ANSI.SYS without the hassle

The ANSI control sequences open up fascinating opportunities for screen display and DOS function-key reassignments. Michael Mefford shows you how to implement them without loading ANSI.SYS.

Would you like to control screen colours with a simple TYPE command? Wouldn't it be nice to redefine your function keys to do meaningful work in DOS? ANSI.COM, this issue's utility, is designed to give you that control without incurring the performance penalties involved in using IBM's own ANSI.SYS to do the job.

ANSI.SYS is a DOS device driver that provides extended screen and keyboard control to any application that uses DOS functions for its input or output. Like other device drivers, it's installed at boot time from the CONFIG.SYS file, and the small divot of memory it carves out can't be recovered after it is installed. Since ANSI looks for its control sequences by examining each character both when it comes in from the keyboard and when it is output to the screen, performance can suffer. The effect is negligible on the performance of fast machines, though it is noticeable with slower processors.

The ANSI driver gives access to the ANSI functions only to programs that use DOS services for I/O. For performance reasons, most of today's regular applications circumvent DOS I/O, either by using the BIOS or addressing the hardware directly. (If you don't have ANSI.SYS installed, you'll know right away whether a particular program requires it, because a telltale series of escape codes will be strewn across the screen.)

The program that all of us use that *can* profit from ANSI, however, is COMMAND.COM. You might not have thought of COMMAND.COM as a *program*, but it is, and the most common way to make use of ANSI control is through COMMAND.COM's PROMPT command. With ANSI's aid, the dull white on black A> prompt can become a colourful and more informative command-awaiting screen. The seldom-used DOS function keys (F3 is normally the only useful one) can be defined to type out things you have to enter frequently, such as DIR, TYPE and COPY.

ANSI.COM is a memory-resident utility that can be installed or uninstalled at any time. The utility performs all the functions of ANSI.SYS, but it adds the abilities to work at higher screen speed, to be activated or deactivated at will, to use a large-size, flushable key-reassignment buffer, and to work with video modes of any width or length — including the 43-line EGA mode.

Getting a copy of ANSI

ANSI.COM is available for downloading from Microtex *6663# on Telecom's Viatel; or by sending a blank formatted 5.25in disk with a stamped, self-addressed package to Cathie Kennedy, *APC*, 47 Glenhuntly Road, Elwood 3184. The source code, ANSI.ASM, together with ANSI.BAS (a Basic listing that will automatically produce ANSI.COM when you run it) is printed here; however, both are also available for downloading from Microtex.

When entered initially, either at the DOS prompt or as part of your AUTOEXEC.BAT file, ANSI installs itself like any other TSR. Since it makes no sense to install a duplicate console driver, however, ANSI.COM will not install if ANSI.SYS is already loaded. If you are presently using ANSI.SYS, therefore, remove the statement DEVICE = ANSI.SYS from your CONFIG.SYS file, add ANSI.COM to your AUTOEXEC.BAT and reboot your machine.

The full ANSI.COM syntax is

```
ANSI [FAST | SLOW] [ON | OFF]
    [/B nnn] [/C] [/U]
```

The first two optional parameter pairs, FAST | SLOW and ON | OFF, control the status of the program. (The vertical bar between the options simply means that you may select only one of each pair.)

The FAST parameter is the default, and it routes ANSI's output directly to the screen buffer. This is the desirable mode for most systems, with the exception of machines that use IBM's CGA adaptor. Writing directly to the CGA video buffer causes 'snow' to appear on the screen. In its SLOW mode, ANSI uses BIOS calls to handle screen output, avoiding the blizzard. Sending output via the BIOS — the method used by ANSI.SYS itself — involves additional overhead that slows output. On the other hand, if you use a screen-capture program and you find that it doesn't work in the FAST

mode, you can simply switch to using the SLOW mode.

The ON | OFF option activates or deactivates the ANSI filter. The default is ON, but if you turn ANSI OFF, your machine will act as if ANSI.COM weren't loaded. You can change either the FAST | SLOW or the ON | OFF options at install time, or you can change them later by entering ANSI followed by the desired mode. ANSI is smart enough not to install itself more than once. Once it's installed, entering ANSI without parameters will produce a status report.

The /B switch controls the amount of memory ANSI reserves for key reassignments. The *nnn* parameter is the decimal number of the desired buffer size in bytes. By default, ANSI.COM reserves the same sized buffer — 220 bytes — as does ANSI.SYS. If you've found yourself restricted by the limited buffer of ANSI.SYS, you'll appreciate the 60k maximum assignable under ANSI.COM. Alternatively, if you don't plan to reassign any keys, you can set the buffer size to 0. Selecting a zero buffer shrinks ANSI.COM down to its minimum resident size, a little under 2k.

'PROMPT provides the easiest way to send ANSI escape commands to the console.'

Since the memory buffer must be allocated at load time, you can only use the /B switch when you are installing the ANSI program.

One of the problems with ANSI.SYS is that once key reassignments are made, they are difficult to remove. Suppose, for example, that you have assigned the double quote to the single quote, eliminating the need to use the Shift key. If you later want to restore the single quote key to its original meaning, your only recourse is to make an additional assignment, defining the single quote back to itself. Since ANSI.SYS does not remove the old reassignment from the buffer, you have that much less room for new key definitions. ANSI.COM takes the same approach; not only can its buffer be much larger, but you can also flush out the entire reassignment buffer at any time by invoking ANSI with the /C (clear) switch.

You can uninstall ANSI.COM by entering its name along with the /U switch. As with all TSRs, you must always uninstall in reverse order of installation if you wish to reclaim the memory it occupies. The ability to uninstall and reinstall makes it possible to change the key reassign-

```
;-----------------------------------------------------------------------;
;  ANSI.COM - Replacement for the ANSI.SYS console device driver.       ;
;  Unlike ANSI.SYS which must be installed at boot time, ANSI.COM       ;
;  can be installed and uninstalled at anytime.  Enhancements include   ;
;  a fast screen write and variable sized keyboard reassignment buffer. ;
;-----------------------------------------------------------------------;
_TEXT           SEGMENT PUBLIC 'CODE'
                ASSUME  CS:_TEXT,DS:_TEXT,ES:_TEXT,SS:_TEXT
                ORG     100H
START:          JMP     INITIALIZE

SIGNATURE       DB      "ANSI 1.0 (C) 1989 Ziff Communications Co.",CR,LF
PROGRAMMER      DB      "PC Magazine ",254," Michael J. Mefford",CR,LF,LF,"$",26

CR              EQU     13
LF              EQU     10
SPACE           EQU     32
ESC_CHAR        EQU     27
SINGLE_QUOTE    EQU     39
DOUBLE_QUOTE    EQU     34
BELL            EQU     7
BS              EQU     8

OFF             EQU     1
ON              EQU     2
SLOW            EQU     4
FAST            EQU     0
STATUS_MASK     EQU     11111100B

ANSI_STATE      DW      ESC_STATE
STATUS          DB      ON OR FAST
PARAMETERS      DB      "OFF ON  SLOWFAST"
LAST_PARAMETER  EQU     $ - PARAMETERS

OLD_INT_29      DW      ?,?
OLD_INT_16      DW      ?,?
OLD_INT_21      DW      ?,?

ATTRIBUTE       DB      7
SAVE_POSITION   DW      0
LINE_WRAP       DB      ON
QUOTE_TYPE      DB      ?
ESC_COUNT       DW      0
NUMBER_COUNT    DW      0

FORMAT_CHARS    DB      CR,LF,BS,BELL
FORMAT_LENGTH   EQU     $ - FORMAT_CHARS

COMMAND_TABLE   LABEL   BYTE
DB   "H", "A", "B", "C", "D", "f", "n", "s", "u", "K", "m", "h", "l", "p", "J"
COMMAND_LENGTH EQU      $ - COMMAND_TABLE

DW   CURS_POSITION, CURSOR_UP,      CURSOR_DOWN, CURS_FORWARD, CURS_BACKWARD
DW   HORZ_VERT_POS, DEVICE_STATUS,  SAVE_CURSOR, RESTORE_CURS, ERASE_IN_LINE
DW   SGR,           SET_MODE,       RESET_MODE,  REASSIGNMENT, CLS
COMMAND_END     EQU     $ - 2

ATTRIBUTE_TABLE         LABEL   BYTE
DB   00,01,04,05,07,08,30,31,32,33,34,35,36,37,40,41,42,43,44,45,46,47
ATTRIBUTE_LENGTH        EQU     $ - ATTRIBUTE_TABLE

;Format: AND mask,OR mask
DB      000H,07H, 0FEH,08H, 0FBH,01H, 0FFH,80H, 0F8H,70H, 088H,00H
DB      0F8H,00H, 0F8H,04H, 0F8H,02H, 0F8H,06H, 0F8H,01H, 0F8H,05H
DB      0F8H,03H, 0F8H,07H, 08FH,08H, 08FH,40H, 08FH,20H, 88FH,60H
DB      08FH,10H, 08FH,50H, 88FH,30H, 88FH,70H
ATTRIBUTE_END           EQU     $ - 2

BIOS_ACTIVE_PAGE        EQU     62H

ACTIVE_PAGE     DB      ?
ADDR_6845       DW      ?
BIOS_CRT_MODE           EQU     49H
CRT_MODE        DB      ?
CRT_COLS        DW      ?
CRT_LEN         DW      ?
CRT_START       DW      ?
CRT_DATA_LENGTH         EQU     $ - CRT_MODE

CURSOR_POSN     LABEL   WORD
CURSOR_COL      DB      ?
CURSOR_ROW      DB      ?
CRT_ROWS        DB      ?

DOS_INPUT       DB      OFF
BUSY_FLAG       DB      OFF
REASSIGN_FLAG   DB      OFF
REMOVE_FLAG     DB      ON
REASSIGN_COUNT  DW      0
REASSIGN_POS    DW      ?
FUNCTION_16     DB      ?
ZR              EQU     10000000B

ESC_BUFFER_SIZE         EQU     126
REASSIGNMENT_DEFAULT    EQU     200
REASSIGNMENT_SIZE       DW      REASSIGNMENT_DEFAULT
REASSIGNMENT_MAX        EQU     60 * 1024
REASSIGN_END            DW      REASSIGNMENT_BUFFER
BUFFER_END              DW      REASSIGNMENT_BUFFER + REASSIGNMENT_DEFAULT

;-----------------------------------------------------------------------;
; INT 29 is an undocumented interrupt called by DOS for output to the console. ;
;-----------------------------------------------------------------------;
ANSI_INT_29     PROC    FAR

                PUSH    AX                      ;Save all registers.
                PUSH    BX
                PUSH    CX
                PUSH    DX
                PUSH    SI
                PUSH    DI
                PUSH    DS
                PUSH    ES
                PUSH    BP

                CLD                             ;All string operations forward.
                MOV     BX,CS                   ;Point to our data segment.
                MOV     DS,BX
                MOV     ES,BX
                CALL    ANSI_STATE              ;Call the current state of
                                                ; the ANSI Esc sequence.
                POP     BP
                POP     ES
                POP     DS
                POP     DI
                POP     SI
                POP     DX
                POP     CX
                POP     BX
                POP     AX                      ;Restore all registers and
                IRET                            ; return.
ANSI_INT_29     ENDP

;-----------------------------------------------;
ANSI_INT_21     PROC    FAR
                PUSHF
                CMP     AH,0AH                  ;If DOS Int 21 functions Ah,
                JZ      INPUT                   ; 7h, 1h or console input (6h)
                CMP     AH,7                    ; then tell INT 16, DOS input
                JZ      INPUT                   ; is active.
                CMP     AH,1
                JZ      INPUT
                CMP     AH,6
                JNZ     QUICK21_EXIT
                CMP     DL,0FFH
                JZ      INPUT
QUICK21_EXIT:   POPF                            ;If not, let the original
                JMP     DWORD PTR CS:OLD_INT_21 ; interrupt process the call.

INPUT:          POPF
                MOV     CS:DOS_INPUT,ON         ;INT 16 handler flag.
                PUSHF
                CALL    DWORD PTR CS:OLD_INT_21 ;Emulate an interrupt.
                MOV     CS:DOS_INPUT,OFF        ;Turn flag back off.
                RET     2                       ;Return with current flags.
```

continued . . .

ANSI.ASM The source code from which to assemble, link and EXE2BIN ANSI.COM. Use the program to implement ANSI control sequences

ment buffer size without fully rebooting your machine. Note that all current key reassignments are lost when you do so, and therefore will have to be reloaded into the resized buffer.

ANSI.COM's own parameters are not case-sensitive, and only the first two characters of the status parameters are required. For example, 'ANSI fa' would change the status to FAST. The only caveat with ANSI.COM is the standard one: there is always the potential for conflicts among multiple TSRs.

ESC sequences

ANSI works as a console filter and looks for an escape sequence as its cue to take some special action. The IBM ANSI commands supported by ANSI.SYS and ANSI.COM are a subset of the ANSI standard. The specific commands you can enter are listed in the box 'The IBM ANSI command set'. Each ANSI escape sequence must begin with two characters: an ESC followed by [, the left bracket. The ESC character does not consist of the three letters E-S-C. ESC instead stands for the control character, decimal 27, hex 1B. ESC is among the 32 characters in the ASCII table that appear before the first text character (the space, character 32). These initial characters are known as control characters because they are used to control devices. ESC is Ctrl-[, which is often printed as ^[. The ASCII display interpretation is a small left arrow.

Entering the ESC character itself is always the hardest part of issuing an escape sequence, so I'll discuss ways to do it.

The DOS PROMPT command provides the easiest way to send ANSI escape commands to the console. PROMPT has a subset of commands all of its own, called metastrings. A metastring consists of two characters, a $ followed by the command character. These are usually used to set a prompt that's quite independent of ANSI. (The DOS manual contains a complete listing of PROMPT metastrings.) The $e metastring generates the ESC character (27), however, and so can be used to introduce an ANSI control sequence. With ANSI.COM (or ANSI.SYS) loaded, a PROMPT command that would keep the screen clear is

```
PROMPT $e[2J
```

Note that while PROMPT metastrings are not case-sensitive, the actual ANSI commands are. In the above example the e in $e could be upper or lowercase, but the ANSI J command must have a capital J.

