

MA-ST6

ST6 Macro Assembler



Reference manual

Revision 01/2000

RAISONANCE

SOFTWARE LICENSE FOR MA-ST6

RAISONANCE grants to the CUSTOMER a license to use the MA-ST6 software (herein referred to as the SOFTWARE) subject to the following provisions:

- The SOFTWARE can only be used on one computer. The CUSTOMER may use the SOFTWARE on as many computers as purchased licenses. The CUSTOMER may use the SOFTWARE on a multi-user or network system if one copy of the software is purchased for each node or terminal on which the SOFTWARE is to be simultaneously used.

- The SOFTWARE is copyrighted by and shall remain the property of RAISONANCE. The CUSTOMER is permitted to make a unique copy of the SOFTWARE for backup purposes only.

REFERENCE MANUAL: No part of this document may be copied or reproduced in any form or by any means without the prior written consent of RAISONANCE S.A. except for personal use by the CUSTOMER. The information contained herein is subject to change. RAISONANCE S.A. assumes no responsibility for the information contained in this document.

Copyright © RAISONANCE S.A. 2000

Contents

1. INTRODUCTION **5**

1.1	MA-ST6 versus AST6 Syntax	6
1.2	About this manual	7
1.2.1	MANUAL ORGANIZATION	7
1.2.2	CONVENTIONS USED IN THIS MANUAL	8

2. OVERVIEW OF AN ASSEMBLY PROGRAM **9**

2.1	Assembly statements	10
2.2	Comments	11
2.3	Keywords	11
2.4	Symbols	12
2.5	Labels	13
2.6	Operands	14
2.6.1	NUMERIC, CHARACTER AND STRING OPERANDS	14
2.6.2	SEGMENT POINTER	15
2.6.3	MAIN REGISTERS	15
2.7	Operators	16
2.7.1	BINARY OPERATORS	16
2.7.2	UNARY OPERATORS	17
2.7.3	OPERATOR PRECEDENCE	18
2.8	Expressions	19
2.9	Memory areas	20

3. PROGRAM BASICS **21**

3.1	Segment	23
3.1.1	DEFINITION	23
3.1.2	RELOCATABLE SEGMENT	23
3.1.3	ABSOLUTE SEGMENT DEFINITION AND SELECTION	27
3.1.4	SEGMENT LOCATION	29
3.2	Memory	30
3.2.1	SYMBOL DEFINITION	30
3.2.2	BANK SWITCHING MODE	34
3.2.3	MEMORY RESERVATION	37
3.2.4	MEMORY INITIALIZATION	39
3.2.5	REGISTER, BIT OF REGISTER OR NUMERICAL VALUE ASSIGNMENT	40
3.3	Object file	43
3.3.1	OBJECT FILE GENERATION	43
3.3.2	OBJECT FILE NAME	44
3.3.3	DEBUGGING INFORMATION	44

3.3.4	LINE NUMBER	45
3.3.5	DBL	46
3.3.6	ASCII	46
3.3.7	ASCIZ	46

4. ADVANCED ASSEMBLY FEATURES **48**

4.1	Source file	49
4.1.1	DEFINITION	49
4.1.2	FILE INCLUSION	49
4.1.3	CONDITIONAL ASSEMBLY	50
4.2	Linking directives	53
4.2.1	DEFINITION	53
4.2.2	SYMBOLS USED IN SEVERAL MODULES.	53
4.2.3	BRIDGE GENERATION	53
4.3	Listing file	56
4.3.1	DEFINITION	56
4.3.2	LISTING FILE GENERATION	57
4.3.3	LISTING FILE TITLE	58
4.3.4	NUMBER OF CHARACTERS IN A LISTING FILE LINE	58
4.3.5	NUMBER OF LINE IN A LISTING FILE PAGE	59
4.3.6	UNASSEMBLED PARTS OF A CONDITIONAL BLOCK	60
4.3.7	DEFINITION OF THE RELOCATION PAGE	61
4.3.8	DATE	61
4.3.9	ERROR MESSAGES	62
4.3.10	MACRO ASSEMBLY INSTRUCTION	63
4.3.11	SOURCE FILE INCLUSION	64
4.3.12	FORM FEED	65
4.3.13	SYMBOL TABLE	66
4.3.14	CROSS REFERENCE TABLE	67
4.4	Macro processor	68
4.4.1	DEFINITION	68
4.4.2	REPETITION OF BLOCKS	69
4.4.3	MACRO OPERATORS	74

5. APPENDICES **79**

5.1	Keywords	80
5.2	Error Messages	82
5.2.1	ERROR TYPES	86
5.2.2	LIST OF ERRORS	86
5.3	Revision History	98

6. INDEX **99**

7. TABLES INDEX **101**

8. BIBLIOGRAPHY **103**

1. Introduction

1.1 MA-ST6 versus AST6 Syntax

1.2 Concerning this manual

The MA-ST6 assembler translates ST6 assembly mnemonics into machine code and generates both a listing file and a relocatable object file that can be combined, using the RL-ST6 linker, with others object files generated by either the MA-ST6 assembler or the C compiler RC-ST6.

The MA-ST6 assembler is one of the components of any of the Raisonance AKit-ST6 or RKit-ST6 packages. It can be used either as a “command-line” translator, or from RIDE, a powerful Integrated Development Environment.

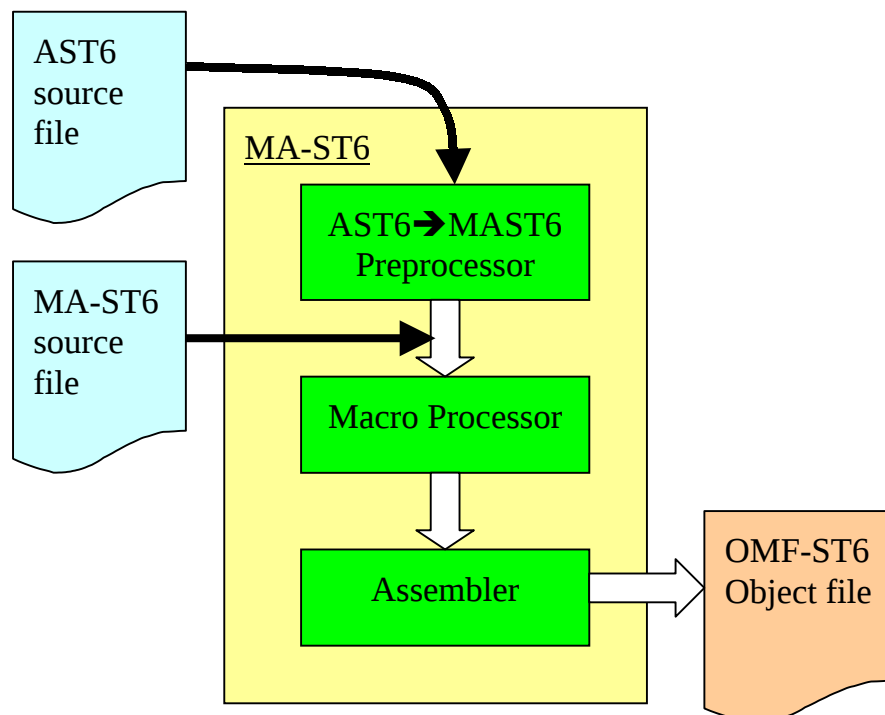
1.1 MA-ST6 versus AST6 Syntax

The Raisonance Macro Assembler MA-ST6 handles two different syntax:

MA-ST6 MA-ST6 is a Raisonance Macro-Assembler dedicated to the ST6 microcontroller family. The assembler directives that are described in this manual have been specified by Raisonance.

AST6 AST6 is a ST-Microelectronics proprietary tool. It’s an element of the ST6-SW package and the present manual **does NOT describe** the syntax of the AST6 Macro-Assembler.

Nevertheless, MA-ST6 is source-compatible with the ST-Microelectronics AST6 macro assembler. Indeed, the **AST6 source files are accepted** thanks to a built-in preprocessor that translates AST6 syntax into MAST6 syntax. Because the ST6 mnemonics are identical, this preprocessing only concerns the directives.



1.2 About this manual

The Raisonance Macro Assembler MA-ST6 Manual provides you with a syntactic description of MA-ST6 version 1.0.

1.2.1 Manual organization

This manual has been made to be as easy to use as possible and is intended for both beginners and experienced programmers. This manual provides a general description of the ST6 microcontroller as well as a detailed description of all directives, instructions and macro syntax.

- Chapter 2** Describes the different components of an assembly program: directives, instructions and symbols, labels, operands, operators.
- Chapter 3** Describes basic assembler features: segment declaration, program location, variable declaration, executable instructions and object file generation.
- Chapter 4** Describes advanced assembler features: conditional assembly, hardware specification, linking directives, macro processor.
- Appendix** Includes program examples and a list of keywords.

1.2.2 Conventions used in this manual

1.2.2.1 Syntax

Each keyword syntax is presented as in the example below:

Syntax: `[$]SET(num [,num...])`

[] optional arguments in command line and option fields are surrounded by square brackets.

\$, SET bold capital text is used for directives and instructions keywords.

num plain text is used to specify which information has to be provided by the programmer.

... dots indicate that the preceding items may be repeated zero or more times.

1.2.2.2 Examples

Examples always follow the rules below:

```
Example_number:
    ; comment concerning the example.

$PRAGMA
Labels:
    INSTRUCTIONS
    ; comments
```

1.2.2.3 Instructions

All instructions are presented in the same way:

MNEMONIC target_operand , source_operand

2. Overview of an assembly program

2.1 *Assembly statements*

2.2 *Comments*

2.3 *Keywords*

2.4 *Symbols*

2.5 *Labels*

2.6 *Operands*

2.7 *Operators*

2.8 *Expressions*

2.9 *Memory areas*

2.1 Assembly statements

There are 2 types of assembly statements:

- Directives: are used to control the way the assembler will process assembly language instructions and will generate the object file and the listing file
- Instructions: specify the source code to assemble.

GLOBAL is a general directive available for all the Raisonance tools. It allows to use a directive whatever the toolchain is. If the used tool implements the specified directive it will take it into account, if the used tool does not implement the directive, it will ignore it. It allows to keep the same code for different tools.

Syntax: global(directive)

2.2 Comments

Comments are used to make a program more readable. They are ignored by the assembler.

Syntax: ; text

Example:

```
Buffer:    DS    3    ;The comment  
            ;begins after the semi-colon.
```

2.3 Keywords

Keywords are symbols used by the assembler that must not be redefined. A full list of all keywords is given in the appendix.

2.4 Symbols

Symbols can be defined before assembly instructions, but not before directives (\$INCLUDE, etc.). In the variables segments (data, code), symbols can be assigned to any address.

A symbol is equivalent to the numerical value of the address at which it is allocated:

```
Example:
CSEG  AT  100H
      ; selection of an absolute code segment
MESSAGE: DB  'HELLO', 0
LABEL:  LDI  A, #LOW(MESSAGE)
```

A declaration cannot begin with a numerical character. Only alphabetical characters (upper or lower case), numerical characters (except in the first position), or the dollar sign (\$), underline (_) and question mark (?) characters may be used in symbol names.

-
- Note:**
1. The symbol length is unlimited.
 2. A symbol cannot have the same name as a keyword.
 3. Declarations are visible throughout the whole program, including INCLUDE files.
 4. The \$ symbol represents the current address in the active segment, which is located at the first byte of the line.
-

```
Example:
cu_ad:  JRC $
      ;branches at the current address (which is
      ;cu_ad in this example) while Carry is 1
```

2.5 Labels

A label is a symbol assigned to an address in an assembly program. It may refer to program code, to variable space in data memory, to constant data stored in the program or to code space.

Syntax: **label_name:**

Example:

```
Init_start:      DB '0', '1'      ; beginning of the initialization block
Init_end:        JP  Start_prog   ; end of the initializing block
                  ; jump to start of the program
...
Start_prog:
```

2.6 Operands

Operands are arguments of instructions or directives. They may be divided into 3 groups:

- Numeric, character and string operands
- Address pointer
- Main registers

2.6.1 Numeric, character and string operands

Numerical operands must be followed by the base in which they are specified. Base types are hexadecimal, decimal, octal or binary, with H (or h), D (or d), O (or o or Q or q), B (or b) as respective suffixes. When a numerical value is specified, it must begin by a 0. If no base type is specified, by default the numerical operand is considered as decimal (d).

