Directive

COLLABORATORS			
	TITLE : Directive		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 23, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Directive

1.1 Directive

```
                                Directive
                                ---------

A Directive is an instruction for the assembler which can be used in the source.

When assembling, the directive influence the way ASM-One assembles the source.

Many directives are used so much, that people think they are actual 68k opcodes,
like the 'DC' directive.

It's very usefull to examine all directives, it will not only make assembling
easier, but it will also give you more power when assembling.

Here's the complete list of ALL directives ASM-One knows:


 >EXTERN
 =
 *

 -- A --

 ADDWATCH
 ALIGN
 AUTO

 -- B --

 BASEREG
 BLK

 -- C --

 CMEXIT
 CNOP

 -- D --
```

```
DC
DCB
DR
DS

-- E --

ELSE
END
ENDB
ENDC
ENDIF
ENDM
ENDOFF
ENDR
ENTRY
EQU
EQUR
EREM
ETEXT
EVEN
EXTRN

-- F --

FAIL
FILESIZE

-- G --

GLOBAL

-- I --

IDNT
IF
IF1
IF2
IFB
IFC
IFD
IFNB
IFNC
IFND
IMAGE
INCBIN
INCDIR
INCIFF
INCIFFP
INCLUDE
INCSRC

-- J --

JUMPERR
JUMPPTR
```

```
-- L --

LINE_A
LINEA
LINE_F
LINEF
LIST
LLEN
LOAD

-- M --

MACRO
MASK2
MEXIT

-- N --

NOLIST
NOPAGE

-- O --

ODD
OFFSET
ORG

-- P --

PAGE
PLEN
PRINTT
PRINTV

-- R --

REM
REG
REPT
RORG
RS
RSRESET
RSSET

-- S --

SECTION
SET
SETCPU
SETFPU
SETMMU
SPC

-- T --

TEXT
TTL
```

```
 -- X --

 XDEF
 XREF
```

1.2 '>EXTERN' Directive

```
NAME
   >EXTERN - Load External Binaries

SYNTAX
   [label] >EXTERN [number,]<file>,<address>[,length]

FUNCTION
   This directive is mainly implemented for backwards compatibilty
   with (Master)Seka.

   With >EXTERN you can load files to a certain address. This is
   handy when you work with absolute addresses.

   When a [number] is specified, you can load that specific file
   with the  E  DLC.

   When no <length> is specified, the whole file will be loaded.

   The External binaries will be loaded after using the  E  DLC.

FROM VERSION
   V1.01
```

1.3 '=' Directive

```
NAME
   = - Assign a Value

SYNTAX
   <label> = <value>

FUNCTION
   This directive assigns the <value> to the <label>.

   Comes in handy when you have difficult to remember values.
   By giving them a name, they are better to remember.

VERSION
   V1.08

SEE ALSO
    EQU
```

1.4 '*' Directive

```
NAME
   * - Current Address

SYNTAX
   <label> *<logical operator>

FUNCTION
   * is the Current Address of the PC, when used in this manner
   (it can also define comments (just like ;) and it can be used
   to multiply values when used in an expresion).

   The <logical operator> will influence the Current Address, so
   it will point to another address.

   This directive is mainly used to point to addresses that can't
   be pointed to by a label directly.

EXAMPLE

   text:  dc.b 'hello there'
   there: *-5

   the label 'there' will point to the start of 'there' in the
   text string defined by label 'text'.

REMARK
   If you use this directive to assign values to an Instruction
   after the program was assembled, it will be regarded as
   'Self Modifing Code'.

   It's mainly used this way in old programs, to save the original
   address of an Interrupt, like this:

   code:      MOVE.L $7c,interrupt

   <more code>

   OwnInterrupt:
   <own interrupt code>
              JMP    $0   ; Original Interrupt Code
   interrupt: *-4

   It's STRONGLY suggested to remove this kind of coding. The 68020
   and higher CPU's have cache memory. There is the possibility that
   the instruction is already loaded into the cache memory, BEFORE
   it could be altered, resulting in completely wrong values

VERSION
   V1.01
```

1.5 'ADDWATCH' Directive

```
NAME
   ADDWATCH - Add label to Debugger Watch Points

SYNTAX
   ADDWATCH <label>

FUNCTION
   Will add the <label> to the Watch Points in the debugger.

VERSION
   V1.16
```

1.6 'ALIGN' Directive

```
NAME
   ALIGN - Align the next address

SYNTAX
   [label] ALIGN <value1>,<value2>

FUNCTION
   This directive is implemented for compatibility reasons.
   It has the same function as the  CNOP  directive.

   ALIGN is used to align the next address to a certain boundry.

   ALIGN looks for the first following address that can be divided by
   <value2>, and then adds <value1> to that address.

EXAMPLE

   ALIGN 0,4  ; The address will be dividable by 4 (is Longword aligned)
   ALIGN 2,4  ; The address will be dividable by 4, and then 2 is added
    ; (is Word aligned)

REMARK
   ALIGN doesn't initilize anything. Bytes that are skipped becose
   ALIGN was used, will have NO particular value. And it is STRONGLY
   suggested you never use them:

   label: ALIGN 0,4
   here:  *-2

   Becose it will produce unpredictable results.

VERSION
   V1.08

SEE ALSO
    CNOP
```

