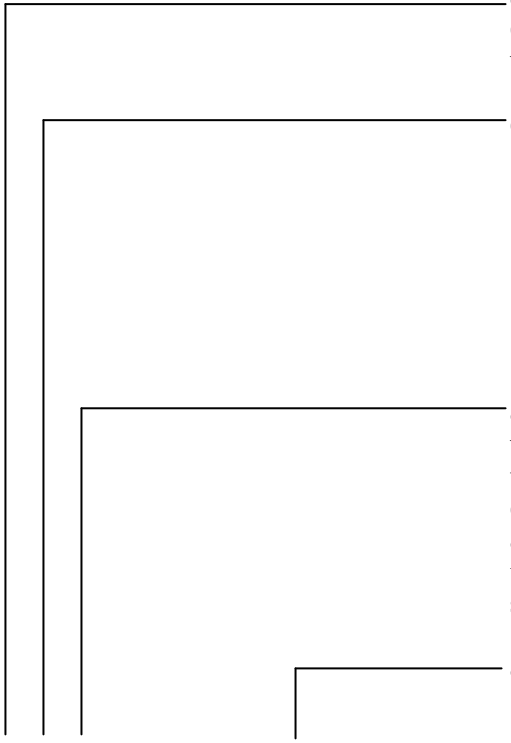
IF THEN, if then

compares value A to value B and branches to another statement if the condition is meet. This statement leads to a shorter codesize than the IF / ELSE / ENDIF statement as well as less typing and easier understanding.

Sizes : ‘long’, ‘word’, ‘byte’ and ‘bit’, positiv / negativ (twos complement)



always a symbolic representation of a numerial or logical value, e.g. a memory location like ‘apos’ or ‘in1’ or any user defined variable

operator :

=	equal
<	smaller
>	greater
<=	smaller or equal
>=	greater or equal
<>	unequal
= true	logical 1
= false	logical 0

another memory location (variable) or any numerical (immediate) value. The size of this variable (B) must not exceed the size of the size of variable A, e.g. a ‘word’ can be compared to a ‘byte’ or a ‘word’, but not to a ‘long’. Variables representing a bit vari - able are recognized automatically and the operators equal and unequal will lead to a faster processing. Bitvariables are only systemvariables like ‘in1’ to ‘in8’, ‘out1’ to ‘out8’ .

a user defined label.

IF <A> = <B> THEN GOTO <C>

This is a single line statement to check a condition and branch to another statement if the condition is true or false.

IF <A> = <B> THEN GOSUB <C>

This is a single line statement to check a condition and branch to a subroutine if the condition is meet. The programm will carry on at the next statement behind this statement after the subroutine has been finished.

IF <A> = <B> THEN EXIT FOR

This statement can be used inside a FOR / NEXT loop, to exit the loop on a certain condition before the loopcount has finished.

1^{st} pass errors : - undefined symbol
- size
- not inside a FOR / NEXT loop

2^{nd} pass errors : - undefined symbol
- undefined label

Runtime errors : - GOSUB to a label wich dosn’t belong to a subroutine or code not terminated by a RETURN statement

.

IF THEN, if then continued

Examples :

A.

```
if variable_A = 20 then goto label_X
print ‘variable_A is not 20 !’
goto label_Y

label_X:
print ‘variable_A is 20 !’
label_Y:
print /n
.
```

B.

```
for variable_A = 1 to 100

        print variable_A
        print /n

        if variable_A <= 80 then exit for

next
.
.
```

C.

```
if variable_A = 20 then gosub label_X
.
.

label_X:
print ‘variable_A equals 20’
print /n
return
```

D. if in1 = true then goto xyz

E. if out3 = 0 then goto xyz

F. if out1 = false then goto xyz

IF /ELSE / ENDIF
IF / ENDIF
if / else / endif
if / endif

compares value A to value B and branches to another statement if the condition is meet.

