BLASM
Manual

Comments
Use the semi-colon character to create a comment.

Examples:

```
; A comment
score = 100       ; This comment starts after the instruction
```

Names
Valid characters:

```
A-Z               Alphabetical
0-9               Digits (names cannot start with a digit)
@                 "At"
.                 Period
_                 Underline
?                 Question
```

Examples:

```
myLabel:
player1 = 100
```

Numbers
Examples:

```
100               Decimal
-100              Negative decimal
0x100             Hexadecimal
100h              Hexadecimal
¤100              Octal
%100              Binary
```

Strings
Examples:

```
'Player 1'
"Player 2"

'Some numbers "1,2,3"'
"Some numbers '1,2,3'"

'Hello, ' + 'World!'

"ABC" + #68
```

Operators
List:

```
+                Addition
-                Subtraction
*                Multiply
/                Division
^                Power

MOD              Modulo
DIV              Division (same as "/")

==               Equal
!=               Not Equal
<                Less than
>                Higher than
<=               Less than or Equal
>=               Higher than or Equal

EQ               Equal
NE               Not Equal
LT               Less than
GT               Greater than
LE               Less than or Equal
GE               Greater than or Equal

( )              Brackets, increases priority
```

Examples:

```
a = 100           ; Assign 100 to an equate
b = (100 + 2) * 6 ; 100 + 2 will be calculated first, then multiplied with 6
c = 100 + 2 * 6   ; 100 will be added with 2 * 6 (= 100+12)
d = 10 MOD 3      ; Modulo
e = 10 DIV 3      ; Division
f = 5 ^ 3         ; Power 5*5*5
```

Number Functions
List:

```
$                Returns the current segment/section location
$$               Returns the start offset of the current section
OFFSET           Returns the offset of an identifier
SIZE             Returns the size of an identifier
```

Examples:

```
a = $                   ; Get Segment Location
b = SIZE myString       ; Get Size of String Equate
c = OFFSET myLabel      ; Get Offset of "myLabel"
```

Type List
This is the general type list. It's used by many instructions.

```
byte              8-bit value
word              16-bit value
dword             32-bit value
qword             64-bit value
farptr            48-bit value
tenbyte           80-bit value
```

Local / Global
Identifiers can be defined locally or globally – LOCAL MODE is enabled by default.

```
name = 100        ; Create a local equate
~name:            ; Create a label in the
                  ; current source file list
~~name = 100      ; Create a global equate ('~' * 2)
```

You can switch to GLOBAL MODE if identifiers should be defined globally by default. It's also possible to move existing identifiers between the lists:

```
.GLOBALMODE       ; Switch to global mode (optional: GLOBALMODE (no dot))
score = 0         ; The identifier is now defined in the global list
LOCAL score       ; Move the identifier to the local list

.LOCALMODE        ; Switch back to local mode (optional: LOCALMODE (no dot))
hiscore = 10000   ; The identifier is defined in the local list
GLOBAL hiscore    ; Move the identifier to the global list
```

Parameter Types
Here is the parameter types, used by the instructions (directives):

```
name              Name parameter
name=expected     Expects a specific name
name (integer)    Name of an Equate, returns an integer value
name (string)     Name of an Equate, returns a string value
value             Value parameter
string            String
on / off          ON or OFF expected
alpha             Alpabetical variable (A-Z) used in headers.
```

Directives
List:

```
@ECHO  string                          Write Message
@ERROR  string                         Generate an Error
@WARNING  string                       Generate a Warning

.ADDMODEL  name  (item1) (item2)..     Add a model
                                       (item =
                                       segName bits limit fill round write)
                                       segName     name        Target segment
                                       bits        value       Segment bits
                                       limit       value       Segment size limit
                                       fill        value       Fill segment
                                       round       value       Round segment
                                       write       on / off    Write this segment?

.ADDRELOC  value                       Add a value to the Relocation list

ALIAS  newName  targetName             Create an Alias
.ALIAS  newName targetName             (see ALIAS)

ALIASCHAR  'n'  't'                    Create an Alias Character
.ALIASCHAR  'n'  't'                   (see ALIASCHAR)

(name) ARRAY  type  value, count       Create an array

.BLOCKSIZE  value                      Change Block Size of the
                                       active Output File

.DEFINE  name                          Define an identifier

DELETE  name                           Deletes an Identifier
.DELETE  name                          (see DELETE)

.DISK  item                            DISK-Object

(name) DB value(s)                     Define Byte
(name) DW value(s)                     Define Word
(name) DD value(s)                     Define Double-word
(name) DQ value(s)                     Define Quad-word
(name) DF value(s)                     Define 48-bit
(name) DP value(s)                     Define 48-bit
(name) DT value(s)                     Define TenByte

.DYNARRAY  item                        DYNAMIC ARRAY-Object

ECHO  string                           Write Message

END (label)                            Ends the current source
(name) ENDM                            End of macro
(name) ENDO                            End of object
(name) ENDP                            End of procedure
(name) ENDS                            End of section

name EQU  value                        Create a number Equate
name EQUS  string                      Create a string Equate

ERROR  string                          Generate an Error

EXITM                                  Leave Macro
EXITO                                  Leave Object
```