1.7 'AUTO' Directive

```
NAME
   AUTO - Automaticaly execute command(s)

SYNTAX
   AUTO <command>[\command..]

FUNCTION
   Usefull to automaticaly execute commands you would have to
   type otherwise.

   Commands are ALLWAYS DLC's.

   The Backslash (\) is used to seperate the commands

REMARK
   The maximum number of characters for AUTO is 256.

VERSION
   V1.01
```

1.8 'BASEREG' Directive

```
NAME
   BASEREG - Set Base for Register

SYNTAX
   BASEREG <label>,<register>

FUNCTION
   Assigns a BASE to an ADDRESS register.

   BASEREG works for all Address registers, EXCEPT A7 (SP) !!

EXAMPLE
   Example without the use of BASEREG:

             LEA     DataArea,A4
             MOVE.W  D0,DataWord-DataArea(A4)
             ---

   DataArea: DCB.B   100,0
   DataWord: DC.W    0

   Same example, but now with the use of BASEREG:

             BASEREG DataArea,A4
             LEA     DataArea,A4
             MOVE.W  D0,DataWord(A4)
             ---

   DataArea: DCB.B   100,0
   DataWord: DC.W    0
```

```
VERSION
   V1.01

SEE ALSO
    ENDB
```

1.9 'BLK' Directive

```
NAME
   BLK - Define a Block of Constants

SYNTAX
   [label] BLK.[size] <value1>,<value2>

FUNCTION
   Mainly here becose of backwards compatibility with older
   Assemblers.

   BLK will define a block of constants, the constants will
   have the size of [size], and will be initialized to
   <value2>.

   <value1> specifies the number of constants to generate.

   Legal sizes are: B (byte), W (word), L (longword), D (double),
   P (packed), S (single) and X (extended)

VERSION
   V1.01

SEE ALSO
    DCB
```

1.10 'CMEXIT' Directive

```
NAME
   CMEXIT - Leave MACRO when Nesting Depth Reached

SYNTAX
   CMEXIT <value>

FUNCTION
   Will leave the MACRO if the nesting depth specified by
   <value> has been reached. Regardless of the fact that
   the MACRO was not fully executed.

VERSION
   V1.01

SEE ALSO
    Building and Using MACRO's
```

1.11 'CNOP' Directive

```
NAME
   CNOP - Align the next address

SYNTAX
   [label] CNOP <value1>,<value2>

FUNCTION
   CNOP is used to align the next address to a certain boundry.

   CNOP looks for the first following address that can be divided by
   <value2>, and then adds <value1> to that address.

EXAMPLE

   CNOP  0,4  ; The address will be dividable by 4 (is Longword aligned)
   CNOP  2,4  ; The address will be dividable by 4, and then 2 is added
    ; (is Word aligned)

REMARK
   CNOP doesn't initilize anything. Bytes that are skipped becose
   CNOP  was used, will have NO particular value. And it is STRONGLY
   suggested you never use them:

   label: CNOP 0,4
   here:  *-2

   Becose it will produce unpredictable results.

VERSION
   V1.01

SEE ALSO
    ALIGN
```

1.12 'DC' Directive

```
NAME
   DC - Define Constant

SYNTAX
   [label] DC.[size] <expresion>[,expresion...]

FUNCTION
   DC defines a constant. The result of <expresion> specifies
   the value of the constant.

   Legal sizes are: B (byte), W (word), L (longword), D (double),
   P (packed), S (single) and X (extended)

   When the size is bytes, you can also use strings:

   text: DC.B 'Hello, here I am'
```

```
VERSION
   V 1.01

SEE ALSO
   For textstrings:  TEXT
```

1.13 'DCB' Directive

```
NAME
   DCB - Define a Block of Constants

SYNTAX
   [label] DCB.[size] <value1>,<value2>

FUNCTION
   DCB will define a block of constants, the constants will
   have the size of [size], and will be initialized to
   <value2>.

   <value1> specifies the number of constants to generate.

   Legal sizes are: B (byte), W (word), L (longword), D (double),
   P (packed), S (single) and X (extended)

VERSION
   V1.01

SEE ALSO
    BLK
```

1.14 'DR' Directive

```
NAME
   DR - Define Relative Value

SYNTAX
   [label] DR.[size] <value>

FUNCTION
   DR gives you the posibility to make a table with relative
   values.

   Each DR hase the following value:

   DR.[size] <value>-*  ; Where * is the Current Address

   Legal sizes are: B (byte), W (word) and L (longword)

EXAMPLE
   Example without the use of DR:
```

```
   JUMP:   LEA    DATA(PC),A0
           ADD.W  D0,D0
           MOVE.W (A0,D0.W),A0
           JMP    (A0)

   DATA:   DC.W   ROUTINE_ONE-DATA
           DC.W   ROUTINE_TWO-DATA
           DC.W   ROUTINE_THREE-DATA

   Example WITH DR:

   JUMP:   LEA    DATA(PC),A0
           ADD.W  D0,D0
           ADD.W  D0,A0
           ADD.W  (A0),A0
           JMP    (A0)

   DATA:   DR.W   ROUTINE_ONE
           DR.W   ROUTINE_TWO
           DR.W   ROUTINE_THREE

VERSION
   V1.01
```

