TIMER 0 interrupt (type 08h) is the interrupt that timer 0 generates. This vector should be replaced with the control program's vector if the control program uses timer 1.

The TIMER 1 interrupt (type 12h) is the interrupt that timer 1 generates. This vector should be replaced with the control program's vector if the control program uses timer 1.

The IRQ7 interrupt (type 37h) is a catch-all interrupt that collects all invalid interrupts.

The SCC_base interrupts are placed every eight bytes (for every two type numbers) in the interrupt vector table, beginning with type 80h. The control program must initialize the SCC interrupts, because the firmware does not initialize these interrupts.

The Interrupt Mask Register

The 80186 has an Interrupt Mask Register (IMR) that is similar to the Intel 8259 mask register. Use this register to individually mask a hardware interrupt request. Write a one (1) to one of the data bits to set the mask for one of the interrupt channels (0 through 7). Write a zero (0) to reset that interrupt channel. As Figure 14 shows, several of the interrupt channels are reserved.

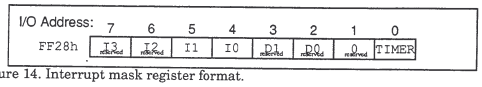


Figure 14. Interrupt mask register format.

The interrupt mask is:

IR mask:	Meaning:
0	Mask reset
1	Mask set

If you mask an interrupt channel, this does not affect other how the other channels operate. Table 19 lists which hardware interrupt is associated with which IMR bit.

Table 19. 80186 hardware interrupts and their respective IMR bits.

Hardware interrupt:	Bit:
TIMER	0
reserved	1
reserved (DMA 0)	2
reserved (DMA 1)	3
8530 SCC's	4
SYSTEM (I/O + 2 write)	5
reserved (interrupt acknowledge)	6
reserved (interrupt acknowledge)	7

When the controllers is powered up, the value set in the interrupt mask register is 00CDh. This enables the SCC interrupts and the SYSTEM interrupt.

Finding the SCC Interrupt Vector Types

Each SCC port can generate four types of interrupts: transmit buffer empty, receive character available, receive special condition, and external/status change.

The SCC interrupts are daisy-chained. You should set VIS=1, NV=0, and STATUS_HIGH/STATUS_LOW=0 in write register 9 of the SCC. When the processor requests an interrupt vector, the SCC places the interrupt vector specified in write register 2 on the bus. This vector is modified to contain status information in bits 1, 2, and 3 that tells the type of interrupt that was generated. Table 20 shows how the vector is modified.

Table 20. SCC vector modification.

Interrupt Vector (binary)	Type of Interrupt
xxxx0000	transmit buffer empty – even numbered port
xxxx0010	external/status change – even numbered port
xxxx0100	receive character available – even numbered port
xxxx0110	special receive condition – even numbered port
xxxx1000	transmit buffer empty – odd numbered port
xxxx1010	external/status change – odd numbered port
xxxx1100	receive character available – odd numbered port
xxxx1110	special receive condition – odd numbered port

The SCC interrupt vectors are placed every eight bytes in the interrupt vector table. Derive the interrupt vector table location by multiplying the modified vector by 4. As an example, the interrupt vector table's location for the receive character available interrupt (for port 5) is:

```
base vector  =      A0h = 10100000 binary
modified bits =     =   xxxx110x
therefore,
modified vector =   =   10101100 = 0ACh

interrupt vector
table location =    =   0ACh * 4 = 2B0h
```

Table 21 shows the interrupt vector table's locations for all SCC interrupts.

Table 21. SCC port interrupt vectors.

Port:	Base Vector type:	Vector table addresses: Transmit buffer empty:	External/ Status change:	Receive character available:	Special receive condition:
1	80h	220h	228h	230h	238h
2	80h	200h	208h	210h	218h
3	90h	260h	268h	270h	278h
4	90h	240h	248h	250h	258h
5	A0h	2A0h	2A8h	2B0h	2B8h
6	A0h	280h	288h	290h	298h
7	B0h	2E0h	2E8h	2F0h	2F8h
8	B0h	2C0h	2C8h	2D0h	2D8h
9	C0h	320h	328h	330h	338h
10	C0h	300h	308h	310h	318h
11	D0h	360h	368h	370h	378h
12	D0h	340h	348h	350h	358h
13	E0h	3A0h	3A8h	3B0h	3B8h
14	E0h	380h	388h	390h	398h
15	F0h	3E0h	3E8h	3F0h	3F8h
16	F0h	3C0h	3C8h	3D0h	3D8h

The following example, from the CPCSTART.ASM header file, initializes the SCC interrupt vectors. Each vector requires 4 bytes. Every second vector is not used, as the SCC modifies bits 3, 2, and 1 of the base vector type, but does not modify bit 0. The unused vectors are already initialized to point to an "invalid interrupt ISR" by the firmware, so they are not altered here:

```
;Name:     vector_init
;Entry:    AX = base vector ype
;Exit:     Nothing
;Registers AX, BX, CX, SI, DI and ES altered

vector_init proc
         shl     ax,1                    ;calculate interrupt vector address
         shl     ax,1
         mov     di,ax
         xor     ax,ax
         mov     es,ax                   ;ES:DI=> destination
         mov     si,offset vector_tbl    ;SI=> vector table
         mov     cx,[si]
         add     si,2
         shr     cx,1                    ;CX = table length (words)
         mov     bx,cs                   ;setup segment address
```



```
;-------------------------------------------------------------------------------
;SCC Interrupt Vector Table - contains addresses of interrupt service routines.

vector_tbl       equ     $
         dw      vector_tbl_end-$-2    ;table length

         dw      line01_TBE
         dw      line01_ESC
         dw      line01_RCA
         dw      line01_SRC

         dw      line00_TBE
         dw      line00_ESC
         dw      line00_RCA
         dw      line00_SRC

         dw      line03_TBE
         dw      line03_ESC
         dw      line03_RCA
         dw      line03_SRC

         dw      line02_TBE
         dw      line02_ESC
         dw      line02_RCA
         dw      line02_SRC


         dw      line05_TBE
         dw      line05_ESC
         dw      line05_RCA
         dw      line05_SRC

         dw      line04_TBE
         dw      line04_ESC
         dw      line04_RCA
         dw      line04_SRC

line07_entry     label   word
         dw      line07_TBE
         dw      line07_ESC
         dw      line07_RCA
         dw      line07_SRC

         dw      line06_TBE
         dw      line06_ESC
         dw      line06_RCA
         dw      line06_SRC

vector_tbl_end   equ     $
```

CHAPTER 8 – Timers

The Intel 80186 has three general purpose timers; however, the Hostess 186 uses timer 2 for controller refresh, so timer 2 is limited in its use. (You can program timer 2's output as a prescaler for the two available timers.)

You control the timers through offsets to an I/O control block. The internal I/O address for the timer's control block is FF00h through FFFFh. Each timer has four registers that regulate how the timer's operate. Programmers can read or write to these registers regardless of the timer's operation. These 16-bit registers appear in Table 22:

Table 22. Timer registers.

Timer Register:	Timer 0 offset (hex):	Timer 1 offset (hex):
Timer count register	50h	58h
Maximum count (A)	52h	5Ah
Maximum count (B)	54h	5Ch
Timer mode/control word	56h	5Eh

Source: Intel

The timer count register holds the incremental count value used by the timer to compare with the maximum count registers. The microprocessor can read or write to this register at any time.

The maximum count registers A or B holds the maximum count value the timer compares with the value accrued in the timer count register. You can write the maximum count value to this register while the timer is operating. The maximum count value can range from 0 to 2^{16} (65,536).

The timer mode/control word is a 16-bit word with reserved bits that manage the timer's operations. The format for this register is:

Bit: 15	14	13	12	11 through 6	5	4	3	2	1	0
Function: EN	INH	INT	RIU	0	MC	RTG	P	EXT	ALT	CONT

Source: Intel

Figure 15. Format of the timer's mode/control word register

Where:
EN – controls the RUN or HALT status of the timer.
INH – sets particular updating of the EN bit.
INT – lets the timer generate interrupts
RIU – determines which maximum count register to use to compare to the timer count value (0 = register A, 1 = register B).
MC – states that the timer reached its maximum count value.
RTG – indicates the status of the timer's external pin if the timer is set for internal clocking (the EXT bit is set to 0).
P – sets the timer input clock at 2 MHz (P = 0) or to use timer 2's output as timer input (P = 1).
EXT – sets the timer either for internal clocking (EXT = 0) or external clocking (EXT = 1).
ALT – indicates which maximum count register to use to compare the timer count with.
CONT – sets the timer to operate continuously.

The Intel 80186's timers are extremely flexible. For an in-depth explanation of how timers function, and other examples of their use, please refer to Intel's *80186/188, 80C186/C188 Hardware Reference Manual.*

Enabling Timers

To enable a timer, you first set the count value, then set the control word.

You set the count register by writing the frequency to the appropriate "timer count register." Table 23 lists count register addresses.
Table 23. Timer count register addresses.

Timer:	Count register address:
0	FF50h
1	FF58h

The following formulae calculate the timer's count register value (in decimal):
- For P bit set to 0, the input clock equal to one-quarter the CPU clock (2 MHz).
  count value = $\frac{2*10^6}{\text{desired frequency}}$
- For P bit set to 1, the input clock is the output of Timer 2.
  count value = $\frac{32,258}{\text{desired frequency}}$

Table 24. Timer frequencies.

P set to 0:		***P set to 1:***	
Frequency Times per Second	Count Register Hexadecimal Value	Frequency Times per Second	Count Register Hexadecimal Value
30	FFFF	0.5	FFFF
40	C350	1	7E02
50	9C40	5	1933
60	8235	10	0C99
70	6F9B	20	064C
80	61A8	30	0433
90	56CE	40	0326
100	4E20	50	0285
110	4705	60	0219
120	411A	70	01CC
130	3C18	80	0193
140	37CD	90	0166
150	3415	100	0142
...	...	110	0125
200	2710	120	010C
		130	00F8
		140	00E6
		150	00D7
		...	...
		200	00A1

To set the control word, write the appropriate value to the control word register. The control word addresses appear in Table 25.
Table 25. Timer control word register addresses.

Timer:	Control word address:
0	FF56h
1	FF5Eh

Here is an example of setting timer 1 to a frequency of 20 times per second, using the output of timer 2 as the clock input:

```
mov   dx,ff5ah     ; dx = timer 1 max count register A
mov   ax,064Ch     ; 20 times per second
out   dx,ax        ; write frequency out
```

Disabling Timers

Disable the two general-purpose timers by writing the value 0 (zero) to the timer's control word register.

Here is an example that clears timer 0 and timer 1:

```
mov   dx,ff52h    ; dx = timer 0 max count register A
mov   ax,0000h    ; zeros
out   dx,ax       ; write zeros out
mov   dx,ff54h    ; dx = timer 0 max count register B
mov   ax,0000h    ; zeros
out   dx,ax       ; write zeros out
mov   dx,ff56h    ; dx = timer 0 control word register
mov   ax,0000h    ; zeros
out   dx,ax       ; write zeros out

mov   dx,ff5ah    ; dx = timer 1 max count register A
mov   ax,0000h    ; zeros
out   dx,ax       ; write zeros out
mov   dx,ff5ch    ; dx = timer 1 max count register B
mov   ax,0000h    ; zeros
out   dx,ax       ; write zeros out
mov   dx,ff5eh    ; dx = timer 1 control word register
mov   ax,0000h    ; zeros
out   dx,ax       ; write zeros out
```

CHAPTER 9 – Serial Communication Controller Port Communication

Talking to the SCC Ports

Each SCC has two ports on it, and each port has a command register and a data register. The command register sets up the communication parameters (baud rate, parity, data bits, stop bits, flow control, and so forth). (Refer to the AMD's or Zilog's 8530 technical manual for specifics on setting up the SCC port.) The data register transmits and receives data.

Each SCC port has preassigned command and data register I/O addresses, which are listed in Table 26. These addresses are accessible from the controller side only.

Table 26. SCC I/O addresses.

Hostess 186 Port:	SCC port:	Command register:	Data register:
1	0	0004h	0006h
2	1	0000h	0002h
3	2	0084h	0086h
4	3	0080h	0082h
5	4	0104h	0106h
6	5	0100h	0102h
7	6	0184h	0186h
8	7	0180h	0182h

The examples that follow show how to write to a command register and a data register for a particular port.

This example writes a value (in this case a 3) to port 1's command register:

```
outp (0x0004, 4);       /* Setup register 4 index on port 1 */
outp (0x0004, 3);       /* Write value (3) to register 4 */
```

This example writes a value (in this case a 31h) to port 1's data register:

```
outp (0x0006, 0x31);    /* Write value (31h) to data register on port 1 */
```

Reading the Modem Status Register

The modem status register is a 16-bit register located at 0200h in the I/O space. This register reports the state of the Data Set Ready (DSR) and Ring Indicator (RI) signals of DCE devices connected to the Hostess 186's ports.

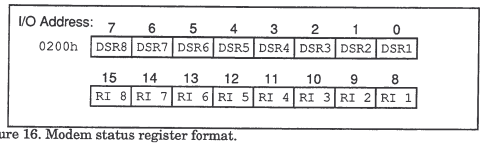


Figure 16. Modem status register format.

The register values are:

Bit value:	Meaning:
0	Signal low (inactive)
1	Signal high (active)

CHAPTER 10 – Downloading and Executing a Control Program

Downloading a Control Program

To download and to execute a control program, follow these steps:

1. Write the control program's executable code to dual-ported RAM, starting at controller memory address 10080h.
2. Compliment the two-byte "interaction flag" at controller memory address 10000h from 55AAh to AA55h.
3. Interrupt the controller by writing to the "I/O_base+2" address. This generates a system-to-controller (SYSTEM) interrupt. The firmware has set up a service routine for this interrupt that verifies that the "interaction flag" is equal to AA55h, and jumps to controller memory address 10080h to execute the control program.
4. Execute the control program. The control program should immediately: disable interrupts , allocate a local stack, initialize the interrupt vectors for the system's interrupts, timers, and SCCs.
   Following this initialization, enable interrupts and continue with normal operation.

We recommend that the control program sets up all its interrupt vectors and then compliments the bytes of the "interaction flag" back to 55AAh as a signal to the system that the control program is functioning properly.

The following paragraph summarizes the steps needed to download a control program to the Hostess 186 controller.

Downloading Summary:
1. Write the control program code into dual-ported memory starting at controller memory address 10080h.
2. Compliment the "interaction flag" at controller memory address 10000h from 55AAh to AA55h.
3. Interrupt the controller.

Using the DPLOADER Program

You can use the DPLOADER program, found on the *Sample Programs* diskette, to download a control program to the Hostess 186. DPLOADER is a DOS program. To use it, follow these steps:

1. Execute DPLOADER.EXE.

```
C: dploader
```

2. DPLOADER will prompt you for values it needs to download the control program.

```
Enter most significant digit of dual port RAM address in hex:     d
Enter I/O base address in hex:     218
     Dual Port Base Address = D000:0000
     I/O Base Address = 218H
Reset HOSTESS 186 controller (Y/N)?     y
     Waiting for reset to complete
Enter control program file name to download:     cpc.exe
Enter number of bytes to strip off file:     640
Invoke Turbo Debugger Remote Kernel on HOSTESS 186 controller (Y/N)?       n
     Downloading cpc.exe...
     XXXXX bytes downloaded successfully.
     COM processor interrupted to start control  program
     Control program started execution
```

(This example uses the CPC.EXE control program. The 640 bytes stripped off the beginning of CPC.EXE include the 512 byte .EXE file header, and the 128 byte "firmware data area" that should not be overwritten.)

CHAPTER 11 – Using Turbo Debugger®

Borland's Turbo Debugger is a source-level debugger that provides a windowing user interface. COMTROL supports the use of this debugger, allowing you to execute and debug programs that operate on the Hostess 186 eight-port controller. There are two debugging environments possible:

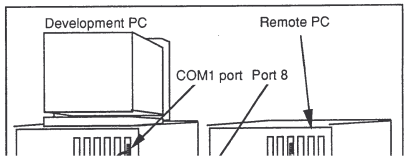
- a PC with both the Hostess 186 and Turbo Debugger installed, and
- two PCs, one with Turbo Debugger installed, and the other with the Hostess 186 installed.

This section explains how to set up the latter system. The examples used in this section use the C versions of the sample programs found on the *Sample Programs* diskette.

Setting Up the Debugging Environment Hardware

The two-PC debugging environment has a development PC that displays Turbo Debugger, and a remote PC system. The remote system runs Hostess 186 control programs under the Turbo Debugger environment. These systems must be configured as follows:

- The development system must be an IBM PC or compatible, with DOS 2.0 or higher, and at least 384K of RAM. A hard disk is recommended.
- The remote system, for these sample programs, must have the Hostess 186 controller installed.
  Set the Hostess 186 for I/O address 218h. The sample program CPC.EXE sets the Hostess 186 to use 64K of system-side memory, starting at D000:0. Make sure that no device occupies this memory space.
- Connect the development PC to the remote PC with a RS-232 null-modem cable. Attach one end of this cable to port 8 of the Hostess 186's interface box. Attach the other end of this cable to the COM1 port of the development PC.



Setting Up the Debugging Environment Software

This section uses examples from the C programs found on the *Sample Programs* diskette. Create directories on both the remote and development PCs, and copy the following programs to the appropriate systems:

Remote system:		Development system:	
DPLOADER	Executable	CPC.C	Source
CPC.EXE	files	CPC.H	files
HITERM.EXE		CPCSTART.ASM	
		CPC.TDS	Symbol table

If you have not done so already, install Turbo Debugger on the development system.

The following steps show how to invoke Turbo Debugger and these files on both systems.

1. On the remote system, invoke DPLOADER.

```
C: dploader
```

2. Identify these values: the dual-port RAM's most significant digit, the I/O base address, the name of the download program, and the strip-off value. Authorize DPLOADER to reset the Hostess 186 and to invoke the remote Turbo Debugger kernel. (For this example, use CPC.EXE as the file to download.)

```
Enter most significant digit of dual port RAM address in hex: d
Enter I/O base address in hex: 218
     Dual Port Base Address = D000:0000
     I/O Base Address = 218H
Reset HOSTESS 186 controller (Y/N)? y
     Waiting for reset to complete
Enter control program file name to download: cpc.exe
Enter number of bytes to strip off file: 640
Invoke Turbo Debugger Remote Kernel on HOSTESS 186 controller (Y/N)? y
     Turbo Debugger Remote Kernel started.
     Downloading cpc.exe...
     XXXXX bytes downloaded successfully.

C:
```


3. On the development system, invoke Turbo Debugger.

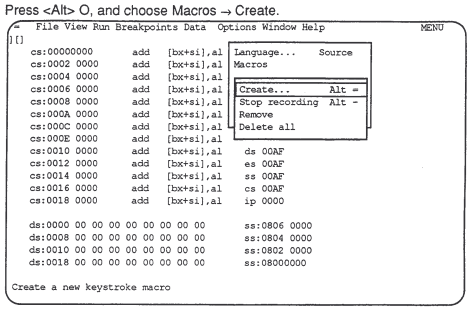
```
C: td -rp1 -rs3
```

The `-rp1` argument specifies the COM1 port. The `-rs2` argument sets the fixed speed of the serial link. For Turbo Debugger versions 2.5 and above, use the `-rs3` argument to specify 38.4Kbaud. For Turbo Debugger versions 2.0 and below, use the `-rs2` argument to specify 38.4Kbaud. An opening window appears first, followed by the CPU window.

```
≡  File View Run Breakpoints Data  Options Window Help                  READY
][] Remote CPU
    cs:00000000       add   [bx+si],al    ax 0000  c=0
    cs:0002 0000      add   [bx+si],al    bx 0000  z=0
    cs:0004 0000      add   [bx+si],al    cx 0000  s=0
    cs:0006 0000      add   [bx+si],al    dx 0000  o=0
    cs:0008 0000      add   [bx+si],al    si 0000  p=0
    cs:000A 0000      add   [bx+si],al    di 0000  a=0
    cs:000C 0000      add   [bx+si],al    bp 0000  i=0
    cs:000E 0000      add   [bx+si],al    sp 0800  d=0
    cs:0010 0000      add   [bx+si],al    ds 00AF
    cs:0012 0000      add   [bx+si],al    es 00AF
    cs:0014 0000      add   [bx+si],al    ss 00AF
    cs:0016 0000      add   [bx+si],al    cs 00AF
    cs:0018 0000      add   [bx+si],al    ip 0000

    ds:0000 00 00 00 00 00 00 00 00       ss:0806 0000
    ds:0008 00 00 00 00 00 00 00 00       ss:0804 0000
    ds:0010 00 00 00 00 00 00 00 00       ss:0802 0000
    ds:0018 00 00 00 00 00 00 00 00       ss:08000000

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

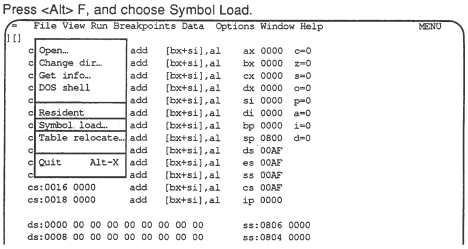
The following steps are tedious, so we recommend that you invoke Turbo Debugger's Macro recording function – this will save you time and keystrokes when you start subsequent debugging sessions. Note that the following examples use the CPC.EXE and CPCSTART files. Your macros will differ, depending on the files to debug. (To invoke the macro function, press <Alt> O and choose Macros from the menu.)

4. Press <Alt> O, and choose Macros → Create.

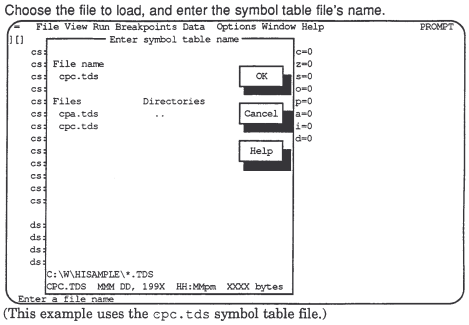


Select Create from the Macros sub-menu. Turbo Debugger will prompt you for the keystroke sequence that starts the macro.

5. Press <Alt> F, and choose Symbol Load.

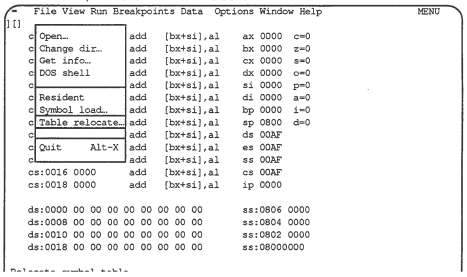


6. Choose the file to load, and enter the symbol table file's name.

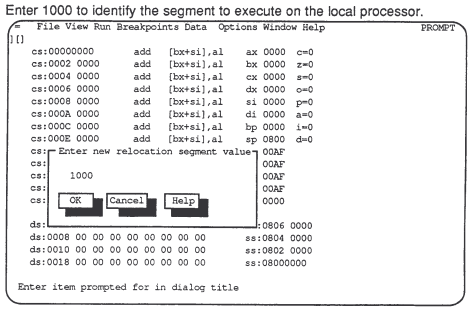


(This example uses the `cpc.tds` symbol table file.)

7. Press <Alt> F, and choose Table relocate....

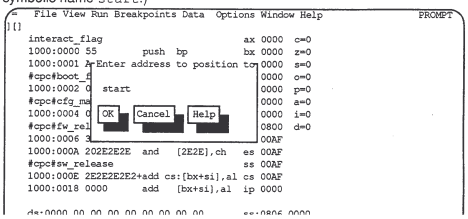


8. Enter 1000 to identify the segment to execute on the local processor.



This specifies the segment to execute on the Hostess 186 controller's processor. You should always specify 1000 for this value.

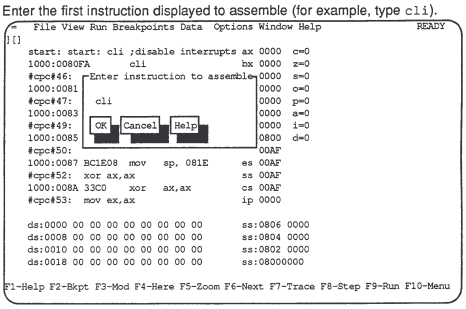
9. Press <Ctrl> G, and enter the address of the entry point. (This example uses the symbolic name `start`.)



10. Press <Ctrl> N.

When you press <Ctrl> N, this updates the registers for the CS:IP (current segment:instruction pointer).

11. Enter the first instruction displayed to assemble (for example, type `cli`).



From this point forward, you may customize your environment to you liking – remember that the macro facility records all of your actions. To stop recording, choose Options → Macros → Stop Recording. To save the macro, choose Options → Save Options. For future debugging sessions, use this macro to automatically replay steps 6 through 11. Finally, to invoke this macro, type the specified macro keystrokes when Turbo Debugger starts.

Configuring Symbol Tables

To use Turbo Debugger, you must generate a symbol table to accompany your program. To create a symbol table:

1. Use the compiler's command options to compile and link your program (Refer to your vendor's documentation for specifics.)
2. Run the resulting .EXE file through Turbo Debugger's symbol table separating utility called TDSTRIP.EXE. This utility removes the symbol table from the executable file and places it in a separate file. The symbol table file has the extension .TDS.

The following examples, from the sample make file TSAMPLE.MK, explain the options used to make a symbol table.

For the assembly language example, the "make" is:

```
    tasm /l /s /zi cpa.asm, cpa.obj
         #/l: Create a listing file
         #/s: Use source code segment ordering
         #/zi: Include full symbolic debug information in object file
    tlink /m /s /v cpa.obj, cpa.exe, cpa.map
         #/m: Create a map file
         #/s: Put detailed map of segments in map file
         #/v: Include full full symbolic debug information in executable file
    tdstrip -s -c cpa.exe cpa.tds
         #/s: Put symbol table in file cpa.tds
         #/c: Create .COM file (this is optional, .EXE file can also be used)
```

For the C example, the "make" is:

```
#CC = tcc                                  # for Turbo C++
CC = bcc                                   # for Borland C++
#CPCLIB = \tc\lib\cs.lib                   # for Turbo C++
CPCLIB = \borlandc\lib\cs.lib              # for Borland C++
...
#CPC (built with symbol table for debugging with Turbo Debug)
cpc.exe: cpc.obj cpcstart.obj tsample.mk
    tlink /m /s /v cpcstart.obj cpc.obj, cpc.exe, cpc.map, $(CPCLIB)
         #/m: Create a map file
         #/s: Put detailed map of segments in map file
         #/v: Include full full symbolic debug information in executable file
         #Note: The start up module CPCSTART.OBJ must be placed first in the
         #      list of object files!
    tdstrip -s cpc.exe
         #/s: Put symbol table in file cpa.tds
cpc.obj: cpc.c tsample.mk
    $(CC) -c -mt -G -v -ocpc.obj cpc.c
         #-c: Compile to object file but do not link
         #-mt: Compile using tiny model
```


Invoking the Remote Turbo Debugger Kernel

The firmware on the Hostess 186 contains a Turbo Debugger remote "kernel" that must be invoked before Turbo Debugger on the remote computer can establish communication with the Hostess 186. Invoke the kernel by executing an `int 27h` interrupt to the Hostess 186's processor.

There are two methods to invoke the kernel. The first is demonstrated in the sample DPLOADER.C program, function `invoke_tdrem()`. Here the system processor writes the `int 27h` opcode value of `27cdh` into dual-port RAM at local processor address 1000:80. Next, the system processor interrupts the local processor, which executes the `int 21h` interrupt service routine to start the kernel. After this, you can download a control program and start the debugging session.

function invoke_tdrem():

```
/* Entry: flag_p - Pointer to interaction flag                              */
/*        io     - Board I/O base address                                   */
/* Returns: 1 if tdrem is invoked, else 0                                   */

int invoke_tdrem(int io,unsigned far *flag_p)
{
   char strbuff[0x81];                     /* Stores string entered by user */
   int do_tdrem,valid_answer,wait;
   unsigned rec_flag;                      /* Received copy of interact flag*/

   valid_answer = 0;
   while(!valid_answer)
   {
      printf("Invoke Turbo Debugger Remote Kernel on Hostess 186 controller (Y/N)? ");
      scanf("%s",strbuff);
      switch(*strbuff)
      {
         case 'y':
         case 'Y':
            valid_answer = 1;
            do_tdrem = 1;
            break;
         case 'n':
         case 'N':
            valid_answer = 1;
            do_tdrem = 0;
            break;
         default:
            break;
      }
   }
```

```
   if(do_tdrem)
   {
      *(flag_p + 0x40) = 0x27cd;        /* Write int 27H instruction used to
                                           invoke TDREM kernel */
      *flag_p = 0xaa55;                 /* Set up interaction flag */
      outp (io + 2,0);                  /* Interrupt COM Processor */
      for(wait = 0;wait < 10;wait++)    /* wait for execution to finish*/
      {
         delay(200);
         if(*flag_p == 0x55aa)
            break;
      }
      if((wait >= 10) && ((rec_flag = *flag_p) != 0x55aa))
      {
         printf("   Turbo Debugger Remote Kernel failed to start, interact flag = %XH\n",rec_flag);
         exit(1);
      }
      printf("   Turbo Debugger Remote Kernel Started\n");
   }
   return(do_tdrem);
}
```

The second method that invokes the Turbo Debugger remote kernel is to embed the `int 27h` interrupt within the downloaded control program. If you do this, in the <Ctrl> G step of the Turbo Debugger startup sequence, enter the starting address of the instruction immediately following the `int 27h`. Also, change the interaction flag at local address 1000:0 to AA55h before executing the `int 27h` interrupt. The interrupt service routine will restore this flag back to 55AAh; this will serve as an indicator that the interrupt was handled correctly. In assembly language this could be written as:

```
         mov  ax,1000h                 ;ES -> interaction flag
         mov  es,ax
         mov  es:0,0aa55h              ;write interaction flag
         int  27h                      ;invoke Turbo Debugger remote kernel
    start_debug:                       ;symbolic address for
                                       ; Turbo Debugger <CTRL G>
```


Notes on Using Turbo Debugger

When debugging code with Turbo Debugger remotely, remember that line 7 (port 8) of the Hostess 186 is used by Turbo Debugger. Therefore, your program must not use nor initialize line 7 of the SCC channel while debugging remotely.

The <CTRL BREAK> feature of Turbo Debugger is not operational with this implementation of Turbo Debugger remote. If Turbo Debugger is "running" and a breakpoint is never reached, there is no method of breaking execution without rebooting DOS on the remote system and restarting the entire debugging session. However, you can use the debug switch to issue an NMI (non-maskable interrupt) to the 80186. This will halt the executing program and return control to Turbo Debugger. Turbo Debugger replaces the non-maskable interrupt vector (type 2) with its own interrupt vector when Turbo Debugger starts up. Turbo Debugger also replaces the TRACE and Breakpoint interrupt vectors (types 1 and 3) with its own interrupt vectors. The remaining interrupt vectors are unaffected.

Single-stepping with the <F7> and <F8> function keys through instructions which result in hardware interrupts may not generate the interrupt reliably. For example, the instruction `outp(0xe1f6,0x31)` writes a character out to line 0. This will normally result in a Transmit Buffer Empty (TBE) interrupt. However, if this instruction is single-stepped, the interrupt may not occur. To avoid this, set a breakpoint after the `outp()` and run to it, rather than single stepping over the `outp()` instruction.

Do not single-step instructions that follow the end-of-interrupt (EOI) to the programmable interrupt controller (PIC). The trace instruction uses a software interrupt (INT 3) to stop the next instruction from executing. If you single-step after an EOI is issued, the interrupt service routine (ISR) for the INT 3 software interrupt issues an IRET. This enables other interrupts to occur before the current interrupt is serviced.

Place a NOP instruction after the last instruction in the main procedure if an interrupt service routine follow the main procedure, and this interrupt could occur during a trace (F7/F8) of the last instruction in the main procedure.

You can debug only code that resides in RAM. Turbo Debugger cannot make firmware debugging calls (for example int 21h, the firmware RAM query) to the Hostess 186.

How to Connect the Debug/Reset Switch to the Controller

This is an option that must be ordered for the Hostess 186. (Order part number HO86DC00A, the development kit for the Hostess 186. There is no extra charge for this option.)

The Debug/Reset switch requires a three-prong header at the top center of the controller. If you want to use the firmware debugger, check that the controller you are using has this header. Please call COMTROL Technical Support if this header is missing from your controller.

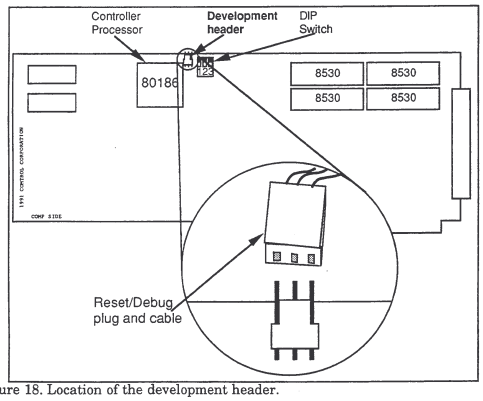


Figure 18. Location of the development header.

CHAPTER 12 – Using the Hostess 186's Firmware Debugger

Debugger Setup

The debugger is provided as part of the firmware installed in the Hostess 186 controller. This debugger can set a breakpoint, display memory, display registers, disassemble instructions, perform input and output to I/O ports, and single step through instructions.

The debugger console is initially assigned to the first serial port on the Hostess 186 controller. The serial communications parameters are defined as follows:

- 9600 bits/second
- no parity
- eight (8) bits per character
- one (1) stop bit

These parameters are fixed; a program cannot alter them. This picture shows how to setup a terminal to port one on a development PC:

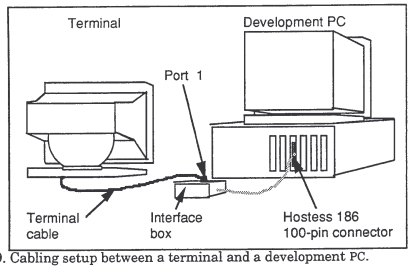


Figure 19. Cabling setup between a terminal and a development PC.

Invoking the Firmware Debugger

The firmware debugger essentially operates as an interrupt service routine. The Hostess 186 firmware provides access to the debugger and its functions through two software interrupts:

INT 20h
: A program may invoke the debugger through this interrupt. The firmware configures serial port 1 as the debug console during system initialization.

INT 22h
: After the system initializes, a program may change the debug console by executing an interrupt 22h. Load a valid device number from 0 to 7 (ports 1 through 8) into register AL before executing the software interrupt.

If you have an interface box with a debug switch, pressing that switch generates a NMI hardware interrupt that invokes the firmware debugger.

A Summary of Debugger Commands

The Hostess 186 debugger supports the following commands:

Command	Name	Function
B	Byte	All succeeding Input or Output commands read or write 8 bit values.
D	Dump	Displays the contents of a specified memory region.
G	Go	Continues execution from the current location with or without a breakpoint.
I	Input	Inputs and displays a byte or word from the specified I/O port.
O	Output	Outputs a byte or word to the specified I/O port.
R	Register	Displays the contents of all registers.
T	Trace	Executes the next instruction (single step).
U	Unassemble	Disassembles a specified memory region.
W	Word	Formats all succeeding Input or Output commands to read or write 16-bit values.

Debugger Command Definitions

This section describes how to use each of the debugger commands. The commands appear in alphabetical order. The function and format of each command appears as well as remarks and examples where appropriate.

For **all** the debugger commands:

- Each command is a single letter, which may be followed by one or more parameters.
- Parameters shown in CAPITAL LETTERS mean that you should substitute your value for that item.
- Optional parameters appear inside square brackets [ ].
- Commands and parameters may be entered using uppercase, lower-case, or a combination of both.
- Commands execute only after you press the <Enter> key.
- The debugger prompt is a hyphen (-).
- If the debugger encounters a syntax error, the pointer error shows the location of the error.

Debugger Commands

B (Byte Mode)

Causes all succeeding Input or Output commands to read or write eight-bit (byte) values.

Format: `B`

No arguments are required.

D (Dump)

Displays the contents of a specified memory region.

Format: `D [STARTING ADDRESS] [ENDING ADDRESS]`

or

```
D [STARTING ADDRESS] [L LENGTH]
```

`STARTING ADDRESS` specifies the first address of a range of addresses to be displayed. It may take any of the following forms:

1. A segment value, offset value pair separated by a colon (:).
2. A segment register mnemonic and an offset value separated by a colon (:).
3. An offset value only. A default segment will be used.

`ENDING ADDRESS` is an offset value, within the segment specified by the `STARTING ADDRESS`, which specifies the last address of a range of addresses to be displayed.

Alternatively, `L LENGTH` specifies the number of bytes to be displayed. If no `ENDING ADDRESS` or `L LENGTH` is specified, the default display length is 128 bytes.

If no arguments are specified and no previous `D` command has been entered, display will start at the current CS:IP location. If a `D` command has previously been entered, display will start with the byte following the last byte previously displayed. Again, the default length is 128 bytes.

For example, both of these commands display the contents of memory from 0040h:000h through 0040h:00FFh:

```
D 40:0 FF
```

or

```
D 40:0 L 100
```

This command displays the contents of memory from offset 1000h, within the segment currently pointed to by the ES register, through offset 107Fh, the default length:

```
D ES:1000
```

G (Go)
Executes from the current location with or without breakpoints.

Format: `G [BREAKPOINT ADDRESS]`

`BREAKPOINT ADDRESS` specifies an address where program execution will be interrupted and control returned to the debugger.

If no breakpoint is specified, the program continues to execute normally.

For example, this command allows program execution to continue from the current location (CS:IP) and sets a breakpoint at address 4000h:0007h. Program execution will be interrupted and control returned to the debugger if the program attempts to execute the instruction at this address:

```
G 4000:7
```

I (Input)
Inputs and displays a byte or word from the specified I/O port.

Format: `I PORTADDRESS`

`PORTADDRESS` specifies a 16 bit I/O address from which data will be input. The size of the input data (byte or word) will depend on the current I/O mode (see the Byte and Word commands).

For example, this command inputs data from the I/O port at address 202h:

```
I 202
```

O (Output)
Outputs a byte or word to the specified I/O port.

Format: `O PORTADDRESS VALUE`

`PORTADDRESS` specifies a 16-bit I/O address to which data will be output.

`VALUE` is the data to be output. The size of the output data (byte or word) will depend on the current I/O mode (see the Byte and Word commands)

R (Register)
Displays the contents of all registers.

Format: `R`
No arguments are required.

T (Trace)
Executes the next instruction (single step).

Format: `T`

No arguments are required.

U (Unassemble)
Disassembles a specified memory region.

Format: `U [STARTING ADDRESS] [COUNT]`

`STARTING ADDRESS` specifies the first address of a range of addresses to be disassembled. It may take any of the following forms:

1. A segment value, offset value pair separated by a colon (:).
2. A segment register mnemonic and an offset value separated by a colon (:).
3. An offset value only. A default segment will be used.

`COUNT` is the number of instructions to disassemble. If no `COUNT` is specified, a default of 16 instructions will be disassembled.

If no arguments are specified and no previous `U` command has been entered, disassembly will start at the current CS:IP location. If a `U` command has previously been entered, disassembly will start with the instruction following the last instruction previously displayed.

For example, this command will disassemble eight instructions starting at address 4000h:0003h:

```
U 4000:3 8
```

W (Word Mode)
Causes all succeeding Input or Output commands to read or write 16-bit (word) values.

Format: `W`

No arguments are required

Notes on Using the Firmware Debugger

Here are a few "features" that you should be aware of when using the firmware debugger:

1. Jump instructions display the next instruction's address as a relative address and not as an absolute address.
2. Non-8086 instructions do not disassemble correctly. These instructions appear as:

   ```
   * data *
   ```

   These instructions will execute correctly, however.
3. Timer, systems, and SCC interrupts may continue to occur when you use the firmware debugger. These system interrupt routines (ISRs) cannot make any assumptions about the state of any registers, including the segment registers, because the firmware debugger has modified these registers for its own use.

   Therefore, when you start debugging, for all the registers used by ISRs, the ISR must save and initialize these registers, and then restore them before you exit the ISR.

How to Connect the Debug/Reset Switch to the Controller

This is an option that must be ordered for the Hostess 186. (Order part number HO86DC00A, the development kit for the Hostess 186. There is no extra charge for this option.)

The Debug/Reset switch requires a three-prong header at the top center of the controller. If you want to use the firmware debugger, check that the controller you are using has this header. Please call COMTROL Technical Support if this header is missing from your controller.

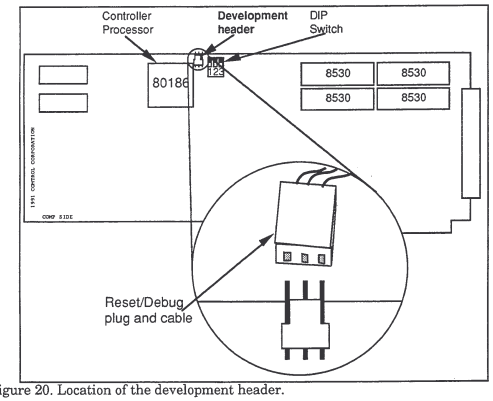


Figure 20. Location of the development header.

APPENDIX A – Assembly Language Listings

The following files are in the 80186 assembly language:

- CPA.ASM – the assembly language source code for CPA.COM.
- CP.EQU – the include file for CPA.ASM

Invoking HITERM

Here are a few guideline for executing the HITERM.EXE program with the CPA.COM assembly language control program:

1. Set the Hostess 186 for I/O address 218h.
2. Check that no other device occupies the D000 base memory address.
   The program uses 64K starting at D000:0.
3. Install the controller in the system.
4. Connect a non-intelligent ASCII terminal to the port on the Hostess 186 that you want to use.
   Set the terminal to:
   - 9600 baud
   - 8 data bits
   - 1 stop bit
   - no parity
   - no flow control.
5. Start-up DOS.
6. Execute DPLOADER.EXE.

```
C: dploader
```

   DPLOADER will prompt you for values it needs to download the control program.

```
Enter most significant digit of dual port RAM address in hex:     d
Enter I/O base address in hex:     218
     Dual Port Base Address = D000:0000
     I/O Base Address = 218H
Reset HOSTESS 186 controller (Y/N)?     y
     Waiting for reset to complete
Enter control program file name to download:     cpa.com
Enter number of bytes to strip off file:     0
Invoke Turbo Debugger Remote Kernel on HOSTESS 186 controller (Y/N)?         n
     Downloading cpc.exe...
     XXXXX bytes downloaded successfully.
     COM processor interrupted to start control  program
     Control program started execution
```

(To run the Turbo Debugger verison; answer <y> to the last question.)

7. Execute HITERM.EXE.
   The HITERM application sends and receives any characters you type on either keyboard. Pressing the <F10> key terminates the transmittal.

CPA.ASM

```
        page    60,80
.186

;******************************************************************************
;File:     cpa.asm
;Purpose: Sample assembly language control program for HOSTESS 186.
;         Supports open, close, read, and write to any asynchronous line.
;Company: Comtrol Corporation
;Author:  Craig Harrison
;Release: 1.00, 2-18-92 - Original release
;******************************************************************************
;Copyright 1992 Comtrol Corporation. All rights reserved. Subject to developers
;license agreement.
;******************************************************************************

          .xlist
          .list
          include cp.equ

_TEXT     segment para public  'CODE'
          assume  cs:_TEXT, ds:_TEXT

                  org 0h
;The first 80h bytes are the "firmware user area" defined by the HOSTESS 186
;firmware

          public  interact_flag
interact_flag     dw      ?             ;processor interaction flag
boot_flag         dw      ?             ;boot/activity flag
cfg_map           dw      ?             ;configuration map
fw_release        db      8 dup (?)     ;firmware release number
sw_release        db      8 dup (?)     ;control program release number
                  dd      ?             ;reserved
dram_map          dd      ?             ;DRAM map
scc_map           dd      ?             ;SCC map
board_id          dd      ?             ;board ID
ii_flag           db      ?             ;invalid interrupt flag
ii_type           db      ?             ;invalid interrupt type
ii_cnt            dw      ?             ;invalid interrupt count
                  db      128-42 dup (?) ;balance of firmware area


                  org     80h             ;firmware jumps here when interrupted
          public  cpmain, start
cpmain    proc
```

```
        xor     ax,ax
        mov     es,ax
        mov     bx,INT1_type*4          ;setup system interrupt vector
        mov     ax,offset system_isr
        mov     es:[bx],ax
        mov     ax,cs
        mov     es:[bx+2],ax

        mov     bx,TIM1_type*4          ;setup timer 1 intr vector
        mov     ax,offset timer1_isr
        mov     es:[bx],ax
        mov     ax,cs
        mov     es:[bx+2],ax

        mov     ax,TIMER_INT_CTRL       ;write to interrupt timer cotrol reg.
        mov     dx,ax
        mov     ax,0000h                ;allow interrupts for timer 1.
        out     dx,ax


        mov     dx,TMR1_MAX_CNTA
        mov     ax,TIMER1_CNT           ;30/sec
        out     dx,ax
        mov     dx,TMR1_CTRL_reg
        mov     ax,0e001h               ;enable timer 1, max count A
        out     dx,ax

        mov     ax,base_vector          ;AX = base vector type
        call    vector_init             ;initialize vector table

        mov     ax,cs
        mov     es,ax
        mov     di,offset interact_flag ;DI=> interaction flag
        mov     [di],55aah              ;restore interaction flag

        mov     di,offset boot_flag     ;indicate control program active
        mov     [di],0ffffh

        mov     di,offset sw_release    ;move software release number
        mov     si,offset release       ;  to shared memory
        mov     cx,release_len
        rep     movsb

        sti                             ;enable interrupts
```



```
;Main processing, is an infinite loop
main_10:
        mov     si,offset line00        ;SI=> 1st line table
main_30:
        mov     ax,[si].line_status     ;get line status
        test    ax,line_active          ;is line active ?
        jnz     short main_40
        jmp     main_70                 ;  no ... move on to next line
main_40:
        test    ax,Tx_active            ;is transmit active ?
        jnz     main_70                 ;  yes ... move on to next line
        call    deq_Tx_data             ;character from xmit queue to send?
        jc      main_70                 ;  no ... continue
main_60:
        cli                             ;protect SCC out from interrupts
        or      [si].line_status,Tx_active   ; indicate transmit active
        mov     dx,[si].io_base         ;get base I/O address
        add     dx,2                    ;offset to data register
        out     dx,al                   ;move character to SCC
        sti
main_70:
        cmp     si,offset line07        ;last table?
        jl      short main_75           ;  no ... continue
        jmp     main_10                 ;otherwise start over ...
main_75:
        add     si,line_entry_len       ;bump to next line table
        jmp     main_30
        nop
cpmain  endp

;---------------------------------------------------------------------------
; Name: timer1_isr
; Purpose: Process interrupt from timer 1.  Doesn't do anything useful, just
;          increments a word in dual port RAM to demonstrate that its working.
; Entry: Nothing
; Exit:  Nothing

timer1_isr proc
        push    ax                      ;save registers
        push    dx
        inc     word ptr ds:7ch         ;increment word
        mov     dx,INTCTL               ;end of interrupt to PIC
        mov     ax,EOI_VAL
        out     dx,ax
        pop     dx                      ;recover registers
        pop     ax
        iret
timer1_isr endp
```

```
;-------------------------------------------------------------------------------
;Name:    system_isr
;Purpose: Process interrupt from sytem processor
;Entry:   Nothing
;Exit:    Nothing

system_isr proc
         pusha                          ;save registers

system_isr_02:
         call    deq_Com_msg            ;Comm Processor message waiting ?
         jc      system_isr_20          ;  no ... exit
         mov     al,msg_area            ;get command
         xor     ah,ah                  ;clear upper byte
         shl     ax,1                   ;double the command value
         mov     bx,offset command_tbl  ;BX=> command table
         add     bx,ax                  ;offset into table
         cmp     bx,offset command_tble ;valid command ?
         jnc     system_isr_10          ;  no ... continue
         call    word ptr [bx]          ;invoke command processor
system_isr_10:
         jmp     system_isr_02          ;check for another message
system_isr_20:
         mov     dx,INTCTL              ;end of interrupt to PIC
         mov     ax,EOI_VAL
         out     dx,ax
         popa
         iret
system_isr endp

command_tbl      equ     $
         dw      null_cmd              ;0 - null command
         dw      open                  ;1 - open  a line
         dw      close                 ;2 - close a line
         dw      int_sys               ;3 - generate interrupt to system
command_tble     equ     $



;-------------------------------------------------------------------------------
;Name:    null_cmd (0)
;Purpose: Handle unimplemented command
;Entry:   Nothing
;Exit:    Nothing

null_cmd proc
         ret
null_cmd endp
```



```
;-------------------------------------------------------------------------------
;Name:    open (1)
;Purpose: Open a line for asynchronous communications
;Entry:   msg_area+1 = line number
;         msg_area+2 = WR3 parameters   (Rx character size)
;         msg_area+3 = WR4 parameters   (stop bits, parity)
;         msg_area+4 = WR5 parameters   (Tx character size)
;         msg_area+5 = WR12 parameter   (lower byte of BRGTC)
;         msg_area+6 = WR13 parameter   (upper byte of BRGTC)
;Exit:    Nothing

open     proc
         call    get_line              ;return SI=> line table entry
         call    configure_line        ;configure line for async communications
         call    enable_line           ;enable line
         ret
open     endp

;-------------------------------------------------------------------------------
;Name:    close (2)
;Purpose: Close a serial line
;Entry:   msg_area+1 = line number
;Exit:    Nothing

close    proc
         call    get_line              ;SI=> line table entry
         mov     dx,[si].io_base       ;get base I/O address
         mov     al,WR1                ;point to WR1
         out     dx,al
         xor     al,al                 ;disable all interrupts
         out     dx,al
         mov     [si].line_status,0    ;disable all in line_status
         ret
close    endp

;-------------------------------------------------------------------------------
;Name:    int_sys (3)
;Purpose: Generate an interrupt to the system.  This would probably never be
;         executed as a command from the system by is included here to show how
;         to interrupt the system.
;Entry:   Nothing
;Exit:    Nothing

int_sys  proc
         mov     dx,C2Sint_reg         ;address of interrupt reg
         mov     ax,C2Sint_low         ;set interrupt line low
         out     dx,ax                 ;do it
```

```
;-------------------------------------------------------------------------------
;Name:    configure_line
;Purpose: Configure the communications parameters in the line table
;Entry:   SI=> line table entry
;         msg_area+1 = line number
;         msg_area+2 = WR3 parameters   (Rx character size)
;         msg_area+3 = WR4 parameters   (stop bits, parity)
;         msg_area+4 = WR5 parameters   (Tx character size)
;         msg_area+5 = WR12 parameter   (lower byte of BRGTC)
;         msg_area+6 = WR13 parameter   (upper byte of BRGTC)
;Exit:    Nothing

configure_line  proc
         push    si                     ;save line table start
         add     si,offset WR3_
         mov     di,si                  ;DI=> line table
         mov     si,offset msg_area+2   ;SI=> message area
         lodsb                          ;WR3 value
         and     al,0c0h                ;isolate Rx character size
         stosb                          ;  and move to line table
         lodsb                          ;WR4 value
         and     al,0fh                 ;isolate stop bits/parity
         stosb                          ;  and move to line table
         lodsb                          ;WR5 value
         and     al,60h                 ;isolate Tx character size
         mov     ah,[di]                ;get current value
         and     ah,0ffh-60h            ;  and clear Tx character size
         or      al,ah                  ;combine the two
         stosb                          ;  and move to line table
         mov     cx,2                   ;setup remaining length
         rep     movsb                  ;move BRGTC to line table
         pop     si                     ;recover line table start
         call    scc_init               ;update params to SCC
         ret
configure_line endp