.EXT string	Change File Extension of the active Output
.FILE item	**FILE**-Object
.FILL value	Fill segment
.FILLSEG value	Fill segment, after assembling
.FIXINSTRPREFIXES	Auto-set instruction prefixes 66h and 67h **NOTE!** Instructions without parameters will not be modified by this instruction.
.GEN item	**Generator**-Object
.GET item	**Get**-Object
INCLUDE string	Include another source file
.INCLUDE string	(see **INCLUDE**)
.INSTRDUMP filename	Dump Instruction List to a file
.HEADER 'name' identname *(ptn)*	Create header
.HEADERITEM hdrName item	Add Item to Header
.LIMIT value	Limit segment size
name **MACRO** *(params)*	Create a Macro
MACRO name *(params)*	Create a Macro
MODEL name	Set Model
.MODEL name	(see **MODEL**)
.MODRM item	**MODR/M**-Object
.MODRMDATA tblName regs bits bracketCode values	Add ModR/M Data to Table
.MODRMTBL tblName columns	Create ModR/M Table
name **OBJECT** stdParam (*(params)*)	Create an Object
OBJECT name stdparam (*(params)*)	Create an object
ORG value	Set ORG
.ORG value	(see **ORG**)
.OUTPUT 'name' identname 'ext' headerName blockSize limit fill round *(pattern)*	Create Output
.OUTLIMIT value	Set size limit of the active Output
.OUTFILL value	Fill the Output to the requested size
.OUTROUND value	Round the Output
.PARSER item	**PARSER**-Object
(name) **PROC**	Create a Procedure
PROC name	Create a Procedure
.PROCESSOR name value	Create processor ID
.REG name code	Create a register
.REGTABLE tblName reg1, reg2,..	Create a register table
.REMOVEINSTRPREFIXES	Remove instruction prefixes 66h / 67h
.RESETFILLPATTERNPOS	Reset the Segments Fill-Pattern-Position

```
.ROUND  value                          Rounds the segment
.ROUNDSEG  value                       Rounds segment, after assembling

.RUNINSTR  name  (par1, par2,..)       Run Processor Instruction

SECTION  sectionName  useSegment       Create a section
(name) SECTION useSegment              Create a section

SEGMENT  'name'  identname  (ptn)      Create a segment
.SEGMENT 'name'  identname  (ptn)      (see SEGMENT)

.SEGREG  name  code  value             Create a segment register

STACK  value                           Set stack memory
.STACK  value                          (see STACK)

.STRING  item                          STRING-Object

TERMINATE   (name=NOBUILD)             Stops assembling

WARNING  string                        Generate a Warning
```

