; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)

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

; U9.ASM (include u9.asm) //// UNIX v1 -> u9.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)

; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)

; 1.44 MB Floppy Disk

; (11/03/2013)

;

; [ Last Modification: 30/06/2015 ] ;;; completed ;;;

;

; Derivation from UNIX Operating System (v1.0 for PDP-11)

; (Original) Source Code by Ken Thompson (1971-1972)

; <Bell Laboratories (17/3/1972)>

; <Preliminary Release of UNIX Implementation Document>

;

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; 01/09/2014

; 28/08/2014

; 28/07/2014

; 27/07/2014

; 23/07/2014

; 20/07/2014

; 12/07/2014

; 04/07/2014

; 30/06/2014

; 27/06/2014

; 25/06/2014

; 11/06/2014

; 03/06/2014

; 02/06/2014

; 05/05/2014

; 30/04/2014

; 17/04/2014

; 15/04/2014

; 04/04/2014 scroll\_up

; 07/03/2014

; 04/03/2014 act\_disp\_page --> tty\_sw

; 03/03/2014 int\_09h, int\_16h

; 28/02/2014 int\_16h

; 17/02/2014

; 14/02/2014

; 01/02/2014 write\_tty

; 18/01/2014

; 17/01/2014

; 13/01/2014 getc, putc

; 12/12/2013

; 10/12/2013

; 07/12/2013

; 04/12/2013 getc, putc, write\_tty

; 04/11/2013 drv\_init

; 24/07/2013 bf\_init

; 20/07/2013 bf\_init

; 19/07/2013 drv\_init

; 18/07/2013 drv\_init

; 17/07/2013 bf\_init

; 14/07/2013

; 13/07/2013 drv\_init, dparam (Retro UNIX 8086 v1 features only!)

; 21/05/2013 'ocvt' & 'ccvt' routines (in U7.ASM)

; 15/05/2013 'rcvt' & 'xmtt' routines (in U6.ASM)

; 11/03/2013

;;rcvt:

;; 'rcvt' routine is in U6.ASM (Retro UNIX 8086 v1 modification!)

;;xmtt:

;; 'xmtt' routine is in U6.ASM (Retro UNIX 8086 v1 modification!)

;;ocvt:

;; 'ocvt' routine is in U7.ASM (Retro UNIX 8086 v1 modification!)

;;ccvt:

;; 'ccvt' routine is in U7.ASM (Retro UNIX 8086 v1 modification!)

drv\_init:

; 04/11/2013

; 19/07/2013

; 18/07/2013

; 14/07/2013

; 13/07/2013

; Retro UNIX 8086 v1 feature only !

;

; Derived from DRVINIT.ASM (DRVINIT4) file of TR-DOS project

; by Erdogan Tan, (26/09/2009 --> 07/08/2011)

;

; Modified/Simplified for Retro UNIX 8086 v1

;

; (LBA disks excluded, hard disk file systems excluded)

;

; ((RUFS and/or TRFS/SINGLIX partitions will be validated

; in future RUNIX/TR-UNIX versions if they will be available.)

;

; Input: none

; Output:

; cf = 0 -> disk drive initialization is ok.

; cf = 1 -> error (error code in ah)

; ((Modified registers: AX, BX, CX, DX, SI, DI))

fd\_init:

xor dx, dx ; fd0

xor si, si ; 0

call dparam

inc si ; 1

cmp al, 2 ; 04/11/2013

jb short hd\_init

inc dl ; fd1

call dparam

hd\_init:

inc si ; 2

mov dl, 80h ; hd0

call dparam

jc short drv\_init\_lbs

; al = number of hard disk drives

cmp al, 2 ; 04/11/2013

jb short drv\_init\_lbs

mov byte ptr [brwdev], al ; 19/07/2013

@@:

dec byte ptr [brwdev] ; 19/07/2013

jz short drv\_init\_lbs

inc si

inc dl

call dparam

jmp short @b

drv\_init\_lbs:

push cs ; 14/07/2013

pop es ; 14/07/2013

xor bx, bx

mov dl, byte ptr [unixbootdrive]

@@:

cmp dl, byte ptr [BX]+drv.pdn

je short @f

cmp bx, si ; 19/07/2013

jnb short drv\_init\_err

inc bl

jmp short @b

drv\_init\_err:

mov ah, byte ptr [BX]+drv.err

stc

retn

@@:

cmp byte ptr [BX]+drv.err, 0

ja short drv\_init\_err

mov si, offset sb0 ; super block buffer

mov byte ptr [SI], bl ; Device Id

mov byte ptr [SI]+1, 4 ; Bit 10,

; read bit

mov byte ptr [rdev], bl ; 19/07/2013

mov bx, si

inc byte ptr [BX]+2 ; physical block number = 1

call diskio

mov byte ptr [BX]+1, 0 ; 18/07/2013

retn

dparam:

; 13/07/2013

; Retro UNIX 8086 v1 feature only !

;

push dx

mov ah, 08h

int 13h

mov byte ptr [SI]+drv.err, ah

jnc short @f

dparam\_error:

pop dx

retn

@@:

mov al, dl ; Number of disk drives

;cmp al, 1

;jb short dparam\_err

; dh = last head number

inc dh

mov dl, dh

xor dh, dh

shl si, 1 ; align to word ptr drv.hds

mov word ptr [SI]+drv.hds, dx

; number of heads

and cx, 3Fh

; SI is already aligned for word ptr drv.spt

mov word ptr [SI]+drv.spt, cx

shr si, 1 ; align to byte ptr drv.pdn

pop dx

mov byte ptr [SI]+drv.pdn, dl

; Physical drive number

retn

bf\_init:

; 24/07/2013 (from last to first)

; 20/07/2013 Device id reset (0FFh)

; 17/07/2013

; Buffer (pointer) initialization !

;

; Retro UNIX 8086 v1 feature only !

;

mov cl, nbuf

mov di, offset bufp

; 24/07/2013

mov ax, offset Buffer + (nbuf\*516)

mov dx, 0FFFFh

@@:

; 24/07/2013

sub ax, 516 ; 4 header + 512 data

stosw

mov si, ax ; 24/07/2013

; mov word ptr [SI], dx ; 0FF00h

mov byte ptr [SI], dl ; 0FFh

; Not a valid device sign

;mov word ptr [SI]+2, dx ; 0FFFFh

; Not a valid block number sign

dec cl

jnz short @b

mov ax, offset sb0

stosw

mov ax, offset sb1

stosw

; 20/07/2013

mov si, ax ; offset sb1

mov byte ptr [SI], dl ; 0FFh

;mov word ptr [SI]+2, dx ; 0FFFFh

;

retn

getc:

;04/07/2014 (rcvc has been removed)

; (serial port interrupts)

;27/06/2014 (rcvc, EOT)

;03/06/2014 (rcvc)

;02/06/2014 (rcvc has been moved here again)

;05/05/2014 (rcvc has been moved from here)

;17/04/2014

;15/04/2014 (rcvc)

;17/02/2014

;14/02/2014

;17/01/2014

;13/01/2014

;10/12/2013

;20/10/2013

;10/10/2013

;05/10/2013

;24/09/2013

;20/09/2013

;29/07/2013 (getc\_s, sleep -> idle)

;28/07/2013 (byte ptr [u.ttyn] = tty number)

;16/07/2013

;20/05/2013

;14/05/2013 (AH input instead of 'mov ax, byte ptr [ptty]')

;13/05/2013

; Retro UNIX 8086 v1 modification !

;

; 'getc' gets (next) character

; from requested TTY (keyboard) buffer

; INPUTS ->

; [u.ttyn] = tty number (0 to 7) (8 is COM1, 9 is COM2)

; AL=0 -> Get (next) character from requested TTY buffer

; (Keyboard buffer will point to

; next character at next call)

; AL=1 -> Test a key is available in requested TTY buffer

; (Keyboard buffer will point to

; current character at next call)

; OUTPUTS ->

; (If AL input is 1) ZF=1 -> 'empty buffer' (no chars)

; ZF=0 -> AX has (current) character

; AL = ascii code

; AH = scan code (AH = line status for COM1 or COM2)

; (cf=1 -> error code/flags in AH)

; Original UNIX V1 'getc':

; get a character off character list

;

; ((Modified registers: AX, BX, -CX-, -DX-, -SI-, -DI-))

;

; 16/07/2013

; mov byte ptr [getctty], ah

;

mov ah, byte ptr [u.ttyn] ; 28/07/2013

getc\_n:

; 10/10/2013

mov bx, offset ttychr

and ah, ah

jz short @f

shl ah, 1

; 17/02/2014

add bl, ah

adc bh, 0

; 24/09/2013

;mov bl, ah

;xor bh, bh

;shl bl, 1

;add bx, offset ttychr

@@:

mov cx, word ptr [BX] ; ascii & scan code

; (by kb\_int)

or cx, cx

jnz short @f

and al, al

jz short getc\_s

xor ax, ax

retn

@@:

and al, al

mov ax, cx

mov cx, 0

jnz short @f

getc\_sn:

mov word ptr [BX], cx ; 0, reset

cmp ax, cx ; zf = 0

@@:

retn

getc\_s:

; 14/02/2014 uquant -> u.quant

; 10/12/2013

; 20/10/2013

; 05/10/2013

; 24/09/2013

; 20/09/2013

; 29/07/2013

; 28/07/2013

; 16/07/2013

; tty of the current process is not

; current tty (ptty); so, current process only

; can use keyboard input when its tty becomes

; current tty (ptty).

; 'sleep' is for preventing an endless lock

; during this tty input request.

; (Because, the user is not looking at the video page

; of the process to undersand there is a keyboard

; input request.)

;; 29/07/2013

; 20/09/2013

;((Modified registers: AX, BX, CX, DX, SI, DI))

;

; 05/10/2013

; ah = byte ptr [u.ttyn] ; (tty number)

;

; 10/10/2013

gcw0:

mov cl, 10 ; ch = 0

gcw1:

call idle

mov ax, word ptr [BX] ; ascii & scan code

; (by kb\_int)

or ax, ax

jnz short gcw3

loop gcw1

;

mov ah, byte ptr [u.ttyn] ; 20/10/2013

; 10/12/2013

cmp ah, byte ptr [ptty]

jne short gcw2

; 14/02/2014

cmp byte ptr [u.uno], 1

jna short gcw0

gcw2:

call sleep

; 20/09/2013

mov ah, byte ptr [u.ttyn]

xor al, al

jmp short getc\_n

gcw3:

; 10/10/2013

xor cl, cl

jmp short getc\_sn

sndc: ; <Send character>

;

; 28/07/2014

; 27/07/2014

; 23/07/2014

; 20/07/2014

; 12/07/2014

; 04/07/2014

; 27/06/2014

; 25/06/2014

; 15/04/2014

; 13/01/2014

; 16/07/2013 bx

; 14/05/2013

;

; Retro UNIX 8086 v1 feature only !

;

; 12/07/2014

xor dh, dh

mov dl, ah

; 27/07/2014

sub dl, 8

; 25/06/2014

push ax

sndcs:

; 28/07/2014

; ; 27/07/2014

; mov cx, 10

;@@:

mov ah, 3 ; Get serial port status

int 14h

test ah, 20h ; Transmitter holding register empty ?

jnz short @f

; call idle

; loop @b

;

push dx

push bx

; 27/07/2014

mov bx, dx

add bx, offset tsleep

;

mov ah, byte ptr [u.ttyn]

;

mov byte ptr [BX], ah ; 27/07/2014

;

call sleep

pop bx

pop dx

jmp short sndcs

@@:

pop ax

@@:

;mov ah, 1 ; Send character

;int 14h

; 13/07/2014

push dx

or dl, dl

mov dx, 2F8h ;data port (COM2)

jnz short @f

add dx, 100h ;3F8h, data port (COM1)

@@:

out dx, al ;send on serial port

pop dx

; 27/07/2014

call idle

;

mov ah, 3 ; Get serial port status

int 14h

cmp ah, 80h ; time out error

cmc ; cf = 0 (OK), cf = 1 (error!)

