Statemets in alphabetical order

Note1: All command are accepted either in uppercase or lowercace. However, case cannot be mixed inside a word.
Note2: All variables and constants are 16 bits long words. If a port is read into a variable then only the lower 8 bits are set, upper 8 bits are zeroed. Similarly if a variable or constant is written to a port, only lower 8 bits are written, upper 8 bits are ignored.
Note3: Numbers can be written as decimal, hexadecimal or binary form. In hexadecimal form both uppercase and lowercase form is accepted. Numbers having no special notation and are represented in 0..9 character range are decimal numbers. A 0x prefix introduces hexadecimal numbers (e.g. 0xa18c) , while 0b prefix introduces binary numbers (e.g. 0B01001110)
Warning! Numbers in illegal form will not cause error message, but will be autodeclared as variables instead! (e.g. 19a, 0x78g, 0b21)

:

Description: Separates different statements within single line
Example: A=7 : B=88

=

Description: assign value to variable
Example: myvar=foo+88*bar
*Expression is evaluated using usual precedence rules, i.e. multiplicatinon precedes addition: 88*bar will be calculated first then added to foo.*

CLR

Description: Clears a bit of a port. Avalible ports in this version are:

DDRB PINB PORTB
DDRC PINC PORTC
DDRD PIND PORTD

Port names can be also written in lowercase. Allowed pins are 0...7. Oversized values will be masked to this range.
Warning! Port name and bit compose a solid compound with a tie dot.
Example: clr portd.3

DELLINE

Description: Deletes command line and all higher numbered lines.
Example: delline 0x8ac
Note: care should be taken deleting lines. Deleting unexisting line may leave fragments of previous line and garbage can be misinterpreted as unwanted statement. It may cause serious crash.

DELSUB

Description: Deletes subroutine (user command) declared before. It also deletes all the subroutines declared later then subroutine being deleted, i.e. if declaratin was done as follows:

sub mycomm …... endsub
sub newerthing …... endsub

… and delete *mycomm* - then *newerthing* will be deleted as well!
Example: delsub mycomm
Note: care should be taken deleting commands. Referenced to unexisting command may

cause serious crash.

DO

Description: Introduces a DO – UNTIL structure. Expression evaluated end of structure at each run and loop continous as long as expression is true. Statement inside DO – UNTIL structure will be executed at least once even if condition is false.
Example:

do { …. } until (a<10)

EXIT

Description: Stops session and exits PC program. Any unsaved change will be lost. To save modifications, run your program. See RUN for details.
Example: exit

FOR

Description: Introduces a FOR – NEXT structure. An optional step is allowed. Starting and ending values may consist of constants, variables or any mixture of them. Step (if used) also may be any of them. Statement inside FOR – NEXT structure will be executed at least once even if ending value is reached at start.
Example:

for x=myvar+7 to myvar+112 { …. } next
for x=myvar+7 to myvar+112 step 10 { …. } next

IF

Description: Introduces an iF-ELSE structure. If condition is true, statement(s) following expression will be executed. If condition is false and there is an *ELSE* keyword in the structure then statement(s) following *ELSE* keyword will be executed.
Example:

if (a<10) { set portb.0 }
if (a<10) { set portb.0 } else { set portb.1 }

NEW

Description: deletes all variables and subroutine declaration and introduces new programming section. ***Warning!*** *Program in chip is still existing as long as new program takes place.*
Example: new

QUIT

Description: Stops session and exits PC program. Any unsaved change will be lost. To save modifications, run your program. See RUN for details.
Example: quit

PORT

Description: Statement is a common form of ports handled by ABS. Avalible ports in this version are:

DDRB PINB PORTB
DDRC PINC PORTC
DDRD PIND PORTD

Port names can be also written in lowercase.
Example1: portd=a+12
Example2: mybyte=pinb and 0x7F

RUN

Description: Runs program in chip. If any modification was made since the last run, program will be downloaded.
Note1: There is no „save" or „download" command. Program will be downloaded and saved just before the run.
Note2: Statement refers to actual session. If chip was removed or connection was lost for any reason, new session will be introduced.
Note3: It is strongly recommended to save your program in a file to recover in case of a crash.
Example: RUN

SUB

Description: Encounters a new user command. The command declaration is opened by *SUB* keyword and closed by *ENDSUB* keyword. The first item after *SUB* keyword will be the name of the command, followed by statement sequence. After succesful declaration the just defined command can be immediately used by itself or encapsulated into another new command. Entering the name a new command all the sequence consisted in it will be executed.
Example:
sub mytest portd=3 : if (pinb < 5) { portc=0 } else { portc=0xff } endsub
Executing *mytest* will write 3 to portd and 0 or 0xff to portc according to pinb state.

