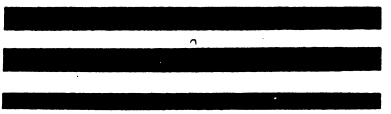
Baby Blue™

CPU Plus User's Manual

Version 1.2



Xedex Corporation
222 Route 59
Suffern, New York 10901
(914) 368-0353

Please refer to the addendum portions of the Manual (pages i - ix) for updated instructions regarding the CONVERT program.

Please read addendum pages x through xii for a description of our newest Conversion Software 3.Ø, which now offers Communications Capabilities with other CP/M computers.

We hope these upgrades will make Baby Blue an even more valuable addition to your IBM Personal Computer.

Xedex Corporation.

Please refer to the addendum portions of the Manual (pages i - ix) for updated instructions regarding the CONVERT program.

Please read addendum pages x through xii for a description of our newest Conversion Software 3.Ø, which now offers Communications Capabilities with other CP/M computers.

We hope these upgrades will make Baby Blue an even more valuable addition to your IBM Personal Computer.

Xedex Corporation.

Preface

HOLD IT!

We want this experience to work out well for you.

If you've been tinkering with microcomputers since the early Altair days (say 1975), you can *probably* take Baby Blue in stride and proceed at will.

But if you're new to the whole microcomputer game, or very new to the IBM Personal Computer, *slow down*. DO NOT start this project at a time like a weekend evening when your PC dealer, your Baby Blue vendor and Xedex Corporation staff will not be available to you.

It's not that you might do anything so disastrous – except damage your confidence in yourself as master of your computer. That would be bad enough, and we don't want it.

How would you feel about taking delivery on a 2-10-10-2 Mallet steam locomotive without having some prior experience with the type, or with locomotives in general? Exactly. Computer hardware and software, when they do anything interesting at all, are considerably more complicated than the most elaborate steam locomotive ever built. So be patient with our product and with yourself, and be in a position to get help if you should need it.

If you're new to all of this, as everyone once was, you can make some silly mistakes. A *stupid* mistake, by contrast, is a silly mistake that never gets corrected. We all have to be silly from time to time – it's in the Fundamental Charter of the universe – but we don't really have to be stupid.

Take your time, choose a time when help can be reached, and don't beat your head against walls if help is not immediately available. Take a break; help *will* be available later.

We want you to succeed. We even want you to enjoy it.

Xedex Corporation

Inventory Checklist

The complete Baby Blue CPU Plus package should contain the following:

- ☐ Printed-circuit board with mounting bracket
- ☐ 5¼-inch floppy diskette in sleeve
- ☐ This manual (includes limited-warranty statement)

If anything is missing or appears to be damaged, inform your vendor immediately.

Table of Contents

Introduction

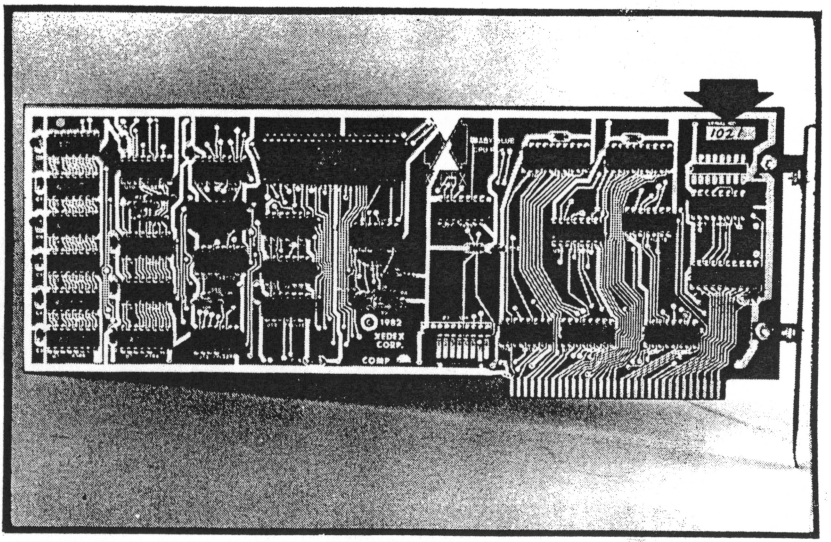
You've made a smart move. With Baby Blue CPU Plus, your IBM Personal Computer – a fine machine in itself – is no longer in a software vacuum.

We want coming in from that vacuum to be a hassle-free experience for you, and we have tried to write this manual accordingly. The topics it covers are:

1. Installing the Baby Blue printed-circuit board
2. The fast way to start *using* Baby Blue
3. The fine points (which many users will never need)
4. Operating systems in general and the importance of CP/M

Then there are the Appendices, which seem the best place to pile up the kind of technical data that you will rarely need, but will want to find quickly when you *do* need it.

As you've guessed, this manual is intended for insertion into an IBM binder. We suggest that you put these pages into either your *Guide to Operations* manual, behind the "Operation" tab, or the *Disk Operating System* manual, just behind the section on Debug.



To register your purchase of Baby Blue for warranty purposes, call us at 914-368-0353 and give us your name, address and serial number. Your Baby Blue serial number can be found in the upper righthand corner of the front (component) side of the printed-circuit board (see photo). Our 90-day limited warranty is spelled out at the

back of this manual. Should you need warranty service, notify your vendor or call us at the number above; we cannot accept unauthorized returns.

Whenever software updates or hardware upgrades to Baby Blue become available, you, as a registered owner, will automatically be informed. Software updates will be made available at a minimal charge in keeping with our duplication, handling and shipping costs; hardware upgrades will be offered to existing Baby Blue owners at the most favorable possible terms.

We want to know what you think of Baby Blue and of this manual, and we want your suggestions for future enhancements. Use the comment form at the back (and please feel free to photocopy and reuse it as many times as you wish) or just write to:

Xedex Corporation
222 Route 59
Suffern, New York 10901
(914) 368-0353

Tell us about your gripes, your inconveniences and your agonies. Tell us about your delights, your glories and your major and minor triumphs, too. Let us hear from you.

Installation

First, a reminder: complete your warranty registration by telephone, as explained in the Introduction.

The only tool you should require is a screwdriver, though you can use a 3/16-in. wrench or nutdriver to tighten the screw that holds the mounting bracket in place.

The Baby Blue printed-circuit board is not especially sensitive, but you should be reasonably gentle in handling it (careless or rough handling *can* damage it). *If* your Personal Computer is in a high-static environment — the kind in which you get a shock every time you touch a doorknob — you should ask your vendor about antistatic sprays and special handling precautions.

Baby Blue will work with a PC that has any amount of memory of its own from 64K bytes up to 544K bytes. We do *not* recommend using Baby Blue with a system that has less than 64K of its own.

To begin the procedure: turn the system OFF, disconnect all power cords from wall sockets, and disconnect all peripheral devices from the System Unit; take your monitor off the top of the System Unit cabinet.

Getting a Slot: Address yourself to the rear panel of the System Unit. There you'll find the two screws that hold its cover in place (Figure 1). Remove these, then remove the cover by sliding it forward (toward the front of the unit) and then up (Figures 2 and 3).