;-------------------------------------------------------------------------------
;Name:    enable_line
;Purpose: Enable a serial line
;Entry:   SI=> line table entry
;Exit:    Nothing
;Sequence to enable interrupts:
;       1.  WR15: specify external/status interrupts
;       2.   WR0: reset external status
;       3.   WR0: reset external status twice
;       4.   WR1: enable receive, transmit and external status interrupts
;       5.   WR9: enable master interrupt enable

enable_line proc
```



```
         mov     al,[si].WR5_           ;make sure RTS is high
         or      al,RTS
         mov     [si].WR5_,al
         mov     dx,[si].io_base        ;get base i/o address
         mov     al,WR5
         out     dx,al
         mov     al,[si].WR5_           ;transmit parameters
         out     dx,al

         ;Enable interrupts
         mov     al,WR15
         out     dx,al
         mov     al,[si].WR15_          ;external/status interrupt control
         out     dx,al

         mov     al,WR0
         out     dx,al                  ;for insurance
         mov     al,reset_ext           ;reset external status interrupts
         out     dx,al

         mov     al,WR1                 ;interrupt enables
         out     dx,al
         mov     al,ext_int_enable+Tx_int_enable+parity_special+Rx_int_enable
         out     dx,al

         mov     al,WR9
         out     dx,al
         mov     al,MIE+status_lo+VIS   ;master interrupt enable
         out     dx,al
         ret
enable_line endp


;-------------------------------------------------------------------------------
;Name:    get_line
;Purpose: Return line table pointer
;Entry:   msg_area+1 = line number
;Exit:    SI=> line table entry

get_line proc
         mov     al,msg_area+1          ;get line number
         xor     ah,ah                  ;clear upper byte
         mov     cx,line_entry_len      ;calculate line table offset
         mul     cl
         add     ax,offset line00
         mov     si,ax
         ret
get_line endp
```

```
;-------------------------------------------------------------------------------
;Name:    scc_init
;Purpose: Initialize SCC for asynchronous operation
;Entry:   SI=> line table entry
;Exit:    Nothing
;Sequence:
;       1.  WR9: reset channel
;       2.  WR4: specify clock mode, number of stop bits and parity
;       3.  WR2: specify base interrupt vector type
;       4.  WR3: specify number of bits/character for receive
;       5.  WR5: specify number of bits/character for transmit
;       6.  WR9: specify interrupt control
;       7.  WR11: specify receive and transmit clock source
;       8.  WR12: specify lower byte of Baud Rate Time Constant
;       9.  WR13: specify upper byte of Baud Rate Time Constant
;      10.  WR14: specify Baud Rate Generator source and enable it
;      11.  WR3: enable receive operation
;      12.  WR5: enable transmit operation

scc_init proc
         mov     dx,[si].io_base       ;get base I/O address

         mov     al,WR4
         out     dx,al
         mov     al,[si].WR4_          ;stop bits and parity
         or      al,x16_clock          ;add clock mode
         out     dx,al

         mov     al,WR2
         out     dx,al
         mov     al,[si].WR2_          ;base interrupt vector type
         out     dx,al

         mov     al,WR3
         out     dx,al
         mov     al,[si].WR3_          ;Rx character size
         out     dx,al

         mov     al,WR5
         out     dx,al
         mov     al,[si].WR5_          ;Tx character size, modem/break
         out     dx,al

         mov     al,WR9
         out     dx,al
         mov     al,status_lo+VIS      ;interrupt control
         out     dx,al

         mov     al,WR11
         out     dx,al
```



```
         mov     al,WR13
         out     dx,al
         mov     al,[si].WR13_         ;upper byte of BRGTC
         out     dx,al

         mov     al,WR14
         out     dx,al
         mov     al,BRG_eq_sys_clk+BRG_enable    ;BRG source
         out     dx,al

         mov     al,WR3
         out     dx,al
         mov     al,[si].WR3_          ;Rx character size
         or      al,Rx_enable          ;enable receive
         mov     [si].WR3_,al
         out     dx,al

         mov     al,WR5
         out     dx,al
         mov     al,[si].WR5_          ;Tx character size
         or      al,Tx_enable          ;enable transmit
         mov     [si].WR5_,al
         out     dx,al
         ret
scc_init endp


;-------------------------------------------------------------------------------
;Name:    deq_Tx_data
;Purpose: Remove character from transmit queue
;Entry:   SI=> line table entry
;Exit:    carry set if queue is empty, else
;         carry clear and AL = character

deq_Tx_data proc
         push    bx
         mov     bx,[si].Txq_tail      ;get queue tail
         cmp     bx,[si].Txq_head      ;is queue empty ?
         stc                           ;(assume it is)
         jz      deq_Tx_data_10        ;  yes ... exit
         push    di                    ;save register
         mov     di,[si].Txq_offset    ;get queue offset
         mov     al,[di+bx]            ;remove character
         pop     di                    ;recover register
         inc     bx                    ;bump pointer
         and     bx,Txq_mask           ;  and mask it
         mov     [si].Txq_tail,bx      ;update pointer
         clc                           ;return carry clear
deq_Tx_data_10:
```

CPA.ASM

```
;-------------------------------------------------------------------------------
;Name:     enq_Sys_msg
;Purpose: Add message to System Processor queue
;Entry:    Nothing
;Exit:     carry set if queue is full, else
;          carry clear


enq_Sys_msg proc
          push    ax                              ;save registers
          push    bx


          mov     bx,offset Sysq          ;BX=> system queue data
          mov     ax,[bx].msgq_head       ;get queue head
          inc     ax                      ;bump pointer
          and     ax,msgq_mask            ;  and mask it
          cmp     ax,[bx].msgq_tail       ;is queue full ?
          stc                             ;(assume it is)
          jz      enq_Sys_msg_10          ;  yes ... exit
          push    cx                      ;save additional registers
          push    si
          push    di
          mov     ax,[bx].msgq_head       ;get queue head again
          mov     cx,msg_len              ;calculate message offset
          mul     cl
          lea     di,[bx].msgq_area
          add     di,ax                   ;got DI
          mov     si,offset msg_area      ;got SI


          rep     movsb                   ;move message queue area
          mov     ax,[bx].msgq_head       ;get queue head again
          inc     ax                      ;bump pointer
          and     ax,msgq_mask            ;  and mask it
          mov     [bx].msgq_head,ax       ;update queue head
          clc                             ;return carry clear
          pop     di
          pop     si
          pop     cx
enq_Sys_msg_10:
          pop     bx
          pop     ax
          ret
enq_Sys_msg endp
```

CPA.ASM

```
;-------------------------------------------------------------------------------
;Name:     deq_Com_msg
;Purpose: Remove message from Communications Processor queue
;Entry:    Nothing
;Exit:     carry set if queue is empty, else
;          carry clear


deq_Com_msg proc
          push    ax                      ;save registers
          push    bx


          mov     bx,offset Comq          ;BX=> comm queue data
          mov     ax,[bx].msgq_tail       ;get queue tail
          cmp     ax,[bx].msgq_head       ;is queue empty ?
          stc                             ;(assume it is)
          jz      deq_Com_msg_10          ;  yes ... exit
          push    cx                      ;save additional registers
          push    si
          push    di
          mov     cx,msg_len              ;calculate message offset
          mul     cl
          lea     si,[bx].msgq_area
          add     si,ax                   ;got SI
          mov     di,offset msg_area      ;got DI


          rep     movsb                   ;move message local area
          mov     ax,[bx].msgq_tail       ;get queue tail again
          inc     ax                      ;bump pointer
          and     ax,msgq_mask            ;  and mask it
          mov     [bx].msgq_tail,ax       ;update queue tail
          pop     di
          pop     si
          pop     cx
deq_Com_msg_10:
          pop     bx
          pop     ax
          ret
deq_Com_msg endp
```

```
;---------------------------------------------------------------------------
;Name:    TBE_isr
;Purpose: Common Transmit Buffer Empty Interrupt Service Routine.
;         This clears the Tx_active flag in the line table to indicate that
;         a character is no longer in the process of being transmitted.
;         To keep the time in the ISR short data writes to the SCC are handled
;         in the main loop.
;Entry:   AX = line table entry
;Exit:    Nothing

TBE_isr  proc
         push    dx                      ; save registers etc.
         push    bx
         push    cx
         push    si
         push    di

         mov     si,ax                   ;si = line table entry
         mov     dx,[si].io_base         ;get base I/O address
         push    dx                      ;save I/O address for isr_ret
         and     [si].line_status,0ffffh-Tx_active    ;inactive

TBE_isr_10:
         mov     al,WR0
         out     dx,al
         mov     al,reset_Tx_int         ;reset pending Tx interrupt
         out     dx,al
TBE_isr_99:
         jmp     isr_ret                 ;common exit ...
TBE_isr endp

;---------------------------------------------------------------------------
;Name:    ESC_isr
;Purpose: Common External Status Change Interrupt Service Routine.  This
;         doesn't do any real work, just demonstrates how to reset and return.
;Entry:   AX = line table entry
;Exit:    Nothing

ESC_isr  proc
         push    dx                      ;save registers etc.
         push    bx
         push    cx
         push    si
         push    di

         mov     si,ax                   ;si = line table entry
         mov     dx,[si].io_base         ;get base I/O address
         push    dx                      ;save I/O address for isr_ret
```



```
;---------------------------------------------------------------------------
;Name:    RCA_isr
;Purpose: Common Receive Character Available Interrupt Service Routine
;Entry:   AX = line table entry
;Exit:    Nothing


RCA_isr  proc
         push    dx                      ;save registers etc.
         push    bx
         push    cx
         push    si
         push    di


         mov     si,ax                   ;si = line table entry
         mov     dx,[si].io_base         ;get base I/O address
         push    dx                      ;save I/O address for isr_ret
         add     dx,2                    ;set up to read data register
         in      al,dx                   ;input character from SCC


         mov     bx,[si].Rxq_tail
         sub     bx,[si].Rxq_head
         dec     bx
         cmp     bx,0
         jge     RCA_isr_30
         add     bx,Rxb_size
RCA_isr_30:
         cmp     bx,1                    ;bx = number of empty spots
         jl      RCA_isr_99              ;if Rx buffer full exit


         mov     bx,[si].Rxq_head        ;get queue head again
         mov     di,[si].Rxq_offset      ;get queue offset
         mov     [di+bx],al              ;add character to queue


         inc     bx                      ;bump head pointer
         and     bx,Rxq_mask             ;  and mask it
         mov     [si].Rxq_head,bx        ;  and update it
RCA_isr_99:
         jmp     isr_ret                 ;common exit ...
RCA_isr endp
```

CPA.ASM

```
;---------------------------------------------------------------------------
;Name:    SRC_isr
;Purpose: Common Special Receive Condition Interrupt Service Routine.  This shows
;         how to get the special receive condition status, but doesn't do any
;         processing on it.
;Entry:   AX = line table entry
;Exit:    Nothing

SRC_isr  proc
         push    dx                     ;save registers etc.
         push    bx
         push    cx
         push    si
         push    di

         mov     si,ax                  ;si = line table entry
         mov     dx,[si].io_base        ;get base I/O address
         push    dx                     ;save I/O address for isr_ret

         mov     al,RR1                 ;allow access to Read Register 1
         out     dx,al
         in      al,dx                  ;get special receive condition status

         ;Do Special Receive Condition processing here

         mov     al,WR0                 ;for insurance
         out     dx,al
         mov     al,error_reset         ;issue error reset command
         out     dx,al

SRC_isr  endp

;---------------------------------------------------------------------------
;Name:    isr_ret
;Purpose: Common Interrupt Service Routine exit processing
;Entry:   DX = SCC base I/O address
;Exit:    To interrupted routine
isr_ret  proc
         pop     dx                     ;get I/O address

         pop     di
         pop     si
         pop     cx
         pop     bx

         mov     al,WR0
         out     dx,al
         mov     al,reset_ius           ;end of interrupt to SCC
         out     dx,al
```

CPA.ASM

```
;---------------------------------------------------------------------------
;Name:    lineXX_TBE
;Purpose: Transmit Buffer Empty Interrupt Service Routine.  There is one of
;         these for each line.  Since each line has identical requirements
;         on TBE, each of these jumps to a common TBE_isr.
;Entry:   Nothing
;Exit:    AX = line table entry

line00_TBE proc                         ;Transmit Buffer Empty
         push    ax                     ;save register
         mov     ax,offset line00       ;   and setup line table offset
         jmp     TBE_isr                ;do common processing ...
line00_TBE endp

line01_TBE proc                         ;Transmit Buffer Empty
         push    ax                     ;save register
         mov     ax,offset line01       ;   and setup line table offset
         jmp     TBE_isr                ;do common processing ...
line01_TBE endp

line02_TBE proc                         ;Transmit Buffer Empty
         push    ax                     ;save register
         mov     ax,offset line02       ;   and setup line table offset
         jmp     TBE_isr                ;do common processing ...
line02_TBE endp

line03_TBE proc                         ;Transmit Buffer Empty
         push    ax                     ;save register
         mov     ax,offset line03       ;   and setup line table offset
         jmp     TBE_isr                ;do common processing ...
line03_TBE endp

line04_TBE proc                         ;Transmit Buffer Empty
         push    ax                     ;save register
         mov     ax,offset line04       ;   and setup line table offset
         jmp     TBE_isr                ;do common processing ...
line04_TBE endp

line05_TBE proc                         ;Transmit Buffer Empty
         push    ax                     ;save register
         mov     ax,offset line05       ;   and setup line table offset
         jmp     TBE_isr                ;do common processing ...
line05_TBE endp

line06_TBE proc                         ;Transmit Buffer Empty
         push    ax                     ;save register
         mov     ax,offset line06       ;   and setup line table offset
         jmp     TBE_isr                ;do common processing ...
line06_TBE endp
```

CPA.ASM

```
;-------------------------------------------------------------------------------
;Name:    lineXX_ESC
;Purpose: External Status Change Interrupt Service Routine.  There is one of
;         these for each line.  Since each line has identical requirements
;         on ESC, each of these jumps to a common ESC_isr.
;Entry:   Nothing
;Exit:    AX = line table entry

line00_ESC proc                         ;External/Status Change
         push    ax                     ;save register
         mov     ax,offset line00       ;   and setup line table offset
         jmp     ESC_isr                ;do common processing ...
line00_ESC endp

line01_ESC proc                         ;External/Status Change
         push    ax                     ;save register
         mov     ax,offset line01       ;   and setup line table offset
         jmp     ESC_isr                ;do common processing ...
line01_ESC endp

line02_ESC proc                         ;External/Status Change
         push    ax                     ;save register
         mov     ax,offset line02       ;   and setup line table offset
         jmp     ESC_isr                ;do common processing ...
line02_ESC endp

line03_ESC proc                         ;External/Status Change
         push    ax                     ;save register
         mov     ax,offset line03       ;   and setup line table offset
         jmp     ESC_isr                ;do common processing ...
line03_ESC endp

line04_ESC proc                         ;External/Status Change
         push    ax                     ;save register
         mov     ax,offset line04       ;   and setup line table offset
         jmp     ESC_isr                ;do common processing ...
line04_ESC endp

line05_ESC proc                         ;External/Status Change
         push    ax                     ;save register
         mov     ax,offset line05       ;   and setup line table offset
         jmp     ESC_isr                ;do common processing ...
line05_ESC endp

line06_ESC proc                         ;External/Status Change
         push    ax                     ;save register
         mov     ax,offset line06       ;   and setup line table offset
         jmp     ESC_isr                ;do common processing ...
line06_ESC endp
```



```
;-------------------------------------------------------------------------------
;Name:    lineXX_RCA
;Purpose: Receive Character Available  Interrupt Service Routine.  There is one
;         of these for each line.  Since each line has identical requirements
;         on RCA, each of these jumps to a common RCA_isr.
;Entry:   Nothing
;Exit:    AX = line table entry

line00_RCA proc                         ;Receive Character Available
         push    ax                     ;save register
         mov     ax,offset line00       ;  and setup interrupt type
         jmp     RCA_isr                ;do common processing ...
line00_RCA endp

line01_RCA proc                         ;Receive Character Available
         push    ax                     ;save register
         mov     ax,offset line01       ;  and setup interrupt type
         jmp     RCA_isr                ;do common processing ...
line01_RCA endp

line02_RCA proc                         ;Receive Character Available
         push    ax                     ;save register
         mov     ax,offset line02       ;  and setup interrupt type
         jmp     RCA_isr                ;do common processing ...
line02_RCA endp

line03_RCA proc                         ;Receive Character Available
         push    ax                     ;save register
         mov     ax,offset line03       ;  and setup interrupt type
         jmp     RCA_isr                ;do common processing ...
line03_RCA endp

line04_RCA proc                         ;Receive Character Available
         push    ax                     ;save register
         mov     ax,offset line04       ;  and setup interrupt type
         jmp     RCA_isr                ;do common processing ...
line04_RCA endp

line05_RCA proc                         ;Receive Character Available
         push    ax                     ;save register
         mov     ax,offset line05       ;  and setup interrupt type
         jmp     RCA_isr                ;do common processing ...
line05_RCA endp

line06_RCA proc                         ;Receive Character Available
         push    ax                     ;save register
         mov     ax,offset line06       ;  and setup interrupt type
         jmp     RCA_isr                ;do common processing ...
line06_RCA endp
```

CPA.ASM

```
;-------------------------------------------------------------------------------
;Name:    lineXX_SRC
;Purpose: Special Receive Condition Interrupt Service Routine.  There is one
;         of these for each line.  Since each line has identical requirements
;         on SRC, each of these jumps to a common SRC_isr.
;Entry:   Nothing
;Exit:    AX = line table entry

line00_SRC proc                         ;Special Receive Condition
         push    ax                     ;save register
         mov     ax,offset line00       ;  and setup interrupt type
         jmp     SRC_isr                ;do common processing ...
line00_SRC endp

line01_SRC proc                         ;Special Receive Condition
         push    ax                     ;save register
         mov     ax,offset line01       ;  and setup interrupt type
         jmp     SRC_isr                ;do common processing ...
line01_SRC endp

line02_SRC proc                         ;Special Receive Condition
         push    ax                     ;save register
         mov     ax,offset line02       ;  and setup interrupt type
         jmp     SRC_isr                ;do common processing ...
line02_SRC endp

line03_SRC proc                         ;Special Receive Condition
         push    ax                     ;save register
         mov     ax,offset line03       ;  and setup interrupt type
         jmp     SRC_isr                ;do common processing ...
line03_SRC endp

line04_SRC proc                         ;Special Receive Condition
         push    ax                     ;save register
         mov     ax,offset line04       ;  and setup interrupt type
         jmp     SRC_isr                ;do common processing ...
line04_SRC endp

line05_SRC proc                         ;Special Receive Condition
         push    ax                     ;save register
         mov     ax,offset line05       ;  and setup interrupt type
         jmp     SRC_isr                ;do common processing ...
line05_SRC endp

line06_SRC proc                         ;Special Receive Condition
         push    ax                     ;save register
         mov     ax,offset line06       ;  and setup interrupt type
         jmp     SRC_isr                ;do common processing ...
line06_SRC endp
```



```
;-------------------------------------------------------------------------------
;Name:    vector_init
;Purpose: Initialize SCC interrupt vectors.  Each vector requires 4 bytes.
;         Since the SCC modifies bits 3, 2, and 1 of the base vector type, but
;         does not modify bit 0, every second vector is unused.  The unused
;         vectors have already been initialized to point to an "invalid
;         interrupt ISR" by the firmware, so they are not altered here.
;Entry:   AX = base vector ype
;Exit:    Nothing
;Registers AX, BX, CX, SI, DI and ES altered

vector_init proc
         shl     ax,1                   ;calculate interrupt vector address
         shl     ax,1
         mov     di,ax
         xor     ax,ax
         mov     es,ax                  ;ES:DI=> destination
         mov     si,offset vector_tbl   ;SI=> vector table
         mov     cx,[si]
         add     si,2
         shr     cx,1                   ;CX = table length (words)
         mov     bx,cs                  ;setup segment address
vector_init_10:
         movsw                          ;move vector offset
         mov     ax,bx
         stosw
         add     di,4                   ;skip unused vector entry
         loop    vector_init_10         ;continue ...
vector_init_99:
         ret
vector_init endp

;-------------------------------------------------------------------------------
;SCC Interrupt Vector Table

vector_tbl       equ     $
         dw      vector_tbl_end-$-2     ;table length

         dw      line01_TBE
         dw      line01_ESC
         dw      line01_RCA
         dw      line01_SRC

         dw      line00_TBE
         dw      line00_ESC
         dw      line00_RCA
         dw      line00_SRC

         dw      line03_TBE
```

```
        dw      line05_TBE
        dw      line05_ESC
        dw      line05_RCA
        dw      line05_SRC

        dw      line04_TBE
        dw      line04_ESC
        dw      line04_RCA
        dw      line04_SRC

line07_entry    label   word
        dw      line07_TBE
        dw      line07_ESC
        dw      line07_RCA
        dw      line07_SRC

        dw      line06_TBE
        dw      line06_ESC
        dw      line06_RCA
        dw      line06_SRC

vector_tbl_end  equ     $
line07_count    equ     (($-line07_entry)/2)


;-------------------------------------------------------------------------------
;Miscellaneous data and stack

        db      'Comtrol HOSTESS 186 Sample Control Program' ,0
        db      'Copyright (C) 1991 Comtrol Corp.  ',0
        db      'All rights reserved.',0

release equ     $
        db      '1.00   ',0
release_len     equ     $-release

msg_area db     16 dup (?)              ;message area

        public  bos
bos     label   word                    ;bottom of stack
        db      512     dup     (?)     ;stack size = 512 bytes
        public  tos
tos     label   word                    ;top of stack


        org     1000h
;-------------------------------------------------------------------------------
        public  Comq,Sysq
```



```
;-------------------------------------------------------------------------------
;Line table, one entry for each line

        public  line00,line01,line02,line03,line04,line05,line06,line07
line00  line_entry      <0004h,0,80h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                         0,0,0,offset line00_Txb,0,0,offset line00_Rxb,\
                         0,0,0,0>
line01  line_entry      <0000h,0,80h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                         0,0,0,offset line01_Txb,0,0,offset line01_Rxb,\
                         0,0,0,0>
line02  line_entry      <0084h,0,90h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                         0,0,0,offset line02_Txb,0,0,offset line02_Rxb,\
                         0,0,0,0>
line03  line_entry      <0080h,0,90h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                         0,0,0,offset line03_Txb,0,0,offset line03_Rxb,\
                         0,0,0,0>
line04  line_entry      <0104h,0,0a0h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                         0,0,0,offset line04_Txb,0,0,offset line04_Rxb,\
                         0,0,0,0>
line05  line_entry      <0100h,0,0a0h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                         0,0,0,offset line05_Txb,0,0,offset line05_Rxb,\
                         0,0,0,0>
line06  line_entry      <0184h,0,0b0h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                         0,0,0,offset line06_Txb,0,0,offset line06_Rxb,\
                         0,0,0,0>
line07  line_entry      <0180h,0,0b0h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                         0,0,0,offset line07_Txb,0,0,offset line07_Rxb,\
                         0,0,0,0>
;-------------------------------------------------------------------------------
;Transmit buffers, one for each line

        public  line00_Txb,line01_Txb,line02_Txb,line03_Txb,line04_Txb
        public  line05_Txb,line06_Txb,line07_Txb
line00_Txb      db      Txb_size dup (?)
line01_Txb      db      Txb_size dup (?)
line02_Txb      db      Txb_size dup (?)
line03_Txb      db      Txb_size dup (?)
line04_Txb      db      Txb_size dup (?)
line05_Txb      db      Txb_size dup (?)
line06_Txb      db      Txb_size dup (?)
line07_Txb      db      Txb_size dup (?)
;-------------------------------------------------------------------------------
;Receive buffers, one for each line

        public  line00_Rxb,line01_Rxb,line02_Rxb,line03_Rxb,line04_Rxb
        public  line05_Rxb,line06_Rxb,line07_Rxb
line00_Rxb      db      Rxb_size dup (?)
line01_Rxb      db      Rxb_size dup (?)
line02_Rxb      db      Rxb_size dup (?)
```

CP.EQU

```
;*****************************************************************************
;File:    cp.equ
;Purpose: Equates for sample control programs for HOSTESS 186.
;Company: Comtrol Corporation
;Release: 1.00, Craig Harrison - Original release.
;Date:    2-18-92
;*****************************************************************************

;Interrupt Controller (PIC) Registers
EOI_VAL          equ     8000h          ;end of interrupt value
INT1_type        equ     0dh            ;system interrupt vector type
INTCTL           equ     0ff22h         ;PIC port
INTCTL1          equ     0ff28h         ;PIC port for initialization command word
TIMER_INT_CTRL   equ     0ff32h         ;timer interrupt control register

;Timer registers
TIMER1_CNT       equ     0ffffh         ;30/sec counter
TIM0_type        equ     08h            ;timer 0 interrupt vector type
TIM1_type        equ     12h            ;timer 1 interrupt vector type
TMR1_CTRL_reg    equ     0ff5eh         ; mode/control register
TMR1_MAX_CNTB    equ     0ff5ch         ; max count B
TMR1_MAX_CNTA    equ     0ff5ah         ; max count A
TMR1_CNT_reg     equ     0ff58h         ; count register
TMR0_CTRL_reg    equ     0ff56h         ; mode/control register
TMR0_MAX_CNTB    equ     0ff54h         ; max count B
TMR0_MAX_CNTA    equ     0ff52h         ; max count A
TMR0_CNT_reg     equ     0ff50h         ; count register


C2Sint_reg       equ     0ef60h         ;COM uP to SYS uP interrupt register
C2Sint_hi        equ     0              ;value to set interrupt line high
C2Sint_low       equ     0008h          ;value to set interrupt line low

Txb_size         equ      512           ;transmit buffer size
Txq_mask         equ      Txb_size-1
Rxb_size         equ      2048          ;receive buffer size
Rxq_mask         equ      Rxb_size-1

base_vector      equ     80h            ;base interrupt vector type


;Message Queue Equates
msg_len          equ     16             ;message length
msgq_size        equ     32             ;number of message queue entries
msgq_mask        equ     msgq_size-1
```



```
;SCC register values
WR2_             db      ?
WR3_             db      ?
WR4_             db      ?
WR5_             db      ?
WR12_            db      ?
WR13_            db      ?
WR15_            db      ?
                 db      ?              ;filler, keeps things on even boundaries


;Transmit queue data
Txq_head         dw      ?
Txq_tail         dw      ?
Txq_offset       dw      ?