@@:

retn

putc:

;27/07/2014

;23/07/2014

;20/07/2014

;27/06/2014 (sndc, EOT)

;25/06/2014

;05/05/2014

;15/04/2014

;13/01/2014

;04/12/2013 write\_tty

;03/12/2013 write\_tty, beep, waitf

; (for video page switch bug-fixing)

;30/11/2013

;04/11/2013

;30/10/2013

;24/09/2013 consistency check -> ok

;20/09/2013 (cx = repeat count)

; (int 10h, function 0Eh -> function 09h)

; (video page can be selected in function 09h only!)

;26/08/2013

;14/05/2013

; Retro UNIX 8086 v1 modification !

;

; 'putc' puts a character

; onto requested (tty) video page or

; serial port

; INPUTS ->

; AL = ascii code of the character

; AH = video page (tty) number (0 to 7)

; (8 is COM1, 9 is COM2)

; OUTPUTS ->

; (If AL input is 1) ZF=1 -> 'empty buffer' (no chars)

; ZF=0 -> AX has (current) character

; cf=0 and AH = 0 -> no error

; cf=1 and AH > 0 -> error (only for COM1 and COM2)

;

; Original UNIX V1 'putc':

; put a character at the end of character list

;

; ((Modified registers: AX, BX, CX, DX, SI, DI))

;

cmp ah, 7

ja short sndc ; send character

write\_tty:

; 01/02/2014

; 18/01/2014

; 12/12/2013

; 04/12/2013

; 03/12/2013

; (Modified registers: AX, BX, CX, DX, SI, DI)

RVRT equ 00001000b ; VIDEO VERTICAL RETRACE BIT

RHRZ equ 00000001b ; VIDEO HORIZONTAL RETRACE BIT

; mov bl, 07h

; Derived from "WRITE\_TTY" procedure of IBM "pc-at" rombios source code

; (06/10/1985), 'video.asm', INT 10H, VIDEO\_IO

;

; 06/10/85 VIDEO DISPLAY BIOS

;

;--- WRITE\_TTY ------------------------------------------------------------------

; :

; THIS INTERFACE PROVIDES A TELETYPE LIKE INTERFACE TO THE :

; VIDEO CARDS. THE INPUT CHARACTER IS WRITTEN TO THE CURRENT :

; CURSOR POSITION, AND THE CURSOR IS MOVED TO THE NEXT POSITION. :

; IF THE CURSOR LEAVES THE LAST COLUMN OF THE FIELD, THE COLUMN :

; IS SET TO ZERO, AND THE ROW VALUE IS INCREMENTED. IF THE ROW :

; ROW VALUE LEAVES THE FIELD, THE CURSOR IS PLACED ON THE LAST ROW, :

; FIRST COLUMN, AND THE ENTIRE SCREEN IS SCROLLED UP ONE LINE. :

; WHEN THE SCREEN IS SCROLLED UP, THE ATTRIBUTE FOR FILLING THE :

; NEWLY BLANKED LINE IS READ FROM THE CURSOR POSITION ON THE PREVIOUS :

; LINE BEFORE THE SCROLL, IN CHARACTER MODE. IN GRAPHICS MODE, :

; THE 0 COLOR IS USED. :

; ENTRY -- :

; (AH) = CURRENT CRT MODE :

; (AL) = CHARACTER TO BE WRITTEN :

; NOTE THAT BACK SPACE, CARRIAGE RETURN, BELL AND LINE FEED ARE :

; HANDLED AS COMMANDS RATHER THAN AS DISPLAY GRAPHICS CHARACTERS :

; (BL) = FOREGROUND COLOR FOR CHAR WRITE IF CURRENTLY IN A GRAPHICS MODE :

; EXIT -- :

; ALL REGISTERS SAVED :

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

;;push ax ; save character and video page number

;;mov bh, ah ; get page setting

;;mov ah, 03h ; (read cursor position)

;;int 10h

;;pop ax ; recover character and video page

cli

; READ CURSOR (04/12/2013)

xor bh, bh

mov bl, ah

shl bl, 1

add bx, offset cursor\_posn

mov dx, word ptr [BX]

;mov cx, word ptr [cursor\_mode]

;

;mov bl, 07h ;

;mov bh, ah ;

mov bl, ah ; video page number

;xor bh, bh

; dx now has the current cursor position

cmp al, 0Dh ; is it carriage return or control character

jbe short u8

; write the char to the screen

u0:

;mov ah, 0Ah ; write character only command

;mov cx, 1 ; only one character

;int 10h ; write the character

mov ah, 07h ; attribute/color

; al = character

; bl = video page number (0 to 7)

;

call write\_c\_current

; position the cursor for next char

inc dl

cmp dl, 80 ; test for column overflow

;jne short u7

jne set\_cpos

mov dl, 0

cmp dh, 25-1 ; check for last row

jne short u6

; scroll required

u1:

;;mov ah, 02h

;;int 10h ; set the cursor

; SET CURSOR POSITION (04/12/2013)

call set\_cpos

; determine value to fill with during scroll

u2:

;;mov ah, 08h ; get read cursor command

;;int 10h ; read char/attr at current cursor

; READ\_AC\_CURRENT :

; THIS ROUTINE READS THE ATTRIBUTE AND CHARACTER

; AT THE CURRENT CURSOR POSITION

;

; INPUT

; (AH) = CURRENT CRT MODE

; (BH) = DISPLAY PAGE ( ALPHA MODES ONLY )

; (DS) = DATA SEGMENT

; (ES) = REGEN SEGMENT

; OUTPUT

; (AL) = CHARACTER READ

; (AH) = ATTRIBUTE READ

; mov ah, byte ptr [crt\_mode] ; move current mode into ah

;

; bl = video page number

;

call find\_position ; get regen location and port address

; dx = status port

;mov si, di ; establish addressing in si

; si = cursor location/address

;push es ; get regen segment for quick access

;pop ds

p11:

sti ; enable interrupts

nop ; allow for small interupts window

cli ; blocks interrupts for single loop

in al, dx ; get status from adapter

test al, RHRZ ; is horizontal retrace low

jnz short p11 ; wait until it is

;

p12: ; now wait for either retrace high

in al, dx ; get status

test al, RVRT+RHRZ ; is horizontal or vertical retrace high

jz short p12 ; wait until either is active

p13:

;lodsw ; get the character and attribute

;

push ds

mov ax, 0B800h

mov ds, ax

mov ax, word ptr [SI]

pop ds

;

; al = character, ah = attribute

;

sti

mov bh, ah ; store in bh

; bl = video page number

u3:

;;mov ax, 0601h ; scroll one line

;;sub cx, cx ; upper left corner

;;mov dh, 25-1 ; lower right row

;mov dl, 80 ; lower right column

;dec dl

;;mov dl, 79

;call scroll\_up ; 04/12/2013

mov al, 1

jmp scroll\_up

;u4:

;;int 10h ; video-call return

; scroll up the screen

; tty return

;u5:

;retn ; return to the caller

u6: ; set-cursor-inc

inc dh ; next row

; set cursor

;u7:

;;mov ah, 02h

;;jmp short u4 ; establish the new cursor

;call set\_cpos

;jmp short u5

jmp set\_cpos

; check for control characters

u8:

je short u9

cmp al, 0Ah ; is it a line feed (0Ah)

je short u10

cmp al, 07h ; is it a bell

je short u11

cmp al, 08h ; is it a backspace

;jne short u0

je short bs ; 12/12/2013

; 12/12/2013 (tab stop)

cmp al, 09h ; is it a tab stop

jne short u0

mov al, dl

cbw

mov cl, 8

div cl

sub cl, ah

ts:

push cx

mov al, 20h

call write\_tty

pop cx

dec cl

jnz short ts

retn

bs:

; back space found

or dl, dl ; is it already at start of line

;je short u7 ; set\_cursor

jz short set\_cpos

dec dx ; no -- just move it back

;jmp short u7

jmp short set\_cpos

; carriage return found

u9:

mov dl, 0 ; move to first column

;jmp short u7

jmp short set\_cpos

; line feed found

u10:

cmp dh, 25-1 ; bottom of screen

jne short u6 ; no, just set the cursor

jmp short u1 ; yes, scroll the screen

beeper: ; 18/01/2014 (sti)

; 17/01/2014 (call from 'kb\_int')

;sti

; bell found

u11:

sti ; 01/02/2014

; 12/12/2013

cmp bl, byte ptr [active\_page]

jne short @f ; Do not sound the beep

; if it is not written on the active page

mov cx, 1331 ; divisor for 896 hz tone

mov bl, 31 ; set count for 31/64 second for beep

;call beep ; sound the pod bell

;jmp short u5 ; tty\_return

;retn

TIMER equ 040h ; 8254 TIMER - BASE ADDRESS

PORT\_B equ 061h ; PORT B READ/WRITE DIAGNOSTIC REGISTER

GATE2 equ 00000001b ; TIMER 2 INPUT CATE CLOCK BIT

SPK2 equ 00000010b ; SPEAKER OUTPUT DATA ENABLE BIT

beep:

; 18/01/2014

; 10/12/2013

; 07/12/2013 (sti)

; 03/12/2013

;

; TEST4.ASM - 06/10/85 POST AND BIOS UTILITY ROUTINES

;

; ROUTINE TO SOUND THE BEEPER USING TIMER 2 FOR TONE

;

; ENTRY:

; (BL) = DURATION COUNTER ( 1 FOR 1/64 SECOND )

; (CX) = FREQUENCY DIVISOR (1193180/FREQUENCY) (1331 FOR 886 HZ)

; EXIT: :

; (AX),(BL),(CX) MODIFIED.

pushf ; 18/01/2014 ; save interrupt status

cli ; block interrupts during update

mov al, 10110110b ; select timer 2, lsb, msb binary

out TIMER+3, al ; write timer mode register

jmp $+2 ; I/O delay

mov al, cl ; divisor for hz (low)

out TIMER+2,AL ; write timer 2 count - lsb

jmp $+2 ; I/O delay

mov al, ch ; divisor for hz (high)

out TIMER+2, al ; write timer 2 count - msb

in al, PORT\_B ; get current setting of port

mov ah, al ; save that setting

or al, GATE2+SPK2 ; gate timer 2 and turn speaker on

out PORT\_B, al ; and restore interrupt status

;popf ; 18/01/2014

sti

g7: ; 1/64 second per count (bl)

mov cx, 1035 ; delay count for 1/64 of a second

call waitf ; go to beep delay 1/64 count

dec bl ; (bl) length count expired?

jnz short g7 ; no - continue beeping speaker

;

;pushf ; save interrupt status

cli ; 18/01/2014 ; block interrupts during update

in al, PORT\_B ; get current port value

or al, not (GATE2+SPK2) ; isolate current speaker bits in case

and ah, al ; someone turned them off during beep

mov al, ah ; recover value of port

or al, not (GATE2+SPK2) ; force speaker data off

out PORT\_B, al ; and stop speaker timer

;popf ; restore interrupt flag state

sti

mov cx, 1035 ; force 1/64 second delay (short)

call waitf ; minimum delay between all beeps

;pushf ; save interrupt status

cli ; block interrupts during update

in al, PORT\_B ; get current port value in case

and al, GATE2+SPK2 ; someone turned them on

or al, ah ; recover value of port\_b

out PORT\_B, al ; restore speaker status

popf ; restore interrupt flag state

@@:

retn

REFRESH\_BIT equ 00010000b ; REFRESH TEST BIT

waitf:

; 03/12/2013

;

; TEST4.ASM - 06/10/85 POST AND BIOS UTILITY ROUTINES

;

; WAITF - FIXED TIME WAIT ROUTINE HARDWARE CONTROLLED - NOT PROCESSOR

;

; ENTRY:

; (CX) = COUNT OF 15.,085737 MICROSECOND INTERVALS TO WAIT

; MEMORY REFRESH TIMER 1 OUTPUT USED AS REFERENCE

; EXIT:

; AFTER (CX) TIME COUNT (PLUS OR MINUS 16 MICROSECONDS)

; (CX) = 0

; delay for (cx)\*15.085737 us

push ax ; save work register (ah)

waitf1:

; use timer 1 output bits

in al, PORT\_B ; read current counter output status

and al, REFRESH\_BIT ; mask for refresh determine bit

cmp al, ah ; did it just change

je short waitf1 ; wait for a change in output line

mov ah, al ; save new lflag state

loop waitf1 ; decrement half cycles till count end

pop ax ; restore (ah)

retn ; return (cx)=0

set\_cpos:

; 01/09/2014

; 12/12/2013

; 10/12/2013

; 04/12/2013

;

; VIDEO.ASM - 06/10/85 VIDEO DISPLAY BIOS

;

; SET\_CPOS

; THIS ROUTINE SETS THE CURRENT CURSOR POSITION TO THE

; NEW X-Y VALUES PASSED

; INPUT

; DX - ROW,COLUMN OF NEW CURSOR

; BH - DISPLAY PAGE OF CURSOR

; OUTPUT

; CURSOR ID SET AT 6845 IF DISPLAY PAGE IS CURRENT DISPLAY

;mov al, bh ; move page number to work register

mov al, bl ; page number

cbw ; convert page to word value

mov si, ax ; ah = 0, al = video page number

shl si, 1 ; word offset

mov word ptr [SI + offset cursor\_posn], dx ; save the pointer

; 01/09/2014

cmp byte ptr [active\_page], bl ; al

jne short m17

;

mov ax, dx ; get row/column to ax

;call m18 ; CURSOR SET

;m17: ; SET\_CPOS\_RETURN

; 01/09/2014

; retn

m18:

call position ; determine location in regen buffer

mov cx, ax

; 01/09/2014

add cx, word ptr [crt\_start]

; add in the start address for this page

;sar cx, 1

shr cx, 1 ; divide by 2 for char only count

mov ah, 14 ; register number for cursor

;call m16 ; output value to the 6845

;retn

;----- THIS ROUTINE OUTPUTS THE CX REGISTER

; TO THE 6845 REGISTERS NAMED IN (AH)

m16:

cli

;mov dx, word ptr [addr\_6845] ; address register

mov dx, 03D4h ; I/O address of color card

mov al, ah ; get value

out dx, al ; register set

inc dx ; data register

jmp $+2 ; i/o delay

mov al, ch ; data

out dx, al

dec dx

mov al, ah

inc al ; point to other data register

out dx, al ; set for second register

inc dx

jmp $+2 ; i/o delay

mov al, cl ; second data value

out dx, al

m17:

; 01/09/2014

retn

position:

; 04/12/2013

;

; VIDEO.ASM - 06/10/85 VIDEO DISPLAY BIOS

;

; POSITION

; THIS SERVICE ROUTINE CALCULATES THE REGEN BUFFER ADDRESS

; OF A CHARACTER IN THE ALPHA MODE

; INPUT

; AX = ROW, COLUMN POSITION

; OUTPUT

; AX = OFFSET OF CHAR POSITION IN REGEN BUFFER

push bx ; save register

mov bl, al

mov al, ah ; rows to al

;mul byte ptr [crt\_cols] ; determine bytes to row

mov bh, 80

mul bh

xor bh, bh

add ax, bx ; add in column value

;sal ax, 1

shl ax, 1 ; \* 2 for attribute bytes

pop bx

retn

find\_position:

; VIDEO.ASM - 06/10/85 VIDEO DISPLAY BIOS

mov cl, bl ; video page number

xor ch, ch

mov si, cx ; ch = 0, cl = video page number

shl si, 1

mov ax, word ptr [SI + Offset cursor\_posn]

jz short p21

;

xor si, si ; else set buffer address to zero

;

p20:

;add si, word ptr [crt\_len] ; add length of buffer for one page

add si, 80\*25\*2

loop p20

p21:

and ax, ax

jz short @f

call position ; determine location in regen in page

add si, ax ; add location to start of regen page

@@:

;mov dx, word ptr [addr\_6845] ; get base address of active display

;mov dx, 03D4h ; I/O address of color card

;add dx, 6 ; point at status port

mov dx, 03DAh

; cx = 0

retn

scroll\_up:

; 04/04/2014 (BugFix)

; 12/12/2013

; 04/12/2013

;

; VIDEO.ASM - 06/10/85 VIDEO DISPLAY BIOS

;

; SCROLL UP

; THIS ROUTINE MOVES A BLOCK OF CHARACTERS UP

; ON THE SCREEN

; INPUT

; (AH) = CURRENT CRT MODE

; (AL) = NUMBER OF ROWS TO SCROLL

; (CX) = ROW/COLUMN OF UPPER LEFT CORNER

; (DX) = ROW/COLUMN OF LOWER RIGHT CORNER

; (BH) = ATTRIBUTE TO BE USED ON BLANKED LINE

; (DS) = DATA SEGMENT

; (ES) = REGEN BUFFER SEGMENT

; OUTPUT

; NONE -- THE REGEN BUFFER IS MODIFIED

;

; ((ah = 3))

; dl = 79

; dh = 24

;

; al = line count (0 or 1) ((0 == clear video page))

; ((al = 1 for write\_tty (putc) procedure))

; bl = video page number (0 to 7)

; bh = attribute to be used on blanked line

;cli

push ax

cmp bl, byte ptr [active\_page]

je short n0

xor si, si

and bl, bl

jz short n9

mov cl, bl

@@:

add si, 25\*80\*2 ; 04/04/2014

dec cl

jnz short @b

jmp short n9

n0:

mov si, word ptr [crt\_start]

n1: ; 04/04/2014

;mov di, si

;

;inc dh

;inc dl ; increment for origin

; dl = 80

; dh = 25

;cmp bl, byte ptr [active\_page]

;jne short n9

;

mov dx, 3DAh ; guaranteed to be color card here

n8: ; wait\_display\_enable

in al, dx ; get port

test al, RVRT ; wait for vertical retrace

jz short n8 ; wait\_display\_enable

mov al, 25h

mov dl, 0D8h ; address control port

out dx, al ; turn off video during vertical retrace

n9:

pop cx ; al = line count

;

mov di, si ; 04/04/2014

;

push es

push ds

mov ax, 0B800h

mov es, ax

mov ds, ax

;

and cl, cl

jnz short @f

; clear video page

mov cx, 25 \* 80

jmp short n3

@@:

;mov ax, 160

; mov al, 160 ; 2 \* (80 columns)

; mul cl

;add si, ax

add si, 160

; ;mov cx, 24

;n2: ; row loop

; ;call n10 ; move one row

; ;add si, ax

; ;add di, ax

; ;loop n2

; mov al, cl

; mov cl, 25

; sub cl, al

; xor ch, ch

; ; cx = line count to move

;@@:

; push cx

n10:

;mov cx, 80

mov cx, 24\*80 ; 24 rows/lines

rep movsw ; move one line (up)

;loop n2

; pop cx

; loop @b

; mov cl, al

mov cl, 80

n3: ; clear entry

mov ah, bh ; attribute in ah

mov al, 20h ; fill with blanks

; cx = word count to clear (80 or 25\*80)

;@@:

; push cx

n11:

; mov cl, 80 ; get # of columns to clear

rep stosw ; store the fill character

; pop cx

; loop @b

n5: ; SCROLL\_END

pop ds

cmp bl, byte ptr [active\_page]

jne short @f

;mov al, byte ptr [crt\_mode\_set] ; get the value of mode set

mov al, 29h ; (ORGS.ASM), M7 mode set table value for mode 3

mov dx, 03D8h ; always set color card port

out dx, al

@@:

pop es

;sti

retn

write\_c\_current:

; 18/01/2014

; 04/12/2013

;

; VIDEO.ASM - 06/10/85 VIDEO DISPLAY BIOS

;

; WRITE\_C\_CURRENT

; THIS ROUTINE WRITES THE CHARACTER AT

; THE CURRENT CURSOR POSITION, ATTRIBUTE UNCHANGED

; INPUT

; (AH) = CURRENT CRT MODE

; (BH) = DISPLAY PAGE

; (CX) = COUNT OF CHARACTERS TO WRITE

; (AL) = CHAR TO WRITE

; (DS) = DATA SEGMENT

; (ES) = REGEN SEGMENT

; OUTPUT

; DISPLAY REGEN BUFFER UPDATED

cli

; bl = video page

; al = character

; ah = color/attribute

push dx

push ax ; save character & attribute/color

call find\_position ; get regen location and port address

; si = regen location

; dx = status port

;

; WAIT FOR HORIZONTAL RETRACE OR VERTICAL RETRACE

;

p41: ; wait for horizontal retrace is low or vertical

sti ; enable interrupts first

cmp bl, byte ptr [active\_page]

jne short p44 ; 18/01/2014

cli ; block interrupts for single loop

in al, dx ; get status from the adapter

test al, RVRT ; check for vertical retrace first

jnz short p43 ; Do fast write now if vertical retrace

test al, RHRZ ; is horizontal retrace low

jnz short p41 ; wait until it is

p42: ; wait for either retrace high

in al, dx ; get status again

test al, RVRT+RHRZ ; is horizontal or vertical retrace high

jz short p42 ; wait until either retrace active

p43: ; 18/01/2014

sti

p44:

pop ax ; restore the character (al) & attribute (ah)

push ds

mov cx, 0B800h

mov ds, cx

mov word ptr [SI], ax

pop ds

pop dx

retn

tty\_sw:

mov byte ptr [u.quant], 0 ; 04/03/2014

;

;act\_disp\_page:

; 30/06/2015

; 04/03/2014 (act\_disp\_page --> tty\_sw)

; 10/12/2013

; 04/12/2013

;

; VIDEO.ASM - 06/10/85 VIDEO DISPLAY BIOS

;

; ACT\_DISP\_PAGE

; THIS ROUTINE SETS THE ACTIVE DISPLAY PAGE, ALLOWING

; THE FULL USE OF THE MEMORY SET ASIDE FOR THE VIDEO ATTACHMENT

; INPUT

; AL HAS THE NEW ACTIVE DISPLAY PAGE

; OUTPUT

; THE 6845 IS RESET TO DISPLAY THAT PAGE

;cli

;push bx

push cx

push dx

;

mov byte ptr [active\_page], al ; save active page value ; [ptty]

;mov cx, word ptr [crt\_len] ; get saved length of regen buffer

mov cx, 25\*80\*2

cbw ; convert AL to word

push ax ; save page value

mul cx ; display page times regen length

; 10/12/2013

mov word ptr [crt\_start], ax ; save start address for later

mov cx, ax ; start address to cx

;sar cx, 1

shr cx, 1 ; divide by 2 for 6845 handling

mov ah, 12 ; 6845 register for start address

call m16

pop bx ; recover page value

;sal bx, 1

shl bx, 1 ; \*2 for word offset

mov ax, word ptr [BX + offset cursor\_posn] ; get cursor for this page

call m18

;

pop dx

pop cx

;pop bx

;

;sti

;

retn

get\_cpos:

; 04/12/2013 (sysgtty)

;

; INPUT -> bl = video page number

; RETURN -> dx = cursor position

push bx

xor bh, bh

shl bl, 1

add bx, offset cursor\_posn

mov dx, word ptr [BX]

pop bx

retn

read\_ac\_current:

; 04/12/2013 (sysgtty)

;

; INPUT -> bl = video page number

; RETURN -> ax = character (al) and attribute (ah)

call find\_position

push ds

mov ax, 0B800h

mov ds, ax

mov ax, word ptr [SI]

pop ds

retn

; 11/06/2014

; Retro UNIX 8086 v1 feature only

; (INPUT -> none)

syssleep:

mov bl, byte ptr [u.uno] ; process number

xor bh, bh

mov ah, byte ptr [BX]+p.ttyc-1 ; current/console tty

call sleep

jmp sysret

; COMMENT $

; 28/02/2014

; Keyboard function variables (for INT 16h)