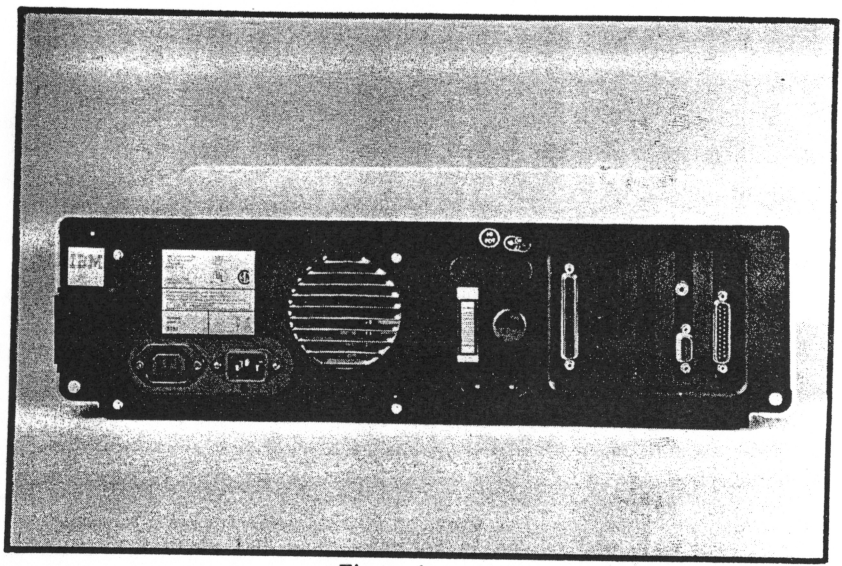


Figure 1

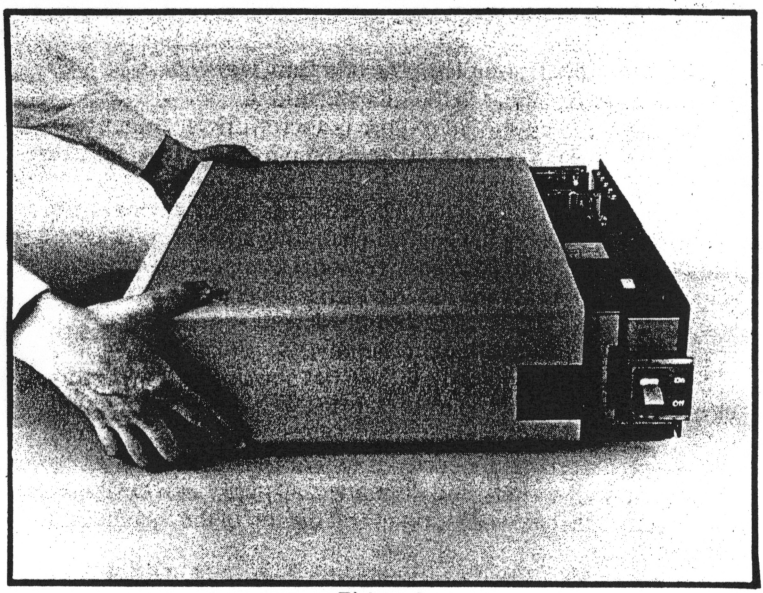


Figure 2

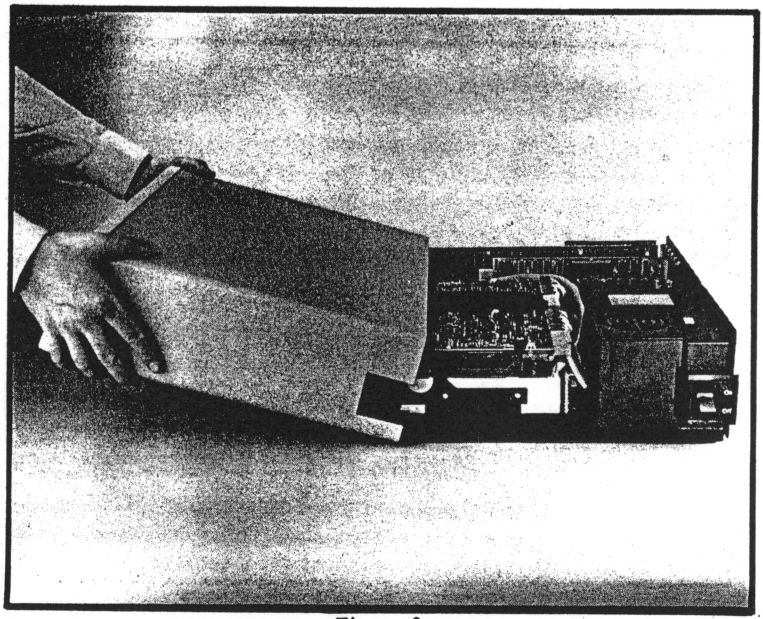


Figure 3

System expansion slots are found at the left rear of the System Unit, as seen from the front. Baby Blue will work in any available (vacant) slot, but we suggest you avoid the leftmost slot because of the (slight) possibility of electrical contact between our board and loudspeaker-mounting hardware.

Remove the back-panel slot cover — L-shaped strip of metal — for the vacant slot you've chosen: unscrew (or draw with nutdriver) the retaining screw (Figure 4), then lift the slot cover clear (Figure 5). Save the screw; do as you please with the slot cover.

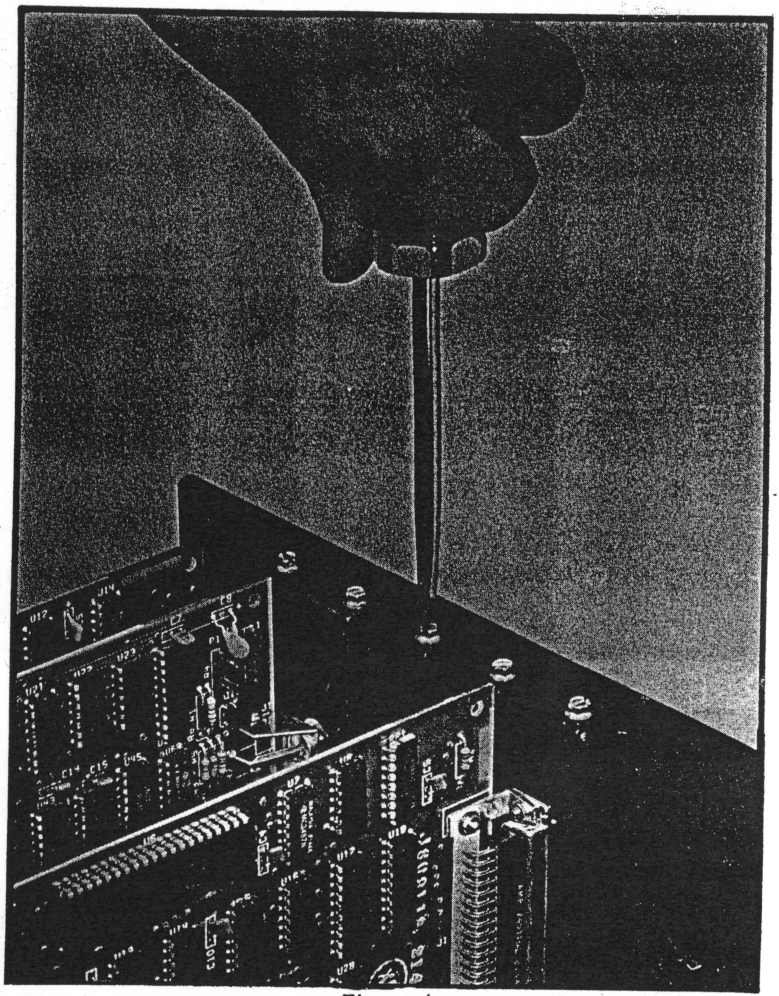


Figure 4

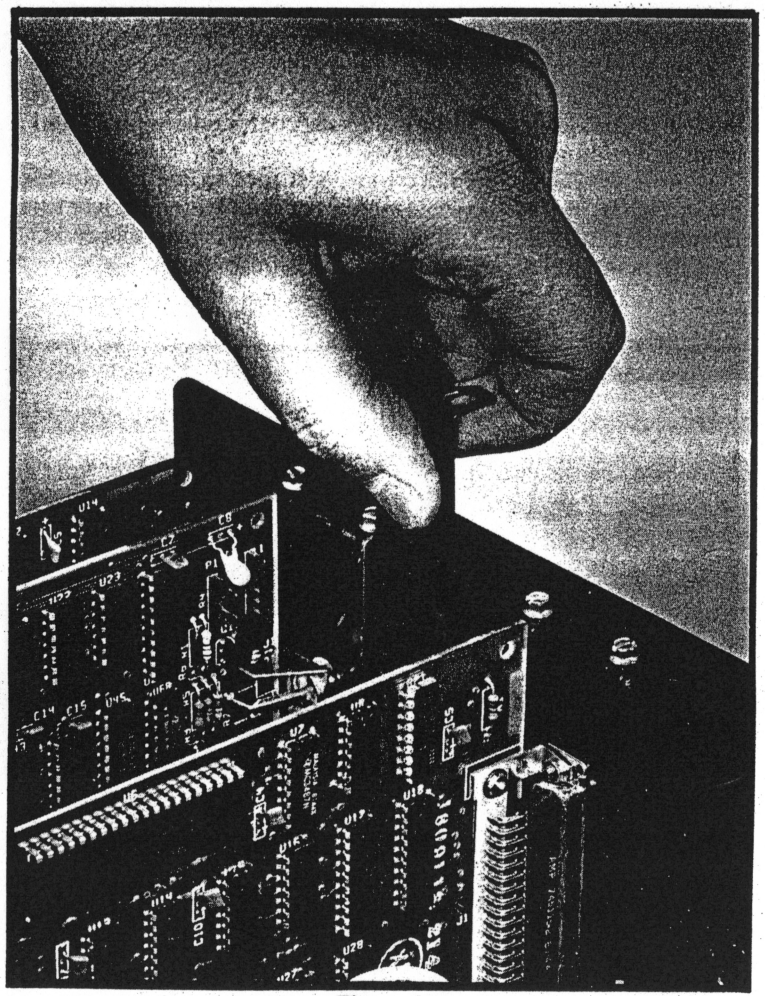


Figure 5

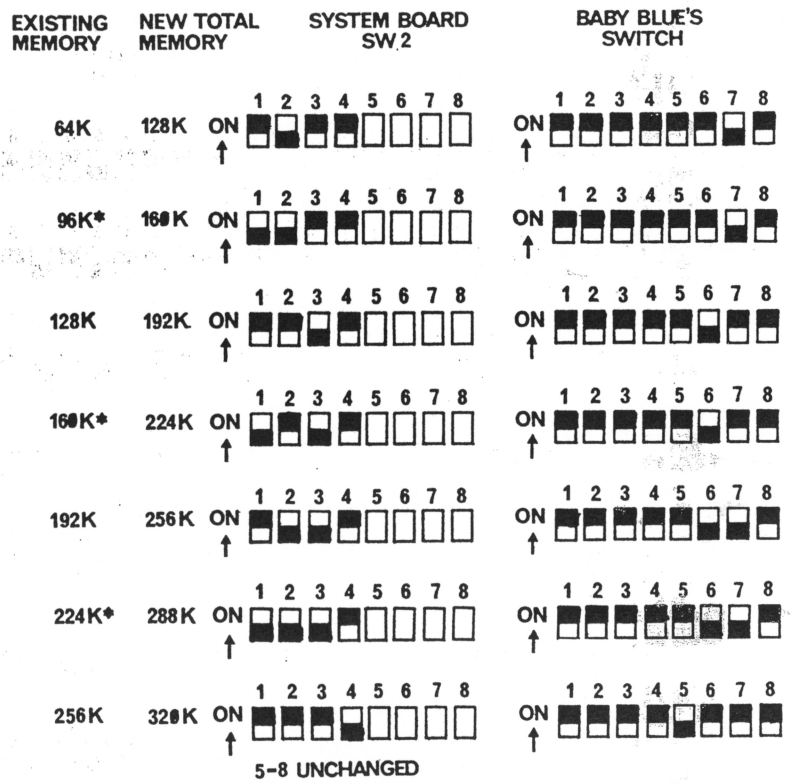
(This is about the halfway point. If you're one of those who are nervous about fooling around with the innards of computers — and we've all been in that position at least once — then congratulations on your coolness so far.)

Switch Settings: The only thing that's user-adjustable on the Baby Blue board is an eight-position DIP (dual inline package) switch. The positions are numbered 1-8; on the righthand side of the switch package you'll see a legend showing which way is on and which way is off for these switches. (On should be up, but check switch-package markings; "in," if the package is so marked, is the same as "on.")

There are two similar DIP switches on the PC's system board (the large horizontal printed-circuit board at the bottom of the System Unit), referred to as SW1 and SW2.

On SW1, positions 1 and 2 and 5-8 are always left unchanged. On any system with at least 64K to begin with make sure that positions 3 and 4 are both OFF. (SW1's positions 3 and 4 reflect the amount of memory present *on the system board*. Note that you'll never have more than 64K on the system board itself.)

Positions 1-4 on SW2 tell the system how much expansion memory, including Baby Blue, it has to work with; positions 5-8, which tell the system other useful things, are not to be changed.



*RECONFIGURE 32K BOARD FOR ADDRESS ABOVE BABY BLUE.

Figure 6

Consult IBM documentation and switch-package markings for correct on and off positions on these switches.

Figure 6 shows appropriate settings of SW2 and Baby Blue's own DIP switch for different amounts of *existing* (pre-Baby Blue) memory. (If you're installing Baby Blue in a system with more than 480K already, call us for details.)

The commonest situation is a system with 64K before Baby Blue. Summarizing for that situation *only*: SW2 has position 1 on, position 2 off, positions 3 and 4 on, positions 5-8 unchanged; Baby Blue's DIP switch has positions 1-6 on, position 7 off and position 8 on.