;Receive queue data
Rxq_head         dw      ?
Rxq_tail         dw      ?
Rxq_offset       dw      ?


                 dw      ?              ;filler
                 dw      ?              ;filler
                 dw      ?              ;filler
                 dw      ?              ;filler


line_entry       ends
line_entry_len   equ     size line_entry

;-----------------------------------------------------------------------------
;line_status definitions

line_active      equ     0001h          ;line is active
Tx_active        equ     0002h          ;transmit is active (char is going out)


;-----------------------------------------------------------------------------
;Message Queue definition

msgq_entry       struc
msgq_head        dw      ?              ;queue head pointer
msgq_tail        dw      ?              ;queue tail pointer
msgq_area        db      msgq_size*msg_len dup (?)   ;queue buffers
msgq_entry       ends
```

CP.EQU

```
;SCC register equates
WR0       equ     0
WR1       equ     1
WR2       equ     2
WR3       equ     3
WR4       equ     4
WR5       equ     5
WR6       equ     6
WR7       equ     7
WR8       equ     8
WR9       equ     9
WR10      equ     10
WR11      equ     11
WR12      equ     12
WR13      equ     13
WR14      equ     14
WR15      equ     15

RR0       equ     0
RR1       equ     1
RR2       equ     2
RR3       equ     3
RR8       equ     8
RR10      equ     10
RR12      equ     12
RR13      equ     13
RR15      equ     15


; Baud Rate Generator Time Constants - x16 Baud Rate Factor
; (based on a 4.9152 MHz clock)
bps50           equ     3070
bps75           equ     2046
bps110          equ     1394            ; 0.026 percent error
bps134          equ     1140            ; 0.001 percent error
bps150          equ     1022
bps300          equ     510
bps600          equ     254
bps1200         equ     126
bps1800         equ     83              ; 0.401 percent error
bps2000         equ     75              ; 1.06 percent error
bps2400         equ     62
bps3600         equ     41              ; 1.62 percent error
bps4800         equ     30
bps7200         equ     19              ; 1.75 percent error
bps9600         equ     14
bps19200        equ     6
```



```
;*****************************************************************************
;
;Write Register Definitions (for basic asynchronous communications)
;
;*****************************************************************************

;Write Register 0 - command register
reset_ext       equ     10h
reset_Tx_int    equ     28h
error_reset     equ     30h
reset_ius       equ     38h

;Write Register 1 - Tx/Rx interrupt and data transfer mode definition
ext_int_enable  equ     01h
Tx_int_enable   equ     02h
parity_special  equ     04h
Rx_int_enable   equ     10h

;Write Register 2 - interrupt vector

;Write Register 3 - Rx parameters and controls
Rx_enable       equ     01h
Rx5_bit_char    equ     00h
Rx7_bit_char    equ     40h
Rx6_bit_char    equ     80h
Rx8_bit_char    equ     0c0h


;Write Register 4 - Tx/Rx miscellaneous parameters and modes
parity_enable   equ     01h
parity_even     equ     02h
parity_odd      equ     00h
one_stop_bit    equ     04h
one$_stop_bits  equ     08h
two_stop_bits   equ     0ch
x16_clock       equ     40h

;Write Register 5 - Tx parameters and controls
RTS             equ     02h
Tx_enable       equ     08h
BREAK           equ     10h
Tx5_bit_char    equ     00h
Tx7_bit_char    equ     20h
Tx6_bit_char    equ     40h
Tx8_bit_char    equ     60h
DTR             equ     80h

;Write Register 6 - sync character or SDLC address field
```

CP.EQU

```
;Write Register 9 - master interrupt control
VIS             equ     01h
NV              equ     02h
DLC             equ     04h
MIE             equ     08h
status_lo       equ     00h
status_hi       equ     10h
reset_ch_B      equ     40h
reset_ch_A      equ     80h
hardware_reset  equ     0c0h

;Write Register 10 - miscellaneous Tx/Rx control bits

;Write Register 11 - clock mode control
Tx_clk_eq_BRG   equ     10h
Rx_clk_eq_BRG   equ     40h

;Write Register 12 - lower byte of baud generator time constant

;Write Register 13 - upper byte of baud generator time constant

;Write Register 14 - miscellaneous control bits
BRG_enable      equ     01h
BRG_eq_sys_clk  equ     02h

;Write Register 15 - external/status interrupt control
DCD_ie          equ     08h
CTS_ie          equ     20h
break_ie        equ     80h

;*************************************************************************
;
;Read Register Definitions (for basic asynchronous communications)
;
;*************************************************************************

;Read Register 0 - Tx/Rx buffer status and external status
Rx_buffer       equ     01h
Tx_buffer       equ     04h
DCD_            equ     08h
CTS_            equ     20h
break_cond      equ     80h

;Read Register 1 - special receive condition status
parity_error    equ     10h
overrun_error   equ     20h
framing_error   equ     40h

;Read Register 2 - interrupt vector
```

Bibliography

The following references were used as research material for this manual:

Advanced Micro Devices, Inc.,® Z8530/Z85C30 SCC Serial Communications Controller Technical Manual

Intel® iAPX 86/88, 186/188 User's Manual Hardware Reference

Intel® iAPX 86/88, 186/188 User's Manual Programmer's Reference

Intel® 80186/188, 80C186/C188 Hardware Reference Manual.

Zilog Inc.,® Z8030/Z8530 SCC Serial Communications Controller Technical Manual

Glossary

controller	The controller is the Hostess 186 board itself.
control program	The control program is the program downloaded to the controller that controls the I/O.
CS:IP	Code Segment register:Instruction Pointer register location used in debugging.
device driver	A device driver is the code running on the host that interfaces with the I/O device.
dual-ported RAM	Dual-ported RAM is the memory on the Hostess 186 controller. It can be accessed by both the host and the controller.
host	The host is the computer in which the controller is placed.
I/O	I/O stands for input and output.
IRQ	The IRQ is the interrupt vector number used to interrupt the host computer.
SCC	The Serial Communications Controller (SCC) for the Hostess 186 is the AMD AmZ8530 or the Intel 82530 chip.

Index

Developer's License Agreement, Warranty, and Technical Support

Developer's License Agreement

At Comtrol, we want to encourage you to develop software products for our hardware products, so we developed this no-nonsense Developer's License Agreement.

The software supplied with the Programmer's Toolkit is protected by United States copyright law and international copyright treaties. In order for Comtrol to protect its copyrights, we need some limitations on reproduction and distribution, so here they are:

1. The software may only be used to develop software products that will operate with Comtrol brand hardware and software.

2. You may not reproduce nor distribute the source code contained in the Programmer's Toolkit.

3. Any reproduced or modified Programmer's Toolkit software distributed in executable object code form must bear either Comtrol's copyright notice (for example, Copyright 1991, 1992 Comtrol Corporation), or your own copyright notice.

Other than these restrictions, programs that you write using the materials in the Programmer's Toolkit may be used, distributed, modified, or licensed by you as you decide.

Sample programs are provided to help you start programming right away. You may edit, modify, or otherwise incorporate these programs and routines; and you may redistribute and license them for use by your customers without any other license fee or restriction.

Of course, you are solely responsible for your own programming and you agree to hold us harmless from all claims, liability, and damage arising from you own products which include any Comtrol Software. Remember that this software is designed for use only with Comtrol Products. It will not function properly with any other brand of controller.

Limited Warranty

PRODUCT UNTIL YOU HAVE CALLED COMTROL'S CUSTOMER SERVICE DEPARTMENT AND OBTAINED A RETURN AUTHORIZATION NUMBER.

The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to the replacement of defective diskette(s) or documentation and shall not include or extend to any claim for or right to recover any other damages, including but not limited to, loss of profits, data or use of the software, or special, incidental or consequential damages or other similar claims, even if Comtrol has been specifically advised of the possibility of such damages. In no event will Comtrol's liability for any damages to you or any other person ever exceed $500.00, regardless of any form of the claim.

COMTROL CORPORATION SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE, OR ARISING FROM ANY COURSE OF DEALING OR PERFORMANCE BETWEEN YOU AND COMTROL. Specifically, Comtrol makes no representation or warranty that the software is fit for any particular purpose and any implied warranty of merchantability is limited to the ninety-day duration of the Limited Warranty covering the physical diskette(s) and physical documentation only (and not the software) and is otherwise expressly and specifically disclaimed. THE SOFTWARE IS PROVIDED AS IS.

This limited warranty gives you specific legal rights; you may have others which may vary from state to state. Some states do not allow the exclusion of incidental or consequential damages, or the limitation on how long an implied warranty lasts, so some of the above may not apply to you.

Limit of Liability

IN NO EVENT WILL COMTROL BE LIABLE FOR ANY DAMAGES, COST OR EXPENSE, INCLUDING WITHOUT LIMITATION, A LOSS OF PROFIT, USE OR DATA, OR ANY SPECIAL, INDIRECT, DIRECT CONSEQUENTIAL OR INCIDENTAL DAMAGES REGARDLESS OF THE BASIS OF YOUR CLAIM, INCLUDING NEGLIGENCE IN EXCESS OF $500.00.

Miscellaneous

This License and Limited Warranty shall be construed, interpreted and governed by the laws of the State of Minnesota and any action hereunder shall be brought only in Minnesota. If any provision is found void, invalid, or unenforceable it will not affect the validity if the balance of this License and Limited Warranty which shall remain valid and enforceable according to its terms. IF any remedy hereunder is

Replacement

To qualify for replacement under the warranty terms, the original purchaser must follow the procedure outlined below:

1. COMTROL CORPORATION must be notified in writing within thirty (30) days of the date that the defect is discovered. COMTROL CORPORATION will then issue a Return Material Authorization (RMA) Number which the purchaser must include with all correspondence and display on the outside of the shipping container when returning the Product.

2. A written description of the defect together with proof of the purchase date must be shipped with the Product.

3. All Product must be shipped freight and insurance prepaid, in the original shipping container, or in a container providing equal or better protection, with the Return Material Authorization (RMA) Number displayed on the outside of the container in prominent manner. Ship the Product to:

   COMTROL CORPORATION
   2675 Patton Road, Dock D
   St. Paul, Minnesota 55113

COMTROL CORPORATION will return a Product which qualifies under this warranty freight and insurance prepaid. COMTROL CORPORATION will replace Products which do not qualify under the terms of this warranty at the option of the purchaser, in which case the purchaser will pay the cost of repair, and return freight and insurance.

Technical Support

COMTROL CORPORATION provides lifetime support for all its products. If you have questions about your Hostess 186 controller, please call or fax COMTROL at:

Toll free: 1-800-926-6876 (US)

Phone: (612) 631-7654 (US), or (44) 869-323-220 (UK)

FAX: 612-631-8117 (US), or (44) 869-323-211 (UK)

Hostess 186 Programmer's Reference

Table of Contents

Hostess 186 Programmer's Reference

List of Figures and Tables

Before You Begin

This book details the features and functionality of revision A of the Hostess 186 controller, part number HO8600nnA, where nn is 04 for the four-port controller, and 08 for the eight-port controller. A white sticker found on the lower right corner of the solder side of the controller shows the revision.

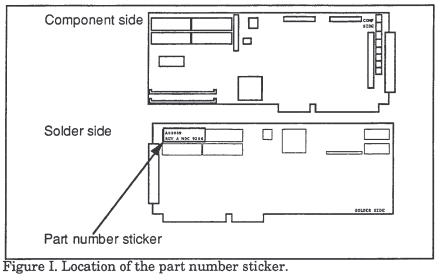


Figure I. Location of the part number sticker.

If your Hostess 186 has a different revision, call COMTROL to receive the corresponding Programmer's Reference.

The purpose of this reference manual is to explain the functionality of the Hostess 186 controller. A diskette titled *Sample Programs* includes examples written in both the C and 80186 assembly languages. If you are not familiar with either language, this reference is not for you. There are several third-party C and assembly programming books available; use those books to increase your technical expertise. Another required book is Advanced Micro Devices' *Z8530/Z85C30 Serial Communications Controller Technical Manual* or Zilog Incorporated's *Z8030/Z8530 SCC Serial Communications Controller Technical Manual*. These books explain the functionality of the Z85C30 serial communications controller, and also include sample programs.

Future Hardware Changes

Hostess 186 Programmer's Reference

"I Never Read Manuals"

If you don't read manuals – you should at least read the "Sample Programs" chapter. After you read this chapter, you can read those chapters in the book that are of interest to you.

This manual contains the following chapters:

Chapter 1: **Hostess 186 Controller Features and Architecture**	Architectural review of the hardware functionality.
Chapter 2: **Sample Programs**	Full listings explanations of the enclosed sample programs.
Chapter 3: **Preview - Initializing the Controller, the Control Registers, and Memory**	An overview of setting up the controller using software.
Chapter 4: **Control Registers Used on the Hostess 186**	How to set up the Hostess 186's control registers.
Chapter 5: **Input/Output Addresses**	Technical descriptions of the I/O addresses you will use.
Chapter 6: **Hostess 186 Dual-port Memory**	Understanding DPRAM.
Chapter 7: **Interrupts**	Setting up interrupts.
Chapter 8: **Timers**	Setting up timers and the values to use.
Chapter 9: **Serial Communication Controller Port Communication**	Technical description of the SCC registers.
Chapter 10:	

Chapter 12: **Using the Hostess 186's Firmware Debugger**	How to use the firmware debugger.
Appendix A: **Assembly Language Listings**	The listing for the assembly language programs.

Read README on the Hostess 186 Sample programs diskette – you should read the ASCII file README on the "Sample Programs" diskette included with your manual. This file lists the latest changes and errata to the sample programs, listings, and this manual.

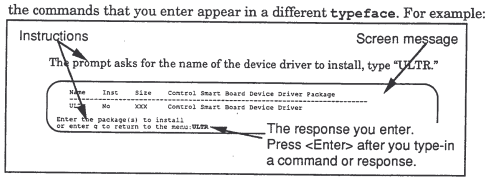
Conventions Used in this Guide

The sample program installation instructions use diagrams to represent what appears on the system administrator's terminal. A screen `typeface` shows what to enter at a prompt. These instructions use a few uniform conventions:

- the menus you use appear boxed in a different typeface. For example:

```
INSTALL REMOVE LIST QUIT                                    CUSTOM
Install software
/                                          Tuesday October 10, 1989  12:

                    Products Currently Installed

  The Operating System
  Software Development System
```

- the commands that you enter appear in a different **`typeface`**. For example:



After you enter a command, select a menu option, or respond to a prompt, always press the <Enter> or <Return> key. If you do not have to press <Enter> or <Return>, the instructions will explicitly state this.

CHAPTER 1 – Hostess 186 Controller Features and Architecture

Hostess 186 Controller Features

The Hostess 186 controller is an intelligent serial communications board with four or eight ports. It has an eight megahertz (MHz), highly-integrated, 16-bit NMOS Intel 80186™ processor. This processor is object code compatible with the Intel 8086 and 8088™ microprocessors.

The 80186 processor has the following integrated components: an 8237 four-channel DMA controller, an 8251 UART communications controller, three 8237 timer-counters, an 8259 interrupt controller, and a refresh controller. The Hostess 186 controller has 128 kilobytes (Kbytes) of dual-ported RAM.

The IRQs available are 3, 4, 5, 9, 10, 11, 12, or 15.

There are four control registers on the Hostess 186. These registers control the addressing, memory window size, interrupts, and mode of operation.

Two programmable timers are available to the control program. The timers can be used to generate interrupts to the control program, at a frequency of 16 to 200 interrupts per second.

The next table summarizes the components and features of the Hostess 186 controller.

Table 1. Hostess 186 components and features.

Components and Features:
• 8 MHz Intel 80186™ processor • Four AMD AmZ8530 or Intel 82530 Serial Communication Controllers, expandable to eight SCCs • 128 Kbytes dual-ported RAM • Switch-definable I/O addresses • Software-definable memory addresses • Software-definable IRQs • Two programmable timers • Software-definable 8- or 16-bit memory transfers

Figures 1 and 2 show the layout of the Hostess 186 four-port and Hostess 186 eight-

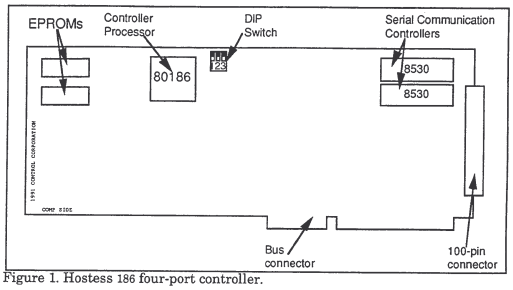


Figure 1. Hostess 186 four-port controller.

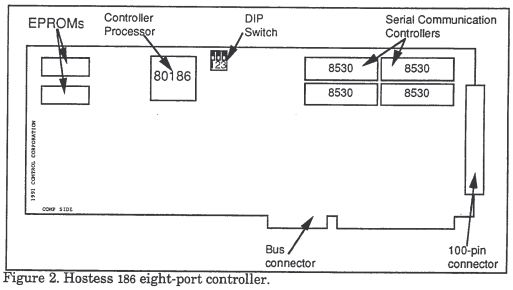


Figure 2. Hostess 186 eight-port controller.

Hostess 186 Architecture

The following block diagram shows the major hardware components of the Hostess 186 controller.

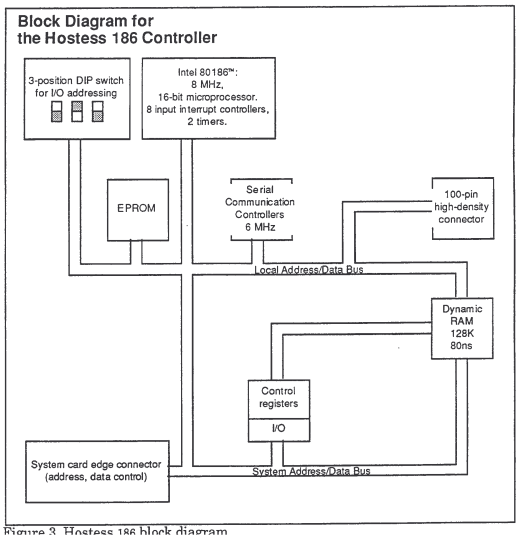


Figure 3. Hostess 186 block diagram

The EPROM block contains 32K bytes mapped at the top of the 80186's one megabyte of memory space. This firmware contains code for the following:

- 80186 bootstrap instruction
- 80186 initialization
- Programmable interrupt controller (PIC) initialization
- Timer initialization
- Interrupt vector initialization
- Interrupt service routines
- Diagnostics for Serial Communication Controllers (SCCs) and memory
- Terminal debugger
- Borland's Turbo Debugger® remote kernel

Although it is possible for users to produce their own EPROMs for the Hostess 186, COMTROL does not recommend it. Instead, users may customize the operation of the Hostess 186 by developing their own control programs and downloading them to the Hostess 186.

The Serial Communication Controllers block contains 8530 Serial Communication Controller chips. These devices implement the four or eight serial ports found on the Hostess 186 controller. The SCCs are mapped into the 80186's I/O address space. For more information about programming the SCCs, please read Advanced Micro Devices' *Z8530/Z85C30 Serial Communications Controller Technical Manual* or Zilog Incorporated's *Z8030/Z8530 SCC Serial Communications Controller Technical Manual*. .

The memory block contains 128K bytes of dual-port RAM mapped at the bottom of the 80186's memory space. A small portion of this memory is reserved for the interrupt vector table and firmware usage. The remainder may be used to customize the Hostess 186 to the user's application. The users accomplish this by developing and downloading their own control programs into the dual-port RAM.

Description:	Starting Address:
unused	10080h
firmware data area	10000h
unused	00c00h
firmware work space	00400h
interrupt vector table	00000h

Figure 4. Hostess 186 standard memory map (as addressed by the local processor).

The I/O block contains functions that can be performed by I/O writes or reads from the system's processor, as follows:

- Write a control register index.
- Write to a control register.
- Enable or disable dual-port RAM.
- Interrupt the 80186.
- Reset the Hostess 186.

The control register block contains the functions that can be performed by I/O writes from the system processor. The control registers are accessed in a two-step process: first an index is written, then the control register value is written. The control register functions are:

- Select the base address of dual-port RAM in the system's memory space.
- Select the size of the system's window into dual-port RAM.
- Select the portion of dual-port RAM visible in the system's window.
- Select the interrupt request line the Hostess 186 uses.

The edge connector block contains the connectors which plug into the system's I/O channel connector. The Hostess 186 may be used in an 8-bit or a 16-bit slot.

The I/O DIP switch box contains the three-position DIP switch you set for the Hostess 186's I/O address.

CHAPTER 2 – Sample Programs

Reading Program Listings

The primary goal of this book is to explain the functionality of the Hostess 186 controller. Included with this book is the *Sample Programs* diskette. This diskette contains the source listings and executable files for a simplified control program model that works on the Hostess 186.

This book uses both C and 80186 assembly language examples in each chapter. Control programs often are a mix both high-level and low-level code. For this reason this chapter, and for most part this entire book, tries to use examples written in both the C and assembly languages.

COMTROL encourages you to use these files on the diskette and examine how the control program works. There are two sets of control programs:
- CPC.EXE – written in the C and the 80186 assembly languages, and
- CPA.COM – written only in 80186 assembly language.

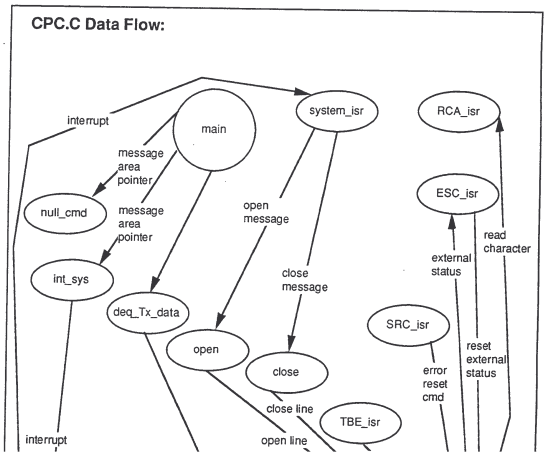
This chapter lists the CPC.EXE source code. Appendix A lists the CPA.COM source code. Both sets are functionally equivalent.

The executable control program model (CPC.EXE or CPA.COM) runs on the Hostess 186. It opens, closes, reads, and writes to any asynchronous line on the controller.

These files make up the CPC.EXE control program model:
- CPC.C – the source code for the control program model.
- CPC.EXE – the executable control program model.
- CPC.H – the header file for CPC.C.
- CPCSTART.ASM – the startup and initialization routines, in assembly code.
- CP.EQU – the equate file for CPCSTART.ASM.
- CPC.TDS – the symbol table (for debugging purposes).
- DPLOADER.C – the source code for the loader program.
- DPLOADER.EXE – the executable loader program.

How the Control Program Works

When the Hostess 186 initially powers up, its processor (the local processor) executes the initialization and diagnostic code out of the firmware. After this process, the system's processor downloads (writes) the control program into dual-port memory (DPM) at the local processor's address 1000:80. Next, the system processor interrupts the local processor, and the firmware's interrupt service routine invokes the control program by jumping to the 1000:80 address. The first section of the control program's code initializes the segment registers, stack, interrupt vectors, timer, and several fields of the firmware user area.



After initializing these data structures, the control program enters an infinite processing loop. The term "line" refers to any one of the eight serial lines (ports) on the Hostess 186 controller. The example programs number the lines from 0 to 7. Each serial line has a line-table entry associated with it.

Table 2. Line table number and corresponding Hostess 186 port.

Line Tables:	Hostess 186 port:
line 0 table	port 1
line 1 table	port 2
line 2 table	port 3
line 3 table	port 4
line 4 table	port 5
line 5 table	port 6
line 6 table	port 7
line 7 table	port 8

The main loop sequentially checks each line's line-table entry, line-status field. If the LINE_ACTIVE bit is not set, processing continues with the next line. If the LINE_ACTIVE bit is set, the line status is checked to see if the TX_ACTIVE bit is set. The TX_ACTIVE bit indicates that the SCC is busy sending a character and it cannot accept another character. If the SCC is free, the main loop calls the `deq_Tx_data` routine to write the next character (if any) from the current line's transmit buffer to the SCC's internal transmit buffer. (Consider these loop operations as background processing. Interrupt service routines (ISRs) handle all other processing in the control program.)

