Description I/O routines IO65, version 2.01 1986

Table of contents Page:

IO65 was developed by: E.J.M. Visschedijk
Drakensteyn 299
7608 TR Almelo
The Netherlands

This manual has been written by: E.J.M. Visschedijk

Translated by: H.A.J.Oort
van Eijsingalaan 28
3527 VL Utrecht
The Netherlands

Introduction

The IO65 part is located in the eprom. This is an 2764 which however is used only half. This is done because, firstly, the DOS65 users of the first hour have an 2764 monitor eprom that they can have reprogrammed. Secondly, an 2764 is cheaper as an 2732. Finaly because probable the other half will be used as well in future.

IO65 is used for the elementery routines like in and output routines, IO device control, device initialization, status line routines and the interrupt dispatch.

Compared to version 1.01 many things have been changed.

Modifications with regard to DOS 1.01 and MON 1.01/2:

If the system is switched on, IO65 will report. The same will occur after a RESET. Then DOS65 is meant to be started by starting the bootstrap. This is done by depressing the key B (b). That is the only key, apart from the function keys, that will be accepted at this point. The bootstrap is started and DOS65 will report in the DOS command mode. At that point we possible can get MON65 from disk to work with the monitor that version 1.01 automaticly started with.

In IO65 also an automatic 'screen off' is included. If during an adjustable time no key was depressed, automaticly the screen will be switched off. This to prevent the burning in of the picture tube.

The start and end addresses for in- and output device 5 have to be set by hand now. This can be done via MON65 or via the DOS command MEMFILL. Before the in- or output device 5 is initialized first the variables WRBEG, WREND and/or REBEG, REEND have to be filled.

Some usefull IO entries have been added. (check for this the jumptable, chapter 5). Other entries which are included in DOS have been left out, like: PRBYT, CRLF etc.. It is possible now by way of the jumptable to initialize and call a device independent of the devices active at that moment.

1. Input and output devices

Every computer system needs input of some kind. The input is digested by the system with a specific program. The result of the input and this program have to be made visible on some peripheral that takes care of the output. From now on we'll call everything that can be used for input, input devices and everything that can be used for output, ouput devices. Accordingly a keyboard is called an input device and a picture screen or printer an output device. Every system knows one or more I/O devices that have to be supported by software. IO65 takes in principle 8 input and 8 output devices into account. To keep track of the device active at present, a byte is reserved for the input as is for the output. Each bit in this byte represents a device. Is a bit 1 then the related device is active. Always one device should be active, for input as for output.

Using the function keys the I/O devices can be selected. If all input and/or output devices were switched off then, because this should not occur, automaticly the default I/O devices will be switched on. The variable in which the output device bits are located is DEVMODOUT and the variable in which the input device bits are located is DEVMODINP. The defauld output device bits are located in the variable OUTRET and the default input device bits are located in the variable INPRET. The variables INPRET and OUTRET are so chosen that on default the keyboard and the screen are switched on. OUTRET and INPRET can not be changed with the functionkeys. Following below is an enumeration of the I/O device numbers with the peripheral they control:

OUTPUT DEVICE NUMBER	DEVICE DESCRIBTION	INPUT DEVICE NUMBER	DEVICE DESCRIBTION
1	Screen	1	Keyboard
2	printer, centronics	2	Not used
3	RS232 output	3	RS232 input
4	VIACOM output	4	VIACOM input
5	Memory output	5	Memory input
6	Free (DV06VEC)	6	Free (DV16VEC)
7	Free (DV07VEC)	7	Free (DV17VEC)
8	Free (DV08VEC)	8	Free (DV18VEC)

The output device # 1 is the screen which will be usually on. In IO65 are all routines needed to control the screen available. The asumption is made that for video-display-hardware the Elector's VDU is used with an 16 MHz crystal. A different character generator is being used, this to get inversed video.

Output device # 2 is used to control a printer. The printer needs to have a centronics parallel interface. If this device is switched on while there is no printer connected, after a few seconds this device will be switched off again to prevent there is being waited for a handshake signal from a printer that is not connected.

Output device # 3 is a serial connection via the 6551 ACIA that is present on the CPU print. The default value for this ACIA is 2400 baud. After selection of in- or output device # 3 the variables ACCTL and ACCMD are loaded into the control and command registers of the ACIA. By changing these variables another baudrate may be chosen. The default value for ACCTL is $BA and for ACCMD $05. With the program RS232 it is possible to adjust these values very quickly and to write these values in a file which is automaticly being loaded when starting the system. This way the always the correct values are written into the variables ACCTL and ACCMD when the system is started.

Output device # 4 is the so called VIACOM output. IO65 is equiped with a special communications program. This communication is accomplished with 8 data-lines and 2 handshake-lines from a VIA. Using this, 2 computers which both have VIACOM can communicate with one another. If during some seconds no handshake signal is received by the VIA, automaticly this device is switched off and the default output device is switched on.

Output device # 5 is the memory. IO65 is able to use the memory like an output device. This way, output that would normally be put on screen now can be put in memory as well. If this device is being switched on first some memory-pointers are copyed to define the area in which the information has to be stored. Is this area used up automaticly output device # 5 is switched off.

Output devices # 6, 7 and 8 are not used. Output routines written by yourself can be called on via these device numbers. To each output device belongs a vector which has to point to the output routine. The names of these vectors may be found in the table.

Input device # 1 is the keyboard. This is also the default input device. If all input devices are switched off this input device is automaticly switched on again to prevent loss of control over the system.

Input device # 2 is not used. Actually he stands beside the centronics output. To make this an centronics input is not very usefull as the VIACOM input also can be seen as a centronics input.

Input device # 3 is the RS232 input. This input uses the same ACIA as the RS232 output uses. The default value for this device also is 2400 baud. To change the baudrate and the number of stopbits and such, the variables ACCTL and ACCMD have to be adjusted. After a system reset these variables have again their default value and need readjusting.