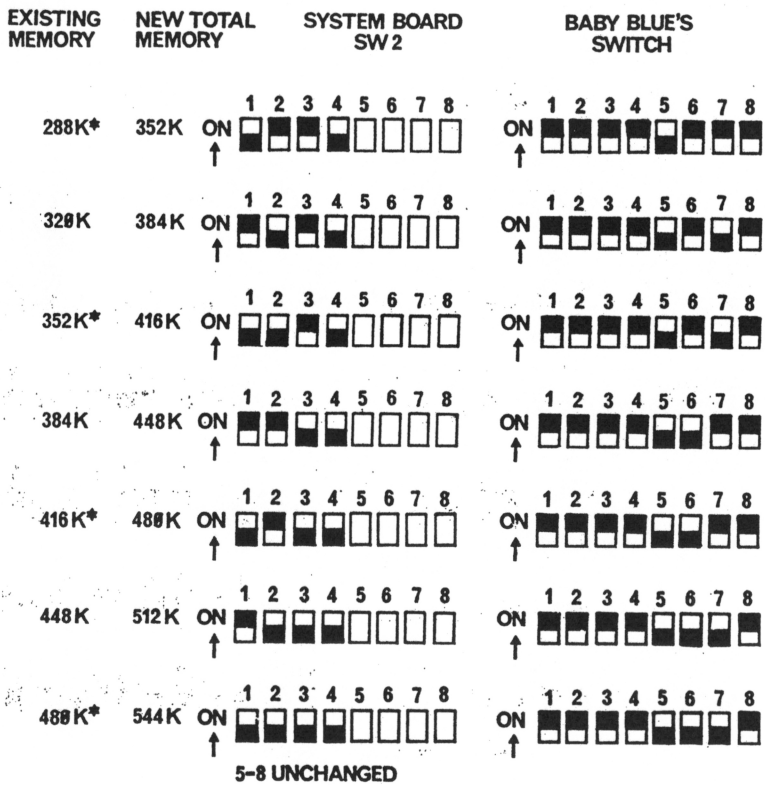


Figure 6 (continued)

(If you're technically-minded and/or hyperalert, you will have noticed that we're installing Baby Blue, in each case, at the top of available memory. As a reward, here are some more goodies for you: positions 1-3 on our DIP switch represent the I/O address, analogous to a telephone number, used by the 8088 to communicate with our Z80B; these positions should all be on except in some unusual cases of conflict with other I/O devices — we will issue a technical bulletin covering those cases if there is sufficient user demand. Positions 4-7 define the memory-segment address for Baby Blue's onboard RAM — which of 16 possible locations it occupies within 8088 memory space. Position 8 enables or defeats parity checking — memory-error detection — for the onboard RAM; it will be discussed further in the troubleshooting appendix.)

Please recheck all your switch settings before proceeding.

Plug-in: Ease the mounting bracket's tongue into the gap between the system board and the back panel and press the Baby Blue board into the expansion slot (Figure 7). The board should seat firmly — you should *feel* that it has gone all the way into the socket. Be careful not to disturb any other expansion-option boards in the process.

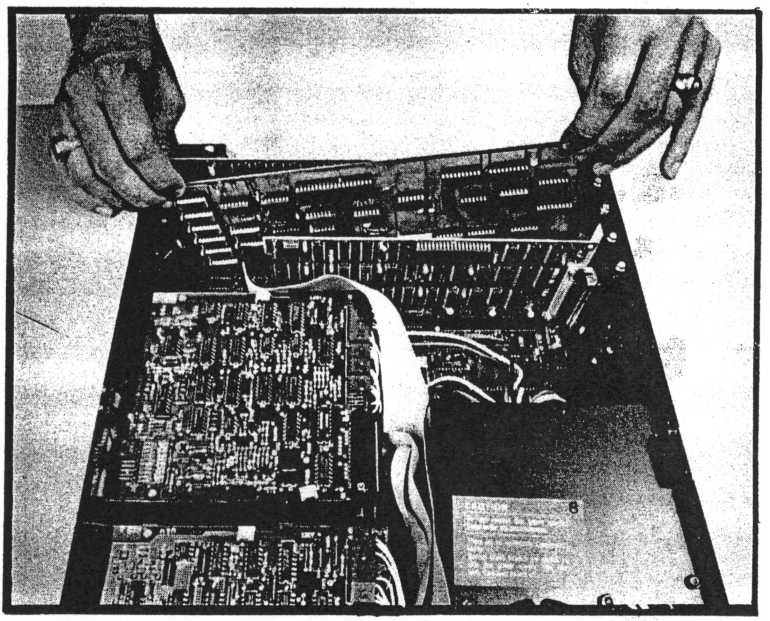


Figure 7

Closing up: Center the back-panel screwhole within the mounting bracket's elongated hole, both left-to-right and front-to-rear (Figure 8). Then replace that screw we asked you to save, and cinch it tight with your screwdriver or nutdriver (Figure 9).

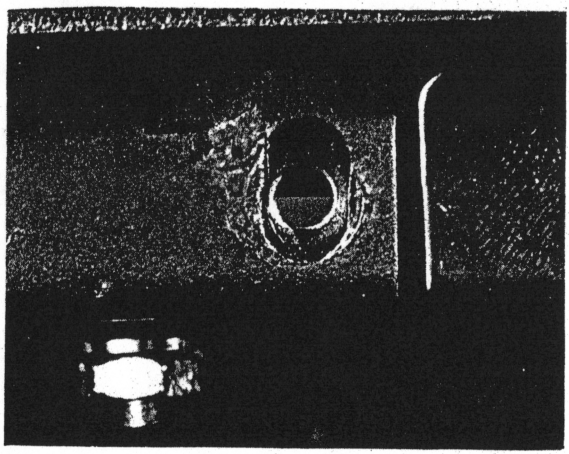


Figure 8

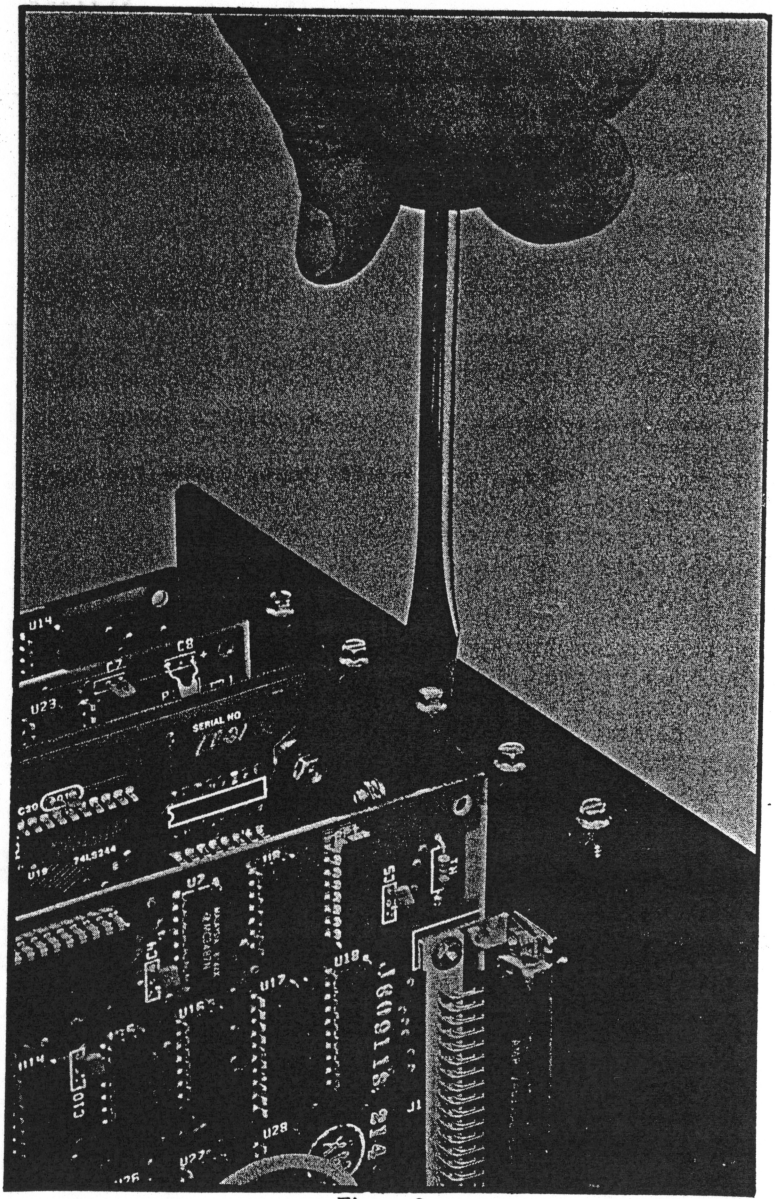

U14
C8
U23
SERIAL NO
C20
U19
74LS244
U2
U18
U17
U16
C10
U28
U27
J1

Figure 9

Replace your System Unit cover, starting by tipping it downward, then leveling it as it slides on. Take care not to bump the Baby Blue board or any expansion-option board. The cover has to go all the way on – if that black tab at the center of the back panel protrudes above it, back off and try again. Replace and tighten the cover-retaining screws.

Reconnect peripheral devices to the System Unit. If your own memory is untrustworthy, see Section 2, "Setup," of the *Guide to Operations*. Reconnect power cables last, turn on the system and make sure that it boots (displays the cursor, ultimately beeps, activates the A drive, then comes up asking you for the date and thereafter responds normally to DOS commands). Note that booting will take longer than it used to, because you've added memory to the system.

First test: Run CHKDSK (available on your DOS master disk) to confirm that your Personal Computer recognizes the memory added with Baby Blue. If your system had 64K of memory to begin with, CHKDSK should now say you have 131072 bytes of total memory. More elaborate memory checkouts can be found on the diagnostics disks supplied with the *Technical Reference* manual; they should confirm the addition of 64K.

IF a disk that brought up the system properly before fails to do so now, or IF memory tests fail to indicate the presence of 64K (65536 bytes) more memory than you started with, or IF indication of a memory error is given, please see the Appendix A on diagnostics and troubleshooting. Most likely causes of trouble are:

1. Incorrect switch settings
2. Baby Blue board not fully seated in socket
3. Seating of Baby Blue or expansion-option boards disturbed during installation process (including replacement of System-Unit cover)
4. Incorrect reconnection of system
5. Defective Baby Blue board

Absence of a video-display cursor may indicate a memory parity error; the Appendix explains a further test to perform.

