ISaGRAF
Version 3.4

Workbench / Kernel Interface Reference

CJ INTERNATIONAL

Publisher

CJ International
3, Rue Hector Berlioz
38600 FONTAINE
FRANCE
Phone: (0033) (0)4 76 26 87 30
Fax: (0033) (0)4 76 26 87 39

http://www.isagraf.com

Printed and bound in France.

CONTENTS

A. Introduction

This document describes the software interface and data exchange between **ISaGRAF Workbench** and **ISaGRAF Kernel**. Most of the information provided here concerns the **Workbench**.

The first chapter gives an overall view of **ISaGRAF**, particularly the architecture and how applications are generated. In addition, the components of the **Workbench** and the **Kernel** are precisely described.

The second chapter describes the interface between the **Workbench** and the **Kernel**.

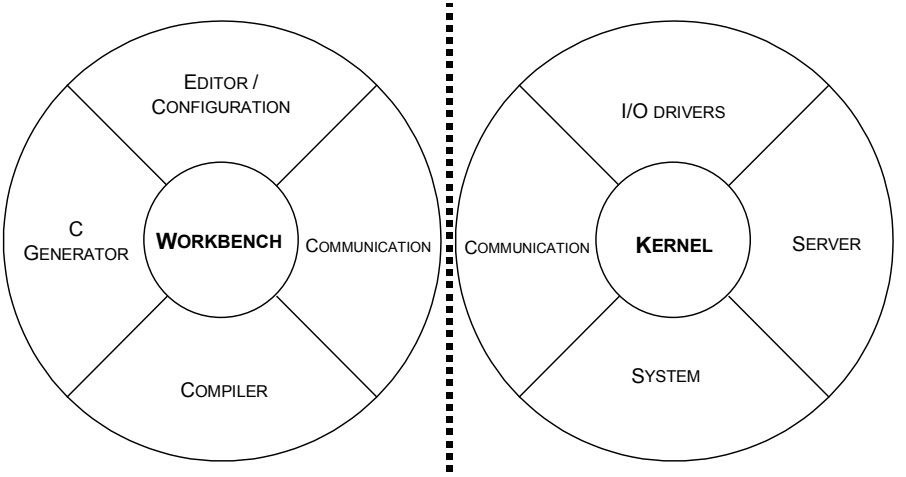
The remaining chapters display a detailed description of the interface elements (included as appendices).

B. ISaGRAF overview

This section gives a general view of the main components of **ISaGRAF**; their names, positions and structures.

B. 1. ISaGRAF architecture

ISaGRAF is made of two parts. The **Workbench**, which is located on the development system. Programs are developed with the 5 languages of the standard **IEC 1131-3**. The **Kernel**, which is located on the target system. Its role is to execute the applications generated by the **Workbench** in a special format.



Workbench

- Editor / Configuration — Used to write an application with the 5 standard languages and to connect the variables to the boards.
- Compiler — Used for translating the applications in a Database which will be directly executed by the Kernel.
- Communication — Used for communicating with the Kernel.
- C code generation — Translates the Database in C files.

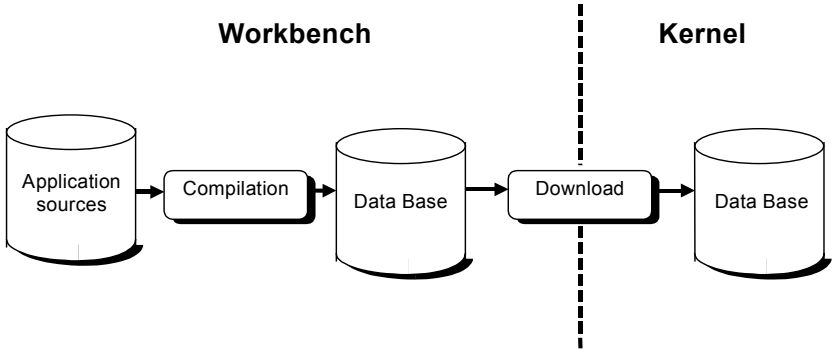
Kernel

- I/O drivers — Used for communicating with the I/O boards.
- System — Used for managing the system resources of the target.
- Communication — Used for communicating with the Workbench.
- Server — Used in a multi-task environment to communicate with other tasks.

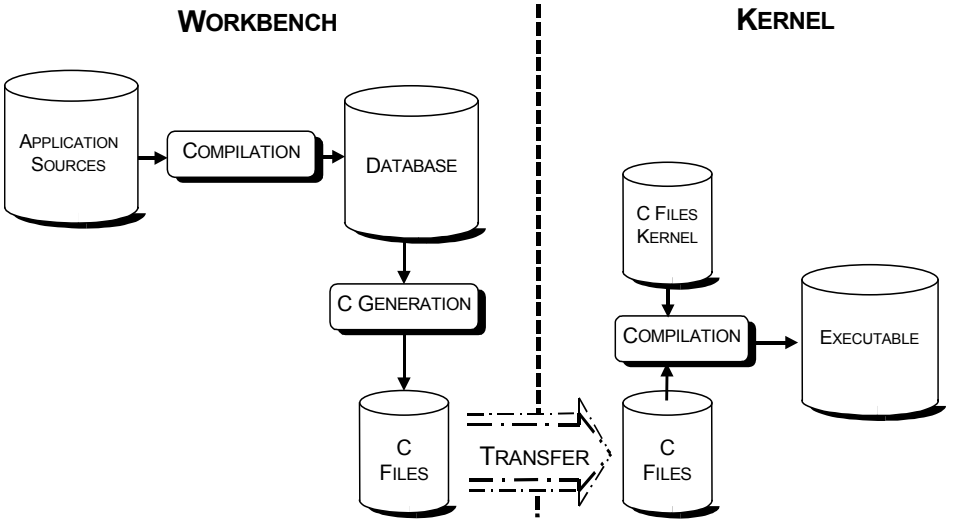
B. 2. Application Generation

Application Generation directly follows the writing of the application. This phase consists of translating the source code into a target code through a compilation process. The result of the compilation is a **Database** containing the data and the code of the application. From here there are two possible uses of the Database:

- The first choice is the standard solution, which consists of downloading the Database into the target and executing it directly from the Kernel.



- The second choice is to obtain a target system dependent code; this is generally more efficient. The **ISaGRAF-Compiler** produces **C code** from the **Database.** These C files are recompiled with the **Kernel** source in order to obtain an executable file.


WORKBENCH
KERNEL
APPLICATION SOURCES
COMPILATION
DATABASE
C GENERATION
C FILES
TRANSFER
C FILES KERNEL
COMPILATION
C FILES
EXECUTABLE

C. Workbench/Kernel Interface

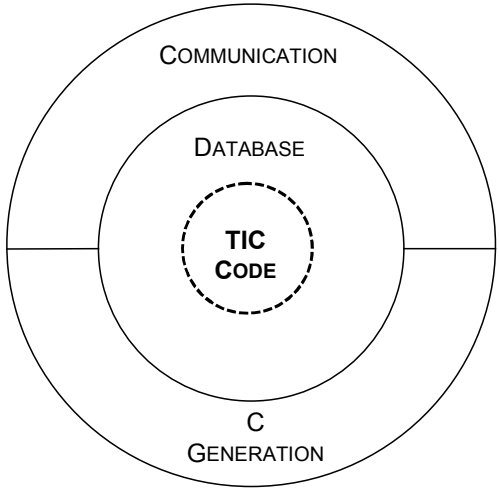
This section identifies the software and data elements providing the interface between the **Workbench** and the **Kernel**. The role of each (for the **Workbench** side only) is defined below.

A detailed description of these interfaces is given in Appendix 3.

C. 1. Interface description

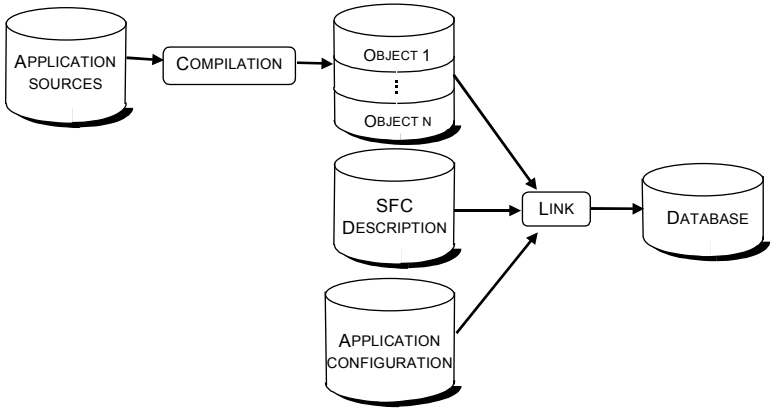
The interface between the **Workbench** and the **Kernel** is constituted of necessary specific database and communication systems to exchange data.

There are 4 parts in the interface organised in the following way:



C. 1. 1. Application Database

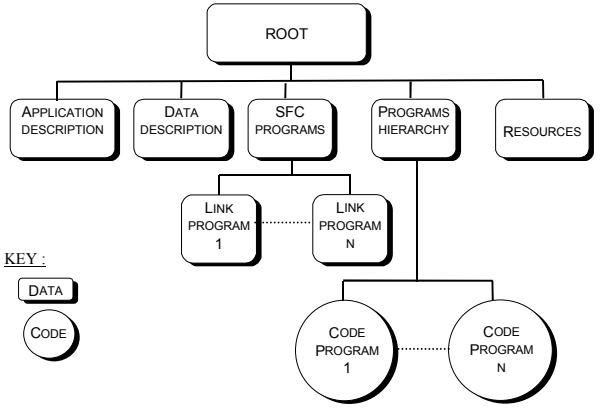
The **Application Database** is a file that contains the data and code of an application after compilation. This section describes briefly the structure and the format of the **Database**. A detailed description of the **Database** is given in Appendix 1.



C. 1. 1. 1. Database structure

An application **Database** is a set of tables producing a hierarchical system including application data and code.

- Data
  The data of an application concerns mainly: variables, initial values, I/O connections etc.
- Code
  The code of an application is the set of instructions forming the body of functions, function blocks, programs, steps and transitions.
  In the **Database,** the application code is written in a format defined by **CJ International**. This format is called **TIC code** (**T**arget **I**ndependent **C**ode). The detail of the TIC code is given in Appendix 2.

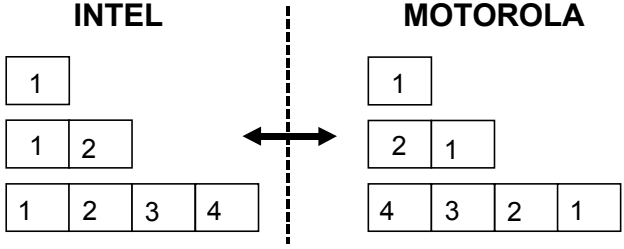


C. 1. 1. 2. Database format

The **Database** format depends on a target processor type and a chosen memory model.

- Processor type:

ISaGRAF can generate a **Database** for **INTEL** or **MOTOROLA** processors. For **MOTOROLA** processors the **HIGH** and **LOW** are switched.



- Memory model

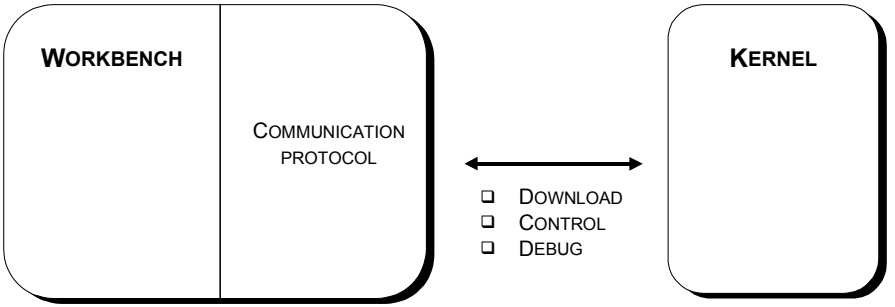
Actually, only the memory model is proposed on **ISaGRAF V3.x.** This model called **Medium** defines the code and operands (maximum 3) of TIC instructions on 16 bits.

Instruction	Size (bits)
code	16 bits
operand 1	16 bits
operand 2	16 bits
operand 3	16 bits

C. 1. 2. ISaGRAF Communication protocol

The **ISaGRAF** communication protocol links the **Workbench** to the **Kernel**. The communication has several roles in **ISaGRAF.**

- It is used to transfer files from the **Workbench** to the target. These files are compiled-codes or symbols of an application.
- The communication allows the **Workbench** to control and monitor the **Kernel** state (stop, run etc.).
- The communication allows the **Workbench Debugger** to monitor the application variables.



ISaGRAF Communication Protocol is a **CJ International** protocol based on **Modbus**. The principle of data exchange from the **Workbench** (master) and the **Kernel** (slave) is the same as **Modbus** but the function codes are different.

A detailed presentation of this protocol is described in Appendix 3.

C. 1. 3. C file generation

C file generation is a possibility offered by **ISaGRAF** to generate an application with a shorter execution time than the **TIC code**.

The principle of C files in an application is to translate the **Database** in C code. This process is defined in Appendix 4.

D. Appendix 1 : Database description

D. 1. Database description

This chapter describes all the tables forming the Database. For each table, the structure, name and size (in bytes) is defined.

D. 1. 1. Specification of the description

This section defines the notation used to describe the tables.

D. 1. 1. 1. Data definition

Generally the data are values (V), however, certain tables contain references to others tables. In this case there are two types of reference:

- a reference to the beginning of the Database. It is an absolute reference named : AR.
- a reference to the beginning of a table. It is a relative reference named : RR.

Mnemonic	**V**	**S**	**RR**	**AR**
Comment	numeric value	string of characters	relative reference	absolute reference

D. 1. 1. 2. Processor dependence

The format of the data depends on the processor on which it is executed. There are two formats : INTEL or MOTOROLA. In addition, two mnemonics classify the table contents: **PD** (processor dependent) and **PI** (processor independent).

Size	1 byte→1	2 bytes→12	4 bytes→1234
Intel format	1	12	1234
Motorola format	1	21	4321

D. 1. 1. 3. Type of data

data type	corresponding 'C' type	Size (bytes)	comments
uint16	unsigned short	2	
uint32	unsigned long	4	can contain an address
uchar	unsigned char	1	
charx	char t[x]	x	
floating point	floating point	4	

D. 1. 2. Table addresses

This is the main table, it is the entry point in the tables' hierarchy.

Name : __MAIN		Size : 64 bytes	
Field No.	**Format**	**Type**	**Comment**
1	uint32	V, PD	indicator of processor validity
2	uint32	AR, PD	address of definition table of the product
3	uint32	AR, PD	address of definition table of the application
4	uint32	AR, PD	address of definition table of the user functions
5	uint32	AR, PD	address of definition table of the function blocks
6	uint32	AR, PD	address of definition table of the conversion functions
7	uint32	AR, PD	address of definition table of conversion tables
8	uint32	AR, PD	address of definition table of variables
9	uint32	AR, PD	address of coding variables
10	uint32	AR, PD	address of definition table of boards definition
11	uint32	AR, PD	address of coding table of OEM
12	uint32	AR, PD	address of definition table of hierarchy project
13	uint32	AR, PD	address of definition table of programs
14	uint32	AR, PD	address of coding table of programs
15	uint32	AR, PD	address of structure SFCs table
16	uint32	AR, PD	address of resources table

Field No.	Notes
1.	It is a integer at 1(value set by the compiler). The kernel must read the same value else the Database and the Kernel are not in accordance. For example, if you compile an application for an INTEL processor and you try to run it on a MOTOROLA target.
2.	It is an absolute address of the definition table of the product (see chapter D.1.2.0). For example, the table being at address 00 00 00 64 is coded (for INTEL format) : 64 00 00 00
3.	It is an absolute address of the definition table of the application (see chapter D.1.2.D. 1. 2. 2.). For example, the table at address 00 0 00 40 is coded (for INTEL format) : 40 00 00 00
4.	It is an absolute address of the definition table of the user functions (see chapter D.1.2.1). For example, the table at address 00 00 02 B4 is coded (for INTEL format) : B4 02 00 00
5.	It is an absolute address of the definition table of the function blocks (see chapter D.1.2.D. 1. 2. 4.). For example, the table at address 00 0 02 B8 is coded (for INTEL format) : B8 02 00 00
6.	It is an absolute address of the definition table of the conversion functions (see chapter D.1.2.D. 1. 2. 5.). For example, the table at address 00 00 00 7C is coded (for INTEL format) : 7C 00 00 00
7.	It is an absolute address of the definition table of the conversion tables (see chapter D.1.2.0). For example, the table at address 00 00 00 80 is coded (for INTEL format) : 80 00 00 00
8.	It is an absolute address of the definition table of variables (see chapter D.1.2.D. 1. 2. 7.). For example, the table at address 00 00 03 FC is coded (for INTEL format) : FC 03 00 00
9.	It is an absolute address of coding variables (see chapter D.1.2.D. 1. 2. 8.). For example, the table at address 00 00 01 D4 is coded (for INTEL format) : D4 01 00 00
10.	It is an absolute address of the definition table of board definitions (see chapter D.1.2.D. 1. 2. 9.). For example, the table at address 00 00 00 84 is coded (for INTEL format) : 84 00 00 00
11.	It is an absolute address of the coding table of OEM (see chapter D.1.2.0). For example, the table at the address 00 00 00 8C is coded (for INTEL format) : 8C 00 00 00
12.	It is an absolute address of the definition table of the project hierarchy (see chapter D.1.2.0). For example, the table at address 00 00 02 98 is coded (for INTEL format) : 98 02 00 00
13.	It is an absolute address of the definition table of programs (see

chapter D.1.2.D. 1. 2. 12.). For example, the table at address 00 00 02 A8 is coded (for INTEL format) : A8 02 00 00

14. It is an absolute address of the coding table of programs (see chapter D.1.2.D. 1. 2. 13.). For example, the table at address 00 00 02 AC is coded (for INTEL format) : AC 02 00 00

15. It is an absolute address of the structure SFC's table (see chapter D.1.2.D. 1. 2. 14.). For example, the table at address 00 00 02 B0 is coded (for INTEL format) : B0 02 00 00

16. Is an absolute address of the resources table (see chapter D.1.2.D. 1. 2. 15.). For example, the table at address 00 00 04 30 is coded (for INTEL format) : 30 04 00 00

D. 1. 2. 1. Product definition

This table defines the kernel for which the application is generated.

Name : __DEFPROD			Size : 24 bytes
Field No.	**Format**	**Type**	**Comment**
1	char16	S, PI	target name
2	char8	S, PI	target version

Field No.	Notes
1.	Target name : It is a string of a maximum of 15 characters terminated by 0. The Kernel uses this information to verify if the Kernel and the Database are compatible. For example, a target named : "CC86M" is coded : 43 43 38 36 4D 00 00 00 00 00 00 00 00 00 00
2.	Target version : It is a string of a maximum of 7 characters terminated by 0. This information is not checked by the Kernel. For example, a target version named : "1.00" is coded : 31 2E 30 30 00 00 00

D. 1. 2. 2. Application definition

This table defines the parameters of the project.

Name : __DEFAPPLI			Size : 36 bytes
Field No.	**Format**	**Type**	**Comment**
1	char16	S, PI	project name
2	uint32	V, PD	generate date
3	uint32	V, PD	check sum
4	uint16	V, PD	application version
5	uint16	V, PD	number of error in the buffer
6	uint16	V, PD	starting mode
7	uint16	V, PD	cycle time
8	uint32	V, PD	application size

Field No.	Notes
1.	Project name : It is a string of a maximum of 15 characters terminated by 0. For example a project named : "rfwash" is coded : 72 66 77 61 73 68 00 00 00 00 00 00 00 00 00
2.	Date : It is the compilation date and time of the application. This information is not directly used by the kernel but is used by the Debugger of the Workbench. In association with the Check sum it identifies an application. For example a date is coded (for INTEL format) : A0 3A EE 32 The date is calculated using the standard "time" function of the "C" run-time library of MSVC 1.52.
3.	Check sum : It is a calculation in relation to the symbols of the application. It is not used directly by the Kernel but is used by the Debugger of the Workbench. In association with the Date it identifies an application. Another utilization is to compare this check sum with the check sum of the symbol table which can be download in the target. For example, a Check sum is coded (for INTEL format) : 5C 5F E9 0A
4.	Version : After each compilation of the application, the version number is increased. For example , after 5 compilations the version number is coded (for INTEL format) : 05 00.
5.	Error : This value defines the number of errors which can be printed. 0 indicates no printed errors. For example if you want to print 1 error message, this number is coded (for INTEL format) : 01 00
6.	Mode : This value indicates how the Kernel starts the application : 0 for real time and 1 for cycle to cycle. For example, the starting mode

is coded : 00 00

7. Cycle Time : It is the cycle time (ms) authorized for the application. If this time is exceeded, the kernel store it. A cycle time of 0 indicates that the Kernel does not manage it. For example, a cycle time of 10 ms is coded (for INTEL format) : 0a 00

8. Size : It is the size in bytes of the Database of the application. In fact, it is the size of all the tables describes in this document. For example a size of 2324 bytes is coded (for INTEL format) : 14 09 00 00

D. 1. 2. 3. Definition table of the user functions

This table defines the user functions written in ' C ' in the Kernel and called by the application.

Name : __DEFUSF			Size : 4+ (n*12) bytes		
		Field No.	Format	Type	Comment
header		1	uint16	V, PD	number of user functions
		2	uint16	V, PD	reserved
function description	repeated n times	3	uint16	V, PD	function number
		4	char9	S, PI	function name
		5	char1	S,PI	terminator

Field No.	Notes
1.	This field defined the number n of user functions called by the application. For example if the application calls one user function, this number is coded (for INTEL format) : 01 00
2.	This reserved field is coded : 00 00
3.	Function number : this number (value is 1 to n) is calculated by the compiler for an application and used by the TIC code. Therefore, if two applications call the same user function, it may have a different number. This number allows the user function to be found. For example, a user function numbered 1 is coded (for INTEL format) : 01 00
4.	Function name : It is a string of a maximum of 8 characters terminated by 0. This name is independent of the application (not the function number). Also it is used by the Kernel to find the code corresponding to the function. For example, the user function named "cfsample" is coded : 43 46 53 41 4D 50 4C 45 00
5.	Terminator : this character is used to align the table fields numbered 3 and 4 to 12 characters. This field is coded : 00

The fields 3,4 and 5 are repeated 'number of user functions (field 1)' times.

D. 1. 2. 4. Definition table of function blocks

This table defines the standard function blocks written in ' C ' in the Kernel (predefined list dependent on ISaGRAF) and the user function blocks written in ' C ' in the Kernel which the instance are declared in the application (list dependent on the application). The difference between standard and user function blocks is the name; prefixed for the standard blocks.

Name : __DEFFBL			**Size : 4+ (n*16) bytes**		
		Field No.	**Format**	**Type**	**Comment**
header		1	uint16	V, PD	number of functions block
		2	uint16	V, PD	reserved
function blocks description	**repeated n times**	3	uint16	V, PD	function block number
		4	uint16	V, PD	number of instance
		5	char10	S,PI	function block name
		6	uint16	V,PD	terminator

Field No.	Notes
1.	This field defines the number n of standard function blocks defined in ISaGRAF and user function blocks call by the application. For example, if the Database defines 20 function blocks, this number is coded (for INTEL format) : 14 00
2.	This reserved field is coded : 00 00
3.	Function number : this number (value 1 to n) is calculated by the compiler for an application and used by the TIC code. The standard blocks are always the same numbers because they are stored first in the table, however, it is different for the user blocks. In fact, if two applications call the same user block it may have different numbers. This number allows the name of the function block to be found. For example, a function block numbered 1 is coded (for INTEL format) : 01 00
4.	Number of instances. It is the number of instanciations of the function block numbered by the field 3. This information is used by the Kernel to manage the memory necessary for the data of the instances. For example, if the application instanciated a function block three times the number is (for INTEL format) : 03 00
5.	Function name : It is a string of a maximum of 9 characters terminated by 0. This name is independent of the application (not the function number). Also it is used by the Kernel to find the code corresponding to the function block. The name of standard blocks are prefixed by the character ' # '.For example, the standard block

named "#sig_gen" is coded : 23 53 49 47 5F 47 45 4E 00 00 and the user block named : "cbsample" is coded : 43 42 53 41 4D 50 4C 45 00 00

6. Terminator : this field is used to align the table fields numbered 3, 4 and 5 to 16 characters. This field is coded : 00 00

Fields 3 to 6 are repeated for each function block ('field 1' times).

D. 1. 2. 5. Conversion functions

This table contains the conversion functions declared in the application.

Name : __DEFFCNV			Size : 4+ (n*16) bytes		
		Field No.	Format	Type	Comment
header		1	uint16	V, PD	number of conv. functions
		2	uint16	V, PD	reserved
conversion function description	repeated n times	3	uint16	V, PD	conversion function number
		4	uint16	V, PD	reserved
		5	char10	S,PI	conversion function name
		6	uint16	V,PD	terminator

Field No.	Notes
1.	Number n of conversion functions. It is the number of conversion functions declared in the application. For example, if one conversion function is declared this number is coded (for INTEL format) : 01 00
2.	reserved
3.	Conversion function number. This number (between 128 and 255) is calculated by the linker. For example, a conversion function with the number 128 is coded (for INTEL format) : 81 00
4.	reserved
5.	Conversion function name. It is a string of a maximum of 9 characters terminated by 0. This name is independent of the application (not the function number). Also, it is used by the Kernel to find the code corresponding to the function. For example, a conversion function named "bcd" is coded : 42 43 44 00 00 00 00 00 00
6.	Terminator : this field is used to align the table fields numbered 3, 4 and 5 to 16 characters. This field is coded : 00 00

Fields 3 to 6 are repeated (field 1) times.

D. 1. 2. 6. Conversion tables

This table contains the conversion tables declared in the application.

Name : __DEFTCNV			Size : 4+ n*(4+p*8) bytes			
			Field No.	Format	Type	Comment
header			1	uint16	V, PD	number of conversion tables (**n**)
			2	uint16	V, PD	reserved
conversion table description	repeated n times		3	uint16	V, PD	conversion table number
			4	uint16	V, PD	number of points (**p**)
		repeated p times	5	floating point	V, PD	value from the device (x)
			6	floating point	V,PD	value for the application (y)

Field No.	Notes
1.	Number of conversion tables. It is the number of conversion tables declared in the application. For example, if one conversion table is declared, this number is coded (for INTEL format) : 01 00
2.	The field 2 is reserved.
3.	Conversion table number. This number (between 1 and 127) is calculated by the linker. For example, a conversion table with the number 1 is coded (for INTEL format) : 01 00
4.	Number of points. It is the number of points (x,y) defining the conversion between value reads from device (x) to value uses by the application(y). For example, a conversion table with 3 points is coded (for INTEL format) : 03 00
5.	Value from device. It is an ' electric value ' read from the device. For example, the 10.0 volts is coded (for INTEL format) : 00 00 20 41
6.	Value for application. It is a ' physical value ' used by the application. For example, the measure of 100.0 units is coded (for INTEL format) : 00 00 C8 42

Fields 5 and 6 are repeated (field 4) times. Fields 3 to 6 are repeated for each conversion table; (field 1) times.

D. 1. 2. 7. Definition table of variables

This table defines the number of variables for each type (Boolean, Analog, Timer and Message) declared in the application.

Name : __DEFVAR			Size : 32 bytes
Field No.	**Format**	**Type**	**Comment**
1	uint16	V, PD	number of internal Boolean
2	unit16	V, PD	number of input Boolean
3	unit16	V, PD	number of output Boolean
4	unit16	V, PD	reserved
5	unit16	V, PD	number of internal analog
6	unit16	V, PD	number of input analog
7	unit16	V, PD	number of output analog
8	unit16	V, PD	reserved
9	unit16	V, PD	number of timer
10	unit16	V, PD	reserved
11	unit16	V, PD	reserved
12	unit16	V, PD	reserved
13	unit16	V, PD	number of internal message
14	unit16	V, PD	number of input message
15	unit16	V, PD	number of output message
16	unit16	V, PD	reserved

Field No.	Notes
1.	Number of internal Boolean variables. For example, one internal Boolean declared in the application is coded (for INTEL format) : 01 00
2.	Number of input Boolean variables connected to a board. For example, one input Boolean declared in the application and connected to a board with 8 inputs is coded (for INTEL format) : 08 00
3.	Number of output Boolean variables connected to a board. For example, one output Boolean declared in the application and connected to a board with 8 outputs is coded (for INTEL format) : 08 00
4.	reserved
5.	Number of internal analogs. For example, two internal analogs declared in the application is coded (for INTEL format) : 02 00
6.	Number of input analog connected to a board. For example, two input analogs declared in the application and connected to a board with 8 inputs is coded (for INTEL format) : 08 00
7.	Number of output analog connected to a board. For example, two output analog declared in the application and connected to a board with 8 outputs is coded (for INTEL format) : 08 00
8.	reserved
9.	Number of internal timer. For example, ten internal timers declared in the application is coded (for INTEL format) : 0A 00
10.	reserved
11.	reserved
12.	reserved
13.	Number of internal messages. For example, one internal message declared in the application is coded (for INTEL format) : 01 00
14.	Number of input messages connected to a board. For example, one input message declared in the application and connected to a board with 8 inputs is coded (for INTEL format) : 08 00
15.	Number of output messages connected to a board. For example, one output message declared in the application and connected to a board with 8 outputs is coded (for INTEL format) : 08 00
16.	reserved

D. 1. 2. 8. Variables coding

This table defines the location of the initialization tables for each type and the number of variables to be initialized.

Name : __CODEVAR			Size : 36 bytes
Field No.	**Format**	**Type**	**Comment**
1	uint32	AR, PD	address of initialization Boolean table
2	uint32	AR, PD	address of initialization analog
3	uint32	AR, PD	address of initialization timer table
4	uint32	AR, PD	address of initialization message table of initialization timer table
5	uint32	AR, PD	address of length message table
6	uint32	AR, PD	address of definition MODBUS table
7	uint32	AR, PD	address of backup table
8	uint16	V, PD	number of Boolean variable to be initialized
9	uint16	V, PD	number of analog variable to be initialized
10	uint16	V, PD	number of timer variable to be initialized
11	uint16	V, PD	number of message variable to be initialized

Field No.	Notes
1.	It is an absolute address of initialization Boolean table(see chapter D.1.2.8.D. 1. 2. 8. 1.). For example, the table at address 00 00 08 42 is coded (for INTEL format) : 42 08 00 00
2.	It is an absolute address of initialization analog (see chapter D.1.2.8.D. 1. 2. 8. 2.). For example, the table at address 00 00 02 4C is coded (for INTEL format) : 4C 02 00 00
3.	It is an absolute address of initialization timer table (see chapter D.1.2.8.0). For example, the table at address 00 00 02 50 is coded (for INTEL format) : 50 02 00 00
4.	It is an absolute address of initialization message table (see chapter D.1.2.8.0). For example, the table at address 00 00 02 54 is coded (for INTEL format) :54 02 00 00
5.	It is an absolute address of length message table (see chapter D.1.2.8.D. 1. 2. 8. 5.). For example, the table at address 00 00 04 1C is coded (for INTEL format) : 1C 04 00 00
6.	It is an absolute address of definition MODBUS table (see chapter D.1.2.8.D. 1. 2. 8. 6.) For example, the table at address 00 00 02 58 is coded (for INTEL format) : 58 02 00 00
7.	It is an absolute address of definition table of conversion tables (see

chapter D.1.2.8.0) For example, the table at address 00 00 01 F8 is coded (for INTEL format) : F8 01 00 00

8. Number of Boolean variable to be initialized
9. Number of analog variable to be initialized
10. Number of timer variable to be initialized
11. Number of message variable to be initialized

D. 1. 2. 8. 1. *Boolean initialization table*

This table is the list of Boolean variables to be set to TRUE. The number (**n**) of elements is defined in field 8 of the table __CODEVAR.

Name : __INITBOO			**Size : (n*2) + 4 - (n*2 % 4) bytes**		
		Field No.	**Format**	**Type**	**Comment**
Boolean description	**repeated n times**	1	uint16	V, PD	variable number
		n+1	charx	S,PI	terminator

Field No.	Notes
1.	Variable number. It is the variable number to be set at TRUE. For example, the Boolean numbered 9 is coded (for INTEL format) : 09 00
n+1.	Terminator. It is a string of x null value. x is : 4 - (n*2 % 4). For example, if there are three Boolean variables to be initialized, the length of the terminator is : 4 - (3*2 % 4) = 2 and it is coded : 00 00

NOTE : % gives the integer remainder of a division. For example, 1%2=1

D. 1. 2. 8. 2. *Analog initialization table*

This table is the list of analog variables to be initialized with a value. The number (**n**) of elements is defined in field 9 of the table __CODEVAR.

Name : __INITANA			**Size : (n*6) + 4 - (n*6 % 4) bytes**		
		Field No.	**Format**	**Type**	**Comment**
analog description	**repeated n times**	1	uint16	V, PD	variable number
		2	uint32 / floating point	V, PD	value
		n*2+1	charx	S,PI	terminator

Field No.	Notes
1.	Variable number. It is the variable number to be set initialized with the value contented in the field 2. For example, the Analog numbered 5 is coded (for INTEL format) : 05 00
2.	value. It is the value (integer or floating pointing point) assigned to the variable defined by the field 1. For example, an integer value 10 is coded (for INTEL format) : 0A 00 and a real value 10.0 is coded (for INTEL format) : 00 00 20 41.
n*2+1.	Terminator. It is a string of x null value. x is : 4 - (n*6 % 4). For example, if there are three analogs to be initialized, the length of terminator is : 4 - (3*6 % 4) = 2 and it is coded : 00 00

NOTE : % gives the integer remainder of a division. For example, 1%2=1

D. 1. 2. 8. 3. *Timer initialization table*

This table is the list of timer variables to be initialized with a value. The number (**n**) of elements is defined in field 10 of the table __CODEVAR.

Name : __INITTMR			**Size : (n*6) + 4 - (n*6 % 4) bytes**		
		Field No.	**Format**	**Type**	**Comment**
timer description	**repeated n times**	1	uint16	V, PD	variable number
		2	uint32	V, PD	value
		n*2+1	charx	S,PI	terminator

Field No.	Notes
1.	Variable number. It is the variable number to be set initialized with the value contented in the field 2. For example, the timer numbered 4 is coded (for INTEL format) : 04 00
2.	value. It is a integer value (millisecond) affected to the variable defined by the field 1. For example a value 10 ms is coded (for INTEL format) : 0A 00.
n*2+1.	Terminator. It is a string of x null characters. x is : 4 - (n*6 % 4). For example, if there are three timers to be initialized, the length of terminator is : 4 - (3*6 % 4) = 2 and it is coded : 00 00

NOTE : % gives the integer remainder of a division. For example, 1%2=1

D. 1. 2. 8. 4. *Message initialization table*

This table is the list of message variables to be initialized with a string of characters. The number (**n**) of elements is defined in field 11 of the table __CODEVAR.

Name : __INITMSG		**Size : (i=1,n)∑3+li+2-((1+l_i)%2) +(4-(∑(5+l_i-(1+li)%2))%4) bytes**			
		Field No.	**Format**	**Type**	**Comment**
message description	**repeated n times**	1	uint16	V, PD	variable number
		2	uchar	V, PI	length
		3	charx	S, PI	string
		4	uchary	S, PI	extension
		n*4+1	charx	S,PI	terminator

Field No.	Notes
1.	Variable number. It is the variable number to be set initialized with the value contented in the field 2. For example, the message numbered 4 is coded (for INTEL format) : 04 00
2.	Length. It is the length l (0-255) of the message defined in the field 3. For example the length of the message "aaa" is 3 and is coded : 03
3.	String. It is a string of x characters. For example the message "aaa" is coded : 61 61 61.
4.	Extension. It is a string of x null characters. It is an extension at an even length of the fields 2 and 3 because MOTOROLA uses even addresses. Also x = 2-((1+l)%2). For example, the message "aaa" has an extension of x=2-((1+3)%2)=2 characters, but the message "aaaa" has an extension of x=2-((1+4)%2)=1 character coded : 00
n*4+1.	Terminator. It is a string of x null characters. x is : 4 - (∑ (5+li-(1+li)%2))%4 with 1≤i≤n. For example, if there are two messages "aa" and "bbb" to be initialized, the length of terminator is 2 and it is coded : 00 00

NOTE : % gives the integer remainder of a division. For example, 1%2=1

D. 1. 2. 8. 5. *Message length table*

This table contains the maximum length of the declared message variable. Notably, this information allows the Kernel to reserve the necessary memory used to store the messages.

Name : __LNGMSG			**Size : n+(4-n%4) bytes**		
		Field No.	**Format**	**Type**	**Comment**
length description	**repeated n times**	1	uchar	V, PI	message length
		n+1.	charx	S, PI	terminator

Field No.	**Notes**
1.	Message length. It is the maximum length of the message variable. The message number corresponds to the order in the table (the second message length is the number variable two). For example, a message declared with a maximum length of 20 characters is coded: 14
n+1.	Terminator. It is a string of **x** null value. x is : 4 - (n%4). For example, if there are three messages the length of terminator is : 4 - (3%4) = 1 and it is coded : 00

NOTE : % gives the integer remainder of a division. For example, 1%2=1

D. 1. 2. 8. 6. *MODBUS definition table*

This table contains the reference to the MODBUS tables for each type.

Name : __DEFMODBUS			Size : 16 bytes
Field No.	**Format**	**Type**	**Comment**
1	uint32	AR, PD	address of Boolean MODBUS definition table
2	uint32	AR, PD	address of analog MODBUS definition table
3	uint32	AR, PD	address of timer MODBUS definition table
4	uint32	AR, PD	address of message MODBUS definition table

Field No.	Notes
1.	It is an absolute address of the Boolean MODBUS definition table (see chapter D.1.2.8.6.D. 1. 2. 8. 7.). For example, the table at address 00 00 02 68 is coded (for INTEL format) : 68 02 00 00
2.	It is an absolute address of the analog MODBUS definition table (see chapter D.1.2.8.6.D. 1. 2. 8. 8.). For example, the table at address 00 00 02 74 is coded (for INTEL format) : 74 02 00 00
3.	It is an absolute address of the timer MODBUS definition table (see chapter D.1.2.8.6.0). For example, the table at address 00 00 02 80 is coded (for INTEL format) : 80 02 00 00
4.	It is an absolute address of the message MODBUS definition table (see chapter D.1.2.8.6.0). For example, the table at address 00 00 02 8C is coded (for INTEL format) : 8C 02 00 00

D. 1. 2. 8. 7. *Boolean MODBUS table definition*

This table defines all the Boolean variables which have a MODBUS address.

Name : __DEFMDBBOO			**Size : 12+n*4 bytes**		
		Field No.	**Format**	**Type**	**Comment**
header		1	uint16	V, PD	number
		2	uint16	V, PI	lowest address
		3	uint16	V, PD	highest address
		4	uint16	V, PD	reserved
message description	**repeated n times**	5	uint16	V, PD	address
		6	uint16	V, PD	variable number
		n*2+5	uint32	V, PI	terminator

Field No.	**Notes**
1.	Number. It is the number of Boolean variables which have a MODBUS address. For example, two Boolean variables with an MODBUS address are coded (for INTEL Format) :02 00
2.	Lowest address. This information indicates which has the lowest address of all Boolean variables which are a MODBUS address. For example, the address 1000 (hexa) is coded (for INTEL format) : 00 10
3.	Highest address. This information indicates which is the highest address of all Boolean which are a MODBUS address. For example, the address 2000 (hexa) is coded (for INTEL format) : 00 20
4.	Reserved
5.	Address. It is the MODBUS address of the Boolean variable numbered in the field 6. For example, the address 1000 (hexa) is coded (for INTEL format) : 00 10
6.	Variable number. It is the Boolean number. For example, the Boolean numbered 4 is coded (for INTEL format) : 04 00
n*2+5.	Terminator. Is a null value coded : 00 00 00 00

D. 1. 2. 8. 8. *Analog MODBUS table definition*

This table defines all the analog variables which have a MODBUS address.