Sizes : ‘long’, ‘word’, ‘byte’ and ‘bit’, positiv / negativ (twos complement)

always a symbolic representation of a numerial or logical value, e.g. a memory location like ‘ist_pos’ or ‘in1’ or any user defined variable

operator :

=	equal
<	smaller
>	greater
<=	smaller or equal
>=	greater or equal
<>	unequal

another memory location (variable) or any numerical (immediate) value. The size of this variable (B) must not exceed the size of the size of variable A, e.g. a ‘word’ can be compared to a ‘byte’ or a ‘word’, but not to a ‘long’. Variables representing a bit vari - able are recognized automatically and the operators equal and unequal will lead to a faster processing. Bitvariables are only systemvariables like ‘in1’ to ‘in8’, ‘out1’ to ‘out8’.

1st pass errors : - undefined symbol
- size
- nesting
- missing endif
- to many ELSE statements

2nd pass errors : - undefined symbol
- undefined label

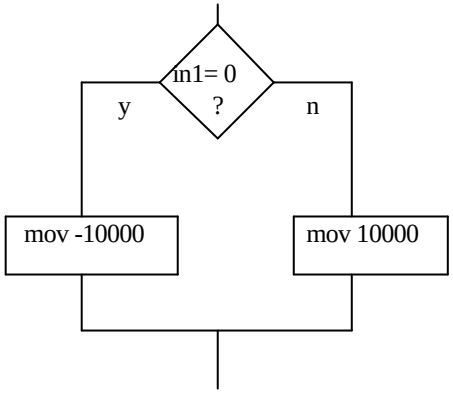
.

IF <A> = <B>

Examples

A.

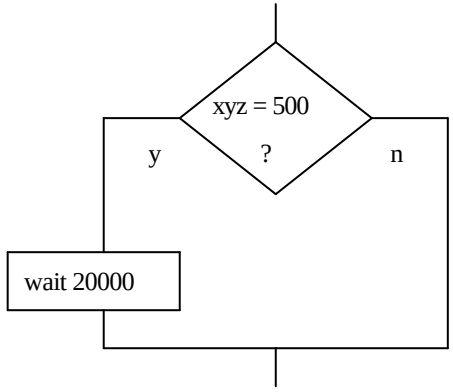
```
if in1 = 0
        mov -10000
else
        mov 10000
endif
```



IF / ELSE / ENDIF continued

B.

```
if xyz = 5
        wait 20000
endif
```



C.

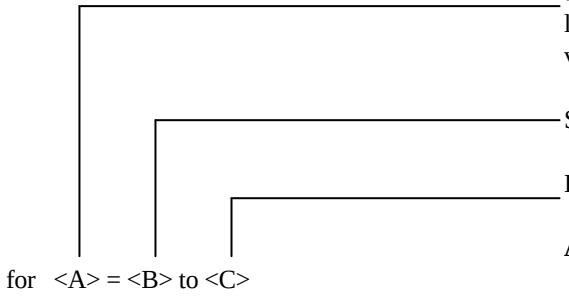
```
if in1 = 0
        out1 = 0
else
        out1 = 1
endif
```

FOR / NEXT

repeats a number of statements a specified number of times

for / next

Sizes : ‘long’, ‘word’ and ‘byte’, positiv / negativ (twos complement)



always a symbolic representation of a numerical value, e.g. a memory location like ‘apos’ or any user defined variable. If the symbolic variable is not defined, it will be generated automatically.

Startvalue, either a symbolic value or an immideate value

Endvalue, either a symbolic value or an immideate value

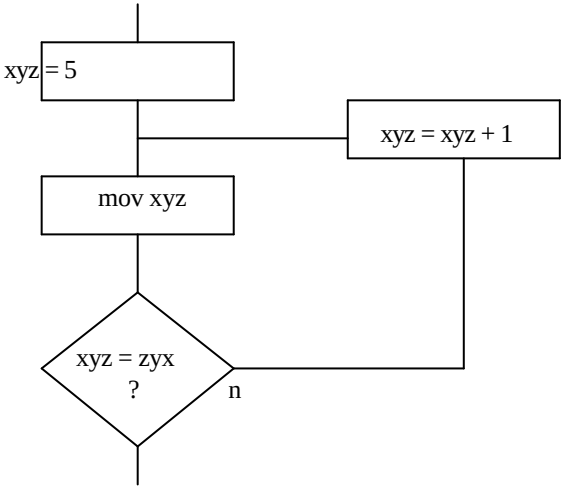
Attention : Counting is always up and the stepsize is always one !