Input device # 4 is the VIACOM input. The intention with VIACOM is stated above already. The use of VIACOM is explaned in chapter 6. If after a few seconds no handshake signal is received by the VIA, then automaticly this device is switched off and the default device is switched on.

Input device # 5 is the memory, used as input device this time. Read is from memory while the system 'thinks' the input is from the keyboard.

Input devices # 6, 7 and 8 are, as with the output devices 6, 7 and 8 free to be defined by the user.

2. Function keys with their functions

It is possible, even if you have a keyboard without function keys, to use function keys. The use of those keys namely is a sequence of common ascii values and not the use of keys with some strange code. To get with a function key first the value $1E has to be sent. On most keyboards this is the controlkey together with a circumflex (^). After this $1E a normal key is depressed. If one should be unable to get $1E from his keyboard, then the variable MONESC has to be changed. The default value is $1E. Because the control-value of ^ in our system of notation ^^ is, are the function keys in this manual preceded by ^^.

The function keys which always may be used if the keyboard is an active input device, are:

```
^^I  - show and change active input devices
^^O  - show and change active output devices
^^S  - on/off switch of the statusline
```

Be sure to use capitals for the 'common'-keys of the functionkeys. If no capitals are used there will be no function key recognized.

Ctrl US in caps mode generates $1E

2.01 Function ^^I

This function key is used to see which input devices are active and to switch an input device on or off. After ^^I is typed a different statusline will be shown. On this statusline is written that we are looking at the input devices. A number from 1 to 8 may now be typed. On the statusline we will see a possible 'on' behind a number change into 'off'. The device corresponding with the typed number has been switch off then. Switching off all input devices is madness. The software therefore will in such case switch on the default device again. If two input devices are switched on simultaneously, then only the one most to the left will be active. This in contradiction to the output devices of which more then one may be switched on at the same time. Input device # 2 is, as stated before, not used.

After one has chosen for input device # 4 and switched off input device # 1 the input has to come from the VIA-port according to the VIACOM protocol. However, is no device connected then, after about 10 seconds, device # 4 will be switched off again; input device # 1 is automaticly switched on then. One should take into account that if another input device is switched on, often there is still being waited for an character from the previous input device. If this device for example was the keyboard still a dummy key has to be depressed. The contents of this dummy key however, will be sent to the active output device.

The input devices 6, 7 and 8 are free to be defined by the user. The corresponding vectors have to point to the input routines, to make them work. In case of a default these vectors point to a routine that will switch on the default input device again and switch off the device that was switched on just now.

2.02 Function ^^O

This function key is used to check which output devices are active and to switch on or off output devices. This function operates analogous to the ^^I function key. It is however possible to switch on several output devices simultaneously.

2.03 Function ^^S

This function key is used to switch off the statusline that appears in the bottom of the screen. By using the same function key again the statusline will be switched on again. If the statusline is switched of it is not possible to have a 25th line on the screen all the same.

The statusline

There is 2 Kbyte ram located on the VDU-print. This enables us to store 25 lines of 80 characters each. As 24 lines of 80 characters form a standard screen it was decided to use the remaining space for putting a statusline on the screen. The statusline is optional, which means it can be switched on and off. It is not possible to chose another screen format, like 25 * 80 characters and no statusline. The statusline can be switched off with ^^S. Normally the statusline is in inverse video. This can be changed, by changing the variable INVST (default on $80) into $00. After this variable has been changed the command CLEAR should be given while in DOS command mode, otherwise some characters will remain in reverse. Those are the characters that are not changed every second or after each time a key is depressed.

There are 3 kinds of statuslines. The most important one is the statusline which appears after switching on or resetting the system. From left to right to following is shown on the statusline.

Time hh:mm:ss	- This is the time in hours, minutes and seconds. Using the DOS command TIME this can be changed.
Date dd-mm-yy	- The date is shown in day, month and year. The date can be changed using the DOS command TIME or, in case you are using a real time clock, with the command SETRTC. Only at 00:00:00 the date is changed by IO65. If the memory addresses corresponding to the date are changed by a program written by yourself then this will not be immediately visible on the screen One has to change the flag DATUPD to $FF; then the date will be adjusted at the moment the monitor jumps to the keyboard input routine.
Col: xx	- This states the column-position of the cursor. Or put differently, the horizontal position of the cursor. The value is 1 if the cursor is in its leftmost position. His maximum value is 80.
Row: yy	- This is the cursor position in vertical direction. The top-line is line 1, the bottom-line is line 24

The next information will only be on the statusline if the screen editor ED is active. After leaving ED again, this part of the statusline will erased.

Ln: z	- During an edit session always the cursor position is shown on the statusline in Col and Row. However this is the position on the screen, it is unknown which line of the file it is. If an edit-file exists of 200 lines and the cursor is on the bottomline, then z will be 200.
Fn: filename	- If one opens a file for editting, here the file-name will be shown. If again a file is opened the name will be changed.

There do exist 2 more statuslines. If the function ^^O or ^^I is chosen to change in- or output devices, on the statusline appears information on the present situation. Shown is which devices are active.
Also it is possible to define a statusline yourself. For this purpose special entries are made in the jumptable. This way it is possible to:

- delete the complete statusline
- put a string, defined by yourself, on the statusline
- call back the old statusline

4. Interrupts

RESET - If the RESET key is depressed all variables will be changed back to their default values. The clock will keep on going and mostly will not require readjusting. (depents on the reason for the RESET). The keyboard and screen are initialized again thus always recovering control over the system. Again a B has to be typed to start the bootstrap.

NMI - After a NMI, via a NMI VECTOR, a jump is made to a dummy RTI. For various purposes the vector can be changed by yourself.