Example1:

Hex:	DB	00fh	; hexadecimal operand
Dec:	DB	003d	; decimal operand
<i>Dec</i>	<i>DB</i>	<i>123</i>	<i>; decimal operand</i>
Oct:	DB	056o	; octal operand
Bin:	DB	010b	; binary operand

(See section 3.2.4.1 for the definition of DB.)

Character operands may be composed of one or two ASCII characters between two quotes and may be used wherever a numerical operand is valid.

Example2:

```
Ch_A EQU 'AB'
```

(See section 3.2.5.2 for the definition of EQU.)

String operands are composed of more than one character between single quotes and must be used with the DB directive.

Example3:

```
Message: DB 'This is a message offered by Raisonance'
```

2.6.2 Segment pointer

Each segment is controlled by a segment pointer that contains the offset of the instruction or data being assembled, relative to the start of the current segment. The segment pointers are initialized to 000h unless the ORG directive is used. The value of the segment pointer of the current segment may be obtained by using the dollar sign character (\$).

Example:

```
RSEG  mycodseg
      ; supposes that mycodseg is a CODE segment
      Loc_count_cod EQU $
      ; Loc_count_cod symbol is assigned the mycodseg
      ; CODE segment pointer value (address)
DSEG  mydatseg
      ; absolute data segment specification
      Loc_count_dat EQU $
      ; Loc_count_dat symbol is assigned the mydatseg
      ; DATA segment address pointer value
      ; which is different from Loc_count_cod
```

(see section 3.2.5.2 for the definition of EQU)

2.6.3 Main registers

ST6 register names are pre-defined. They are listed below:

- A: general purpose accumulator
- V, W, X, Y: general purpose registers (8-bit registers)
- C, Z: carry and zero flags

2.7 Operators

Binary operators contain 2 operands whereas unary operators are used with a single operand. Operands are used to show the evaluation of an expression, e.g. in the evaluation of a constant. They are different from the instruction set of the ST6 microprocessor. A list of all the different operators is given in tables 1 to 7.

2.7.1 Binary operators

Binary operators are operators that require 2 operands.

2.7.1.1 Arithmetic operators

Operator	Syntax	Description
+	exp + exp	Addition
-	exp - exp	Subtraction
*	exp * exp	Multiplication
/	exp / exp	Integer division
MOD	exp MOD exp	Remainder

Table 1: binary arithmetic operators

2.7.1.2 Relational operators

Operator	Syntax	Description
GTE	exp1 GTE exp2	true if exp1 is greater than or equal to exp2
>=	exp1 >= exp2	
GT	exp1 GT exp2	true if exp1 is greater than exp2
>	exp1 > exp2	
LTE	exp1 LTE exp2	true if exp1 is less than or equal to exp2
<=	exp1 <= exp2	
LT	exp1 LT exp2	true if exp1 is less than exp2
<	exp1 < exp2	
EQ	exp1 EQ exp2	true if exp1 is equal to exp2
=	exp1 = exp2	
NE	exp1 NE exp2	true if exp1 is not equal to exp2

Table 2: relational operators

2.7.1.3 Shifting operators

Operator	Syntax	Description
SHR	exp SHR num	shifts num times exp towards the right
SHL	exp SHL num	shifts num times exp towards the left

Table 3: shifting operators

2.7.1.4 Binary operands operators

Operator	Syntax	Description
AND	exp1 AND exp2	true if exp1 and exp2 are both true
OR	exp1 OR exp2	true if one or both expressions, exp1 and exp2, are true

Table 4: binary operand operators

2.7.2 Unary operators

Unary operators are operators that require only one operand.

2.7.2.1 Arithmetic operators

Operator	Syntax	Description
+	+ exp	returns exp
-	- exp	returns exp opposite

Table 5: unary arithmetic operators

2.7.2.2 Miscellaneous operators

Operator	Syntax	Description
NOT	NOT exp	bit-wise complement
LOW	LOW exp	low-order byte of 16-bit expression
HIGH	HIGH exp	high-order byte of 16-bit expression
WINDOW	#WINDOW (expr)	High-order 6 bits of a 12-bit address, to be loaded into DRWR.
WINOFFSET	#WINOFFSET (expr)	Low-order 6 bits of a 12-bit address, to access the DATAROM Window. This operator takes into account the 0x40 offset
PAGE	#PAGE(expr)	Defines the ROM page for a symbol located either in banked ROM or RAM area. To be loaded in DRBR

Table 6: miscellaneous unary operators

2.7.3 Operator precedence

Expression may include a great number of operators. The way in which an expression containing more than one operator is evaluated depends on the operator precedence.

The following table lists all operators with their respective priorities.

Operator	Priority
()	1 (first operator to be evaluated)
HIGH, LOW, NOT, WINDOW, DATA, PAGE	2
+ (unary), - (unary)	3
*, -, MOD	4
+, -	5
SHR, SHL	6
AND, OR, XOR	7
GTE, >=, LTE, <=, GT, >, LT, <, EQ, =, NE	8 (last operator to be evaluated)

Table 7: operator precedence

Note: it is highly recommended that you use parenthesis with complex expressions (see example).

Example:

```

IF( X=1    AND Y=2) ...
IF( (X=1) AND (Y=2)) ...
    ; these 2 expressions are not processed in the
    ; same way because the first one is the same as
IF( (X= (1 AND Y)) =2)

```

2.8 Expressions

An expression is a combination of operands and operators that is evaluated by the assembler. It may become complex if symbols, used in an expression come from different segments.

Expressions containing a relocatable or an external symbol are called relocatable expressions.

2.9 Memory areas

The ST6 processor recognizes 3 different memory segments.

<u>CODE</u>	ROM of up to 4 Kbytes of read-only executable code or constants. With a technique of banking, this segment may be extended up to 8Kb.
<u>DATA</u>	RAM of up to 64 bytes addressable directly or indirectly. With a technique of banking, this segment may be extended up to 512 bytes.
<u>DATAROM</u>	<i>Equivalent to “CODE INWINDOW”</i>
<u>EEPROM</u>	EEPROM of up to 128 bytes

3. Program basics

3.1 *Segment*

3.2 *Memory*

3.3 *Object file*

This section shows how to create an assembler program:

- defining all segments, relocatable or absolute, data or code,
- defining and initializing symbols,
- writing the program heart,
- generating the object file.

All other directives are secondary and will be dealt with in section IV (Advanced assembler features).

3.1 Segment

3.1.1 Definition

A segment can be placed either in relocatable mode, or in absolute mode (if any address has been specified). This can be done regardless of the output format.

At each instruction, the current segment address will be incremented by 2 or 4 depending on the instruction being assembled.

Overlapping of absolute code segments generates an error.

Overlapping of absolute data segments is allowed but the assembler will generate a warning.

The following sections describe the directives that must be used to declare a relocatable segment, to select a relocatable segment or to select an absolute segment.

3.1.2 Relocatable segment

All relocatable segments must be declared by the **SEGMENT** directive. The name of the segment is specified by the symbol name immediately preceding **SEGMENT** directive.

Its type is specified immediately after **SEGMENT** directive.

The following types are allowed: **CODE** and **DATA**.

Syntax: SegmentName **SEGMENT** SegmentType [PAGE num] [AT
addr] [OVERLAYABLE] [NODUPLICATION]
[RelocationType]

SegmentName: is the identifier of the relocatable segment declared.

SegmentType: must be chosen among the following values

CODE characterizes a relocatable segment in code memory space.

DATA characterizes a relocatable segment in data memory

DATAROM characterizes a relocatable segment in code memory space

EEPROM Characterizes a relocatable segment in the EEPROM

DATAROM is like a **CODE** segment with a specific relocation label **INWINDOW** (relocation in a 64 –byte window). The following two definitions are equivalent:

MySegment **SEGMENT** **CODE** **INWINDOW**

MySegment **SEGMENT** **DATAROM**

PAGE num: (optional parameter)

specifies that the PAGE num is used. 'num' may be replaced by STATIC or AUTO which are keywords. The default is PAGE AUTO (which means that the linker chooses the PAGE) except if a directive (PRPAGE, DTPAGE or EEPAGE) has been defined (see section 4.3.8).

STATIC corresponds to the PAGE 1 for the code and to the segment 84h-C0h for the data.

'PAGE num' needs the PROGPAGING directive. (see section 3.2.2.1)

'PAGE AUTO' does not need the PROGPAGING directive for CODE segments but needs the DATAPAGING directive for DATA segments.

'num' can take different values according to the **SegmentType**:

SegmentType	value
CODE	0, 2, 3, AUTO, STATIC
DATA	1, 2, AUTO, STATIC
DATAROM	0, 2, 3, AUTO, STATIC
EEPROM	0, 1, AUTO

AT addr (optional)

specifies an absolute segment starting at address. addr. AT and PAGE can be specified together, but they must match.

If num = AUTO, [AT addr] is forbidden

If num = STATIC, [AT addr] can be used
 for CODE: x800 -> FEF
 for DATA: x80 -> BF

For CODE or DATAROM If num = 0, 2 or 3, [AT addr] can be used: 0 -> 7FF

For DATA if num = 1 or 2, [AT addr] can be used: 0 -> 3F

For EEPROM if num = 0 or 1, [AT addr] can be used: 0 -> 3F

OVERLAYABLE (optional parameter):

Only available for DATA segment

NODUPLICATION (optional parameter):

Only available for CODE segment

RelocationType (optional parameter, default is UNIT):

INWINDOW	insertion in a 64-byte block.
INBLOCK	insertion in a 256-byte block.
INPAGE	insertion in a 2048-byte block

Note: When a segment is declared without the 'AT addr' relocation type it can be selected with the RSEG directive.

When declared with the 'AT addr' relocation type, a segment may be selected with the ASEG directive.

The current segment remains selected until a new segment is found.

Syntax: **RSEG** SegmentName
 ASEG SegmentName

Example: ;

```

CSEG AT 0FFEH
    JP      MyProgramEntry
                ; this is an absolute vector at address 0 which is
                ; the reset vector. Reset vector points to
                ; MyProgramEntry label
mycodseg      SEGMENT CODE
                ; this is a relocatable code segment
mydat1seg     SEGMENT DATA
                ; mydat1seg relocatable DATA segment declaration
mydat2seg     SEGMENT DATA
                ; SEGMENT statement allows several segment
                ; declaration even with the same type

RSEG         mycodseg
                ; mycodseg relocatable CODE segment selection
MyProgramEntry:
    LDI      A, #55H
    LD       X, A
                ; instructions of the mycodseg CODE segment
RSEG         mydat1seg
                ; mydat1seg DATA segment selection
    COUNTER:  DB  1
                ; memory reservation
RSEG         mycodseg
                ; mycodseg CODE segment may be reselected
                ; the mycodseg address pointer is automatically
                ; increased depending on the instruction length

```

<pre>NOP RSEG mydat2seg ; mydat2seg DATA segment selection</pre>
--

3.1.3 Absolute segment definition and selection

3.1.3.1.1 Selecting a code segment

The CSEG directive is used to select an absolute segment within the CODE address space. The segment starting address is specified using keyword AT.

Syntax: **CSEG [AT ad] PAGE num**

If **PAGE** = 0, 2 or 3, ad goes from 0 to 7FF

If **PAGE** = STATIC, ad goes from 800 to FEF

If ad > FF0 (to FFF), the **PAGE** directive is forbidden because those addresses correspond on the area of vectors

```

Example:
mycodseg    SEGMENT    CODE
              ; CODE relocatable segment declaration
CSEG  AT    780h    PAGE 0
              ; absolute CODE segment declaration
              ; its starting address is 800h
      JP      700H
              ; instruction length: 2 bytes
RSEG        mycodseg
              ; mycodseg CODE segment selection
      NOP
      NOP
CSEG
              ; absolute segment declaration
              ; since no address is specified, address is 782h
      ...