Statements

List:

```
.BREAK                                 Break a loop
.CONTINUE                              Continue loop

.ENDIF                                 ENDIF

.ELSE                                  ELSE (run if expression is false)

.FOR var = startVal TO endVal (STEP value)
                                       Starts FOR-loop

.IF value1 operator value2             IF expression is true
.IFB param                             IF Parameter is blank
.IFBYTE value                          IF Value is a valid BYTE
.IFDEF name                            IF name is Defined
.IFDIF name1, name2                    IF Different
.IFDIFI name1, name2                   IF Different
.IFDWORD value                         IF Value is a valid DWORD
.IFEQU name                            IF Name is an Equate
.IFEQUSTR name                         IF Name is an Equate String
.IFEQUVALUE name                       IF Name is an Equate Number
.IFFALSE value                         IF False ( = 0)
.IFFARPTR value                        IF Value is a valid 48-bit
.IFHEADER name                         IF Name is a Header
.IFIDN name1, name2                    IF Identical
.IFIDNI name1, name2                   IF Identical
.IFMODEL name                          IF Name is a Model
.IFNAME something                      IF Name
.IFNAMES  name n1 (, n2(, n3...))      IF "name" match any of the other names
.IFNAMEEQUSTRING name  string          IF Name is Equal to String
.IFNB param                            IF NOT Parameter is blank
.IFNDEF name                           IF NOT name is Defined
.IFOUTPUT name                         IF Name is an Output
.IFPARSENAME name                      IF Parse Name is Equal to Name
.IFPARSENAMES n1(, n2(, n3...))        IF Parse Name is Equal to any of the names
.IFPARSENAMESTR string                 IF Parse Name Equal to String
.IFPROCESSOR name                      IF Name is a Processor
.IFREG name                            IF Name is a Register
```

```
.IFREGTABLE name                    IF Name is a Register Table
.IFS string1  string2               IF string1 is Equal to string2
.IFSEGMENT name                     IF Name is a Segment
.IFSTRING something                 IF String
.IFTRUE value                       IF True ( = non-zero)
.IFVALUE something                  IF Value
.IFWORD value                       IF Value is a valid WORD

.NEXT                               End of FOR-loop

.WEND                               End of WHILE-loop
.WHILE value1 operator value2       Starts WHILE-loop
```

Include Files

Use the .INCLUDE-instruction to include a file:

```
.INCLUDE  'c:\blasm\test.inc'       ; Include a file
```

Equates

There is two types of Equates:

```
score  EQU  100                     Number Equate
player  EQUS  'Player 1'            String Equate
hiscore = 10000                     Number Equate
```

Example:

```
hiScore     EQU  100000             ; Number Equate
DD hiScore                          ; Define Dword (32-bit value)
```

The #-character adds a character by using an ASCII-value:

```
myText EQUS 'Hello, World!' + #13 + #10 + 'Good bye!'
DB myText                           ; Define Bytes (8-bit values)
```

Define Memory Data

Use the D(x) instructions to define data into the selected segment.

Types:

```
DB                Define Byte (8-bit)
DW                Define Word (16-bit)
DD                Define Double-Word (32-bit)
DQ                Define Quad-Word (64-bit)
DF                Define Far 48-bit Pointer
DP                Define Far 48-bit Pointer
DT                Define TenByte (10 bytes)
```

Usage:

```
(name) DX value(s)      Define values in the selected segment
                        The name is optional and can be used
                        to return the offset of the data.
                        Use a comma (,) to separate values.
```

Examples:

```
DB 'Hello!'             ; Write 'Hello!' to the segment
score DW 0              ; Write 16-bit value to the segment (=0)
                        ; "score" holds the offset to the data

mulValues DB 0, 1, 2, 3 ; Write 4 bytes to the segment

memTable DW score       ; Write the offset of "score" to the segment
DD segName              ; Write the offset of a segment to the selected segment.
                        ; "segName" = get offset of this segment
```

Labels

A label holds the current offset:

```
myLabel:                ; Creating a label named "myLabel"
```

Example:

```
DB 'Hello!'             ; Write 'Hello!' into the current segment

myLabel:                ; Creating a label
ofs = OFFSET myLabel    ; ofs = offset of label
ECHO ofs                ; Display the offset
```

Procedures

Use PROC to set the beginning of a procedure, and ENDP to end the procedure:

```
name PROC         ; Start the procedure
; content here
name ENDP         ; End of procedure
```

Macros / Objects

Macros and Objects works basically in the same way, but they have different syntax - objects is using a main-parameter and the other parameters must be inside brackets ( ).

Usage:

```
name MACRO (params)                 ; Start of a macro
; code here...
(name) ENDM                         ; End of macro

name OBJECT mainParam ((params))    ; Start an object
; code here...
(name) ENDO                         ; End of object
```

Example 1:

```
NewMessage  MACRO str          ; Create a macro
   ECHO str                    ; Echo content from parameter
   ENDM                        ; End of macro

NewMessage 'Hello!'            ; Run macro
```

Example 2:

```
NewMessage MACRO str           ; Create a macro
 .IFNB {str}                   ; If not parameter is blank
   ECHO str                    ; Echo content from parameter
 .ELSE                         ; ELSE
   ECHO 'Is Blank!!!'          ; ECHO Message if blank
 .ENDIF                        ; ENDIF
   ENDM

NewMessage 'Hello!'            ; Run macro
NewMessage                     ; Run macro without parameter
```

