MACROTECH MI-286 AND COMPUPRO CPU 8085/88

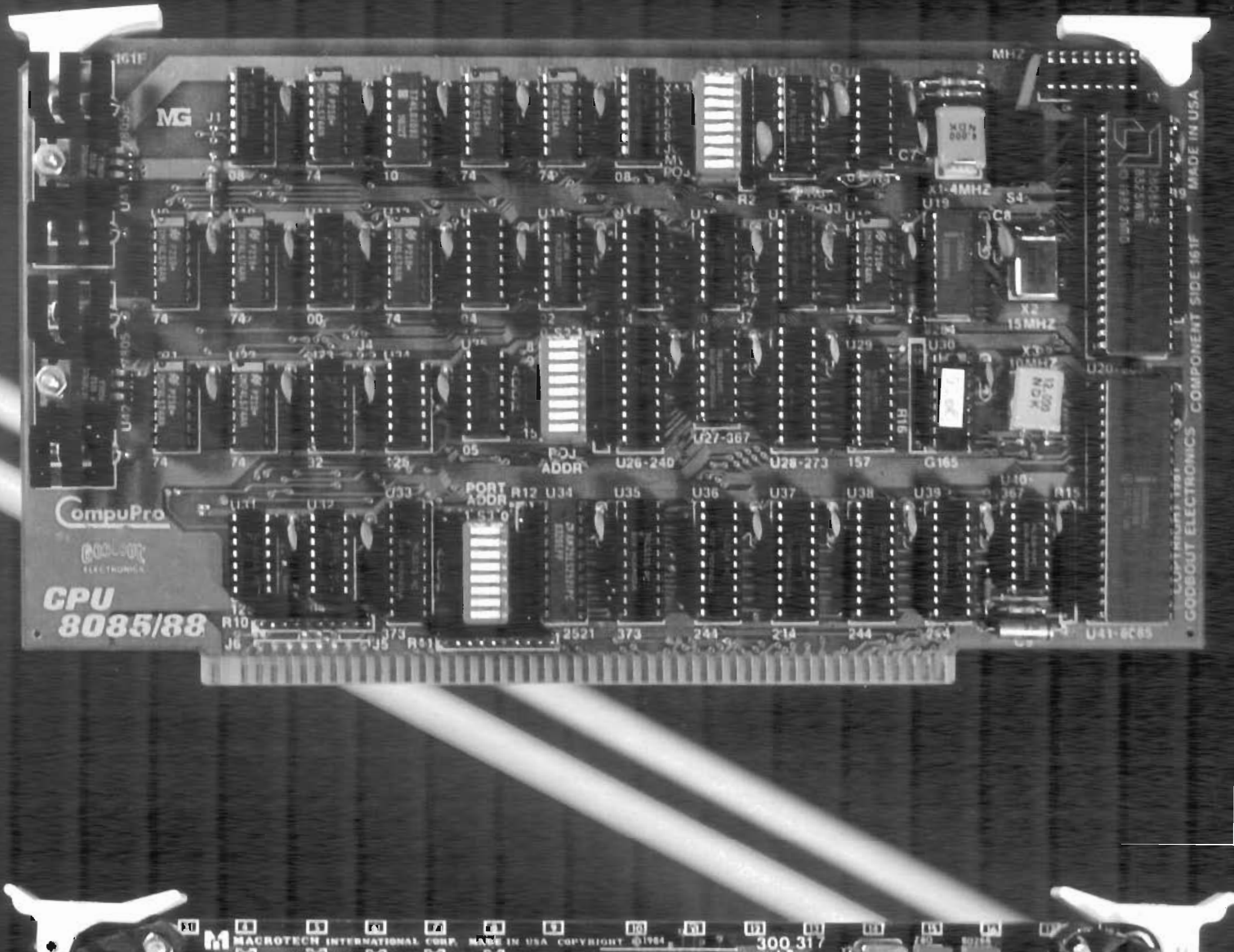
Jay Vilhena

	CompuPro	Macrotech
CPUs	16-bit: 8088 8-bit: 8085	16-bit: 80286 8-bit: Z80H
Math coprocessor	8087 (as a piggy-back board from Hudson Assoc.)	80287 optional
Maximum Memory Addressable	16 megabytes	16 megabytes
Data Bus	8 bits	8 bits (restricted) or 16 bits
Processor active on power-up	8085	Z80
8-bit speed	2 or 6 MHz	2 or 8 MHz
16-bit speed	10 MHz	6 MHz
I/O wait states	0 or 1 - switch selectable	Z80: 0 or 1 - switch selectable 286: up to 3 - switch selectable
MWRITE generator	switch selectable	switch selectable
Cost	$350	$1095
Manufacturer	CompuPro 26538 Danti Court Hayward, CA 94545-3999 800-842-7961 800-842-7962 (California)	Macrotech 9551 Irondale Ave. Chatsworth, CA 91311 800-824-3181 818-700-1501 (California)

Since so much of this issue is dedicated to dual-processor boards, I decided to throw in this small article with an at-a-glance table showing you the main features of the CompuPro CPU 8085/88 and the Macrotech MI-286.

Although 16-bit systems (and the CPU 8085/88 in particular) have been available for several years, many readers have 8-bit-only machines and may now be considering a 16-bit upgrade to run extra software. A dual-processor board is a good choice because all your older 8-bit software can still run. (Another alternative is to buy a 16-bit slave or coprocessor board available from several manufacturers.)