1.15 'DS' Directive

```
NAME
   DS - Defines space for variables

SYNTAX
   [label] DS.[size] <value>

FUNCTION
   Defines space for variables. <value> specifies
   the number of spaces to define. All spaces will
   have the size of [size]

   Legal sizes are: B (byte), W (word), L (longword), D (double),
   P (packed), S (single) and X (extended)

   The space defined will NOT be initialized !!!!

REMARK
   DS can ONLY be used in an BSS setion !!!

VERSION
   V1.01
```

1.16 'ELSE' Directive

```
NAME
   ELSE - Jumps to alternative code when the IF-statement is FALSE
```

```
SYNTAX
   ELSE

FUNCTION
   Gives you the posibility to define code to execute when the
   IF-statemant is FALSE.

VERSION
   V1.01

SEE ALSO
    Conditional Branches with IF
```

1.17 'END' Directive

```
NAME
   END - Define the end of the source

SYNTAX
   END

FUNCTION
   Sourcecode after this directive will be skipped by
   ASM-One.

   Normaly ASM-One will put END at the end of the Source.
   But you can define it yourself if you wish.

VERSION
   V1.01
```

1.18 'ENDB' Directive

```
NAME
   ENDB - End BASEREG Section

SYNTAX
   ENDB <address register>

FUNCTION
   Will deactivate the function of BASEREG for the given
   <address register>.

VERSION
   Untill version V1.09: BASEREG could only be used ONE time
   V1.09 and up: After ENDB, you can use BASEREG again for the SAME register
```

1.19 'ENDC' Directive

```
NAME
   ENDC - End IF-block

SYNTAX
   ENDC

FUNCTION
   Ends an IF-block.

   Implemented for compatibility reasons.

VERSION
   V1.01

SEE ALSO
    ENDIF ,  Conditional Branches with IF
```

1.20 'ENDIF' Directive

```
NAME
   ENDIF - End IF-block

SYNTAX
   ENDIF

FUNCTION
   Ends an IF-block.

VERSION
   V1.01

SEE ALSO
    ENDC ,  Conditional Branches with IF
```

1.21 'ENDM' Directive

```
NAME
   ENDM - End MACRO definition

SYNTAX
   ENDM

FUNCTION
   Specifies the end of a MACRO definition.

VERSION
   V1.01

SEE ALSO
    MACRO ,  Building and Using MACRO's
```

1.22 'ENDOFF' Directive

```
NAME
   ENDOFF - End OFFSET definition

SYNTAX
   ENDOFF

FUNCTION
   Specifies the end of an OFFSET definition.

VERSION
   V1.15

SEE ALSO
    OFFSET
```

1.23 'ENDR' Directive

```
NAME
   ENDR - End REPT block

SYNTAX
   ENDR

FUNCTION
   Specifies the end a REPT block.

VERSION
   V1.01

SEE ALSO
    REPT
```

1.24 'ENTRY' Directive

```
NAME
   ENTRY - External Definition

SYNTAX
   ENTRY <label>[,label...]

FUNCTION
   Will define <label> as an external value, so it can be
   used by other modules.

   This is mainly interesting for linking assembeler routines
   into code made by higher programming languages.

    You will need a linker (like BLink) to link the code.

EXAMPLE
```

```
   See  XREF

REMARK
   Sources that contain ENTRY, are not executables. And therefore
   can NOT be executed !!

   You will need the  WO  to write the
   assembled object file.

   ENTRY has the same function as XDEF, EXTRN and GLOBAL.

VERSION
   V1.01

SEE ALSO
    XDEF ,  XREF ,  GLOBAL ,  EXTRN ,  WO
```

1.25 'EQU' Directive

```
NAME
   EQU - Assign a Value

SYNTAX
   <label> EQU <value>

FUNCTION
   This directive assigns the <value> to the <label>.

   Comes in handy when you have difficult to remember values.
   By giving them a name, they are better to remember.

VERSION
   V1.01

SEE ALSO
    = ,  SET
```

1.26 'EQUR' Directive

```
NAME
   EQUR = Give Register a Name

SYNTAX
   <label> EQUR <register>

FUNCTION
   Assigns a <label> to the specified <register>, so
   it's easier to remember.

   Only Address- and Data registers are allowed to be used.

EXAMPLE
```

```
   Bitplane1 EQUR    A3
             MOVE.L  D0,(Bitplane1)+
```

REMARK
   <label> can’t be defined again, but registers can have
   more than one name.