In this example, the ANSI escape sequence is issued with every new DOS prompt. There are times when it's unnecessary to continually send the same

```
ANSI_INT_21     ENDP

;-----------------------------------------------------------------;
; If we get here via a DOS console input call, any awaiting key   ;
; in keyboard buffer is checked to see if it is to be reassigned. ;
;-----------------------------------------------------------------;
ANSI_INT_16     PROC    FAR
                STI                             ;Interrupts back on.
                PUSHF                           ;Preserve flags on stack.
                PUSH    BP
                PUSH    DS
                PUSH    AX
                PUSH    CS                      ;Point to our data.
                POP     DS

                CMP     DOS_IMPUT,OFF           ;If not called via a DOS
                JZ      QUICK16_EXIT            ; console input call, exit.
                CMP     BUSY_FLAG,ON            ;If already processing a call,
                JZ      QUICK16_EXIT            ; exit.
                CLD                             ;All string operations forward.
                MOV     BP,SP                   ;Base reference to flags on stack
                MOV     FUNCTION_16,AH          ;Save function request.
                AND     AH,NOT 10H              ;Strip possible extended request.
                JZ      GET_KEY                 ;If zero, it's wait for a key.
                DEC     AH                      ;Else, decrement.  If zero it's
                JZ      KEY_STATUS              ; key status, else exit.

QUICK16_EXIT:   POP     AX                      ;Restore registers and lea
                POP     DS                      ; original interrupt handler
                POP     BP                      ; process the call.
                POPF
                JMP     QWORD PTR CS:OLD_INT_16

KEY_STATUS:     OR      BYTE PTR [BP+6],ZR      ;Assume none ready; ZR = 1.
GET_KEY:        MOV     BUSY_FLAG,ON            ;Non-reentrant; flag busy.
                POP     AX
                PUSH    BX                      ;Save some more registers.
                PUSH    CX
                PUSH    DX
                PUSH    SI
                PUSH    DI
                CMP     REASSIGN_FLAG,ON        ;Already in process of reassign?
                JZ      CK_BUFFER               ;If yes, check kbd buffer.

                PUSHF                           ;Else, emulate an interrupt.
                CALL    DWORD PTR OLD_INT_16
                PUSHF                           ;Status call results in flags.
                TEST    FUNCTION_16,1           ;Was it a status call?
                JZ      CK_DUP                  ;If no, check key returned.
                POPF                            ;Else, retrieve status results.
                JZ      INT16_EXIT              ;If zero, no key waiting.
                AND     BYTE PTR [BP+6],NOT ZR  ;Else, ZR = 0.
                PUSHF                           ;PUSHF just to keep stack right.

CK_DUP:         POPF                            ;Fix stack.
                MOV     BX,AX                   ;Match procedure wants key in BX.
                CALL    CK_MATCH                ;Check reassignment buffer.
                JC      INT16_EXIT              ;If no match, exit.
                MOV     AX,[DI]                 ;Else, retrieve string length.
                SUB     AX,3                    ;Adjust for length word and match
                ADD     DI,3                    ; byte; same for string pointer.
                OR      BL,BL                   ;Extended key? ie. key = 0.
                JNZ     STORE_COUNT             ;If no, save length and pointer.
                DEC     AX                      ;Else, adjust for extended code.
                INC     DI
STORE_COUNT:    MOV     REASSIGN_COUNT,AX       ;Save string length.
                MOV     REASSIGN_POS,DI         ;Save pointer to string.
                MOV     REASSIGN_FLAG,ON        ;Flag that replacement in effect.
CK_BUFFER:      TEST    FUNCTION_16,1           ;Was it a key wait function call?
                JNZ     RETRIEVE                ;If no, key status; get it.
                CMP     REMOVE_FLAG,OFF         ;Removed it from kbd buffer?
                JZ      RETRIEVE                ;If yes, get reassignment.
                MOV     AH,FUNCTION_16          ;Else, eat the replaced character
                INT     16H                     ; via INT 16.
                MOV     REMOVE_FLAG,OFF         ;Flag character eaten.

RETRIEVE:       MOV     DI,REASSIGN_POS         ;Retrieve pointer to string.
                MOV     AX,[DI]                 ;Retrieve replacement character.
                TEST    FUNCTION_16,1           ;Is it a key wait function call?
                JZ      REMOVE                  ;If yes, bump pointer up one.
                AND     BYTE PTR [BP+6],NOT ZR  ;If no, status call; ZR = 0
                JMP     SHORT INT16_EXIT        ;All done.

REMOVE:         INC     REASSIGN_POS            ;Move pointer to next character.
                DEC     REASSIGN_COUNT          ;One less character to replace.
                JZ      REASSIGN_DONE           ;If end of string, reset flags.
                OR      AL,AL                   ;Else, extended replacement?
                JNZ     INT16_EXIT              ;If no, done here.
                INC     REASSIGN_POS            ;Else adjust pointers.
                DEC     REASSIGN_COUNT
                JNZ     INT16_EXIT              ;End of string?
REASSIGN_DONE:  MOV     REASSIGN_FLAG,OFF       ;If yes, reset flags.
                MOV     REMOVE_FLAG,ON

INT16_EXIT:     MOV     BUSY_FLAG,OFF           ;Reset the busy flag.
                POP     DI                      ;Restore registers.
                POP     SI
                POP     DX
                POP     CX
                POP     BX
                POP     DS
                POP     BP
                POPF                            ;Flags on stack have status
                RET     2                       ; results; kill old flags.
ANSI_INT_16     ENDP

;************* ANSI ESCAPE STATE ROUTINES ************ ;

ESC_STATE:      MOV     BX,OFFSET BRACKET_STATE ;Assume Esc character.
                CMP     AL,ESC_CHAR             ;Is it Esc?  If yes, store
                JZ      SHORT JUMP              ; char. and next state.
                JMP     WRITE_CHAR              ;Else, normal char; write it.

;-----------------------------------------------;
BRACKET_STATE:  MOV     BX,OFFSET SPECIAL_STATE ;Assume left bracket.
                CMP     AL,"["                  ;Is it left bracket?  If yes,
SHORT_JUMP:     JZ      STORE_STATE             ;store char. and next state.
                JMP     FLUSH_BUFFER            ;Else, flush Esc out of buffer.

;-----------------------------------------------------------------;
; Must process <ESC>[2J (CLS) regardless if ANSI.COM ON or OFF. ;
;-----------------------------------------------------------------;
SPECIAL_STATE:  MOV     BX,OFFSET CLS_STATE     ;Assume Erase in Display code.
                CMP     AL,"2"                  ;Is it the "2" ?
                JZ      STORE_STATE             ;If yes, progress to next state.
                TEST    STATUS,OFF              ;Else, is ANSI OFF ?
                JNZ     FLUSH_BUFFER            ;If yes, flush Esc, bracket.
                MOV     BX,OFFSET PARAM_STATE   ;Else, check for first Set,
                CMP     AL,"="                  ; Reset Mode characters of
                JZ      STORE_STATE             ; "=" and "?".
                CMP     AL,"?"
                JZ      STORE_STATE             ;If found, store.
                JMP     SHORT DO_PARAMETER      ;Else, check other codes.

;-----------------------------------------------;
CLS_STATE:      CMP     AL,"J"                  ;Is it second character of CLS?
                MOV     DI,OFFSET COMMAND_END   ;If yes, execute
                JZ      EXECUTE                 ; Erase in Display procedure.
                TEST    STATUS,OFF              ;ANSI OFF?  If yes, flush buffer.
                JNZ     FLUSH_BUFFER            ; If not fall through to process.
                MOV     BYTE PTR NUMBER_BUFFER,2   ;Store the previous 2.
DO_PARAMETER:   MOV     ANSI_STATE,OFFSET PARAM_STATE  ;Parameter state.

;-----------------------------------------------;
PARAM_STATE:    CALL    CK_QUOTE                ;Is it single or double quotes?
                JZ      STORE_STATE             ;If yes, store string state.
                CMP     AL,";"                  ;Is it semi-colon delimiter?
                JNZ     CK_NUMBER               ;If no, check if number.
                INC     NUMBER_COUNT            ;Else, increment number count.
                JMP     SHORT BUFFER_CHAR       ;Buffer the semi-colon.

CK_NUMBER:      CMP     AL,"0"                  ;Is it below 0 ?
                JB      FLUSH_BUFFER            ;If yes, illegal; flush buffer.
                CMP     AL,"9"                  ;Else, is it above 9 ?
                JA      DO_COMMAND              ;If yes, check for command char.
                CALL    ACCUMULATE              ;Else it's a number; accumulate.
                JMP     SHORT BUFFER_CHAR       ;Buffer the character.

DO_COMMAND:     MOV     DI,OFFSET COMMAND_TABLE ;Point to legal ANSI commands.
```

continued . . .

escape sequence. Changing the ANSI display attribute is one example of this, since you want the attribute to stay in effect until it is changed. To accomplish this, TYPEing a file that consists of an ANSI command to the console is a better idea. For example, to change the screen output to white characters on a blue background, you create a short file that consists of the sequence

ESC[37;34m

Again, the ESC must be the escape character; but how do you get an ESC character into a text file?

In most word processors, pressing the ESC key alone won't do. Most word processors *will* let you enter control characters, but you must first signal your intentions to the program. In WordStar and SideKick, for example, you first hold down the Ctrl key, press P and then press the Esc key. A ^[will appear on the screen. You must now add ANSI's 'regular' left bracket and the command parameters. For the white on blue attribute above, the complete sequence appears as **^[[37;34m**. Don't be confused by the pair of left brackets. The ESC represented is characterised by ^[, and the second character in the escape sequence is a left bracket. Once you've entered the sequence, save it to a file named BLUE, for example, and enter the command TYPE BLUE. Nothing from the file will appear on the screen. ANSI has absorbed the string as a command, however, and all subsequent screen output, including the next prompt, will be white on blue. Enter CLS and the whole screen will clear to blue.

If you have trouble getting your favourite word processor to accept the ESC character, everyone has at least one program that will do the job: EDLIN. To create the same white-on-blue file, start EDLIN by entering **EDLIN BLUE**. Then at the asterisk prompt press I to insert a line. To enter the starting ESC in EDLIN, you press Ctrl-V and then the left bracket, [. Then enter the 'normal' left

```
A
MOV     SI,0080
MOV     CL,[SI]
XOR     CH,CH
INC     CX
MOV     Word Ptr [SI],5B1B
LODSB
INT     29
LOOP    010C
RET

N ESCAPE.COM
RCX
12
W
Q
```

