Enhancing CP/M-80

by William F. Dudley

Add A Command History Function

Imagine that you are working at your CP/M system and you type the following command:

```
A>pip lst:+some.fil,another.fil,andmore.fil[t8p]
```

and instead of hearing the printer start, you get:

```
INVALID PIP FORMAT: +SOME.FIL,ANOTHER.FIL,ANDMORE.FIL[T8P]
A>
```

Wouldn't it be nice to get that last line back to correct the typo? It's so tedious retyping the whole line again just for one small mistake.

Or imagine you are debugging a C program. Edit-Compile-Assemble-Link-Run-See errors. Edit-Compile-Assemble-Link-Run-See errors. Etc. etc. etc. ad nauseam.

It would be less tedious if you didn't have to retype the same five commands over and over again. (Even using a submit file, I still count three commands repeated endlessly.) Somebody *must* have thought of this before.

Well, campers, *they* did. Several big name operating systems have a command history feature:

a. Berkeley Unix. The cshell is several years old and it allows execution and editing of commands executed earlier. I find its syntax painful, however.

b. VMS 4.x. When Digital Equipment Corp. upgraded VMS, they added command history. They only allow access to the last twenty commands, which is sufficient for most needs. I think the editing characters are somewhat arbitrarily chosen, (unless you have a VT-100 terminal, where the arrow keys work).

c. AT&T UNIX V. A new shell for AT&T UNIX is called K-shell (Korn authored it). I find K-shell's command history very nice. It allows you to choose your command editor to look like either emacs or vi, both very popular editors on UNIX machines. It keeps the command history in a file, and stores many more than I need.

HOW DOES CP/M COMMAND HISTORY WORK?

To be useful for a CP/M environment, I worked up these requirements:

a. The command history is kept in (banked) ram to keep the speed to access commands high. I didn't want to slow down an already slow machine.

b. Sparse but useful feature set so that code size would fit within a standard CP/M system size. I still wanted to be able to use SYSGEN to make a bootable single-sided single-density 8" disk.

c. Use editor commands of an editor I was familiar with. I would have liked to use vi, but requirement (b) above dictated emacs, which was easier to implement.

d. Some trick features of the public domain CCP I was using had to be sacrificed to make room for history. These, plus other little used features and commands that history needed, are or could be supported by transient programs.

The resulting history feature works in the following manner. When faced with the CP/M prompt:

```
B>
```

you can type in a command just as before; however, the command line editing characters have changed and expanded:

^H	destructive backspace (delete left)
DEL	destructive backspace (delete left)
^U	discard and erase entire line
^X	discard and erase entire line
^B	non-destructive move cursor left
^F	non-destructive move cursor right
^A	move cursor to beginning of line
^E	move cursor to end of line
^D	delete character under cursor (+ close up line)
^L	toggle printer slaved to console
^C	warm boot CP/M if typed as first char on line
^M	(carriage return) execute line
^J	(line feed) execute line
^P	discard command buffer, fetch previous command
^N	discard command buffer, fetch "next" command (not implemented)

Otherwise: insert the char into the command line. If not at the end of the command, do a real insert (i.e. move characters to the right over to make room.) Control characters not listed above are echoed just as CP/M used to (e.g. ctrl-Y is echoed ^Y).

So now, fixing the command line offered at the beginning of the article would look like this:

```
A>pip lst:+some.fil,another.fil,andmore.fil[t8p]
INVALID PIP FORMAT: +SOME.FIL,ANOTHER.FIL,ANDMORE.FIL[T8P]
A>^P
A>pip lst:+some.fil,another.fil,andmore.fil[t8p]
```

Move the cursor to the "+" character. Use ^A to go to the beginning of the line. Type eight (8) ^F's to move right to the "+". Type ^D to delete the "+". Your cursor is under the "s" in some, and the line now looks like this:

```
A>pip lst:some.fil,another.fil,andmore.fil[t8p]
```

Type an "=" to insert it in front of the "some". Type a ⟨cr⟩ (carriage return) to execute the line.

```
A>pip lst:=some.fil,another.fil,andmore.fil[t8p]
```

To make the edit-compile-run loop a little easier, you now need type the commands only once each:

```
A>edit foo.c
   editing . . .
A>submit cc foo.c
   compiler runs . . .
A>foo
   foo runs . . .
```

Now, to re-edit, just type three (3) ^P's:

```
A)^P
A)foo^P
A)submit cc foo.c^P
A)edit foo.c(cr)
```

From now on, you need type only three ^P's to recover the next command in the cycle.