; DS = 40h

;;DDSDATA equ 40h

;

;;KB\_FLAG equ 17h ; byte

;;;KB\_FLAGS equ 17h ; word ; initial value = 0

;;BUFF\_HEAD equ 1Ah ; word ; initial value = offset KB\_BUFF

;;BUFF\_TAIL equ 1Ch ; word ; initial value = offset KB\_BUFF

;;BUFF\_START equ 80h ; word ; initial value = offset KB\_BUFF

;;BUFF\_END equ 82h ; word ; initial value = offset KB\_BUFF + 32

;;;KB\_BUFF equ 1Eh ; 32 bytes ; Keyboard buffer (circular queue buffer)

; 03/03/2014

BIOS\_DSEGM equ 40h

RESET\_FLAG equ 72h ; WORD=1234H IF KEYBOARD RESET UNDERWAY

; (40h:72h)

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

; VIDEO DISPLAY DATA AREA ;

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

CRT\_MODE equ 49h ; CURRENT DISPLAY MODE (TYPE)

CRT\_MODE\_SET equ 65h ; CURRENT SETTING OF THE 3X8 REGISTER

;--------- 8042 COMMANDS -------------------------------------------------------

ENA\_KBD equ 0AEh ; ENABLE KEYBOARD COMMAND

DIS\_KBD equ 0ADh ; DISABLE KEYBOARD COMMAND

;--------- 8042 KEYBOARD INTERFACE AND DIAGNOSTIC CONTROL REGISTERS ------------

STATUS\_PORT equ 064h ; 8042 STATUS PORT

INPT\_BUF\_FULL equ 00000010b ; 1 = +INPUT BUFFER FULL

PORT\_A equ 060h ; 8042 KEYBOARD SCAN CODE/CONTROL PORT

;---------- 8042 KEYBOARD RESPONSE ---------------------------------------------

KB\_ACK equ 0FAh ; ACKNOWLEDGE PROM TRANSMISSION

KB\_RESEND equ 0FEh ; RESEND REQUEST

KB\_OVER\_RUN equ 0FFh ; OVER RUN SCAN CODE

;---------- KEYBOARD/LED COMMANDS ----------------------------------------------

KB\_ENABLE equ 0F4h ; KEYBOARD ENABLE

LED\_CMD EQU 0EDH ; LED WRITE COMMAND

;---------- KEYBOARD SCAN CODES ------------------------------------------------

ID\_1 equ 0ABh ; 1ST ID CHARACTER FOR KBX

ID\_2 equ 041h ; 2ND ID CHARACTER FOR KBX

ALT\_KEY equ 56 ; SCAN CODE FOR ALTERNATE SHIFT KEY

CTL\_KEY equ 29 ; SCAN CODE FOR CONTROL KEY

CAPS\_KEY equ 58 ; SCAN CODE FOR SHIFT LOCK KEY

DEL\_KEY equ 83 ; SCAN CODE FOR DELETE KEY

INS\_KEY equ 82 ; SCAN CODE FOR INSERT KEY

LEFT\_KEY equ 42 ; SCAN CODE FOR LEFT SHIFT

NUM\_KEY equ 69 ; SCAN CODE FOR NUMBER LOCK KEY

RIGHT\_KEY equ 54 ; SCAN CODE FOR RIGHT SHIFT

SCROLL\_KEY equ 70 ; SCAN CODE FOR SCROLL LOCK KEY

SYS\_KEY equ 84 ; SCAN CODE FOR SYSTEM KEY

;---------- FLAG EQUATES WITHIN @KB\_FLAG----------------------------------------

RIGHT\_SHIFT equ 00000001b ; RIGHT SHIFT KEY DEPRESSED

LEFT\_SHIFT equ 00000010b ; LEFT SHIFT KEY DEPRESSED

CTL\_SHIFT equ 00000100b ; CONTROL SHIFT KEY DEPRESSED

ALT\_SHIFT equ 00001000b ; ALTERNATE SHIFT KEY DEPRESSED

SCROLL\_STATE equ 00010000b ; SCROLL LOCK STATE HAS BEEN TOGGLED

NUM\_STATE equ 00100000b ; NUM LOCK STATE HAS BEEN TOGGLED

CAPS\_STATE equ 01000000b ; CAPS LOCK STATE HAS BEEN TOGGLED

INS\_STATE equ 10000000b ; INSERT STATE IS ACTIVE

;---------- FLAG EQUATES WITHIN @KB\_FLAG\_1 -------------------------------------

SYS\_SHIFT equ 00000100b ; SYSTEM KEY DEPRESSED AND HELD

HOLD\_STATE equ 00001000b ; SUSPEND KEY HAS BEEN TOGGLED

SCROLL\_SHIFT equ 00010000b ; SCROLL LOCK KEY IS DEPRESSED

NUM\_SHIFT equ 00100000b ; NUM LOCK KEY IS DEPRESSED

CAPS\_SHIFT equ 01000000b ; CAPS LOCK KEY IS DEPRE55ED

INS\_SHIFT equ 10000000b ; INSERT KEY IS DEPRESSED

;---------- FLAGS EQUATES WITHIN @KB\_FLAG\_2 -----------------------------------

KB\_LEDS equ 00000111b ; KEYBOARD LED STATE BITS

; equ 00001000b ; RESERVED (MUST BE ZERO)

KB\_FA equ 00010000b ; ACKNOWLEDGMENT RECEIVED

KB\_FE equ 00100000b ; RESEND RECEIVED FLAG

KB\_PR\_LED equ 01000000b ; MODE INDICATOR UPDATE

KB\_ERR equ 10000000b ; REYBOARD TRANSMIT ERROR FLAG

;----------- FLAGS EQUATES WITHIN @KB\_FLAG\_3 -----------------------------------

KBX equ 00000001b ; KBX INSTALLED

LC\_HC equ 00000010b ; LAST SCAN CODED WAS A HIDDEN CODE

GRAPH\_ON equ 00000100b ; ALL GRAPHICS KEY DOWN (W.T. ONLY)

; equ 00011000b ; RESERVED (MUST BE ZERO)

SET\_NUM\_LK equ 00100000b ; FORCE NUM LOCK IF READ ID AND KBX

LC\_AB equ 01000000b ; LAST CHARACTER WAS FIRST ID CHARACTER

RD\_ID equ 10000000b ; DOING A READ ID (MUST BE BIT0)

;

;----- THIS CODE CONTAINS THE KBX SUPPORT FOR INT 09H

; EQUATES

F11\_M equ 217 ; FUNC 11 MAKE

F11\_B equ 215 ; FUNC 11 BREAK

F12\_M equ 218 ; FUNC 12 MAKE

F12\_B equ 216 ; FUNC 12 BREAK

K102\_M equ 86 ; KEY 102 MAKE

K102\_B equ 214 ; KEY 102 BREAK

;

INS\_M equ 82 ; INSERT KEY MAKE

DEL\_M equ 83 ; DELETE KEY MAKE

LEFT\_M equ 75 ; CURSOR LEFT MAKE

RIGHT\_M equ 77 ; CURSOR RIGHT MARE

UP\_M equ 72 ; CURSOR UP MAKE

DN\_M equ 80 ; CURSOR DOWN MAKE

PGUP\_M equ 73 ; PG UP MAKE

PGDN\_M equ 81 ; PG DN MAKE

HOME\_M equ 71 ; HOME MAKE

END\_M equ 79 ; END MAKE

;

FUNC11 equ 133 ; FUNCTION 11 KEY

HC equ 224 ; HIDDEN CODE

;----------- INTERRUPT EQUATES -------------------------------------------------

EOI equ 020h ; END OF INTERRUPT COMMAND TO 8259

INTA00 equ 020h ; 8259 PORT

int\_16h:

; 28/08/2014

; 30/06/2014

; 03/03/2014

; 28/02/2014

; Derived from "KEYBOARD\_IO\_1" procedure of IBM "pc-at"

; rombios source code (06/10/1985)

; 'keybd.asm', INT 16H, KEYBOARD\_IO

;

; 06/10/85 KEYBOARD BIOS

;

;--- INT 16 H -------------------------------------------------------------------

; KEYBOARD I/O :

; THESE ROUTINES PROVIDE READ KEYBOARD SUPPORT :

; INPUT :

; (AH)= 00H READ THE NEXT ASCII CHARACTER ENTERED FROM THE KEYBOARD, :

; RETURN THE RESULT IN (AL), SCAN CODE IN (AH). :

; :

; (AH)= 01H SET THE ZERO FLAG TO INDICATE IF AN ASCII CHARACTER IS :

; AVAILABLE TO BE READ FROM THE KEYBOARD BUFFER. :

; (ZF)= 1 -- NO CODE AVAILABLE :

; (ZF)= 0 -- CODE IS AVAILABLE (AX)= CHARACTER :

; IF (ZF)= 0, THE NEXT CHARACTER IN THE BUFFER TO BE READ IS:

; IN (AX), AND THE ENTRY REMAINS IN THE BUFFER. :

; (AH)= 02H RETURN THE CURRENT SHIFT STATUS IN (AL) REGISTER :

; THE BIT SETTINGS FOR THIS CODE ARE INDICATED IN THE :

; EQUATES FOR @KB\_FLAG :

; OUTPUT:

; AS NOTED ABOVE, ONLY (AX) AND FLAGS CHANGED :

; ALL REGISTERS RETAINED :

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

sti

push ds ; SAVE CURRENT DS

push bx ; SAVE BX TEMPORARILY

mov bx, cs

mov ds, bx ; PUT SEGMENT VALUE OF DATA AREA INTO DS

or ah, ah ; CHECK FOR (AH)= 00H

jz short k1b ; ASCII\_READ

;

dec ah

jz short k2 ; CHECK FOR (AH)= 01H

; ASCII\_STATUS

dec ah ; CHECK FOR (AH)= 02H

jz short k3 ; SHIFT STATUS

pop bx ; RECOVER REGISTER

pop ds ; RECOVER SEGMENT

iret ; INVALID COMMAND EXIT

;----- READ THE KEY TO FIGURE OUT WHAT TO DO

k1b:

mov bx, word ptr [BUFFER\_HEAD] ; GET POINTER TO HEAD OF BUFFER

cmp bx, word ptr [BUFFER\_TAIL] ; TEST END OF BUFFER

;; 28/08/2014

;;jne short k1c ; IF ANYTHING IN BUFFER SKIP INTERRUPT

jne short k1d

;;mov ax, 09002h ; MOVE IN WAIT CODE A TYPE

;;int 15h ; PERFORM OTHER FUNCTION

k1: ; ASCII READ

sti ; INTERRUPTS BACK ON DURING LOOP

nop ; ALLOW AN INTERRUPT TO OCCUR

k1c: cli ; INTERRUPTS BACK OFF

mov bx, word ptr [BUFFER\_HEAD] ; GET POINTER TO HEAD OF BUFFER

cmp bx, word ptr [BUFFER\_TAIL] ; TEST END OF BUFFER

k1d:

; 30/06/2014 (original code again)

push bx ; SAVE ADDRESS

pushf ; SAVE FLAGS

call make\_led ; GO GET MODE INDICATOR DATA BYTE

mov bl, byte ptr [KB\_FLAG\_2] ; GET PREVIOUS BITS

xor bl, al ; SEE IF ANY DIFFERENT

and bl, KB\_LEDS ; ISOLATE INDICATOR BITS

jz short k1a ; IF NO CHANGE BYPASS UPDATE

call snd\_led1

cli

k1a:

popf ; RESTORE FLAGS

pop bx ; RESTORE ADDRESS

jz short k1 ; LOOP UNTIL SOMETHING IN BUFFER

;

mov ax, word ptr [BX] ; GET SCAN CODE AND ASCII CODE

call k4 ; MOVE POINTER TO NEXT POSITION

; 03/03/2014

mov word ptr [BUFFER\_HEAD], bx ; STORE VALUE IN VARIABLE

pop bx ; RECOVER REGISTER

pop ds ; RECOVER SEGMENT

iret ; RETURN TO CALLER

;----- ASCII STATUS

k2:

cli ; INTERRUPTS OFF

mov bx, word ptr [BUFFER\_HEAD] ; GET HEAD POINTER

cmp bx, word ptr [BUFFER\_TAIL] ; IF EQUAL (Z=1) THEN NOTHING THERE

mov ax, word ptr [BX]

; 30/06/2014 (original code again)

pushf ; SAVE FLAGS

push ax ; SAVE CODE

call make\_led ; GO GET MODE INDICATOR DATA BYTE

mov bl, byte ptr [KB\_FLAG\_2] ; GET PREVIOUS BITS

xor bl, al ; SEE IF ANY DIFFERENT

and bl, KB\_LEDS ; ISOLATE INDICATOR BITS

jz short sk2 ; IF NO CHANGE BYPASS UPDATE

;

call snd\_led1

sk2:

pop ax ; RESTORE CODE

popf ; RESTORE FLAGS

sti ; INTERRUPTS BACK ON

pop bx ; RECOVER REGISTER

pop ds ; RECOVER SEGMENT

retf 2 ; THROW AWAY FLAGS

;----- SHIFT STATUS

k3:

mov al, byte ptr [KB\_FLAG] ; GET THE SHIFT STATUS FLAGS

pop bx ; RECOVER REGISTERS

pop ds

iret ; RETURN TO CALLER

; 03/03/2014

;----- INCREMENT A BUFFER POINTER

k4: inc bx

inc bx ; MOVE TO NEXT WORD IN LIST

cmp bx, word ptr [BUFFER\_END] ; AT END OF BUFFER?