Both the Macrotech and the CompuPro are excellent dual-processor boards and are now essentially bug-free. The MI-286 offers outstanding performance due to a fast Z80 processor and a 16-bit data path for the 80286. The CPU 8085/88 can be obtained at extremely attractive prices. So the choice, as usually, depends on your intentions. If what you want is an economical and dependable upgrade to 16 bits, the 8085/88 is ideal. Since the 8085/88 uses an 8-bit data path for both processors, it will work fine with all your old 8-bit memory. However, if you are after the ultimate speed, want to take advantage of 16-bit memories, or plan to use the board in multiuser environments, you need the MI-286. You may also prefer the MI-286 if your current CPU is a Z80, since some Z80-specific programs may not run with the 8085.


CompuPro
CPU 8085/88
GODBOUT ELECTRONICS COMPONENT SIDE 161F
MADE IN USA
MACROTECH INTERNATIONAL CORP. MADE IN USA COPYRIGHT ©1984
300 317
DUAL PROCESSOR

(continued from page 13)
block. The 8088 registers are converted to the 8085 equivalents and written to the task block memory area. The 8088 also writes into the task block memory the function code for the requested BIOS function. The 8088 then issues an input instruction to activate the 8085.

3. The 8085 wakes up and loads all of its registers from the information in the task block memory, and calls the existing CP/M BIOS code for the requested function. When the existing BIOS returns, the 8085 writes all of its registers back to the task block memory area and in turn issues an input instruction that reactivates the 8088.

4. The 8088 reconverts, into their 8088 equivalents, the 8085 registers stored in the task block memory area. It then returns to CP/M-86.

Of course, things are not always as simple as one would like. The above scenario becomes more complicated when disk I/O is involved, as you will see when each BIOS function is examined in individual detail.

Memory Allocation

Memory must be carefully managed when both microprocessors on the Dual CPU are used because both processors can address and alter the same physical memory. It would be very easy (and probably disastrous) for one of the microprocessors to clobber information used by the other. Not only that, but CP/M and CP/M-86 differ significantly in their memory layout. CP/M uses low memory for a number of important fields, such as the BDOS and warm-boot vectors, and the IOBYTE. CP/M itself runs in high memory in the 8085 address space. CP/M-86 is forced by the architecture of the 8088 to use low memory for interrupt vectors. CP/M-86 itself is relocatable — that means it can run anywhere within the address space of the 8088. Typically, CP/M-86 occupies a memory area starting at address 400 (hex), with the BIOS starting at address 2500 (hex).

Somewhere in this messy picture, a secure area must be found for the BIOS task blocks. The memory allocation I used is pictured in Table 3. Low memory does cause some confusion due to conflicting usage by CP/M and CP/M-86. The code that processes the task block interface has to do some saving and restoring of key low-memory areas.

The Software Pieces

In order to implement the BIOS task block approach, I wrote three separate programs. The first program is the CP/M-86 BIOS. This is a special BIOS which runs on the 8088 and creates BIOS task blocks for the 8085 to process. The second program is called the BIOS Task Block Processor, and it runs on the 8085. It loads CP/M-86 into memory and causes the 8088 to start executing CP/M-86. A portion of the second program remains in memory to act as an interface between the CP/M-86 BIOS and the CP/M BIOS. The third program is a very simple program, called the CP/M Reboot Program, which runs on the 8088. It causes your computer to cease running CP/M-86 and resume running CP/M. This allows you to freely alternate between CP/M and CP/M-86 without the need to reset (reboot) your computer.

CP/M (8085)	CP/M-86 (8088)
A	AL
B	CH
C	CL
D	DH
E	DL
H	BH
L	BL

Table 2. *CP/M versus CP/M-86 Register Allocation. Whenever a BIOS call is invoked, processor registers are used to pass the parameters. This table depicts the translation conventions used in CP/M-86. For example, if a CP/M BIOS call used the 8085 register C to pass a parameter, the corresponding CP/M-86 BIOS call will use 8088 register CL for that same parameter.*

The CP/M-86 BIOS

Listing 1 (CBIOS86.A86) is a BIOS for CP/M-86. This code is really only a translator to allow your existing CP/M BIOS to do all the hard work. Let's examine each of the routines in this BIOS. During this discussion I will frequently refer to program labels which you can find in Listing 1.

The beginning of the BIOS contains several equates. A few of them are very noteworthy. The equate called IFAREA is used to locate the BIOS task block in memory (see Table 3). In Listing 1, the BIOS task block is located at D000 (hex). In the installation section of this article, you will learn how to locate the BIOS task block in your system. The definition of the task block format starts at the equate called CODE. The task block contains the CODE which determines the BIOS function the 8085 is to perform, the 8085 registers, the current IOBYTE and CDISK settings of CP/M (remember that CP/M and CP/M-86 conflict in their usage of low memory, so these important low-memory fields must be saved and restored), and a disk buffer (labelled DSKBUF). The disk buffer will be used by the CP/M BIOS whenever a disk read or write is requested by the 8088. The 8088 will then be responsible for moving the disk buffer's contents to wherever CP/M-86 has currently set its DMA address. Since the 8085 is not capable of addressing more than 64K, and the CP/M-86 DMA address can be outside of the bottom 64K of memory, a fixed disk buffer is required in the BIOS task block.

The BIOS INIT routine is very standard. It initializes all interrupt vectors, prints a sign-on message, and enters CP/M-86.

