Avise5 Kernel Operator Glossary version 5.9 ©2023 Wolfgang Schemmert 10.Nov.2023

TOS is "Top of Stack". TOSH is high byte(bits 8-15), TOSL is low byte(bits0-7)
NOS is 2nd ("next") element of stack

Basic Data Stack Operators and Arithmetic Operators:

DROP	(x2 x1 -- x2)	delete TOS
DUP	(x -- x x)	duplicate TOS
SWAP	(x2 x1 – x1 x2)	exchange NOS and TOS
OVER	(x2 x1 – x2 x1 x2)	NOS is copied over TOS and new TOS then
ROT	(x3 x2 x1 -- x2 x1 x3)	rotates 3rd stack entry up
-ROT	(x3 x2 x1 – x1 x3 x2)	rotates TOS down to be 3rd on stack
PICK	(xn x(n-1). .x2 x1 n -- xn x(n-1). .x2 x1 xn) Example: (x3 x2 x1 3 -- x3 x2 x1 x3)	drops TOS,keeps n in mind.**Copies** xn up to new TOS.Count starts at x1 "after" indices shown in stack diagram are assigned as before operation 0 PICK ignored silently, 1 PICK performs DUP, 2 PICK performs OVER
ROLL	(xn x(n-1). .x2 x1 n -- x(n-1)..x2 x1 xn) Example: (x4 x3 x2 x1 3 – x4 x2 x1 x3)	drops TOS, keeps n in mind. **Pops** xn up to TOS, deleted at old position Count starts at x1 0 ROLL, 1 ROLL ignored silently. Wouldn't have an effect. 3 ROLL performs ROT, 4 ROLL 4 ROLL performs 2SWAP
INJECT	(x(n+1) xn x2 x1 n -- x(n-1) x1 xn…x2 Example: (x4 x3 x2 x1 3 – x4 x1 x3 x2)	drops TOS, keeps n in mind. Pushes NOS down to aove the n-th item. n counts stack elements starting from x1. 0 INJECT, 1 INJECT ignored silently. Wouldn't have an effect. 2 INJECT performs SWAP, 3 INJECT performs –ROT
+	(x2 x1 -- x2+x1)	signed 16bit addition. 0x7FFF-0x8000 crossover handled unsigned
1+	(x -- x+1)	add 1 to TOS. 0x7FFF-0x8000 crossover handled unsigned
-	(x2 x1 -- x2-x1)	signed 16 bit subtraction. 0x7FFF-0x8000 crossover handled unsigned
1-	(x -- x-1)	subtract 1 from TOS, 0x7FFF-0x8000 crossover handled unsigned
*****	(factor factor -- factor*factor)	signed 16 bit multipication. Error message if (result is > 16bit signed int.) For 32bit result use ***/** with divisor = 1
/	(dividend divisor ----- dividend/divisor)	signed 16 bit division
MOD	(dividend divisor -- remainder)	The remainder has the same sign as the dividend
***/**	(factor factor divisor -- quotientL quotientH)	first multiply factor*factor, then divide the intermediate result by divisor 32 bit result of multipication.
***/MOD**	(factor factor divisor -- remaind quotL quotH)	same as ***/** plus 16 bit remainder on 3rd stack element. The remainder has the same sign as the multiplication result
ABS	(x -- abs(x))	transform TOS into it's absolute value
SHIFT	(x2 x1 – x2 << x1) (x2 x1 – x2 >> **-**x1)	X1 positve: shift left, shift range X1:0 ... 15 X1 negative: shift right, shift range X1:0 ...-15 **Unsigned 16bit shift operation**. May be combined with CARRY to ROL or ROR. For signed SHIFT, user can handle bit15 with extra action.
AND	(x2 x1 -- x1&x2)	bitwise AND
OR	(x2 x1 – x21|x1)	bitwise OR
XOR	(x2 x1 -- x2^x1)	bitwise XOR
NOT	(x ---xmodified)	bitwise 0/1 inversion (one's complement of x)
+/-	(x -- **-x**)	change sign of TOS (two's complement of x)
BSET	(x bitN -- xmodified)	bitN of x is set to 1 (bitN 0..15)
BCLR	(x bitN -- xmodified)	bitN of x is cleared (set to 0) (bitN 0..15)
BTST	(x bitN -- bitvalue))	bitvalue = boolean value (=0 or 1) of bitN