```

3.1.3.1.2 Selecting a data segment

The DSEG directive is used to select an absolute segment within the DATA address space. The segment starting address is specified using the keyword AT.

If no starting address is specified, the segment starting address is either the physical segment starting address, or the next address after the last address used in a segment of the same. The segment name does not have to be specified and is generated implicitly.

Syntax: **DSEG [AT ad]**

If **PAGE** = 1 or 2, ad goes from 0 to 3F

If **PAGE** = STATIC, ad goes from 80 to 8F

```

Example:
DSEG  AT      90h    PAGE STATIC
              ; absolute DATA segment selection
              ; DSEG has the same syntax as CSEG

```

3.1.3.1.3 Selecting an eeprom segment

The ESEG directive is used to select an absolute segment within the EEPROM address space. The segment starting address is specified using the keyword AT.

If no starting address is specified, the segment starting address is either the physical segment starting address, or the next address after the last address used in a segment of the same. The segment name does not have to be specified and is generated implicitly.

Syntax: **ESEG [AT ad]**

If **PAGE** = 0 or 1, ad goes from 0 to 3F

3.1.3.1.4 Selecting a datarom segment

The DRSEG directive is used to select an absolute segment within the DATAROM address space. The segment starting address is specified using keyword AT.

Syntax: **DRSEG [AT ad] PAGE num**

If **PAGE** = 0, 2 or 3, ad goes from 0 to 7FF

If **PAGE** = STATIC, ad goes from 800 to FEF

3.1.4 Segment location

The ORG is used to specify the offset for subsequent code or data. The address assigned after an ORG directive must be specified by an absolute or simple relocatable expression without forward references.

An ORG statement is often used to define the program start address.

Syntax: ORG exp

-
- Note:**
1. The ORG directive does not create a new segment; it sets the segment pointer to the value of exp.
 2. When used in a relocatable segment, the segment pointer is incremented.
-

The END directive specifies the end of the current module.

Syntax: END

Note: any text following an END directive is ignored by assembler.

Example:		
DSEG	AT	90h
		; absolute DATA segment declaration
ORG	97h	
		; data address pointer is now 97h
		; space between 90h and 97h is empty
message:	DS	8
		; reserves 8 bytes from 97h to 9Fh
ORG	0A0h	
		; data address pointer is now 0A0h
table:	DS	10
		; reserves 10 bytes from A0h to AAh
		; be careful not to rewrite on message

(See section 3.2.3.2 for the definition of DS.)

3.2 Memory

When developing an assembly program, it will usually be necessary to:

- reserve memory spaces,
- initialize memory spaces when they are in CODE space,
- assign symbols to specific addresses, registers or values.

3.2.1 Symbol definition

3.2.1.1 Defining a symbol referring to a program address

The CODE directive associates a program address to the specified symbol.

The name of the symbol to be defined is specified by the string immediately preceding CODE directive.

The program address is set to the value Address.

When the PROGAGING directive is set, the authorized addresses for the CODE directive are chosen from 0000 to FFFF. When the NOPROGAGING directive is set, the authorized addresses are chosen from 000 to 0FFF. This corresponds to the pages 0, 1, 2, 3 and 32

Syntax: SymbName **CODE** Address

Example:

```
int0    CODE    0FF2h
          ; defines int0 as being address 0FF2h
int1    CODE    int0 + 2h
          ; relative definition may be processed
int2    CODE    int1 + 2h
CSEG    AT      600h
          ; absolute CODE segment declaration
JA      int0
          ; goto 0FF2h
```

Note: SymbName cannot be redefined or changed.

3.2.1.2 Defining a symbol referring to a data address

The DATA directive associates a program to a specified symbol.

The name of the symbol to be defined is specified by the string immediately preceding the DATA directive.

The address is set to the value Address.

When the DATAPAGING directive is set, the authorized addresses for the DATA directive are chosen from 0000 to 003F and from 080 to 0FF and for each of the following pages from nn00 to nn 3F (nn is the page number). When the NODATAPAGING directive is set, the authorized addresses are chosen from 0000 to 003F and from 080F to 0FF which corresponds to the page 0. The number of the active page is fixed by the DATAPAGENUMBER directive.

Syntax: SymbName **DATA** Address

Example:

```
buff_st     DATA   23h
            ; buff_st symbol now represents address 23h
buff_en     DATA   buf_st + 10
            ; relative assignment may be processed
```

Note: SymbName cannot be redefined or changed.

3.2.1.3 Defining a symbol referring to a data address

The DATAROM directive associates a program to a specified symbol.

The name of the symbol to be defined is specified by the string immediately preceding the DATAROM directive.

3.2.1.4 Defining a symbol referring to a eeprom address

The EEPROM directive associates a program to a specified symbol.

The name of the symbol to be defined is specified by the string immediately preceding the EEPROM directive.

Syntax: SymbName **EEPROM** Address

3.2.1.5 Defining a 32-bit long constant data

NUMBER directive allows 32-bit constant data values to be defined.

Syntax: SymbolName **NUMBER** expression

Example:

```
CSEG AT 234h
          ; absolute CODE segment selection
lab1: DB 1
NUM_TSKS  NUMBER 12
SWP_NUM1  NUMBER (lab1>>8)+(lab1<<8)
```

Note: The NUMBER directive may be used with PUBLIC and EXTERN directives. The Symbol_name can be used in different segments, as the reference is resolved at the linking stage.

3.2.2 Bank Switching Mode

3.2.2.1 Authorization of the Bank Switching Mode for the Code space

The PROGPAGING directive enables the use of Bank Switching on the Code space.

Syntax: **\$PROGPAGING**

The NOPROGPAGING directive disables the use of bank switching on the Code space.

When the NOPROGPAGING is set, the PAGE and the DATA operators are forbidden in the code space and the definition of PAGE is not authorized on code segment.

Syntax: **\$NOPROGPAGING**

Note: the default value is \$NOPROGPAGING

3.2.2.2 Authorization of the Bank Switching Mode for the Data space

The DATAPAGING directive enables the use of bank switching on the Data space.

Syntax: **\$DATAPAGING**

The NODATAPAGING directive disables the use of bank switching on the Data space.

When the NOPROGPAGING is set, the PAGE operator is forbidden for the data space and the definition of PAGE is not authorized on the data segment.

Syntax: **\$NODATAPAGING**

Note: the default value is NODATAPAGING

3.2.2.3 Definition of the active page

The DATAPAGENUMBER directive defines the active page for the variable definition of the DATA space when the DATAPAGING directive is set. It is available when '*symp DATA addr*' is used with *addr* between 0 and 3F

Syntax: **\$DATAPAGENUMBER (val)**

val must be either 1 or 2.

The EEPROMPAGENUMBER directive defines the active page for the variable definition of the EEPROM space when '*symp EEPROM addr*' is used with *addr* between 0 and 3F.

Syntax: **\$EEPROMPAGENUMBER (val)**

val must be either 0 or 1.

3.2.2.4 Definition of the mask

The DRBRMask directive defines the mask value for the DRBR in the case of Data bank switching.

Syntax: \$DRBRMask(hexval)

3.2.2.5 Definition of the DRBR address

The DRBRAddr directive defines the address of the DRBR register, in the case of Data bank switching.

Syntax: \$DRBRAddr(hexval)

3.2.2.6 Definition of the microcontroller

The VERS directive allows to define version the microcontroller in use. It needs as parameter a string defining the microcontroller. This automatically sets the values for DRBRMask and DRBRAddr if they are enabled.

Syntax: \$VERS(string)

3.2.3 Memory reservation

At any time within a segment, you may reserve some space without initializing it. Space may be reserved in DATA space.

3.2.3.1 Size of the ROM

The ROMSIZE directive defines the size of the ROM available for the microcontroller used.

Syntax: **\$ROMSIZE(val)**

By default, val is 2 and must be chosen from 2, 4, 8 and 16. It defines the size of the ROM in Kbytes.

3.2.3.2 Data space

DS, DSB, DSW and DSD directive are used to reserve space in the data segment.

The address of the reserved memory is referred to by a label preceding the DS directive.

The number of bytes to reserve is specified by a number following the DS directive which cannot contain forward references, relocatable symbols or external symbols.

Syntax:

```
[lab] DS NumberOfBytes
[lab] DSB NumberOfBytes
[lab] DSW NumberOfWords
[lab] DSD NumberOfDoubleWords
```

Example:

```
DSEG      AT      90h
           ; absolute DATA segment selection
Table1:    DS      6
           ; reserves 6 bytes for Table, which are
           ; located at addresses 90h to 95h
Table2:    DSB     6
           ; reserves 6 bytes for Table, which are
           ; located at addresses 96h to 9Bh
Buff1:     DSW     2
           ; reserves 2 words for Buff1
           ; which are located from 9Ch to 9Fh
Buff2:     DSD     1
           ; reserves 1 double-word for Buff2
           ; which are located from 0A0h to 0A3h
CSEG
           ; absolute CODE segment selection
LDI        A, #Table1
LD         X, A
LDI        A, #0FFH
LD         (X), A
           ; first byte of Table1 initialization
```

Note: The Space reservation is done in the current segment, at the current address. The current address is increased by the value of the expression.

3.2.4 Memory initialization

Memory initialization may be done either at the start address or within the program.

One difference between code space initialization and data space initialization is that in code space you can use the DB and DW directives for initializing byte and word memory spaces.

3.2.4.1 Byte values

The DB directive is used to initialize CODE memory with byte values.

The address of the initialized memory is referred to by a label preceding the DB directive.

A symbol, string or an expression can be used to initialize the memory.

Syntax: [lab] **DB** exp [, exp ...]

Example:

```
CSEG      AT    100h
           ; absolute CODE segment declaration
list:     DB    0,1,2,3,'Raisonance',Low(list)
           ; memory content      100h  -> 0
           ;                    101h  -> 1
           ;                    102h  -> 2
           ;                    103h  -> 3
           ;                    104h  -> R
           ;                    105h  -> a ...
```

3.2.4.1.1 Word values

DW directive is used to initialize CODE memory with word values, using the BIG ENDIAN storage convention.

The address of the initialized memory is referred to by the label preceding the DW directive.

A symbol , a string or an expression can be used to initialize the memory.

Syntax: [lab] **DW** exp [, exp ...]

Example:

```
CSEG      AT    30h
Buffer:   DW    24h,25h
Timer:    DW    $
           ; memory content      30h  -> 00
           ;                    31h  -> 24
           ;                    32h  -> 00
           ;                    33h  -> 25
```

3.2.5 Register, bit of register or numerical value assignment

Register, register bit or numerical value assignment may be performed using either the SET or the EQU directives. Unlike EQU, the SET directive allows the assigned symbol to be reassigned.

3.2.5.1 Assignment or re-assignment

SET can be used in two different ways:

1) The SET directive is used to assign a symbol name to: a numerical value, a register or a register bit.

- The symbol name is defined by the string preceding the SET directive.
- If a numerical value is assigned it must immediately follow the SET directive and must not contain a forward reference.
- If a register symbol is assigned it must be chosen from X,Y,V and W.

Syntax: symb **SET** exp
 symb **SET** reg

Example1:

```
RegX      SET    X
timer     SET    V
ref       SET    20
counter   SET    ref + 2
CSEG
LDI       A, #counter
LD        RegX, A
```

Note: 1. Each occurrence of a defined symbol is replaced in the assembly program by the corresponding numerical value or register symbol.

2. symb can be changed by another SET statement.

2) The SET directive assigns values to symbols.

- These symbols are only used in \$IF \$ELIF directive statements for conditional assembly.
- Symbols declared with the SET directive do not interfere with CODE and DATA symbols.

Syntax: **\$SET(symbol[=number] [,symbol[=number]...])**

Note: The SET directive is preceded by a dollar sign (\$) only if used in the assembly file. The dollar sign is not necessary in the command line.

If no number is specified, the symbol is initialized with 0FFFFh.