Some Transient Programs: Three transient programs are used to support history. INITHIST.COM clears the history buffer to the empty state. This is unnecessary, but nice. If you ^P back to before you started the current session, the garbage in the history buffer can make a small mess on your CRT screen.

GETHIST.COM copies the history buffer to the CP/M transient area so it can be saved with the CP/M SAVE command. This command is good for debugging the history feature, but mainly used with:

PUTHIST.COM to allow keeping the history buffer on disk from one session to the next. This way, you need type a command only once, ever, ever, ever. (Well, almost).

SYSTEM REQUIREMENTS

The code presented here can be used with little change on CP/M 2.2 systems using a Z80 or 64180 CPU. I use a 64180, so the memory management instructions reflect that. However, these can be easily changed to accomodate a typical Z80 bank select system.

The second requirement is some free ram that CP/M doesn't use. This can be (as in my system) another 64K bank, or in the simplest instance, just a couple of pages of ram above the BIOS.

A third requirement is an assembler, and some experience SYSGEN'ing CP/M systems. I use SLR180 by SLR Systems in Pennsylvania. Microsoft's M80 and L80 can be used just as well.

WHAT PARTS OF THE O.S. ARE MODIFIED?

Hold onto your hats. This job is most cleanly done by modifying both the CCP and the BDOS. Yes, I said BDOS, but its not as bad as it sounds. Honestly.

In order to shoehorn all this code into a nearly stock CP/M system, I had to scrap the BDOS system call 0AH, Get Line from CON:. The history line editor replaces this code completely, and there was no sense in wasting the ram occupied by the line editor built into the BDOS. There are two approaches to the BDOS mod: the patch and the gung-ho version. More about this later.

The actual command history buffer management is handled by the modified CCP. I have been using an old public domain CCP replacement for years, (like ZCPR1 or something). I gutted out some functions in it that I never used to make room for the history buffer management code.

INSTALLING THE CODE

Modifying the BDOS: The BDOS change is the first to go in, and gives you a good command line editor right away, without fixing the CCP.

I disassembled my BDOS, and then made room by changing most of the absolute jumps to relative jumps. When I then excised the existing Get Line function code, I had just enough room for the new line editor (Listing 1). I also needed to add a second entry point for the getline function, to allow editing of a line already in the line buffer. I *stole* the unused CP/M function 26H. This function has never been used, in CP/M 2.2, CP/M 3.0 (Plus), or even MS-DOS (which copied CP/M's BDOS function calls.)

Public domain disassemblers are available to disassemble the BDOS, or you can buy the product advertised as a source code generator for CP/M and do it automatically.

If disassembling the BDOS seems like too much work, the other way is to patch the BDOS to call an external routine that you locate in high ram above the BIOS. This could be loaded at cold boot time or kept in a BIOS rom that is resident at all times.

To facilitate the patch technique, some useful locations in a standard CP/M 2.2 BDOS are shown in Listing 2.

Modifying the CCP: First, you need the source to your CCP, either the disassembled Digital Research CCP, or one of the public domain CCP replacements. I use an old public domain CCP. (It's so old, it doesn't have wheel bytes or named directories).

Next, you need to decide which features you will sacrifice to make room for the history code. I took out LIST (TYPE to printer) since I always use PIP for that, and JUMP (to start execution at an arbitrary memory location) since I use DDT if I need something like that.

Now the code in Listing 3 can be merged with your CCP source. It fits in just after the code to print the CP/M prompt ")", and replaces the code from there until the code to convert the command line to upper case.

As an aside, I also added code to toggle on/off the lower to upper case conversion of command lines. This allows you to use the PIP Start and Quit flags with lower case strings.

The only hardware dependent parts of the code are the pointers to where the history buffer is kept in memory. If you have a 64180 and more than 64K, then the code will run as supplied. I keep a copy of the CCP and BDOS from 10000H to 115FFH for warm boots from ram. The history buffer is kept from 11600H to 12000H. Subroutines BANK0 and BANK1 do what you'd expect, respectively disabling and enabling the memory from 10000H to 1CFFFH. When enabled, that memory resides at logical addresses 0000H to 0CFFFH.