Name : __DEFMDBANA			**Size : 12+n*4 bytes**		
		Field No.	**Format**	**Type**	**Comment**
header		1	uint16	V, PD	number
		2	uint16	V, PI	lowest address
		3	uint16	V, PD	highest address
		4	uint16	V, PD	reserved
message description	**repeated n times**	5	uint16	V, PD	address
		6	uint16	V, PD	variable number
		n*2+5	uint32	V, PI	terminator

Field No.	**Notes**
1.	Number. It is the number of Boolean variables which have a MODBUS address. For example, two analogs with a MODBUS address are coded (for INTEL Format) :02 00
2.	Lowest address. This information indicates which is the lowest address of all analogs which are a MODBUS address. For example, the address 1000 (hexa) is coded (for INTEL format) : 00 10
3.	Highest address. This information indicates which is the highest address of all analogs which are a MODBUS address. For example, the address 2000 (hexa) is coded (for INTEL format) : 00 20
4.	Reserved
5.	Address. It is the MODBUS address of the analog variable numbered in the field 6. For example, the address 1000 (hexa) is coded (for INTEL format) : 00 10
6.	Variable number. It is the analog number. For example, the analog numbered 4 is coded (for INTEL format) : 04 00
n*2+5.	Terminator. Is a null value coded : 00 00 00 00

D. 1. 2. 8. 9. *Timer MODBUS table definition*

This table defines all the timer variables which have an MODBUS address.

Name : __DEFMDBTMR		**Size : 12+n*4 bytes**			
		Field No.	**Format**	**Type**	**Comment**
header		1	uint16	V, PD	number
		2	uint16	V, PI	lowest address
		3	uint16	V, PD	highest address
		4	uint16	V, PD	reserved
message description	**repeated n times**	5	uint16	V, PD	address
		6	uint16	V, PD	variable number
		n*2+5	uint32	V, PI	terminator

Field No.	**Notes**
1.	Number. It is the number of timer variable which have a MODBUS address. For example, two timers with an MODBUS address are coded (for INTEL Format) :02 00
2.	Lowest address. This information indicates which is the lowest address of all timers which are a MODBUS address. For example, the address 1000 (hexa) is coded (for INTEL format) : 00 10
3.	Highest address. This information indicates which is the highest address of all timers which are a MODBUS address. For example, the address 2000 (hexa) is coded (for INTEL format) : 00 20
4.	Reserved
5.	Address. It is the MODBUS address of the timer variable numbered in the field 6. For example, the address 1000 (hexa) is coded (for INTEL format) : 00 10
6.	Variable number. It is the timer number. For example, the timer numbered 4 is coded (for INTEL format) : 04 00
n*2+5.	Terminator. Is a null value coded : 00 00 00 00

D. 1. 2. 8. 10. *Message MODBUS table definition*

This table defines all the message variables which have a MODBUS address.

Name : __DEFMDBMSG		Size : 12+n*4 bytes			
		Field No.	**Format**	**Type**	**Comment**
header		1	uint16	V, PD	number
		2	uint16	V, PI	lowest address
		3	uint16	V, PD	highest address
		4	uint16	V, PD	reserved
message description	**repeated n times**	5	uint16	V, PD	address
		6	uint16	V, PD	variable number
		n*2+5	uint32	V, PI	terminator

Field No.	Notes
1.	Number. It is the number of message variable which have a MODBUS address. For example, two messages with an MODBUS address are coded (for INTEL Format) :02 00
2.	Lowest address. This information indicates which is the lowest address of all messages which are a MODBUS address. For example, the address 1000 (hexa) is coded (for INTEL format) : 00 10
3.	Highest address. This information indicates which is the highest address of all messages which are a MODBUS address. For example, the address 2000 (hexa) is coded (for INTEL format) : 00 20
4.	Reserved
5.	Address. It is the MODBUS address of the message variable numbered in the field 6. For example, the address 1000 (hex) is coded (for INTEL format) : 00 10
6.	Variable number. It is the message number. For example, the message numbered 4 is coded (for INTEL format) : 04 00
n*2+5.	Terminator. Is a null value coded : 00 00 00 00

D. 1. 2. 8. 11. *Backup table*

This table contains all the information to be saved by the Kernel.

Name : __BACKUP			Size : 80 bytes
Field No.	**Format**	**Type**	**Comment**
1	char64	S, PI	memory area
2	uint16	V, PD	number of Boolean
3	uint16	V, PD	first Boolean
4	uint16	V, PD	number of Analog
5	uint16	V, PD	first Analog
6	uint16	V, PD	number of Timers
7	uint16	V, PD	first Timer
8	uint16	V, PD	number of Message
9	uint16	V, PD	first Message

Field No. Notes

1. Memory area. It contents the backup's parameters in Ascii mode : address and size for each type. For example, you want to save the Boolean at the address 1000 and reserved an area of 16 bytes for it. It is coded :

   31 30 30 30 2C 31 36 00 - 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00

2. Number of Boolean variables. This is the number of Boolean variables to be saved. One Boolean to be saved is coded (for INTEL format) : 01 00
3. First Boolean. This is the number of the first Boolean variables to be saved. The Boolean numbered 9 to be saved is coded (for INTEL format) : 09 00
4. Number of analog. This is the number of analog variables to be saved. Two analogs to be saved is coded (for INTEL format) : 02 00
5. First analog. This is the number of the first analog variables to be saved. The analog numbered 10 to be saved is coded (for INTEL format) : 0A 00
6. Number of timer. This is the number of timer variables to be saved. Five timers to be saved is coded (for INTEL format) : 05 00

7. First timer. This is the number of the first timer variables to be saved. The timer numbered 16 to be saved is coded (for INTEL format) : 10 00

8. Number of message. This is the number of message variables to be saved. One message to be saved is coded (for INTEL format) : 01 00

9. First message. This is the number of the first message variables to be saved. The Boolean numbered 8 to be saved is coded (for INTEL format) : 08 00

NOTE : The linker arranges all the variables to be saved from the first variable number.

D. 1. 2. 9. Boards definition

This table contains the information concerning the number of input/output boards or the number of real variables.

Name : __DEFOEM			Size : 8 bytes
Field No.	**Format**	**Type**	**Comment**
1	uint16	V, PD	Number of inputs board
2	uint16	V, PD	Number of outputs board
3	uint16	V, PD	Number of analog variables (floating point, convert)
4	uint16	V, PD	Reserved

Field No.	Notes
1.	Number of inputs board. It is the total number of inputs board (Boolean, analog and message). For example 3 inputs board are coded (for INTEL format) : 03 00
2.	Number of outputs board. It is the total number of outputs board (Boolean, analog and message). For example 2 outputs board are coded (for INTEL format) : 02 00
3.	Number of variable to convert. It is the number of input/output analog variables to convert in real. For example, 1 variable to convert is coded (for INTEL format) : 01 00
4.	Reserved.

D. 1. 2. 10. OEM table coding

This table contains the references to the table OEM.

Name : __CODEOEM			Size : 16 bytes
Field No.	**Format**	**Type**	**Comment**
1	uint32	AR, PD	Address of Boolean table OEM
2	uint32	AR, PD	Address of Analog table OEM
3	uint32	AR, PD	Address of Message table OEM
4	uint32	AR, PD	Address of I/O boards definition

Field No.	Notes
1.	Address of Boolean table OEM (see the chapter D.1.2.10.D. 1. 2. 10. 1.). For example, the table at address 00 00 00 D0 is coded (for INTEL format) : D0 00 00 00
2.	Address of analog table OEM (see the chapter D.1.2.10.D. 1. 2. 10. 2.). For example, the table at address 00 00 01 10 is coded (for INTEL format) : 10 01 00 00
3.	Address of message table OEM (see the chapter D.1.2.10.D. 1. 2. 10. 3.). For example, the table at address 00 00 01 D0 is coded (for INTEL format) : D0 01 00 00
4.	Address of I/O boards definition (see the chapter D.1.2.10.D. 1. 2. 10. 4.). For example, the table at address 00 00 00 9C is coded (for INTEL format) : D0 00 00 00

D. 1. 2. 10. 1. *Boolean table (OEM)*

This table contains the information necessary to connect the Boolean to the input or output Boolean board.

Name : __DEFBOO		**Size : n*8 bytes**			
		Field No.	**Format**	**Type**	**Comment**
Boolean description	**repeated n times**	1	uint16	V, PD	number
		2	uchar	V, PI	rack
		3	uchar	V, PI	slot
		4	uchar	V, PI	channel
		5	uchar	V, PI	I/O
		6	uchar	V, PI	conversion/length
		7	uchar	V, PI	real

Field No.	Notes
1.	Number. It is the Boolean variable number connected. For example, the variable numbered 1is coded (for INTEL Format) :01 00
2.	Rack number. It is the rack number (0-255) where a set of boards are plugged. A PLC can have several racks. For example, a rack numbered 2 is coded : 02
3.	Slot number. It is the slot number (0 to 255) where the board is plugged. For example, a slot numbered 1 is coded : 01
4.	Channel number. It is the channel number (0 to 255) where the variable is connected. For example, a channel numbered 1 is coded : 01
5.	I/O. It is an indicator of input board (value equal 0) or output board (value different 0). For example an output board is coded : 01
6.	Conversion/length. It is a null value coded : 00
7.	Real. It is a null value coded : 00

NOTE : n is defined in the table __DEFEOM

D. 1. 2. 10. 2. *Analog table (OEM)*

This table contains the information necessary to connect the analog variables to the input or output analog board.

Name : __DEFANA		**Size : n*8 bytes**			
		Field No.	**Format**	**Type**	**Comment**
analog description	**repeated n times**	1	uint16	V, PD	number
		2	uchar	V, PI	rack
		3	uchar	V, PI	slot
		4	uchar	V, PI	channel
		5	uchar	V, PI	I/O
		6	uchar	V, PI	conversion/length
		7	uchar	V, PI	real

Field No.	Notes
1.	Number. It is the analog variable number connected. For example, the variable numbered 1is coded (for INTEL Format) :01 00
2.	Rack number. It is the rack number (0-255) where a set of boards are plugged. A PLC can have several racks. For example, a rack numbered 2 is coded : 02
3.	Slot number. It is the slot number (0 to 255) where the board is plugged. For example, a slot numbered 1 is coded : 01
4.	Channel number. It is the channel number (0 to 255) where the variable is connected. For example, a channel numbered 1 is coded : 01
5.	I/O. It is an indicator of input board (value equal 0) or output board (value different 0). For example an output board is coded : 01
6.	Conversion/length. It is the function or table conversion number used to convert the analog (see table __DEFFCNV and __DEFTCNV). For example a conversion function numbered 81 (hexa) is coded: 81
7.	Real. It is a null value if the variable is an analog or a none null value if the variable is a real. For example, a real is coded : 01

NOTE : n is defined in the table __DEFEOM

D. 1. 2. 10. 3. *Message table (OEM)*

This table contains the information necessary to connect the message to the input or output analog board.

Name : __DEFMSG		Size : n*8 bytes			
		Field No.	Format	Type	Comment
message description	repeated n times	1	uint16	V, PD	number
		2	uchar	V, PI	rack
		3	uchar	V, PI	slot
		4	uchar	V, PI	channel
		5	uchar	V, PI	I/O
		6	uchar	V, PI	conversion/length
		7	uchar	V, PI	real

Field No.	Notes
1.	Number. It is the message variable number connected. For example, the variable numbered 1is coded (for INTEL Format) :01 00
2.	Rack number. It is the rack number (0-255) where a set of boards are plugged. A PLC can have several racks. For example, a rack numbered 2 is coded : 02
3.	Slot number. It is the slot number (0 to 255) where the board is plugged. For example, a slot numbered 1 is coded : 01
4.	Channel number. It is the channel number (0 to 255) where the variable is connected. For example, a channel numbered 1 is coded : 01
5.	I/O. It is an indicator of input board (value equal 0) or output board (value different 0). For example an output board is coded : 01
6.	Conversion/length. It is a the length of the message (see the table __LNGMSG). For example a input message declares with a maximum 100 (decimal) is coded : 64
7.	Real. It is a null value coded : 00

NOTE : n is defined in the table __DEFEOM

D. 1. 2. 10. 4. *I/O boards definition*

This table defines all the I/O boards numbered from 1 to n.

Name : __DEFBRD		**Size : 12*n + (4 - (Σ(12+x_i))%4) bytes 1≤i≤n**			
		Field No.	**Format**	**Type**	**Comment**
board description	**repeated n times**	1	uint16	V, PD	size
		2	uint16	V, PD	OEM key
		3	uint16	V, PD	first variable
		4	uchar	V, PI	rack
		5	uchar	V, PI	slot
		6	uchar	V, PI	mode
		7	uchar	V, PI	type
		8	uchar	V, PI	I/O
		9	uchar	V, PI	number of connected variable
		10	charx	S, PI	OEM board parameters
		11	chary	S, PI	terminator

Field No.	Notes
1.	Size. It is the size in bytess of the OEM board description (field 1 to 10). For example, a description size of 12 is coded (for INTEL format) : 0C 00
2.	OEM key. It is the hardware supplier number (different for each supplier). For example, hardware supplier with the number 1000 (hexa) is coded (for INTEL format) : 00 10
3.	First variable. It is the number of the first variable connected. For example, the variable numbered 9 connected to the board is coded (for INTEL format) : 09 00.
4.	Rack number. It is the rack number (0-255) where a set of boards are plugged. A PLC can have several racks. For example, a rack numbered 2 is coded : 02
5.	Slot number. It is the slot number (0 to 255) where the board is plugged. For example, a slot numbered 1 is coded : 01
6.	Mode. It indicates if the board is in real mode (value equal to 1) or in simulation mode (null value). For example, a board in real mode is coded : 01
7.	Type. It is the type of the variables connected to the board. The type

value is 1 for Boolean, 2 for analog and 3 for message. For example, a message board is coded : 03

8. I/O. The value of this field indicated if the board is an input board (value equal to 0) or an output board (value equal to 1). For example, an output board is coded : 01

9. Number of connected variable. It is the number of possible connection on the board (0 to 255). For example, a board of 8 inputs is coded : 08

10. OEM board parameters. The contents depends of the configuration made by the OEM supplier. The size x of this field is : x=size-12.

11. Terminator. It is a string of y null characters. y is : 4 - (Σ(12+xi))%4 with $1 \leq i \leq n$. For example, if a length of terminator of 2 is coded : 00 00

NOTE : The rack and slot number (field 4 and 5) allow to link the I/O variables define in the tables : __DEFBOO, __DEFANA, __DEFMSG.

D. 1. 2. 11. Hierarchy project definition table

This table contains all the programs, functions and function blocks and defines the SFC hierarchy.

Name : __DEFHIE			Size : 16+n*4 bytes		
		Field No.	Format	Type	Comment
header		1	uint16	V, PD	number of BEGIN programs
		2	uint16	V, PD	number of END programs
		3	uint16	V, PD	number of SFC main
		4	uint16	V, PD	number of SFC child
		5	uint16	V, PD	number of functions or blocks
		6	uint16	V, PD	reserved
SFC description	repeated n times	7	uint16	V, PD	SFC number
		8	uint16	V, PD	level
		n*2+7	char4	V, PI	terminator

Field No.	Notes
1.	Number of BEGIN program. For example, an application with two begin programs is coded (for INTEL format) : 02 00
2.	Number of END program. For example, an application with three end programs is coded (for INTEL format) : 03 00
3.	Number of SFC main. SFC main is the SFC having the highest level in the hierarchy. For example, an application with three SFC main is coded (for INTEL format) : 03 00
4.	Number of SFC child. A SFC child depends of a SFC main. For example, an application with five SFC child is coded (for INTEL format) : 05 00
5.	Number of function or IEC function blocks. For example, an application with three functions is coded (for INTEL format) : 03 00
6.	Reserved
7.	SFC number. For example a SFC numbered 10 is coded (for INTEL format) : 0A 00
8.	Level. It is the hierarchy level number of a SFC. Level 0 identifies a main SFC and the other value (1..) are decreasing level for SFC child. For a SFC child with the level number 2 is coded (for INTEL format) : 02 00. Also, it indicates which there are two SFC above in the SFC hierarchy.

n*2+7. Terminator. It is string of 4 null value coded : 00 00 00 00

D. 1. 2. 12. Programs definition table

This table defines each program of an application. The programs are numbered from 1 to n.

Name : __DEFPROG			Size : n*4 bytes		
		Field No.	Format	Type	Comment
program description	repeated n time	1	uint16	V, PD	program type
		2	uint16	V, PD	level

Field No.	Notes
1.	Program type. The programs type are identified by a number : 1 for BEGIN, 2 for END, 3 for SFC main, 4 for SFC child and 5 for function or IEC function block. For example a SFC main is coded (for INTEL format) : 03 00
2.	Level. It is the level of the SFC hierarchy. Also, the programs with the type 1,2,3 and 5 have a level numbered 0. A SFC child (type value : 1 to n) numbered 2 for example is coded (for INTEL format) : 02 00

NOTE : The number of programs is calculated with the table __DEFHIE (it is the total of the fields of 1 to 5).

D. 1. 2. 13. Coding programs table

This table contains the address of all the program sequences (code of programs, SFC, function, IEC function blocks).

Name : __CODEPROG			**Size : n*4 bytes**	
	Field No.	**Format**	**Type**	**Comment**
repeated n time	1	uint32	AR, PD	Address of program sequence

Field No.	Notes
1.	Address of program sequence. It is the absolute address of the programs code. For example, the table at address 00 00 00 D0 is coded (for INTEL format) : D0 00 00 00

NOTE : The number of programs is calculated with the table __DEFHIE (it is the total of the fields of 1 to 5).

D. 1. 2. 13. 1. *Program sequence*

This table contains some information in relation to the code of the application.

<table>
<tr><td colspan="5">Name : __ProgName_CODE</td><td colspan="4">Size : 12*m+Σ(s1_k+s2_k+s3_k)
+4*n Σ(s1_l) +x bytes 1≤k≤m, 1≤l≤n</td></tr>
<tr><td colspan="5"></td><td>Field</td><td>Fmt</td><td>Type</td><td>Comment</td></tr>
<tr><td rowspan="6">Step SFC or non SFC desc.</td><td rowspan="6">Rept'd m times</td><td rowspan="2">BEGIN action or non SFC program</td><td colspan="2"></td><td>1</td><td>uint16</td><td>RR, PD</td><td>address block</td></tr>
<tr><td colspan="2"></td><td>2</td><td>uint16</td><td>V, PD</td><td>size block (s1)</td></tr>
<tr><td rowspan="2">COMPLEX action</td><td colspan="2"></td><td>3</td><td>uint16</td><td>RR, PD</td><td>address block</td></tr>
<tr><td colspan="2"></td><td>4</td><td>uint16</td><td>V, PD</td><td>size block (s2)</td></tr>
<tr><td rowspan="2">END action</td><td colspan="2"></td><td>5</td><td>uint16</td><td>RR, PD</td><td>address block</td></tr>
<tr><td colspan="2"></td><td>6</td><td>uint16</td><td>V, PD</td><td>size block (s3)</td></tr>
<tr><td colspan="3">transition SFC description</td><td>repeated n times</td><td>7</td><td>uint16</td><td>RR, PD</td><td>address block</td></tr>
<tr><td colspan="3"></td><td colspan="2"></td><td>8</td><td>uint16</td><td>V, PD</td><td>size block (t1)</td></tr>
<tr><td rowspan="3">block desc.</td><td rowspan="3">rept'd i times</td><td rowspan="3">TIC code desc.</td><td colspan="2"></td><td>9</td><td>uint16</td><td>V, PD</td><td>TIC code</td></tr>
<tr><td>rept'd p time</td><td></td><td>10</td><td>uint16</td><td>V, PD</td><td>parameter</td></tr>
<tr><td colspan="3"></td><td>11</td><td>charx</td><td>S, PI</td><td>terminator</td></tr>
</table>

Field No.	Notes
1.	Address block. It is an offset from the begin the table __ProgName_CODE. This address correspond to the block if code of the BEGIN action or a non SFC program. For example, the table being at the relative address 01 AC is coded (for INTEL format) : AC 01
2.	Size block. It is the size in bytess of the block of code of the BEGIN action or a non SFC program. For example a block of code of 28 bytes is coded (for INTEL format) : 1C 00
3.	Address block. It is an offset from the begin the table __ProgName_CODE. This address correspond to the block of code of the COMPLEX action. For example, the table being at the relative address 01 AC is coded (for INTEL format) : AC 01
4.	Size block. It is the size in bytess of the block of code of the COMPLEX action. For example a block of code of 28 bytes is coded (for INTEL format) : 1C 00
5.	Address block. It is an offset from the begin the table

__ProgName_CODE. This address correspond to the block of code of the END action. For example, the table being at the relative address 01 AC is coded (for INTEL format) : AC 01

6. Size block. It is the size in bytess of the block of code of the END action. For example a block of code of 28 bytes is coded (for INTEL format) : 1C 00

7. Address block. It is an offset from the begin the table __ProgName_CODE. This address correspond to the block of code of a SFC transition. For example, the table being at the relative address 01 AC is coded (for INTEL format) : AC 01

8. Size block. It is the size in bytess of the block of code of a SFC transition. For example a block of code of 28 bytes is coded (for INTEL format) : 1C 00

9. TIC code. It is the TIC code number. For more information see the TIC code document. For example, the TIC code number 03 (hexa) is coded (for INTEL format) : 03 00

10. Parameter. It a parameter of TIC code. For more information see the TIC code document. For example, a variable numbered 7 is coded (for INTEL format) : 07 00

11. Terminator. It is a string of x null characters. x is : (2*p*i ΣΣ(1+pu,v))%4 with 1≤u≤i and 1≤v≤p. For example, a terminator length of 2 is coded : 00 00

D. 1. 2. 14. Table of the SFC's structure

This table contains the address of all the program structure (structure of SFC programs).

Name : __LINKSFC			Size : n*4 bytes	
	Field No.	Format	Type	Comment
repeated n time	1	uint32	AR, PD	Address of program structure

Field No.	Notes
1.	Address of program sequence. It is the absolute address for SFC programs and null address for none SFC programs. For example; the table at address 00 00 00 D0 is coded (for INTEL format) : D0 00 00 00

NOTE : The number of programs is calculated with the table __DEFHIE (it is the total of the fields of 1 to 5).

D. 1. 2. 14. 1. *Address of the SFCs structure*

This table defines all the steps and transitions and their links of a SFC programs. Be careful, because a SFC program can content several SFC.

Name : __ProgName_LINK		**Size : 8+2*m*Σ(iu+iu+2) + 2*m*Σ(jv+jv+2) + (8+2*m*Σ(i_u+i_u+2) + 2*m*Σ(j_v+j_v+2))%4 with 1≤u≤i and 1≤v≤j bytes**			
		Field No.	**Format**	**Type**	**Comment**
header		1	uint16	V, PD	first transition
		2	uint16	V, PD	number of transition
		3	uint16	V, PD	first step
		4	uint16	V, PD	number of step
step link	**repeated**	5	uint16	RR, PD	address predecessors
description	**m times**	6	uint16	RR, PD	addr successors
transition link	**repeated**	7	uint16	RR, PD	addr predecessors
description	**n time**	8	uint16	RR, PD	address successors
predecessor	**repeated i times**	9	uint16	V, PD	transition number
of step		10	uint16	V, PD	terminator
successor	**repeated j times**	11	uint16	V, PD	transition number
of step		12	uint16	V, PD	terminator
predecessor	**repeated k times**	13	uint16	V, PD	step number
of transition		14	uint16	V, PD	terminator
successor	**repeated l times**	15	uint16	V, PD	step number
of transition		16	uint16	V, PD	terminator
		17	charx	V, PD	terminator

Field No.	Notes
1.	First transition. It is the first transition number of a SFC program. For example, the first transition numbered 1 is coded (for INTEL format) : 01 00
2.	Number of transition. It is the total number of transition of the SFC program. For example, a SFC with 10 transitions is coded (for INTEL format) : 0A 00
3.	First step. It is the first step number of a SFC program. For example, the first step numbered 100 is coded (for INTEL format) : 64 00

4. Number of step. It is the total number of step of the SFC program. For example, a SFC with 20 steps is coded (for INTEL format) : 14 00

5. Address predecessors. It is an address of the list of the predecessors of a step. It is an offset in bytes from the beginning of the table. For example, an offset of 30 bytes is coded (for INTEL format) : 1E 00

6. Address successors. It is an address of the list of the successors of a step. It is an offset in bytes from the beginning of the table. For example, an offset of 20 bytes is coded (for INTEL format) : 14 00

7. Address predecessors. It is an address of the list of the predecessors of a transition. It is an offset in bytes from the beginning of the table. For example, an offset of 30 bytes is coded (for INTEL format) : 1E 00

8. Address successors. It is an address of the list of the successors of a transition. It is an offset in bytes from the beginning of the table. For example, an offset of 20 bytes is coded (for INTEL format) : 14 00

9. Transition number. It is the transition number preceding the step. For example, the transition numbered 2 is coded (for INTEL format) : 02 00

10. Terminator. It is a null value terminating the list of transitions preceding the step. This value is coded : 00 00

11. Transition number. It is the transition number succeeding the step. For example, the transition numbered 4 is coded (for INTEL format) : 04 00

12. Terminator. It is a null value terminating the list of transitions succeeding the step. This value is coded : 00 00

13. Step number. It is the step number preceding the transition. For example, the step numbered 2 is coded (for INTEL format) : 02 00

14. Terminator. It is a null value terminating the list of steps preceding the step. This value is coded : 00 00

15. Step number. It is the step number succeeding the transition. For example, the step numbered 4 is coded (for INTEL format) : 04 00

16. Terminator. It is a null value terminating the list of steps succeeding the step. This value is coded : 00 00

17. Terminator. It is a string of x null value. $x = (8+2*m*\Sigma(iu+iu+2) + 2*m*\Sigma(jv+jv+2))\%4$ with $1 \leq u \leq i$ and $1 \leq v \leq j$. A value of x of 1 is coded : 00

D. 1. 2. 15. Resources table

This table contains all the information in relation to the resources.

Name : __RESOURCE		Size : 4+32*i + $\Sigma(x_u+y_u)$ + (4 + 32*l + $\Sigma(x_u+y_u)$)%4, 1≤u≤j bytes			
		Field No.	**Format**	**Type**	**Comment**
header		1	uint32	V, PD	number of resources
resource	**repeated i time**	2	char16	S, PI	name
		3	uint32	V, PD	type
		4	uint32	V, PD	data size
		5	uint32	V, PD	address of data
		6	uint32	V, PD	address of path
data	**repeated j time**	7	charx	S, PI	data
		8	chary	S, PI	path
		9	charz	S, PI	terminator

Field No.	Notes
1.	Number of resources. It is the number of resource of the application. For example, It five resources to be transfer are coded (for INTEL format) : 05 00 00 00
2.	Name. It the name of the resource. It is a string of 16 maximum characters terminated by 0. For example, a resource named "ANA" is coded : 41 4E 41 00 00 00 00 00 00 00 00 00 00 00 00 00
3.	Type. It is the type of resource. 1 for binary file, 2 for text file, 3 for ulongdata and 4 for varlist. For example, the type of a text file to be transfer is coded (for INTEL format) : 02 00 00 00
4.	Data size. It is the size in bytess of the data of the resource. For example, the size of a text file of 20 bytes to be transfer is coded (for INTEL format) : 14 00 00 00
5.	Address of data. It is an offset in bytes from the beginning of the current resource (field 1) which located the data. For example, an offset of 30 bytes id coded (for INTEL format) : 1E 00 00 00
6.	Address of path. It is an offset in bytes from the beginning of the current resource (field 1) which located the path for a file to be transfers. For example, an offset of 20 bytes id coded (for INTEL format) : 14 00 00 00
7.	Data. It is a string of x characters. X is defined by the field 4. For example the data "AAA" is coded : 41 41 41

8. Path. It is a string of y characters terminated by 0. The path indicated the location in the target of a file to be transfer. For example the path "/PRJ" is coded : 2F 50 52 4A

9. Terminator. It is a string of x null value. x = (4 + 32*i + Σ(xu+yu))%4 with 1$\leq$u$\leq$j.This value is coded : 00

E. Appendix 2 : TIC code definition

E. 1. Introduction

This section is a whole description of the **TIC code V2.00**.

E. 2. Notations in the instructions

E. 2. 1. Data type

The '**Type**' field describes the data type of an object, and can take one of the following values:

Type Name	Description	Format
NIL	no type is associated to the object	
INTEGER	integer analog	16 bits
REAL	real analog	
BOOLEAN	Boolean	
MESSAGE	message string	0-255 chars
TIMER	time of day in MS	
SFC	SFC program	

E. 2. 2. TIC code significance

The name of the TIC code contents certain mnemonic which indicate the type of the instruction.

Attributes Name	Description	Example
OUT	instructions used to assign an output variable	TIC**_OUT_**A3I
SYS	instructions used kernel functions	TIC**_SYS_**TAC
USF	instructions represent a call to a user written C function	TIC**_USF_**A4B

TRS	instructions used to assign a SFC transition receptivity	TIC**_TRS_**A3B
STD	instructions represent a call to a standard C function	TIC**_STD_**FBL
_A1	unary operation	TIC**_A1**B
_A2	binary operation	TIC**_A2**I
_A3	simple copy operation	TIC**_A3**M
_A4	call function of function block	TIC**_A4**I
_A5	copy operation from a array to a simple variable	TIC**_A5**B
_A6	copy operation from a simple variable to an array	TIC**_A6**R
_C1	conditional jump	TIC**_C1**B
_C2	conditional jump	TIC**_C2**B
_I1	unconditional jump	TIC**_I1**
_I2	unconditional jump	TIC**_I2**
_L1	label	TIC**_L1**
_P1	parameter passing	TIC**_P1**
B	Boolean instruction	TIC_A3**B**
I	integer instruction	TIC_A3**I**
R	real instruction	TIC_A3**R**
T	timer instruction	TIC_A3**T**
M	message instruction	TIC_A3**M**

E. 2. 3. Symbols

The following notation is used to describe the operation and TIC syntax

Symbol	Comments
d	displacement or constant value
<a>	effective address of the variable a
#<a>	immediate data
label	assemble program label

E. 2. 4. Intel or Motorola implementation

Size	Intel format	Motorola format
1 byte→1	1	1
2 bytes→12	12	21
4 bytes→1234	1234	4321

E. 3. Basic instructions

The basic instructions performing single operations between variables are detailed below.

E. 3. 1. Integer instructions

TIC_A1SI Integer Negation TIC_A1SI

Operation :

0 - <a2> → <a1>

TIC syntax :

<a1> = - <a2>

Type :

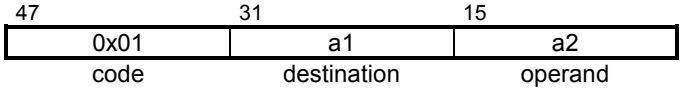
a1	integer
a2	integer

Description :

Subtracts the operand **<a2>** from zero and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2MI Integer Multiply TIC_A2MI

Operation :

<a2> * <a3> → <a1>

TIC syntax :

<a1> = <a2> * <a3>

Type :

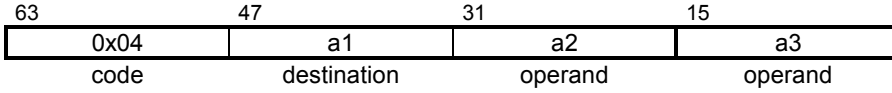
a1	integer
a2	integer
a3	integer

Description :

Multiplies the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2DI Integer Divide TIC_A2DI

Operation :

<a2> / <a3> → <a1>

TIC syntax :

<a1> = <a2> / <a3>

Type :

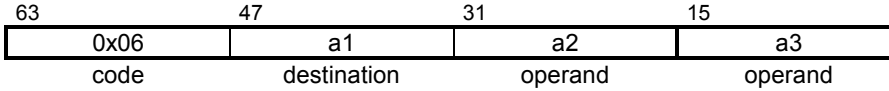
a1	integer
a2	integer
a3	integer

Description :

Divides the operand **<a2>** by the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2AI Integer Add TIC_A2AI

Operation :

<a2> + <a3> → <a1>

TIC syntax :

<a1> = <a2> + <a3>

Type :

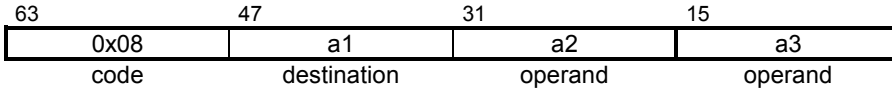
a1	integer
a2	integer
a3	integer

Description :

Adds the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2SI Integer Subtract TIC_A2SI

Operation :

<a2> - <a3> → <a1>

TIC syntax :

<a1> = <a2> - <a3>

Type :

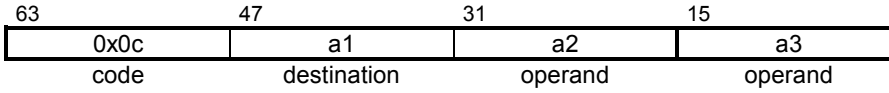
a1	integer
a2	integer
a3	integer

Description :

Subtracts the operand **<a3>** from the operand **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2LTI Integer Test TIC_A2LTI

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

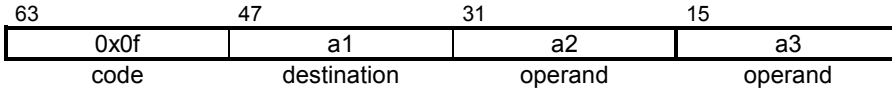
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2GTI Integer Test TIC_A2GTI

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

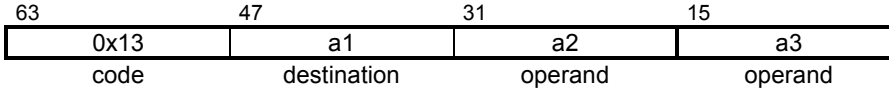
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2LEI Integer Test TIC_A2LEI

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

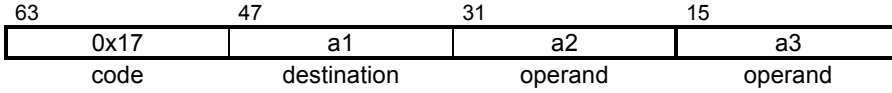
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is less than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2GEI Integer Test TIC_A2GEI

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

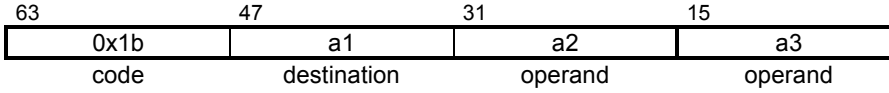
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is greater or equal than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2EQI Integer Test TIC_A2EQI

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

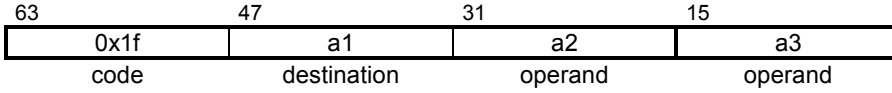
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2NEI Integer Test TIC_A2NEI

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

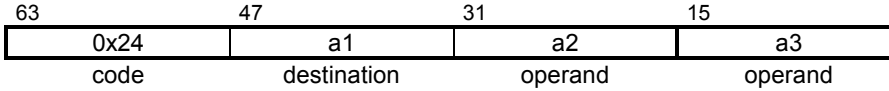
a1	boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A3I Integer Assignment TIC_A3I

Operation :

<a2> → <a1>

TIC syntax :

<a1> = <a2>

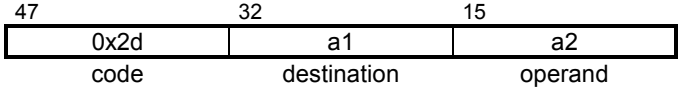
Type :

a1	integer
a2	integer

Description : Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_A5I Integer Assignment of array element TIC_A5I

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

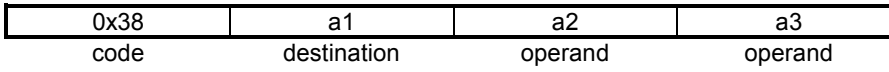
a1	integer
a2	integer
a3	integer

Description :

Reads an element of the index **<a2>** from the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A6I Integer Assignment array TIC_A6I

Operation :

<a3> →<a1> [<a2>]

TIC syntax :

<a1> [<a2>] = <a3>

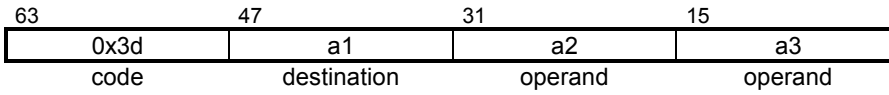
Type :

a1	integer
a2	integer
a3	integer

Description : Stores **<a3>** in array **<a1>** at the index **<a2>**

Format :

- **Medium**



TIC_TRS_A2LTI Integer Test TIC_TRS_A2LTI

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

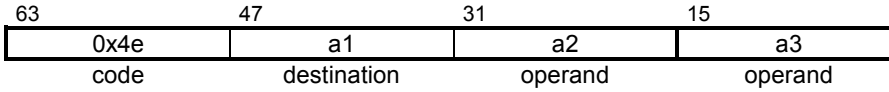
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2GTI Integer Test TIC_TRS_A2GTI

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

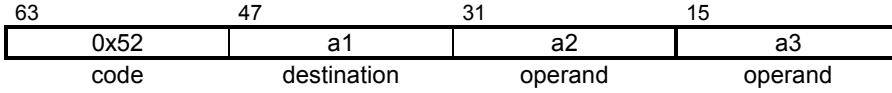
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2LEI Integer Test TIC_TRS_A2LEI

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

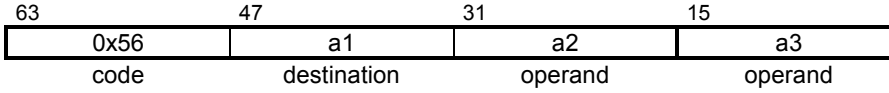
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is less or equal than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2GEI Integer Test TIC_TRS_A2GEI

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

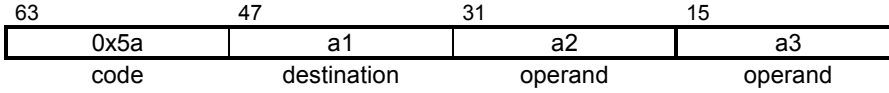
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2EQI Integer Test TIC_TRS_A2EQI

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

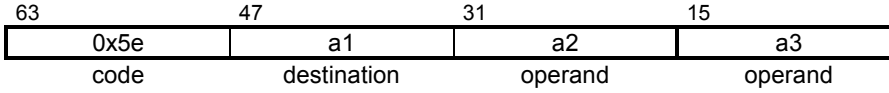
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2NEI Integer Test TIC_TRS_A2NEI

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

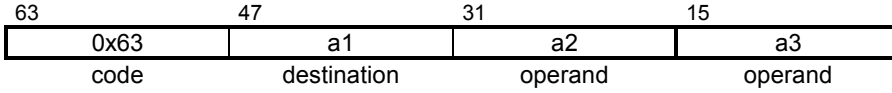
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A1SI Integer Negation TIC_OUT_A1SI

Operation :

0 - <a2> → <a1>

TIC syntax :

<a1> = - <a2>

Type :

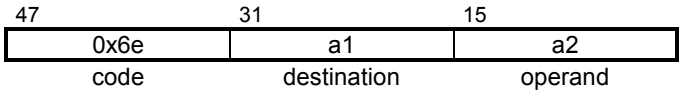
a1	integer
a2	integer

Description :

Subtracts the operand **<a2>** from zero and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2MI Integer Multiply TIC_OUT_A2MI

Operation :

<a2> * <a3> → <a1>

TIC syntax :

<a1> = <a2> * <a3>

Type :

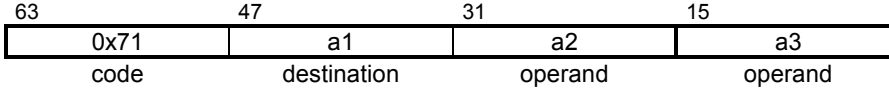
a1	integer
a2	integer
a3	integer

Description :

Multiplies the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2DI Integer Divide TIC_OUT_A2DI

Operation :

<a2> / <a3> → <a1>

TIC syntax :

<a1> = <a2> / <a3>

Type :

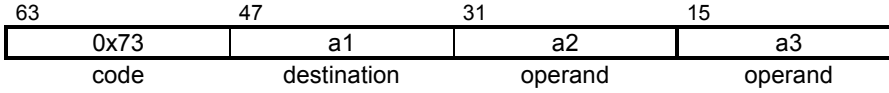
a1	integer
a2	integer
a3	integer

Description :

