Балт-Систем
Balt-System

NC-110, NC-210, NC-201M

PLC
INTERFACE PROGRAMMING

Saint-Petersburg
2008

PUBLICATION ISSUED BY:

Balt-System Ltd.
198206, Sankt-Petersburg
Petergofskoje Shosse, 73

Telefax: +7(812)744 70 59
Telephone: +7(812)744 70 59

Important User Information

PREFACE

This manual describes the PLC development system that can be used on the NC-110, NC-210 systems, and provides the user with the all the information he needs to program a machine logic. The manual has been written for machine tool manufacturers intending to fit their machines with NC-110, NC-210 control systems, and for machine logic programmers in general.

To use this manual, a general understanding of the NC-110, NC-210 hardware and software architecture, and the principles of Boolean algebra, on which PLC programming is based, is required.

CONTENTS

1. INTRODUCTION

The PLC (Programmable Interface System) is a development system. This manual explains how the PLC system can be used to develop a machine logic on the NC-110, NC-210 systems and to adapt its typical functions to the control of a particular machine tool.

The PLC user can program his own personal machine logic according to his special requirements and the machine tool configuration. This means that the NC-110, NC-210 systems can be installed on any machine tool irrespective of its system features. The machine logic installed on the NC-110 system acts as the communication interface between the machine tool and the numerical control, and will hereafter be referred to as the PLC interface.

The PLC includes a set of programming tools that the user can use directly on the NC-110, NC-210 systems to develop the machine logic. These tools can be employed in all the stages that determine machine logic programming. The first part of the manual explains how to use these tools.

The PLC interface handles a series of standard activities in the process control operated by the NC-110, NC-210 systems. The second part of the manual describes how these activities are handled by the NC-110, NC-210 systems, and indicates all the interchange signals involved.

2. FEATURES AND PERFORMANCES

The PLC is a development system that provides the user with a set of tools for programming a machine logic that can be used to adapt the NC-110 system to a particular machine tool according to the user's specific requirements and the machine tool system features.

2.1. MACHINE LOGIC AND PLC INTERFACE

The machine logic is basically a program that is written in a particular language, and contains all the information needed to customize and interface the numerical control with the machine tool.

A compiler is used to translate the machine logic program into a language that can be understood by the system, and the program is then installed in the system memory as part of the system software in order to be processed.

In this respect, the machine logic can be considered a system software module that acts as software interface, handling a series of standard control activities and all the activities defined by the user in the numerical control customization.

The terms program and logic program are used indifferently in this manual when referring to the machine logic in all its various phases of programming.

2.2. DEVELOPMENT TOOLS

The machine logic is developed on the NC-110 system by using the programming tools offered by the PLC. This chapter provides all the information and references needed to use the PLC programming tools listed below:

- PLC language;
- PLC compiler;
- Debugger.

Language

The PLC language is a Boolean type language based on simple logical expressions that reflect the combinatorial logic hardware elements with which users of the system are familiar. This language consists of a set of elements that are linked accordirng to established criteria to write the individual statements of a logic program.

The basic language elements consist of signals and groups of signals (words) that contain the information handled by the interface. The statements that make up the machine logic basically consist of signals or words that are linked by appropriate operators.

The user may handle the standard activities with a set of signals that has already been defined (see Chapter 8), or he may define new signals according to his own application requirements. Chapters 3 and 4 of this manual give an operational description of the language.

Compiler

The PLC compiler is a program that interprets the PLC language statements and translates them into a language that can be understood by the NC-110 system computer.

All the information on how to use the compiler and the logic program compiling procedures is given in Chapter 5.

Debugger

The PLC debugger is a program that offers the user a series of procedures for executing the logic program operational debugging. This operational debugging makes it possible to correct, improve and test the logic program during execution.

Refer to Chapter 5 for information on debugging.

2.3. DEVELOPMENT AMBIENT

All the various phases in the machine logic programming and its installation in the memory are performed on the NC-110 system according to defined hardware-software system settings.

Software Ambient

The machine logic performs the interface function as a software task that is part of the system software. The system software is based on a series of tasks that are divided into two groups according to their functions: service software and process software.

The PLC interface is therefore considered a software task that forms part of the process software, i.e., it belongs to the system software group that performs all the functions relating to the numerical control of a class of machine tools.

The PLC development system represents a service software task, i.e., it belongs to the group of system software that gives the user access to the system, and offers a series of support features and service utilities.

The diagram below illustrates how the PLC development system and the PLC interface are located in the NC-110 system software structure.

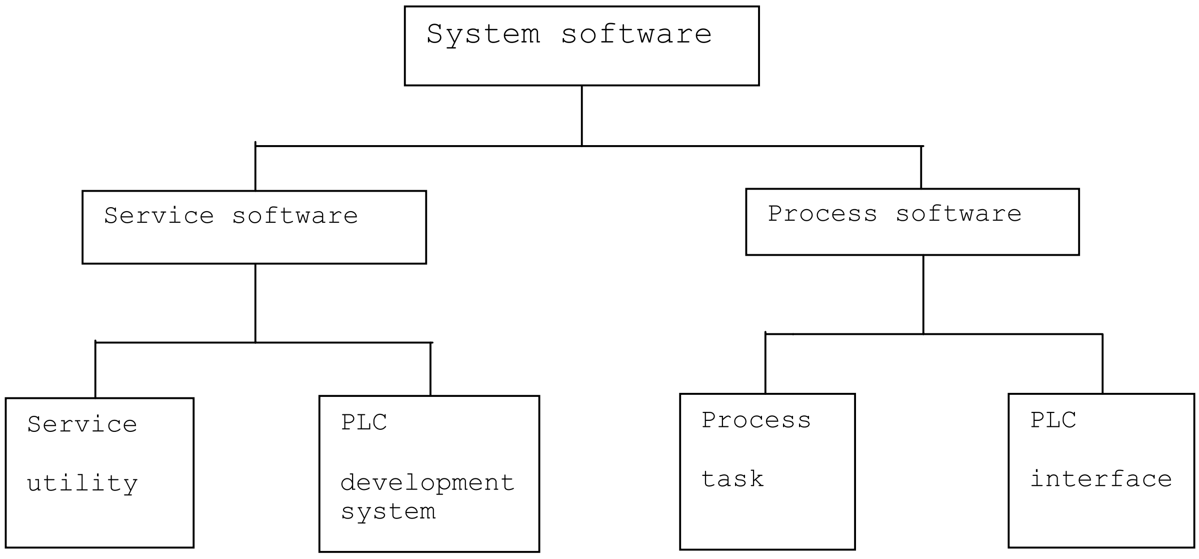


Fig. 2.l. - PLC and System Software

Software Presettings

To program a machine logic, it is first necessary to preset a special structure, called AMBIENT, that contains the parameters for the execution of the compiling and debugging phases (refer to Chapter 5, "Setting the PLC AMBIENT"). In particular, the memory addresses for object logic loading and execution, and the memory addresses where the logical and physical I/O signals are loaded and executed during machine logic execution must be declared.

The time dedicated by the system to machine logic execution must be declared in the IOCFIL configuration file (refer to "Machine logic execution procedure" described in this chapter).

After compiling and debugging have been completed, the machine logic can be stored on the EPROMs. This is done by updating the FCRSYS configuration file by means of appropriate settings, and if necessary, modifying parameters in AMBIENT.

Hardware Ambient

The machine logic is written and introduced into the system by using the NC-110 system console keyboard. A system editor is used to perform all the operations involving user program catalogueing and modification.

A work memory area is available on the CMOS RAM modules for program entry. This memory is a permanent memory that retains data even after control switch-off.

To process the executable object program (after debugging), it must first be transferred to the system memory, where there are 64 Kbytes available on the EPROM or CMOS RAM modules.

The NC-110 system CPU processes the machine logic while concurrently performing other typical system activities, such as axis handling, user program analysis etc.

Hardware Presettings

As has already been said, the PLC development system offers the user a series of features that enable him to translate the logic program into machine language, and test it before installing it on the EPROM. These operations are performed by the compiler and the debugger, and are typical features of the service software.

2.4. MACHINE LOGIC EXECUTION PROCEDURE

The system dedicates a certain amount of cyclic time, defined during configuration, to machine logic processing, while concurrently performing other activities such as axis handling, program analysis etc. This amount of time is not normally sufficient to execute the entire machine logic, which means that the machine logic has to be executed in several stages spread over successive cycles. Each cycle is called sampling and lasts as long as the cyclic period defined during configuration.

Fast Logic and Slow Logic

During machine logic processing, certain events must be analyzed at very frequent intervals as they require extremely short operate times. The machine is therefore divided into two parts, fast logic and slow logic.

The fast logic contains the statements relevant to the operations that must be executed very frequently. The whole of this logic is performed at each sampling.

The slow logic contains the statements that may be executed less frequently than the fast logic statements. The slow logic can therefore be interrupted during one sampling operation and resumed at the next.

In short all the fast logic and a part of the slow logic is executed at each sampling operation; slow logic execution is spread over several sampling operations and the amount of time it takes to complete it (equal to an integer multiple of the sampling time) is called logic cycle. Each sampling operation not only executes the fast logic but also updates the timers defined in the machine logic.

Software Settings

The total time taken by the system to execute the whole logic (slow and fast), and the partial periods dedicated to executing the slow logic and fast logic are defined in the IOCFIL configuration file and in the PLC AMBIENT.

The amount of cyclic time that the system dedicates to processing the logic (slow and fast), while concurrently performing the other process tasks, and the amount of time dedicated to processing the slow logic must be defined in the IOCFIL file by setting the following parameters in the triletteral CLO:

- Logic tick time: cyclic time (in milliseconds) dedicated to analyzing the machine logic (slow and fast) and to the various process tasks. This time defines the amount of time it takes to perform a single sampling operation.

- Logic task time: time (in milliseconds) dedicated to analyzing the slow logic in each sampling operation.

The logic task time parameter must be less than or equal to half the logic tick time parameter. In this manual, all the time values refer to the standard logic tick that is equal to 10 milliseconds. If a logic tick is adopted that is a multiple of 10 milliseconds according to a certain ratio, the timer time values must be incremented in the same ratio.

The maximum periods dedicated to analyzing the fast logic and the slow logic are declared in AMBIENT with the following parameters:

- Max. fast time: maximum amount of time (in microseconds) dedicated to analyzing the fast logic in each sampling operation.

- Max. slow time: maximum amount of time (in milliseconds) dedicated to analyzing the slow logic in several sampling operations.

The max. fast time parameter must be less than or equal to half the logic task time declared in the IOCFIL file. Refer to Chapter 5, "Setting the PLC ambient" for further details.

The slow logic and fast logic can be thought of as tasks belonging to the process software described previously. The actual amount of time required to execute the fast logic task, the slow logic task and the timers task is displayed by the system after the machine logic has been compiled.

Example:

The example given below illustrates a machine logic execution sequence based on values given to the parameters that establish the execution times of the various tasks:

IOCFIL file setting:

```
CLO=10,2
Logic tick time = 10 milliseconds
Logic task time =  2 milliseconds
```

PLC AMBIENT setting:

```
Max. fast time: 1000 (microseconds)
Max. slow time: 10 (milliseconds)
```

The system displays the results of compilation as follows:

1) Timer task 500 microseconds;
2) Fast task 500 microseconds;
3) Slow task 7826 microseconds.

Fig. 2.2. illustrates the execution sequence of the various tasks according to the settings indicated above. It can be seen that the slow logic task is executed over several sampling operations and the fast logic task and the timers task are executed at each sampling. The symbol "#" indicates that a task has been completed. The following notations are used:

LV	Fast logic task
LL	Slow logic task
TI	Timers task
TP	Process task and other activities

```
 -------------------------------------------------------------
        |--|--|       |              |  |  |       |         |
        |LV|TI|LL (1) |      TP      |LV|TI|LL (2) |   TP    |
        |  |  |       |              |  |  |       |         |
        |  |  |       |              |  |  |       |         |
        |  |  |       |              |  |  |       |         |
        |# | #|       |              |# | #|       |         |
-time-> |--|--|-------|------------>-|--|--|-------|------->-|
 (ms)   0                          10                      20

        |--|--|-------|              |--|--|-------|         |
        |  |  |       |              |  |  |       |         |
        |LV|TI|LL (3) |      TP      |LV|TI|LL (4) |   TP    |
        |  |  |       |              |  |  |       |         |
        |  |  |       |              |  |  |       |         |
        |  |  |       |              |  |  |       |         |
        |# |# |       |              |# |# |#      |         |
        |--|--|-------|------------>-|--|--|-------|------->-|
       20                          30                      40
```

Fig. 2.2. - Execution sequence

The slow logic execution requires four logic tick cycles (sampling) equal to a total of 40 ms. The number placed after LL indicates the number of sampling cycles necessary.

3. PLC LANGUAGE

3.1. LANGUAGE CHARACTERISTICS

The PLC development system includes a programming language that provides the user with the tools necessary for writing the machine logic. The PLC language is a Boolean type language consisting of a series of elements that can be combined according to certain rules to write the statements that form a logic program.

The statements use logical expressions to define and control the physical and logical I/O signals that constitute the variables containing the information handled by the interface. The variables used by the machine logic are stored in special storage areas called connector racks.

3.1.1. ELEMENTS

The PLC language consists of a set of elements that can be divided into five functional groups:

- Operands;
- Metaoperands;
- Functions;
- Blocks;
- Operators.

The operands represent the language variables and the information handled by the PLC interface. This group includes: Signals and words.

The metaoperands emulate the functions of a series of electrical and electronic devices such as counters, relays, etc. This group includes: Timers, Counters, Pulse Generators, ASCII Comparators.

The functions provide a series of support features used to convert data and handle its representation format. This group includes: Encoders, Decoders, BCD Conversion, Binary Conversion, Modulus, Sign, Multiplexers, Half Words.

The blocks are special statements that permit the execution of blocks of statements. This group includes the statements: DOF, ENDF, DOE, ENDE.

The operators link operands, metaoperands and functions to make up the program statements. These operators are subdivided into three categories: mathematical, comparison and logical.

3.1.2. VARIABLES AND CONSTANTS

Variables

The language variables, that consist of operands, are used to represent the information processed by the system machine logic in real time, and indicate the current process status.

The simplest type of variable is the signal that can assume only two logic values: 0 and 1. These two values are considered logic levels and correspond respectively to the boolean concepts "false" and "true". More complex word variables can be obtained by arranging the signals in groups. A word represents a group of eight signals and consequently can assume a wider range of values: from 0 to 255 in decimal notation.

The variables (signals and words) are univocally identified by a symbolic representation format. The user can use a set of pre-defined variables, included in the standard activities handled via the interface (initialization, machine tool safety devices etc.), but he can also define new variables according to the customization required by the control.

The variables storage areas are identified by grouping the signals and words in connectors. A connector conventionally represents 32 signals divided into 4 words, and the connector racks are the variables storage areas.

Constants

When writing the machine logic, it may be necessary to define constant values for certain elements or parameters. This occurs, for example, when a constant value is to be assigned to a word, or if a timer time base or a counter preset value are to be defined. In these cases, the constant value can be represented with the following expression:

nnn f

where:

nnn	Numerical value that can be expressed in three different formats.

F	Numerical value representaion format. This parameter can assume the following values:
	D = decimal format;
	0 = octal format;
	H = hexadecimal format.

The numerical value can vary within the range of values indicated below according to the format selected:

- Decimal format: 000-255;

- Octal format: 000-377;

- Hexadecimal format: OOO-OFF.

If the most significant figure in the hexadecimal format is a letter, it must be preceded by a zero. For example, if the hexadecimal value FF is to be represented, OFF must be written. The letter D can be omitted in the decimal format.

Example:

The expression 2DH represents a constant value in hexadecimal format. The same value can be represented as follows in the other formats:

- Decimal format: 45;
- Octal format: 550.

3.1.3. CONNECTOR RACKS

The connectors, that by definition consist of 32 signals, are split up onto three segments, called connector racks, in the system memory. These three racks, indicated below, are defined according to the functions of the signals in the connectors:

- Rack A containins connectors physical I/O. The numbers are declared in the IOCFIL configuratoin file.

- Rack K containing 256 connectors numbered 0 to 255.

- Rack T containing 16 connectors numbered 0 to 15.

Rack A

Rack A stores the signals that relate to the physical and logical I/Os of the machine tool system. The user assigns these signals by means of the machine logic; input signals are sent to the control from the electric cabinet, and output signals are sent from the control to the electric cabinet.

The NC-110 system is capable of recognizing the inputs from the machine tool and sending the outputs to the actuators with a 24 V.D.C voltage. The contents of rack A are reset when the control is switched off.

Rack K

Rack K stores the logical I/O signals for interface/process software communication. One part of rack K contains pre-defined signals relevant to the standard control activities handled by the interface. These signals are described in Chapter 8 "Interface Signals". The user may use these signals but must not alter their contents. The rest of rack K is available for the user to define the signals he requires.

26 connectors are reserved for the pre-defined signals in each process, and therefore the number of connectors available for user signals depends on the number of processes configured. The first 10 connectors in each configured process are reserved for the machine logic input signals. The remaining 16 connectors are reserved for the machine logic output signals.

The table given below indicates the number of the configured process and the corresponding numbers of the reserved and free connectors, and the input/output signals.

Table 3.1. - Rack K connectors

CONFIGURED PROCESS	RESERVED CONNECTORS	FREE CONNECTORS	INPUT SIGNALS	OUTPUT SIGNALS
1	0-25	26-255	0-9	10-25
2	26-51	52-255	26-35	36-51
3	52-77	78-255	52-61	62-77
4	78-1O3	104-255	78-87	88-103
5	104-129	130-255	104-113	114-129

If three processes are configured, the connectors are divided into reserved connectors and free connectors as follows:

- Reserved connectors: 0-77 (standard activity signals);
- Free connectors: 78-255 (user signals).

If a rack K word is used as a counter preset value or a timer time base, the next word must be set to zero (see Chapter 4, "List of elements").

It should be remembered that the contents of rack K will be deleted when the control is switched off, and there will therefore be no variables in this storage area when the control is switched back on. Rack T should be used if varibles are to be stored after switch-off.

Rack T

Rack T contains logical I/O variables that are retained even after control switch-off. This feature is particularly useful when constant values of certain parameters (preset values, time bases etc.) are used in the machine logic, or if the last value assumed by one or more variables before switch-off is to be retrieved at control switch-on.

The variables in rack T are handled by the machine logic in one of the two modes indicated below:

- Variables accessible by the machine logic only in read operations.

- Variables accessible by the machine logic in read and write operations.

The read only variables are declared by the user in section 4 of the IOCFIL characterization file. The variables specified appear to the right of the equals sign in the machine logic assignments and are therefore input signals with values that are read directly by the machine logic in rack T.

The IOCFIL file contains 64 records, numbered 1 to 64, to define up to 64 words in the 16 connectors of rack T. The words declared in IOCFIL represent constant values that are to be retained after the control is switched off. The numbers of the words must increase starting from connector zero. The IOCFIL file records and rack T words correspond as follows:

```
T01 = W00T0
T02 = W00T1
T03 = W00T2
...
T63 = W15T2
T64 = W15T3
```

where TOl, TO2,..., Tl28 are the IOCFIL file records. The word values are given in hexadecimal.

The read/write accessible variables are stored by the machine logic by means of assignment statements, and must not be declared in IOCFIL. These variables appear to the left of the equals sign in the assignment statements and are therefore output signals with values that are stored in rack T. Of course, once these variables have been stored in rack T, they are also read accessible.

The read only words and the read/write words have no particular position order in rack T. If a word in rack T is used as a counter preset value or as a timer time base, the next word must be set to zero (refer to Chapter 4, "List of elements").

Figure 3.1. gives an example of the organization inside rack T.

Fig. 3.1. - Rack T structure

WORD TYPE	BUFFER T
word declared in IOCFIL	W0T0
word declared in IOCFIL	W0T1
word declared in IOCFIL	W0T2
word written by machine logic	W0T3
word declared in IOCFIL	W1T0
word containing a preset value	W1T1
word set to zero	W1T2 = 0
	....
	....

The read only handling of rack T signals permits the constant values of certain parameters to be modified when necessary, without modifying the machine logic. A simple modification of only a few of the records in section 4 of the IOCFIL file, and the values of the parameters involved will be updated to the required value the first time the control is switched on.

Example:

The example given below shows how a rack T variable im used in the machine logic, and indicates the setting to be performed in IOCFIL.

Machine logic statement:
T26I(W1T0)=U50K1

This statement defines timer 26 with time base equal to the value of the word WO1TO in rack T.

IOCFIL setting:
T05=3C
T06=0

The constant value of the word WO1TO is established in record TO5 as equal to 3C hexadecimal i.e. 60 decimal. Timer 26 has therefore a time base of 60 and therefore a delay time of 6 seconds.

The time base value can be changet simply by altering the hexadecimal value specified in record TO5 in section 4 of IOCFIL, leaving the machine logic statement relating to the timer input unaltered. Record TO6, that follows the record containing the count task, is set to zero which is equivalent to setting W1T1=0.

3.1.4. VARIABLES MEMORY MAP

The variables used in the programming of the machine logic are stored in the connector racks described in the previous section of this manual.
Table 3.2. below indicates the storage addresses (in hexadecimal) of the variables that are grouped according to the connector rack they belong to and their functions. Each line of the table gives the initial address (in hexadecimal), the final address (in hexadecimal), and the amound of memory (in bytes) taken up by each group of variables in a given process.

Table 3.2. - Variables memory map

PROC	INITIAL ADDRESS	FINAL ADDRESS	TYPE OF VARIABLES
			RACK K INPUT VARIABLES AREA
1	2705:0000	2705:0027	40 bytes from 00K to 09K
2	2705:0068	2705:008F	40 bytes from 26K to 35K
3	2705:00D0	2705:00F7	40 bytes from 52K to 61K
4	2705:0138	2705:015F	40 bytes from 78K to 87K
5	2705:01A0	2705:01C7	40 bytes from 144K to 113K
			RACK K OUTPUT VARIABLES AREA
1	2705:0028	2705:0067	64 bytes from 10K to 25K
2	2705:0090	2705:00CF	64 bytes from 36K to 51K
3	2705:00F8	2705:0137	64 bytes from 62K to 77K
4	2705:0160	2705:019F	64 bytes from 88K to 103K
5	2705:01C8	2705:0207	64 bytes from 114K to 129K
			RACK K USER VARIABLES AREA
1	2705:0068	2705:03FF	920 bytes from 26K to 255K
2	2705:00D0	2705:03FF	816 bytes from 52K to 255K
3	2705:0138	2705:03FF	712 bytes from 78K to 255K
4	2705:01A0	2705:03FF	608 bytes from 104K to 255K
5	2705:0208	2705:03FF	504 bytes from 130K to 255K
	2705:0590	2705:060F	RACK T VARIABLES AREA
all			64 bytes from 0T to 15T
	2762:0000	2762:007F	PHYSICAL I/Os TO/FROM M.T.AREA
all			24 bytes from 0A to 5A

where: BASE-address for relative version SOFTWARE (for V1.5-BASE=2705)

3.1.5. STATEMENTS

A machine logic program consists of a series of statements that translate information for machine tool interfacing into PLC language. Each statement takes up a program line. The program lines must not be numbered and are processed in the order in which they are written.

The statements contain all the information needed for customization and machine tool interfacing, and are written by using all the language elements defined in this chapter. A statement basically consists of an assignment operation based on a Boolean expression.

A Boolean expression is an expression consisting of operands, metaoperands, constants and functions that are linked by operators. Two types of statements can be identified depending on the type of Boolean expression:

- Signal statements.

- Word statements.

Signal Statements

The signal statements are assignments using boolean expressions on signals according to the following format:

signal=Boolean expression on signal

where:

signal	Output signal to which the logic level in the Boolean expression on signal is assigned.
Boolean expression on signal	Expression containing operands, metaoperands, constants and functions linked by operators. The result of this expression is a signal.

In this type of assignment, only output signals may appear to the left of the equals sign, whereas both input and output signals may appear to the right of the equals sign.

Example:

In this example an output signal is defined by assigning it to a Boolean expression on signal.

U26K0=U50K2&[W26K1>W27K0]+U50K1

Word Statements

The word statements consist of assignments by Boolean expressions on words according to the following format:

word=Boolean expression on word

where:

word	Word to which the value in the Boolean expression on word is assigned.
Boolean expression on word	Expression containing operands, metaoperands, constants and functions linked by operators. The result of this expression is a word.

Example:

In this example a word is defined by assigning it to a Boolean expression on word.

W26K0=[W26K1+W50K0]*[W26K2-W50K1]

The Boolean expression on word cannot contain the MUX function. The MUX function can be assigned to a word only if it is alone. This means that the MUX function must appear to the right of the equals sign by itself, as illustrated below:

word=MUX(word1,word2,...,wordn),(sig1,sig2,...,sign)

Block statements

The block statements DOE, DOF, ENDE, ENDF, however, are different. These statements already have their own meaning and are not defined by assignment. Each statement is entered separately on a program line and does not appear in the Boolean expressions. For further information, refer to the section on "Blocks" that is given later in this chapter.

3.1.6. LOGIC SYNCHRONIZATION

With reference to the information given on the machine logic in Chapter 2, "Machine Logic execution procedure", it should be noted that the machine logic is executed with a given synchronization.

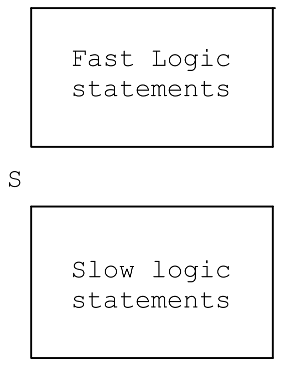
It should also be remembered that the statements to be executed frequently must be entered in the fast logic, and those statements that are to be executed less frequently must be entered in the slow logic.

The fast logic and the slow logic are separated by the "S" character. This character tells the system that the statements that follow belong to the slow logic.

The "S" character must be entered on a program line between the last fast logic statement and the first slow logic statement. Figure 3.2. illustrates the machine logic structure and the "S" character separating the two sections.

If the source program consists of several source tasks, the fast logic and the "S" character must be contained in the first task that is declared in AMBIENT in the "source pathname" parameter.

Fig. 3.2. - Separation of Fast Logic and Slow Logic



3.1.7. COMMENTS

Comments can be entered in the machine logic to make the machine logic source code more comprehensible. These comments are not translated into machine language and therefore do not take up space in the object program.

The first character in a comment line must be ";". This character tells the compiler that the characters that follow belong to a comment and must not be compiled.

3.2. LANGUAGE ELEMENTS

3.2.1. OPERANDS

Operands are language elements that represent the information handled by the machine logic, and constitute the language variables. This group includes the following elements:

- Signals;
- Words.

Signals

The signals represent elementary information consisting of a single digital electrical signal that may assume two logic levels, 0 and 1. The signals may be of the logical type or the physical type. The logical signals are used to communicate between the PLC interface and the process software, and they carry indications and requests from the interface or process software. The physical signals relate to the physical devices on the machine tool such as solenoid valves, relays, actuators, microswitches, sensors, pressure switches etc.

All the signals can be either input or output signals. The input signals contain information sent to the interface, the output signals contain information that is output from the interface. Special symbols make it possible to distinguish the input and output signals (see Chapter 4).

The signals defined in the machine logic are stored in appropriate system memory areas and are grouped on connectors that are divided into connector racks. A connector is the name given to a group of 32 signals numbered 0 to 31.

Words

A word consists of a group of eight consecutive signals on a given connector. Four words numbered 0 to 3 can be identified on each connector. The words and groups of eight signals on a connector correspond as follows:

WORD	**GROUP OF SIGNALS**
0	0-7
1	8-15
2	16-23
3	24-31

The value assumed by a word can be represented in three different formats: decimal, octal, hexadecimal. The word can assume the extreme values indicated below according to its format:

FORMAT	**EXTREME VALUES**
Decimal	000-255
Octal	000-377
Hexadecimal	000-0FF

Refer to Chapter 4 "List of elements" for a description of the representation format of signals and words.

Mnemonic representation

The words and signals can be identified by sbols, as described in Chapter 4, or by mnemonic symbols defined by the user. This mnemonic representation means that a variable can be assigned a mnemonic name that immediately identifies its function. For example the signal U100K0 can be assigned the mnemonic name EMERGENCY if this name helps identify the signal.

The mnemonic name consists of an alphanumerical string that is limited to a maximum length of 127 characters, i.e. the maximum number accepted by the PLC compiler. Only the first six characters, however, are significant. For example in the case of the variable EMERGENCY, the name assigned by the compiler will be EMERGE and therefore names such as EMERGENCY AXES and EMERGENCY SPINDLE will both be identified by the name EMERGE, which is ambiguous. In this case, the second definition entered will generate the error message "Mnemonic name already declared".

Chapter 8, "Interface signals", lists the mnemonic names of the variables corresponding to the standard system activities. These names should be used to simplify source program reading.

The user, however, can define the variables according to his own numerical control. Simbolic names and mnemonic names can be used indifferently within the same program. The mnemonic names must be preceded by the character "!". The symbolic name of a variable, however, must be defined.

Cross reference file

If mnemonic names are to be used, a mnemonic-sbol cross reference file must be prepared to establish the correspondence between the symbolic name and the mnemonic name with an expression of the following type:

mnemonic name = symbolic name

The character "*" must be entered at the beginning of this cross reference file, after which a reference comment can be entered.

Example:

```
* EXAMPLE OF A CROSS REFERENCE FILE
SUPPERMAN=W26K0
OLIVE OIL=I27K3
24H=24H
POPEYE=I19A2
BATMAN=W26K1
ROBIN=W26K2
```

During compilation, the cross reference file must be declared in AMBIENT as the first file to be compiled (source pathname 0), otherwise an error message is given. If an error is encountered in a file record, an error message is given and the relevant mnemonic name is rejected. All references by the logic program to the rejected name produce error messages. After compilation, the mnemonic names in the cross reference file are arranged in alphabetical order.

Mnemonic names table

A table, that lists the mnemonic names in alphabetical order, is created during cross reference file compilation. When using the debugger, it is necessary to make sure that this mnemonic table is stored in the memory as the debugger only has access to the mnemonic names table and not to the cross reference file.

Each time the logic program is compiled, a new table is created with the list of names that are currently defined in the cross reference file. If a logic program containing mnemonic names has been compiled without the corresponding cross reference file, simply compile the cross reference file to obtain the mnemonic names table.

3.2.2. METAOPERANDS

Metaoperands are elements consisting up of a group of operands defined to emulate functions that are typical of a series of hardware devices. The language includes the following metaoperands:

- timers;
- counters;
- pulse generators;
- ASCII comparators

Timers

A timer features a set of signals and duplicates the functions of a delay relay. Its main characteristic is the user-defined delay time expressed in seconds. This metaoperand normally has the function of timing a given event (coolant on, spindle start, logic enable etc.) in accordance with the delay time. This means for example, that a given event can last for a period of time equal to the delay time, or it can be triggered after a period of time equal to the programmed delay has elapsed.

Appropriate symbols represent the various signals that define the timer (refer to Chapter 4, "List of elements"). The user can employ 64 timers as follows:

- Forty eight long programmable timers with a delay time of 0.1 to 25.5 seconds (T00-T47);
- Sixteen short progranunable timers with a delay time of 0.01 to 2.55 seconds (T48-T63).

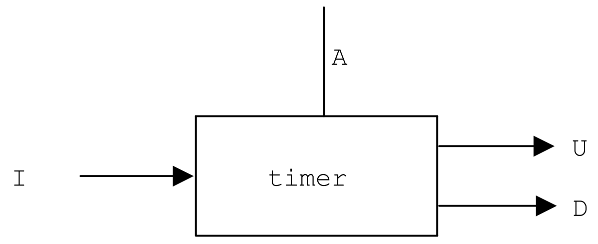
Timer operation features four signals that are identified with the names and sbols indicated below (see Chapter 4):

- TxxI = input;
- TxxA = enable;
- TxxU = delayed output;
- TxxD = pulse output.

The characters "xx", in the identification sumbology, represent the number of the timer.

From the logic point of view, a timer can be represented as a device with two inputs (input and enable) and two outputs (pulse and delayed).

Fig. 3.3. - Timer Logic Diagram



Input. Input is a user-defined signal that initializes the timer and defines the delay time. If the input is not defined, the timer outputs are not activated. Timing is activated on the input rising edge (transaction from 0 to 1), and deactivated on the input falling edge (transaction from 1 to 0).

Enable. Enable is a signal defined by the user that enables or disables timer operation depending on its status:

- If A=0 the timer is enabled;
- If A=1 the timer is disabled.

The enable signal is only effective when the timing is active (i.e. if I=1). If the enable signal rises to 1, the time value reached and the status of the delayed and pulse outputs are frozen, while the input continues to operate. When the enable signal returns to 0, the timing starts from the value it was stopped at and the outputs continue from the point at which they were blocked.

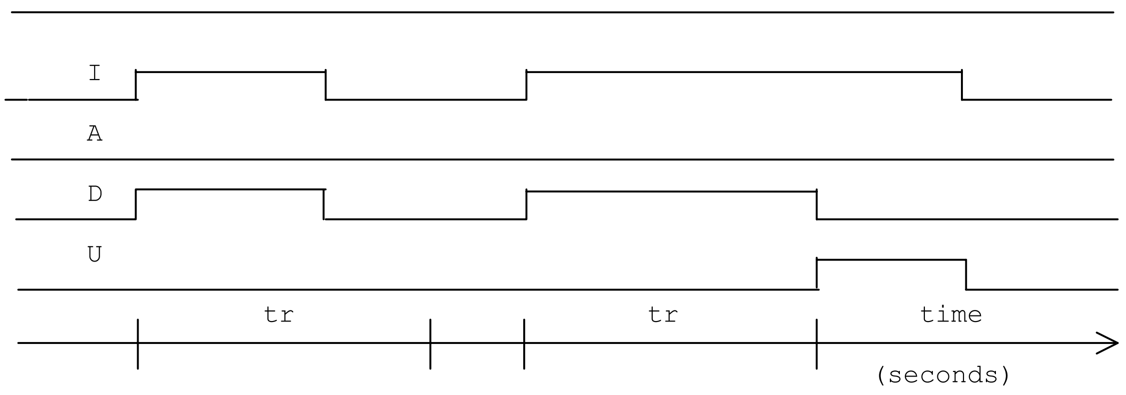
Pulse output. The pulse output, or simply pulse, is an output signal that goes to 1 on the input rising edge and stays at 1 for a period of time equal to the programmed delay time plus the amount of time the enable signal is at 1. If the input goes to 0 before the delay time has expired, the pulse goes to 0.

Delayed output. The delayed output is an output signal that goes to 1 with a delay with respect to the input rising edge, and stays at 1 as long as the input is at 1. The delay is equal to the amount of time programmed for the delay. When the input goes to 0, the delayed output goes to 0.

Example:

Figure 3.4. illustrates the evolution of the pulse and delayed outputs.

Fig. 3.4. - Timer operation



tr = delay time

The delay time count starts at each input rising edge, and is reset, together with the pulse status, when the input falls to 0.

The timer is started, and the values assigned to the output signals, the moment the input definition statement is encountered. The timing situation and the timer signals status, however, may differ. This difference depends on the length of the machine logic, its execution time, and the type of logic to which the timer statement belongs (slow or fast logic), and may be equal to a maximum of one logic cycle.

Counters

A counter features a series of operands, and has the function of an electronic pulse counter. Its main feature is the preset value that represents the number of pulses counted by the counter. This metaoperand normally has the function of generating an output according to the number of rising edges of an assigned input signal. A typical counter application is to count the phases of a tool holder turret.

A set of symbols is used to represent the various signals defining the timer (refer to Chapter 4, "List of elements"). The user may employ 32 programmable counters, numbered from 0 to 31, with preset value from 2 to 255.

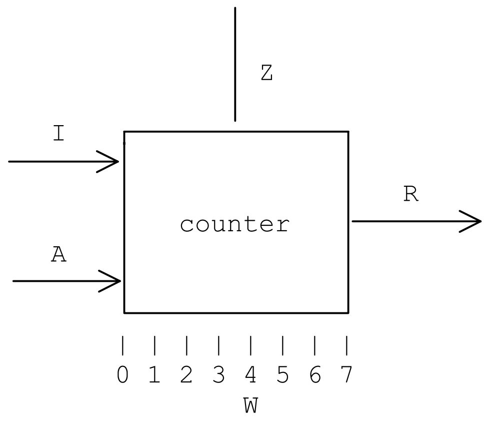
Counter operation features five signals identified with the names and symbols indicated below (see Chapter 4):

- CxxI = input;
- CxxR = carry-over;
- CxxZ = reset;
- CxxA = mode;
- CxxW = count word.

The identification symbols "xx" represent the counter number.

From the logic point of view, a counter can be represented as a device with five terminals: three input terminals, one output and one that can be either an input or an output.

Fig. 3.5. - Counter Logic Diagram



Input. The input is a user-defined signal that initializes the counter and defines the preset value. If the input is not defined, the counter outputs are not activated. The count is activated on the input rising edge and is incremented or decremented by one unit at each subsequent input rising edge (depending on the mode status).

Mode. The mode signal is user-defined and selects the count up or count down in accordance with its status:

- If A = 0 the count increases and goes from zero to the preset value;
- If A = 1 the count decreases and goes from the preset value to zero.

When the count has been completed (the preset value or zero has been reached), the counter is reset and the count is reactivated on the first input rising edge.

Count word. The count word can be used in read and write. In read its contents can be read by assigning it to a support word. In write, it is defined by the user assigning a constant value.

In read, the count word permits the current count status to be read. In this case the word contents are incremented or decremented according to the mode status:

- If A = 0 the word contents are incremented starting from 0 until the preset value is reached;
- If A = 1 the word contents are decremented starting from the preset value until zero has been reached.

The count end value (preset value, or zero) can never be read since the counter is automatically reset as soon as the count reaches this value.

In write, the count word can be defined by assignment at any time during counter operation. This means that the starting point can be changed or intermediate counts skipped. The value declared in word assignment must be less than the preset value.

Reset. It is a user-defined signal. When set to 1, it resets all the signals of the counter. In this state, the only allowable operation is the direct assignment of CxxW. It is advisable to program the reset of a counter neither inside a conditional block nor inside alternative blocks (DOF-DOE).