If you have only 64K of ram, another possibility is to configure your system size to 60K, which will leave some ram above the BIOS that CP/M doesn't know about. The history buffer can be kept there just as well.

After you have made your decisions concerning history buffer location, modify your source code and get it to assemble without errors. Then make up a new system with DDT:

```
A)SYSGEN
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)A
SOURCE ON A, THEN TYPE RETURN
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)(cr)

A)SAVE 38 CPM.COM
A)DDT CPM.COM
.iNEWCCP.HEX
.r[offset]
.iNEWBDOS.HEX
.r[offset]
.^C
A)SAVE 38 NEWCPM.COM
A)SYSGEN NEWCPM.COM
SYSGEN VER 2.0
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B
DESTINATION ON B, THEN TYPE RETURN
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)(cr)
```

Now the moment of truth. If you now reset and cold boot off your new system track (on the B diskette in this case) and get the A) prompt, congratulations. If "DIR" still works, a standing ovation. If ^P(cr) does a DIR again, then it's ticker tape parade time.

For the rest of you, I would suggest putting the BDOS mod in without changing the CCP right away. Everything should still work like a box-stock CP/M system, without history, but with a really fine command line editor facility.

Once the new BDOS (of BDOS patch) is in, the new CCP can go in. In order to help debugging the CCP mods for history, you may want to put in the transient programs GETHIST, PUTHIST, and INITHIST. (Listings 4, 5, and 6).

Another debugging trick for this is to temporarily put the history buffer into ordinary transient area ram, say at 8000H. This will allow you to examine the history buffer with DDT without having to worry about the complication of dealing with banked ram with DDT.

HOW IT WORKS

GETLINE function: The line editor in the BDOS follows closely the model of the original Digital Research code. Relative jumps are used to keep the code size down (they are slower than absolute jumps but speed is unimportant here). Some cleverness was required to make the routines for deleting and inserting characters work properly on any CRT with backspace the only cursor positioning sequence allowed. The temptation to use CRT escape sequences was immense, but the fact that I own two radically different CRT's forced me to do it right.

Backspaces are used to move the cursor left. To move the cursor right, the characters on the line are retyped. To erase, spaces are output, and then backspaces used to recover the cursor position.

The normal entry point, BDOS function 0AH. works as before, that is, the line buffer is cleared before the first character is gotten from the console in routine.

The second entry point, BDOS function 26H, (formerly unused) prints the current contents of the line buffer on the CRT, leaving the cursor at the end of the line, and then goes for an input character.

The CCP needs to know how the line was terminated, for if a carriage return or line feed was typed, the line is to be executed. If however, a ^P or ^N was typed, the line in the buffer is to be discarded and the previous or next line in the history buffer is to be fetched.

For this reason, BDOS function 0AH and the new function 26H now return the last character typed to the caller in the Accumulator (and the L register). CP/M applications that don't need this information are unaffected since the accumulator is expected to be clobbered by the BDOS call.

CCP History Function: When the CCP is first entered, it calls CP/M function 0AH to get a command line. If the call returns with a value of carriage return or linefeed, the command in the buffer is to be executed. When a command is executed by the CCP, it is first put into the history buffer. The history buffer is designed as a combination fifo (first in first out) buffer and a doubly linked list. The exact layout is as follows:

```
cbuff0:    dw    len?h   ;pointer to previous command length byte
len0h:     db    len0    ;length of first command
com0:      ds    len0    ;first command
len0t:     db    len0    ;length of first command
len1h:     db    len1    ;length of second command
com1:      ds    len1    ;first command
len1t:     db    len1    ;length of second command
```

To copy a command into the history buffer, first the existing commands in the buffer are moved up in the buffer to make room at the input end. The new command is then copied into the buffer beginning, with a byte representing its length at each end of the command copy in the buffer. Finally, the previous command buffer pointer is reset to point to the most recent command in the buffer.

The copying of the latest command into the history buffer is done so that the history buffer never expands beyond its defined limits. The limits must be on page boundaries.