Fig 1 This short program solves the problem of entering escape sequences at the DOS prompt. After DEBUG is loaded, these commands send the ESC, bracket and command string to ANSI

```
                MOV     CX,COMMAND_LENGTH       ;Number of commands.
                REPNZ   SCASB                   ;Check for a match.
                JNZ     FLUSH_BUFFER            ;If no match, flush sequence.
                INC     NUMBER_COUNT            ;Else, increment for last number.
                MOV     DI,OFFSET COMMAND_END   ;Point to appropriate command
                SHL     CX,1                    ; processing procedure.
                SUB     DI,CX
EXECUTE:        CALL    GET_BIOS_DATA           ;Get cursor position data.
                CALL    DS:[DI]                 ;Do command subroutine.
                JMP     SHORT FLUSH_END         ;Clear buffer.

;-----------------------------------------------;
QUOTE_STATE:    CMP     AL,QUOTE_TYPE           ;Is it an ending string quote?
                JNZ     BUFFER_CHAR             ;If no, buffer literal.
                MOV     BX,OFFSET PARAM_STATE   ;Else, back to parameter parsing.

;-----------------------------------------------;
STORE_STATE:    MOV     ANSI_STATE,BX           ;Store the ANSI state.

;-----------------------------------------------;
BUFFER_CHAR:    MOV     DI,ESC_COUNT            ;Buffer the character in case
                CMP     DI,ESC_BUFFER_SIZE      ; ending ANSI command is illegal.
                JZ      FLUSH_BUFFER            ;If buffer full, flush.
                ADD     DI,OFFSET ESC_BUFFER    ;Point to next buffer position.
                STOSB                           ;Store the character.
                INC     ESC_COUNT               ;Increment the sequence count.
                RET

;-----------------------------------------------;
FLUSH_BUFFER:   PUSH    AX                      ;Save the current character.
                MOV     SI,OFFSET ESC_BUFFER    ;Point to the sequence buffer.
                MOV     CX,ESC_COUNT            ;Count of buffered characters.
NEXT_FLUSH:     LODSB                           ;Retrieve one.
                PUSH    CX                      ;Save counter and pointer.
                PUSH    SI
                CALL    WRITE_CHAR              ;Write character to screen.
                POP     SI                      ;Restore counter and pointer.
                POP     CX
                LOOP    NEXT_FLUSH              ;Flush entire buffer.
                POP     AX                      ;Retrieve last character.
                CALL    WRITE_CHAR              ;Write it also.
FLUSH_END:      MOV     ESC_COUNT,0                     ;Reset counter.
                MOV     ANSI_STATE,OFFSET ESC_STATE     ;Back to Esc state.
                MOV     NUMBER_COUNT,0                  ;Reset parameter counter.
                PUSH    CS                              ;Point to our data.
                POP     ES
                MOV     DI,OFFSET NUMBER_BUFFER         ;Clear number buffer
                MOV     CX,ESC_BUFFER_SIZE / 2          ; to zeros.
                XOR     AX,AX
                REP     STOSW
                RET

;-----------------------------------------------;
; Slow video writes are via BIOS Write TTY.     ;
;-----------------------------------------------;
WRITE_CHAR:     MOV     DI,OFFSET FORMAT_CHARS  ;Let BIOS process carriage
                MOV     CX,FORMAT_LENGTH        ; return, linefeed, backspace
                REPNZ   SCASB                   ; and bell via
                JZ      WRITE_TTY               ; TTY.

                CALL    GET_BIOS_DATA           ;Get BIOS video data.
                CMP     AL,9                    ;Is character a TAB?
                JNZ     CK_ACTIVE               ;If no, process normally.
                MOV     CX,CURSOR_POSN          ;Else, expand TAB to
                AND     CX,7                    ; appropriate space characters.
                NEG     CX
                ADD     CX,8
NEXT_TAB:       PUSH    CX
                MOV     AL,SPACE
                CALL    CK_ACTIVE
                POP     CX
                LOOP    NEXT_TAB
                RET

CK_ACTIVE:      TEST    STATUS,OFF              ;Is ANSI OFF?
                JNZ     WRITE_TTY               ;If yes, write via BIOS TTY.
CK_FAST:        CALL    CK_SLOW_TEXT            ;Is ANSI SLOW or in graphics
                JNC     WRITE_FAST              ; mode? If no, write fast.

WRITE_SLOW:     PUSH    AX                      ;Else, write character/attribute
                MOV     CX,1                    ; at current cursor position
                MOV     BH,ACTIVE_PAGE          ; via BIOS.
                MOV     BL,ATTRIBUTE
                MOV     AH,9
                INT     10H
                POP     AX

                CMP     LINE_WRAP,ON            ;Is line wrap on?
                JZ      TTY                     ;If yes, continue.
                MOV     CX,CRT_COLS             ;Else, cursor at rightmost
                DEC     CL                      ; column?
                CMP     CL,CURSOR_COL           ;If yes, continue, else
                JNZ     TTY                     ; return without writing.
                RET

;-----------------------------------------------;
WRITE_TTY:      MOV     BL,7                    ;Attribute in graphics mode.
TTY:            MOV     AH,0EH
                INT     10H
                RET

;-----------------------------------------------------------------------;
; Fast screen writes are directly to the video buffer without retrace check. ;
;-----------------------------------------------------------------------;
WRITE_FAST:     PUSH    ES                      ;Preserve extra segment.
                MOV     DX,CURSOR_POSN          ;Retrieve cursor position.
                CALL    VIDEO_SETUP             ;Calculate video address.
                MOV     AH,ATTRIBUTE            ;Retrieve attribute.
                STOSW                           ;Put char/attrib in video buffer.
                INC     DL                      ;Increment cursor column.
                CMP     DL,BYTE PTR CRT_COLS    ;End of row?
                JB      UPDATE_CURSOR           ;If no, update cursor.

                CMP     LINE_WRAP,OFF           ;Else, line wrap off?
                JZ      FAST_END                ;If yes, don't move cursor.
                XOR     DL,DL                   ;Else, column zero.
                INC     DH                      ;Next row.
                CALL    INFORMATION             ;Get displayable row info.
                CMP     DH,AL                   ;Beyond the bottom of screen?

                JBE     UPDATE_CURSOR           ;If no, update cursor.

                DEC     DH                      ;Else, cursor to original row.
                MOV     DI,CRT_START            ;Point destination to top.
                MOV     SI,DI                   ;Point source to second row
                MOV     AX,CRT_COLS             ; by adding width in columns
                PUSH    AX                      ; twice for char/attribute.
                ADD     SI,AX
                ADD     SI,AX
                MUL     DH                      ;Times displayable rows - 1.
                MOV     CX,AX                   ; equals char/attrib to scroll.
                PUSH    DS                      ;Save data segment and
                PUSH    ES                      ; point to video segment.
                POP     DS
                REP     MOVSW                   ;Scroll the screen.
                POP     DS                      ;Restore data segment.
                POP     CX                      ;Retrieve CRT columns.
                MOV     AL,SPACE                ;Write space/attrib to
                MOV     AH,ATTRIBUTE            ; bottom row.
                REP     STOSW

UPDATE_CURSOR:  CALL    SET_CURSOR              ;Update the cursor position.
FAST_END:       POP     ES                      ;Restore extra segment.
                RET

;************* SUPPORT ROUTINES *************;

CK_QUOTE:       MOV     BX,OFFSET QUOTE_STATE   ;Assume quote state.
                MOV     AH,DOUBLE_QUOTE
                CMP     AL,AH                   ;Is it double quote?
                JZ      GOT_QUOTE               ;If yes, string delimiter.
                MOV     AH,SINGLE_QUOTE         ;Is it single quote?
                CMP     AL,AH                   ;Is yes, string delimiter.
                JNZ     QUOTE_END               ;Else, return ZR = 0.
GOT_QUOTE:      MOV     QUOTE_TYPE,AH           ;Store as matching string end.
QUOTE_END:      RET
```

continued . . .

The IBM ANSI command set

The ANSI.SYS device driver is mentioned in the DEVICE command section of the regular DOS 'Reference' manual. The command set is not included there, however, and is found instead in the 'Technical Reference' manual (called the 'Command Reference' for DOS 4.0). Since many readers do not have the 'Technical Reference' manual, the ANSI command set is listed here.

Control sequence syntax

The control sequences have the following general form:

ESC[*parameters*COMMAND

The ESC is not the three-character sequence E-S-C but rather the 1-byte ASCII code for decimal 27 (hex 1B). The left bracket is the second control character and is followed by the parameters for the specific ANSI command. The parameters in the IBM ANSI-supported commands listed below are either a decimal number, represented by #, or a literal string, designated by beginning and ending matching single or double quotation marks. To include a quote within a string, use the other kind of quote as the delimiter. For example, "This ' is a single quote" or 'This " is a double quote'.

A semicolon is used to separate multiple parameters. If a parameter value is omitted or specified as 0, then the default parameter is used for the command. The COMMAND is the last character(s) in the sequence. In all but a few cases the COMMAND is constituted by a single, case-sensitive alpha character.

Cursor control sequences

In all cursor-control commands, the parameters are checked to keep the cursor within the boundaries of the screen. That is, you cannot move the cursor off-screen. The default value of 1 is used for any omitted parameters.

Cursor Position

ESC[#;#H

or the alternative sequence

ESC[#;#f

Either of these sequences moves the cursor to the row that is specified by the first parameter and to the column that is specified by the second. One or both of the parameters may be omitted. For example, **ESC[;5H** is equivalent to **ESC[1;5H**, and the cursor would be moved to the first row, column 5. If both parameters are omitted, then the cursor is moved to the home position.

Cursor up

ESC[#A

The sequence above moves the cursor up # rows without changing its column position.

Cursor down

ESC[#B

This sequence moves the cursor down by # rows without changing its column position.

Cursor forward

ESC[#C

This sequence moves the cursor forward # columns without changing its row.

Cursor backward

ESC[#D

The above sequence moves the cursor backward by # columns without changing its row.

Cursor position report

ESC[#;#R

The sequence above reports the row and column of the cursor through the standard input device. The **ESC[6n** sequence (Device Status Report) outputs the position report. Most applications will find the Device Status Report of no use to them. The DOS prompt will become uncontrolled if you use it as part of a PROMPT command.

Save cursor position

ESC[s

This sequence saves the current cursor position so that it can be restored later.

SGR parameters

Parameter	Attribute
0	All attributes off (normal white on black)
1	Bold on (high intensity)
4	Underline for monochrome display; blue foreground for colour
5	Blink on
7	Reverse video on
8	Cancelled on (invisible)
30	Black foreground
31	Red foreground
32	Green foreground
33	Yellow foreground
34	Underline for monochrome display; blue foreground for colour
35	Magenta foreground
36	Cyan foreground
37	White foreground
40	Black background
41	Red background
42	Green background
43	Yellow background
44	Blue background
45	Magenta background
46	Cyan background
47	White background

Fig A The colour and attribute parameters of the IBM ANSI Set Graphics Rendition control sequence

bracket, again followed by the desired command parameters. The EDLIN screen display in this case will be

```
1:* ^V[[37;34m
```

Now hit Enter to insert the line, Ctrl-Break to stop inserting, and E to end and save the editing session. The result will be the BLUE file with the escape sequence. (Note, do not try to enter the ESC code by typing the caret character followed by a capital V. It won't work. Instead, press and hold the Ctrl key and then press V.)

Still another way to enter escape sequences — and one of the easiest when you're experimenting with escape commands — is to enter them directly at the DOS prompt; however, it's almost impossible to do. If you try to start an escape sequence by pressing the ESC key at

Restore cursor position

`ESC[u`

This sequence restores the cursor to the position saved with the last 'save cursor position' sequence.

Erasing control sequences

Erase in display or CLS

`ESC[2J`

This sequence clears the screen with the current SGR (Set Graphics Rendition) attribute (see below) and moves the cursor to the home (top-left) position.

Erase in line

`ESC[K`

This sequence shown above erases the line containing the cursor from (and including) the current cursor position. It uses the SGR attribute from the current cursor position.

Set graphics rendition

`ESC[#;...;#m`

This sequence selects the attribute used to display characters. Attributes have a cumulative effect. For example, `ESC[31;5;43m` would make the attribute red on yellow and blink.

To undo this, you first change the attribute back to normal and then reset the colour. For example, `ESC[0;34;47m` would first change any predefined attribute to normal (effectively cancelling any bold or blink attribute) and change the display attribute to blue on white. A listing of the SGR parameters and their definitions are shown in Fig A, 'SGR Parameters'.

Set mode

`ESC[=#h`

This sequence changes the video mode to #, where # has one of the values shown in Fig B, 'Video modes'. Note: the mode options numbered 14 through 19 were added to the IBM subset with DOS 4.0. (ANSI.COM supports the added modes with any DOS version if your system supports them.) Chances are that you won't find much use for this or the following Reset Mode command.

Video modes

Parameter	Video mode or action
0	40x25 black and white
1	40x25 colour
2	80x25 black and white
3	80x25 colour
4	320x200 colour
5	320x200 black and white
6	640x200 black and white
7	Wrap at end of line (typing past end of line wraps to new line)
14	640x200 colour EGA
15	640x350 mono EGA
16	640x350 colour EGA
17	640x480 colour VGA
18	640x480 colour VGA
19	320x200 colour VGA

Fig B The video modes supported by the IBM ANSI Set Mode control sequence

Reset mode

`ESC[=#l`

Note that the command character is a lowercase l, not the numeral 1. All the parameters are the same as those shown in the 'Video mode' table, except that a parameter 7 turns line wrap off (the cursor does not wrap to the next line).

Key reassignment

The sequences used to reassign the keyboard keys is one of the more useful functions of the ANSI expanded control system. They are as follows:

```
ESC[#;#...#p
or ESC[''string''p
or ESC['string'p
or ESC[#;''string'';#;#;
''string'';#p
or any combination of the above
```

The first # is the ASCII character code for the key to be reassigned. It's followed by a single or string of codes that are to replace it. If the first parameter is 0, then the first and second parameters make up an extended ASCII code, and the third parameter (instead of the second) becomes the first reassignment ASCII character. For example,

`ESC[0;68;"Dir";13p`

causes a directory listing to be produced when the F10 (extended code 0;68) is pressed. Replacement characters can also have extended codes. Fig C, 'Extended ASCII codes', lists these codes.

Extended ASCII codes

Extended ASCII code	Meaning or key(s)
3	NUL (null character)
15	Shift-Tab
16-25	Alt-Q, -W, -E, -R, -T, -Y, -U, -I, -O, and -P
30-38	Alt-Z, -X, -C, -V, -B, -M, and -N
59-68	F1 through F10
71	Home
72	Cursor up
73	PgUp
75	Cursor left
77	Cursor right
79	End
80	Cursor down
81	PgDn
82	Ins
83	Del
84-93	F11 through F20 (Shift-F1 through F10)
94-103	F20 through F30 (Ctrl-F1 through F10)

Fig C The extended ASCII codes used in the 2-byte ANSI key redefinitions

the DOS prompt, DOS interprets the keypress as a command to abort the current process and spits out a confusing backslash character, \. You never get the chance to add the [that starts the ANSI sequence. Trying to trick DOS by using the numeric keypad technique of holding down the Alt key and entering 27 results in the same DOS abort reaction.

To solve the problem, I've included a short program here that will send both the ESC, the bracket and then the command string to ANSI. To create the program, you must load DEBUG and enter the commands that are shown in Fig 1.