RANDOM	(--- random)	returns a 16 bit random number on TOS ("xorshift16" algorithm)
CARRY	(-- carry of last action)	Overflow of arithmetic operations is collected in the background.and is put on stack with CARRY. Helpul for 32 bit operations Used by SHIFT, too. Logic CARRY combination w. result "rotates" bits

Comparison Operators

==	(x2 x1 -- 1|0)	1 if x2 == x1 x2 and x1 are removed from stack
!=	(x2 x1 -- 1|0)	1 if x2 != x1 x2 and x1 are removed from stack
0=	(x -- 1|0)	1 if x == 0 x is removed from stack
>	(x2 x1 -- 1|0)	1 if x2 > x1 x2 and x1 are removed from stack (signed compare)
U>	(x2 x1 -- 1|0)	1 if x2 > x1 x2 and x2 are removed from stack (**unsigned** compare)
>=	(x2 x1 -- 1|0)	1 if x2 >= x1 x2 and x1 are removed from stack (signed compare)
<	(x2 x1 -- 1|0)	1 if x2 < x1 x2 and x2 are removed from stack (signed compare)
U<	(x2 x1 -- 1|0)	1 if x2 < x1 x2 and x2 are removed from stack (**unsigned** compare)
<=	(x2 x1 -- 1|0)	1 if x1 <= x1 x2 and x2 are removed from stack (signed compare)

Memory Access Operators

W	(word addr ---)	writes word into addr. Replaces Standard Forth ! (typing is more easy)
WDM	(byte addr ---)	write BYTE into ATmega Data Memory (TOSH ignored) if (addr <=63) <addr> is increased by 0x20 (AVR instruction special)
WREEP	(word addr ---)	write word into EEProm (used by RECON or User Function modification)
R	(addr --- word)	reads word from addr. Replaces Standard Forth @ (typing is more easy)
RDM	(addr --- byte)	read BYTE from ATmega Data Memory (TOSH=0)(if (addr <=63) <addr> is increased by 0x20 (AVR instruction special)
VAR	(---) IMMEDIATE	creates a global VARiable with initial value 0. Max 32 VAR possible If a VARiable is called by name, it's value ADDRESS is put on TOS Syntax: VAR VALLY <return>
VA	(--- addr)	kernel preinstalled variable. May be written by background keystroke CTRL_A(=ASCII code1). Else behaviour like user installed VARiable
VB	(--- addr)	kernel preinstalled variable. May be written by background keystroke CTRL_B(=ASCII code2). Else behaviour like user installed VARiable
VC	(--- addr)	kernel preinstalled variable. May be written by background keystroke CTRL_C(=ASCII code3). Else behaviour like user installed VARiable
ARRAY	(index --- address)	16bit integer array in SRAM, number of entries CPU depending Puts the corresponding value address on TOS. Else used like VAR Syntax: 23 3 ARRAY W writes 23 into 3rd ARRAY index memory
CONST	(x ---) IMMEDIATE	creates a CONSTant with initial value x. If a CONSTant is called by name, it's VALUE is put on TOS. Syntax: 234 CONST CONNY <return>
RECON	(x ---) IMMEDIATE	replaces value of named CONST with value of TOS (next use and in prevoiusly compiled User Functions). Syntax: 789 RECON CONNY<ret>
CODEOF new v5.9	(--- FlashAddr) IMMEDIATE	Syntax: CODEOF <KernelOp name or UserFunction name> returns the start address of the function binary code in Flash memory Examples: **: X…;** **CODEOF X VC W VCALL** or **CODEOF X MEM**
VCALL new v5.9	(---)	starts execution of binary code w. start address stored in Kernel Var VC. Primarily indended for experimental operations: Call newer UserFunctions from older UserFunction frames. With background OP VC, program flow may be changed at runtime.

Terminal Operators

KEY?	(--- number)	returns the actual number of bytes in terminal input buffer, = 0 if none
KEY	(--- byte)	execution blocks until a terminal byte is received, gets returned on TOS Unblock e.g. w. CTRL_K: KEY DUP 0xB != IF <your code> ELSE DROP THEN
EMIT	(byte ---)	the lowest 8 bit of TOS are sent via terminal as raw byte

.	(number ---)	TOS is sent as ASCII text using actual SYSTEM NUMBER BASE
		if DECIMAL: the content of TOS is sent as 16 bit **signed** integer.
		if HEX: the content of TOS is sent as 16 bit **unsigned** integer.