The console status (CONST) routine is the first place where anything unusual happens. This routine sets the CODE byte in the BIOS task block. The codes used are the offsets from the base of the CP/M BIOS jump table to the location of the jump for the requested BIOS service. Console status uses a code of 6 because the console-status jump is the third jump in the BIOS jump table; therefore, it is at offset 6 from the beginning of the table. Table 4 contains a listing of the BIOS functions and their CODES. After setting the code, the console status routine does a jump to the routine that will swap processors to perform the BIOS function.

Most of the BIOS functions are as simple as console status. The functions LISTOUT (list device output), LISTST (list device status), PUNCH (punch device output), READER (reader device input), HOME (cause the disk drive head to seek to track zero), SETTRK (select the track for the next disk operation), and SETSEC (select the sector for the next disk operation) all work the same as console status and will not be further discussed.

In CP/M-86 the IOBYTE was moved from low memory to a location inside the BIOS. Two new BIOS functions, GETIOBF (get IOBYTE) and SETIOBF (set IOBYTE), are provided. In this case, the IOBYTE itself is in the BIOS task block area so that the 8085 can examine it for any changes.

The SELDSK (select disk) routine is the most complicated routine in the CP/M-86 BIOS. Like most of the other BIOS routines, it starts out by setting the CODE byte and swapping to the 8085 to perform the SELDSK routine in the CP/M BIOS. If no errors were detected in the CP/M BIOS, a pointer to a Disk Parameter Header (DPH) is returned. The CP/M-86 BIOS uses the information returned to build a local copy of the DPH for CP/M-86 to use. When the CP/M BIOS returned its pointer to a DPH, it returned an absolute pointer to a location. Within the CP/M-86 BIOS, all references to memory are made relative to the 8088's DS segment register. (If the concept of segment registers is unfamiliar, you might want to read the series of articles on the architecture of the 8086 which have appeared in BYTE [4]. The 8086 and 8088 architectures are identical from a software viewpoint.) The DS register is always set to the base of CP/M-86, in this case 400 (hex). Before the pointer returned from the CP/M BIOS can be used, that pointer must be adjusted by subtracting the bias added by the DS register. This is accomplished by the instruction *sub bx,cpm-offset*. After the pointer is adjusted, the necessary fields are copied to the local DPH, and a pointer to the local DPH is returned to

CP/M-86. One of the fields in the copied DPH (the sector translate table pointer) must also be adjusted for the DS register offset. Incidentally, when I originally wrote this BIOS, I tried not making a local copy of a DPH and just passing to CP/M-86 the adjusted pointer to the DPH in the CP/M BIOS. This did not work properly, and I was not able to find out why.

The SETDMA (set DMA address) and SETDMAB (set DMA base address) BIOS functions are handled without calling the CP/M BIOS. These two functions save pointers to where disk records are found on disk reads or disk writes.

The READ function calls the CP/M BIOS to read one disk record. Remember that the CP/M BIOS uses a fixed disk buffer which is stored in the BIOS task block area. After calling the CP/M BIOS, the READ function transfers the data from the BIOS task block area to the address specified by the current DMA address. The WRITE function is very similar. Before calling the CP/M BIOS, the WRITE function copies the data from the current DMA address to the BIOS task block area. The CP/M BIOS is then called to do the actual disk write.

The CALLCB is the routine where the CP/M BIOS gets called. First, the register translation is performed according to the mapping shown in Table 2. Second, the IN AL,SWAP instruction shuts down the 8088 and wakes up the 8085. After the 8085 is finished processing the BIOS task block, it reawakens the 8088 which starts executing right where it left off. Lastly, the register translation is performed in reverse, and the routine exits.

The remainder of the CP/M-86 BIOS consists of data areas. The only one of any real interest is the *segtable*. This table defines for CP/M-86 what memory is available in the computer. In Listing 1, this table is set so the area between the end of the CP/M-86 BIOS and the BIOS task block is available for program use (see Table 3). If you have memory at or above address 10000 (hex), you will want to change this table to add the other memory areas. The first byte of the table tells how many entries are in the table. The remaining entries are two words each. The first word is the base paragraph address (physical address divided by sixteen) of the available memory; the second word is the number of paragraphs (16-byte chunks) available in this region. Up to eight noncontiguous memory regions may be defined.