VERSION
   V1.01

1.27 ’EREM’ Directive

NAME
   EREM - End REM block

SYNTAX
   EREM

FUNCTION
   Specifies the end of a REM block

VERSION
   V1.15

SEE ALSO
   REM

1.28 ’ETEXT’ Directive

NAME
   ETEXT - End TEXT block

SYNTAX
   ETEXT

FUNCTION
   Specifies the end of a TEXT block.

VERSION
   V1.15

SEE ALSO
   TEXT

1.29 ’EVEN’ Directive

NAME
   EVEN - Make next address even

SYNTAX

```
   [label] EVEN

FUNCTION
   Will make the next address an EVEN address.

   EVEN has the same function as:

   CNOP   0,2

VERSION
   V1.01

SEE ALSO
    CNOP ,  ODD ,  ALIGN
```

1.30 'EXTRN' Directive

```
NAME
   EXTRN - External Definition

SYNTAX
   EXTRN <label>[,label...]

FUNCTION
   Will define <label> as an external value, so it can be
   used by other modules.

   This is mainly interesting for linking assembeler routines
   into code made by higher programming languages.

    You will need a linker (like BLink) to link the code.

EXAMPLE
   See  XREF

REMARK
   Sources that contain EXTRN, are not executables. And therefore
   can NOT be executed !!

   You will need the  WO  to write the
   assembled object file.

   EXTRN has the same function as XDEF, ENTRY and GLOBAL.

VERSION
   V1.01

SEE ALSO
    XDEF ,  XREF ,  GLOBAL ,  ENTRY ,  WO
```

1.31 'FAIL' Directive

```
NAME
   FAIL - Generate Error

SYNTAX
   FAIL

FUNCTION
   FAIL will generate the 'User made FAIL' error.

   You can use FAIL in IF Statements, just using them in
   your source, will have an irretating effect.

VERSION
   V1.01

SEE ALSO
    Conditional Branches with IF
```

1.32 'FILESIZE' Directive

```
NAME
   FILESIZE - Get the size of a file

SYNTAX
   FILESIZE(<file>)

FUNCTION
   Enables you to get the size of a file without including
   or opening it.

   Great for allocating memory for files.

REMARK
   Can also be used as expresion !!

VERSION
   V1.29
```

1.33 'GLOBAL' Directive

```
NAME
   GLOBAL - External Definition

SYNTAX
   GLOBAL <label>[,label...]

FUNCTION
   Will define <label> as an external value, so it can be
   used by other modules.

   This is mainly interesting for linking assembeler routines
   into code made by higher programming languages.

    You will need a linker (like BLink) to link the code.

EXAMPLE
   See  XREF

REMARK
   Sources that contain GLOBAL, are not executables. And therefore
   can NOT be executed !!

   You will need the  WO  to write the
   assembled object file.

   GLOBAL has the same function as XDEF, ENTRY and EXTRN.

VERSION
   V1.01

SEE ALSO
    XDEF ,  XREF ,  EXTRN ,  ENTRY ,  WO
```

1.34 'IDNT' Directive

```
NAME
   IDNT - Identify Program

SYNTAX
   IDNT <string>

FUNCTION
   Normaly, a program with more than 2 sections should have a name.
   When no name is given, ASM-One will assign an empty string
   as name.

VERSION
   V1.01
```

1.35 'IF' Directive

```
NAME
   IF - Conditional Branch Option

SYNTAX
   IF(cc) <bolean>

FUNCTION
   IF allows you to include/exclude parts of the source when
   assembling, based on the test of the <bolean>. With (cc)
   you can specify on which condition something happens.

   EQ = Equal
   NE = Not Equal
   GT = Greater Than
   GE = Greater or Equal
   LT = Lower Than
   LE = Lower or Equal

   Generaly, the <bolean> is tested. And the result is compared
   to 0.

VERSION
   V1.01

SEE ALSO
    Conditional Branches with IF
```

1.36 'IF1' Directive

```
NAME
   IF1 - Assemble Pass 1

SYNTAX
   IF1

FUNCTION
   What follows between IF1 and ENDIF, will only be assembled
   in Pass 1.

VERSION
   V1.01

SEE ALSO
    Conditional Branches with IF
```

1.37 'IF2' Directive

```
NAME
   IF2 - Assemble Pass 2

SYNTAX
   IF2

FUNCTION
   What follows between IF1 and ENDIF, will only be assembled
   in Pass 2.

VERSION
   V1.01

SEE ALSO
    Conditional Branches with IF
```

1.38 'IFB' Directive

```
NAME
   IFB - Assembles when empty

SYNTAX
   IFB <symbol>

FUNCTION
   I've tested this, but the only thing I come up with is that
   what ever <symbol> is, the result is always FALSE !!!

   Except when NO <symbol> is given, then the result is TRUE !!

VERSION
   V1.01

SEE ALSO
    IFNB ,  Conditional Branches with IF
```

1.39 'IFC' Directive

```
NAME
   IFC - Assembles when strings are equal

SYNTAX
   IFC <string1>,<string2>

FUNCTION
   Compares both strings, when equal, the result is TRUE.

VERSION
   V1.01

SEE ALSO
    IFNC ,  Conditional Branches with IF
```