Divides the operand **<a2>** by the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2AI Integer Add TIC_OUT_A2AI

Operation :

<a2> + <a3> → <a1>

TIC syntax :

<a1> = <a2> + <a3>

Type :

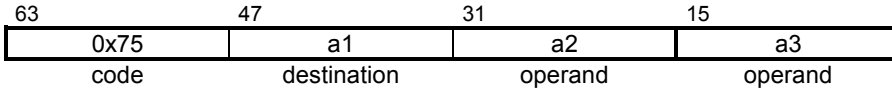
a1	integer
a2	integer
a3	integer

Description :

Adds the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2SI Integer Subtract TIC_OUT_A2SI

Operation :

<a2> - <a3> → <a1>

TIC syntax :

<a1> = <a2> - <a3>

Type :

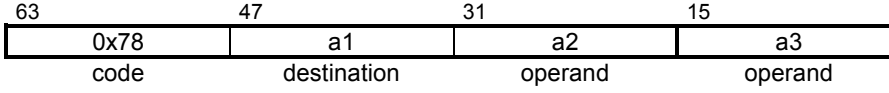
a1	integer
a2	integer
a3	integer

Description :

Subtracts the operand **<a3>** from the operand **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2LTI Integer Test TIC_OUT_A2LTI

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

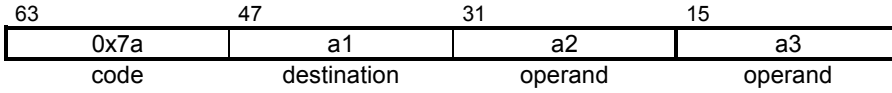
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2GTI Integer Test TIC_OUT_A2GTI

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

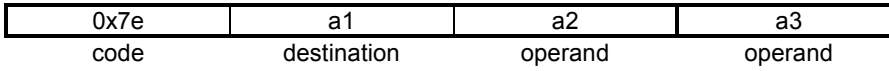
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2LEI Integer Test TIC_OUT_A2LEI

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

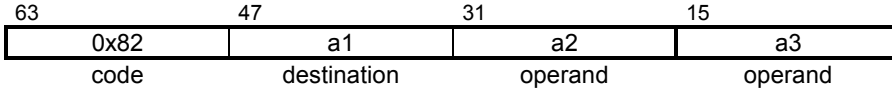
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is less or equal than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2GEI Integer Test TIC_OUT_A2GEI

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

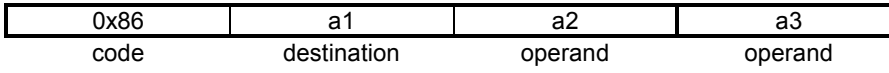
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2EQI Integer Test TIC_OUT_A2EQI

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

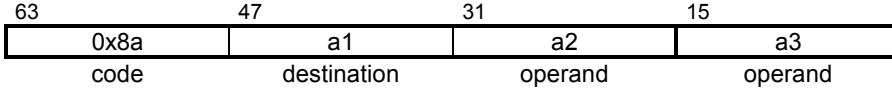
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2NEI Integer Test TIC_OUT_A2NEI

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

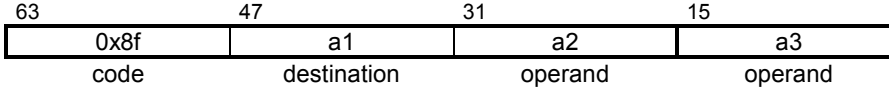
a1	Boolean
a2	integer
a3	integer

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A3I Integer Assignment TIC_OUT_A3I

Operation :

<a2> → <a1>

TIC syntax :

<a1> = <a2>

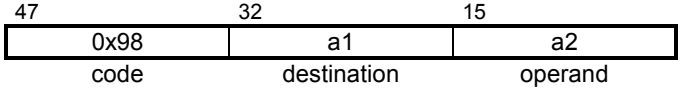
Type :

a1	integer
a2	integer

Description : Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A5I Integer Assignment of array element TIC_OUT_A5I

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

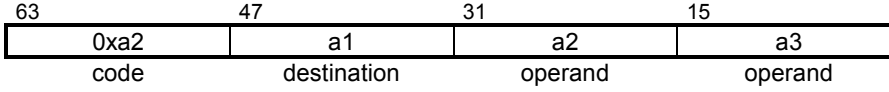
a1	integer
a2	integer
a3	integer

Description :

Reads an element of the index **<a2>** from the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A6I Integer TIC_OUT_A6I Assignment array

Operation : <a3> →<a1> [<a2>]

TIC syntax : <a1> [<a2>] = <a3>

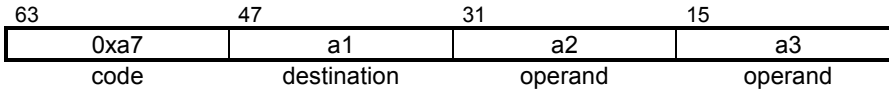
Type :

a1	integer
a2	integer
a3	integer

Description : Stores **<a3>** in array **<a1>** at the index **<a2>**

Format :

- **Medium**



E. 3. 2. Real instructions

TIC_A1SR Real Negation TIC_A1SR

Operation :

0 - <a2> → <a1>

TIC syntax :

<a1> = - <a2>

Type :

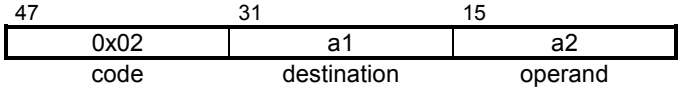
a1	Real
a2	Real

Description :

Subtracts the operand **<a2>** from zero and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2MR Real Multiply TIC_A2MR

Operation :

<a2> * <a3> → <a1>

TIC syntax :

<a1> = <a2> * <a3>

Type :

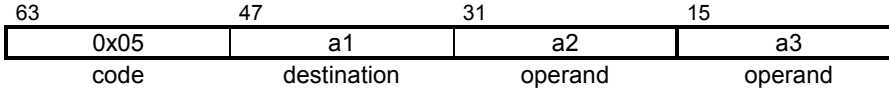
a1	Real
a2	Real
a3	Real

Description :

Multiplies the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2DR Real Divide TIC_A2DR

Operation :

<a2> / <a3> → <a1>

TIC syntax :

<a1> = <a2> / <a3>

Type :

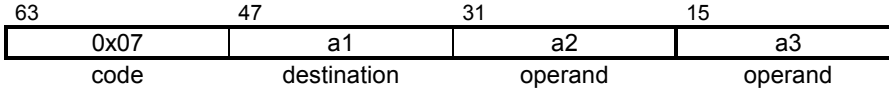
a1	Real
a2	Real
a3	Real

Description :

Divides the operand **<a2>** by the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2AR Real Add TIC_A2AR

Operation :

<a2> + <a3> → <a1>

TIC syntax :

<a1> = <a2> + <a3>

Type :

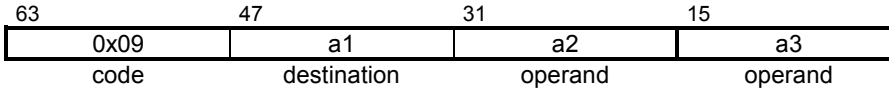
a1	Real
a2	Real
a3	Real

Description :

Adds the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2SR Real Subtract TIC_A2SR

Operation :

<a2> - <a3> → <a1>

TIC syntax :

<a1> = <a2> - <a3>

Type :

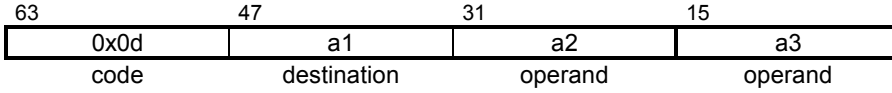
a1	Real
a2	Real
a3	Real

Description :

Subtracts the operand **<a3>** from the operand **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2LTR Real Test TIC_A2LTR

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

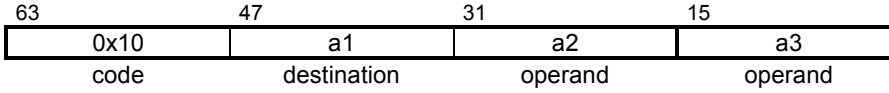
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2GTR Real Test TIC_A2GTR

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

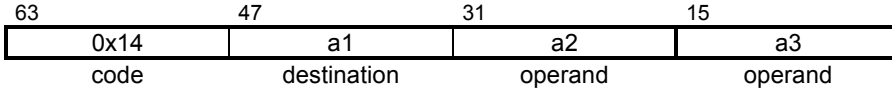
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2LER Real Test TIC_A2LER

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

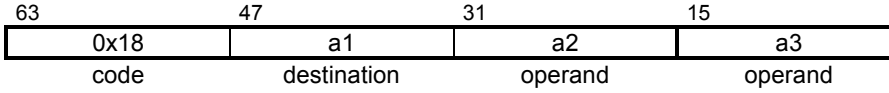
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is less than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2GER Real Test TIC_A2GER

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

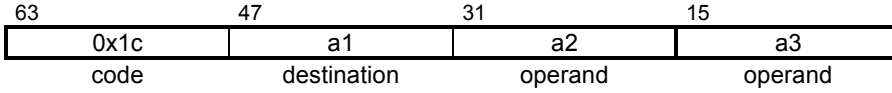
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2EQR Real Test TIC_A2EQR

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

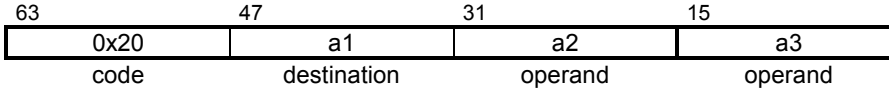
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2NER Real Test TIC_A2NER

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

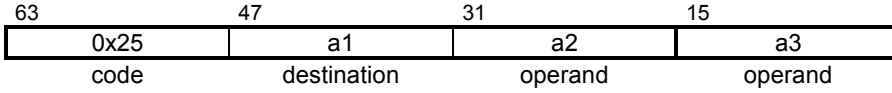
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A3R Real Assignment TIC_A3R

Operation :

<a2> → <a1>

TIC syntax :

<a1> = <a2>

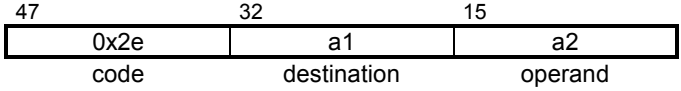
Type :

a1	Real
a2	Real

Description : Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_A5R Real Assignment of array element TIC_A5R

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

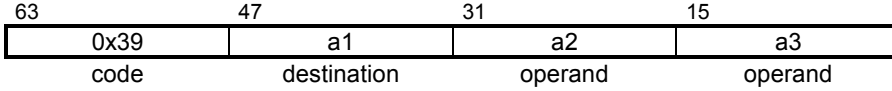
a1	Real
a2	integer
a3	integer

Description :

Reads an element of the index **<a2>** from the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A6R Real Assignment array TIC_A6R

Operation :

<a3> →<a1> [<a2>]

TIC syntax :

<a1> [<a2>] = <a3>

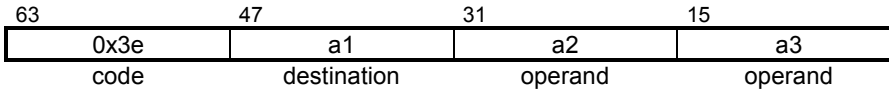
Type :

a1	integer
a2	integer
a3	Real

Description : Stores **<a3>** in array **<a1>** at the index **<a2>**

Format :

- **Medium**



TIC_TRS_A2LTR Real Test TIC_TRS_A2LTR

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

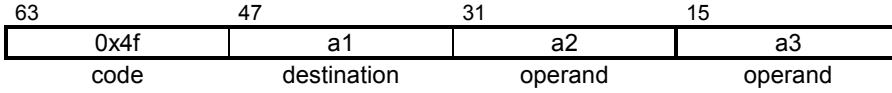
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2GTR Real Test TIC_TRS_A2GTR

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

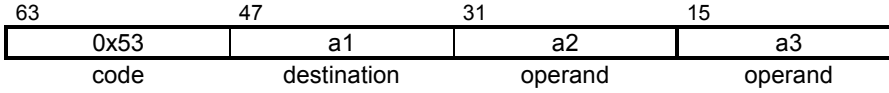
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2LER Real Test TIC_TRS_A2LER

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

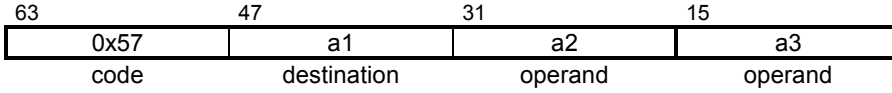
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is less than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2GER Real Test TIC_TRS_A2GER

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

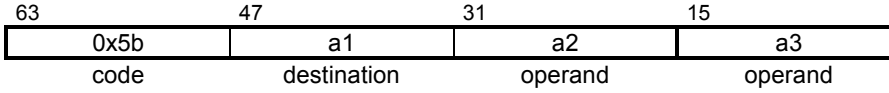
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2EQR Real Test TIC_TRS_A2EQR

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

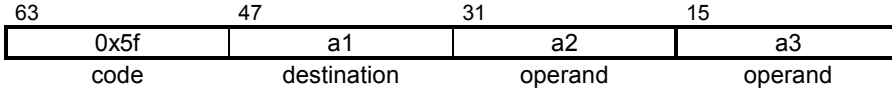
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2NER Real Test TIC_TRS_A2NER

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

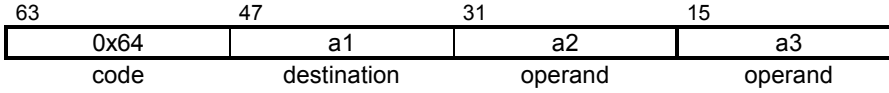
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A1SR Real Negation TIC_OUT_A1SR

Operation :

0 - <a2> → <a1>

TIC syntax :

<a1> = - <a2>

Type :

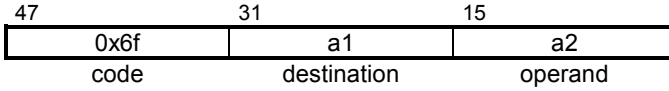
a1	Real
a2	Real

Description :

Subtracts the operand **<a2>** from zero and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2MR Real Multiply TIC_OUT_A2MR

Operation :

<a2> * <a3> → <a1>

TIC syntax :

<a1> = <a2> * <a3>

Type :

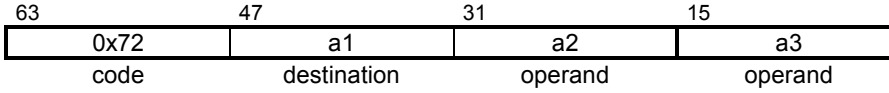
a1	Real
a2	Real
a3	Real

Description :

Multiplies the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2DR Real Divide TIC_OUT_A2DR

Operation :

<a2> / <a3> → <a1>

TIC syntax :

<a1> = <a2> / <a3>

Type :

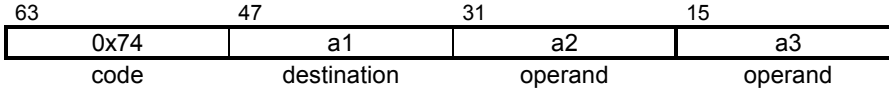
a1	Real
a2	Real
a3	Real

Description :

Divides the operand **<a2>** by the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2AR Real Add TIC_OUT_A2AR

Operation :

<a2> + <a3> → <a1>

TIC syntax :

<a1> = <a2> + <a3>

Type :

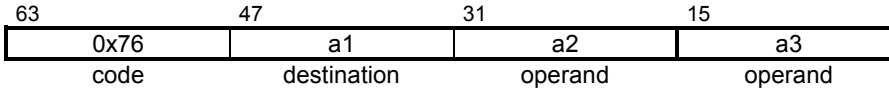
a1	Real
a2	Real
a3	Real

Description :

Adds the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2SR Real Subtract TIC_OUT_A2SR

Operation :

<a2> - <a3> → <a1>

TIC syntax :

<a1> = <a2> - <a3>

Type :

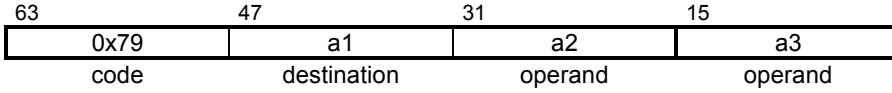
a1	Real
a2	Real
a3	Real

Description :

Subtracts the operand **<a3>** from the operand **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2LTR Real Test TIC_OUT_A2LTR

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

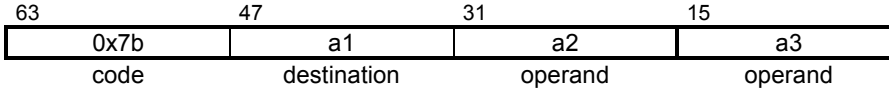
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2GTR Real Test TIC_OUT_A2GTR

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

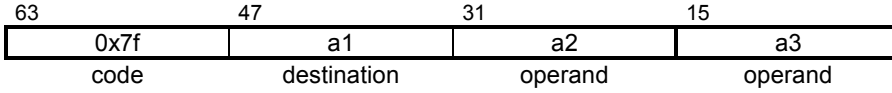
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2LER Real Test TIC_OUT_A2LER

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

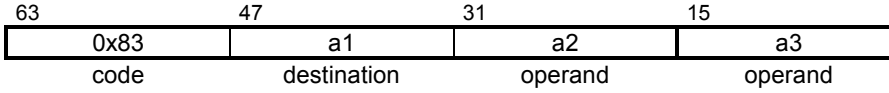
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is less than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2GER Real Test TIC_OUT_A2GER

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

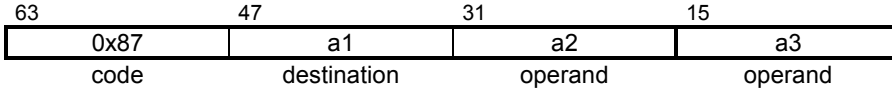
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2EQR Real Test TIC_OUT_A2EQR

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

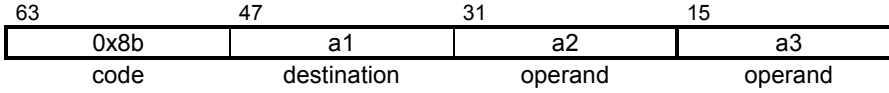
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2NER Real Test TIC_OUT_A2NER

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

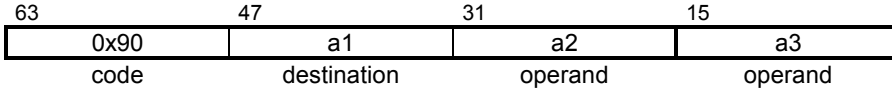
a1	Boolean
a2	Real
a3	Real

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A3R Real Assignment TIC_OUT_A3R

Operation : <a2> → <a1>

TIC syntax : <a1> = <a2>

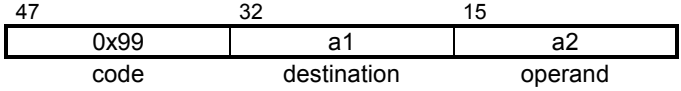
Type :

a1	Real
a2	Real

Description : Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A5R Real Assignment of array element TIC_OUT_A5R

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

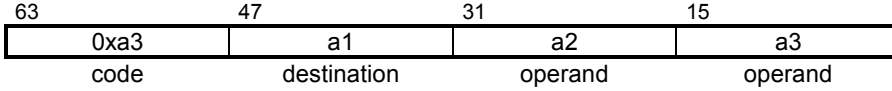
a1	Real
a2	integer
a3	integer

Description :

Reads an element of the index **<a2>** from the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A6R Real Assignment array TIC_OUT_A6R

Operation :

<a3> →<a1> [<a2>]

TIC syntax :

<a1> [<a2>] = <a3>

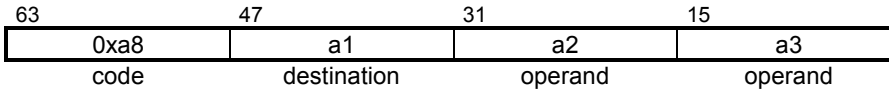
Type :

a1	integer
a2	integer
a3	Real

Description : Stores **<a3>** in array **<a1>** at the index **<a2>**

Format :

- **Medium**



E. 3. 3. Boolean instructions

TIC_A1NO. Boolean Negation TIC_A1NO.

Operation :

~ <a2> → <a1>

TIC syntax :

<a1> = ! <a2>

Type :

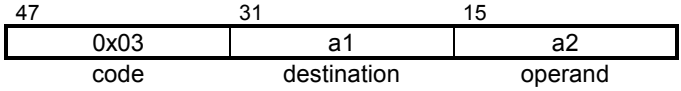
a1	Boolean
a2	Boolean

Description :

Calculates the logical-negation of the operand **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2EQB Boolean Test TIC_A2EQB

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

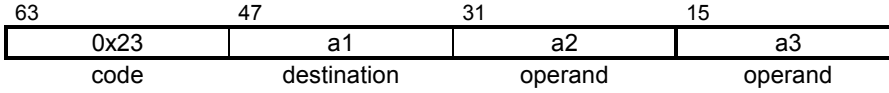
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2NEB Boolean Test TIC_A2NEB

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

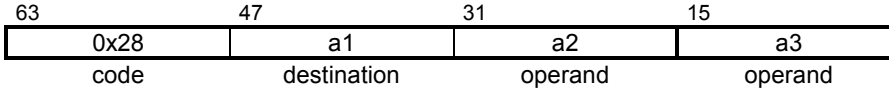
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2ORB OR Logical TIC_A2ORB

Operation :

<a2> OR <a3> → <a1>

TIC syntax :

<a1> = <a2> || <a3>

Type :

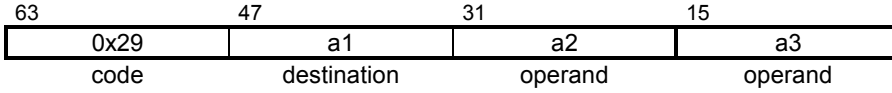
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Performs an inclusive OR on the operand **<a2>** and the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2ANO. AND Logical TIC_A2ANO.

Operation :

<a2> AND <a3> → <a1>

TIC syntax :

<a1> = <a2> && <a3>

Type :

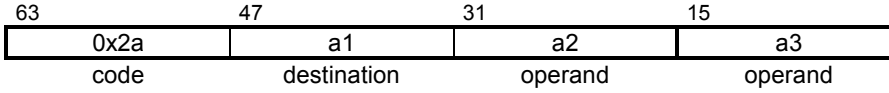
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Performs an AND operation of the operand **<a2>** and the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2XOB Logical XOR TIC_A2XOB

Operation :

<a2> XOR <a3> → <a1>

TIC syntax :

<a1> = <a2> ^^ <a3>

Type :

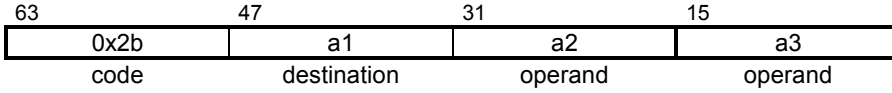
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Performs an exclusive OR on the operand **<a2>** and the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A3B Boolean Assignment TIC_A3B

Operation :

<a2> → <a1>

TIC syntax :

<a1> = <a2>

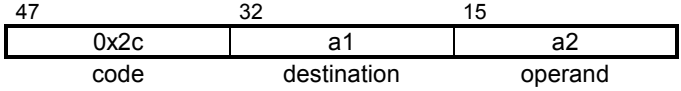
Type :

a1	Boolean
a2	Boolean

Description : Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_A5B Boolean Assignment of array element TIC_A5B

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

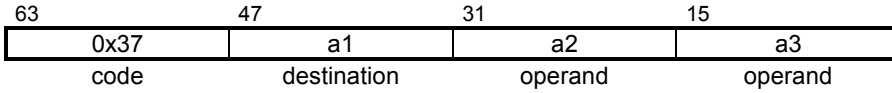
a1	Boolean
a2	integer
a3	integer

Description :

Reads an element of the index **<a2>** from the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A6B Boolean Assignment array TIC_A6B

Operation :

<a3> →<a1> [<a2>]

TIC syntax :

<a1> [<a2>] = <a3>

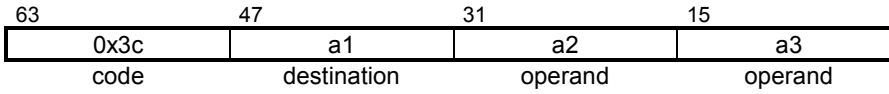
Type :

a1	integer
a2	integer
a3	Boolean

Description : Stores **<a3>** in array **<a1>** at the index **<a2>**

Format :

- **Medium**



TIC_TRS_A1NO. Boolean Negation TIC_TRS_A1NO .

Operation :

~ <a2> → <a1>

TIC syntax :

<a1> = ! <a2>

Type :

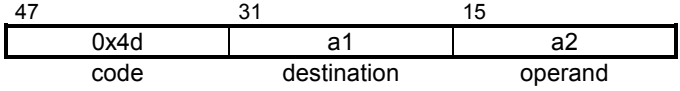
a1	Boolean
a2	Boolean

Description :

Calculates the logical-negation of the operand **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2EQB Boolean Test TIC_TRS_A2EQB

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

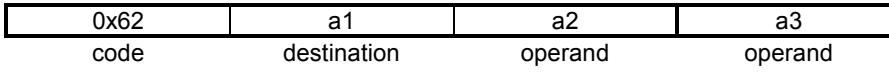
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2NEB Boolean Test TIC_TRS_A2NEB

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

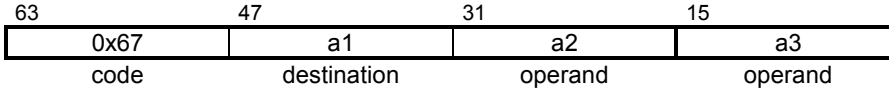
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2ORB OR Logical TIC_TRS_A2ORB

Operation :

<a2> OR <a3> → <a1>

TIC syntax :

<a1> = <a2> || <a3>

Type :

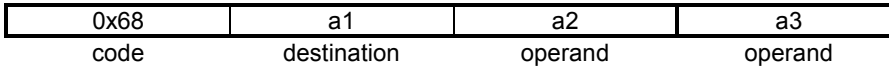
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Performs an inclusive OR on the operand **<a2>** and the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2ANO AND Logical TIC_TRS_A2ANO.

.

Operation :

<a2> AND <a3> → <a1>

TIC syntax :

<a1> = <a2> && <a3>

Type :

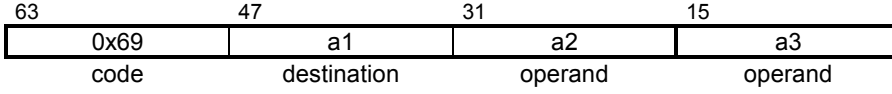
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Performs an AND operation on the operand **<a2>** and the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2XOB Logical XOR TIC_TRS_A2XOB

Operation :

<a2> XOR <a3> → <a1>

TIC syntax :

<a1> = <a2> ^^ <a3>

Type :

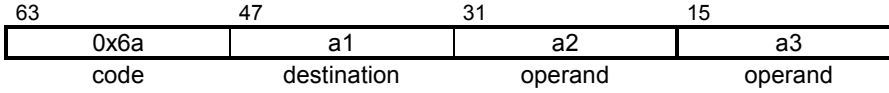
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Performs an exclusive OR on the operand **<a2>** and the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A3B Boolean Assignment TIC_TRS_A3B

Operation :

<a2> → <a1>

TIC syntax :

<a1> = <a2>

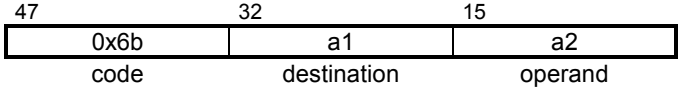
Type :

a1	Boolean
a2	Boolean

Description : Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A5B Boolean Assignment of array element TIC_TRS_A5B

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

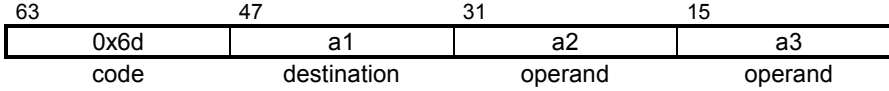
a1	Boolean
a2	integer
a3	integer

Description :

Reads an element of the index **<a2>** from the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A1NO . Boolean Negation TIC_OUT_A1NO.

Operation :

~ <a2> → <a1>

TIC syntax :

<a1> = ! <a2>

Type :

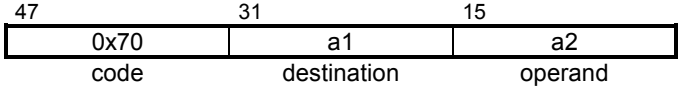
a1	Boolean
a2	Boolean

Description :

Calculates the logical-negation of the operand **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2EQB Boolean TIC_OUT_A2EQB Test

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

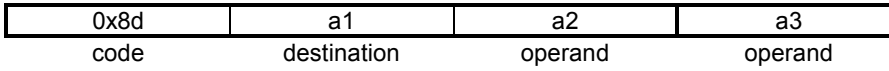
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2NEB Boolean Test TIC_OUT_A2NEB

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

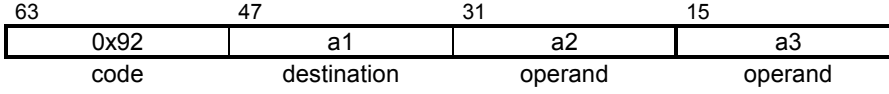
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2ORB OR Logical TIC_OUT_A2ORB

Operation :

<a2> OR <a3> → <a1>

TIC syntax :

<a1> = <a2> || <a3>

Type :

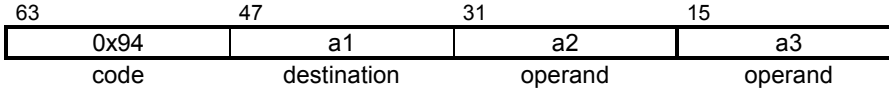
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Performs an inclusive OR on the operand **<a2>** and the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2ANO. AND Logical TIC_OUT_A2ANO.

Operation :

<a2> AND <a3> → <a1>

TIC syntax :

<a1> = <a2> && <a3>

Type :

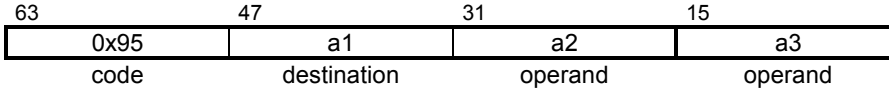
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Performs an AND operation on the operand **<a2>** and the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2XOB Logical XOR TIC_OUT_A2XOB

Operation :

<a2> XOR <a3> → <a1>

TIC syntax :

<a1> = <a2> ^^ <a3>

Type :

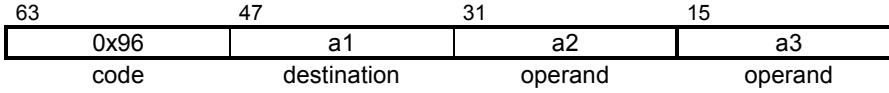
a1	Boolean
a2	Boolean
a3	Boolean

Description :

Performs an exclusive OR on the operand **<a2>** and the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A3B Boolean Assignment TIC_OUT_A3B

Operation :

<a2> → <a1>

TIC syntax :

<a1> = <a2>

Type :

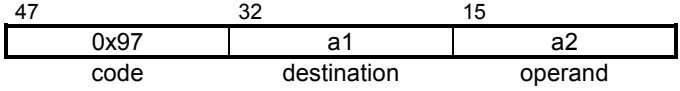
a1	Boolean
a2	Boolean

Description :

Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A5B Boolean Assignment of array element TIC_OUT_A5B

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

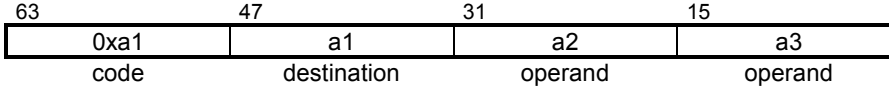
a1	Boolean
a2	integer
a3	integer

Description :

Reads an element of the index **<a2>** from the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A6B Boolean TIC_OUT_A6B Assignment array

Operation :

<a3> →<a1> [<a2>]

TIC syntax :

<a1> [<a2>] = <a3>

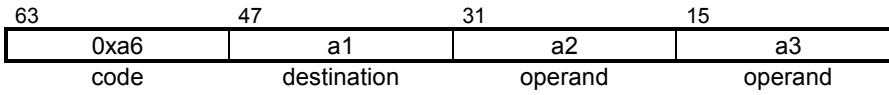
Type :

a1	integer
a2	integer
a3	Boolean

Description : Stores **<a3>** in array **<a1>** at the index **<a2>**

Format :

- **Medium**



E. 3. 4. Message instructions

For further information on the operations on the messages variables refers to **ISaGRAF Guide**.

TIC_A2AM Message Add TIC_A2AM

Operation :

<a2> + <a3> → <a1>

TIC syntax :

<a1> = <a2> + <a3>

Type :

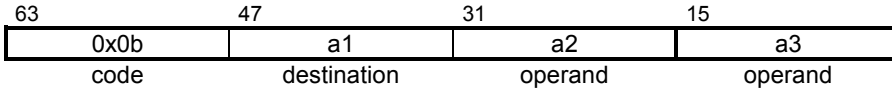
a1	message
a2	message
a3	message

Description :

Adds the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2LTM Message Test TIC_A2LTM

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

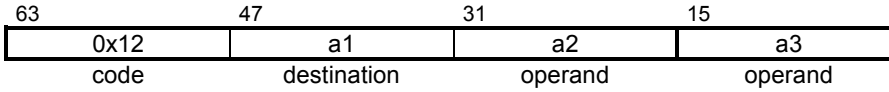
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2GTM Message Test TIC_A2GTM

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

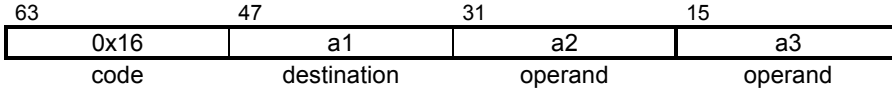
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2LEM Message Test TIC_A2LEM

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

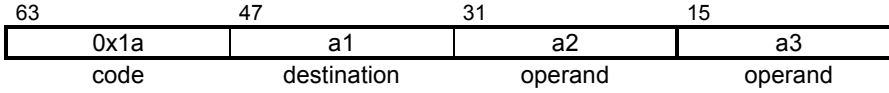
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is less than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2GEM Message Test TIC_A2GEM

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

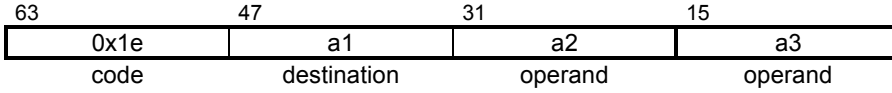
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2EQM Message Test TIC_A2EQM

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

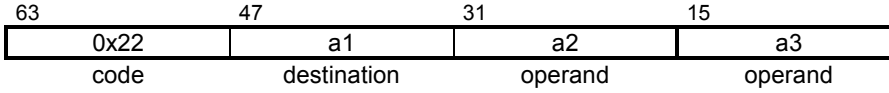
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2NEM Message Test TIC_A2NEM

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

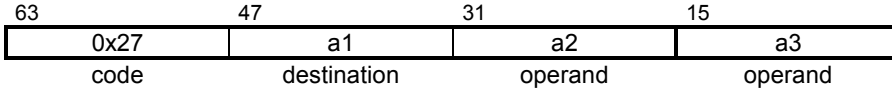
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A3M Message Assignment TIC_A3M

Operation :

<a2> → <a1>

TIC syntax :

<a1> = <a2>

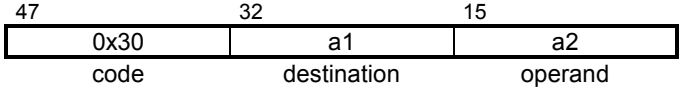
Type :

a1	message
a2	message

Description : Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_A5M Message Assignment of array element TIC_A5M

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

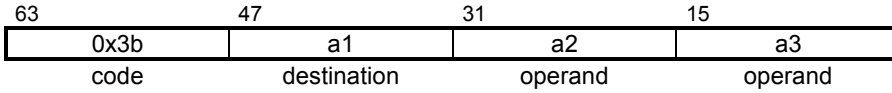
a1	Message
a2	integer
a3	integer

Description :

Reads an element at index **<a2>** of the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A6M Message Assignment array TIC_A6M

Operation :

<a3> →<a1> [<a2>]

TIC syntax :

<a1> [<a2>] = <a3>

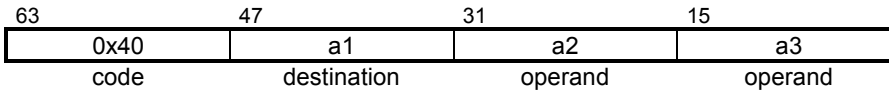
Type :

a1	integer
a2	integer
a3	Message

Description : Stores **<a3>** in array **<a1>** at the index **<a2>**

Format :

- **Medium**



TIC_TRS_A2LTM Message Test TIC_TRS_A2LTM

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

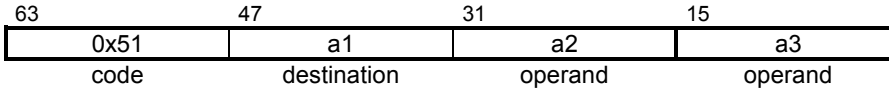
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2GTM Message Test TIC_TRS_A2GTM

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

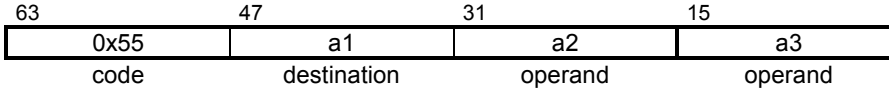
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2LEM Message Test TIC_TRS_A2LEI

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

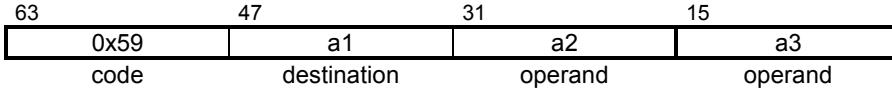
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is less than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2GEM Message Test TIC_TRS_A2GEM

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

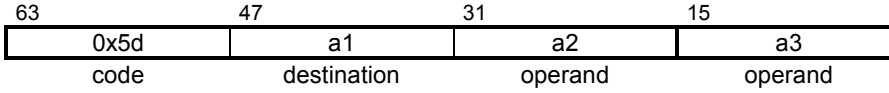
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2EQM Message Test TIC_TRS_A2EQM

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

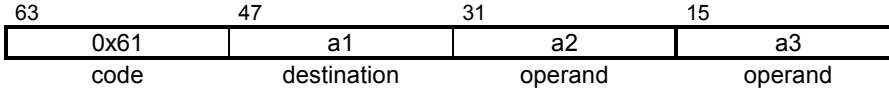
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2NEM Message Test TIC_TRS_A2NEM

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

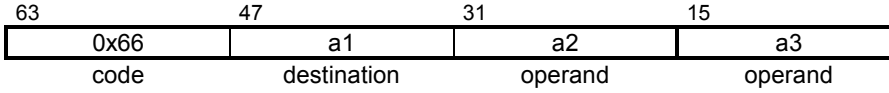
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2AM Message Add TIC_OUT_A2AM

Operation :

<a2> + <a3> → <a1>

TIC syntax :

<a1> = <a2> + <a3>

Type :

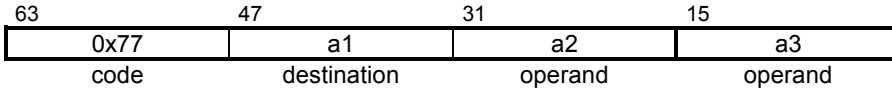
a1	message
a2	message
a3	message

Description :

Adds the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2LTM Message Test TIC_OUT_A2LTM

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

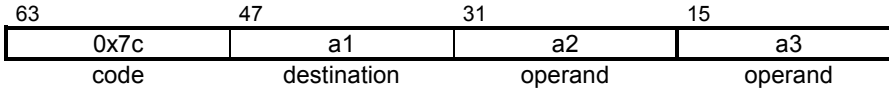
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2GTM Message Test TIC_OUT_A2GTM