When the CCP gets a return value of ^P from the BDOS getline call, it goes to the fifo for the previous command. The command pointer is always pointing to the length byte for the previous command in the history buffers. That command is copied to the CCP command line buffer, and the previous command pointer is incremented by the length byte plus two so that it points to the next command back in the fifo (back in time).

The CCP now calls BDOS function 26H to edit the command which is in the command line buffer. Then, depending on the return value, the CCP either saves and executes the command, or fetches the previous command for editing again.

BUGS & NOTES

The only known bug is that command lines with embedded control characters are displayed incorrectly when editing. This means that when editing those lines, it is necessary to keep the real cursor position in your head. Another way is to always move using ^A and ^F's, so that the line characters will be echoed properly. I wanted to use the code, and not be debugging it for the rest of my life, so I decided to just use the code this way. If anyone fixes this bug, I'd like a copy of the fix.

The ^N command to get to the *next* command in the history buffer was not implemented due to lack of space in the CCP. If you overshoot the command you want with an extra ^P, the only way to recover is to send an empty line with a carriage return to the CCP, which will reset the command pointer to the beginning of the buffer.

VMS is a trademark of Digital Equipment Corp.
M80, L80, and MSDOS are trademarks of Microsoft.
SLR180 is a trademark of SLR Systems.

Bill Dudley is a Member of Technical Staff at AT&T Bell Laboratories in New Jersey, where he designs hardware for Cellular Mobile Telephone Service. He has a BSEE and M.Eng. from Cornell University. When not hacking his CP/M system or justifying why he hasn't got an MS-DOG system yet, he can be found riding his motorcycle.