SET

Description: Sets a bit of a port. Avalible ports in this version are:

DDRB PINB PORTB
DDRC PINC PORTC
DDRD PIND PORTD

Port names can be also written in lowercase. Allowed pins are 0...7. Oversized values will be masked to this range.
Warning! Port name and bit compose a solid compound with a tie dot.
Example: set portd.3

TOGGLE

Description: Toggles a bit of a port. Avalible ports in this version are:

DDRB PINB PORTB
DDRC PINC PORTC
DDRD PIND PORTD

Port names can be also written in lowercase. Allowed pins are 0...7. Oversized values will be masked to this range.
Warning! Port name and bit compose a solid compound with a tie dot.
Example: toggle portd.3

WHILE

Description: Introduces a while structure. Statements inside structure will be executed until condition is true. If condition is false already at start, all structure is skipped and statements will be never executed.
Example:

while (a>0) { …. }
while (a) { …. }

To compose a forever loop use an expression never come true.

Example:

while (1) { …. }

Az empty statement is also allowed:

while (timer0) {}

Note the missing statement(s) between brackets. This structure can be used in special cases when user wants keep stayind as long as an external event happens. (E.g. a time elapses, a pin activated, etc.)

Care should be taken this structure because it may lead to endless running.

Arithmetic and logic operators

Note: Expression is evaluated using usual precedence rules, i.e. multiplicatinon precedes addition.In expression

*myvar=foo+88*bar*

*88*bar will be calculated first then added to foo.*

:

Description: Separates different stataements within single line

Example: A=7 : B=88

=

Description: assign value to variable

Example: myvar=foo+88*bar

Expression is evaluated using usual precedence rules, i.e. multiplicatinon precedes addition: 88*bar will be calculated first then added to foo.

+

Description: addition

Example: myvar=foo+bar

-

Description: substraction

Example: myvar=foo-bar

*

Description: multiplication

*Example: myvar=foo*bar*

Note1: Overflow part will be saved in temporary register and taken account in subsequent division. As a result a= 2*50000/4 will yield correctly a=25000.

Note2: FORTH users may recognise in this solution the ***/*** operator, but the need for three oparans is avoided.

/

Description: division

Example: myvar=foo/bar

Note1: Proceeding multiplication may leave some overflow in a temporary register and it is taken account in division. If no overflow was in proceeding multiplication, the temporary register is cleared, not influencing division.

Note2: FORTH users may recognise in this solution the ***/*** operator, but need for handling three operands is avoided.

%
Description: modulo
Example: myvar=foo%bar
Note1: Proceeding multiplication may leave some overflow in a temporary register and it is taken account in calculating modulo. If no overflow was in proceeding multiplication, the temporary register is cleared, not influencing calculating modulo.
Note2: FORTH users may recognise in this solution the ***/*** operator, but need for handling three operands is avoided.

&
Description: bitwise logical and
Example: myvar=foo&bar

|
Description: bitwise logical or
Example: myvar=foo|bar

^
Description: bitwise logical xor
Example: myvar=foo^bar

!
Description: bitwise logical not
Example: myvar=!foo
results zero (false, actually 0) if the expression was nonzero (true)
resuls nonzero (true, actually 1) if the expression was zero (false)

Comparisional operators

<

Description: compares left side operator to right side operator. Returns true if left side is less then right side.
Example: if (a<8) …. returns true if *foo* is in range 0...7
Note: All values are represented as unsigned 16 bit integers. No -1, -2... -n is interpreted, 0-1 will result 65535