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

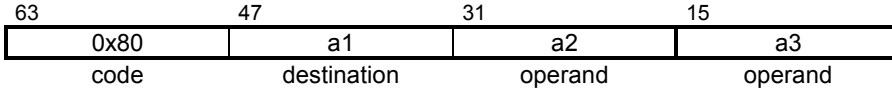
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2LEM Message Test TIC_OUT_A2LEM

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

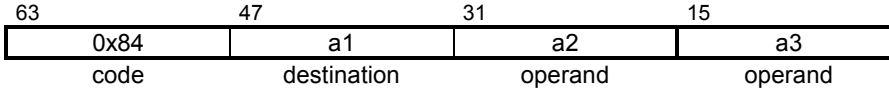
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is less than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2GEM Message Test TIC_OUT_A2GEM

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

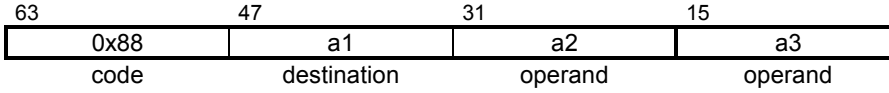
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2EQM Message Test TIC_OUT_A2EQM

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

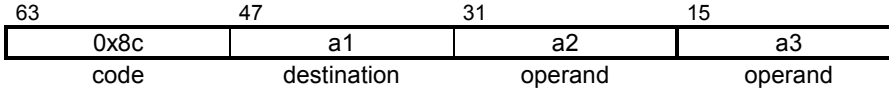
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2NEM Message Test TIC_OUT_A2NEM

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

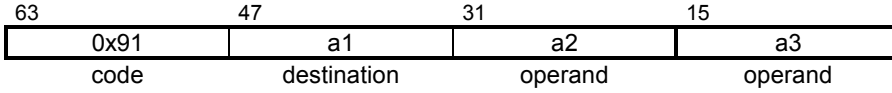
a1	Boolean
a2	message
a3	message

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A3M Message Assignment TIC_OUT_A3M

Operation :

<a2> → <a1>

TIC syntax :

<a1> = <a2>

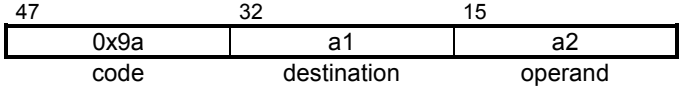
Type :

a1	message
a2	message

Description : Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A5M Message Assignment of array element TIC_OUT_A5M

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

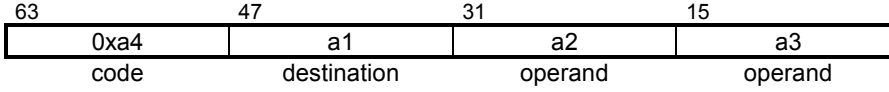
a1	Message
a2	integer
a3	integer

Description :

Reads an element of the index **<a2>** from the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A6M Message TIC_OUT_A6M Assignment array

Operation :

<a3> →<a1> [<a2>]

TIC syntax :

<a1> [<a2>] = <a3>

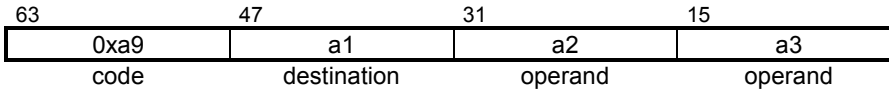
Type :

a1	integer
a2	integer
a3	Message

Description : Stores **<a3>** in array **<a1>** at the index **<a2>**

Format :

- **Medium**



E. 3. 5. Timer instructions

TIC_A2AT Timer Add TIC_A2AT

Operation :

<a2> + <a3> → <a1>

TIC syntax :

<a1> = <a2> + <a3>

Type :

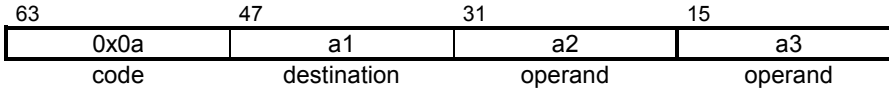
a1	timer
a2	timer
a3	timer

Description :

Adds the two operands **<a2>** and **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2ST Timer Subtract TIC_A2ST

Operation :

<a2> - <a3> → <a1>

TIC syntax :

<a1> = <a2> - <a3>

Type :

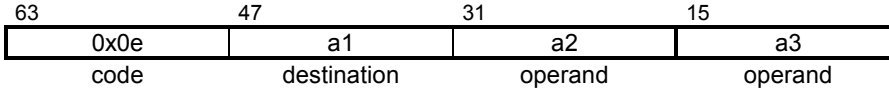
a1	timer
a2	timer
a3	timer

Description :

Subtracts the operand **<a3>** from the operand **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2LTT Timer Test TIC_A2LTT

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

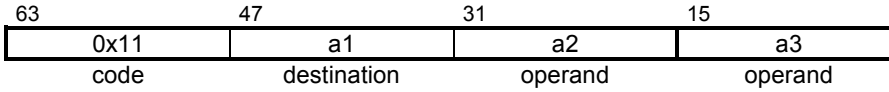
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2GTT Timer Test TIC_A2GTT

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

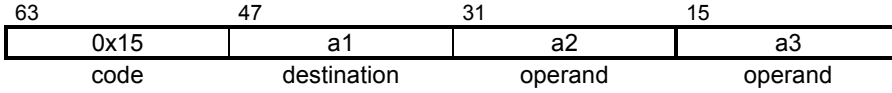
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2LET Timer Test TIC_A2LET

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

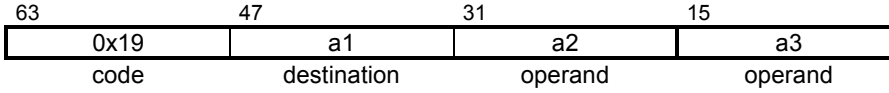
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is less than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2GET Timer Test TIC_A2GET

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

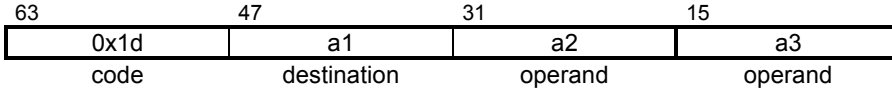
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2EQT Timer Test TIC_A2EQT

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

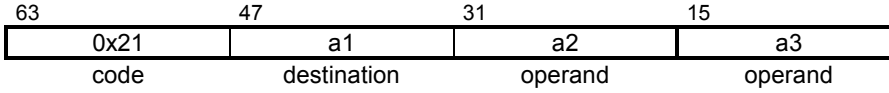
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A2NET Timer Test TIC_A2NET

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

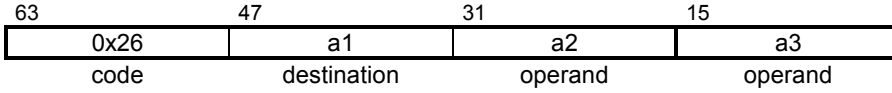
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A3T Timer Assignment TIC_A3T

Operation :

<a2> → <a1>

TIC syntax :

<a1> = <a2>

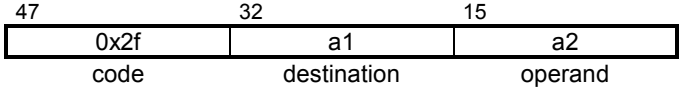
Type :

a1	timer
a2	timer

Description : Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_A5T Timer Assignment of array element TIC_A5T

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

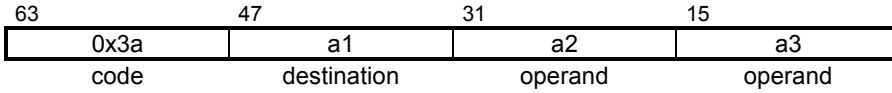
a1	Timer
a2	integer
a3	integer

Description :

Reads an element of the index **<a2>** from the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_A6T Timer Assignment array TIC_A6T

Operation : <a3> →<a1> [<a2>]

TIC syntax : <a1> [<a2>] = <a3>

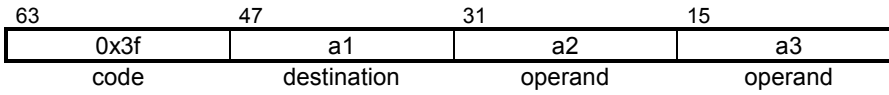
Type :

a1	integer
a2	integer
a3	Timer

Description : Stores **<a3>** in array **<a1>** at the index **<a2>**

Format :

- **Medium**



TIC_TRS_A2LTT Timer Test TIC_TRS_A2LTT

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

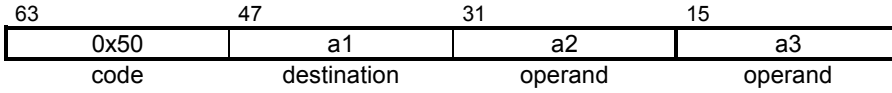
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>**

Format :

- **Medium**



TIC_TRS_A2GTT Timer Test TIC_TRS_A2GTT

Operation :

<a2> > <a3> → <a1>

TIC syntax :

<a1> = <a2> > <a3>

Type :

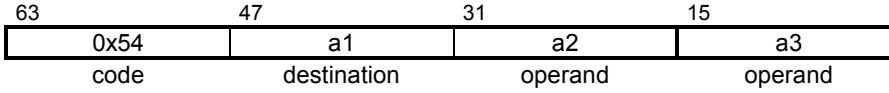
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2LET Timer Test TIC_TRS_A2LET

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

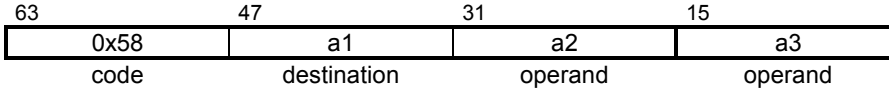
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is less than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2GET Timer Test TIC_TRS_A2GEI

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

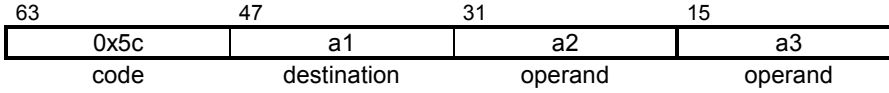
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2EQT Timer Test TIC_TRS_A2EQT

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

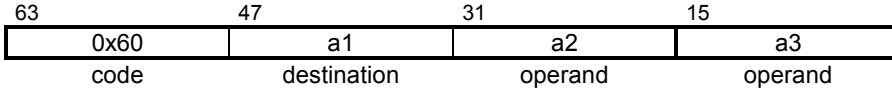
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A2NET Timer Test TIC_TRS_A2NET

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

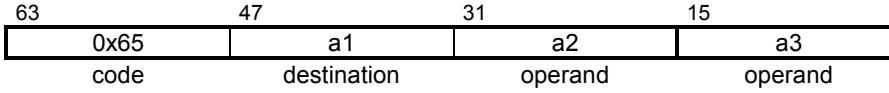
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2LTT Timer Test TIC_OUT_A2LTT

Operation :

<a2> < <a3> → <a1>

TIC syntax :

<a1> = <a2> < <a3>

Type :

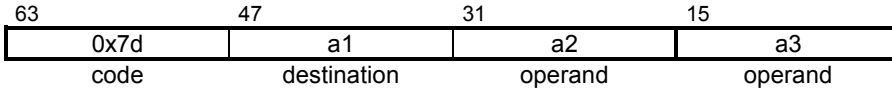
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is less than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2GTT Timer Test TIC_OUT_A2GTT

Operation : <a2> > <a3> → <a1>

TIC syntax : <a1> = <a2> > <a3>

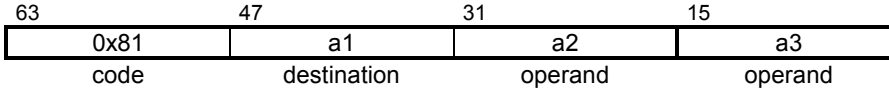
Type :

a1	Boolean
a2	timer
a3	timer

Description : Checks if the operand **<a2>** is greater than the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2LET Timer Test TIC_OUT_A2LET

Operation :

<a2> ≤ <a3> → <a1>

TIC syntax :

<a1> = <a2> <= <a3>

Type :

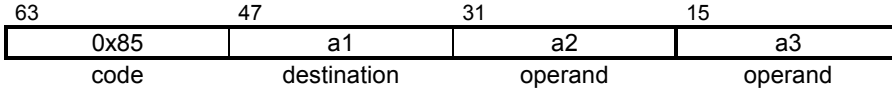
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is less than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2GET Timer Test TIC_OUT_A2GEI

Operation :

<a2> ≥ <a3> → <a1>

TIC syntax :

<a1> = <a2> >= <a3>

Type :

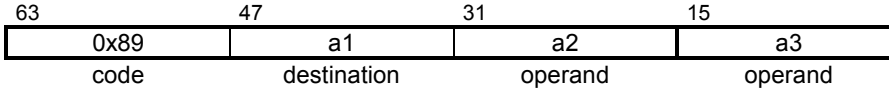
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is greater than or equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2EQT Timer Test TIC_OUT_A2EQT

Operation :

<a2> = <a3> → <a1>

TIC syntax :

<a1> = <a2> == <a3>

Type :

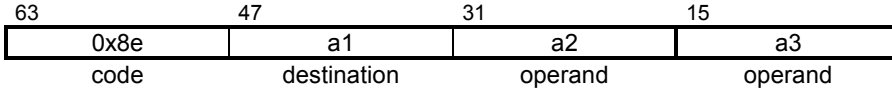
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A2NET Timer Test TIC_OUT_A2NET

Operation :

<a2> <> <a3> → <a1>

TIC syntax :

<a1> = <a2> != <a3>

Type :

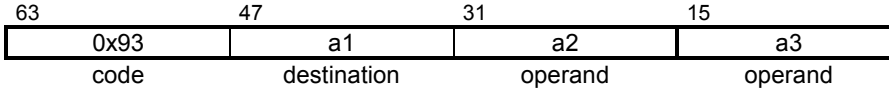
a1	Boolean
a2	timer
a3	timer

Description :

Checks if the operand **<a2>** is not equal to the operand **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A3T Timer Assignment TIC_OUT_A3T

Operation :

<a2> → <a1>

TIC syntax :

<a1> = <a2>

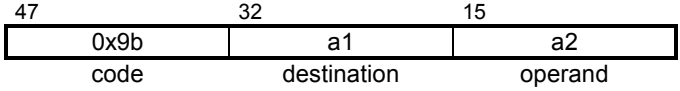
Type :

a1	timer
a2	timer

Description : Stores the operand **<a2>** in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A5T Timer Assignment of TIC_OUT_A5T array element

Operation :

<a2> [<a3>] → <a1>

TIC syntax :

<a1> = <a2> [<a3>]

Type :

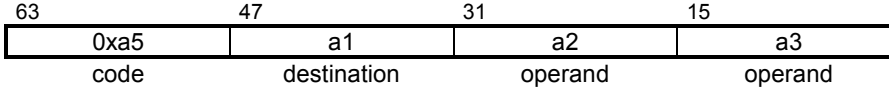
a1	Timer
a2	integer
a3	integer

Description :

Reads an element of the index **<a2>** from the array **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 4. Labels and jumps

The following instructions perform operations on labels. The arguments of a 'goto' instruction are signed offsets from the current code position. The argument of the instruction TIC_L1 (label marker) is not used by the target, and is reserved for future extensions.

TIC_C1B Conditional Jump TIC_C1B

Operation :

if <a1> == TRUE then goto <a2>

TIC syntax :

if <a1> goto <a2>

Type :

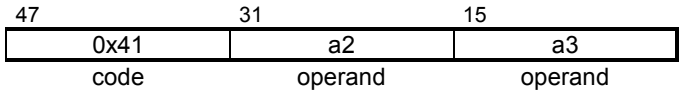
a1	Boolean
a2	integer

Description :

Compares the Boolean value **<a1>** with the Boolean constant TRUE. If the result is true jumps of **<a2>** bytes from the current position.

Format :

- **Medium**



TIC_C2B Conditional Jump TIC_C2B

Operation :

if <a1> == FALSE then goto <a2>

TIC syntax :

if !<a1> goto <a2>

Type :

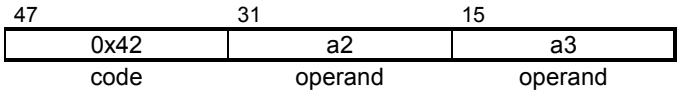
a1	Boolean
a2	integer

Description :

Compares the Boolean value **<a1>** with the Boolean constant FALSE. If the result is true the program continues at **<a2>** bytes from the current position.

Format :

- **Medium**



TIC_I1 Unconditional Jump TIC_I1

Operation : goto <a1>

TIC syntax : goto <a1>

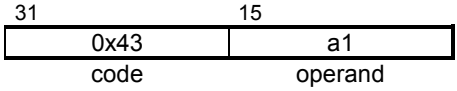
Type :

a1	Integer

Description : The program continues at **<a1>** bytes from the current position.

Format :

- **Medium**



TIC_I2 Return TIC_I2

Operation :

Restore the context of the caller

TIC syntax :

ret

Type :

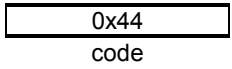
a1	Integer

Description :

The program leaves the function and restores the context of the caller

Format :

- **Medium**



TIC_L1 Label TIC_L1

Operation :

null effect

TIC syntax :

label :

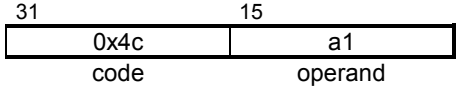
Type :

Description :

This instruction has no effect. It's just a mark. The operand <a1> is reserved for a future extension.

Format :

- **Medium**



E. 5. Parameter passing instructions

The following instructions aer used in a calling program, to push parameters before calling a function or a function block.

E. 5. 1. Integer instructions

TIC_P1I Integer Parameter TIC_P1I

Operation :

pushes <a1> in the stack

TIC syntax :

param <a1>

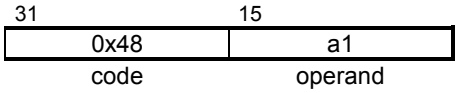
Type :

a1	integer

Description : Pushes the parameter **<a1>** in a stack or memory block.

Format :

- **Medium**



E. 5. 2. Real instructions

TIC_P1R Real Parameter TIC_P1R

Operation :

pushes <a1> in the stack

TIC syntax :

param <a1>

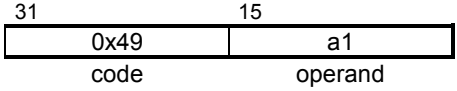
Type :

a1	Real

Description : Pushes the parameter **<a1>** in a stack or memory block.

Format :

- **Medium**



E. 5. 3. Boolean instructions

TIC_P1B Boolean Parameter TIC_P1B

Operation :

pushes <a1> in the stack

TIC syntax :

param <a1>

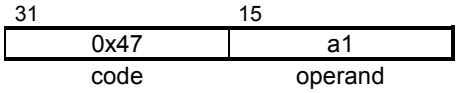
Type :

a1	Boolean

Description : Pushes the parameter **<a1>** in a stack or memory block.

Format :

- **Medium**



E. 5. 4. Message instructions

TIC_P1M Message Parameter TIC_P1M

Operation :

pushes <a1> in the stack

TIC syntax :

param <a1>

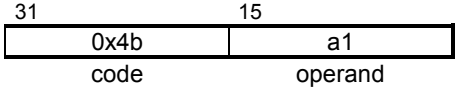
Type :

a1	Message

Description : Pushes the parameter **<a1>** in a stack or memory block.

Format :

- **Medium**



E. 5. 5. Timer instructions

TIC_P1T Timer Parameter TIC_P1T

Operation :

pushes <a1> in the stack

TIC syntax :

param <a1>

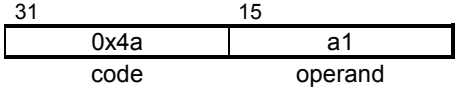
Type :

a1	Timer

Description : Pushes the parameter **<a1>** in a stack or memory block.

Format :

- **Medium**



E. 5. 6. SFC instructions

TIC_P1C SFC Parameter TIC_P1C

Operation :

pushes <a1> in the stack

TIC syntax :

param <a1>

Type :

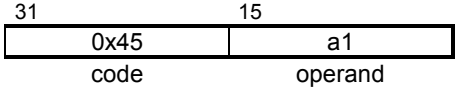
a1	SFC

Description :

Pushes the parameter **<a1>** which is a SFC-child in a stack or memory block.

Format :

- **Medium**



TIC_P1F SFC Parameter TIC_P1F

Operation :

pushes <a1> in the stack

TIC syntax :

param <a1>

Type :

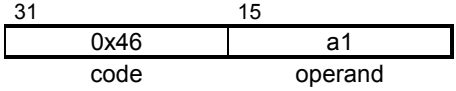
a1	SFC

Description :

Pushes the parameter **<a1>** which is a SFC-parent in a stack or memory block.

Format :

- **Medium**



TIC_P1S SFC Parameter TIC_P1S

Operation :

pushes <a1> in the stack

TIC syntax :

param <a1>

Type :

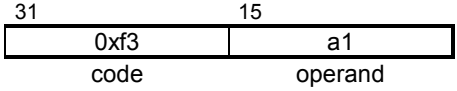
a1	SFC

Description :

Pushes the parameter **<a1>** which is an SFC-parent in a stack or memory block.

Format :

- **Medium**



E. 6. Parameter access from a function

The following instructions occur in the body of a function written in IEC language (ST/IL/LD/FBD) to assign its return parameter. The second argument (#b) is an absolute value which indicates the order number, and is always equal to 0 for a function.

E. 6. 1. Integer instructions

TIC_SYS_GPARI Integer TIC_SYS_GPARI Parameter

Operation :

GetPar (#<a2>) → <a1>

TIC syntax :

<a1> = GetPar (#<a2>)

Type :

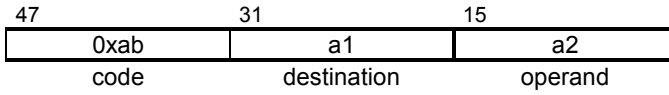
a1	integer
a2	integer

Description :

Reads the parameter number **<a2>** in a stack or memory block and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_SYS_PPARI Integer TIC_SYS_PPARI Parameter

Operation :

PutPar (#<a2>) → <a1>

TIC syntax :

PutPar (<a1>,#<a2>)

Type :

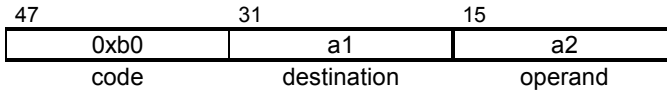
a1	integer
a2	integer

Description :

Reads the parameter number **<a2>** in a stack or memory block and stores the result in the destination location **<a1>.**

Format :

- **Medium**



Notes :

In this case, #<a2> is equal to 0.

E. 6. 2. Real instructions

TIC_SYS_GPARR Real TIC_SYS_GPARR Parameter

Operation :

GetPar (#<a2>) → <a1>

TIC syntax :

<a1> = GetPar (#<a2>)

Type :

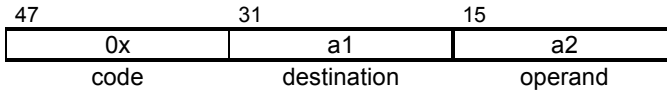
a1	Real
a2	integer

Description :

Reads the parameter number **<a2>** in a stack or memory block and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_SYS_PPARR Real TIC_SYS_PPARR Parameter

Operation :

PutPar (#<a2>) → <a1>

TIC syntax :

PutPar (<a1>,#<a2>)

Type :

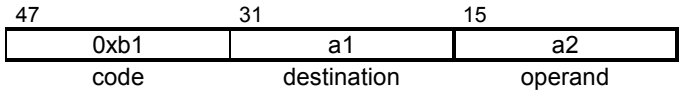
a1	Real
a2	integer

Description :

Reads the parameter number **<a2>** in a stack or memory block and stores the result in the destination location **<a1>.**

Format :

- **Medium**



Notes :

In this case, #<a2> is equal to 0.

E. 6. 3. Boolean instructions

TIC_SYS_GPARB Boolean TIC_SYS_GPARB Parameter

Operation :

GetPar (#<a2>) → <a1>

TIC syntax :

<a1> = GetPar (#<a2>)

Type :

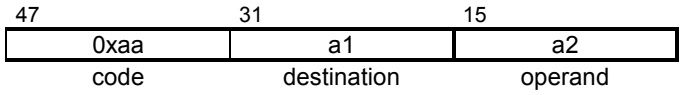
a1	Boolean
a2	integer

Description :

Reads the parameter number **<a2>** in a stack or memory block and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_SYS_PPARB Boolean TIC_SYS_PPARB Parameter

Operation :

PutPar (#<a2>) → <a1>

TIC syntax :

PutPar (<a1>,#<a2>)

Type :

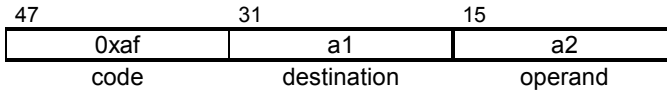
a1	Boolean
a2	integer

Description :

Reads the parameter number **<a2>** in a stack or memory block and stores the result in the destination location **<a1>.**

Format :

- **Medium**



Notes :

In this case, #<a2> is equal to 0.

E. 6. 4. Message instructions

TIC_SYS_GPARM Message TIC_SYS_GPARM Parameter

Operation :

GetPar (#<a2>) → <a1>

TIC syntax :

<a1> = GetPar (#<a2>)

Type :

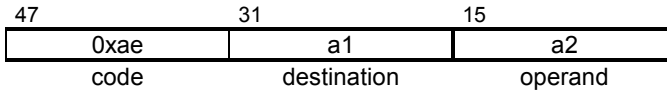
a1	message
a2	integer

Description :

Reads the parameter number **<a2>** in a stack or memory block and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_SYS_PPARM Message TIC_SYS_PPARM Parameter

Operation :

PutPar (#<a2>) → <a1>

TIC syntax :

PutPar (<a1>,#<a2>)

Type :

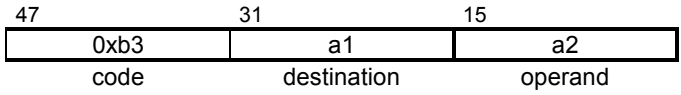
a1	Message
a2	integer

Description :

Reads the parameter number **<a2>** in a stack or memory block and stores the result in the destination location **<a1>.**

Format :

- **Medium**



Notes :

In this case, #<a2> is equal to 0.

E. 6. 5. Timer instructions

TIC_SYS_GPART Timer TIC_SYS_GPART Parameter

Operation :

GetPar (#<a2>) → <a1>

TIC syntax :

<a1> = GetPar (#<a2>)

Type :

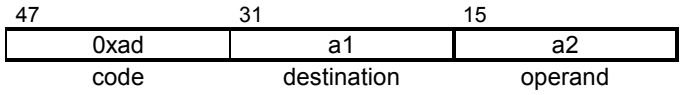
a1	timer
a2	integer

Description :

Reads the parameter number **<a2>** in a stack or memory block and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_SYS_PPART Timer TIC_SYS_PPART Parameter

Operation :

PutPar (#<a2>) → <a1>

TIC syntax :

PutPar (<a1>,#<a2>)

Type :

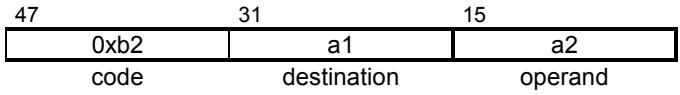
a1	Timer
a2	integer

Description :

Reads the parameter number **<a2>** in a stack or memory block and stores the result in the destination location **<a1>.**

Format :

- **Medium**



Notes :

In this case, #<a2> is equal to 0.

E. 7. Access to FB output parameters

The following instructions are used to access an output parameter of a called function block. A different code is used if the parameter value is used to assign an SFC transition. The second argument (#b) is an absolute value which indicates the order number of the accessed parameter. The last arguments (c and d) are the address and the instance number of a function block.

E. 7. 1. Integer instructions

TIC_SYS_GPARBLKI Integer Output Parameter TIC_SYS_GPARBLKI

Operation : GetFBout (#<a2>, <a3>, <a4>) → <a1>

TIC syntax : <a1> = GetFBout (#<a2>,<a3>,<a4>)

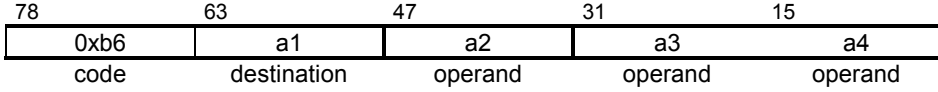
Type :

a1	integer
a2	integer
a3	integer
a4	integer

Description : Reads the output parameter number **<a2>** in a stack or memory block associated to the instance **<a4>** of the function block **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 7. 2. Real instructions

TIC_SYS_GPARBLKR Real Output Parameter TIC_SYS_GPARBLKR

Operation :

GetFBout (#<a2>, <a3>, #<a4>) → <a1>

TIC syntax :

<a1> = GetFBout (#<a2>,<a3>,#<a4>)

Type :

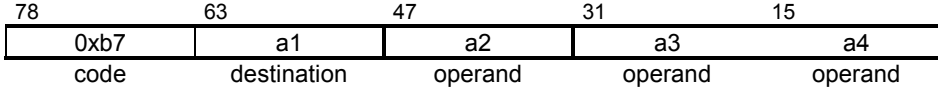
a1	Real
a2	integer
a3	integer
a4	integer

Description :

Reads the output parameter number **<a2>** in a stack or memory block associated to the instance number **<a4>** of the function block **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 7. 3. Boolean instructions

TIC_SYS_GPARBLKB Boolean Output Parameter TIC_SYS_GPARBLKB

Operation :

GetFBout (#<a2>, <a3>, #<a4>) → <a1>

TIC syntax :

<a1> = GetFBout (#<a2>,<a3>,#<a4>)

Type :

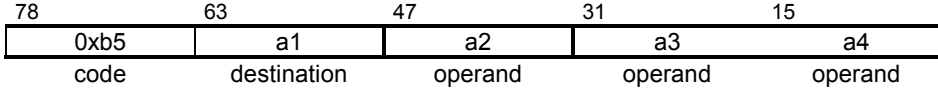
a1	Boolean
a2	integer
a3	integer
a4	integer

Description :

Reads the output parameter number **<a2>** in a stack or memory block associated to the instance number **<a4>** of the function block **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_GPARBLKB Boolean Output Parameter TIC_TRS_GPARBLKB

Operation : GetFBout (#<a2>, <a3>, #<a4>) → <a1>

TIC syntax : <a1> = GetFBout (#<a2>,<a3>,#<a4>)

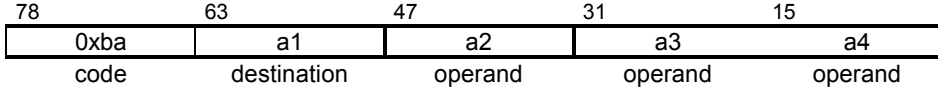
Type :

a1	Boolean
a2	integer
a3	integer
a4	integer

Description : Reads the output parameter number **<a2>** in a stack or memory block associated to the instance number **<a4>** of the function block **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 7. 4. Message instructions

TIC_SYS_GPARBLKM Message Output Parameter TIC_SYS_GPARBLKM

Operation :

GetFBout (#<a2>, <a3>, #<a4>) → <a1>

TIC syntax :

<a1> = GetFBout (#<a2>,<a3>,#<a4>)

Type :

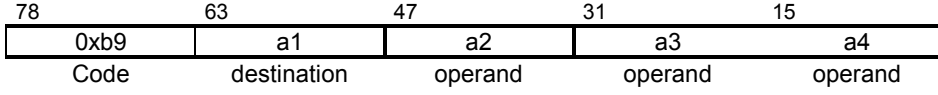
a1	message
a2	integer
a3	integer
a4	Integer

Description :

Reads the output parameter number **<a2>** in a stack or memory block associated to the instance number **<a4>** of the function block **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 7. 5. Timer instructions

TIC_SYS_GPARBLKT Timer Output Parameter TIC_SYS_GPARBLKT

Operation :

GetFBout (#<a2>, <a3>, #<a4>) → <a1>

TIC syntax :

<a1> = GetFBout (#<a2>,<a3>,#<a4>)

Type :

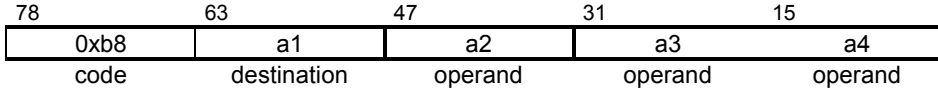
a1	integer
a2	integer
a3	integer
a4	integer

Description :

Reads the output parameter number **<a2>** in a stack or memory block associated to the instance number **<a4>** of the function block **<a3>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 8. Calling function instructions

The following instructions represent a call to a user function written in an IEC language (ST, IL, LD or FBD). The calling parameters of the function have been pushed by 'param' instructions. The second argument (b) is the address of the function.

E. 8. 1. Nil instructions

TIC_A4N Nil Function TIC_A4N

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

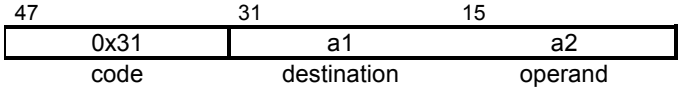
a1	Nil
a2	Integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 8. 2. Integer instructions

TIC_A4I Integer Function TIC_A4I

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

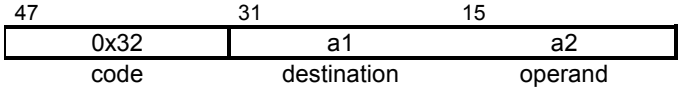
a1	integer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A4I Integer Function TIC_OUT_A4I

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

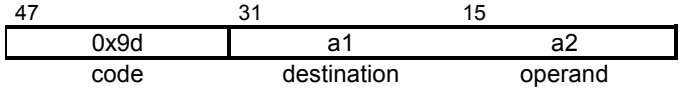
a1	integer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_A4I Integer Function TIC_USF_A4I

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

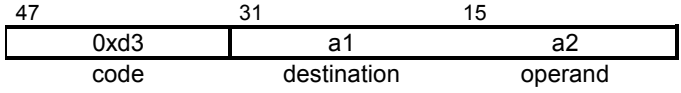
a1	integer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_STD_A4I Integer Function TIC_STD_A4I

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

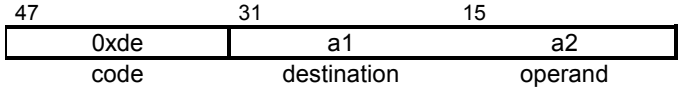
a1	integer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_OUT_A4I Integer TIC_USF_OUT_A4I Function

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

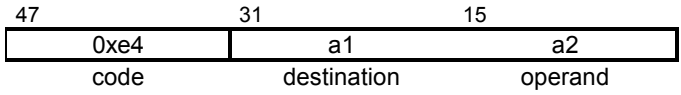
a1	integer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 8. 3. Real instructions

TIC_A4R Real Function TIC_A4R

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

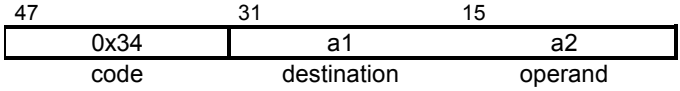
a1	Real
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A4R Real Function TIC_OUT_A4R

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

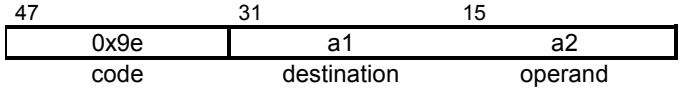
a1	Real
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_A4R Real Function TIC_USF_A4R

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

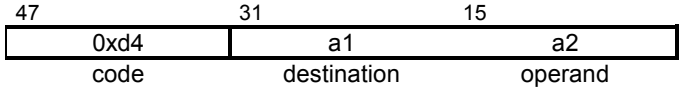
a1	Real
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_OUT_A4R Real TIC_USF_OUT_A4R Function

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

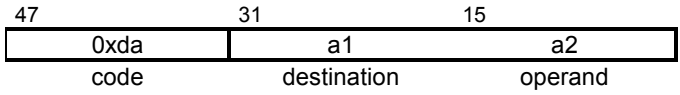
a1	Real
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_STD_A4R Real Function TIC_STD_A4R

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

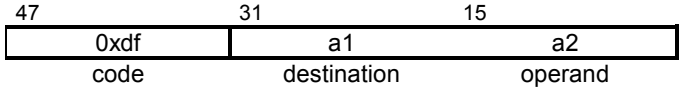
a1	Real
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_STD_OUT_A4R Real TIC_STD_OUT_A4R Function

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

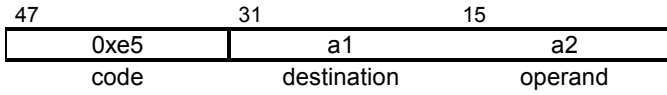
a1	Real
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 8. 4. Boolean instructions

TIC_A4B Boolean Function TIC_A4B

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

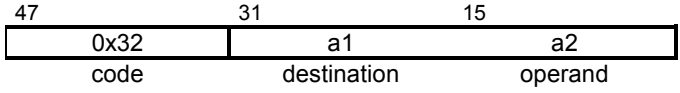
a1	Boolean
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_A4B Boolean Function TIC_TRS_A4B

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

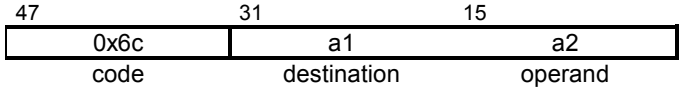
a1	Boolean
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A4B Boolean Function TIC_OUT_A4B

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

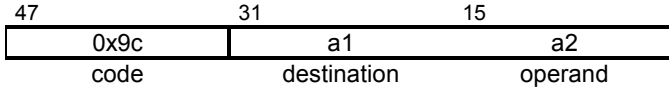
a1	Boolean
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_A4B Boolean Function TIC_USF_A4B

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

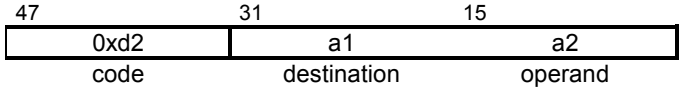
a1	Boolean
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_TRS_A4B Boolean TIC_USF_TRS_A4B Function

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

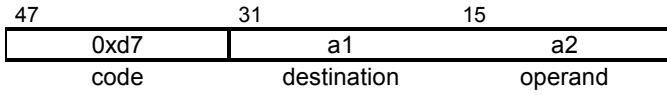
a1	Boolean
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_OUT_A4B Boolean TIC_USF_OUT_A4B Function

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

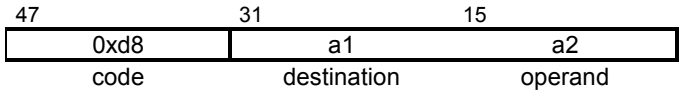
a1	Boolean
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_STD_A4B Boolean Function TIC_STD_A4B

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

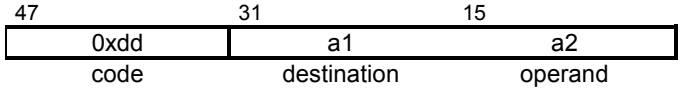
a1	Boolean
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_STD_TRS_A4B Boolean TIC_STD_TRS_A4B Function

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

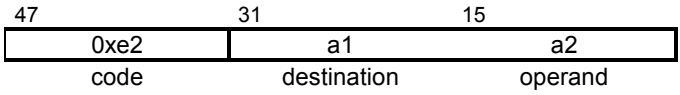
a1	Boolean
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_STD_OUT_A4B Boolean TIC_STD_OUT_A4B Function

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

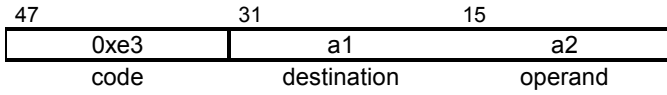
a1	Boolean
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 8. 5. Message instructions

TIC_A4M Message Function TIC_A4M

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

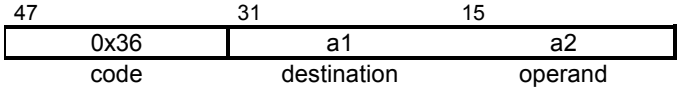
a1	message
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A4M Message Function TIC_OUT_A4M

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

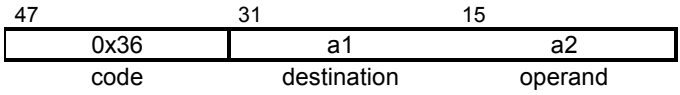
a1	message
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_A4M Message Function TIC_USF_A4M

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

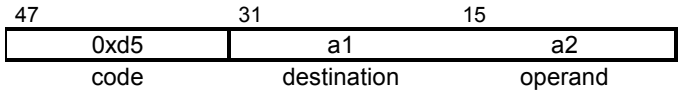
a1	message
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_OUT_A4M Message TIC_USF_OUT_A4M Function

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

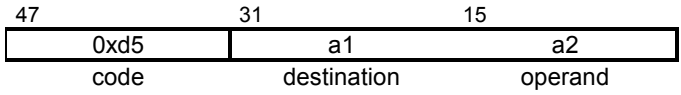
a1	message
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_STD_A4M Message Function TIC_STD_A4M

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

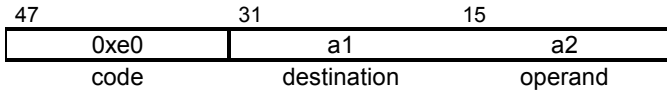
a1	message
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_STD_OUT_A4M Message TIC_STD_OUT_A4M Function

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

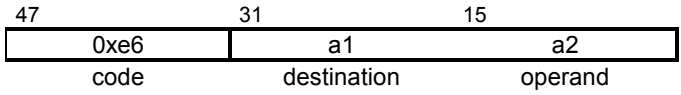
a1	message
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 8. 6. Timer instructions

TIC_A4T Timer Function TIC_A4T

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

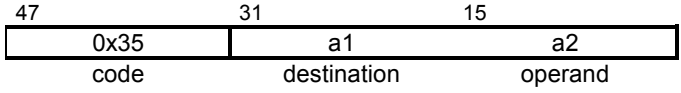
a1	timer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_A4T Timer Function TIC_OUT_A4T

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

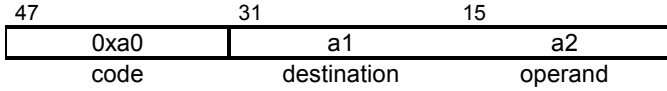
a1	timer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_A4T Timer Function TIC_USF_A4T

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

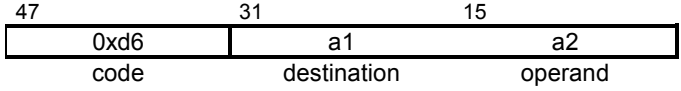
a1	timer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_USF_OUT_A4T Timer TIC_USF_OUT_A4T Function

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

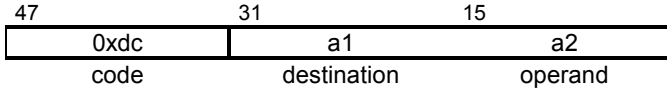
a1	timer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_STD_A4T Timer Function TIC_STD_A4T

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

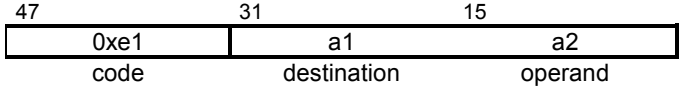
a1	timer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_STD_OUT_A4T Timer Function TIC_STD_OUT_A4T

Operation :

call <a2> → <a1>

TIC syntax :

<a1> = call <a2>

Type :

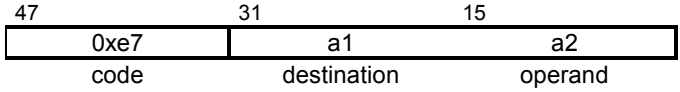
a1	timer
a2	integer

Description :

Calls the function **<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 9. SFC control functions instructions

Such instructions are function calls occuring in a parent SFC program, to control one of its child SFC programs. The address of the child program has been pushed as a parameter.