.H	(number ---)	TOS is sent as ASCII text HEX formatted - 16 bit **unsigned** integer.
.D	(number ---)	TOS is sent as ASCII text DECIMAL formatted - 16 bit **signed** integer.
.U	(number ---)	TOS is sent as ASCII text DECIMAL formatted - 16bit **UNsigned** integer
." ...."	(---)	send the text between " ...." as byte stream via terminal .Max 78 bytes. Between P" and the first text letter MUST be a SPACE which is not sent Differing from other FORTH, string is 0 terminated. Recommended for printable char only !
DZ	(---) IMMEDIATE BACKGRND	set system number base DECIMAL In this case hex numbers can be entered with leading 0x or $. Even 0x–… is accepted: 0x-FFFFFFFF = 1
HX	(---) IMMEDIATE BACKGRND	set number base HEXADECIMAL Decimal numbers can be entered with leading &. The number base can be changed by DZ and HX token in compile mode, but is reset to the system number base at compile end (;)

NOTE: after chip programming, number base is DECIMAL. Change to HEX via terminal: "0 0xD WREEP", **change back to DECIMAL**:"0xFF 0xD WREEP". Gets effective after reset. During session modified with HX,DZ

Serial I/O Operators
(only supported by ATmega644, ATmega1284 and ATmega32U4 - if USART1 assembled)

U1BAUD	(baudrate --)	baudrate = (CPUclock(Hz) / 16 / BaudRate(Hz)) **– 1** (must be almost integer !) **-1 U1BAUD=OFF** Examples: 16MHz/16/38400)-1 = 25 921600/16/57600)-1 = 0
RX1?	(--- 0 | 1)	returns the actual number of bytes in USART1 input buffer, = 0 if none
RX1	(--- byte)	blocks until I/O byte is received, byte gets supplied on TOS .
TX1	(byte ---)	the least 8 bit of TOS are sent via serial I/O as **raw byte** (EMIT counterpart)
NOBAK	(0|1 ---)	1 NOBAK disables reaction on background ops: ESC, CTRL_A,B,C, CTRL_R, CTRL_S - 0 NOBAK allows(default). **CPUs w. USART1 only**

Time Operators

MS	(mstime ---)	starts countdown of 'mstime' milliseconds. Calling User Function BLOCKS until MS-countdown is finished to zero. Blocking can be released with terminal input CTRL_R (ASCII code 0x12)
TIX	(time ---)	starts NON-BLOCKING countdown of "time" steps of 1/10 seconds.
TIME	(--- time)	returns remaining 1/10s steps of TIX-countdown (timeline, max 6553 s)

Peripheral Operators

IP	(portPin ---)	sets addressed port pin as INPUT with CPU internal PULL-UP resistor PortPin is entered as HEX byte. A0 is decimal 160,B3 is decimal 179 etc Syntax: 0xB3 IP configures PB3 as input with pull-up
IZ	(portPin ---)	sets addressed port pin as INPUT with HIGH IMPEDANCE PortPin is entered as HEX byte, detail see at 'IP'
OH	(portPin ---)	sets addressed port pin as push/pull OUTPUT HIGH PortPin is entered as HEXbyte, detail see at 'IP'
OL	(portPin ---)	sets addressed port pin as push/pull OUTPUT LOW PortPin is entered as HEXbyte, detail see at 'IP'
PH	(portPin ---)	changes ATmega data register "PORT" bit only (saves runtime) i.e. change from OL to OH or from IZ to IH portPin is entered as HEXbyte, detail see at 'IP'
PL	(portPin ---)	changes ATmega data register "PORT" bit only (saves runtime) i.e. change from OH to OL or from IH to IZ portPin is entered as HEXbyte, detail see at 'IP'
RDI	(portPin -- 0|1)	returns the digital level at port pin. Independent of actual configuration and user availabilty of this pin. PortPin is entered as HEXbyte
AIN	(PinNo -- value)	reads the 10 bit level of addressed ADC. VREF pin config. see AREF ATmega168,328: PinNo 0 ... 6 = PC0 … PC5 ATmega644,1284,32: PinNo 0 ... 7 = PA0 … PA7 ATmega32U4:PinNo0,1=PF0,PF1, PinNo2 … 5 = PF4 .. PF7 Called first, the A/D converter gets initalized. Stays active, takes current!