With this short program, you can change the attribute to white on blue, for example, by entering `ESCAPE 37;34m`. The ESCAPE.COM program will take care of the ESC-plus-bracket combina-

```
;-----------------------------------------------;
ACCUMULATE:     PUSH    AX                      ;Preserve number character.
                SUB     AL,"0"                  ;Convert ASCII to binary.
                MOV     CL,AL                   ;Save the number.
                MOV     AX,10                   ;Multiply previous count by 10.
                MOV     BX,NUMBER_COUNT
                MUL     BYTE PTR NUMBER_BUFFER[BX]
                ADD     AL,CL                           ;Add in new number
                MOV     BYTE PTR NUMBER_BUFFER[BX],AL   ; and store.
                POP     AX
                RET

;-----------------------------------------------------------------------;
; OUTPUT:  CY = 1 if write SLOW mode or in graphics mode; CY = 0 otherwise. ;
;-----------------------------------------------------------------------;
CK_SLOW_TEXT:   TEST    STATUS,SLOW
                JNZ     SLOW_MODE
                CMP     CRT_MODE,7
                JZ      TEXT_MODE
                CMP     CRT_MODE,3
                JA      SLOW_MODE
TEXT_MODE:      CLC
                RET

SLOW_MODE:      STC
                RET

;-----------------------------------;
; OUTPUT:  AL = Screen rows minus one ;
;-----------------------------------;
INFORMATION:    PUSH    DS                      ;Save data segment.
                MOV     AX,40                   ;Point to BIOS data.
                MOV     DS,AX
                MOV     AL,DS:[84H]             ;Retrieve rows - 1.
                OR      AL,AL                   ;BIOS supported?
                JNZ     INFO_END                ;If yes, done here.
                MOV     AL,24                   ;Else, assume 25 lines.
INFO_END:       POP     DS
                RET

;-----------------------------------------------------------------------;
; INPUT:  DX = Cursor position.                                         ;
; OUTPUT: ES = Video buffer segment; DI - Video buffer offset; AX,DX preserved ;
;-----------------------------------------------------------------------;
VIDEO_SETUP:    PUSH    AX
                MOV     AX,CRT_COLS             ;Retrieve CRT columns.
                MUL     DH                      ;Times cursor row.
                MOV     BL,DL
                XOR     BH,BH
                ADD     AX,BX                   ;Plus cursor column.
                SHL     AX,1                    ;Times two for attribute.
                MOV     DI,CRT_START            ;Plus starting video offset.
                ADD     DI,AX                   ;Equals destination.

                MOV     BX,0B000H               ;Assume mono card.
                CMP     ADDR_6845,3B4H          ;Is it mono port?
                JZ      VIDEO_SEGMENT           ;If yes, guessed right.
                ADD     BX,800H                 ;Else, point to color segment.
VIDEO_SEGMENT:  MOV     ES,BX
                POP     AX
                RET

;-----------------------------------------------;
; Move BIOS video data into our data segment.   ;
;-----------------------------------------------;
GET_BIOS_DATA:  PUSH    DS
                PUSH    DI                      ;Point to BIOS data segment.
                MOV     BX,40H
                MOV     DS,BX
                MOV     SI,BIOS_ACTIVE_PAGE     ;Start with active page.
                MOV     DI,OFFSET ACTIVE_PAGE
                MOVSB                           ;Retrieve active page
                MOVSW                           ; and address of 6845 port.
                MOV     SI,BIOS_CRT_MODE        ;Retrieve CRT mode, CRT columns,
                MOV     CX,CRT_DATA_LENGTH      ; CRT length, CRT start.
                REP     MOVSB
                MOV     BL,ES:ACTIVE_PAGE       ;Use active page as index
                XOR     BH,BH                   ; of active cursor position.
                SHL     BX,1
                ADD     SI,BX
                MOVSW
                POP     DI
                POP     DS
                RET

;-----------------------------------------------------------------------;
; OUTPUT: BL = First parameter; BH = Second parameter; DX = cursor position. ;
;-----------------------------------------------------------------------;
ADJUST_NUMBER:  MOV     BX,WORD PTR NUMBER_BUFFER
                OR      BH,BH
                JNZ     ADJUST_AL               ;If either parameter zero,
                INC     BH                      ; increment to convert
ADJUST_AL:      OR      BL,BL                   ; to base one.
                JNZ     ADJUST_END
                INC     BL
ADJUST_END:     MOV     DX,CURSOR_POSN          ;Return cursor position.
                RET

;-----------------------------------------------------------------------;
; INPUT:  AX = number; BP = 0 if console output; BP = 1 if storage output. ;
;-----------------------------------------------------------------------;
DECIMAL_OUT:    MOV     BX,10                   ;Divisor of ten.
                XOR     CX,CX                   ;Zero in counter.
NEXT_COUNT:     XOR     DX,DX                   ;Zero in high half.
                DIV     BX                      ;Divide by ten.
                ADD     DL,"0"                  ;Convert to ASCII.
                PUSH    DX                      ;Save results.
                INC     CX                      ;Also increment count.
                CMP     AX,0                    ;Are we done?
                JNZ     NEXT_COUNT              ;Continue until zero.
                OR      BP,BP                   ;If BP zero, output to screen.
                JNZ     DEVICE_OUTPUT           ;Else, store in buffer.

REPORT_OUTPUT:  POP     AX                      ;Retrieve number.
                CALL    PRINT_CHAR              ;Display it.
                LOOP    REPORT_OUTPUT
                RET

DEVICE_OUTPUT:  POP     AX                      ;Retrieve number.
                STOSB                           ;Store it
                LOOP    DEVICE_OUTPUT
                RET

;************* ANSI COMMAND SUBSET *************;

CLS:            CALL    INFORMATION             ;Get displayable rows.
                MOV     DH,AL                   ;Store in DH.
                MOV     BH,7                    ;Assume normal attribute.
                TEST    STATUS,OFF              ;Is ANSI OFF?
                JNZ     CLS_SLOW                ;If yes, CLS via BIOS.
                MOV     BH,ATTRIBUTE            ;Else, requested attribute.
                CALL    CK_SLOW_TEXT
                JC      CLS_SLOW                ;If AHSI SLOW, via BIOS.

CLS_FAST:       MOV     AH,BH                   ;Else, attribute in high half.
                MOV     AL,SPACE                ;Space character in low half.
                XOR     DX,DX                   ;Cursor poeition home.
                CALL    VIDEO_SETUP             ;Calculate video address.
                MOV     CX,CRT_LEN              ;Retrieve CRT length.
                SHR     CX,1                    ;Times two for attribute.
                REP     STOSW                   ;Write directly to video buffer.
                JMP     SHORT SET_CURSOR        ;Set the cursor top left corner.

CLS_SLOW:       MOV     DL,BYTE PTR CRT_COLS    ;Retrieve CRT columns.
                DEC     DL                      ;Adjust to zero base.
                XOR     CX,CX                   ;Scroll active page.
                MOV     AX,600H
                INT     10H
                XOR     DX,DX                   ;Home the cursor.
                JMP     SHORT SET_CURSOR

;-----------------------------------------------;
CURS_POSITION:                                  ;These two commands are the same.
HORZ_VERT_POS:  CALL    INFORMATION             ;Returns AL = screen rows.
                CALL    ADJUST_NUMBER           ;Returns BL,BH = parameters.
                SUB     BX,101H                 ;Zero based.
                CMP     BL,AL                   ;If cursor request within
                JA      CURSOR_END              ; boundaries of screen, set it.
                CMP     BH,BYTE PTR CRT_COLS
                JAE     CURSOR_END
                XCHG    BH,BL
                MOV     DX,BX

SET_CURSOR:     MOV     BH,ACTIVE_PAGE          ;Set cursor via BIOS.
                MOV     AH,2
                INT     10H
CURSOR_END:     RET

;-----------------------------------------------;
CURSOR_UP:      CALL    ADJUST_NUMBER           ;Move cursor up one or more rows
                OR      DH,DH                   ; without changing column.
                JZ      CURSOR_END              ;If already at top, ignore.
                SUB     DH,BL
                JNC     SET_CURSOR              ;Stay in bounds.
                XOR     DH,DH
                JMP     SHORT SET_CURSOR

;-----------------------------------------------;
CURSOR_DOWN:    CALL    INFORMATION             ;Returns AL = screen rows.
                CALL    ADJUST_NUMBER           ;Returns BL,BH =nos.; DX = cursor
                CMP     DH,AL                   ;Move cursor down requested
                JZ      CURSOR_END              ; rows without changing column.
                ADD     DH,BL
                CMP     DH,AL
                JNA     SET_CURSOR
                MOV     DH,AL                   ;Stay in bounds.
                JMP     SHORT SET_CURSOR

;-----------------------------------------------;
CURS_FORWARD:   CALL    ADJUST_NUMBER           ;Returns BL,BH =nos.; DX = cursor
                MOV     BH,BYTE PTR CRT_COLS    ;Retrieve displayable columns.
                DEC     BH                      ;Adjust to zero base.
                CMP     DL,BH                   ;Move cursor forward one or more
                JZ      CURSOR_END              ; columns without changing row.
                ADD     DL,BL
                CMP     DL,BH
                JNA     SET_CURSOR
                MOV     DL,BH                   ;Stay in bounds.
                JMP     SHORT SET_CURSOR
;-----------------------------------------------;
CURS_BACKWARD:  CALL    ADJUST_NUMBER           >Returns BL,BH =nos.; DX = cursor
                OR      DL,DL                   ;Move cursor backward one or
                JZ      CURSOR_END              ; more columns without changing
                SUB     DL,BL                   ; row.  Ignore if already left.
                JNC     SET_CURSOR
                XOR     DL,DL                   ;Stay in bounds.
                JMP     SHORT SET_CURSOR

;-----------------------------------------------;
; Report cursor position via console; Format: ESC[#;#R   ;
;-----------------------------------------------;
DEVICE_STATUS:  MOV     DI,OFFSET DEVICE_STATUS_BUFFER
                MOV     AL,ESC_CHAR
                STOSB                           ;Store leading Esc, bracket
                MOV     AL,"["                  ;in special Device Status Buffer.
                STOSB

                MOV     AL,CURSOR_ROW           ;Retrieve cursor row.
                XOR     AH,AH                   ;Zero in high half.
                INC     AX                      ;Convert to base one.
                MOV     BP,1                    ;Flag as storage.
                CALL    DECIMAL_OUT             ;Convart to ASCII.
                MOV     AL,";"                  ;Store delimiting semi-colon.
                STOSB
                MOV     AL,CURSOR_COL           ;Do same with cursor column.
                XOR     AH,AH
                INC     AX
                CALL    DECIMAL_OUT
                MOV     AL,"R"                  ;Add command character
                STOSB
                MOV     AL,CR                   ; and carriage return.
                STOSB
                SUB     DI,OFFSET DEVICE_STATUS_BUFFER   ;Setup for console out,
                MOV     REASSIGN_COUNT,DI
                MOV     REASSIGN_POS,OFFSET DEVICE_STATUS_BUFFER
                MOV     REASSIGN_FLAG,ON
                MOV     REMOVE_FLAG,OFF
                RET

;-----------------------------------------------;
SAVE_CURSOR:    MOV     DX,CURSOR_POSN
                MOV     SAVE_POSITION,DX
                RET

;-----------------------------------------------;
RESTORE_CURS:   MOV     DX,SAVE_POSITION
                JMP     SET_CURSOR

;-----------------------------------------------;
ERASE_IN_LINE:  MOV     DX,CURSOR_POSN          ;Erase from the cursor to
                MOV     CX,CRT_COLS             ; the end of the line, including
                SUB     CL,DL                   ; the current cursor position.
                CALL    CK_SLOW_TEXT
                JC      ERASE_SLOW              ;If ANSI ON and not graphice,
                CALL    VIDEO_SETUP             ; write directly to video buffer
                MOV     AL,SPACE                ; with space/attribute.
                MOV     AH,ATTRIBUTE
                REP     STOSW
                RET

ERASE_SLOW:     MOV     CX,DX                   ;Else, erase SLOW via
                MOV     DL,BYTE PTR CRT_COLS    ; BIOS scroll active page.
                DEC     DL
                MOV     BH,ATTRIBUTE
                MOV     AX,601H
                INT     10H
                RET
```