<=

Description: compares left side operator to right side operator. Returns true if left side is less or equal to right side.
Example: if (a<=8) …. returns true if *foo* is in range 0...8
Note: All values are represented as unsigned 16 bit integers. No -1, -2... -n is interpreted, 0-1 will result 65535

>

Description: compares left side operator to right side operator. Returns true if left side is greather then right side.
Example: if (a>8) …. returns true if *foo* is in range 9...65535
Note: All values are represented as unsigned 16 bit integers. No -1, -2... -n is interpreted, 0-1 will result 65535

>=

Description: compares left side operator to right side operator. Returns true if left side is greather or equal to right side.
Example: if (a>=8) …. returns true if *foo* is in range 8...65535
Note: All values are represented as unsigned 16 bit integers. No -1, -2... -n is interpreted, 0-1 will result 65535

==

Description: compares left side operator to right side operator. Returns true if left side is different then right side.
Example: if (a==8) …. returns true if *foo* is 8.
Note: All values are represented as unsigned 16 bit integers. No -1, -2... -n is interpreted, 0-1 will result 65535.
Warning! Due to unsigned representation comparing 65535 to -1 (65534 to -2, etc.) will be handled as equal values!

<>

Description: compares left side operator to right side operator. Returns true if left side is different then right side.
Example: if (a=8) …. returns true if *foo* is not 8.
Note: All values are represented as unsigned 16 bit integers. No -1, -2... -n is interpreted, 0-1 will result 65535.
Warning! Due to unsigned representation comparing 65535 to -1 (65534 to -2, etc.) will be handled as equal values!

!=

Description: just another notation, but fully equvivalent to **<>**, see there.

Internal variables

Some internal variables are available for users. These variables can be written and read as other variables, but they influence the operating system running in background. Some of them may even be modifyed by the system.

System variables updated from time to time:

timer0
timer1
timer2
timer3

As names suggest these user variables act as timers. Each of them count backward. At bootup stepping rate is 1 Hz, i.e. each of them steps down at each second. Reaching 0, a turn to 0xffff (or 65535 if you like it better) will take place and counting goes on.
Timers can be set to expected delay time. Wring e.g. 10 to timer0 (simply write timer0=10) your timer0 will count back 10 seconds. During tis time you may do anithing else provided sometimes check if timer0 equals to zero. Due to the system quartz crystal referenced timing you get three advantages:
- you are free from running useless delay loops for timing
- you can handle unexpected external pin events during delay
- however your timer0 polling may jitter in time, the timer runs without any jitter, resulting stability of a quartz controlled clock – because actually it ***is*** a quartz controlled clock.
Each timers are independent and have no influence to each other.

System variables belonging to timing system are:

scale0
scale1
scale2
scale3

scale0 belongs to timer0, scale1 to timer1, etc., respectively. These variables are never changed by the system, however theirs content control the counting rate of respective timer. Value hold by a scale variable represent how many milliseconds (i.e. 1/1000 seconds) are necessary to make a timer to step. As the system starts, each of the scale variables hold 1000, so 1000 millisec, i.e. 1 second will be the stepping rate for each timer. Having no influence to each other, different scale and timer valuse can be used.

Autorun variables belonging to timing system are:

ontick0
ontick1
ontick2
ontick3

ontick0 belongs to timer0, ontick1 to timer1, etc., respectively. These variables are never changed by the system, however theirs content ….. to be continued.

Event variables:

captured
This variable is continously updated when „capture pin” is activated.

Port related variables:

Note: All ports are 8 bits long. If a port is written, lower 8 bits of variable is written to it. If a port is read into a variable, upper 8 bits are cleared to zero.

ddrb *or* ***DDRB***	*Example: ddrb=3*
pinb *or* ***PINB***	*Example: foo=pinb*
portb *or* ***PORTB***	*Example: portb=3*
ddrc *or* ***DDRC***	*Example: ddrc=3*
pinc *or* ***PINC***	*Example: foo=pinc*
portc *or* ***PORTC***	*Example: portc=3*
ddrd *or* ***DDRD***	*Example: ddrd=3*
pind *or* ***PIND***	*Example: foo=pind*
portd *or* ***PORTD***	*Example: portd=3*