IRQ - Via the vector IRQVECTOR a jump is made to the IRQ routine. In which routine it is checked what caused the interrupt. Then from a table the address of the routine required is fetched. In this table are the absolute addresses minus 1. This because via the stack is jumped to these addresses. This table contains 16 vectors. Both VIA's may cause 7 interrupts. Further a interrupt may be caused by the ACIA and also a software interrupt is possible. The software interrupt vector points to a dummy RTS. In the program MON65 this vector is diverted to be able to generate a software break.
The addresses of the 16 vectors are described in chapter 7 with the variables.
Is however no cause for the interrupt found, then via the UNRINT vector will be jumpted to a routine that will put 'IRQ ignored' on the screen. The cause of the interrupt however is not removed, thus causing this information to keep on coming. The system only can be saved mostly by depressing the RESET key. The UNRINT vector points directly to an absolute address. Did one connect an extra peripheral which may cause an interrupt, then will be jumpted to the UNRINT vector. While initializing this peripheral (the real time clock for example) the UNRINT vector has to be diverted and has to point to a routine that will check if the interrupt was caused by this device and if not, there's yet to be jumped to 'IRQ ignored'. If the peripheral did cause the interrupt then may be continued with the interrupt routine for this device.

The interrupt routines that are pointed at by the table are to be concluded with an RTS thus allowing the main interrupt routine in IO65 to get back the values put on stack and then to continue with the main program. Below the IRQ table is printed. The addresses of the vectors may be found in chapter 7.

```
INTV1  -  T1    VIA 1    used by system clock
INTV2  -  T2    VIA 1
INTV3  -  CB1   VIA 1
INTV4  -  CB2   VIA 1
INTV5  -  SR    VIA 1
INTV6  -  CA1   VIA 1    used by keyboard
INTV7  -  CA2   VIA 1
INTV8  -  T1    VIA 2
INTV9  -  T2    VIA 2
INTV10 -  CB1   VIA 2
INTV11 -  CB2   VIA 2
INTV12 -  SR    VIA 2
INTV13 -  CA1   VIA 2
INTV14 -  CA2   VIA 2
INTV15 -  ACIA           used by IO device # 3
INTV16 -  Software interrupt.
```

The jumptable

The jumptable has been made to have fixed entries for the important routines which are called on from the DOS or other programs. Following below is a list and describtion of these routines.

$F000 - General output routine. Put a character in the accumulator and call on this routine. All active output devices will receive this character for output. The registers A, X and Y remain intact.

$F003 - General input routine. If a jump is made to this routine the routine will return with the input value from the active input device in accu. If more then one input device is switched on, only the most significant one is used. The X and Y registers remain intact.

$F006 - Also this is an output routine. However here the output device that is in the X register will be regarded as being active. Has the X register for example the value $02 then the character will go to the printer.
The numbers that should be in the X register are similar to those used for the output devices with the function key ^^O. Valid are only numbers from 1 to 8. If however 9 is chosen then something will be put on the statusline. In such case also the Y register is of importance. In this register is the X position on the statusline located. Would one want to write an 'A' on the 25th position of the statusline, then in the accu should be $41, in the X register $09 and in the Y register $19. At the end of this routine the registers are destroyed.

$F009 - This is an input routine which uses the device that is pointed to by the X register. The numbers 1 to 8 can be used and correspond to those of funtionkey ^^I. However if $09 is located in the X register then a jump will be made to a key-board input routine which checks if there's still anything in the input buffer. If not then will be jumped back with the accu $00, else with the key-value in accu. The X and Y register are destroyed in this routine.

$F00C - With this entry it is possible to initialize devices. The number of the device is put in the X register. To make a distinction between input and output devices the most signifi-cant bit should be set in case of input devices. If one want to initialize the printer from an application-program, then a jump should be made to this routine with $02 in the X register. The carry indicates whether the initialization was succesfull or not. If no printer was connected or selected then will be returned with the carry set. If one wants to initialize VIACOM as an input device then should be jumped to this entry with $83 in the accumulator.

$F00F - This entry is used by the editor to initialize the right part of the statusline. 'Ln:' and 'Fn:' is put on the statusline.

$F012 - In the accu (high byte) and the Y register (low byte) is the value located which should be printed in decimal behind 'Ln:' on the statusline. This entry is used by the editor.

$F015 - In the accu (high byte) and the Y register (low byte) is the address of the string located that represents the filename which has to be put behind 'Fn:' on the statusline. This entry is used by the editor.

$F018 - After calling this routine the right part of the statusline is erased. Also this routine is used by the editor.

$F01B - After calling this routine the complete statusline is erased. An erased statusline shows spaces in inverse video only.

$F01E - In the accu (high byte) and the Y register (low byte) the address of the string is located which is printed on the status line. Using this one is able to define a statusline himself. The string should be concluded with $00 and should not be longer then 80 characters. After a 'clear screen' this routine should be called again, because the IO65 clear screen routine does not know were the user defined statusline is. Also one perhaps just should omit 'clear screen' in this case.

$F021 - With this routine it is possible to call back the standard statusline again.

$F024 - After calling this routine the cursorposition is changed into the position indicated by the X and Y register. The position on the upper-left side of the screen is 1,1. X goes from $01 to $50 (is 80 in decimal). Y goes from $01 to $19 (is 25 in decimal).
If now the next routine is called the cursor will be put back to its previous position.

$F027 - If the routine at $F024 was called then with this routine the cursorposition from before can be restored. However these routines can not be called on repeatedly. So by calling $F024 twice without calling $F027 in between, the first return position will be destroyed.

6. I/O routines

In this chapter we'll go further into certain IO65 routines. Described is how these routines work and also what they can be used for.

6.01 The screen output routine

On default all output goes to the screen. Only if there are output devices changed the ouput will go to other devices. However the screen remains the most important ouput device. Because of this some very special possibilitys were made for this device.

- Cursor control
- Invers video
- Grafics

It is possible to control the cursor by 'direct cursor addressing'. Which means that the cursor can be moved to any screen-position by a fixed sequence of characters. This can be done by using the jumptable-entries $F024 and $F027 but also in a way as is decribed below.