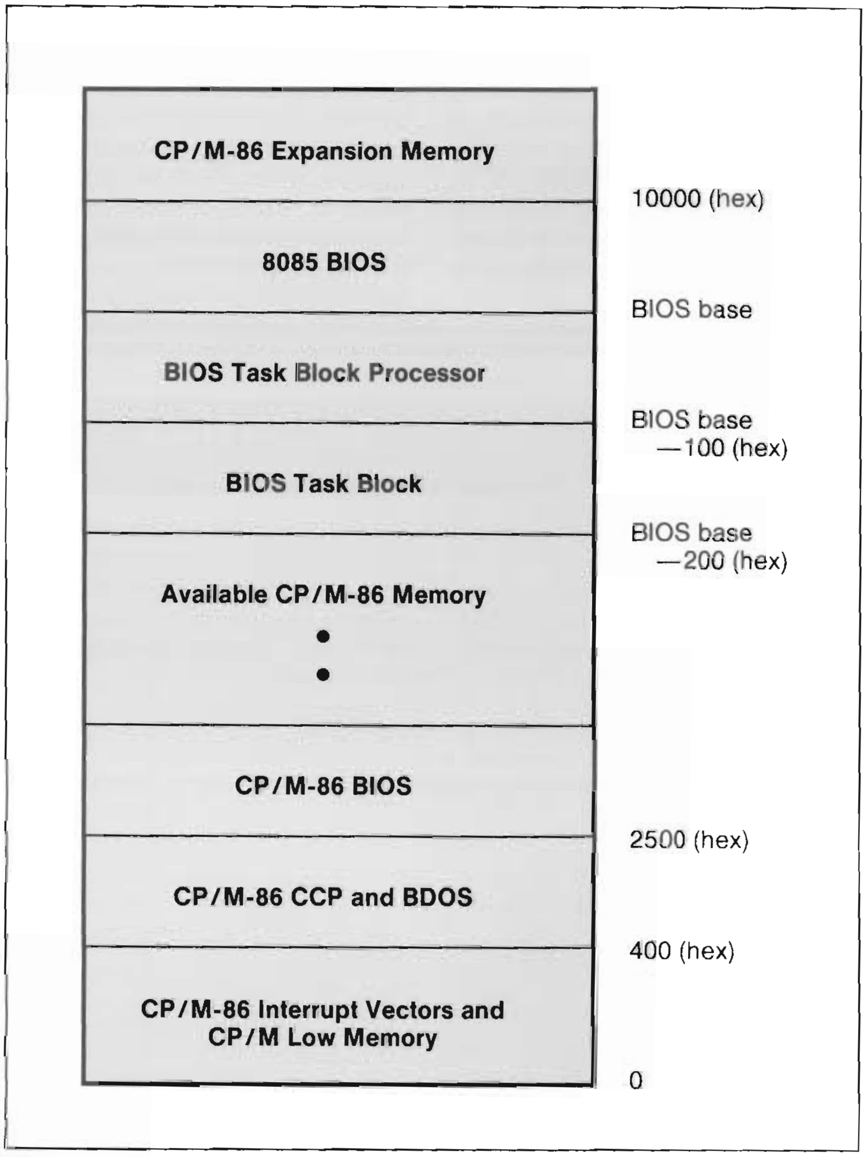


Table 3. *Memory Allocation. This table depicts the memory of the computer as it looks when CP/M-86 is running, and the 8085 is providing BIOS services for the 8088. The area between the top of the CP/M-86 BIOS and the bottom of the BIOS task block is available for running programs. If more than 64K of memory is available, any memory above the bottom 64K (memory above address 10000 hex) can also be used for running programs.*

The BIOS Task Block Processor

Listing 2 (BOOT86.ASM) is a program which runs on the 8085. This program has four responsibilities. Its first job is to load the CP/M-86 system file (a file containing the CP/M-86 CCP, BDOS, and BIOS) into memory. Second, it attempts to provide for the 8088 reset vector. Third, it saves part of the 8085 environment in the task block area so that some important variables are not smashed by CP/M-86. Fourth, it relocates a portion of itself to the BIOS task block area, and functions as the interface between the BIOS task block and the CP/M BIOS. The major sections of Listing 2 are identified by comments, and you may want to refer to each section during the following discussion.

Loading the CP/M-86 system file is straightforward. However, beware that object files under CP/M-86 have, as their first sector, an information sector on how to set segment registers after loading. This sector is of no use and must be discarded. Since the CP/M-86 file must be loaded at 400 (hex), loading is started at 380 (hex) to discard this first sector. The load code terminates at the label EOF.

The section of code that saves some of the important 8085 memory regions simply copies the contents of this memory to secure locations in the task block. The regions copied are those that might be smashed by the 8088 when running CP/M-86. The area at FFF0 is saved as part of providing for an 8088 reset vector.

The next section of code attempts to provide for the 8088 reset vector (the reset vector is an 8088 jump instruction to the address where the 8088 should start executing). This is a tricky problem. A restart vector must also be provided. The reset vector is used the very first time that the 8088 is turned on. The restart vector is used every subsequent time that the 8088 is turned on. The text box on page 28 details the problems of providing for reset and restart vectors on the Dual CPU.

Following the code for the 8088 reset vector, there is a short loop which relocates the remainder of the program to the BIOS task block area. This relocation places the remaining code in a secure area of memory (just after the task block). The relocated code must remain resident during CP/M-86 operation because it processes BIOS requests for the 8088.

CODE (hex)	Function
00	Cold Boot
03	Warm Boot
06	Console Status
09	Console Input
0C	Console Output
0F	List Device Output
12	Punch Device Output
15	Reader Device Input
18	Home Disk
1B	Select Disk
1E	Set Disk Track
21	Set Disk Sector
24	Set Disk DMA Address
27	Read Disk
2A	Write Disk
2D	List Device Status
30	Translate Logical Disk Sector to Physical Disk Sector
FF	Terminate CP/M-86 and Resume CP/M Operation

Table 4. *BIOS Functions and BIOS Task Block CODES. This table lists the codes used in the CODE field of the BIOS task blocks. The CODE field is used by the 8088 to inform the 8085 which BIOS function is to be performed. The values for the codes are actually the offset of the requested jump vector from the beginning of the BIOS jump vector table.*

This code is relocated to the BIOS task block area, rather than simply being loaded at the task block area, because it overlays a portion of the CP/M BDOS.

The definition of the task block area follows. This definition must match the definition provided in the CP/M-86 BIOS. The code that is relocated starts at the label HIMEM. All the jump instructions in this section look funny because of the relocation.