The Crash Course

So far so good. Baby Blue is in the system and its onboard memory checks out. It's ready to use.

In what follows, we're going to assume that you have a working operator's knowledge of Personal Computer DOS – its rudimentary commands and facilities. If you suspect that you don't, pull out your DOS manual and leaf through it – trying things out as you go (but always using copy disks, not your original master disks; if setting up copy disks presents *any* problem, ask the folks you got your PC from). When you feel a little more seasoned, return to this place in your Blue Pages.

Now that you're ready, here's what to do:

1. Using a copy of your DOS master disk, format a blank disk with the /S (copy system) option. Put that disk into your A drive and put our Conversion Software disk into your B drive and copy all files (*.*) from B to A. (If you have just one disk drive, you'll be told when to switch disks in that drive, which is considered both A and B.) Take the original Conversion Software disk out of the B drive and store it in a safe place.

2. Take a DIRectory of the A drive. The file names displayed should include the following:

HEADER	
BIND	.COM
CONVERT	.COM
SAMPLE	.CPM
TESTZ80	.COM

3. Ease your mind: run TESTZ80 by typing just that, followed by a carriage return. It's a CP/M-80 program that already has the special header required for execution on the Personal Computer with Baby Blue. It will not run at all if the Z80B on your Baby Blue board is not functioning properly. When TESTZ80 does run, it doesn't do much more than announce itself, do a little memory testing and exercise your video, keyboard and printer (if present) a bit, then say goodbye. But that can be pleasant to see.

Here comes some theory: To make a CP/M-80 program usable with Baby Blue, you have to do two things. The first is to convert the program from CP/M file format to PC DOS file format, which is quite different. The second is to attach – bind – our special header to its front end. (We'll defer detailed discussion of what the header actually does until the next section; consider it a black box for now.) All the files you copied from the Conversion Software disk were already in PC DOS file format; TESTZ80 already had the header attached, so it had no trouble running.

Our CONVERT program performs both tasks for you. We'll get to it in a moment. HEADER is obviously the header; it must be present when BIND is used. When CONVERT is used or when a CP/M-80 program running on the PC, a linker for instance, generates a new file with the extension .COM – the header is automatically bound to the new file; the file called HEADER need not be present. BIND is a program that attaches HEADER to programs that are already in PC DOS file format – there are just a few circumstances in which you should need it – and SAMPLE.CPM is simply a sample program to use BIND on. BIND does require that HEADER be present.

The only times BIND should come into play are when: A) You've used an *8088* communications program to receive a CP/M-80 .COM file and stored that file on a PC DOS disk. You would then use the DOS rename command to give the file the extension .CPM, which BIND requires. B) We've issued a new version of HEADER and you want to attach it to already-bound programs.

4. Try out BIND. (If you're a really hard case, you may wish to satisfy yourself that a CP/M-80 program without the header will not run. Rename SAMPLE.CPM as SAMPLE.COM and attempt to run it. Fascinating. Reboot the system by cycling power and don't forget to rename SAMPLE.COM back to SAMPLE.CPM.) Type BIND SAMPLE. When the DOS prompt returns, take a directory. You should now see both SAMPLE.CPM and SAMPLE.COM in the list of files. Type SAMPLE and a carriage return. The program you've just bound doesn't do much more than TESTZ80 did, but now you can see that it runs. BIND accepts the normal DOS syntax: command source destination. Useful versions include BIND OLDNAME NEWNAME and BIND A:SAMPLE B: (taking SAMPLE.CPM from your A disk and writing SAMPLE.COM to your B disk). To bind a program located on the B disk and write the result there, while logged onto A, you would use BIND B:PROGRAM B: (to avoid writing to the default drive).

The CONVERT program is used, as we said, when you have programs and/or other files on a CP/M disk (of an acceptable format) that you want to use on the Personal Computer. In what follows, we'll assume that you have such a disk at hand. The *first* thing to do is make sure that that disk has a write-protect tab over its write-permit notch (just in case of operator error).

Run CONVERT. As soon as the disk-drive-in-use light goes off, you can remove the disk containing CONVERT, if you wish. The program asks you to nominate one of your drives, A or B, to take the CP/M disk. Let's say it's B. Type the letter B, no carriage return. CONVERT then asks which drive to use for the target PC DOS disk. Answer with A. (Note that even if you have just one drive, you still have access to

the two "logical" drives A and B, and will be prompted to swap diskettes at the appropriate times. On a one-drive system, it is *essential* that the drive you specify for the PC DOS disk be the same one that you were logged onto when you invoked CONVERT.)

Put the CP/M and PC DOS diskettes into their respective designated drives. (One-drive system: put the PC DOS disk into the drive.) Any properly formatted PC DOS diskette can be the destination. CONVERT next needs to know which of the several acceptable formats your CP/M represents, and provides you with a menu of choices. Pick one by typing its letter. After requiring confirmation of your selection, CONVERT gives you its main commands menu:

S – Select disk type (change assignments and/or format)
F – Convert files from CP/M to PC DOS
T – Convert files to CP/M from PC DOS
L – List CP/M files (DIR)
Q – Quit

S we've already taken care of in our initial dialog with CONVERT. T doesn't concern us right now (it will be discussed in the next section). Try L. The program asks you to specify a file name to look for, or a global file name (one containing ? and/or *) indicating a group of files of interest – or to just type a carriage return to display the names of all files on the CP/M disk. Hit return, and the CP/M directory appears on the screen. (One-drive system: you're prompted to switch disks first.) CP/M file names, you'll notice, use the same format as PC DOS file names.

Press any key when you're through looking at the directory, and CONVERT's main menu returns. Now let's bring some files across – press F. You're again asked to provide a specific file name, a global file name or carriage return for "all" (specifying *.* would have the same effect). As each CP/M file is converted into PC DOS format, its name is displayed. All files with the extension .COM get the Baby Blue header bound to them.

That's about it. When the conversion process is complete, the main menu returns, and you can Q for quit, and remove that CP/M diskette from its drive and put it in a safe place. The files you've brought over are ready for use – PC DOS can read the data files, and the .COM files can be executed. Go have fun with them.

One point to remember: Some CP/M programs will ask you to name the terminal you're using (because cursor-control schemes differ among the many hardware environments CP/M has had to deal with). Tell them you have a Televideo 950. If that popular terminal fails to appear on a menu of choices, answer with ADM-3A. Where neither terminal is on the menu, you can in most cases specify the individual cursor-control codes and sequences to be used; they are listed in Appendix B.

To summarize what you've learned:

- CP/M files, even data files, must be converted to PC DOS format before they can be used. The CONVERT program accomplishes this in a relatively straightforward manner.
- The Baby Blue header must be bound to a CP/M program (.COM file) before you can run that program.
- Binding is accomplished either automatically during the conversion process or by ordering BIND *filename* (omitting the optional filename extension, which is expected to be .CPM) and getting an executable *filename*.COM as the result. When BIND is used, HEADER must be present on the disk on which BIND.COM resides.
- For the sake of CP/M programs that want to do fancy things with the display screen, Baby Blue emulates a Televideo 950 terminal.

The SETCOM Utility

A late addition to the Baby Blue package is the utility called SETCOM, which allows you to use the Asynchronous Communications Adapter to drive a serial-interface printer, such as a Diablo, Qume or NEC, and allows CP/M-80 programs to use that printer as the list device. It also lets you set the adapter's baud rate (the speed at which it sends and receives data), for the sake of communications software that requires a predefined rate.

SETCOM extracts your intentions from you entirely by means of menus. It first asks whether you need to describe a printer or merely to determine a baud rate. If you're just setting a baud rate, you pick it from a menu (offering rates from 110 to 9600 baud) and that's that. If you're talking about a printer, you can select parallel interface (as in the Epson printer supplied by IBM) or serial interface with your choice of protocols – XON/XOFF, ETX/ACK or none. (Don't worry too much about what those mean; just go with what your printer supplier says.) With serial printers, you will also be asked for a baud rate, and for specification of parity (odd, even or none), the number of bits in a character (either seven or eight) and the size in bytes of your printer's internal buffer. (Here again, follow the recommendations of the printer vendor.)

Since SETCOM actually modifies PC DOS in memory, the changes it makes remain in force until the next cold boot (system reset), performed either by turning power off and then on or by pressing CTRL-ALT-DEL (from outside a CP/M program – see page 18). In addition, those changes are in force for your 8088 programs, so that they can use your serial printer too.

The cable used to connect the Asynchronous Communications Adapter to a serial-interface printer requires some special attention. (If this paragraph gets confusing, just show it to your hardware supplier.) The Adapter is wired to look like a terminal, not a computer; consequently, pins 2 and 3 of the cable must be swapped. Additionally, the Adapter does not allow you to disable hardware handshaking. If you want to use handshaking, you must swap pins 4 and 5, and swap pins 6 and 20. If you don't want to use handshaking, tie pins 4 and 5 together and tie pins 6 and 20 together, on *both sides* of

the cable; these lines *must* be broken. In any case, the PC side of the cable requires a female connector.

You may wish to set up an AUTOEXEC.BAT file that invokes SETCOM whenever the system is powered up or rebooted.

The Fine Points

The discussions that follow require some understanding of how Baby Blue works. If the theory seems thick to you, just skim it the first time. Here goes:

Baby Blue is an additional computer within your Personal Computer. On the hardware side, that means a Z80B microprocessor and 64K bytes of random-access memory (RAM). On the software side, that means the instructions to the Z80B and to your system's 8088 microprocessor that allow them to communicate with one another. The memory on the Baby Blue board is dual-ported, which means that both the Z80B and the 8088 have access to it. That's not only a nice thing — because it means that the extra memory is useful even when you're not running CP/M programs — it's also essential to the way the total combination of PC and Baby Blue works.

The special header that we treated as a black box in the last section actually consists of two parts: 8088 instructions and Z80B instructions. The 8088 instructions:

1. Get the CP/M program following the header into the right part of memory (specifically the 64K that's on Baby Blue's board).
2. Put the Z80B instructions where the Z80B can use them best (specifically the top 1/2K of Baby Blue's 64K). The greater part of the translator software is actually 8088 code in 8088 memory that's inaccessible to the Z80B.
3. Make the 8088 turn the Z80B on and keep it looking in the Z80B's direction for I/O requests (attempts to communicate with video, keyboard, disks, printer and Communications Adapter) to handle.