AREF	(0|1|2 --)	configures VREF pin behavour **and deactivates ADC**. Default = 1 Take care of your HW wiring of this pin to avoid overload !! 0=external, 1=internal Vcc, 2=internal 2.56V (ATmega168/328:1.1V)
CNT	(--- count)	return number of upwards counted negative pulses at CMT pin CNT pin= PD2, except ATmega32U4: CNT pin = PD0
ENC	(--- count)	return number of up/down counted pulses at CNT pin Count **UP** if level of ENC pin=1, Count **DOWN** if level of ENC pin=0 ENC pin= ATmega168,328:PD5. ATmega644,1284:PB1. ENC pin= ATmega3:PD4, ATmega32U4:PD4.
CZ	(---)	reset counter
CNTX	(--- count)	return number of uwards counted pulses at CNTX pin **CNTX and ENCX not implemented for ATmega168, ATmega 328** CNTX pin= ATmega644,1284:PB2 ATmega32:PD3, ATmega32U4:PD1
ENCX	(--- count)	return number of up/down counted pulses at CNTX pin Count **UP** if level of ENCX pin=1,Count**DOWN** if level of ENCX pin=0 ENCX pin= ATmega644,1284:PB1, ATmega32:PD4, ATmega32U4:PB7
CXZ	(---)	reset counter X not implemented for ATmega168, ATmega 328
PWM1	(timing --)	8 or 10 bit PWM output: ATmega168,328: PB1, ATmega644,1284,32: PD5, ATmega32U4: PB6 **If TOSH<0x10**: 8 bit resolution, TOSL is high phase portion. bits8,9,10 of TOS are prescaler (1…5). Freq is CPU speed depending **Else**: 10bit resolution. Bits 9-0 are high phase portion. Bits12,13,14 of TOS are prescaler (1…5). Freq is CPU speed depending PWM and WAVE generation exclusively switched off with OL,OH,PZ,PH
WAVE	(duration ---)	PWM with more precise adjustment than PWM1. Same output pins TOS is duration of pulse, strongly CPU speed depending. WAVEHI must be set before and WAVE>WAVEHI, else default config. Very short pulses are replaced by a minimal default.
WAVEHI	(highphase ---)	TOS is duration of high phase duration of WAVE
PWM2	(timing--)	8 bit PWM exclusively. **NOT available for ATmega32U4**. Output: ATmega168,328: PD3, ATmega644,1284,32: PD7 TOSL is high phase portion. bits8,9,10 of TOS are prescaler (1…7). Freq is CPU speed depending PWM2 generation exclusively switched off with OL,OH,PZ,PH
PWM3	(timing --)	8 or 10 bit PWM-**exclusively** implemented at **ATmega32U4** output: PC6 Setup and switch-off see PWM1
INITSPI	(byte ---)	intializes an SPI master with **8 bit** transfer length TOSH is ignored, TOSL must be coded as follows High nibble: **0,1,2,3** = Motorola **modes 0,1,2,3**. (sometimes referenced as (0,0) (1,0) (0,1) (1,1)) Low nibble defines clock rate (0---6). Strongly CPU clock dependent **/CS must be handled separately** by user with any pin, /SS preferred. **Used I/O pins:** (/SS is restricted by hardware:must be OUT or IN/pullup) ATmega168,328: /SS:PB2, MOSI=PB3, MISO=PB4, SCK=PB5 ATmega32,644,1284: /SS:PB4, MOSI=PB5, MISO=PB6, SCK=PB7 ATmega32U4: /SS:PB0, MOSI=PB2, MISO=PB3, SCK=PB1
SPI	(TxByte -- RxByte)	One data BYTE is transferred (transmit and received), TOSH is ignored. **SPI must be initialized before** else error message