The code that starts at the label HIMEM processes task block requests. The first thing it does is to allow the 8088 to run. CP/M-86 has been loaded into memory, and the reset vectors have been set for the 8088 to begin running at the CBOOT (cold boot) entry point of the CP/M-86 BIOS. After the 8088 gets an initial chance to run, the memory at FFF0 which was altered to provide a reset vector is restored from the values saved in the task block area.

The code starting at the label NXTCMD begins a loop that will continue to execute for as long as CP/M-86 is running. NXTCMD is the entry point for processing task block requests. The CP/M-86 low-memory environment is pushed onto the stack and the CP/M low-memory environment is restored from the values saved in the task block area. A special check is made to see if the task block is requesting a return to CP/M operation (the task code for this special request is FF). If CP/M is to be resumed, the warm-boot vector is rebuilt in low memory, and a jump to 0 warm boots CP/M back into memory. If the task block is requesting a BIOS operation for CP/M-86, the

(continued on page 25)

PROGRAM LISTINGS

The listings for the three programs described in this article are published in the S-100 Journal Supplement distributed with this issue.

They are also available on standard IBM format 8" single-sided, single-density diskettes from Howard Spindel, 20877 S.W. Winema Drive, Tualatin, Oregon 97062. There is a handling charge of $30.00 which includes the disk and shipping by United States Postal Service.

(continued from page 19)
8085 registers are loaded from the information that the 8088 left in the task block area. Then, based on the task block CODE, the correct routine in the CP/M BIOS is called. Since the 8085 instruction set does not provide an indirect call instruction, the BIOS call is performed with a common 8085 trick. The return address is pushed on the stack, followed by the address which is to be called. A return instruction is then executed which actually calls the intended routine. The return instruction at the end of the called routine will pull the return address which was pushed on the stack.

After the CP/M BIOS returns to the task block processor, the 8085 registers are stored in the task block area. The possibly updated CP/M low-memory environment is saved again in the task block area. The CP/M-86 environment is rebuilt in low memory. Then the 8088 is allowed control again so it can use the information which the 8085 built in the task block area. When the 8085 next wakes up, it will begin processing by jumping back to NXTCMD.

The CP/M Reboot Program

Listing three (BOOT80.A86) is a program that runs on the 8088. It will cause CP/M-86 to terminate and CP/M to be warm booted. This program is quite short, but contains an interesting trick. A task block is built (in the task block area) which uses the task code FF to request that the 8085 task block processor warm boot CP/M. The trick is that this program must leave the 8088 executing at a known address so that, if CP/M-86 is restarted, the 8085 can write a restart vector into memory. As mentioned in the text box of page 28, I have used the convention that the 8088 will always start executing again at absolute address 10 (hex). It seems obvious that the way to leave the 8088 executing at address 10 is to put at address 0E a 2-byte IN instruction which swaps processors back to the 8085. However, the 8088 hardware has a feature which is called a *fetch ahead queue* (also called instruction pipeline) for its instructions. This feature allows the 8088 to look ahead in memory and ready some future instructions for execution, during the execution of another instruction. This means that, if an IN instruction were executed at address 0E, the 8088 would already have decoded the instruction at location 10. The 8085 will want to write a different instruction at location 10, but can't because the 8088 won't look at it. The way to solve this problem is to place several NOP instructions between the IN instruction and location 10. That way, the *fetch ahead queue* fills up with the NOP instructions, and the 8088 does not decode the instruction at location 10 until the 8085 has had a chance to write something there. This was one of the trickier problems to figure out when debugging these programs.

SYSTEM REQUIREMENTS FOR INSTALLING THESE PROGRAMS

• *CP/M 2.2 computer system.*
You must have CP/M 2.2 working in your computer using the CompuPro CPU 8085/88 (or Macrotech MI-286 — see the text box on page 13) as a

processor board. The BIOS code must not use any interrupt-driven code because the 8085 will not be available to handle interrupts when the 8088 is in control.

- *CP/M-86 1.1 distribution disk from Digital Research.*
- *CompuPro CPU 8085/88 (or Macro-tech MI-286) Dual-Processor board.*

The CompuPro CPU board must be strapped so that both the 8085 and 8088 reset-on-swap switches (labelled 5RS and 8RS on the circuit board) are OFF, and the extended address clear-on-reset switch (labelled XAC) should be ON.

- *Recommended minimum 48K CP/M System.*

You will rapidly discover that CP/M-86 programs tend to eat up core, and many programs probably will not run without additional memory boards (beyond 64K). If you plan to add memory past 64K, you may need to upgrade your current memory to respond to all 24 bits of the IEEE 696 (S-100) bus.

- *Provision for an 8088 Reset Vector.*

5 bytes of Global RAM Memory (memory which responds to CPU requests without checking the upper 12 S-100 address bits) at address FFF0, or, alternatively, an EPROM installed at address FFFF0 (or FFF0 if FFF0 is global memory) which contains a 5-byte 8088 instruction to execute a far jump to absolute address 0:0. This memory requirement is necessary to provide a restart vector for the 8088 on initial power-up (as discussed in the text box on page 28).

INSTALLATION INSTRUCTIONS

Installation of these programs (once you get them typed in!) is fairly simple. Usually, the only customization that will be necessary for your system is to determine where the task block will reside in memory.

Examine the three source files (BOOT86.ASM, CBIOS86.ASM, and BOOT80.ASM). Near the beginning of each of the files, there is an equate called IFAREA. In the listings, there is a constant 0D000H as part of the calculation of IFAREA. There may be other terms involved in the calculation, but the 0D000H term is what you will need to change in order to make these programs run on your system.