;jne short k5 ; NO, CONTINUE

jb short k5

mov bx, word ptr [BUFFER\_START] ; YES, RESET TO BUFFER BEGINNING

k5:

retn

int\_09h:

; 07/03/2014

; 03/03/2014

; Derived from "KEYBOARD\_INT\_1" procedure of IBM "pc-at"

; rombios source code (06/10/1985)

; 'keybd.asm', INT 16H, KEYBOARD\_IO

;

; 06/10/85 KEYBOARD BIOS

;

;--- HARDWARE INT 09 H - ( IRQ LEVEL 1 )-----------------------------------------

;

; KEYBOARD INTERRUPT ROUTINE

;

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

sti ; ENABLE INTERRUPTS

push bp

push ax

push bx

push cx

push dx

push si

push di

push ds

push es

cld ; FORWARD DIRECTION

;call dds ; SET UP ADDRESSING

;mov ax, offset DDSData ;

mov ax, cs

mov ds, ax

mov es, ax

;

;----- WAIT FOR KEYBOARD DISABLE COMMAND TO BE ACCEPTED

mov al, DIS\_KBD ; DISABLE THE KEYBOARD COMMAND

call ship\_it ; EXECUTE DISABLE

cli ; DISABLE INTERRUPTS

;sub cx, cx ; SET MAXIMUM TIMEOUT

xor cx, cx

kb\_int\_01:

in al, STATUS\_PORT ; READ ADAPTER STATUS

test al, INPT\_BUF\_FULL ; CHECK INPUT BUFFER FULL STATUS BIT

loopnz kb\_int\_01 ; WAIT FOR COMMAND TO BE ACCEPTED

;

;----- READ CHARACTER FROM KEYBOARD INTERFACE

in al, PORT\_A ; READ IN THE CHARACTER

;

;----- SYSTEM HOOK INT 15H - FUNCTION 4FH (ON HARDWARE INTERRUPT LEVEL 9HI

;mov ah, 04Fh ; SYSTEM INTERCEPT - KEY CODE FUNCTION

;stc ; SET CY= 1 (IN CASE OF IRET)

;int 15h ; CASSETTE CALL (AL)= KEY SCAN CODE

; RETURNS CY= 1 FOR INVALID FUNCTION

;jc short kb\_int\_02 ; CONTINUE IF CARRY FLAG SET ((AL)=CODE)

;

;jmp short k26 ; EXIT IF SYSTEM HANDLED SCAN CODE

; EXIT HANDLES HARDWARE EOI AND ENABLE

;jnc k26

;----- CHECK FOR A RESEND COMMAND TO KEYBOARD

kb\_int\_02: ; (AL)= SCAN CODE

sti ; ENABLE INTERRUPTS AGAIN

cmp al, KB\_RESEND ; IS THE INPUT A RESEND

je short kb\_int\_03 ; GO IF RESEND

;

;----- CHECK FOR RESPONSE TO A COMMAND TO KEYBOARD

cmp al, KB\_ACK ; IS THE INPUT AN ACKNOWLEDGE

jne short kb\_int\_04 ; GO IF NOT

;

;----- A COMMAND TO THE KEYBOARD WAS ISSUED

cli ; DISABLE INTERRUPTS

or byte ptr [KB\_FLAG\_2], KB\_FA ; INDICATE ACK RECEIVED

jmp k26 ; RETURN IF NOT (ACK RETURNED FOR DATA)

;

;----- RESEND THE LAST BYTE

kb\_int\_03:

cli ; DISABLE INTERRUPTS

or byte ptr [KB\_FLAG\_2], KB\_FE ; INDICATE RESEND RECEIVED

jmp k26 ; RETURN IF NOT ACK RETURNED FOR DATA)

kb\_int\_04:

;----- UPDATE MODE INDICATORS IF CHANGE IN STATE

push ax ; SAVE DATA IN

call make\_led ; GO GET MODE INDICATOR DATA BYTE

mov bl, byte ptr [KB\_FLAG\_2] ; GET PREVIOUS BITS

xor bl, al ; SEE IF ANY DIFFERENT

and bl, KB\_LEDS ; ISOLATE INDICATOR BITS

jz short up0 ; IF NO CHANGE BYPASS UPDATE

call snd\_led ; GO TURN ON MODE INDICATORS

up0: pop ax ; RESTORE DATA IN

mov ah, al ; SAVE SCAN CODE IN AH ALSO

;

;----- TEST FOR OVERRUN SCAN CODE FROM KEYBOARD

cmp al, KB\_OVER\_RUN ; IS THIS AN OVERRUN CHAR

;jne short k16 ; NO, TEST FOR SHIFT KEY

;jmp short k62 ; BUFFER\_FULL\_BEEP

je k62

k16:

and al, 07Fh ; REMOVE BREAK BIT

;push cs

;pop es ; ESTABLISH ADDRESS OF TABLES

;

test byte ptr [KB\_FLAG\_3], RD\_ID+LC\_AB ; ARE WE DOING A READ ID?

jz short not\_id ; CONTINUE IF NOT

jns short tst\_id\_2 ; IS THE RD\_ID FLAG ON?

cmp ah, ID\_1 ; IS THIS THE 1ST ID CHARACTER?

jne short rst\_rd\_id

or byte ptr [KB\_FLAG\_3], LC\_AB ; INDICATE 1ST ID WAS OK

rst\_rd\_id:

and byte ptr [KB\_FLAG\_3], NOT RD\_ID ; RESET THE READ ID FLAG

;jmp short do\_ext

jmp k26

tst\_id\_2:

and byte ptr [KB\_FLAG\_3], NOT LC\_AB ; RESET FLAG

cmp ah, ID\_2 ; IS THIS THE 2ND ID CHARACTER?

;jne short do\_ext ; LEAVE IF NOT

jne k26

;

;----- A READ ID SAID THAT IT WAS KBX

or byte ptr [KB\_FLAG\_3], KBX ; INDICATE KBX WAS FOUND

test byte ptr [KB\_FLAG\_3], SET\_NUM\_LK ; SHOULD WE SET NUM LOCK?

;jz short do\_ext ; EXIT IF NOT

jz k26

or byte ptr [KB\_FLAG], NUM\_STATE ; FORCE NUM LOCK ON

call snd\_led ; GO SET THE NUM LOCK INDICATOR

;jmp short exit

jmp k26

;

not\_id:

test byte ptr [KB\_FLAG\_3], LC\_HC ; WAS THE LAST CHARACTER A HIDDEN CODE

jz short not\_lc\_hc ; JUMP IF NOT

;

;----- THE LAST CHARACTER WAS A HIDDEN CODE

and byte ptr [KB\_FLAG\_3], NOT LC\_HC ; RESET LAST CHAR HIDDEN CODE FLAG

cmp al, INS\_M ; WAS IT THE INSERT KEY?

je short not\_i

test ah, 80h ; IS THIS A BREAK CODE

;jnz short exit ; IGNORE BREAK ON REST OF THESE KEYS

jnz k26

not\_i:

mov di, offset K\_TAB1 ; TEST FOR ONE OF THE KEYPAD CURSOR FUNC

mov cx, L\_TAB1

repne scasb ; SCAN FOR THE KEY

jne short not\_cur ; GO ON IF NOT FOUND

test byte ptr [KB\_FLAG\_1], HOLD\_STATE ; ARE WE IN HOLD STATE?

jz short n\_hld

and byte ptr [KB\_FLAG\_1], NOT HOLD\_STATE ; EXIT HOLD STATE

;do\_ext:

; jmp short exit ; IGNORE THIS KEY

jmp k26

n\_hld:

test byte ptr [KB\_FLAG], ALT\_SHIFT ; IS ALT DOWN?

jz short not\_alt

test byte ptr [KB\_FLAG], CTL\_SHIFT ; HOW ABOUT CTRL?

;jz short exit ; IGNORE ALL IF ONLY ALT DOWN

jz k26

cmp al, DEL\_M ; WAS IT THE DELETE KEY'

;jne short exit ; IGNORE IF NOT

jne k26

jmp k29 ; GO DO THE CTL, ALT, DEL RESET

;

not\_alt:

test byte ptr [KB\_FLAG], CTL\_SHIFT ; IS CTL DOWN?

jnz short ctl\_on ; SPECIAL CASE IF SO

cmp al, INS\_M ; IS THIS THE INSERT KEY?

;jne short n\_ins

jne k49

;

;----- SPECIAL HANDLING FOR INSERT KEY

mov al, ah ; RECOVER SCAN CODE

mov ah, INS\_SHIFT ; AH = MASK FOR INSERT

test al, 80h ; WAS THIS A BREAK CODE?

;jnz short b\_c

jnz k24

jmp k22 ; GO HANDLE INSERT SHIFT

;b\_c:

; jmp short k24 ; HANDLE BREAK

;n\_ins:

; jmp short k49 ; HANDLE & IGNORE NUMLOCK

ctl\_on:

cmp cl, 5 ; WAS IT INS, DEL, UP OR DOWN?