When PC DOS loads a .COM file for execution, it reads it into memory starting from the lowest available address (remember that every byte of memory in the system has a unique numerical address, so that "low memory" and "high memory" follow from those number-addresses). That load address is (almost always — exceptions are covered below) in the lowest 64K of system memory, while Baby Blue's RAM is always set up as at least the second 64K of system memory, never lower. So we've got the Baby Blue header, including Z80B instructions, and all or most of the CP/M program it was bound to, down in that bottom segment, while the Z80B can read and write only memory that's in the higher segment where Baby Blue's RAM is. Consequently the first task of the 8088-instructions part of the header is to move a few things around.

The Z80B instructions, part of the translator (we'll get to that), have to be moved to the top of Baby Blue's 64K segment. The CP/M program has to be moved into Baby Blue's memory, starting at address 256 of that segment (because that's where CP/M programs expect to find themselves, and if you move them they can get *very*

confused). And what's known as Page Zero — addresses 0 to 255, a kind of CP/M scratchpad — has to be set up properly within the Baby Blue segment, too. The lower part of that segment may have been loaded by PC DOS with the upper part of the CP/M program, if it's a really large one, but the 8088 instructions in the header are smart enough to keep from losing any part of the program by writing on top of it.

Then the 8088 activates the Z80B, telling it to begin executing instructions starting from address 256 in its own segment. After that, the 8088 is in a loop within its own instructions in the header, one in which it's continually checking the Z80B for I/O to cope with, which it can do by using PC DOS.

The Z80B is off and running now, running at the same clock speed the 8088 uses and executing instructions within the CP/M program. Every so often, those instructions will include a call to a CP/M function to handle input or output (to or from video, keyboard, disks, etc.). Now, since there's no CP/M operating system anywhere in the PC, CP/M I/O function calls are referred to the translator we put (in part) at the top of Baby Blue's 64K segment. The Z80B part of the translator converts the CP/M-style I/O request into something the 8088 can cope with, and the 8088 part of the translator then changes it into a PC DOS-style I/O request and passes it along for action. That simple.

When the CP/M program is finished, the Z80B is turned off again, and you get your DOS prompt back. There are a few further niceties, and we'll spell out below the areas in which total CP/M compatibility is slightly compromised, but that's about the size of it.

In the remainder of this section we'll treat a variety of topics, each as briefly as possible. Not every topic will be of earthshaking interest to every Baby Blue user, but we suggest that you at least scan all the topic headings.

The Less-than-64K System

As noted in a previous section, we *do not* recommend that Baby Blue be used only with a system that has less than 64K of its own.

Rebooting

Attempting a keyboard reboot — CTRL-ALT-DEL — while in the middle of a CP/M program will not reboot the system but will turn off the Z80B and return you to DOS and its prompt. If nothing seems to happen, type a carriage return. If you next type CTRL-ALT-DEL, DOS will reboot normally.

CONVERT Errors and Error Messages

The CONVERT program is a pretty forgiving one — you can't do too much to keep it from working the way it was meant to. If you give it the same drive-letter designation for both the CP/M disk and the PC

DOS disk, it will tell you that that isn't very sensible (since a single-drive system lets you use both A and B, even if they *are* both in the same place). If you type a character that corresponds to no menu item or command, it will be ignored or you'll get a "no such command" message. And you'll get an error message if you use a global filename in telling CONVERT what to transfer *to* the CP/M disk (see below).

About the only other ways you can go wrong – not necessarily your own fault – are with either a full disk or a full directory at your destination, or a CP/M disk that doesn't match the format you selected. Each condition produces an appropriate error message. If you run out of disk space, CONVERT will erase the incompletely-transferred file from the destination disk. If you run out of directory space, you won't have an unfinished file to erase. If you tell CONVERT to expect one CP/M disk format and then feed it another, you'll get a message and no conversion will take place. BUT – some CP/M disk formats may be so similar (so subtly dissimilar) that CONVERT cannot detect the discrepancy, and conversions occur but yield scrambled files (the scrambling will be readily apparent with text files you're already familiar with – chunks of text will be out of order); so it's in your own interest to take some responsibility for giving CONVERT only the CP/M disk format you said you would.

Whenever a CONVERT error message appears, you clear it by pressing any key. Whatever was the current command is aborted, and you take it from there.

Note that the Baby Blue header adds about 4K to each .COM file converted. Allow for this in planning your disk-space needs.

CONVERTing *to* CP/M

The program has the ability to make file transfers *from* PC DOS format *to* your selected CP/M format. It can be handy to be able to pass files on disk to a user of a system other than a PC. Two special comments are in order:

Global filenames are not accepted for transfer in this direction. An error message appears if you try using a ? or * in the filename specification.

CONVERT will strip the 4K Baby Blue header off every .COM file translated to CP/M format. You know, of course, that an 8088 (native PC) .COM file cannot be converted and expected to run on a CP/M system; the CONVERT program, in fact, won't even recognize it.

But note that since CP/M compilers, interpreters, assemblers and linkers will run with Baby Blue and since CONVERT can put the results back into CP/M format, you *can* use the Personal Computer as a development system for software that is ultimately to be used on CP/M-only systems.

BIND Errors

Just a few ways to crash-land here: non-existent source file, HEADER not present, out of disk space or directory space when trying to write the output .COM file. In any of these cases, you get an appropriate message from either BIND or PC DOS, and your DOS prompt returns; any incomplete files are removed for you.

BIND looks first for the source filename with extension .CPM. When that can't be found, it looks for a previously bound source COM and replaces its header with the current version of HEADER. It cannot find 8088 .COM files, much less alter them.

Note that BIND does *not* accept global filenames.

ROM-BIOS Access

Those who do assembly-language programming (and some devious high-level-language programmers) may be interested to note that Baby Blue gives you access to the PC's ROM (read-only memory) BIOS (Basic Input/Output Subsystem) via a new CP/M-style BDOS (Basic Disk Operating System) function call that emulates the 8088 INT (software interrupt) instruction. The function number is FD hexadecimal (253 decimal), and on entry the HL register-pair points to a table in memory that represents an 8088 register layout:

Interrupt number – 1 byte
Register AX – 2 bytes
Register BX – 2 bytes
Register CX – 2 bytes
Register DX – 2 bytes

On return from the function call, contents of these table locations will have been appropriately updated.

Direct I/O

Although direct input and output involving Z80B ports are *not* supported, direct I/O with 8088 ports *is*, by means of extended BDOS calls. For output, call function number FE hexadecimal (254 decimal) with the 8088 port number (3F8H for the Asynchronous Communications Adapter, for example) in the HL register-pair and the byte to be output in the E register. For input, call function number FF hexadecimal (255 decimal) with the 8088 port number in the HL register-pair; the input byte is returned in the A register. A call to function number F1 hexadecimal (241 decimal) returns the Asynchronous Communications Adapter's status in the A register.

Programs Calling Programs

PC DOS does some pretty neat things that CP/M never dreamed of. One of these is a relocation scheme by which a program can call for another program to be loaded into memory above the first one and executed; when the subsidiary program is done, the original one takes up where it left off. Not only that, but the second program

called by the first can itself call a third, and so on – a veritable stairway to the stars.

If you're not an assembly-language programmer you may never even notice that such a hierarchy of calling-and-called programs is being built up in memory. But what happens when a Baby Blue-converted CP/M program is invoked as one of the latter steps in the stairway? Disaster, perhaps, if the invocation is not rejected with an error message.

CP/M programs under Baby Blue are somewhat sensitive as to their initial load-address in system memory. That's because the CP/M program *per se* and the Z80B part of the translator have to go into specific spots in Baby Blue's own 64K segment, while the part of the header that is 8088 instructions has to remain available *outside* Baby Blue's segment. If you have too large a calling program (or hierarchy or calling-and-called programs), you can get the 8088 instructions in the header overwritten when the attempt is made to set up Page Zero, the CP/M program and the translator in the Baby Blue segment – this can happen with a caller or callers totaling 50K or more in a system that had 64K before Baby Blue arrived. If you have more memory (and you've installed Baby Blue at the top of available memory, as we advise), you can build a correspondingly taller staircase before reaching the Babel-point. (If Baby Blue is not at the top of memory, and you have a *very* large structure of callers-and-called, you could get into a situation in which the CP/M program runs fine, but the bottom steps have been blown away – which you won't learn until you attempt to descend them in order.)

While Baby Blue is relatively new, the PC user is unlikely to encounter such situations. *If* you're a programmer who's out to put together the ultimate system of programs that includes both PC DOS-native software and CP/M programs in a grand, world-beating design (and more power to you!), don't *you* be the one who sets up a caller(s) staircase greater than 50K.

Corollary: Don't ever assume that a CP/M-native program can be used as an overlay to a PC DOS-native program (or vice versa). CP/M programs, however, can use their own overlays freely.

One-Drive Operations

...just take care and patience. We've already discussed the use of CONVERT with a single drive. CP/M programs that expect two disk drives, as handled by our translator and PC DOS, will prompt you to switch diskettes as required.

CP/M-80 Compatibility

We have never promised anyone perfect compatibility with CP/M-80. After all, CP/M-80 (a/k/a CP/M) is far from perfectly compatible with itself; between versions, between hardware implementations and between specific disk formats you find a multitude of gulfs.

What's of interest to us and to the majority of users is compatibility with plain-vanilla, no-tricks, maximally-portable, hardware-independent CP/M programs. That's the kind of compatibility that's long been of interest to the people who've been doing serious, useful programming within CP/M environments.

Below we'll detail the points of incompatibility, or incomplete compatibility, with the CP/M rainbow of possibilities. Keep in mind that responsible programmers, those not infatuated with particular hardware setups, will have avoided most of the traps for you already. Programs that fail to run with Baby Blue would probably fail when carried from one CP/M-native system to another.