Example 3:

```
TmyObj OBJECT name (val)       ; Create an object
   ~~name&.Value = val         ; Set value to global equate
   ENDO

TmyObj myList (100)            ; Run object
ECHO myList.Value              ; Write value
```

Example 4:

```
mac1 MACRO par1                ; Create a macro
   ECHO {par1}                 ; Echo "par1"
   ECHO par1                   ; Echo contents of "par1"
   ENDM                        ; End of macro

   mac1 'ABC...'               ; Run macro
```

Example 5:

```
mac2 MACRO par2                ; Create a macro
newlabel&par2:                 ; Create a label
   ENDM                        ; End of macro

mac2 x                         ; Run macro
```

Sections

A section is a part of a segment.
Use the **$$**-function to get it's start offset.

```
name SECTION (targetSeg)       ; Create a section in the current segment
; code here...
(name) ENDS                    ; End of section
```

Models

What is a Model?

A model contains settings for program segments.

Creating a Model

```
.ADDMODEL name (settings1) (settings2)...

name              The name of the model to be created
settings          Settings for each segment
                  (syntax)
                  segment-name      : name      : Name of the segment to modify
                  bits              : value     : Set bit value
                  limit             : value     : Set maximum size of segment
                                                  0 = disable limit
                  fill              : value     : Fill segment to the
                                                  requested size
                  round             : value     : Round the segment
                  write             : on / off  : Write the segment to the Output
```

Example:

```
.ADDMODEL myModel (seg1 16 0 0 0 16 ON) (seg2 16 0 0 0 16 OFF)
```

Set a model

Use the **.model**-instruction to set a model.

```
.MODEL name
```

Headers

What is a header?

A header is the first part of an Output File. It holds information about the executable, like type of executable, initial segment addresses, sizes, data tables and so on.

Create a Header

```
.HEADER  name  identname (pattern)
                 name              : string    : Name / Description of Header
                 identname         : name      : Identifier name
                 pattern           : name      : Use String Equate as
                                                 fill/round pattern
```

Insert Items to a header

```
.HEADERITEM  header  item
                 header            : name      : Target Header identifier name
                 item

                 Item List
                 TEXT  string
                                   string            : string

                 Write text to the header.
                 ..

                 BYTE  value
                                   value             : value

                 Write byte value to the header.
                 ..

                 WORD  value
                                   value             : value

                 Write word value to the header.
                 ..

                 DWORD  value
                                   value             : value

                 Write dword value to the header.
                 ..

                 QWORD  value
                                   value             : value

                 Write qword value to the header.
                 ..

                 FARPTR  value
                                   value             : value

                 Write far-pointer value to the header.
                 ..

                 TENBYTE  value
                                   value             : value

                 Write ten-byte value to the header.
                 ..
```

RELOCTBL type
type : name (see Type List)

Write Relocation Table to the header. "type" sets
the size of each item.
..

ROUND value
value : value

Round the header.
..

FILL value
value : value

Fill the header to the requested size.
..

MEMO alpha
alpha : alpha (A-Z)

Memorize the current location.
..

WRITEMEMO type alpha1 *(**operator** alpha2)*
type : name (see Type List)
alpha1 : alpha (A-Z)

Write memorized locations to header.
..

GETSEGSIZE type segment
type : name (see Type List)
segment : name

Write size of a segment to header.
..

GETSEGOFFSET type segment
type : name (see Type List)
segment : name

Write offset of a segment to header.
..

WRITE type item2 *(**operator** value)*
type : name (see Type List)
item2 : name (see List below)

Write header item.
..

Item List (2)

BLOCKCNT	Block Count
LASTBLOCKSIZE	Size of the last block
RELOCCNT	Relocation Count
SIZE	Header Size
STACK	Stack Size

Outputs

What is an Output?

An "Output" holds settings for output files.

Create an Output

```
.OUTPUT name  identname  ext  header  blockSize  limit  fill  round  (pattern)
                name              : string    : Name / Description of Output
                identname         : name      : Identifier name
                ext               : string    : File Extension
                header            : name      : Set Header
                blockSize         : value     : Block Size
                limit             : value     : Set maximum size of Output
                                                0 = unlimited size
                fill              : value     : Fill to requested size
                round             : value     : Round Output
                pattern           : name      : Use String Equate as
                                                fill/round pattern
```

Example:

```
.OUTPUT 'Com File' com 'com' com_hdr 0 0 0 0    ; Create Output
com                                             ; Set Output
```

System Objects

Disk Object
Contains instructions to control disks.