Carry-over. The carry-over signal is an output signal that goes to 1 in the following conditions:

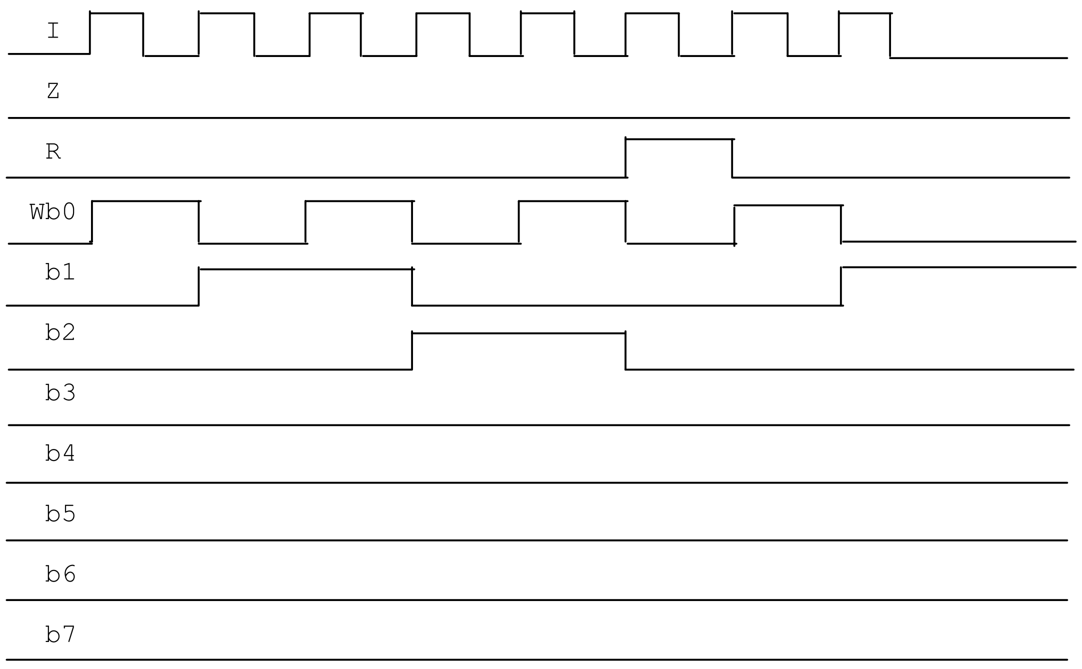
- If the count increases (A = 0), the carry-over goes to 1 when the count reaches the count end value (preset value), and remains at 1 until the first input rising edge reactivates the count;

- If the count decreases (A = 1), the carry-over signal goes to 1 when the count reaches the count end value (zero), and remains at 1 until the input rising edge reactivates the count.

Example:

Given an input that develops as illustrated, and a preset value equal to 6, Fig. 3.6. illustrates the evolution of the carry-over and the count word contents bit by bit. Bit bO corresponds to the count word W.

Fig. 3.6. - Counter operation



Refer to Chapter 4 for further details on how counters are used.

Pulse generators

The pulse generator acts as a one-shot circuit and consists of a signal-type operand. This metaoperand generates a signal on the rising edge of an assigned signal, and the generated signal remains at 1 for one logic cycle. The pulse generator can be used to keep a given signal at 1 for a complete logic cycle.

There are 32 pulse generators numbered 0 to 31, identified with the following symbols:

Pxx

where xx represents the pulse generator number.

Pulse operation features only one signal-type operand that is assigned by the user. On the first rising edge of this signal, the pulse generator generates a signal that remains at logic level 1 for one logic cycle. The status assumed by the generated signal can be read by assigning it to a support signal. Refer to Chapter 4 "List of elements" for further details.

ASCII comparators

The ASCII comparator compares an ASCII character string, sent from the keyboard by the user pressing pushbutton P2, and an ASCII character string contained in the machine logic.

If the two strings are the same, the comparator generates a signal that remains at logic level 1 for two logic cycles, and then goes to zero. If the two strings are not the same, the signal remains at zero.

The string programmed in the machine logic must be written between inverted commas as follows:

"ASCII string"

where ASCII string is an ASCII string containing a maximum of 32 characters.

The string set on the keyboard is sent by pressing pushbutton **P2**.

Four decimal digits, sent from the keyboard after the ASCII string, can be stored in rack K, in BCD format. The digits must be separated from the ASCII string by pressing the CR key. Storing will be performed on the following words:

- Least significant digits: W07K0;

- Most significant digits: W07K1.

The decimal digits are stored on these words until the next ASCII string is sent.

Example:

If the value 1234 is sent from the keyboard, it is stored as follows:

- The digits 3 and 4 are stored in the word W07K0 in the format 0011 0100 (two BCD digits).

- The digits 1 and 2 are stored in the word W07K1 in the format 0001 0010 (two BCD digits).

Refer to Chapter 4 "List of elements" for further details.

3.2.3. FUNCTIONS

The functions represent a class of elements that have operands and mathematical expressions for argument and give a single operand as the result. Functions normally have one input fora given output. To read or use this output, it must be assigned by the machine logic to a support operand that contains the output value. The functions in the language are normally used in assignment statements with the following format:

operand = function(arg)

where "arg" is the function argument that can be an operand, a set of operands or a mathematical expression; "function(arg)" represents the output operand generated by the function; "operand" represents the support operand assigned by the user from the machine logic to read the output operand contents.

Function Categories

The functions can be divided into three categories according to the operations they perform:

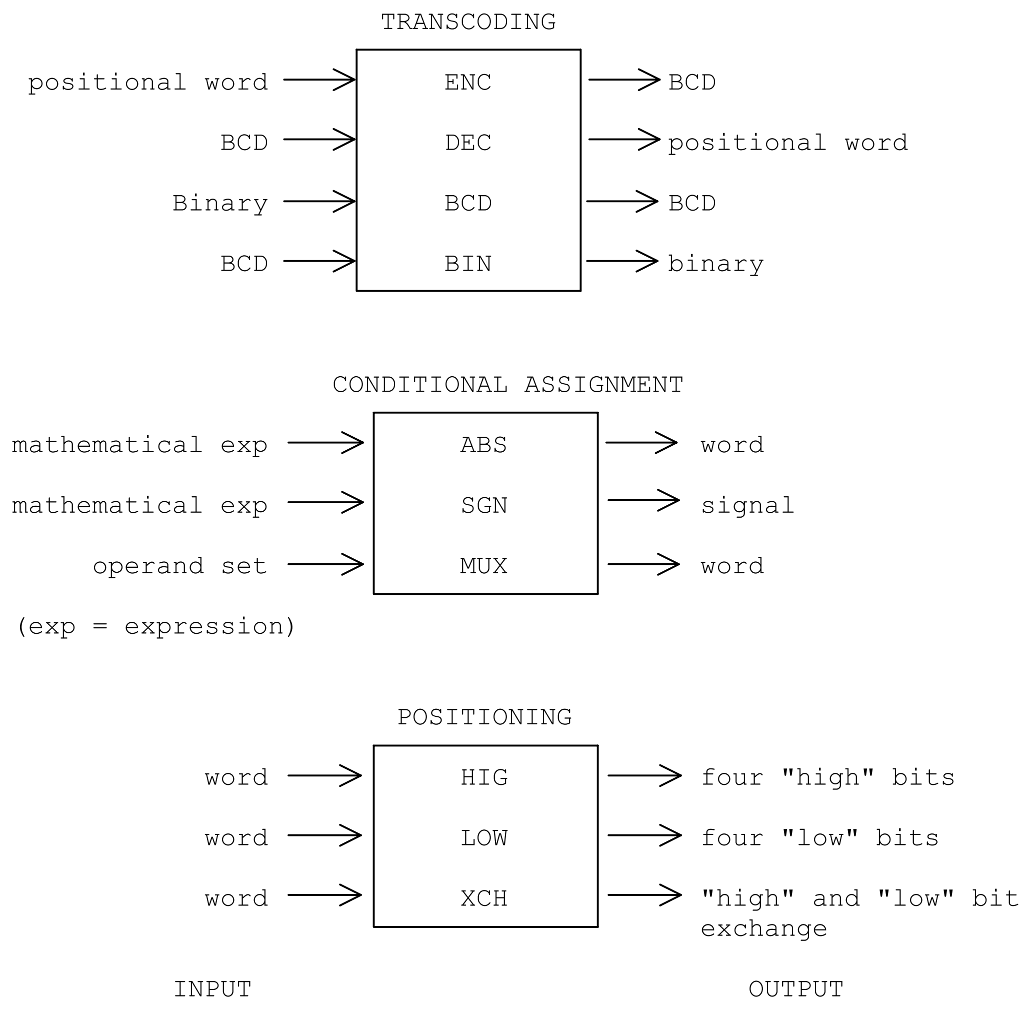
- Transcoding functions: ENC, DEC, BCD, BIN. These functions change the code of an input word specified by the user.

- Conditional assignment functions: ABS, SGN, MUX. These functions give an operand whose value depends on a certain condition taking place or on the result of a mathematical expression.

- Positioning functions: HIG, LOW, XCH. These functions operate on the position occupied by the individual bits of an input word specified by the user.

The diagram below illustrates the operational features of the various functions available, in the three categories described previously.

Fig. 3.7. - Function Categories



Function Characteristics

The user can utilize a set of functions that constitute a support tool used in writing the machine logic. The table below gives a summary of the functions available. Refer to Chapter 4 "List of elements" for further details.

FUNCTION	**OPERATION PERFORMED**
ENC	Converts a positional value into a BCD value between 0 and 8.
DEC	Converts a BCD value between 0 and 8 into a positional value.
BCD	Converts the contents of a word from binary format to BCD format.
BIN	Converts the contents of a word from BCD format to binary format.
ABS	Gives a word with a value equal to the modulus of a mathematical expression.
SGN	Gives a signal whose status depends on the sign of a mathematical expression.
MUX	Gives a word whose value is selected from a series of words according to the status of a corresponding signal.
HIG	Gives the four most significant bits in a word.
LOW	Gives the four least significant bits in a word.
XCH	Exchanges the four least significant bits with the four most significant bits in a word.

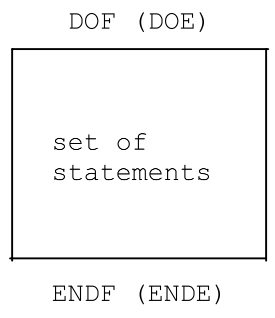
3.2.4. BLOCKS

PLC programming allows the execution of a block of statements according to a particular condition. This feature can be obtained by using the block statements DOF, ENDF, DOE, ENDE.

Unlike all the other program statements, the block statements do not contain operands and metaoperands linked by operators (assignments, comparisons etc.), and they stand alone on a program line. These special structures are normally used to write sophisticated programs to reduce the machine logic execution times.

A block consists of a set of consecutive statements that begins with a DO type statement (DOF, DOE) and ends with an END type statement (ENDF, ENDE). The statements that make up the block are executed when a particular expression is true. The block is ignored if the expression is false. Fig. 3.8. illustrates the structure of a block within a program.

Fig. 3.8. - Block structure



A block cannot be contained in both the fast logic and the slow logic. For example, a block that starts in the fast logic cannot end in the slow logic, but the whole block must be contained in the fast logic.

There are two types of blocks with their corresponding block statements:

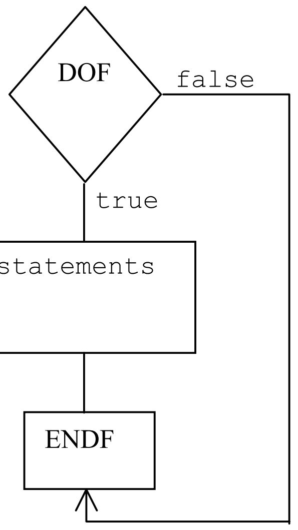
- Conditional blocks.

- Alternative blocks.

Conditional Blocks

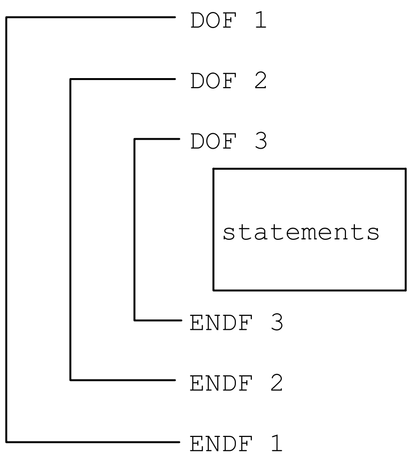
A conditional block is defined by using statements DOF and ENDF, and it is executed only if a particular expression associated to the DOF statement is true, otherwise it is ignored. Conditional block operation can be represented by a flow diagram as illustrated in Fig. 3.9.

Fig. 3.9. - Conditional Block Operation



The conditional blocks can be nested one inside the other. The nesting level represents the number of blocks nested. When writing nested structures, it is particularly important not to leave open blocks (no ENDF) and to correctly nest the blocks. Fig. 3.10. illustrates how to nest a series of conditional blocks; the different blocks are identified by the number placed after the block statements.

Fig. 3.10. - Nested Blocks



It should be noted that if the blocks are to be nested correctly, the lines joining the DOF statements with the corresponding ENDF statements must not intersect each other. It should also be remembered that in this particular structure, the innermost statements will be executed only if the expressions of all the other blocks are true.

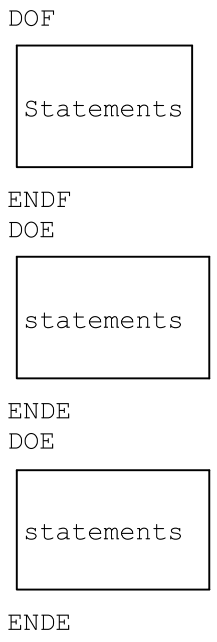
Alternative Blocks

An alternative block is defined by using the statements DOE and ENDE, and is only executed if a particular expression associated to the DOE statement is true and the expression associated with the DOE (or DOF) statement of a block that precedes it is false, otherwise it is ignored.

An alternative block cannot be used by itself, but only if followed by another conditional or alternative block. An alternative block is processed, in fact, as an alternative to a block that precedes it if this block is not executed.

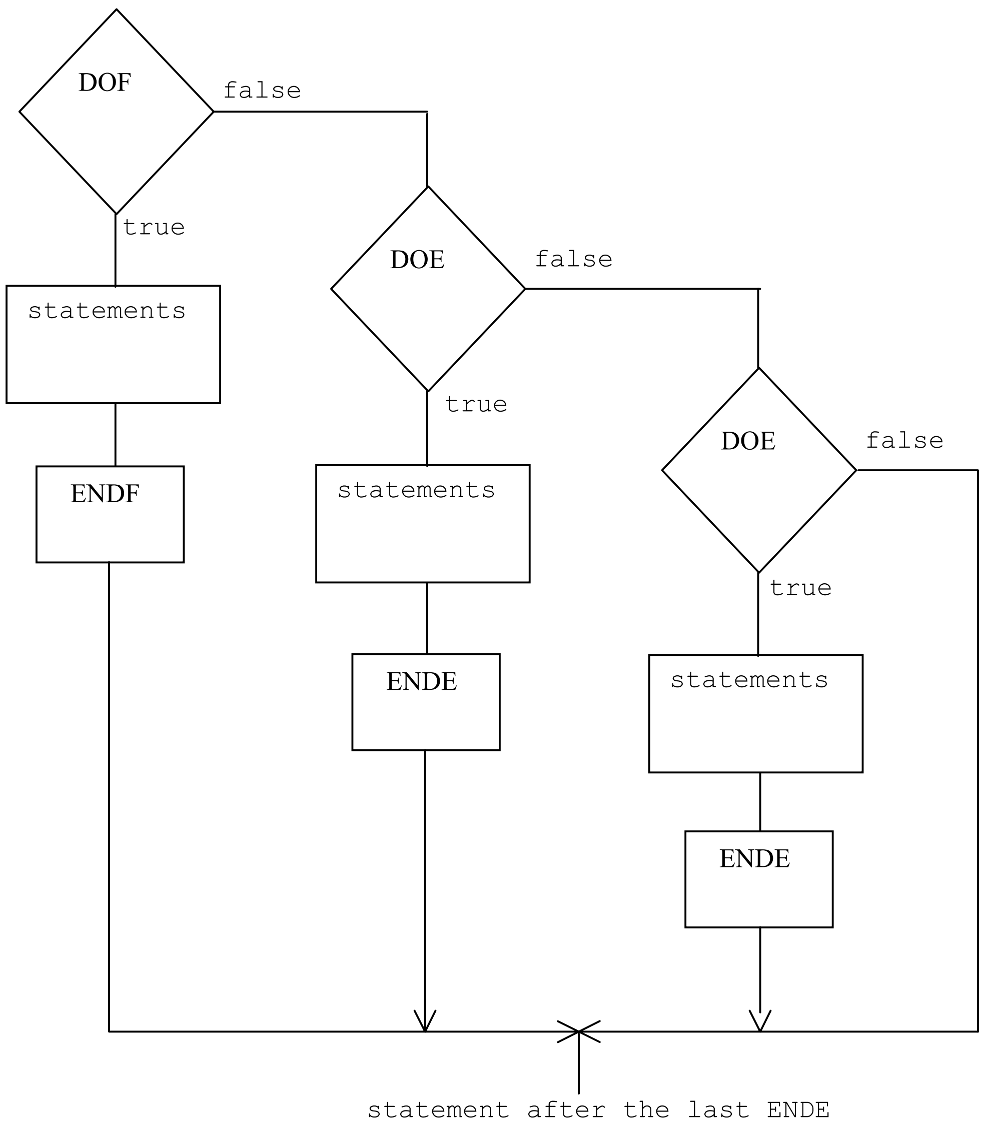
Several alternative blocks can be linked together in series. The diagram below illustrates how to introduce a set of alternative blocks in sequence inside a program.

Fig. 3.11. - Alternative Blocks in Sequence



The operation of an alternative block structure can be represented by a flow diagram, as illustrated in Fig. 3.12.

Fig. 3.12. - Block Structure Operation



A complete set of alternative blocks arranged in sequence will never be executed, even if the corresponding expressions are true - Only the first alternative block with true expression encountered during processing will be executed. Chapter 4, "List of elements" describes how to use the blocks.

3.2.5. OPERATORS

The PLC language has a set of operators that are used to link the operands. These operators can be divided into the following three groups:

- Mathematical operators;

- Comparison operators;

- Logical operators.

Mathematical operators

The following mathematical operators are used to perform mathematical operations between operands:
- Sum;
- Difference.

The table given below lists the mathematical operators with a description of the operation performed and the syntax representing the operations.

Table 3.3. - Mathematical operators

OPERATOR	**OPERATION**	**SYNTAX**
Sum	Addition of two operands in two's complement.	[ope1+ope2]
Difference	Subtraction of two operands in two's complement.	[ope1-ope2]

Mathematical Operations

Mathematical operations are performed between words in the binary system in two's complement module eight. In the above table, ope1 and ope2 represent the added or subtracted operands and may be of the following type:
- Word;
- ABS[op-mat];
- DEC(word);
- ENC(word);
- BIN(word).

Mathematical operations must always be enclosed inside square brackets.

The result of mathematical expressions is a word, and they can therefore be used in a Boolean expression to write a machine logic statement.

Examples:

The following example gives the sum of two words.

```
W100K0=[W100K1+W100K2]
```

The following example gives the difference between the decoder of a word and the encoder of a word.

```
W100K0=DEC(W110K0)
W100K1=ENC(W110K1)
W100K2=[W100K0-W100K1]
```

Comparison Operators

The logical comparison operators are used to test word operands. The outcome of the test can be true or false, and can be made to correspond with a logic level signal at 1 or 0 respectively. The table given below lists the types of comparison carried out by the various operators and the syntax representing the operations.

Table 3.4. - Comparison Operators

OPERATOR	COMPARISON	SYNTAX
=	Equality	[word1=word2]
>	Greater than	[word1>word2]
<	Less than	[word1<word2]

The words compared (word1 and word2) can be represented either with symbols (Wxxyz) or with mnemonics. One of the two words can also represent a constant value of the nnnf type. In this case the operations have the following format:

```
[word=nnnf]

[word>nnnf]

[word<nnnf]
```

Comparison Operations

The table given below summarizes the possible results of the various comparison operations defined previously.

Table 3.5. - Comparison Operations

OPERATION	RESULTS
[word1=word2]	True, if word1 and word2 are the same. False, if word1 and word2 are different.
[word1>word2]	True, if word1 is greater than word2. False, if word1 is less than or equal to word2.
[word1<word2]	True, if word1 is less than word2. False, if word1 is greater than or equal to word2.

The comparison operations must always be enclosed inside square brackets.

Comparison operations give a result that to all intents and purposes is a signal, and they can therefore be used in a Boolean expression to write a machine logic statement.

Example 1:

Given the following expression:

U100K26=[W100K1=16D]

If the word W100K1 is equal to the decimal value 16, then U100K26=1. If it is not, U100K26=0.

Example 2:

Given the following expression:

U100K12=I12A18*/ [W15K1=W13T3]

If I12A18 = 1 and the words W15K1 and W13T3 are different, then U100K12=1.

Logical Operators

The logical operators are used to perform logical operations with the signal/word type operands. The logical operations are performed in accordance with rules laid down by Boolean algebra. The result of a logical operation between two signals is a signal, and the result of a logical operation between two words is a word. The table given below lists the logical operators with a description of the operation executed and the syntax representing the operations.

Table 3.6. - Logical Operators

OPERATOR	OPERATION	SYNTAX
NOT	Executes the logical negation of a signal.	/sign1
AND	Executes the logical product of two signals or two words.	sign1*sign2 word1*word2
OR	Executes the logical sum of two signals or two words.	sign1+sign2 word1+word2
XOR	Executes the exclusive OR operation of two signals.	sign1&sign2

In the above table, sign1, sign2, word1, word2 represent the signals and the words set in relation to each other. It should be noted that a signal cannot be related to a word.

Logical Operations

The above-mentioned logical operations are executed in accordance with the hierarchical priorities binding the operators. The NOT operator has priority over a11 the other operators; the AND operator has priority over OR or XOR; OR and XOR have the same priority.

In operations between two words, the operation is carried out in two's complement and module eight. The result of logical operations between signals is clearly shown in the following truth table that refers to the above-mentioned logical operators.

Fig. 3.13. - Logical Operators Truth Table

NOT	SIGN	/SIGN
	0	1
	1	0

AND	SIGN1	SIGN2	SIGN1*SIGN2
	0	0	0
	0	1	0
	1	0	0
	1	1	1

OR	SIGN1	SIGN2	SIGN1+SIGN2
	0	0	0
	0	1	1
	1	0	1
	1	1	1

XOR	SIGN1	SIGN2	SIGN1&SIGN2
	0	0	0
	0	1	1
	1	0	1
	1	1	0

The result of logical operations between signals is a signal, and therefore they can be used in a Boolean expression to write a machine logic statement.

Example 1:

Given the following equation:

```
U100K1=U100K2*U100K3
if U100K2 = 0 and U100K3 = 1 then U100K1 = 0
if U100K2 = 1 and U100K3 = 1 then U100K1 = 1
```

Example 2:

Given the following equation:

```
U100K1=U100K2+/U100K3
if U100K2 = 0 and U100K3 = 1 then U100K1 = 0
if U100K2 = 0 and U100K3 = 0 then U100K1 = 1
```

Use of Brackets

Since the logical operators have different priorities, the outcome of a sequence of logical operations may have a different meaning depending on the position of the brackets.

For example, given the following operation:

U20A5=I19A1+I19A2*I02K6

This expression corresponds to the electricity network indicated in Fig. 3.14(a).

If the brackets are positioned as follows:

U20A5=(I19A1+I19A2)*U02K6

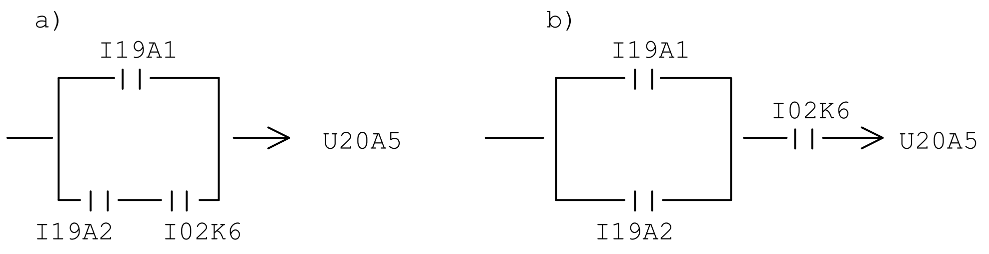
The equivalent network is the one indicated in Fig. 3.14(b).

If the brackets are positioned as follows:

U20A5=I19A1+(I19A2*U02K6)

The equivalent network is exactly the same as the network that corresponds to the expression without brackets (Fig. 3.14(a)). In this case, the introduction of brackets does not after the meaning of the initial expression and the brackets can therefore be omitted.

Fig. 3.14. - Equivalent Electricity Networks



4. LANGUAGE ELEMENTS

4.1. INTRODUCTION

This chapter gives an operational description of the individual PLC language elements to help the user to employ it correctly when writing a program. The first part of the chapter gives a summary of the elements for quick reference. The second part describes each element in detail and includes all the information concerning the element's function and how it can be used by the user.

The table below lists all the elements described in this chapter in alphabetical order. A short functional description is given for each element and its functional class is indicated.

ELEMENT	**FUNCTION**	**CLASS**
COUNTER	Acts as an electronic pulse counter.	Metaoperand
CONVERSION INTO BCD	Converts a word into BCD format.	Function
CONVERSION INTO BINARY	Converts a word into binary format.	Function
DECODER	Gives the positional value of a word.	Function
PULSE GENERATOR	Acts as a one-shot circuit.	Metaoperand
DOE ENDE	Permit the execution of a block of statements.	Block

ELEMENT	**FUNCTION**	**CLASS**
DOF ENDF	Permit the execution of a block of statements.	Block
ENCODER	Gives the BCD value of a positional value.	Function
HALF WORDS	They exchange positions on the bits of a word.	Functions
MODULUS	Gives a word according to the modulus of a mathematical expression.	Function
MULTIPLEXER	Gives a word according to the status of a set of assigned signals.	Function
ASCII COMPARATOR	Compares a string sent from the console with a string programmed in machine logic.	Metaoperand
SIGNAL	Represents a logical or physical digital signal.	Operand
SIGN	Gives a signal according to the sign of a mathematical expression.	Function
TIMER	Acts as a delay relay.	Metaoperand
WORD	Represents eight signals.	Operand

4.2. LIST OF ELEMENTS

This part of the chapter describes the individual language elements. These elements are listed in alphabetical order and the following information is given for each one of them:

- The element function.

- The element syntax and usage, which are expressed using the following conventions:

. All the characters belonging to the element's syntax are written in bold type. These characters must be keyed in when using the element;

. The key words are written in bold type and capital letters and must be written exactly as illustrated. It should be noted that square or round brackets may appear. In this case, the brackets are a syntactical element and as such have to be keyed in;

. The symbolic names of the parameters (syntax elements) are written in bold type in small letters and must be substituted by appropriate values indicated separately.

- The description of the parameters written with the following conventions:

. The optional parameters are indicated in square brackets (not in bold type) and can be used at the user's discretion. In this case the square brackets are not a syntactical element and may therefore be omitted;

. The alternative parameters are enclosed inside brace brackets (not bold) and are separated by a vertical line. In this case the brace brackets are not a syntactical element and may therefore be omitted;

. The vertical line always separates alternative parameters.

- The element's functional characteristics and any remarks.

- Application examples.

Some elements have more than one syntax. In these cases, the different syntaxes are indicated by the words "syntax 1, syntax 2 etc" and are considered alternatives.

4.2.1. COUNTER Metaoperand

Acts as an electronic pulse counter.

Input
CxxI (count) =sign-bool-expr

Mode
AxxA=sign-bool-expr

Reset
CxxZ=sign-bool-expr

Count word (in read)
out-word=CxxW

Count word (in write)
CxxW=value

Carry-over
sign=CxxR

Where

SYNTAX ELEMENT	**MEANING**
xx	Counter number: 0 to 31.
count	Counter preset value: from 2 to 255. Can be a constant expressed in the nnnf format, or a word expressed with symbols or mnemonics.
sign-bool-expr	Signal Boolean expression.
out-word	Support word assigned by the user that contains the current count value.
value	Value assigned to the count word by the user. Can be expressed with a word Boolean expression or with a MUX function.
sign	Signal assigned by the user used as support signal to read the carry-over status (CxxR).

Characteristics

If the count parameter is a constant, the preset value is a constant and can only be modified by reprogramming the CxxI signal. If, however, the count parameter is a word, the preset value is a variable in relation to the current value of the word specified.

If the count parameter is a word, it must be remembered that the system analyzes the preset values on a 16 bit basis (the word containing the preset value plus the next word), whereas the PLC language only programs 8 bits (a word). For this reason, the word that follows a word containing a preset value must not be used and must be set to zero.

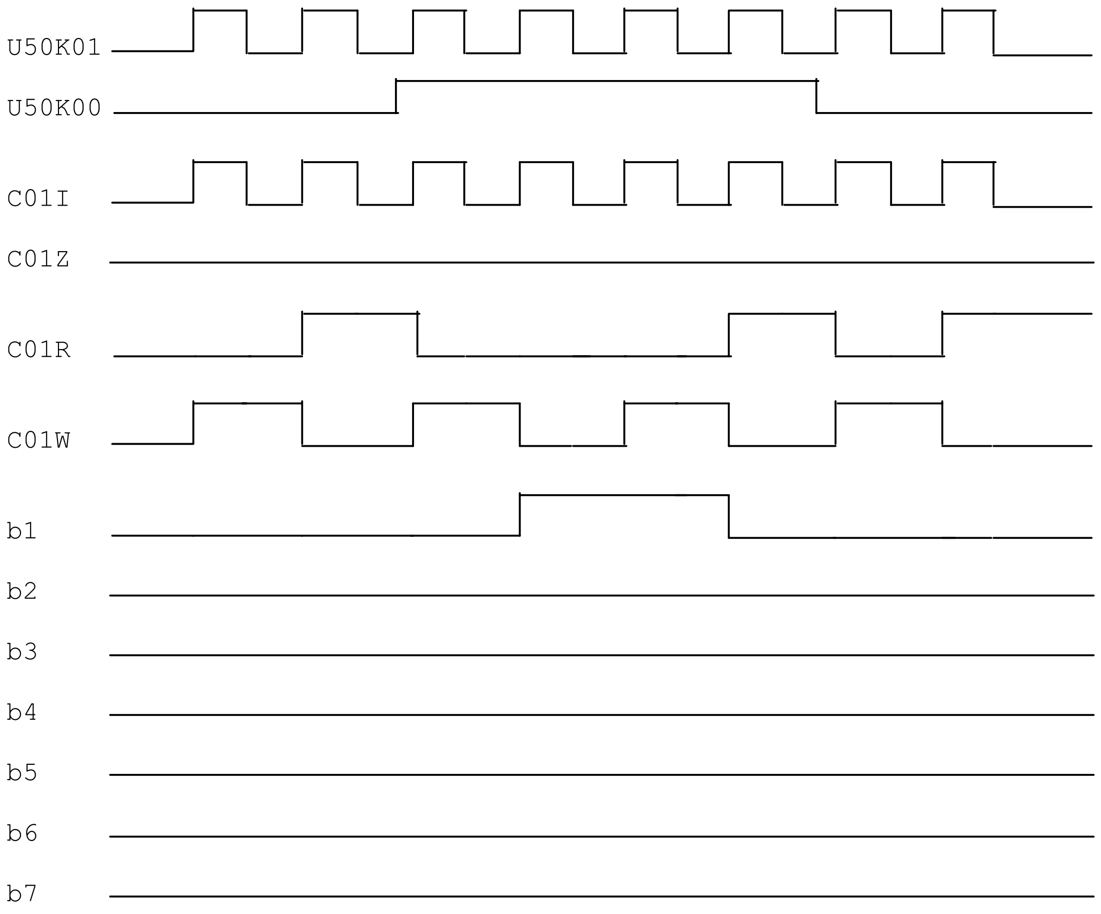
If the mode (CxxA) is not prograrnmed, the system assumes an up count mode by default.

Example

The following example shows how a counter with variable preset value is programmed using a conditional block.

```
W100K00=2
DOF:U50K00
W100K00=4
ENDF
C01I (W100K00) =U50K01
W100K01=0
```

The diagram below illustrates the status of the signals relating to counter C01 and the individual bits of the count word (C01W):



In this example, the word W100K00 is used to define the counter preset value, and consequently the word W100K01 following this word is set to zero. It should be noted that bit bO corresponds to the count word C01W.

4.2.2. CONVERSION INTO BCD Function

Converts the binary value of a word or constant into the corresponding BCD value.

Syntax 1

```
out-word=BCD(in-word)
```

Syntax 2

```
out-word=BCD(const)
```

Where

SYNTAX ELEMENT	MEANING
out-word	Word assigned by the user that contains the word given by the function.
in-word	Input word to be converted supplied by the user.
Const	Constant numerical value to be converted supplied by the user in nnnf format.

Example

If the user programs:	W100K0=26D
	W100K1=BCD(W100K0)

then:	W100K0 = 0001 1010 (binary)
	W100K1 = 0010 0110 (BCD)

4.2.3. CONVERSION INTO BINARY Function

Converts the BCD value of a word or constant into the corresponding binary value.

Syntax 1

```
out-word=BIN(in-word)
```

Syntax 2

```
out-word=BIN(const)
```

Where

SYNTAX ELEMENT	MEANING
out-word	Word assigned by the user that contains the word given by the function.
in-word	Input word to be converted supplied by the user.
Const	Constant numerical value to be converted supplied by the user in nnnf format.

Example

If the user programs:

```
W50K0=26D
W50K1=BIN(W50K0)
```

then:

```
W50K0 = 0010 0110 (BCD)

W50K1 = 0001 1010 (binary)
```

4.2.4. DECODER Function

Gives the positional value contained in the word or constant specified.

Syntax 1

```
out-word=DEC(in-word)
```

Syntax 2

```
out-word=DEC(const)
```

Where

SYNTAX ELEMENT	MEANING
out-word	Word assigned by the user containing the word given by the function.
in-word	Input word supplied by the user containing a positional value.
Const	Constant numerical value supplied by the user in format nnnf containing a positional value.

Remarks

The input word must be supplied in the binary format which is obtained by applying the ENC function.

Characteristics

The positional value contained in a word or in a constant represents the position of the most significant bit at 1.

The function output is represented by a word having only one bit at 1. The position of this bit represents the positional value contained in the input constant or word; the bit furthest right is considered bit 1 in the output word. The diagram given below shows all the possible ways of decoding a positional value; the positional value and the output given by the function (word) are indicated for each function input (in binary).

INPUT WORD	POSITIONAL VALUE	OUTPUT WORD
		8|7|6|5|4|3|2|1
00000000	-	0 0 0 0 0 0 0 0
00000001	1	0 0 0 0 0 0 0 1
00000010	2	0 0 0 0 0 0 1 0
00000011	3	0 0 0 0 0 1 0 0
00000100	4	0 0 0 0 1 0 0 0
00000101	5	0 0 0 1 0 0 0 0
00000110	6	0 0 1 0 0 0 0 0
00000111	7	0 1 0 0 0 0 0 0
00001000	8	1 0 0 0 0 0 0 0

Any other binary input value, gives an output value equal to zero. The maximum input value is the binary value 00001000, i.e. decimal eight.

Example

If the binary value 00011001 is input, the output of the function is zero.

4.2.5. PULSE GENERATOR metaoperand

Generates a signal lasting one logic cycle on the rising edge of a specified signal.

Pulse

```
Pxx=sign-bool-expr
```

Read signal

```
sign=Pxx
```

Where

SYNTAX ELEMENT	MEANING
xx	Number identifying the pulse generator: from 8 to 31.
sign-bool-expr	Signal Boolean expression. A signal lasting one logic cycle is generated on the rising edge of this signal.
sign	Signal assigned by the user used as support signal to read the status of the pulse (Pxx).

Characteristics

The signal generated by the pulse generator remains at 1 for one logic cycle regardless of how the signal assigned by means of the parameter sign-bool-expr evolves after the rising edge.

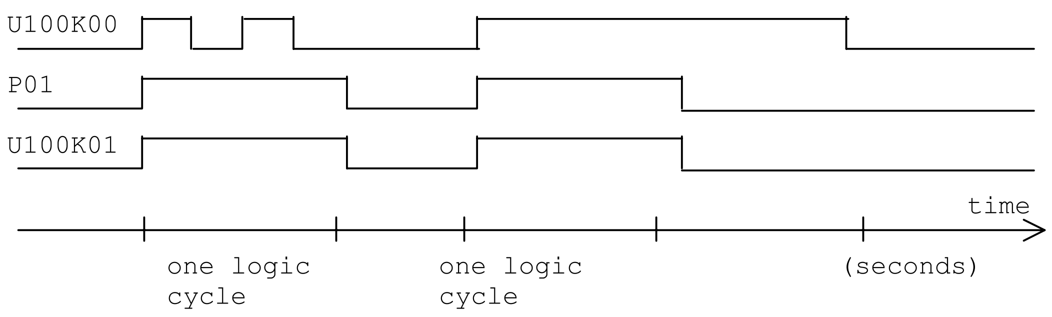
Example

The following example shows how to program a pulse generator and illustrates the status of the signals associated to it.

If the user programs:

```
P01=U100K00
U100K01=P01
```

The status of the signals relating to pulse generator P01 is as follows:



Important. If the pulse (i.e. P01) is written in the low logic, the interval during which it is in SET is equivalent to the duration of one turn of the entire low logic.

If the pulse is written in the fast logic, the interval during which it is in SET is equivalent to the duration of one or two turns of the low logic.

4.2.6. DOE...ENDE Statements

These statements permit a block of statements to be executed in alternative to a previous block of statements only when the expression specified is true.

```
DOE[block-name]expr
.
.
.
statements
.
.
.
ENDE[block-name]
```

Where

SYNTAX ELEMENT	MEANING
block-name	Alphanumerical string that identifies the block of statements. This string can have up to 128 characters, but only the first six characters are significant.
expr	Expression containing operands, metaoperands, functions and constants linked by operators, that may be true or false.

Characteristics

The block statements DOE and ENDE define an alternative type of block (See Chapter 3, "Blocks"). This type of block can be entered in a program only after a conditional block or another alternative block. A DOE statement can consequently only be used after ENDF and ENDE statements.

The DOE statement is only considered if the previous block expression is false. In this case, if the expression associated to DOE is true, the block statements between DOE and ENDE are executed. A jump is then made to the statement following the ENDE statement of the last block in the series. If the statement associated to the DOE is false, the block statements are ignored and a jump is made to the statement irnmediately after the ENDE statement.

The expression can be omitted in the DOE statement of the last alternative block in series. In this case the statement is called unconditional DOE and is equivalent to a DOE whose expression is always true. The last alternative block in a series defined by an unconditional DOE is always executed if the expressions in all the blocks that precede it are false.

If the block is identified with a name by means of the block-name parameter, the name specified in the ENDE statement must be exactly the same as the one specified in the DOE statement. Otherwise an error is indicated in the compilation phase.

Only comments or blank lines are allowed between the DOE statement of a block and the ENDE or ENDF statement of the previous block.