- BDOS functions: Supported — granted that those that are meaningless in a PC DOS environment return constant values or are simply ignored. Appendix D spells it all out.
- BIOS jump table: It's all there, pointed to by the customary jump at address 0 (within Baby Blue's segment). It's all functional, except for the disk primitives (hint: don't use them).
- IOBYTE: Not implemented, because PC DOS (the version current as of this writing) offers no corresponding facility for I/O redirection.
- Users: Not supported, because unavailable under PC DOS. Everyone is always user 0, despite any attempt to change user number.
- File attributes: Unsupported and ignored. The only PC DOS file attribute analogous to a CP/M attribute is invisibility, one we do not consider particularly useful for anything but files that are parts of the operating system.
- Utilities: Certain CP/M disk utilities, such as STAT, will run but give wrong answers. Use DOS functions and utilities instead.
- Block or group map: Bytes 16 through 31 (numbered from 0) of a CP/M file-control block are used for a representation of the actual allocation of disk-storage groups or blocks to a file. According to documentation, these bytes are reserved for system use and are to be left untouched by user programs. Some existing programs, primarily disk utilities, may violate this prohibition. Don't expect them to work with Baby Blue.
- Software write-protection of disks: Not supported.
- File extents: Fully emulated, even though PC DOS does not use them.
- SUBMIT: In general — as for a compile-and-link batch operation — rename your .SUB file as a .BAT file and let PC DOS handle it (initiate execution by typing the filename without extension). There is, however, no XSUB equivalent; also, each command line within the batch file must be either a proper program invocation or a valid PC DOS command.

■ $$$.SUB: Supported. This primitive method of chaining programs, used when high-level languages offered no better, will still work with Baby Blue, which is to say that the next (reading from bottom up) line of the $$$.SUB file will be executed, assuming it invokes an available program.

■ Transient Program Area (TPA) size: 63.5K — the equivalent of running a 70.5K native-CP/M system. Translation: no sweat, you have all the user memory you need. (Note that the 4K Baby Blue header is *not* a 4K TPA penalty, because 3.5K of it resides outside Baby Blue's RAM.)

■ Direct I/O: Meaning a specific value to or from a specific 8080/8085/Z80 port number — can't be done. This kind of thing is exceedingly hardware-specific, as are writing to screen memory and handling specific peripherals. Note also that software supplied by hardware manufacturers has usually been customized to run on *their* hardware only.

■ Page Zero: Is handled exactly as under CP/M — default file control block and command-line copy are there, as usual. The jumps you expect to find at 0 and at 5 are there. What you should not expect to use are interrupt vectors within Page Zero — because there's nothing that can interrupt-drive Baby Blue's Z80B.

■ Asynchronous Communications Adapter (PC DOS aux in and aux out): Permanently defined as CP/M's reader and punch, also accessible directly through extended BDOS calls. If absent, do not call.

■ Strange characters: Although their filename formats are essentially the same, CP/M and PC DOS do differ slightly in the choice of characters they permit in filenames and extensions. Specifically, CP/M permits / " and + while PC DOS does not. Contrariwise, PC DOS permits < and > while CP/M does not. You may encounter CP/M programs that use or create files with the incompatible characters in their names. Baby Blue's software automatically changes them (during the CONVERT process or when they are created by a user program):

/ becomes <
+ becomes >
" becomes ` (accent grave or backward apostrophe)

The only possibility for trouble is with the ` character, which is permitted in CP/M filenames, so that two filenames that differ only by virtue of one having the " where the other has the ` cannot be told apart; such a situation should be vanishingly rare.

■ Case in filenames: PC DOS treats all letters in filenames as uppercase, at all times. CP/M, although it makes access to them a little difficult, does permit and distinguish filenames containing lowercase letters. To work correctly with Baby Blue, programs that use filenames that are distinguishable only by virtue of the *same* letter(s) being either uppercase or lowercase will have to be modified, either by their authors or by means of careful patching (usually requiring either source code or the program author's help).

■ Random-access-file gaps: CP/M and some high-level languages running under it permit a questionable programming practice — writing to different records of a random-access file without necessarily allocating disk space for the intervening records. PC DOS performs this allocation automatically. At best, this will mean that an unfilled random-access data file will require more space under PC DOS than it did under CP/M. At worst, that file will be truncated in conversion, because CONVERT will stop the process after finding a file extent that is not fully allocated (written to). Compare file sizes before and after conversion to make sure this hasn't happened to your data files.

The CP/M Connection

Suppose you were a native speaker of Finnish, knowing no other language, and living in the United States. Wouldn't you agree that the number of people you could talk with would be somewhat limited?

Suppose two people have just had a baby. Will they teach the child a brand-new language they've just invented for his or her exclusive use, and allow her or him to hear no other?

Suppose each individual in the country minted a unique currency, with no standardization of value, denominations or even what to call a unit of money? How easy would it be to buy a week's groceries or to go on a trip?

Suppose human beings varied greatly in the locations, numbers and functions of their organs. Would you want to be a physician?

Paranoid fantasies of chaos, you say. Well, the analogies may be imperfect, but they convey a feeling for the situation that existed when microcomputers first appeared, that exists to a degree today and that could have been much worse if CP/M had never been devised.

Let's take it from the basics. Every computer (mainframe, mini, or micro) is just a box that does calculations and makes decisions, and needs to communicate with what it considers the outside world. Every computer, if it's to do anything even a little bit useful, has to have a means of storing information in some relatively permanent form, something that won't forget everything the next time the power is shut off. That's two fundamental problems: communication and mass storage.

Each of those problems has a hardware aspect. The hardware of communication includes the many kinds of terminals, keyboards, video units, printers, plotters and digitizers. The hardware of mass storage includes — with the devices that read and write them — punched paper tape, magnetic tape, hard disks, and floppy diskettes (these last two also magnetic). We'll take the hardware as given, and we'll limit our further discussion of mass storage to floppy diskettes, since they're what we have at hand.

Neither the problem of communication nor the problem of mass storage is solved until we also provide software — instructions that tell the computer, in these cases, *how* to use the communications devices we've attached to it, *how* to employ that floppy-diskette gear. So now we have two software problems.

There are a zillion possible ways of solving each of those problems. If you like, the rest of us will wait right here while you go off in a corner and solve them for yourself; we have no doubt that given time, sufficient interest and lots of patience, you can come up with quite respectable, perhaps even brilliant solutions.

Back so soon? We're impressed. But we'll bet you that (unless you've been around little computers for a while and probably don't need to read this section) your solutions have something in common with the stranded Finn, with young John or Jane Doe who speaks and understands only John-or-Jane-Doe-ish, with the homemade money and with someone whose liver looks and acts like skin and vice versa. The residual problem is that of standardization.

Why do we want to impose standards (with their ever-present implication of compromises)? Why shouldn't each computer do its own thing? Some degree of standardization, built into the software solutions to the problems of communication and mass storage, is important for these simple reasons:

1. It is a good thing for any given computer to be able to use the largest possible number of programs and pieces of data.
2. It is a good thing for any given program or piece of data to be accessible to the largest possible number of computers.
3. The only way to achieve those two good things, given that there are a lot of different computers out there, is to find a way to A) make all the programs and pieces of data look the same to all the computers and B) make all the computers look the same to the programs and pieces of data.

And *that,* dear friends, is where a standard or quasi-standard operating system comes in. An operating system, since we haven't defined it before, is a hunk of software that handles communication and mass storage. The Personal Computer's DOS is one example; CP/M-80 is another, though it's really a family of semi-compatible operating systems.

We will refer to CP/M-80 henceforward as just plain CP/M, since that's the way its own authors referred to it for quite a while. CP/M is really a whole sheaf of operating systems because it runs on a lot of different computers, and therefore includes a chunk that's privy to the detailed inner workings of each of those computers (a hardware-specific part), *and* a chunk that's always the same regardless of the computer it lives in — the part that makes the computers all look the same to the programs and pieces of data. It's that second part that includes a *general* way of communicating with the outside world and a *general* way of storing data and programs on floppy diskettes (and getting them back when needed!), and those *general* methods are all that another piece of software needs to know about as it blithely skips from specific machine to specific machine. It's almost as though everyone in the world instantly became fluent in Esperanto.

CP/M is a brilliant solution, make no mistake. You'll hear people criticize it, from their advanced perspectives of 1982 and beyond. But CP/M has been kicking around since about 1975 (ancient times, by microcomputer standards), and it does the job. PC DOS, like every other important microcomputer operating system you're going to see, owes something to CP/M – for borrowing from its successes, and for learning from its shortcomings. CP/M is a watershed.

And that fact has had a snowballing effect (mixture of precipitation metaphors noted). No prudent manufacturer of a new computer with eligible hardware (we'll get to that in a minute) today fails to take CP/M into account – that company either provides CP/M or provides an operating system that looks just like it to the programs and pieces of data. Similarly, the majority of programmers who want to sell serious programs for serious dollars write for the CP/M environment. The manufacturers reinforce the programmers who reinforce the manufacturers who... beautiful synergism!

Alas, there are several flies in this dish of soup. Most relate to hardware. CP/M requires one of a specific trio of microprocessors – 8080, 8085, or Z80 – and was written for the lowest common denominator in that group, the essentially obsolete 8080.

That leaves a lot of interesting machines out in the cold; an unmodified Apple, Atari, or Commodore PET/CMB can't run CP/M, because they use the 6502 as a central processing unit (CPU). Some others, even with an acceptable microprocessor, are beyond the pale because of other hardware quirks; a stock TRS-80 Model I or Model III can't run regular CP/M because of read-only memory in inconvenient locations (and the kind of modifications to CP/M that half-solves the problem leaves most CP/M programs unavailable to those systems). Users of those popular but un-CP/M microcomputers have been forced to evolve their own software literature, and have done so.

The larger hardware problem lies in the development of advanced microprocessors incompatible with the 8080-8085-Z80 kinship group. It had to happen. Suppose you were a newly graduated automotive engineer, hired by a major car manufacturer and then told that everything you did would have to be usable as a simple accessory or add-on upgrade to a 1919 Model T. After expending considerable ingenuity, you might devise enhancements that would allow a Model T to cruise at 55 mph *and* meet Federal safety and emissions standards circa 1982. More likely, you would find another job and look on with glee as your first employer went belly up trying to make a 1982 product that retained 1919 compatibility.

Real innovation virtually guarantees incompatibility with the past; the alternative is stagnation. The new generation of 16-bit microprocessors proves the point. IBM chose the 8088 processor for the PC because it offered significantly better performance than the

old 8-bit processors. With that choice, IBM also took a bold leap into the unknown, because the programs and pieces of data wouldn't be out there, ready and waiting, as they had been for the legions of manufacturers who introduced CP/M-compatible, me-too machines in 1981 and the early part of 1982. Besides a new processor, IBM chose a new operating system, PC DOS (also sold as MS-DOS and SB-86 by others, with the hardware-specific part either set up for different machines or left up to the buyer to figure out), that didn't even *pretend* to be compatible in any way with CP/M.

(What about CP/M-86, you ask? All right, let's discuss it within these discreet parentheses. It's the CP/M idea applied to the 8086 and 8088 microprocessors. What it isn't is compatible with all that existing CP/M-80 stuff. Yes, conversions can be made, but you have to start with the original programmer's working files — source files. The process is labor intensive, and the end results seem to take more memory and more execution time than the CP/M-80 versions did. It's also completely compatible with PC DOS — what one operating systems knows, it won't be able to share with the other.)