continued . . .

```
;-----------------------------------------------;
;            Set Graphics Rendition             ;
;-----------------------------------------------;
SGR:            MOV     SI,OFFSET NUMBER_BUFFER     ;Point to number parameters.
NEXT_SGR:       LODSB                               ;Retrieve parameter.
                MOV     DI,OFFSET ATTRIBUTE_TABLE   ;Look up attribute in
                MOV     CX,ATTRIBUTE_LENGTH         ; translation table.
                REPNZ   SCASB
                JNZ     SGR_LOOP

                MOV     DI,OFFSET ATTRIBUTE_END
                SHL     CX,1
                SUB     DI,CX
                MOV     AX,[DI]
                AND     ATTRIBUTE,AL            ;If match, AND with strip mask.
                OR      ATTRIBUTE,AH            ;OR with add mask.
SGR_LOOP:       DEC     NUMBER_COUNT            ;Do all parameters.
                JNZ     NEXT_SGR
                RET
;-----------------------------------------------;
SET_MODE:       MOV     AH,ON                   ;Assume command 7,
                JMP     SHORT CK_WRAP           ; turn line wrap on.

;-----------------------------------------------;
RESET_MODE:     MOV     AH,OFF                      ;Assume turn line wrap off.
CK_WRAP:        MOV     AL,BYTE PTR NUMBER_BUFFER   ;Retrieve number parameter.
                CMP     AL,7                        ;Is it 7?
                JZ      SET_WRAP                ;If yes, set wrap mode.
                JB      DO_MODE                 ;1 - 6 valid modes.
                CMP     AL,14                   ;14 - 16 valid modes.
                JB      MODE_END
                CMP     AL,15
                JA      MODE_END                ;If above 16, illegal.
DO_MODE:        XOR     AH,AH                   ;Else, set video mode
                INT     10H                     ; to parameter.
                RET

SET_WRAP:       MOV     LINE_WRAP,AH
MODE_END:       RET

;-----------------------------------------------------------------------;
; Keyboard Key Reassignment:  First convert ASCII numbers to binary,    ;
; parse out delimiting semi-colons, and quote string delimiters.        ;
;-----------------------------------------------------------------------;
REASSIGNMENT:   MOV     CX,ESC_COUNT            ;Retrieve length of Esc sequence.
                DEC     CX                          ;Adjust.
                MOV     SI,OFFSET ESC_BUFFER + 2    ;Point past Esc, bracket.
                MOV     DI,OFFSET PARSE_BUFFER      ;Point to parse storage.

NEXT_STRING:    MOV     QUOTE_TYPE,0                ;Reset quote type.
NEXT_NUM:       XOR     DL,DL                   ;Use DL to carry number; init = 0
                XOR     BP,BP                   ;Use BP as number flag.
NEXT_PARAM:     LODSB                           ;Retrieve a byte.
                DEC     CX                      ;Decrement string length counter.
                JZ      PARSE_END               ;If zero, done.
                MOV     AH,QUOTE_TYPE           ;Else, retrieve quote type.
                OR      AH,AH                   ;If zero, not in string.
                JNZ     QUOTE_MODE              ;Else, go to string mode.
                CALL    CK_QUOTE                ;Is character a quote?
                JZ      NEXT_PARAM              ;If yes, ignore.
                CMP     AL,";"                  ;Else, is it semi-colon?
                JZ      STORE_NUMBER            ;If yes, number delimiter.

                SUB     AL,"0"                  ;Else, must be number; convert
                MOV     DH,AL                   ; to binary; save in DH.
                MOV     AX,10                   ;Multiply current total by ten.
                MUL     DL
                ADD     AL,DH                   ;Add new number.
                MOV     DL,AL                   ;Save in DL.
                MOV     BP,1                    ;Flag that number mode active.
                JMP     SHORT NEXT_PARAM        ;Next parameter.

STORE_NUMBER:   OR      BP,BP                   ;If number mode flag not set then
                JZ      NEXT_PARAM              ;semi-colon not prefaced with no.
                MOV     AL,DL                   ;Else, store number.
                STOSB
                JMP     SHORT NEXT_NUM          ;Reset number carrying registers.

QUOTE_MODE:     CMP     AL,AH                   ;Is current char a string ending
                JZ      NEXT_STRING             ;quote?  If yes, exit quote mode
                STOSB                           ;Else, store char as literal.
                JMP     SHORT NEXT_PARAM
;-----------------------------------------------------------------------;
; Remove duplicate assignments if removal leaves room for new assignment.   ;
; If no duplicate and room, add new assignment, else flush buffer to screen. ;
;-----------------------------------------------------------------------;
PARSE_END:      OR      BP,BP                   ;Was last parameter a number?
                JZ      CK_LENGTH               ;If no, skip.
                MOV     AL,DL                   ;Else, store the last number.
                STOSB
CK_LENGTH:      SUB     DI,OFFSET PARSE_BUFFER - 2  ;Calculate string length
                MOV     AX,DI                       ; including word for length.
                MOV     BX,WORD PTR PARSE_BUFFER    ;BL=new first ASCII; BH=2nd.
                MOV     CX,5                        ;String length has to be at
                OR      BL,BL                   ;least word + 2 for old + new = 5
                JZ      CK_LEGAL                ; for extended key reassignment.
                DEC     CX                      ;And word + 1 for old + new = 4
CK_LEGAL:       CMP     AX,CX                   ; for regular ASCII reassignment.
                JB      ASSIGN_FLUSH            ;If not, illegal; flush buffer.
                MOV     BP,BUFFER_END           ;BP to carry buffer end pointer.
                CALL    CK_MATCH                ;Is char already reassigned?
                JC      CK_ROOM                 ;If no, check room for new.

CK_REMOVE:      MOV     CX,DX                   ;REASSIGN_END - string end
                SUB     CX,SI                   ; = bytes to move.
                JZ      CK_ROOM                 ;If zero, last string.
                SUB     DX,[DI]                 ;Is REASSIGN_END - old string
                ADD     DX,AX                   ; + new string >
                CMP     DX,BP                   ; BUFFER_END?
                JA      ASSIGN_FLUSH            ;If yes, flush buffer to screen.
                REP     MOVSB                   ;Else, move them down in buffer.
                JMP     SHORT STORE_NEW

CK_ROOM:        ADD     DX,AX                   ;New string + current strings.
                CMP     DX,BP                   ;Greater than buffer size?
                JA      ASSIGN_FLUSH            ;If yes, flush new string.
STORE_NEW:      MOV     SI,OFFSET PARSE_BUFFER  ;Else, room for new string.
                STOSW                           ;Store string length first.
                MOV     CX,AX                   ;Adjust counter.
                DEC     CX
                DEC     CX
                REP     MOVSB                   ;Store the actual string.
                MOV     REASSIGN_END,DI         ;Store new string end.
                RET

ASSIGN_FLUSH:   MOV     AL,"p"                  ;If buffer full, flush string
                JMP     FLUSH_BUFFER            ; including ending "p" command.

;-----------------------------------------------------------------------;
; INPUT:  BL = first ASCII code to match; BH = extended if BL = 0       ;
; OUTPUT: CY = 1 if no match found; CY = 0 if match found.              ;
;         DI points to string length of match; DI+2 points to start of string. ;
;         SI = end of string; DX = REASSIGN_END; BP = BUFFER_END        ;
;-----------------------------------------------------------------------;
CK_MATCH:       MOV     DX,REASSIGN_END                 ;Current strings end.
                MOV     SI,OFFSET REASSIGNMENT_BUFFER   ;Point to buffer.
NEXT_MATCH:     MOV     DI,SI                           ;Current record.
                CMP     DI,DX                   ;End of strings?
                JAE     NO_MATCH                ;If yes, no match.
                MOV     CX,[DI+2]               ;CL=current first ASCII; CH=2nd
                ADD     SI,[DI]                 ;Point to next record.
                CMP     BL,CL                   ;First characters match?
                JNZ     NEXT_MATCH              ;If no, check next record.
                OR      BL,BL                   ;Extended code? ie zero.
                JNZ     MATCH                   ;If no, then match.
                CMP     BH,CH                   ;Else, check second code.
                JNZ     NEXT_MATCH              ;If not same, no match.
MATCH:          CLC                             ;Else return with CY = 0
                RET

NO_MATCH:       STC                             ;No match; CY = 1.
                RET


;-----------------------------------------------------------------------;
; Buffer area will write over disposable data and initislization code. ;
;-----------------------------------------------------------------------;
ESC_BUFFER              =       $
NUMBER_BUFFER           =       ESC_BUFFER + ESC_BUFFER_SIZE
PARSE_BUFFER            =       NUMBER_BUFFER
DEVICE_STATUS_BUFFER    =       PARSE_BUFFER + ESC_BUFFER_SIZE
DEVICE_STATUS_SIZE      =       11
REASSIGNMENT_BUFFER     =       DEVICE_STATUS_BUFFER + DEVICE_STATUS_SIZE

;               DISPOSABLE DATA
;               ---------------

SYNTAX          LABEL   BYTE
DB      "Usage:  ANSI [FAST | SLOW][ON | OFF][/# nnn][/C][/U]",CR,LF
DB      "FAST = direct screen writes; default",CR,LF
DB      "SLOW = screen writes via BIOS",CR,LF
DB      "ON/OFF = active/inactive; default is ON",CR,LF
DB      "nnn = buffer size in bytes (0-60K) reserved for key reassignment; "
DB      "default 200",CR,LF
DB      "/U = Uninstall"
CR_LF   DB      CR,LF,LF,"$"

STATUS_MSG      DB      "Status: $"
BUFFER_MSG      DB      CR,LF,"Buffer size: $"
BYTES_FREE      DB      CR,LF,"Bytes free:  $"
ANSI_SYS_MSG    DB      "ANSI.SYS is installed so "
NOT_INSTALLED   DB      "ANSI.COM not installed",CR,LF,"$"
CON             DB      "CON"
CON_OFFSET      EQU     10

SIZE_MSG        DB      "Uninstall to change buffer size",CR,LF,LF,"$"
UNLOAD_MSG      DB      "ANSI can't be uninstalled",CR,LF
                DB      "Uninstall resident programs in reverse order",CR,LF,"$"
NOT_ENOUGH      DB      "Not enough memory",CR,LF,"$"
ALLOCATE_MSG    DB      "Memory allocation error",CR,LF,BELL,"$"
INSTALL_MSG     DB      "Installed",CR,LF,"$"
UNINSTALL_MSG   DB      "Uninstalled",CR,LF,"$"


;-----------------------------------------------------------------------;
; Search memory for a copy of our code, to see if already installed. ;
;-----------------------------------------------------------------------;
INITIALIZE      PROC    NEAR
                CLD                             ;All string operations forward.
                MOV     BX,OFFSET START         ;Point to start of code.
                NOT     BYTE PTR [BX]           ;Change a byte so no false match.
                XOR     DX,DX                   ;Start at segment zero.
                MOV     AX,CS                   ;Store our segment in AX.
NEXT_PARAG:     INC     DX                      ;Next paragraph.
                MOV     ES,DX
                CMP     DX,AX                   ;Is it our segment?
                JZ      ANNOUNCE                ;If yes, search is done.
                MOV     SI,BX                   ;Else, point to our signature.
                MOV     DI,BX                   ; and offset of possible match.
                MOV     CX,16                   ;Check 16 bytes for match.
                REP     CMPSB
                JNZ     NEXT_PARAG              ;If no match, keep looking.

;-----------------------------------------------;
ANNOUNCE:       MOV     DX,OFFSET SIGNATURE     ;Display our signature.
                CALL    PRINT_STRING
                MOV     DX,OFFSET SYNTAX        ;And syntax.
                CALL    PRINT_STRING
                MOV     SI,81H                  ;Point to command line.
NEXT_CAP:       LODSB                           ;Capitalize parameters.
                CMP     AL,CR
                JZ      PARSE
                CMP     AL,"a"
                JB      NEXT_CAP
                CMP     AL,"z"
                JA      NEXT_CAP
                AND     BYTE PTR [SI - 1],5FH
                JMP     SHORT NEXT_CAP

;-----------------------------------------------;
PARSE:          MOV     SI,81H                  ;Point to command line again.
NEXT_PARA:      XOR     AX,AX                   ;Position in status parameters.
                MOV     BX,4                    ;Status parameters each 4 bytes.
NEXT_STATUS:    MOV     DI,OFFSET PARAMETERS    ;Point to "OFF ON  SLOWFAST"
                ADD     DI,AX                   ;Point to next parameter.
                ADD     AX,BX
                CMP     AX,LAST_PARAMETER       ;Check all four possible statuses
                JA      CK_SWITCHES
                PUSH    SI                      ;Save command line pointer.
                MOV     CX,2                    ;Check first two bytes for match.
                REP     CMPSB
                POP     SI                      ;Recover command line pointer.
                JNZ     NEXT_STATUS
                DIV     BL                      ;If match, divide offset by four.
                MOV     CL,1                    ;Set a bit to match offset.
NEXT_SHIFT:     SHL     CL,1
                DEC     AL
                JNZ     NEXT_SHIFT
                SHR     CL,1                    ;Adjust.
                MOV     AH,STATUS_MASK          ;Retrieve appropriate ON/OFF
                CMP     CL,ON                   ; or FAST/SLOW mask.
                JBE     ADD_STATUS
                ROL     AH,1
                ROL     AH,1

ADD_STATUS:     AND     ES:STATUS,AH            ;Mask off old status.
                OR      ES:STATUS,CL            ;Add new status.
                INC     SI                      ;Bump command line pointer.
                INC     SI
```