Example

In the example given below, the expression associated to the various defined blocks consists of a logical comparison. Comments are inserted between blocks.

```
DOF BLOCK1:[W26K0=2]
U00K2=[W26K0=3]
U00K3=[W26K0=4]
ENDF BLOCK1

;IF AN OPERATIONAL STATEMENT IS ENTERED IN THE  PLACE  OF
;THIS COMMENT THE COMPILER INDICATES AN ERROR

DOE BLOCK2:[W26K0=5]
U00K2=[W26K0=6]
U00K3=[W26K0=7]
ENDE BLOCK2

;THE NEXT BLOCK IS NOT IDENTIFIED BY A NAME

DOE:[W26K0=8]
U00K2=[W26K0=9]
U00K3=[W26K0=10]
ENDE

;THE NEXT BLOCK IS DEFINED BY AN UNCONDITIONAL DOE

DOE BLOCK4:
U00K2=[W26K0=11]
U00K3=[W26K0=12]
ENDE BLOCK4
```

If W26K0 = 2 (result of "true" logical comparison) the statements of the first block are executed and a jump is then made to the statement following the ENDE statement relating to the last block. If W26K0 is not equal to two (result of "false" logical comparison) the first alternative block with true expression is executed, the other alternative blocks are ignored and a jump is made to the statement following the ENDE statement of the last block. If, however, W26K0 is not 2, 5, or 8, the last block of the unconditional DOE, that is equivalent to a block with expression that is always true, is executed.

4.2.7. DOF...ENDF Statements

These statements permit a block of statements to be executed only when the expression specified is true.

```
DOF[block-name]:expr
.
.
.
statements
.
.
.
ENDF[block-name]
```

Where

SYNTAX ELEMENT	MEANING
block-name	Alphanumerical string that identifies the block of statements. This string can have up to 128 characters, but only the first six characters are significant.
expr	Expression containing operands metaoperands, functions and constants linked by operators, that may be true or false.

Characteristics

The block statements DOF and ENDF define a conditional type of block (see Chapter 3, "Blocks"). If the expression is true, the block statements between DOF and ENDF and the statement following the ENDF statement are executed. If the expression is false, the block statements are ignored and a jump is made to the statement immediately following the ENDF statement.

If the block is identified with a name by means of the block-name parameter, the name specified in the ENDF statement must be exactly the same as the one specified in the DOF statement. Otherwise an error is indicated in the compilation phase. Maximum block nesting level is 10.

Examples

In the example given below, the expression related to the block is a signal, which is the simplest type of Boolean expression that can be defined. The signal at logic level 1 corresponds to a true expression.

```
DOF:U10K0
U10K3=U10K2
U10K4=U10K3*/UOOTO
ENDF
W13K3=MUX(W13A0),(I19A1)
```

If the signal U10K0 is at logic level 1 (equivalent to a true expression), the conditional block statements are executed, if it is not, a jump is made to the statement following the ENDF statement of the block.

The example given below shows how the nested block structures are to be used.

Incorrect block nesting:

```
DOF BLOCK1:[W26K0=2]
U00K2=[W26K0=3]
U00K3=[W26K0=4]
DOF INCORRECT NESTING:I27K3
U00K4=[W26K0=5]
ENDF BLOCK1
ENDF NESTS
```

In this example the block "BLOCK1" is incorrectly closed before the block called "INCORRECT NESTING". Correct block nesting is performed as follows:

```
DOF BLOCK1:[W26K0=2]
U00K2=[W26K0=3]
U00K3=[W26K0=4]
DOF CORRECT NESTING:I27K3
U00K4=[W26K0=5]
ENDF NESTS
ENDF BLOCK1
```

In the previous two examples, the block name specified in the ENDF statement is not exactly the same as the one specified in the DOF statement. This does not generate an error, however, because the first six characters of the declared names are the same, and any characters after the sixth character are ignored by the compiler.

4.2.8. ENCODER Function

Provides the binary value corresponding to the positional value contained in the specified constant or word.

Syntax 1

```
out-word=ENC(in-word)
```

Syntax 2

```
out-word=ENC(const)
```

Where

SYNTAX ELEMENT	MEANING
out-word	Word assigned by the user containing the word given by the function.
in-word	Input word supplied by the user containing a positional value.
const	Constant numerical value supplied by the user in the nnf format containing a positional value.

Characteristics

The binary number output from the function is considered the binary value of the word and represents the word in all respects. The ENC function determines the positional value of the most significant bit (signal) of the word at 1 (considering the bit furthest right as bit 1) and puts it in the binary code. If the contents of the word is zero, the function output is also zero. The diagram below indicates all the possible conversions of a word into the relevant binary value: the positional value of the most significant bit and the output given by the function (binary) are given for each function input (word).

INPUT WORD 8|7|6|5|4|3|2|1	POSITIONAL VALUE	OUTPUT WORD
0 0 0 0 0 0 0 0	-	00000000
0 0 0 0 0 0 0 1	1	00000001
0 0 0 0 0 0 1 -	2	00000010
0 0 0 0 0 1 - -	3	00000011
0 0 0 0 1 - - -	4	00000100
0 0 0 1 - - - -	5	00000101
0 0 1 - - - - -	6	00000110
0 1 - - - - - -	7	00000111
1 - - - - - - -	8	00001000

The dashes represent insignificant signals.

Example

This example indicates the binary values output by the function, corresponding to certain input words.

```
INPUT WORD                                  OUTPUT WORD

01010010 ---->  (ENC function) ----> 00000111
00100000 ---->  (ENC function) ----> 00000110
00111111 ---->  (ENC function) ----> 00000110
```

4.2.9. HALF WORDS Functions

Perform positional exchanges on the bits of the specified word.

LOW

out-word=LOW(in-word)

HIG

out-word=HIG(in-word)

XCH

out-word=XCH(in-word)

Where

SYNTAX ELEMENT	MEANING
out-word	Output word assigned by the user containing the word given by the function.
in-word	Input word supplied by the user.

Characteristics

The LOW function supplies an output word containing the four least significant bits of the input word. The bits are aligned from the right of the output word.

Example

The following example shows how the function transforms certain input words.

```
INPUT WORD                          OUTPUT WORD

0101 1111 ----> (LOW function) ----> 0000 1111
1111 0000 ----> (LOW function) ----> 0000 0000
1110 1010 ----> (LOW function) ----> 0000 1010
```

The four most significant bits of the output word are always at logic level 0 and the maximum value that this word can assume is 00001111.

The HIG function supplies an output word containing the four most significant bits of the input word. The bits are aligned from the right of the output word.

Example

The following example shows how the function transforms certain input words.

```
INPUT WORD                          OUTPUT WORD

0101 1111 ----> (HIG function) ----> 0000 0101
1111 0000 ----> (HIG function) ----> 0000 1111
0000 1111 ----> (HIG function) ----> 0000 0000
```

The four most significant bits of the output word are always at logic level 0 and the maximum value the word can assume is 00001111.

The XCH function exchanges the four least significant bits with the four most significant bits in the input word. The bits are aligned from the right of the output word. The following diagram shows how the positions occupied by the input word bits are exchanged.

```
                 INPUT WORD               OUTPUT WORD

BIT        8|7|6|5|4|3|2|1            4|3|2|1|8|7|6|5
```

Example

The following example shows how the function transforms certain input words.

```
INPUT WORD                              OUTPUT WORD

1111 0000 ----> (XCH function) ----> 0000 1111
0000 1111 ----> (XCH function) ----> 1111 0000
1100 1010 ----> (XCH function) ----> 1010 1100
```

To simplify reading, the four "low" bits are separated from the four "high" bits in the input and output words in all the examples given.

4.2.10. MODULUS Function

Supplies a word with a value equal to the modulus of the mathematical expression specified.

```
out-word=[ABS(word-mat-exp)]
```

Where

SYNTAX ELEMENT	MEANING
out-word	Output word assigned by the user containing the word given by the function.
word-mat-expr	Word mathematical expression

Remarks

Square brackets are to be used in this function, as for a mathematical expression.

Example

If the user programs:

```
W100K2=160D
W100K3=200D
W100K1=[ABS(W100K2-W100K3)]
```

then W100K1 = 40 decimal.

4.2.11. MULTIPLEXER Function

Outputs a word with a value that has been selected from the values of a set of specified words. A value is selected according to the logic level of a set of specified signals.

```
out-word=MUX(word1[,word2].....[,wordn], [sign1[,sign2].....
[,signn])
```

Where

SYNTAX ELEMENT	MEANING
out-word	Output word assigned by the user containing the value given by the function.
word1,..., wordn	Set of words from which the value assigned to the output word is selected.
sign1,...,.signn	Set of signals whose logic level selects a given value from those specified.

Remarks

The number of specified words must be equal to the number of specified signals. Positional correspondence exists between the words and signals.

Characteristics

The output word assumes the value of word1 if the signal sign1 is at logic level 1. It assumes the value of word2 if sign2 is at logic level 1, and so on. In short, the output word assumes the value of the word that corresponds to the first signal at 1 in the set specified.

Example 1

```
W10K1=MUX(W19A3),(I27K05)
```

If signal I27K05 is at logic level 1, the word W10K1 assumes the value of the word W19A3. If I27K05 is at logic level 0, the value of W10K1 remains unaltered.

Example 2

If the user programs:

```
U100K4 = [W100K1=W100K2]
W10K1=MUX(W19A0,W19A1,16H) , (U100K3,U100K4,U100K5)
```

and if words W100K1 and W100K2 are equal and U100K3 = 0, then W10K1 = W19A1.

4.2.12. ASCII COMPARATOR   Metaoperand

Compares a string sent from the keyboard with a string contained in the machine logic.

Syntax 1

```
out-sign="ASCII string"
```

Syntax 2

```
out-sign=sign-bool-expr{/|+|*|&}"ASCII string"
```

Where

SYNTAX ELEMENT	**MEANING**
out-sign	Output signal assigned by the user containing the status of the signal generated by the comparator.
ASCII string	Alphanumerical string containing a maximum of 32 ASCII characters.
sign-bool-expr	Signal Boolean expression. This expression is in its turn connected to the ASCII string by one of the logical operators indicated in syntax 2.

Characteristics

If the string sent from the keyboard is the same as the string specified in the ASCII string, the comparator generates a signal that remains at 1 for two logic cycles, otherwise the signal is not generated.

Example 1

If the user programs:

U100K01="PIPPO"

and sends the string "PIPPO" from the keyboard by pressing pushbutton **P2**, then the signal U100K01 goes to 1 for two logic cycles.

Example 2

If the following statement is entered in the machine logic:

U17A3=U12K6*"RIF C.U."

when the user sends the string "RIF C.U." from the keyboard by pressing pushbutton **P2**, and the signal U12K6 is at 1, then signal U17A3 goes to 1 for two logic cycles.

Four decimal digits sent from the keyboard can be stored in rack K by using the ASCII comparator. Refer to Chapter 3, "Metaoperands" for further details.

Remarks

The ASCII string must be enclosed within inverted commas which are considered syntactical elements.

4.2.13. SIGNAL Operand

Represents a single signal that belongs to the connector rack specified.

Input signal
Ixxyzz

Output signal
Uxxyzz

Where

SYNTAX ELEMENT	MEANING
xx	Number of connector to which the signal belongs.
y	Type of rack. It can have the following values: A = signal belonging to rack A K = signal belonging to rack K T = signal belonging to rack T
zz	Number of the signal in the specified connector: from 0 to 31.

Examples

The expression I0A16 represents the signal on input 16 of connector 0 in rack A.

The expression U12K26 represents the signal on output 26 of connector 12 in rack K.

4.2.14. SIGN Function

Provides a signal according to the sign of the mathematical expression specified.

```
sign=SGN(mat-expr)
```

Where

SYNTAX ELEMENT	MEANING
Sign	Signal assigned by the user containing the status of the signal given by the function.
mat-expr	Word mathematical expression.

Characteristics

If the result of the mathematical expression is positive, the signal is output at logic level 0.

If the result of the mathematical expression is negative, the signal is output at logic level 1.

Example

If the user programs:

```
W50K0=160D
W50K1=90D
U51K0=SGN(W50K1-W50K0)
```

then U51K0 = 1.

4.2.15. TIMER Metaoperand

Acts as a delay relay.

Input
TxxI (time)=sign-bool-expr

Enable
TxxA=sign-bool-expr

Output
sign=TxxU

Pulse
sign=TxxD

Where

SYNTAX ELEMENT	MEANING
Xx	Timer number: from 0 to 63.
time	Timer time base: from 1 to 255
sign-bool-expr	Signal Boolean expression.
sign	Signal assigned by the user containing the output and pulse status.

Characteristics

The timer delay time is given by the product of the time base (time parameter) and a time factor that depends on the type of timer (long or short) according to the following formula:

delay time = time x ft

where ft is the time factor that may have the following values:

- 0.1 for the long timers (xx = 0-47);
- 0.01 for the short timers (xx = 48-63).

Example 1

Timer T26(150) gives a delay time of 15 seconds, i.e.:
delay time = 150 x 0.1 = 15 secondi.

If the time parameter is a constant, the time base is a constant and can only be modified by reprogramming the signal TxxI. If, however, the time parameter is a word, the time base is variable in relation to the current value of the word specified.

If the time parameter is a word, it should be remembered that the system analyzes the time bases on the basis of 16 bits (the word containing the time base plus the next word), whereas only 8 bits (one word) are programmed in the PLC language. For this reason the word following a word containing a time base must not be used and must be set to zero.

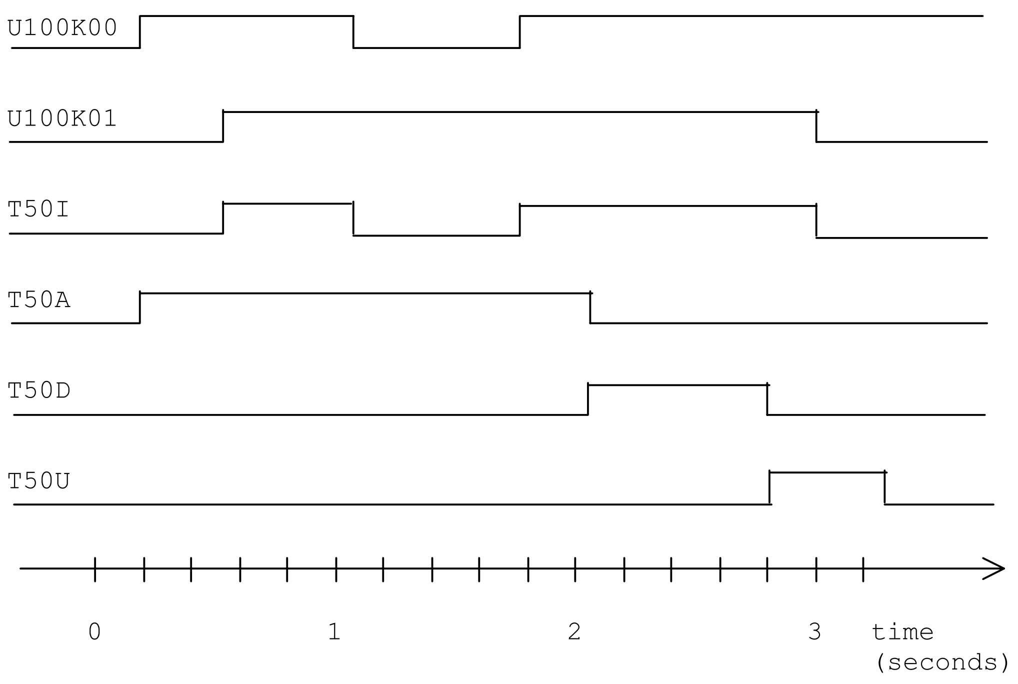
Example 2

The following example shows the programming of a timer where the enable signal (TxxA) goes to 1 and disables the timing.

If the user programs:

```
T50I(100)=U100K00*U100K01
T50A=U100K02
U100K03=T50D
U100K04=T50U
```

The following diagram shows the status of the signals relating to timer T50:



This timer has a delay time of:
delay time = 100 x ft = 100 x 0.01 = 1 second

4.2.16. WORD Operand

Represents eight consecutive signals on one connector.

Wxxyz

Where

SYNTAX ELEMENT	MEANING
xx	Number of the connector to which the word belongs.
y	Type of rack. It may have the following values: A = signal belonging to rack A K = signal belonging to rack K T = signal belonging to rack T
z	Number of word in the specified connector: from 0 to 3.

Characteristics

A word consists of eight consecutive signals on a connector. A connector therefore has four words numbered from 0 to 3. The words and groups of eight signals correspond as follows:

WORD	GROUP OF SIGNALS
0	0 - 7
1	8 - 15
2	16 - 23
3	24 - 31

Examples

The expression W01A2 represents word 2 of connector 01 in rack A.

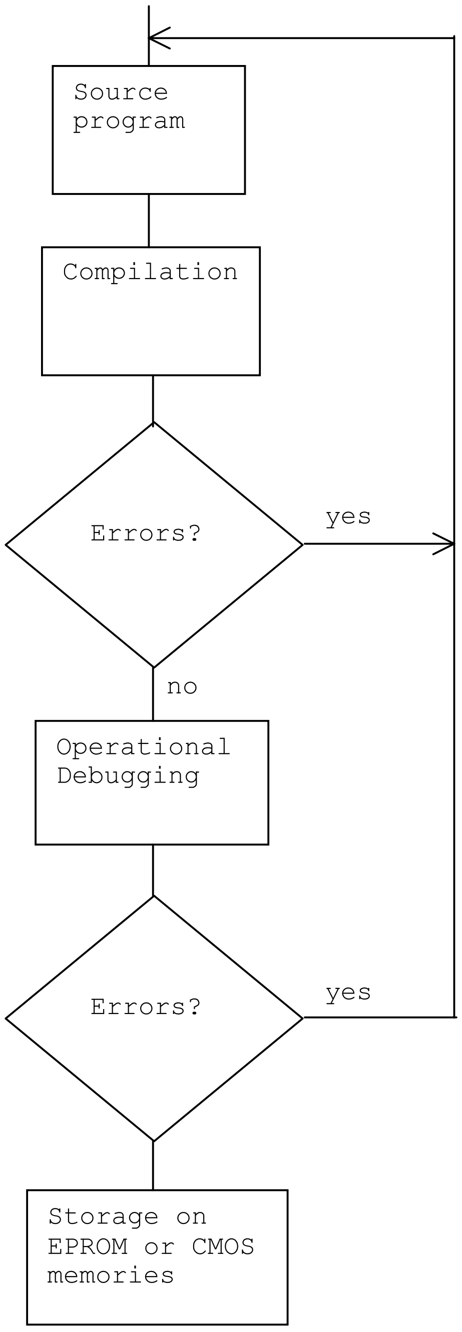
The expression W12K3 represents word 3 of connector 12 in rack K.

5. CREATING A MACHINE LOGIC

5.1. PROCEDURE

Machine logic creation is split up into different stages that require the use of special tools (see Chapter 3, Development Tools). The flow diagram illustrated below shows the various stages in this procedure.

Fig. 5.1. - Machine Logic Creation Procedure



Source Program Programming

At this stage the user draws up the machine logic algorithm to write the machine logic to satisfy his own personal requirements. The algorithm represents the series of operations to be performed by the machine logic.

Flow diagrams can be profitably used to represent the algorithm, as they graphically illustrate the various steps in the handling of the machine tool/control communication. The second part of the manual therefore contains the flow diagrams illustrating the process control activities described in Chapter 7.

The user then translates the algorithm into a program made up of a sequence of statements, with the help of the PLC language. This program constitutes the source program that is entered in the system in the JOB status, using the directives offered by the system editor.

At this stage, the source program is stored in the available characterized memory.

Compilation

Compilation consists in translating the logic program into machine language, to create the object program. Compilation is carried out by the PLC compiler board. When compilation has been completed, the compiler indicates any user or system errors. There are normally two types of errors:

- Fatal.
- Non-Fatal.

The fatal errors occur when the limits set by the system's resources are exceeded (memory overflow, time-out etc.). These errors immediately interrupt the compilation procedure.

The non-fatal errors are syntax errors in the programming made by the user, and they do not interrupt compilation (unless the "exit" option is requested). The resulting object program cannot normally be used unless the errors are of the type that do not affect the object program's validity. In this case the non-fatal errors are of the warning type.
The system indicates the presence of an error by displaying one of the following messages, according to the type of error that has occurred:

```
* compilation error * nn

* compilation warning * nn

COMPILATION ABORTED motive
```

where:

nn	Represents the error code: from 1 to 49.
motive	Represents the cause of the error.

Appendix "Error Messages and Codes" lists the error codes and their meanings.

The first type of message is displayed when a non-fatal error occurs. In this case the cause of the error must be eliminated and compilation repeated.

The second type of message is displayed when a warning type non-fatal error occurs. In this case recompilation is not required.

The third type of message is displayed when a fatal error occurs, or when a non-fatal error occurs having requested the "exit" option in the compilation phase.

A fatal error is caused by the system resource limits being exceeded. The cause of the error must be eliminated and compilation repeated.

When the error makes the object program unusable, the cause of the error must be eliminated and the program recompiled by repeating this procedure until there are no error indications.

If the system is switched off during this phase, the system must be switched back on in the emergency status (ON CYCLE START) or the machine logic must be disabled in the IOCFIL file in order to carry out the necessary corrections.

Refer to the section on "Machine Logic Compilation" in this chapter for a description of the compilation procedures.

Operational Debugging

Operational debugging consists in testing and improving the object program during its execution. This phase is used to check that the machine logic is operating correctly by performing a series of operations such as the displaying of the variable values in real time, connection of a certain number of lines etc. At the end of the debugging stage the executable object program is ready for installation in the memory.

A special object program, called the debug file, must be created in order to debug the machine logic. This file is created by the system upon request from the user, and it is enabled for execution with all the debugging options (connection, disconnection, variables display etc) requested by the user.

Any errors in the writing of the machine logic made by the user are displayed during debugging. If any errors are encountered at this stage, the cause of the error must be eliminated, and compilation and debugging repeated. This operation must be repeated until no more errors are indicated.

Debugging is performed by the PLC debugger.

Refer to "Machine Logic Operational Debugging" in this chapter for further details.

Storage

The executable object program can be processed by the system only if it is first installed in the system memory, which can be done in two different storage areas.

- RAM CMOS memory.
- EPROM memory.

In the first case, the executable object program is loaded n the CMOS RAMs, added as a CMOS RAM support, at the memory address declared in AMBIENT during compilation. The CMOS RAM is mainly used during the debugging phase to allow alterations during object programm execution.

In the second case, the executable object program is transferred from the CMOS RAMs to the EPROMs, by means of the RP 600 or ELAN C41 programmer, connected to the NC-110 system through the serial line. The programmed EPROMs are then installed directly on the NC-110 system at the memory address declared in AMBIENT. This second procedure is executed at the end of debugging when the machine logic has been tested and certified.

Refer to Chapter 7 "Machine Logic Installation" for a description of the machine logic storing procedures.

5.2. CREATION AMBIENT

All the machine logic creation phases described above are carried out by using special tools in NC-110 system configurations.

5.2.1. PROGRAMMING LAYOUTS

To write the source program, the user can make use of a set of special sheets called programming layouts. These layouts help the user to program, find signals and metaoperands, and create a complete documentation of the machine logic source program. The following layouts are available:

- signal layouts;
- timer layouts;
- counter layouts;
- pulse generator layouts;
- program layouts.

These layouts are enclosed in Appendix B of this manual, and must be compiled by the user during machine logic creation.

The layouts referring to the signals on connectors 0 to 25 of each process are described in Chapter 8, "Interface Signals". These layouts list the reserved signals used by the interface to handle a series of standard system control activities. The user is responsible for compiling the layouts of the signals he has defined.

Tne timer, counter and pulse generator layouts must be compiled by the user during machine logic creation. These layouts help organize the metaoperands used in the machine logic clearly and concisely.

The source program statements are written in the program layouts. Each layout can group together the logical statements relating to a certain activity. In this way, each layout identifies a particular stage in the control carried out by the machine logic.

5.2.2. PLC AMBIENT

The PLC AMBIENT must be set in order to compile or debug the machine logic. The AMBIENT consists of two display pages, displayed on the system display, that list a set of parameters that must be assigned a value by the user. These parameters give the system the basic information needed to execute the compilation and debugging phases correctly.

The parameter values entered during an AMBIENT session are stored in the SIPCON file. This file must be present in the memory when the compiler is called. The AMBIENT can be called up in two ways, according to whether it has already been set or not:

- If it has not been set (no SIPCON file), it is displayed by sending the command RUN, PLC, from the keyboard when the system is in the JOB status.

- If it has been set (SIPCON file present), it can be called up by selecting option A in menu 1 (see "Compilation and debugging menu" in this chapter).

Fig. 5.2. and Fig. 5.3. illustrate the composition and the contents of the two display pages that make up the AMBIENT. It is possible to change page by pressing the CR key.

Fig. 5.2. - AMBIENT page 1

```
 8086  PLC COMPILER/DEBUGGER * vers. ____ambient option


select entry:

use ↑ and ↓ keys followed by <send> or for page change
-------------------------------------------------------------
Program Pathname :                Compile options   :

Print device     :

Source pathname 0:                Source pathname 1:

Source pathname 2:                Source pathname 3:

Source pathname 4:                Source pathname 5:

Source pathname 6:                Source pathname 7:

Source pathname 8:                Source pathname 9:
```

Fig. 5.3. - AMBIENT page 2

```
* 8086  PLC COMPILER DEBUGGER * vers.___ambient option

select entry:

use ↑ and ↓ keys followed by <send> or for page change
------------------------------------------------------------
Program Pathname   :         Compile options      :

Print device       :

Fast time max      :         Slow time max        :

Load address       :         Debug Load address   :

Exec address       :         Debug exec address   :

Program max (Kb)   :         Dbg prog max (Kb)    :

I/O load address   :         Logic load address   :

I/O exec address   :         Logic exec address   :
```

The two AMBIENT display pages have a common heading followed by several lines that list the names of the AMBIENT parameters. The first three parameters on each page are the same, but can only be characterized on the first page. The other parameters are different.

5.2.3. SETTING THE PLC AMBIENT

The AMBIENT is set by assigning appropriate values to the AMBIENT parameters. The values are assigned to each parameter by positioning the selection cursor on the parameter required.

The selection cursor is a luminous rectangle that covers the Parameter name, and it can be positioned by using the **LINE BACK** and **LINE SKIP** keys. After positioning the selection cursor on the parameter, press **SEND**, then enter the desired value and press **SEND** again to send it to the system.

This part of the chapter describes the individual parameters listed on the two AMBIENT display pages with their meaning and assignable values.

Program Pathname

The user uses this parameter to give the system the name of the object program and its logical device, in the following format:

name/dev

where:

name	Name of the object program that may have up to five alphanumerical characters. The first character cannot be a number.
dev	Logical device where the object program resides. Enter MEM to indicate the system memory at the Address declared in the AMBIENT "load address" parameter.

Compile Options

The user can use this parameter to select the compilation options he wishes to activate for fast compilation (refer to "Fast compilation" in this chapter). These options are exactly the same as those listed in menu 2 and can be requested with this menu for compilation with options (refer to "Compilation with options" in this Chapter).

The options are selected by keying in the initials of the desired options after the "compile options" parameter. When the user keys F in menu 1 to request fast compilation, compilation is activated with all the options requested by the "compile options" parameter. The options that can be entered in this parameter are listed below.

Debug. This option automatically creates the debug file when compilation is performed. When the system receives this request, it creates a file with the name of the object program, previously declared by the user, and adds the letter "D" to the end of it. (Refer to "Creating the Debug file" in this Chapter).

Bit. This option creates an object program with the possibility of reading and writing the individual bits in the rack A variables during program execution. The bits are accessed at the physical address of rack A relating to the input-output board and not at the memory address where the variables are processed. If this option is selected, the I/O Load address and the I/O Exec address must be set in AMBIENT. The physical address of rack A is 1000 hexadecimal.

Interrupt. This option creates an object program with system interrupt enable during program execution. If this option is selected and the system interrupt occurs while a statement is being executed, this statement is executed in the next sampling operation. The system reserves an area of five bytes between one statement and the next to load any statements that are truncated by the interrupt.

This option should only be used if there is a CPU dedicated to machine logic processing to avoid problems arising during statement execution. If this option is not used (interrupt disabled), the control reserves 5 bytes of memory for each program statement.

Save. This option can be used to save the last version of the object program if a new version is compiled with fatal errors that make it unusable. The old object program is only substituted at the end of a new compilation in which no fatal errors have occurred. Both the old object program and the new object program are therefore present in the memory during compilation.

This option should not be used, however, if the memory does not have sufficient space to contain both old and new object programs. If this option is not used, the old version of the object program is irreversibly deleted before beginning a new compilation.

Video on. This option creates an object program where each source program statement is displayed on the system display as it is compiled.

Exit on error. This option interrupts the compilation if a non-fatal error occurs (normally the system only interrupts compilation if fatal errors occur). The interrupt is made on the program line containing the error. This option, however, has no effect in the case of "warning" type errors, and compilation is not interrupted.

Print Device

This parameter gives the system the name of the default logical device where the print file is loaded. The following names are available:

- LPO for the printer;
- TYO for the teletype;
- MPx for the CMOS RAMs according to the system's capacity.

Source Pathname

The names and logical devices of the 10 tasks into which the source program can be devided are given by means of the source pathname parameters (10 parameters numbered 0 to 9). The parameters are entered with the following format:

name/dev

where:

name	Name of the source program task that may have up to six alphanumerical characters. The first character cannot be a number.
dev	Source program task logical device. It may be selected from the following: MPx, HD0.

The source task containing the fast logic and the seporator "$" must be declared before all the other source tasks. If a cross reference file is created, it must be declared as task 0.

Fast Time Max

This parameter defines the maximum amount of time the system can dedicate to fast logic execution. The amount of time must be expressed in microseconds. Refer to Chapter 3, "Machine Logic execution procedure" for the definition of the value to be supplied.

Slow Time Max

This parameter defines the maximum amount of time that the system can dedicate to the whole slow logic execution (sampling cycles). The time must be expressed in milliseconds. Refer to Chapter 3 "Machine Logic execution procedure" for the definition of the value to be supplied.

Load Address

This parameter defines the object program load address. This address must be expressed in hexadecimal. The object program is loaded starting from this address.

Debug Load Address

This parameter defines the debug file load address. This address must be expressed in hexadecimal. The debug file is loaded starting from this memory address.

Exec Address

This parameter defines the object program execution address. This address must be expressed in hexadecimal and corresponds to the object program load address (load address parameter). This address must also be indicated in the IOCFIL file, section 1, triletteral ALM.

Debug Exec Address

This parameter defines the debug file execution address. This address must be expressed in hexadecimal and corresponds to the debug file load address (debug load address parameter).

Program Max (Kb)

This parameter defines the maximum extension of the object program expressed in Kbyte. The value must be expressed in hexadecimal and must not exceed the value of 32 kbyte (20 hex).

Dbg prog max (Kb)

This parameter defines the maximum debug file extension expressed in kbyte. The value must be expressed in hexadecimal and must not exceed the value of 32 kbyte (20 hex).

I/O Load Address

This parameter defines the load address of the physical I/O variables in rack A. This address must be expressed in hexadecimal. Refer to Chapter 3, "Variables memory map", for the value to be entered.

Logic Load Address

This parameter defines the load address of the logical variables in rack K. This address must be expressed in hexadecimal. Refer to Chapter 3, "Variables memory map" for the value to be entered.

I/O Exec Address

This parameter defines the memory address where the physical I/O variables of rack A are stored during machine logic execution. This address must be expressed in hexadecimal and corresponds to the physical I/O variables load address (I/O load address parameter).

Logic Exec Address

This parameter defines the memory address where the logical variables in rack K are stored during machine logic execution. This address must be expressed in hexadecimal and corresponds to the logical variables load address (logic load address parameter).

The parameters of the second AMBIENT screen must be set as follows:

```
*  8086  PLC COMPILER DEBUGGER * vers. _____ambient  option

select entry:

use ↑ and ↓ Keys followed by <send> or for page change
----------------------------------------------------------------
Programm Pathname   :          Compile options      :

Print devicet       :

Fast time max       :          Slow time max        :

Load address        : 4808     Debug Ioad adress    :  4808

Exec adress         : 4808     Debug exec adress    :  4808

Program  max (Kb)   : 0020     Debug prog max (Kb)  :  0020

I/O load address    : 2762     Logic load address   :  2705

I/O exec address    : 2762     Logic exec adress    :  2705
```

AMBIENT storage

The parameters entered on the display pages proposed by the AMBIENT are stored in the SIPCON file. Once the parameter values have been entered on the two AMBIENT display pages, press the **ESCAPE** key and the following command menu will be displayed:

[UPDATE|ABORT|MODIFY]

A command is selected by pressing its initial letter. The meanings of the commands are given below.

COMMAND	**MEANING**
UPDATE	Exit and save. The system exits the AMBIENT and save the values entered during the last AMBIENT session.
ABORT	Exit without saving. The system exits the AMBIENT without saving the values entered during the last AMBIENT session.
MODIFY	Save without exiting. The values entered during the last AMBIENT session are saved and the system returns to AMBIENT.

If the ABORT command is sent at the end of the first AMBIENT session, all the values entered are lost and the SIPCON file is not created. If, however, the ABORT command is sent at the end of a session that comes after the first session, only the values entered during the current session are lost and the SIPCON file remains in the memory with the values that were entered in the previous sessions.

5.2.4. COMPILATION AND DEBUGGING MENU

The compilation and debugging procedures are performed by the user making a number of selections from a series of 7 menus. Each of these menus consists of a list of items that correspond to executive commands relating to compilation or debugging operations. The commands in the various menus are alternatives to each other, except for menu 2 where several commands can be selected at the same time.

The user can select a particular command by keying its initial letter on the keyboard. This brings about the immediate execution of the corresponding command. Except for menu 2, it is not necessary to press the **SEND** key after the initial letter has been keyed.

Information on how to use the commands can be obtained for all the menus simply by pressing the character "?". A message containing information on how to use a particular command can be displayed by keying in a question mark and then selecting the relevant menu command.

This part of the chapter describes the individual menus displayed during compilation and debugging. Fig. 5.4. and Fig. 5.5. illustrate the interconnection between the various menus, and summarize all the possible commands that can be selected.

Menu 1

Menu 1 consists of the following:

[Compile|Fast compile|Debug|Ambient|Exit]

This menu is displayed (if the AMBIENT has already been set) by entering RUN, PLC. This calls up the PLC utility and displays menu 1 which is the main menu in the series of menus displayed. The functions of the commands available are described below.

Compile: Compilation with source program options.

Fast compile: Fast source program compilation according to the "compile options" parameter set in AMBIENT.

Debug: Entry into debugging status.

Ambient: Entry into AMBIENT.

Exit: Exit from PLC utility.

Menu 2

Menu 2 consists of the following:

[Debug|bit|Interrupt|Save|Video on|Exit on error]

This menu contains the commands for the compilation options that can be set in the AMBIENT "compile options" parameter. Several options can be selected at the same time by entering one option after the other. After each option has been selected, the system waits for the next option. To send the selected options, press **SEND** at the end of the options sequence. The sending of the options sequence executes compilation with the selected options. At the end of the procedure menu 1 will reappear on the display.

The names and meanings of the options presented by this menu are the same as the options in the AMBIENT "compile options" parameter. (Refer to "SETTING THE PLC AMBIENT" in this chapter).

Menu 3

Menu 3 consists of the following:

[R-M|I-M|P-M|Ass|Go|Stop|Loa|Trace|]

Depending on the command selected, either a menu will be displayed on the display or menu 3 will reappear. The functions of the commands available are described below.

R-M (real time monitor): Displays the variable values in real time.

I-M (interval monitor): Graphically displays the variable values in real time (using an oscilloscope).

P-M (print monitor): Stores the variables currently displayed on the monitor in the trace file.

Ass (assign): Assigns variable values from the keyboard.

Go: Enables debug file.

Stop: Disables debug file.

Loa (load): Loads debug file.

Trace: Creates trace file.

Menu 4

Menu 4 consists of the following:

[R-M|I-M|P-M|Ass|Con|Dcon|Go|Stop|Load|Trace]

This menu has a series of commands that are common to menu 3 plus the commands, Con and Dcon. Depending on the command selected, either another menu is displayed on the monitor or menu 4 reappears. The functions of the commands available are described below.

The commands R-M, I-M, P-M, Ass, Go, Stop, Loa, Trace have already been described in menu 3. The functions of Con and Dcon are described below.

Con (connect): Connects debug file lines.

Dcon (disconnect): Disconnects debug file lines.

Menu 5

Menu 5 consists of the following:

[Add|Delete|Go|Stop]

Either menu 4 or menu 5 will appear after selecting a command. The functions of the commands available are described below.

Add: Requests the display of the numerical value of variables in real time.

Delete: Deletes the request for the display of the numerical value of variables in real time.

Go: Activates monitor to display the values of the variables requested by Add in real time.

Stop: Disables monitor.

Menu 5b

Menu 5b consists of the following:

[Add|Delete|Go]

Either menu 4 or menu 5b will be displayed, after selecting a command. The functions of the commands available are described below.

Add: Requests graphic display of variables.

Delete: Deletes request for graphlc display of variables.

Go: Activates monitor for graphic display of variables requested with Add.

Menu 6

Menu 6 consists of the following:

[Initialize|Status info|No status info]

Either menu 4 or menu 6 will be displayed after selecting a command. The functions of the command available are described below.

Initialize: Initializes trace file.

Status info: Activates storage of the system messages, displayed during the debugging procedure, in the trace file.

No status info: Deactivates storage of the system messages, displayed during the debugging procedure, in the trace file.

5.2.5. MENU DIAGRAM

To simplify compilation and debugging procedures performed by following the various menus described earlier, fig. 5.4. and fig. 5.5. illustrate the interconnection between the various menus and summarize the items.