1.40 'IFD' Directive

```
NAME
   IFD - Assembles when symbol is defined

SYNTAX
   IFD <symbol>

FUNCTION
   The test is set to TRUE when de <symbol>
   is defined.

VERSION
   V1.01
```

```
SEE ALSO
    IFND ,  Conditional Branches with IF
```

1.41 'IFNB' Directive

```
NAME
   IFNB - Assembles when empty

SYNTAX
   IFNB <symbol>

FUNCTION


VERSION
   V1.01

SEE ALSO
    IFB ,  Conditional Branches with IF
```

1.42 'IFNC' Directive

```
NAME
   IFNC - Assembles when strings are not equal

SYNTAX
   IFNC <symbol1>,<symbol2>

VERSION
   V1.01

SEE ALSO
    IFC ,  Conditional Branches with IF
```

1.43 'IFND' Directive

```
NAME
   IFND - Assmbles when symbol is not defined

SYNTAX
   IFND <symbol>

VERSION
   V1.01

SEE ALSO
    IFD ,  Conditional Branches with IF
```

1.44 'IMAGE' Directive

```
NAME
   IMAGE - Include Binary File

SYNTAX
   IMAGE <file>[,<address>]

FUNCTION
   Will load the file specified by <file> in to memory.

   When an <address> is specified, the file will be loaded at this
   address.

   IMAGE has the same functions as INCBIN.

VERSION
   V1.01

SEE ALSO
    INCBIN ,  INCDIR
```

1.45 'INCBIN' Directive

```
NAME
   INCBIN - Include Binary File

SYNTAX
   INCBIN <file>[,<address>]

FUNCTION
   Will load the file specified by <file> in to memory.

   When an <address> is specified, the file will be loaded at this
   address.

VERSION
   V1.01

SEE ALSO
    IMAGE ,  INCDIR
```

1.46 'INCDIR' Directive

```
NAME
   INCDIR - Specify Include Directory

SYNTAX
   INCDIR <path>

FUNCTION
   Normaly, ASM-One will only look in the current directory for
   INCLUDEs/INCBINs when no path was specified.

   With this directive you can set a directory where ASM-One
   should look first for INCLUDEs/INCBINs.

   INCDIR works for:

    IMAGE
    INCBIN
    INCIFF
    INCIFFP
    INCLUDE

REMARK
   You will have to include the whole path as <path>:

   INCDIR "Work:sources"

   will not work, but"

   INCDIR "Work:sources/"

   will.....

VERSION
   V1.01
```

1.47 'INCIFF' Directive

```
NAME
   INCIFF - Include IFF file

SYNTAX
   INCIFF

FUNCTION


VERSION
   V1.25

SEE ALSO
    INCIFFP ,  INCDIR
```

1.48 'INCIFFP' Directive

```
NAME
   INCIFFP - Include IFF palette

SYNTAX
   INCIFFP
```

FUNCTION

VERSION
   V1.25

SEE ALSO
    INCIFF , INCDIR

1.49 'INCLUDE' Directive

NAME
   INCLUDE - Include Source

SYNTAX
   INCLUDE <file>

FUNCTION
   Will include the file specified by <file> as source
   into the current source.

REMARK
   To make assembling faster, INCLUDEs are only loaded once.
   When includes changes, you can use the ZI DLC to flush
   all includes.

VERSION
   V1.01

SEE ALSO
    ZI , INCDIR , INCSRC

1.50 'INCSRC' Directive

NAME
   INCSRC - Include Source from ASM-One

SYNTAX
   INCSRC <sourcenumber>

FUNCTION
   With this directive you can include one of the 10 source
   of ASM-One into your current source.

   <sourcenumber> can be a number from 0 till 9.

EXAMPLE
   You are in Source 0, and you want to include Source 1:

```
   INCSRC 1
```

REMARK

```
   Sources included with INCSRC are assembled every time you
   assemble the source which has the INCSRC directive.

   This means you will not have to use the  ZI  DLC to flush
   certain includes you are working on.

VERSION
   V1.25
```

1.51 'JUMPERR' Directive

```
NAME
   JUMPERR - Jump to your own error routine

SYNTAX
   JUMPERR <label>

FUNCTION
   Gives you to possibility to specify your own error routine
   when something happens when you ran your program with the
    J  DLC.

   ASM-One will first jump to <label> before returning to
   the DLC.

REMARK
   This will not work hen a fatal error occures.

VERSION
   V1.01
```

1.52 'JUMPPTR' Directive

```
NAME
   JUMPPTR

SYNTAX
   JUMPPTR <label>

FUNCTION
   Specifies the label where ASM-One should start with debugging, or
   the starting label when you use the  J  or  G
   without an address or label.

VERSION
   V1.01
```