By first sending the character $14 (control T) to the screen the routine is notified that the next two characters represent a cursor position. The screen consists of 80 columns and 24 lines. (The status-line not counted here.). The position specified therefore should be inside this 80 * 24 field. Values are to be entered in hexadecimal, so the field limits are: $01...$50 and $01...$18. (The routine itself uses values from $00, so 1 less, but the user will not notice this.). After the character $14 first the column co-ordinate and then the line co-ordinate is to be stated. If one would wish to print an '*' on the 10th line in the 40th column, then the following sequence of characters are to be sent to the screen output routine: $14 $28 $0A $2A. Ofcoarse also it is possible to print several characters this way. Also several cursor control commands can be used in a row. To print something on the statusline the line co-ordinate should be $19. Then care has to be taken not to have this character erased by the clock-display. (for example). This can be prevented by making the variable STATTOG $FF. The statusline then is no longer kept up to date. One should first call on a routine which clears the statusline.

Also ther are other special (cursor control) characters or characters which cause the erasure of the whole screen or just a part of it. Those characters are listed below.

- $07 - Bell. Produces a beeb if the necessary hardware is there.
- $08 - Back space. Puts the cursor one position back. If the cursor position was 1,1 then a scroll down follows.
- $09 - Horizontal tab. The cursor will be moved to the next tab-position. There is a tabposition every 8 positions. If the cursor is past the last tab position of the current line there will be scrolled up.
- $0A - Line feed. The cursor moves one line down. If the cursor was on the last line then a scroll up will follow.
- $0B - Vertical tab. The cursor is moved one line up. If the cursor was on the topline a scroll down will follow.
- $0C - Form feed. The screen is cleared. The statusline remains unchanged. The cursor is moved to position 1,1.
- $0D - Carriage return. The cursor is moved to the leftmost column of the current line.
- $14 - Cursor direct addressing. After this code 2 co-ordinates are expected. If one of both co-ordinates are not inside the available field the command is not executed. If the first co-ordinate entered was wrong already then still the second will be expected, but nothing will be done with it.
- $19 - Clear to end of screen. This clears the screen from the position to the end of the screen. The cursor position does not change.
- $1A - Clear to end of line. Clears from the cursorposition to the end of the line. The cursor position is not changed.
- $1B - Escape. Following this character the screen can be changed into invers or grafics. Also with an escape sequence that mode is restored.
- $1C - Home. The cursor moves to position 1,1. The screen remains unchanged.

Invers video is another possibility of the screen. With this parts of text can be accentuated. This is amongst others done with the status line. To put an character in invers video on the screen the most significant bit of this character should be set or the character should be preceded by an escape sequence. This sequence is <ESCAPE> and i ($1B $69). The characters following this sequence are all in inverse video. After the escape sequence <ESCAPE> and n ($1B $6E) the characters will be printed normal again.

It also is possible to get a limited number of grafic characters on the screen. By first switching on invers video also those characters can be displayed inversed. The escape sequence for grafics is <ESCAPE> and f ($1B $46). To switch off the grafics mode, use the sequence: <ESCAPE> and g ($1B $47).

7.02 The Viacom routines

The VIACOM I/O routines are developed to be able to have two systems, which both are provided with a protocol like this, communicate with one another. Communications is accomplisched with 8 bits parallel, using a VIA. Also 2 handshake lines are needed and a 'ground', so, at least a cable with 11 cores is needed to connect the 2 systems. VIACOM is designed to sent programs, parts of memory, etc. from one system to the other. The VIACOM routines are part of the I/O devices. (Number 4). If output device 4 is switched on together with the screen then data which is sent to the screen also is sent to the output VIACOM routine. If no other computer is connected to the computer then seemingly the system 'hangs'. However after about 10 seconds this device will be switched off automaticly. The same apply's to input device # 4. This device however cannot function simultaneously with the keyboard as the input routine only selects the most significant input device which is the keyboard. So the keyboard has to be switched off. Then one switches on the VIACOM input and again it seems as if the system 'hangs' if no other system is connected to the other side of the VIA. Both systems should be connected as is shown in fig. 1. Using the CPU-print of Elector that is the A side of IC 3. If the hardware is in order then it should be possible to sent a program from DOS65 using the discribtion below.

The procedure is as follows:

Sender:

1 - 'Type' the file with the command TYPE in this way:
    TYPE - N filename
2 - Before depressing <RETURN> behind the filename, first ouput-device # 4 should be switched on. (^^4).
3 - Then depress <RETURN>. Sometimes another return is needed.

Receiver:

1 - 'Create' a file using the CREATE command in this manner:
    CREATE filename
2 - An '&' appears as prompt. Now switch off input device # 1 and switch on input device # 4. (^^I 1 4).
3 - Sometimes a return is needed.

After the file is sent over another 10 seconds will pass before the receiver regains control over his keyboard. Then still ^D and <RETURN> have to be typed to close the CREATE correctly. In the file some editting is needed to remove an ^J (linefeed) which turned up at the beginning of the file, and also an '$' at the end of the file.
The sender still has control over his keyboard and so, can switch off device # 4 without problem. (^^O 4).

The operation of Viacom is based on two VIA's which communicate with one another using full auto handshake. Fig. 2 should make this a little clearer. The sender is ready to transmit data and checks the CA1 line on becoming zero, which means he can start sending. If this line is zero the sender puts the data on the VIA-bus and also changes CA2 to zero thus making the CA1 line in the receiver zero. The data on the sender VIA-bus now is valid. A CA1 line becoming zero automaticly changes the CA2 line into high again. At the receiver now CA1 becomes zero thus causing his CA2 and the CA1 of the sender becoming high. The receiver also checkes his CA1 line and as soon as this line becomes zero the receiver reads the VIA data bus. After the data has been read by the receiver he will change his CA2 line to low which causes the CA1 line at the sender becoming zero as well. If the receiver was not yet ready digesting previous data then his CA1 line will have been low for some time already before new data could be read. If the senders CA1 becomes zero again the sender is allowed to put something on the VIA-bus again. If the sender is still busy getting the next data then the CA1 line will become zero before he is able to sent. Now data will be sent as soon as the sender is ready.