Fig. 5.4. - Compilation and Debugging Menu

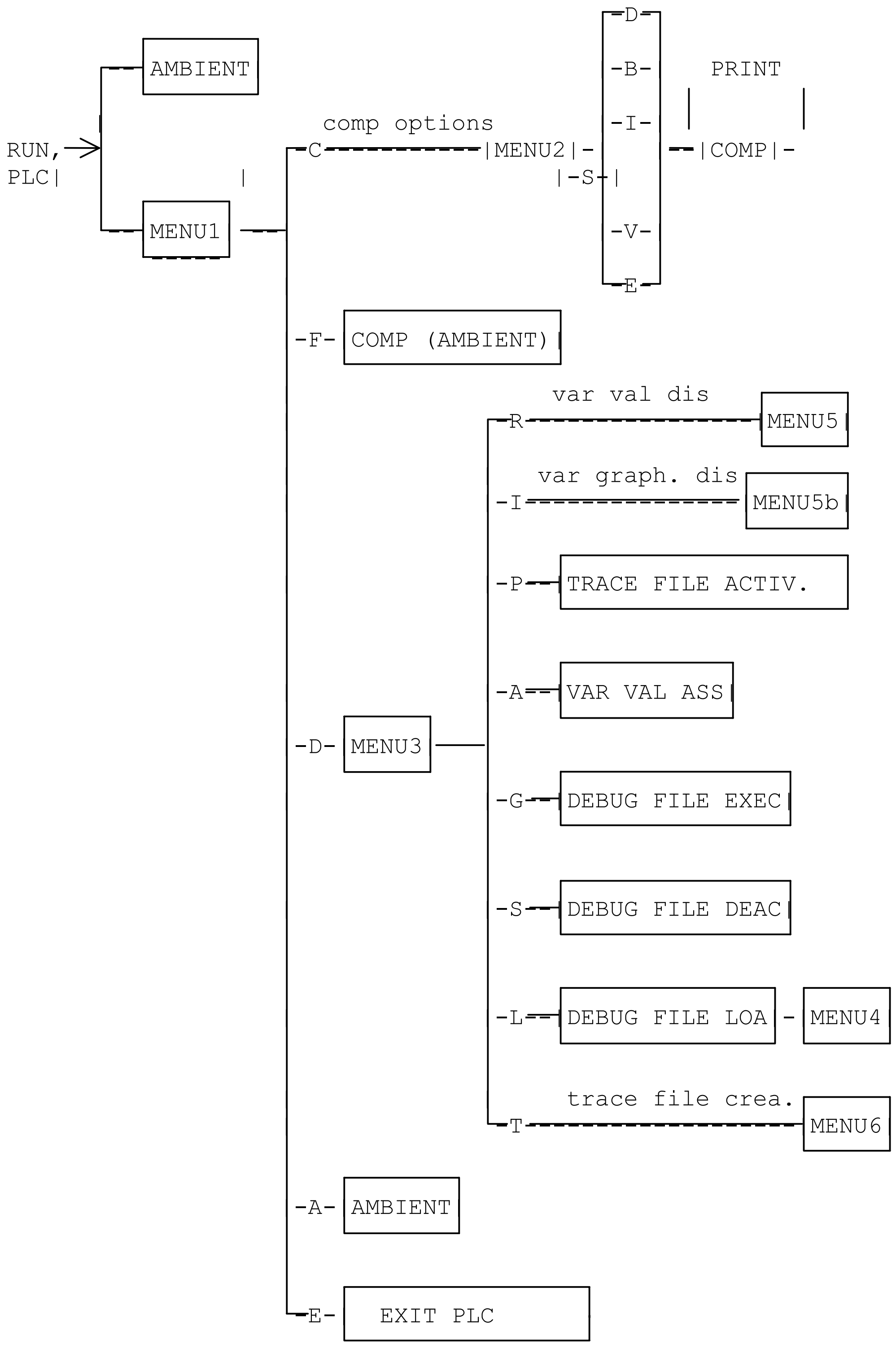
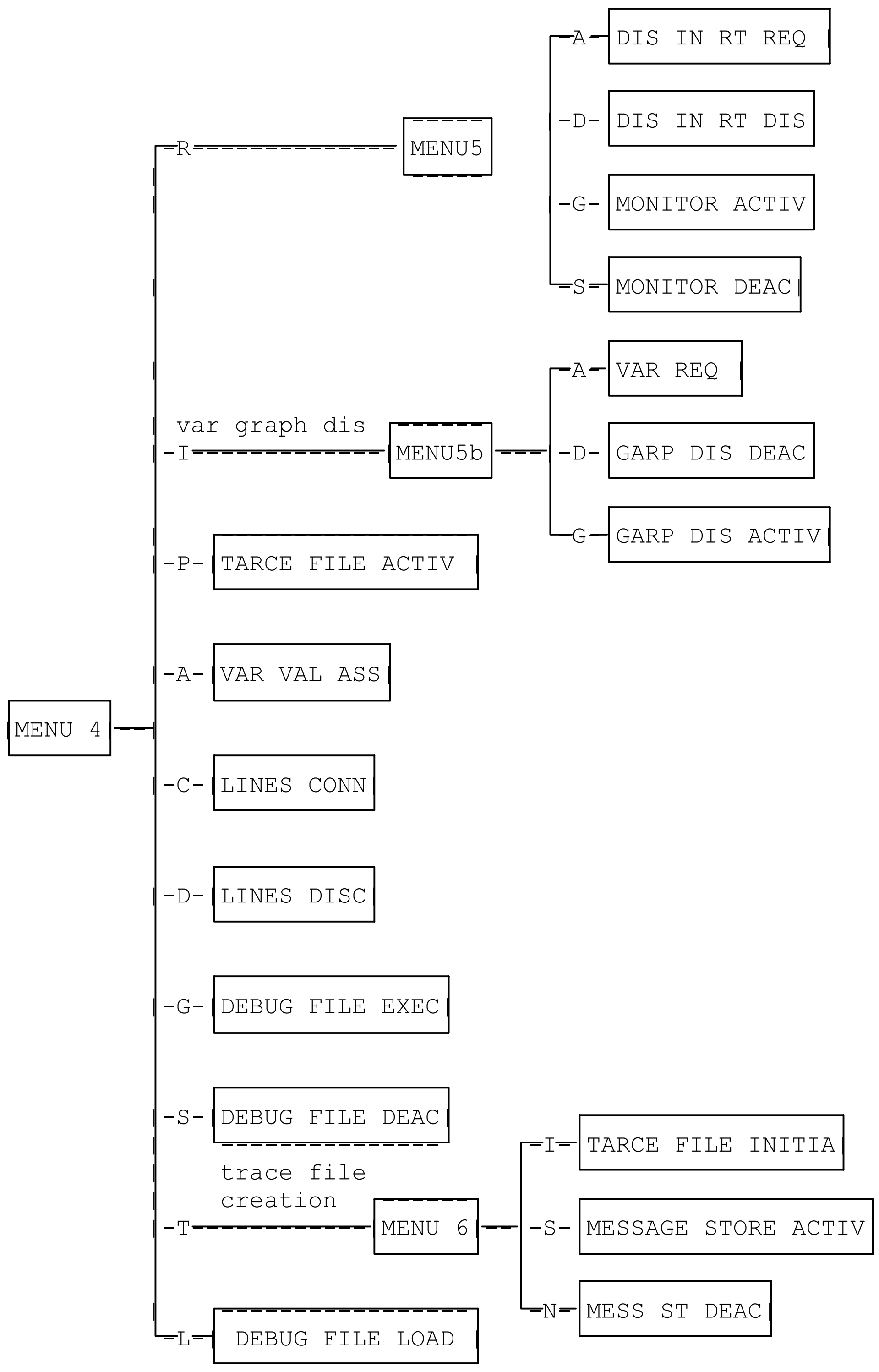


Fig. 5.5. - Compilation and Debugging Menu



Key to the Diagram

The meanings of the abbreviations used in the diagram are given below:

ACTIV = activation	INTR = introduction	VAR = variables
ASS = assignment	ST = store	DIS = display
LOA = loading	MESS = messages	crea = creation
COMP = compilation	OBJ = object	val = values
CONN = connection	REQ = request	var = variables
DEAC = deactivation	RT = real time	dis = display
EXEC = execution	DISC = disconnection	GRAP = graphic
VAL = values		

The diagram illustrates all the menu configurations offered by the system during the various logic program compilation and debugging phases. Each menu branches off into the various options that the user can select by keying in the characters indicated.

The boxed expression that corresponds to a particular item in the menu indicates the operation or the command that will be executed when that particular item is selected. For example, if L is selected in menu 4, the debug file is loaded. The boxed expression "DEBUG FILE LOAD" is indicated on the corresponding branch of the diagram.

If the selection of an item directly displays a menu, a comment relating to the functions of the menu is given on the branch of the diagram corresponding to that item. For example, in the case of menu 3, the selection of item R directly calls up menu 5. The comment "var val dis" is written on the branch of the diagram corresponding to this item indicating that menu 5 refers to the displaying of the variable values.

5.3. MACHINE LOGIC COMPILATION

Before beginning the compilation procedure, the user must first set the PLC AMBIENT (refer to SETTING THE PLC AMBIENT in this chapter) that contains information that is essential for the system to compile and debug the machine logic.

For all types of compilation that will be described below, it is advisable to carry out compilation with the process deactivated. This means that the NC-110 system is switched on in the INIT ABORT status (pressing the pushbuttons **ON + CYCLE START**) and then moving into the JOB status by pressing the P0 key. In this status the functions of the NC-110 are active, while the M.T. auxiliaries are OFF.

The following directive must be send to start the compilation procedure:

RUN,PLC

At this point one of the following two situations may occur:

- The PLC AMBIENT is displayed, if it has not yet been set (SIPCON file not stored).

- Menu 1 is displayed, if the AMBIENT has already been set.

Menu 1 is the main menu of the various menus displayed, and it is used to compile the object program. There are two types of compilation:

- Fast compilation.

- Compilation wiitn options.

In both cases, when compilation is requested, the system creates an object program with the name:

nameO

where "name" is the name of the object program declared by the user and the final letter "O" identifies the type of program.

The object program that is created after compilation is automatically executed at every system power-up.

Refer to "Compilation and Debugging Menu" in this chapter, whenever compilation and debugging are referred to in the following sections.

5.3.1. FAST COMPILATION

Fast compilation is compilation with all the options set in AMBIENT in the "compile options" parameter (refer to "Setting the PLC AMBIENT" in this chapter). This procedure is activated by selecting the Fast compile command in menu 1. If the names of the source program and the object program ("source pathname" and "program pathname" parameters) are not declared in AMBIENT, the system requests the missing data before beginning compilation.

Compilation can be interrupted by pressing the ESCAPE key, and resumed or not by giving the replies Y or N to a prompt from the system.

5.3.2. COMPILATION WITH OPTIONS

Compilation with options is compilation with a series of options offered by a special menu, which can be selected by the user according to his own particular requirements. This procedure is activated by selecting the Compile command in menu 1. The system then requests the name and the logical device of the source program modules which must be entered with the following format:

name/dev

where:

name	Name of the source program module that can have up to five alphanumerical characters. The first character cannot be a number.
dev	Logical device where the source program resides. (MPx).

The source program can be subdivided into a maximum of 10 tasks numbered 0 to 9. The source task containing the fast logic and the separator "$" must be declared before all the other source tasks.

It should be remembered that if a cross-reference file is generated, it must be declared as task 0. Compilation gives a single object program regardless of the number of source tasks entered. If the logical device where the object program resides is omitted, the system assumes the device declared in the AMBIENT "source pathname" parameter by default.

After the names of the source tasks have been entered, the system displays menu 2 that offers the compilation options. Options that are not requested assume the disabled status by default. Once the compilation options have been selected, the system prompts for the name and the logical device of the object program which must be entered with the following format:

name/dev

where:

name	Name of the object program that can have up to five alphanumerical characters. The first character cannot be a number.
dev	Logical device where the object program resides. Enter MEM to indicate the system memory at the address declared in the AMBIENT "load address" parameter.

The system creates an object program with name nameO. If the object name is omitted, the system assumes the source program name by default.

The same options selected in the fast compilation can also be selected in compilation with options, except that in fast compilation the options are selected only once (until the AMBIENT is altered), whereas in compilation with options, the user selects the options he requires at each new compilation.

Creating the Print File

Once the name of the object program and the logical device on which it resides have been entered, the system prompts for the name and the logical device of a possible print file. The print file contains the source program with the line numbers, the list of possible errors, execution time and the amount of memory occupied by the object program.

Ths system prompts for:

Print file
(name) (/dev) or "=" for ambient print file

the user must enter the name and logical device of the print file with the following format:

name/dev

where:

name	Name of the print file that may have up to five alphanumerical characters. The first character cannot be a number.
dev	Logical device where the print file resides. Possible devices are: MPx, LPO, TYO.

The system creates a print file with the name nameL. If the user enters the character "=" a print file is created with the name of the object program with final letter "L" on the logical device declared in the AMBIENT "print device" parameter.

5.4. MACHINE LOGIC OPERATIONAL DEBUGGING

Machine logic operational debugging can only be carried out if the object program has been written and a special debug file has been created. Debugging consists of the following checking and verification operations that enable the user to confirm that the machine logic is operating correctly:

- creating the debug file;
- loading the debug file;
- connecting the debug file lines;

- disconnecting the debug file lines;

- executing the debug file;

- stopping the debug file;

- displaying the variable values;

- graphically displaying the variables;

- assigning values to variables;

- creating the trace file.

Refer to "Compilation and Debugging Menu" in this chapter whenever the compilation and debugging menus are referred to in the following sections.

5.4.1. DEBUG FILE

The debug file is an object program that is enabled for execution with the user-selected debugging operations. This file can only be executed during debugging and under the control of the PLC debugger.

Unlike the object program created simply by performing compilation, the debug file is not executed automatically at system power-up. If a request is made for a debug file during the last compilation, the machine logic will be disabled at system power-up. To reactivate it, the user must enter the debug status via menu 1 and request debug file loading.

The following sections describe all the debugging operations that can be activated in the debug file, by using the commands in the compilation and debugging menus.

5.4.2. CREATING THE DEBUG FILE

The debug file can be created during the source program compilation phase in the following two ways:

- During fast compilation, if the "debug" option has be set in the AMBIENT "fast compile" parameter.

- During compilation with options, if the "debug" option is selected in menu 2.

After the debug file request, the system creates a file with the name:

nameD

where "name" is the name of the object program previously declared by the user, and the final letter "D" identifies the type of file.

After compilation has been completed, the debug file must be loaded in the memory in order to be executed.

5.4.3. LOADING THE DEBUG FILE

This operation transfers the debug file into the system memory at the address specified in the AMBIENT "debug load address" parameter. The debug file is loaded by selecting the Debug command in menu 1. Menu 3 is then displayed where the Loa (load) command is to be selected. At this point the system prompts for the name of the debug file and the logical device where it resides, which must be entered with the following format:

name/dev

where:

name	Name of the debug file previously created by the system.
dev	Logical device where the debug file resides. Enter MEM to indicate the system memory at the address specified in the AMBIENT "debug load address" parameter.

If the logical device is omitted, the system assumes the object program logical device daclared in the AMBIENT "program pathname" parameter by default.

5.4.4. CONNECTING THE DEBUG FILE LINES

This operation activates a certain number of debug file lines. When a debug file execution request is made, only the connected lines are executed. This command can be used only if the debug file has been loaded with the Loa (load) command from menu 4, and implicitly contains the file disable (Stop) command. It should be remembered that the debug file is created with all lines disconnected and therefore cannot be executed unless a certain number of lines are connected.

The debug file lines are connected by sending the Con (connect) command, and then entering, in response to the system's prompt, the lines that are to e connected. The numbers of the lines entered refer to the source program. The connected lines, however, are object program lines that correspond to the declared lines. The line numbers must be entered with the following format:

lineam,linean

where:

lineam	Number of a line referring to the source program.
linean	Number of a line referring to the source program.

The lines are connected according to the lineam and linean values as follows:

- If lineam is less than linean the lines between lineam and linean are connected.

- If lineam is greater than linean the lines between line 1 and linean, and the lines between lineam and the last line of the file are connected.

The character "*" can be used to indicate a group of lines or all the file lines according to the following:

If

***,linean**

is entered, all the lines between line 1 and linean are connected.

If

lineam,*

is entered, all the lines between lineam and the last line of the file are connected.

If only an asterisk is entered

*****

All the file lines are connected.

Examples:

If

7,21

is entered, the lines numbered 7 to 21 are connected.

If

21,7

is entered, the lines between 1 and 7 and the lines between 21 and the last line of the file are connected.

5.4.5. DISCONNECTING THE DEBUG FILE LINES

This operation disconnects a certain number of debug file lines. When debug file execution is requested, all the debug file lines are executed except for those that are disconnected. This command can only be used if the debug file has been loaded with the Loa (load) command from menu 4, and implicitly contains the disable (Stop) command.

This operation is performed by selecting the Dcon (disconnect) command in menu 4 and entering the lines to be disconnected in response to the system's prompt. The line numbers to be entered refer to the source program. The disconnected lines, however, are object program lines that correspond to the declared lines. The line numbers must be entered with the following format:

lineam,linean

where:

lineam	Number of a debug file line.
linean	Number of a debug file line.

The lines are disconnected according to the lineam and linean values as follows:

- If lineam is less than linean the lines between lineam and linean are disconnected.

- If lineam is greater than linean the lines between line 1 and linean and the lines between lineam and the last line of the file are disconnected.

The character "*" can be used to indicate a group of lines or all the lines in the file as follows:

If

***,lineam**

is entered, all the lines between line 1 and lineam are disconnected.

If

lineam,*

is entered, all the lines between lineam and the last line of the file are disconnected.

If only an asterisk is entered:

*****

All the file lines are disconnected.

5.4.6. EXECUTING THE DEBUG FILE

The debug file can be executed by selecting the Go conunand in menu 3 or menu 4. This command can only be used if the debug file has been loaded with the Loa command (load) selected in menu 4. The Go command only executes the lines connected with the Con (connect) command. The Go command is activated only if the file is deactivated (Stop). File activation is indicated with three vertical lines displayed after the file name as shown below:

nameD | | |

where the name is the debug file name.

It should be noted that the commands Con, Dcon and Ass implicitly contain the Stop command (program deactivated). These commands, in fact, do not launch debug file execution when they are sent. The only way of executing the debug file is by sending the Go command.

5.4.7. STOPPING THE DEBUG FILE

Debug file execution can be stopped with the Stop command selected in menu 3 or menu 4. This command can only be used if the debug file has been loaded with the Loa (load) conunand selected in menu 4. The Stop command is activated only if the file is activated (Go).

5.4.8. DISPLAYING THE VARIABLE VALUES

The values of the variables contained in the machine logic can be displayed during machine logic debugging. The requested variables are loaded in a memory area called monitor that can hold up to 16 variables.

The variable values are displayed by selecting the R-M command (real time monitor) in menu 3 or menu 4. Menu 5 is then displayed where the Add command must be selected to make the display request. The system then prompts:

ADD variable
NAME

Enter the variable name and press **SEND**. The system will display the value of the variable requested. To enter additional variables, press the Add command again. A maximum of 16 variables can be entered. If more than 16 variables are entered the system will display the message "MONITOR WINDOW FULL".

To display the values of all the variables entered in the monitor in real time, the Go command must be selected in menu 5. The value (assumed when this command is sent) of all the variables in the monitor is displayed each time this command is sent.

If the monitor is full and its contents are to be updated or other variables are to be displayed, some of the previously requested variables must be deleted with the Delete command in menu 5. After selecting the Delete command the system displays:

SELECT ENTRY

```
Variable 1
Variable 2
Variable 3
...
Variable 16
```

The cursor is positioned on the variable to be deleted by using the **LINE SKIP** and **LINE BACK** keys, and then the SEND key is pressed. The variable is then deleted and the list of variables is shifted upwards.

The names of the variables can also be given in mnemonics provided that the mnemonic table is stored in the memory (refer to Chapter 3, "Operands").

The system displays the values of the variables requested with the following format:

variable type: variable name = variable value

where:

variable type	Type of variable requested. It may be one of the following: signal, byte, connector. Note that if the variable requested is identified with a mnemonic name the mnemonic name will be displayed instead of the variable type.
variable name	Symbolic or mnemonic name of the variable requested.
value	Value of the variable requested that is given with the following formats according to the type of variable: - binary for "signal" type variables; - binary and decimal for "byte" type variables; - binary for "connector" type variables.

The types of variables displayed and the types defined in Chapter 3 correspond as follows:

- signal = signal type variable;
- byte = word type variable;
- connector = all the words contained in a connector.

The variables display can be interrupted by disabling the monitor with the Stop command in menu 5.

5.4.9. GRAPHICALLY DISPLAYING THE VARIABLES

The evolution of the variables contained in the machine logic can be displayed graphically during machine logic debugging. The requested variables are loaded in a memory area called monitor that can hold up to 16 variables.

The frequency and the delay time of the variables graphic display with respect to a synchronization signal are defined by the user. To enable the graphic display, the I-M command (interval monitor) must be selected in menu 3 or menu 4. Menu 5b is then displayed.

Key in Add to request the graphic display. The system displays:

ADD variable
NAME

Enter the name of the variable and press **SEND**. Additional variables can be entered by repeating this operation for a total number of 16 variables. If more than 16 variables are entered, the system displays the message "MONITORINDOW FULL".

To enable the variables graphic display, the Go command must be selected in menu 5b, the system will then prompt for the time base and the display delay:

TIME/BASE

the user must enter these parameters with the following format:

time base/delay

where:

time base	Sampling time base and updating of the graphic display of the variables requested expressed in sampling operations (one sampling operation = 10 ms)
delay	Delay, with respect to a synchronization signal, with which the graphic display of the variables requested starts. The delay is expressed in TIME BASE units. The synchronization signal (trigger signal) is then requested.

Example:

If the user enters 2/3 then:

- time base = 20 ms;
- delay = 60 ms.

This means that the status of the requested variables is read and displayed every two sampling cycles, i.e. every 20 milliseconds, and the delay with which the displaying starts after the synchronization signal is three times the TIME BASE, i.e. 60 milliseconds.

When these two parameters have been entered, the system prompts for the name of the synchronization signal:

TRIGGER SIGNAL
(NAME(|)):

The user must enter the name of the synchronization signal. The display is enabled on the synchronization signal rising edge and falling edge by pressing the **LINE SKIP** and **LINE BACK** keys respectively after the name.

Once the trigger signal has been specified, press **SEND** and the displaying will begin as requested.

If the monitor is full and its contents are to be updated or other variables are to be displayed, the previously requested variables must be deleted with the Delete command in menu 5. The system then displays:

SELECT ENTRY

Variable 1
Variable 2
Variable 3
...
Variable 16

The cursor is positioned on the variable to be deleted by using the **LINE SKIP** and **LINE BACK** keys, and the **SEND** key is pressed. The variable is deleted and the list of variables is shifted upwards.

The names of the variables can be also be given in mnemonics provided that the mnemonic table is stored in the memory (refer to Chapter 3, "Operands").

5.4.10. ASSIGNING VALUES TO VARIABLES

Values can be assigned to the variables contained in the machine logic during machine logic debugging. A value is assigned to a variable by selecting the command Ass (Assign) in menu 3 or menu 4. The system then prompts for the name of the variable and the value to be assigned. The user must give the name and the variable value with the following format:

variable=value

where:

variable	Name of the variable to which a value is to be assigned.
value	Value in decimal to be assigned to the variable.

The variable name can be also be given in mnemonics provided that the mnemonic table is stored in the memory (refer to Chapter 3, "Operands").

Once the variable name and the value to be assigned have been entered, press the **SEND** key and the assignment will be displayed with the following format:

variable type: variable name = variable value

where:

variable type	Type of variable requested. It may be either signal, byte. Note that if the requested variable is identified with a mnemonic name the mnemonic name is displayed instead of the variabla type.
variable name	Symbolic or mnemonic name of the variable requested.
value	Value assigned to the variable that is given with the following formats according to the type of variable: - binary for "signal" type variables; - binary and decimal for "byte" type variables.

Note that the types of variables displayed and the types defined in Chapter 3 have the following correspondence:
- signal = signal type variable;
- byte = word type variable.

5.4.11. CREATING THE TRACE FILE

A trace file containing information generated during the debugging phase can be created during machine logic debugging. This file can be redirected onto the print file.

The trace file may contain two types of information:

- Contents of the current variables display monitor (graphic or value display).

- Status messages, displayed by the system, that are generated in the debugging phase by the commands: Ass, Loa, Con, Dcon, Go, Stop.

The trace file is created by selecting the Trace command in menu 5. Menu 6 is then displayed. The trace file must first Sof all be initialized by selecting the Initialize command in menu 6. At this point, the system prompts for the name and the print file logical device where the trace file is to be directed.

Print file
(name) (/dev) or "=" for ambient print file

the user must enter the name and logical device of the trace file with the following format:

name/dev

where:

name	Name of the trace file consisting of a maximum of five alphanumerical characters. The first character cannot be a number.
dev	Logical device where the trace file resides. Possible devices are: MPx, LPO, TYO.

The system creates a file with the name nameT. If the user enters the character "=", a trace file is created with the object program name with final letter "T" on the logical device declared in the AMBIENT "print device" parameter.

The system then indicates that the debug file is active and displays menu 4. At this point, if the user wishes to enter the monitor status in the trace file he must select the P-M command (print monitor) in menu 4. When the P-M command is sent, the status messages are also automatically entered in the trace file.

If the status messages are not to be entered, the no status info command must be selected in menu 6. To re-enable the entry of these messages, the status info command must be selected in menu 6.

6. FINAL STORING OF THE MACHINE LOGIC

After the debugging stage, carried out to check that the cycles and the functions executed by the logic program are functioning and optimized, it is possible to execute the final storing of the logic program on special supports.

This chapter describes how to store the machine logic on the following supports:

- EPROM memories;
- Punched tape;
- Magnetic cassette;
- CMOS RAM memories.

6.1. SETTING THE OBJECT PROGRAM

Setting the control mask.

The mask is calculated by the DEBUG and CKSUM utilities. The procedure is as follows:

Power on the system in emergency and key in:

RUN,DEBUG

which gives access to the main debug menu.

The next operation is necessary to reset segment 5.

Press the F (FILL) key. The system will then request the following data:

THE SYSTEM ASKS	**THE USER MUST INSERT**
MODE:	L (length)
TYPE:	B (byte)
FROM:	4800:0 (start address)
LENGTH:	8000 (32 Kbytes)
VALUE:	0 (to reset the 32 Kbytes)

Exit by pressing the **ESCAPE** key and the E (exit) key.

Send the following instruction:

RUN,PLC

menu 1 is displayed. Press A to access the first page of AMBIENT. Press CR to display the second page. If different values have been declared during the AMBIENT configuration stage, the following values must be entered:

```
LOAD ADDRESS: 4808
DEBUG LOAD ADDRESS:  4808
EXEC ADDRESS: 5008
DEBUG EXEC ADDRESS: 5008
```

Exit from AMBIENT by pressing the **ESCAPE** key and store the data introduced by pressing the U (UPDATE) key.

From menu 1 send the fast compile command by pressing the F key in order to compile the program. If no errors have been found, at the end of the compilation the following message will be displayed:

END OF PROGR. COMPILATION ERRORS 0000

If there are any errors, correct them and then repeat the compilation. Exit from the PLC utility by sending the exit command from menu 1 pressing the E key.

Calculating and storing the control Mask.

Send the instruction:

RUN,CKSUM

the calculation of the control mask is enabled and the system requests the following data:

THE SYSTEM ASKS	THE USER MUST INSERT
TYPE OF EPROM	4 (EPROMs type 27256)
BASE ADDRESS	4800 (start address)
EXE BASE ADD	5000 (start address)
NUMBER OF EPROMS	2 object lower than 32 Kbytes

The values of the checksums of both eproms (HIGH and LOW) will subsequently appear. They are stored in the following 128 bytes at the address 4800. Press the **SEND** key to return automatically to the JOB screen.

At this point it is possible to:

- transfer the machine logic on Eprom memories or on File-Devices;
- set up an external support (tape, cassette) to store the machine logic;
- store the machine logic directly on CMOS RAM system memories.

Enabling the peripherals driver

According to the type of support chosen to store the machine logic, the following peripherals can be used for program transfer:

- EPROM programmer;
- tape punch for tape perforation;
- magnetic cassette recorder;
- teleprinter.

Once the type of support and the peripheral have been chosen, it is necessary to enable to relative driver in section *2 of the FCRSYS characterization file. The allowable parameters are:

SETTING	DRIVER ENABLED
XAPROM,0E,0,5	Eprom programmer
XAPUNC,0A,0,2	Tape punch
XATEAC,0B,0,3	Magnetic cassette
XAT485,07,0,1	Teleprinter

Once the driver has been enabled, it is important to check whether the PIMP utility is present in section 8 of the FCRSYS file. Power down the NC-110 system and re-power it on in emergency status (**ON + CYCLE START**) to activate the new characterization.

N.B. With the eprom programmer (XAPROM) the data transfer is executed with an INTEL MDS format.

The serial line of the eprom programmer must be characterized as follows:

- 1 bit for start;
- 2 bits for stop;
- 8 bits for each datum;
- NO parity.

and the programmer must be enabled in the INPUT FROM SERIAL LINE mode.

Transfer initialization

After the PIMP utility has been enabled, the procedure of program transfer on the previously enabled support begins. Through the serial line connect the chosen peripheral (previously characterized in the FCRSYS file) to the NC-110 system. Switch the peripheral on and execute the following operations:

- EPROM programmer = select the type of eprom and set the programmer in the "INPUT from serial line" mode;

- OLIVETTI TC 485 = set the teleprinter on "LINE".

At the end of these operations, send the instruction RUN,PIMP.

The system will request the following data:

THE SYSTEM ASKS	THE USER MUST INSERT
FUNCTION	D (download)
START ADDRESS	4800:0 (start address)
FUNCTION	L (length)
LENGTH	8000 (for 2 eprom modules)
STEP (Kb)	16 (for eproms 27256)

After having entered the STEP value, the system is ready for the actual data transfer. The transfer begins with the request:

DEVICE:

to which the user must reply by pressing the key aassociated to the name of the peripheral used. This procedure is detailed in the paragraphs that follow.

6.2. TRANSFER ON EPROM PROGRAMMER

Press the **U** key

the following message will be displayed:

Confirm download High Section number 01 (Y/N)

By pressing Y the transfer of data relating to eprom OH onto the memory of the programmer will be executed.

The transfer will be highlighted by the flashing message:

Working

At the end of the transfer the following message will be displayed:

Download successfully terminated. Press SEND to continue

At this point by setting the programmer correctly, the user can burn the EPROM memory marking it with the value OH and with the value of the CKSUM shown by the programmer display.

At the end, it is necessary to set once more the programmer in INPUT FROM SERIAL LINE mode, after which, by pressing the **SEND** key, the following message will be displayed:

Confirm download Low Section number 01 (Y/N)

By pressing the **Y** key, the data relating to eprom OL are transferred onto the memory of the programmer.

The transfer will be highlighted by the flashing message:

Working

At the end of the transfer the following message will be displayed:

Download successfully terminated. Press SEND to continue

At this point by setting the programmer correctly, the user can burn the EPROM memory marking it with the value OL and with the value of the CKSUM shown by the programmer display.

At the end of the burning session of the two eproms, by pressing the **SEND** and then the E keys in succession, the system exits the eprom burning status and goes back to the initial screen (JOB).

Software calibrations to restore the system

Described hereunder are the software calibrations which must be executed to restore the initial working conditions of the system.

- Section 1 of FCRSYS
Define EP4=Y4,5000,0002 (logic on 2 Eproms)

- Section 2 of FCRSYS
Remove XAPROM,OE,1,5 (driver of the eprom programmer)

- Section 1 of IOCFIL
If necessary, modify the value of the ALM record by declaring address 5008.

- Power down the system. Remove the board CPU and insert the eproms just burnt in the appropriate sockets, enabling them by means of the appropriate calibration. Re-insert the board.

6.3. TRANSFER ON PUNCH OR MAGNETIC CASSETTE

Press **C** (Cassette)

or **P** (punch)

the following message will be displayed:

Confirm download High Section number 01 (Y/N)

Press **Y**

- If the punch is utilized, the transfer of the data relating to eprom OH will immediately begin. The transfer will be highlighted by the flashing message:

Working

- If the cassette is utilized, the following message will be displayed:

File name:

Enter the name of the file (a maximum of 6 alphanumeric characters, the first of which must be a letter) and then press **SEND**. The transfer of data relating to eprom OH on the cassette will begin. The transfer will be highlighted by the flashing message:

Working

At the end of the data transfer, the following message will be displayed:

Download successfully terminated. Press SEND to continue

After pressing **SEND** the following message will be displayed:

Confirm download Low Section number 01 (Y/N)

Press **Y**

- If the punch is utilized, the transfer of the data relating to eprom OL will immediately begin. The transfer will be highlighted by the flashing message:

Working

- If the cassette is utilized, the following message will be displayed:

File name:

Enter another file name (a maximum of 6 alphanumeric characters, the first of which must be a letter) and then press **SEND**. The transfer of data relating to eprom OL on the cassette will begin. The transfer will be highlighted by the flashing message:

Working

At the end of the data transfer, the following message will be displayed:

Download successfully terminated. Press SEND to continue

Then press **SEND** and E in succession. The system will exit the transfer status and go back to the initial screen (JOB).

Once the transfer procedure is completed, enclose to the machine the punched tape or the magnetic cassette. It is also a good procedure to enclose the tape and the print-out of the source program.

6.4. STORING ON CMOS RAM

It is possible to store the machine logic on CMOS RAM memory of the NC-110 system, if an area sufficient to contain the entire logic is available. The storing is executed directly in the system CMOS RAM memories. For this type of storing the PIMP utility is NOT utilized. The procedure is as follows:

- Preset an area large enough to contain the machine logic on the CMOS RAM modules of the system memory.

- Send the instruction RUN, PLC and access AMBIENT by pressing **A**. Set the initial address (in hexadecimal) corresponding to the CMOS area reserved to store the machine logic in the following AMBIENT parameters:

LOAD ADDRESS
DEBUG IOAD ADDRESS
EXEC ADDRESS
DEBUG EXEC ADDRESS

- Exit AMBIENT by pressing **ESCAPE**, store the data entered by pressing **U** (UPDATE) and activate a fast compilation by pressing **F** (FAST COMPILE) from menu 1.

- At the end of the compilation, set the same address declared in the abovementioned AMBIENT parameters in record 1 of section 1 of the IOCFIL file.

- Power down the system. Re-power it on and check the system cycles and functions.

If required, by means of the PIMP utility, the object program can be downloaded from the CMOS area in which it resides and saved on tape or cassette.

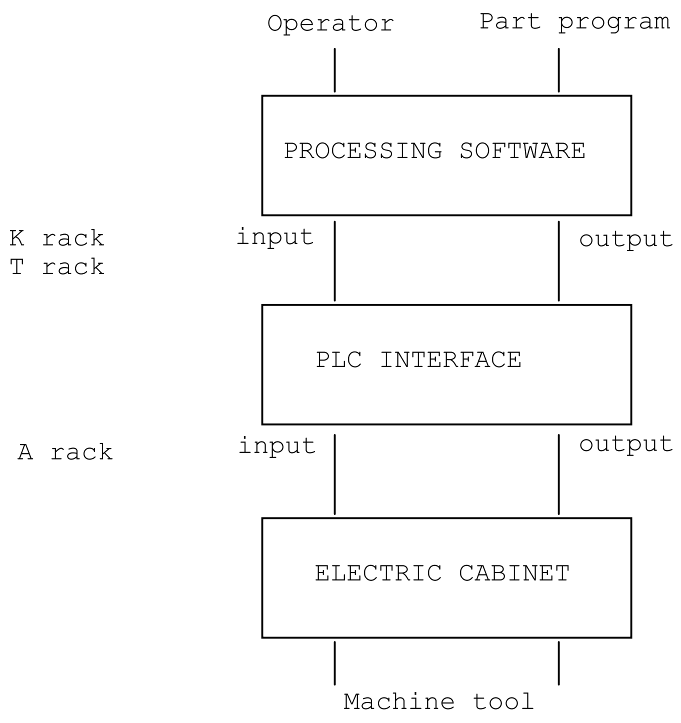
7. PLC INTERFACE

7.1. INTRODUCTION

The PLC interface is a software module that has been developed for the NC-110 line by means of the development tools described in the first part of this manual. The purpose of this interface is to provide a support for communication between the numerical control and the machine tool. These two systems communicate by means of interchange signals which may be either logical or physical. Both logical and physical signals may be input signals (input to interface) or output signals (output from interface).

In accordance with the description given in Chapter 3, the logical signals belong to the K and T racks, while the physical signals belong to the A rack. The signals from the K and T racks interface the system processing software. The signals from the A rack interface the machine tool's electric cabinet. The logic diagram shown in Fig. 7.1. shows how the numerical control and machine tool are interfaced.

Fig. 7.1. - Control-Machine Tool Interfacing



The interface for the NC-110 has two basic functions:

- Control of statuses and enabling by operator and part program by means of the following K rack signals:
  . from 0 to 9 for the first process;
  . from 26 to 35 for the second process;
  . from 52 to 61 for the third process;
  . from 78 to 87 for the fourth process;
  . from 104 to 113 for the fifth process.

- Analysis and execution of requests by the machine logic, for completion of the cycles required, using the following K rack signals:
  . from 10 to 25 for the first process;
  . from 36 to 51 for the second process;
  . from 62 to 77 for the third process;
  . from 88 to 103 for the fourth process;
  . from 114 to 129 for the fifth process.

In carrying out these tasks, the interface makes use of:
- A set of signals which satisfy the regulations contained in ISO and which are therefore widely known;
- A wide range of elementary cycles which can be activated through requests made by the logic using the previously mentioned K rack signals.

The PLC interface allows the numerical control to be customized on the machine tool through the handling of a series of activities typical of the process control. These activities can be summarized as follows:
- Initialization stages;
- Handling of protocol signals which define the various system operating modes;
- Handling of machine tool safety devices;
- Execution of auxiliary S, T, M, H, and indexing axis functions;
- Handling of requests from logic.

These activities are performed through a series of cycles that are described in the next few sections of this chapter. The flow-charts relating to the cycles described are given at the end of the chapter.

For the sake of brevity, the control unit will henceforth be known as CU and the machine tool as MT.

7.2. INITIALIZATION

Initialization involves various operations, the overall purpose of which is to set the CU to stand-by status. The control can be initialized by means of the following cycles:
- Apro;
- MT switch-on after emergency;
- Reset.

7.2.1. APRO

Initialization begins when the power supply is switched on by pressing the **ON** pushbutton. In this case, after reset and self-diagnostics, the CU deactivates the SPEPN signal for the auxiliary circuits, provided that no irregularities (memory or I/O errors) have been detected. At this stage the MUSPE signal must be set to 1. If **ON** is pressed again, the system waits until the MUSPE signal goes to zero, after which the CONP (control ready) signal will be activated for the logic.

7.2.2. MT SWITCH-ON AFTER EMERGENCY

After an emergency (servo error, alarm, MT off), the system waits for the MUSPE to go to zero before de-activating the diagnostics message and activating the CONP signal.

7.2.3. RESET

The system is reset in order to interrupt all operations in progress, whether these relate to all the processes or to one in particular. A reset puts the CU (or the process) in stand-by status. Reset has the following functions: to stop the axes, abort all auxiliary functions that have previously been programmed and inform the machine logic of the minimum duration of two logic cycles, by means of the impulsive RESET signal (I00K1).

All processes are automatically reset if process 0 (system process) is selected and requested by the operator via the console. Alternatively, one particular process can be reset, after making a declaration by means of the CWP statement in the PGCFIL file relating to the process. In this case the reset is asynchronous. Only one process is reset in the following cases:

- By the operator via console, after selection of a process other than the system process;
- By the logic via the REAZ signal;
- By various M functions using settings performed during configuration.

In the first two cases, the RESET is completely asynchronous. In the third, it is synchronized on the M function that requests it.

7.3. OPERATING MODES

The CU has various operating modes. The mode is generally chosen by the operator via a console selector, or by the logic through a suitable request. The interface supplies the logic with a set of signals in accordance with a protocol that clearly defines the mode selected. The operating modes are listed below:
- Disconnected axes;
- Abandoned axes;
- Switched axes;

- Manual (continuous-incremental), return to contour, machine home;
- Auto-semiauto-MDI.

7.3.1. DISCONNECTED AXES

This mode makes it possible to display the values relating to the calculated position of the stationary axes. In this mode, the axes are slaved in their current position and the "servo error" check is performed. This mode can be called up by means of the three-letter code UAS=1.

7.3.2. ABANDONED AXES

This mode makes it possible to drive the axis by means of an external console and to display the motion-related values, without enabling the usual speed and position checks. This mode can be called up by not setting the bits in the WRABI word corresponding to the axes to be abandoned.