In each of the source files, the term 0D000H should be changed (using your favorite editor) to become the base of your CP/M BIOS minus at least 200 (hex). If you do not know the base of your BIOS, use the DDT program to list (L command) the code at 0 in your system. Take the address of the jump listed at 0, subtract 3, and you have the base of your BIOS. Now subtract 200 (hex), and edit the resulting number into the source files where it now has 0D000H in the IFAREA equate. Note that all three source files must be set exactly the same!

GLOSSARY

This article contains some of the jargon with which all computer articles seem afflicted. To help explain some buzzwords, here is a short glossary.

CP/M (also called **CP/M-80**)
Control Program for Microcomputers. This is a widely used operating system provided by Digital Research. It runs on 8080, Z80, and 8085 microprocessors.

CP/M-86
This is a version of the CP/M operating system which runs on the 8086 family of microprocessors (8086, 8088, 80186, 80286). Data files are stored in the same format as CP/M, allowing disk compatibility between the two operating systems.

CCP
Console Command Processor. This is the first of three parts of the CP/M family of operating systems. The CCP is responsible for processing keyboard input and generating the appropriate calls on the other parts of CP/M.

BDOS
Basic Disk Operating System. This is the second of three parts of the CP/M family of operating systems. The BDOS is primarily responsible for maintaining all of the disk structures (directories and files) and allowing an easy, structured access to the disks. The BDOS will also make appropriate calls to the BIOS.

BIOS
Basic Input Output System. This is the third of three parts of the CP/M family of operating systems. The BIOS contains all the drivers for any hardware devices in a CP/M system. All the machine dependent code in the CP/M operating system is isolated in the BIOS. When CP/M is ported (moved) to a new machine, only the BIOS needs to be rewritten.

DDT
Dynamic Debugging Tool. This is a program supplied with CP/M which allows users to examine memory and interactively debug programs.

STAT
Another program supplied with CP/M which allows the user to configure the current active devices, control some disk parameters, and generally report system status.

Warm Boot
When CP/M (the 8080 version only) is running, the CCP and BDOS may be destroyed by a running program in order to gain more useful

If your dual CPU board is set so that the processor swap port is not the standard value of 0FD (hex), you will also need to change the equate for SWAP found near the front of each source file. No other changes should be necessary to customize these programs for your system. You may want to change some of the messages (the logon banner is a good example) in CBIOS86.A86. If you have more than 64K of memory in your system, you will want to eventually change the *segtable* memory table entries in CBIOS86.A86. *Segtable* entries were briefly discussed above in the description of the CP/M-86 BIOS. The CP/M-86 Operating System Guide, provided by Digital Research with CP/M-86, contains complete information on how to change the *segtable* entries.

memory space for running the program. A special BIOS function, called the Warm Boot, may be called by the program to cause the CCP and BDOS to be reloaded into memory from the disk.

Warm Boot Vector
A special jump stored at address 0 in the 8080 version of CP/M which is used to activate the Warm Boot routine in the BIOS. A program wishing to cause a Warm Boot need only jump to address zero.

IOBYTE
A byte of memory which contains the current active device mappings for CP/M. There are four logical devices in CP/M, the console, the reader, the punch, and the list device. The IOBYTE allows each of the four logical devices to be mapped to one of four physical devices controlled by the BIOS. The STAT program is used to examine or change the setting of the IOBYTE.

CDISK
A byte of memory which contains the current default disk being accessed by CP/M.

DPH
Disk Parameter Header. A table of information about the disk which is stored in the BIOS and used by the BDOS. The disk parameter header contains pointers to a sector translation table, a DPB, and scratchpad areas for use by the BDOS.

DPB
Disk Parameter Block. Another table of information about the disk which is also stored in the BIOS and used by the BDOS. The DPB contains several fields which determine the storage capacity of the disk, how many directory entries the disk can contain (and therefore how many files), and some other information which allows disks of varying capacities and capabilities to be used by CP/M.

Sector Translation Table
A table which is used by CP/M to convert logical disk sector numbers into physical disk sector numbers. The physical sectors of a disk are usually numbered consecutively on each track. In order to minimize rotational delays when accessing disks, it is usually necessary to avoid accessing physical sectors consecutively. Consecutive logical sectors in a file (as maintained by the BDOS) are therefore rarely stored as consecutive physical sectors on the disk. Given a logical sector number, the sector translation table tells how to find the physical sector on the disk.

DMA Address
Disk Memory Address. The address of a 128-byte buffer which is used to contain one disk sector on any disk read or disk write operation.

Assuming that the address equates are now correctly set for your system, it is time to assemble the sources. BOOT86.ASM is compatible with the standard Digital Research 8080 assembler (ASM); CBIOS86.A86 and BOOT80.A86 are compatible with the 8086 assembler distributed on Digital Research's CP/M-86 distribution disks (ASM86). To assemble and link the set of programs, perform the following steps:

```
ASM BOOT86
LOAD BOOT86
ASM86 CBIOS86
PIP CPMX.H86=CPM.H86,CBIOS86.H86
GENCMD CPMX 8080 CODE[A40]
ASM86 BOOT80
GENCMD BOOT80 8080
```

CPM.H86, ASM86.COM, and GENCMD.COM will be found on the CP/M-86 distribution disk. ASM.COM and LOAD.COM will be found on the CP/M distribution disk. Note that the above entire sequence can be performed while running under CP/M because Digital Research thoughtfully provides both CP/M and CP/M-86 versions of ASM86 and GENCMD. It would be inconvenient, to say the least, to require running ASM86 and GENCMD under CP/M-86 to bring up CP/M-86. The filename CPMX in the above sequence is an arbitrary choice; you may call it anything you like. All of the other filenames are fixed.