continued . . .

```
CK_SWITCHES:  LODSB                                  ;Get a byte.
              CMP     AL,CR                          ;Is it carriage return?
              JZ      INSTALL                        ;If yes, done here.
              CMP     AL,"/"                         ;Is there a switch character?
              JNZ     NEXT_PARA                      ;If no, keep looking.
              LODSB                                  ;Else, get the switch character.
              CMP     AL,"B"                         ;Is it "B" ?
              JNZ     CK_C                           ;If no, check "c".


              CALL    CK_INSTALLED                   ;Else, see if already installed.
              JZ      GET_SIZE                       ;If no, get buffer request size.
              MOV     DX,OFFSET SIZE_MSG             ;Else, display error message.
              CALL    PRINT_STRING
              JMP     SHORT NEXT_PARA
GET_SIZE:     CALL    DECIMAL_INPUT                  ;Get number parameter.
              CMP     BX,REASSIGNMENT_MAX            ;If greater than maximum, use
              JBE     STORE_BUFFER                   ; maximum, else use requested
              MOV     BX,REASSIGNMENT_MAX            ; size.
STORE_BUFFER: MOV     REASSIGHMENT_SIZE,BX
              ADD     BX,OFFSET REASSIGNMENT_BUFFER  ;Calculate end of buffer.
              MOV     BUFFER_END,BX
              JMP     SHORT NEXT_PARA

CK_C:         CMP     AL,"C"                         ;Is it "C" ?
              JNZ     CK_U                           ;If not, check "U".
              MOV     ES:REASSIGN_END,OFFSET REASSIGNMENT_BUFFER  ;Clear buffer

CK_U:         CMP     AL,"U"                         ;Is it "U" ?
              JZ      DO_U                           ;If yes, try to uninstall.
              JMP     NEXT_PARA                      ;Else, next parmater.
DO_U:         CALL    CK_INSTALLED                   ;Else, see if installed.
              MOV     DX,OFFSET NOT_INSTALLED        ;If no, exit with error message.
              JZ      LILLY_PAD                      ;Too far for short jump.
              JMP     UNINSTALL                      ;Else, uninstall.

;-----------------------------------------------------;
INSTALL:      CALL    STATUS_REPORT                  ;Display status.
              CALL    CK_INSTALLED                   ;Check if already installed.
              JZ      CK_AVAILABLE                   ;If no, see if enough memory.
              OR      AL,AL                          ;Else, done.
              JMP     EXIT                           ;Exit with ERRORLEVEL = 0.


;-------------------------------;
; This is the install procedure. ;
;-------------------------------;
CK_AVAILABLE: MOV     BP,OFFSET REASSIGNMEHT_BUFFER  ;TSR ends at end
              ADD     BP,REASSIGNMENT_SIZE           ; of reassignment buffer.
              ADD     BP,15                          ;Round up.
              CMP     BP,DS:[6]                      ;Buffer > PSP bytes in segment?
              MOV     DX,OFFSET NOT_ENOUGH           ;If yes, exit without installing
              JA      MSG_EXIT                       ; with message.

              MOV     AX,3529H                       ;Get undocumented INT 29 vector.
              INT     21H
              MOV     SI,OFFSET CON                  ;Does it point to ANSI.SYS?
              MOV     DI,CON_OFFSET                  ;Check by looking for "CON"
              MOV     CX,3                           ; as device name.
              REP     CMPSB
              MOV     DX,OFFSET ANSI_SYS_MSG         ;Exit with error message if
LILLY_PAD:    JZ      MSG_EXIT                       ;ANSI.SYS loaded.

              MOV     OLD_INT_29[0],BX               ;Else save old interrupt.
              MOV     OLD_INT_29[2],ES
              MOV     DX,OFFSET ANSI_INT_29          ;Install new interrupt.
              MOV     AX,2529H
              INT     21H
              MOV     AX,3516H                       ;Get INT 16 vector.
              INT     21H
              HOV     OLD_INT_16[0],BX               ;Save old interrupt.
              MOV     OLD_INT_16[2],ES
              MOV     DX,OFFSET ANSI_INT_16          ;Install new interrupt.
              MOV     AX,2516H
              INT     21H
              MOV     AX,3521H                       ;Get DOS 21h interrupt.
              INT     21H
              MOV     OLD_INT_21[0],BX               ;Save old interrupt.
              MOV     OLD_INT_21[2],ES
              MOV     DX,OFFSET ANSI_IHT_21          ;Install new interrupt.
              MOV     AX,2521H
              INT     21H

              MOV     AX,DS:[2CH]                    ;Get environment segment.
              MOV     ES,AX
              MOV     AH,49H                         ;Free up environment.
              INT     21H


              MOV     DX,OFFSET INSTALL_MSG          ;Display install message.
              CALL    PRINT_STRING
              CALL    FLUSH_END                      ;Setup the number buffer.
              MOV     DX,BP                          ;Retrieve resident byte request.
              MOV     CL,4
              SHR     DX,CL                          ;Convert to paragraphs.
              MOV     AX,3100H                       ;Return error code of zero.
              INT     21H                            ;Terminate but stay resident.

;-----------------------------------------------------------------;
; Exit.  Return ERRORLEVEL code 0 if successful, 1 if unsuccessful. ;
;-----------------------------------------------------------------;
MSG_EXIT:     CALL    PRINT_STRING
ERROR_EXIT:   MOV     AL,1                           ;ERRORLEVEL = 1.
EXIT:         MOV     AH,4CH                         ;Terminate.
              INT     21H

;-----------------------------------------------------;
; This subroutine uninstalls the resident ANSI.COM. ;
;-----------------------------------------------------;
UNINSTALL:    AND     ES:STATUS,NOT OH               ;Turn OFF ANSI, just in case
              OR      ES:STATUS,OFF                  ; can't uninstall.
              HOV     CX,ES                          ;Save segment in CX.
              MOV     AX,3529H                       ;Get interrupt 29h.
              INT     21H
              CMP     BX,OFFSET ANSI_INT_29          ;Has it been hooked by another?
              JNZ     UNINSTALL_ERR                  ;If yes, exit with error message.
              MOV     BX,ES
              CMP     BX,CX                          ;Is the segment vector same?
              JNZ     UHIHSTALL_ERR                  ;If no, exit with error message.

              MOV     AX,3516H                       ;Get interrupt 16h.
              IHT     21H
              CMP     BX,OFFSET ANSI_INT_16          ;Has it been hooked by another?
              JHZ     UNIHSTALL_ERR                  ;If yes, exit with error message.
              MOV     BX,ES
              CMP     BX,CX                          ;Is the segment vector same?
              JNZ     UNINSTALL ERR                  ;If no, exit with error message.

              MOV     AX,3521H                       ;Get interrupt 21h.
              IHT     21H
              CMP     BX,OFFSET ANSI_INT_21          ;Has it been hooked by another?
              JNZ     UNINSTALL_ERR                  ;If yes, exit with error message.
              MOV     BX,ES
              CMP     BX,CX                          ;Is the segment vector same?
              JNZ     UNINSTALL_ERR                  ;If no, exit with error message.

              MOV     AH,49H                         ;Return memory to system pool.
              INT     21H
              MOV     DX,OFFSET ALLOCATE_MSG
              JC      MSG_EXIT                       ;Display message if problem.

              MOV     DX,ES:OLD_INT_29[0]            ;Restore old INT 29.
              MOV     DS,ES:OLD_INT_29[2]
              MOV     AX,2529H
              INT     21H
              MOV     DX,ES:OLD_INT_16[0]            ;Restore old IHT 16.
              MOV     DS,ES:OLD_INT_16[2]

              MOV     AX,2516H
              IHT     21H
              MOV     DX,ES:OLD_INT_21[0]            ;Restore old INT 21.
              MOV     DS,ES:OLD_INT_21[2]
              MOV     AX,2521H
              INT     21H

              PUSH    CS
              POP     DS                             ;Point to our data.
              MOV     DX,OFFSET UHINSTALL_MSG        ;Display uninstall message.
              CALL    PRIMT_STRIHG
              OR      AL,AL                          ;Exit with ERRORLEVEL = 0.
              JMP     EXIT

UNINSTALL_ERR: MOV    ES,CX                          ;If error, display OFF status.
              CALL    STATUS_REPORT
              MOV     DX,OFFSET UNLOAD_MSG           ;And exit with error message.
              JMP     MSG_EXIT
IHITIALIZE    ENDP

;-----------------------------------------------------;
; INPUT:  SI points to parameter start.               ;
; OUTPUT: SI points to parameter end; BX = number. ;
;-----------------------------------------------------;
DECIMAL_IHPUT PROC    NEAR
              XOR     BX,BX                          ;Start with zero as number.
NEXT_DECIMAL: LODSB                                  ;Get a character.
              CMP     AL,CR                          ;Carriage return?
              JZ      ADJUST_DEC                     ;If yes, done here.
              CMP     AL,"/"                         ;Forward slash?
              JZ      ADJUST_DEC                     ;If yes, done here.
              SUB     AL,"0"                         ;ASCII to binary.
              JC      NEXT_DECIMAL                   ;If not between 0 and 9, skip.
              CMP     AL,9
              JA      NEXT_DECIMAL
              CBW                                    ;Convert byta to word.
              XCHG    AX,BX                          ;Swap old and new number.
              MOV     CX,10                          ;Shift to left by multiplying
              MUL     CX                             ; last entry by ten.
              JC      DECIMAL_ERROR                  ;If carry, too big.
              ADD     BX,AX                          ;Add new number and store in BX.
              JNC     NEXT_DECIMAL                   ;If not carry, next number.
DECIMAL_ERROR: MOV    BX,-1                          ;Else, too big; return -1.

ADJUST_DEC:   DEC     SI                             ;Adjust pointer.
              RET
DECIMAL_INPUT ENDP

;-----------------------------------------------------------;
; OUTPUT: ZR = 1 if not installed; ZR = 0 if installed. ;
;-----------------------------------------------------------;
CK_INSTALLED: MOV     AX,ES
              MDV     BX,CS
              CMP     AX,BX                          ;Compare segments.


              RET

;-----------------------------------------------------;
STATUS_REPORT: MOV    DX,OFFSET STATUS_MSG
              CALL    PRINT_STRING                   ;Display "Status: ".
              MOV     BL,ES:STATUS
              MOV     BH,1
HEXT_REPORT:  TEST    BL,BH
              JZ      LOOP_STATUS

              MOV     DL,BH
              MOV     AX,-1
NEXT_BIT:     INC     AX
              SHR     DL,1
              JNC     NEXT_BIT
              SHL     AX,1
              SHL     AX,1
              MOV     SI,OFFSET PARAMETERS           ;Display appropriate ON or OFF,
              ADD     SI,AX                          ; SLOW or FAST.
              MOV     CX,4
REPORT:       LODSB
              CALL    PRINT_CHAR
              LOOP    REPORT

LOOP_STATUS:  SHL     BH,1
              CMP     BH,FAST
              JBE     NEXT_REPORT
              MOV     DX,OFFSET BUFFER_MSG           ;Display "Buffer size: ".
              CALL    PRINT_STRING
              MOV     AX,ES:REASSIGNMENT_SIZE
              PUSH    AK
              XOR     BP,BP
              CALL    DECIMAL_OUT                    ;Display the size.
              MOV     DX,OFFSET BYTES_FREE           ;Display "Bytes free: ".
              CALL    PRINT_STRING
              POP     AX
              ADD     AX,OFFSET REASSIGNMENT_BUFFER
              SUB     AX,ES:REASSIGN_END
              CALL    DECIMAL_OUT                    ;Display the bytes free.
              MOV     DX,OFFSET CR_LF
              CALL    PRINT_STRING
              RET

;-----------------------------------------------------;
PRINT_CHAR:   MOV     DL,AL
              MOV     AH,2                           ;Print character via DOS.
              JMP     SHORT DOS_INT

PRIHT_STRING: MOV     AH,9                           ;Print string via DOS.
DOS_INT:      INT     21H
              RET

_TEXT         ENDS
              END     START
```