System Operators

.S	(---) IMMEDIATE	sends all entries of the DataStack, TOS last followed by the actual values of VA, VB, BC. Next line actual values of first 8 global VARiables Can be triggered as background op. with CTRL_S(ASCII oode 0x13)
.RS	(---) IMMEDIATE	sends upper 4 items of CPU Return Stack, top first. Followed by CPU Program Counter value before call of .RS
OPS	(---) IMMEDIATE	Sends a list of all Kernel Operator names

USER	(---) IMMEDIATE	Sends a list of all actually compiled User Functions. First the function code start address in Flash is sent, next the name. For VARiables, the storage address and its actual value is sent. For CONSTants, the actual value is sent.
SEE	(---) IMMEDIATE	Syntax: SEE <User Function name> The User Function binary code is de-compiled as ASCII text. first line EEProm: followed by start address (count byte) of code entry.
MEM	(0|1 --) IMMEDIATE	**0 MEM** sends approx. remaining memory space (decimal numbers) S=Symbol Table, C=Flash code, V = number of remaining variables **1 MEM** additionally sends content of memory actually worked on (hex). First EEProm symbol table, then Flash page. CBUF=temp SRAM buffer **<FlashAddress> MEM** sends Flash block. 1st line contains this address
FLUSH	(xn…x1 --) IMMEDIATE	deletes all items from the Data Stack, removes garbage, reorganizes
FORGET	(---) IMMEDIATE	Syntax: FORGET <User Function name> The named User Function **and all newer ones** are deleted
ABORT	(---) IMMEDIATE	Clears all stacks and resets all system parameters. Executed automatically in case of user input / compiler / runtime errors
AUTOEXE	(---) IMMEDIATE	Syntax: AUTOEXE <User Function name> The specified UserFunction is automatically executed at system start. Effect removed/disabled by “AUTOEXE<return>” (or invalid User Function)
SLEEP	(---)	sets CPU into 'low power" sleep mode. **Awake with LevelChange** at **PD0** (= awake by terminal) or **PD4**, **except** ATmega32: **LOW** pulse at **PB2**. **Not** available for **ATmega32U4**
SPEED	(1|2 ---)	1: CPU clock = 921.6kHz, 2: CPU clock = crystal freq. Same baud rate. Implemented only for firmware with 1.8432MHz and 7.3728MHz crystal
LED	(0|1|2 ---)	onboard LED behaviour. 0 LED (default): LED reflects terminal input 1 LED permanently OFF, 2 LED: permanently ON
QT	(0|1 ---)	1 QT suppresses ASCII feedback at terminal input, 0 QT allows(default). Sometimes a problem when source file loaded fast at low CPU speed.
//	not explicitly listed as Kernel OP !	// initiated **comments are suppressed until end of the line** (=CR) Comment is deleted automatically directly in the "query" input handler In contrast to comment handling implemented in other Forth standards, no leading and trailing SPACE is needed. Character sequence **// may not be included** in User Function names ! **Suppression of /*…*/** can be managed with DTERM terminal software.

Structuring Operators

IF	(-- addr) IMMEDIATE, COMPILEONLY	starts compilation of a IF..ELSE..THEN conditional. Compiles doIF Essentially provides a placeholder for jump address and sends its code address via stack to ELSE or THENI
ELSE	(addr -- addr) IMMEDIATE, COMPILEONLY	compiles doELSE, the middle runtime part of a IF...ELSE...THEN conditional. Essentially compiles its code address into the placeholder behind doIF and provides a placeholder for jump address. The stack entry with the code address provided by IF is replaced by a stack entry that contains the code address of doELSE
THEN	(addr --) IMMEDIATE, COMPILEONLY	compiles termination of IF or IF….ELSE Essentially it compiles the jump target address into the placeholder behind doIF or doELSE. Leaves no token in User Function code, is unvisible in SEE output, but it's target address is 2nd word of machine code of doIF or doELSE
DO (=BEGIN in traditonal Forth)	(-- addr) IMMEDIATE, COMPILEONLY	starts compilation of a DO...WHILE, DO..UNTIL or DO…AGAIN loop. puts the actual compiler code pointer on TOS for WHILE, UNTIL, AGAIN
UNTIL	(addr --) IMMEDIATE, COMPILEONLY	finishes compilation of a DO ... UNTIL loop: compiles doUNTIL plus one Flash word with the loopback adress provided by DO. Compiles RS-1UP
WHILE	(addr --) IMMEDIATE, COMPILEONLY	finishes compilation of a DO .. WHILE loop: compiles doWHILE plus one Flash word with the loopback adress provided by DO. Compiles RS-1UP
AGAIN	(addr --) IMMEDIATE, COMPILEONLY	finishes compilation of a DO .. AGAIN loop: compiles doAGAIN plus one Flash word with the loopback adress provided by DO. Compiles RS-1UP