Listing 1

```
;   GETLIN.MAC
;
;   bdos getlin replacement
;   entry point to allow editing of line already
;   in the buffer.
;
;   William F. Dudley
;   Dec. 22, 1985
;
wboot0   equ    0
lf       equ    10
cr       equ    13
bs       equ    8     ;delete previous char
ctrlb    equ    2     ;emacs back char  <--
ctrlf    equ    6     ;emacs fwd char   -->
ctrln    equ    0eh   ;emacs next line    ^
                      ; (returns to caller if typed)
ctrla    equ    1     ;emacs begin line  <<--
ctrll    equ    0ch   ;echo to lst: (replaces ^p)
ctrlp    equ    10h   ;emacs prev line
                      ; (returns to caller if typed)
ctrle    equ    5     ;emacs end of line -->>
ctrlk    equ    0bh   ;emacs erase to eol
ctrld    equ    4     ;emacs delete char
ctrlc    equ    3
ctrlx    equ    18h
ctrlu    equ    15h
del      equ    7fh
;
;<<< system call >>>   buffer input from console
;
; buffer format:          mx,n,c,c,c,c....
;       mx= size of buffer
;        n= # of characters entered
;        c= characters
;  special characters:
;       rub/del = remove and echo last character
;       ctrla   = emacs move cursor to beginning of line
;       ctrlb   = emacs move cursor left one character
;       ctrlc   = reboots system if typed at start of input
;       ctrld   = emacs delete character under cursor
;       ctrle   = emacs move cursor to end of line
;       ctrlf   = emacs move cursor right one character
;       ctrlh   = remove and do crt rubout
;       ctrlj   = (line feed) end of input
;       ctrlm   = (return) end of input
;       ctrll   = toggle list device slave
;       ctrlu   = scratch input buffer after new line
;       ctrlx   = scratch input buffer & back up to
;                     start of line
;
code0a: ld      hl,(userde) ;hl= buffer pointer
        ld      c,(hl)      ;c= size of the buffer.
        inc     hl
        push    hl          ;save pointer to # of characters
        ld      b,0         ;clear character counter
        ld      a,(curcol)  ;get current column position
        jr      coment      ;common entry point
```

```
code26:
        ld      hl,(userde) ;hl= buffer pointer
        ld      c,(hl)      ;c= size of the buffer
        inc     hl
        push    hl          ;save pointer to # of characters
        ld      b,(hl)      ;b=no. of chars in buffer already
        ld      a,(curcol)
        push    af
        push    bc
        call    tline    ;print the line
        pop     bc     ;restore buffer size, line length
        pop     af     ;restore cursor position after prompt
coment:
        ld      (colsav),a  ;save it
        ld      d,b         ;cursor position == end of line
cod0aa: push    bc          ;save the character count
        push    hl          ;save the buffer pointer
cod0ab: call    gconch      ;get a character from console
        and     7fh         ;strip parity
        ld      e,a         ;current char to e
        pop     hl          ;restore buffer pointer
        pop     bc          ;restore character count
        cp      cr
        jr      z,aqvec         ;if end of input
        cp      ctrlp
        jr      z,aqvec         ;if end of input
        cp      ctrln
        jr      z,aqvec         ;if end of input
        cp      lf
aqvec:  jp      z,cod0ag        ;if end of input
        cp      bs
        jr      z,bckspc
        cp      del
        jr      nz,delchk       ;if not delete
                                ;here for back space
bckspc: ld      a,d             ;a= character count
        or      a
        jr      z,aavec         ;if buffer empty, forget it
        call    bspcc
        push    hl
        push    de
movlin: inc     d
        inc     hl
        push    bc
        ld      a,b
        sub     d       ;no of chars to the right of cursor
        ld      b,a
        push    bc
        push    hl
; print line to end of line omitting current char
        call    tline
        ld      c,' '
        call    pchar
        call    pchar
        pop     hl
        pop     bc
        push    bc
        inc     b
        inc     b
; backspace up to original position minus 1
bslp:   call    bspc
        djnz    bslp
        pop     bc
        ld      a,b
        or      a
        jr      z,bckext
        ld      c,b
        ld      b,0
        ld      d,h
        ld      e,l
        inc     hl
; move chars in buffer to close up gap
        ldir
bckext: pop     bc
        dec     b
        pop     de
        pop     hl
aavec:  jp      cod0aa
delchk: cp      ctrld           ;^d ?
        jr      nz,fwd
        ld      a,b
        cp      d
        jr      z,aavec  ;cursor is after last char in buffer
        push    hl
        push    de
        jr      movlin
fwd:    cp      ctrlf           ;^f ?
        jr      nz,eolchk
        ld      a,b
        sub     d
        jr      z,aavec
        ld      a,1
        jr      foward   ;end of line check will occur there
eolchk: cp      ctrle           ;^e ?
        jr      nz,back
        ld      a,b
        sub     d
```

```
; print (move) 'a' chars forward
foward: ld      e,a
        add     d
        ld      d,a