;ja short exit ; IGNORE IF DO

ja k26

jmp k42 ; GO HANDLE CTRL CASE

;

not\_lc\_hc: ; LAST CHARACTER WAS NOT A HIDDEN CODE

cmp ah, HC ; IS THIS CHARACTER A HIDDEN CODE?

jne short not\_cur

or byte ptr [KB\_FLAG\_3], LC\_HC+KBX ; SET LAST CHAR WAS A HIDDEN CODE & KBX

;exit:

jmp k26 ; THROW AWAY THIS CODE

;

not\_cur:

cmp ah, F11\_M ; WAS IT F11?

jne short t\_f12 ; HANDLE IF SO

mov cl, FUNC11 ; SET BASE FUNCTION 11

cmp ah, F11\_B ; IS THIS A BREAK CODE

;je short exit ; IGNORE SPEAK CODES

je k26

cmp ah, F12\_B ; IS THIS A BREAK CODE

;je short exit ; IGNORE BREAK CODES

je k26

jmp short do\_fn

t\_f12:

cmp ah, F12\_M ; WAS IT F12?

jne short t\_sys\_key ; GO TEST FOR SYSTEM KEY

mov cl, FUNC11+1 ; SET BASE FUNCTION 12

do\_fn:

test byte ptr [KB\_FLAG\_1], HOLD\_STATE ; ARE WE IN HOLD STATE?

jz short n\_hld1

and byte ptr [KB\_FLAG\_1], NOT HOLD\_STATE ; EXIT HOLD STATE

;jmp short exit ; IGNORE THIS KEY

je k26

n\_hld1:

mov ah, cl

;

test byte ptr [KB\_FLAG], ALT\_SHIFT ; ARE WE IN ALT

jz short t\_ctl

add ah, 6 ; CNVT TO ALT FN 11-12

jmp short set\_fn

t\_ctl:

test byte ptr [KB\_FLAG], CTL\_SHIFT ; ARE WE IN CTRL

jz short t\_shf

add ah, 4 ; CNVT TO CTRL FN 11-12

jmp short set\_fn

t\_shf:

test byte ptr [KB\_FLAG], LEFT\_SHIFT+RIGHT\_SHIFT ; IS EITHER SHIFT ON?

jz short set\_fn

add ah, 2 ; CNVT TO SHIFT FN 11-12

set\_fn:

sub al, al ; FORCE PSEUDO SCAN CODE

jmp k61 ; PUT IT INTO BUFFER

;

;----- TEST FOR SYSTEM KEY

t\_sys\_key:

cmp al, SYS\_KEY ; IS IT THE SYSTEM KEY?

jnz short k16a ; CONTINUE IF NOT

;

test ah, 80h ; CHECK IF THIS A BREAK CODE

jnz short k16c ; DO NOT TOUCH SYSTEM INDICATOR IF TRUE

;

test byte ptr [KB\_FLAG\_1], SYS\_SHIFT ; SEE IF IN SYSTEM KEY HELD DOWN

;jnz short k16b ; IF YES, DO NOT PROCESS SYSTEM INDICATOR

jnz k26

;

or byte ptr [KB\_FLAG\_1], SYS\_SHIFT ; INDICATE SYSTEM KEY DEPRESSED

mov al, EOI ; END OF INTERRUPT COMMAND

out INTA00, al ; SEND COMMAND TO INTERRUPT CONTROL PORT

; INTERRUPT-RETURN-NO-EOI

mov al, ENA\_KBD ; INSURE KEYBOARD 15 ENABLED

call ship\_it ; EXECUTE ENABLE

;mov ax, 8500h ; FUNCTION VALUE FOR MAKE OF SYSTEM KEY

;sti ; MAKE SURE INTERRUPTS ENABLED

;int 15h ; USER INTERRUPT

jmp k27a ; END PROCESSING

;k16b:

; jmp short k26 ; IGNORE SYSTEM KEY

k16c:

and byte ptr [KB\_FLAG\_1], NOT SYS\_SHIFT ; TURN OFF SHIFT KEY HELD DOWN

mov al, EOI ; END OF INTERRUPT COMMAND

out INTA00, al ; SEND COMMAND TO INTERRUPT CONTROL PORT

; INTERRUPT-RETURN-NO-EOI

mov al, ENA\_KBD ; INSURE KEYBOARD IS ENABLED

call ship\_it ; EXECUTE ENABLE

;mov ax, 08501h ; FUNCTION VALUE FOR BREAK OF SYSTEM KEY

;sti ; MAKE SURE INTERRUPTS ENABLED

;int 15h ; USER INTERRUPT

jmp k27a ; IGNORE SYSTEM KEY

k16a:

mov di, offset K6 ; SHIFT KEY TABLE

mov cx, K6L ; LENGTH

repne scasb ; LOOK THROUGH THE TABLE FOR A MATCH

mov al, ah ; RECOVER SCAN CODE

;je short k17 ; JUMP IF MATCH FOUND

;jmp short k25 ; IF NO MATCH, THEN SHIFT NOT FOUND

jne k25

;

;------ SHIFT KEY FOUND

k17:

sub di, offset K6+1 ; ADJUST PTR TO SCAN CODE MATCH

add di, offset K7

mov ah, byte ptr [DI] ; GET MASK INTO AH

test al, 80h ; TEST FOR BREAK KEY

;jz short k17c ; BREAK\_SHIFT\_FOUND

;jmp short k23 ; CONTINUE

jnz short k23

;

;----- DETERMINE SET OR TOGGLE

k17c:

cmp ah, SCROLL\_SHIFT

jae short k18 ; IF SCROLL SHIFT OR ABOVE, TOGGLE KEY

;

;----- PLAIN SHIFT KEY, SET SHIFT ON

or byte ptr [KB\_FLAG], ah ; TURN ON SHIFT BIT

jmp k26 ; INTERRUPT\_RETURN

;

;----- TOGGLED SHIFT KEY, TEST FOR 1ST MAKE OR NOT

k18: ; SHIFT-TOGGLE

test byte ptr [KB\_FLAG], CTL\_SHIFT ; CHECK CTL SHIFT STATE

jnz short k25 ; JUMP IF CTL STATE

;

cmp al, INS\_KEY ; CHECK FOR INSERT KEY

jnz short k22 ; JUMP IF NOT INSERT KEY

test byte ptr [KB\_FLAG], ALT\_SHIFT ; CHECK FOR ALTERNATE SHIFT

jnz short k25 ; JUMP IF ALTERNATE SHIFT

;

test byte ptr [KB\_FLAG], NUM\_STATE ; CHECK FOR BASE STATE

jnz short k21 ; JUMP IF NUM LOCK IS ON

test byte ptr [KB\_FLAG], LEFT\_SHIFT+RIGHT\_SHIFT

jz short k22 ; JUMP IF BASE STATE

;

k20: ; NUMERIC ZERO, NOT INSERT KEY

mov ax, 5230h ; PUT OUT AN ASCII ZERO

jmp k57 ; BUFFER FILL

k21: ; MIGHT BE NUMERIC

test byte ptr [KB\_FLAG], LEFT\_SHIFT+RIGHT\_SHIFT

jz short k20 ; JUMP NUMERIC, NOT INSERT

;

k22: ; SHIFT TOGGLE KEY HIT; PROCESS IT

test ah, byte ptr [KB\_FLAG\_1] ; IS KEY ALREADY DEPRESSED

jz short k22a0 ; GO IF NOT

jmp short k26 ; JUMP IF KEY ALREADY DEPRESSED

k22a0:

or byte ptr [KB\_FLAG\_1], ah ; INDICATE THAT THE KEY IS DEPRESSED

xor byte ptr [KB\_FLAG], ah ; TOGGLE THE SHIFT STATE

;

;----- TOGGLE LED IF CAPS OR NUM KEY DEPRESSED

test ah, CAPS\_SHIFT+NUM\_SHIFT+SCROLL\_SHIFT ; SHIFT TOGGLE?

jz short k22b ; GO IF NOT

;

push ax ; SAVE SCAN CODE AND SHIFT MASK

call snd\_led ; GO TURN MODE INDICATORS ON

pop ax ; RESTORE SCAN CODE

k22b:

cmp al, INS\_KEY ; TEST FOR 1ST MAKE OF INSERT KEY

jne short k26 ; JUMP IF NOT INSERT KEY

mov ax, INS\_KEY\*100h ; SET SCAN CODE INTO AH, 0 INTO AL

jmp k57 ; PUT INTO OUTPUT BUFFER

;

;----- BREAK SHIFT FOUND

k23: ; BREAK-SHIFT-FOUND

cmp ah, SCROLL\_SHIFT ; IS THIS A TOGGLE KEY

jae short k24 ; YES, HANDLE BREAK TOGGLE

not ah ; INVERT MASK

and byte ptr [KB\_FLAG], ah ; TURN OFF SHIFT BIT

cmp al, ALT\_KEY+80h ; IS THIS ALTERNATE SHIFT RELEASE

jne short k26 ; INTERRUPT\_RETURN

;

;----- ALTERNATE SHIFT KEY RELEASED, GET THE VALUE INTO BUFFER

mov al, byte ptr [ALT\_INPUT]

mov ah, 0 ; SCAN CODE OF 0

mov byte ptr [ALT\_INPUT], ah ; ZERO OUT THE FIELD

cmp al, 0 ; WAS THE INPUT=0

je short k26 ; INTERRUPT\_RETURN

jmp k58 ; IT WASN'T, SO PUT IN BUFFER

;

k24: ; BREAK-TOGGLE

not ah ; INVERT MASK

and byte ptr [KB\_FLAG\_1], ah ; INDICATE NO LONGER DEPRESSED

jmp short k26 ; INTERRUPT\_RETURN

;

;----- TEST FOR HOLD STATE

k25: ; NO-SHIFT-FOUND

cmp al, 80h ; TEST FOR BREAK KEY

jae short k26 ; NOTHING FOR BREAK CHARS FROM HERE ON

test byte ptr [KB\_FLAG\_1], HOLD\_STATE ; ARE WE IN HOLD STATE

jz short k28 ; BRANCH AROUND TEST IF NOT

cmp al, NUM\_KEY

je short k26 ; CAN'T END HOLD ON NUM\_LOCK

and byte ptr [KB\_FLAG\_1], NOT HOLD\_STATE ; TURN OFF THE HOLD STATE BIT

;

k26: ; INTERRUPT-RETURN

cli ; TURN OFF INTERRUPTS

mov al, EOI ; END OF INTERRUPT COMMAND

out INTA00, al ; SEND COMMAND TO INTERRUPT CONTROL PORT

k27: ; INTERRUPT-RETURN-NO-EOI

mov al, ENA\_KBD ; INSURE KEYBOARD IS ENABLED

call ship\_it ; EXECUTE ENABLE

k27a:

cli ; DISABLE INTERRUPTS

pop es ; RESTORE REGISTERS

pop ds

pop di

pop si

pop dx

pop cx

pop bx

pop ax

pop bp

iret ; RETURN, INTERRUPTS ON WITH FLAG CHANGE

;----- NOT IN HOLD STATE

k28: ; NO-HOLD-STATE

test byte ptr [KB\_FLAG], ALT\_SHIFT ; ARE WE IN ALTERNATE SHIFT

;jnz short k29 ; JUMP IF ALTERNATE SHIFT

;jmp short k38 ; JUMP IF NOT ALTERNATE

jz short k38

;

;----- TEST FOR CONTROL KEY AND RESET KEY SEQUENCE (CTL ALT DEL)

k29: ; TEST-RESET

test byte ptr [KB\_FLAG], CTL\_SHIFT ; ARE WE IN CONTROL SHIFT ALSO

jz short k31 ; NO RESET

cmp al, NUM\_KEY ; CHECK FOR INVALID NUM LOCK KEY

je short k26 ; THROW AWAY IF (ALT-CTL)+NUM-LOCK

cmp al, SCROLL\_KEY ; CHECK FOR INVALID SCROLL-LOCK KEY

je short k26 ; THROW AWAY IF (ALT-CTL)+SCROLL\_LOCK

cmp al, DEL\_KEY ; CTL-ALT STATE, TEST FOR DELETE KEY

jne short k31 ; NO-RESET

;

;----- CTL-ALT-DEL HAS BEEN FOUND

;;mov byte ptr [RESET\_FLAG], 1234h ; SET FLAG FOR RESET FUNCTION

;;jmp short START\_1 ; JUMP TO POWER ON DIAGNOSTICS

mov bx, BIOS\_DSEGM

mov ds, bx

mov bx, RESET\_FLAG

mov word ptr [BX], 1234h ; warm reset

; 07/03/2014

jmp cpu\_reset

;cpu\_reset:

; 07/03/2014

; CPU reset (power on) address

;db 0EAh ; far jump (jmp 0FFFFh:0000h)

;dw 0

;dw 0FFFFh ; F000:0FFF0h

;khere: hlt

; jmp short khere

;

;----- IN ALTERNATE SHIFT, RESET NOT FOUND

k31: ; NO-RESET

cmp al, 57 ; TEST FOR SPACE KEY

jne short k32 ; NOT THERE

mov al, ' ' ; SET SPACE CHAR

jmp k57 ; BUFFER\_FILL

;