ends

```
100 REM - BASIC PROGRAM TO CREATE ANSI.COM
110 CLS:PRINT "Creating ANSI.COM      ": OPEN "ANSI.COM     " AS #1 LEN = 1
120 FIELD #1, 1 AS A$: CHECKSUM#=0
130 FOR I = 1 TO   182
140  LINESUM#=0: LOCATE 2,3: PRINT "Countdown: "   182 - I ;
150  FOR J = 1 TO  16: READ BYTE$: BYTE = VAL("&H" + BYTE$)
160   CHECKSUM# = CHECKSUM# + BYTE: LINESUM# = LINESUM# + BYTE
170   IF (BYTE < 256) THEN LSET A$=CHR$(BYTE): PUT #1
180  NEXT J
190  READ LINETOT$: LINECHECK# = VAL("&H"+LINETOT$)
200  IF LINECHECK# = LINESUM# THEN GOTO 220
210  LOCATE 4,2: PRINT "Error in line #"  ;  260 +  10 * I: GOTO 260
220 NEXT I
230 CLOSE: READ FILETOT$ : FILECHECK# = VAL(FILETOT$)
240 IF CHECKSUM# <> FILECHECK# THEN GOTO 260
250 PRINT: PRINT "ANSI.COM      created successfully": SYSTEM
260 PRINT: PRINT "ANSI.COM      is not valid!": END
270 DATA E9, D3,  0, 41, 4E, 53, 49, 20, 31, 2E, 30, 20, 28, 43, 29, 20,   472
280 DATA 31, 39, 38, 39, 20, 5A, 69, 66, 66, 20, 43, 6F, 6D, 6D, 75, 6E,   519
290 DATA 69, 63, 61, 74, 69, 6F, 6E, 73, 20, 43, 6F, 2E,  D,  A, 50, 43,   504
300 DATA 20, 4D, 61, 67, 61, 7A, 69, 6E, 65, 20, 2D, 4D, 69, 63, 68, 6B,   6B8
310 DATA 61, 65, 6C, 20, 4A, 2E, 20, 4D, 65, 66, 66, 6F, 72, 64,  D,  A,   4C4
320 DATA  A, 24, 1A, 37,  3,  A, 4F, 46, 46, 20, 4F, 4E, 20, 20, 53, 4C,   383
330 DATA 4F, 57, 46, 41, 53, 54,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,   1D4
340 DATA  0,  0,  7,  0,  0,  2,  0,  0,  0,  0,  0,  D,  A,  8,  7, 48,    77
350 DATA 41, 42, 43, 44, 66, 6E, 73, 75, 4B, 6D, 68, 6C, 70, 4A, D0,  5,   5E1
360 DATA F1,  5,  0,  6, 14,  6, 2B,  6, D0,  5, 3A,  6, 7A,  6, 83,  6,   365
370 DATA 8A,  6, B7,  6, DD,  6, E1,  6, FE,  6, 95,  5,  0,  1,  4,  5,   4BF
380 DATA  7,  8, 1E, 1F, 20, 21, 22, 23, 24, 25, 28, 29, 2A, 2B, 2C, 2D,   21A
390 DATA 2E, 2F,  0,  7, FF,  8, F8,  1, FF, 00, F8, 70, 88,  0, F8,  0,   6CB
400 DATA F0,  4, F0,  2, F8,  6, F8,  1, F0,  5, F0,  3, F8,  7, 0F,  8,   773
410 DATA 0F, 40, 0F, 20, 8F, 60, 8F, 10, 8F, 50, 0F, 30, 0F, 70,  0,  0,   5A9
420 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  2,  0,  0,     5
430 DATA  0,  0,  0,  0, C8,  0, C4,  0, 8C,  9, 50, 53, 51, 52, 56, 57,   41C
440 DATA 1E,  6, 55, FC, 8C, CB, 8E, DB, 8E, C3, FF, 16, 53,  1, 5D,  7,   753
450 DATA 1F, 5F, 5E, 5A, 59, 5B, 58, CF, 9C, 00, FC,  A, 74, 1A, 00, FC,   73D
460 DATA  7, 74, 15, 00, FC,  1, 74, 10, 80, FC,  6, 75,  5, 00, FA, FF,   706
470 DATA 74,  6, 9D, 2E, FF, 2E, 6E,  1, 9D, 2E, C6,  6, FB,  1,  2, 9C,   612
480 DATA 2E, FF, 1E, 6E,  1, 2E, C6,  6, FB,  1,  1, CA,  2,  0, FB, 9C,   614
490 DATA 55, 1E, 50,  E, 1F, 00, 3E, FB,  1,  1, 74, 17, 80, 3E, FC,  1,   4F1
500 DATA  2, 74, 10, FC, 8B, EC, 88, 26,  3,  2, 80, E4, EF, 74, 11, FE,   782
510 DATA CC, 74,  9, 50, 1F, 5D, 9D, 2E, FF, 2E, 6A,  1, 80, 4E,  6, 48,   594
520 DATA C6,  6, FC,  1,  2, 50, 53, 51, 52, 56, 57, 80, 3E, FD,  1,  2,   584
530 DATA 74, 37, 9C, FF, 1E, 6A,  1,  9, FC, FE,  6,  3,  2,  1, 74,  8,   9D6
540 DATA 74, 75, 80, 66,  6, BF, 9C, 9D, 8B, D8, E0, DE,  4, 72, 68, 0B,   85F
550 DATA  5, 2D,  3,  8, 83, C7,  3,  A, DB, 75,  2, 48, 47, A3, FF,  1,   510
560 DATA 89, 3E,  1,  2, C6,  6, FD,  1,  2, F6,  6,  3,  2,  1, 75, 12,   41F
570 DATA 80, 3E, FE,  1,  1, 74,  B, 8A, 26,  3,  2, CD, 16, C6,  6, FE,   59F
580 DATA  1,  1, 8B, 3E,  1,  2, 0B,  5, F6,  6,  3,  2,  1, 74,  6, 80,   35A
590 DATA 66,  6, BF, EB, 22, FF,  6,  1,  2, FF,  E, FF,  1, 74,  E,  A,   5D9
600 DATA C0, 75, 14, FF,  6,  1,  2, FF,  E, FF,  1, 75,  A, C6,  6, FD,   6A6
610 DATA  1,  1, C6,  6, FE,  1,  2, C6,  6, FC,  1,  1, 5F, 5E, 5A, 59,   509
620 DATA 5B, 1F, 5D, 9D, CA,  2,  0, BB, 41,  3, 3C, 1B, 74,  8, E9, CB,   5C6
630 DATA  0, BB, 4B,  3, 3C, 5B, 74, 70, E9, 8C,  0, BB, 66,  3, 3C, 32,   593
640 DATA 74, 6E, F6,  6, 55,  1,  1, 75, 7E, BB, 7F,  3, 3C, 3D, 74, 60,   5B2
650 DATA 3C, 3F, 74, 5C, EB, 13, 3C, 4A, BF, AA,  1, 74, 43, F6,  6, 55,   641
660 DATA  1,  1, 75, 63, C6,  6, 3B,  8,  2, C7,  6, 53,  1, 7F,  3, E8,   476
670 DATA 38,  1, 74, 3C, 3C, 3B, 75,  6, FF,  6, 79,  1,  1, EB, 36, 3C,   3E7
680 DATA 72, 45, 3C, 39, 77,  5, E8, 35,  1, EB, 29, BF, 7F,  1, B9,  F,   5E1
690 DATA  0, F2, AE, 75, 32, FF,  6, 79,  1, BF, AA,  1, D1, E1, 2B, F9,   806
700 DATA E8, 84,  1, FF, 15, EB, 36, 3A,  6, 76,  1, 75,  7, BB, 7F,  3,   612
710 DATA 89, 1E, 53,  1, 8B, 3E, 77,  1, B3, FF, 7E, 74,  A, 81, C7, BD,   6BF
720 DATA  7, AA, FF,  6, 77,  1, C3, 50, BE, BD,  7, 8B,  E, 77,  1,  AC,   680
730 DATA 51, 56, E0, 27,  0, 5E, 59, E2, F6, 58, E8, 1F,  0, C7,  6, 77,   6E8
740 DATA  1,  0,  0, C7,  6, 53,  1, 37,  3, C7,  6, 79,  1,  0,  0,  E,   2B1
750 DATA  7, BF, 3B,  8, B9, 3F,  0, 33, C0, F3, AB, C3, BF, 7B,  1, B9,   749
760 DATA  4,  0, F2, AE, 74, 4E, E8, 1E,  1, 3C,  9, 75, 16, 8B,  E, F8,   5CE
770 DATA  1, 83, E1,  7, F7, D9, 83, C1,  8, 51, 80, 20, E8,  4,  0, 59,   6EE
780 DATA E2, F7, C3, F6,  6, 55,  1,  1, 75, 2A, E8, A9,  0, 73, 2C, 50,   70E
790 DATA B9,  1,  0, 8A, 3E, EE,  1, 8A, 1E, 72,  1, B4,  9, CD, 10, 58,   57E
800 DATA 80, 3E, 75,  1,  2, 74,  F, 8B,  E, F2,  1, FE, C9, 3A,  E, F0,   64C
810 DATA  1, 75,  3, C3, B3,  7, B4,  E, CD, 10, C3,  6, 0B, 16, F0,  1,   5F8
820 DATA E8, 9D,  0, 0A, 26, 72,  1, AB, FE, C2, 3A, 16, F2,  1, 72, 35,   6FD
830 DATA 00, 3E, 75,  1,  1, 74, 31, 32, D2, FE, C6, E0, 71,  0, 3A, F0,   725
840 DATA 76, 23, FE, CE, 8B, 3E, F6,  1, 8B, F7, A1, F2,  1, 50,  3, F0,   07E
850 DATA  3, F0, F6, E6, 8B, C8, 1E,  6, 1F, F3, A5, 17, 59, B0, 20, 0A,   7CF
860 DATA 26, 72,  1,  1, F3, AB, E0, 32,  1,  7, C3, BB, B7,  3, B4, 22,   3A,   69F
870 DATA C4, 74,  6, B4, 27, 3A, C4, 75,  4, 80, 26, 76,  1, C3, 50, 2C,   5F4
880 DATA 30, 0A, C8, BB,  A,  0, BB, 1E, 79,  1, F6, A7, 3B,  8,  2, C1,   60A
890 DATA 88, 87, 3B,  8, 58, C3, F6,  6, 55,  1,  4, 75, 10, 80, 3E, F1,   5F7
900 DATA  1,  7, 74,  7, 80, 3E, F1,  1,  3, 77,  2, F8, C3, F9, C3, 1E,   644
910 DATA B0, 20,  0, BE, D8, A0, 04,  0,  A, C0, 75,  2, B0, 18, 1F, C3,   655
920 DATA 50, A1, F2,  1, F6, E6, 8A, DA, 32, FF,  3, C3, D1, E0, 8B, 3E,   995
930 DATA F6,  1,  3, F8, BB,  0, B0, B1, 3E, EF,  1, B4,  3, 74,  4, 81,   6BC
940 DATA C3,  0,  8, 8E, C3, 58, C3, 1E, 57, BB, 40,  0, 8E, DB, BE, 62,   730
950 DATA  0, BF, EE,  1, A4, A5, BE, 49,  0, B9,  7,  0, F3, A4, 26, 8A,   705
960 DATA 1E, EE,  1, 32, FF, D1, E3,  3, F3, A5, 5F, 1F, C3, 8B, 1E, 3B,   7B2
970 DATA  0,  A, FF, 75,  2, FE, C7,  A, DB, 75,  2, FE, C3, 8B, 16, F8,   803
980 DATA  1, C3, BB,  A,  0, 33, C9, 33, D2, F7, F3, 80, C2, 30, 52, 41,   779
990 DATA 3D,  0,  0, 75, F2,  B, ED, 75,  7, 58, E8, C6,  6, E2, FA, C3,   7C3
1000 DATA 50, AA, E2, FC, C3, E8, 67, FF, 0A, F0, B7,  7, F6,  6, 55,  1,   97B
1010 DATA  1, 75, 1C, 8A, 3E, 72,  1, E8, 3C, FF, 72, 13, 8A, E7, B0, 20,   6B6
1020 DATA 33, D2, E0, 5B, FF, 0B,  E, F4,  1, D1, E9, F3, AB, EB, 29, 8A,   9CB
1030 DATA 16, F2,  1, FE, CA, 33, C9, B0,  0,  6, CD, 10, 33, D2, EB, 10,   77F
1040 DATA E8, 2C, FF, E8, 87, FF, 81, EB,  1,  1, 3A, D8, 77, 12, 3A, 3E,   802
1050 DATA F2,  1, 73,  C, 86, FB, 8B, D3, 8A, 3E, EE,  1, B4,  2, CD, 10,   79B
1060 DATA C3, E8, 69, FF,  A, F6, 74, F8, 2A, F3, 73, EC, 32, F6, EB, E8,   AF6
1070 DATA E8, FC, FE, E8, [illegible], FF, 3A, F0, 74, E6,  2, F3, 3A, F0, 76, D8,   B11
1080 DATA 8A, F0, EB, D4, [illegible] 46, FF, 8A, 3E, F2,  1, FE, CF, 3A, D7, 74,   A73
1090 DATA CF,  2, D3, 3A, [illegible], 76, C1, 8A, D7, EB, BD, EB, 2F, FF,  A, D2,   9E7
1100 DATA 74, BE, 2A, D3, 73, B2, 32, D2, EB, AE, BF, B9,  8, B0, 1B, AA,   8E6
1110 DATA B0, 5B, AA, A0, F9,  1, 32, E4, 40, BD,  1,  0, E8, 23, FF, B0,   81D
1120 DATA 3B, AA, A0, F8,  1, 32, E4, 40, E8, 17, FF, B0, 52, AA, B0,  D,   83B
1130 DATA AA, 81, EF, B9,  8, 89, 3E, FF,  1, C7,  6,  1,  2, B9,  8, C6,   6F9
1140 DATA  6, FD,  1,  2, C6,  6, FE,  1,  1, C3, 8B, 16, F8,  1, 89, 16,   5CE
1150 DATA 73,  1, C3, 8B, 16, 73,  1, E9, 5E, FF, 88, 16, F8,  1, 8B,  E,   6C5
1160 DATA F2,  1, 2A, CA, E8, 4F, FE, 72,  C, E8, 74, FE, B0, 20, 8A, 26,   874
1170 DATA 72,  1, F3, AB, C3, 8B, CA, 8A, 16, F2,  1, FE, CA, 8A, 3E, 72,   8BE
1180 DATA  1, B8,  1,  6, CD, 10, C3, BE, 3B,  8, AC, BF, AC,  1, B9, 16,   648
1190 DATA  0, F2, AE, 75, 11, BF, EC,  1, D1, E1, 2B, F9, 8B,  5, 20,  6,   75E
1200 DATA 72,  1,  8, 26, 72,  1, FF,  E, 79,  1, 75, DE, C3, B4,  2, EB,   652
1210 DATA  2, B4,  1, A0, 3B,  8, 3C,  7, 74,  F, 72,  8, 3C,  E, 72, D,   3A3
1220 DATA 3C, 13, 77,  9, 32, E4, CD, 10, C3, 88, 26, 75,  1, C3, 88,  E,   605
1230 DATA 77,  1, 49, BE, BF,  7, BF, 3B,  8, C6,  6, 76,  1,  0, 32, D2,   58E
1240 DATA 33, ED, AC, 49, 74, 33, 8A, 26, 76,  1,  A, E4, 75, 24, E8, 99,   6EB
1250 DATA FD, 74, EF, 3C, 3B, 74, 12, 2C, 30, 8A, F0, B8,  A,  0, F6, E2,   7CD
1260 DATA  2, C6, 8A, D0, BD,  1,  0, EB, D9,  B, ED, 74, D5, 8A, C2, AA,   8DB
1270 DATA EB, CC, 3A, C4, 74, C3, AA, EB, C9,  B, ED, 74,  3, 8A, C2, AA,   9AF
1280 DATA 81, EF, 39,  8, 8B, C7, 8B, 1E, 3B,  8, B9,  5,  0,  A, DB, 74,   606
1290 DATA  1, 49, 3B, C1, 72, 30, 8B, 2E,  8,  2, E8, 2E,  0, 72, 12, 8B,   4D0
1300 DATA CA, 2B, CE, 74,  C, 2B, 15,  3, D0, 3B, D5, 77, 19, F3, A4, EB,   778
1310 DATA  6,  3, D0, 3B, D5, 77, 1F, BE, 3B,  8, AB, 8B, C8, 49, 49, F3,   6F3
1320 DATA A4, 89, 3E,  6,  2, C3, B0, 70, E9, 3C, FC, 8B, 16,  6,  2, BE,   6DE
1330 DATA C4,  8, 8B, FE, 3B, FA, 73, 13, 8B, 4D,  2,  3, 35, 3A, D9, 75,   6AA
1340 DATA F1,  A, DB, 75,  4, 3A, FD, 75, E9, F8, C3, F9, C3, 55, 73, 61,   984
1350 DATA 67, 65, 3A, 20, 20, 41, 4E, 53, 49, 20, 5B, 46, 41, 53, 54, 20,   43A
1360 DATA 7C, 20, 53, 4C, 4F, 57, 5D, 5B, 4F, 4E, 20, 7C, 20, 4F, 46, 46,   4CD
1370 DATA 5D, 5B, 2F, 42, 20, 6E, 6E, 6E, 5D, 5B, 2F, 43, 5D, 5B, 2F, 55,   4F9
1380 DATA 5D,  D,  A, 46, 41, 53, 54, 20, 3D, 20, 64, 69, 72, 65, 63, 74,   49A
1390 DATA 20, 73, 63, 72, 65, 65, 6E, 20, 77, 72, 69, 74, 65, 73, 3B, 20,   5B9
1400 DATA 64, 65, 66, 61, 75, 6C, 74,  D,  A, 53, 4C, 4F, 57, 20, 3D, 20,   4BE
1410 DATA 73, 63, 72, 65, 65, 6E, 20, 77, 72, 69, 74, 65, 73, 20, 76, 69,   63D
1420 DATA 61, 20, 42, 49, 4F, 53,  D,  A, 4F, 4E, 2F, 4F, 46, 46, 20, 3D,   3C9
1430 DATA 20, 61, 63, 74, 69, 76, 65, 2F, 69, 6E, 61, 63, 74, 69, 76, 65,   61E
1440 DATA 3B, 20, 64, 65, 66, 61, 75, 6C, 74, 20, 69, 73, 20, 4F, 4E,  D,   506
1450 DATA  A, 6E, 6E, 6E, 20, 3D, 20, 62, 75, 66, 66, 65, 72, 20, 73, 69,   547
1460 DATA 7A, 65, 20, 69, 6E, 20, 62, 79, 74, 65, 73, 20, 28, 30, 2D, 36,   4F8
1470 DATA 38, 4B, 29, 20, 72, 65, 73, 65, 72, 76, 65, 64, 20, 66, 6F, 72,   58B
1480 DATA 20, 6B, 65, 79, 20, 72, 65, 61, 73, 73, 69, 67, 6E, 6D, 65, 6E,   625
1490 DATA 74, 3B, 20, 64, 65, 66, 61, 75, 6C, 74, 20, 32, 30, 38,  D,  A,   47D
1500 DATA 2F, 55, 20, 3D, 20, 55, 6E, 69, 6E, 73, 74, 61, 6C, 6C,  D,  A,   4D2
1510 DATA  A, 24, 53, 74, 61, 74, 75, 73, 3A, 20, 24,  D,  A, 42, 75, 66,   464
1520 DATA 66, 65, 72, 20, 73, 69, 7A, 65, 3A, 20, 24,  D,  A, 42, 79, 74,   6DC
1530 DATA 65, 73, 20, 66, 72, 65, 65, 3A, 20, 20, 24, 41, 4E, 53, 49, 2E,   491
1540 DATA 53, 59, 53, 20, 69, 73, 20, 69, 6E, 73, 74, 61, 6C, 6C, 65, 64,   5DB
1550 DATA 20, 73, 6F, 20, 41, 4E, 53, 49, 2E, 43, 4F, 4D, 20, 6E, 6F, 74,   4CB
1560 DATA 20, 69, 6E, 73, 74, 61, 6C, 6C, 65, 64,  D,  A, 24, 43, 4F, 4E,   4FB
1570 DATA 55, 6E, 69, 6E, 73, 74, 61, 6C, 6C, 20, 74, 6F, 20, 63, 68, 61,   689
1580 DATA 6E, 67, 65, 20, 62, 75, 66, 66, 65, 72, 20, 73, 69, 7A, 65,  D,   5BC
1590 DATA  A,  A, 24, 41, 4E, 53, 49, 20, 63, 61, 6E, 27, 74, 20, 62, 65,   437
1600 DATA 20, 75, 6E, 69, 6E, 73, 74, 61, 6C, 6C, 65, 64,  D,  A, 55, 6E,   59D
1610 DATA 69, 6E, 73, 74, 61, 6C, 6C, 20, 72, 65, 73, 69, 64, 65, 6E, 74,   675
1620 DATA 20, 70, 72, 6F, 67, 72, 61, 6D, 73, 20, 69, 6E, 20, 72, 65, 76,   5EF
1630 DATA 65, 72, 73, 65, 20, 6F, 72, 64, 65, 72,  D,  A, 24, 4E, 6F, 74,   557
1640 DATA 20, 65, 6E, 6F, 75, 67, 68, 20, 6D, 65, 6D, 6F, 72, 79,  D,  A,   576
1650 DATA 24, 4D, 65, 6D, 6F, 72, 79, 20, 61, 6C, 6C, 6F, 63, 61, 74, 69,   606
1660 DATA 6F, 6E, 20, 65, 72, 72, 6F, 72,  D,  A,  7, 24, 49, 6E, 73, 74,   507
1670 DATA 61, 6C, 6C, 65, 64,  D,  A, 24, 55, 6E, 69, 6E, 73, 74, 61, 6C,   58B
1680 DATA 6C, 65, 64,  D,  A, 24, FC, BB,  0,  1, F6, 17, 33, D2, 8C, C8,   68E
1690 DATA 42, 8E, C2, 3B, D0, 74,  B, 8B, F3, 8B, FB, B9, 10,  0, F3, A6,   882
1700 DATA 75, EE, BA,  3,  1, E8, 61,  2, BA, BD,  7, E8, 5B,  2, BE, 81,   76E
1710 DATA  0, AC, 3C,  D, 74,  E, 3C, 61, 72, F7, 3C, 7A, 77, F3, 80, 64,   681
1720 DATA FF, 5F, EB, ED, BE, 81,  0, 33, C0, BB,  4,  0, BF, 56,  1,  3,   740
1730 DATA F8,  3, C3, 3D, 10,  0, 77, 2C, 56, B9,  2,  0, F3, A6, 5E, 75,   62B
1740 DATA EB, F6, F3, B1,  1, D0, E1, FE, CB, 75, FA, D0, E9, B4, FC, 80,   C55
1750 DATA F9,  2, 76,  4, D0, C4, D0, C4, 26, 20, 26, 55,  1, 26,  8,  E,   59B
1760 DATA 55,  1, 46, 46, AC, 3C,  D, 74, 4D, 3C, 2F, 75, BA, AC, 3C, 42,   55C
1770 DATA 75, 27, E8, 8A,  1,  74,  8, BA, 20,  9, E8, EC,  1, EB, A8, E8,   7BE
1780 DATA 58,  1, 81, FB,  0, F8, 76,  3, BB,  0, F0, 89, 1E,  4,  2, 81,   617
1790 DATA C3, C4,  8, 89, 1E,  8,  2, EB, 8E, 3C, 43, 75,  7, 26, C7,  6,   5A7
1800 DATA  6,  2, C4,  8, 3C, 55, 74,  3, E9, 7C, FF, E8, 51,  1, BA,  4,   638
1810 DATA  9, 74, 36, E9, 95,  0, E8, 4D,  1,  1, E8, 43,  1, 74,  5,  A,   C0,   5D6
1820 DATA E9, 84,  0, BD, C4,  8,  3,  2E,  4,  2, 83, C5,  F, 3B, 2E,  6,   4F3
1830 DATA  8, BA, 8D,  9, 77, 6C, B0, 29, 35, CD, 21, BE, 1D,  9, BF,  A,   5E4
1840 DATA  0, B9,  3,  0, F3, A6, BA, EB, 8B, 74, 57, 89, 1E, 66,  1, 8C,   667
1850 DATA  6, 68,  1, BA,  A,  2, B8, 29, 25, CD, 21, B8, 16, 35, CD, 21,   51A
1860 DATA 89, 1E, 6A,  1, 8C,  6, 6C,  1, BA, 5D,  2, B8, 16, 25, CD, 21,   50C
1870 DATA B8, 21, 35, CD, 21, 89, 1E, 6E,  1, 8C,  6, 70,  1, BA, 28,  2,   4F9
1880 DATA B8, 21, 25, CD, 21, A1, 2C,  0, 8E, C0, B4, 49, CD, 21, BA, BC,   768
1890 DATA  9, E8, 35,  1, E8, C6, F8, 8B, D5, B1,  4, D3, EA, B8,  0, 31,   888
1900 DATA CD, 21, E8, 24,  1, B0,  1, B4, 4C, CD, 21, 26, 80, 26, 55,  1,   5BC
1910 DATA FD, 26, 80,  E, 55,  1,  1, 8C, C1, B8, 29, 35, CD, 21, 81, FB,   6D5
1920 DATA  A,  2, 75, 6B, 8C, C3, 3B, D9, 75, 65, B8, 16, 35, CD, 21, 81,   69B
1930 DATA FB, 5E,  2, 75, 5A, 8C, C3, 3B, D9, 75, 54, B8, 21, 35, CD, 21,   752
1940 DATA 81, FB, 28,  2, 75, 49, 8C, C3, 3B, D9, 75, 43, B4, 49, CD, 21,   76A
1950 DATA BA, A1,  9, 72, AD, 26, 8B, 16, 66,  1, 26, 8E, 1E, 68,  1, B8,   5A4
1960 DATA 29, 25, CD, 21, 26, 8B, 16, 6A,  1, 26, 8E, 1E, 6C,  1, B8, 16,   47B
1970 DATA 25, CD, 21, 26, 8B, 16, 6E,  1, 26, 8E, 1E, 70,  1, B8, 21, 25,   48A
1980 DATA CD, 21,  E, 1F, BA, C8,  9, E8, 9F,  0,  A, C0, E9, 78, FF, 8E,   7E5
1990 DATA C1, E8, 32,  0, BA, 43,  9, E9, 68, FF, 33, DB, AC, 3C,  D, 74,   7A8
2000 DATA 1C, 3C, 2F, 74, 18, 2C, 30, 72, F3, 3C,  9, 77, EF, 98, 93, B9,   663
2010 DATA  A,  0, F7, E1, 72,  4,  3, D8, 73, E2, BB, FF, FF, 4E, C3, 8C,   8DE
2020 DATA C0, 8C, CB, 3B, C3, C3, BA, C2,  8, E8, 5D,  0, 26, 8A, 1E, 55,   7C4
2030 DATA  1, B7,  1, 84, DF, 74, 1C, 8A, D7, B8, FF, FF, 40, D0, EA, 73,   930
2040 DATA FB, D1, E8, BE, 56,  1,  3, F8, B9,  4, 00, AC, E8, 32,  8E8
2050 DATA  0, E2, FA, D0, E7, 80, FF,  8, 76, D9, BA, CB,  8, E8, 29,  0,   987
2060 DATA 26, A1,  4,  2, 50, 33, ED, E8, 38, F9, BA, DB,  8, E8, 19,  0,   6F4
2070 DATA 58,  5, C4,  8, 26, 2B,  6,  6,  2, E8, 26, F9, BA, 8E,  8, E8,   5F7
2080 DATA  7,  0, C3, 8A, D0, B4,  2, EB,  2, B4,  9, CD, 21, C3,  0,  0,   635
2090 DATA 294000
```