So IBM got itself a new processor and a new operating system, and that was all very high-minded, but where did it leave you? Apart from the use you could get from VisiCalc, it left you looking at those BASIC demo programs, most of which were unrunnable if you'd bought the machine for business purposes and didn't have the color-graphics board. It left you listening to "Humoresque" or the "Mexican Hat Dance" and, perhaps, feeling a little bit foolish.

Well, at that point we had to step in. There's so much good stuff, *useful* stuff out there written for CP/M machines that it would have been a crime not to turn it over to you. So we found a way to make CP/M programs and pieces of data feel at home with the Personal Computer and its DOS. Such hospitality required both hardware and software, but IBM was wise enough to provide expansion slots for the hardware, and we've made the software practically take care of itself.

We don't expect the original Baby Blue to hang around forever, although it will always be 64K of usable RAM. Someday, we should see a majority of the solid CP/M programs (and pieces of data) rewritten for the 8088 and PC DOS. Eventually, we should also see plenty of really good software written for the Personal Computer as a native environment. But in the meanwhile — why wait?

After all, we're providing transparency, with the convenience of dealing with just one operating system; it's as though you were using a CP/M-native machine, but with greater speed and more usable memory. And Baby Blue will never really be obsolete, so long as the programs it runs are of use.

Appendix A — Diagnostics and Troubleshooting

If you get no video-display cursor on attempting to bring up your system with Baby Blue in place:

1. Make sure you have reconnected the system correctly. Make sure Baby Blue and all other expansion-slot boards are seated. Try booting again if any discrepancy is found.

2. With all power off and disconnected, turn position 8 on Baby Blue's DIP switch off. Then try to boot again. If the system comes up *now*, with position 8 off, but indicates a memory fault, your Baby Blue board may be defective and may require return for repair or replacement — but recheck *all* switch settings first. You should be able to confirm the defect in detail with your IBM diagnostics software.

If your system comes up, but either 1) indicates faulty memory or 2) does not respond to DOS commands, then switch settings — on Baby Blue and/or on the system board — are probably wrong. Go through the entire installation procedure again with special attention to switch settings. If the system still fails to come up, contact your vendor or Xedex Corporation.

Appendix B – Terminal Emulation

The IBM Personal Computer with Baby Blue CPU Plus emulates a large subset of the functions of the Televideo 950 terminal.

Control codes:

	ASCII decimal	ASCII hexadecimal	
CTRL G	7	07H	Bell
CTRL H	8	08H	Backspace, cursor left
CTRL I	9	09H	Tab
CTRL J	10	0AH	Line feed
CTRL K	11	0BH	Cursor up
CTRL M	13	0DH	Carriage down
CTRL V	22	16H	Cursor down
CTRL Z	26	1AH	Clear screen
CTRL ^	30	1EH	Home cursor
CTRL _	31	1FH	New line (carriage return-line feed)

Escape Sequences:

	ASCII decimal	ASCII hexadecimal	
ESC $	27, 36	1BH, 24H	Graphics mode on (IBM, not Televideo, graphics set)
ESC %	27, 37	1BH, 25H	Graphics mode off
ESC (	27, 40	1BH, 28H	Set high intensity
ESC)	27, 41	1BH, 29H	Set low intensity
ESC *	27, 42	1BH, 2AH	Clear screen
ESC +	27, 43	1BH, 2BH	Clear screen
ESC ,	27, 44	1BH, 2CH	Clear screen
ESC . *n*	27, 46	1BH, 2EH	Set cursor attributes (permitted: n = 0, no cursor; n = 1 or 2, blinking block cursor; n = 3 or 4, blinking underline cursor)
ESC = *r c*	27, 61	1BH, 3DH	Position cursor, where *r* and *c* are row and column, with offset of 32 (20H) added to each
ESC ?	27, 63	1BH, 3FH	Transmit current cursor position (row, column) and carriage return to host
ESC E	27, 69	1BH, 45H	Insert line
ESC F	27, 70	1BH, 46H	Load second user buffer
ESC G *a*	27, 71	1BH, 47H	Set attribute (permitted for *a*; 0, normal; 1, blank; 2, blink; 3 blank; 4 reverse; 5, reverse blank; 6, reverse blink; 7, reverse blank; 8, underline; 9, underline blank; :, underline blink; ;, underline blank; <, underline; =, underline reverse blank; >, reverse blink; ?, underline reverse blank. The preceding sequences all step the cursor forward; to set attributes *without* cursor motion, use the characters @, A, B,... in place of the sequence 0, 1, 2,...

ESC N	27, 78	1BH, 4EH	Set page edit mode
ESC O	27, 79	1BH, 4FH	Set line edit mode
ESC Q	27, 81	1BH, 51H	Insert character
ESC R	27, 82	1BH, 52H	Delete line
ESC T	27, 84	1BH, 54H	Clear to end of line
ESC V *a*	27, 86	1BH, 56H	Set color, where *a* may be:

- 0 — foreground black
- 1 — foreground blue
- 2 — foreground green
- 3 — foreground cyan
- 4 — foreground red
- 5 — foreground magenta
- 6 — foreground brown
- 7 — foreground white
- 8 — foreground gray
- 9 — foreground light blue
- : — foreground light green
- ; — foreground light cyan
- < — foreground light red
- = — foreground light magenta
- > — foreground yellow
- ? — foreground high-intensity white
- @ — background black
- A — background blue
- B — background green
- C — background cyan
- D — background red
- E — background magenta
- F — background brown
- G — background white
- H — border black
- I — border blue
- J — border green
- K — border cyan
- L — border red
- M — border magenta
- N — border brown
- O — border white

(Text mode only. Color-setting sequences do not move the cursor.)

ESC W	27, 87	1BH, 57H	Delete character
ESC Y	27, 89	1BH, 59H	Clear to end of screen
ESC f text CR	27, 102 CR = ASCII 13 or 0DH)	1BH, 66H	Load user buffer (80 characters maximum)
ESC g	27, 103	1BH, 67H	Display user buffer
ESC h	27, 104	1BH, 68H	Display second user buffer
ESC j	27, 106	1BH, 6AH	Reverse line feed
ESC t	27, 116	1BH, 74H	Clear to end of line
ESC y	27, 121	1BH, 79H	Clear to end of screen
ESC |	27, 124	1BH, 7CH	Load function keys (must be loaded in order, with 256 characters maximum total)

Function-key programming:
ESC I (27, 124 or 1BH, 7CH) is followed by a character indicating the key to be programmed and whether it is shifted:

Key	Character	ASCII decimal	ASCII hexadecimal
F1	1	49	31
F2	2	50	32
F3	3	51	33
F4	4	52	34
F5	5	53	35
F6	6	54	36
F7	7	55	37
F8	8	56	38
F9	9	57	39
F10	:	58	3A
F1 shifted	<	60	3C
F2 shifted	=	61	3D
F3 shifted	>	62	3E
F4 shifted	?	63	3F
F5 shifted	@	64	40
F6 shifted	A	65	41
F7 shifted	B	66	42
F8 shifted	C	67	43
F9 shifted	D	68	44
F10 shifted	E	69	45

An ASCII 1 (49 or 31H) comes next, followed by the message to be loaded; any non-printing character must be preceded by a CTRL P (16 or 10H). The message is terminated by CTRL Y (25 or 19H). Note that because function keys use a single buffer, they must be programmed in order. A total of 256 characters is available.

Codes and sequences generated by function, cursor and editing keys:

	Unshifted	Shifted	Control	Alt
F1	01H 40H 0DH	01H 60H 0DH	02H 5EH	02H 68H
F2	01H 41H 0DH	01H 61H 0DH	02H 5FH	02H 69H
F3	01H 42H 0DH	01H 62H 0DH	02H 60H	02H 6AH
F4	01H 43H 0DH	01H 63H 0DH	02H 61H	02H 6BH
F5	01H 44H 0DH	01H 64H 0DH	02H 62H	02H 6CH
F6	01H 45H 0DH	01H 65H 0DH	02H 63H	02H 6DH
F7	01H 46H 0DH	01H 66H 0DH	02H 64H	02H 6EH
F8	01H 47H 0DH	01H 67H 0DH	02H 65H	02H 6FH
F9	01H 48H 0DH	01H 68H 0DH	02H 66H	02H 70H
F10	01H 49H 0DH	01H 69H 0DH	02H 67H	02H 71H
Left Cursor	08H		02H 73H	
Right Cursor	0CH		02H 74H	
Up Cursor	0BH			
Down Cursor	16H			
Home	1EH		02H	
End	02A 4FH		02H 75H	
Page Up	02H 49H		02H 77H	
Page Down	02H 51H		02H 76H	
Insert	1BH 51H			
Delete	1BH 57H			
←	7FH			

Appendix C — Memory Map

Addresses indicated are within Baby Blue's 64K memory segment.