Example2:

```
                ; command line
MAST6 raison.a SET(type=TT)

                ; content of raison.a
$IF(type=TT)
    bw: DB    16
                ; in this case bw is assigned 16
$ELSE
    bw: DB    8
$ENDIF
```

The RESET directive performs the same initialization as the SET directive with 0 as number.

Syntax: **RESET(symbol)**

Example3:

```
$RESET(sb)
                ;sb is initialized with 0
                ;the same initialization could have been done with
$SET(sb=0)
```

3.2.5.2 Absolute assignment

The EQU directive is used to assign a symbol name to: a numerical value, a register bit or a register symbol.

The symbol name is defined by the string immediately preceding the EQU directive.

If a numerical value is assigned it must immediately follow the EQU directive and must not contain forward reference.

If a register symbol is assigned it must be chosen from X,Y,V and W.

Syntax: symb EQU exp
 symb EQU reg

Example1:			
nb_elem	EQU	130	
elem_size	EQU	3	
buff_ptr	EQU	W	
buu_size	EQU	nb_elem * elem_size	

Note: 1. Each occurrence of the defined symbol is replaced in the assembly program by the specified numerical value or register symbol.

2. symb cannot be redefined or changed.

3.3 Object file

Object directives control the object file generation, its name, debugging information presence and line number definitions.

3.3.1 Object file generation

The OBJECT directive specifies the object file which will be generated.

The default name for the object file is the source file name with the OBJ extension.

Syntax: **OBJECT(object_file_name.obj)**

Abbreviation: OJ

Example 1: MAST6 raison.a OBJECT(raison.obj)
--

Example 2: \$OBJECT(raison.obj)

The NOOBJECT directive prevents the object file from being generated.

Syntax: **NOOBJECT**

Abbreviation: NOOJ

Example 3: MAST6 raison.a NOOJ
--

3.3.2 Object file name

When assembling the current module an object file is generated. Its name is defined by the string following the NAME directive.

Syntax: **NAME** str

Example: NAME rais_mod

Note: By default if the object module name is not specified the object file name will be the same as the source file name.

3.3.3 Debugging information

The DEBUG directive controls the inclusion of debugging information in the object file.

Syntax: **DEBUG**

Abbreviation: DB

Example 1: MAST6 raison.a DEBUG

Example 2: \$DB

The NODEBUG directive prevents the debugging information from appearing in the object file.

Syntax: **NODEBUG**

Abbreviation: NODB

Example 3: MAST6 raison.a NODB
--

3.3.4 Line number

The LINE directive allows line numbers to be generated in an assembly source file, similar to those generated by compilers.

The text following the LINE directive is treated as a comment and does not need to be prefixed by a semi colon.

The line number generated will correspond to the absolute line number of the current file.

Syntax: **LINE** text_comment

-
- Note:**
1. The LINE directive generates a line number only if the current segment is a 'CODE' (either absolute or relocatable) segment.
 2. The LINE directive is ignored in DATA segments and in 'INCLUDE' files.
-

Some directives have been defined in order to accept the syntax of the AST6 ST assembler.

3.3.5 DBL

The DBL directive allows to reserve a segment in a space. For example, the following definition:

```
?PR?SEGCODE SEGMENT CODE
RSEG ?PR?SEGCODE
...
    DBL 64
...
```

reserves 64 bytes in the ?PR?SEGCODE segment.

This directive allows to the ends of the segments to be aligned.

3.3.6 ASCII

The ASCII directive allows an identifier to be associated to a character string **without** final zero.

For example the following declaration:

```
MyString1  ASCII "Hello"
```

has a size of 5 bytes. The 5 bytes are respectively the characters for which the ASCII codes correspond to the letters H, e, l, l and o.

Message: ASCII 'Hello' is equivalent to Message: db 'Hello' = db 'H','e','l','l','o'

3.3.7 ASCIZ

The ASCIZ directive allows an identifier to be associated to a character string **with** the final zero.

For example the following declaration:

```
MyString2  ASCIZ    "World"
```

has a size of 6 bytes. The 6 bytes are respectively the characters for which the ASCII codes correspond to the letters W, o, r, l and d plus the final zero.

Message: ASCIZ 'Hello' is equivalent to Message: db "Hello" = db 'H','e','l','l','o',0

4. Advanced assembly features

4.1 *Source file*

4.2 *Linking directives*

4.3 *Listing file*

4.4 *Macro processor*

4.1 Source file

4.1.1 Definition

In many cases it is not practical to write an entire assembly project as a single file. In this case it is necessary to control the project architecture. The source directive presented in this section allows you to use objects and / or functions declared in other files.

4.1.2 File inclusion

The INCLUDE directive allows an external file to be inserted into the current file from the current directory. It can be used several times and allows a program to be divided into several files.

The assembler will automatically look for the include file in the current directory. If the include file is in another directory, the file path must be specified (either a relative file path or an absolute file path).

The filename must be specified with either a relative path or the full file path.

The INCLUDE directive is recursive and INCLUDE files can also contain the INCLUDE directive.

The input or output of an INCLUDE file does not modify the active segment; the active segment is therefore not restored if a directive modifies the active segment.

Syntax: `[$]INCLUDE (FileName)`

where FileName is the name of the file to include.

Example:

```
$INCLUDE (c:\RIDE\INCST6\DEFS.INC)
```

4.1.3 Conditional assembly

Some sections of an assembly program can be assembled selectively by using the conditional assembly directives presented in this section: IF, ELSEIF, ELSE and ENDIF.

Note:

1. Those directives may be used with or without a dollar sign (\$).
2. When prefixed by a dollar sign, they only access symbols defined with \$SET or \$RESET directives.
3. When not prefixed by a dollar sign, they first access symbols defined with the SET and the EQU directives and then those defined with the \$SET and the \$RESET directives.

- The IF directive defines the start of a block which must terminate with an ENDIF directive. The block constructed with these 2 directives is assembled provided the expression following the IF directive is true. In the other case, it is not assembled.

Syntax: **IF** expression
 ...
 ENDIF

Example 1:

```
$IF (version>200)
    call INIT_SERIAL_PORT
        ; if version is greater than 200 function
        ; INIT_SERIAL_PORT will be called
$ENDIF
```

- The IF directive can also be used in other blocks constructed with the ELSE directive.

Syntax: **IF** expression
 block1
 ELSE
 block2
 ENDIF

If 'expression' is true, only block1 is assembled, otherwise only block2 will be assembled.

Example 2:

```
$IF (version>200)
    callr    INIT_SERIAL_PORT
            ; if version is greater than 200,
            ; function INIT_SERIAL_PORT will be called
$ELSE
    callr    INIT_PARALLEL_PORT
            ; function INIT_PARALLEL_PORT is called
            ; if version is less or equal than 200
$ENDIF
```

- The IF directive can be used in blocks constructed with the ELSEIF directive.

Syntax:

```
IF expression1
    block1
ELSIF expression2
    block2
ENDIF
```

If 'expression1' is true, only block1 will be assembled, otherwise only block2 will be assembled.

Note: The ELSEIF directive can be used any number of times within a block.

Example:

```
$IF (bdt==1)
    callr func1
            ; if bdt is equal to 1, func1 is called
$ELSEIF (bdt==2)
    callr func2
            ; if bdt is equal to 2, func2 is called
$ELSEIF (bdt==3)
    callr func3
            ; if bdt is equal to 3, func3 is called
$ENDIF
```

- The IF, ELSEIF, ELSE and ENDIF directives can be used in the same block.

Syntax:

```
IF expression1
    block1
ELSEIF expression2
    block2
ELSE
    block3
ENDIF
```

Example:

```
$IF (bdt==1)
    call func1
$ELSEIF (bdt==2)
    call func2
$ELSEIF (bdt==3)
    call func3
$ELSE
    call func4
        ; func4 is called only if bdt is different
        ; from 1, 2 or 3
$ENDIF
```

In the example above, callr func4 is assembled only if bdt is less than 1 or bdt greater than 3.

4.2 Linking directives

4.2.1 Definition

Developing complex projects usually requires the use of several files. It may be necessary to use symbols and functions declared in other files.

4.2.2 Symbols used in several modules.

The PUBLIC directive is used to specify which symbols declared in the current module may be used in other modules.

The symbols following the PUBLIC directive must be declared in the current module (forward references are allowed).

Syntax: **PUBLIC** symb [, symb ...]

Note: 1. Registers and segment symbols cannot be declared as public symbols.

2. The PUBLIC directive cannot be used in the command line.

The EXTERN directive is used to specify symbols that are used in the current module but are declared in other modules.

The segment type must be specified and must be one of the following types: CODE, DATA, DATAROM, EEPROM and NUMBER.

Syntax: **EXTERN** SegmentType (symb [, symb ...])

Note: The EXTERN directive cannot be used in the command line.

4.2.3 Bridge generation

The BRIDGEON directive is used to automatically generate the bridges in the linker.

Syntax: **\$BRIDGEON**

Example 1:

```

; module1
$TITLE Module1
; module1 title definition
PUBLIC Buffer
; Buffer is declared as a public symbol
; It may be used in another module in which it is
; declared as external
Buffer DATA 20h

; module2
$TITLE Module2
EXTERN DATA (Buffer)
; specifies that Buffer has been declared in
; another module

```

The implementation of the PUBLIC and EXTERN symbols may become tedious for large applications.

To avoid this, it is recommended that you:

- Group all the extern declarations in the same file, independently of the modules in which they are actually used.

Example 2:

```

;Declaration file EX_DCL.INC
EXTERN CODE (MESURE1, MESURE2, RESULT)
EXTERN CODE ....
EXTERN CODE ...
EXTERN DATA (NUMBER, COUNTER)

```

- Insert this file at the beginning of EACH file, using the INCLUDE directive.

Example 3:

```

; Module i
$INCLUDE(EX_DCL.INC)
NAME MAIN
; specifies which name to use for the
; object file
; Module j
$INCLUDE(EX_DCL.INC)
NAME SERIE
; Module k
$INCLUDE(EX_DCL.INC)
NAME ACQUISITION

```

Conditions to be satisfied for correct linking are listed below:

1. A symbol must NOT be declared with the PUBLIC attribute in two different modules.
2. Whenever a module refers to an external symbol (via a jump or call instruction) the corresponding symbol must have been declared with the PUBLIC attribute in another module.

If the same symbol is declared as being both EXTERN and PUBLIC within the same module, the PUBLIC attribute will be used.

Example 4:

```
EXTRN CODE (TRT_INT1)
                ; this declaration will be ignored
?PR?SEG SEGMENT CODE
RSEG ?PR?SEG
PUBLIC TRT_INT1
                ; the PUBLIC attribute will be used
```

This property allows the 'EXTERN' declarations to be grouped in a single file and to be included at the beginning of each module.

External symbols NOT used in a module are ignored by the assembler.

4.3 Listing file

4.3.1 Definition

A file listing information concerning the assembly operation may be generated by using the PRINT directive.

This file listing may include some information such as the unassembled parts of the source program, error messages, current date, the program source listing, etc.

These information are controlled by the listing directives presented in the following section:

TITLE, PAGESWIDTH, PAGESLENGTH, COND, NOCOND, DATE, EJECT, ERRORPRINT, NOERRORPRINT, GEN, NOGEN, LIST, NOLIST, SYMBOLS, NOSYMBOLS, XREF, NOXREF.

4.3.2 Listing file generation

A listing file including indication on the assembly operation may be generated with the PRINT directive.

The listing file name is specified by the string following the PRINT directive.

If no file name is specified, the listing file name will be the source file name with .lst extension.

Syntax: **[\$]PRINT (file_name.lst)**

[\$]PRINT

Abbreviation: **[\$]PR**

Example 1: MAST6 raison.a PRINT

Example 2: MAST6 raison.a PRINT(listing.lst)
--

Example 3: \$PRINT

The NOPRINT directive prevents the assembler from generating a listing file.

Syntax: **[\$]NOPRINT**

Abbreviation: **[\$]NOPR**

Example 4: \$NOPR

Example 5: MAST6 raison.a NOPRINT

Note: \$PRINT (resp. \$PR, \$NOPRINT and \$NOPR) is used when specified in the assembly file whereas PRINT (resp. PR, NOPRINT and NOPR) is used in the command line. The default is \$PRINT.