        push    bc
        ld      b,e
        call    tline
        pop     bc
        jr      aavec
back:   cp      ctrlb           ;^b ?
        jr      nz,beglin       ;nope
        ld      e,1             ;1 char to backup
        jr      backup
beglin: cp      ctrla           ;^a ?
        jr      nz,eraeol       ;nope
        ld      e,d             ;no of chars to backup
; backup e char's
backup: ld      a,d
        or      a
        jr      z,aavec         ;at beginning, forget it
        push    bc
        ld      b,e
backlp: call    bspcc
        djnz    backlp
        pop     bc
        jr      aavec
eraeol: cp      ctrlk           ;^k ?
        jr      nz,tglist
        ld      a,b
        sub     d               ;no of chars to erase
        jr      z,aavec         ;at end, ignore
        ld      b,d
        push    bc
        ld      b,a
        ld      a,(curcol)
        push    af
        push    hl          ;save so current hl is final hl
; print chars to move cursor to end of line
        call    tline
        pop     hl
        ld      a,(curcol)
        ld      (colsvl),a
        pop     af
        ld      (curcol),a
; now go rubout back over the stuff to be erased
        jr      cod0al
tglist:
        cp      ctrll
        jr      nz,cod0af       ;if not ^l
                                ;here to toggle the list slave
        push    hl              ;save hl-reg
        ld      hl,listsw       ;point to list slave switch
        ld      a,1             ;toggle it
        sub     (hl)
        ld      (hl),a
        pop     hl              ;restore hl-reg
        jr      aavec           ;loop back
cod0af: cp      ctrlx
        jr      z,clrlin        ;yes, ^x
        cp      ctrlu
        jr      nz,cod0an       ;no, not ^u
                                ;here for crt line clear
clrlin: pop     hl              ;clean stack
cod0ag: ld      a,(colsav)      ;check print column position
        ld      hl,curcol       ;against cursor position
        cp      (hl)            ;and:
        jp      nc,code0a       ;if end of rubout, start over
        dec     (hl)            ;else dec column position count
        call    rubout          ;do the crt rubout
        jr      cod0ag          ;loop till entire entry cleared
cod0al: push    hl              ;retype done,
        ld      a,(colsvl)      ;check print column
        or      a               ;jump if at beginning
        jr      z,abvec
        ld      hl,curcol
        sub     (hl)
        ld      (colsvl),a
cod0am: call    rubout
        ld      hl,colsvl
        dec     (hl)
        jr      nz,cod0am
abvec:  jp      cod0ab
;echo character to console
cod0an: call    putcon
        ld      a,b
        sub     d
        jr      z,eolchr  ;last char in line, no move needed
        push    bc
        push    de
        ld      b,a             ;no of chars to print
        ld      c,a             ;save in c for later
        push    bc
        call    tline
        pop     bc
        push    hl              ;save buffer pointer to end
inbklp: call    bspcc           ;back up from end of line to
                                ; insert position
        djnz    inbklp
        pop     hl
        ld      d,h             ;hl points to end of buffer now
        ld      e,l
        inc     de
        lddr                    ;move rest of line out one char
        pop     de
        pop     bc
;       place character in buffer
eolchr: inc     b
        inc     d
        inc     hl                      ;bump buffer pointer
        ld      (hl),e                  ;install character
        ld      a,(hl)                  ;get the previous character
        cp      ctrlc
        ld      a,b                     ;a= count
        jr      nz,cod0ap               ;if not abort
        cp      1                       ;check for first character
        jp      z,wboot0                ;if first character
cod0ap: cp      c
        jp      c,cod0aa                ;if not at end of buffer
; here on end of input
cod0aq: pop     hl                      ;restore pointer to count
        ld      (hl),b                  ;return the count in buffer
        ld      c,cr                    ;send a return and exit
        call    pchar
        ld      a,e
        jp      retaf
; bspcc-print a backspace, 2 if current char is control char
; bspc -print a backspace
bspcc:  call    bspc
        dec     d
        ld      a,' ' - 1
        cp      (hl)
        dec     hl                      ;doesn't affect flags
        ret     c                       ;not control char, only one
                                        ; screen position
bspc:   ld      e,bs
; print char in e register
putcon: push    bc
        push    hl
        ld      c,e
        call    pchrcon
        pop     hl
        pop     bc
        ret
; print chars from current cursor position to end of line
tline:  ld      a,b
        or      a
        ret     z
tlinel: inc     hl
        ld      e,(hl)
        call    putcon
        djnz    tlinel
        ret
;end needed if code is to be assembled as standalone patch
;       end
```