TIC_SYS_GSTART SFC TIC_SYS_GSTART control

Operation : starts a SFC child

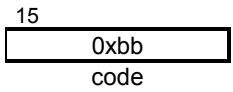
TIC syntax : call GSTART

Type :

Description : Starts the SFC child which the name has been pushed.

Format :

- **Medium**



TIC_SYS_GFREEZE SFC TIC_SYS_GFREEZE control

Operation : freezes a SFC child

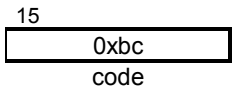
TIC syntax : call GFREEZE

Type :

Description : Freezes the SFC child which the name has been pushed.

Format :

- **Medium**



TIC_SYS_GKILL SFC control TIC_SYS_GKILL

Operation : kills a SFC child

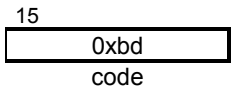
TIC syntax : call GKILL

Type :

Description : Kills the SFC child which the name has been pushed.

Format :

- **Medium**



TIC_SYS_GRST SFC control TIC_SYS_GRST

Operation :

restarts a SFC child

TIC syntax :

call GRST

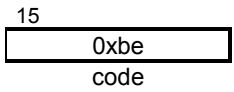
Type :

Description :

Restarts the SFC child which the name has been pushed.

Format :

- **Medium**



E. 10. Activity control timer instructions

Such instructions are function calls occuring in a program, to control the updating of a timer variable. The address of the timer has been pushed as a parameter.

TIC_SYS_TSTART Timer control TIC_SYS_TSTART

Operation : starts a timer

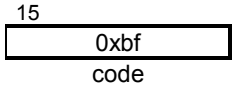
TIC syntax : call TSTART

Type :

Description : Starts the timer which the name has been pushed.

Format :

- **Medium**



TIC_SYS_TSTOP Timer control TIC_SYS_TSTOP

Operation : stops a timer

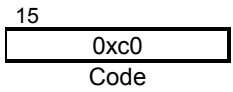
TIC syntax : call TSTOP

Type :

Description : Stops the timer which the name has been pushed.

Format :

- **Medium**



E. 11. Function system instructions

Such instructions are function calls occurring in a program, to system level functions supported by the ISaGRAF target. The parameter addresses have been pushed by 'param' instructions.

E. 11. 1. Integer Instructions

TIC_SYS_GSTATUS Status system TIC_SYS_GSTATUS

Operation :

status of system → <a1>

TIC syntax :

<a1> = call GSTATUS

Type :

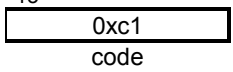
a1	integer

Description :

Reads the status of the system and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_SYS_SYSTEM System call TIC_SYS_SYSTEM

Operation :

call system command→ <a1>

TIC syntax :

<a1> = call SYSTEM

Type :

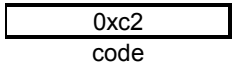
a1	integer

Description :

Calls the function 'system' and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_SYS_OPERATE Operate TIC_SYS_OPERATE

Operation :

calls operate function→ <a1>

TIC syntax :

<a1> = call OPERATE

Type :

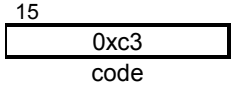
a1	integer

Description :

Calls the ' operate 'function and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_GSTATUS Status TIC_OUT_GSTATUS system

Operation :

status of system → <a1>

TIC syntax :

<a1> = call GSTATUS

Type :

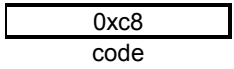
a1	integer

Description :

Reads the status of the system and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_SYSTEM System call TIC_OUT_SYSTEM

Operation :

call system command→ <a1>

TIC syntax :

<a1> = call SYSTEM

Type :

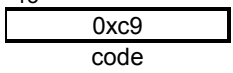
a1	integer

Description :

Calls the function ' system ' and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_OPERATE Operate TIC_OUT_OPERATE

Operation :

calls operate function→ <a1>

TIC syntax :

<a1> = call OPERATE

Type :

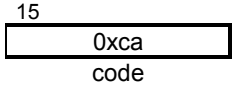
a1	integer

Description :

Calls the ' operate 'function and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 11. 2. Boolean instructions

TIC_SYS_FEDGE Fedge TIC_SYS_FEDGE

Operation :

falling edge state → <a1>

TIC syntax :

<a1> = call FEDGE

Type :

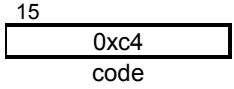
a1	Boolean

Description :

Reads if a falling edge is detected and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_SYS_REDGE Redge TIC_SYS_REDGE

Operation :

rising edge state → <a1>

TIC syntax :

<a1> = call REDGE

Type :

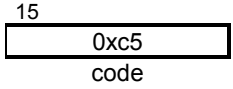
a1	Boolean

Description :

Reads if a rising edge is detected and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_FEDGE Fedge TIC_TRS_FEDGE

Operation :

falling edge state → <a1>

TIC syntax :

<a1> = call FEDGE

Type :

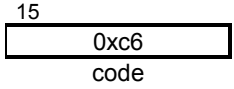
a1	Boolean

Description :

Reads if a falling edge is detected and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_REDGE Redge TIC_TRS_REDGE

Operation :

rising edge state → <a1>

TIC syntax :

<a1> = call REDGE

Type :

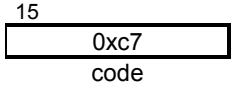
a1	Boolean

Description :

Reads if a rising edge is detected and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_FEDGE Fedge TIC_OUT_FEDGE

Operation :

falling edge state → <a1>

TIC syntax :

<a1> = call FEDGE

Type :

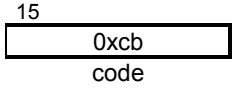
a1	Boolean

Description :

Reads if a falling edge is detected and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_OUT_REDGE Redge TIC_OUT_REDGE

Operation :

rising edge state → <a1>

TIC syntax :

<a1> = call REDGE

Type :

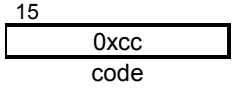
a1	Boolean

Description :

Reads if a rising edge is detected and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 12. Conversion function instructions

The following instructions are used to call 'type conversion' functions. The value to be converted has been pushed as a parameter. The second and third arguments (#b and #c) are absolute values which identifies destination and source types, according to the following numbering: 1=boo / 2=ana / 3=real /4=msg.

TIC_SYS_CNV Convert function TIC_SYS_CNV

Operation :

converts parameter→ <a1>

TIC syntax :

<a1> = call CONVERTION (#<a2>, #<a3>)

Type :

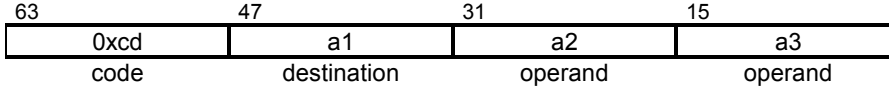
a1	depend of #<a2>
a2	integer
a3	integer

Description :

Converts the parameter which has a type defined by **#<a3>** to a type defined by **#<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



TIC_TRS_CNV Convert function TIC_TRS_CNV

Operation :

converts parameter→ <a1>

TIC syntax :

<a1> = call CONVERTION (#<a2>, #<a3>)

Type :

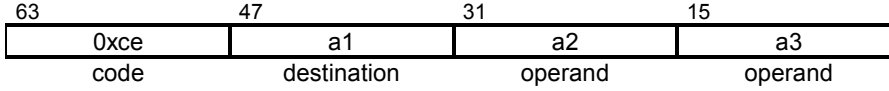
a1	depend of #<a2>
a2	integer
a3	integer

Description :

Converts the parameter which has a type defined by **#<a3>** to a type defined by **#<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



Notes :

#<a2> must be defined as a Boolean.

TIC_OUT_CNV Convert function TIC_OUT_CNV

Operation : converts parameter→ <a1>

TIC syntax : <a1> = call CONVERTION (#<a2>, #<a3>)

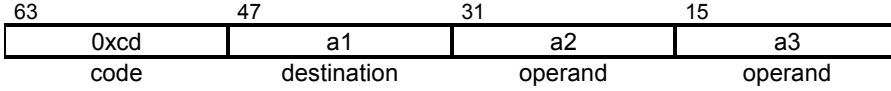
Type :

a1	depend of #<a2>
a2	integer
a3	integer

Description : Converts the parameter which has a type defined by **#<a3>** to a type defined by **#<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 13. Step activity instructions

The following instructions give access to the attributes of an SFC step (activity flag and activity duration). The second argument is the address of the SFC step.

E. 13. 1. Boolean instructions

TIC_SYS_TAC Test Activity TIC_SYS_TAC

Operation :

reads activity of step number #<a2>→ <a1>

TIC syntax :

<a1> = StepAct (#<a2>)

Type :

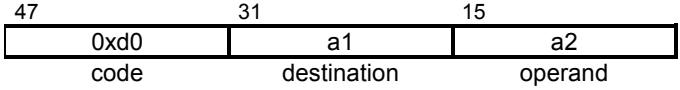
a1	Boolean
a2	integer

Description :

Reads the activity of step number **#<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 13. 2. Timer instructions

TIC_SYS_DAC Duration Activity TIC_SYS_DAC

Operation :

reads duration of step number #<a2>→ <a1>

TIC syntax :

<a1> = StepTime (#<a2>)

Type :

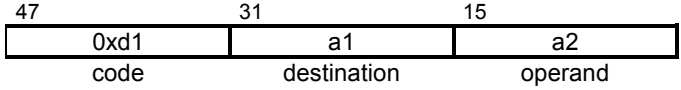
a1	timer
a2	integer

Description :

Reads the duration of step number **#<a2>** and stores the result in the destination location **<a1>.**

Format :

- **Medium**



E. 14. Calling function block instructions

TIC_FBL User C Blocks TIC_FBL

Operation :

Actives instance number #<a2> of the function block address <a1>

TIC syntax :

call <a1>, #<a2>

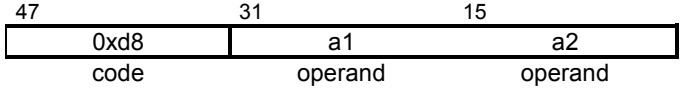
Type :

Description :

Actives instance number #<a2> of the function block address <a1> with the parameters pushed

Format :

- **Medium**



TIC_STD_FBL Standard C Blocks TIC_STD_FBL

Operation : Actives instance number #<a2> of the function block address <a1>

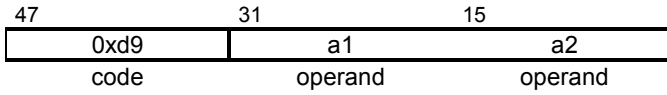
TIC syntax : call <a1>, #<a2>

Type :

Description : Actives instance number #<a2> of the function block address <a1> with the parameters pushed

Format :

- **Medium**



E. 15. End of block instruction

This instruction occurs as a marker to indicate the end of a code sequence. It does not occur in a code sequence for the evaluation of a transition condition.

TIC_END_BLOCK End Code TIC_END_BLOCK

Operation : marks the end of a block of code.

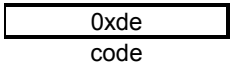
TIC syntax : EndBlock

Type :

Description : Indicates the end of a code sequence.

Format :

- **Medium**



E. 16. Reserved TIC code

These instructions are never generated by the ISaGRAF compiler. They are reserved for the target kernel to perform debug operations on the downloaded code.

TIC_BKP_BEG Reserved TIC_BKP_BEG

Operation :

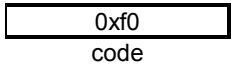
TIC syntax :

Type :

Description :

Format :

- **Medium**



TIC_BKP_END Reserved TIC_BKP_END

Operation :

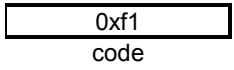
TIC syntax :

Type :

Description :

Format :

- **Medium**



TIC_BKP_TRA Reserved TIC_BKP_TRA

Operation :

.

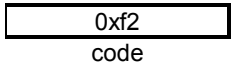
TIC syntax :

Type :

Description :

Format :

- **Medium**



E. 17. Instruction summary

F. Appendix 3 : Communication Protocol definition

F. 1. Overview

This document gives information about the communication protocol implemented between the ISaGRAF Workbench and Target. It only covers the communication requests linked to the programming environment (program downloading and debugging). It does not describe the other communications capabilities of the ISaGRAF target (such as pure Modbus frame handling). The protocol described in this document is the one implemented in ISaGRAF V3.xx. It may evolve to included further capabilities in future releases of the ISaGRAF product.

F. 2. Main format

This section gives an overview of the protocol implemented between ISaGRAF Workbench and target. This document gives the detail of the frames exchanged on a standard serial link. The protocol is based on Modbus protocol, with dedicated function codes. The same frames, at the lowest level, are used in the case of a TCP-IP communication on Ethernet, or when the protocol is customized using the ISDK_COM programming tool.

F. 2. 1. Exchange mechanism

ISaGRAF communication is based on a **master/slave** mechanism. The master is the debugger, the slave is the target system. The rules of exchange are:

- the master sends a question
- the slave answers the question

The slave system never sends data spontaneously. Several slaves can be connected to the same master, if the hardware link enables it. Each ISaGRAF target kernel is identified by an ISaGRAF slave number, used as a Modbus slave number in the standard communication protocol.

F. 2. 2. Frame format

Below is the main format of the frames during a single exchange (question and answer). Each frame has a maximum length of 255 bytes (including header and CRC16). Any number expressed in more than one byte follows the Motorola convention: from the highest significant byte to the lowest significant byte.

```
snd:  [slv][fc][id][pn][No.][info(?)][crc(2)]
rcv:  [slv][fc][id][pn][No.][info(?)][crc(2)]
```

slv (1byte): ..... Modbus-like slave number
fc (1 byte):..... Modbus function code
id (1 byte):..... Exchange identifier
pn (1 byte):..... Packet number (if splitted in several frames)
No. (1 byte): ..... Number of information bytes
info ('No.' bytes): ..... information data
crc (2 bytes): ..... CRC16 checksum

Important: In the case of TCP-IP implementation, each question or answer is embedded in a TCP-IP telegram. Each telegram contains exactly 256 bytes. If the frame is smaller than 256, the trailing transmitted bytes are not relevant and should be ignored.

F. 2. 3. Slave number

The slave number stored in the frames is the ISaGRAF slave number which fully identifies an ISaGRAF target kernel. This is enough for serial link communication. In the case of TCP-IP or customized protocols, this slave number may be completed with the identification of a physical or logical node of the network. The slave number can take any value between 1 and 255. Code 0, reserved in Modbus protocol for broadcasting, is not implemented and is never used in the ISaGRAF protocol.

F. 2. 4. Function codes

The Modbus function codes used for communication between the Workbench and the target are described below. They are extensions of the set of Modbus function codes (expressed here in decimal):

65..... system/application control
67..... application request
68..... application request (see note)
69..... start download
70..... download
71..... unused (reserved for extensions)
72..... unused (reserved for extensions)

Note: The function code 67 is normally used for all application requests. In the case of a request splitted in several packets, the first exchange (1rst packet) uses functuon code 67, and the following exchanges (packets 2 to N) use function code 68.

Other standard Modbus function codes (read/write bits or words) are also supported by the ISaGRAF target, but are not used by the Workbench. Refer to the ISaGRAF Target User's Guide for further information about standard Modbus function code handling on the target.

F. 2. 5. Frame identification

Each exchange (question and answer) is identified by a frame number expressed in one byte and stored in the third byte of the frame. This number is increased by the Workbench (the master) on each new exchange. The frame number is answered by an echo by the target (the slave).

When automatic retries are allowed on the Workbench, the same frame number is used if a single exchange fails.

F. 2. 6. Packet numbering

Because each frame is limited to 255 characters, some messages (questions or answers) can be splitted into many single frames. This also occurs when using download messages. In such a case, the fourth byte of the frame contains the packet number. The following numbering is used:

0..... this is the last packet
FDhex (253)..... other packets are expected
FEhex (254)..... other packets are expected (during file transfer)
FFhex (255)..... exchange must be stopped after this packet
other ..... other packets are expected (during data transfer)

F. 2. 7. CRC16 checksum

All the frames are completed by a CRC16 checksum. ISaGRAF uses an enhanced algorithm using preset tables for the calculation of the CRC16. All the bytes of the frame (including the header) are operated on by the CRC16 calculation. Below is the coding of the CRC16 used by ISaGRAF:

```
#define POLY_CRC16   0xA001

static BYTE TABLE1[256];   /* crc16 values */
static BYTE TABLE2[256];   /* crc16 values */

/*
 crc_init: prepare crc16 preset tables
```

```
 must be called once at init
 (before any crc calculation)
*/

void crc_init (void)
{
   WORD mask, bit, crc, mem;

   for( mask = 0; mask < 0x100; mask++) {
      crc = mask;
      for( bit = 0; bit < 8; bit++) {
         mem = crc & 0x0001 ;
         crc /=  2;
         if ( mem != 0 ) crc ^= POLY_CRC16 ;
      }
      TABLE2[mask] = crc & 0xff;
      TABLE1[mask] = crc >> 8;
   }
}

/*
 crc_make: calculates a crc16
 size: size of the frame (including header)
 buff: pointer to the first byte of the frame
       (slave number)
 return: CRC16 in MOTOROLA format (high byte / low byte)
*/

WORD crc_make (WORD size, BYTE *buff)
{
   BYTE car, i;
   BYTE crc[2];

   crc[0] = 0xff;
   crc[1] = 0xff;
   for ( i = 0; i < size; i++ ) {
      car = buff[i];
      car ^= crc[0];
      crc[0] = crc[1] ^ TABLE2[car];
      crc[1] = TABLE1[car];
   }
   return (*(WORD*)(&crc[0]));
}
```

F. 3. Download exchanges

Download exchanges are used to transfer files from the Workbench to the target. This applies to the following features:

- download the code of the application

- download the symbols of the application
- download new application code for on-line modification

The download communication is divided into three sequential phases:

- begin download (1 exchange)
- download packets (N exchanges)
- end download (1 exchange)

F. 3. 1. Beginning of download

The first exchange of a download identifies the type of data to be downloaded, and gives the target the opportunity to accept or refuse the download. It uses the function code 69:

```
snd:  [slv] [69] [id] [0] [No.] [typ] [name(s)] [0] [size(4)] [crc(2)]
rcv:  [slv] [69] [id] [0] [2] [rc(2)] [crc(2)]
```

slv (1byte): ..... Modbus-like slave number
id (1 byte): ..... Exchange identifier
No. (1 byte): ..... Number of information bytes = (strlen (name) + 6)
typ (1 byte): ..... Type of downloaded file
name (x bytes): ..... Name of the downloaded file (various length)
size (4 bytes): ..... Size of the downloaded file
rc (2 bytes): ..... Return check answered by the target
crc (2 bytes): ..... CRC16 checksum

The null byte stored after the name of the downloaded file is used as a marker to separate the 'name' and 'size' fields. The names used by the Workbench for download commands are:

<projname>**.tel** ..... normal download of the application code
<projname>**.tst** ..... download of the application symbols
<projname>**.upd** ..... download of the code for an on-line modification

The <projname> string is the name of the project such as defined in the Workbench. The 'typ' field is one ascii character which can take one of the following values:

'1' (49dec): ..... normal download of the application code
'2' (50dec): ..... download of the application symbols
'5' (53dec): ..... download for immediate On Line modification
'6' (54dec): ..... download for delayed On Line modification

Values 0, 3 and 4 are reserved for future extensions. The 'rc' field is a return check, expressed in two bytes, answered by the target to accept or refuse the downloading. The possible values for the return check are:
0..... download is accepted
other ..... download s refused

F. 3. 2. Downloading packets

The following exchanges are used to send the contents of the downloaded file. The packet number is used in this case to distinguish the last sent packet. These exchanges use the function code 70:

```
snd:  [slv] [70] [id] [pn] [No.] [data(?)] [crc(2)]
rcv:  [slv] [70] [id] [0] [2] [rc(2)] [crc(2)]
```

slv (1byte): ..... Modbus-like slave number
id (1 byte):..... Exchange identifier
pn (1 byte):..... Packet identifier
No. (1 byte):..... Number of data bytes in the packet
data (No. bytes): ..... Packet of data bytes
rc (2 bytes):..... Return check answered by the target
crc (2 bytes):..... CRC16 checksum

The 'pn' field is used to distinguish the last downloaded packet. It can take one of two values:

0..... this is the last packet
FEhex (254)..... other packets will be sent after this one

The 'rc' field is a return check, expressed in two bytes, answered by the target to accept or refuse the packet. The possible values for the return check are:

0..... OK
other..... invalid packet or error during packet handling

F. 3. 3. End of download

When all the packets have been sent, another single exchange is performed. It indicates that the target must perform the operation on the downloaded data. The operation performed by the target depends on the type of data downloaded. For the downloading of the application code, the target stores the code start of the new application. This exchange uses the function code 67:

```
snd:  [slv] [67] [id] [0] [No.] [63] [typ] [pathname(s)] [0] [crc(2)]
rcv:  [slv] [67] [id] [0] [3] [63] [rc(2)] [crc(2)]
```

slv (1byte): ..... Modbus-like slave number
id (1 byte):..... Exchange identifier
No. (1 byte):..... Number of information bytes = (strlen (pathname) + 2)
typ (1 byte):..... Type of downloaded file
pathname (x bytes): ..... Pathame of the downloaded file (various length)
size (4 bytes): ..... Size of the downloaded file
rc (2 bytes):..... Return check answered by the target
crc (2 bytes):..... CRC16 checksum

The 'pathname' contains the same string as the 'name' indicated at the beginning of the download. It can include a full pathname in future extensions of the protocol. The pathname is always followed by a null byte. The 'typ' field is one ascii character which can take one of the following values:

1:..... normal download of the application code
2:..... download of the application symbols
5:..... download for immediate On Line modification
6:..... download for delayed On Line modification

Values 0, 3 and 4 are reserved for future extensions. The 'rc' field is a return check, expressed in two bytes, answered by the target to indicate whether the requested end of download has been correctly performed. The possible values for the return check are:

0..... OK
other ..... operation has failed

F. 4. Requests used for Upload

This section details the format of the communication requests used to upload EZS (Embedded Source Code) from ISaGRAF 3.3 Target to ISaGRAF 3.3 Workbench. The implementation introduces a generic request used to upload any kind of target object. The EZS upload is implemented through ISaGRAF resources. This is an application request sent through function code #67, according to the standard ISaGRAF format.

Request ID = **64** (decimal) - Name = **GETOBJECT**

Question:

slv	67	fn	0	nb	**64**	**...**	**info**	**...**	Crc(2)

Answer:

slv	67	fn	0	nb	**64**	**Rc(2)**	**...**	**info**	**...**	crc(2)

Arguments:
slv ..... slave number
fn..... frame number (always changing)
nb..... nb of bytes in request (request id + info)
crc..... checksum
rc..... request return check
info..... arguments

Below are format and description of the request arguments:

Question:

64	size	offset	ident

1	1	4	<=31 + \0	(bytes)

Answer

64	rc	size	offset	Ident	data	
1	2	1	4	<=31 + \0	<=128	(bytes)

Arguments

size ..... number of bytes to read (cannot exceed 128 bytes)
or 0 to get object length (4 bytes returned in this case)
offset..... offset of the first byte to read
ident..... object identification (type and name)
text string (max 31 characters) plus null termination character
data..... read data (cannot exceed 128 bytes)
or object length if size is 0
rc..... 0=ok
1=invalid object
2=invalid size
3=invalid offset

Below is the standard format of the "object identification" used in communication protocol to identify a resource. All names are expressed in uppercase letters. The total string cannot exceed 31 characters

`<type>[:<name>]`

Below are the predefined object types:

type	*name*	*description*
R	**resource name**	application resource "EZS" is the name of embedded source code
		(others to be defined in the future)

Examples:

`R:EZS` ..... resource used to store zipped source code
`R:MYRES` ..... user defined resource

F. 5. Control exchanges

Control exchanges are used to control or monitor the state of the target PLC (start / stop application), and to monitor the state of the application which is actually running in the PLC (run, stop, breakpoint...). All these requests use the function code 65:

`snd: [slv]` **`[65]`** `[id][0][No.]` **`[rq][args`** `(?)` **`]`** `[crc(2)]`

```
rcv: [slv][65][id][0][No.][rq][rc(2)][data(?)][crc(2)]
```

slv (1byte):..... Modbus-like slave number
id (1 byte):..... Exchange identifier
No. (1 byte):..... Number of information bytes (not including CRC)
rq (1 byte):..... Control request code
args (x bytes):..... Request arguments (various length)
rc (2 bytes):..... Return check answered by the target
data (x bytes):..... Answered data (various length)
crc (2 bytes):..... CRC16 checksum

The following sections of this chapter details the contents of questions and answers, for the fields 'rq', 'rc', 'args' and 'data' only.

F. 5. 1. Initiate communication

This exchange is the first sent by the debugger to open a communication session. It is mainly used by the target to initiate each of its communictaion layers. The request code is 0. There is no argument for this request.

```
snd: [0]
rcv: [0][rc(2)]
```

The 'rc' field is a return check expressed in two bytes. Its value is **0** if the request has been succesfully executed.

F. 5. 2. Control PLC

This exchange is used to query the name and state of the application installed in the PLC. It is also used to start or stop the application (commmands of the Files menu in the debugger window). The request code is 10 (decimal). When the debugger queries the name and state of the current application the exchanged frames are:

```
snd: [10][0][0]
rcv: [10][rc(2)][0][state][vers(4)][len][name(s)][0]
```

rc (2 bytes):..... Return check answered by the target
state (1 byte):..... Status of the application (if exist)
vers (4 bytes):..... Version number of the communication protocol
len (1 byte):..... Length of the application name (if exist)
name (len bytes):..... Name of the application (if exist)

The state of the application installed on the PLC can take one of the following values:

0..... No application or application inactive
3..... Application running in the PLC

When no application is running, the application name is empty. The 'len' field is equal to 0, and the null byte after the 'name' field still exists. Examples of typical answers are:

```
rcv: [10][rc][0][0][vers][0][0]          = no application
rcv: [10][rc][0][3][vers][3]['ABC'][0] = appli 'ABC' running'
```

The 'vers' field is a 4 byte string containing the version number of the communication protocol. Current version number is '2.00'. The frames exchanged when the debugger send a 'STOP' command to deactivate an application are:

```
snd: [10][0][1]
rcv: [10][rc(2)]
```

The frames exchanged when the debugger send a 'START' command to activate an application are:

```
snd: [10][0][2][len][name(s)][0]
rcv: [10][rc(2)]
```

len (1 byte):..... Length of the name of the application to be started
name (len bytes):..... Name of the application to be started

The 'rc' field is a return check expressed in two bytes. Its value is **0** if the request has been succesfully executed.

F. 5. 3. Monitor application state

Monitor exchanges are used by the Workbench to query the state of the application which is currently running on the PLC. The request code is 20 (decimal):

```
snd: [20]
rcv: [20][rc(2)][state]
```

rc (2 bytes):..... Return check answered by the target
state (1 byte):..... Running application state

The application state can take one of the following values:

0..... Application is in normal 'real time' mode
1..... Application is in 'cycle to cycle' mode
2..... Application is stopped on a breakpoint
255..... Application is aborted due to a fatal error

The 'rc' field is a return check expressed in two bytes. Its value is **0** if the request has been succesfully executed.

F. 6. Application level requests

Application level exchanges are used to monitor the components of the application (variables, programs, steps, transitions...). They also enable actions on application objects (write variable, set breakpoint...).

Such requests are always relative to the application currently running on the PLC. All these requests use the function code 67:

```
snd:  [slv] [67] [id] [0] [No.] [rq] [args(?)] [crc(2)]
rcv:  [slv] [67] [id] [0] [No.] [rq] [rc(2)] [data(?)] [crc(2)]
```

slv (1byte): ..... Modbus-like slave number
id (1 byte):..... Exchange identifier
No. (1 byte):..... Number of information bytes (not including CRC)
rq (1 byte): ..... Control request code
args (x bytes):..... Request arguments (various length)
rc (2 bytes):..... Return check answered by the target
data (x bytes):..... Answered data (various length)
crc (2 bytes):..... CRC16 checksum

The 'rc' field is a return check expressed in two bytes. Its value is **0** if the request has been succesfully executed. Most of these requests have access to the ISaGRAF application object. The identification of an object is made through its **VA** (**Virtual Address**) which is the logical address of the ISaGRAF in its internal application Database. VAs are expressed in two bytes. The most significant 4 bits indicate the type of the object. The other bits contain the index of the object. Below is the coding of the ISaGRAF object types:

1..... Boolean variable
2..... Analog variable
3..... Timer variable
4..... Message string variable
5..... Program
6..... Real (floating pointing) analog variable
7..... SFC step
8..... SFC transition
13..... (reserved for extensions)
15..... System level information

The following sections of this chapter details the content of questions and answers for the fields 'rq', 'rc', 'args' and 'data' only. The list of supported requests is:

R_BKT ..... 23..... set breakpoint
R_WILR..... 24..... (reserved for extensions)
R_WBOO..... 25..... write boolean variable
R_WANA ..... 26..... write analog variable
R_WTMR..... 27..... write timer variable
R_WMSG ..... 28..... write message variable

R_TSTART ..... 30 ..... start timer
R_TSTOP ..... 31 ..... stop timer
R_CC ..... 32 ..... set cycle by cycle mode
R_RT ..... 33 ..... set real time mode
R_EC ..... 34 ..... execute one cycle
R_TUC ..... 35 ..... transition unconditional clearing
R_TCC ..... 36 ..... transition conditional clearing
R_BKDEL ..... 40 ..... remove breakpoint
R_TCW ..... 43 ..... set cycle timing
R_TCR ..... 44 ..... read cycle timing
R_GSTART ..... 45 ..... start SFC program
R_GFREEZE ..... 46 ..... freeze SFC program
R_GKILL ..... 47 ..... kill SFC program
R_GRST ..... 48 ..... restart SFC program
R_GER ..... 53 ..... get last application error
R_UNLOCK ..... 55 ..... unlock I/O variable
R_LOCK ..... 57 ..... lock I/O variable
R_LOCKR ..... 70 ..... get list of locked I/O variables
R_APPL ..... 72 ..... get application identification
R_RANY ..... 80 ..... read a list of variables (all types)
R_UPMDF ..... 62 ..... realize delayed On-Line modification
R_WLIST ..... 83 ..... (reserved for extensions)

F. 6. 1. R_BKT: set breakpoint

This request is sent by the debugger to set a breakpoint on a step or a transition. Its request code is 23. Below is the format of question and answers:

```
snd:  [23][va(2)][typ][ln(2)]
rcv:  [23][rc(2)]
```

va (2 bytes): ..... VA of the concerned step or transition
typ (1 byte): ..... Type of breakpoint
ln (2 bytes): ..... (reserved for extensions)

The 'typ' field indicates the type of breakpoint to be applied. It can take one of the following values:

1 ..... breakpoint on step activation
2 ..... breakpoint on step de-activation
3 ..... breakpoint on transition clearing
4 ..... (reserved for extensions)

F. 6. 2. R_WBOO: write boolean variable

This request is sent by the debugger to force the value of a boolean variable. Its request code is 25. The format of questions and answers is:

```
snd:  [25][va(2)][act][1][value]
rcv:  [25][rc(2)]
```

va:..... VA of the concerned boolean variable
act (1 byte):..... Type of action to be performed
value (1 byte):..... Boolean value (1=true / 0=false)

The 'act' field is the type of action that must be performed, according to the attribute of the variable:

0:..... internal variable
2:..... input variable (never used)
3:..... output variable (I/O device must be updated)

F. 6. 3. R_WANA: write analog variable

This request is sent by the debugger to force the value of an analog (integer or real) variable. Its request code is 26. Below is the format of question and answers:

```
snd:  [26][va(2)][act][1][value(4)]
rcv:  [26][rc(2)]
```

va:..... VA of the analog variable concerned
act (1 byte):..... Type of action to be performed
value (4 bytes):..... Analog (integer or floating pointing) value

The 'act' field is the type of action that must be performed, according to the attribute of the variable:

0:..... internal variable
2:..... input variable (never used)
3:..... output variable (I/O device must be updated)

F. 6. 4. R_WTMR: write timer variable

This request is sent by the debugger to force the value of a timer variable. Its request code is 27. Below is the format of question and answers:

```
snd:  [27][va(2)][0][1][value(4)]
rcv:  [27][rc(2)]
```

va:..... VA of the concerned timer variable
value (4 bytes):..... Time value (expressed in milliseconds)

F. 6. 5. R_WMSG: write message variable

This request is sent by the debugger to force the value of a message variable. Its request code is 28. Below is the format of question and answers:

```
snd: [28][va(2)][act][1][No.c][value(s)][0]
rcv: [28][rc(2)]
```

va: ..... VA of the concerned analog variable
act (1 byte): ..... Type of action to be performed
No.c (1 byte): ..... Number of character in string value
value (No.c bytes): ..... String value (characters)

The string value is always followed by a null byte. The 'act' field is the type of action that must be performed, according to the attribute of the variable:

0: ..... internal variable
2: ..... input variable (never used)
3: ..... output variable (I/O device must be updated)

F. 6. 6. R_TSTART: start timer

This request is sent by the debugger to start a timer variable. Its request code is 30. The format of questions and answers is:

```
snd: [30][1][va(2)]
rcv: [30][rc(2)]
```

va (2 bytes): ..... VA of the timer variable to be started

F. 6. 7. R_TSTOP: stop timer

This request is sent by the debugger to stop a timer variable. Its request code is 31. The format of questions and answers is:

```
snd: [31][1][va(2)]
rcv: [31][rc(2)]
```

va (2 bytes): ..... VA of the timer variable to be stopped

F. 6. 8. R_CC: set cycle by cycle mode

This request is sent by the debugger to set the 'cycle to cycle' execution mode. Its request code is 32. The format of questions and answers is:

```
snd: [32]
```

```
rcv:  [32][rc(2)]
```

F. 6. 9. R_RT: set real time mode

This request is sent by the debugger to set the 'real time' execution mode. Its request code is 33. The format of questions and answers is:

```
snd:  [33]
rcv:  [33][rc(2)]
```

F. 6. 10. R_EC: execute one cycle

This request is sent by the debugger to execute one cycle, when the target is in 'cycle to cycle' mode. Its request code is 34. The format of questions and answers is:

```
snd:  [34]
rcv:  [34][rc(2)]
```

F. 6. 11. R_TUC: transition unconditional clearing

This request is sent by the debugger to force the unconditional clearing of an SFC transition. Its request code is 35. The format of questions and answers is:

```
snd:  [35][1][va(2)]
rcv:  [35][rc(2)]
```

va (2 bytes):..... VA of the SFC transition to be cleared

F. 6. 12. R_TCC: transition conditional clearing

This request is sent by the debugger to force the conditional clearing of an SFC transition. Its request code is 36. The format of questions and answers is:

```
snd:  [36]  [1][va(2)]
rcv:  [36][rc(2)]
```

va (2 bytes):..... VA of the SFC transition to be cleared

F. 6. 13. R_BKDEL: remove breakpoint

This request is sent by the debugger to remove a breakpoint currentlt installed on an SFC step or transition. Its request code is 40. The format of questions and answers is:

```
snd:  [40][1][va(2)]
```

```
rcv:  [40][rc(2)]
```

va (2 bytes): ..... VA of step or transition where breakpoint is installed

This request can also be used to remove all the existing installed breakpoints, using the following format:

```
snd:  [40][0]
rcv:  [40][rc(2)]
```

F. 6. 14. R_TCW: set cycle timing

This request is sent by the debugger to force the application cycle timing (sampling period). Its request code is 43. The format of questions and answers is:

```
snd:  [43][timing(4)]
rcv:  [43][rc(2)]
```

timing (4 bytes): ..... New cycle timing

F. 6. 15. R_TCR: read cycle timing

This request is sent by the debugger to read the application cycle timing components. Its request code is 44. The format of questions and answers is:

```
snd:  [44]
rcv:  [44][rc(2)][smp(4)][curr(4)][max(4)][over(4)]
```

smp (4 bytes): ..... Programmed time (sampling period) in milliseconds
curr (4 bytes):..... Last measured cycle timing in milliseconds
max (4 bytes): ..... Maximum detected cycle timing in milliseconds
over (4 bytes):..... Number of cycle timing overflows

F. 6. 16. R_GSTART: start SFC program

This request is sent by the debugger to start an SFC program. Its request code is 45. The format of questions and answers is:

```
snd:  [45][1][pnum]
rcv:  [45][rc(2)]
```

pnum (1 byte):..... SFC program number

The program number is the lower byte of its VA, and does not include the 'program' object type. There is a maximum of 255 SFC programs in an application.