1.53 'LINE_A' Directive

```
NAME
   LINE_A - Generate a LINE A Exception

SYNTAX
   LINE_A <word>

FUNCTION
   Will generate a LINE A exception. The <word> is the
   a value that is usefull for the routine that handles
   the LINE A exception.

REMARK
   A LINE A is normaly used to specify your on routine
   for instructions that are not implemented in the CPU, FPU
   or MMU.

   Make sure you have read abook about the M68000 Family of
   processors, because most LINE A and LINE F possibilities are
   already used.

VERSION
   V1.01
```

1.54 'LINEA' Directive

```
NAME
   LINEA - Generate a LINE A Exception

SYNTAX
   LINEA <word>

FUNCTION
   Will generate a LINE A exception. The <word> is the
   a value that is usefull for the routine that handles
   the LINE A exception.

REMARK
   A LINE A is normaly used to specify your on routine
   for instructions that are not implemented in the CPU, FPU
   or MMU.

   Make sure you have read abook about the M68000 Family of
   processors, because most LINE A and LINE F possibilities are
   already used.

VERSION
   V1.01
```

1.55 'LINE_F' Directive

```
NAME
```

```
   LINE_F - Generate a LINE F Exception

SYNTAX
   LINE_F <word>

FUNCTION
   Will generate a LINE F exception. The <word> is the
   a value that is usefull for the routine that handles
   the LINE F exception.

REMARK
   A LINE F is normaly used to specify your on routine
   for instructions that are not implemented in the CPU, FPU
   or MMU.

   Make sure you have read abook about the M68000 Family of
   processors, because most LINE A and LINE F possibilities are
   already used.

VERSION
   V1.01
```

1.56 'LINEF' Directive

```
NAME
   LINEF - Generate a LINE F Exception

SYNTAX
   LINEF <word>

FUNCTION
   Will generate a LINE F exception. The <word> is the
   a value that is usefull for the routine that handles
   the LINE F exception.

REMARK
   A LINE F is normaly used to specify your on routine
   for instructions that are not implemented in the CPU, FPU
   or MMU.

   Make sure you have read abook about the M68000 Family of
   processors, because most LINE A and LINE F possibilities are
   already used.

VERSION
   V1.01
```

1.57 'LIST' Directive

```
NAME
   LIST - Activate Listing
```

```
SYNTAX
   LIST

FUNCTION
   Will override the 'List File' Preference.

   The list will start where LIST was in the source.

VERSION
   V1.01

SEE ALSO
    NOLIST , ASM-One's Preferences,  ASM-One Pref file
    PAGE ,  SPC ,  TTL
```

1.58 'LLEN' Directive

```
NAME
   LLEN - Specify line length

SYNTAX
   LLEN <length>

FUNCTION
   Specifies the length of each line when you print something.

   <length> must be a number between 60 and 132.

VERSION
   V1.01

SEE ALSO
    PLEN
```

1.59 'LOAD' Directive

```
NAME
   LOAD - Specify load address

SYNTAX
   [label] LOAD <address>

FUNCTION
   If specified, the code will be assembled starting at
   <address>.

   In combination with  ORG  you can assemble code
   at an absolute address.

VERSION
   V1.01
```

1.60 'MACRO' Directive

```
NAME
   MACRO - Start MACRO definition

SYNTAX
   [label] MACRO

FUNCTION
   Allowes you to specify MACRO's.

VERSION
   V1.01

SEE ALSO
    Building and Using MACRO's ,  CMEXIT ,  MEXIT ,  ENDM
```

1.61 'MASK2' Directive

```
NAME
   MASK2 - Unknown

SYNTAX
   MASK2

FUNCTION
   Unknown

   MASK2 is implemented for compatibility reasons with some
   old Assemblers !!!

VERSION
   V1.01
```

1.62 'MEXIT' Directive

```
NAME
   MEXIT - Leave MACRO

SYNTAX
   MEXIT

FUNCTION
   ASM-One will leave a MACRO when it encounters this directive.
   No matter if the macro was completed.

VERSION
   V1.01

SEE ALSO
    Building and Using MACRO's ,  MACRO ,  CMEXIT ,  ENDM
```

1.63 'NOLIST' Directive

```
NAME
   NOLIST - Stops the function of LIST

SYNTAX
   NOLIST

FUNCTION
   Stops the list that was started with LIST.

   This way you can generate a listing for only a part of the
   complete program.

VERSION
   V1.01

SEE ALSO
    LIST
```

1.64 'NOPAGE' Directive

```
NAME
   NOPAGE - Deactivate the PAGE option

SYNTAX
   NOPAGE

VERSION
   V1.01

SEE ALSO
    PAGE
```

1.65 'ODD' Directive

```
NAME
   ODD - Make next address odd

SYNTAX
   [label] ODD

FUNCTION
   Will make the next address an ODD address.

   ODD has the same function as:

   CNOP   1,2

VERSION
   V1.01
```

SEE ALSO
    CNOP , EVEN , ALIGN

1.66 'OFFSET' Directive

NAME
   OFFSET - Define Offsets

SYNTAX
   [label] OFFSET <value>

FUNCTION
   Allowes you to build offsets without calculation what the
   actual offset would be.