4.3.3 Listing file title

You may include a title in the listing file by using the TITLE directive. The title name is specified by the string immediately following this directive.

Syntax: **TITLE**(text)

Abbreviation: **TT**

Example: \$TITLE(Raisonance benchmark)
--

Note: TITLE cannot be used in the command line.

4.3.4 Number of characters in a listing file line

The maximum number of characters in a listing file line may be controlled by the PAGEWIDTH directive. Lines that are longer than the specified value automatically wrap around the next line. The default is 120.

Syntax: **[\$]PAGEWIDTH**(number)

Abbreviation: **[\$]PW**

Example 1: MAST6 raison.a PW(50)
--

Example 2: \$PAGEWIDTH(130)

Note: \$PAGEWIDTH (resp. \$PW) is specified in the assembly file whereas PAGEWIDTH (resp. PW) is typed in the command line.

4.3.5 Number of line in a listing file page

The number of lines per page printed in the listing file may be specified with the `PAGELength` directive. This number must be between 10 and 65535.

The default value is 60.

Syntax: **[\$]PAGELength(number)**

Abbreviation: **[\$]PL**

Example 1:

```
MAST6 raison.a PL(120)
```

Example 2:

```
$PAGELength(20)
```

`PAGELength(0)` is accepted as well and means that no page setting is processed. It may be an interesting option when exporting the listing file to another software program.

Note: `$PAGELength` (resp. `$PL`) is specified in an assembly file whereas `PAGELength` (resp. `PL`) is typed in a command line.

4.3.6 Unassembled parts of a conditional block

It may sometimes be useful to include the unassembled parts of a conditional IF,ELSIF,ELSE,ENDIF block of the source file in the listing file.

This is possible using the COND directive either in a command line, or at the top of the source file.

Syntax: **[\$]COND**

Example 1: MAST6 raison.a COND
--

Example 2: \$COND

The NOCOND prevents unassembled portions of a conditional IF,ELSIF,ELSE,ENDIF block of the source file from appearing in the listing file.

Syntax: **[\$]NOCOND**

Example 1: MAST6 raison.a NOCOND
--

Example 2: \$NOCOND

Note: \$COND (resp. \$NOCOND) is specified in an assembly file whereas COND (resp. NOCOND) is typed in a command line. The default is COND.

4.3.7 Definition of the relocation page

The page of the relocation segment is AUTO by default except if it has been defined by a directive.

PRPAGE is used for the CODE and DATAROM segment types

Syntax: **[\$]PRPAGE** (num)

num is taken from: 0, 2, 3, AUTO, STATIC

DTPAGE is used for the DATA segment type

Syntax: **[\$]DTPAGE** (num)

num is taken from: 1, 2, AUTO, STATIC

EEPAGE is used for the EEPROM segment type

Syntax: **[\$]EEPAGE** (num)

num is taken from: 0, 1, AUTO

MODULE is used to define a value for the CODE page and the DATA page

Syntax: **[\$]MODULE** (num)

num is taken from: 0 -> 11, AUTO (see RL-ST6 manual §3.3.4)

4.3.8 Date

The DATE directive allows you to specify the current date in the header of each page of the listing file.

Syntax: **[\$]DATE**(dd/mm/yy)

Abbreviation: **[\$]DA**

Example 1: MAST6 raison.a DATE(25/12/98)
--

Example 2: \$DATE(25/12/98)

Note: \$DATE (resp. \$DA) is specified in an assembly file whereas DATE (resp. DA) is typed in a command line.

4.3.9 Error messages

Error messages may be displayed by using the ERRORPRINT directive.

Error messages will either be output to the console or written in a specified file depending on whether there is a filename after the ERRORPRINT directive or not.

If no file name is specified, all error messages are sent to the console.

Syntax: **[\$]ERRORPRINT**
 [\$]ERRORPRINT(file_name)

Abbreviation: **[\$]EP**

Example 1: MAST6 raison.err ERRORPRINT(no_error.err)
--

Example 2: MAST6 raison.err EP
--

Example 3: \$ERRORPRINT(no_error.err)

No error file is generated with NOERRORPRINT

Syntax: **[\$]NOERRORPRINT**

Abbreviation: **[\$]NOEP**

Example 4: MAST6 raison.err NOEP
--

Note: \$ERRORPRINT (resp. \$EP, \$NOERRORPRINT, \$NOEP) is specified in an assembly file whereas ERRORPRINT (resp. EP, NOERRORPRINT, NOEP) is typed in a command line. The default is NOERRORPRINT.

4.3.10 Macro assembly instruction

The GEN directive is used to expend a macro definition in the listing file.

Syntax: **[\$]GEN**

Example 1: MAST6 raison.a GEN

Example 2: \$GEN

The NOGEN directive is used to prevent the macro assembler definitions from being expanded in the listing file.

Syntax: **[\$]NOGEN**

Example 3: MAST6 raison.a NOGEN

Note: \$GEN (resp. \$NOGEN) is specified in an assembly file whereas GEN (resp. NOGEN) is typed in a command line. The default is GEN.

4.3.11 Source file inclusion

It may sometimes be useful to have the source file written in the listing file
This is possible with the LIST directive.

Syntax: **[\$]LIST**

Abbreviation: **[\$]LI**

Example 1: MAST6 raison.a LIST
--

Example 2: \$LI

The NOLIST directive prevents the source program from appearing in the
listing file (unless an error occurs at that line).

Syntax: **[\$]NOLIST**

Abbreviation: **[\$]NOLI**

Note: \$LIST (resp. \$LI, \$NOLIST, \$NOLI) is specified in an assembly file
whereas LIST (resp. LI, NOLIST, NOLI) is typed in a command line.
The default is NOLIST.

4.3.12 Form feed

The EJECT directive is used to insert a form feed into the listing file after the line containing this directive.

Syntax: **EJECT**

Abbreviation: **EJ**

Note: 1. This directive is ignored if the NOLIST directive or the NOPRINT directive has been previously specified.

2. This directive cannot be specified in a command line.

Example: \$EJECT

4.3.13 Symbol table

The SYMBOL directive writes all symbols used by the assembly program in the listing file..

Syntax: **[\$]SYMBOLS**

Abbreviation: **[\$]SB**

Example 1: MAST6 raison.a SYMBOLS

Example 2: \$SB

The NOSYMBOLS directive is used to prevent the assembler from including this symbol table in the listing file.

Syntax: **[\$]NOSYMBOLS**

Abbreviation: **[\$]NOSB**

Example 3: MAST6 raison.a NOSB
--

Note: \$SYMBOLS (resp. \$SB, \$NOSYMBOLS, \$NOSB) is specified in an assembly file whereas SYMBOLS (resp. SB, NOSYMBOLS, NOSB) is typed in a command line. The default is NOSYMBOLS.

4.3.14 Cross reference table

The XREF directive is used to generate a cross reference table of all symbols used in the source module in the listing file.

Syntax: **[\$]XREF**

Abbreviation: **[\$]XR**

Example 1: MAST6 raison.a XREF
--

Example 2: \$XREF

The NOXREF directive prevents the assembler from generating the cross reference table in the listing file.

Syntax: **[\$]NOXREF**

Abbreviation: **[\$]NOXR**

Example3: MAST6 raison.a NOXR

Note: \$XREF (resp. \$XR, \$NOXREF, \$NOXR) is specified in an assembly file whereas XREF (resp. XR, NOXREF, NOXR) is typed in a command line. The default is NOXREF.

4.4 Macro processor

The macro syntax developed in this chapter is the macro syntax used by default by Raisonance MA-ST6.

4.4.1 Definition

A macro is a group of one or several instruction lines and is declared using the **MACRO** directive.

The macro can be used an unlimited number of times within the file by using the macro name.

At assembly, the corresponding lines of code are substituted each time the name of the macro is encountered (they are placed between the keyword **MACRO** and the keyword **ENDM**).

The corresponding instructions are then assembled as if they had been written in place of the macro.

Syntax: macro_name **MACRO** par1,par2,..park
 {instruction lines}
 ENDM

Example:

```
SAVE MACRO
    ; declaration of a macro whose name is SAVE
    LD  A,X
    LD  SAVE_X,A
    LD  A,Y
    LD  SAVE_Y,A
    LD  A,V
    LD  SAVE_V,A
    LD  A,W
    LD  SAVE_W,A
ENDM
    ; end of macro declaration
restore MACRO
    LD  A,SAVE_X
    LD  X,A
    LD  A,SAVE_Y
    LD  Y,A
    LD  A,SAVE_V
    LD  V,A
    LD  A,SAVE_W
    LD  W,A
ENDM
```

4.4.2 Repetition of blocks

A block within a macro can be repeated once for each occurrence of a formal parameter.

This is possible due to the macro repetition directives: REPT, WHILE, IRP and IRPC.

Repeated blocks may be nested due to the LOCAL directive.

4.4.2.1 Sequential block repetition

4.4.2.1.1 Repetition of blocks controlled by a number

The REPT directive is used to repeat a block of text.

The number of repetition must be specified after REPT directive.

This directive must be used with the ENDM directive which specifies the end of the block to repeat.

Syntax: **REPT** NumberOfRepetition
 {instruction lines}
 ENDM

Example: delay MACRO REPT 3 NOP ENDM ; end of repeated block ENDM ; end of the macro
--

; When the delay macro is invoked it will generate the following code:

NOP NOP NOP

4.4.2.1.2 Repetition of blocks controlled by a Boolean expression

The WHILE directive is used to repeat a block of text. This block is repeated until the Boolean expression specified after the WHILE statement is true.

This directive must be used with the ENDM directive which specifies the end of the block to repeat.

Syntax: **WHILE**(exp)
 {instruction lines}
 ENDM

4.4.2.1.3 Repetition of a block of text controlled by an argument list

The IRP directive is also used to repeat a block of text.

The repetition number is defined by the number of arguments specified in the list following IRP statement.

This directive must be used with the ENDM directive which specifies the end of the block to repeat.

Syntax: **IRP** parameter,<arg [, arg ...]>
 {instruction lines}
 ENDM

```
Example:
init_tab  MACRO
            IRP    table_element, <055h,0AAh,05Ah>
            LDI    A, #table_element
            LD      (X), A
            INC     X
            ENDM
        ENDM

; When invoked the init_tab macro will generate the following
; code:
            LDI    A, #055h
            LD      (X), A
            INC     X
            LDI    A, #0AAh
            LD      (X), A
            INC     X
            LDI    A, #05Ah
            LD      (X), A
            INC     X
```

4.4.2.1.4 Repetition of a block of text controlled by a set of characters

The IRPC directive is also used to repeat a block of text.

The repetition number is not explicitly specified by a number but is implicitly calculated regarding the character number composing the argument.

This directive must be used with ENDM directive which specifies the end of the block to repeat.

Syntax: **IRPC** parameter,<string>

...

ENDM

Example:

```
Send_string MACRO
    IRPC    param, <Raisonance>
    LDI     A, #'param'
    CALL    MYPUTCHAR
    ENDM
ENDM
```

; When invoked the Send_string macro will send the characters
; as follows:

```
    LDI     A, , #'R'
    CALL    MYPUTCHAR
    LDI     A, , #'a'
    ...
```

4.4.2.2 Nested block repetition

The macro repetition directives (REPT, WHILE, IRP and IRPC) may be nested up to 9 levels deep.

Example 1:

```
fast_send_string MACRO
    REPT 8
        IRPC param, <initvalue>
            LDI A, #'param'
            CALL MyPutChar
        ENDM
        ; end of IRPC block
    ENDM
    ; end of REPT block
ENDM
; end of fast_send_string macro
```

With the LOCAL directive, you even can declare up to 16 local symbols (labels, data declarations...) for use in a macro.

Syntax: macro_name **MACRO** par1,par2,..park
 LOCAL symb [,symb ...]
 {instruction lines}
 ENDM

Example 2:

```
slow_send_string MACRO
    LOCAL delay
    delay MACRO
        REPT 12
            NOP
        ENDM
    ENDM
    ; end of repeated block
    ; end of delay macro
    IRPC param, <initvalue>
        LDI A, #'param'
        CALL MyPutChar
        delay
    ENDM
    ; end of IRPC block
ENDM
; end of slow_send_string macro
```

4.4.3 Macro operators

Some operators may be used in macros.