Listing 2

```
; useful locations in the bdos
; referenced to the BDOS start
; which is found in locations 6 and 7
; on "normal" origin 0 CP/M systems
;
code0ap equ     bdos + 05BH
code26p equ     bdos + 093H
codetb  equ     bdos + 47H
colsav  equ     bdos + 030BH
colsvl  equ     bdos + 30AH
curcol  equ     bdos + 030Ch
gconch  equ     bdos + 0FBH
listsw  equ     bdos + 30DH
pchar   equ     bdos + 148H
pchrcon equ     bdos + 17FH
retaf   equ     bdos + 301H
rubout  equ            7FH
userde  equ     bdos + 343H
userhl  equ     bdos + 345H
wboot0  equ              0
```

Listing 3

```
        title   'enhanced z80 ccp for 2.x  3/18/82'
; William F. Dudley
; Dec. 06, 1985 - modified for last command memory.
;
;       equates
        include listing.4a      ;global equates for history
;
; TUNE THESE FOR YOUR SYSTEM SIZE
ccploc  equ     0d400h  ;start of ccp in memory
;
;I had to put CCP's stack above my BIOS to make room for code
histk   equ     0F7FAH  ;tune this as your system allows
; STOP TUNING HERE
;
ctrlp   equ     10h     ;previous command line
ctrln   equ     0eh     ;previous command line
supres  eq      true    ;suppress user 0 prompt (true)
sprmpt  equ     '='     ;submit prompt character
cprmpt  equ     '>'     ;keyboard prompt character
defusr  equ     0       ;default user # (com files)
maxusr  equ     15      ;maximum user number
biosiz  equ     380h    ;bios size (get protection)
wboot   equ     0       ;cp/m warm boot address
udflag  equ     4       ;user number in high nybble,
;                       ;disk in low
tbuff   equ     80h     ;default disk i/o buffer
tfcb    equ     5ch     ;default fcb buffer
; macro to use 64180 output instruction
; replace out0 references with normal Z80 out
; instruction if using a Z80 cpu
out0    macro   port
        db      0edh,39h,port
        endm
cbar    equ     0bah    ;64180 common bank register
bbr     equ     0b9h    ;64180 base bank register
; input command line from user console
rbl:    call    subkil  ;erase $$$.sub if present
        call    setud   ;set user and disk
        ld      a,cprmpt        ;print '>' prompt
        call    conout
        ld      c,0ah   ;read (possibly old) command line
                        ; from user
; from HERE on is neat new stuff
rcmd    equ     $ - 1   ;allows changing of getlin entry point
        ld      de,mbuff
        call    bdos
        push    af
        call    bank1
        pop     af
        cp      ctrlp
        jr      nz,newcmd
        ld      hl,(hdptr)
        ld      b,0
        ld      a,(hl)
        cp      buflen
        jr      c,rb2           ; command is legal buffer length
        ld      a,buflen
rb2:    ld      de,cbuff ; dest will be ccp buffer (mbuff + 1)
        ld      c,a
        inc     c               ; add 1 for length byte
        ldir                    ; hl now points to end length byte
        inc     hl              ; skip length byte
        ld      a,h
        cp      high hbuffe
        jr      nc,pchk1        ; hl points past hist buffer end
        cp      high cbuff0
        jr      nc,pchk2        ; hl doesn't point below hist
                                ;  buffer beginning
pchk1:  ld      hl,cbuff0       ;past end of hstbuf, wrap to
                                ;  beginning
pchk2:  ld      (hdptr),hl
        call    bank0
        ld      a,26h
        ld      (rcmd),a   ;patch so cpm line editor invoked
vrstrt: jp      restrt
newcmd:
        ld      a,(cbuff)
        ld      c,a
        ld      b,0
        inc     c               ; buffer length + length byte
        push    bc              ; length of command
        inc     c               ; add 1 for trailing cmdlen byte
        ld      hl,hbuffl
        or      a
        sbc     hl,bc
        push    hl              ; # of chars in histbuf to move
        ld      hl,hbuffe
        push    hl              ; destinatin
        sbc     hl,bc           ; source
        pop     de              ; dest --> de
        pop     bc              ; # to move --> bc
        lddr
        ld      de,cbuff0 ; dest is start of hist buffer
        ld      hl,cbuff  ; source is ccp command buffer
        pop     bc              ; get command length
        push    de              ; save start of hist buffer
        push    bc              ; save command length
        ldir                    ; copy command to hist buffer
        pop     bc              ; get command length
        ex      de,hl           ; hl-->end of command in hist buffer
        dec     c
        ld      (hl),c          ; put at end of buffer image in hbuff
```

```
         pop      hl         ; get start of hist buffer
         ld       (hdptr),hl; put in hist buffer pointer
         call     bank0
         ld       a,0ah
         ld       (rcmd),a ;unpatch so normal cpm getlin invoked
; capitalize string (ending in 0) in cbuff
cnvbuf:  ld       hl,cbuff         ;pt to user's command
         ld       b,(hl)  ;char count in b
         inc      b       ;+1 to cover 0 case
cbl:     inc      hl      ;next char
         ld       a,(hl)  ;get character
ucasev   equ      $ + 1   ;allows patching out ucase call
         call     ucase   ;force upper case
         ld       (hl),a  ;store character
         djnz     cbl     ;do rest of command line
         ld       (hl),b  ;store ending <null> (b=0)
         ld       hl,cibuff ;set cmd ptr to 1st char
         ld       (cibptr),hl
         ret
;  This ^^^ ret needs to be ahead of ucase for CASE to work
;  convert char in a to upper case
ucase:   cp       'a'     ;less than lower case a?
         ret      c       ;yes
         cp       'z'+1   ;greater than lower-case z?
         ret      nc      ;yes
         res      5,a     ;no, clear lower case bit
         ret
;  form:          case
case:    ex       (sp),ix ;put exit address on stack
casflg   equ      $ + 1
         ld       a,0     ;overwritten, initialized to ucase
                          ;  mode
         xor      1
         ld       (casflg),a
         ld       a,low(ucase)
         ld       b,'1'
         jr       z,caspat
         dec      a       ;points to the return before ucase
         dec      b       ;is a '0' now
caspat:  ld       (ucasev),a
         ld       a,lf
         call     conout
         ld       a,b
         jp       conout  ;restart ccp without login
; make the second bank active
bank1:   ld       a,10h   ;second 64k bank
         jr       obbr
; make the first bank active
bank0:   xor      a
obbr:    out0     bbr
         ret
```