The above works perfectly once all got going. The routine is very simple: check CA1, if low then write or read to or from the VIA-bus. If CA1 is high then wait, but no more then 10 seconds. However, the big problem occurs when the sender has to sent the first character. At that moment the receiver has not yet received anything and the sender's CA1 is still high. The result is that the first character will never be sent. The solution however is rather simple. The output routine uses a vector. When the output device is initialized, then the vector is set so that the first character which has to be sent directly is put on the VIA's data-bus, without checking CA1. After this character has been sent the vector is changed so that before the next characters are sent first CA1 is checked.

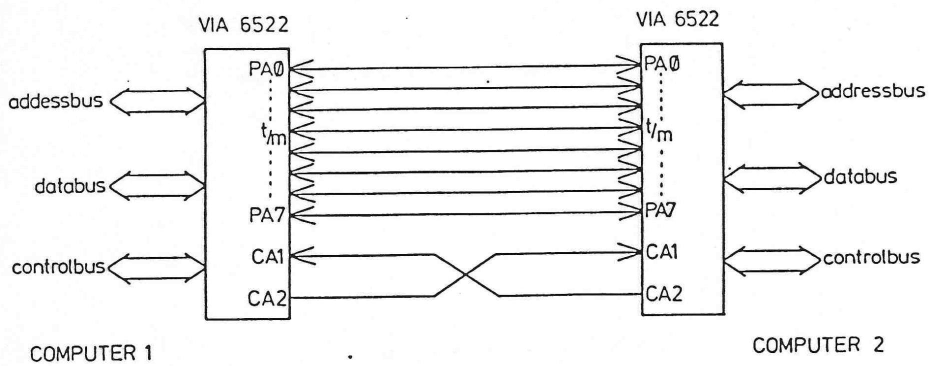
VIA 6522
VIA 6522
PAØ
t/m
PA7
CA1
CA2
addessbus
databus
controlbus
addressbus
databus
controlbus
COMPUTER 1
COMPUTER 2

FIG.1

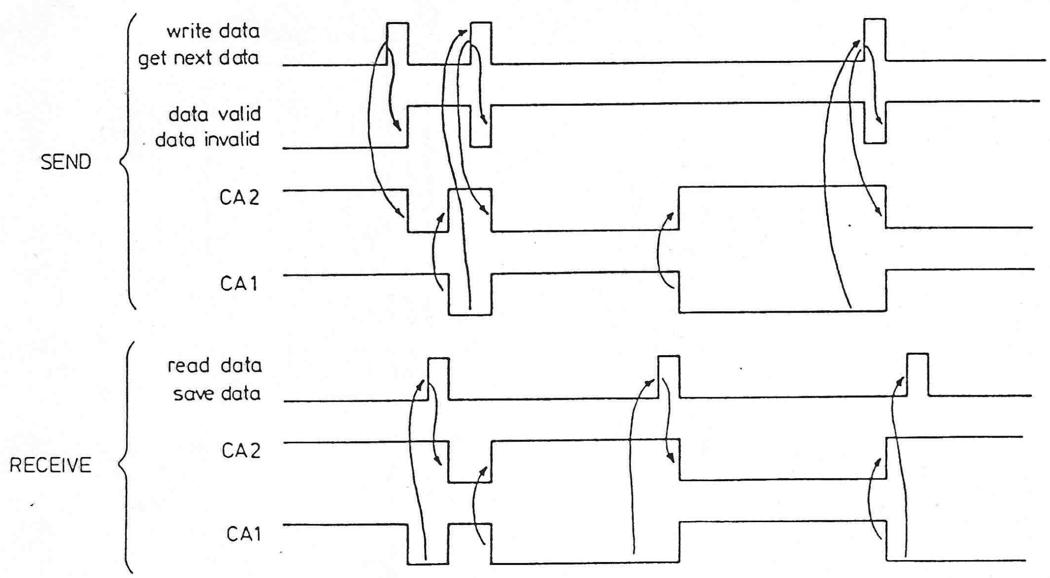
SEND
write data
get next data
data valid
data invalid
CA2
CA1
RECEIVE
read data
save data
CA2
CA1

FIG.2

6.03 RS232 Interface

There is both an in- and output RS232 routine included in the monitor. With the input routine device # 3 has to be switched on and # 1 to be switched off. At the receiving of an 'receiver register full' interrupt the received character is put into the keyboard buffer. The RS232 receiver routine reads those characters from the keyboard buffer. The result of which is that, while receiving via RS232 also the characters will be received which are keyed in on the keyboard.
The RS232 input routine does not use a handshake procedure.

The RS232 output routine can, for example, be used for controlling a serial printer. From DOS the printer will not work with the PRINT command, as this command uses the centronics output.
The RS232 output does work with a handshake procedure. For if characters are sent then this output can be blocked be sending an ^S to the RS232 input. Following an ^Q the output goes on. This software handshake is used by many printers. Also many modem connections use this protocol. Switching off this device will cause no problems as the keyboard still can be used.

Variables

The intention of this chapter is to make the user of DOS65 and IO65 clear which variables are where and also for what purpose one could use the variables oneself. Following below is a list of all variables used in I/O65. First the variables absolute address is listed followed by the name and an explanation. The name used is the same as used in the source listing of IO65 which is available as well.

A great number of zero-page variables are used. Which means used temporaryly. The variables located on the addresses from $FB to $FF are used by the bootstrap routine. The variable PTA and PTB are used by IO65 itself, but after use the original contents is restored. So these addresses may be used for other purposes. Which means, in normal programs. Errors will occur when these variables are used in interrupt routines.

0000	PTA	Is used to copy the statusline in the scroll routine, with scroll up and scroll down. (2 bytes)
0002	PTB	Also these variables are used in the scroll routines. Zero-page addresses were chosen because of speed-considerations. (2 bytes)

The variables below are used by the bootstrap routine.

00FB	SECC	TSL location.
00FC	RPOIN	Pointer to system block (input buffer). (2 bytes)
00FE	ERCNT	Number of read errors.
00FF	ERC1	Density indicator.