```
void deq_Tx_data(LINE_ENTRY_T *lt_p)
{
   int tail;

   tail = lt_p->Txq_tail;
   if(lt_p->Txq_head == tail)              /* Tx queue empty */
      return;
   lt_p->line_status |= TX_ACTIVE;         /* indicate transmit active */
   outp(lt_p->io_base + 2,*(lt_p->Txq + tail)); /* write char to SCC */
   lt_p->Txq_tail = (tail + 1) & TXQ_MASK;  /* update tail pointer */
}
```

During the control program's initialization phase, an interrupt service routine named `system_isr` replaces the firmware routine that first invoked the control program. `system_isr` is responsible for processing messages sent by the system processor to the control program in the `Comq` buffer. Four messages have been defined: `null_cmd` (does nothing), `open`, `close`, and `int_sys` (an example of


```
system_isr proc
        pusha                           ;save registers

system_isr_02:
        call    deq_Com_msg             ;Comm Processor message waiting ?
        jc      system_isr_20           ;  no ... exit
        mov     bx,offset msg_area
        mov     al,[bx]                 ;get command
        xor     ah,ah                   ;clear upper byte
        shl     ax,1                    ;double the command value
        mov     bx,offset command_tbl   ;BX=> command table
        add     bx,ax                   ;offset into table
        cmp     bx,offset command_tble  ;valid command ?
        jnc     system_isr_10           ;  no ... continue

        mov     ax,offset msg_area     ;parameter is ptr to msg_area
        push    ax                      ;  pass it
        call    word ptr [bx]           ;invoke C command processor
        add     sp,2                    ;remove parameter from stack
system_isr_10:
        jmp     system_isr_02           ;check for another message
system_isr_20:
        mov     dx,INTCTL               ;end of interrupt to PIC
        mov     al,EOI_VAL
        out     dx,al
        popa
        iret
system_isr endp

;-------------------------------------------------------------------------------
;Command table
command_tbl     equ     $
        dw      _null_cmd              ;0 - null command
        dw      _open                  ;1 - open  a line
        dw      _close                 ;2 - close a line
        dw      _int_sys               ;3 - generate interrupt to system
command_tble    equ     $
```

Examples of system-side processing for `open` and `close` are contained in HILIB.ASM, the `hiopen` and `hiclose` routines illustrate the system-side processing for the `open` and `close` messages.

The function `hiopen` opens a serial line on the Hostess 186. After a line has been opened, data may be transmitted to that line. To transmit a character, the system processor writes the character to that line's transmit buffer, using the normal queue operations, as used in the HILIB.ASM, `hiwrite` routine.

The control program's main loop will remove the character from the queue and

```
int TBE_isr(LINE_ENTRY_T *lt_p)
{
   int scc;                              /* SCC command register address */

   scc = lt_p->io_base;                  /* get SCC command register address */
   lt_p->line_status &= 0xfffff - TX_ACTIVE; /* Tx is now inactive */
   outp(scc,WR0);                        /* reset pending Tx interrupt */
   outp(scc,RESET_TX_INT);
   return(scc);
}
```

After a line has been opened, data may also be received from that line. When the SCC receives a serial character, it issues an interrupt to the local processor, which invokes the `RCA_isr` routine for that line. `RCA_isr` reads the character from the SCC and places it in that line's receive buffer queue.

```
int RCA_isr(LINE_ENTRY_T *lt_p)
{
   int scc;                              /* SCC command register address */
   unsigned char ch;                     /* character read from SCC */
   int head;                             /* Rx queue head pointer */
   int num_full;                         /* number Rx queue locations filled */

   scc = lt_p->io_base;                  /* get SCC command register address */
   ch = inp(scc+2);                      /* read character from SCC */
   head = lt_p->Rxq_head;
   if((num_full = head - lt_p->Rxq_tail) < 0)  /* num queue locations full */
      num_full += (RXQ_MASK + 1);                /*   adjust for queue wrap */
   if(num_full < RXQ_MASK)               /* if Rx queue has empty space */
   {
      *(lt_p->Rxq + head) = ch;          /* add received character to queue */
      lt_p->Rxq_head = (head + 1) & RXQ_MASK; /* bump head pointer */
   }
   return(scc);
}
```

The system processor may then remove that character from the queue, as used in HILIB.ASM's `hiread` routine.

The SCC's are also capable of generating interrupts for external status changes or special receive conditions. These interrupts are handled by CPC.C's interrupt service routines `ESC_isr` and `SRC_isr`:

```
/* Function: ESC_isr                                                        */
int ESC_isr(LINE_ENTRY_T *lt_p)
{
   int scc;                              /* SCC command register address */
   unsigned char status;                 /* saves the external status */
   scc = lt_p->io_base;                  /* get SCC command register address */
```



```
/* Function: SRC_isr                                                        */

int SRC_isr(LINE_ENTRY_T *lt_p)
{
   int scc;                              /* SCC command register address */
   unsigned status;                      /* saves the SRC status */

   scc = lt_p->io_base;                  /* get SCC command register address */
   outp(scc,RR1);                        /* read the SRC status */
   status = inp(scc);

   /* Do Special Receive Condition processing here */

   outp(scc,WR0);                        /* for insurance */
   outp(scc,ERROR_RESET);                /*   issue error reset command */
   return(scc);
}
```

The local processor's timer 1 is initialized by the control program to generate an interrupt 30 times a second. These are handled by the interrupt service routine `timer1_isr`, which does nothing except increment the word count found at local address 1000:7C.

```
; Name: timer1_isr

timer1_isr proc
         push    ax                   ;save registers
         push    dx
         inc     word ptr cs:7ch      ;increment word
         mov     dx,INTCTL            ;end of interrupt to PIC
         mov     al,EOI_VAL
         out     dx,al
         pop     dx                   ;recover registers
         pop     ax
         iret
timer1_isr endp
```

Dual-port RAM Configuration for CPC.EXE

Information stored in dual-port memory (DPM) includes:
1. General information about the controller as defined by firmware (the firmware data area)
2. Area for messages for the communications processor (Comq)
3. Area for messages for the system processor (Sysq).
4. Line tables for each of the 16 ports that describe the port.
5. Transmit and receive buffer for each of the 16 lines.

Transmit and receive (circular) buffer sizes:
1. Transmit buffer – 512 bytes.
2. Receive buffer – 2048 bytes.

To write data to the transmit buffer:
1. Write data to the buffer at the head.
2. Update the head pointer in the line table.

To read data from the receive buffer:
1. Read data from the buffer at the tail.
2. Update the tail pointer in the line table

The control programs use only 64K of dual-port memory, beginning at the local processor's address 1000:0. The system processor views this same memory beginning at address D000:0. The following is a map of this area of dual-port memory.

Table 3. 64K Dual-port memory map.

Offset in hex:	Use:	Length in hex bytes:
	Firmware Data Area:	
0	Processor interaction flag	2
2	Boot/activity flag	2
4	Configuration map (obsolete)	2
6	Firmware release number	8
E	Control program release number	8
16	Reserved	4
1A	DRAM map	4
1E	SCC map	4
22	Controller ID	4

Offset in hex:	Use:	Length in hex bytes:
	Communication Message Queue:	
1000	head pointer	2
1002	tail pointer	2
1004	message area	200
	System Message Queue:	
1204	head pointer	2
1208	tail pointer	2
120A	message area	200
1408	filler	8
	Line Tables:	
1410	line 0 table	20
1430	line 1 table	20
1450	line 2 table	20
1470	line 3 table	20
1490	line 4 table	20
14B0	line 5 table	20
14D0	line 6 table	20
14F0	line 7 table	20
	Transmit Buffers:	
1510	line 00	200
1710	line 01	200
1910	line 02	200
1B10	line 03	200
1D10	line 04	200
1F10	line 05	200
2110	line 06	200
2310	line 07	200
	Receive Buffers:	
2510	line 00	800
2D10	line 01	800
3510	line 02	800
3D10	line 03	800
4510	line 04	800

Table 4. Line table map.

Offset in hex:	Use:	Length in hex bytes:
0	SCC base I/O address	2
2	line status	2
4	write register 2 value	1
5	write register 3 value	1
6	write register 4 value	1
7	write register 5 value	1
8	write register 12 value	1
9	write register 13 value	1
A	write register 15 value	1
B	filler, keep pointers on even address	1
C	transmit buffer head pointer	2
E	transmit buffer tail pointer	2
10	transmit buffer DPM offset	2
12	receive buffer head pointer	2
14	receive buffer tail pointer	2
16	receive buffer DPM offset	2
18	filler	8
20	end of line table data	

The pages that follow show the listings for:

- CPC.C,
- CPC.H,
- CPCSTART.ASM,
- CP.EQU,
- DPLOADER.C,
- HILIB.ASM,
- HITERM.C, a terminal emulation program, and
- TSAMPLE.MK, the "make" file for these sample programs.

Read through these listings to get a feel for how a control program works with the controller. You will see some of this code again as examples in the chapters that follow.

CPC.C

```
/*****************************************************************************
 File:    cpc.c
 Purpose: Sample C language control program for the HOSTESS 186.
          Supports open, close, read, and write to any asynchronous line.
 Company: Comtrol Corporation
 Release: 1.00, Craig Harrison - Original release
*****************************************************************************/

#include <dos.h>
#include "cpc.h"

LINE_ENTRY_T *line[16];

/* Function prototypes */
void deq_Tx_data(LINE_ENTRY_T *lt_p);
int TBE_isr(LINE_ENTRY_T *lt_p);
int RCA_isr(LINE_ENTRY_T *lt_p);
int ESC_isr(LINE_ENTRY_T *lt_p);
int SRC_isr(LINE_ENTRY_T *lt_p);
void null_cmd(char *msg_area);
void open(char *msg_area);
void close(char *msg_area);
void int_sys(char *msg_area);

main()
{
   int linenum;
   LINE_ENTRY_T *lt_p;                    /* ptr to line table entry */

   /* Initialize pointers to line tables */
   line[0] = (LINE_ENTRY_T *)0x1410;    line[1] = (LINE_ENTRY_T *)0x1430;
   line[2] = (LINE_ENTRY_T *)0x1450;    line[3] = (LINE_ENTRY_T *)0x1470;
   line[4] = (LINE_ENTRY_T *)0x1490;    line[5] = (LINE_ENTRY_T *)0x14b0;
   line[6] = (LINE_ENTRY_T *)0x14d0;    line[7] = (LINE_ENTRY_T *)0x14f0;

   /* Main processing loop, handles Tx characters only */
   for(linenum = 0;;linenum = (linenum + 1) & 0x07) /* infinite loop */
   {
      lt_p = line[linenum];                /* get ptr to line table entry */
      if(!(lt_p->line_status & LINE_ACTIVE)) /* line active? */
         continue;                         /*    no - move on to next line */
      if(lt_p->line_status & TX_ACTIVE)    /* transmit active? */
         continue;                         /*    yes - move on to next line */
      deq_Tx_data(lt_p);                   /* send char from xmit queue */
   }
}
```

```
/*************************************************************************/
/* Function: deq_Tx_data                                                 */
/* Purpose:  Remove character from transmit queue, write it to SCC.      */
/*           SCC Tx buffer must be empty before calling this function.   */
/* Entry:    lt_p - Pointer to line table entry                          */
/* Exit:     Nothing                                                     */

void deq_Tx_data(LINE_ENTRY_T *lt_p)
{
   int tail;

   tail = lt_p->Txq_tail;
   if(lt_p->Txq_head == tail)              /* Tx queue empty */
      return;
   lt_p->line_status |= TX_ACTIVE;         /* indicate transmit active */
   outp(lt_p->io_base + 2,*(lt_p->Txq + tail)); /* write char to SCC */
   lt_p->Txq_tail = (tail + 1) & TXQ_MASK;  /* update tail pointer */
}
```

```
/*************************************************************************/
/* Function: TBE_isr                                                     */
/* Purpose:  Common Transmit Buffer Empty Interrupt Service Routine.     */
/*           This clears the Tx_active flag in the line table to indicate*/
/*           that a character is no longer in the process of being       */
/*           transmitted.To keep the time in the ISR short data writes   */
/*           to the SCC are handled in the main loop.                    */
/* Entry:    lt_p - Pointer to line table entry                          */
/* Exit:     Returns SCC command register address                        */

int TBE_isr(LINE_ENTRY_T *lt_p)
{
   int scc;                               /* SCC command register address */

   scc = lt_p->io_base;                   /* get SCC command register address */
   lt_p->line_status &= 0xffff - TX_ACTIVE; /* Tx is now inactive */
   outp(scc,WR0);                         /* reset pending Tx interrupt */
   outp(scc,RESET_TX_INT);
   return(scc);
}
```



```
/*************************************************************************/
/* Function: ESC_isr                                                     */
/* Purpose:  Common External Status Change Interrupt Service Routine.  This  */
/*           shows how to get the external status, but doesn't do any    */
/*           processing on it.                                           */
/* Entry:    lt_p - Pointer to line table entry                          */
/* Exit:     Returns SCC command register address                        */

int ESC_isr(LINE_ENTRY_T *lt_p)
{
   int scc;                               /* SCC command register address */
   unsigned char status;                  /* saves the external status */

   scc = lt_p->io_base;                   /* get SCC command register address */
   status = inp(scc);                     /* read the external status */

   /* Do External Status Change processing here */

   outp(scc,WR0);                         /* reset external status interrupts */
   outp(scc,RESET_EXT);
   return(scc);
}
```

```
/*************************************************************************/
/* Function: RCA_isr                                                     */
/* Purpose:  Common Receive Character Available Interrupt Service Routine.   */
/*           Add received character to Rx queue if there is room, else   */
/*           throws it away.                                             */
/* Entry:    lt_p - Pointer to line table entry                          */
/* Exit:     Returns SCC command register address                        */

int RCA_isr(LINE_ENTRY_T *lt_p)
{
   int scc;                               /* SCC command register address */
   unsigned char ch;                      /* character read from SCC */
   int head;                              /* Rx queue head pointer */
   int num_full;                          /* number Rx queue locations filled */

   scc = lt_p->io_base;                   /* get SCC command register address */
   ch = inp(scc+2);                       /* read character from SCC */
   head = lt_p->Rxq_head;
   if((num_full = head - lt_p->Rxq_tail) < 0)  /* num queue locations full */
      num_full += (RXQ_MASK + 1);               /*   adjust for queue wrap */
   if(num_full < RXQ_MASK)                /* if Rx queue has empty space */
```

```
/*************************************************************************/
/* Function: SRC_isr                                                     */
/* Purpose:  Common Special Receive condition Interrupt Service Routine. */
/*           This shows how to get the special receive condition status, */
/*           but doesn't do any processing on it.                        */
/* Entry:    lt_p - Pointer to line table entry                          */
/* Exit:     Returns SCC command register address                        */

int SRC_isr(LINE_ENTRY_T *lt_p)
{
   int scc;                          /* SCC command register address */
   unsigned status;                  /* saves the SRC status */

   scc = lt_p->io_base;              /* get SCC command register address */
   outp(scc,RR1);                    /* read the SRC status */
   status = inp(scc);

   /* Do Special Receive Condition processing here */

   outp(scc,WR0);                    /* for insurance */
   outp(scc,ERROR_RESET);            /*   issue error reset command */
   return(scc);
}
```

```
/*************************************************************************/
/* Function: null_cmd (0)                                                */
/* Purpose:  Handle unimplemented command                                */
/* Entry:    Pointer to message area                                     */
/* Exit:     Nothing                                                     */

void null_cmd(char *msg_area)
{
   ;
}
```



```
/*************************************************************************/
/* Function: open (1)                                                    */
/* Purpose:  Open a line for asynchronous communications                 */
/* Entry:    msg_area+1 = line number                                    */
/*           msg_area+2 = WR3 parameters   (Rx character size)           */
/*           msg_area+3 = WR4 parameters   (stop bits, parity)           */
/*           msg_area+4 = WR5 parameters   (Tx character size)           */
/*           msg_area+5 = WR12 parameter   (lower byte of BRGTC)         */
/*           msg_area+6 = WR13 parameter   (upper byte of BRGTC)         */
/* Exit:     Nothing                                                     */

void open(char *msg_area)
{
   LINE_ENTRY_T *lt_p;                /* ptr to line table entry */
   unsigned char msgb;                /* a byte from msg_area */
   int scc;                           /* SCC command register address */

   lt_p = line[msg_area[1]];          /* get ptr to line table entry */
   scc = lt_p->io_base;               /* get SCC command reg address */

   /* Configure line table entry using message passed from system */
   lt_p->WR3_ = msg_area[2] & 0xc0;     /* WR3, isolate Rx char size */
   lt_p->WR4_ = msg_area[3] & 0x0f;     /* WR4, isolate stop bits/parity*/
   msgb = msg_area[4] & 0x60;          /* WR5, isolate Tx char size */
   lt_p->WR5_ = (lt_p->WR5_ & (0xff-0x60)) | msgb; /* update WR5 Tx char size*/
   lt_p->WR12_ = msg_area[5];           /* WR12, BRGT low byte */
   lt_p->WR13_ = msg_area[6];           /* WR13, BRGT high byte */

   /* Initialize SCC from line table entry */
   outp(scc,WR4);                     /* stop bits/parity + clock mode */
   outp(scc,lt_p->WR4_ | X16_CLOCK);
   outp(scc,WR2);                     /* base interrupt vector type */
   outp(scc,lt_p->WR2_);
   outp(scc,WR3);                     /* Rx character size */
   outp(scc,lt_p->WR3_);
   outp(scc,WR5);                     /* Tx char size, modem/break */
   outp(scc,lt_p->WR5_ | RTS);         /*    make sure RTS is active */
   outp(scc,WR9);                    /* interrupt control */
   outp(scc,STATUS_LO+VIS);
   outp(scc,WR11);                    /* clock sources */
   outp(scc,RX_CLK_EQ_BRG+TX_CLK_EQ_BRG);
   outp(scc,WR12);                    /* lower byte of BRGTC */
   outp(scc,lt_p->WR12_);
   outp(scc,WR13);                    /* upper byte of BRGTC */
   outp(scc,lt_p->WR13_);
   outp(scc,WR14);                    /* BRG source */
   outp(scc,BRG_EQ_SYS_CLK+BRG_ENABLE);
   outp(scc,WR3);                     /* enable receive */
   outp(scc,lt_p->WR3_ | RX_ENABLE);
   outp(scc,WR5);                     /* enable transmit */
```

```
   /* Enable interrupts */
   outp(scc,WR15);                          /* external/status interrupt control */
   outp(scc,lt_p->WR15_);
   outp(scc,WR0);              /* for insurance reset external status interrupts */
   outp(scc,RESET_EXT);
   outp(scc,WR1);                           /* interrupt enables */
   outp(scc,EXT_INT_ENABLE+TX_INT_ENABLE+PARITY_SPECIAL+RX_INT_ENABLE);
   outp(scc,WR9);                           /* master interrupt enable */
   outp(scc,MIE+STATUS_LO+VIS);
}


/*****************************************************************************/
/* Function: close (2)                                                       */
/* Purpose:  Close a serial line                                             */
/* Entry:    msg_area+1 = line number                                        */
/* Exit:     Nothing                                                         */

void close(char *msg_area)
{
   LINE_ENTRY_T *lt_p;                      /* ptr to line table entry */
   int scc;                                 /* SCC command register address */

   lt_p = line[msg_area[1]];                /* get ptr to line table entry */
   scc = lt_p->io_base;                     /* get SCC command reg address */

   outp(scc,WR1);                           /* disable all interrupts */
   outp(scc,0);
   lt_p->line_status = 0;                   /* disable all in line_status */
}


/*****************************************************************************/
/* Function: int_sys (3)                                                     */
/* Purpose:  Generate an interrupt to the system.  This would probably never */
/*           be executed as a command from the system by is included here to */
/*           show how to interrupt the system.                               */
/* Entry:    Pointer to message area                                         */
/* Exit:     Nothing                                                         */

void int_sys(char *msg_area)
{
   outp(C2SINT_REG,C2SINT_LOW);        /* set interrupt line low */
   outp(C2SINT_REG,C2SINT_HI);         /* set interrupt line high  */
}
```

CPC.H

```
/*****************************************************************************
 File:     cpc.c
 Purpose: Header file for sample C language control program for the HOSTESS i
                and HOSTESS 186.
 Company: Comtrol Corporation
 Release: 1.00, Craig Harrison - Original release
 Date:    8-23-91
*****************************************************************************/

/* ----- Line Table data structure ---------------------------------------- */
/* This matches the assembly language structure line_entry in cp.equ */

struct line_entry
{
   int io_base;                             /* SCC base I/O address */
   unsigned line_status;                    /* line status (defined below) */

   /* SCC register values */
   unsigned char WR2_;                      /* write register 2 */
   unsigned char WR3_;                      /* write register 2 */
   unsigned char WR4_;                      /* write register 2 */
   unsigned char WR5_;                      /* write register 2 */
   unsigned char WR12_;                     /* write register 2 */
   unsigned char WR13_;                     /* write register 2 */
   unsigned char WR15_;                     /* write register 2 */
   unsigned char fill1;                     /* filler for word allignment */

   /* Transmit queue data */
   int Txq_head;
   int Txq_tail;
   char *Txq;

   /* Receive queue data */
   int Rxq_head;
   int Rxq_tail;
   char *Rxq;
};
typedef struct line_entry LINE_ENTRY_T;


/* ----- line_status definitions ------------------------------------------ */
#define LINE_ACTIVE 0x0001       /* line is active */
#define TX_ACTIVE   0x0002       /* transmit is active (char is going out) */
```

```
/* ----- SCC register defines ------------------------------------------------ */
#define WR0      0
#define WR1      1
#define WR2      2
#define WR3      3
#define WR4      4
#define WR5      5
#define WR6      6
#define WR7      7
#define WR8      8
#define WR9      9
#define WR10     10
#define WR11     11
#define WR12     12
#define WR13     13
#define WR14     14
#define WR15     15

#define RR0      0
#define RR1      1
#define RR2      2
#define RR3      3
#define RR8      8
#define RR10     10
#define RR12     12
#define RR13     13
#define RR15     15

/* ----- Baud Rate Generator Time Constants - x16 Baud Rate Factor ------- */
/* (based on a 4.9152 MHz clock) */
#define BPS50           3070
#define BPS75           2046
#define BPS110          1394            ; 0.026 percent error
#define BPS134          1140            ; 0.001 percent error
#define BPS150          1022
#define BPS300          510
#define BPS600          254
#define BPS1200         126
#define BPS1800         83              ; 0.401 percent error
#define BPS2000         75              ; 1.06 percent error
#define BPS2400         62
#define BPS3600         41              ; 1.62 percent error
#define BPS4800         30
#define BPS7200         19              ; 1.75 percent error
#define BPS9600         14
#define BPS19200        6
#define BPS38400        2
#define BPS56000        1               ; 74.3 percent error
#define BPS76800        0
```

CPC.H

```
/* Write Register 1 - Tx/Rx interrupt and data transfer mode definition */
#define EXT_INT_ENABLE    0x01
#define TX_INT_ENABLE     0x02
#define PARITY_SPECIAL    0x04
#define RX_INT_ENABLE     0x10

/* Write Register 2 - interrupt vector */

/* Write Register 3 - Rx parameters and controls */
#define RX_ENABLE         0x01
#define RX5_BIT_CHAR      0x00
#define RX7_BIT_CHAR      0x40
#define RX6_BIT_CHAR      0x80
#define RX8_BIT_CHAR      0x0c0


/* Write Register 4 - Tx/Rx miscellaneous parameters and modes */
#define PARITY_ENABLE     0x01
#define PARITY_EVEN       0x02
#define PARITY_ODD        0x00
#define ONE_STOP_BIT      0x04
#define ONE5_STOP_BITS    0x08
#define TWO_STOP_BITS     0x0c
#define X16_CLOCK         0x40

/* Write Register 5 - Tx parameremeters and controls */
#define RTS               0x02
#define TX_ENABLE         0x08
#define BREAK             0x10
#define Tx5_BIT_CHAR      0x00
#define Tx7_BIT_CHAR      0x20
#define Tx6_BIT_CHAR      0x40
#define Tx8_BIT_CHAR      0x60
#define DTR               0x80

/* Write Register 6 - sync character or SDLC address field */

/* Write Register 7 - sync character or SDLC flag */

/* Write Register 8 - transmit buffer */

/* Write Register 9 - master interrupt control */
#define VIS               0x01
#define NV                0x02
#define DLC               0x04
#define MIE               0x08
#define STATUS_LO         0x00
#define STATUS_HI         0x10
#define RESET_CH_B        0x40
#define RESET_CH_A        0x80
```

```
/* Write Register 12 - lower byte of baud generator time constant */

/* Write Register 13 - upper byte of baud generator time constant */

/* Write Register 14 - miscellaneous control bits */
#define BRG_ENABLE       0x01
#define BRG_EQ_SYS_CLK   0x02

/* Write Register 15 - external/STATUS interrupt control */
#define DCD_IE           0x08
#define CTS_IE           0x20
#define BREAK_IE         0x80

/* ----- Read Register Definitions (for basic asynCHronous communications) */
/* Read Register 0 - Tx/Rx BUFFER STATUS and external STATUS */
#define RX_BUFFER        0x01
#define TX_BUFFER        0x04
#define DCD_             0x08
#define CTS_             0x20
#define BREAK_COND       0x80

/* Read Register 1 - special receive condition STATUS */
#define PARITY_ERROR     0x10
#define OVERRUN_ERROR    0x20
#define FRAMING_ERROR    0x40

/* Read Register 2 - interrupt vector */

/* Read Register 3 - interrupt pending STATUS */

/* Read Register 8 - receive data register */

/* Read Register 10 - miscellaneous STATUS bits */

/* Read Register 12 - value stored in WR12 */

/* Read Register 13 - value stored in WR13 */

/* Read Register 15 - value stored in WR15 */


/* ----- Other defines ------------------------------------------------------ */
#define TXQ_MASK    512-1         /* Tx queue pointer mask */
#define RXQ_MASK    2048-1        /* Rx queue pointer mask */
#define C2SINT_REG  0xef60        /* COM uP to SYS uP interrupt register */
#define C2SINT_HI   0             /* value to set interrupt line high */
#define C2SINT_LOW  0x0008        /* value to set interrupt line low  */
```

CPCSTART.ASM