1st pass errors : - size
- missing NEXT statement
- undefined symbol

Examples :

A.

```
for xyz = 5 to zxy
        mov xyz
next
```



B.

```
for variable_a = 0 to 10000
        print “ VARIBLE_A = “
        print variable_A
        print /n
next
```

PRINT, print

Returns values and strings to the remote computer

Sizes of values : all

PRINT$ abcdefghijklmno...	will send the string ‘abcdefghijklmno...’ to the remote computer, no CR/LF is attached
PRINT „abcdefghijkmno...“	will send the string ‘abcdefghijklmno...’ to the remote computer, no CR/LF is attached
PRINT /n	will send CR/LF to the remote computer
PRINT <symbol>/H	will send the hex presentation of the symbol’s value to the remote computer, no CR/LF is attached
PRINT <symbol>	will send the decimal representation of the symbols value to the remote computer, no CR/LF is attached

1st pass errors : - undefined symbol

Examples :

A.

```
.
.
paul = 200
print$   The current value of paul is :
print    paul
print    /n
print    “The current value of paul as hex representation is : “
print    paul/h
print    /n
.
.
```

Output :

```
The current value of paul is :200
The current value of paul as hex representation is :  C8H
```

GOTO, goto

Branch to a label

Example :

```
                out1 = 1
                out2 = 1
                movr -200
                goto label_A
                .
                .
label_A:
                out1 = 0
                out2 = 0
                .
                .
```

2nd pass erros : - undefined label
- multidefined labels

GOSUB, gosub

Branch to subroutine, either a user defined label or a system label is valid (see system routines)

Example :

```
                gosub port_on
                gosub run_motor
                gosub port_off
                .
                .
port_on:
                out1 = 1
                out2 = 1
                return
port_off:
                out1 = 0
                out2 = 0
                return

run_motor:
                if rmp_stat <> 0 then goto run_motor

                mov 10000
                return
```

1st pass errors: - none

2nd pass errors: - undefined label
- multidefined labels

Runtime errors: - missing RETURN statement at the end of a subroutine

RETURN, return

Terminates a subroutine

Example: see GOSUB

DEFINE BYTE / WORD / LONG
define byte / word / long

Define a memory variable, this statement will save storagespace in the controllers R/W memory. The minimum space reserved is two bytes for the DEFINE BYTE statement and four bytes for the DEFINE LONG statement

The symbolic name of the defined variable can be referenced in all statements dealing with variables.

Any valid symbolic name except protected names which will be used further on in the programm.

DEFINE BYTE <label/symbol>

Examples :

1st pass errors: - protected variable name

```
define byte variable_C          ; definition
define long variable_E

let variable_C = 100            ; initialisation
let variable_E = -200000
```

DEFINE BYTE_E / WORD_E / LONG_E

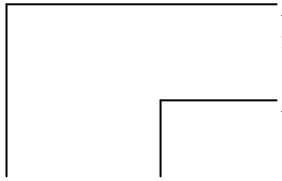
define byte_e / word_e / long_e

Define a memory variable including ist initalization, this statement will save storagespace in the controllers nonvolatile memory. The minimum space reserved is two bytes for the DEFINE BYTE_E statement and four bytes for the DEFINE LONG_E statement.

This statement is usually used to store system parameters which will be used at power up time. The defined storage will be setup during programm load time. There is no need to write to it within a running programm.

The symbolic name of the defined variable can be referenced in all statements dealing with variables.

Attention : **This memory is made of EEPROM technology and has a limited number of write cycles. Don't used it instead of standard R/W memory.**



Any valid symbolic name except protected names which will be used further on in the programm.