FOR (replaces DO in traditonal Forth)	(-- addr) IMMEDIATE, COMPILEONLY needs to be compiled before: doLIT<start index> doLIT<stop index> VAR R sequence	initaliazes a FOR… NEXT loop. Though not obvious, essentially it does: The doFOR token is compiled, next a word with zero content is compiled into Flash. Then FOR puts the Flash compile pointer **on TOS for LOOP**. But **ahead of FOR, 3 items must be compiled** to be used by doFOR: First values of <start> and <stop> loop index compiled as doLIT<start> doLIT<stop>, followed by a <VARaddress>. The VAR will hold the actual loop index. This way the loop index may be changed during looping. **Syntax:** <start> <stop> <variable name> FOR … code of loop …LOOP
LOOP	(addr --) IMMEDIATE, COMPILEONLY	finishes compilation of a FOR ... LOOP loop: It compiles doLOOP, next 1 Flash word with (TOS+2)=loopback addr. Then it's (code address+2) is written into the zero Flash word of doFOR.Finally RS-4UP is compiled
BREAK	(---) COMPILEONLY	Leaves the actually performed loop immediately: jumps to RS-1UP or RS-4UP, where the looop specific Return Stack entries are removed.
CONTI	(---) COMPILEONLY	starts next "looping" now:In a FOR.. loop, the loop index gets updated. If finish, loop is terminated. In a DO.. loop implicitly doAGAIN is executed.
RETURN	(---) COMPILEONLY	Return from User Function,**1 Return Stack level only**! See RS1-UP,RS-4UP

Compiler Operators

: (colon)	(---) IMMEDIATE	Starts a new compilation - separated by SPACE - the intended name of the new User Function is followed. Hereafter follows the sequence of Kernel Operator names and User Function names, which shall be executed by the new function, each separated by SPACE.
; (semis)	(---) IMMEDIATE	Terminates compilation, **always last token of a User Function**. Compiles instr 0x8 95. Helpful when reading User Function code (op MEM)

Runtime Operators (Runtime Primitives) compiled by other operators,control their runtime behaviour **IF** compiles doIF, **ELSE** compiles doELSE, **WHILE** compiles doWHILE, **UNTIL** compiles doUNTIL, **AGAIN** compiles doAGAIN, **FOR** compiles doFOR, **LOOP** compiles doLOOP. **No direct handling or access by user**

doLIT	(---) COMPILEONLY	compiled by a number literal. In the compiled thread, at runtime doLIT puts the content of the next code word (=number value) on TOS.
doSTR	(---) COMPILEONLY	compiled by ." In the compiled thread it is followed by the bytes to be transmitted. The byte sequence is 0 terminated,so only printable characters are safe
doIF	(criterion --) COMPILEONLY	if 'criterion' == 0, a jump is performed to the Flash address, which is contained in the Flash word following this token. Else linear progress. Performs same code as doUNTIL. Separated for better code readability
doELSE	(--) COMPILEONLY	performs an unconditional jump to the Flash address, which is contained in the Flash word following this token. Performs same code as doAGAIN. Separated for better code readability
doUNTIL	(criterion --) COMPILEONLY	if 'criterion'== 0, a jump is made to the Flash address, which is contained in the next Flash word. Else program flow continues linearly via RS-1UP
doWHILE	(criterion --) COMPILEONLY	if 'criterion' != 0, a jump is made to the Flash address, which is contained in the next Flash word. Else program flow continues linearly via RS-1UP
doAGAIN	(--) COMPILEONLY	performs an unconditional jump to the Flash address, which is contained in the next Flash word.
doFOR	(start stop addr ----) COMPILEONLY	stack parameters must be compiled ahead of FOR (or available else). <start>:loopstart index, <stop>:loopend index,VAR <addr>:for loop index **doFOR** is only active before first "looping" and performs following action: get <start> from TOS and write it into variable at addr (first loop index) get <stop> from TOS, compare with <start> **if up or down count**. Insert this direction flag as bit15 of <addr>. Index step only +/-1 possible Push <stop> word on Return Stack and next push modified <addr> increase doFOR parameter address, push the content on ReturnStack (doFOR parameter contains the BREAK addr (see FOR and LOOP) Loopback later goes behind doFOR parameter (jumps back to two Flash words behind doFOR instruction code).