Syntax:

```
.DISK item

item                Item list
                    FILEEXISTS  targetEquate  filename
                                    targetEquate      : name   (integer)
                                    filename          : string

                    Returns true (non-zero) if a file exists.
                    ..

                    GETDIR  targetEquate
                                    targetEquate      : name   (string)

                    Returns the current path of the active disk.
                    ..

                    GETDISKDIR  targetEquate  driveID
                                    targetEquate      : name (string)
                                    driveID           : value

                    Returns the current path of the selected disk.
                    DriveID's
                    0 = Current Disk
                    1 = A
                    2 = B
                    3 = C
                    4 = D...
                    ..
```

Example:

```
.DISK FileExists  ex  'c:\windows\notepad.exe'
.IF ex != 0
   ECHO 'File exists!!!'
.ELSE
   ECHO 'Not found...  :-('
.ENDIF
```

Dynamic Array Object
Dynamic Arrays can store values, strings and names.

Syntax:

```
.DYNARRAY item

item                Item list
                    NEW  objName  type  initial-size
                                    objName          : name
                                    type             : name=value
                                                       name=string
                                                       name=name
                                    initial-size     : value

                    Create a new dynamic array.
                    ..

                    SETLENGTH  objName  size
                                    objName          : name
                                    size             : value

                    Set new length of array.
                    ..

                    GETLENGTH  objName  targetEquate
                                    objName          : name
                                    targetEquate     : name (integer)

                    Get length of array.
                    ..

                    GETTYPE  objName  targetEquate
                                    objName          : name
                                    targetEquate     : name (integer)

                    Get array type (0 = value, 1 = string, 2 = name).
                    ..

                    SETITEM  objName  index  data
                                    objName          : name
                                    index            : value
                                    data             : value / string

                    Set array item.
                    ..

                    GETITEM  objName  index  targetEquate
                                    objName          : name
                                    index            : value
                                    targetEquate     : name (integer or string)

                    Get array item. Works only with Value- and String-arrays.
                    ..

                    GETNAME  objName  index
                                    objName          : name
                                    index            : value

                    Get name from array. Return the name with @ITEMNAME@.
                    ..
```

Example:

```
.DYNARRAY  New  myArray  value  10
.FOR i = 0 TO 9
   .DYNARRAY  SetItem  myArray  i  100 + i
.NEXT

.FOR i = 0 TO 9
   .DYNARRAY  GetItem  myArray  i  val
   ECHO  val
.NEXT

.DYNARRAY  New  array2  name  3
.DYNARRAY  SetItem  array2  0  score
.DYNARRAY  SetItem  array2  1  myLbl
.DYNARRAY  SetItem  array2  2  something

.DYNARRAY  GetName  array2  1

@itemname@:
```

File Object
Create, write and read files.

Syntax:

.FILE *item*

item Item list

NEW objName filename
objName : name
filename : string

Create file object.
..

OPEN objName filename method
objName : name
filename : string
method : name=write / name=read

Open file for writing / reading.
..

CLOSE objName
objName : name

Close an open file.
..

SEEK objName position
objName : name
position : value

Move the file pointer.
..

GETPOS objName targetEquate
objName : name
targetEquate : name *(integer)*

Get position of file pointer.
..

GETLEN objName targetEquate
objName : name
targetEquate : name *(integer)*

Get file size (in bytes).
..

GETMODE objName targetEquate
objName : name
targetEquate : name *(integer)*

Get File Mode.
0 = Not opened
1 = Open for reading
2 = Open for writing
..

GETERROR objName targetEquate
objName : name
targetEquate : name *(integer)*

Get Error/Status Code.
0 = Status OK.
..

ISEOF objName targetEquate
objName : name
targetEquate : name *(integer)*

Is pointer at end of file?
0 = No
1 = Yes
..

WRITE objName *item(2)*
objName : name
item(2) : name (see list below)

Write to file.
..

READ objName *item(3)*
objName : name
item(3) : name (see list below)

Read from file.
..

Item list (2)
VALUE type value
type : name (see Type List)
value : value

Write value to file.
..

STRING string
string : string

Write a string to file.
..