7.3.3. SWITCHED AXES

Two axes can be driven on the same analog channel (the names must be declared during configuration). By setting the bit corresponding to the axis to be temporarily released at W17K1 (the system only carries out the servo error check with axes at a standstill), so that another axis can be driven, the system informs the logic that the axis has been released, by means of WO6KO.

7.3.4. MANUAL

This mode is identified by a mode selector position. The only operations allowed in this operating mode are axis motions requested by the operator from console, after the process concerned has been selected, or by the logic. Only one axis at a time can be selected. The motion can be performed either in continuous or incremental mode. In addition, "machine home" can be used in order to provide the absolute axis reference and allow a return to the abandoned contour. If the GU is operating in manual mode, it informs the logic using signals MANUC, MANUJ, RIPRO, and RIMZE, depending on the type of manual movement requested.

When axis motion is requested, the signal related to the axis (MOV1 for axis 1, MOV2 for axis 2, MOV3 for axis 3 etc) is set. The motion is performed as soon as the motion enable signal is displayed (COMU=1). Axis speed and direction are defined by the position of the manual potentiometer. When the motion has been completed, the motion signal is reset. The system informs the logic of the direction of motion of the selected axis using the DIRMN signal (DIRMN=1 negative motion).

7.3.5. AUTO-SEMIAUTO-MDI

In this mode, the CU operates at three different mode selector positions. Despite these three different positions, the type of work performed by the CU is, conceptually speaking, identical. This work involves processing and executing the programmed block (auxiliary and axis motion functions.) If the system process is selected, the operating mode applies to all the processes configured. Otherwise it applies only to the process selected. From one position to another, all that varies is the way in which each block is considered by the system:

- In the AUTO position, the system automatically considers one block after another in the same part program;
- In the SEMIAUTO position, the system considers one block at a time only; it analyzes it and displays it, and then the system enters stand-by mode until the cycle start pushbutton is pressed. In other words, the block displayed is the one that must be executed with the cycle start;
- In the MDI position, the block previously set from keyboard is considered and subsequently executed.

Block execution involves the following stages:

- The AUTO signal is set when in AUTO position, SEMI if in SEMIAUTO position, while EMDI is set when in position MDI;
- The signal relating to the block in progress is set (CYCLE);
- Motion start S, T, M, H, and indexing axis auxiliary functions are performed. Emission takes place in the order listed if "serial emission" is defined during configuration; if "parallel emission" is defined, functions S, T, M and H are performed simultaneously, while the indexing axis function is carried out in the next two logic cycles;
- The stand-by (STABY) signal is reset, the block signals in GOO (FGOO) and fixed cycle are updated in accordance with the features of the block being processed;
- If necessary, an axis motion is performed, after switching;
- Expedite M and H functions are performed;
- The CYCLE signal is reset;
- Expedite M and H functions are reset;
- Motion end auxiliary functions M and H are performed;
- If compensations have been changed, the tools and compensations are updated;
- The STABY signal is set, thus ending execution.

7.4. MACHINE TOOL SAFETY DEVICES

The CU has the following machine tool safety statuses at its disposal:

- Travel limits;
- Emergency;
- Hold;
- Feed hold.

7.4.1. TRAVEL LIMITS

The NC-110 system handles the travel limits internally. This means that the CU checks all the axes to ensure they are within the safety limits. A setting establishes whether the travel limits are handled by hardware and/or software. For handling by hardware there are two travel limit signals (positive or negative) for each axis, while the two limits provided for software handling consist of values defined during configuration, and a home position which represents the reference point for these values.

In software handling, travel limit control is activated only after the axis has been zeroed. It should be noted that the NC-110 system is capable of simultaneously controlling axes handled by both software and hardware.

When the CU esablishes that a given axis has reached its travel limit during a movement, it zeroes the references of all axes involved in the process to which the axis in question belongs, and displays the travel limit concerned.

To exit from the travel limit, the axis concerned must be selected in manual mode and the movement carried out in the direction from which the work envelope is entered. If the direction is not exact, the system suspends the movement. The hardware and software travel limits can be only by-passed simultaneously.

7.4.2. EMERGENCY

Emergency status is entered in the event of an irregularity or danger which jeopardises correct control unit operation so seriously that it must be switched off. Should this occur, the CU triggers a logic which displays the type of emergency causing the switch-off.

There are three types of emergency status:
- Emergency status requested by the system;
- Emergency status requested by the logic;
- Emergency status requested by the operator.

Emergency status is requested by the system when the system detects an irregularity in one of its own components or elements (memories, I/O modules, etc.), or on an axis (axis runaway or loss of feedback).

The emergency requested from logic occurs when the logic itself finds an anomaly in the functioning of the machine tool. In this case it is the logic which has to request the power down of the machine tool through the RISPE signal accompanied by the display of a message apt to identify the type of anomaly.

Emergency status is requested by the logic when the interface detects an irregularity. Should this occur the interface must request that the MT be switched off by means of the RISPE output signal while simultanously displaying the diagnostic message.

If the emergency has been requested by the operator, power has been removed only from the MT auxiliary circuits.

In all three cases, when the MUSPE signal is set to 1, the control displays it in reverse.

7.4.3. HOLD

In hold status, the current machine tool status is frozen, without loss of position. This status can be requested by:
- the CU, for axis return to the contour;
- the operator, by means of the **HOLD** pushbutton on the console;
- the logic, by means of signals HLDR or ROHE.

If the system process (SYSTEM) is selected, the axes involved in all the processes enter hold status. If any of the non-system processes is selected, only the axes involved in the selected process enter hold status. If it is defined in PCGFIL, the process which has made the request by means of CWP enters hold.

Hold status is handled in the following three stages:

- Hold entry;
- Hold exit enable;
- Hold exit.

During the first stage, all axes are stopped in controlled mode, the motion signals (MOV) are zeroed, and the HOLDA signal is set, thus informing the logic that hold status has been entered. During the second stage, signals HLDR and ROHE are examined. If they are activated, a diagnostic message is displayed. To exit hold status, the operator must press the **HOLD** pushbutton. Hold exit is only allowed if the interface approves it by means of HLDR=0 or ROHE=0.

During the third stage, the MOV signals are re-activated to the status they were in when hold was requested, the HOLDA signal is reset and the interupted axis motion is resumed, provided that COMU=1.

While the GU is in hold status, the operations listed below are allowed:

Manual axis motion (the axis must later be returned to the contour).

Sending of a block from keyboard in order to activate auxiliary functions accepted in hold (see IOCFIL file).

7.4.4. FEED HOLD

This request suspends the axis motion in progress, be it automatic or manual. When the request is terminated, the interrupted motion is resumed. The feed and feed hold requests are not accepted in the threading and tapping cycles.

7.5. AUXILIARY FUNCTIONS

There are three different types of auxiliary function, all performed in diffferent ways:

- Motion start functions;
- Motion end functions;
- Expedite functions.

There are also five families of auxiliary functions which differ in the type of operation they perform. These families are as follows:

- S functions;
- T functions;
- M functions;
- H functions;
- INDEXING AXIS functions.

The auxiliary functions can be handled normally or in stored mode. In the first case, they are performed in synchronized mode within the block. In the second case, the CU carries the machine tool to the point in the program that had previously been interrupted.

When the RCM request is made, the system executem the program without generating any movement or transferring any auxiliary function to the K rack. When the atored search has been completed, the auxiliary functions previously stored in the buffer are transmitted to the K rack by means of three-letter code ERM, in the following order: S, T (spindle), Indexing Axis, M - H by search class, programmed T (if present). Finally, a return to the axis contour is requested, after which the previously interrupted program conditions can be restored.

If the indexing axis has been defined as incremental during configuration, the GU transfers the figures corresponding to the difference between the starting point (home position) and the point calculated in the stored search, when the search has been completed. If, on the other hand, the axis has been defined as absolute, the last indexing axis figures programmed are sent to the K rack, when the stored search has been completed. If there are jmps in the part program, the operator must restore the previous conditions of the parameters analyzed.

7.5.1. MOTION START FUNCTIONS

The auxiliary motion start functions are carried out before any axis motion. They are as follows:

- Functions S and T;
- Functions M and H defined as "motion start" functions during configuration;
- INDEXING ARIS functions.

The auxiliary functions, which can be selected during configuration, can be handled in two different ways:

- Parallel handling;
- Serial handling.

In parallel handling, if more than one motion start auxiliary function belonging to different families (except the INDEXING AXIS function, which is always executed last), they are executed simultaneously. If there is more than one auxiliary function belonging to the same family, these are performed together in a second stage, one per family, in the order in which they were programmed.

Auxiliary functions can only be executed if the CEFA signal is at logic level 1.

7.5.2. MOTION END FUNCTIONS

These functions are performed after the axis motion in the block in which they were programmed. The motion end auxiliary functions are M and H only if these are defined as "motion end" functions during configuration.

There are two types of motion end auxiliary functions:

- Common auxiliary functions;
- Calculation blocking auxiliary functions.

The common auxiliary functions are motion end functions which can only be executed if the CEFA signal is at logic level 1.

The calculation blocking auxiliary functions are motion end functions which can request axis motion from the system (MAS - system axis motion via logic). To permit the MAS to be executed, the logic must be synchronized with the system by means of the CEFA and CEFAB functions.

Motion end functions can only be executed if both CEFA and CEFAB are at 1. If the function requires a compensation change request, updating must take place before the function can be executed. To request compensation change, the auxiliary function must request "calculation blocking." To execute a MAS, the CEFAB signal must be held at 0 for the time required to complete the axis motion by the system. By setting CEFAB to 1 (and, in the case of functions requiring compensations, updating the new compensation), the next motion end auxiliary function can then be executed.

The CEFAB signal permits motion start and/or end functions to be programmed in an axis motion record from the system. The only condition is that the functions entered in the MAS must not repeat the calculation blocking request.

7.5.3. EXPEDITE FUNCTIONS

The expedite auxiliary functions are communicated to the logic in the form of two BCD figures after the motion start auxiliary functions, and last for the duration of the movement. In other words, if the expedite M function is programmed in the "continuous" blocks, it remains set until a non "continuous" block or a reset is programmed. These functions are sent to K rack locations different from those for motion start and end functions M and H. These functions are not affected by the status of the CEFA signal.

7.5.4. S FUNCTIONS

The S functions are motion start functions that are acceptable in hold also. They are used to determine spindle speed. These functions are expressed by means of 5 figures which are transferred to the logic in BCD format plus the FUAS strobe. The BGD figures remain latched.

Four K rack signals (SGAM1-SGAM4) are also associated with the S functions. These identify the range to which the programmed S function belongs. The fields defining the angular velocity rates for each range are defined in the characterization file. The reference is updated only after a "range entered" signal has been activated. The characterization file also defines the type of spindle handling in the AXCFIL file TPA parameter:

- Spindle handling with DC motor;
- Spindle handling without slaving.

When the spindle is handled with a DG motor, the spindle is driven directly with an analog reference proportional to the programmed S function. This reference, however, must be controlled by the logic through signals ROMAO (clockwise rotation) and ROMAA (anti-clockwise).

When the spindle is handled with an AC motor, only the 5 BCD figures relating to the programmed S function, and the impulsive FUAS strobe on two logic cycles, are transferred to the logic. The logic is also informed, by signals INVER and STOPR during the G84 and G86 cycles, when spindle rotation must be reversed or stopped. It should be remembered that for spindle control with an AC motor, the system must be characterized and the MT logic equatimns executed as for a DC motor with or without transducer.

7.5.5. T FUNCTIONS

Auxiliary T functions determine which tool is to work. They are executed at motion start and can be accepted in HOLD. T functions T are programmed in the T1234 format, with a maximum of four numbers after the T.

T functions can be managed as follows:

1) NORMAL MANAGEMENT;
2) RANDOM MANAGEMENT.

Both types of management provide the following options:

- TOOL LIFE AND INTEGRITY option;
- TOOL POCKET option.

With the RANDOM management there can also be a further option:

- TOOL IN UNLOAD option.

1) **NORMAL MANAGEMENT**

During normal management the interface does not execute any checks on the programmed T function. It simply sends the 4 BCD figures and the FUAT strobe to the logic. The BCD figures are stored whereas the FUAT strobe is impelling on two logic revolutions.

2) **RANDOM MANAGEMENT**

With random management, it is necessary to generate a file (FInRAN) by means of a special instruction (CRE). The file must be characterized in record FIL of the PGCFIL file and then edited.

The file editing operation consists in declaring the number of the tool or the number of the pocket (if this optional feature is also utilized) for each record.

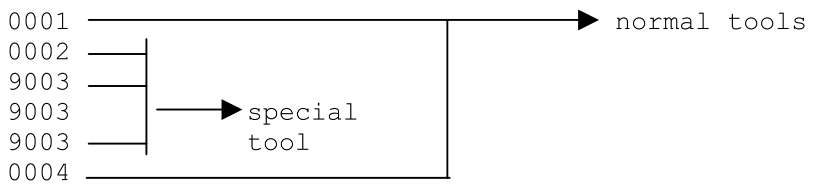
The position of the record identifies the number of the station of the crib whereas the number edited in the record identifies:

1 - the number of the tool
2 - the number of the pocket (if this option is used)

To each tool or pocket, the system will subsequently associate the corresponding station when the tool update is requested.

Both for "normal" tools and "special" tools (tools which occupy 3 stations), the format always has 4 digits.

Ex.:



With random management, the system executes a series of checks on the programmed T function. The checks establish the adequacy of the programmed T function and the necessity for a search in the tool crib.

In particular cases the programmed T function does not require a search in the tool crib, since the next tool will have to be loaded from the ground (manual tool change).

If a search is necessary, the BCD figures and the FUAT strobe are sent to the logic exactly as per the first type of management. On pack K are issued both the figures relating to the programmed T function and the figures relating to the station in which the programmed tool resides.

The random management is represented by the succession of the stages described in the following sections.

Offset change

The programming of a TX (TX = tool at present in the spindle) is tantamount to a booking for an offset change without any physical search. In this case the system informs the logic by sending the number equal to 0 to pack K. The logic is therefore not informed (pack K) about the station to search but only about the programmed T function and the FUAT strobe (or FUTKO).

Manual tool change

It is possible to program a manual tool change by programming a T function which is not contained in the FInRAN file. In this case signal CUMAN is set. The logic is informed (pack K) about the programmed T function (as well as about the FUAT strobe) but not about the station to search. The tool in the spindle must be manually unloaded by the user and is automatically deleted from the FInRAN file.

Normal tool change

If a T function present in the FInRAN file is programmed, the following information is sent to the logic:

- programmed T;
- station where the programmed T resides;
- function T strobe (FUAT).

N.B. After executing a manual tool change by programming a T function present in the FInRAN file and requesting the tool update, the tool previously loaded manually in the spindle is stored in the FInRAN file.

Loading of tools in the Crib

The loading of tools can be executed in two different ways:

1) Normally the operator manually loads the tools directly into the crib and declares the tools loaded, through the station-tool association in the FInRAN file.

2) The loading operation in the crib and the tool declaration in the FInRAN file is automatically executed through the sequence of operations described hereunder:

- Programming TxM6 (x = number of the tool to be loaded);

- The system forces manual CU (CUMAN), due to the fact that Tx is not present in the FInRAN file;

- The operator manually loads the tools in the spindles;

- Programming TOM6;
The programming of TOM6 causes the spindle to be emptied. In this way on pack K is issued the code corresponding to the first "empty" station of the crib and, when the tool update is requested, the tool previously loaded in the spindle is stored in the PInRAN file.
If the system sees that there are no "empty" stations in the crib, a new manual tool change message is forced.

- For other tools to be loaded, the same procedure applies.

Special tools

RANDOM MANAGEMENT offers the possibility of utilizing special tools i.e. tools which occupy 3 stations.

The system singles out a special tool if 4 figures, the first of which being 9, are programmed (it is possible to have 999 special tools). The system interrupts a tool change of the NORML TOOL $\longleftrightarrow$ SPECIAL TOOL type and displays the error message FILMS4 90. It is therefore essential that the tool present in the spindle be unloaded by programming TOM6, before requesting the change of the tool in the spindle with a tool of a different type.

TOOL LIFE AND INTEGRITY Option

Through this option, for each tool it is possible to declare a tool life time which allows to check the wear and tear and, therefore to avoid working with tools whose conditions are not perfect. It is also possible to request the replacement of the worn or broken tool with a good one of the same technological characteristics. The management of the tool life cycle is executed with the help of a table contained in a file which describes the characteristics of the tools (see the use and programming manual).

The system enables the tool life counter in the spindle during the intervals of time in which the CYCLE START is ON for movements in FEED and the SPINDLE is ROTATING. The logic can prevent the counter from working through the DITVU signal. The check for the tool integrity is completely managed by the system through cycle G74 described in the use and programming manual. If all the tools in the same family are worn or broken and the alternative tool of the last one is TO, when programming the main tool of this family, the system informs the logic about the number of tool programmed in BCD with the FUTKO strobe instead of the FUAT strobe.

Through this option, it can appear that the programmed tool is not the tool which should go in the spindle, since it can be replaced (when worn) by the alternative tool of the same family defined in the table. However, the system informs the logic not only about the code of the programmed tool but also about the code of the tool which will in fact have to go in the spindle.

TOOL IN UNLOAD Option

By characterizing value 0400 in the record CWP of the PGCFIL file, the option TOOL IN UNLOAD is enabled (this option is to be utilized only with the RANDOM MANAGEMENT of a crib and with tool change supplied with INTERMEDIATE STATION).

For the tool changes supplied with an intermediate station between the tool crib and the spindle, this option allows to execute a direct exchange (between intermediate station and spindle) of NORMAL tools $\longleftrightarrow$ SPECIAL tools without passing through the normal procedure which foresees the unloading of the tool present in the spindle, before programming the new tool.

This option therefore allows the exchange between "mixed" tools. In this case the error message FILMS4-90 will not be diagnosed and in the 25K connector are issued the figures that identify the position of the crib in which the tool present in the spindle will have to be unloaded.

7.5.6. M FUNCTIONS

These functions are expressed by two figures which are transferred to the logic in BCD format, plus the FUAM strobe. The features of each of these are defined in the characterization file by means of three bytes in hexadecimal format which have the meanings given below:

First Byte

BIT	**MEANING**
0	1 = Motion start function
1	1 = Motion end function
2	1 = Function acceptable in hold
3	1 = Function that must not be displayed
4	1 = Expedite M function
5	- Reserved -
6	1 = Modal function
7	1 = Function that continues to be displayed after reset

Second Byte

BIT	**MEANING**
0	- Not significant -
1	1 = Forces conditional semiauto
2	1 = Calculation block
3	1 = Forces semiauto
4	1 = Compensation change request
5	1 = Reset request at execution end
6	- Not significant -
7	- Not significant -

Third Byte

BIT	MEANING
0-3	Display class code
4-7	Stored search class code

The M functions must be characterized during initialization. If an M function that has not been defined in the characterization file is programmed, an error is indicated. Depending on how they are defined in the characterization file, these are performed either at motion start or motion end (synchronized by the CEFA signal) or as expedite functions.

If defined as motion start or motion end functions, they will be sent to the logic as two BCD digits plus an impulsive FUAM, the minimum duration of which is two logic turns. If dmfined as expedite functions, they will be sent to the logic as two BCD digits that will remain stored all through the execution of the axis motion block in which they have been programmed. Up to four motion start and end M functions can be programmed, plus an expedite M function, in the same block.

7.5.7. H FUNCTIONS

The H functions are a category of auxiliary functions that are completely at the user's disposal. The feature of each of thesa functions is defined in the characterization fiie by means of two bytes which have the meanings given below.

First Byte

BIT	MEANING
0	1 = Motion start function
1	1 = Motion end function
2	1 = Function acceptable in hold
3	1 = Function that must not be displayed
4	- Reserved -
5	1 = Expedite H function
6	1 = Modal function
7	1 = Function that must continue to be displayed after reset

Second Byte

BIT	MEANING
0-3	Display class code
4-7	Stored search class code

The H functions must be characterized during initialization. If an H function that is not defined in the charactmrization file is programmed, an error is indicated. Depending on how they are defined in the characterization file, they are executed at motion start or end (synchronized with the CEFA signal), or as expedite functions.

If defined as motion start or motion end functions, they will be sent to the logic as two BCD digits plus an impulsive FUAM, the minimum duration of which is two logic turns. If dafined as expedite functions, they will be sent to the logic as two BCD digits that will remain stored all through the execution of the axis motion block in which they have been programmed. Up to 4 motion start or end H functions plus one expedite H function can be programmed in the same block.

7.5.8. INDEXING AXIS FUNCTIONS

These are auxiliary motion start functions that are not acceptable in hold. Up to three indexing axes can be handled in the same block. The logic is informed of the indexing axis programming by means of eight BCD figures in 5.3 format, which are common for all three axes, plus strobes TASCl, TASC2 and TASC3, which select the axis to be moved. The BCD figures are latched, while the strobe is impulsive on two logic cycles. The SESC signal is common to the three axes and defines the sign programmed for the axis that is currently in motion (this signal is only active if the axis has been defined as incremental). Each axis can be configured in either absolute or incremental mode.

In absolute mode, the position to be reached is always supplied; this gives 99999.999 distinct positions.

In incremental mode, the number of displacements to be performed and the programmed sign are always supplied; this gives increments of 99999.999 displacements.

The first programming operation after control unit switch-on must define an axis zeroing. Position 0000 must therefore be programmed, or the system must be informed that zeroing has been completed by means of signals MIZE1, MIZE2, and MIZE3, which must always remain active (otherwise an error is indicated).

7.6. LOGIC REQUESTS

The PLC interface provides the machine logic with a certain number of cycles in order to increase the programmer's operational capabilities. The cycles can be activated by means of request signals. The request signals can be divided into two categories:

- Asynchronous Requests (accepted at any time);
- Synchronous Requests (accepted in particular conditions).

The following asynchronous requests are possible:

- Updating of spindle reference;
- Point-to-point axis positioning;
- Spindle updating;
- Message display;
- Reference forcing on D/A converter channel;
- Console handling from machine logic;
- Spindle switching.

The following synchronous requests are possible:

- Axis motion file record execution from system;
- Program selection;
- Program stop request;
- Cycle start request;
- Program jump from logic.

Important. All the asynchronous requests from the machine logic will be accepted by the system at any moment. They will be executed after the last equation written in the low logic has been executed.

7.6.1. UPDATING OF SPINDLE REFERENCE

The NC-110 system provides an analog reference voltage that can be used in order to drive a DC motor. The process is capable of providing the analog reference voltage level, moment by moment, with the help of suitable settings and information requests provided by the logic. To this purpose, four "range entered" signals (GAMMA1, GAMM2, GAMM3, GAMM4) are used in order to evaluate the currently applicable transmission ratio, and five signals which enable the reference output in different ways.

The five reference output enable signals are listed below.

ANGOM - The spindle is transduced using a resolver or an encoder (ratio 1:1) and is positioned using the acceleration, speed, gain and positioning tolerance parameters defined during configuration. The spindle is kept in position until the request is terminated.

FOMAO - FOMAA - A reference voltage based on two BCD figures provided by the logic is generated. The two figures express the percentage with respect to the maximum level required on output. The maximum possible value is 99%.

ROMAO - ROMAA - A reference voltage proportional to the programmed S, and in the range declared by the logic, is generated. The interface informs the logic which range the programmed S belongs to, by means of signals SGAMM1, SGAMM2, SGAMM3 and SGAMM4.

Should requests be issued simultaneously, the following order of priority is respected: ANGOM, FOMAO, FOMAA, ROMAO, ROMAA).

7.6.2. SPINDLE AXIS SWITCHING

From the machine logic it is possible to make a request for sequentially switching between the spindle axes previously defined in the AXCFIL file.

To request spindle switching, the name of the new spindle to be slaved must be declared in W11K3, in ASCII code.

If the switching request is not accepted, the system sets NCKCM to 1. NCKCM remains in this status until the request that generated it is reset.

If the request is accepted, the system goes into the WAIT status for two logic cycles and sets ACKCM to zero. After the switching has been completed, ACKCM goes to 1 and the "new" spindle will be activated with all the characteristics (speed, sense of rotation) of the "old" one. In turn, the "old" spindle will go on rotating at the same speed and in the same sense as before. Nevertheless, it cannot be used for special machining cycles (canned cycles, etc.).

Notes:

• If no code is written in the W11K3 word at power-up, the system will slave the spindle that has been declared in the ASM record during configuration and manage it as the main spindle.

• In case of reset, the system resets the zero analog reference and stops the rotation of both the spindle written in W11K3 and the spindle declareed in the ASM record.

• Since subsequent rotating spindles can be stopped by subsequent reset cycles, the machine logic must decide if the old spindle must be stopped by setting the rotation requests (ROMAO, ROMAA, etc.) to zero before switching to the new spindle.

7.6.3. POINT-TO-POINT AXIS POSITIONING

The NC-110 system provides the logic with two cycles for positioning eight point-to-point axes (two simultaneous). The axes can be positioned in a specified number of stations equally spaced along the axis (99999.999).

The request is made by means of two distinct words, which identify which axis pair must be moved. When a request has been activated on one of two software channels, the system informs the logic that the channel is busy, throughout the time required for positioning, by means of signals BUSY1 and BUSY2, for the first and second channels, respectively.

If while one axis is being positioned a positioning request is activated for another axis on the same channel, the system aborts the request and indicates an error via signals KOSY1 e KOSY2. In point-to-point axis positioning, the system uses settings performed during configuration, in addition to the requests from logic.

The cycle consists of the following stages:

- Activation of the BUSY signal for the busy channel.

- Check to determine whether or not the home pomition has been set; if not, a "machine home" cycle is activated, using the home position direction, speed and address values defined during configuration (if the axis is not absolute or is absolute with encoder).

- If direction has not been forced by the logic using signals FOPA and FONA, the system decides on the direction of motion by reading the transducer, after which it informs the logic using signals ROPO and RONE. Obviously, signals FOPA and FONA are only significant if the axis is cyclic (rotary).

- Wait period, until the first point defined during configuration has been passed.

- Activation of the ROLE signal (first deceleration).

- Wait period, until the second point defined during configuration has been passed.

- Activation of the ROLLE signal (second deceleration) and ROLE reset.

- Wait period, until the axis movs within the tolerance limits set during configuration.

- Activation of the axis signal in POSI position and ROLLE reset.

- Check on request reset.

- POSI and BUSY reset.

Note. If the axis motor has been defined as a non-DC motor during configuration, and the analog reference voltage is therefore not used, the logic must home the axis and position it correctly, using the direction, deceleration and positioning signals.

7.6.4. UPDATE OP TOOL OFFSET

The logic can request the system to update and display the tool in the spindle and the related offset, using the AGTOL signal. Depending on the result of this operation, the CU replies with either ACKTO (tool update performed) or NCKTO (tool update request not congruent).

7.6.5. MESSAGE DISPLAY

You can display as many as 255 messaqes per process. These messages must be resident in the FInMSG file (where n stands for the process number) that has been previously created, edited and confiqured in record FLC from the PGCFIL file.
Two display commands are allowable:

- By means of connectors 21K-47K-73K-99K-125K and 22K-48K-73K-100K-126K. Using this method, you can display the first 64 messages edited in the FInMSG file on the fifth line of the screen. These messages will appear one at a time, with decreasing priority.

- By means of words W17K3-W43K3-W69K3-W95K3-W121K3. Using this method, all 255 messages in the FInMSG file will be displayed on the fourth line of the screen. These messages will appear one at a time, with decreasing priority.

Examples:

```
U21K3=I00A1
```

When I00A1 is at 1, the message contained in record 4 of the FI1MS5 file is displayed.

```
W17K3=MUX(1,13,54,253),(I00A1,I00A2,I00A3,I00A4)
```

When signals I0A1, I0A2, I0A3 or I0A4 go to 1, messages 1, 13, 54 and 253 of file FIlMSG will be displayed on the fourth line.

7.6.6. FORCING A VOLTAGE ON A D/A CONVERTER CHANNEL

The system forces a reference voltage on three possible D/A analog channels. These correspond to three corresponding point-to-point axes that are declared during configuration. One channel at a time can be slaved by combining the two request signals FORZ1 and FORZ2, and by using the FORZN signal, which requests that the reference voltage value be negative. For example:

FORZ1	FORZ2	MEANING
1	0	voltage forced on channel assigned to axis 1
0	1	voltage forced on channel assigned to axis 2
1	1	voltage forced on channel assigned to axis 3

In addition to selecting the channel, the logic must define the voltage value by means of words W13K2 and W13K3.

7.6.7. CONSOLE CONTROL BY LOGIC

The NC-110 system allows the console to be handled by the machine logic using suitable requests to the K rack. In other words, all activities performed by the console can be removed from the operator's control and assigned by the logic. These activities are listed below:

- Operating mode selection;

- Axes selection;

- Assignment of manual JOG selector value and direction;

- Cycle start request;

- Assiqnment of manual feedrate selector;

- Assiqnment of speed override selector.

The console can be disabled partially, by setting the individual W15K0 signals. Alternatively, it can be disabled completely, by setting the word in its entirety.

Note. Before the cycle start request is generated, the console should be configured in accordance with the request to be executed. For further details on console handling by the MT loqic, refer to the description of the signals on connectors W15K and W16K.

7.6.8. AXIS MOTION FILE RECORD EXECUTION FROM SYSTEM

The logic can request that one of the 255 records available in the axis motion from system file be executed (see characterization PGCFIL), by assigning the W12K1 in binary code. The W12K1 identifies the number of the record to be executed. The request is accepted durinq execution of a motion end M function with the "calculation blockinq" feature (see the description of the M functions) and with CEFA = 1 and CEFAB = 0.

In the axis motion from system file, axis motion, preliminary G functions, M, S, T and indexing axis functions, etc., can be edited as a normal part program, except the M function with the "calculation blocking" feature. When the record being executed has terminated, the system sets the POSIA signal, which remains active until the request has been executed. It is advisable to enter G79 in the axis motion record (positioning at an absolute value with reference to the home position).

7.6.9. SELECTING PROGRAM OR COMMAND FROM KEYBOARD

The logic may request that a part program be selected, or that a command named in the FILCMD file be either issued from keyboard, or written in ASCII, starting from a connector of the K rack declared.

In the first case, a file linked with logical name FILCMD in FCRSYS must be edited. The name and the device (one for each record) of the possible part programs to be selected, or the function to be assigned (for example, URL=1; E8=40) must then be entered in the available records, by means of the EDI directive.

In both cases, if the request is accepted, the system informs the logic that the operation has been completed successfully, by means of the ACKSPC acknowledge. If the request is not accepted, it sets the NKSPC signal (request not executed).

For more information about the modality and sequencing with which the loqic executes a program selection or a command from keyboard, refer to the explanatory note for signals SPCCOM-CMDLOG-FILCMD in Chapter 8.

7.6.10. CYCLE START REQUEST

The logic may request execution of a program previously selected by means of SPG, using the CYST signal. While the request is being considered, the system is in stand-by status. For further details, refer to the section describing CYCLE START ACTIVATION AND USE.

7.7. INTERFACE CYCLE FLOW-CHARTS

This section of the chapter contains copies of the flow-charts relating to the cycles described in the previous sections. The names of the flow-charts are the same as those of the related cycles.

8. INTERFACE SIGNALS

This chapter describes the signals and words belonging to the first 26 connectors in the K rack for each process. These connectors and relating signals are common to the five processes that can be configured on the NC-110 system. This manual specifically describes the connectors of the first process, but this description applies equally to the other four processes. For the numbering of the first 26 connectors in all the processes, refer to Table 3.1.

The first part of the chapter consists of the connector layouts that indicate the names and functions of the various signals.

The second part of the chapter gives a functional description of the individual signals and words on the connectors.

8.1. SIGNAL LAYOUTS

The signal layouts identify the position and function of the interface signals in the K rack. Each layout gives the numbering of the words and signals on a connector. The mnemonic name (if it exists) and a brief functional description is provided for each signal.

The number of the connector relating to the first process is written at the top of each layout. The numbers of the connectors that correspond to the other four processes are given in brackets and refer to the second, third, fourth and fifth process, respectively.

Blank layout sheets, similar to the layouts described here, are enclosed at the end of the manual for the user to record the signals he defines himself in the machine logic.

I00K LAYOUT

INPUT SIGNALS Connector IOOK (26K-52K-78K-1O4K)			
WORD	SIG		
0	0	EMERG	process in emergency status
	1	RESE	reset cycle in progress
	2	CONP	process ready to slave the axes
	3	CYCLE	process in block execution
	4	STABY	process in stand-by
	5		reserved
	6	RCM	stored search in progress
	7	CYON	cycle start switched on
1	8	ABI1	axis 1 slaving enabled
	9	ABI2	axis 2 slaving enabled
	10	ABI3	axis 3 slaving enabled
	11	ABI4	axis 4 slaving enabled
	12	ABI5	axis 5 slaving enabled
	13	ABI6	axis 6 slaving enabled
	14	ABI7	axis 7 slaving enabled
	15	ABI8	axis 8 slaving enabled
2	16	MOV1	axis 1 motion request
	17	MOV2	axis 2 motion request
	18	MOV3	axis 3 motion request
	19	MOV4	axis 4 motion request
	20	MOV5	axis 5 motion request
	21	MOV6	axis 6 motion request
	22	MOV7	axis 7 motion request
	23	MOV8	axis 8 motion request
3	24	POSIA	MAS axes motion record executed
	25	POSIM	spindle positioning performed
	26	NCKTO	T update on spindle failed
	27	ACKTO	T update on spindle performed
	28	HOLDA	control in HOLD status
	29	ACKCM	spindle axis switching performed
	30	NCKCM	spindle axis switching incorrect
	31	DIRMN	manual axis motion in minus direction

I01K LAYOUT

INPUT SIGNALS **Connector I01K (27K-53K-79K-105K)**		
WORD	SIG	
0	0 1 2 3	thousandths
	4 5 6 7	hundredths
1	8 9 10 11	tenths
	12 13 14 15	INDEXING AXIS units
2	16 17 18 19	LATCHED IN BCD (5.3) tens
	20 21 22 23	hundreds
3	24 25 26 27	thousands
	28 29 30 31	tens of thousands

I02K LAYOUT

INPUT SIGNALS

Connector I02K (28K-54K-80K-106K)

WORD	SIG		
0	0 1 2 3		units
	4 5 6 7	T POSITION TO BE SOUGHT, LATCHED	tens
1	8 9 10 11	IN BCD (4 FIGURES) (FOR RANDOM MANAGEMENT ONLY)	hundredsi
	12 13 14 15		thousands
2	16 17 18 19		unitsi
	20 21 22 23	T FUNCTION TO BE SOUGHT TO SPINDLE, LATCHED IN BCD (4 FIGURES)	tens
3	24 25 26 27	(FOR NON-RANDON MANAGEMENT ONLY)	hundreds
	28 29 30 31		thousands

i

I03K LAYOUT

INPUT SIGNALS **Connector I03K (29K-55K-81K-107K)**			
WORD	SIG		
0	0 1 2 3	IMPULSIVE M FUNCTION (MOTION START OR END)	units
	4 5 6 7	ON 2 LOGIC CYCLES (2 figures)	tens
1	8 9 10 11	EXPEDITE M FUNCTION	units
	12 13 14 15	LATCHED IN THE BLOCK IN BCD (2 figures)	tens
2	16 17 18 19	IMPULSIVE H FUNCTION (MOTION START OR END)	units
	20 21 22 23	ON 2 LOGIC CYCLES (2 figures)	tens
3	24 25 26 27	EXPEDITE H FUNCTION	units
	28 29 30 31	LATCHED IN THE BLOCK IN BCD (2 figures)	tens

I04K LAYOUT

INPUT SIGNALS Connector I04K (30K-56K-82K-108K)		
WORD	SIG	
0	0 1 2 3	units
	4 5 6 7	tens T FUNCTION PROGRAMMED
1	8 9 10 11	LATCHED IN BCD (4 FIGURES) hundreds
	12 13 14 15	thousands
2	16 17 18 19 20 21 22 23	FUAS S function strobe FUAT T function strobe FUAM M function strobe FUAH H function strobe TASC1 indexing axis 1 function strobe TASC2 indexing axis 2 function strobe TASC3 indexing axis 3 function strobe FUTKO T function strobe, tool life expired
3	24 25 26 27 28 29 30 31	SESC indexing axis, negative programmed sign MPROFI axes actually in contour CUMAN manual tool change (random handling) PWMAN spindle rotation request SOFIT air flow request GOMAN spindle rotation in progress ACKCY cycle start request executed NCKCY cycle start request not accepted

I05K LAYOUT

INPUT SIGNALS

Connector I05K (31K-57K-83K-109K)

WORD	SIG	
0	0	FG00 Rapid traverse control
	1	PROFIT Contour motion control
	2	MISU Fixed measurement cycle control
	3	INTUT Tool integrity cycle in progress
	4	FG8.1 ┐
	5	FG8.2 ├ Standard fixed cycle coding
	6	FG8.4 │
	7	FG8.8 ┘
1	8	ASRI1 axis 1 homed
	9	ASRI2 " 2 "
	10	ASRI3 " 3 "
	11	ASRI4 " 4 "
	12	ASRI5 " 5 "
	13	ASRI6 " 6 "
	14	ASRI7 " 7 "
	15	ASRI8 " 8 "
2	16	ROPO1 point-to-point axis 1 positive motion
	17	ROPO2 point-to-point axis 2 positive motion
	18	RONE1 point-to-point axis 1 negative motion
	19	RONE2 point-to-point axis 2 negative motion
	20	ROLE1 point-to-point axis 1 slow motion
	21	ROLE2 point-to-point axis 2 slow motion
	22	ROLLE1 point-to-point axis 1 very slow motion
	23	ROLLE2 point-to-point axis 2 very slow motion
3	24	POSI1 point-to-point axis 1 in position
	25	POSI2 point-to-point axis 2 in position
	26	BUSY1 channel 1 busy
	27	BUSY2 channel 2 busy
	28	KOSI1 blocking error on channel 1
	29	KOSI2 blocking error on channel 2
	30	ACKSPC spg or keyboard command executed
	31	NCKSPC spg or keyboard command not executed

I06K LAYOUT

INPUT SIGNALS Connector I06K (32K-58K-84K-110K)			
WORD	SIG		
0	0	COM1 axis 1 switch executed	
	1	COM2 axis 2 switch executed	
	2	COM3 axis 3 switch executed	
	3	COM4 axis 4 switch executed	
	4	COM5 axis 5 switch executed	
	5	COM6 axis 6 switch executed	
	6	COM7 axis 7 switch executed	
	7	COM8 axis 9 switch executed	
1	8		
	9		
	10		
	11		
	12		
	13		
	14		
	15		
2	16	SGAM1 programmed S belongs to range 1	
	17	SGAM2 programmed S belongs to range 2	
	18	SGAM3 programmed S belongs to range 3	
	19	SGAM4 programmed S belongs to rangm 4	
	20	RETRA retrace mode	
	21		
	22		
	23	ABIM Spindle active and enabled	
3	24	Type of coded emergency (binary)	Emergency code
	25		
	26		
	27		
	28		Axis index
	29		
	30		
	31		

I07K LAYOUT

INPUT SIGNALS
Connector I07K (33K-59K-85K-111K)