The variables following below are used in IO65.

E700 -	SAVSTACKP	This is a software stack pointer for a routine which saves the A, X and Y registers.
E701 -	STATTOG	Variable which keeps track of which statusline is in use. The most significant bit indicates on or off, 1 = on.
E702 -	CRPX	Variable in which the contents of CURPX is stored before a jump is made to the routine which prints Row and Col on the statusline.
E703 -	CRPY	Variable in which the contents of CURPY is stored before a jump is made to the routine which prints Row and Col on the statusline.
E704 -	CURPX	The position of the cursor in X (horizontal) direction The value of this variable is one less then Col states. The value of this variable can be from $00 to a maximum of $4F.
E705 -	CURPY	The position of the cursor in Y (vertical) direction. The value of this variable is one less then Row states The value of this variable can be from $00 to a maximum of $17. Is something printed on the statusline then its value will be $18.
E706 -	CURP	Flag which indicates whether direct cursor addressing takes place. Also it shows wether the first entered co-ordinate was inside its limits.
E707 -	SCURPX	Temporary store address for CURPX.
E708 -	SCURPY	Temporary store address for CURPY.
E709 -	XTMP	Temporary store address for the X direction pointer.
E70A -	XPSOLD	Old X position for call on POSIT routine.
E70B -	YPSOLD	Old Y position for call on POSIT routine.
E70C -	OUTCHR	Save address for the character to be printed.
E70D -	MAXSTL	Maximum stringlenght in the routine which puts characters on the statusline.
E70E -	HSFLG	Flag which sees to the handshake procedure ^S and ^Q in the RS232 routines.
E70F -	DEVCHOIC	If function ^^O is chosen, this variable will be temporaryly $00. With ^^I this variable will be $01.
E710 -	DEVMODOUT	The active output device bits are located here.
E711 -	DEVMODINP	The active input device bits are located here.
E712 -	SDEVMODOUT	Store address for DEVMODOUT.
E713 -	SDEVMODINP	Store address for DEVMODINP.
E714 -	SDVOUT	Store address for DEVMODOUT in the output routine.
E715 -	SDVINP	Store address for DEVMODINP in the input routine.
E716 -	OUTRET	Default setting for DEVMODOUT. If VIACOM must wait for a handshake to long, or if the memory output device is no longer active, or if device # 6, 7 or 8 is activated without this device being initialized, then this value is put in DEVMODOUT and the original situation is restored.
E717 -	INPRET	Default setting for DEVMODINP. The same story goes here as with OUTRET, but now in relation to the input devices and DEVMODINP.
E718 -	CLSEC	Only during the input routine, while waiting for a depressed key the clock is adjusted on the screen. This variable has the value of the seconds after the clock has been set on time. Then it is checked if CLSEC is still the same as the seconds counter. If so, then will be waited some before the clock is adjusted again.

E719 - FUNENA		Flag which is set after ^^ is depressed, so, when a functionkey is expected.
E71A - FNCEN		If the value of this variable is not $00, the function keys are switched off.
E71B - VIAVRA		If there has been no handshake for 10 seconds, a jump will be made from VIACOM. VIAVRA keeps track of the time passed.
E71C - GRFLG		Grafics flag. Contains $46 if the grafics are on. On the screen grafic instead of ascii characters are printed. This flag is set by sending the character sequence <ESCAPE> and F to the screen output routine, so $1B and $46. The grafics mode is switched of by sending <ESCAPE> and G to the screen routine, so $1B and $47. Then normal ascii characters will be printed again. The variable GRFLG will contain $00 then.
E71D - PNTXLSAVE		Save address for PNTXL.
E71E - PNTXHSAVE		Save address for PNTXH.
E71F - PNTX		Because the designer wished to use as little zeropage addresses as possible some addressing possibilities are not available anymore. For example STA (ADR),Y. To simulate this kind of addressing elsewhere in memory self modifying programs are used. At address PNTX an operationcode is stored. PNTXL and PNTXH now are the absolute addresses which can be changed. At the first address after PNTXH is an RTS. The opcodes which are put in PNTX depents on the use.
E720 - PNTXL		Low byte of the absolute address of the selfmodifying program PNTX.
E721 - PNTXH		High byte of PNTX.
E723 - DMPRAM		Selfmodifying program which is used to put characters on the screen. This is the opcode address, it always contains STA ($8D).
E724 - DUMPL		Low byte of the VDU ram address which will be used when the next character goes to the screenroutine.
E725 - DUMPH		High byte of the above mentioned.
E727 - LDALET		Selfmodifying program which can be used to output a series of characters to the active output devices. Its contents is always LDA ($AD).
E728 - LETADRL		Low byte of the absolute address of the character which this routine gets.
E729 - LETADRH		High byte of the above mentioned.
E72B - FRWR		Selfmodifying program which is used to put a character somewhere in memory with output device # 5, the memory as an output device.
E72C - FRWRL		Low byte of the absolute address of FRWR.
E72D - FRWRH		High byte of the absolute address of FRWR.
E72F - FRRE		Self modifying program which is used to read a character from somewhere in memory with input device # 5, the memory as an input device.
E730 - FRREL		Low byte of the absolute address of FRRE.
E731 - FRREH		High byte of the absolute address of FRRE.
E733 - MONESC		In here is default $1E. Or the character after which the monitor the next character assumes to be a functionkey.