;----- LOOK FOR KEY PAD ENTRY

k32: ; ALT-KEY-PAD

mov di, offset K30 ; ALT-INPUT-TABLE

mov cx, 10 ; LOOK FOR ENTRY USING KEYPAD

repne scasb ; LOOK FOR MATCH

jne short k33 ; NO\_ALT\_KEYPAD

sub di, offset K30+1 ; DI-NOW-HAS ENTRY VALUE

mov al, byte ptr [ALT\_INPUT] ; GET THE CURRENT BYTE

mov ah, 10 ; MULTIPLY BY 10

mul ah

add ax, di ; ADD IN THE LATEST ENTRY

mov byte ptr [ALT\_INPUT], al ; STORE IT AWAY

jmp short k26 ; THROW AWAY THAT KEYSTROKE

;

;----- LOOK FOR SUPERSHIFT ENTRY

k33: ; NO-ALT-KEYPAD

mov byte ptr [ALT\_INPUT], 0 ; ZERO ANY PREVIOUS ENTRY INTO INPUT

mov cx, 26 ; (DI),(ES) ALREADY POINTING

repne scasb ; LOOK FOR MATCH IN ALPHABET

jne short k34 ; NOT FOUND, FUNCTION KEY OR OTHER

mov al, 0 ; ASCII CODE OF ZERO

jmp k57 ; PUT IT IN THE BUFFER

;

;----- LOOK FOR TOP ROW OF ALTERNATE SHIFT

k34: ; ALT-TOP-ROW

cmp al, 2 ; KEY WITH '1' ON IT

je short k35 ; NOT ONE OF INTERESTING KEYS

cmp al, 14 ; IS IT IN THE REGION

jae short k35 ; ALT-FUNCTION

add ah, 118 ; CONVERT PSEUDO SCAN CODE TO RANGE

mov al, 0 ; INDICATE AS SUCH

jmp k57 ; BUFFER\_FILL

;

;----- TRANSLATE ALTERNATE SHIFT PSEUDO SCAN CODES

k35: ; ALT-FUNCTION

; 59 = scan code of F1 key

cmp al, 59 ; TEST FOR IN TABLE

;jae short k37 ; ALT-CONTINUE

jb k26

;k36: ; CLOSE-RETURN

; jmp short k26 ; IGNORE THE KEY

k37: ; ALT-CONTINUE

cmp al, 71 ; IN KEYPAD REGION

;jae short k36 ; IF SO, IGNORE

jae k26

mov bx, offset K13 ; ALT SHIFT PSEUDO SCAN TABLE

jmp k63 ; TRANSLATE THAT

;

;----- NOT IN ALTERNATE SHIFT

k38: ; NOT-ALT-SHIFT

test byte ptr [KB\_FLAG], CTL\_SHIFT ; ARE WE IN CONTROL SHIFT

jz short k44 ; NOT-CTL-SHIFT

;

;----- CONTROL SHIFT, TEST SPECIAL CHARACTERS

;----- TEST FOR BREAK AND PAUSE KEYS

cmp al, SCROLL\_KEY ; TEST FOR BREAK

jne short k39 ; NO-BREAK

mov bx , word ptr [BUFFER\_START] ; RESET BUFFER TO EMPTY

mov word ptr [BUFFER\_HEAD], bx

mov word ptr [BUFFER\_TAIL], bx

mov byte ptr [BIOS\_BREAK], 80h ; TURN ON @BIOS\_BREAK BIT

;

;----- ENABLE KEYBOARD

mov al, ENA\_KBD ; ENABLE KEYBOARD

call ship\_it ; EXECUTE ENABLE

int 1Bh ; BREAK INTERRUPT VECTOR

sub ax, ax ; PUT OUT DUMMY CHARACTER

jmp k57 ; BUFFER\_FILL

k39: ; NO\_BREAK

cmp al, NUM\_KEY ; LOOK FOR PAUSE KEY

jne short k41 ; NO-PAUSE

or byte ptr [KB\_FLAG\_1], HOLD\_STATE ; TURN ON THE HOLD FLAG

;

;----- ENABLE KEYBOARD

mov al, ENA\_KBD ; ENABLE KEYBOARD

call ship\_it ; EXECUTE ENABLE

mov al, EOI ; END OF INTERRUPT TO CONTROL PORT

out INTA00, al ; ALLOW FURTHER KEYSTROKE INTERRUPTS

;

;----- DURING PAUSE INTERVAL, TURN COLOR CRT BACK ON

push ds

mov bx, BIOS\_DSEGM

mov ds, bx

mov bx, offset CRT\_MODE

cmp byte ptr [BX], 7 ; IS THIS THE MONOCHROME CARD

je short k40p ; YES, NOTHING TO DO

mov dx, 03D8h ; PORT FOR COLOR CARD

mov al, byte ptr [CRT\_MODE\_SET] ; GET THE VALUE OF THE CURRENT MODE

out dx, al ; SET THE CRT MODE, SO THAT CRT 15 ON

;

;----- SUSPEND SYSTEM OPERATION (LOOP) TILL NEXT KEY CLEARS HOLD STATE FLAG

k40p:

pop ds

k40: ; PAUSE-LOOP

test byte ptr [KB\_FLAG\_1], HOLD\_STATE ; CHECK HOLD STATE FLAG

jnz short k40 ; LOOP UNTIL FLAG TURNED OFF

;

jmp k27a ; INTERRUPT\_RETURN\_NO\_EOI

;

;----- TEST SPECIAL CASE KEY 55

k41: ; NO-PAUSE

cmp al, 55

jne short k42 ; NOT-KEY-55

mov ax, 114\*100h ; START/STOP PRINTING SWITCH

jmp k57 ; BUFFER\_FILL

;

;----- SET UP TO TRANSLATE CONTROL SHIFT

k42: ; NOT-KEY-55

mov bx, offset K8 ; SET UP TO TRANSLATE C7L

cmp al, 59 ; IS IT IN TABLE

js short k56 ; YES, GO TRANSLATE CHAR

; CTL-TABLE-TRANSLATE

mov bx, offset K9 ; CTL TABLE SCAN

jmp k63 ; TRANSLATE\_SCAN

;

;----- NOT IN CONTROL SHIFT

k44: ; NOT-CTL-SHIFT

cmp al, 71 ; TEST FOR KEYPAD REGION

jae short k48 ; HANDLE KEYPAD REGION

test byte ptr [KB\_FLAG], LEFT\_SHIFT+RIGHT\_SHIFT

jz short k54 ; TEST FOR SHIFT STATE

;

;----- UPPER CASE, HANDLE SPECIAL CASES

cmp al, 15 ; BACK TAB KEY

jne short k45 ; NOT-BACK-TAB

mov ax, 15\*100h ; SET PSEUDO SCAN CODE

jmp short k57 ; BUFFER\_FILL

k45: ; NOT-BACK-TAB

cmp al, 55 ; PRINT SCREEN KEY

jne short k46 ; NOT-PRINT-SCREEN

;

;----- ISSUE INTERRUPT TO INDICATE PRINT SCREEN FUNCTION

mov al, ENA\_KBD ; INSURE KEYBOARD IS ENABLED

call ship\_it ; EXECUTE ENABLE

mov al, EOI ; END OF CURRENT INTERRUPT

out INTA00, al ; SO FURTHER THINGS CAN HAPPEN

;push bp ; SAVE POINTER

;int 05h ; ISSUE PRINT SCREEN INTERRUPT

;pop bp ; RESTORE POINTER

jmp k27 ; GO BACK WITHOUT EOI OCCURRING

k46: ; NOT-PRINT-SCREEN

cmp al, 59 ; FUNCTION KEYS

js short k47 ; NOT-UPPER-FUNCTION

mov bx, offset K12 ; UPPER CASE PSEUDO SCAN CODES

jmp k63 ; TRANSLATE\_SCAN

;

k47: ; NOT-UPPER-FUNCTION

mov bx, offset K11 ; POINT TO UPPER CASE TABLE

jmp short k56 ; OK, TRANSLATE THE CHAR

;

;----- KEYPAD KEYS, MUST TEST NUM LOCK FOR DETERMINATION

k48: ; KEYPAD-REGION

test byte ptr [KB\_FLAG], NUM\_STATE ; ARE WE IN NUM LOCK

jnz short k52 ; TEST FOR SURE

test byte ptr [KB\_FLAG], LEFT\_SHIFT+RIGHT\_SHIFT ; ARE WE IN SHIFT STATE

jnz short k53 ; IF SHIFTED, REALLY NUM STATE

;----- BASE CASE FOR KEYPAD

k49: ; BASE-CASE

cmp al, 74 ; SPECIAL CASE FOR A COUPLE OF KEYS

je short k50 ; MINUS

cmp al, 78

je short k51

sub al, 71 ; CONVERT ORIGIN

mov bx, offset K15 ; BASE CASE TABLE

jmp k64 ; CONVERT TO PSEUDO SCAN

k50:

mov ax, (74\*100h)+'-' ; MINUS

jmp short k57 ; BUFFER\_FILL

k51:

mov ax, (78\*100h)+'+' ; PLUS

jmp short k57 ; BUFFER\_FILL

;

;----- MIGHT BE NUM LOCK, TEST SHIFT STATUS

k52: ; ALMOST-NUM-STATE

test byte ptr [KB\_FLAG], LEFT\_SHIFT+RIGHT\_SHIFT

jnz short k49 ; SHIFTED TEMP OUT OF NUM STATE

;

k53: ; REALLY NUM STATE

sub al, 70 ; CONVERT ORIGIN

mov bx, offset K14 ; NUM STATE TABLE

jmp short k56 ; TRANSLATE\_CHAR

;

;----- PLAIN OLD LOWER CASE

k54: ; NOT-SHIFT

cmp al, 59 ; TEST FOR FUNCTION KEYS

jb short k55 ; NOT-LOWER-FUNCTION

mov al, 0 ; SCAN CODE IN AH ALREADY

jmp short k57 ; BUFFER\_FILL

k55: ; NOT-LOWER-FUNCTION

mov bx, offset K10 ; LC TABLE

;

;----- TRANSLATE THE CHARACTER

k56: ; TRANSLATE-CHAR

dec al ; CONVERT ORIGIN

xlat ; CONVERT THE SCAN CODE TO ASCII

;

;----- PUT CHARACTER INTO BUFFER

k57: ; BUFFER\_FILL

cmp al, -1 ; IS THIS AN IGNORE CHAR

;je short k59 ; YES, DO NOTHING WITH IT

je k26

cmp ah, -1 ; LOOK FOR -1 PSEUDO SCAN

;je short k59 ; NEAR\_INTERRUPT\_RETURN

je k26

;

; ; 07/03/2014

;; DELETE key handling (ASCII = 127)

;; (This code part was not in original INT 09h handler)

;; AX = 53E0h => AX = 007Fh <= AX = 5300h

; cmp ah, DEL\_KEY

; jne short k58

; cmp al, 0E0h

; je short @f

; and al, al

; jnz short k58

;@@:

; mov ax, 127

; jmp short k61

;

;----- HANDLE THE CAPS LOCK PROBLEM

k58: ; BUFFER\_FILL-NOTEST

test byte ptr [KB\_FLAG], CAPS\_STATE ; ARE WE IN CAPS LOCK STATE

jz short k61 ; SKIP IF NOT

;

;----- IN CAPS LOCK STATE

test byte ptr [KB\_FLAG], LEFT\_SHIFT+RIGHT\_SHIFT ; TEST FOR SHIFT STATE

jz short k60 ; IF NOT SHIFT, CONVERT LOWER TO UPPER

;

;----- CONVERT ANY UPPER CASE TO LOWER CASE

cmp al, 'A' ; FIND OUT IF ALPHABETIC

jb short k61 ; NOT-CAPS-STATE

cmp al, 'Z'

ja short k61 ; NOT\_CAPS STATE

add al, 'a'-'A' ; CONVERT TO LOWER CASE

jmp short k61 ; NOT\_CAPS\_STATE

;k59: ; NEAR-INTERRUPT-RETURN

; jmp short k26 ; INTERRUPT\_RETURN

;