WORD	SIG		
	0		
	1		
	2		units
	3		
0	4		
	5		
	6		tens
	7	BCD FIGURES ASSOCIATED	
	8	WITH THE ASCII COMPARATORS	
	9		
	10	(4 FIGURES)	hundreds
	11		
1	12		
	13		thousands
	14		
	15		
	16		
	17		
	18		units
	19		
2	20		
	21		tens
	22		
	23	ACTIVE OFFSET	
	24	IN BCD (4 FIGURES)	
	25		
	26		hundreds
	27		
3	28		
	29		thousands
	30		
	31		

I08K LAYOUT

INPUT SIGNALS

Connector I08K (34K-60K-86K-112K)

WORD	SIG		
0	0 1 2 3		units
	4 5 6 7	PROGRAMMED S FUNCTION	tens
1	8 9 10 11	LATCHED IN BCD (5 FIGURES)	hundreds
	12 13 14 15		thousands
2	16 17 18 19		tens of thousands
	20 21 22 23		
3	24	EMDI	manual data input (MDI)
	25	AUTO	control in automatic mode
	26	SEMI	control in semi-automatic mode
	27	MANUC	control in continuous manual mode
	28	MANUJ	control in JOG manual mode
	29	RIPRO	return to profile
	30	RIMZE	home pomition mearch
	31	RESET	selector in Reset status

I09K LAYOUT

INPUT SIGNALS Connector I09K (35K-61K-87K-113K)		
WORD	SIG	
0	0 1 2 3 4 5 6 7	(FILMS 4)
1	8 9 10 11 12 13 14 15	BILA1 balance signal for axis 1 BILA2 balance signal for axis 2 BILA3 balance signal for axis 3 BILA4 balance signal for axis 4 BILA5 balance signsl for axis 5 BILA6 balance signal for axis 6 BILA7 balance signal for axis 7 BILA8 balance signal for axis 8
2	16 17 18 19 20 21 22 23	
3	24 25 26 27 28 29 30 31	INVER spindle inversion STOPR spindle stop

U10K LAYOUT

OUTPUT SIGNALS Connector U10K (36K-62K-88K-114K)		
WORD	SIG	
0	0	MUSPE machine tool off
	1	REAZ reset request
	2	HLDR request for hold with restart enable
	3	RHOE request for hold with automatic restore
	4	CYST cycle start request
	5	FOLD feed hold request
	6	WAIC siqnal for expedite COMU
	7	RISPE machine tool switch-off request
1	8	RABI1
	9	RABI2
	10	RABI3
	11	RABI4 SLAVING ENABLE REQUEST
	12	RABI5
	13	RABI6 FOR AXES 1 TO 8
	14	RABI7
	15	RABI8
2	16	REGTOL spindle tool reset request
	17	DITVU tool life count disable
	18	SPCCOM keyboard command SPG request
	19	CMDLOG keyboard command activation
	20	
	21	AGTOL spindle tool update request
	22	
	23	FILCMD SPG or keyb. command from file or K rack
3	24	COMU motion enable
	25	CEFA auxiliary function "emission"enable
	26	CEFAB aux. func. emiss. enable in calc. Blocking
	27	MIZE1 home position for indexing axis 1
	28	MIZE2 home position for indexing axis 2
	29	MIZE3 home position for indexing axis 3
	30	RMORE retrace operation request
	31	

U11K LAYOUT

OUTPUT SIGNALS

Connector U11K (37K-63K-89K-115K)

WORD	SIG	
	0	ANGOM spindle orientation
	1	FOMAO forcing of clockwise rotation
	2	FOMAA forcing of anticlockwise rotation
	3	ROMAO clockwise rotation request
0	4	ROMAA anticlockwise rotation request
	5	FORID forced division by 10
	6	
	7	
	8	
	9	tenths of volt
	10	FORCED VOLTAGE
	11	
1	12	
	13	WITH FOMAO OR FOMAA
	14	volt
	15	
	16	GAMM1
	17	GAMM2 RANGE CURRENTLY ENABLED
	18	GAMM3
	19	GAMM4
2	20	
	21	
	22	
	23	
	24	
	25	
	26	
	27	SELECTED SPINDLE AXIS NAME (ASCII)
3	28	
	29	
	30	
	31	

U12K LAYOUT

OUTPUT SIGNALS Connector U12K (38K-64K-90K-116K)		
WORD	SIG	
0	0	FORZ1 number of axis where forcing is to occur
	1	FORZ2 number of axis where forcing is to occur
	2	FORZN negative reference on converter
	3	
	4	FOPA1 force cyclic axis 1 in positive direction
	5	FOPA2 force cyclic axis 2 in positive direction
	6	FONA1 force cyclic axis 1 in negative direction
	7	FONA2 force cyclic axis 2 in negative direction
1	8 9 10 11 12 13 14 15	MOTION FILE RECORD NUMBER REQUESTED FROM LOGIC: 0 = No motion required 1-255 = Number of record
2	16	DI1+ axis 1 positive travel limits disable
	17	DI1- axis 1 negative travel limits disable
	18	DI2+ axis 2 negative travel limitm disable
	19	DI2- axis 2 negative travel limits disable
	20	DI3+ axis 3 positive travel limits disable
	21	DI3- axis 3 negative travel limits disable
	22	DI4+ axis 4 positive travel limits disable
	23	DI4- axis 4 negative travel limits disable
3	24	DI5+ axis 5 positive travel limits disable
	25	DI5- axis 5 negative travel limits disable
	26	DI6+ axis 6 positive travel limits disable
	27	DI6- axis 6 negative travel limits disable
	28	DI7+ axis 7 positive travel limits disable
	29	DI7- axis 7 negative travel limits disable
	30	DI8+ axis 8 positive travel limits disable
	31	DI8- axis 8 negative travel limits disable

U13K LAYOUT

OUTPUT SIGNALS

Connector U13K (39K-65K-91K-117K)

WORD	SIG		
0	0 1 2 3 4 5 6 7	ARESE request for active reset	
1	8 9 10 11 12 13 14 15		
2	16 17 18 19		millivolts
	20 21 22 23	VOLTAGE SET	hundredths of a volt
3	24 25 26 27	ON THE D/A CONVERTER CHANNEL	tenths of a volt
	28 29 30 31		VOLT

U14K LAYOUT

OUTPUT SIGNALS Connector U14K (40K-66K-92K-118K)		
WORD	SIG	
0	0 1 2 3	thousandths
	4 5 6 7	hundredths
1	8 9 10 11	tenths
	12 13 14 15	SPINDLE ORIENTATION DISPLACEMENT units
2	16 17 18 19	IN DEGREES (5.3 BCD FORMAT) tens
	20 21 22 23	hundreds
3	24 25 26 27	thousands
	28 29 30 31	hundreds of thousands

U15K LAYOUT

OUTPUT SIGNALS Connector U15K (41K-67K-93K-119K)		
WORD	SIG	
6	0	cycle start disable on console
	1	
	2	mode select disable on console
	3	manual feed disable on console
	4	feed override disable on console
	5	spindle override disable on console
	6	axis selection disable on console
	7	JOG selection disable on console
1	8	MDI mode selection
	9	automatic mode selection
	10	semi-automatic mode selection
	11	continuous manual mode selection
	12	JOG manual mode selection
	13	return to profile mode select
	14	home position search mode selection
	15	reset mode selection
2	16	axis 1 selection
	17	axis 2 selection
	18	axis 3 selection
	19	axis 4 selection
	20	axis 5 selection
	21	axis 6 selection
	22	axis 7 selection
	23	axis 8 selection
3	24	manual feed percentage 1%
	25	manual feed percentage 2%
	26	manual feed percentage 4%
	27	manual feed percentage 8%
	28	manual feed percentage 16%
	29	manual feed percentage 32%
	30	manual feed percentage 64%
	31	FDILO Negative axis direction from logic (C.L)

i

U16K LAYOUT

OUTPUT SIGNALS Connector U16K (42K-68K-94K-120K)		
WORD	SIG	
0	0	feed override percentage 1%
	1	feed override percentage 2%
	2	feed override percentage 4%
	3	feed override percentage 8%
	4	feed override percentage 16%
	5	feed override percentage 32%
	6	feed override percentage 64%
	7	
1	8	spindle override percentage 1%
	9	spindle override percentage 2%
	10	spindle override percentage 4%
	11	spindle override percentage 8%
	12	spindle override percentage 16%
	13	spindle override percentage 32%
	14	spindle override percentage 64%
	15	
2	16	JOG length 0.001)
	17	JOG length 0.01 (mm
	18	JOG length 0.1)
	19	JOG length 1 (or
	20	JOG length 10)
	21	JOG length 100 (inches
	22	
	23	complete JOG execution request
3	24	
	25	
	26	
	27	
	28	
	29	
	30	
	31	

U17K LAYOUT

OUTPUT SIGNALS Connector U17K (43K-69K-95K-121K)		
WORD	SIG	
0	0 1 2 3 4 5 6 7	if FILCMD=0 number (hexadec.) of the first ten connectors in rack K that stores the name of the program selected from logic if FILCMD=1 number of the record (hexadec.) that contains the name of the pro-gram selected from logic
1	8 9 10 11 12 13 14 15	RCOM1 axis 1 switch request RCOM2 axis 2 switch request RCOM3 axis 3 switch request RCOM4 axis 4 switch request RCOM5 axis 5 switch request RCOM6 axis 6 switch request RCOM7 axis 7 switch request RCOM8 axis 8 switch request
2	16 17 18 19 20 21 22 23	
3	24 25 26 27 28 29 30 31	NUMBER OF MESSAGE TO BE DISPLAYED CODED IN BINARY

U18K LAYOUT

INPUT SIGNALS

Connector U18K (44K-70K-96K-122K)

WORD	SIG		
0	0 1 2 3		thousandths
	4 5 6 7		hundredths
1	8 9 10 11		tenths
	12 13 14 15	STATION REQUESTED	units
2	16 17 18 19	FOR POINT-TO-POINT AXIS 1 BCD (5.3)	tens
	20 21 22 23		hundreds
3	24 25 26 27		thousands
	28 29 30 31		tens of thousands

U19K LAYOUT

INPUT SIGNALS **Connector U19K (45K-71K-97K-123K)**			
WORD	SIG		
0	0 1 2 3		thousandths
	4 5 6 7		hundredths
1	8 9 10 11		tenths
	12 13 14 15	STATION REQUESTED	units
2	16 17 18 19	FOR POINT-TO-POINT AXIS 2 BCD (5.3)	tens
	20 21 22 23		hundreds
3	24 25 26 27		thousands
	28 29 30 31		tens of thousands

U20K LAYOUT

OUTPUT SIGNALS Connector U20K (46K-72K-98K-124K)		
WORD	SIG	
0	0 1 2 3 4 5 6 7	BINARY CODING OF AXIS TO BE MOVED POSITIONING REQUEST AS POINT-TO-POINT AXIS 1 (0 = AXIS NOT ACTIVE)
1	8 9 10 11 12 13 14 15	BINARY CODING OF AXIS TO BE MOVED POSITIONING REQUEST AS POINT-TO-POINT AXIS 2 (0 = AXIS NOT ACTIVE)
2	16 17 18 19 20 21 22 23	feedrate percent. for point-to-point axis 1 1% feedrate percent. for point-to-point axis 1 2% feedrate percent. for point-to-point axis 1 4% feedrate percent. for point-to-point axis 1 8% feedrate percent. for point-to-point axis 1 16% feedrate percent. for point-to-point axis 1 32% feedrate percent. for point-to-point axis 1 64%
3	24 25 26 27 28 29 30 31	feedrate percent. for point-to-point axis 2 1% feedrate percent. for point-to-point axis 2 2% feedrate percent. for point-to-point axis 2 4% feedrate percent. for point-to-point axis 2 8% feedrate percent. for point-to-point axis 2 16% feedrate percent. for point-to-point axis 2 32% feedrate percent. for point-to-point axis 2 64%

U21K LAYOUT

OUTPUT SIGNALS Connector U21K (47K-73K-99K-125K)		
WORD	SIG	
0	0	MSG1 message 1 display command
	1	MSG2 " 2 " "
	2	MSG3 " 3 " "
	3	MSG4 " 4 " "
	4	MSG5 " 5 " "
	5	MSG6 " 6 " "
	6	MSG7 " 7 " "
	7	MSG8 " 8 " "
1	8	MSG9 " 9 " "
	9	MSG10 " 10 " "
	10	MSG11 " 11 " "
	11	MSG12 " 12 " "
	12	MSG13 " 13 " "
	13	MSG14 " 14 " "
	14	MSG15 " 15 " "
	15	MSG16 " 16 " "
2	16	MSG17 " 17 " "
	17	MSG18 " 18 " "
	18	MSG19 " 19 " "
	19	MSG20 " 20 " "
	20	MSG21 " 21 " "
	21	MSG22 " 22 " "
	22	MSG23 " 23 " "
	23	MSG24 " 24 " "
3	24	MSG25 " 25 " "
	25	MSG26 " 26 " "
	26	MSG27 " 27 " "
	27	MSG28 " 28 " "
	28	MSG29 " 29 " "
	29	MSG30 " 30 " "
	30	MSG31 " 31 " "
	31	MSG32 " 32 " "

U22K LAYOUT

OUTPUT SIGNALS Connector U22K (48K-74K-100K-126K)		
WORD	SIG	
0	0	MSG33 message 33 display command
	1	MSG34 " 34 " "
	2	MSG35 " 35 " "
	3	MSG36 " 36 " "
	4	MSG37 " 37 " "
	5	MSG38 " 38 " "
	6	MSG39 " 39 " "
	7	MSG40 " 40 " "
1	8	MSG41 " 41 " "
	9	MSG42 " 42 " "
	10	MSG43 " 43 " "
	11	MSG44 " 44 " "
	12	MSG45 " 45 " "
	13	MSG46 " 46 " "
	14	MSG47 " 47 " "
	15	MSG48 " 48 " "
2	16	MSG49 " 49 " "
	17	MSG50 " 50 " "
	18	MSG51 " 51 " "
	19	MSG52 " 52 " "
	20	MSG53 " 53 " "
	21	MSG54 " 54 " "
	22	MSG55 " 55 " "
	23	MSG56 " 56 " "
3	24	MSG57 " 57 " "
	25	MSG58 " 58 " "
	26	MSG59 " 59 " "
	27	MSG60 " 60 " "
	28	MSG61 " 61 " "
	29	MSG62 " 62 " "
	30	MSG63 " 63 " "
	31	MSG64 " 64 " "

U23K LAYOUT

OUTPUT SIGNALS Connector U23K (49K-75K-101K-127K)		
WORD	SIG	
0	0 1 2 3 4 5 6 7	ASCII CODE ASSOCIATED TO THE DISPLAY OF THE FIRST BCD OF THE FIRST FIELD
1	8 9 10 11	UNITS
	12 13 14 15	TENS
		BCD VALUE OF FIRST FIELD (4 FIGURES)
2	16 17 18 19	HUNDREDS
	20 21 22 23	THOUSANDS
3	24 25 26 27 28 29 30 31	ASCII CODE ASSOCIATED TO THE DISPLAY OF THE SECOND BCD OF THE SECOND FIELD

U24K LAYOUT

OUTPUT SIGNALS Connector U24K (50K-76K-102K-128K)		
WORD	SIG	
0	0 1 2 3	units
	4 5 6 7	tens
		BCD VALUE OF SECOND FIELD (4 FIGURES)
1	8 9 10 11	hundreds
	12 13 14 15	thousands
2	16 17 18 19 20 21 22 23	ASCII CODE ASSOCIATED TO THE DISPLAY OF THE BINARY VALUE OF THE THIRD FIELD
3	24 25 26 27	units
		BINARY VALUE OF THIRD FIELD
	28 29 30 31	tens

U25K LAYOUT

OUTPUT SIGNALS

Connector U25K (51K-77K-103K-129K)

WORD	SIG		
0	0 1 2 3	TOOL DISCHARGE POSITION	units
	4 5 6 7	LATCHED IN BCD (4 FIGURES)	tens
1	8 9 10 11	(RANDOM TOOL MANAGEMENT ONLY)	hundreds
	12 13 14 15		thousands
2	16 17 18 19 20 21 22 23	CONTROL (%) OF SPINDLE IN G84 (FOR DC SPINDLE WITH TRANSDUCER ONLY)	
3	24 25 26 27 28 29 30 31		

8.2. CONNECTOR SIGNALS

This section of the chapter gives a functional description of the signals and the words relating to the first 26 connectors of the K rack of each process. The variables described here are used by the interface in handling the process control operations described in Chapter 7.

The definition of input and output signals is related to the interface, in the sense that the inputs enter the interface and the outputs emerge from it. Connectors 0 to 9 (for the first process) contain input signals that start from the process software and go to the interface. Connectors 10 to 25 (for the first process) contain signals that start from the interface and go to the process software.

For the sake of brevity and easy comprehension, this description uses expressions that directly indicate the status of the signals. For example, the expression EMERG = 1 means that the EMERG signal is at logic level 1, or that the EMERG signal is activated.

In other cases, groups of words or signals that have strict functional connections are described. Any mnemonic names are given in brackets after the symbolic names.

8.2.1. I00K SIGNALS

I0K0 (EMERG)

The EMERG signal is activated by the system when an emergency occurs on a given axis. In this case, the control sets SPEPN = 0 (an internal system signal) so as to switch off the auxiliaries, and CONP = 0. It also declares the type of emergency in progress (first four bits) and the axis concerned (last four bits), by means of the word W06K3. The EMERG signal is set to 0 either by a reset command or when the machine tool is switched back on and MUSPE is set = 0.

I0K1 (RESE)

The RESE signal is activated after one of the following events:

- Reset requested by operator;
- Reset requested by machine logic;
- Machine tool switch-off (MUSPE = 1);
- Machine tool switch-on.

This signal is of the impulsive type with duration equal to two logic cycles, and can be used to reset the MT logic equations.

I0K2 (CONP)

The CONP is activated when the system is switched on for the second time with MUSPE = 0. This signal communicates that the control has been correctly initialized. Once CONP = 1 the control is capable of slaving and controlling the axes, by activating all the machine protection devices. The CONP signal goes to 0 after an emergency and when MUSPE = 1 (machine tool off).

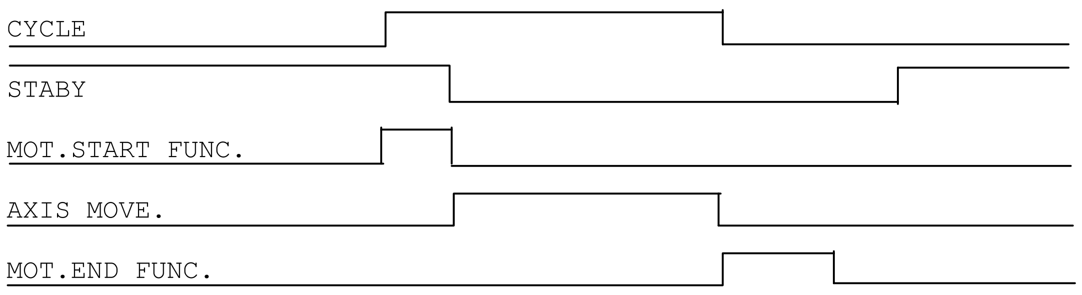
I0K3 (CYCLE)

The CYCLE signal is activated at the beginning of a block and remains at 1 throughout the motion, except for manual motion, and during the emission of motion start auxiliary functions. This signal informs the logic that the system is executing a block sent from keyboard or from the program.

I0K4 (STABY)

This signal informs the interface that axes are stationary in position. It changes to logic level 0 whenever the axes are moved (even in manual); during movement in GOO and G29 this signal changes logic level at every motion start and end. During movement in G28 and G27 it remains constant at logic level 0.

Notes on CYCLE and STABY signal timing



The diagram shows that the logic must take into account the combination of Cycle and Staby signals to know when the axes actually move. That is, the axes move when CYCLE = 1 and STABY = 0. In all other cases the control is emitting motion end or start functions.

I0K6 (RCM)

The RCM signal is activated by the system when a stored search is in progress.
In detail, it is set to 1 after inputting RCM press **SEND** and pressing **CYCLE START** in Auto mode; it stays at level 1 until the axes are moved back to the profile, and pushbutton **HOLD** is pressed to resume program execution.

I0K7 (CYON)

The CYON signal remains to 1 as long while the cycle start lamp is on.

WOK1 (ABI1, ABI2,..., ABI8)

In this word, which consists of signals ABI1, ABI2, , ABI8, the signals corresponding to the axes slaved after a request for axes enable made by the logic with the word W10K1 (RABI1, RABI2,..., RABI8), are set to 1. The relation between axes and bits is declared in the interpolator, in the AXCFIL file. For example, the first axis declared in AXCFIL corresponds to the first bit in the ABI1 word, and so on.

WOK2 (MOV, MOV2,..., MOV8)

This word consists of signals MOV1, MOV2,..., MOV8. When at logic level 1, these signals inform the interface that the system is about to move the axis corresponding to the signal activated. The signals of this word are activated after the motion end functions are sent, and remains at level 1 during axis movement until positioning has been completed. The signals at 1 go to 0 after the axis has come within the positioning tolerance. In a block with G27 or G28, the word signals corresponding to the axes to be moved are activated at the beginning of the block. Axis correspondence is declared in the interpolator, in the AXCFIL file.

I0K24 (POSIA)

The POSIA signal is activated when an axis motion request is made by the W12K1 word. It remains at logic level 1 until the request is terminated or another request is received. This signal informs the logic that the axis motion in the corresponding record of the FInMOV file has been performed.

I0K25 (POSIM)

When the POSIM signal is at logic level 1, it informs the logic that the spindle has been positioned. It is activated when the transducer gives a signal to indicate that the spindle is within the positioning tolerance set in the AXCFIL file, and goes to 0 when ANGOM = 0. This signal is only read by the machine logic during a spindle angle orientation request. If the spindle exceeds tolerance before the request is terminated, the system attempts to return it to its position. If it fails to do so it enters emergency status.

I0K26 (NACKTO)

When the NACKTO signal is at logic level 1, it informs the logic that the T function update request has not been accepted for one of the following reasons:

- Compensation file does not exist;
- Compensation does not exist;
- Compensation out of tolerance;
- I/O logic error;
- Characterized tool life file not present or over;
- Tool in the spindle not acknowledged.

After this situation has occurred, the interface resets the logic.

I0K27 (ACKTO)

When the ACKTO signal is at logic level 1, it informs the logic that the T function on the spindle and the related compensation have been updated.

I0K28 (HOLDA)

When the HOLDA signal is at logic level 1, it informs the logic that the system is in hold status. This signal is activated after the hold request, so that the axes can be slowed down if necessary. It remains at 1 until the operator requests hold status exit. The hold request can be made by pressing a pushbutton on the console, or by the machine logic using signals HLDR or RHOE. In order to exit hold status, COMU = 1 is set by the machine logic.

I0K29 (ACKCM)

The ACKCM signal is normally at logic level 1. During a commutation request between two spindles, it is set at logic level 0 for the whole commutation execution of the system.

I0K30 (NCKCM)

When at logic level 1, the NCKCM signal informs the logic that the request to switch the spindle axis with another axis has not been accepted because it is incongruent. It will remain at 1 until the switching request that generated it is reset.

I0K31 (DIRMN)

When at logic level 1, the DIRMN signal informs the logic that a manual axis motion in the negative direction is in progress.

8.2.2. I01K SIGNALS

This connector consists of four words which contain the indexing axis function latched in BCD, in 5.3 format. The function is latched in the following words:

W01K0
W01K1
W01K2
W01K3

These words inform the logic of the latched indexing axis function. Values from thousandths to tens of thousands of degrees can be programmed. The order in which the various figures are stored is described in the I01K layout, in the "Signal Layouts" section of this chapter.

8.2.3. I02K SIGNALS

This connector informs the logic of the T function programmed either in RANDOM or in NORMAL mode. In the first two-word group (W02K0 and W02K1), the T function is provided in RANDOM mode, while in the second two-group word (W02K2 e W02K3) the T function is provided in NORMAL mode.

In both cases, the T functions programmed are filtered by means of the optional TOOL LIFE. The TOOL LIFE file lists a family of tools that are interchangeable. When a certain tool in the family is defective or used, the next tool listed in the table is selected. For this reason, an alternative T function may be sent to the spindle in place of the T function programmed, in TOOL LIFE mode.

W02K0 - W02K1

These words are significant only with RANDOM tool crib management. The code set in BCD format represents the position in the crib of the tool to be mounted on the spindle. If the TOOL IN UNLOAD option is not used, it is also the position in the crib into which the tool taken from the spindle will have to be unloaded. The position is expressed in units, tens, hundreds and thousands. The order in which the various figures are set is described in the I02K layout.

W02K2 - W02K3

These words are significant only with the NORMAL tool crib management. The code set in BCD format represents the actual code of the tool which is to mounted on the spindle, filtered by the optional TOOL LIFE management. This tool code or position is expressed in units, tens, hundreds and thousands. The order in which the various figures are set is described in the I02K layout.

Notes:

The notes that follow apply to both NORMAL and RANDOM management.

• For possible and particular processing to be executed via machine logic, the tool code actually programmed is always issued and set in the following words: W04K0 and W04K1.

• When all the tool codes of a "family" have expired and the last tool code has TO as an "alternative" code, the system issues strobe FUTKO instead of strobe FUAT but the error message FILMS4-72 is not displayed. It is therefore essential that the machine logic should stop the running part program.

8.2.4. I03K SIGNALS

W03K0

This word provides the logic with the M functions in BCD code. The BCD code im written in the word and is stored there for two logic cycles if the CEFA signal is at logic level 1. This type of function can be executed at axis motion start or end if it has not been declared as an "expedite" function in the IOCFIL file. For the order in which the digits are written, refer to layout IO3K.

W03K1

This word provides the logic with the expedite M functions in BCD code. In the word, the BCD code is written at the beginning of the axis motion block in which it has been specified and lasts all through the axis motion programmed in the block. The expedite M functions are defined as such during characterization in the IOCFIL file. This type of function is not affected by the CEFA signal status and can also be programmed in continuous G28 and G27 mode. For the order in which the digits are written, refer to layout I03K.

W03K2

This word provides the logic with the H functions in BCD code. The BCD code is written in the word and remains stored there for two logic cycles only if the CEFA signal is at logic level 1. This type of function can be executed at the start or end of an axis motion if it has not been declared as an expedite function in the IOCFIL file. For the order in which the digits are written, refer to layout I03K.

W03K3

This word provides the logic with the expedite H functions in BCD code. In the word, the BCD code is written at the beginning of the axis motion block in which it has been specified and lasts all through the axis motion programmed in the block.

The expedite H functions are defined as such during characterization in the IOCFIL file. This type of function is not conditioned by the status of the CEFA signal and can be programmed in continuous G28 and G27 mode. For the order in which the digits are written, refer to layout I03K.

8.2.5. I04K SIGNALS

W04K0 - W04K1

These words provide the logic with the T function programmed in BCD code. The BCD code is only written on the two words if the CEFA signal is a logic level 1. The code remains stored in the words until the next T function is programmed or until a reset is performed. The T function is always a motion start function. For the order in which the digits are written, refer to layout I04K of the "Connector layouts" section of this chapter.

I04K16 (FUAS)

The FUAS signal represents the S function strobe. When at logic level 1, it informs the logic that a new S function code has been issued. This signal remains at 1 for two logic cycles.

I04K17 (FUAT)

The FUAT signal represents the T function strobe. When at logic level 1, it informs the logic that a new T function code has been issured. This signal remains at 1 for two logic cycles.

I04K18 (FUAM)

The FUAM signal represents the M function strobe. When at logic level 1, it informs the logic that a new M function code has been issued. This signal remains at 1 for two logic cycles.

I04K19 (FUAH)

The FUAH signal represents the H function strobe. When at logic level 1, it informs the logic that a new H function code has been issued. This signal remains at 1 for two logic cycles.

I04K29 (TASC1)

The TASC1 signal represents the indexing axis 1 function strobe. When at logic level 1, it informs the logic that a new indexing axis 1 function code has been issued. This signal remains at 1 for two logic cycles.

I04K21 (TASC2)

The TASC2 signal represents the indexing axis 2 function strobe. When at logic level 1, it informs the logic that a new indexing axis 2 function code has been issued. This signal remains at 1 for two logic cycles.

I04K22 (TASC3)

The TASC3 signal represents the indexing axis 3 function strobe. When at logic level 1, it informs the logic that a new indexing axis 3 function code has been issued. This signal remains at 1 for two logic cycles.

I04K23 (FUTKO)

The FUTKO signal represents the T function strobe with tool life expired. This signal is activated only if the TOOL LIFE option is used, to indicate that the programmed tool belongs to a family whose members are all broken or unusable (see GEnTOL file). In this case, the system informs the interface only of the T function programmed with the FUKTO strobe signal at 1 for two logic cycles rather than with the FUAT strobe signal at 1.

Important. The system activates the FUTKO strobe only when, in the record that manages the life of the programmed or the selected tool, the alternative tool is TO.

If the alternative tool is the selected tool itself, the system activates neither the FUTKO strobe nor the T code and a locking error will occur. The following error message will be displayed:

FILMS4 72

I04K24 (SESC)

The SESC signal is activated during a negative indexing axis motion. It remains activated until another motion is programmed or a reset is performed.

I04K25 (MPROFI)

The MPROFI signal is activated when the axes move in contour G1, G2, G3.

I04K26 (CUMAN)

The CUMAN signal is activated only if RANDOM tool management is used, to indicate that the programmed T function does not exist in the FInRAN file or is a TO and there are no free records for storing the unloaded tool.

I04K27 (PWMAN)

The PWMAN signal remains active throughout a spindle rotation request. This signal represents the result of an OR operation performed on the signals requesting spindle rotation (ROMAO/A, FOMAO/A, ANGOM). This signal can go to 1 even if the spindle is not actually rotating. This signal is only valid for spindles that have transducers and that are driven by DC motors .

I04K28 (SOFIT)

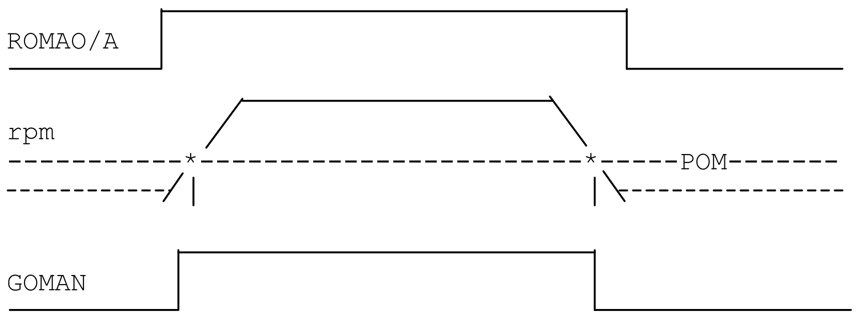
This signal is enabled during cycles G72 - G73 - G74. It is set to 1 at the beginning of the start cycle and set to 0 at the end of the rapid movement in the same cycle. The timing of signals SOFIT - MISU - INTUT in cycles G72-G73 and G74 is as follows:

```
                <-------FORWARD CYCLE----->|<--RETURN CYCLE->|
                .   rapid     .   feedrate   .   rapid rate    .
                .  approach   .   probing    .                 .
                .____________.______________.__________________.
     MEASURE    |            |                                 |
     CYCLE      |            |                                 |________
                 ____________|_________________________________
     MISU       |            |                                 |
     ________   |            |                                 |________
                 ____________|_________________________________
     INTUT      |            |                                 |
     ________   |            |                                 |________
                 ____________|
     SOFIT      |            |
     ________   |            |_________________________________________
```

I04K29 (GOMAN)

The GOMAN signal is set to 1 when a spindle rotation request is made and the number of spindle revolutions is greater than the value set in the POM statement. This signal goes to 0 when the number of spindle revolutions is below the value set in the POM statement. This signal is only valid for spindles that have transducers and that are driven by DC motors.

The following diagram shows the status of the GOMAN signal in relation to the number of spindle revoultions and the status of signals ROMAO and ROMAA:



I04K30 (ACKCY)

The ACKCY signal is only activated in console control by the logic in order to inform the logic that the request for axis motion from console has been accepted and executed.

I04K31 (NCKCY)

The NCKCY signal is only activated in console control by the logic in order to inform the logic that the request for axis motion from the console has not been accepted because it is incongruent with the state of the system.

8.2.6. I05K SIGNALS

I05K0 (FGOO)

The FGOO signal represents the status of the control unit in GOO (rapid). This signal is activated when power-on has been completed. It is set to 0 by a motion in G1, G2 and G3 and is re-activated if a rapid motion or a reset is programed.

I05K1 (PROFI)

The PROFI signal represents the control status in contour. This signal is activated when a motion in G1, G2, or G3 is programmed, and remains at 1 until a motion in GO occurs.

I05K2 (MISU)

This signal is activated during measure cycles G72-G73. MISU goes to 1 during the initial phase and goes to 0 at the end of the return phase. Its synchronization has been shown for the SOFIT signal.

I05K3 (INTUT)

This signal is activated during probing cycle G74. INTUT goes to 1 during the initial phase and goes to 0 at the end of the return phase. Its synchronization has been shown for the SOFIT signal.

I05K4 (FG8.1) - I05K5 (PG8.2)
I05K6 (FG8.4) - I05K7 (PG8.8)

Signals FG8.1, FG8.2, FG8.4, FG8.8 inform the system of the type of fixed cycle in progress (G81 - G89).

W05K1 (ASRI1, ASRI2,..., ASRI8)

When the signals in this word go to 1, they inform the logic that the corresponding axes refer to the home position. The relation between HOMED axis and word signal is determined by the interpolator setting in the AXCFIL file. For example, the first signal (bit 1) corresponds to the first axis, and so on.

I05K16 (ROPO1) - I05Kl7 (ROPO2)
I05K18 (RONE1) - I05K19 (RONE2)

These signals are activated during point-to-point axis motion. When they go to 1, these signals infom the interface of the positive direction (ROPO1, ROPO2) or the negative direction (RONE1, RONE2) of the point-to-point axes 1 and 2. These signals are used when the related motors are not of the DC type.

I05K20 (ROLE1) - I05K21 (ROLE2)
I05K22 (ROLLE1) - I05K23 (ROLLE2)

These signals are activated by the system, upon the correct characterization of record TAx in the IOCFIL file, when a point-to-point axis powered by a motor NOT in D.C. is positioned.
When these signals go to 1, they communicate to the machine logic the possibility of executing the first (ROLE1, ROLE2) and second (ROLLE1, ROLLE2) deceleration of the point-to-point axis. This allows the logic to execute a maximum of two axis velocity changes before it actually stops in its position.

I05K24 (POSI1) - I05K25 (POSI2)

When these signals go to 1, they inform the logic that the corresponding point-to-point axis is on the positioning tolerance threshold. These signals remain at 1 until the rotation request is activated. If the axis goes out of tolerance with request active, the AXIS MANAGER attempts to bring it back within the tolerance. If it fails to do so it gives the "axis out of tolerance" message (FILMS4 69). The POSI signals always remain at 0 throughout the period required for tolerance re-entry.

I05K26 (BUSY1) - I05K27 (BUSY2)

When these signals are at 1, they inform the logic which channels are busy with positioning requests for point-to-point axes 1 and 2:

- BUSY1 = 1 - channel 1 busy;
- BUSY2 = 2 - channel 2 busy.

I05K28 (KOSY1) - I05K29 (KOSY2)

When these signals are at 1, they inform the logic that a point-to-point axis positioning request has been made on a channel that is already busy with a previous request.

I05K30 (ACKSPC)

When activated, the ACKSPC signal informs the logic that the request for program selection or command activation from keyboard has been accepted. This signal remains active until the request is reset.

I05K31 (NCKSPC)

When active, the NCKSPC informs the logic that the request for program selection or command activation from keyboard has not been accepted because it is incorrect or cannot be activated. This signal remains active until the request is reset.

8.2.7. I06K SIGNALS

W06K0 (COM1, COM2,..., COM8)

In this word, the signals relating to the axis on which switching has been requested by the logic are activated. The COM1 signal refers to axis 1, the COM2 signal to axis 2, and so on. The relation between switched axes and activated signals must be defined during AXCFIL file characterization, in record INx. After the request, the switched axis is no longer slaved for calculation or position, but the positioning tolerance is constantly checked.
Example:

The axis-signal correspondence in the AXCFIL file record Inx is defined as follows:

```
IN3,XYZC,S,30,16

X = axis 1, COM1
Y = axis 2, COM2
Z = axis 3, COM3
C = axis 4, COM4
```

I06K16 (SGAMM1) - I06K17 (SGAMM2)
I06K18 (SGAMM3) - I06K19 (SGAMM4)

When at 1, these signals inform the logic of the range in which the spindle must operate, on the basis of the programmed S function and the setting performed in AXCFIL (GM1-GM2-GM3-GM-4). The logic must declare which range is enabled by means of signals GAMM1,..., GAMM4. The spindle reference in the range change cycle is updated two logic cycles after the SGAMM1 signal, the S function in BCD on words W08K0, W08K1 and W08K2 and the FUAS strobe have been updated. If the range change does not exist, the logic must activate the signal relating to GAMMA 1 (U11K16).

I06K20 (RETRA)

The RETRA signal is activated at the first axis motion in the MBR = 1 status and is held at 1 until the part program block for which the retrace status (MBR = 0) has been activated is returned to, i.e.: this signal remains at 1 during the execution of all part program blocks which are re-executed.

I06K23 (ABIM)

The ABIM signal is activated after requests to the spindle (ROMAO, ROMAA, FOMAO, FOMAA, ANGOM). It goes to 0 when the requests have been executed and terminated. This signal is the result of an OR operation between the spindle request signals:

ABIM=ROMAO+ROMAA+FOMAO+FOMAA+ANGOM

This signal is used to synchronize the switching between two spindles when a single analog channel is used, to ensure that the channel is free.

W06K3

The type of emergency and the number of the axis concerned are given in binary code in this word. The type of emergency is coded on the first group of four signals as follows:

Signal 4	3	2	1	Meaning
0	0	0	1	Switch-off by logic
0	0	1	0	Servo error switch-off
0	0	1	1	Velocity ERROR
0	1	0	0	Calculation exception
0	1	0	1	Transducer anomaly
0	1	1	0	Axis out of tolerance

The number of the axis in emergency status is coded in the second group of four signals. These 4 signals are sent to the logic when an emergency is declared by the system or by the logic and are set to 0 when MUSPE = 0.

8.2.8. I07K SIGNALS

W07K1 - W07K2

In these two words, the four digits sent from keyboard that are associated to the ASCII comparators are stored in BCD. The digits remain stored until a new offset is programmed or a reset occurs. The order in which the various digits are stored is described in layout I07K in the "Signal Layouts" section of this chapter. For further details, see also Chapter 3, section on "ASCII Comparators."

W07K2 - W07K3

The four digits of the compensation associated to the programmed tool, or the alternative tool if the programmed one is worn, are stored in these two words. The digits are stored until the T function has been updated. The order in which the various digits are stored is described in layout I07K of the "Signal Layouts" section of this chapter.