F. 6. 17. R_GFREEZE: freeze SFC program

This request is sent by the debugger to freeze an SFC program. Its request code is 46. The format of questions and answers is:

```
snd:  [46][1][pnum]
rcv:  [46][rc(2)]
```

pnum (1 byte): ..... SFC program number

The program number is the lower byte of its VA, and does not include the 'program' object type. There is a maximum of 255 SFC programs in an application.

F. 6. 18. R_GKILL: kill SFC program

This request is sent by the debugger to kill an SFC program. Its request code is 47. The format of questions and answers is:

```
snd:  [47][1][pnum]
rcv:  [47][rc(2)]
```

pnum (1 byte): ..... SFC program number

The program number is the lower byte of its VA, and does not include the 'program' object type. There is a maximum of 255 SFC programs in an application.

F. 6. 19. R_GRST: restart SFC program

This request is sent by the debugger to restart a frozen SFC program. Its request code is 48. The format of questions and answers is:

```
snd:  [48][1][pnum]
rcv:  [48][rc(2)]
```

pnum (1 byte): ..... SFC program number

The program number is the lower byte of its VA, and does not include the 'program' object type. There is a maximum of 255 SFC programs in an application.

F. 6. 20. R_GER: get last application error

This request is sent by the debugger to read the code of the last detected application error. Its request code is 53. The format of questions and answers is:

```
snd:  [53]
rcv:  [53][rc(2)][code(2)][arg(2)]
```

code (2 bytes): ..... Error code number (0 if no error detected)
arg (2 bytes):..... Error argument (VA or numerical info)

F. 6. 21. R_UNLOCK: unlock I/O variable

This request is sent by the debugger to unlock an I/O variable. Its request code is 55. The format of questions and answers is:

```
snd: [55] [1] [va(2)]
rcv: [55] [rc(2)]
```

va (2 bytes): ..... VA of the I/O variable to be unlocked

The same request can also be used to unlock all the currently locked I/O variable, using the following format:

```
snd: [55] [0]
rcv: [55] [rc(2)]
```

F. 6. 22. R_LOCK: lock I/O variable

This request is sent by the debugger to lock an I/O variable. Its request code is 57. The format of questions and answers is:

```
snd: [57] [1] [va(2)]
rcv: [57] [rc(2)]
```

va (2 bytes): ..... VA of the I/O variable to be locked

F. 6. 23. R_LOCKR: get list of locked I/O variables

This request is sent by the debugger to get the list of thhe currently locked I/O variables. Its request code is 70 The format of questions and answers is:

```
snd: [70]
rcv: [70] [rc(2)] [No.(2)] { [va(2)] }
```

No. (2 bytes): ..... number of locked variables
va (2 bytes): ..... VA of a locked I/O variable
(the frame contains a list of No. VAs)

F. 6. 24. R_APPL: get application identification

This request is sent by the debugger to get the identification of the application currently running in the PLC. Its request code is 72. The format of questions and answers is:

```
snd: [72]
rcv: [72][rc(2)][vers(2)][dttm(4)][crc(4)]
```

vers (2 bytes):..... Application code version number
dttm (4 bytes):..... Date and time of code generation
crc (4 bytes):..... CRC of the application symbols

F. 6. 25. R_RANY: read a list of variables (all types)

This request is sent by the debugger to read the values of a list of objects from all types. Its request code is 80. The format of questions and answers is:

```
snd: [rq][No.] {[va(2)][act]}
rcv: [rq][rc(2)][AB(hex)][CD(hex)][No.]{[type][value(?)]}
```

No. (1 byte):..... number of variables
act (1 byte):..... Type of action to be performed
va (2 bytes):..... VA of a locked I/O variable
(the frame contains a list of No. [act+va] fields)
type (1 byte):..... Type of value (see below)
value (X bytes):..... Value of an object (various length: see format below)
(the frame contains a list of No. [type+value] fields)

The 'act' field is the type of action that must be performed, according to the attribute of the variable. This field is 0 if the object to be read is not a variable.

0:..... internal variable (or not a variable)
1:..... output variable
3:..... input variable

The format and the meaning of a value depends on its type. The following table gives the relation ship between the data type (expressed in the 'type' field), the length of the 'value' field, and its meaning

type	*length*	*meaning*
1	1	boolean value (0=false / 1=true)
2	4	analog (integer or floating pointing) value
3	4	timer value (expressed in milliseconds)
4	2+No.c	string value: length + No.c characters + null byte
5	1	SFC program state (4=inactive / 2=active / 6=frozen)
7	1	step: bit 0(mask 01hex) = step activity flag bit 7(mask 80hex) = breakpoint on activation bit 6(mask 40hex) = breakpoint on de-activation

8..... 1.....transition: bit 0(mask 01hex) = transition validity
bit 7(mask 80hex) = breakpoint
13..... 8.....(reserved for extensions)
15..... 4.....detected error code and argument (see GER request)
("f000" is used as a VA to query detected errors)

F. 6. 26. R_UPMDF: realize delayed On-Line modification

This request is sent by the debugger to realize a delayed On Line modification (corresponds to the 'realize update' command of the debugger 'Files' menu). Its request code is 62. The format of questions and answers is:

```
snd: [62]
rcv: [62][rc(2)]
```

F. 7. Summary

Start download:

```
snd:  [slv][69][id][0][No.][typ][..name..][0][size(4)][crc(2)]
rcv:  [slv][69][id][0][2][rc(2)][crc(2)]
```

Downloading packets:

```
snd:  [slv][70][id][pn][No.][..data..][crc(2)]
rcv:  [slv][70][id][0][2][rc(2)][crc(2)]
```

End of download:

```
snd:  [slv][67][id][0][No.][63][typ][pathname(2)][0][crc(2)]
rcv:  [slv][67][id][0][3][63][rc(2)][crc(2)]
```

Control exchanges: Main format:

```
snd:  [slv][65][id][0][No.][rq][...args...][crc(2)]
rcv:  [slv][65][id][0][No.][rq][rc(2)][...data...][crc(2)]
```

Control exchanges: Request format:

```
Init:                       snd:  [0]
                            rcv:  [0][rc(2)]
Read PLC status:            snd:  [10][0][0]
(no appli)                  rcv:  [10][rc][0][0][vers(4)][0][0]
Read PLC status:            snd:  [10][0][0]
(appli active)              rcv:  [10][rc(2)][0][3]
                                  [vers(4)][len][name(s)][0]
Stop appli:                 snd:  [10][0][1]
                            rcv:  [10][rc(2)]
Activate appli:             snd:  [10][0][2][len][name(s)][0]
                            rcv:  [10][rc(2)]
Read status:                snd:  [20]
                            rcv:  [20][rc(2)][state]
```

Application level exchanges: Main format:

```
snd:  [slv][67][id][0][No.][rq][...args...][crc(2)]
rcv:  [slv][67][id][0][No.][rq][rc(2)][...data...][crc(2)]
```

Application level exchanges: Request format:

```
R_WBOO:                 snd: [25][va(2)][act][1][value]
                        rcv: [25][rc(2)]
R_WANA:                 snd: [26][va(2)][act][1][value(4)]
                        rcv: [26][rc(2)]
R_WTMR:                 snd: [27][va(2)][0][1][value(4)]
                        rcv: [27][rc(2)]
R_WMSG:                 snd: [28][va(2)][act][1]
                             [No.c][value(s)][0]
                        rcv: [28][rc(2)]

R_TSTART:               snd: [30][1][va(2)]
                        rcv: [30][rc(2)]
R_TSTOP:                snd: [31][1][va(2)]
                        rcv: [31][rc(2)]

R_CC:                   snd: [32]
                        rcv: [32][rc(2)]
R_RT:                   snd: [33]
                        rcv: [33][rc(2)]
R_EC:                   snd: [34]
                        rcv: [34][rc(2)]

R_TUC:                  snd: [35][1][va(2)]
                        rcv: [35][rc(2)]
R_TCC:                  snd: [36][1][va(2)]
                        rcv: [36][rc(2)]

R_BKT:                  snd: [23][va(2)][typ][ln(2)]
                        rcv: [23][rc(2)]
R_BKDEL:                snd: [40][1][va(2)]
                        rcv: [40][rc(2)]
R_BKDEL all:            snd: [40][0]
                        rcv: [40][rc(2)]

R_TCW:                  snd: [43][timing(4)]
                        rcv: [43][rc(2)]
R_TCR:                  snd: [44]
                        rcv: [44][rc(2)]
                             [smp(4)][cur(4)][max(4)][ovr(4)]

R_GSTART:               snd: [45][1][pnum]
                        rcv: [45][rc(2)]
R_GFREEZE:              snd: [46][1][pnum]
                        rcv: [46][rc(2)]
R_GKILL:                snd: [47][1][pnum]
                        rcv: [47][rc(2)]
R_GRST:                 snd: [48][1][pnum]
                        rcv: [48][rc(2)]

R_GER:                  snd: [53]
                        rcv: [53][rc(2)][code(2)][arg(2)]
```

```
R_UNLOCK:                 snd: [55][1][va(2)]
                          rcv: [55][rc(2)]
R_UNLOCK all:             snd: [55][0]
                          rcv: [55][rc(2)]
R_LOCK:                   snd: [57][1][va(2)]
                          rcv: [57][rc(2)]
R_LOCKR:                  snd: [70]
                          rcv: [70][rc(2)][No.(2)] { [va(2)] }

R_APPL:                   snd: [72]
                          rcv: [72][rc(2)]
                               [vers(2)][dttm(4)][crc(4)]

R_UPMDF:                  snd: [62]
                          rcv: [62][rc(2)]

R_RANY:                   snd: [rq][No.] { [va(2)] [act]}
                          rcv: [rq][rc(2)][AB][CD]
                               [No.] { [typ][values(?)] }
```

R_RANY: format of values:

```
1=boolean                 [boo_val] (0=false / 1=true)
2=analog                  [anaval(4)]
3=timer                   [tmrval(4)]
4=string                  [len][characters(s)][0]
5=program                 [state]
                          (0=inact. / 1=act. / 3= frozen)
7=step                    [state]
                          (80h=bk_act | 40h=bk_de | 01h=activity)
8=transition              [state] (80h=breakpoint | 01h=validity)
15=appli error            [code(2)][arg(2)]
```

G. Appendix 4 : C code definition

G. 1. Introduction

ISaGRAF V3.04 or later allows the generation of various target codes. One of them is the C code which can be compiled with the **ISaGRAF Kernel**.

This manual contains detailed information about the contents of the files generated by the **C code generator**. In particular, all the tables and data structures generated are described as are the set of C code instructions produced.

This document assumes user knowledge of certain concepts. The knowledge prerequisites are :

- **ISaGRAF** architecture
- **TIC code**.

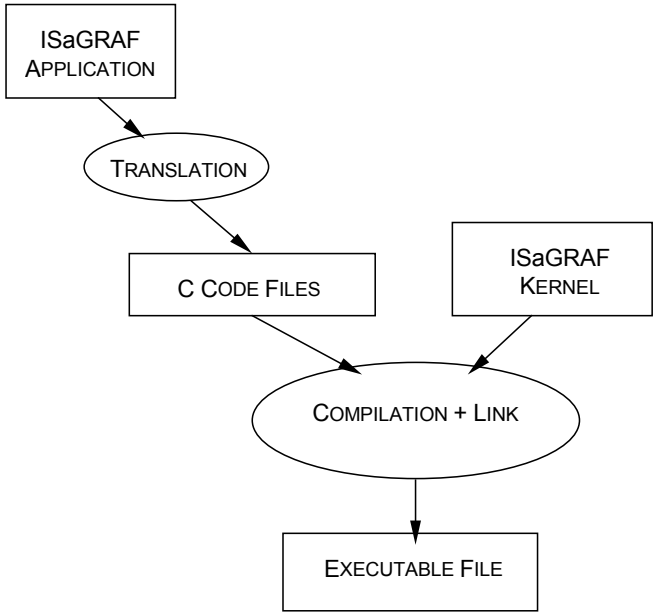
For further information on these concepts you can consult the official documents on **ISaGRAF**.

G. 2. General principle of C code generation

G. 2. 1. Principle

The aim of the C code generation is to produce one executable file including the application written with **ISaGRAF Workbench** and **ISaGRAF Kernel.** This executable file is able to run on the chosen target.

To achieve it, the application ; programs and I/O connections etc. are translated into C code files**.** These files are compiled and linked with **ISaGRAF Kernel** across a C compiler compatible with the target, to obtain an executable file.



The compilation technique (as opposed to the interpretation technique) suppresses a few functions:

- on-line modification

  It's impossible to modify an application during its execution.

- breakpoints

  It's impossible to put breakpoints into processing code (IL, ST, FBD, LD).

- resources management

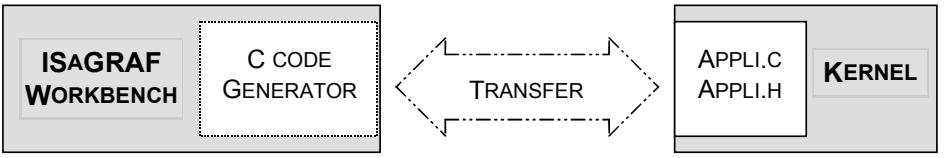
  In this version of **C code generator**, the resource management is not supported, because it does not represent a great interest.

- automatic downloads

  The generated code must be manually downloaded.

G. 3. Architecture

Translating an **ISaGRAF** application creates two files named : **appli.h** and **appli.c**. **appli.h** is an include file which contains external declarations only used by **ISaGRAF Kernel**. **Appli.c** is a source file which contains the data and the code of application.



This chapter describes the data structures and C instructions generated.

G. 3. 1. Compliance

For obvious reasons normalized C code (according to **C-Ansi**) is generated.

G. 3. 2. Data representation

Data represents the application parameters for the **Kernel** : number of variables for each type, number of steps or transitions in SFC charts, etc.

Data is represented by structures and organized in tables. The size of tables depends on the application. Therefore, the size of data structures is wholly defined.

All the tables are defined with a global scope but they are initialized by function. The access to these tables is ' read only ' to the rest of the **Kernel**.

G. 3. 3. Code representation

Application code represents the translation of the application instructions described with the **IEC 1131-3** language (**IL, ST, FBD, LD**) into C code.

A subset of the C language is generated by the **C code generator**. This chapter describes all the possibilities of these generated C code instructions.

G. 3. 3. 1. Set of the basic C instructions

This chapter syntactically describes the set of basic C instructions used by the **C code generator**.

G. 3. 3. 1. 1. *Variables*

The variables employed in the instructions can represent a *single-element* or *multi-elements*.
A *single-element* is a value or an address. A *multi-element* is an array.

G. 3. 3. 1. 2. *Operation instructions*

- **Assignment instruction**

<variable> **=** <variable> **;**

- **Unary operator**

<variable> **= -** <variable> **;**
<variable> **= !**<variable> **;**

- **Binary analog operator**

<variable> **=** <variable> **+** <variable> **;**
<variable> **=** <variable> ***** <variable> **;**
<variable> **=** <variable> **-** <variable> **;**
<variable> **=** <variable> **/** <variable> **;**

- **Binary Boolean operator**

The result of a Boolean expression is stored in a Boolean variable. **For ISaGRAF Kernel**, a Boolean is represented by an unsigned char (**uchar**). Also, all the Boolean expressions are converted to **uchar** type.

<*variable*> **= (uchar)(**<*variable*> **>** <*variable*>**);**
<*variable*> **= (uchar)(**<*variable*> **<=** <*variable*>**);**
<*variable*> **= (uchar)(**<*variable*> **>=** <*variable*>**);**
<*variable*> **= (uchar)(**<*variable*> **==** <*variable*>**);**
<*variable*> **= (uchar)(**<*variable*> **!=** <*variable*>**);**
<*variable*> **= (uchar)(**<*variable*> **|** <*variable*>**);**
<*variable*> **= (uchar)(**<*variable*> **&** <*variable*>**);**
<*variable*> **= (uchar)(**<*variable*> **^** <*variable*>**);**

G. 3. 3. 1. 3. *Control instructions*

- **if**

if (<*expression*>**) goto** <*label*>
if (!(<*expression*>**)) goto** <*label*>
if (<*expression*>**) { <***instructions***> } else { <***instructions***> }**

- **goto**

goto <*label*>

- **label**

<*identifier*> **:**

- **return**

return ((unsigned char) (<*expression*> **))**

- **switch**

switch (<*ana_expression***>) { { case <***value***> : <***instruction***> ; break; } default :** <*instruction*> **break; }**

G. 3. 4. Function call of Kernel

The following functions are defined in the **kernel**. They can be called in **appli.c**.

Management	Function prototype
TIC code	uchar tic_dec(uint16 *)
	extern uint16 *blk_seq(uint16);
	void tic_cnv(uint16,uint16,uint16,uint16)
I/O access	void ios_ana_ref(uint16)
	void ios_boo_ref(uint16)
	void ios_msg_ref(uint16)
SFC evolution	uchar evo_sfc_start(uchar)
	uchar evo_sfc_rstart(uchar)
	uchar evo_sfc_kill(uchar)
	uchar evo_sfc_freeze(uchar)
	uchar ste_testact(uint16)
	uint32 ste_readtim(uint16)
Timer	uchar tmr_active(uint16)
	uchar tmr_deactive(uint16)

	uint32 tmr_read(uint16)
	void tmr_write(uint16, uint32);
	void tmr_duplicate(uint16, uint16);
Message	uchar msg_len (uint16)
	void msg_concat (uint16, uint16, uint16)
	uchar msg_equ (uint16, uint16)
	uchar msg_lexcmp (uint16, uint16)
	void msg_set(uint16, uint16)
	void msg_write(uint16, uchar, uchar *)
	uchar sys_strlen(char*)
Function and Function Block	int32 usf_mic(uint16, uint16 *, uchar *)
	int32 usf_std(uint16, uint16 *, uchar *)
	void fbl_active(uint16, uint16, uint16 *, uchar *)
	void fbl_std(uint16, uint16, uint16 *)
	int32 fbl_readpa(uint16, uint16, uint16)
Program	extern uchar prg_get_status(uchar)
System	int32 sys_call(int32, int32)
	void sys_err(uint16,uint16)
I/O special process	int32 ios_boo_operate(uint16, int32, int32)
	int32 ios_ana_operate(uint16, int32, int32)
	int32 ios_msg_operate(uint16, int32, int32)

G. 3. 5. TIC code representation

The translation method is to translate each **TIC code** into a list of C instructions. The following table details the correspondence between **TIC code** and C code.

Notation :

- *op1*, *op2*, *op3* and *op4* are index values resolved by **ISaGRAF linker**.
- *label* is an identifier generated by the **C code generator**.
- *Index* is a value generated by the **C code generator**.

TIC code	Corresponding C code
TIC_OUT_A1SI	`ios_ana_ref(op1);` `BF_ANA[op1] = -BF_ANA[op2];`
TIC_A1SI	`BF_ANA[op1] = -BF_ANA[op2];`

TIC_OUT_A1NO.	ios_boo_ref(*op1*); BF_BOO[*op1*] = !BF_BOO[*op2*];
TIC_A1NO.	BF_BOO[*op1*] = !BF_BOO[*op2*];
TIC_OUT_A2MI	ios_ana_ref(*op1*); BF_ANA[*op1*] = BF_ANA[*op2*] * BF_ANA[*op3*];
TIC_A2MI	BF_ANA[*op1*] = BF_ANA[*op2*] * BF_ANA[*op3*];
TIC_OUT_A2LTI	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] < BF_ANA[*op3*]);
TIC_A2LTI	BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] < BF_ANA[*op3*]);

TIC_OUT_A2DI	ios_ana_ref(*op1*);
	if (BF_ANA[*op3*]) {
	BF_ANA[*op1*] = BF_ANA[*op2*] / BF_ANA[*op3*] ;
	}
	else
	{
	sys_err(64L,*op3* | 0x2000L);
	if (BF_ANA[*op2*] > 0) {
	BF_ANA[*op1*] = MAX_INT;
	}
	else
	{
	BF_ANA[*op1*] = - MAX_INT;
	}
	;
	}
	;

TIC_A2DI	if (BF_ANA[*op3*]) {
	BF_ANA[*op1*] = BF_ANA[*op2*] / BF_ANA[*op3*] ;
	}
	else
	{
	sys_err(64L,*op3* | 0x2000L);
	if (BF_ANA[*op2*] > 0) {
	BF_ANA[*op1*] = MAX_INT;
	}
	else
	{
	BF_ANA[*op1*] = - MAX_INT;
	}
	;
	}
	;

TIC_OUT_A2AI	ios_ana_ref(*op1*); BF_ANA[*op1*] = BF_ANA[*op2*] + BF_ANA[*op3*];
TIC_A2AI	BF_ANA[*op1*] = BF_ANA[*op2*] + BF_ANA[*op3*];
TIC_OUT_A2SI	ios_ana_ref(*op1*); BF_ANA[*op1*] = BF_ANA[*op2*] - BF_ANA[*op3*];
TIC_A2SI	BF_ANA[*op1*] = BF_ANA[*op2*] - BF_ANA[*op3*];

TIC_OUT_A1SR	ios_ana_ref(*op1*); ((floating point*)BF_ANA)[*op1*] = -((floating point*)BF_ANA)[*op2*];
TIC_A1SR	((floating point*)BF_ANA)[*op1*] = -((floating point*)BF_ANA)[*op2*];
TIC_OUT_A2MR	ios_ana_ref(*op1*); ((floating point*)BF_ANA)[*op1*] = ((floating point*)BF_ANA)[*op2*] * ((floating point*)BF_ANA)[*op3*];
TIC_A2MR	((floating point*)BF_ANA)[*op1*] = ((floating point*)BF_ANA)[*op2*] * ((floating point*)BF_ANA)[*op3*];

TIC_OUT_A2DR	ios_ana_ref(*op1*);
	if (((floating point*)BF_ANA)[*op3*]) {
	((floating point*)BF_ANA)[*op1*] = ((floating point*)BF_ANA)[*op2*] / ((floating point*)BF_ANA)[*op3*] ;
	}
	else
	{
	sys_err(64L,*op3* | 0x2000L);
	if (((floating point*)BF_ANA)[*op2*] > 0) {
	((floating point*)BF_ANA)[*op1*] = MAX_REA;
	}
	else
	{
	((floating point*)BF_ANA)[*op1*] = - MAX_REA;
	}
	;
	}
	;

TIC_A2DR	if (((floating point*)BF_ANA)[*op3*]) {
	((floating point*)BF_ANA)[*op1*] = ((floating point*)BF_ANA)[*op2*] / ((floating point*)BF_ANA)[*op3*] ;
	}
	else
	{
	sys_err(64L,*op3* | 0x2000L);
	if (((floating point*)BF_ANA)[*op2*] > 0) {
	((floating point*)BF_ANA)[*op1*] = MAX_REA;
	}
	else
	{
	((floating point*)BF_ANA)[*op1*] = - MAX_REA;
	}
	;
	}
	;

TIC_OUT_A2AR	ios_ana_ref(*op1*); ((floating point*)BF_ANA)[*op1*] = ((floating point*)BF_ANA)[*op2*] + ((floating point*)BF_ANA)[*op3*];
TIC_A2AR	((floating point*)BF_ANA)[*op1*] = ((floating point*)BF_ANA)[*op2*] + ((floating point*)BF_ANA)[*op3*];
TIC_OUT_A2SR	ios_ana_ref(*op1*); ((floating point*)BF_ANA)[*op1*] = ((floating point*)BF_ANA)[*op2*] - ((floating point*)BF_ANA)[*op3*];
TIC_A2SR	((floating point*)BF_ANA)[*op1*] = ((floating point*)BF_ANA)[*op2*] - ((floating point*)BF_ANA)[*op3*];
TIC_OUT_A2LTR	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] < ((floating point*)BF_ANA)[*op3*]);
TIC_A2LTR	BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] < ((floating point*)BF_ANA)[*op3*]);
TIC_OUT_A2GTR	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] > ((floating point*)BF_ANA)[*op3*]);
TIC_A2GTR	BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] > ((floating point*)BF_ANA)[*op3*]);
TIC_OUT_A2LER	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] <= ((floating point*)BF_ANA)[*op3*]);
TIC_A2LER	BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] <= ((floating point*)BF_ANA)[*op3*]);
TIC_OUT_A2GER	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] >= ((floating point*)BF_ANA)[*op3*]);
TIC_A2GER	BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] >= ((floating point*)BF_ANA)[*op3*]);
TIC_OUT_A2EQR	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] == ((floating point*)BF_ANA)[*op3*]);
TIC_A2EQR	BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] == ((floating point*)BF_ANA)[*op3*]);

TIC_OUT_A2NER	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] != ((floating point*)BF_ANA)[*op3*]);
TIC_A2NER	BF_BOO[*op1*] = (unsigned char) (((floating point*)BF_ANA)[*op2*] != ((floating point*)BF_ANA)[*op3*]);
TIC_OUT_A3R	ios_ana_ref(*op1*); ((floating point*)BF_ANA)[*op1*] = ((floating point*)BF_ANA)[*op2*];
TIC_A3R	((floating point*)BF_ANA)[*op1*] = ((floating point*)BF_ANA)[*op2*];
TIC_A4R	tic_dec(blk_seq(*op2*)); ((floating point*)BF_ANA)[*op1*] = LGC_RET_SUB;
TIC_SYS_GPARR	BF_ANA[*op1*] = BF_ANA[LGC_P_NUM[*op2* - 1L]];
TIC_SYS_PPARR	LGC_RET_SUB = BF_ANA[*op2*];
TIC_TRS_A2LTR	return ((unsigned char)((((floating point*)BF_ANA)[*op2*] < ((floating point*)BF_ANA)[*op3*]))));
TIC_TRS_A2GTR	return ((unsigned char)((((floating point*)BF_ANA)[*op2*] > ((floating point*)BF_ANA)[*op3*]))));
TIC_TRS_A2LER	return ((unsigned char)((((floating point*)BF_ANA)[*op2*] <= ((floating point*)BF_ANA)[*op3*]))));
TIC_TRS_A2GER	return ((unsigned char)((((floating point*)BF_ANA)[*op2*] >= ((floating point*)BF_ANA)[*op3*]))));
TIC_TRS_A2EQR	return ((unsigned char)((((floating point*)BF_ANA)[*op2*] == ((floating point*)BF_ANA)[*op3*]))));
TIC_TRS_A2NER	return ((unsigned char)((((floating point*)BF_ANA)[*op2*] != ((floating point*)BF_ANA)[*op3*]))));
TIC_A2AT	tmr_write(*op1*, tmr_read(*op2*) + tmr_read(*op3*));
TIC_OUT_A2LTT	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) < tmr_read(*op3*));
TIC_A2LTT	BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) < tmr_read(*op3*));
TIC_OUT_A2GTT	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) > tmr_read(*op3*));
TIC_A2GTT	BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) > tmr_read(*op3*));
TIC_OUT_A2LET	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) <= tmr_read(*op3*));

TIC_A2LET	BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) <= tmr_read(*op3*));
TIC_A2ST	tmr_write(*op1*, tmr_read(*op2*) - tmr_read(*op3*));
TIC_OUT_A2GET	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) >= tmr_read(*op3*));
TIC_A2GET	BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) >= tmr_read(*op3*));
TIC_OUT_A2EQT	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) == tmr_read(*op3*));
TIC_A2EQT	BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) == tmr_read(*op3*));
TIC_OUT_A2NET	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) != tmr_read(*op3*));
TIC_A2NET	BF_BOO[*op1*] = (unsigned char) (tmr_read(*op2*) != tmr_read(*op3*));
TIC_A3T	tmr_write(*op1*, tmr_read(*op2*));
TIC_A4T	tic_dec(blk_seq(*op2*)); tmr_write(*op1*, LGC_RET_SUB);
TIC_SYS_GPART	tmr_write(*op1*, tmr_read(LGC_P_NUM[*op2* - 1L]));
TIC_SYS_PPART	LGC_RET_SUB = tmr_read(*op2*);
TIC_SYS_A3T	tmr_duplicate(*op1*,*op2*) ;
TIC_SYS_TSTART	tmr_active(LGC_P_NUM[0L]);
TIC_SYS_TSTOP	tmr_deactive(LGC_P_NUM[0L]);
TIC_TRS_A2LTT	return ((unsigned char)((tmr_read(*op2*) < tmr_read(*op3*))));
TIC_TRS_A2GTT	return ((unsigned char)((tmr_read(*op2*) > tmr_read(*op3*))));
TIC_TRS_A2LET	return ((unsigned char)((tmr_read(*op2*) <= tmr_read(*op3*))));
TIC_TRS_A2GET	return ((unsigned char)((tmr_read(*op2*) >= tmr_read(*op3*))));
TIC_TRS_A2EQT	return ((unsigned char)((tmr_read(*op2*) == tmr_read(*op3*))));
TIC_TRS_A2NET	return ((unsigned char)((tmr_read(*op2*) != tmr_read(*op3*))));
TIC_OUT_A2AM	ios_msg_ref (*op1*); msg_concat (*op1*, *op2*,*op3*);
TIC_A2AM	msg_concat (*op1*, *op2*,*op3*);
TIC_OUT_A2LTM	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (msg_lexcmp(*op2*,*op3*,' == 2'));
TIC_A2LTM	BF_BOO[*op1*] = (unsigned char) (msg_lexcmp(*op2*,*op3*,' == 2'));

TIC_OUT_A2GTM	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (msg_lexcmp(*op2*,*op3*,' == 4'));
TIC_A2GTM	BF_BOO[*op1*] = (unsigned char) (msg_lexcmp(*op2*,*op3*,' == 4'));
TIC_OUT_A2LEM	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (msg_lexcmp(*op2*,*op3*,' != 4'));
TIC_A2LEM	BF_BOO[*op1*] = (unsigned char) (msg_lexcmp(*op2*,*op3*,' != 4'));
TIC_OUT_A2GEM	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (msg_lexcmp(*op2*,*op3*,' != 2'));
TIC_A2GEM	BF_BOO[*op1*] = (unsigned char) (msg_lexcmp(*op2*,*op3*,' != 2'));
TIC_OUT_A2EQM	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (msg_equ(*op2*,*op3*));
TIC_A2EQM	BF_BOO[*op1*] = (unsigned char) (msg_equ(*op2*,*op3*));
TIC_OUT_A2NEM	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (!msg_equ(*op2*,*op3*));
TIC_A2NEM	BF_BOO[*op1*] = (unsigned char) (!msg_equ(*op2*,*op3*));
TIC_OUT_A3M	ios_msg_ref (*op1*); msg_set(*op1*,*op2*);
TIC_A3M	msg_set(*op1*,*op2*);
TIC_SYS_GPARM	msg_set(*op1*, LGC_P_NUM[*op2* -1]);
TIC_TRS_A2LTM	return ((unsigned char)((unsigned char) (msg_lexcmp(*op2*,*op3*) == 2)));
TIC_TRS_A2GTM	return ((unsigned char)((unsigned char) (msg_lexcmp(*op2*,*op3*) == 4)));
TIC_TRS_A2LEM	return ((unsigned char)((unsigned char) (msg_lexcmp(*op2*,*op3*) != 4)));
TIC_TRS_A2GEM	return ((unsigned char)((unsigned char) (msg_lexcmp(*op2*,*op3*) != 2)));
TIC_TRS_A2EQM	return ((unsigned char)((unsigned char) (msg_equ(*op2*,*op3*))));
TIC_TRS_A2NEM	return ((unsigned char)((unsigned char) (!msg_equ(*op2*,*op3*))));
TIC_OUT_A2GTI	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] > BF_ANA[*op3*]);
TIC_A2GTI	BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] > BF_ANA[*op3*]);

TIC_OUT_A2LEI	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] <= BF_ANA[*op3*]);
TIC_A2LEI	BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] <= BF_ANA[*op3*]);
TIC_OUT_A2GEI	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] >= BF_ANA[*op3*]);
TIC_A2GEI	BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] >= BF_ANA[*op3*]);
TIC_OUT_A2EQI	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] == BF_ANA[*op3*]);
TIC_A2EQI	BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] == BF_ANA[*op3*]);
TIC_OUT_A2EQB	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_BOO[*op2*] == BF_BOO[*op3*]);
TIC_A2EQB	BF_BOO[*op1*] = (unsigned char) (BF_BOO[*op2*] == BF_BOO[*op3*]);
TIC_OUT_A2NEI	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] != BF_ANA[*op3*]);
TIC_A2NEI	BF_BOO[*op1*] = (unsigned char) (BF_ANA[*op2*] != BF_ANA[*op3*]);
TIC_OUT_A2NEB	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_BOO[*op2*] != BF_BOO[*op3*]);
TIC_A2NEB	BF_BOO[*op1*] = (unsigned char) (BF_BOO[*op2*] != BF_BOO[*op3*]);
TIC_OUT_A2ORB	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_BOO[*op2*] | BF_BOO[*op3*]);
TIC_A2ORB	BF_BOO[*op1*] = (unsigned char) (BF_BOO[*op2*] | BF_BOO[*op3*]);
TIC_OUT_A2ANO.	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_BOO[*op2*] & BF_BOO[*op3*]);
TIC_A2ANO.	BF_BOO[*op1*] = (unsigned char) (BF_BOO[*op2*] & BF_BOO[*op3*]);
TIC_OUT_A2XOB	ios_boo_ref(*op1*); BF_BOO[*op1*] = (unsigned char) (BF_BOO[*op2*] ^ BF_BOO[*op3*]);
TIC_A2XOB	BF_BOO[*op1*] = (unsigned char) (BF_BOO[*op2*] ^ BF_BOO[*op3*]);
TIC_OUT_A3B	ios_boo_ref(*op1*); BF_BOO[*op1*] = BF_BOO[*op2*];
TIC_A3B	BF_BOO[*op1*] = BF_BOO[*op2*];

TIC_OUT_A3I	ios_ana_ref(*op1*); BF_ANA[*op1*] = BF_ANA[*op2*];
TIC_A3I	BF_ANA[*op1*] = BF_ANA[*op2*];
TIC_USF_A4B	LGC_P_TYP [*index*] = 0; BF_BOO[*op1*] = (unsigned char) (usf_mic(*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0]));
TIC_STD_A4B	LGC_P_TYP [*index*] = 0; BF_BOO [*op1*] = (unsigned char) (usf_std (*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0]));
TIC_USF_A4I	LGC_P_TYP_ [*index*] = 0; BF_ANA[*op1*] = usf_mic(*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0]);
TIC_STD_A4I	LGC_P_TYP [*index*] = 0; BF_ANA[*op1*] = usf_std(*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0]);
TIC_USF_A4R	LGC_P_TYP[*index*] = 0; BF_ANA[*op1*] = usf_mic(*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0]);
TIC_STD_A4R	LGC_P_TYP[*index*] = 0 ; BF_ANA[*op1*] = usf_std(*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0]);
TIC_USF_A4T	LGC_P_TYP[*index*] = 0; tmr_write((*op1*, usf_mic(*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0]));
TIC_USF_A4M	LGC_P_TYP[*index*] = 0; msg_write(*op1*,sys_strlen(usf_mic(*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0])), usf_mic(*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0]));
TIC_STD_A4M	LGC_P_TYP[*index*] = 0; msg_write(*op1*,sys_strlen(usf_std(*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0])), usf_std(*op2*,&LGC_P_NUM[0],&LGC_P_TYP[0]));
TIC_TRS_A4B	tic_dec(blk_seq(*op2*)) ; return ((unsigned char)(((unsigned char) (LGC_RET_SUB))));

TIC_OUT_A4B	ios_boo_ref(*op1*);
	tic_dec(blk_seq(*op2*));
	BF_BOO[*op1*] = (unsigned char) (LGC_RET_SUB);

TIC_A4B	tic_dec(blk_seq(*op2*)); BF_BOO[*op1*] = (unsigned char) (LGC_RET_SUB);
TIC_OUT_A4I	ios_ana_ref(*op1*);
TIC_A4I	tic_dec(blk_seq(*op2*)); BF_ANA[*op1*] = LGC_RET_SUB;

TIC_OUT_A4M	ios_msg_ref (*op1*);
	tic_dec(blk_seq(*op2*));
	msg_set(*op1*,(unsigned char) (LGC_RET_SUB));

TIC_A4M	tic_dec(blk_seq(*op2*)); msg_set(*op1*,(unsigned char) (LGC_RET_SUB));

TIC_FBL	LGC_P_TYP [*index*] = 0; fbl_active(*op1*, *op2*, &LGC_P_NUM[0], &LGC_P_TYP[0]);
TIC_STD_FBL	fbl_std(*op1*, *op2*, &LGC_P_NUM[0]);
TIC_SYS_GPARBLKB	BF_BOO[*op1*] = (unsigned char) (fbl_readpa(*op3*, *op4*, *op2*));
TIC_SYS_GPARBLKI	BF_ANA[*op1*] = fbl_readpa(*op3*, *op4*, *op2*);
TIC_SYS_GPARBLKR	BF_ANA[*op1*] = fbl_readpa(*op3*, *op4*, *op2*);
TIC_SYS_GPARBLKT	tmr_write((*op1*, fbl_readpa(*op3*, *op4*, *op2*));
TIC_SYS_GPARBLKM	msg_write(*op1*,sys_strlen(fbl_readpa(*op3*,*op4*,*op2*)),fbl_readpa(*op3*,*op4*,*op2*));
TIC_C1B	if (BF_BOO[*op1*]) goto *label*;
TIC_C2B	if (!(BF_BOO[*op1*])) goto *label*
TIC_I1	goto *label*
TIC_I2	return ((unsigned char)(0));
TIC_P1S	LGC_P_NUM[*index*] = *op1*;
TIC_P1C	LGC_P_NUM[*index*] = *op1*;
TIC_P1F	LGC_P_NUM[*index*] = *op1*;
TIC_P1B	LGC_P_TYP[*index*] = BOO; LGC_P_NUM[*index*] = *op1*;
TIC_P1I	LGC_P_TYP[*index*] = ANA; LGC_P_NUM[*index*] = *op1*;
TIC_P1R	LGC_P_TYP[*index*] = REA; LGC_P_NUM[*index*] = *op1*;
TIC_P1T	LGC_P_TYP[*index*] = TMR; LGC_P_NUM[*index*] = *op1*;
TIC_P1M	LGC_P_TYP [*index*] = MSG; LGC_P_NUM [*index*] = *op1*;
TIC_L1	*label* :
TIC_SYS_GPARB	BF_BOO[*op1*] = BF_BOO[LGC_P_NUM[*op2* - 1L]];
TIC_SYS_GPARI	BF_ANA[*op1*] = BF_ANA[LGC_P_NUM[*op2* - 1L]];
TIC_SYS_PPARB	LGC_RET_SUB = BF_BOO[*op2*];
TIC_SYS_PPARI	LGC_RET_SUB = BF_ANA[*op2*];
TIC_SYS_PPARM	LGC_RET_SUB = *op2*;
TIC_TRS_PPARB	return ((unsigned char)(BF_BOO[*op2*]));
TIC_SYS_GSTART	evo_sfc_start((unsigned char) LGC_P_NUM[0L]);
TIC_SYS_GFREEZE	evo_sfc_freeze((unsigned char) LGC_P_NUM[0L]);
TIC_SYS_GKILL	evo_sfc_kill((unsigned char) LGC_P_NUM[0L]);
TIC_SYS_GRST	evo_sfc_rstart((unsigned char) LGC_P_NUM[0L]);
TIC_OUT_GSTATUS	ios_ana_ref(*op1*); BF_ANA[*op1*] = prg_get_status((unsigned char)LGC_P_NUM[0]);

TIC_SYS_GSTATUS	BF_ANA[*op1*] = prg_get_status((unsigned char)LGC_P_NUM[0]);

TIC_SYS_OPERATE	LGC_P_TYP[*index*] = 0;
	switch (LGC_P_TYP[0L])
	{
	case BOO :
	BF_ANA[*op1*] = ios_boo_operate(LGC_P_NUM[0], BF_ANA[LGC_P_NUM[2]], BF_ANA[LGC_P_NUM[1]]);
	break;
	case ANA :
	BF_ANA[*op1*] = ios_ana_operate(LGC_P_NUM[0], BF_ANA[LGC_P_NUM[2]], BF_ANA[LGC_P_NUM[1]]) ;
	break;
	case REA :
	BF_ANA[*op1*] = ios_ana_operate(LGC_P_NUM[0], BF_ANA[LGC_P_NUM[2]], BF_ANA[LGC_P_NUM[1]]) ;
	break;
	case MSG :
	BF_ANA[*op1*] = ios_msg_operate(LGC_P_NUM[0], BF_ANA[LGC_P_NUM[2]], BF_ANA[LGC_P_NUM[1]]) ;
	break;
	default :
	sys_err(69L,1L);
	break;
	}

TIC_SYS_FEDGE	LGC_BVAL = BF_BOO(LGC_P_NUM[0L]); BF_BOO[*op1*] = (unsigned char) (!BF_BOO(LGC_P_NUM[0L]) & BF_BOO(LGC_P_NUM[1L])); BF_BOO(LGC_P_NUM[1L] = LGC_BVAL;
TIC_SYS_REDGE	LGC_BVAL = BF_BOO(LGC_P_NUM[0L]); BF_BOO[*op1*] = (unsigned char) (BF_BOO(LGC_P_NUM[0L]) & !BF_BOO(LGC_P_NUM[1L])); BF_BOO(LGC_P_NUM[1L] = LGC_BVAL;

TIC_OUT_CNV	switch(*op1*)
	{
	case:TIC_BOO: ios_boo_ref(*op1*); break;
	case:TIC_ANA: ios_ana_ref(*op1*); break;
	case:TIC_MSG: ios_msg_ref (*op1*); break;
	default: sys_err(14L,7L); break;
	}
	tic_cnv(LGC_P_NUM[0],*op1*,*op2*,*op3*);

TIC_SYS_CNV	tic_cnv(LGC_P_NUM[0],*op1*,*op2*,*op3*);
TIC_SYS_TAC	BF_BOO[*op1*] = (unsigned char) (ste_testact(*op2*));
TIC_SYS_DAC	tmr_write(*op1*, ste_readtim(*op2*));
TIC_SYS_SYSTEM	BF_ANA[*op1*] = sys_call(BF_ANA[LGC_P_NUM[0]],BF_ANA[LGC_P_NUM[1]]);

TIC_TRS_A1NO.	`return ((unsigned char)( !BF_BOO[op2]));`
TIC_TRS_A2LTI	`return ((unsigned char)( (BF_ANA[op2] < BF_ANA[op3])));`
TIC_TRS_A2GTI	`return ((unsigned char)( (BF_ANA[op2] > BF_ANA[op3])));`
TIC_TRS_A2LEI	`return ((unsigned char)( (BF_ANA[op2] <= BF_ANA[op3])));`
TIC_TRS_A2GEI	`return ((unsigned char)( (BF_ANA[op2] >= BF_ANA[op3])));`
TIC_TRS_A2EQI	`return ((unsigned char)( (BF_ANA[op2] == BF_ANA[op3])));`
TIC_TRS_A2NEI	`return ((unsigned char)( (BF_ANA[op2] != BF_ANA[op3])));`
TIC_TRS_A2EQB	`return ((unsigned char)( (BF_BOO[op2] == BF_BOO[op3])));`
TIC_TRS_A2NEB	`return ((unsigned char)( (BF_BOO[op2] != BF_BOO[op3])));`
TIC_TRS_A2ORB	`return ((unsigned char)( (BF_BOO[op2] | BF_BOO[op3])));`
TIC_TRS_A2ANO.	`return ((unsigned char)( (BF_BOO[op2] & BF_BOO[op3])));`
TIC_TRS_A2XOB	`return ((unsigned char)( (BF_BOO[op2] ^ BF_BOO[op3])));`
TIC_TRS_A3B	`return ((unsigned char)( BF_BOO[op2]));`
TIC_END_BLOCK	`return ((unsigned char)( 0 ));`

G. 4. Description of generated files

This chapter describes wholly the generated files.