Any number within the possible range of figures (singned numbers)

1st pass errors: - protected variable name
- size

2nd pass errors: - multible defind labels

DEFINE BYTE_E <label/symbol> = <value>

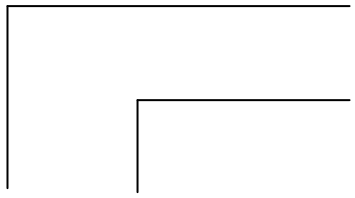
Examples :

define byte_e variable_C = -120

define long_e variable_E = 231231231

DIM BYTE / WORD / LONG
dim byte / word / long

Defines an array or a table in R/W memory. This array can be accessed by a label and a pointer. The kind of access is name ‘indirect’



The arrayname is actually a label or the symbolic address in memory
Only names are valid !

This is always a figure representing the size of the array. Bytearrays will be internally aligned to an even size, wordarrays will use twice as much memory and longarrays will use four times as much memory as the defined size.

DIM BYTE <arrayname> <(size)>

Examples :

```
dim byte array1(5)
dim byte xyz(5)

for cnt = 0 to 4                    ; copy contents of array ‘array1’
                                    ; to array ‘xyz’
        a_byte = array1(cnt)
        xyz(cnt) = a_byte
        print cnt                   ; and show movement on PC
        print$ ,
        print a_byte
        print /n

next
```

LET, let

Assign values to symbols and performs basic calculations

Sizes : all except BIT operators, in most cases, the sizes are getting aligned to the size of the variable left of the equal sign. If an alignement is not posiible, an error will be generated. Moving a smaller size (byte) to a bigger size (word), the upper bits (sign) will be set accordingly. All calculations and movements are done with signed number (two's complement).

On 'BIT' (sytem) variables, only 0 and 1 are valid !

Options : The keyword 'LET' is optional

Operators : + = add
- = substract
/ = divide
* = multiply

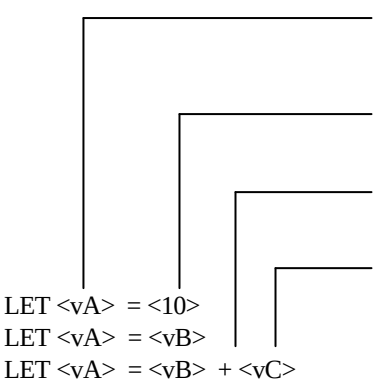
Operations :

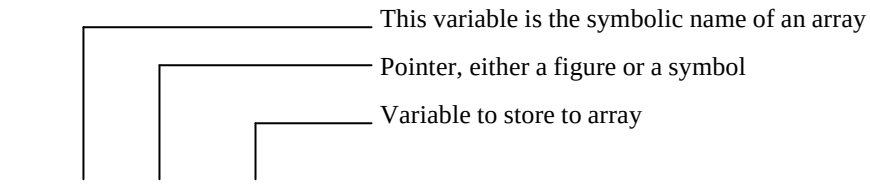
load immediate byte with	-128 to	127
load immediate word with	-128 to	127
load immediate long with	-128 to	127
load immediate word with	-32767 to	32766
load immediate long with	-32767 to	32766
load immediate long with	-2147483647 to	2147483647

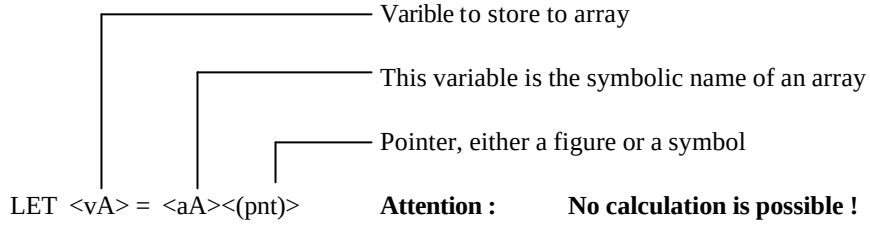
add byte to byte ---> byte	sub byte from byte ---> byte
add byte to byte ---> word	sub byte from word ---> word
add byte to byte ---> long	sub byte from long ---> long
add byte to word ---> word	sub word from word ---> word
add byte to word ---> long	sub word from long ---> long
add byte to long ---> long	sub long from long ---> long
add word to word ---> word	sub byte from byte ---> word
add word to word ---> long	sub byte from byte ---> long
add word to long ---> long	sub word from word ---> long
add long to long ---> long	

div byte by byte ---> byte	mul byte by byte ---> byte
div word by byte ---> word	mul byte by word ---> word
div word by word ---> word	mul word by word ---> long
div long by word ---> long	