Lastly, make sure that your system is providing for the restart vector needed by the 8088. Reread now the system requirements section and the text box of page 28 to determine how to provide for the restart vector. If you will be providing an EPROM with an 8088 far jump to 0:0, note that the codes to program into your EPROM (in hexadecimal) are EA,00,00,00,00.

Your system should now be completely configured to run CP/M-86.

OPERATING INSTRUCTIONS

Booting your CP/M-86 system is very simple with the provided programs. Under CP/M, execute the command:

```
BOOT86 CPMX.CMD
```

The CPMX.CMD file is the CP/M-86 system file which you generated using ASM86 and GENCMD. Note that since BOOT86 accepts a filename input, you may command BOOT86 to pick between multiple CP/M-86 versions on a single disk.

After some disk access time, you will see the CP/M-86 sign-on message, followed by a familiar-looking CCP prompt. You are now running CP/M-86. Pat yourself on the back, and play with CP/M-86 for a while.

To return to CP/M-80 at any time, execute (under CP/M-86) the following command:

```
BOOT80
```

This will warm boot your CP/M-80 system back into memory (make sure that a disk with your CP/M system on the system tracks is inserted in drive A:). You may use BOOT86 and BOOT80 to swap back and forth between operating systems as often as you wish. There is no need to reset your computer between operating system swaps.

These programs are set up in a way that preserves the currently logged disk across operating system swaps. This means that if your system is displaying a B> prompt under CP/M, it will also display B> after booting in CP/M-86. The logged disk is again preserved when using BOOT80 to return to CP/M.

The programs will also preserve IOBYTE changes across operating system swaps. If you use STAT to change the IOBYTE assignments while running CP/M-86, you will find that the CP/M IOBYTE also reflects the change you made.

If you are using another package besides CBIOS86 to run your 8088, it will be necessary to reboot (front panel reset) your computer if the other package has been executed since the last reset. This is necessary so that the 8088 will be in a known state at the start of BOOT86 execution. Most likely, it will also be necessary to reset your computer if you want to run another 8088 application after running these programs.

TIPS IF YOU HAVE PROBLEMS

This section is intended to give some additional guidelines in case these programs do not work immediately.

1. Have you ever executed code on your 8088 before? It is possible that your 8088 does not work.

Many CPU 8085/88 boards are shipped with the 8088 set up to run at 8 MHz, and this may be too fast for your memory boards or other system components. This could be your problem. It may be corrected by replacing the 8088 crystal on your processor board with a slower crystal (remember that the 8088 uses a crystal three times the desired operating frequency).

2. Make certain that the IFAREA equates in all of the source files are exactly the same!

3. Be sure the switches on your CPU 8085/88 are set the way that the above system-requirements section specifies.

4. These programs have to maintain a low-memory (0-100 hex) environment for CP/M. For almost all users, this means rebuilding the IOBYTE and CDISK areas after every processor swap between the 8085 and the 8088 (since CP/M is not running, it is not necessary to rebuild the BDOS jump or the BIOS warm-boot jump). If your BIOS for the 8085 is using any of the low memory areas, other than IOBYTE and CDISK, you will probably have problems. To verify if this is the case, edit your CBIOS86.A86 file to change the equate for SAVLOWMEM to "true." This will cause the CP/M-86 BIOS to save and restore all 256 bytes of the low-memory area. Now go through the installation steps again and try running the system. You will notice a substantial reduction in the operating speed of your system. But, if the problems go away, then they are caused by your 8085 BIOS using some of the low-memory areas.

The reduction in operating speed makes saving all 256 bytes an undesirable long-term solution. The best solution is to rewrite the 8085 BIOS, so that it does not use any low-memory areas except the IOBYTE and CDISK. Another possible solution is, first, to determine the addresses that need saving and restoring. Second, edit the BOOT86.ASM file to save and restore those locations in a manner similar to the way that the IOBYTE and CDISK are handled (this requires some skill with 8080 assembly language).

5. Have you correctly provided for an initial starting vector for the 8088 as detailed in the system requirements?

6. If you have created a very large version of your CP/M-86 BIOS, you may need to change the ORG 6000H statement in BOOT86.ASM. BOOT86 assumes that the last address of the CP/M-86 BIOS will be less than 6000H. If it is more than 6000H, then a portion of the BOOT86 program will get clobbered when CP/M-86 is loaded in. If the last address of the CP/M-86 BIOS is greater than 6000H then change the ORG 6000H statement in BOOT86.ASM to be anything greater than the last address of the CP/M-86 BIOS. But, it should still be small enough for your CP/M loader to be willing to load it (small enough so that it loads below the BDOS in your system).

SUGGESTION FOR ENHANCEMENT

You may notice that your newly-running CP/M-86 system is slower than your CP/M system. This is largely due to the CP/M-86 BDOS which runs a lot more slowly than the CP/M BDOS. Swapping microprocessors back and forth to do I/O operations does cause some additional overhead processing which may further slow down CP/M-86. To eliminate some of the swapping overhead, you can begin to rewrite portions of your BIOS in 8088 assembler and to incorporate them directly into the CBIOS86.ASM file. One easy change, likely to have a noticeable effect, is to rewrite your console output driver. Typically the console output driver is a very easy portion of the BIOS to rewrite.