doLOOP	(---) COMPILEONLY	pop loop params from Return Stack (BREAK addr, VAR addr with loop index, STOP value) extract up/down flag, update count index in variable. Push all items back on CPU Return Stack.Check if loop is finished or not **if loop goes on:** read loopback addr from Flash word after doLOOP instruction code and jump back to behind doFOR parameter. i.e. program flow continues behind doFOR token + 2 Flash words. **if loop finished**: subsequent RS-4UP is executed, which removes loop parameters from CPU Return Stack. Next linear progression goes on.
RS-1UP	(---) COMPILEONLY	removes 1 item from CPU Return Stack. Automatically compiled behind doUNTIL, doWHILE, doAGAIN as target of BREAK.
RS-4UP	(---) COMPILEONLY	removes 4 items from CPU Return Stack. Automatically compiled behind doLOOP as target of BREAK. **To leave a User Function completely from inside a loop, RS1-UP or RS4-UP must be compiled before RETURN**, correspondingly several times to return completely from nested loops.

Following operators are exclusivelly supported by ATmega32U4

VID and **PID**	(---) IMMEDIATE	Specifiy the VID/PID used for USB communication. To avoid errors, enter VID or PID in HEX,4 digits w. 0x + leading zeroes New VID/PID is active after system restart. **Be careful not to shoot USB access** (appropriate Windows .inf file !) By default, the VID/PID and Windows .inf of Arduino Micro is used. For non-evaluation and public use, a legal VID/PID must be configured.

Hotkeys and Background features used by 'Avise 5

Key sequence ASCII "//" ignore rest of actual line as **comment**. Details see above at "System Operators"

Avise default number base: After chip programming, default number base is DECIMAL.
Change to HEX via terminal: "0 0xD WREEP". Change back to DECIMAL:"0xFF 0xD WREEP".
Change is effective after reset and next powerON. Enter WREEP data carefully not do destruct neighbour bytes!

Generally, 'Avise' accepts only 'printable characters'. These are: ASCII codes between hex20 and hex7E. Execptions are: 'carriage return' =hexD (<RETURN>; key) and 'backspace' = 8, but no control characters and country specific special characters like german 'Umlaute' are accepted. If LineFeed (0xA) is needed for terminal operation, change value of CRR in source code and re-assemble.

Nevertheless, following ASCII codes have a special meaning for 'Avise' - they are already filtered out in the terminal input handler:

The 'ESC' key (ASCII code hex1D, decimal 27) calls ABORT directly out of the serial receiver interrupt handler. This feature is very useful to cancel a never-ending AUTOEXE function and return control to the user console. Performing a reset only would restart the AUTOEXE and clear values of VARiables.

Key combination 'CTRL_S' (ASCII code hex13, decimal19) performs the .S and .RS commands as background process and returns the actual **data stack**, some VARiables and actual **CPU Return** Stack and PC. Any executing User Function is not interrupted or changed. **In practice useful for debugging only**

Key combination 'CTRL_R' (ASCII code hex12, decimal18) releases any blocking MS timer immediately. The actual timer countdown is kept and can further be evaluated with TIME.

Key combinations 'CTRL_A or CTRL_B or CTRL_C' (ASCII codes 1,2,3), followed by ASCII text of a valid number value, terminated by <RETURN> key: this numeric input is **stored immediately into the kernel based variable VA or VB orVC**. Without leading prefix, then the actual system number base is assumed. May be overridden with leading 0x or $ (number is interpreted hex) or with leading & (number is interpreted decimal). This is handled in the background, possible influence on the executing User Function (if VA,VB,VC is read). This feature is intended to modify the flow of this User Function interactively due to change of variable value.

ASCII codes hex7F and higher are ignored by 'Avise'.

contact: wschemmert@t-online.de, <www.midi-and-more.de/more>