;----- CONVERT ANY LOWER CASE TO UPPER CASE

k60: ; LOWER-TO-UPPER

cmp al, 'a' ; FIND OUT IF ALPHABETIC

jb short k61 ; NOT\_CAPS\_STATE

cmp al, 'z'

ja short k61 ; NOT CAPS STATE

sub al, 'a'-'A' ; CONVERT TO UPPER CASE

k61: ; NOT-CAPS-STATE

mov bx, word ptr [BUFFER\_TAIL] ; GET THE END POINTER TO THE BUFFER

mov si, bx ; SAVE THE VALUE

call k4 ; ADVANCE THE TAIL

cmp bx, word ptr [BUFFER\_HEAD] ; HAS THE BUFFER WRAPPED AROUND

je short k62 ; BUFFER\_FULL\_BEEP

mov word ptr [SI], ax ; STORE THE VALUE

mov word ptr [BUFFER\_TAIL], bx ; MOVE THE POINTER UP

cli ; TURN OFF INTERRUPTS

mov al, EOI ; END OF INTERRUPT COMMAND

out INTA00, al ; SEND COMMAND TO INTERRUPT CONTROL PORT

mov al, ENA\_KBD ; INSURE KEYBOARD IS ENABLED

call ship\_it ; EXECUTE ENABLE

;mov ax, 09102h ; MOVE IN POST CODE & TYPE

;int 15h ; PERFORM OTHER FUNCTION

jmp k27a ; INTERRUPT\_RETURN

;

;----- TRANSLATE SCAN FOR PSEUDO SCAN CODES

k63: ; TRANSLATE-SCAN

sub al, 59 ; CONVERT ORIGIN TO FUNCTION KEYS

k64: ; TRANSLATE-SCAN-ORGD

xlat ; CTL TABLE SCAN

mov ah, al ; PUT VALUE INTO AH

mov al, 0 ; ZERO ASCII CODE

jmp short k57 ; PUT IT INTO THE BUFFER

k62:

mov al, EOI ; ENABLE INTERRUPT CONTROLLER CHIP

out INTA00, al

mov cx, 678 ; DIVISOR FOR 1760 HZ

mov bl, 4 ; SHORT BEEP COUNT (1/16 1/64 DELAY)

call beep ; GO TO COMMON BEEP HANDLER

jmp k27 ; EXIT

snd\_data:

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

; SND\_DATA

; THIS ROUTINES HANDLES TRANSMISSION OF COMMAND AND DATA BYTES

; TO THE KEYBOARD AND RECEIPT OF ACKNOWLEDGEMENTS. IT ALSO

; HANDLES ANY RETRIES IF REQUIRED

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

;

push ax ; SAVE REGISTERS

push bx

push cx

mov bh, al ; SAVE TRANSMITTED BYTE FOR RETRIES

mov bl, 3 ; LOAD RETRY COUNT SOOT

cli ; DISABLE INTERRUPTS

and byte ptr [KB\_FLAG\_2], not (KB\_FE+KB\_FA) ; CLEAR ACK AND RESEND FLAGS

;

;----- WAIT FOR ANY PENDING COMMAND TO BE ACCEPTED

sub cx, cx ; MAXIMUM WAIT COUNT

sd1:

in al, STATUS\_PORT ; READ KEYBOARD PROCESSOR STATUS PORT

test al, INPT\_BUF\_FULL ; CHECK FOR ANY PENDING COMMAND

loopnz sd1 ; WAIT FOR COMMAND TO BE ACCEPTED

;

mov al, bh ; REESTABLISH BYTE TO TRANSMIT

out PORT\_A, al ; SEND BYTE

sti ; ENABLE INTERRUPTS

;mov cx, 01A00h ; LOAD COUNT FOR 10 ms+

xor cx, cx

sd3:

test byte ptr [KB\_FLAG\_2], KB\_FE+KB\_FA ; SEE IF EITHER BIT SET

jnz short sd7 ; IF SET, SOMETHING RECEIVED GO PROCESS

;

loop sd3 ; OTHERWISE WAIT

sd5:

dec bl ; DECREMENT RETRY COUNT

jnz short sd1 ; RETRY TRANSMISSION

or byte ptr [KB\_FLAG\_2], KB\_ERR ; TURN ON TRANSMIT ERROR FLAG

jmp short sd9 ; RETRIES EXHAUSTED FORGET TRANSMISSION

sd7:

test byte ptr [KB\_FLAG\_2], KB\_FA ; SEE IF THIS IS AN ACKNOWLEDGE

jz short sd5 ; IF NOT, GO RESEND

sd9:

pop cx ; RESTORE REGISTERS

pop bx

pop ax

retn ; RETURN, GOOD TRANSMISSION

snd\_led:

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

; SND\_LED

; SND\_LED1

;

; THIS ROUTINES TURNS ON THE MODE INDICATORS.

;

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

;

cli ; TURN OFF INTERRUPTS

test byte ptr [KB\_FLAG\_2], KB\_PR\_LED ; CHECK FOR MODE INDICATOR UPDATE

jnz short sl9 ; DON'T UPDATE AGAIN IF UPDATE UNDERWAY

;

or byte ptr [KB\_FLAG\_2], KB\_PR\_LED ; TURN ON UPDATE IN PROCESS

mov al, EOI ; END OF INTERRUPT COMMAND

out INTA00, al ; SEND COMMAND TO INTERRUPT CONTROL PORT

jmp short sl3 ; GO SEND MODE INDICATOR COMMAND

snd\_led1:

cli ; TURN OFF INTERRUPTS

test byte ptr [KB\_FLAG\_2], KB\_PR\_LED ; CHECK FOR MODE INDICATOR UPDATE

jnz short sl9 ; DON'T UPDATE AGAIN IF UPDATE UNDERWAY

;

or byte ptr [KB\_FLAG\_2], KB\_PR\_LED ; TURN ON UPDATE IN PROCESS

sl3:

mov al, LED\_CMD ; LED CMD BYTE

call snd\_data ; SEND DATA TO KEYBOARD

cli

call make\_led ; GO FORM INDICATOR DATA BYTE

and byte ptr [KB\_FLAG\_2], not KB\_LEDS ; CLEAR MODE INDICATOR BITS

or byte ptr [KB\_FLAG\_2], al ; SAVE INDICATORS STATES FOR NEXT TIME

test byte ptr [KB\_FLAG\_2], KB\_ERR ; TRANSMIT ERROR DETECTED

jnz short sl5 ; IF SO, BYPASS SECOND BYTE TRANSMISSION

;

call snd\_data ; SEND DATA TO KEYBOARD

cli ; TURN OFF INTERRUPTS

test byte ptr [KB\_FLAG\_2], KB\_ERR ; TRANSMIT ERROR DETECTED

jz short sl7 ; IF NOT, DON'T SEND AN ENABLE COMMAND

sl5:

mov al, KB\_ENABLE ; GET KEYBOARD CSA ENABLE COMMAND

call snd\_data ; SEND DATA TO KEYBOARD

cli ; TURN OFF INTERRUPTS

sl7:

and byte ptr [KB\_FLAG\_2], not (KB\_PR\_LED+KB\_ERR) ; TURN OFF MODE INDICATOR

sl9: ; UPDATE AND TRANSMIT ERROR FLAG

sti ; ENABLE INTERRUPTS

retn ; RETURN TO CALLER

make\_led:

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

; MAKE\_LED

;

; THIS ROUTINES FORMS THE DATA BYTE NECESSARY TO TURN ON/OFF

; THE MODE INDICATORS.

;

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

;

push cx ; SAVE CX

mov al, byte ptr [KB\_FLAG] ; GET CAPS & NUM LOCK INDICATORS

and al, CAPS\_STATE+NUM\_STATE+SCROLL\_STATE ; ISOLATE INDICATORS

mov cl, 4 ; SHIFT COUNT

rol al, cl ; SHIFT BITS OVER TO TURN ON INDICATORS

and al, 07h ; MAKE SURE ONLY MODE BITS ON

pop cx

retn ; RETURN TO CALLER

ship\_it:

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

; SHIP\_IT

;

; THIS ROUTINES HANDLES TRANSMISSION OF COMMAND AND DATA BYTES

; TO THE KEYBOARD CONTROLLER.

;

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

;

push ax ; SAVE DATA TO SEND

;----- WAIT FOR COMMAND TO ACCEPTED

cli ; DISABLE INTERRUPTS TILL DATA SENT

sub cx, cx ; CLEAR TIMEOUT COUNTER

s10:

in al, STATUS\_PORT ; READ KEYBOARD CONTROLLER STATUS

test al, INPT\_BUF\_FULL ; CHECK FOR ITS INPUT BUFFER BUSY

loopnz s10 ; WAIT FOR COMMAND TO BE ACCEPTED

pop ax ; GET DATA TO SEND

out STATUS\_PORT, al ; SEND TO KEYBOARD CONTROLLER

sti ; ENABLE INTERRUPTS AGAIN

retn ; RETURN TO CALLER

;----- TABLE OF SHIFT KEYS AND MASK VALUES (EARLY PC)

K6: db INS\_KEY ; INSERT KEY

db CAPS\_KEY,NUM\_KEY,SCROLL\_KEY,ALT\_KEY,CTL\_KEY

db LEFT\_KEY,RIGHT\_KEY

K6L equ $-K6

;----- SHIFT\_MASK\_TABLE

K7: db INS\_SHIFT ; INSERT MODE SHIFT

db CAPS\_SHIFT,NUM\_SHIFT,SCROLL\_SHIFT,ALT\_SHIFT,CTL\_SHIFT

db LEFT\_SHIFT,RIGHT\_SHIFT

;----- SCAN CODE TABLES

K8: db 27,-1,0,-1,-1,-1,30,-1,-1,-1,-1,31

db -1,127,-1,17,23,5,18,20,25,21,9,15

db 16,27,29,10,-1,1,19,4,6,7,8,10

db 11,12,-1,-1,-1,-1,28,26,24,3,22,2

db 14,13,-1,-1,-1,-1,-1,-1,' ',-1

;----- CTL TABLE SCAN

K9: db 94,95,96,97,98,99,100,101,102,103,-1,-1

db 119,-1,132,-1,115,-1,116,-1,117,-1,118,-1

db -1

;----- LC TABLE

K10: db 01Bh,'1234567890-=',08h,09h

db 'qwertyuiop[]',0Dh,-1,'asdfghjkl;',027h

db 60h,-1,5Ch,'zxcvbnm,./',-1,'\*',-1,' '

;----- UC TABLE

K11: db 27,'!@#$',37,05Eh,'&\*()\_+',08h,0

db 'QWERTYUIOP{}',0Dh,-1,'ASDFGHJKL:"'

db 07Eh,-1,'|ZXCVBNM<>?',-1,0,-1,' ',-1

;----- UC TABLE SCAN

K12: db 84,85,86,87,88,89

db 90,91,92,93

;----- ALT TABLE SCAN

K13: db 104,105,106,107,108

db 109,110,111,112,113

;----- NUM STATE TABLE

K14: db '789-456+1230.'

;----- BASE CASE TABLE

K15: db 71,72,73,-1,75,-1

db 77,-1,79,80,81,82,83

;----- TABLE OF KEYPAD CURSOR ; CONTROL KEYS

K\_TAB1:

db UP\_M, DN\_M, INS\_M, DEL\_M, LEFT\_M, RIGHT\_M

db PGUP\_M, PGDN\_M, HOME\_M, END\_M

L\_TAB1 equ $-K\_TAB1

;----- ALT-INPUT-TABLE

K30: db 82,79,80,81,75,76

db 77,71,72,73 ; 10 NUMBERS ON KEYPAD

;

;----- SUPER-SHIFT-TABLE

db 16,17,18,19,20,21 ; A-Z TYPEWRITER CHARS

db 22,23,24,25,30,31

db 32,33,34,35,36,37

db 38,44,45,46,47,48

db 49,50

; $