G. 4. 1. Appli.h structure

Appli.h contains all the declarations in external mode of data and functions defined in **appli.c** and used by the **Kernel**. Also, it is not necessary to include **appli.h** in **appli.c**.

G. 4. 2. Appli.c structure

Appli.c is structured in two parts : a declaration zone and a definition zone. The first part contains all the declarations : data structures, tables, functions etc. The second part consists of functions which contain the application code and the initialization of the data tables.

Declaration of data and functions	Include
	Data structures and tables
	External functions
	External data
	Static data
	Declaration of the initialization functions
	Declaration of the processing code functions
Definition of functions	Initialization code
	Application code

G. 4. 2. 1. Include

This section contains all the target Includes useful for the file compilation. These Includes are the same for all C code files generated.

Includes need in appli.c
#include <tasy0def.h>
#include <take0pro.h>
#include <take0blk.h>
#include <take0ste.h>
#include <take0msg.h>
#include <take0prg.h>
#include <tasy0apl.h>
#include <taio0def.h>

G. 4. 2. 2. Data structures and tables

This section declares the data structures and the tables which contain them.
The number and the size of these tables depend on the application.

G. 4. 2. 2. 1. *Specification of the description*

The following table specifies the notation used to describe the tables and data structures.

Data type	Size in bytess	Corresponding ' C ' type	Comments
uint16	2	unsigned short	
int32	4	long	
uint32	4	unsigned long	can contain an address
uchar	1	unsigned char	
charx	x	char t[x]	x depends on the application

G. 4. 2. 2. 2. *Data structure and table descriptions*

G. 4. 2. 2. 2. 1. *List of Tables*

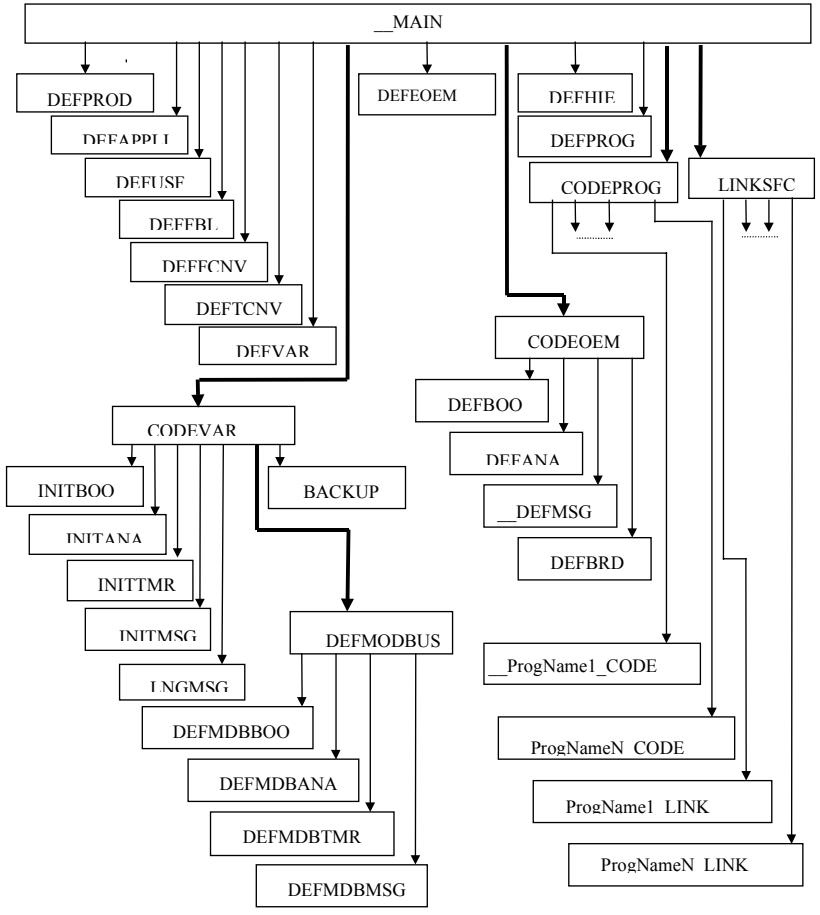
The following table contains the list of tables generated in **appli.c**. For each table, the size (in bytes) is calculated. Sometimes, the table size is not fixed, it is application dependant. In these cases, the table size is represented by a formula containing a variable 'x'.

Name Table	Size (bytes)	Comments
__MAIN	64	
__DEFPROD	24	
__DEFAPPLI	32	
__DEFUSF	4+12x	x = number of functions
__DEFFBL	6+16x	x = number of blocks
__DEFFCNV	4+16x	x = number of conversion functions
__DEFTCNV	4+8x	x = number of tables
__DEFVAR	32	
__CODEVAR	36	
__INITBOO	2x	x = number of Boolean variables to be set
__INITANA	6x	x = number of analogs to be initialized
__INITTMR	6x	x = number of timers to be initialized
__INITMSG	7x	x = number of messages to be initialized
__LNGMSG	x	x = number of messages
__DEFMODBUS	16	the information are not structured
__DEFMDBBOO	2x	x = number of Boolean variables Modbus
__DEFMDBANA	2x	x = number of analogs Modbus
__DEFMDBTMR	2x	x = number of timers Modbus
__DEFMDBMSG	2x	x = number of messages Modbus
__BACKUP	80	
__DEFOEM	6	
__CODEOEM	16	
__DEFBOO	8x	x = number of I/O Boolean variables
__DEFANA	8x	x = number of I/O analogs
__DEFMSG	8x	x = number of I/O messages
__DEFBRD	4x	x = number of board
__DEFBRD_n	14+ 2r ı (if type = uint16) 16+ 4r ı (if type = uint32) 28+ 16 m (if type=16 uchar)	m = number of specific fields
__DEFHIE	12+2x	x=begin+end+main+child+function

__DEFPROG	2x	x = number of programs * 2
__CODEPROG	4x	x = number of programs
__PrgName_CODE	4x	x = number of block programs
__LINKSFC	4x	x = number of programs
__PrgName_LINK	2x	x = depend of SFC structures
__RESSOURCE	x	

G. 4. 2. 2. 2. 2. Table hierarchies

To improve the understanding of the table organization, the following figure presents the hierarchy of the tables.



G. 4. 2. 2. 2. 3. *Detailed description*

This chapter describes all the data structures and tables manipulated in **appli.c**. and includes:

- the structure of their elements
- the table declaration
- the table size in bytes

G. 4. 2. 2. 2. 3. 1 *Table name : __MAIN*

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   uint32 def_flag;
   uint32 def_prod;
   uint32 def_appli;
   uint32 def_usf;
   uint32 def_fbl;
   uint32 def_fcnv;
   uint32 def_tcnv;
   uint32 def_var;
   uint32 tab_init;
   uint32 def_ios;
   uint32 def_oem;
   uint32 def_hie;
   uint32 def_prog;
   uint32 tab_code;
   uint32 tab_link;
   uint32 def_res;
}
str_header;
```

- **Table declaration**

str_header __MAIN;

- **Table Size**

16*4 = 64

G. 4. 2. 2. 2. 3. 2 *Table name : __DEFPROD*

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   char name[16];
   char version[8];
}
str_def_prod;
```

- **Table declaration**

str_def_prod __DEFPROD;

Structure defined in ' tasy0apl.h '

- **Table Size**

24

G. 4. 2. 2. 2. 3. 3 *Table name : __DEFAPPLI*

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   char name[16];
   uint32 date;
   uint32 crc;
   uint16 version;
   uint16 error_level;
   uint16 start_mode;
   uint16 cycle_duration;
}
str_def_appli;
```

- **Table declaration**

str_def_appli __DEFAPPLI;

- **Table Size**

32

G. 4. 2. 2. 2. 3. 4 ***Table name : __DEFUSF***

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   uint16 libno
   char name[10];
}
str_dfusf;
```

Structure defined in ' appli.c ' and ' appli.h '

```
typedef struct
{
   uint16 usfNo.;
   uint16 reserved;
   str_dfusf usf[x];
}
str_dfusf_head;
```

- **Table declaration**

str_dfusf_head __DEFUSF;

- **Table Size**

(2+2) + (2+10)*x = 4+12x

G. 4. 2. 2. 2. 3. 5 Table name : __DEFFBL

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   uint16 libno;
   uint16 No.inst;
   char name[12];
}
str_def_fbl;
```

Structure defined in ' appli.c ' and ' appli.h '

```
typedef struct
{
   uint16 fblNo.;
   uint16 reserved;
```

```
   str_def_fbl fbl[x];
   uint16 align16;
}
str_def_fbl_head;
```

- **Table declaration**

str_def_fbl_head __DEFFBL;

- **Table Size**

(2+2) + (2+2+12)*x + 2 = 6+16x

G. 4. 2. 2. 2. 3. 6 *Table name : __DEFFCNV*

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   uint16 libno;
   uint16 reserve;
   char name[10];
   uint16 reserve2;
}
str_dfcnv;
```

Structure defined in ' appli.c ' and ' appli.h '

```
typedef struct
{
   uint16 No.;
   uint16 reserve;
   str_dfcnv fcnv[x];
}
str_def_fcnv_head;
```

- **Table declaration**

str_def_fcnv_head __DEFFCNV;

- **Table Size**

(2+2)+(2+2+10+2)*x = 4+16x

G. 4. 2. 2. 2. 3. 7 *Table name : __DEFTCNV*

- **Data structure**

Structure defined in ' taio0tab.c ', ' appli.c ' and ' appli.h '

```
typedef struct
{
   floating point x;
   floating point y;
}
str_point;
```

Structure defined in ' appli.c ' and ' appli.h '

```
typedef struct
{
   uint16 tabNo.;
   uint16 No.pt;
   str_point *pt;
}
str_dftcnv;
```

Structure defined in ' appli.c ' and ' appli.h '

```
typedef struct
{
   uint16 No.;
   uint16 reserve;
   str_dftcnv tcnv[x];
}
str_def_tcnv_head;
```

- **Table declaration**

str_def_tcnv_head __DEFTCNV;

- **Table Size**

(2+2)+(2+2+4)*x = 4+8x

G. 4. 2. 2. 2. 3. 8 Table name : __DEFVAR

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   uint16 boo_int;
   uint16 boo_inp;
   uint16 boo_out;
   uint16 reserve1;
```

```
   uint16 ana_int;
   uint16 ana_inp;
   uint16 ana_out;
   uint16 reserve2;
   uint16 tmr_int;
   uint16 reserve3;
   uint16 reserve4;
   uint16 reserve5;
   uint16 msg_int;
   uint16 msg_inp;
   uint16 msg_out;
   uint16 reserve6;
}
str_def_var;
```

- **Table declaration**

str_def_var __DEFVAR;

- **Data size**

2*16 = 32

G. 4. 2. 2. 2. 3. 9 Table name : __CODEVAR

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   uint32 init_boo;
   uint32 init_ana;
   uint32 init_tmr;
   uint32 init_msg;
   uint32 lng_msg;
   uint32 def_mdb;
   uint32 retain;
   uint16 No._var_boo;
   uint16 No._var_ana;
   uint16 No._var_tmr;
   uint16 No._var_msg;
}
str_def_init;
```

- **Table declaration**

str_def_init __CODEVAR;

- **Table Size**

4*7+ 4*2 = 36

G. 4. 2. 2. 2. 3. 10 *Table name : __INITBOO*

- **Data structure**

uint16

- **Table declaration**

uint16 __INITBOO[x];

- **Table Size**

2x

G. 4. 2. 2. 2. 3. 11 *Table name : __INITANA*

- **Data structure**

Structure defined in ' appli.c '

```
typedef struct
{
   uint16 No.;
   uint32 value;
}
str_init_ana;
```

- **Table declaration**

str_init_ana __INITANA[x];

- **Data size**

(2+4)x = 6x

G. 4. 2. 2. 2. 3. 12 *Table name : __INITTMR*

- **Data structure**

Structure defined in ' appli.c '

```
typedef struct
{
   uint16 No.;
```

```
   uint32 value;
}
str_init_tmr;
```

- **Table declaration**

str_init_tmr __INITTMR[x];

- **Table Size**

(2+4)x = 6x

G. 4. 2. 2. 2. 3. 13 Table name : __INITMSG

- **Data structure**

Structure defined in ' appli.c '

```
typedef struct
{
   char *mes;
   uint16 No.;
   uchar len;
}
str_init_msg;
```

- **Table declaration**

str_init_msg __INITMSG[x];

- **Table Size**

(4+2+1)x = 7x

G. 4. 2. 2. 2. 3. 14 Table name : __LNGMSG

- **Data structure**

unsigned char

- **Table declaration**

unsigned char __LNGMSG [x];

- **Table Size**

1x = x

G. 4. 2. 2. 2. 3. 15 Table name : __DEFMODBUS

- **Data structure**

unsigned char

- **Table declaration**

unsigned char __DEFMODBUS [16];

- **Table Size**

16

G. 4. 2. 2. 2. 3. 16 *Table name : __DEFMDBBOO*

- **Data structure**

uint16

- **Table declaration**

uint16 __DEFMDBBOO [x];

- **Table Size**

2x

G. 4. 2. 2. 2. 3. 17 *Table name : __DEFMDBANA*

- **Data structure**

uint16

- **Table declaration**

uint16 __DEFMDBANA [x];

- **Table Size**

2x

G. 4. 2. 2. 2. 3. 18 *Table name : __DEFMDBTMR*

- **Data structure**

uint16

- **Table declaration**

uint16 __DEFMDBTMR [x];

- **Table Size**

2x

G. 4. 2. 2. 2. 3. 19 Table name : __DEFMDBMSG

- **Data structure**

uint16

- **Table declaration**

uint16 __DEFMDBMSG [x];

- **Table Size**

2x

G. 4. 2. 2. 2. 3. 20 Table name : __BACKUP

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   char param[64];
   uint16 No._boo;
   uint16 num_boo;
   uint16 No._ana;
   uint16 num_ana;
   uint16 No._tmr;
   uint16 num_tmr;
   uint16 No._msg;
   uint16 num_msg;
}
str_def_retain;
```

- **Table declaration**

str_def_retain __BACKUP;

- **Table Size**

64+(8*2) = 80

G. 4. 2. 2. 2. 3. 21 Table name : __DEFOEM

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   uint16 inp_brd;
   uint16 out_brd;
   uint16 cnv_real;
}
str_def_ios;
```

- **Table declaration**

str_def_ios __DEFOEM;

- **Table Size**

2*3 = 6

G. 4. 2. 2. 2. 3. 22 Table name : __CODEOEM

- **Data structure**

Structure defined in ' tasy0apl.h '

```
typedef struct
{
   uint32 oem_boo;
   uint32 oem_ana;
   uint32 oem_msg;
   uint32 oem_brd;
}
str_def_oem;
```

- **Table declaration**

str_def_ oem __ CODEOEM;

- **Table Size**

4*4 = 16

G. 4. 2. 2. 2. 3. 23 Table name : __DEFBOO

- **Data structure**

Structure defined in ' taio0def.h '

```
typedef struct
{
   uint16 vnum;
   uchar rack;
   uchar slot;
   uchar channel;
   uchar out;
   uchar conv_len;
   uchar rea_ana;
}
str_dfvar;
```

- **Table declaration**

str_dfvar __DEFBOO[x];

- **Table Size**

(2+6)x = 8x

G. 4. 2. 2. 2. 3. 24 Table name : __DEFANA

- **Data structure**

Structure defined in ' taio0def.h '

```
typedef struct
{
   uint16 vnum;
   uchar rack;
   uchar slot;
   uchar channel;
   uchar out;
   uchar conv_len;
   uchar rea_ana;
}
str_dfvar;
```

- **Table declaration**

str_dfvar __DEFANA[x];

- **Table Size**

(2+6)x = 8x

G. 4. 2. 2. 2. 3. 25 Table name : __DEFMSG

- **Data structure**

Structure defined in ' taio0def.h '

```
typedef struct
{
   uint16 vnum;
   uchar rack;
   uchar slot;
   uchar channel;
   uchar out;
   uchar conv_len;
   uchar rea_ana;
}
str_dfvar;
```

- **Table declaration**

str_dfvar __DEFMSG[x];

- **Table Size**

(2+6)x = 8x

G. 4. 2. 2. 2. 3. 26 Table name : __DEFBRD

- **Data structure**

uchar *

- **Table declaration**

uchar *__DEFBRD[x];

- **Table Size**

4x

G. 4. 2. 2. 2. 3. 27 Table name : __DEFBRD_n

- **Data structure**

Structure defined in ' appli.c '

```
typedef struct
{
```

```
   uint16 total_size;
   uint16 oem_key;
   uint16 first_var;
   uchar rack;
   uchar slot;
   uchar real;
   uchar type;
   uchar out_brd;
   uchar No.channel ;
   <type> field_0;
         ...
   <type> field_m;
}
str_dfbrd_n;
```

Note :

<*type*> is : uint16 , uint32 or table of 16 uchar.

- **Table declaration**

str_dfbrd_n __DEFBRD_n;

- **Table Size**

3*2 + 6*1 + (m+1)*2 = 14+ 2m (if type = uint16)
3*2 + 6*1 + (m+1)*4 = 16+ 4m (if type = uint32)
3*2 + 6*1 + (m+1)*16 = 28+ 16m (if type = 16 uchar)

G. 4. 2. 2. 2. 3. 28 Table name : __DEFHIE

- **Data structure**

Structure defined in ' appli.c ' and ' appli.h '

```
typedef struct
{
   uint16 begin;
   uint16 end;
   uint16 main;
   uint16 child;
   uint16 function;
   uint16 reserve;
   uint16 numlevel[x];
}
str_dfhie;
```

- **Table declaration**

str_dfhie __DEFHIE;

- **Table Size**

2*6+(begin+end+main.child+function)*2 = 12+(begin+end+main+child+function)*2

G. 4. 2. 2. 2. 3. 29 *Table name : __DEFPROG*

- **Data structure**

uint16

- **Table declaration**

uint16 __DEFPROG[x];

- **Table Size**

2x

G. 4. 2. 2. 2. 3. 30 *Table name : __CODEPROG*

- **Data structure**

unsigned char (**)(void);

- **Table declaration**

unsigned char (**__CODEPROG [x])(void);

- **Table Size**

4x

G. 4. 2. 2. 2. 3. 31 *Table name : __ ProgName_CODE*

- **Data structure**

unsigned char (*)(void);

- **Table declaration**

unsigned char (*__ProgName_CODE [x])(void);

- **Table Size**

4x

G. 4. 2. 2. 2. 3. 32 Table name : __LINKSFC

- **Data structure**

uint16 *

- **Table declaration**

uint16 *__LINKSFC [x];

- **Table Size**

4x

G. 4. 2. 2. 2. 3. 33 Table name : __ ProgName_LINK

- **Data structure**

uint16

- **Table declaration**

uint16 __ProgName_LINK [x];

- **Table Size**

2x

G. 4. 2. 2. 2. 3. 34 Table name : __RESSOURCE

- **Data structure**

uchar

- **Table declaration**

uchar __RESOURCE [x];

- **Table Size**

x

G. 4. 2. 3. External function

Declaration

```
extern void tic_cnv(unsigned short,unsigned short,unsigned short,unsigned short );
```

G. 4. 2. 4. External data

BF_BOO and **BF_ANA** are the two external variables used in this file. **BF_BOO** is a pointer to the memory block containing all the Boolean data of the application. **BF_ANA** is same as **BF_BOO** for the analog data.

Declaration	Meaning
extern long *BF_ANA	
extern unsigned char *BF_BOO	

G. 4. 2. 5. Static data

Local variables are use to store the relative address of function parameters.

Declaration	Meaning
static long LGC_RET_SUB	
static unsigned short LGC_P_NUM[32]	
static unsigned char LGC_P_TYP[32]	
static unsigned char LGC_BVAL	

G. 4. 2. 6. Declaration of the initialization function

This section declares the functions use to initialize the data tables. The number of functions is variable and depends on the application.

Initialization Function Prototype	Comment
independent of the application	
void apl_init_TABLE (void) ;	This function calls all others
void apl_init___MAIN (void) ;	
void apl_init___DEFAPPLI (void) ;	
void apl_init___DEFPROD (void) ;	
void apl_init___DEFFCNV (void) ;	
void apl_init___DEFTCNV (void) ;	
void apl_init___DEFOEM (void) ;	
void apl_init___CODEOEM (void) ;	
void apl_init___DEFBRD (void) ;	
void apl_init___DEFBOO (void) ;	
void apl_init___DEFANA (void) ;	
void apl_init___DEFMSG (void) ;	
void apl_init___CODEVAR (void) ;	
void apl_init___BACKUP (void) ;	
void apl_init___INITBOO (void) ;	
void apl_init___INITANA (void) ;	
void apl_init___INITTMR (void) ;	
void apl_init___INITMSG (void) ;	

void apl_init___DEFMODBUS (void) ;	
void apl_init___DEFMDBBOO (void) ;	
void apl_init___DEFMDBANA (void) ;	
void apl_init___DEFMDBTMR (void) ;	
void apl_init___DEFMDBMSG (void) ;	
void apl_init___DEFHIE (void) ;	
void apl_init___DEFPROG (void) ;	
void apl_init___CODEPROG (void) ;	
void apl_init___LINKSFC (void) ;	
void apl_init___DEFUSF (void) ;	
void apl_init___DEFFBL (void) ;	
void apl_init___DEFVAR (void) ;	
void apl_init___LNGMSG (void) ;	
void apl_init___RESOURCE (void);	
application dependent	
void apl_init___ProgName1_LINK (void);	for the program named : ProgName1
void apl_init___ProgName2_LINK (void);	for the program named : ProgName2
void apl_init___ProgName3_LINK (void);	for the program named : ProgName3
.....	.....
void apl_init___ProgName1_CODE (void);	for the program named : ProgName1
void apl_init___ProgName2_CODE (void);	for the program named : ProgName2
void apl_init___ProgName3_CODE (void);	for the program named : ProgName3
.....	.....

G. 4. 2. 7. Declaration of the processing code function

This section declares the functions containing the application code. The number of functions is variable and depends on the application.

Function Prototype	Comment
unsigned char __ ProgName1_CODE_TRT (void)	TRT = processing code for ProgName1
unsigned char __ ProgName2_CODE_ESTP0001 (void)	ESTP0001 = step number 1 of the SFC ProgName2
unsigned char __ ProgName2_CODE_TRS0002 (void)	TRS0002 = transition 2 of the SFC ProgName2
.....	.....

NOTE :
ProgName1 declaration means *ProgName1* is not a SFC (program, function etc.).
ProgName2 declaration means *ProgName2* is a SFC.

G. 4. 2. 8. Initialization data

These functions allow to initialize the data tables.

NOTE : the values used to initialize the tables are just some examples.

G. 4. 2. 8. 1. *apl_init_TABLE*

```
void apl_init_TABLE (void)
{
apl_init___MAIN ();
apl_init___DEFAPPLI ();
apl_init___DEFPROD ();
apl_init___DEFFCNV ();
apl_init___DEFTCNV ();
apl_init___DEFOEM ();
apl_init___CODEOEM ();
apl_init___DEFBRD ();
apl_init___DEFBOO ();
apl_init___DEFANA ();
apl_init___DEFMSG ();
apl_init___CODEVAR ();
apl_init___BACKUP ();
apl_init___INITBOO ();
apl_init___INITANA ();
apl_init___INITTMR ();
apl_init___INITMSG ();
apl_init___DEFMODBUS ();
apl_init___DEFMDBBOO ();
apl_init___DEFMDBANA ();
apl_init___DEFMDBTMR ();
apl_init___DEFMDBMSG ();
apl_init___DEFHIE ();
apl_init___DEFPROG ();
apl_init___CODEPROG ();
apl_init___LINKSFC ();
apl_init___DEFUSF ();
apl_init___DEFFBL ();
apl_init___DEFVAR ();
apl_init___LNGMSG ();
apl_init___ ProgName1_LINK ();
apl_init___ ProgName2_LINK ();
.................................................
apl_init___RESOURCE ();
apl_init___ ProgName1_CODE ();
apl_init___ ProgName2_CODE ();
.................................................
}
```

G. 4. 2. 8. 2. *apl_init___MAIN*

```
void apl_init___MAIN (void)
{
__MAIN.def_flag = 0x00000001;
```

```
__MAIN.def_prod = 0x00000064;
__MAIN.def_appli = 0x00000040;
__MAIN.def_usf = 0x000001f0;
__MAIN.def_fbl = 0x000001f4;
__MAIN.def_fcnv = 0x0000007c;
__MAIN.def_tcnv = 0x00000080;
__MAIN.def_var = 0x00000348;
__MAIN.tab_init = 0x000000ac;
__MAIN.def_ios = 0x00000084;
__MAIN.def_oem = 0x0000008c;
__MAIN.def_hie = 0x000001a8;
__MAIN.def_prog = 0x000001c0;
__MAIN.tab_code = 0x000001d0;
__MAIN.tab_link = 0x000001e0;
__MAIN.def_res = 0x0000091c;

}
```

G. 4. 2. 8. 3. *apl_init___DEFAPPLI*

```
void apl_init___DEFAPPLI (void)
{
sys_strcpy(__DEFAPPLI.name, 'rfwash');
__DEFAPPLI.date = 0x32a4ce5a;
__DEFAPPLI.crc = 0x0ae95f5c;
__DEFAPPLI.version = 0x0032;
__DEFAPPLI.error_level = 0x0001;
__DEFAPPLI.start_mode = 0x0000;
__DEFAPPLI.cycle_duration = 0x0000;

}
```

G. 4. 2. 8. 4. *apl_init___DEFPROD*

```
void apl_init___DEFPROD (void)
{
sys_strcpy(__DEFPROD.name, 'CC86M');
sys_strcpy(__DEFPROD.version, '1.00');

}
```

G. 4. 2. 8. 5. *apl_init___DEFFCNV*

```
void apl_init___DEFFCNV (void)
```

```
{
__DEFFCNV.No. = 0x0000;
__DEFFCNV.reserve = 0x0000;

}
```

G. 4. 2. 8. 6. *apl_init___DEFTCNV*

```
void apl_init___DEFTCNV (void)
{
__DEFTCNV.No. = 0x0000;
__DEFTCNV.reserve = 0x0000;

}
```

G. 4. 2. 8. 7. *apl_init___DEFOEM*

```
void apl_init___DEFOEM (void)
{
__DEFOEM.inp_brd = 0x0000;
__DEFOEM.out_brd = 0x0000;
__DEFOEM.cnv_real = 0x0000;

}
```

G. 4. 2. 8. 8. *apl_init___CODEOEM*

```
void apl_init___CODEOEM (void)
{
__CODEOEM[0]=0xa0;
.....................................
__CODEOEM[15]=0x00;
}
```

G. 4. 2. 8. 9. *apl_init___DEFBRD*

```
void apl_init___DEFBRD (void)
{
__DEFBRD[0] = (void*)0;

}
```

G. 4. 2. 8. 10. *apl_init___DEFBOO*

```
void apl_init___DEFBOO (void)
{
__DEFBOO[0].vnum = 0x0000;
__DEFBOO[0].rack = 0x00;
__DEFBOO[0].slot = 0x00;
__DEFBOO[0].channel = 0x00;
__DEFBOO[0].out = 0x00;
__DEFBOO[0].conv_len = 0x00;
__DEFBOO[0].rea_ana = 0x00;

}
```

G. 4. 2. 8. 11. *apl_init___DEFANA*

```
void apl_init___DEFANA (void)
{
__DEFANA[0].vnum = 0x0000;
__DEFANA[0].rack = 0x00;
__DEFANA[0].slot = 0x00;
__DEFANA[0].channel = 0x00;
__DEFANA[0].out = 0x00;
__DEFANA[0].conv_len = 0x00;
__DEFANA[0].rea_ana = 0x00;

}
```

G. 4. 2. 8. 12. *apl_init___DEFMSG*

```
void apl_init___DEFMSG (void)
{
__DEFMSG[0].vnum = 0x0000;
__DEFMSG[0].rack = 0x00;
__DEFMSG[0].slot = 0x00;
__DEFMSG[0].channel = 0x00;
__DEFMSG[0].out = 0x00;
__DEFMSG[0].conv_len = 0x00;
__DEFMSG[0].rea_ana = 0x00;

}
```

G. 4. 2. 8. 13. *apl_init___CODEVAR*

```
void apl_init___CODEVAR (void)
{
__CODEVAR.init_boo = 0x00000120;
__CODEVAR.init_ana = 0x00000124;
__CODEVAR.init_tmr = 0x00000150;
__CODEVAR.init_msg = 0x00000164;
__CODEVAR.lng_msg = 0x00000164;
__CODEVAR.def_mdb = 0x00000168;
__CODEVAR.retain = 0x000000d0;
__CODEVAR.No._var_boo = 0x0000;
__CODEVAR.No._var_ana = 0x0001;
__CODEVAR.No._var_tmr = 0x0000;
__CODEVAR.No._var_msg = 0x0000;

}
```

G. 4. 2. 8. 14. *apl_init___BACKUP*

```
void apl_init___BACKUP (void)
{
__BACKUP.param[0] = 0x00;
__BACKUP.param[1] = 0x00;
.........................................
__BACKUP.param[63] = 0x00;
__BACKUP.No._boo = 0x0000;
__BACKUP.num_boo = 0x0000;
__BACKUP.No._ana = 0x0000;
__BACKUP.num_ana = 0x0000;
__BACKUP.No._tmr = 0x0000;
__BACKUP.num_tmr = 0x0000;
__BACKUP.No._msg = 0x0000;
__BACKUP.num_msg = 0x0000;

}
```

G. 4. 2. 8. 15. *apl_init___INITBOO*

```
void apl_init___INITBOO (void)
{
__INITBOO[0] = 0x000d;
__INITBOO[1] = 0x0000;
...................................
}
```

G. 4. 2. 8. 16. *apl_init___INITANA*

```
void apl_init___INITANA (void)
{
__INITANA[0].No. = 0x0001;
__INITANA[0].value = 0x00000064;
__INITANA[1].No. = 0x0009;
__INITANA[1].value = 0x00000000;
.....................................................
}
```

G. 4. 2. 8. 17. *apl_init___INITTMR*

```
void apl_init___INITTMR (void)
{
__INITTMR[0].No. = 0x0001;
__INITTMR[0].value = 0x0000012c;
.....................................................
}
```

G. 4. 2. 8. 18. *apl_init___INITMSG*

```
void apl_init___INITMSG (void)
{
__INITMSG[0].No. = 0x0000;
__INITMSG[0].len = 0x00;
__INITMSG[0].mes = 0x00000000;
.....................................................
}
```

G. 4. 2. 8. 19. *apl_init___DEFMODBUS*

```
void apl_init___DEFMODBUS (void)
{
__DEFMODBUS[0]=0x78;
.....................................
__DEFMODBUS[15]=0x00;
}
```

G. 4. 2. 8. 20. *apl_init___DEFMDBBOO*

```
void apl_init___DEFMDBBOO (void)
{
__DEFMDBBOO[0] = 0x0000;
```

```
.............................................
__DEFMDBBOO[5] = 0x0000;
}
```

G. 4. 2. 8. 21. *apl_init___DEFMDBANA*

```
void apl_init___DEFMDBANA (void)
{
__DEFMDBANA[0] = 0x0000;
.............................................
__DEFMDBANA[5] = 0x0000;

}
```

G. 4. 2. 8. 22. *apl_init___DEFMDBTMR*

```
void apl_init___DEFMDBTMR (void)
{
__DEFMDBTMR[0] = 0x0000;
.............................................
__DEFMDBTMR[5] = 0x0000;

}
```

G. 4. 2. 8. 23. *apl_init___DEFMDBMSG*

```
void apl_init___DEFMDBMSG (void)
{
__DEFMDBMSG[0] = 0x0000;
.............................................
__DEFMDBMSG[5] = 0x0000;

}
```

G. 4. 2. 8. 24. *apl_init___DEFHIE*

```
void apl_init___DEFHIE (void)
{
__DEFHIE.begin = 0x0000;
__DEFHIE.end = 0x0001;
__DEFHIE.main = 0x0001;
__DEFHIE.child = 0x0001;
```

```
__DEFHIE.function = 0x0001;
__DEFHIE.reserve = 0x0000;
__DEFHIE.sfc_num = 0x0002;
__DEFHIE.sfc_level = 0x0000;

}
```

G. 4. 2. 8. 25. *apl_init___DEFPROG*

```
void apl_init___DEFPROG (void)
{
__DEFPROG[0] = 0x0002;
.....................................
}

void apl_init___CODEPROG (void)
{
__CODEPROG[0]=__SIMULATE_CODE;
.....................................................................
..
}
```

G. 4. 2. 8. 26. *apl_init___LINKSFC*

```
void apl_init___LINKSFC (void)
{

__LINKSFC[0]=(void*)0;
__LINKSFC[1]=__WASHMAIN_LINK;
__LINKSFC[2]=__AGITATE_LINK;
__LINKSFC[3]=(void*)0;
}
```

G. 4. 2. 8. 27. *apl_init___DEFUSF*

```
void apl_init___DEFUSF (void)
{
__DEFUSF.usfNo. = 0x0000;
__DEFUSF.reserved = 0x0000;
}
```

G. 4. 2. 8. 28. *apl_init___DEFFBL*

```
void apl_init___DEFFBL (void)
{
__DEFFBL.fblNo. = 0x0015;
__DEFFBL.reserved = 0x0000;
__DEFFBL.fbl[0].libno = 0x0001;
__DEFFBL.fbl[0].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[0].name, '#SIG_GEN');
__DEFFBL.fbl[1].libno = 0x0002;
__DEFFBL.fbl[1].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[1].name, '#RS');
__DEFFBL.fbl[2].libno = 0x0003;
__DEFFBL.fbl[2].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[2].name, '#SR');
__DEFFBL.fbl[3].libno = 0x0004;
__DEFFBL.fbl[3].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[3].name, '#R_TRIG');
__DEFFBL.fbl[4].libno = 0x0005;
__DEFFBL.fbl[4].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[4].name, '#F_TRIG');
__DEFFBL.fbl[5].libno = 0x0006;
__DEFFBL.fbl[5].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[5].name, '#CTU');
__DEFFBL.fbl[6].libno = 0x0007;
__DEFFBL.fbl[6].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[6].name, '#CTD');
__DEFFBL.fbl[7].libno = 0x0008;
__DEFFBL.fbl[7].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[7].name, '#CTUD');
__DEFFBL.fbl[8].libno = 0x0009;
__DEFFBL.fbl[8].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[8].name, '#TON');
__DEFFBL.fbl[9].libno = 0x000a;
__DEFFBL.fbl[9].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[9].name, '#TOF');
__DEFFBL.fbl[10].libno = 0x000b;
__DEFFBL.fbl[10].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[10].name, '#TP');
__DEFFBL.fbl[11].libno = 0x000c;
__DEFFBL.fbl[11].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[11].name, '#SEMA');
__DEFFBL.fbl[12].libno = 0x000d;
__DEFFBL.fbl[12].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[12].name, '#CMP');
__DEFFBL.fbl[13].libno = 0x000e;
__DEFFBL.fbl[13].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[13].name, '#STACKINT');
__DEFFBL.fbl[14].libno = 0x000f;
__DEFFBL.fbl[14].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[14].name, '#DERIVATE');
__DEFFBL.fbl[15].libno = 0x0010;
__DEFFBL.fbl[15].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[15].name, '#HYSTER');
__DEFFBL.fbl[16].libno = 0x0011;
```

```
__DEFFBL.fbl[16].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[16].name, '#AVERAGE');
__DEFFBL.fbl[17].libno = 0x0012;
__DEFFBL.fbl[17].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[17].name, '#LIM_ALRM');
__DEFFBL.fbl[18].libno = 0x0013;
__DEFFBL.fbl[18].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[18].name, '#BLINK');
__DEFFBL.fbl[19].libno = 0x0014;
__DEFFBL.fbl[19].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[19].name, '#INTEGRAL');
__DEFFBL.fbl[20].libno = 0x0015;
__DEFFBL.fbl[20].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[20].name, '#PID');
}
```

G. 4. 2. 8. 29. apl_init___DEFVAR

```
void apl_init___DEFVAR (void)
{
__DEFVAR.boo_int = 0x0011;
__DEFVAR.boo_inp = 0x0000;
__DEFVAR.boo_out = 0x0000;
__DEFVAR.reserve1 = 0x0000;
__DEFVAR.ana_int = 0x0012;
__DEFVAR.ana_inp = 0x0000;
__DEFVAR.ana_out = 0x0000;
__DEFVAR.reserve2 = 0x0000;
__DEFVAR.tmr_int = 0x0009;
__DEFVAR.reserve3 = 0x0000;
__DEFVAR.reserve4 = 0x0000;
__DEFVAR.reserve5 = 0x0000;
__DEFVAR.msg_int = 0x0000;
__DEFVAR.msg_inp = 0x0000;
__DEFVAR.msg_out = 0x0000;
__DEFVAR.reserve6 = 0x0000;
}
```

G. 4. 2. 8. 30. apl_init___LNGMSG

```
void apl_init___LNGMSG (void)
{
__LNGMSG[0]=0x00;
.................................
}
```

G. 4. 2. 8. 31. *apl_init___ProgName1_LINK*

```
void apl_init___ProgName1_LINK (void)
{
__ ProgName1_LINK[0] = 0x0001;
.....................................
}
```

G. 4. 2. 8. 32. *apl_init___ProgName2_LINK*

```
void apl_init___ProgName2_LINK (void)
{
__ ProgName2_LINK[0] = 0x000e;
.....................................
}
```

G. 4. 2. 8. 33. *apl_init___RESOURCE*

```
void apl_init___RESOURCE (void)
{
__RESOURCE[0]=0x00;
.....................................
}
```

G. 4. 2. 8. 34. *apl_init___ProgName_CODE*

```
void apl_init___ProgName1_CODE (void)
{
__ ProgName1 _CODE[0]=__ ProgName1_CODE_TRT;
.....................................
}

void apl_init___ ProgName2_CODE (void)
{
__ ProgName2_CODE[0]=__ ProgName2_CODE_BSTP0000;
__ ProgName2_CODE[1]=(void*)0;
.....................................
}
```

G. 4. 2. 9. Application code

These functions contain the code of the application. The details of the C instructions representing the body of these functions are defined in chapter C.