E734 - ACCTL	This value is put in the acia control register when I/O device # 3 is switched on. (RS232). Default this value is set to $BA, which means 2 stopbits and 7 data bits are used. Also the internal baudrate generator is selected which is set for 2400 baud. After a reset the default value is put in ACCTL again.
E735 - ACCMD	When switching on I/O device # 3 the contents of this variable is put into the command register of the acia. Default this value is set to $05. Which means no parity bits are sent or checked, and also the echo mode is deselected. A transmitter control interrupt will occur and the RTS level is active when low. Also the IRQ interrupt of the acia is switched on. Finally bit 0 is made 1 to let the acia work.
E736 - TIMDAT	When the time is printed on the statusline its contents is ':', with the date its '-'.
E737 - SEC1/20	There is an interrupt running via the timer inside the VIA. This timer gives an interrupt every 1/20 second. In this variable the 1/20 parts of one second are counted. This value goes to a maximum of $13, the minimum is $00. So this counter can count to 20.
E738 - TOUTF	Flag which is normally $FF and is set to $00 by the timer when the screen time-out has passed.
E739 - TREMP	When the transmit register of the acia is empty an interrupt will occur. Then this flag is made $00 thus enabling the acia output routine to see whether the transmitter register is empty.
E73A - DECI1	These 3 variables are used for computing from hex to decimal. The computed decimal number is put as decimal bytes in these variables.
E73B - DECI2	Part of decimal number. (See above).
E73C - DECI3	Part of decimal number. (See above).
E73D - LNRL	Low byte of an 16 bit number which is to be converted into decimal and then is put on the statusline just behind Ln:. The screen editor uses this to indicate in which line of the file the cursor is.
E73E - LNRH	High byte of the above mentioned.
E73F - COMP	By sending the ascii-value $19 to the screen routine the screen is cleared from the cursor to the end of the screen. In this 2 byte variable the address of the last character on the screen to be erased is located.
E741 - ESCFLG	This flag is set after an <ESCAPE> is sent to the screen ($1B).
E742 - STATFG	Internal I065 flag which indicates there is to be printed on the statusline.
E743 - INVERS	To get characters in invers video on the screen this flag has to be set. This is done by sending <ESCAPE> and i to the screenroutine. The socalled escaperoutine becoming: $1B $69. To switch off invers video <ESCAPE> and n, so $1B and $6E, is sent to the screenroutine.
E744 - INVERSS	This is the save address for INVERS during some routines.
E745 - INVST	Default its contents is $80 as the statusline is default switched to invers. Does one want an ordinary statusline its value has to be changed into $00. Do use afterwards the command CLEAR to remake the screen.

At the next addresss are the vectors located which point to the interrupt routines. Remember that the addresses in this table are one less then the startaddress of an interrupt routine. The interrupt routine must end with an RTS, not an RTI. The registers which were put on stack by the main interrupt routine now can be retrieved by the same program.

E750 - INTV1	interrupt 1 vector,	T1 VIA 1, System clock
E752 - INTV2	interrupt 2 vector,	T2 VIA 1
E754 - INTV3	interrupt 3 vector,	CB1 VIA 1
E756 - INTV4	interrupt 4 vector,	CB2 VIA 1
E758 - INTV5	interrupt 5 vector,	SR VIA 1
E75A - INTV6	interrupt 6 vectot,	CA1 VIA 1, Keyboard
E75C - INTV7	interrupt 7 vector,	CA2 VIA 1
E75E - INTV8	interrupt 8 vector,	T1 VIA 2
E760 - INTV9	interrupt 9 vector,	T2 VIA 2
E762 - INTV10	interrupt 10 vector,	CB1 VIA 2
E764 - INTV11	interrupt 11 vector,	CB2 VIA 2
E766 - INTV12	interrupt 12 vector,	SR VIA 2
E768 - INTV13	interrupt 13 vector,	CA1 VIA 2
E76A - INTV14	interrupt 14 vector,	CA2 VIA 2
E76C - INTV15	interrupt 15 vector,	ACIA, RS232
E76E - INTV16	interrupt 16 vector,	Software break, MON65

E770 - SVAINT	Accu save address in interrupt routine.
E771 - COUD	When the keyboard routine is asking for input, then the cursor is switched on. Which cursor is defined by a register in the CRTC. The value COUD is put into the CRTC. If the cursor mode is changed then the new CRTC value is also put in COUD.
E772 - TOUTIL	IO65 is equiped with an automatic screen-off utility. Which means that if for some time no key has been depressed the screen automaticly will be extinguished. As soon as a key is depressed again the screen will be switched on again. For this variable an 16 bits variable is used. That one is called: TOUTH and TOUTL. The default value is 1800 seconds. (30 minutes). Right after a key is being depressed the default value has to be put in TOUTH and TOUTL again. This default value is fetched from TOUTIL and TOUTIH. To be able to adjust this time the value is fetched from a variable. The DOS program DPTIME adjusts these values. Using this program the screen-off time can be adjusted from 0 to 65535 seconds.
E773 - TOUTIH	See under TOUTIL.
E774 - TOUTL	See under TOUTIL.
EE75 - TOUTH	See under TOUTIL.

E776 - WRBEG	To be able to use the memory for an output device, this pointer has to point to the start of the memory area to which data has to be transferred. WREND points to the end of this area. These values have to be set before output device # 5 is initialized. Such can be accomplisched by writing an auxiliary program or, by the monitor and also from the DOS command mode using the command MEMFILL.
E778 - WREND	See under WRBEG.
E77A - REBEG	To be able to use the memory for an input device, this pointer has to point to the start of the memory area from where data has to be read. REEND point to the end of this area. The values have to be set before input device # 5 is initialized. Such can be accomplisched by writing an auxiliary program or, by the monitor and also from the DOS command mode using the command MEMFILL.
E77C - REEND	See under REBEG.
E77E - DATUPD	If an interrupt of the timer, used for the clock, occurs the relevant registers are adjusted. Only during the keyboard input routine the time, stated on the statusline, is adjusted. The date is not copyed again and again from his registers to the statusline. This is only done when a day-transition occurs, so at 12 pm. At that moment the flag DATUPD is set to $FF and the routine which adjusts the time on the status-line then knows also the date has changed.
E77F - DAY	Variable in which the date is saved. This value is changed at midnight by the interrupt routine. A RESET does not change this value.
E780 - MONTH	Variable in which the month is kept up to date. This value is changed by the interrupt routine at midnight and only after the last day of the month. Months with 30 or 31 days and leap-years are included in the routine. After a RESET the value of this variable remains the same.
E781 - YEAR	Variable in which the year is stored. Is changed only at midnight 31 december, again; by the interrupt routine. The value is not changed after a RESET.
E782 - HOURS	In here the hours of the clock which runs on interrupt basis are stored. A RESET does not change this value.
E783 - MINUTES	In here the minutes of the clock which runs on interrupt basis are stored. After a RESET its value is not changed.
E784 - SECONDS	In here the seconds of the clock which runs on interrupt basis are stored. A RESET does not change its value.
E785 - DV04VEC	Output device # 4 changes its vector after the first character has been sent. This is done because after the first character has to be waited for a handshake signal before the next character can be sent off. Without this the output device would be waiting for a handshake signal before a character has been sent.
E786 - DV14VEC	To achief uniformity also the input device goes via a vector.