STRINGLN string
string : string

Write a string to file and insert a new line.
..

BLOCK type dynArray startIndex count
type : name (see Type List)
dynArray : name
startIndex : value
count : value

Write contents of a dynamic array to file.
..

```
Item List (3)
VALUE  type  targetEquate
                  type              : name (see Type List)
                  targetEquate      : name (integer)

Read value from file.
..

STRING  targetEquate
                  targetEquate      : name (string)

Read string from file.
..

BLOCK  type  dynArray  startIndex  count
                  type              : name (see Type List)
                  dynArray          : name
                  startIndex        : value
                  count             : value

Read block from file and store it into a dynamic array.
..
```

Example:

```
.FILE  new  myFile  ''
.FILE  open  myFile  'c:\test.txt'  write
.FILE  write  myFile  stringln  'Hello, World!'
.FILE  write  myFile  string  'Good bye!'
.FILE  close  myFile
```

Generator Object
This object generates output code, from values to memory addresses.

Syntax:

```
.GEN  item

item                Item List
                    VALUE  type  value
                                    type              : name (see Type List)
                                    value             : value

                    Write value to segment.
                    ..

                    TEXT  string
                                    string            : string

                    Write string to segment.
                    ..

                    RELADDRESS  type  label (operator  value)
                                    type              : name (see Type List)
                                    label             : name

                    Write relative adress to segment.
                    ..

                    MEMADDRESS  type  memory (operator  value)
                                    type              : name (see Type List)
                                    memory            : name

                    Write memory adress to segment.
                    ..

                    SEGOFFSET  type  segname (operator  value)
                                    type              : name (see Type List)
                                    segname           : name

                    Write offset of a segment to segment.
                    ..

                    SEGSIZE  type  segname (operator  value)
                                    type              : name (see Type List)
                                    segname           : name

                    Write size of a segment to segment.
                    ..

                    GETRELOCITEM  index  targetEquate
                                    index             : value
                                    targetEquate      : name (integer)

                    Get value from relocation table.
                    ..
```

```
GETRELOCLEN  targetEquate
                  targetEquate      : name (integer)

Get number of relocation items.
..

GETPROCESSORVALUE  targetEquate
                  targetEquate      : name (integer)

Get the current processor value.
..

GETREGCODE  regName  targetEquate
                  regName           : name
                  targetEquate      : name (integer)

Get register code.
..

GETREGVALUE  regName  targetEquate
                  regName           : name
                  targetEquate      : name (integer)

Get register value.
..
```

Get Object

Get application and parameter values.

Syntax:

```
.GET  item

item              Item List
                  PARAMETER  targetEquate  paramIndex
                                    targetEquate      : name (string)
                                    paramIndex        : value

                  Get macro/object parameter.
                  ..

                  PARAMCOUNT  targetEquate
                                    targetEquate      : name (integer)

                  Get number of macro/object parameters.
                  ..

                  MAINSOURCE  targetEquate
                                    targetEquate      : name (string)

                  Get filename of main source.
                  ..

                  INITDATASOURCE  targetEquate
                                    targetEquate      : name (string)

                  Get filename of initial data source.
                  ..

                  ENDDATASOURCE  targetEquate
                                    targetEquate      : name (string)
```

Get filename of end data source.
..

OUTPUTFILE targetEquate
targetEquate : name *(string)*

Get filename of output file.
..

CURRENTFILE targetEquate
targetEquate : name *(string)*

Get filename of current file.
..

MAJORVERSION targetEquate
targetEquate : name *(integer)*

Get BLASM major version.
..

MINORVERSION targetEquate
targetEquate : name *(integer)*

Get BLASM minor version.
..

BUILD targetEquate
targetEquate : name *(integer)*

Get BLASM build version.
..

RELEASE targetEquate
targetEquate : name *(integer)*

Get BLASM release version.
..

DATE targetEquate
targetEquate : name *(string)*

Get current date.
..

TIME targetEquate
targetEquate : name *(string)*

Get current time.
..

EXTRACTFILEDIR targetEquate filename
targetEquate : name *(string)*
filename : string

Extract Directory of a filename.
..