4.4.3.1 Determining if a macro argument is null

The NUL operator is used to determine whether the argument following its statement is null or not.

Note: NUL is not equivalent to ‘ ‘.

Syntax: NUL argument

Example:

```
raison MACRO argu
    IF NUL argu
        ...
        ; this block will be treated provided
        ; argu is NUL
    ENDIF
    ...
ENDM
```

4.4.3.2 Concatenation of text and macro parameters

The Operator & is used to concatenate text and macro parameters.

Example:

```
routine_title MACRO title
    title&1: NOP
ENDM
```

```
; When invoked by routine_title RAISON the routine_title macro
; will generate the following code:
RAISON1: NOP
```

4.4.3.3 Keeping the literal text of an expression

The angle brackets operators (< and >) are used to enclose text that should remain the same until macro code generation.

Example1:

```
new_macro_example MACRO argu
    IRP table_name,<argu>
        LDI V,#table_name&1
        LDI W,#table_name&2
    ENDM
ENDM
```

```
; When invoked by « new_macro_example 10 » the welcome macro
; will generate the following code:
        LDI V,#101
        LDI W,#102
```

The exclamation mark operator may be used to pass characters such as commas.

4.4.3.4 Evaluating of an expression

The percent character operator (%) is used to evaluate an expression and to pass its value to the macro rather than the literal expression.

Example:

```
init_Rn MACRO argu
    IRPC register_number,<0123>
        LDI REG&register_number,#argu
    ENDM
ENDM
ten      EQU 0ah
init_Rn %ten
```

; When invoked by « init_Rn 0ah », the macro will generate the following
; code:

```
LDI REG0,#0ah
LDI REG1,#0ah
LDI REG2,#0ah
LDI REG3,#0ah
```

4.4.3.5 Macro local comments

The double semicolon operator (;;) is used to insert comments into the macros without generating the text when the macro code is expanded.

Example:

```
registers01_init MACRO
    LDI REG0,#0 ;; REG0 initialization
    LDI REG1,#0 ;; REG1 initialization
ENDM
```

5. Appendices

5.1 *Keywords*

5.2 *Error Messages*

5.1 Keywords

The table below lists all keywords:

The Keywords in plain text are recognized as assembler instructions by the assembler.

ADD	ADDI	AND	ANDI	ASCII
ASCIZ	ASEG			
BRIDGEON				
	CALL	CLR	CODE	COM
CP	CPI	CSEG		
DATA	DATAROM	DATE	DB	DEC
DEBUG	DRSEG	DS	DSB	DSD
DSEG	DSW	DTPAGE	DW	
EEPROM	EEPAGE	EJ	EJECT	ELSE
ELSEIF	END	ENDIF	ENDM	ENDP
EP	EQ	EQU	ERRORPRINT	ESEG
EXTERN				
GEN	GT	GTE		
HIGH				
IF	INCLUDE	INC	IRP	IRPC
JP	JRC	JRNC	JRZ	JRNZ
JRR	JRS			
LI	LINE	LIST	LOCAL	LOW
LT	LTE			
MACRO	MO	MOD	MODULE	
NAME	NE	NOCOND	NODB	NODEBUG
NOEP	NOERRORPRINT	NOGEN	NOLIST	NOLI
NOMO	NOOJ	NOOBJECT		NOPRINT
NOSB	NOSYMBOLS	NOXR	NOXREF	NUL
NUMBER				
OBJECT	OJ	ORG	OVERLAYABLE	
PAGE	PAGELENGTH	PAGEWIDTH	PL	PR
PRPAGE	PRINT	PUBLIC	PW	
REPT	RET	RETI	RES	RLC
RSEG				
SB	SEGMENT	SET	SLA	STOP
SUB	SUBI	SYMBOLS		
TITLE	TT			

UNIT

V

W

X

Y

WAIT

XR**WHILE****XREF****WINDOW****WINOFFSET**

5.2 Error Messages

1	ILLEGAL CHARACTER IN NUMERIC CONSTANT	
2	INVALID CHARACTER IN NUMERIC CONSTANT	
3	MISSING STRING TERMINATOR	
4	BAD (OR MISUSED) SPECIAL CHARACTER	
5	BAD INDIRECT REGISTER	
6	'Name' ILLEGAL USE OF A RESERVED WORD	
7	DEFINITION STATEMENT EXPECTED	
8	LABEL NOT PERMITTED	
9	ATTEMPT TO REDEFINE LABEL 'Name'	
10	SYNTAX ERROR	
11	ATTEMPT TO REDEFINE SYMBOL 'Name'"	
12	STRING CONTAINS MORE THAN TWO CHARACTERS	
13	ILLEGAL FACTOR	
14	)' EXPECTED	
15	BAD RELOCATABLE EXPRESSION	
16	MISSING FACTOR	
17	DIVIDE BY ZERO ERROR	
18	INVALID BYTE BASE IN BIT ADDRESS EXPRESSION	
19	OUT OF RANGE OR NON-TYPELESS BIT-OFFSET	
20	INVALID REGISTER FOR EQU/SET	
21	INVALID SIMPLE RELOCATABLE EXPRESSION	
22	EXPRESSION WITH FORWARD REFERENCE NOT PERMITTED	
23	EXPRESSION TYPE DOES NOT MATCH INSTRUCTION	
24	NUMERIC EXPRESSION EXPECTED	
25	SEGMENT-TYPE EXPECTED	
26	RELOCATION-TYPE EXPECTED	
27	BAD RELOCATION-TYPE	
28	LOCATION COUNTER MAY NOT POINT BELOW SEGMENT-BASE	
29	ABSOLUTE EXPRESSION REQUIRED	
30	SEGMENT-LIMIT EXCEEDED	
31	SEGMENT-SYMBOL EXPECTED	
32	PUBLIC-ATTRIBUTE NOT PERMITTED	
33	ATTEMPT TO RESPECIFY MODULE NAME	
34	CONFLICTING ATTRIBUTES	
35	',' EXPECTED	
36	(' EXPECTED	
37	INVALID NUMBER FOR REGISTER BANK	
38	OPERATION INVALID IN THIS SEGMENT	
39	NUMBER OF OPERANDS DOES NOT MATCH INSTRUCTION	
40	REGISTER-OPERAND EXPECTED	
41	INVALID REGISTER FOR THIS INSTRUCTION	

42	PREMATURE END OF FILE (NO END STATEMENT	
43	'LABEL' PHASE ERROR (PASS-2)",	
44	RESPECIFIED PRIMARY DIRECTIVE, LINE IGNORED	
45	MISPLACED PRIMARY DIRECTIVE, LINE IGNORED	
46	UNDEFINED SYMBOL (PASS-2) ' <i>Name</i> '	
47	CODE-ADDRESS EXPECTED	
48	XDATA-ADDRESS EXPECTED	
49	DATA-ADDRESS EXPECTED	
50	IDATA-ADDRESS EXPECTED	
51	BIT-ADDRESS EXPECTED	
52	TARGET OUT OF RANGE	
53	VALUE HAS BEEN TRUNCATED TO 8 BITS	
54	MISSING 'USING' INFORMATION	
55	MISPLACED CONDITIONAL DIRECTIVE	
56	BAD CONDITIONAL EXPRESSION	
57	UNBALANCED IF-ENDIF-DIRECTIVES	
58	SAVE STACK UNDERFLOW	
59	SAVE STACK OVERFLOW",	
60	ATTEMPT TO DEFINE AN ALREADY DEFINED MACRO",	
61	SOURCE-FILE LINE CONTAINS MORE THAN 250 CHARACTERS	FATAL
62	ERRORPRINT- AND LIST-FILE CANNOT BE THE SAME	FATAL
63	NON-NULL ARGUMENT EXPECTED	FATAL
64	CONFLICTING DIRECTIVE	FATAL
65	CANNOT HAVE GENERAL DIRECTIVE IN INVOCATION-LINE	FATAL
66	ARGUMENT TOO LONG	FATAL
67	DELIMITER '(' AFTER OPTION EXPECTED	FATAL
68	DELIMITER ')' AFTER OPTION EXPECTED	FATAL
69	UNDEFINED DIRECTIVE	FATAL
70	BAD NUMERIC CONSTANT	FATAL
71	OUT OF RANGE NUMERIC VALUE	FATAL
72	SYMBOL-TABLE SPACE EXHAUSTED	FATAL
73	TOO MANY WARNINGS	
74	IDENTIFIER EXPECTED	
75	DB/DW/DD' ALLOWED ONLY IN CODE SEGMENT	
76	'DSB' NOT ALLOWED OUTSIDE OF A 'BYTEADDRESSABLE' SEGMENT	
77	INVALID BIT OFFSET",	
78	SYMBOL NATURE ERROR	
79	UNABLE TO CREATE LISTING FILE ' <i>Name</i> '	
80	UNABLE TO CREATE OBJECT FILE ' <i>Name</i> '	
81	UNABLE TO CREATE TEMPORARY FILE ' <i>Name</i> '	
82	BAD PARAM NUMBER IN MACRO INVOCATION	
83		
84	CANNOT OPEN FILE ' <i>Name</i> '	

85	INTERNAL ERROR	
86	BAD REGISTER	
87	MISSING 'END' STATEMENT	
88	UNEXPECTED END OF FILE	
89	TOO MANY ERRORS	
90	TOO MANY SEGMENTS	
91	UNABLE TO OPEN INCLUDE FILE ' <i>Name</i> '	
92	UNABLE TO OPEN SERIAL NUMBER FILE",	
93	BAD SERIAL NUMBER	
94	POSITIVE VALUE EXPECTED	
95	VALUE OUT OF RANGE, IT HAS BEEN TRUNCATED	
96	INTERNAL ERROR	
97	INCLUDE LOOP	
98	ELSE WITHOUT IF	
99	ELSE IN ELSE STATEMENT	
100	ELSE TYPE DOESN'T MATCH IF TYPE	
101	ELSEIF WITHOUT IF	
102	ELSEIF IN ELSE STATEMENT	
103	ELSEIF TYPE DOESN'T MATCH IF TYPE	
104	ENDIF WITHOUT IF	
105	ENDIF TYPE DOESN'T MATCH IF TYPE	
106	ILLEGAL OBJECT FILE EXTENSION	
107	ILLEGAL LISTING FILE EXTENSION	
108	UNREACHABLE ADDRESS	
109	OFFSET IS TOO BIG	
110	INVALID BIT ADDRESS	
111	WRONG NUMBER OF OPERANDS	
112	VALUE HAS BEEN TRUNCATED TO 4 BITS	
113	INVALID OPERAND	
114	'INPAGE' RELOCATION-TYPE INVALID WITH 'AT'	
115	""INPAGE' RELOCATION-TYPE INVALID WITH 'OFFS'	
116	""INPAGE' RELOCATION-TYPE INVALID WITH 'OFFS'	
117	'LOWBYTEADDR' APPLIED TO A BYTE LABEL	
118	'HIGHBYTEADDR' APPLIED TO A BYTE LABEL	
119	'BYTELABEL' NOT ALLOWED IN A WORD SEGMENT	
120	'Rel8' ON BYTE SYMBOL ' <i>Name</i> ': WORD ADDRESS ASSUMED	
121	ONE BYTE INSERTED FOR ALIGNMENT	
122	WORD LABEL ON AN ODD BYTE	
123	ADDRESS OUT OF RANGE	
125	'DSB' OUTSIDE OF A DATA SEGMENT	
126	'DSW' OUTSIDE OF A DATA SEGMENT	
127	SYMBOL NOT COMPATIBLE	
128	LABEL EXPECTED	
129	OPERAND OUT OF RANGE [0..15]",	