Listing 4

```
         title    'get history buffer into save image'
; gethist.mac
;
; to use:
;        A>gethist
;        A>save 13 histname.ext
;
; for z80 with simple bank select: write to port 0 to
;               change bank
;        1 to port 0 -> normal ram from 0000 to 7fffh
;        2 to port 0 -> second bank from 0000 to 7fffh
;
         include listing.4a  ;global includes for history
;
         aseg
         org      tpa
;
; copy real program up to NONBNK above bank selected memory.
         ld       hl,mstart
         ld       de,nonbnk
         ld       bc,pend-pstart
         ldir
         jp       pstart
mstart:
         .phase   nonbnk
; phase pseudo-op causes assembler to generate
; code for different run location than load address
pstart:
         ld       a,bank2          ;switch in second bank
         out      (bank),a
         ld       hl,hbuff
         ld       de,empty  ;copy history buffer to unbanked
                            ; ram above this code
         ld       bc,hbuffl
         ldir
         ld       a,bank1   ;switch back to first bank
         out      (bank),a
         ld       hl,empty
         ld       de,tpabuf ;copy buffer image down to low
                            ; memory for CP/M SAVE command
         ld       bc,hbuffl
         ldir
         ret                ;return to cpm without warm boot
pend:
empty:   ds       1
         .dephase
;
         end
```

Listing 4a

```
; listing.4a -- global equates for history
; these should be tuned for your system hardware,
;  particularly the history buffer location
;
hbuff    equ      1600h            ;pointer to current hbuff line
hdptr    equ      hbuff
cbuff0   equ      hbuff + 2        ;contents is length of most
                                   ; recent command
hbuffe   equ      2000h            ;last location of history
                                   ; buffer + 1
hbuffl   equ      hbuffe - hbuff ;length of history buffer
bank     equ      0       ;bank register
bank2    equ      2
bank1    equ      1
nonbnk   equ      8000h   ;start of non-banked memory
; STOP TUNING
tab      equ      09h     ;horizontal tab
lf       equ      0ah     ;line feed
ff       equ      0ch     ;form feed
cr       equ      0dh     ;carriage return
false    equ      0
true     equ      not false
;
wboot    equ      0       ;cp/m warm boot address
;                         ;disk in low
bdos     equ      5       ;bdos function entry point
tpa      equ      100h    ;base of tpa
tpabuf   equ      tpa + 100h
```

Listing 5

```
         title    'put history buffer from save image'
; puthist.mac
;
; to use:
;        A>get 100 histname.ext
;        A>puthist
; or if using CCP without GET command
;        A>ddt histnam.com
;        .^C
;        A>puthist
;
; for z80 with simple bank select
         include listing.4a      ;global equates for history
;
         aseg
         org      tpa
;
; move real program to high memory above bank selected ram
         ld       hl,mstart
         ld       de,nonbnk
         ld       bc,pend-pstart
         ldir
         jp       pstart
mstart:
         .phase   nonbnk
pstart:
         ld       de,empty
; move hist buffer image to high ram above bank selected ram
         ld       hl,tpabuf
         ld       bc,hbuffl
         ldir
         ld       a,bank2          ;select the second bank
         out      (bank),a
; move hist buffer image to proper place in banked ram
         ld       de,hbuff
         ld       hl,empty
         ld       bc,hbuffl
         ldir
         ld       a,bank1          ;re-select the first bank
         out      (bank),a
         ret
; return to CP/M CCP without re-booting
pend:
empty:   ds       1
         .dephase
;
         end
```

Listing 6

```
         title    'clear history buffer'
; inithist.mac
;
; to use:
```

continued on page 85