8.2.9. I08K SIGNALS

W08K0 - W08K1 - W08K2

These three words inform the logic, in BCD, of the S function programmed. These digits remain stored until a new S function is programmed or a reset performed. The maximum programmable value is 99999. The order in which the various digits are stored is given in layout I08K of the "Signal Layout" section of this chapter.

W08K3

This word informs the logic of the system operating mode. It represents the status of the "Mode Selector" which is handled either by console or logic (refer to the layout of the I08K connector).

8.2.10. I09K SIGNALS

W09K1 (BILA1, BILA2,..., BILA8)

This word informs the logic of the direction in which all the process axes move. This information is given in real-time so that one or more axes (normally vertical) can be mechanically balanced. The signal, referred to the axis being moved, is set to 1 when the axis direction is positive. The value remains stored even after a reset, until the axis is moved in the negative direction.

I09K24 (INVER)

INVER is activated in the G84 cycle with spindle characterized "without servo control". This signal is set to 1 in advance compared to the point of bottom tapping to allow the logic to give a command of spindle rotation reversal synchronized with the axis movement reversal. The advance time with wich the signal is activated depends from the value set in parameter time-max-inver of record TSM.

I09K25 (STOPR)

STOPR is activated in the G86 cycle with the spindle characterized "without servo control". This signal is set to level 1 in advance compared to the point of end hole to allow the logic to command the spindle stop synchronized with the axis stop (at the end of the boring operation).

8.2.11. U10K SIGNALS

U10K0 (MUSPE)

The MUSPE signal must be set to 1 by the logic every time the auxiliary circuits of the machine tool are switched off. This is to allow the system to set CONP to 0. It should be remembered that MUSPE must be set to 0 only after the auxiliary circuits of the machine tool have been switched on (by pressing the **ON** pushbutton at the end of the system initialization).

U10K1 (REAZ)

The REAZ signal can be activated asynchronously and requests a reset cycle from the logic (see Chapter 7, section on "Reset cycle").

U10K2 (HLDR)

Request for hold with restart enabled. When at logic level 1, the HLDR signal requests that the system stop axis motion. After the request has been made, the system sets HOLDA = 1. To exit the hold status, it is necessary to abort the request, press the **HOLD** pushbutton on the console, enable axis motion with COMU=1, and press cycle start.

U10K3 (RHOE)

Hold request with automatic restart. When at logic level 1, the RHOE signal requests that the system stop axis motion. After the request has been made, the system sets HOLDA = 1. To exit the hold status, set COMU to 1 and cancel the RHOE request.

U10K4 (CYST)

The logic can request that a program previously selected by means of SPG be executed or that axes be displaced by G.L. (console from logic) when the CYST signal is at logic level 1. The request is considered while the system remains in stand-by.

U10K5 (FOLD)

When at logic level 1, the FOLD signal temporarily interrupts any axis motion that may be in progress, unless a threading or tapping cycle is being executed. When the request is aborted, the axes restart in a way that is congruent with the movement previously requested.

N.B. The FOLD request is accepted by the system only on its rising edge. If, therefore, a reset is given during a FOLD request, the FOLD request must be set to zero for the period of reset.

U10K6 (WAIC)

WAIC is only examined by the system before an axis motion takes place. In this case, if WAIC = 1 the logic checks that COMU = 1 and slaves the axis home position. If WAIC = 0, it waits two logic cycles before checking that COMU = 1.

N.B. During the G84 fixed cycle, the WAIC signal must be set to 0.

This signal is normally used to bypass the wait status imposed for two logic cycles in the event of axis motion, and must not be used for axes fitted with blocking device.

U10K7 (RISPE)

RISPE is activated by the logic each time the machine tool has to be switched off because anomalies have been detected by the machine logic. After the switch-off request has been made, the system switches off the auxiliaries and sets internal signal SPEPN to 0. To re-enable a correct switch-on, the request must be aborted.

W10K1 (RABI1, RABI2,..., RABI8)

By means of this word, the logic requests the system to slave the axes corresponding to the signals set to 1. If this request is not executed, the axes remain abandoned (with servo loop open). The axes enable request must only be made after the system has set the CONP signal to 1.

U10K16 (REGTOL)

In normal TOOL control, (non-RANDOM), the REGTOL signal clears the display of the Tool on the spindle. In RANDOM TOOL control, this signal clears the display and the storing of the Tool on the spindle.

U10K17 (DITVU)

If the DITVU signal is set to 1, it stops the life count for the tool in the spindle for the whole period in which the signal remains in that status.
This request is accepted in an asynchronous way, both with the normal tool life count (which foresees the spindle rotation and the working movement) and the "forced" life count by means of signal U10K20 (ABTVU).

U10K18 (SPCCOM)

The SPCCOM signal is activated when a part program is to be selected by the machine logic in order to launch its execution (see Chapter 8, section "Program Launch procedures"). For more information, refer to note 2 in the description of signal FILCMD.

U10K19 (CMDLOG)

The CMDLOG signal represents the request switch carried out by the SPCCOM signal. When the CMDLOG signal is at 1, the keyboard command activation request is generated. When the CMDLOG is at 0, a part program selection request is activated. For more information, refer to note 2 in the description of signal FILCMD.

U10K20 (ABTVU)

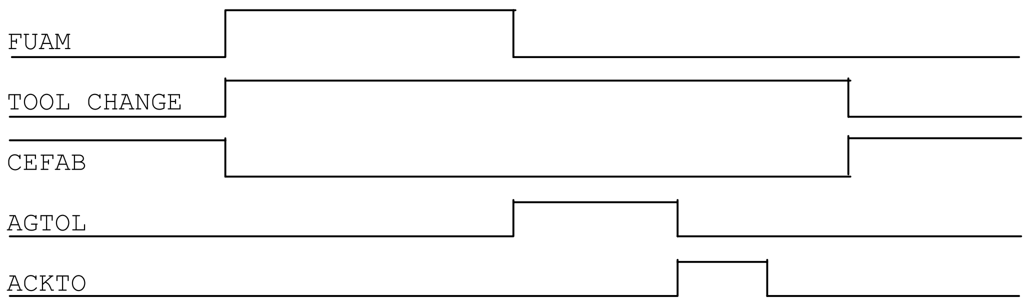
If the ABTVU signal is set to 1, it enables the life count of the tool in the spindle. In this case --as opposed to normal life count of the tool in the spindle which contemplates both the spindle rotation and the working movement-- it is not restrained by any condition. The life count is executed for the whole period in which the signal remains at 1.

U10K21 (AGTOL)

When the AGTOL signal is at 1, it requests the system to update the T function and activate the corresponding compensation. The request must be executed by a motion end function with the following features:
- Compensation change;
- Calculation blocking.

If the function requested by the AGTOL signal is to be correctly executed, the GEFAB signal must be at 0 for an interval longer than the time during which the AGTOL signal is at 1.

The diagram below shows an example of correct AGTOL signal timing.



U10K23 (FILCMD)

If the signal is set to 1, the system interprets any active code in the W17K0 as the No. of the record in the previously edited FILCMD file that must be executed.

If the signal is set to 0, the system interprets any active code in the W17K0 as the number of the initial connector of buffer K where the logic has written, for example, one of the following pieces of information:

a) the keyboard command that is to be executed (e.g. URL=1), in hexadecimal code;

b) the name of the program to be selected and the relevant device (e.g. CIFIX/MP0), in hexadecimal code.

Note:

• Every piece of information written in hexadecimal code in buffer K must end with a hexadecimal code (OAH) corresponding to character <LF>.

• After the SPCCOM request has been executed, the system sets ACKSPC to 1. If the request has not been accepted by the system or if a system error has been detected, ACKSPC will be set to 0. In either case, the logic must reset SPCCOM to 0 when the system sets ACK or NCK to 1 before another SPCCOM request can occur.

If SPCCOM is set to 0 before the acknowledge arrives, the system executes the request without sending either ACK or NCK. For example:

- In order to activate the SPCCOM signal, the SPG,A/MPO ASCII string written in the K buffer connectors must be executed. If SPPCOM is set to 0 while the string is being executed, execution will be accomplished but the ACK will not be set to 1;

- To activate the SPCCOM signal it is necessary to execute, by means of an "indirect file", a series of records that contain keyboard commands:

· the PROTEC file contains the $PROT1/MP0 record;
· the PROT1/MP0 film contains the following records:
```
RAP=1
URL=1
USB=0
UCV=2
SPG,A/MPO
```

The $ indicates to the system that the PROT1/MP0 file must not be selected but that the records in the "indirect" PROT1/MP0 file must be executed.

If SPCCOM is set to 0 while PIPPO/MPO is being executed, the system will only accomplish the current record. The remaining records will not be taken into consideration and ACK will not be set to 1.

If there is a request via SPCCOM, the system will respond by activating NCKSPC in the following cases:

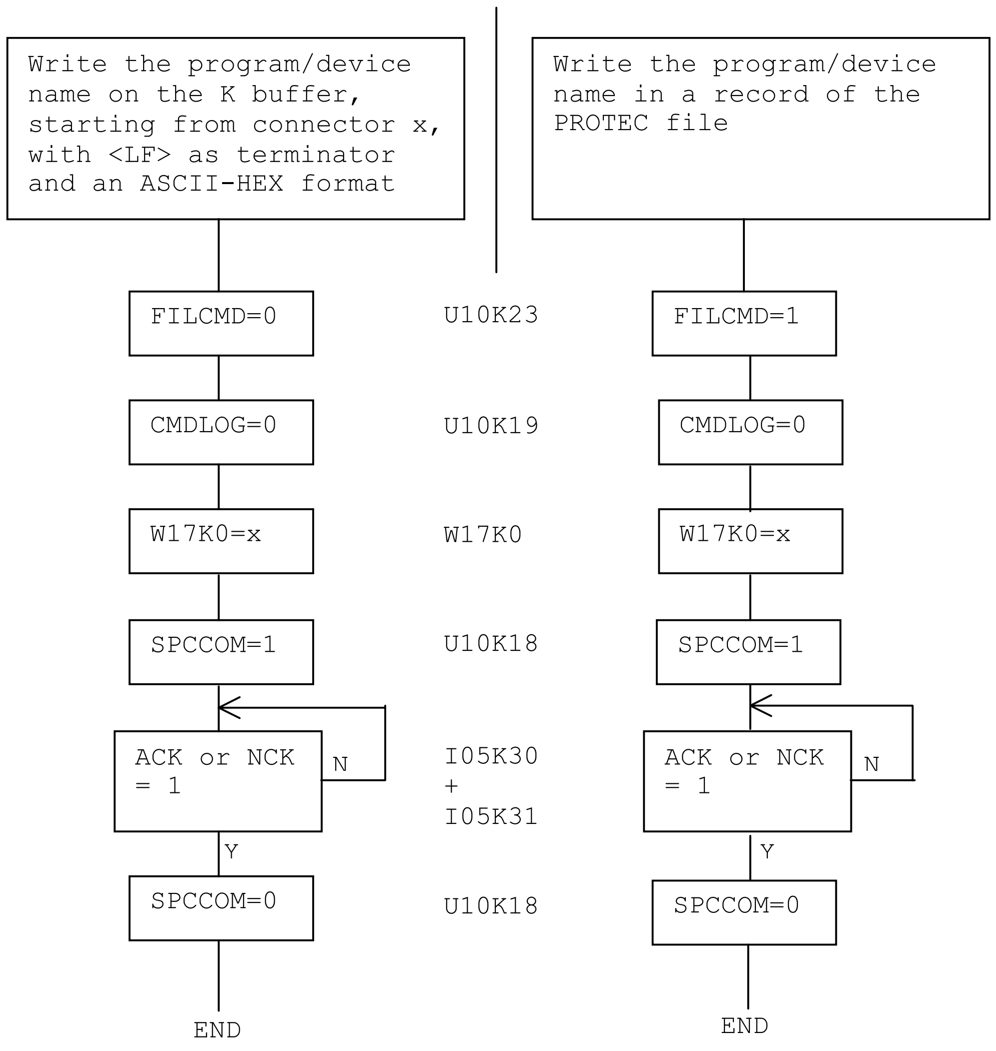
- the content of word W17K0 is zero;
- the ASCII string in the K buffer does not have a terminator equivalent to the <LF> character;
- SPCCOM has been set to 1 and the system is in the Reset status;
- the system is in the Error status or W09K0 is 1 (FILMS401);
- there is no request record in the PROTEC file;
- the indirect file does not exist or does not contain any record;
- the syntax or the format of the requested record is illegal.

The examples that follow illustrate how the logic must temporize the requests to the system in order to request the activation of the code contained in word W17K0.

Example 1: "SELECT PROGRAM" request

Mode A: from K buffer

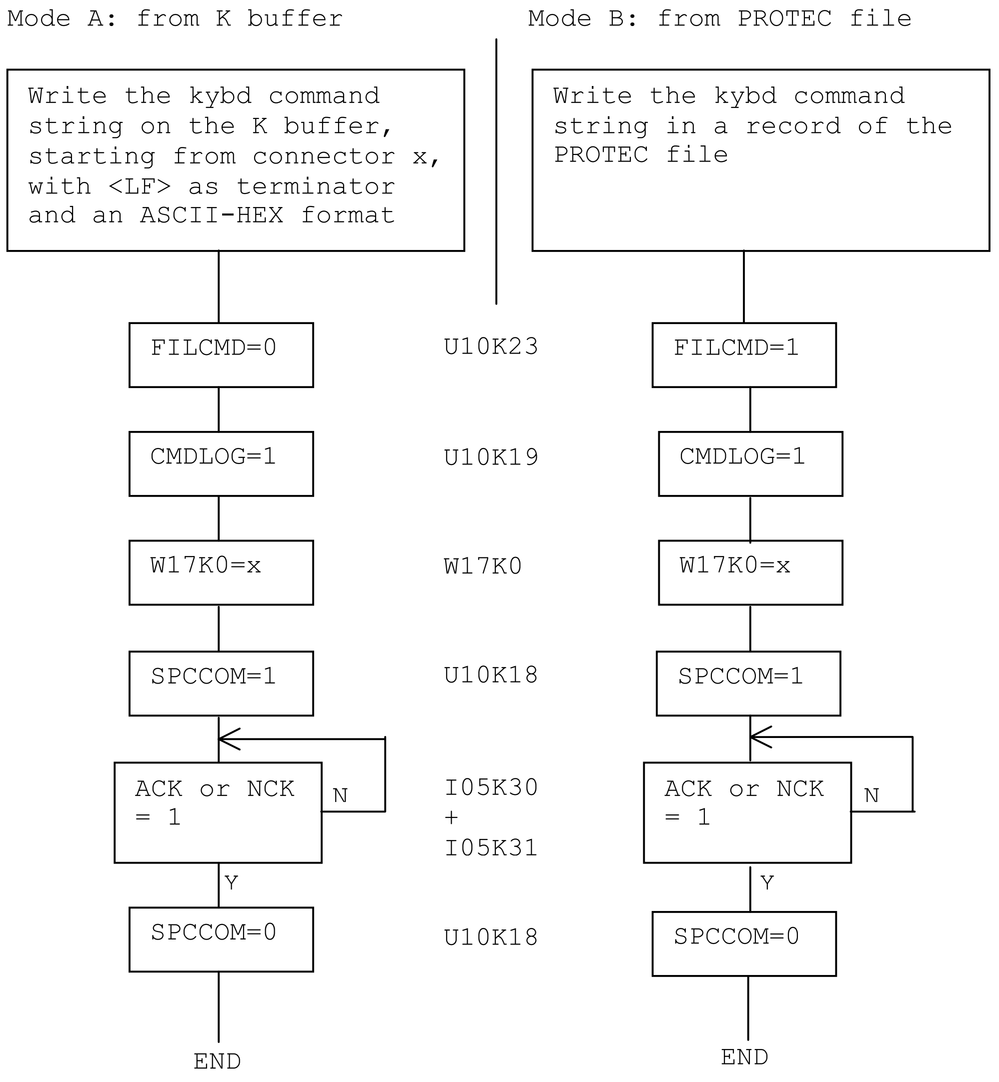
Mode B: from PROTEC file


Write the program/device name on the K buffer, starting from connector x, with <LF> as terminator and an ASCII-HEX format
Write the program/device name in a record of the PROTEC file
FILCMD=0
U10K23
FILCMD=1
CMDLOG=0
U10K19
CMDLOG=0
W17K0=x
W17K0
W17K0=x
SPCCOM=1
U10K18
SPCCOM=1
ACK or NCK = 1
N
I05K30 + I05K31
ACK or NCK = 1
N
Y
Y
SPCCOM=0
U10K18
SPCCOM=0
END
END

x = Number of the K buffer connector from which the program/device name starts

x = Number of the PROTEC file record that stores the program/device name (in binary)

Example 2: "EXECUTE A COMMAND FROM KEYBOARD" request



x = Number of the K buffer connector from which the kybd command string starts

x = Number of the PROTEC file record that stores the kybd command string (in binary)

U10K24 (COMU)

When at logic level 1, the COMU signal enables the system to carry out movements. This signal is analyzed in the following cases:

- Before every manual movement;

- Before every movement in G00;

- Before every movement in G29;

- Before exiting hold status with the HOLDA signal still at logic level 1;

- In the first block before beginning a cycle in G28.

Whenever the COMU signal is analyzed before a movement, and found to be at logic level 0, the system is unable to accept an axis motion end until it goes to 1.

U10K25 (CEFA)

When at logic level 1, the CEFA signal enables the system to issue S, T indexing axis, M (not expedite) and H (not expedite) auxiliary functions.

U10K26 (CEFAB)

The CEFAB signal is analyzed by the system after issuing an M function of the "calculation blocking" type. If the M function is deEND d as a motion end function, FILMAS file records can be executed by the system in synchronized mode. When CEFAB is at 0, axis motions from the system (FILMAS record) and execution of auxiliary functions entered in the record being executed (CEFA = 1 is necessary), are possible. The system remains in the "calculation blocking" M function until CEFAB goes to 0.

U10K27 (MIZE1) - U10K28 (MIZE2) - U10K29 (MIZE3)

These signals are analyzed when the corresponding indexing axes (axes 1, 2 and 3) are programmed to move. After the indexing axis motion request, the system analyzes this signal. If the signal is at 1, the system prepares the programmed displacements. If the signal is at 0, it checks that the displacements programmed are = 0. If they are not it gives an error indication.

U10K30 (RMORE)

The RMORE signal permits the enabling of the "retrace" operating mode requested by the logic. This signal must be set to 1 only after linear motion has been stopped or interpolated with an impulsive hold request (HLDR) and after the system has set HOLDA = 1.

When RMORE is at 1, the cycle start pushbutton can be pressed or a cycle start requested by the logic, in order to make the system exit the hold status and begin executing part program blocks, in semiauto or backwards auto, for a maximum number of blocks declared with the MBR statement.

When "normal" operation is to be resumed, the logic must again request a hold by following the same procedure as that used for the previous request. When the system sets HOLDA = 1, RMORE = 0 must be set. At this point the cycle start pushbutton can be pressed or a cycle start requested by the logic to make the control exit from the hold status and resume "norma1" operation (not retrace mode).

8.2.12. U11K SIGNALS

U11K0 (ANGOM)

When at logic level 1, the ANGOM signal requests the system to position the spindle according to the value declared in the characterization or defined by the logic through the 14K connector. The reference output by the converter is related to the configuration parameters. After the request, the spindle is positioned by the system, which sends the POSIM = 1 signal to the logic when the transducer indicates that the spindle is within the positioning tolerance limits.

Note. When the POSIM signal goes to 1, the system waits for ANGOM to be set to 0 (in this period POSIM remains at 1 and the spindle is held in position), after which the spindle is abandoned by the system. The signals ROMAO, ROMAA, FOMAO, and FOMAA are not considered until the spindle positioning request has teninated.

U11K1 (FOMAO) - U11K2 (FOMAA)

When these signals are at logic level 1, they request the system to force a clockwise (FOMAO) or anticlockwise (FOMAA) reference on the spindle, in accordance with the normal W11K1 word requests. The rotation forcing request has priority over the normal rotation requests (ROMAO e ROMAA). These signals can be used with DC spindles only. The timing for these signals is identical to that used for the ROMAO signal.

U11K3 (ROMAO) - U11K4 (ROMAA)

When these signals are at logic level 1, they request the system to provide the spindle with a clockwise (ROMAO) or anticlockwise (ROMAA) reference in accordance with the last S function programmed and the range entered. These signals can be analyzed by the system at any point. They must also be activated in the case of an AC spindle motor (without slaving).

U11K5 (FORID)

When at logic level 1, the FORID signal divides the reference written on the W11K1 word by 10 with FOMAA = 1 or FOMAO = 1.

W11K1

This word defines the reference on the spindle channel with FOMAA = 1 or FOMAO = 0.

The voltage values definable via this word are a percentage of the 10V max. that can be supplied by the D/A converter. For example, if you write value 01110101 in word W11K, the reference is forced to 7.5V; i.e., the first 4 bits force the tenths of volt, the second 4 force the units of volt.

U11K16 (GAMM1) - U11K17 (GAMM2)
U11K18 (GAMM3) - U11K19 (GAMM4)

These signals must be set to 1 by the logic when the range programmed is mechanically entered, so that the spindle rotation value prescribed is in line with the specifications made during configuration. These signals must also be activated in the case of an AC spindle motor (without slaving).The machine logic can set the above signals (GAMM1 through GAMM4) to zero only if the spindle has been "dropped" by the system, i.e. if ABIM=0.

W11K3

This word must contain the hexadecimal code of the spindle axis name that the logic requests for activations. Without this code, the system activates the spindle axis declared in the ASM record of the IOCFIL file. If the request is not accepted, the system sets signal NCKCM to level 1. If the request is accepted, the system sets signal ACKCM to 0. Once the commutation is ended, the ACKCM signal is set to level 1 and the new spindle will be servo controlled.

8.2.13. U12K SIGNALS

U12K0 (FORZ1) - U12K1 (FORZ2)

These two signals permit three different converters on which forcing can be carried out to be defined (using the definition of three possible axis names given in the IOCFIL file).

The code for the type of converter on the two signals is given by the table shown below:

FORZ1	FORZ2	
0	0	No output on the two channels
1	0	Output on the first axis defined in UCDA
0	1	Output on the second axis defined in UCDA
1	1	Output on the third axis defined in UCDA

U12K2 (FORZN)

When at logic level 0, the FORZN forces a positive reference on the channel selected (by means of signals FORZ1 FORZ2). When this signal is at logic level 1, a negative reference is forced.

U12K4 (FOPA1) - U12K5 (FOPA2)

When at 1, these two signals force positive direction on the point-to-point 1 (FOPA1) and point-to-point 2 (FOPA2) cyclic axes for which motion is requested. If these signals are at 0, the interface selects the shortest route for axis positioning. These signals must be active before the positioning request so that, as far as the hydraulic motors are concerned, the interface can respond correctly by means of positioning signals ROLE, ROLLE, ROPO, RONE.

U12K6 (FONA1) - U12K7 (FONA2)

When at 1, these signals force the negative direction on the point-to-point 1 (FONA1) and point-to-point 2 (FONA2) cyclic axes for which motion is requested. If these signals are at 1, the interface selects the shortest route for axis positioning. These signals must be active before the positioning request so that, as far as the hydraulic motors are concerned, the interface can respond correctly by means of positioning signals ROLE, ROLLE, ROPO, RONE.

W12K1 (WMAS)

If this word is assigned by the logic, it executes an axes motion (process) request at the position shown in the FILMAS file record (axes motion from the system). The record is identified by the value in binary of the W12K1 word. This request can only be activated by a motion end function with "calculation blocking" feature, CEFA = 1 and CEFAB = 0. These limits permit a function that does not have the calculation blocking feature to be programmed in the record to be executed, in addition to an axis motion.

W12K2 (WDIFC1) - W12K3 (WDIFC3)

These two words (signals DI1+, DI1-, DI2+, DI2-..., DI8+, DI8-) make it possible to disable the positive and/or negative travel limit on one or more of the eight processing axes available. This request is asynchronous. This assignment must only be performed after the axis has been zeroed as the system only considers the request on the signal's rising edge.

If in an axis subsection (sect. 2 AXCFIL) you characterize one of the two hardware travel limits as home microswitch (MCZ), using this word you must disable that hardware travel limit for all the time the concerned axis is being reset. For the meaning of the signals, see the layout for connector U12K described in this chapter. In order to disable hardware and software travel limits, the bit must be set to 1.

8.2.14. U13K SIGNALS

W13K2 (WPRDA1) - W13K3 (WPRDA2)

These two words define the reference value (from 0 to 10 V) that should be activated if FORZ1 = 1 or FORZ2 = 1 on a channel defined during configuration. The request can be asynchronous and updating is carried out in real time. For the meaning of the signals contained in this word, refer to the layout for connector U13K given in this chapter.

8.2.15. U14K SIGNALS

W14K0 (COSPM1) - W14K1 (COSPM2)
W14K2 (COSPM3) - W14K3 (COSPM4)

The displacement value, as regard the transducer electrical zero towards which the spindle is to be oriented, is normally characterized in record POM of the sub-section relating to the axis.

To orientate a spindle towards values other than the characterized ones, it is necessary for the logic to write in the words of this connector, the displacement values expressed in degrees and/or fractions of degrees in BCD 5.3 format (resolution up to a thousandth of a degree).

For the meanings of the signals contained in these words, refer to the layout of connector U14K in this chapter.

8.2.16. U15K SIGNALS - U16K SIGNALS

The purpose of these 2 connectors (8 words) is to permit the MT logic to perform a "console from logic" (henceforth referred to as C.L.), i.e. to perform the requests sent to the system that are normally performed with the control console operators (pushbuttons, selectors), by means of the M.T. logic.

In particular, the words from W15K1 to W16K2 can be used by the C.L. to give the system the value to be attributed to the operators that have been disabled by means of the W15K0. W16K0, for example, is the word dedicated to the FEED OVERRIDE selector.

The system considers the requests (activation of the bits on both connectors) in asynchronous way; i.e., at the end of every turn of slow logic, it checks whether any bits have been activated, and it only carries out the requests of the C.L. if they result complete and congruent.

W15K0

The bits of thes word must be activated in order to inform the system which operators must be disabled.

For the meaning of the signals contained in this word, refer to the layout of connector U15K described in this chapter.

W15K1

The bits of this word are used to request the MODE SELECTED functions from the system. The bits of this word can be set one at a time. If more than one bit is set to 1, the system will not consider the value set, taking it to be incorrect. For the meaning of the signals contained in this word, refer to the layout of connector 15K described in this chapter.

W15K2

The bits of this word are used to ask the system which axis is to be selected for manual motion. The bits are activated in the same way as for the W15K1. For the meaning of the signals contained in this word, refer to the layout of connector U15K described in this chapter.

W15K3

The bits of this word are used to ask the system the FEED MANUAL value. This value is a percentage of the value characterized with the MAN statement set in the AXCFIL file. The first seven bits are used to indicate the feed percentage (binary), the eighth bit, bit = 1, is used to tell the system that the set value is negative.

W16K0 - W16K1

The bits of these two words, that are used in the same way as the W15K3 word except for the fact that the most significant bit (sign) is not considered, are used to ask the system for the FEED OVERRIDE and SPINDLE OVERRIDE percentage respectively. For the meaning of the bits of these 2 words, refer to the layout of connector 16K described in this chapter.

W16K2

The bits of this word are used to ask the system for the manual JOG value. A particular JOG value corresponds to each bit (0 to 5) which are activated in the same way as the W15K1 bits.

Bit 6 is unused, whereas if bit 7 is set to 1, it is used to tell the system that the set JOG value execution must be completed even if the cycle start command (CYST-U10K4) is given during axis displacement.
The condition set via bit 7 is only valid for JOG movements executed by C.L.

W16K3 (WSEDP)

Through the bits of this word it is possible to force from machine logic, the selection of video screen (#0÷#6) and/or the selection of a process (SYSTEM - SUBSYS1 ÷ SUBSYS5) by writing a code in binary format in the word.

This allows the function keys **P1** and **P3** to work.

In particular, through a code comprised in the first 4 bits (U16K24-25-26-27) a video screen is forced, whereas through a code which comprises the second 4 bits the selection of a process is forced.

To force the selection of the system video screen (#0) and/or the system process (SYSTEM), it is necessary to write a code which forces all the respective bits to 1.

It should be remembered that the forcing of these bits to 1 has priority on the pressing of keys **P1** and **P3**. The function of the keys will therefore be restored only when the value of the word (corresponding to the bits of the **P1** and/or **P3** key) will be brought back to zero.

This forcing can be requested, through the suitable word of pack K, for one process at the time.

Example:

By writing code 022D = 0001 0110 in W16K3 the selection of the first process (SUBSYS1) and the video screen of graphic partition (#6) are forced.

NOTES ON HOLD ACTIVATION AND USE (HLDR-U10K2,RHOE-U10R3)

The Hold must be handled by the C.L. in a special way.
The operator console Hold pushbutton cannot be disabled by the word W15K0. Two types of HOLD can be requested from C.L.:

1) HOLD with exit by means of operator enable (HLDR-U10K2);
2) HOLD with exit and automatic restart (RHOE-U10K3).

If the hold bit "HLDR" is changed to 1, the system enters the Hold status.

To exit the Hold status, the hold request must first terminate and the operator must then take the machine back to normal working conditions by pressing the HOLD pushbutton on the console. To avoid the machine exiting the block status unexpectedly, the hold pushbutton on the operator console will be disabled until the hold request terminates (from C.L.).

The changing of the hold bit "RHOE" makes the system enter the Hold status.

The HOLD status is automatically exited by means of the C.L. at the end of the hold request. The system will not restart if it is already in hold and a "RHOE" hold request is made, or if a "HLDR" hold request has been made while the system is in the hold status.

This means that the system can only be restarted if RHOE is active, and a HLDR with greater priority is not activated.

NOTES ON CYCLE START ACTIVATION AND USE

The Cycle Start (U10K4) is rather special as regards its use.

The system is informed of changes in the signal U10K4 even if the W15K0 zero bit is not active. The only purpose of this bit (U15K0) is to disable the **CYCLE** pushbutton.

As regards Cycle Start control from logic, the system guarantees the activation of a signal (ACKCY-I04K30) that permits synchronization with the system. In other words the C.L. can activate the Cycle Start request (U10K4) only when the signal ACKCY-I04K30 is at zero, and it must reset the cycle start request when the system sets the signal ACKCY-I04K30 to 1.

Apart from this timing, the system does not consider any Cycle Start requests.

8.2.17. U17K SIGNALS

W17K0

The value in binary format that the logic can assign to this word indicates to the system where the names of the technological program or the keyboard command are placed.
The selection or activation of both is requested by the logic.
In particular, the code written in the word assumes two different meanings according to the status of the FILCMD bit:

FILCMD = 1 -	the code identifies the number of the record of the PROTEC file where the name of the program or the keyboard command are written.
FILCMD = 0 -	the code identifies the number of the first connector of the K pack where the name of the program or the keyboard command have been codified.

W17K1 (RCOM1, RCOM2,..., RCOM8)

By means of the bits of this word it is possible to ask the system to commutate (among them) two axes belonging to the same process and correctly characterized.

In the process it is possible to have a maximum of four "pairs" of commutated axes.

The commutation between two axes is executed by setting the bit relating to the axis to be "released" to 1. If the system does not find any incongruities in the commutation request, at the end of it, it sets the bit, (in W06K0) relating to the axis which has been "released", to level 1.

The "released" axis is controlled into position by the system via the stand-by servo error. The relation bit-axis is given by the position with which the axes have been characterized in the interpolator.

W17K3

By assigning a binary value in this word, it is possible to ask the system to display on the fourth line one of the 255 messages reserved for the machine logic, contained in the FinMSG file characterized by means of record FLC of the PGCFIL file.

The number assigned to the word identifies the numer of the record containing the message which is to be displayed.

If this file (FInMSG) has not been created or has not been characterized (record FLC - file PGCFIL) or, if in file FCRSYS the declaration FILMS5,MESSAG/MPx is missing, the message requested is displayed in the system code with the number of the message. For example:

Ex. FILMS5 190

8.2.18. U18K SIGNALS

The position to be reached by point-to-point axis 1 is defined in this connector. It is possible to request positioning at 99999 different positions (the number must be declared during configuration in the IOCFIL file) and 999 equidistant points inside each station.

Example:

With a one-metre long linear point-to-point axis, and if the transducer permits it, a resolution of 1 hundredth of a micron can be programmed for the point-to-point axis.

8.2.19. U19K SIGNALS

The position to be reached by point-to-point axis 2 is defined in this connector. It is possible to request positioning at 99999 different positions (the number must be declared during configuration in the IOCFIL file) and 999 equidistant points inside each station.

8.2.20. U20K SIGNALS

W20K0 (WASM01) - W20K1 (WASM02)

These words contain respectively the number of the point-to-point axis 1, and the number of the point-to-point axis 2 that must be moved to the position programmed respectively on connectors 18K and 19K.

W20K2 (WFESS1) - W20K3 (WFEAS2)

By activating these two words, a speed percentage (in relation to the rapid speed defined during configuration) can be assigned for point-to-point axis 1 (WFESA1) and 2 (WFESA2), respectively.

8.2.21. U21K SIGNALS - U22K SIGNALS

The bits in these connectors allow to display the first 64 machine logic messages on line five. These messages are stored in the file characterized by the FLC record in the PGCFIL file.

If this file (FInMSG) has not been created or has not been characterized or the declaration FILMS5,MESSAG/MPx is missing from FCRSYS, the message will be displayed in the system code with the number of the message. For example:

```
FILMS5 22               where 22 is the message number.
```

8.2.22. U23K SIGNALS - U24K SIGNALS

By assigning the words of these two connectors, alphanumerical values can be displayed in three different fields on one video line (subsystem screen). BCD codes can be written in fields 1 and 2 to display values up to 9999, and BINARY codes can be written in field 3 to display values up to 255. An ASCII character, which is to be written in the appropriate words, can be associated to the three assigned values. See layouts of connectors 23K and 24K.

Example:

By assigning:
W23K0 = 42H
W23K1 = 97H

the first field displays:
B 0097

8.2.23. PHYSICAL I/O SIGNALS HANDLED BY THE CONTROL UNIT

In the interests of security, the travel limit, home position and computer emergency output signals are directly handled by the system and not by the machine logic.

Travel Limits

These signals consist of two inputs, from the MT, for each axis, which are characterized in the AXCFIL file in PLC format. It should be remembered that the travel limit that is not pressed must give the system an input = 1.

If an input that has been declared as travel limit input is used as a home position input declaration, the travel limit input involved must be disabled during the axis zeroing cycle by means of the bits available in the words W12K2-W12K3.

Home position

These signals consist of an input for each axis. It should be rembered that the home position (e.g. a microswitch) that is not pressed, must give the system an input = 1. One of the inputs declared as "travel limit" can be declared as "home position".

Computer Emergency

When the SPEPN signal is at logic level 1, the auxiliary switch-off relay is triggered. This signal cannot be addressed by the machine logic but can be indirectly activated by means of interface signal RISPE. Each process is associated to the related RISPE signal and affects the same SPEPN signal.

9. ERROR CODES AND MESSAGES

This chapter describes the error codes and messages which can be displayed by the system in the various environments where the user operates through PLC programming.

9.1. SYSTEM ERRORS

This section lists the errors communicated by the system in different environments during the various PLC programming stages. In the table shown below, the error messages, their meaning, user operations and the environment where the error originated, are all provided. The possible causes of an error are given below:

- A = environment;
- C = compiler;
- D = debugger;
- S = system.

The error messages displayed in small letters usually relate to errors committed by the user, while those in capital letters and reverse video usually indicate errors made by the system.

MESSAGE	OPERATION	ORIGIN
invalid command	Check the entry line	C D
invalid format	Check the entry line	C D
invalid option	Check the entry line	C D
invalid name – name	Specify the correct name	C D
file not found: nameT/dev	Specify the correct file	C D
illegal device: /dev	Specify the correct logic support	C D

MESSAGE	OPERATION	ORIGIN
device not ready: /dev Logic device specified is not available (not configured)	Specify the correct logic support	C D
file already open: nameT/dev	Check the programs that have already been opened	C D
insufficient space for file: nameT/dev	If the saved program exists it is possible to re-compile without the S option, or delete the unnecessary programs	C
compile errors in program	Eliminate the error and re-compile	D
overflow	Introduce a smaller value	A
VIDEO SGALE NOT AVAILABLE	The small-scale video partition is reserved. Exit the function, reserving the partition, and try again	S
NO FREE CHANNELS	Try again. Check the system configuration	C D
LIST DEVICE ERROR – xx	Check that the device is enabled	C D
I/O ERROR – xx	Check the error code	C D
AMBIENT FILE READ ERROR – xx	Check the file device status	A
AMBIENT FILE WRITE ERROR - xx I/O error xx in reading SIPCON file	Check the file device status	A
SOURCE READ ERROR – xx I/O error xx in writing SIPCON file	Check the program device status	C

MESSAGE	**OPERATION**	**ORIGIN**
SYMBOL FILE UPDATE ERROR - xx	Check the program device status	C
FILE CANCELLATION ERROR: nameT/dev I/O error xx. Program cancelled	Check the existing programs	C
FILE RESTORATION ERROR: nameT/dev I/O error xx. Program restoration	Check the existing programs	C
COMPILER ABORTED Fatal error	Check the line where the error has occurred	C
NON DEBUG PROGRAM - name/dev Attempt to load a program that is not enabled for debugging	Load the debug file	D
INVALID RECORD TYPE IN name/dev The debug file contains a type of record that is not recognised	Delete the invalid record	D
I/O ERROR XX ON PROGRAM name/dev I/O error xx in loading the program specified by "name/dev"	Check the program device status	D
CHECK SUM ERROR ON PROGRAM - name/dev Contaminated program. Fatal error	Re-compile the program	D
DEBUGGER ABORTED Fatal error	Check the line where the error has occurred	D
HELP MESSAGE ERROR – z This error message should never appear	Consult servicing technicians	C D

9.2. NON-FATAL COMPILATION ERRORS

The system indicates compilation errors by means of an error code which represents a given message. The errors described here are of the non-fatal type. The system indicates them by means of the following message:

*** compilation error * nn ***

where nn is the code for the error given in the table shown below.

If the non-fatal error is of the warning type (see Chapter 5, Section on "Compiling the machine logic"), the error is indicated by the following message:

*** compilation warning * nn ***

where nn is the code for the error given in the table shown below.

The errors of the warning type are marked with an asterisk that is placed after the code number.

The table shown below gives the code number and meaning of each error.