EXTRACTFILEEXT targetEquate filename
targetEquate : name *(string)*
filename : string

Extract File Extension of a filename.
..

```
EXTRACTFILENAME  targetEquate  filename
                 targetEquate      : name (string)
                 filename          : string

Extract filename (remove directory part) of a filename.
..
EXTRACTFILEPATH  targetEquate  filename
                 targetEquate      : name (string)
                 filename          : string

Extract File Path of a filename.
..

ARRAYCOUNT  targetEquate  array
                 targetEquate      : name (value)
                 array             : name

Get number of items of an array.
..
```

ModR/M Object

Generates values from ModR/M Tables.

Syntax:

```
.MODRM  item

item             Item List
                 RESET

                 Reset object.
                 ..

                 GETRETURNVALUE  targetEquate
                                   targetEquate      : name (integer)

                 Get the result after generating value.
                 ..

                 GENSTANDARD

                 Standard ModR/M generation, reg-to-reg.
                 ..

                 GENDIGIT

                 ModR/M generation using a digit.
                 ..

                 GENSPECIAL

                 Special ModR/M generation using a value.
                 ..

                 SETTABLE  table
                                   table             : name

                 Set ModR/M Table.
                 ..
```

SETBITS bits
bits : value

Set Bit Value (Filter).
..

SETREG reg
reg : name

Set register.
..

SETREG1 reg
reg : name

Set register 1.
..

SETREG2 reg
reg : name

Set register 2.
..

SETREG3 reg
reg : name

Set register 3.
..

CLEARTABLE

Remove ModR/M Table from Object.
..

CLEARREG

Remove register from Object.
..

CLEARREG1

Remove register 1 from Object.
..

CLEARREG2

Remove register 2 from Object.
..

CLEARREG3

Remove register 3 from Object.
..

SETBRACKETCODE code
code : value

Set Bracket Code, **0-6**.
0 = No brackets used
1 = [reg1]+reg2+reg3
2 = reg1+[reg2]+reg3
3 = reg1+reg2+[reg3]
4 = [reg1+reg2]+reg3

```
5 = reg1+[reg2+reg3]
6 = [reg1+reg2+reg3]
..

SETDIGIT  digit
                  digit             : value

Set digit.
..

SETVALUE  value
                  value             : value

Set value.
..
```

Parser Object

This object can be used for reading names/values from String Equates, Macro/Object Parameters and Files. You can use this for making your own assemblers/compilers etc.

Syntax:

```
.PARSER  item

item              Item List
SET  string
                  string            : string

Set String Equate.
..

GETITEM  dynArray  index
                  dynArray          : name
                  index             : value

Get item from dynamic array.
..

GETPARAM  paramIndex
                  paramIndex        : value

Get macro/object parameter.
0 = first parameter, 1 = second and so on,...
..

SETPOS  position
                  position          : value

Set reading position. 0 = first character.
..

NEXTCHAR

Point to the next character.
..

SKIPBLANK

Skip spaces and TABs from the current position.
..
```

IGNORECHAR char
char : value

Ignore/Skip characters from the current position.
..

GETNAME

Read a name from the current position. The readed name can be returned by using **@PARSENAME@**.
..

GETNUMBER type targetEquate
type : name (see Type List)
targetEquate : name *(integer)*

Read a value (expressions) from the current position.
..

GETCALC targetEquate
targetEquate : name *(integer)*

Read an operator + expression.
..

GETOPR targetEquate
targetEquate : name *(integer)*

Get the initial operator, after using **GETCALC**.
0 = none
1 = addition
2 = subtraction
3 = multiply
4 = division
5 = modulo
6 = power ^
7 = shl
8 = shr
..

GETSTRING targetEquate
targetEquate : name *(string)*

Read a string from the current position.
..

GETBASICSTRING targetEquate
targetEquate : name *(string)*

Same as **GETSTRING**, but doesn't support Equates and **"+"**.
..

GETNUM targetEquate
targetEquate : name *(integer)*

Read a value from the current position.
..

NAMETOEQU targetEquate
targetEquate : name *(string)*

After using **GETNAME**, you can use this instruction
to create a String Equate containing the readed name.
..

ISENDLINE targetEquate
targetEquate : name *(integer)*

Is the end of the parse string reached?
0 = no
1 = yes
..

GETPOS targetEquate
targetEquate : name *(integer)*

Get the current position.
..

GETLEN targetEquate
targetEquate : name *(integer)*

Get the length of the parse string.
..

GETCHAR targetEquate
targetEquate : name *(integer)*

Get the current character.
..

GETCHARAT position targetEquate
position : value
targetEquate : name *(integer)*

Get character from the selected position.
..