```
        page    60, 80
;*******************************************************************************
;File:    cpcstart.asm
;Purpose: Startup code for C language control program for HOSTESS 186.
;Company: Comtrol Corporation
;Release: 1.00, Craig Harrison - Original release
;*******************************************************************************

.186
.MODEL tiny      ;Must use the simplified segment directives with tiny
                 ;model.  This causes C language global variables to
                 ;have offsets relative to the start of the _TEXT segment.
                 ;Otherwise these variables are given offsets relative to
                 ;DGROUP, which is incorrect because we are initializing
                 ;DS to CS, which is _TEXT, not DGROUP.
                 ;
                 ;The technique used here is to keep everything in the
                 ;_TEXT segment so that no fixups are needed. This way
                 ;the .EXE header can be thrown away on download without
                 ;doing the normal .EXE file relocation.

        include cp.equ

        assume  cs:_TEXT,ds:_TEXT

        extrn   _main:near
        extrn   _null_cmd:near, _open:near, _close:near, _int_sys:near
        extrn   _TBE_isr:near, _ESC_isr:near, _RCA_isr:near, _SRC_isr:near

.CODE

                 org 0h
;The first 80h bytes are the "firmware user area" defined by the HOSTESS 186
;firmware

        public  i_flag
i_flag          dw      ?               ;processor interaction flag
boot_flag       dw      ?               ;boot/activity flag
cfg_map         dw      ?               ;configuration map
fw_release      db      8 dup (?)       ;firmware release number
sw_release      db      8 dup (?)       ;control program release number
                dd      ?               ;reserved
dram_map        dd      ?               ;DRAM map
scc_map         dd      ?               ;SCC map
board_id        dd      ?               ;board ID
ii_flag         db      ?               ;invalid interrupt flag
ii_type         db      ?               ;invalid interrupt type
```

```
;Initialize segment registers, interrupt vectors, etc.
start:   cli                          ;disable interrupts while initializing them
         mov     ax,cs
         mov     ds,ax                ;set up data segment
         mov     ss,ax                ;  and stack segment
         mov     sp,offset tos        ;  and stack pointer

         xor     ax,ax
         mov     es,ax
         mov     bx,INT1_type*4       ;setup system interrupt vector
         mov     ax,offset system_isr
         mov     es:[bx],ax
         mov     ax,cs
         mov     es:[bx+2],ax

         mov     bx,TIM1_type*4       ;setup timer 1 intr vector
         mov     ax,offset timer1_isr
         mov     es:[bx],ax
         mov     ax,cs
         mov     es:[bx+2],ax

         mov     ax,TIMER_INT_CTRL    ;write to interrupt timer cotrol reg.
         mov     dx,ax
         mov     ax,0000h             ;allow interrupts for timer 1.
         out     dx,ax


         mov     dx,TMR1_MAX_CNTA
         mov     ax,TIMER1_CNT        ;30/sec
         out     dx,ax
         mov     dx,TMR1_CTRL_reg
         mov     ax,0e001h            ;enable timer 1 max count A
         out     dx,ax

         mov     ax,base_vector       ;AX = base vector type
         call    vector_init          ;initialize vector table

         mov     ax,ds
         mov     es,ax                ;set up extra segment
         mov     di,offset i_flag     ;DI=> interaction flag
         mov     [di],55aah           ;restore interaction flag

         mov     di,offset boot_flag  ;indicate control program active
         mov     [di],0ffffh

         mov     di,offset sw_release ;move software release number
         mov     si,offset release    ;  to shared memory
         mov     cx,release_len
         rep     movsb
```

CPCSTART.ASM

```
;---------------------------------------------------------------------------
; Name: timer1_isr
; Purpose: Process interrupt from timer 1.  Doesn't do anything useful, just
;          increments a word in dual port RAM to demonstrate that its working.
; Entry: Nothing
; Exit:  Nothing

timer1_isr proc
          push    ax                  ;save registers
          push    dx
          inc     word ptr cs:7ch     ;increment word
          mov     dx,INTCTL           ;end of interrupt to PIC
          mov     ax,EOI_VAL
          out     dx,ax
          pop     dx                  ;recover registers
          pop     ax
          iret
timer1_isr endp


;---------------------------------------------------------------------------
;Name:    system_isr
;Purpose: Process interrupt from sytem processor
;Entry:   Nothing
;Exit:    Nothing

system_isr proc
          pusha                       ;save registers

system_isr_02:
          call    deq_Com_msg         ;Comm Processor message waiting ?
          jc      system_isr_20       ;  no ... exit
          mov     bx,offset msg_area
          mov     al,[bx]             ;get command
          xor     ah,ah               ;clear upper byte
          shl     ax,1                ;double the command value
          mov     bx,offset command_tbl  ;BX=> command table
          add     bx,ax               ;offset into table
          cmp     bx,offset command_tble ;valid command ?
          jnc     system_isr_10       ;  no ... continue

          mov     ax,offset msg_area  ;parameter is ptr to msg_area
          push    ax                  ;  pass it
          call    word ptr [bx]       ;invoke C command processor
          add     sp,2                ;remove parameter from stack
system_isr_10:
          jmp     system_isr_02       ;check for another message
system_isr_20:
          mov     dx,INTCTL           ;end of interrupt to PIC
          mov     ax,EOI_VAL
```

```
;-------------------------------------------------------------------------------
;Command table
command_tbl     equ     $
        dw      _null_cmd               ;0 - null command
        dw      _open                   ;1 - open  a line
        dw      _close                  ;2 - close a line
        dw      _int_sys                ;3 - generate interrupt to system
command_tble    equ     $


;-------------------------------------------------------------------------------
;Name:    enq_Sys_msg
;Purpose: Add message to System Processor queue
;Entry:   Nothing
;Exit:    carry set if queue is full, else
;         carry clear

enq_Sys_msg proc
        push    ax                              ;save registers
        push    bx

        mov     bx,offset Sysq          ;BX=> system queue data
        mov     ax,[bx].msgq_head       ;get queue head
        inc     ax                      ;bump pointer
        and     ax,msgq_mask            ;  and mask it
        cmp     ax,[bx].msgq_tail       ;is queue full ?
        stc                             ;(assume it is)
        jz      enq_Sys_msg_10          ;  yes ... exit
        push    cx                      ;save additional registers
        push    si
        push    di
        mov     ax,[bx].msgq_head       ;get queue head again
        mov     cx,msg_len              ;calculate message offset
        mul     cl
        lea     di,[bx].msgq_area
        add     di,ax                   ;got DI
        mov     si,offset msg_area      ;got SI

        rep     movsb                   ;move message queue area
        mov     ax,[bx].msgq_head       ;get queue head again
        inc     ax                      ;bump pointer
        and     ax,msgq_mask            ;  and mask it
        mov     [bx].msgq_head,ax       ;update queue head
        clc                             ;return carry clear
        pop     di
        pop     si
        pop     cx
enq_Sys_msg_10:
        pop     bx
```



```
;-------------------------------------------------------------------------------
;Name:    deq_Com_msg
;Purpose: Remove message from Communications Processor queue
;Entry:   Nothing
;Exit:    carry set if queue is empty, else
;         carry clear

deq_Com_msg proc
        push    ax                      ;save registers
        push    bx

        mov     bx,offset Comq          ;BX=> comm queue data
        mov     ax,[bx].msgq_tail       ;get queue tail
        cmp     ax,[bx].msgq_head       ;is queue empty ?
        stc                             ;(assume it is)
        jz      deq_Com_msg_10          ;  yes ... exit
        push    cx                      ;save additional registers
        push    si
        push    di
        mov     cx,msg_len              ;calculate message offset
        mul     cl
        lea     si,[bx].msgq_area
        add     si,ax                   ;got SI
        mov     di,offset msg_area      ;got DI

        rep     movsb                   ;move message local area
        mov     ax,[bx].msgq_tail       ;get queue tail again
        inc     ax                      ;bump pointer
        and     ax,msgq_mask            ;  and mask it
        mov     [bx].msgq_tail,ax       ;update queue tail
        pop     di
        pop     si
        pop     cx
deq_Com_msg_10:
        pop     bx
        pop     ax
        ret
deq_Com_msg endp
```

```
;-------------------------------------------------------------------------------
;Name:    isr_ret
;Purpose: Common Interrupt Service Routine exit processing
;Entry:   AX = SCC base I/O address
;Exit:    To interrupted routine

isr_ret  proc
         mov     dx,ax                  ;get I/O address
         mov     al,WR0
         out     dx,al
         mov     al,reset_ius           ;end of interrupt to SCC
         out     dx,al

         mov     dx,INTCTL              ;end of interrupt to PIC
         mov     ax,EOI_VAL
         out     dx,ax

         popa
         iret
isr_ret  endp


;-------------------------------------------------------------------------------
;Name:    lineXX_TBE
;Purpose: Transmit Buffer Empty Interrupt Service Routine.  There is one of
;         these for each line.  Since each line has identical requirements
;         on TBE, each of these calls a common TBE_isr C language function.
;Entry:   Nothing
;Exit:    To isr_ret with AX = SCC base address (returned from TBE_isr)

line00_TBE proc                         ;Transmit Buffer Empty
         pusha                          ;save registers
         mov     ax,offset _line00      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _TBE_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line00_TBE endp

line01_TBE proc                         ;Transmit Buffer Empty
         pusha                          ;save registers
         mov     ax,offset _line01      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _TBE_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line01_TBE endp

line02_TBE proc                         ;Transmit Buffer Empty
         pusha                          ;save registers
```



```
line03_TBE proc                         ;Transmit Buffer Empty
         pusha                          ;save registers
         mov     ax,offset _line03      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _TBE_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line03_TBE endp

line04_TBE proc                         ;Transmit Buffer Empty
         pusha                          ;save registers
         mov     ax,offset _line04      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _TBE_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line04_TBE endp

line05_TBE proc                         ;Transmit Buffer Empty
         pusha                          ;save registers
         mov     ax,offset _line05      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _TBE_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line05_TBE endp

line06_TBE proc                         ;Transmit Buffer Empty
         pusha                          ;save registers
         mov     ax,offset _line06      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _TBE_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line06_TBE endp

line07_TBE proc                         ;Transmit Buffer Empty
         pusha                          ;save registers
         mov     ax,offset _line07      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _TBE_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line07_TBE endp
```

```
;-------------------------------------------------------------------------------
;Name:    lineXX_ESC
;Purpose: External Status Change Interrupt Service Routine.  There is one of
;         these for each line.  Since each line has identical requirements
;         on ESC, each of these calls a common ESC_isr C language function.
;Entry:   Nothing
;Exit:    To isr_ret with AX = SCC base address (returned from ESC_isr)

line00_ESC proc                         ;External/Status Change
         pusha                          ;save registers
         mov     ax,offset _line00      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _ESC_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line00_ESC endp

line01_ESC proc                         ;External/Status Change
         pusha                          ;save registers
         mov     ax,offset _line01      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _ESC_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line01_ESC endp

line02_ESC proc                         ;External/Status Change
         pusha                          ;save registers
         mov     ax,offset _line02      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _ESC_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line02_ESC endp

line03_ESC proc                         ;External/Status Change
         pusha                          ;save registers
         mov     ax,offset _line03      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _ESC_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line03_ESC endp

line04_ESC proc                         ;External/Status Change
         pusha                          ;save registers
         mov     ax,offset _line04      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _ESC_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
```

CPCSTART.ASM

```
line05_ESC proc                         ;External/Status Change
         pusha                          ;save registers
         mov     ax,offset _line05      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _ESC_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line05_ESC endp

line06_ESC proc                         ;External/Status Change
         pusha                          ;save registers
         mov     ax,offset _line06      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _ESC_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line06_ESC endp

line07_ESC proc                         ;External/Status Change
         pusha                          ;save registers
         mov     ax,offset _line07      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _ESC_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line07_ESC endp



;-------------------------------------------------------------------------------
;Name:    lineXX_RCA
;Purpose: Receive Character Available  Interrupt Service Routine.  There is one
;         of these for each line.  Since each line has identical requirements
;         on RCA, each of these calls a common RCA_isr C language function.
;Entry:   Nothing
;Exit:    To isr_ret with AX = SCC base address (returned from RCA_isr)

line00_RCA proc                         ;Receive Character Available
         pusha                          ;save registers
         mov     ax,offset _line00      ;line table offset is parameter
         push    ax                     ;  pass it
         call    _RCA_isr               ;do common processing in C
         add     sp,2                   ;remove parameter from stack
         jmp     isr_ret                ;do common exit processing
line00_RCA endp

line01_RCA proc                         ;Receive Character Available
```

```
line02_RCA proc                         ;Receive Character Available
        pusha                           ;save registers
        mov     ax,offset _line02       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _RCA_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line02_RCA endp

line03_RCA proc                         ;Receive Character Available
        pusha                           ;save registers
        mov     ax,offset _line03       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _RCA_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line03_RCA endp

line04_RCA proc                         ;Receive Character Available
        pusha                           ;save registers
        mov     ax,offset _line04       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _RCA_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line04_RCA endp

line05_RCA proc                         ;Receive Character Available
        pusha                           ;save registers
        mov     ax,offset _line05       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _RCA_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line05_RCA endp

line06_RCA proc                         ;Receive Character Available
        pusha                           ;save registers
        mov     ax,offset _line06       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _RCA_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line06_RCA endp

line07_RCA proc                         ;Receive Character Available
        pusha                           ;save registers
        mov     ax,offset _line07       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _RCA_isr                ;do common processing in C
```

CPCSTART.ASM

```
;-------------------------------------------------------------------------------
;Name:    lineXX_SRC
;Purpose: Special Receive Condition Interrupt Service Routine.  There is one
;         of these for each line.  Since each line has identical requirements
;         on SRC, each of these calls a common SRC_isr C language function.
;Entry:   Nothing
;Exit:    To isr_ret with AX = SCC base address (returned from SRC_isr)

line00_SRC proc                         ;Special Receive Condition
        pusha                           ;save registers
        mov     ax,offset _line00       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _SRC_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line00_SRC endp

line01_SRC proc                         ;Special Receive Condition
        pusha                           ;save registers
        mov     ax,offset _line01       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _SRC_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line01_SRC endp

line02_SRC proc                         ;Special Receive Condition
        pusha                           ;save registers
        mov     ax,offset _line02       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _SRC_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line02_SRC endp

line03_SRC proc                         ;Special Receive Condition
        pusha                           ;save registers
        mov     ax,offset _line03       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _SRC_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line03_SRC endp

line04_SRC proc                         ;Special Receive Condition
        pusha                           ;save registers
        mov     ax,offset _line04       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _SRC_isr                ;do common processing in C
```

```
line05_SRC proc                         ;Special Receive Condition
        pusha                           ;save registers
        mov     ax,offset _line05       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _SRC_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line05_SRC endp

line06_SRC proc                         ;Special Receive Condition
        pusha                           ;save registers
        mov     ax,offset _line06       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _SRC_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line06_SRC endp

line07_SRC proc                         ;Special Receive Condition
        pusha                           ;save registers
        mov     ax,offset _line07       ;line table offset is parameter
        push    ax                      ;  pass it
        call    _SRC_isr                ;do common processing in C
        add     sp,2                    ;remove parameter from stack
        jmp     isr_ret                 ;do common exit processing
line07_SRC endp

;-------------------------------------------------------------------------------
;Name:    vector_init
;Purpose: Initialize SCC interrupt vectors.  Each vector requires 4 bytes.
;         Since the SCC modifies bits 3, 2, and 1 of the base vector type, but
;         does not modify bit 0, every second vector is unused.  The unused
;         vectors have already been initialized to point to an "invalid
;         interrupt ISR" by the firmware, so they are not altered here.
;Entry:   AX = base vector ype
;Exit:    Nothing
;Registers AX, BX, CX, SI, DI and ES altered

vector_init proc
        shl     ax,1                    ;calculate interrupt vector address
        shl     ax,1
        mov     di,ax
        xor     ax,ax
        mov     es,ax                   ;ES:DI=> destination
        mov     si,offset vector_tbl    ;SI=> vector table
        mov     cx,[si]
        add     si,2
        shr     cx,1                    ;CX = table length (words)
        mov     bx,cs                   ;setup segment address
vector_init_10:
```



```
;-------------------------------------------------------------------------------
;SCC Interrupt Vector Table

vector_tbl      equ     $
        dw      vector_tbl_end-$-2      ;table length

        dw      line01_TBE
        dw      line01_ESC
        dw      line01_RCA
        dw      line01_SRC

        dw      line00_TBE
        dw      line00_ESC
        dw      line00_RCA
        dw      line00_SRC

        dw      line03_TBE
        dw      line03_ESC
        dw      line03_RCA
        dw      line03_SRC

        dw      line02_TBE
        dw      line02_ESC
        dw      line02_RCA
        dw      line02_SRC

        dw      line05_TBE
        dw      line05_ESC
        dw      line05_RCA
        dw      line05_SRC

        dw      line04_TBE
        dw      line04_ESC
        dw      line04_RCA
        dw      line04_SRC

line07_entry    label   word
        dw      line07_TBE
        dw      line07_ESC
        dw      line07_RCA
        dw      line07_SRC

        dw      line06_TBE
        dw      line06_ESC
        dw      line06_RCA
        dw      line06_SRC

vector_tbl_end  equ     $
line07_count    equ     (($-line07_entry)/2)
```

```
;---------------------------------------------------------------------------
;Miscellaneous data

        db      'Comtrol HOSTESS 186 Sample Control Program' ,0
        db      'Copyright (C) 1991 Comtrol Corp.  ',0
        db      'All rights reserved.',0

release equ     $
        db      '1.00   ',0
release_len     equ     $-release


        dw      256 dup (?)             ;stack and heap
        public  tos
tos     label   word                    ;top of stack


        org     0ff0h
msg_area db     16 dup (?)              ;message area
;---------------------------------------------------------------------------
        public  Comq,Sysq
;Comq - Queue for messages from System Processor to Communications Processor
Comq    msgq_entry      <>

;Sysq - Queue for messages from Communications Processor to Systems Processor
Sysq    msgq_entry      <>

        dw      4 dup (?)           ;filler

;---------------------------------------------------------------------------
;Line table, one entry for each line

        public  _line00,_line01,_line02,_line03,_line04,_line05,_line06,_line07
_line00  line_entry     <0004h,0,80h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                        0,0,0,offset line00_Txb,0,0,offset line00_Rxb,\
                        0,0,0,0>
_line01  line_entry     <0000h,0,80h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                        0,0,0,offset line01_Txb,0,0,offset line01_Rxb,\
                        0,0,0,0>
_line02  line_entry     <0084h,0,90h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                        0,0,0,offset line02_Txb,0,0,offset line02_Rxb,\
                        0,0,0,0>
_line03  line_entry     <0080h,0,90h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                        0,0,0,offset line03_Txb,0,0,offset line03_Rxb,\
                        0,0,0,0>
_line04  line_entry     <0104h,0,0a0h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                        0,0,0,offset line04_Txb,0,0,offset line04_Rxb,\
                        0,0,0,0>
_line05  line_entry     <0100h,0,0a0h,0c0h,04h,60h,bps9600,bps9600 shr 8,0,\
                        0,0,0,offset line05_Txb,0,0,offset line05_Rxb,\
```

CPCSTART.ASM

```
;---------------------------------------------------------------------------
;Transmit buffers, one for each line

        public  line00_Txb,line01_Txb,line02_Txb,line03_Txb,line04_Txb
        public  line05_Txb,line06_Txb,line07_Txb
line00_Txb      db      Txb_size dup (?)
line01_Txb      db      Txb_size dup (?)
line02_Txb      db      Txb_size dup (?)
line03_Txb      db      Txb_size dup (?)
line04_Txb      db      Txb_size dup (?)
line05_Txb      db      Txb_size dup (?)
line06_Txb      db      Txb_size dup (?)
line07_Txb      db      Txb_size dup (?)

;---------------------------------------------------------------------------
;Receive buffers, one for each line

        public  line00_Rxb,line01_Rxb,line02_Rxb,line03_Rxb,line04_Rxb
        public  line05_Rxb,line06_Rxb,line07_Rxb
line00_Rxb      db      Rxb_size dup (?)
line01_Rxb      db      Rxb_size dup (?)
line02_Rxb      db      Rxb_size dup (?)
line03_Rxb      db      Rxb_size dup (?)
line04_Rxb      db      Rxb_size dup (?)
line05_Rxb      db      Rxb_size dup (?)
line06_Rxb      db      Rxb_size dup (?)
line07_Rxb      db      Rxb_size dup (?)

        end     start
```

CP.EQU

```
;*****************************************************************************
;File:    cp.equ
;Purpose: Equates for sample control programs for HOSTESS 186.
;Company: Comtrol Corporation
;Release: 1.00, Craig Harrison - Original release.
;Date:    2-18-92
;*****************************************************************************

;Interrupt Controller (PIC) Registers
EOI_VAL          equ     8000h          ;end of interrupt value
INT1_type        equ     0dh            ;system interrupt vector type
INTCTL           equ     0ff22h         ;PIC port
INTCTL1          equ     0ff28h         ;PIC port for initialization command word
TIMER_INT_CTRL   equ     0ff32h         ;timer interrupt control register

;Timer registers
TIMER1_CNT       equ     0ffffh         ;30/sec counter
TIM0_type        equ     08h            ;timer 0 interrupt vector type
TIM1_type        equ     12h            ;timer 1 interrupt vector type
TMR1_CTRL_reg    equ     0ff5eh         ; mode/control register
TMR1_MAX_CNTB    equ     0ff5ch         ; max count B
TMR1_MAX_CNTA    equ     0ff5ah         ; max count A
TMR1_CNT_reg     equ     0ff58h         ; count register
TMR0_CTRL_reg    equ     0ff56h         ; mode/control register
TMR0_MAX_CNTB    equ     0ff54h         ; max count B
TMR0_MAX_CNTA    equ     0ff52h         ; max count A
TMR0_CNT_reg     equ     0ff50h         ; count register


C2Sint_reg       equ     0ef60h         ;COM uP to SYS uP interrupt register
C2Sint_hi        equ     0              ;value to set interrupt line high
C2Sint_low       equ     0008h          ;value to set interrupt line low

Txb_size         equ      512           ;transmit buffer size
Txq_mask         equ      Txb_size-1
Rxb_size         equ      2048          ;receive buffer size
Rxq_mask         equ      Rxb_size-1

base_vector      equ     80h            ;base interrupt vector type


;Message Queue Equates
msg_len          equ     16             ;message length
msgq_size        equ     32             ;number of message queue entries
msgq_mask        equ     msgq_size-1
```

CP.EQU

```
;-----------------------------------------------------------------------------
;Line Table data structure

line_entry       struc

io_base          dw      ?              ;SCC base I/O address
line_status      dw      ?              ;line status (defined below)

;SCC register values
WR2_             db      ?
WR3_             db      ?
WR4_             db      ?
WR5_             db      ?
WR12_            db      ?
WR13_            db      ?
WR15_            db      ?
                 db      ?               ;filler, keeps things on even boundaries

;Transmit queue data
Txq_head         dw      ?
Txq_tail         dw      ?
Txq_offset       dw      ?