G. 4. 2. 9. 1. __ *ProgName_CODE_Name*

```
unsigned char __ ProgName1_CODE_TRT(void)
{
........   processing code of the application
........   see chapter G. 3.  for detail of C code
}

unsigned char __ ProgName2_CODE_BSTP0000(void)
{
........   processing code of the application
........   see chapter G. 3.  for detail of C code
}
```

G. 5. EXAMPLE : RFWASH

This chapter shows an example of C code generated by ISaGRAF V3.20. For this example C code is produced for the application RFWASH (as delivered with the product). RFWASH is compiled without optimization.

G. 5. 1. Appli.h

```
#ifndef LGC_VERS2
typedef struct
{
   uint32 def_flag;
   uint32 def_prod;
   uint32 def_appli;
   uint32 def_usf;
   uint32 def_fbl;
   uint32 def_fcnv;
   uint32 def_tcnv;
   uint32 def_var;
   uint32 tab_init;
   uint32 def_ios;
   uint32 def_oem;
   uint32 def_hie;
   uint32 def_prog;
   uint32 tab_code;
   uint32 tab_link;
   uint32 def_res;
}
str_header;
#endif
extern str_header __MAIN;
extern str_def_appli __DEFAPPLI;
extern str_def_prod __DEFPROD;
```

```
#ifndef LGC_VERS2
typedef struct
{
   uint16 libno;
   uint16 reserve;
   char name[10];
   uint16 reserve2;
}
str_dfcnv;
#endif
typedef struct
{
   uint16 No.;
   uint16 reserve;
   str_dfcnv fcnv[1];
}
str_def_fcnv_head;
extern str_def_fcnv_head __DEFFCNV;
typedef struct
{
   floating point x;
   floating point y;
}
str_point;
typedef struct
{
   uint16 tabNo.;
   uint16 No.pt;
   str_point *pt;
}
str_dftcnv;
typedef struct
{
   uint16 No.;
   uint16 reserve;
   str_dftcnv tcnv[1];
}
str_def_tcnv_head;
extern str_def_tcnv_head __DEFTCNV;
extern str_def_ios __DEFOEM;
extern unsigned char __CODEOEM [16];
typedef struct
{
   uint16 total_size;
   uint16 oem_key;
   uint16 first_var;
   uchar rack;
   uchar slot;
   uchar real;
   uchar type;
   uchar out_brd;
   uchar No.channel;
}
```

```
str_dfbrd_0;
extern str_dfbrd_0 __DEFBRD_0;
typedef struct
{
   uint16 total_size;
   uint16 oem_key;
   uint16 first_var;
   uchar rack;
   uchar slot;
   uchar real;
   uchar type;
   uchar out_brd;
   uchar No.channel;
}
str_dfbrd_1;
extern str_dfbrd_1 __DEFBRD_1;
extern uchar *__DEFBRD[2];
extern str_dfvar __DEFBOO[16];
extern str_dfvar __DEFANA[1];
extern str_dfvar __DEFMSG[1];
extern str_def_init __CODEVAR;
extern str_def_retain __BACKUP;
extern uint16 __INITBOO[2];
typedef struct
{
   uint16 No.;
   uint32 value;
}
str_init_ana;
extern str_init_ana __INITANA[1];
typedef struct
{
   uint16 No.;
   uint32 value;
}
str_init_tmr;
extern str_init_tmr __INITTMR[1];
typedef struct
{
   char *mes;
   uint16 No.;
   uchar len;
}
str_init_msg;
extern str_init_msg __INITMSG[1];
extern unsigned char __DEFMODBUS [16];
extern uint16 __DEFMDBBOO[6];
extern uint16 __DEFMDBANA[6];
extern uint16 __DEFMDBTMR[6];
extern uint16 __DEFMDBMSG[6];
extern str_def_hie __DEFHIE;
extern uint16 __DEFPROG[4];
extern unsigned char (**__CODEPROG [2])(void);
```

```
extern uint16 *__LINKSFC [2];
#ifndef LGC_VERS2
typedef struct
{
   uint16 libno;
   char name[10];
}
str_dfusf;
#endif
typedef struct
{
   uint16 usfNo.;
   uint16 reserved;
}
str_dfusf_head;
extern str_dfusf_head __DEFUSF;
#ifndef LGC_VERS2
typedef struct
{
   uint16 libno;
   uint16 No.inst;
   char name[12];
}
str_def_fbl;
#endif
typedef struct
{
   uint16 fblNo.;
   uint16 reserved;
   str_def_fbl fbl[20];
}
str_def_fbl_head;
extern str_def_fbl_head __DEFFBL;
extern str_def_var __DEFVAR;
extern unsigned char __LNGMSG [4];
extern uint16 __RUNSTOP_LINK[46];
extern unsigned char __RESOURCE [8];
extern unsigned char (*__COMMAND_CODE [1])(void);
extern unsigned char (*__RUNSTOP_CODE [13])(void);
extern void apl_init_TABLE (void);
```

G. 5. 2. Appli.c

INCLUDE

```
#include <tasy0def.h>
#include <take0pro.h>
#include <take0blk.h>
#include <take0ste.h>
#include <take0msg.h>
#include <take0prg.h>
```

```
#include <tasy0apl.h>
#include <taio0def.h>

                    DATA STRUCTURES AND TABLES
#ifndef LGC_VERS2
typedef struct
{
   uint32 def_flag;
   uint32 def_prod;
   uint32 def_appli;
   uint32 def_usf;
   uint32 def_fbl;
   uint32 def_fcnv;
   uint32 def_tcnv;
   uint32 def_var;
   uint32 tab_init;
   uint32 def_ios;
   uint32 def_oem;
   uint32 def_hie;
   uint32 def_prog;
   uint32 tab_code;
   uint32 tab_link;
   uint32 def_res;
}
str_header;
#endif
str_header __MAIN;
str_def_appli __DEFAPPLI;
str_def_prod __DEFPROD;
#ifndef LGC_VERS2
typedef struct
{
   uint16 libno;
   uint16 reserve;
   char name[10];
   uint16 reserve2;
}
str_dfcnv;
#endif
typedef struct
{
   uint16 No.;
   uint16 reserve;
   str_dfcnv fcnv[1];
}
str_def_fcnv_head;
str_def_fcnv_head __DEFFCNV;
typedef struct
{
   floating point x;
   floating point y;
}
```

```
str_point;
typedef struct
{
   uint16 tabNo.;
   uint16 No.pt;
   str_point *pt;
}
str_dftcnv;
typedef struct
{
   uint16 No.;
   uint16 reserve;
   str_dftcnv tcnv[1];
}
str_def_tcnv_head;
str_def_tcnv_head __DEFTCNV;
str_def_ios __DEFOEM;
unsigned char __CODEOEM [16];
typedef struct
{
   uint16 total_size;
   uint16 oem_key;
   uint16 first_var;
   uchar rack;
   uchar slot;
   uchar real;
   uchar type;
   uchar out_brd;
   uchar No.channel;
}
str_dfbrd_0;
str_dfbrd_0 __DEFBRD_0;
typedef struct
{
   uint16 total_size;
   uint16 oem_key;
   uint16 first_var;
   uchar rack;
   uchar slot;
   uchar real;
   uchar type;
   uchar out_brd;
   uchar No.channel;
}
str_dfbrd_1;
str_dfbrd_1 __DEFBRD_1;
uchar *__DEFBRD[2];
str_dfvar __DEFBOO[16];
str_dfvar __DEFANA[1];
str_dfvar __DEFMSG[1];
str_def_init __CODEVAR;
str_def_retain __BACKUP;
uint16 __INITBOO[2];
```

```
typedef struct
{
   uint16 No.;
   uint32 value;
}
str_init_ana;
str_init_ana __INITANA[1];
typedef struct
{
   uint16 No.;
   uint32 value;
}
str_init_tmr;
str_init_tmr __INITTMR[1];
typedef struct
{
   char *mes;
   uint16 No.;
   uchar len;
}
str_init_msg;
str_init_msg __INITMSG[1];
unsigned char __DEFMODBUS [16];
uint16 __DEFMDBBOO[6];
uint16 __DEFMDBANA[6];
uint16 __DEFMDBTMR[6];
uint16 __DEFMDBMSG[6];
str_def_hie __DEFHIE;
uint16 __DEFPROG[4];
unsigned char (**__CODEPROG [2])(void);
uint16 *__LINKSFC [2];
#ifndef LGC_VERS2
typedef struct
{
   uint16 libno;
   char name[10];
}
str_dfusf;
#endif
typedef struct
{
   uint16 usfNo.;
   uint16 reserved;
}
str_dfusf_head;
str_dfusf_head __DEFUSF;
#ifndef LGC_VERS2
typedef struct
{
   uint16 libno;
   uint16 No.inst;
   char name[12];
}
```

```
str_def_fbl;
#endif
typedef struct
{
   uint16 fblNo.;
   uint16 reserved;
   str_def_fbl fbl[20];
}
str_def_fbl_head;
str_def_fbl_head __DEFFBL;
str_def_var __DEFVAR;
unsigned char __LNGMSG [4];
uint16 __RUNSTOP_LINK[46];
unsigned char __RESOURCE [8];
unsigned char (*__COMMAND_CODE [1])(void);
unsigned char (*__RUNSTOP_CODE [13])(void);
```

EXTERNAL FUNCTION

```
extern   void   tic_cnv(unsigned   short,unsigned   short,unsigned
short,unsigned short );
```

EXTERNAL DATA

```
#ifndef DEF_NO_GLOBALS
extern long *BF_ANA;
extern unsigned char *BF_BOO;
#endif
```

STATIC DATA

```
static long LGC_RET_SUB;
static unsigned short LGC_P_NUM[32];
static unsigned char LGC_P_TYP[32];
static unsigned char LGC_BVAL;
```

DECLARATION OF THE INITIALIZATION FUNCT ONS

```
void apl_init_TABLE (void);
void apl_init___MAIN (void);
void apl_init___DEFAPPLI (void);
void apl_init___DEFPROD (void);
void apl_init___DEFFCNV (void);
void apl_init___DEFTCNV (void);
void apl_init___DEFOEM (void);
void apl_init___CODEOEM (void);
void apl_init___DEFBRD (void);
void apl_init___DEFBOO (void);
void apl_init___DEFANA (void);
void apl_init___DEFMSG (void);
void apl_init___CODEVAR (void);
void apl_init___BACKUP (void);
void apl_init___INITBOO (void);
void apl_init___INITANA (void);
void apl_init___INITTMR (void);
void apl_init___INITMSG (void);
```

```
void apl_init___DEFMODBUS (void);
void apl_init___DEFMDBBOO (void);
void apl_init___DEFMDBANA (void);
void apl_init___DEFMDBTMR (void);
void apl_init___DEFMDBMSG (void);
void apl_init___DEFHIE (void);
void apl_init___DEFPROG (void);
void apl_init___CODEPROG (void);
void apl_init___LINKSFC (void);
void apl_init___DEFUSF (void);
void apl_init___DEFFBL (void);
void apl_init___DEFVAR (void);
void apl_init___LNGMSG (void);
void apl_init___RUNSTOP_LINK (void);
void apl_init___RESOURCE (void);
void apl_init___COMMAND_CODE (void);
void apl_init___RUNSTOP_CODE (void);
```

DECLARATION OF THE PROCESSING CODE FUNCTIONS

```
unsigned char __COMMAND_CODE_TRT (void);
unsigned char __RUNSTOP_CODE_BSTP0002 (void);
unsigned char __RUNSTOP_CODE_ESTP0002 (void);
unsigned char __RUNSTOP_CODE_TRS0002 (void);
unsigned char __RUNSTOP_CODE_TRS0003 (void);
```

INITIALIZATION CODE

```
void apl_init_TABLE (void)
{
apl_init___MAIN ();
apl_init___DEFAPPLI ();
apl_init___DEFPROD ();
apl_init___DEFFCNV ();
apl_init___DEFTCNV ();
apl_init___DEFOEM ();
apl_init___CODEOEM ();
apl_init___DEFBRD ();
apl_init___DEFBOO ();
apl_init___DEFANA ();
apl_init___DEFMSG ();
apl_init___CODEVAR ();
apl_init___BACKUP ();
apl_init___INITBOO ();
apl_init___INITANA ();
apl_init___INITTMR ();
apl_init___INITMSG ();
apl_init___DEFMODBUS ();
apl_init___DEFMDBBOO ();
apl_init___DEFMDBANA ();
apl_init___DEFMDBTMR ();
apl_init___DEFMDBMSG ();
apl_init___DEFHIE ();
apl_init___DEFPROG ();
```

```
apl_init___CODEPROG ();
apl_init___LINKSFC ();
apl_init___DEFUSF ();
apl_init___DEFFBL ();
apl_init___DEFVAR ();
apl_init___LNGMSG ();
apl_init___RUNSTOP_LINK ();
apl_init___RESOURCE ();
apl_init___COMMAND_CODE ();
apl_init___RUNSTOP_CODE ();

}
void apl_init___MAIN (void)
{
__MAIN.def_flag = 0x00000001;
__MAIN.def_prod = 0x00000064;
__MAIN.def_appli = 0x00000040;
__MAIN.def_usf = 0x00000230;
__MAIN.def_fbl = 0x00000234;
__MAIN.def_fcnv = 0x0000007c;
__MAIN.def_tcnv = 0x00000080;
__MAIN.def_var = 0x00000378;
__MAIN.tab_init = 0x00000140;
__MAIN.def_ios = 0x00000084;
__MAIN.def_oem = 0x0000008c;
__MAIN.def_hie = 0x00000204;
__MAIN.def_prog = 0x00000218;
__MAIN.tab_code = 0x00000220;
__MAIN.tab_link = 0x00000228;
__MAIN.def_res = 0x00000470;

}
void apl_init___DEFAPPLI (void)
{
sys_strcpy(__DEFAPPLI.name, 'rfsample');
__DEFAPPLI.date = 0x32a86613;
__DEFAPPLI.crc = 0x0034eb4e;
__DEFAPPLI.version = 0x0009;
__DEFAPPLI.error_level = 0x0010;
__DEFAPPLI.start_mode = 0x0000;
__DEFAPPLI.cycle_duration = 0x0000;

}
void apl_init___DEFPROD (void)
{
sys_strcpy(__DEFPROD.name, 'CC86M');
sys_strcpy(__DEFPROD.version, '1.00');

}
void apl_init___DEFFCNV (void)
{
__DEFFCNV.No. = 0x0000;
__DEFFCNV.reserve = 0x0000;
```

```
}
void apl_init___DEFTCNV (void)
{
__DEFTCNV.No. = 0x0000;
__DEFTCNV.reserve = 0x0000;

}
void apl_init___DEFOEM (void)
{
__DEFOEM.inp_brd = 0x0001;
__DEFOEM.out_brd = 0x0001;
__DEFOEM.cnv_real = 0x0000;

}
void apl_init___CODEOEM (void)
{

__CODEOEM[0]=0xb8;__CODEOEM[1]=0x00;__CODEOEM[2]=0x00;__CODEOEM[3
]=0x00;
__CODEOEM[4]=0x38;__CODEOEM[5]=0x01;__CODEOEM[6]=0x00;__CODEOEM[7
]=0x00;
__CODEOEM[8]=0x3c;__CODEOEM[9]=0x01;__CODEOEM[10]=0x00;__CODEOEM[
11]=0x00;
__CODEOEM[12]=0x9c;__CODEOEM[13]=0x00;__CODEOEM[14]=0x00;__CODEOE
M[15]=0x00;
}
void apl_init___DEFBRD (void)
{
__DEFBRD[0] = (uchar *)&__DEFBRD_0;
__DEFBRD_0.total_size = 0x000c;
__DEFBRD_0.oem_key = 0x0000;
__DEFBRD_0.first_var = 0x0001;
__DEFBRD_0.rack = 0x00;
__DEFBRD_0.slot = 0x00;
__DEFBRD_0.real = 0x01;
__DEFBRD_0.type = 0x01;
__DEFBRD_0.out_brd = 0x00;
__DEFBRD_0.No.channel = 0x08;
__DEFBRD[1] = (uchar *)&__DEFBRD_1;
__DEFBRD_1.total_size = 0x000c;
__DEFBRD_1.oem_key = 0x0000;
__DEFBRD_1.first_var = 0x0009;
__DEFBRD_1.rack = 0x00;
__DEFBRD_1.slot = 0x01;
__DEFBRD_1.real = 0x01;
__DEFBRD_1.type = 0x01;
__DEFBRD_1.out_brd = 0x01;
__DEFBRD_1.No.channel = 0x08;

}
void apl_init___DEFBOO (void)
{
```

```
__DEFBOO[0].vnum = 0x0001;
__DEFBOO[0].rack = 0x00;
__DEFBOO[0].slot = 0x00;
__DEFBOO[0].channel = 0x00;
__DEFBOO[0].out = 0x00;
__DEFBOO[0].conv_len = 0x00;
__DEFBOO[0].rea_ana = 0x00;
__DEFBOO[1].vnum = 0x0002;
__DEFBOO[1].rack = 0x00;
__DEFBOO[1].slot = 0x00;
__DEFBOO[1].channel = 0x01;
__DEFBOO[1].out = 0x00;
__DEFBOO[1].conv_len = 0x00;
__DEFBOO[1].rea_ana = 0x00;
__DEFBOO[2].vnum = 0x0003;
__DEFBOO[2].rack = 0x00;
__DEFBOO[2].slot = 0x00;
__DEFBOO[2].channel = 0x02;
__DEFBOO[2].out = 0x00;
__DEFBOO[2].conv_len = 0x00;
__DEFBOO[2].rea_ana = 0x00;
__DEFBOO[3].vnum = 0x0004;
__DEFBOO[3].rack = 0x00;
__DEFBOO[3].slot = 0x00;
__DEFBOO[3].channel = 0x03;
__DEFBOO[3].out = 0x00;
__DEFBOO[3].conv_len = 0x00;
__DEFBOO[3].rea_ana = 0x00;
__DEFBOO[4].vnum = 0x0005;
__DEFBOO[4].rack = 0x00;
__DEFBOO[4].slot = 0x00;
__DEFBOO[4].channel = 0x04;
__DEFBOO[4].out = 0x00;
__DEFBOO[4].conv_len = 0x00;
__DEFBOO[4].rea_ana = 0x00;
__DEFBOO[5].vnum = 0x0006;
__DEFBOO[5].rack = 0x00;
__DEFBOO[5].slot = 0x00;
__DEFBOO[5].channel = 0x05;
__DEFBOO[5].out = 0x00;
__DEFBOO[5].conv_len = 0x00;
__DEFBOO[5].rea_ana = 0x00;
__DEFBOO[6].vnum = 0x0007;
__DEFBOO[6].rack = 0x00;
__DEFBOO[6].slot = 0x00;
__DEFBOO[6].channel = 0x06;
__DEFBOO[6].out = 0x00;
__DEFBOO[6].conv_len = 0x00;
__DEFBOO[6].rea_ana = 0x00;
__DEFBOO[7].vnum = 0x0008;
__DEFBOO[7].rack = 0x00;
__DEFBOO[7].slot = 0x00;
__DEFBOO[7].channel = 0x07;
```

```
__DEFBOO[7].out = 0x00;
__DEFBOO[7].conv_len = 0x00;
__DEFBOO[7].rea_ana = 0x00;
__DEFBOO[8].vnum = 0x0009;
__DEFBOO[8].rack = 0x00;
__DEFBOO[8].slot = 0x01;
__DEFBOO[8].channel = 0x00;
__DEFBOO[8].out = 0x01;
__DEFBOO[8].conv_len = 0x00;
__DEFBOO[8].rea_ana = 0x00;
__DEFBOO[9].vnum = 0x000a;
__DEFBOO[9].rack = 0x00;
__DEFBOO[9].slot = 0x01;
__DEFBOO[9].channel = 0x01;
__DEFBOO[9].out = 0x01;
__DEFBOO[9].conv_len = 0x00;
__DEFBOO[9].rea_ana = 0x00;
__DEFBOO[10].vnum = 0x000b;
__DEFBOO[10].rack = 0x00;
__DEFBOO[10].slot = 0x01;
__DEFBOO[10].channel = 0x02;
__DEFBOO[10].out = 0x01;
__DEFBOO[10].conv_len = 0x00;
__DEFBOO[10].rea_ana = 0x00;
__DEFBOO[11].vnum = 0x000c;
__DEFBOO[11].rack = 0x00;
__DEFBOO[11].slot = 0x01;
__DEFBOO[11].channel = 0x03;
__DEFBOO[11].out = 0x01;
__DEFBOO[11].conv_len = 0x00;
__DEFBOO[11].rea_ana = 0x00;
__DEFBOO[12].vnum = 0x000d;
__DEFBOO[12].rack = 0x00;
__DEFBOO[12].slot = 0x01;
__DEFBOO[12].channel = 0x04;
__DEFBOO[12].out = 0x01;
__DEFBOO[12].conv_len = 0x00;
__DEFBOO[12].rea_ana = 0x00;
__DEFBOO[13].vnum = 0x000e;
__DEFBOO[13].rack = 0x00;
__DEFBOO[13].slot = 0x01;
__DEFBOO[13].channel = 0x05;
__DEFBOO[13].out = 0x01;
__DEFBOO[13].conv_len = 0x00;
__DEFBOO[13].rea_ana = 0x00;
__DEFBOO[14].vnum = 0x000f;
__DEFBOO[14].rack = 0x00;
__DEFBOO[14].slot = 0x01;
__DEFBOO[14].channel = 0x06;
__DEFBOO[14].out = 0x01;
__DEFBOO[14].conv_len = 0x00;
__DEFBOO[14].rea_ana = 0x00;
__DEFBOO[15].vnum = 0x0010;
__
```

```
__DEFBOO[15].rack = 0x00;
__DEFBOO[15].slot = 0x01;
__DEFBOO[15].channel = 0x07;
__DEFBOO[15].out = 0x01;
__DEFBOO[15].conv_len = 0x00;
__DEFBOO[15].rea_ana = 0x00;

}
void apl_init___DEFANA (void)
{
__DEFANA[0].vnum = 0x0000;
__DEFANA[0].rack = 0x00;
__DEFANA[0].slot = 0x00;
__DEFANA[0].channel = 0x00;
__DEFANA[0].out = 0x00;
__DEFANA[0].conv_len = 0x00;
__DEFANA[0].rea_ana = 0x00;

}
void apl_init___DEFMSG (void)
{
__DEFMSG[0].vnum = 0x0000;
__DEFMSG[0].rack = 0x00;
__DEFMSG[0].slot = 0x00;
__DEFMSG[0].channel = 0x00;
__DEFMSG[0].out = 0x00;
__DEFMSG[0].conv_len = 0x00;
__DEFMSG[0].rea_ana = 0x00;

}
void apl_init___CODEVAR (void)
{
__CODEVAR.init_boo = 0x000001b4;
__CODEVAR.init_ana = 0x000001b8;
__CODEVAR.init_tmr = 0x000001bc;
__CODEVAR.init_msg = 0x000001c0;
__CODEVAR.lng_msg = 0x000001c0;
__CODEVAR.def_mdb = 0x000001c4;
__CODEVAR.retain = 0x00000164;
__CODEVAR.No._var_boo = 0x0000;
__CODEVAR.No._var_ana = 0x0000;
__CODEVAR.No._var_tmr = 0x0000;
__CODEVAR.No._var_msg = 0x0000;

}
void apl_init___BACKUP (void)
{
__BACKUP.param[0] = 0x00;
__BACKUP.param[1] = 0x00;
__BACKUP.param[2] = 0x00;
__BACKUP.param[3] = 0x00;
__BACKUP.param[4] = 0x00;
__BACKUP.param[5] = 0x00;
```

```
__BACKUP.param[6] = 0x00;
__BACKUP.param[7] = 0x00;
__BACKUP.param[8] = 0x00;
__BACKUP.param[9] = 0x00;
__BACKUP.param[10] = 0x00;
__BACKUP.param[11] = 0x00;
__BACKUP.param[12] = 0x00;
__BACKUP.param[13] = 0x00;
__BACKUP.param[14] = 0x00;
__BACKUP.param[15] = 0x00;
__BACKUP.param[16] = 0x00;
__BACKUP.param[17] = 0x00;
__BACKUP.param[18] = 0x00;
__BACKUP.param[19] = 0x00;
__BACKUP.param[20] = 0x00;
__BACKUP.param[21] = 0x00;
__BACKUP.param[22] = 0x00;
__BACKUP.param[23] = 0x00;
__BACKUP.param[24] = 0x00;
__BACKUP.param[25] = 0x00;
__BACKUP.param[26] = 0x00;
__BACKUP.param[27] = 0x00;
__BACKUP.param[28] = 0x00;
__BACKUP.param[29] = 0x00;
__BACKUP.param[30] = 0x00;
__BACKUP.param[31] = 0x00;
__BACKUP.param[32] = 0x00;
__BACKUP.param[33] = 0x00;
__BACKUP.param[34] = 0x00;
__BACKUP.param[35] = 0x00;
__BACKUP.param[36] = 0x00;
__BACKUP.param[37] = 0x00;
__BACKUP.param[38] = 0x00;
__BACKUP.param[39] = 0x00;
__BACKUP.param[40] = 0x00;
__BACKUP.param[41] = 0x00;
__BACKUP.param[42] = 0x00;
__BACKUP.param[43] = 0x00;
__BACKUP.param[44] = 0x00;
__BACKUP.param[45] = 0x00;
__BACKUP.param[46] = 0x00;
__BACKUP.param[47] = 0x00;
__BACKUP.param[48] = 0x00;
__BACKUP.param[49] = 0x00;
__BACKUP.param[50] = 0x00;
__BACKUP.param[51] = 0x00;
__BACKUP.param[52] = 0x00;
__BACKUP.param[53] = 0x00;
__BACKUP.param[54] = 0x00;
__BACKUP.param[55] = 0x00;
__BACKUP.param[56] = 0x00;
__BACKUP.param[57] = 0x00;
__BACKUP.param[58] = 0x00;
```

```
__BACKUP.param[59] = 0x00;
__BACKUP.param[60] = 0x00;
__BACKUP.param[61] = 0x00;
__BACKUP.param[62] = 0x00;
__BACKUP.param[63] = 0x00;
__BACKUP.No._boo = 0x0000;
__BACKUP.num_boo = 0x0000;
__BACKUP.No._ana = 0x0000;
__BACKUP.num_ana = 0x0000;
__BACKUP.No._tmr = 0x0000;
__BACKUP.num_tmr = 0x0000;
__BACKUP.No._msg = 0x0000;
__BACKUP.num_msg = 0x0000;

}
void apl_init___INITBOO (void)
{
__INITBOO[0] = 0x0012;
__INITBOO[1] = 0x0000;

}
void apl_init___INITANA (void)
{
__INITANA[0].No. = 0x0000;
__INITANA[0].value = 0x00000000;

}
void apl_init___INITTMR (void)
{
__INITTMR[0].No. = 0x0000;
__INITTMR[0].value = 0x00000000;

}
void apl_init___INITMSG (void)
{
__INITMSG[0].No. = 0x0000;
__INITMSG[0].len = 0x00;
__INITMSG[0].mes = 0x00000000;

}
void apl_init___DEFMODBUS (void)
{

__DEFMODBUS[0]=0xd4;__DEFMODBUS[1]=0x01;__DEFMODBUS[2]=0x00;__DEF
MODBUS[3]=0x00;
__DEFMODBUS[4]=0xe0;__DEFMODBUS[5]=0x01;__DEFMODBUS[6]=0x00;__DEF
MODBUS[7]=0x00;
__DEFMODBUS[8]=0xec;__DEFMODBUS[9]=0x01;__DEFMODBUS[10]=0x00;__DE
FMODBUS[11]=0x00;
__DEFMODBUS[12]=0xf8;__DEFMODBUS[13]=0x01;__DEFMODBUS[14]=0x00;__
DEFMODBUS[15]=0x00;
}
void apl_init___DEFMDBBOO (void)
```

```
{
__DEFMDBBOO[0] = 0x0000;
__DEFMDBBOO[1] = 0x0000;
__DEFMDBBOO[2] = 0x0000;
__DEFMDBBOO[3] = 0x0000;
__DEFMDBBOO[4] = 0x0000;
__DEFMDBBOO[5] = 0x0000;

}
void apl_init___DEFMDBANA (void)
{
__DEFMDBANA[0] = 0x0000;
__DEFMDBANA[1] = 0x0000;
__DEFMDBANA[2] = 0x0000;
__DEFMDBANA[3] = 0x0000;
__DEFMDBANA[4] = 0x0000;
__DEFMDBANA[5] = 0x0000;

}
void apl_init___DEFMDBTMR (void)
{
__DEFMDBTMR[0] = 0x0000;
__DEFMDBTMR[1] = 0x0000;
__DEFMDBTMR[2] = 0x0000;
__DEFMDBTMR[3] = 0x0000;
__DEFMDBTMR[4] = 0x0000;
__DEFMDBTMR[5] = 0x0000;

}
void apl_init___DEFMDBMSG (void)
{
__DEFMDBMSG[0] = 0x0000;
__DEFMDBMSG[1] = 0x0000;
__DEFMDBMSG[2] = 0x0000;
__DEFMDBMSG[3] = 0x0000;
__DEFMDBMSG[4] = 0x0000;
__DEFMDBMSG[5] = 0x0000;

}
void apl_init___DEFHIE (void)
{
__DEFHIE.begin = 0x0001;
__DEFHIE.end = 0x0000;
__DEFHIE.main = 0x0001;
__DEFHIE.child = 0x0000;
__DEFHIE.function = 0x0000;
__DEFHIE.reserve = 0x0000;
__DEFHIE.sfc_num = 0x0002;
__DEFHIE.sfc_level = 0x0000;

}
void apl_init___DEFPROG (void)
{
```

```
__DEFPROG[0] = 0x0001;
__DEFPROG[1] = 0x0000;
__DEFPROG[2] = 0x0003;
__DEFPROG[3] = 0x0000;

}
void apl_init___CODEPROG (void)
{

__CODEPROG[0]=__COMMAND_CODE;__CODEPROG[1]=__RUNSTOP_CODE;
}
void apl_init___LINKSFC (void)
{

__LINKSFC[0]=(void*)0;__LINKSFC[1]=__RUNSTOP_LINK;
}
void apl_init___DEFUSF (void)
{
__DEFUSF.usfNo. = 0x0000;
__DEFUSF.reserved = 0x0000;

}
void apl_init___DEFFBL (void)
{
__DEFFBL.fblNo. = 0x0014;
__DEFFBL.reserved = 0x0000;
__DEFFBL.fbl[0].libno = 0x0001;
__DEFFBL.fbl[0].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[0].name, '#SIG_GEN');
__DEFFBL.fbl[1].libno = 0x0002;
__DEFFBL.fbl[1].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[1].name, '#RS');
__DEFFBL.fbl[2].libno = 0x0003;
__DEFFBL.fbl[2].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[2].name, '#SR');
__DEFFBL.fbl[3].libno = 0x0004;
__DEFFBL.fbl[3].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[3].name, '#R_TRIG');
__DEFFBL.fbl[4].libno = 0x0005;
__DEFFBL.fbl[4].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[4].name, '#F_TRIG');
__DEFFBL.fbl[5].libno = 0x0006;
__DEFFBL.fbl[5].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[5].name, '#CTU');
__DEFFBL.fbl[6].libno = 0x0007;
__DEFFBL.fbl[6].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[6].name, '#CTD');
__DEFFBL.fbl[7].libno = 0x0008;
__DEFFBL.fbl[7].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[7].name, '#CTUD');
__DEFFBL.fbl[8].libno = 0x0009;
__DEFFBL.fbl[8].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[8].name, '#TON');
```

```
__DEFFBL.fbl[9].libno = 0x000a;
__DEFFBL.fbl[9].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[9].name, '#TOF');
__DEFFBL.fbl[10].libno = 0x000b;
__DEFFBL.fbl[10].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[10].name, '#TP');
__DEFFBL.fbl[11].libno = 0x000c;
__DEFFBL.fbl[11].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[11].name, '#SEMA');
__DEFFBL.fbl[12].libno = 0x000d;
__DEFFBL.fbl[12].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[12].name, '#CMP');
__DEFFBL.fbl[13].libno = 0x000e;
__DEFFBL.fbl[13].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[13].name, '#STACKINT');
__DEFFBL.fbl[14].libno = 0x000f;
__DEFFBL.fbl[14].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[14].name, '#DERIVATE');
__DEFFBL.fbl[15].libno = 0x0010;
__DEFFBL.fbl[15].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[15].name, '#HYSTER');
__DEFFBL.fbl[16].libno = 0x0011;
__DEFFBL.fbl[16].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[16].name, '#AVERAGE');
__DEFFBL.fbl[17].libno = 0x0012;
__DEFFBL.fbl[17].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[17].name, '#LIM_ALRM');
__DEFFBL.fbl[18].libno = 0x0013;
__DEFFBL.fbl[18].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[18].name, '#BLINK');
__DEFFBL.fbl[19].libno = 0x0014;
__DEFFBL.fbl[19].No.inst = 0x0000;
sys_strcpy(__DEFFBL.fbl[19].name, '#INTEGRAL');

}
void apl_init___DEFVAR (void)
{
__DEFVAR.boo_int = 0x0004;
__DEFVAR.boo_inp = 0x0008;
__DEFVAR.boo_out = 0x0008;
__DEFVAR.reserve1 = 0x0000;
__DEFVAR.ana_int = 0x0001;
__DEFVAR.ana_inp = 0x0000;
__DEFVAR.ana_out = 0x0000;
__DEFVAR.reserve2 = 0x0000;
__DEFVAR.tmr_int = 0x0000;
__DEFVAR.reserve3 = 0x0000;
__DEFVAR.reserve4 = 0x0000;
__DEFVAR.reserve5 = 0x0000;
__DEFVAR.msg_int = 0x0000;
__DEFVAR.msg_inp = 0x0000;
__DEFVAR.msg_out = 0x0000;
__DEFVAR.reserve6 = 0x0000;
__
```

```
}
void apl_init___LNGMSG (void)
{

__LNGMSG[0]=0x00;__LNGMSG[1]=0x00;__LNGMSG[2]=0x00;__LNGMSG[3]=0x
00;
}
void apl_init___RUNSTOP_LINK (void)
{
__RUNSTOP_LINK[0] = 0x0001;
__RUNSTOP_LINK[1] = 0x0004;
__RUNSTOP_LINK[2] = 0x0001;
__RUNSTOP_LINK[3] = 0x0003;
__RUNSTOP_LINK[4] = 0x0024;
__RUNSTOP_LINK[5] = 0x0028;
__RUNSTOP_LINK[6] = 0x002c;
__RUNSTOP_LINK[7] = 0x0032;
__RUNSTOP_LINK[8] = 0x0036;
__RUNSTOP_LINK[9] = 0x003a;
__RUNSTOP_LINK[10] = 0x0000;
__RUNSTOP_LINK[11] = 0x003e;
__RUNSTOP_LINK[12] = 0x0042;
__RUNSTOP_LINK[13] = 0x0046;
__RUNSTOP_LINK[14] = 0x004a;
__RUNSTOP_LINK[15] = 0x004e;
__RUNSTOP_LINK[16] = 0x0052;
__RUNSTOP_LINK[17] = 0x0056;
__RUNSTOP_LINK[18] = 0x0001;
__RUNSTOP_LINK[19] = 0x0000;
__RUNSTOP_LINK[20] = 0x0002;
__RUNSTOP_LINK[21] = 0x0000;
__RUNSTOP_LINK[22] = 0x0004;
__RUNSTOP_LINK[23] = 0x0002;
__RUNSTOP_LINK[24] = 0x0000;
__RUNSTOP_LINK[25] = 0x0003;
__RUNSTOP_LINK[26] = 0x0000;
__RUNSTOP_LINK[27] = 0x0003;
__RUNSTOP_LINK[28] = 0x0000;
__RUNSTOP_LINK[29] = 0x0004;
__RUNSTOP_LINK[30] = 0x0000;
__RUNSTOP_LINK[31] = 0x0001;
__RUNSTOP_LINK[32] = 0x0000;
__RUNSTOP_LINK[33] = 0x0001;
__RUNSTOP_LINK[34] = 0x0000;
__RUNSTOP_LINK[35] = 0x0002;
__RUNSTOP_LINK[36] = 0x0000;
__RUNSTOP_LINK[37] = 0x0002;
__RUNSTOP_LINK[38] = 0x0000;
__RUNSTOP_LINK[39] = 0x0003;
__RUNSTOP_LINK[40] = 0x0000;
__RUNSTOP_LINK[41] = 0x0003;
__RUNSTOP_LINK[42] = 0x0000;
```

```
__RUNSTOP_LINK[43] = 0x0002;
__RUNSTOP_LINK[44] = 0x0000;
__RUNSTOP_LINK[45] = 0x0000;

}
void apl_init___RESOURCE (void)
{

__RESOURCE[0]=0x00;__RESOURCE[1]=0x00;__RESOURCE[2]=0x00;__RESOUR
CE[3]=0x00;
__RESOURCE[4]=0xa2;__RESOURCE[5]=0xf3;__RESOURCE[6]=0xf1;__RESOUR
CE[7]=0xff;
}
void apl_init___COMMAND_CODE (void)
{

__COMMAND_CODE[0]=__COMMAND_CODE_TRT;
}
void apl_init___RUNSTOP_CODE (void)
{

__RUNSTOP_CODE[0]=(void*)0;__RUNSTOP_CODE[1]=(void*)0;__RUNSTOP_C
ODE[2]=(void*)0;__RUNSTOP_CODE[3]=(void*)0;
__RUNSTOP_CODE[4]=(void*)0;__RUNSTOP_CODE[5]=(void*)0;__RUNSTOP_C
ODE[6]=__RUNSTOP_CODE_BSTP0002;__RUNSTOP_CODE[7]=(void*)0;
__RUNSTOP_CODE[8]=__RUNSTOP_CODE_ESTP0002;__RUNSTOP_CODE[9]=(void
*)0;__RUNSTOP_CODE[10]=(void*)0;__RUNSTOP_CODE[11]=__RUNSTOP_CODE
_TRS0002;
__RUNSTOP_CODE[12]=__RUNSTOP_CODE_TRS0003;
}
unsigned char __RUNSTOP_CODE_BSTP0002 (void)
{
ios_boo_ref(9);
BF_BOO[9] = BF_BOO[18];
return ((unsigned char)(0));

}
```

APPLICATION CODE

```
unsigned char __COMMAND_CODE_TRT (void)
{
BF_BOO[20] = (unsigned char)(BF_BOO[1] & BF_BOO[2]);
BF_BOO[17] = BF_BOO[20];
_LAB1:;
return ((unsigned char)(0));

}
unsigned char __RUNSTOP_CODE_ESTP0002 (void)
{
ios_boo_ref(9);
BF_BOO[9] = BF_BOO[19];
return ((unsigned char)(0));
```

```
}
unsigned char __RUNSTOP_CODE_TRS0002 (void)
{
return ((unsigned char)(BF_BOO[17]));

}
unsigned char __RUNSTOP_CODE_TRS0003 (void)
{
BF_BOO[20] =  ! BF_BOO[17];
return ((unsigned char)(BF_BOO[20]));

}
```