E789 - DV06VEC Output device # 6 vector. To be defined by the user. If this output device is activated without the vector has been defined, then the in and output devices are put to their default state.
Also see chapter 1 about I/O devices.

E78B - DV16VEC Input device # 6 vector. To be defined by the user. If this input device is activated without the vector has been defined, then the in and output devices are put to their default state.
Also see chapter 1 about I/O devices.

E78D - DV07VEC Output device # 7 vector. See under DV06VEC.

E78F - DV17VEC Input device # 7 vector. See under DV16VEC.

E791 - DV08VEC Output device # 8 vector. See under DV06VEC.

E793 - DV18VEC Input device # 8 vector. See under DV16VEC.

E795 - ASAVESTACK Software stack for accu contents. 8 bytes.

E79D - XSAVESTACK Software stack for X register contents. 8 bytes.

E7A5 - YSAVESTACK Software stack for Y register contents. 8 bytes.

E7AD - START Pointer which keeps track of the start of the video-ram on the screen. The video ram is located from $E800 to $EFFF. However the upper-left-position on the screen (1,1) can only be found at address $E800 after an reset. After a scroll-up this is $E800 + $50 = $E850 already.

E7AF - UNRINT When an interrupt is generated which is not recognized the interrupt routine jumps via this vector to the error-report 'interrupt ignored'. So if one writes an program oneself, in which a interrupt is generated in a pheripheral or a timer then this vector can be derouted via a selfmade interrupt routine which checks whether the interrupt came from such device. If not, still the jump to the error-report 'interrupt ignored' has to be made.

E7B1 - NMIVECTOR The systemvector NMI at address $FFFA point via an indirect jump in the jumptable to this address. The NMI vector points to a dummy RTI.

E7B3 - BRKVECTOR When a break is detected by DOS65 then the routine is started to which BRKVECTOR points.

E7B5 - IRQVECTOR The system vector IRQ at $FFFE points via an indirect jump in the jumptable to this address. So, does an IRQ occur and has the interrupt flag in the processor not been set then the interrupt routine is started to which IRQVECTOR points.

E7B7 - KEYPNT The keyboard works with interrups. Which means that when a key is depressed its value is stored in the keyboard buffer. After each key-input the variable KEYPNT is incremented by one. When the inputroutine now asks for a key then the first character from the buffer is taken, KEYPNT is decremented and the buffer contents moves one place.

E7B8 - KEYBUF The keyboard input buffer, which can contain 40 characters, uses these addresses.

E400 - SYSB The DOS65 bootstrap needs 2 buffers of 256 bytes. This buffer is used for 'system block'.

E500 - TSLB The second buffer is used for 'tsl block'.

E000 -	PAD	Data and data direction A.	PIA on FDC print.
E001 -	PAC	Command register.	
E002 -	PBD	Data and data direction B.	
E003 -	PBC	Command register.	
E004 -	CMR	Command register.	FDC on FDC print.
E004 -	STR	Status register.	
E005 -	TKR	Track register.	
E006 -	SCR	Sector register.	
E007 -	DTR	Data register.	
E100 -	VAPBD	Port B data.	VIA 1 on the CPU extention print.
E101 -	VAPAD	Port A data.	
E102 -	VAPBDD	Port B data direction.	
E103 -	VAPADD	Port A data direction.	
E104 -	VATACL	T1, latch low, counter low.	
E105 -	VATACH	T1, counter high.	
E106 -	VATALL	T1, latch low.	
E107 -	VATALH	T1, latch high.	
E108 -	VATBCH	T2, latch low, counter low.	
E109 -	VATBCH	T2, counter high.	
E10A -	VASR	Shift register.	
E10B -	VAACR	Auxiliary control register.	
E10C -	VAPCR	Peripheral control register.	
E10D -	VAIFR	Interrupt flag register.	
E10E -	VAIER	Interrupt enable register.	
E10F -	VAADN	Port A data, no handshake.	
E110 -	VBPBD	Port B data.	VIA 2 on the CPU extention print.
E111 -	VBPAD	Port A data.	
E112 -	VBPBDD	Port B data direction.	
E113 -	VBPADD	Port A data direction.	
E114 -	VBTACL	T1, latch low, counter low.	
E115 -	VBTACH	T1, counter high.	
E116 -	VBTALL	T1, latch low.	
E117 -	VBTALH	T1, latch high.	
E118 -	VBTBCH	T2, latch low, counter low.	
E119 -	VBTBCH	T2, counter high.	
E11A -	VBSR	Shift register.	
E11B -	VBACR	Auxiliary control register.	
E11C -	VBPCR	Peripheral control register.	
E11D -	VBIFR	Interrupt flag register.	
E11E -	VBIER	Interrupt enable register.	
E11F -	VBADN	Port A data, no handshake.	
E130 -	RECREG	Receiver register.	ACIA on the CPU
E130 -	TRAREG	Transmitter register.	
E131 -	ACIASR	Status register.	
E132 -	ACICMD	Command register.	
E133 -	ACICTL	Control register.	
E140 -	CRTCAR	Address register.	CRTC on the VDU-print.
E141 -	CRTCRF	Register file.	