;Receive queue data
Rxq_head         dw      ?
Rxq_tail         dw      ?
Rxq_offset       dw      ?

                 dw      ?               ;filler
                 dw      ?               ;filler
                 dw      ?               ;filler
                 dw      ?               ;filler

line_entry       ends
line_entry_len   equ     size line_entry




;-----------------------------------------------------------------------------
;line_status definitions

line_active      equ     0001h          ;line is active
Tx_active        equ     0002h          ;transmit is active (char is going out)
```

```
;-------------------------------------------------------------------------------
;Message Queue definition

msgq_entry      struc
msgq_head       dw      ?               ;queue head pointer
msgq_tail       dw      ?               ;queue tail pointer
msgq_area       db      msgq_size*msg_len dup (?)    ;queue buffers
msgq_entry      ends

;SCC register equates
WR0     equ     0
WR1     equ     1
WR2     equ     2
WR3     equ     3
WR4     equ     4
WR5     equ     5
WR6     equ     6
WR7     equ     7
WR8     equ     8
WR9     equ     9
WR10    equ     10
WR11    equ     11
WR12    equ     12
WR13    equ     13
WR14    equ     14
WR15    equ     15

RR0     equ     0
RR1     equ     1
RR2     equ     2
RR3     equ     3
RR8     equ     8
RR10    equ     10
RR12    equ     12
RR13    equ     13
RR15    equ     15

; Baud Rate Generator Time Constants - x16 Baud Rate Factor
; (based on a 4.9152 MHz clock)
bps50           equ     3070
bps75           equ     2046
bps110          equ     1394            ; 0.026 percent error
bps134          equ     1140            ; 0.001 percent error
bps150          equ     1022
bps300          equ     510
bps600          equ     254
bps1200         equ     126
bps1800         equ     83              ; 0.401 percent error
bps2000         equ     75              ; 1.06 percent error
bps2400         equ     62
```

CP.EQU

```
;*******************************************************************************
;
;Write Register Definitions (for basic asynchronous communications)
;
;*******************************************************************************

;Write Register 0 - command register
reset_ext       equ     10h
reset_Tx_int    equ     28h
error_reset     equ     30h
reset_ius       equ     38h

;Write Register 1 - Tx/Rx interrupt and data transfer mode definition
ext_int_enable  equ     01h
Tx_int_enable   equ     02h
parity_special  equ     04h
Rx_int_enable   equ     10h

;Write Register 2 - interrupt vector

;Write Register 3 - Rx parameters and controls
Rx_enable       equ     01h
Rx5_bit_char    equ     00h
Rx7_bit_char    equ     40h
Rx6_bit_char    equ     80h
Rx8_bit_char    equ     0c0h


;Write Register 4 - Tx/Rx miscellaneous parameters and modes
parity_enable   equ     01h
parity_even     equ     02h
parity_odd      equ     00h
one_stop_bit    equ     04h
one5_stop_bits  equ     08h
two_stop_bits   equ     0ch
x16_clock       equ     40h

;Write Register 5 - Tx parameters and controls
RTS             equ     02h
Tx_enable       equ     08h
BREAK           equ     10h
Tx5_bit_char    equ     00h
Tx7_bit_char    equ     20h
Tx6_bit_char    equ     40h
Tx8_bit_char    equ     60h
DTR             equ     80h

;Write Register 6 - sync character or SDLC address field

;Write Register 7 - sync character or SDLC flag
```

```
;Write Register 9 - master interrupt control
VIS             equ     01h
NV              equ     02h
DLC             equ     04h
MIE             equ     08h
status_lo       equ     00h
status_hi       equ     10h
reset_ch_B      equ     40h
reset_ch_A      equ     80h
hardware_reset  equ     0c0h

;Write Register 10 - miscellaneous Tx/Rx control bits

;Write Register 11 - clock mode control
Tx_clk_eq_BRG   equ     10h
Rx_clk_eq_BRG   equ     40h

;Write Register 12 - lower byte of baud generator time constant

;Write Register 13 - upper byte of baud generator time constant

;Write Register 14 - miscellaneous control bits
BRG_enable      equ     01h
BRG_eq_sys_clk  equ     02h

;Write Register 15 - external/status interrupt control
DCD_ie          equ     08h
CTS_ie          equ     20h
break_ie        equ     80h

;*************************************************************************
;
;Read Register Definitions (for basic asynchronous communications)
;
;*************************************************************************

;Read Register 0 - Tx/Rx buffer status and external status
Rx_buffer       equ     01h
Tx_buffer       equ     04h
DCD_            equ     08h
CTS_            equ     20h
break_cond      equ     80h

;Read Register 1 - special receive condition status
parity_error    equ     10h
overrun_error   equ     20h
framing_error   equ     40h

;Read Register 2 - interrupt vector
```

DPLOADER.C

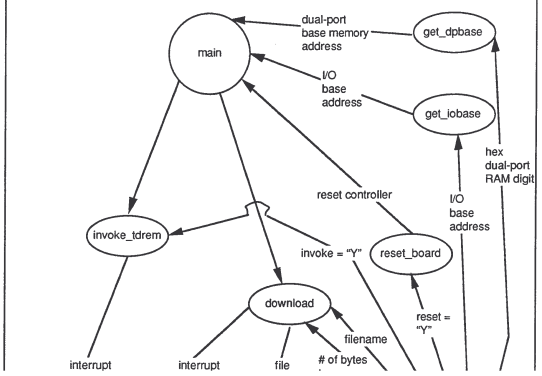
How DPLOADER Works

DPLOADER is a DOS program written in C. This program can:

- reset the Hostess 186 controller,
- remove header bytes before downloading,
- download a binary file into dual-port RAM on the Hostess 186, and
- start the Turbo Debugger debugger kernel code on the Hostess 186.

The diagram that follows show the data flow of DPLOADER.C.

DPLOADER.C Data Flow:



```
/* File:    dploader.c
   Company: Comtrol Corporation
   Purpose: Reset controller and start Turbo Debugger Remote if needed.
            Load a binary file into the Smart HOSTESS dual port RAM and
            signal the COM processor to begin execution.   Strip off the
            first ? bytes of the binary file before downloading, if needed.
   Release: 1.00: February 4, 1988 - Craig Harrison - Original
            1.01: June 23, 1991 - Craig Harrison
                  1. Add support for HOSTESS i control register initialization.
            1.02: Aug  23, 1991 - Craig Harrison
                  1. Add reset option
                  2. Add Turbo Debugger Remote startup option
            1.03: 12-3-91 - Craig Harrison
                  Changed all "outportb" to "outp" for compatability with
                  Microsoft C.
            1.04: 1-2-92 - Scott Davidson
                  Changed program to work with Hostess 186 instead of Hostess i

*/
#include <stdio.h>
#include <fcntl.h>
#include <dos.h>


#define EN_RAM     0x80                    /* Enable DPRAM */
#define HiCR0      0x01                    /* Select Control Reg 0 */
#define HiCR1      0x02                    /* Select Control Reg 1 */
#define DIS_RAM    0x00                    /* Disable DPRAM */
#define HiCR2      0x04                    /* Select Control Reg 2 */
#define HiCR3      0x08                    /* Select Control Reg 3 */


char crlval;                               /* HOSTESS 186 CR1 value */

/* HOSTESS 186 CR0 values for different window sizes */
char win16k = 0x07;
char win32k = 0x06;
char win64k = 0x04;


/* HOSTESS 186 CR2 values for various offsets */
char w16k[8] = {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07};  /* 16K windows */
char w32k[4] = {0x00,0x02,0x04,0x06};             /* 32K windows */
char w64k[2] = {0x00,0x04};                       /* 64K windows */
```



```
void cr_init(int io);
unsigned far *get_dpbase();
int get_iobase();
void download(int io,unsigned far *flag_p);
void reset_board(int io,unsigned far *flag_p);
int invoke_tdrem(int io,unsigned far *flag_p);


main()
{
   int io_base;                            /* I/O ports base address */
   unsigned far *dp_base;                  /* dual port base address */

   dp_base = get_dpbase();                 /* get dual port base address */
   io_base = get_iobase();                 /* get I/O base address */
   reset_board(io_base,dp_base);           /* reset board if needed */
   download(io_base,dp_base);              /* download & start ctrlpgm */
   return(0);
}



/*************************************************************************/
/* Purpose: HOSTESS 186 control register initialization                    */
/* Formal parms: None                                                      */
/* Parms modified: None                                                    */
/* Returns: Nothing                                                        */

void cr_init(int io)
{
   outp(io+1,HiCR0 | DIS_RAM);             /* access CR0, PC mode */
   outp(io,win64k);                        /* 64k window size */
   outp(io+1,HiCR1 | DIS_RAM);             /* access CR1 */
   outp(io,crlval);                        /* below 1 Meg RAM address */
   outp(io+1,HiCR2 | DIS_RAM);             /* access CR2 */
   outp(io,w64k[1]);                       /* 64K upper window */
   outp(io+1,HiCR3 | DIS_RAM);             /* access CR3 */
   outp(io,cr4[0]);                        /* no IRQ set */
   outp(io+1,EN_RAM);                      /* enable DPRAM */
}
```

```
/*************************************************************************/
/* Purpose: Get dual port memory base address                            */
/* Formal parms: None                                                    */
/* Parms modified: None                                                  */
/* Returns: dp memory base address                                       */

unsigned far *get_dpbase()
{
   int addrin, valid;

   printf("Enter most significant digit of dual port RAM address in hex: ");
   do
   {
      valid = 1;
      scanf("%x",&addrin);
      switch(addrin)
      {
         case 8:
            crlval = 0x08;
            return((unsigned far *)0x80000000);
         case 9:
            crlval = 0x09;
            return((unsigned far *)0x90000000);
         case 0xA:
            crlval = 0x0A;
            return((unsigned far *)0xA0000000);
         case 0xB:
            crlval = 0x0B;
            return((unsigned far *)0xB0000000);
         case 0xC:
            crlval = 0x0C;
            return((unsigned far *)0xC0000000);
         case 0xD:
            crlval = 0x0D;
            return((unsigned far *)0xD0000000);
         case 0xE:
            crlval = 0x0E;
            return((unsigned far *)0xE0000000);
         case 0xF:
            crlval = 0x0F;
            return((unsigned far *)0xF0000000);
         default:
            printf("Invalid, try again: ");
            valid = 0;
            break;
      }
   }while(!valid);
   return((unsigned far *)0); /* This never reached, but it prevents compiler
                                 warning */
```

DPLOADER.C

```
/*************************************************************************/
/* Purpose: Get I/O base address                                         */
/* Formal parms: None                                                    */
/* Parms modified: None                                                  */
/* Returns: I/O base address                                             */

int get_iobase()
{
   int addrin;

   addrin = 0;
   printf("Enter I/O base address in hex: ");
   while(addrin == 0)
   {
      scanf("%x",&addrin);
      if((addrin < 0x218) || (addrin > 0xFFC))
      {
         printf("Invalid I/O address, try again: ");
         addrin = 0;
      }
   }
   return addrin;
}
```

```
/*****************************************************************************/
/* Purpose: Reset board                                                      */
/* Entry: Nothing                                                            */
/* Exit:  Nothing                                                            */

void reset_board(int io,unsigned far *flag_p)
{
   char strbuff[81];
   int i;
   unsigned rec_flag;

   printf("   Dual Port Base Address = %p\n",(char far *)flag_p);
   printf("   I/O Base Address = %XH\n",io);
   while(1)
   {
      printf("Reset HOSTESS 186 controller (Y/N)? ");
      scanf("%s",strbuff);
      switch(*strbuff)
      {
         case 'y':
         case 'Y':
            outp(io+3,0);
            delay(100);
            outp(io+3,0xff);
            printf("   Waiting for reset to complete\n");
            delay(500);
            cr_init(io);
            for(i = 0;i < 21;i++)
            {
               delay(1000);
               if(*flag_p == 0x55aa)
                  break;
            }
            if(((rec_flag = *flag_p) != 0x55aa) && i >= 21)
            {
               printf("   Reset failed, interaction flag = %XH\n",rec_flag);
               exit(1);
            }
            return;
         case 'n':
         case 'N':
            cr_init(io);
            return;
         default:
            break;
      }
   }
}
```

DPLOADER.C

```
/*****************************************************************************/
/* Purpose: Read a binary file into dual port memory, signal the COM         */
/*          Processor to start execution, verify execution started.          */
/* Entry:       flag_p - Pointer to interaction flag                         */
/*              io     - I/O base address                                    */
/* Returns: Nothing                                                          */
/* Notes: The file to be downloaded must:                                    */
/*        Have "org 0"                                                       */
/*        Have 80h bytes preceeding the code                                 */
/*        Write the value 55AAh to interaction flag when it starts execution */

void download(int io,unsigned far *flag_p)
{
   char far *ram_p;
   char fname[13];                        /* Name of file to download */
   unsigned wait;                         /* Do nothing loop counter */
   unsigned rec_flag;                     /* Received copy of interact flag*/
   unsigned long strip;                   /* Number bytes to strip off file*/
   char buffer[256];                      /* Buffer for data to be xfered
                                             to dual port RAM */
   long int nbytes;                       /* Number of bytes downloaded */
   int retry,                             /* Loop control flag */
       i,
       sector,                            /* File sector beign transfered */
       f_h,                               /* File handle */
       bytesread,                         /* Bytes read on current read */
       tdrem_invoked;                     /* Set of TDREM is invoked */

   /* Check interaction flag */
   if((rec_flag = *flag_p) != 0x55aa)
   {
      printf("   Bad interaction flag = %XH before download\n",rec_flag);
      exit(1);
   }

   /* Get file name and open file for read */
   retry = 1;
   while(retry)
   {
      printf("Enter control program file name to download: ");
      scanf("%s12",fname);
      if((f_h = open(fname,O_BINARY)) == -1)
         printf("   Unable to open file %s, try again.\n",fname);
      else
         retry = 0;
   }

   /* Strip off first ? bytes from file */
   printf("Enter number of bytes to strip off file: ");
```

```
   /* Download file from disk to dual port RAM */
   *flag_p = 0xAA55;                    /* Write to interaction flag */
   printf("   Downloading %s ... \n",fname);
   nbytes = 0;
   ram_p = (char far *)flag_p + 0x80;
   while((!eof(f_h)) && (nbytes < 0xFFFF))
   {
      bytesread = read(f_h,buffer,256); /* Read 256 bytes into buffer */
      nbytes = nbytes + bytesread;
      for(i = 0;i < bytesread;i++)
      {
         *ram_p++ = buffer[i]; /* Move buffer into DPRAM */
      }
   }
   if(close(f_h) != 0)
   {
      printf("   Error closing file %s after reading into DPRAM\n",fname);
      exit(1);
   }
   if(bytesread <= -1)
   {  /* Error reading file */
      printf("   Error reading file %s\n",fname);
      exit(1);
   }
   else if(nbytes >= 0xFFFF)
   {  /* File to big */
      printf("   File too large to fit in one segment\n");
      exit(1);
   }
   else
   {  /* Download successful, interrupt COM Processor if TDREM not invoked */
      printf("   %ld bytes downloaded successfully\n",nbytes);
      if(tdrem_invoked)
         return;
      outp(io + 2,0);                  /* Interrupt COM Processor */
      printf("   COM processor interrupted to start control program\n");
      for(wait = 0;wait < 10;wait++)     /* wait for control program to */
      {                                  /*   restore interaction flag */
         delay(400);
         if(*flag_p == 0x55aa)
            break;
      }
      if((wait >= 10) && ((rec_flag = *flag_p) != 0x55aa))
      {
         printf("   Control program failed to start, interact flag = %XH\n",
                 rec_flag);
         exit(1);
      }
      else
```

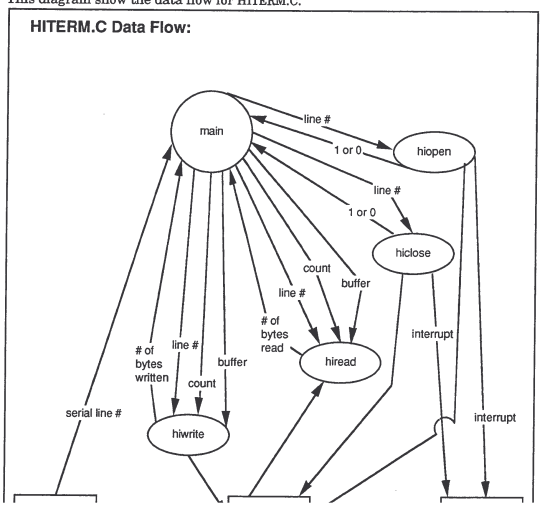
DPLOADER.C

```
/*************************************************************************/
/* Purpose: Invoke Turbo Debugger Remote kernel, if needed.              */
/* Entry: flag_p - Pointer to interaction flag                           */
/*        io     - Board I/O base address                                */
/* Returns: 1 if tdrem is invoked, else 0                                */

int invoke_tdrem(int io,unsigned far *flag_p)
{
   char strbuff[0x81];                    /* Stores string entered by user */
   int do_tdrem,valid_answer,wait;
   unsigned rec_flag;                     /* Received copy of interact flag*/

   valid_answer = 0;
   while(!valid_answer)
   {
      printf("Invoke Turbo Debugger Remote Kernel on HOSTESS 186 controller (Y/N)? ")
      scanf("%s",strbuff);
      switch(*strbuff)
      {
         case 'y':
         case 'Y':
            valid_answer = 1;
            do_tdrem = 1;
            break;
         case 'n':
         case 'N':
            valid_answer = 1;
            do_tdrem = 0;
            break;
         default:
            break;
      }
   }
   if(do_tdrem)
   {
      *(flag_p + 0x40) = 0x27cd;       /* Write int 27H instruction used to
                                          invoke TDREM kernel */
      *flag_p = 0xaa55;                /* Set up interaction flag */
      outp(io + 2,0);                /* Interrupt COM Processor */
      for(wait = 0;wait < 10;wait++)   /* wait for execution to finish*/
      {
         delay(200);
         if(*flag_p == 0x55aa)
            break;
      }
      if((wait >= 10) && ((rec_flag = *flag_p) != 0x55aa))
      {
         printf("   Turbo Debugger Remote Kernel failed to start, interact flag =
%XH\n",rec_flag);
```

HITERM.C

How HITERM Works

The HITERM program runs on the system and emulates a terminal. This DOS program, linked with the HILIB.ASM file routines, works with the control program. This diagram show the data flow for HITERM.C.



HITERM.C

Invoking HITERM

To use the executable file HITERM.EXE with CPC.EXE, follow these steps:

1. Set the Hostess 186 for I/O address 218h.
2. Check that no other device occupies the D000 base memory address. The program uses 64K starting at D000:0.
3. Install the controller in the system.
4. Connect a non-intelligent ASCII terminal to the port on the Hostess 186 that you want to use.
   Set the terminal to:
   - 9600 baud
   - 8 data bits
   - 1 stop bit
   - no parity
   - no flow control.
5. Start-up DOS.
6. Execute DPLOADER.EXE.

```
C: dploader
```

DPLOADER will prompt you for values it needs to download the control program.

```
Enter most significant digit of dual port RAM address in hex:     d
Enter I/O base address in hex:     218
     Dual Port Base Address = D000:0000
     I/O Base Address = 218H
Reset HOSTESS 186 controller (Y/N)?     y
     Waiting for reset to complete
Enter control program file name to download:     cpc.exe
Enter number of bytes to strip off file:     640
Invoke Turbo Debugger Remote Kernel on HOSTESS 186 controller (Y/N)?          n
     Downloading cpc.exe...
     XXXXX bytes downloaded successfully.
     COM processor interrupted to start control  program
     Control program started execution
```

```
/*****************************************************************************
 File:    hiterm.c
 Purpose: A sample terminal emulation program to demonstrate the use of
          hilib.asm and cp.asm with the HOSTESS i and Hostess 186 controllers.
 Company: Comtrol Corporation
 Release: 1.00, Craig Harrison - Original release
 Date:    8-23-91
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

/* Function prototypes */
int hiopen(int linenum);
int hiclose(int linenum);
int hiread(char *buffer,int cnt,int linenum);
int hiwrite(char *buffer,int cnt,int linenum);


main()
{
        int i;
        int pnum;
        int cnt;
        char buf[80];

        fprintf(stderr,"Serial Line Number (0-7): ");
        gets(buf);                                          /* get serial line */
        sscanf(buf,"%d",&pnum);
        system("cls");                                      /* clear screen */
        fprintf(stderr,"Serial Line Number %d\t\t\t\t\tHit F10 to Quit\n",
        pnum);

        if (!hiopen(pnum))
        {
                fprintf(stderr,"Can't open line %d\n",pnum);
                exit(0);
        }

        while(1)                                            /* infinite loop */
        {
                /* attempt to read char from device and write to screen */
                if ((cnt = hiread(buf,80,pnum)) > 0)
                        for (i=0;i<cnt;i+=2)
                        {
                                if (buf[i] == 015)          /* CR = CRLF */
                                        fprintf(stderr,"\n");
                                else if (buf[i] != 012)         /* not LF */
```



```
                /* attempt to read char from keyboard and write to device */
                if ((bdos(11,0,0)&0xff) == 0xFF)        /* if char waiting */
                {
                        buf[0] = bdos(8,0,0) & 0xff;    /* read keybd char */
                        if ((buf[0] == NULL) &&         /* 2 char key */
                        ((bdos(11,0,0)&0xff) == 0xff))
                        {
                                buf[1] = bdos(8,0,0) & 0xff;    /* next */
                                if(buf[1] == 0x44)      /* F10 = quit */
                                        break;
                                else
                                        hiwrite(buf,2,pnum);
                        }
                        else if (buf[0] == 015)         /* CR = CRLF */
                        {
                                buf[1] = 012;
                                hiwrite(buf,2,pnum);
                        }
                        else if (buf[0] != 012)         /* not LF */
                                hiwrite(buf,1,pnum);
                }
        }
        hiclose(pnum);
        return(0);
}
```

HILIB.ASM

HILIB.ASM is the file that contains four Hostess 186 routines: hiopen(), hiclose(), hiread(), and hiwrite(). Assemble and link the HILIB.ASM file with your application program to access the Hostes i's serial lines. The paragraphs that follow explain these routines, with examples in C syntax.

The `hiopen` routine opens a requested serial line on the Hostess 186, initializes the line to 9600 baud, 8 data bits, 1 stop bit, and no parity:

```
int hiopen(linenum)
int linenum;                /* number (0-7) of Hostess 186 line */
```

Returns 1 if successful, 0 if unsuccessful.

The function `hiclose` closes a requested serial line on the Hostess 186:

```
int hiclose(linenum)
int linenum;                /* number (0-7) of Hostess 186 line */
```

Returns 1 if successful, 0 if unsuccessful.

The `hiread` routine reads up to a maximum "cnt" bytes into the line's receive buffer. The routine does not wait for the bytes to read:

```
int hiread(buffer,cnt,linenum)
char *buffer;               /* local receive buffer */
int cnt;                    /* number of bytes to read */
int linenum;                /* number (0-7) of Hostess 186 line */
```

Returns the number of bytes read (0 - 'cnt').

The `hiwrite` routine writes up to a maximum "cnt" bytes from the line's receive buffer into dual-port memory . The routine does not wait for enough space to write if the request is too large

```
hiwrite(buffer,cnt,linenum)
char *buffer;               /* local transmit buffer */
int cnt;                    /* number of bytes to write */
int linenum;                /* number (0-7) of Hostess 186 line */
```

Returns the number of bytes written (0 - 'cnt').

HILIB.ASM

```
;*****************************************************************************
;File:    hilib.asm
;Purpose: A library of sample routines for the HOSTESS i and HOSTESS 186
;         controllers.
;Company: Comtrol Corporation
;Release: 1.00, Craig Harrison - Original release
;Date:    8-23-91
;*****************************************************************************

        .xlist
        include cp.equ
        .list


;If the sample program is to be run with the HOSTESS 186 installed at a different
;address these two equates must be changed
DPRAM_ADDR        equ     0d000h      ;System address HOSTESS 186 dual port RAM
IO_ADDR           equ     218h        ;System I/O base address of HOSTESS 186

.MODEL   small
.DATA

hiopenmsg       db      1,0,0c0h,44h,60h,0eh,0,0,0,0,0,0,0,0,0,0
hiclosmsg       db      2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0


.CODE
;-----------------------------------------------------------------------------
;Function:   int hiopen(linenum)
;Parameters: int linenum;        /* number of line to open (0-7) */
;Return:     1 if successful, 0 if failed
;Purpose:    Open a serial line on the HOSTESS 186

        public  _hiopen
_hiopen proc    near
        push    bp                      ;prolog, establish stack frame
        mov     bp,sp

        assume  es:nothing
        mov     ax,DPRAM_ADDR
        mov     es,ax                   ;es is dpram address

        mov     bx,offset Comq          ;start of comm proc queue
        mov     ax,es:[bx].msgq_head    ;ax = current msg number
        inc     ax                      ;bump pointer
        and     ax,msgq_mask            ;mask it
        cmp     ax,es:[bx].msgq_tail    ;is msg area full?
        mov     ax,0                    ;set up return value for fail
        jz      _hiopen_10              ;if it's is full exit now
```

```
        mov     ax,es:[bx].msgq_head    ;get head again
        mov     cx,msg_len              ;length of a message
        mul     cl                      ;get message offset
        lea     di,[bx].msgq_area       ;get destination
        add     di,ax                   ;es:di = destination
        mov     si,offset hiopenmsg     ;ds:si = source
        mov     ax,[bp+4]               ;line number to open
        mov     [si+1],al               ;2nd byte of msg is line number
        rep     movsb                   ;move the message

        mov     ax,es:[bx].msgq_head    ;get head again
        inc     ax                      ;increment it
        and     ax,msgq_mask            ;mask it
        mov     es:[bx].msgq_head,ax    ;store new head

        mov     dx,IO_ADDR+2            ;interrupt comm proc
        out     dx,al

        mov     ax,1                    ;set up success return value
        pop     ds
        pop     di
        pop     si

_hiopen_10:
        pop     bp                      ;epilog
        ret
_hiopen  endp

;---------------------------------------------------------------------------
;Function:   int hiclose(linenum)
;Parameters: int linenum;        /* number of line to close (0-7) */
;Returns:    1 if successful, 0 if failed
;Purpose:    Close a serial line on the HOSTESS 186

        public  _hiclose
_hiclose proc   near
        push    bp                      ;prolog, establish stack frame
        mov     bp,sp

        mov     ax,DPRAM_ADDR
        mov     es,ax                   ;es = dpram address
        assume  es:nothing

        mov     bx,offset Comq          ;start of comm proc queue
        mov     ax,es:[bx].msgq_head    ;ax = current msg number
        inc     ax                      ;bump pointer
        and     ax,msgq_mask            ;mask it
        cmp     ax,es:[bx].msgq_tail    ;is msg area full?
        mov     ax,0                    ;set up return value for fail
        jz      hiclose_10              ;if it's is full exit now
```



```
        mov     ax,es:[bx].msgq_head    ;get head again
        mov     cx,msg_len              ;length of a message
        mul     cl                      ;get message offset
        lea     di,[bx].msgq_area       ;get destination
        add     di,ax                   ;es:di = destination
        mov     si,offset hiclosmsg     ;ds:si = source
        mov     ax,[bp+4]               ;line number to close
        mov     [si+1],ax               ;2nd byte of msg is line number
        rep     movsb                   ;move the message

        mov     ax,es:[bx].msgq_head    ;get head again
        inc     ax                      ;increment it
        and     ax,msgq_mask            ;mask it
        mov     es:[bx].msgq_head,ax    ;store new head

        mov     dx,IO_ADDR+2            ;interrupt comm proc
        out     dx,al
        pop     ds
        pop     di
        pop     si
        mov     ax,1                    ;set up sucess return value

_hiclose_10:
        pop     bp                      ;epilog
        ret
_hiclose endp