ANSI.BAS This Basic program will automatically create ANSI.COM if you don't have a macro assembler

tion. All you have to do is add the ANSI command parameters.

Before turning to the technical side of ANSI.COM, however, I do want to share with you a couple of my favourite ANSI command sequences, starting with the following date and time display:

```
PROMPT $e[s$e[;50H$d
    $t$h$h$h$h$h$h$h$e[u$p$g
```

This sequence uses a combination of the DOS PROMPT and ANSI cursor control. It saves the cursor position, moves the cursor to the top right of the screen, and displays the date, a few delimiting spaces and the time. PROMPT's $h destructive backspace metastring then erases the distracting second, and hundredth second places from the time display, after which the cursor is returned to the saved position where the current drive and directory are displayed, followed by the familiar greater-than-sign prompt. Note that the time will be updated only when you generate a new prompt.

Another sequence that I use on my own machine is

```
PROMPT $p =$g $e[8m $e[0m$e [D
```

This sequence displays the current drive

and directory, followed by a space, and a directing combination of an equals sign and a greater-than sign as the prompt. Since I hate a blinking cursor, the next part of the sequence changes the attribute to invisible ($e[8m) and displays a space. The attribute is then returned to a normal white on black ($e[0m) — use a colour of your liking here — and the cursor is moved backward one column to the visible space. The result is an invisible blinking cursor at an idle prompt. Of course as soon as you start to type a command, the cursor moves on to a space with a normal attribute and becomes visible again.

Undocumented DOS call

DOS calls on the undocumented INT 29h for all its screen output. If you trace through the INT 29 interrupt handler (without ANSI.SYS or ANSI.COM loaded), you'll find a very short procedure that makes a BIOS INT 10, Write TTY 0Eh function call. The BIOS Write TTY (teletype) is an all-purpose screen output function that takes care of updating the cursor position. It includes line wrap and even scrolls the screen when the cursor wraps on the last line. Since it's ANSI's job to filter all DOS output, all that ANSI has to do to accomplish its mission is steal the INT 29. ANSI thus gets a chance to look through all DOS output for its ESC-[combination.

The one sticky part about the escape sequence is that the alpha command character doesn't come until the *end* of the string. That means ANSI has to buffer all potential escape sequences (the tip, of course, is the starting ESC-[) until it gets the ending command character. If the command is invalid, the buffer is flushed to the screen. Otherwise, the string never makes it to the screen and ANSI takes the appropriate command-requested action. You probably noticed that in the ESCAPE.COM program the escape string is passed directly to ANSI.COM via an INT 29 call.

A state machine

The escape sequence structure provides a perfect opportunity for ANSI to implement what is called a state-machine technique. A state machine jumps to a procedure according to a particular stage in a sequence of events. For ANSI, the command syntax dictates what ANSI is to expect next. In the first stage, ANSI jumps directly to a short procedure that checks for the ESC character, passing all others through to the display. Once the ESC is detected, ANSI's interest is piqued, and on the next DOS INT 29 call, it can skip the ESC monitor and branch directly to a procedure that looks for the left bracket. If the left bracket is found, on subsequent calls ANSI goes directly to the third number-parameter-parsing procedure, which converts decimal numbers to binary. The parameter stage can go into a substage of its own if either a single or double quote is encountered. (The quote stage buffers string characters as literals until a closing matching quote is encountered, which kicks ANSI back to the parsing stage.)

If any stage is not satisfied — if, for example, the left bracket is not the second character or the command is invalid — the buffer is flushed and ANSI reverts to the first ESC-monitoring stage. The state machine technique makes for more efficient processing by avoiding unnecessary branching.

The special case of CLS

Normally DOS processes the DOS Clear Screen command with its own internal CON device handler. If DOS finds that the INT 29h vector does not point to the same segment as the kernel, DOS knows someone else has hooked the vector and assumes it is ANSI.SYS. If DOS thinks ANSI.SYS is installed, it passes the CLS job on to the driver by sending INT 29 an ESC[2J (Erase in Display) string. This is so that ANSI.SYS can clear the screen with any user-chosen attribute. That means ANSI.COM will receive an ESC[2J whenever you enter CLS. This would seem to be no problem since ANSI is already geared up for the task. You could, however, assign ANSI an OFF status even while it's still installed. In the OFF state, ANSI is supposed to pass all characters on to the screen, but you would certainly be disappointed if the ESC[2J were echoed to the screen instead of its clearing the screen. The ANSI program, therefore, provides a special state so that it can process the ESC[2J string even when ANSI is OFF. When ANSI is OFF, the screen is cleared, but with a normal white on black instead of with the assigned attribute.

Checking the input

DOS INT 29 makes filtering output escape sequences a breeze. Filtering input from the keyboard for key reassignment is a little trickier for ANSI. The DOS keyboard input calls are INT 21 functions 1, 6, 7 and Ah. These four DOS input functions serve only as a middleman, and they end up calling the BIOS INT 16 to do the actual work of retrieving keystrokes from the keyboard buffer. DOS adds little to the process other than overhead, which is why most programs are written to go directly to the BIOS INT 16 for key input. For those programs that *do* use DOS, however, ANSI must monitor the keyboard throughput. This is accomplished by hooking both the INT 21 and INT 16 interrupts.

The chain of events then goes something like this: A program like COMMAND.COM calls INT 21, function Ah (Buffered Keyboard Input) for a keystroke. Since ANSI has hooked INT 21, it gets called in the process and sets a flag if the function is one of the four key-input functions before it passes control on to the original (DOS) INT 21 handler. DOS gets control and calls INT 16 for the keystroke. Since ANSI has hooked INT 16 also, the call ends up in ANSI's lap again. ANSI sees that the flag has been set by the ANSI INT 21 handler, so after it calls on INT 16 to get a keystroke, it checks the returned key against its reassignment buffer. If a match is found, the reassigned string is passed back, a byte at a time. For any INT 16 calls that have not been called by DOS, the flag will not have been set, so ANSI passes the call on to the real INT 16 handler untouched.

In closing

The .ASM listing for ANSI.COM is amply documented and will repay inspection by present and prospective assembly language programmers. For all PC users, however, if you didn't use ANSI before because it wasn't flexible enough, you've run out of excuses not to experiment with extended screen and keyboard control.

END