APPLICABILITY TO OTHER MACHINES

While the programs presented here are specifically tailored to the CompuPro Dual CPU, the idea of using a BIOS task block interface should be generally useful with any dual-processor board. For example, the same approach could be taken to bring up CP/M-68K on the Z80/68000 dual-processor board made by Cromemco. This approach might also be useful to someone running a Zenith Z-100 which uses an 8085 and an 8088. Operating systems like MP/M 816 (furnished by CompuPro) use similar techniques in reverse — the 8088 is used as an I/O processor for the 8085. Using the 8088 as an I/O processor has an additional advantage because the 8085 BIOS becomes very small, allowing a larger CP/M transient program area.

CONCLUSION

This article has shown a way to bring up CP/M-86 on older S-100 computers. The method presented can result in considerable savings in cost, time, and effort over alternative methods of getting into 16-bit processing. The running system generated with this method is suitable for long-term use. It is also suitable for use as a bootstrap to other implementations of 16-bit operating systems. The most difficult part of upgrading to a 16-bit operating system — getting the initial system to work — has been greatly simplified. ■

References

1. Bray, David W. Upgrading Older S-100 Computers to the CompuPro Dual Processor. *Microsystems*, Vol. 4 No. 9, September 1983; page 80.
2. Ratoff, Bruce R. The Godbout Dual Processor Board and CP/M-86. *Microsystems*, July/August 1981.
3. Kalish, Richard L. Upgrade Problems and Solutions. *CompuPro Product Users Manuals*, Vol. 2, January/September 1981; page 4.
4. Heywood, Stephen A. The 8086 — An Architecture for the Future. *BYTE*, Vol. 8, No. 6, June 1983; page 450 (part 1). *BYTE*, Vol. 8, No. 7, July 1983; page 299 (part 2). *BYTE*, Vol. 8, No. 8, August 1983; page 404 (part 3).

The Macrotech MI-286 Dual Processor Board

Although this article was originally written with the CompuPro 8085/88 Dual Processor in mind, through the good will of S-100 Journal and Macrotech I recently had the opportunity to test my program for a few hours with the Macrotech MI-286 Dual Processor. (The Macrotech MI-286 uses an Intel 80286 and a Zilog Z80 combination and is advertised as a direct plug replacement for the CompuPro Dual Processor.)

Since the MI-286 manual said that it came pre-configured as a CompuPro replacement, I pulled out my CompuPro board, put the Macrotech board in, and hit the power button. I was gratified to see the system boot CP/M 2.2 as usual. Next I tried running the BOOT86 program to fire up CP/M-86. My system crashed. I hit the reset button and tried it again, and this time CP/M-86 booted up and ran! A few more tries showed me that the system was a little flaky. I had been afraid of this because my memory boards are older 8-bit-only boards and I thought they might not be fast enough for the 80286. However, Macrotech had the foresight to also lend me the V-RAM, a 512K-byte static RAM board. I flipped through the V-RAM manual, configured the V-RAM board for 512K of system memory at address zero, yanked out my RAM boards, and put in the V-RAM. Power on, run BOOT86, and I've got a reliably running system!

I spent a little bit of time playing around with CP/M-86 and marvelling at the extra speed of the Macrotech board. The MI-286 has LEDs which show which processor is active. It was really fun to try different things and watch the relative brightness of the Z80 and 80286 LEDs shift back and forth.

I then ran BOOT80 to switch back to CP/M and decided to run BOOT86 again to make sure that the restart vector worked. Oops, crashed again. A couple of retries convinced me that this was a software failure. A little more thinking and the answer came to me — I'd been bitten by the fetch-ahead queue again. The 80286 has a much larger fetch-ahead queue than an 8088, and the 80286 was reading the restart vector before the Z80 could write it. The fix is simple and is left "as an exercise for the reader." For the time being, I decided to just live with having to reset the computer every time before booting CP/M-86.

About this time, I decided to read the MI-286 manual more thoroughly, and discovered that I could enable memory wait states. This sounded like just the thing I needed to use my older memory boards. A quick jumper change and a couple of board swaps later I had a reliable system using my older memory boards.

I also wanted to get some idea of how much faster the MI-286 was than the 8085/88. For a simple test, I put both CBIOS86.A86 and ASM86.CMD on my Digital Research RAM disk and assembled the BIOS. I got the following results (hand timed with a stopwatch):

MI-286 with Macrotech 16-bit memory	15 seconds
MI-286 with my 8-bit memory (1 wait state)	28 seconds
MI-286 with my 8-bit memory (0 wait states)	21 seconds
CompuPro CPU 8085/88	27 seconds

The above table shows that the system RAM has an enormous effect upon the system performance. To use the MI-286 effectively, you should plan on using it with 16-bit memory boards.

I also became curious about how much performance degradation was incurred by using the Z80 to drive the BIOS routines as opposed to having true CP/M-86 BIOS routines. Recently, I broke down and bought and installed CP/M-816 from Viasyn. I tried booting CP/M-816 using the MI-286 with the Macrotech memory board and it worked perfectly. I then tried the above timing test and found that the assembly took 14.5 seconds. This is only slightly faster than the CBIOS86 approach and may in fact be within the error limits of hand timing.

Dear Macrotech,
Your boards were wonderful to use and very fast. I truly hated to ship them back to S-100 Journal. Since they are technically used equipment now, would you consider selling them to me inexpensively? Pretty please?

Howard Spindel