;---------------------------------------------------------------------------
;Function:   int hiread(buffer,cnt,linenum)
;Parameters: char *buffer;       /* points to storage buffer */
;            int cnt;            /* maximum count to read */
;            int linenum;        /* number (0-7) of serial line */
;Returns:    Integer number of bytes read
;Purpose:    Read bytes from an open serial line on the HOSTESS 186

        public  _hiread
_hiread  proc   near
        push    bp                      ;prolog, establish stack frame
        mov     bp,sp
        sub     sp,2
        push    si                      ;save registers
        push    di
        push    ds

        mov     ax,[bp+8]               ;ax = line number
        call    higet_line              ;si = line table
        mov     di,si                   ;di = line table
        mov     [bp-2],si               ;also save it
```

```
        mov     ax,[bp+6]               ;ax = count
        mov     cx,ds:[si].Rxq_head     ;get head
        mov     bx,ds:[di].Rxq_tail     ;get tail
        sub     cx,bx                   ;cx = head - tail
        jb      _hiread_10              ;if cx < 0 jump
        call    himin                   ;get min of cx and ax
        mov     dx,cx                   ;store min in dx
        jcxz    _hiread_30              ;exit if min = 0 (i.e. empty)

        mov     si,ds:[di].Rxq_offset   ;si = receive buffer
        add     si,bx                   ;si = current loc in buffer
        mov     di,[bp+4]               ;di = local buffer
        rep     movsb                   ;read bytes into buffer
        jmp     short _hiread_20        ;jump
_hiread_10:
        add     cx,Rxq_mask             ;adjust cx
        inc     cx
        call    himin                   ;now get min of cx and ax again
        mov     dx,cx                   ;store min
        jcxz    _hiread_30              ;exit if min = 0

        mov     ax,Rxq_mask             ;ax = mask
        sub     ax,bx                   ;ax = mask - tail
        inc     ax                      ;ax = buffsize - tail
        mov     cx,ax                   ;cx = count I can read here
        mov     si,ds:[di].Rxq_offset   ;si = receive buffer
        add     si,bx                   ;si = current loc in buffer
        mov     di,[bp+4]               ;di = local buffer
        rep     movsb                   ;read bytes into buffer
        mov     cx,dx                   ;cx = full min count
        sub     cx,ax                   ;subtract what we just read
        jcxz    _hiread_20              ;exit if 0

        mov     bx,[bp-2]               ;restore start of line table
        mov     si,ds:[bx].Rxq_offset   ;restore start of buffer
        rep     movsb                   ;read rest into local buffer

_hiread_20:
        mov     di,[bp-2]               ;di = line table
        mov     bx,ds:[di].Rxq_offset   ;bx = receive buffer
        sub     si,bx                   ;get new tail
        and     si,Rxq_mask             ;mask it
        mov     ds:[di].Rxq_tail,si     ;store it

_hiread_30:
        mov     ax,dx                   ;return value = number of bytes read
        pop     ds                      ;epilog
        pop     di
```

HILIB.ASM

```
;---------------------------------------------------------------------------
;Function:   int hiwrite(buffer,cnt,linenum)
;Parameters: char *buffer;       /* points to buffer to write */
;            int cnt;            /* number of bytes to write */
;            int linenum;        /* number (0-7) of serial line */
;Returns:    Integer number of bytes written
;Purpose:    Write bytes to an open serial line on the HOSTESS 186

        public  _hiwrite
_hiwrite proc   near
        push    bp                      ;prolog, establish stack frame
        mov     bp,sp
        sub     sp,2
        push    si                      ;save registers
        push    di
        push    es

        mov     ax,DPRAM_ADDR           ;set up dual line memory seg
        mov     es,ax
        assume  es:nothing

        mov     ax,[bp+8]               ;get line number
        call    higet_line              ;si = line table
        mov     [bp-2],si               ;save the pointer

        mov     cx,es:[si].Txq_tail     ;get tail
        mov     bx,es:[si].Txq_head     ;get head
        sub     cx,bx                   ;get difference
        jbe     _hiwrite_10             ;branch if tail <= head

        mov     ax,[bp+6]               ;ax = count
        dec     cx                      ;adjust tail - head
        call    himin                   ;return minimum of ax and cx
        mov     dx,cx                   ;save minimum in dx
        jcxz    _hiwrite_30             ;exit if min = 0 (i.e. full)

        mov     di,es:[si].Txq_offset   ;get offset of buffer
        add     di,bx                   ;get starting loc in buffer
        mov     si,[bp+4]               ;si = start of write buffer
        rep     movsb                   ;write buffer to dpm
        jmp     short _hiwrite_20       ;done

_hiwrite_10:
        mov     ax,[bp+6]               ;ax = count
        add     cx,Txq_mask             ;adjust tail - head
        call    himin
        mov     dx,cx                   ;store minimum in dx
        jcxz    _hiwrite_30             ;exit if min = 0
```

```
        mov     cx,dx                   ;cx = count
        sub     cx,ax                   ;subtract what we just wrote
        jcxz    _hiwrite_20             ;exit if done

        mov     bx,[bp-2]               ;get line table pointer
        mov     di,es:[bx].Txq_offset   ;reset buffer pointer to start
        rep     movsb                   ;write rest of buffer
_hiwrite_20:
        mov     si,[bp-2]               ;si = line table
        mov     bx,es:[si].Txq_offset   ;get offset of buffer
        sub     di,bx                   ;get new head pointer
        and     di,Txq_mask             ;mask it
        mov     es:[si].Txq_head,di     ;store new head

_hiwrite_30:
        mov     ax,dx                   ;return value = bytes written
        pop     es
        pop     di                      ;epilog
        pop     si
        mov     sp,bp
        pop     bp
        ret
_hiwrite endp

;-------------------------------------------------------------------------

higet_line proc
        mov     cl,line_entry_len       ;cl = length of a line table
        mul     cl                      ;mult linenum by length
        add     ax,offset line00        ;add start of line tables
        mov     si,ax                   ;return in si
        ret
higet_line endp

;-------------------------------------------------------------------------

himin   proc
        cmp     ax,cx                   ;compare the two values
        jnc     himin_10                        ; branch if ax >= cx
        mov     cx,ax                   ;make cx = ax
        jmp     short himin_99
himin_10:
        mov     ax,cx                   ;make ax = cx
himin_99:
        ret
himin   endp
```



```
;-------------------------------------------------------------------------
;This segment forms a template for dual port RAM so that offsets can be
;computed.  It does not actually reserve any memory.

DPRAM   segment para AT 0
                org 0h
;The first 80h bytes are defined by the firmware
        public  interact_flag
interact_flag   dw      ?               ;processor interaction flag
boot_flag       dw      ?               ;boot/activity flag
cfg_map         dw      ?               ;configuration map
fw_release      db      8 dup (?)       ;firmware release number
sw_release      db      8 dup (?)       ;control program release number
                dd      ?               ;reserved
dram_map        dd      ?               ;DRAM map
scc_map         dd      ?               ;SCC map
board_id        dd      ?               ;board ID
ii_flag         db      ?               ;invalid interrupt flag
ii_type         db      ?               ;invalid interrupt type
ii_cnt          dw      ?               ;invalid interrupt count
                db      128-42 dup (?) ;balance of firmware area

        org     1000h
;-------------------------------------------------------------------------
;Comq - Queue for messages from System Processor to Communications Processor
Comq    msgq_entry      <>

;Sysq - Queue for messages from Communications Processor to Systems Processor
Sysq    msgq_entry      <>

        dw      4 dup (?)               ;filler

;-------------------------------------------------------------------------
;Line table, one entry for each line

line00  line_entry      <>
line01  line_entry      <>
line02  line_entry      <>
line03  line_entry      <>
line04  line_entry      <>
line05  line_entry      <>
line06  line_entry      <>
line07  line_entry      <>

;-------------------------------------------------------------------------
;Transmit buffers, one for each line

line00_Txb      db      Txb_size dup (?)
line01_Txb      db      Txb_size dup (?)
line02_Txb      db      Txb_size dup (?)
```

```
;-------------------------------------------------------------------------------
;Receive buffers, one for each line

line00_Rxb      db      Rxb_size dup (?)
line01_Rxb      db      Rxb_size dup (?)
line02_Rxb      db      Rxb_size dup (?)
line03_Rxb      db      Rxb_size dup (?)
line04_Rxb      db      Rxb_size dup (?)
line05_Rxb      db      Rxb_size dup (?)
line06_Rxb      db      Rxb_size dup (?)
line07_Rxb      db      Rxb_size dup (?)

DPRAM   ends
        end
```

TSAMPLE.MK

```
#CC = tcc                              # for Turbo C++
CC = bcc                               # for Borland C++
#CPCLIB = \tc\lib\cs.lib               # for Turbo C++
CPCLIB = c:\borlandc\lib\cs.lib        # for Borland C++

nothing: hiterm.exe dploader.exe cpa.com cpc.exe

dploader.exe: dploader.c tsample.mk
    $(CC) -mc -v dploader.c

hiterm.exe: hiterm.c hilib.asm tsample.mk
    $(CC) -ms -v hiterm.c hilib.asm


#CPA (built with symbol table for debugging with Turbo Debug)
cpa.com: cpa.asm cp.equ tsample.mk
    tasm /l /s /zi cpa.asm, cpa.obj
    tlink /m /s /v cpa.obj, cpa.exe, cpa.map
    tdstrip -s -c cpa.exe cpa.tds


#CPC (built with symbol table for debugging with Turbo Debug)
cpc.exe: cpc.obj cpcstart.obj tsample.mk
    tlink /m /s /v cpcstart.obj cpc.obj, cpc.exe, cpc.map, $(CPCLIB)
    tdstrip -s cpc.exe

cpc.obj: cpc.c tsample.mk
    $(CC) -c -mt -G -v -ocpc.obj cpc.c

cpcstart.obj: cpcstart.asm cp.equ tsample.mk
   tasm /l /s /zi cpcstart.asm, cpcstart.obj
```

CHAPTER 3 – Preview: Initializing the Controller, the Control Registers, and Memory

To start up the controller, you must first reset and initialize the controller, initialize the control registers, and finally enable the memory on the controller. The chapters that follow explain in detail how these actions occur. All of these actions occur by writing to the I/O base address plus an offset:

I/O Address:	Description:
I/O base+0	writes to control registers
I/O base+1	switches memory ON or OFF, control register index
I/O base+2	interrupts controller
I/O base+3	resets controller

After the system boots, the Hostess 186's firmware code executes and initializes the controller's registers and data structures to a preliminary state.

Controller State at Startup

When the Hostess 186's firmware code executes, and it sets the controller to the following state:

- the control registers are set to the following values:

Control Register:	Default Setting:
1	8 bit memory transfer, 128K window, below 1 MB base address.
2	0 base memory address
3	Zero offset into dual-port memory
4	IRQ disabled

- the timer interrupts are not enabled.
- the 80186's timer 2 and DMA channel 1 are reserved for controller memory refresh. (Timer 2 can be used as a clock input for timer 0 and timer 1. DMA channel 1

- the following interrupt vectors are initialized:

Hostess 186 Interrupt:	Vector type number:	Vector table location:
NMI	2h	8h
DEBUGGER	20h	80h
RAM_QUERY	21h	84h
DEBUG_PORT	22h	88h
CONFIG_QUERY	23h	8Ch
TURBO_DEBUGGER_REMOTE	27h	9Ch
8530 bank 1	0Ch	*
SYSTEM	0Dh	34h
IRQ7 (invalid interrupts)	37h	DCh
SCC base	80h	200h

* Operates in cascade mode and therefore does not use this vector table entry.

- the firmware user area is initialized in dual-port memory.

From this preliminary state, the Hostess 186 is ready to be reset by the control program.

Resetting and Initializing the Controller

Writing to the address I/O_base+3 resets the controller. Write the value 00h, delay one-tenth of a second, then write the value 0FFh. The controller's memory will come up as disabled after the controller is reset. Writes to the system or reads to dual-port RAM are not allowed between these two I/O writes. (For UNIX device drivers, you can protect these two I/O writes using an spl7() kernel call.)

An example of these two I/O writes is:

```
outp (I/O_base+3, 00h);
delay (HZ/10);
outp (I/O_base+3, 0FFh)       Resets the Hostess 186 controller
```

Initializing Control Registers

The control registers are written through a two-step process:

- an index value is written out to `I/O_base+1` to select the control register:

Control Register:	Index with RAM disabled:
1	01h
2	02h
3	04h
4	08h

- the register contents are written out to `I/O_base+0`. The index will remain fixed until an `I/O_base+1` is written again. Subsequent writes to the same control register are permitted without intervening index writes (this is useful for applications that use a sliding window into dual-port RAM).

Initialize all control registers before enabling the controller's memory. This means that the data bit SD7 must bet set to a 0 whenever you write out to `I/O_base+1`. (See the table column titled "Index with RAM disabled.")

For example:

```
outp (I/O_base+1, 01h);       Setup for Control Register #1
outp (I/O_base+0, <value>)

outp (I/O_base+1, 02h);       Setup for Control Register #2
outp (I/O_base+0, <value>)

outp (I/O_base+1, 04h);       Setup for Control Register #3
outp (I/O_base+0, <value>)

outp (I/O_base+1, 08h);       Setup for Control Register #4
outp (I/O_base+0, <value>)
```

After initializing the control registers, enable memory by executing the following:

```
outp (I/O_base+1, 80h)
```

Once the control registers are initialized, you access the registers with new addresses.

Control	Index with

CHAPTER 4 – Control Registers Used on the Hostess 186

Register Features

There are four control registers on the Hostess 186. These write-only registers control the memory addressing, and select the memory window size, interrupts, and mode of operation (either PC (8-bit) or AT (16-bit)). You access the control registers by writing an index value to the `I/O base + 1` address, then the register contents to the `I/O base + 0` address. (The chapter titled "Input/Output Addresses" has a section that explains writing to the `I/O base + offset` address.) The three-position DIP switch SW1 selects the I/O base address.

Overall, the registers function in this manner:
- Control register #1 selects the "sliding window" size, the AT/PC data transfer mode, and part of the controller's system address.
- Control register #2 selects the remainder of the controller's system address.
- Control register #3 selects the "sliding window" of dual-port memory.
- Control register #4 selects the interrupt request (IRQ).

Control Register #1

This write-only register selects the:
- mode of memory transfer between the controller and the system.
- windowing capability for the controller.
- size of the controller's "sliding window."

Writing a value to data bits SD0 to SD2 sets the size of dual-port memory's sliding window. This "window" is the portion of the dual-port memory the system processor sees at any one time. Writing a value to data bit SD4 determines if the Hostess 186 uses a sliding window. Writing a value to data bit SD5 determines the mode of data transfer between the controller and the system. Finally, writing a value to data bits SD6 and SD7 specifies part of the controller's system address. (You set the Hostess 186's system address with data bits SD6 and SD7 of control register #1, and data bits SD0 through SD7 of control register #2.)

Data bit:	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0
Field:	SA15	SA14	AT/PC mode	Window enable	not used	64K	32K	16K

Figure 8. Control register #1 format.

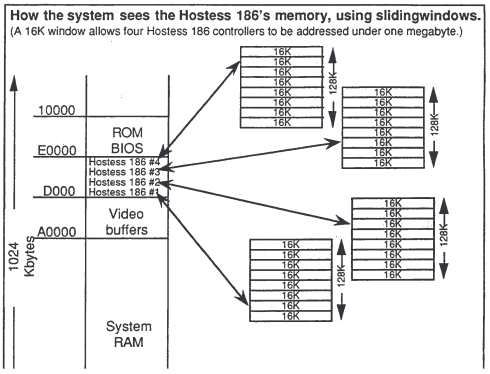
memory space. All boards within a 128K block that begins on a 128K boundary must use the same mode of operation.

Data bits SD0 to SD2 selects the size of dual-port memory's sliding window.

Table 5. Control register #1 sliding window size format.

Control Register #3 Data Bits: SD2 2	SD1 1	SD0 0	Window Size:
0	0	0	128K
1	0	0	64K
1	1	0	32K
1	1	1	16K

Setting the sliding window size determines how much of the dual-port RAM the system may access at one time. A 16K window allows four Hostess 186 controllers to be configured under one megabyte.



This example selects the below one megabyte base address D000:0h, using the I/O base 218h, with a 64K window:

```
outp (219h,01h);                    /* access CR#1, */
outp (218h,04h);                    /* set 64K upper window ,
                                       PC mode */
outp (219h,02h);                    /* access CR#2 */
outp (218h,0Dh);                    /* below 1 Meg RAM address */
outp (219h,04h);                    /* access CR#3 */
outp (218h,04h);                    /* set 64K offset */
...
...
outp (219h,80h);                    /* enable DPRAM */
```


Control Register #2

This write-only register selects the system memory address for the controller. Registers #1, #2, and #3 together manage the Hostess 186's addressing.

For example, if you address the Hostess 186 controller below one megabyte and use a sliding window; you must set the AT/PC mode bit, the WINDOW ENABLE bit, and sliding window size bits in control register #1, set the system address bits in control register #1 and #2, and the sliding window offset bits in control register #3

Control register # 1's data bits SD6 and SD7, combined with control register #2's data bits SD0 to SD7 set the system address for the Hostess 186 controller:

Data bit:	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0
Field:	SA23	SA22	SA21	SA20	SA19	SA18	SA17	SA16

Figure 10. Control register #2 format.

The range of base addresses possible is from 80000h to FE0000h. Table 6 lists the popular system addresses under sixteen megabytes, whereas Table 7 lists the popular system addresses under one megabyte.

Table 6. Memory locations addressed under sixteen megabytes.

Address:	Value for Control Register #2, Data bits SD7 to SD0:	Value for Control Register #1, Data bits SD7 to SD6:	Address:	Value for Control Register #2, Data bits SD7 to SD0:	Value for Control Register #1, Data bits SD7 to SD6:
F00000h	0F0h	30h	F80000h	0F8h	30h
F20000h	0F2h	30h	FA0000h	0FAh	30h
F40000h	0F4h	30h	FC0000h	0FCh	30h
F60000h	0F6h	30h	FE0000h	0FEh	30h
E00000h	0E0h	30h	E80000h	0E8h	30h
E20000h	0E2h	30h	EA0000h	0EAh	30h
E40000h	0E4h	30h	EC0000h	0ECh	30h
E60000h	0E6h	30h	EE0000h	0EEh	30h
D00000h	0D0h	30h	D80000h	0D8h	30h
D20000h	0D2h	30h	DA0000h	0DAh	30h
D40000h	0D4h	30h	DC0000h	0DCh	30h
D60000h	0D6h	30h	DE0000h	0DEh	30h

This table defines the popular memory base locations for Hostess 186 controllers addressed under one megabyte:

Table 7. Memory locations addressed under one megabyte.

Controller Memory Address and Offset	Value for Control Register # 2 Bits 0 to 7 (SD7 to SD0)	Value for Control Register # 1 Bits 6 and 7 (SD7 to SD6)	Valid System Window Sizes (set with Control Register #1)
8000:0000	08h	00h	16K, 32K, 64K
8000:4000	08h	01h	16K
8000:8000	08h	02h	16K, 32K
8000:C000	08h	03h	16K
9000:0000	09h	00h	16K, 32K, 64K
9000:4000	09h	01h	16K
9000:8000	09h	02h	16K, 32K
9000:C000	09h	03h	16K
A000:0000	0Ah	00h	16K, 32K, 64K
A000:4000	0Ah	01h	16K
A000:8000	0Ah	02h	16K, 32K
A000:C000	0Ah	03h	16K
B000:0000	0Bh	00h	16K, 32K, 64K
B000:4000	0Bh	01h	16K
B000:8000	0Bh	02h	16K, 32K
B000:C000	0Bh	03h	16K
C000:0000	0Ch	00h	16K, 32K, 64K
C000:4000	0Ch	01h	16K
C000:8000	0Ch	02h	16K, 32K
C000:C000	0Ch	03h	16K
D000:0000	0Dh	00h	16K, 32K, 64K
D000:4000	0Dh	01h	16K
D000:8000	0Dh	02h	16K, 32K
D000:C000	0Dh	03h	16K
E000:0000	0Eh	00h	16K, 32K, 64K
E000:4000	0Eh	01h	16K
E000:8000	0Eh	02h	16K, 32K

The following example selects the above one megabyte base address FA0000h, using the I/O base 218h:

```
outp (219h,01h);                    /* access CR#1 */
outp (218h,30h);                    /* set address, AT mode */
outp (219h,02h);                    /* access CR#2 */
outp (218h,FAh);                    /* zero out */
outp (219h,04h);                    /* access CR#3 */
outp (218h,00h);                    /* zero out */
...
...
outp (219h,80h);                    /* enable DPRAM */
```

Control Register #3

This write-only register selects and controls the dual-port memory window offset. Remember, this "window" is the portion of the dual-port memory the system processor sees at any one time.

Data bit:	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0
Field:	not used	not used	not used	not used	not used	WA16	WA15	WA14

Figure 11. Control register #3 format.

The fields WA14, WA15, and WA16 control the window's offset from the beginning of the 128K block of dual-port memory.

Table 8. Control register #3 window offset format.

Control Register #3 Bits 0 to 2 (LD0 to LD2)	Data Bits	LD2 2	LD1 1	LD0 0	16K Window Offset:	32K Window Offset:	64K Window Offset:
00h		0	0	0	+ 0	+ 0	+ 0
01h		0	0	1	+ 16K	+ 0	+ 0
02h		0	1	0	+ 32K	+ 32K	+ 0
03h		0	1	1	+ 48K	+ 32K	+ 0
04h		1	0	0	+ 64K	+ 64K	+ 64K
05h		1	0	1	+ 80K	+ 64K	+ 64K
06h		1	1	0	+ 96K	+ 96K	+ 64K
07h		1	1	1	+ 112K	+ 96K	+ 64K

This example selects the below one megabyte base address D000:0h, using the I/O base 218h, and sets a 64K sliding window with a 64K offset:

```
outp (219h,01h);                    /* access CR#1, */
outp (218h,04h);                    /* PC mode, 64K window */
outp (219h,02h);                    /* access CR#2 */
outp (218h,0Dh);                    /* below 1 Meg RAM address */
outp (219h,04h);                    /* access CR#3 */
outp (218h,04h);                    /* set 64K offset */
...
...
outp (219h,80h);                    /* enable DPRAM */
```


Control Register #4

This write-only register selects the IRQ used to interrupt the system. Open-collector outputs allow more than one Hostess 186 controller to share the same IRQ. (The open-collector output is a feature not used by COMTROL's device drivers.)

The corresponding value for each IRQ appears in the next table.
Table 9. Control register #4 interrupt values.

Interrupt:	Value for Control Register #4:
IRQ3	08h
IRQ4	09h
IRQ5	0Ah
IRQ9	0Bh
IRQ10	0Ch
IRQ11	0Dh
IRQ12	0Eh
IRQ15	0Fh
Disabled	00h

This example selects the below one megabyte base address D000:0h, using the I/O base 218h, with a 64K window and a 64K offset, and selects IRQ11:

```
outp (219h,01h);                    /* access CR#1, */
outp (218h,04h);                    /* PC mode, 64K window */
outp (219h,02h);                    /* access CR#2 */
outp (218h,0Dh);                    /* below 1 Meg RAM address */
outp (219h,04h);                    /* access CR#3 */
outp (218h,04h);                    /* set 64K offset */
outp (219h,08h);                    /* access CR#4 */
outp (218h,0Dh);                    /* set IRQ11 */
outp (219h,80h);                    /* enable DPRAM */
```

CHAPTER 5 – Input/Output Addresses

Setting the I/O Addresses

The three-position DIP switch block on the controller sets the system I/O addresses. The controller reserves four consecutive I/O addresses, starting with the address set by the switches. Table 10 shows the possible I/O addresses and their switch settings.

Table 10. Hostess 186 I/O addresses.

I/O Address Range:	Dip Switch Settings 1 2 3	I/O Address Range:	Dip Switch Settings 1 2 3
218h - 21Bh		318h - 31Bh	
21Ch - 21Fh		31Ch - 31Fh	
238h - 23Bh		338h - 33Bh	
23Ch - 23Fh		33Ch - 33Fh	

Writing to I/O_Base + Offset

The Hostess 186 controller reserves four consecutive system I/O addresses: `I/O_base+0`, `I/O_base+1`, `I/O_base+2`, and `I/O_base+3`.

Table 11. Hostess 186 I/O map.

I/O Address:	Description:
`I/O base+0`	writes to control registers
`I/O base+1`	switches memory ON or OFF, control register index
`I/O base+2`	interrupts controller
`I/O base+3`	resets controller

Use `I/O_base+1` to switch the memory either ON or OFF, and as an index register when writing to the control register. A write of value 1 to bit 7 switches the memory ON. A write of value 0 to bit 7 switches the memory OFF. This example shows how to turn on the memory at the I/O base address 218h:

```
outp (0x219,80h);
```

The next example shows how to use `I/O_base+1` as an index register. This example first selects control register #4 at I/O base address 218h, and then selects IRQ 3:

```
out (0x219, 08h)          ; Access Control Register #4, keep memory disabled
out (0x218, 08h)          ; Set IRQ 3
```

Any access to the control registers after the controller is initialized have the following addresses.

Table 12. Index with RAM enabled or disabled.

Control Register:	Index with RAM enabled:	Index with RAM disabled:
1	81h	01h
2	82h	02h
3	84h	04h
4	88h	08h

Use `I/O_base+2` to interrupt the controller. Writing any byte value generates an interrupt to the controller. It is the controller's responsibility to clear this interrupt. This example shows how to interrupt a controller whose I/O base address is 218h:

writes using an `spl7()` kernel call.) This example shows how to reset a controller whose I/O base address is 218h:

```
outp (0x21b,0x00);          /* set the reset */
delay(HZ/10);               /* delay 1/10 second */
outp (0x21b,0xff);          /* remove the reset */
```

After removing the reset, you must wait between approximately five seconds (for the 128 Kbytes of memory) to allow the reset diagnostics to complete.