EXAMPLE

```
   start:     OFFSET    100
   dat0:      dc.b      1
   dat1:      dc.b      9
              ENDOFF
```

   This will give the labels 'start' and 'dat0' the value 100,
   and the label 'dat1' the value 100+1.

REMARK
   An OFFSET only works in the same section.

   An OFFSET is ended when ENDOFF , SECTION , OFFSET or END .

VERSION
   V1.01

1.67 'ORG' Directive

NAME
   ORG - Set absolute program start

SYNTAX
   [label] ORG <address>

FUNCTION
   In combination with LOAD, your program will be assembled starting
   at the address by LOAD, and will also be executed at the address
   specified by ORG.

VERSION
   V1.01

SEE ALSO
    LOAD

1.68 'PAGE' Directive

```
NAME
   PAGE - Start at new page in the listing

SYNTAX
   PAGE

FUNCTION
   PAGE will start a new page, and will also start a new page
   if the previous was full.

   It functions in combination with LIST.

VERSION
   V1.01

SEE ALSO
    NOPAGE ,  LIST
```

1.69 'PLEN' Directive

```
NAME
   PLEN - Set page length

SYNTAX
   PLEN <page-length>

FUNCTION
   Sets the length of the page for the printer.

   <page-length> must be a number between 20 and 100.

VERSION
   V1.01

SEE ALSO
    LLEN
```

1.70 'PRINTT' Directive

```
NAME
   PRINTT - PRINTT a string

SYNTAX
   PRINTT <string>

FUNCTION
   While assembling, the <string> will be printed (on screen) where
   PRINTT was in the source.

   Nice in combination with MACRO's.
```

```
VERSION
   V1.01

SEE ALSO
    Building and Using MACRO’s
```

1.71 ’PRINTV’ Directive

```
NAME
   PRINTV - Print value

SYNTAX
   PRINTV <label/value>[,label/value...]

FUNCTION
   Prints the value of <label> or just the <value> (on screen).

   Also very handy in combination with REPT and MACRO’s

EXAMPLE

   PRINTV StartingAddress

   ; -- Actual Source --

   StartingAddress:
         MOVEQ  #0,D0
         RTS

VERSION
   V1.01

SEE ALSO
    Building and Using MACRO’s ,  REPT
```

1.72 ’REM’ Directive

```
NAME
   REM - Add Remark

SYNTAX
   REM

FUNCTION
   Enables you (together with EREM) to out comment large pieces
   of code without the need to start every line with an semi-colon
   (;).

   Everything between REM and EREM will be ignored by ASM-One.

VERSION
```

```
   V1.15

SEE ALSO
    EREM
```

1.73 'REG' Directive

```
NAME
   REG - Assign label to RegisterList

SYNTAX
   <label> REG <registerlist>

FUNCTION
   Enables you to assign a name to a registerlist. This is
   very handy for MOVEM.

EXAMPLE

   AllRegs:  REG  D0-A6

             MOVEM.L  AllRegs,-(A7)

REMARK
   Only Data- and Address registers are allowed.
   The <label> can not be used again for something else.

VERSION
   V1.01
```

1.74 'REPT' Directive

```
NAME
   REPT - Repete something

SYNTAX
   REPT <number>

FUNCTION
   Allowes you to repete the same line(s) of coding several times
   automatically.

EXAMPLE

         REPT    20
         MOVE.B  (A3)+.(A2)+
         ENDR

REMARK
   From version V1.29 a minimum of 2 repts is needed, or an error
   will be generated.
```

```
VERSION
   V1.01
   Error from verion V1.29 and up.

SEE ALSO
    ENDR
```

1.75 'RORG' Directive

```
NAME
   RORG - Set relative start

SYNTAX
   [label] RORG <value>

FUNCTION
   Not complete clear, but:

   Will add <value> to the starting address of a section, so
   that the next address ill be the starting address plus <value>.

VERSION
   V1.01
```

1.76 'RS' Directive

```
NAME
   RS - Add value to RS counter

SYNTAX
   [label] RS.[size] <value>

FUNCTION
   Does add the <value> to the value of the internal RS counter.

   Legal sizes are: B (byte), W (word) and L (longword)

EXAMPLE

           RSRESET
           RS.B     -30   ; Start value
   Open:   RS.B     -6
   Close:  RS.B     -6
   Read:   RS.B     -6
   Write:  RS.B     -6

VERSION
   V1.01

SEE ALSO
    RSRESET ,  RSSET
```

1.77 'RSRESET' Directive

```
NAME
   RSRESET - Reset the RS counter

SYNTAX
   [label] RSRESET

FUNCTION
   Will reset the internal RS counter to zero.

VERSION
   V1.01

SEE ALSO
    RS ,  RSSET
```

1.78 'RSSET' Directive

```
NAME
   RSSET - Set RS counter

SYNTAX
   [label] RSSET <value>

FUNCTION
   Will reset the internal RS counter, and set the internal RS
   counter to <value>.

VERSION
   V1.01

SEE ALSO
    RS ,  RSRESET
```