130	'DSD' OUTSIDE OF A DATA SEGMENT	
131	OPERAND OUT OF RANGE [0..F]",	
132	OPERAND OUT OF RANGE [0..FF]	
133	HARD REGISTER INVALID WITH 'REG'	
134	EVALUATION VERSION DOES NOT GENERATE OBJECT FILE	
135	EVALUATION VERSION: MAXIMUM SIZE OF OBJECT FILE REACHED	
136	MULTIPLE ALIGNMENT SPECIFICATION	
137	ADDRESS EXPECTED	
138	MULTIPLE RELOCATION SPECIFICATION	
139	'ENDP' WITHOUT 'PROC'	
140	'ENDP' ALLOWED ONLY IN CODE SEGMENTS	
141	PROCEDURE CANNOT BE NESTED	
143	UNCLOSED PROCEDURE	
144	'PROC' ALLOWED ONLY IN CODE SEGMENTS",	
145	HARD REGISTER INVALID WITH 'RBIT'	
146	REGISTER INVALID WITH 'HRBIT'	
147	INVALID NUMBER	
148	USER BREAK	
149	UNKNOWN DIRECTIVE, LINE IGNORED	
150	UNKNOWN MNEMONIC	
153	INITIALIZERS NUMBER ERROR	
154	ORG ASSUMED	
155	INVALID DATA	
156	OBSOLETE FO DIRECTIVE	
157		
158	COMMAND LINE TOO LONG	
159		
160		
161	INVALID OPERATION	
162	'Name' INVALID OUTSIDE A MACRO	
163	LISTING LINE HAS BEEN TRUNCATED	
164	MPL-PREPROCESSOR ERROR	
165		
166	RELOCATABLE SEGMENT OPEN WITH ASEG	
167	ABSOLUTE SEGMENT OPEN WITH RSEG	
168	ATTEMPT OF MACRO REDEFINITION	
169		
170		
171		
172		

5.2.1 Error types

A warning is generated if any abnormalities are detected in the program that may have been caused by a programming error. A warning indicates source code that is allowed by the standard but either with a style that does not conform to good programming rules or with a highly probable programming error.

An error does not usually abort the assembling process, but suspends code generation.

A fatal error causes immediate termination of the compilation. The most likely cause is generally a system problem such as a full disk or memory space.

5.2.2 List of errors

Error 1: ILLEGAL CHARACTER IN NUMERIC CONSTANT

Numeric constants must begin with a decimal digit. Dollar signs are allowed in constants to improve readability. This error occurs if the dollar sign is the last character in a numeric constant.

Error 2: INVALID CHARACTER IN NUMERIC CONSTANT

Indicates that an invalid character was found in a numeric constant.

Error 3: MISSING STRING TERMINATOR

The ending string terminator is missing.

Error 4: BAD (OR MISUSED) SPECIAL CHARACTER

A special character as %, &, and ! is not used correctly.

Error 5: BAD INDIRECT REGISTER

Combined registers are entered incorrectly, e.g. @R2, @R3, @DPTR, @A+PC, @A+DPTR

Error 6: 'symb' ILLEGAL USE OF A RESERVED WORD

A label has the same name as a reserved keyword (see MA-51 user manual).

Error 8: LABEL NOT PERMITTED

The use of this label is not permitted in this context.

Error 9: ATTEMPT TO REDEFINE LABEL '*name*'

Error 10: SYNTAX ERROR

The MA-51 assembler encountered an error of syntax when processing the specified line.

Error 11: ATTEMPT TO REDEFINE SYMBOL 'symb'

The symbol "symb" has been define more than once. You must keep only one definition of the symbol "symb".

Error 14: ')' EXPECTED

A closing parenthesis is expected. This error usually occurs for the definition of external symbols:

Example: EXTRN DATA (i,j,buf

The ')' is missing at the end of the expression.

Error 15: BAD RELOCATABLE EXPRESSION

This error is generated, for example, when operands of an instruction are not of one of the following types:

1. constant_numeric_value
2. relocatable_symbolic_value
3. relocatable_symbolic_value + constant_numeric_value
4. relocatable_symbolic_value - relocatable_symbolic_value

NOTES:

1. In case 4, both symbols must be defined in the file and in the same segment. In that case, the difference is equivalent to a constant numeric value (i.e. "gap" between these two symbols in the segment).
2. The symbolic operators (HIGH-LOW) are allowed on the symbolic values.
3. A constant numeric value can be made of every expression established by combination of the operators described in the manual.
Then, $2*3+4$ is also a constant_numeric_value because the expression will be evaluated during assembling.
However, replacement of "2" and "3" by relocatable symbols will produce an error:
extrn code (lab1,lab2)
toto set lab1 + Lab2 ; << error

Error 16: MISING FACTOR**Error 17: DIVIDE BY ZERO ERROR**

Division by zero is forbidden. A division by zero is attempted while calculating an expression. The value calculated is undefined.

Error 18: INVALID BYTE BASE IN BIT ADDRESS EXPRESSION

In the expression EXP.BITOFFS, EXP is not a correct symbolic expression

Example:

setb DPTR.1 ;error..

Error 19: OUT OF RANGE OR NON-TYPELESS BIT-OFFSET

In the expression EXP.BITOFFS, BITOFFS is not a numerical constant included in the interval [0,7].

Example:

setb P1.8 ;error

Error 20: INVALID REGISTER FOR EQU/SET**Error 21: INVALID SIMPLE RELOCATABLE EXPRESSION**

Example:

DEMOSEG segment BIT

rseg DEMOSEG

LABS: DBIT 1

CSEG at 0

org LABS ; a simple expression is required here!!

Error 23: EXPRESSION TYPE DOES NOT MATCH INSTRUCTION

The expression is not complete. A #, /Bit, register, or numeric expression was expected.

Example: DIV A

Division needs two elements

Error 24: NUMERIC EXPRESSION EXPECTED

A numeric expression is expected but an expression of another type is found.

Error 25: SEGMENT-TYPE EXPECTED

demoxdataseg SEGMENT XDATA

demoxxxx SEGMENT XXX ; XXX is not a valid segment type.

Segment type is among CODE, DATA, XDATA, BIT

Error 26: RELOCATION-TYPE EXPECTED

Example:

democodeseg SEGMENT CODE OUTPAGE

The relocation type is among INPAGE, PAGE, INBLOCK but OUTPAGE is not a valid relocation page.

Error 27: BAD RELOCATION-TYPE

Example:

democodeseg SEGMENT CODE BITADDRESSABLE

The relocation type is not allowed for the segment type (here, ibtaddressable can be used only for DATA segments).

Error 29: ABSOLUTE EXPRESSION REQUIRED

The operand of an RSEG directive must be a segment symbol. The operand of the RSEG directive has not been defined or not as a segment symbol.

Check that this operand is defined as a segment symbol.

Error 30: SEGMENT-LIMIT EXCEEDED

dataseg segment DATA

rseg DATASEG

org 400H ; DATA segment is up to 256 bytes

Error 31: SEGMENT-SYMBOL EXPECTED

The operand of an RSEG directive must be a segment symbol

The following example is valid:

```
fct3CD SEGMENT CODE
```

```
RSG fct3CD
```

Error 32: PUBLIC-ATTRIBUTE NOT PERMITTED

The PUBLIC attribute is not permitted for the specified symbol

Error 33: ATTEMPT TO RESPECIFY MODULE NAME

The NAME directive is used more than once. The NAME directive allows which name to use for the object module generated for the current program. So only one name can be defined.

Error 34: CONFLICTING ATTRIBUTES**Error 35: ',' EXPECTED**

A coma ',' is expected in a list of expressions or symbols.

Error 36: '(' EXPECTED

A left parenthesis is expected at the specified line.

Example: EXTRN DATA i,j,buf) \\ INVALID

EXTRN DATA (i,j,buf) \\ VALID

Error 37: INVALID NUMBER FOR REGISTER BANK**Error 38: OPERATION INVALID IN THIS SEGMENT**

The operation is valid only in a CODE segment

The following example is not valid:

```
fct3CD SEGMENT DATA
```

```
RSEG fct3CD
```

It should be

```
fct3CD SEGMENT CODE
```

```
RSEG fct3CD
```

Error 39: NUMBER OF OPERANDS DOES NOT MATCH INSTRUCTION

NOP A ; NOP does not take any operand

JC A,\$; JC takes only one operand

Error 43: 'LABEL' PHASE ERROR (PASS-2)

Occurs if a symbol in PASS 2 contains a value different that in PASS 1.

Error 45: MISPLACED PRIMARY CONTROL, LINE IGNORED

The primary control is misplaced. Primary controls may be entered in the invocation line or at the beginning of the source file (as \$ instruction). If no primary control is define in the first line, the other defined primary controls are ignored.

Error 46: UNDEFINED SYMBOL (PASS-2) 'symb'

The symbol that is used here is not defined. You must define it before using it.

Error 47: CODE-ADDRESS EXPECTED

A CODE or typeless expression is expected

Error 48: XDATA-ADDRESS EXPECTED

An XDATA or typeless expression is expected

Error 49: DATA-ADDRESS EXPECTED

A DATA or typeless expression is expected

Error 50: IDATA-ADDRESS EXPECTED

An IDATA or typeless expression is expected

Error 51: BIT-ADDRESS EXPECTED

A BIT or typeless expression is expected

Error 52: TARGET OUT OF RANGE

```
org 100h
sjmp label ; label is too far for a sjmp/ Use a LJMP
org 200h
label: nop
```

Error 53: VALUE HAS BEEN TRUNCATED TO 8 BITS

```
MOV A,#1234H
```

Error 55: MISPLACED CONDITIONAL DIRECTIVE**Error 56: BAD CONDITIONAL EXPRESSION**

An IF or ELSEIF control is invalid. An ELSEIF may be defined without IF before.

These expressions must be absolute and may not contain relocatable symbols.

Error 57: UNBALANCED IF-ENDIF-CONTROLS

Each IF block has to be terminated with an ENDIF control. There should be the same number of IF and ENDIF controls.

Error 58: SAVE STACK UNDERFLOW

You can restore only what has been saved before. A \$RESTORE control instruction is valid only if a \$SAVE control was previously executed.

Error 59: SAVE STACK OVERFLOW

The context of the GEN, COND and LIST controls may be stored by the \$SAVE control a maximum of 9 times.

Fatal 63: NON-NULL ARGUMENT EXPECTED

The name of the source file to assemble or a file name in a command line option was omitted

Fatal 64: CONFLICTING CONTROL

An option on the command line conflicts with another option.

Example:

MA51.EXE EXAMPLE.A51 **NOOBJECT OBJECT**(EXAMPLE.OBJ)

Fatal 65: CANNOT HAVE GENERAL CONTROL IN INVOCATION-LINE

Some controls can be used only in the source file but cannot be used in the command line. See the specified control to have more information on its use;

Fatal 66: ARGUMENT TOO LONG

An Argument contains too many characters.

Fatal 69: UNDEFINED CONTROL

The specified control in the command line is not recognized by the MA-51 Assembler

Fatal 71: OUT OF RANGE NUMERIC VALUE

A numeric argument is out of range. This occurs for example, when the PAGELNGTH option has an argument that is smaller than 10 or greater than 65535

Fatal 72: SYMBOL-TABLE SPACE EXHAUSTED

There is no more room for symbol information in the symbol table. The symbol table can accommodate approximately 40Kbytes of symbols.

Error 73: TOO MANY WARNINGS

Too many warnings have been found when translating the file.

Error 75: 'DB/DW' ALLOWED ONLY IN CODE SEGMENT

The use of DB and DW directives are allowed only in segment that have been defined as CODE type.

Check that you have not tried to use them in another space segment.

Error 76: 'DS' NOT ALLOWED IN BIT SEGMENT

The DS directive is used to reserve space in the internal data space or external data space. So, the DS directive cannot be used in BIT segment.

NOT VALID

```
demobitseg    SEGMENT BIT
```

```
RSEG demobitseg
```

```
COUNTER:    DS    1
```

VALID

```
demodataseg   SEGMENT DATA
```

```
RSEG demodataseg
```

```
COUNTER:    DS    1
```

Error 77: 'DBIT' ALLOWED ONLY IN BIT SEGMENT

The use of DBIT is allowed only in segment that have been defined as BIT type.