Hexadecimal		Decimal
FFFF		65535
	Z80B Portion of Translator	
FE00*		65024*
FDFF		65023
	Transient Program Area — Space for User Programs	
0100		256
00FF		255
	Page Zero	
0005	Jump Vector to BDOS translation	5
0000	Jump Vector to jump table in BIOS translation	0

*Subject to change

Appendix D – BDOS Function Calls

Number	*Function*	*Notes*
0	System reset	return to PC DOS
1	Console input	
2	Console output	
3	Reader input	input from Asynchronous Communications Adapter
4	Punch output	output to Asynchronous Communications Adapter
5	List output	
6	Direct console I/O	
7	Get iobyte	ignored
8	Set iobyte	ignored
9	Print string	
10	Read console buffer	uses PC DOS line editing
11	Get console status	
12	Return version number	returns version 2.2
13	Reset disk system	ignored
14	Select disk	
15	Open file	ignores header on bound COM files; unbound .COM files (CP/M or 8088) will not be found
16	Close file	
17	Search for file	
18	Search for next	
19	Delete file	
20	Read sequential	
21	Write sequential	
22	Make file	automatically binds header when file to be created is of the type .COM; a can't-open message may mean directory or disk full.
23	Rename file	automatically binds header whenever new name is of type .COM; removes it when .COM file is changed to other type. If disk space is insufficient for bound file, type CPM is produced.
24	Return log-in vector	returns A and B logged in
25	Return current disk	

Number	*Function*	*Notes*
26	Set DMA address	
27	Return allocation vector	ignored
28	Write protect disk	ignored
29	Return R/O vector	returns no drive read-only
30	Set file attributes	ignored
31	Return address of disk parameters	returns pointer to a dummy disk parameter table
32	Get/set user code	does not set user code; always returns 0
33	Read random	
34	Write random	
35	Compute file size	
36	Set random record	
37	Reset drive	ignored
38	not used	
39	not used	
40	Write random with zero fill	

Extended functions (see "Fine Points" section):

240	Initialize Asynchronous Communications Adapter (See Appendix E)
241	Return Asynchronous Communications Adapter status
251	Peek (load DE with memory address, HL with memory segment; memory value is returned in A)
252	Poke (load DE with memory address, HL with memory segment, B with value to be placed in memory)
253	Direct ROM-BIOS call
254	Direct output
255	Direct input

Appendix E — Sample Communications Code

```
bdos  equ     5

; Code to initialize port

        mvi c,0F0h
        mvi e,0E3h          ; 9600 baud — 1 stop bit — no parity
        call bdos

; code to get comml status

        mvi c,0F1H          ; Status is returned in A register
        call bdos           ; bit 0 high for char received
                            ; bit 5 high for output ready

; code to transmit character in Register A to comml

        push psw            ; save character
loop:   mvi c,0F1H          ; get status
        call bdos
        ani 20h             ; check for transmit ready
        jz loop             ; not ready try again

        pop psw             ; restore character
        mov e,a             ; move character to E
        mvi c,0FEh          ; direct output
        lxi h,3F8h          ; to comml data port
        call bdos

; code to get character from comml

loopl:  mvi c,0F1h          ; get status
        call bdos
        ani 01h             ; check for character received
        jz loopl            ; not there — try again

        mvi c,0FFh          ; direct input
        lxi h,3F8h          ; from comml data port
        call bdos           ; character returned in A register
```

Index

Baby Blue CPU Plus Limited Warranty

Xedex Corporation congratulates you on the purchase of your new Baby Blue CPU Plus. Your Baby Blue's circuit board and disk, including the program (software) on the disk, are covered by a limited warranty for 90 days from the time the Baby Blue is purchased from Xedex or an authorized Xedex dealer.

If your Baby Blue's circuit board or disk is physically defective, or the disk program is unreadable, within the 90-day warranty period, Xedex will repair or replace it. If the program on the disk fails to perform as described in the User's Manual within the 90-day warranty period – and the defect can be demonstrated by repetition – Xedex will repair or replace it within 60 days of notification. Note, however, that certain CP/M programs will not function with Baby Blue for reasons beyond Xedex's control, and Xedex will not be responsible for a specific program not working with Baby Blue.

To obtain warranty service. If you think your Baby Blue may be defective, call (914) 368-0353. Xedex will try to pinpoint what the problem is. At its option, Xedex will send you a replacement or ask you to return the circuit board or disk or both for repair. We will reimburse you for shipping charges on items we ask you to return.

Limitations. This warranty does not include service for damage resulting from accidents, misuse or failing to use Baby Blue according to the directions in the User's Manual.

Any implied warranty of fitness for a particular purpose or warranty of merchantability is limited to the 90-day warranty period. Also, Xedex is not responsible for lost profits, lost savings or any other incidental or consequential damages.

Some states do not allow limitations on how long an implied warranty lasts or the exclusion or limitation of incidental or consequential damages, so the limitations or exclusions stated above may not apply to you.

This warranty gives you specific legal rights and you may have other rights which vary from state to state.

Baby Blue CPU Plus comment form

Your reactions to our products – positive, negative or pointedly neutral – are important to us. It is understood that we may use your comments or information in any way we consider appropriate, without incurring obligation. Your name and affiliation will not, however, be used without your consent. If you wish a reply to your comment, please include your name and mailing address; such inclusion will not be considered permission to use your name and/or affiliation unless you so state. If convenient, and to help us respond to your comment, please include a brief summary of your prior experience with microcomputers. Thank you.

BABY BLUE CONVERSION SOFTWARE
VERSION 2.Ø
ADDENDUM

Congratulations! You have received the enhanced version (version 2.Ø) of the BABY BLUE CONVERSION SOFTWARE. The new version offers you two major new advantages:

1. Compatibility with the new IBM PC DOS Version 1.1
2. A more user-friendly program which supports five CP/M formats:

    NEC PC-8ØØ1
    IMS 5ØØØ
    DEC VT-18
    Heath/Zenith
       (48 TPI)
    CP/M-86

Compatibility with the new IBM PC DOS Version 1.1 is the result of an enhanced version of the "HEADER" portion of the conversion software. Supporting the five CP/M-8Ø formats in a more user- friendly fashion is the result of a completely new version of the "CONVERT" portion of the software. "BIND" remains unchanged.

If this is your first experience with Baby Blue, we recommend that you first read the Baby Blue instruction manual and then read this addendum.

PROGRAMS PREVIOUSLY CONVERTED (using version 1.Ø)

If you have programs which have already been converted to the DOS format using version 1.Ø of the Baby Blue Conversion Software, it is necessary to re-BIND these programs. As discussed in the manual, you must run BIND whenever a new version of HEADER is released (see page 14). Running BIND will bind the new version of HEADER onto all of your previously reformatted files.

USING YOUR NEW CONVERSION SOFTWARE (version 2.Ø)

As you know from the Baby Blue manual, in order to run CP/M software on your IBM PC + Baby Blue, you must first convert your CP/M-8Ø files to PC DOS format. To invoke the conversion program, type:

d:convert <CR>

where "d" represents the disk drive location of your CONVERT program. The screen will clear and the following message will appear:

CP/M IBM File Trans. Utility V 2.Ø
Copyright (c) 1982 Microlog Inc.

This message is followed by the prompt:

IBM Disk :_____

Enter that drive which contains the PC DOS diskette. No <CR> is necessary. The next prompt will be:

CP/M Disk :_____

Enter that drive which contains the CP/M diskette. No <CR> is necessary. The next prompt will be:

AVAILABLE FORMATS:

1. NEC PC-8001
2. IMS 5000
3. DEC VT-18X
4. Heath/Zenith
5. CP/M-86

SELECT FORMAT :_____

Now, select the format which matches your CP/M format. No <CR> is necessary. The next prompt you see will be:

1. Copy from CP/M to IBM
2. Copy from IBM to CP/M
3. Print Directory of IBM disk
4. Print Directory of CP/M disk
5. Change parameters
6. Exit program

ENTER SELECTION :_____

You may now choose your function.

EXPLANATION OF FUNCTIONS

1. Copy from CP/M to IBM

This function is used to convert files from your CP/M disk onto your PC DOS disk. It is similar to using the COPY program (supplied with your PC DOS disk), where the CP/M disk is the source and the DOS disk is the destination. After selecting number 1, the following prompt will appear:

ENTER FILE NAME (<CR> to exit) :___

You may now enter the name(s) of the CP/M program to be transferred, followed by <CR>. (Global file name characters and the file naming conventions in general, which are discussed in the DOS manual pages 2-8 through 2-11, are supported.)

As a file is transferred, the file selection line will disappear and will be replaced with:

COPYING FILE d: filename.ext

If you are transferring multiple files, the above message will be updated as new files are being copied.

When the copy is complete, the file selection prompt will reappear. If this is the extent of your copying procedures, simply enter a <CR>.

2. Copy from IBM to CP/M

This function is in all ways similar to copying from CP/M to IBM, except that the IBM disk should be in the source drive and the CP/M disk should be in the destination drive.

3. Print Directory of IBM Disk

Selection of this function will PRINT the entire IBM disk directory on the bottom of your screen.

4. Print Directory of CP/M Disk

Selection of this function will PRINT the entire CP/M disk directory on the bottom of your screen.

5. Change Parameters

This function clears the screen and then displays the sign-on message. You are now ready to start over again with new file information.

6. Exit the Program

This function clears the screen and returns you to PC DOS.

EXPLANATION OF ERROR MESSAGES

You may encounter any of the five error messages explained below. To clear the error message and return to the selection menu, simply hit any key.

1. File Not Found

During a copy function, the file that you entered is not found on the source diskette.

2. Bad File Name

You have entered a file name either without a period or without a <CR> after eight characters.

3. Directory Full

You have completely filled the directory of your destination diskette. You must now insert a new diskette and "Change Parameters".

4. Disk Full

Like DIR FULL, except there is no more room on the diskette for data. You must now insert a new disk and "Change Parameters".

5. Fatal CP/M Error

Your diskette is unreadable for one of the following reasons:

- worn diskette
- wrong CP/M format
- inserting new diskette without restarting program
- any interference with the proper operation of the drive

Three new extended BDOS functions have been added to facilitate the movement of blocks of data throughout the entire PC memory address space under control of the Z-80 CPU.

FUNCTION 249:BLOCK MOVE DATA DOWN

Entry Parameters:
Register C: F9H

FUNCTION 250: BLOCK MOVE DATA UP

Entry Parameters:
Register C:FAH
Registers HL: pointer to block table

```
TABLE:
      DW    SOURCE OFFSET
      DW    SOURCE SEGMENT
      DW    DESTINATION OFFSET
      DW    DESTINATION SEGMENT
      DW    # OF BYTES TO MOVE
example:
      BDOS EQU 5
      Move:       LD  HL, MOVETBL
                  LD  C, 0FAH

                  CALL BDOS
      MOVTBL:
                  DW   HERE
                  DW   HERE$SEG
                  DW   THERE
                  DW   THERE$SEG
                  DW   2000
```

This example would cause 2000 bytes to be moved from "here" to "there".

The third added function is F8: This is essentially the same as ROM-BIOS ACCESS (FD), but adds ES: and Flags. The format is:

Register ES: 2 bytes
FLAGS : 1 byte, defined below

bit 7=SF
bit 6=ZF
bit 5=not used
bit 4=AF
bit 3=not used
bit 2=PF
bit 1=not used
bit 0=CF

Additionally, function FOH in the Baby Blue Manual bears further explanation. This function is used to intialize the Asynchronous Communications Adapter.

REGISTER BITS DEFINED

Bits 0-1 Word Length
10 = 7 bits
11 = 8 bits

Bit 2 Stop Bit
0 = 1 stop bit
1 = 2 stop bits

Bits 3-4 Parity
X0 = none
01 = odd
11 = even

Bits 5-7 Baud Rate
000 = 110
001 = 150
010 = 300
011 = 600
100 = 1200
101 = 2400
110 = 4800
111 = 9600