1.79 'SECTION' Directive

```
NAME
   SECTION - Define a new section

SYNTAX
   [label] SECTION <name>[,type][_memory]

FUNCTION
   Will start a new section with <name> as name, of type [type],
   allocated in memory [memory]

   Legal types are: CODE, DATA and BSS (case is not important)

   Legal memory types are: _C (chip), _F (fast) and _P (public)

REMARK
```

```
   If no section is specified, ASM-One will start with a CODE
   section with the name 'Text'.

   A maximum of 255 section is allowed.

VERSION
   V1.01
```

1.80 'SET' Directive

```
NAME
   SET - Assign value

SYNTAX
   [label] SET <value>

FUNCTION
   Works the same as EQU, but the [label] can get another
   value assigned by SET is needed.

VERSION
   V1.01

SEE ALSO
    EQU
```

1.81 'SETCPU' Directive

```
NAME
   SETCPU - Set CPU type

SYNTAX
   SETCPU <option>

FUNCTION
   Allowes you to override the current settings for
   the CPU (as specified in the Preferences).

   Legal values for <option> are:

   000  : 68000
   010  : 68010
   020  : 68020
   030  : 68030
   040  : 68040
   060  : 68060
   PUSH : Store current value
   PULL : Restore stored value

VERSION
   V1.3x
```

```
SEE ALSO
    SETFPU ,  SETMMU
```

1.82 'SETFPU' Directive

```
NAME
   SETFPU - Set current FPU

SYNTAX
   SETFPU <option>

FUNCTION
   Will override the current Preference settings for the FPU.

   Legal values for <option> are:

   ON   : Set FPU on
   OFF  : SET FPU off
   PUSH : Store current value
   PULL : Restore stored value

VERSION
   V1.3x

SEE ALSO
    SETCPU ,  SETMMU
```

1.83 'SETMMU' Directive

```
NAME
   SETMMU - Set current MMU

SYNTAX
   SETMMU <option>

FUNCTION
   Allowes you to set the MMU regardless of the Preferences.

   Legal values for <option> are:

   ON   : Set MMU on
   OFF  : Set MMU off
   PUSH : Store current value
   PULL : Restore storde value

VERSION
   V1.3x

SEE ALSO
    SETCPU ,  SETFPU
```

1.84 'SPC' Directive

```
NAME
   SPC - Add empty lines

SYNTAX
   SPC <value>

FUNCTION
   When showing a listing (with LIST), you can specify with SPC
   a number of empty lines. The number of empty lines is specified
   by <value>

VERSION
   V1.01

SEE ALSO
    LIST
```

1.85 'TEXT' Directive

```
NAME
   TEXT - Start a text block

SYNTAX
   [label] TEXT

FUNCTION
   TEXT allowes you (together with ETEXT) to enter text without the
   addition of DC.B's.

   Adding a | will start the hex mode. Of every character between two
   | (pipes), ASM-One will substract $30.

VERSION
   V1.15

SEE ALSO
    ETEXT ,  DC
```

1.86 'TTL' Directive

```
NAME
   TTL - Set program title

SYNTAX
   TTL <string>

FUNCTION
   Sets the program title for a listing.

VERSION
```

```
   V1.01

SEE ALSO
    LIST
```

1.87 'XDEF' Directive

```
NAME
   XDEF - External Definition

SYNTAX
   XDEF <label>[,label....]

FUNCTION
   Will define <label> as an external value, so it can be
   used by other modules.

   This is mainly interesting for linking assembeler routines
   into code made by higher programming languages.

    You will need a linker (like BLink) to link the code.

EXAMPLE
   See  XREF

REMARK
   Sources that contain XDEF, are not executables. And therefore
   can NOT be executed !!

   You will need the  WO  to write the
   assembled object file.

   XDEF has the same function as ENTRY, EXTRN and GLOBAL.

VERSION
   V1.01

SEE ALSO
    XREF ,  ENTRY ,  GLOBAL ,  EXTRN ,  WO
```

1.88 'XREF' Directive

```
NAME
   XREF - Define External Definition

SYNTAX
   XREF <label>[,label....]

FUNCTION
   Will tell ASM-One that <label> is defined outside
   this program. So that ASM-One ill keep on assembling.
   This is mainly interesting for linking assembeler routines
   into code made by higher programming languages.

   You will need a linker (like BLink) to link the code.

EXAMPLE

   Program 1:

            XDEF    ClearScreen
   ClearScreen:
            CLR.L   Screen
            RTS

   Screen:  DC.L    0


   Program 2:

            XREF    ClearScreen
   GoClear: JMP     ClearScreen

REMARK
   Sources that contain ENTRY, are not executables. And therefore
   can NOT be executed !!

   You will need the  WO  to write the
   assembled object file.

   ENTRY has the same function as XDEF, EXTRN and GLOBAL.

VERSION
   V1.01

SEE ALSO
    XDEF ,  ENTRY ,  GLOBAL ,  EXTRN ,  WO
```