LET, let continued







Examples :

```
let variable_A = -120
let variable_A = variable_B + 10
let array_A(pnt) = variable_A
let array_A(15) = variable_B
let variable_A = variable_A - variable_B
let variable_B = array_A(pnt)
```

1st pass errors : - undefined symbol
- size
- array not defined
- invalid operation

MOV, mov

Move motor to absolut postion.

size : long only

Can be a figure or a symbolic variable

mov <vA>

Example :

mov variable_A

mov -120

1st pass errors : - undefined symbol
- size

MOVR, movr

Move motor to relativ postion.

size : long only

Can be a figure or a symbolic variable

movr <vA>

Example :

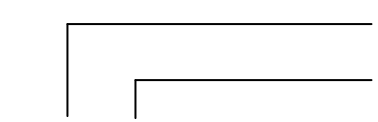
movr variable_A

movr -120

1st pass errors : - undefined symbol
- size

out1 ... out8

Set output ports to on or off



1st pass errors : - wrong number

Example :

```
out3 1          ; set port 3 to on
out5 0          ; set port 5 to off
```

RPO, rpo

Reset position or set current position to zero

STOP, stop

Signals the end of Source file to the compiler

END, end

Signals the end of Source file to the compiler

SET EEPROM
set eeprom

Signals to the compiler to compile the code so, that it can be runned from nonvolatile memory

wait

Stops the program-execution for a fixed time in milli-seconds
Example: wait 1000 stops the programm for 1 second

Systemvariables

These are variables defined by the BGE9010 controller and they are protected

Variblename	Size	Access	Figure-range	Description
apos	Long	R	any	Actual position
aposw	Long	R/W	positiv only	Detection window for current position flag in incremensts
aspd	word	R	any	Motorspeed in rpm
bal	byte	R	0/1	Break-Resistor circuitry, 1 = activ
brt	word	R/W	positiv only	Ontime of breakresistor circuitry in increments of 5ms
brv	word	R/W	positiv only	Thresholdvoltage of breakresistor circuitry
in1 ... in8	byte	R	0/1	Inputport status
inapos	byte	R	0/1	In current position flag. 1 = in position
intpos	byte	R	0/1	In position flag, 1 = in position
led1	byte	R/W	0/1	Special output port – green LED
mpos	long	R/W	any	Intermediate target position
mvend	byte	R	0/1	Positioning status, 1 = busy, 0 = stopped
out1... out8	byte	R/W	0/1	Outputport status
pkp	word	R/W	positiv only	Proportional gain of position-controller
pvmax	word	R/W	positiv only	Max. speed in rpm out of the position-controller
ski	word	R/W	positiv only	Integral gain of speed-controller
skp	word	R/W	positiv only	Proportional gain of speed_controller
tf	word	R/W	positiv only	Break or ramp down time in ms (for 1000 rmp)
tmp	Word	R	positiv only	Current heatsink temperatur in degree centigrad
tpos	Long	R	any	Target – Position
tposw	Long	R/W	positv only	Detection window for position flag in increments
tr	Word	R/W	postiv only	Acceleration-time for ramp up in ms (for 1000 rpm)
va1	Word	R	any	Inputvoltage of analog port in mV
va1off	Word	R	any	Offsetvoltage
va2	Word	R	any	Inputvoltage of analog port in mV
va2off	word	R	any	Offsetvoltage
vs	word	R	positiv only	Current power supply voltage
enable	byte	R/W	0/1	1= power stage on , if inhibit=0
id	word	R/W	positiv only	long-time current limit in mA
iki	word	R/W	positiv only	Integral gain of current-controller
ikp	word	R/W	positiv only	Proportional gain of current-controller
inhibit	byte	R/W	0/1	1= power stage is off
ip	word	R/W	positiv only	short-time current limit in mA
mode	byte	R/W	any	BGE9010- Operationmode
nmul	word	R/W	positiv only	factor va1 to nominal speed - 4Q mode only
spd1	word	R/W	any	nominal speed in rpm