Example:

```
demobitseg          SEGMENT CODE //NOT VALID
```

```
demobitseg          SEGMENT BIT   //VALID
```

```
RSEG demobitseg
```

```
MES_SEG:    DBIT 1
```

Error 79: UNABLE TO CREATE LISTING FILE '%s'

'lstfilename' cannot be opened: the disk is full, or a protected (read-only) file already exists.

Fatal 80: UNABLE TO CREATE OBJECT FILE '%s'

'objfilename' cannot be opened: the disk is full, or a protected (read-only) file already exists.

Fatal 81: UNABLE TO CREATE TEMPORARY FILE '%s'

'tmpfilename' cannot be opened: the disk is full, or a protected (read-only) file already exists

Warning 82: BAD PARAM NUMBER IN MACRO INVOCATION

The macro is invoked with less or more parameters as declared in the definition.

Fatal 84: CANNOT OPEN FILE '%s'

The specified file cannot be open; Check that this file exist and that it is well placed.

Error 85: INTERNAL ERROR

This error should not occur, ... contact the technical support.

Warning 87: MISSING 'END' STATEMENT

The program must be terminated by an END statement. This END statement is missing. Put an END statement at the end of your program.

Fatal 88: UNEXPECTED END OF FILE

This error appears when an assembling is ended while a macro is still running (the endm instruction is missing) or a conditional assembling block (\$if) is not closed.

Fatal 89: TOO MANY ERRORS

Too many errors have been found. The translation process has been stopped.

Fatal 90: TOO MANY SEGMENTS

Too many segments have been defined.

Fatal 91: UNABLE TO OPEN INCLUDE FILE '%s'

At least one of the include file (.inc) that are called from your source file cannot be open.

Check that they exist and that they are placed in the right directory.

Warning 92: UNABLE TO OPEN SERIAL NUMBER FILE

The files concerning the tool serialization have not been opened successfully. Contact Raisonance.

Warning 93: BAD SERIAL NUMBER

The serial number is not correct. Contact Raisonance.

Error 94: POSITIVE VALUE EXPECTED

The value that defines the expression or the segment must be a positive value.

Warning 95: VALUE OUT OF RANGE, IT HAS BEEN TRUNCATED

The specified numerical value is outside the allowed interval.

Example:

iseg at 1000H ; the address of an idata segment has to be less than 256.

Error 96: INTERNAL ERROR

This error should not occur, ... contact the technical support.

Fatal 97: INCLUDE LOOP

The use of the file inclusion with the INCLUDE directive leads to loop: the included file include itself either directly or indirectly.

Error 98: ELSE WITHOUT IF

The ELSE control is used for a block program without the IF control for the previous block program.

Error 99: ELSE IN ELSE STATEMENT

On a conditional assembling, two “else” blocks follow each other. Syntax allows only one.

Error 100: ELSE TYPE DOESN'T MATCH IF TYPE

An "else" block is encountered, but no current "if" condition.

Error 101: ELSEIF WITHOUT IF

An ELSEIF control has been used without the use of a previous IF control.

Error 102: ELSEIF IN ELSE STATEMENT

On a conditional assembling, an else block can follow an elseif block, but the reverse is not allowed (an else block, if it exists, should be always in last position).

Error 103: ELSEIF TYPE DOESN'T MATCH IF TYPE

An “elseif” block" is encountered, without a current “if” condition

Error 104: ENDIF WITHOUT IF

The ENDIF control is used without any IF control. If several ENDIF and IF controls are used, check that there is the same number of IF as ENDIF controls.

Error 105: ENDIF TYPE DOESN'T MATCH IF TYPE

An "endif" directive is encountered, without a current "if" condition.

Error 106: ILLEGAL OBJECT FILE EXTENSION

Not all the files are possible for the generation of an object file. For example, the LST suffix is reserved for the listing files and cannot be used simultaneously for the object file.

Error 107: ILLEGAL LISTING FILE EXTENSION

Not all the files are possible for the generation of a listing file. For example, the OBJ suffix is reserved for the object files and cannot be used simultaneously for the listing file.

Error 109 : OFFSET IS TOO BIG**Error 110 : INVALID BIT ADDRESS****Error 111 : WRONG NUMBER OF OPERANDS**

Error 112 : VALUE HAS BEEN TRUNCATED TO 4 BITS

Error 113 : INVALID OPERAND

Error 114 : 'INPAGE' RELOCATION-TYPE INVALID WITH 'AT'

Error 115 : 'INPAGE' RELOCATION-TYPE INVALID WITH 'OFFS'

Error 117 : 'LOWBYTEADDR' APPLIED TO A BYTE LABEL

Error 118 : 'HIGHBYTEADDR' APPLIED TO A BYTE LABEL

Error 119 : 'BYTELABEL' NOT ALLOWED IN A WORD SEGMENT

Error 120 : 'Rel8' ON BYTE SYMBOL '*Name*': WORDCADDRESS ASSUMED

Error 121 : ONE BYTE INSERTED FOR ALIGNMENT

Error 122 : WORD LABEL ON AN ODD BYTE

Error 123 : ADDRESS OUT OF RANGE

Error 124 : 'DSB' OUTSIDE OF A DATA SEGMENT

Error 124 : 'DSW' OUTSIDE OF A DATA SEGMENT

Error 127: SYMBOL NOT COMPATIBLE

Error 128: LABEL EXPECTED

Error 129: OPERAND OUT OF RANGE [0..15]

Error 130: 'DSD' OUTSIDE OF A DATA SEGMENT

Error 131: OPERAND OUT OF RANGE [0..FFFF]

Error 132: OPERAND OUT OF RANGE [0.FF]

Error 133: HARD REGISTER INVALID WITH 'REG'

Fatal 134: EVALUATION VERSION DOES NOT GENERATE OBJECT FILE

You are using an evaluation version of the assembler. In the evaluation version, MA-XA is limited to 8Kb of generated code. To get a full version of the Assembler, please contact us at info@raisonance.com

Fatal 135: EVALUATION VERSION : MAXIMUM SIZE OF OBJECT FILE REACHED

You are using an evaluation version of the assembler. In the evaluation version, MA-XA is limited to 8Kb of generated code. To get a full version of the Assembler, please contact us at info@raisonance.com

Error 136: MULTIPLE ALIGNMENT SPECIFICATION**Error 137: ADDRESS EXPECTED****Error 138: MULTIPLE RELOCATION SPECIFICATION****Error 139: 'ENDP' WITHOUT 'PROC'****Error 140: 'ENDP' ALLOWED ONLY IN CODE AND FAR CODE SEGMENTS****Error 141: PROCEDURE CANNOT BE NESTED",****Error 142: 'FAR' PROCEDURE IN NEAR CODE SEGMENT****Error 143: UNCLOSED PROCEDURE****Error 144: 'PROC' ALLOWED ONLY IN CODE AND FAR CODE SEGMENTS****Error 145: HARD REGISTER INVALID WITH 'RBIT'****Error 146: REGISTER INVALID WITH 'HRBIT'****Error 147: INVALID NUMBER**

The defined number is not a valid one.

Fatal 148: USER BREAK

The user stopped assembling during the processing (with the CANCEL button for example)

Warning 149: UNKNOWN CONTROL, LINE IGNORED

An unknown directive is placed on the command line.

Error 150: UNKNOWN MNEMONIC**Error 153: INITIALIZERS NUMBER ERROR**

When a table has been initialized in code, too many initializers were present (number greater than the table size).

Warning 154: ORG ASSUMED

This warning appears when the former syntax is used for the statement of absolute segments.

Error 155: INVALID DATA**Error 156: OBSOLETE FO CONTROL****Error 158: COMMAND LINE TOO LONG****Error 161: INVALID OPERATION****Error 162: '%s' INVALID OUTSIDE A MACRO**

A macro operator (EXITM for example) has been encountered outside every macro definition.

Warning 163: LISTING LINE HAS BEEN TRUNCATED

The lines exited in the listing file are truncated, because they are too long.

Error 164: MPL-PREPROCESSOR ERROR

An error has been encountered during the MPL preprocessing of the file. The preprocessor usually gives some details on the encountered error.

Warning 166: RELOCATABLE SEGMENT OPEN WITH ASEG**Warning 167: ABSOLUTE SEGMENT OPEN WITH RSEG****Error 168: ATTEMPT TO DEFINE AN ALREADY DEFINED MACRO**

An attempt was made to redefine an already defined macro with the same name.

5.3 Revision History

DATE	SECTION	DESCRIPTION

6. Index

-	16, 17, 18
\$	12
%	
macros.....	76
&	
macros.....	75
()	18
*	16, 18
/	16
::	
macros.....	76
+	16, 17, 18
<	16, 18
macros.....	75
<=	16, 18
=	16, 18
>	16, 18
macros.....	75
>=	16, 18
absolute.....	23
AND	17, 18
ASEG	25
AT.....	24
AUTO.....	24
binary.....	14
C	15
Character	14
CODE	20, 23, 30
Comments.....	11
COND.....	60
Conditional assembly	50
CSEG.....	27
DA	61
DATA.....	20, 23, 31
DATAPAGENUMBER	35
DATAPAGING.....	34
DATAROM.....	20, 23, 32
DATE	61
DB	39, 44
DEBUG	44
decimal	14
Directives	10
DRSEG.....	28
DS.....	38, 92
DSB	38
DSD	38
DSEG.....	27
DSW	38
DW.....	39
EEPROM	20, 23, 32
EEPROMPAGENUMBER.....	35
EJ 65	
EJECT.....	65
ELSE.....	50
ELSEIF	50
END	29
ENDIF.....	50
ENDM.....	68, 73
EP62	
EQ.....	16, 18
EQU	42
ERRORPRINT.....	62
ESEG	28
Expressions	19
EXTERN.....	53
GEN	63
GLOBAL	10
GT	16, 18
GTE.....	16, 18
hexadecimal	14
HIGH	17, 18
HR0-HR15	15
IF 50	
INBLOCK.....	25
Include	
directive.....	49
include file.....	49
INCLUDE.....	49
INPAGE.....	25
Instructions	10
INWINDOW.....	25
IRP	71
IRPC	72
Keyword	80
label.....	13
LI 64	
LINE	45
Linkage directives.....	53
LIST	64
Listing file.....	56

Location counter.....	15	OR.....	17, 18
LOW	17, 18	ORG	15, 29
LT	16, 18	Overlapping	23
LTE.....	16, 18	PAGE.....	17, 23, 27
MACRO	68, 73	PAGELength.....	59
Memory	20	PAGEWIDTH.....	58
MOD.....	16, 18	PL59	
MODULE	61	PR	57
NAME	44	precedence	18
NE.....	16, 18	PRINT	57
NOCOND.....	60	PROGPAGING.....	34
NODATAPAGING	34	PUBLIC	53
NODB.....	44	PW	58
NODEBUG	44	relocatable.....	23
NOEP	62	REPT.....	69
NOERRORPRINT	62	RSEG	25
NOGEN.....	63	SB	66
NOLI	64	Segment	23
NOLIST.....	64	INCLUDE directive	49
NOOBJECT.....	43	SEGMENT	23
NOOJ.....	43	SET	40
NOPR	57	SHL.....	17, 18
NOPRINT.....	57	SHR.....	17, 18
NOPROGPAGING	34	Source file	49
NOSB	66	STATIC	24
NOSYMBOLS	66	String.....	14
NOT.....	17, 18	Symbols	12
NOXR.....	67	SYMBOLS	66
NOXREF	67	TITLE	58
NUL.....	74	TT	58
NUMBER.....	33	Warning	86
OBJECT	43	WHILE	70
Object file	43	WINDOW	17
octal	14	WINOFFSET	17
OJ43		XOR.....	18
Operands.....	14	XR.....	67
operators.....	16	XREF	67

7. Tables index

Table 1: binary arithmetic operators; p 16

Table 2: relational operators; p 16

Table 3: shifting operators; p 17

Table 4: binary operand operators; p 17

Table 5: unary arithmetic operators; p 17

Table 6: miscellaneous unary operators; p 17

Table 7: operator precedence; p 18

8. Bibliography