CODE	**MEANING**
1	Variable definition error
2	Syntax error
3	Incorrect operand
4	Message identifier error
5	Message number not provided
6	Message number error
7	Message too long
8	Message already defined
9	MUX function identifier error
10	Separator incorrect
11	Bracket "(" missing before the MUX function parameter
12	Bracket ")" missing after the MUX function parameter
13	Word parameter different from signal parameter

CODE	MEANING
14	Elements not recognised after a MUX function
15	Statement too complex
16	Excessive bracket nesting
17	SCM function identifier error
18	ABS function identifier error
19	Symbol for invalid relation in a comparison operation
20	Bracket "]" missing in a comparison operation
21	ASCII comparator null
22	ASCII comparator too long
23	Inverted commas (") missing from the ASCII string definition of an ASCII comparator
24	DEC function identifier error
25	ENC function identifier error
26	BIN function identifier error
27	BCD function identifier error
28	LOW function identifier error
29	HIG function identifier error
30	XCG function identifier error
31	Bracket ")" missing from a DEC function
32	Bracket ")" missing from an ENC function
33	Bracket ")" missing from a BIN function
34	Bracket ")" missing from a BCD function
35	Bracket ")" missing from a LOW function
36	Bracket ")" missing from a HIG function
37	Bracket ")" missing from an XCH function

CODE	MEANING
39 *	Time limit for slow logic task has expired
40	Block nesting level too high
41	Block name not congruent
42	DOF, ENDF statements not congruent
43	DO does not have the corresponding END in a block
44	END does not have the corresponding DO in a block
45	DOE statement does not have an ENDE or ENDF statement before it
46 *	Mnemonic syntax error
47 *	Mnemonic already declared
48 *	Mnemonic table full
49	Mnemonic not found

9.3. FATAL COMPILATION ERRORS

The system indicates compilation errors by the system by means of an error code which represents a given message. The errors described here are fatal. The system indicates these errors with the following message:

COMPILATION ABORTED motive

where "motive" is the message given in the table shown below.

CODE	MEANING
50	TIME OVERFLOW
51	CODE OVERFLOW
52	DEBUG PROGRAM LINE MAXIMUM EXCEEDED
60	I/O ERROR ON PROGRAM WRITE
62	I/O ERROR ON SOURCE READ
63	OPERATOR ABORT REQUEST

9.4. DEBUGGING ERRORS

The system indicates debugging errors by displaying an error message. The errors that occur during the debugging stage are listed below. The following table shows the message, its meaning and the operation that the user must perform in order to eliminate the error.

MESSAGE	OPERATION
Variable definition error	Specify the correct variable name
Invalid value	Specify the correct value.
Line number range error	Specify the correct field limits
No monitor variable present	The monitor input cannot be cancelled because the window is empty
Monitor window full	No other variable can be added because the window is full

APPENDIX

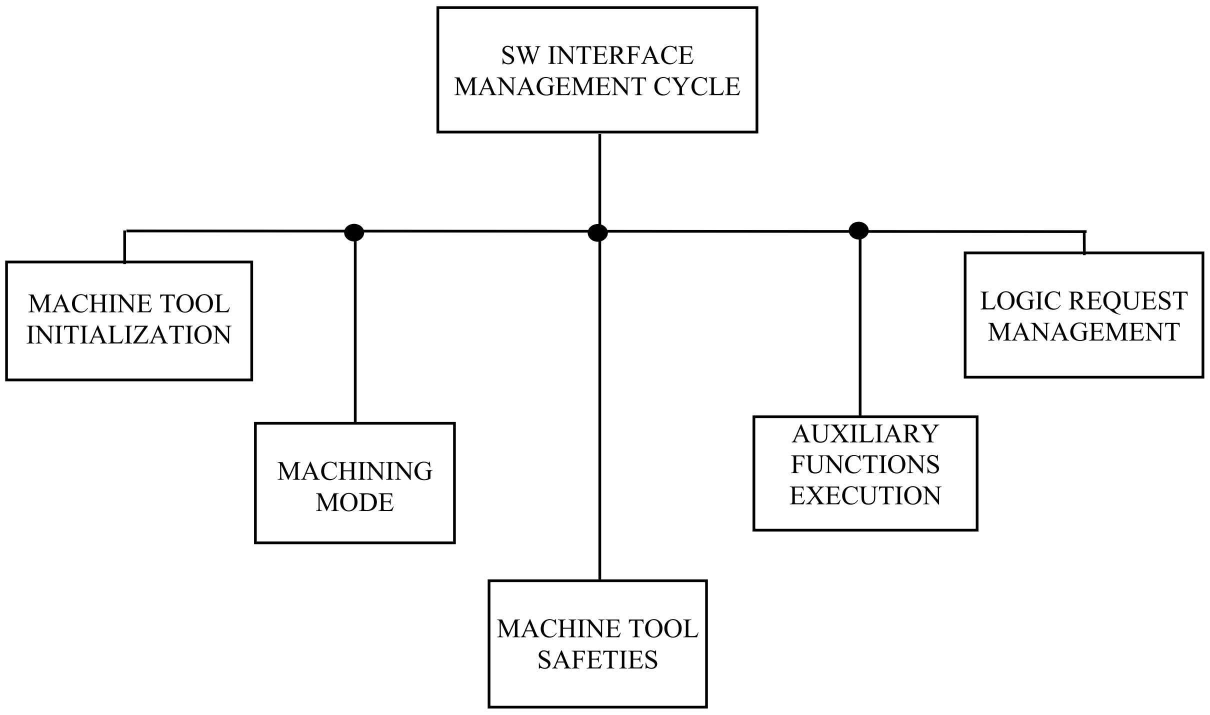

SW INTERFACE MANAGEMENT CYCLE
MACHINE TOOL INITIALIZATION
MACHINING MODE
MACHINE TOOL SAFETIES
AUXILIARY FUNCTIONS EXECUTION
LOGIC REQUEST MANAGEMENT

Fig.1. SOFTWARE MANAGEMENT

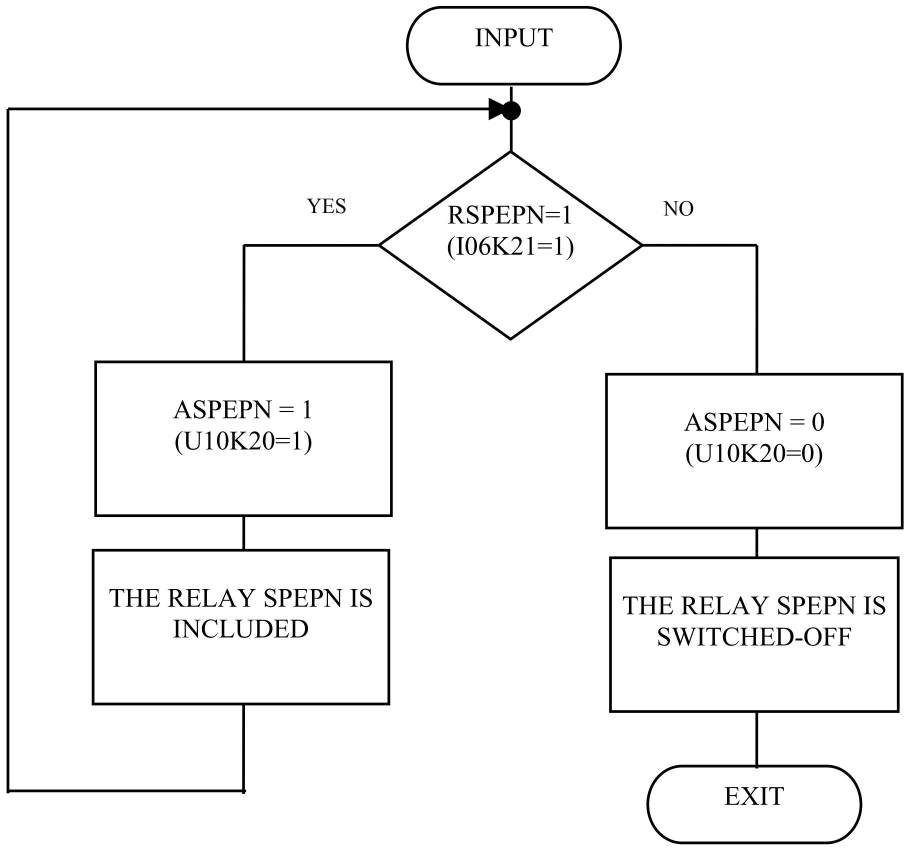


Fig.2. PROCEDURE «INCLUSION OF THE RELAY SPEPN»

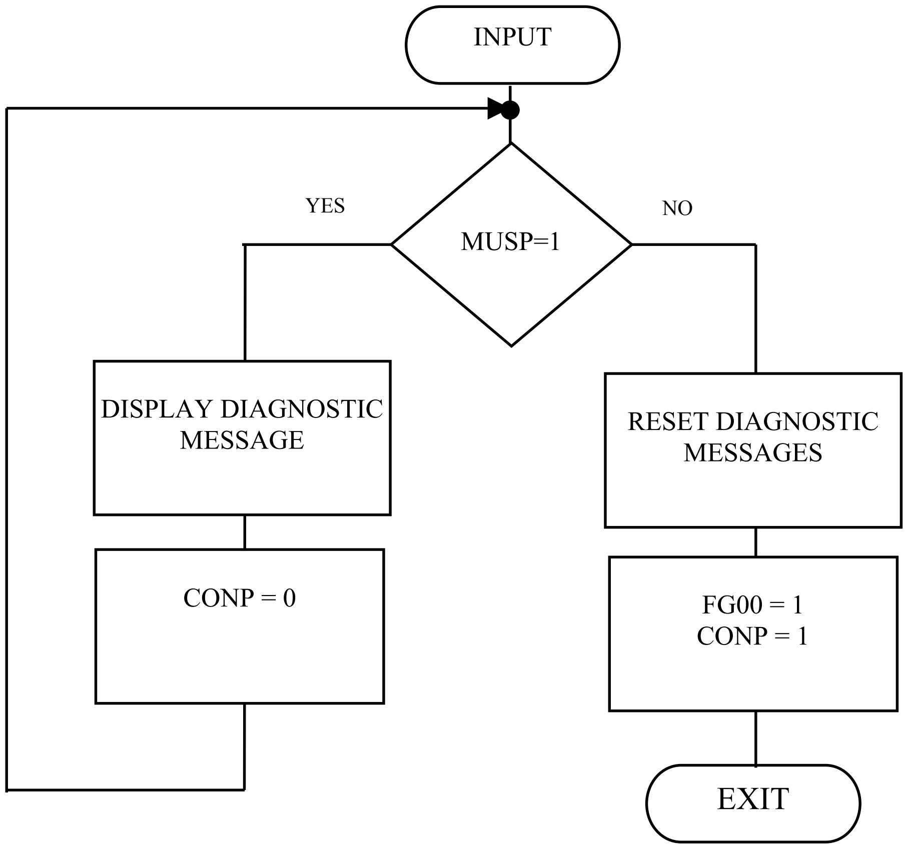


Fig.3. PROCEDURE «OPENING»

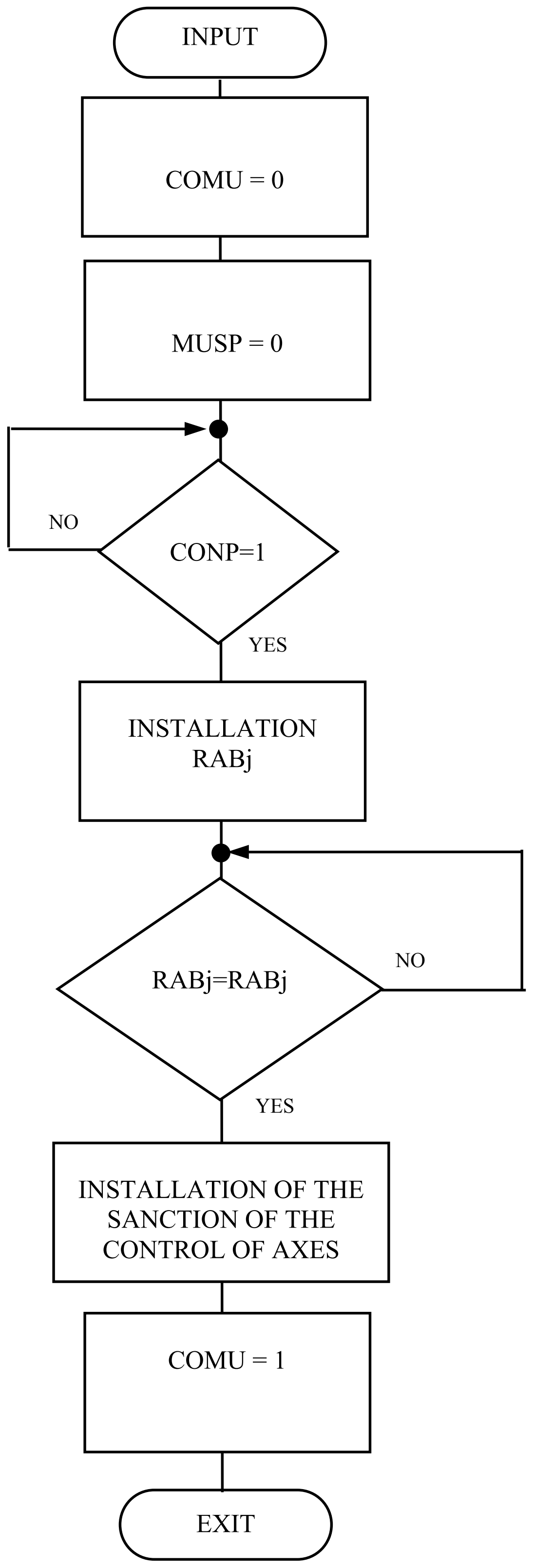


Fig.4. THE SANCTION OF AXES

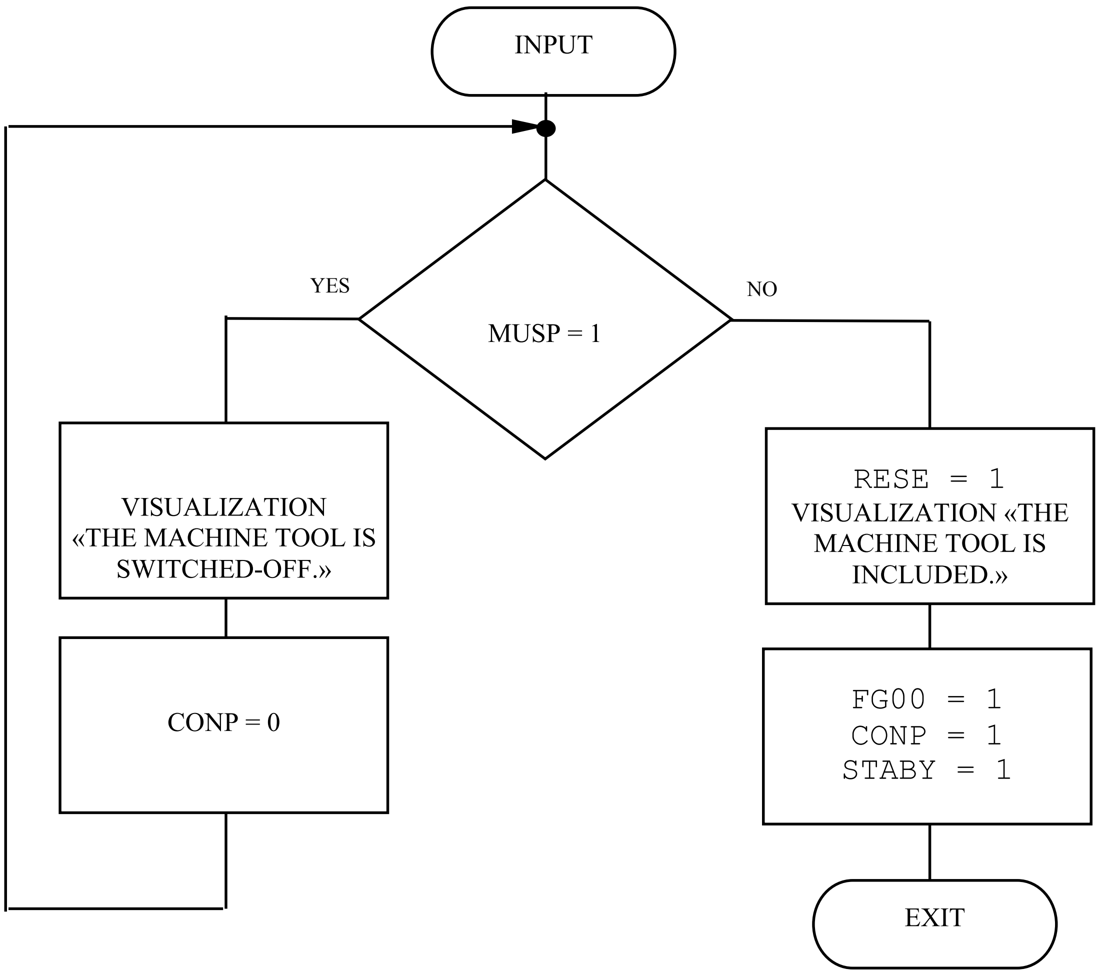

INPUT
YES
MUSP = 1
NO
VISUALIZATION «THE MACHINE TOOL IS SWITCHED-OFF.»
CONP = 0
RESE = 1
VISUALIZATION «THE MACHINE TOOL IS INCLUDED.»
FG00 = 1
CONP = 1
STABY = 1
EXIT

Fig.5. INCLUSION AFTER FAILURE

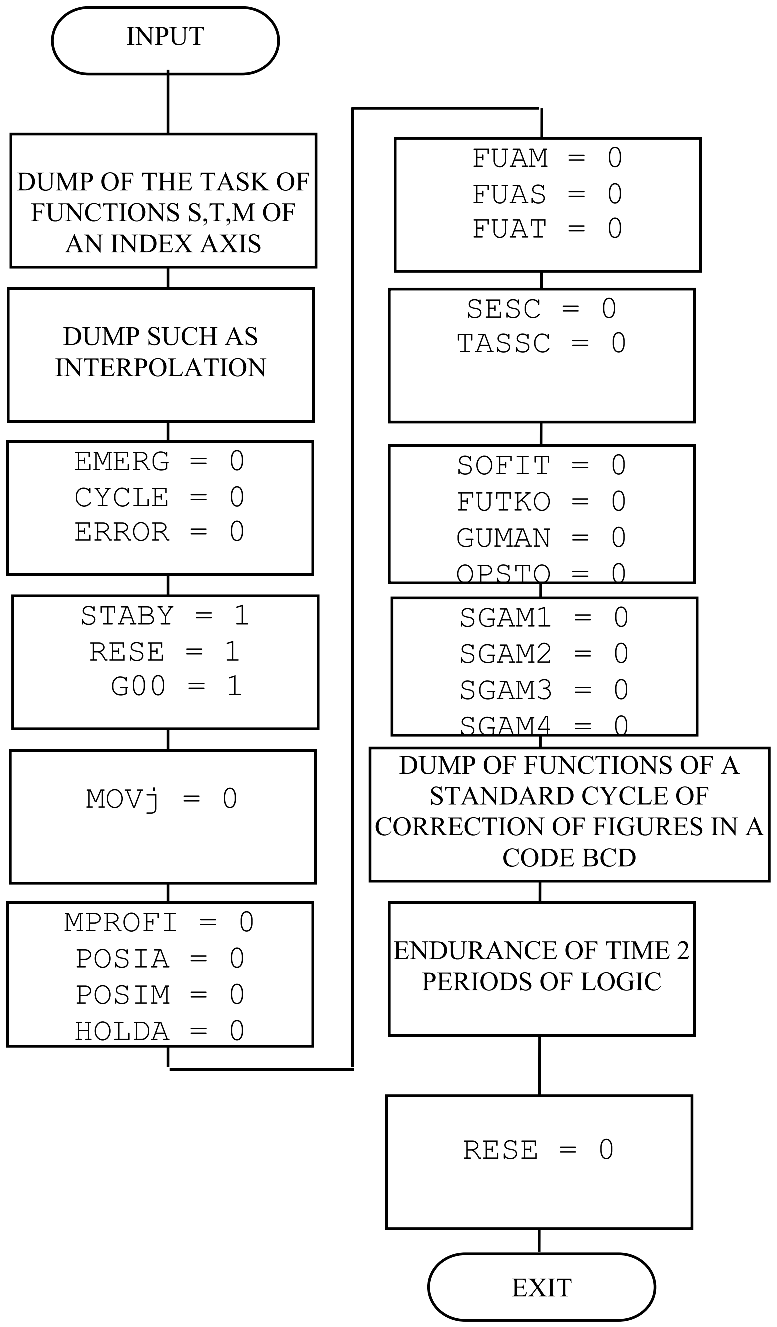


Fig.6. PROCEDURE «RESET»

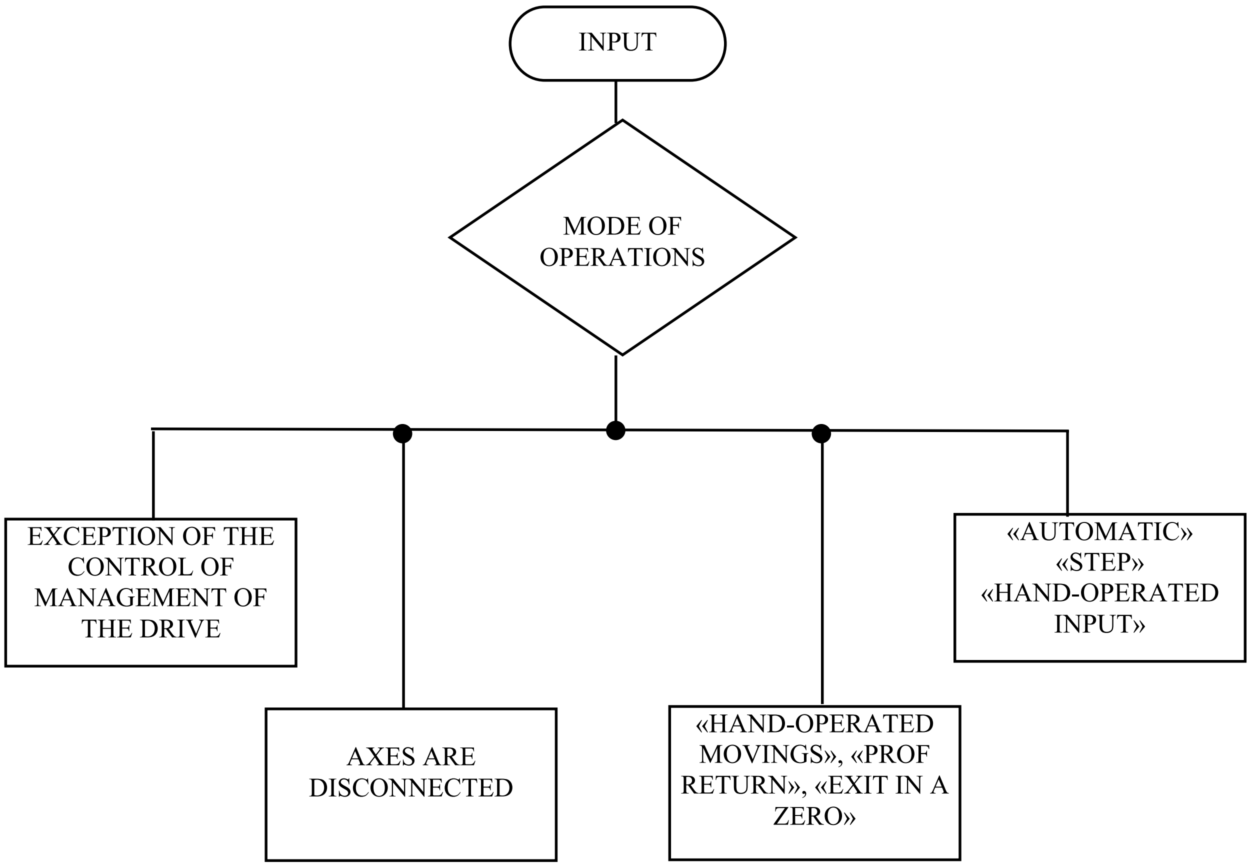

INPUT
MODE OF OPERATIONS
EXCEPTION OF THE CONTROL OF MANAGEMENT OF THE DRIVE
AXES ARE DISCONNECTED
«HAND-OPERATED MOVINGS», «PROF RETURN», «EXIT IN A ZERO»
«AUTOMATIC» «STEP» «HAND-OPERATED INPUT»

Fig.7. CYCLES OF MODES OF OPERATIONS

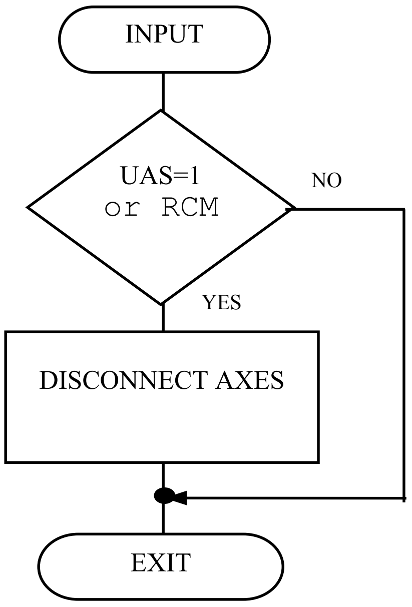


Fig.8. DISCONNECTED AXES CYCLE

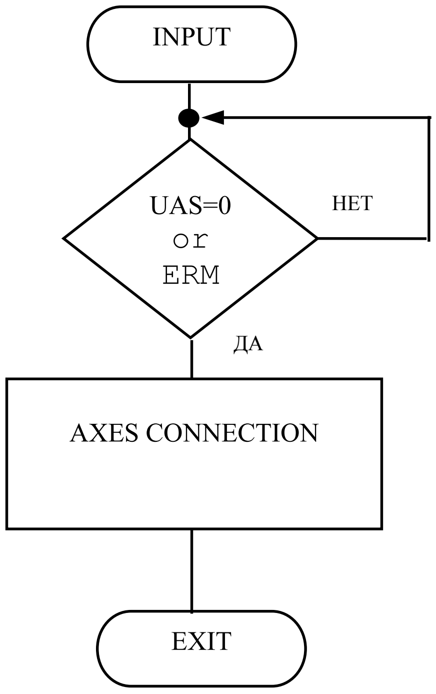


Fig.9. EXIT DISCONNECTED AXES CYCLE

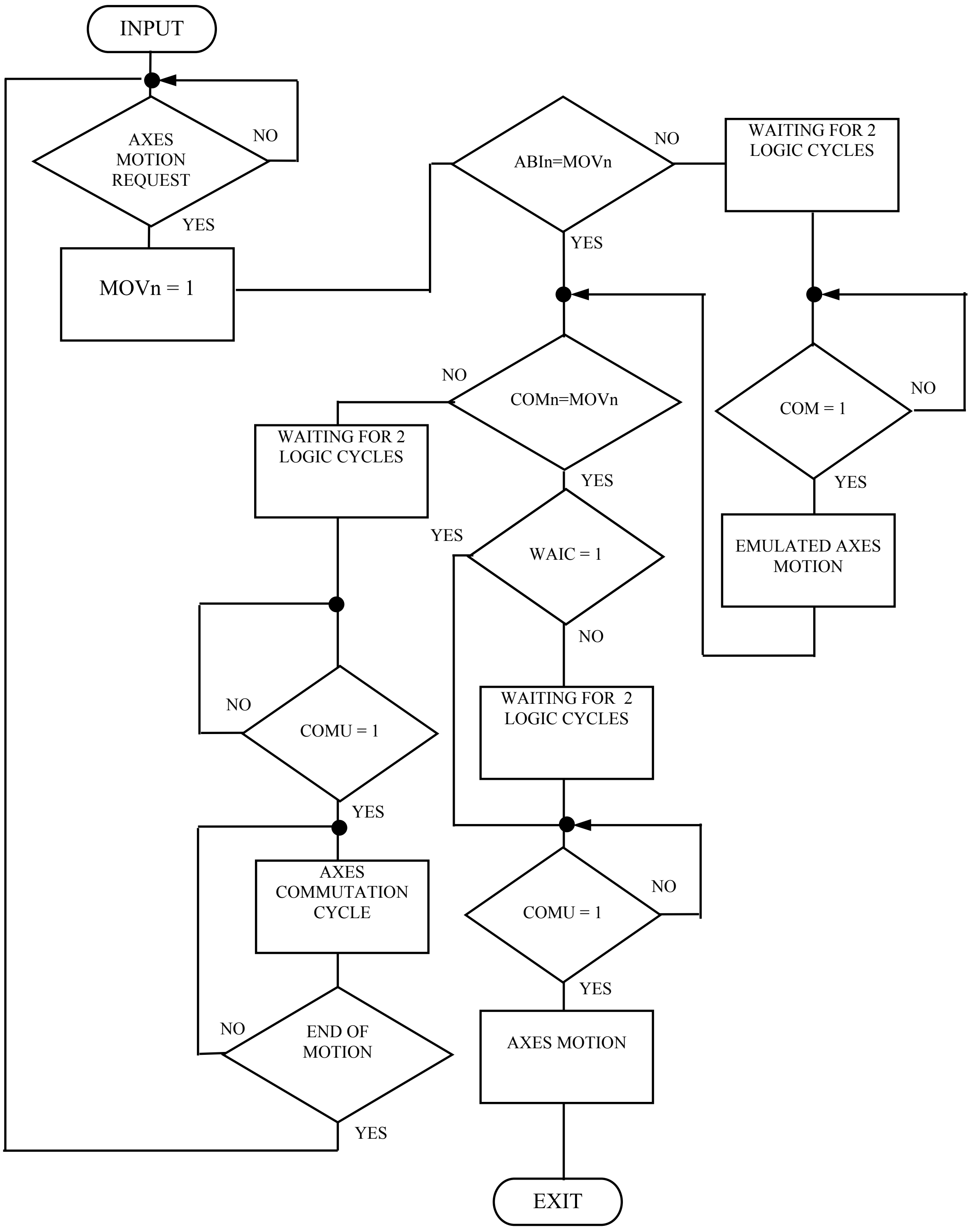


Fig.10. AXES COMMUTATION AND ENABLING CYCLE

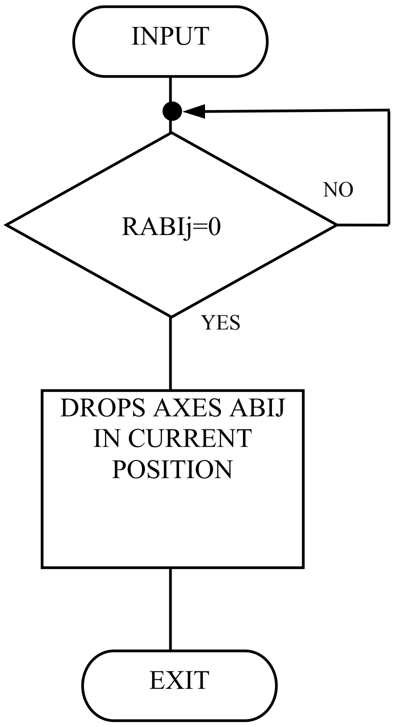


Fig.11. AXIS DROPPING CYCLE

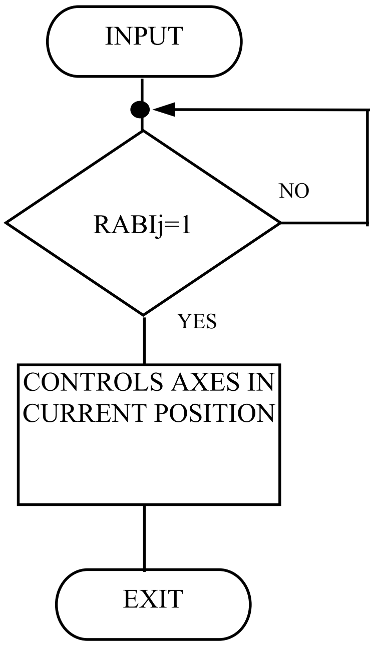


Fig.12. EXIT FROM AXIS DROPPING CYCLE

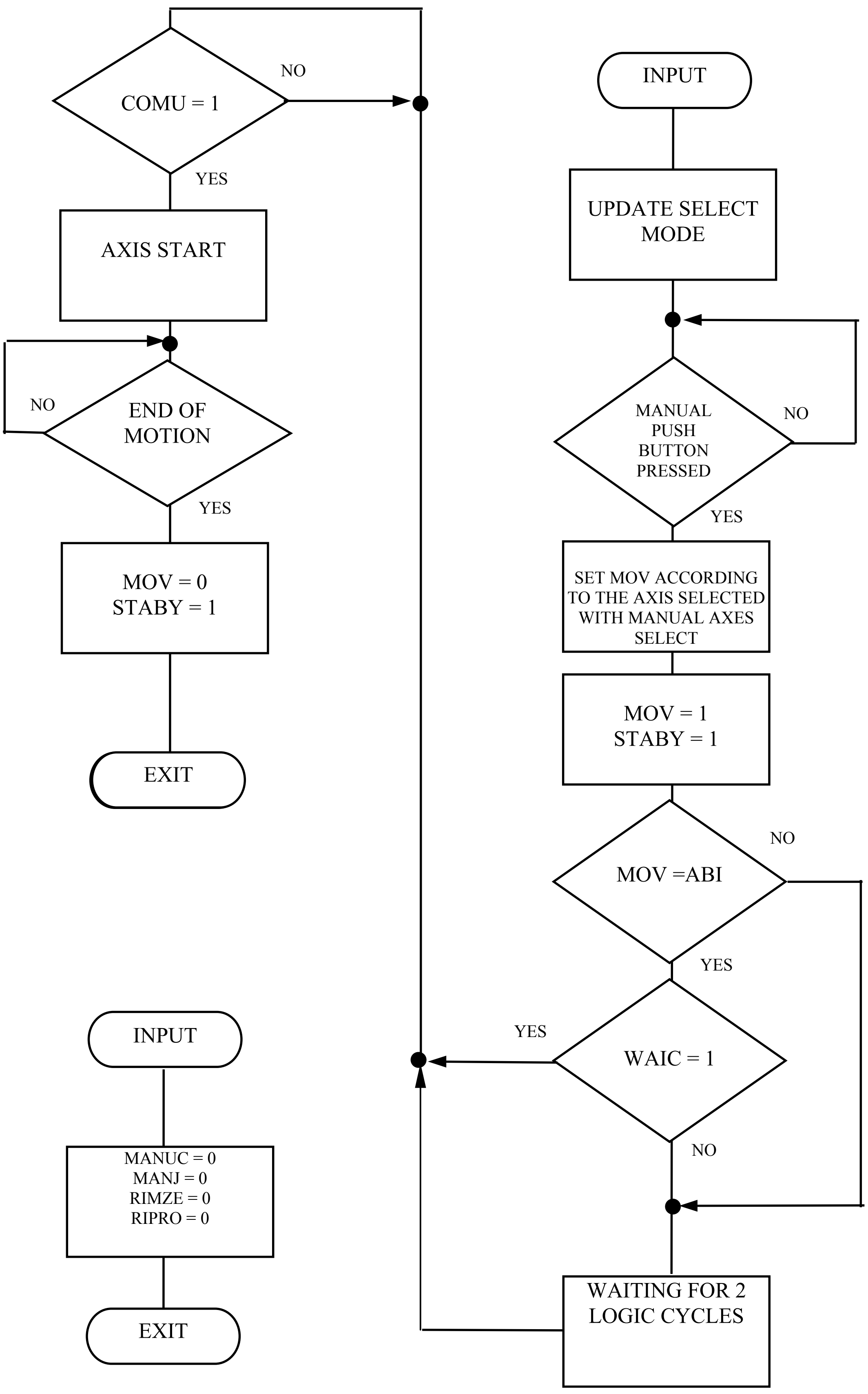


Fig.13. MANUAL MOVEMENT CYCLE

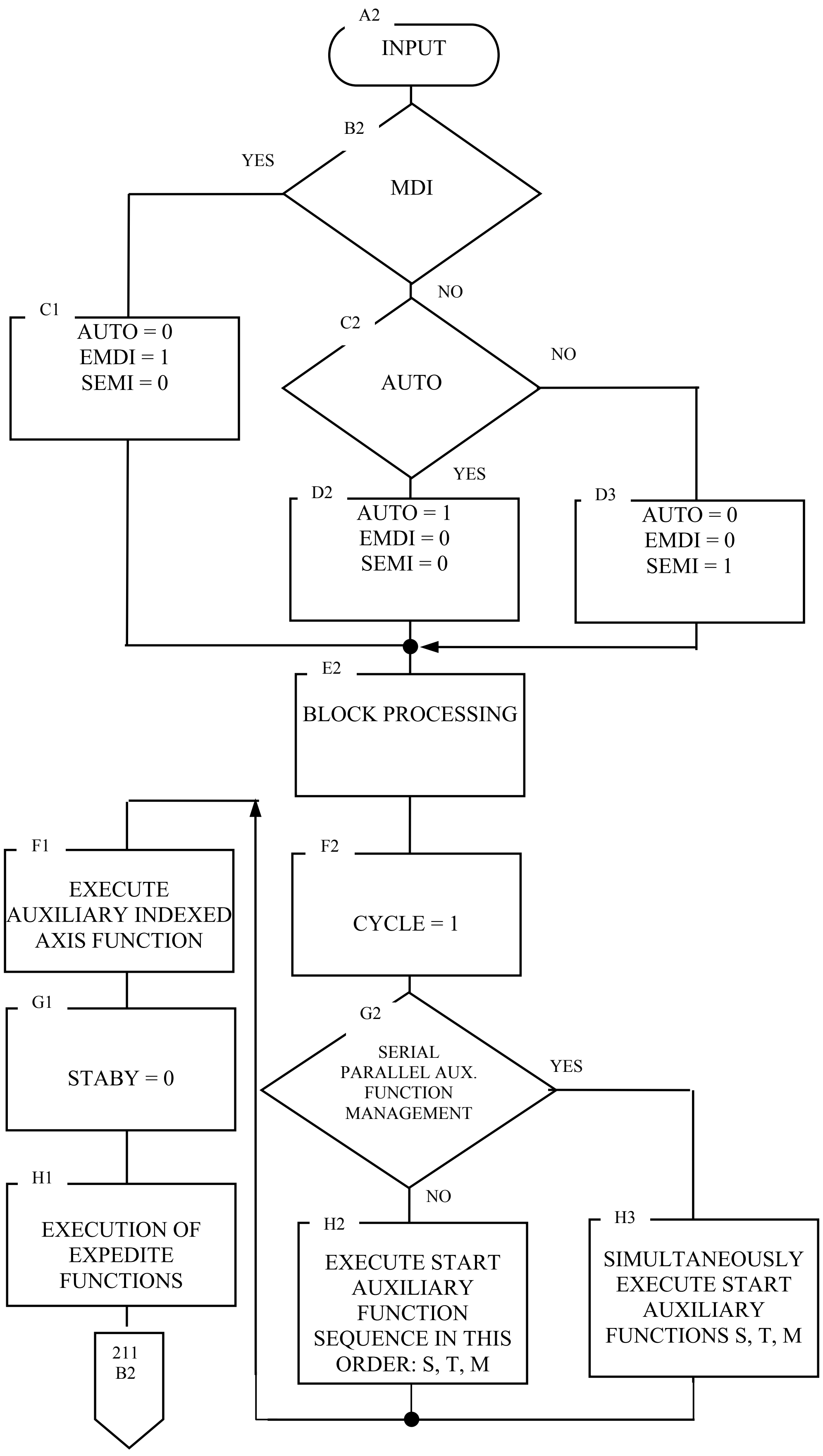
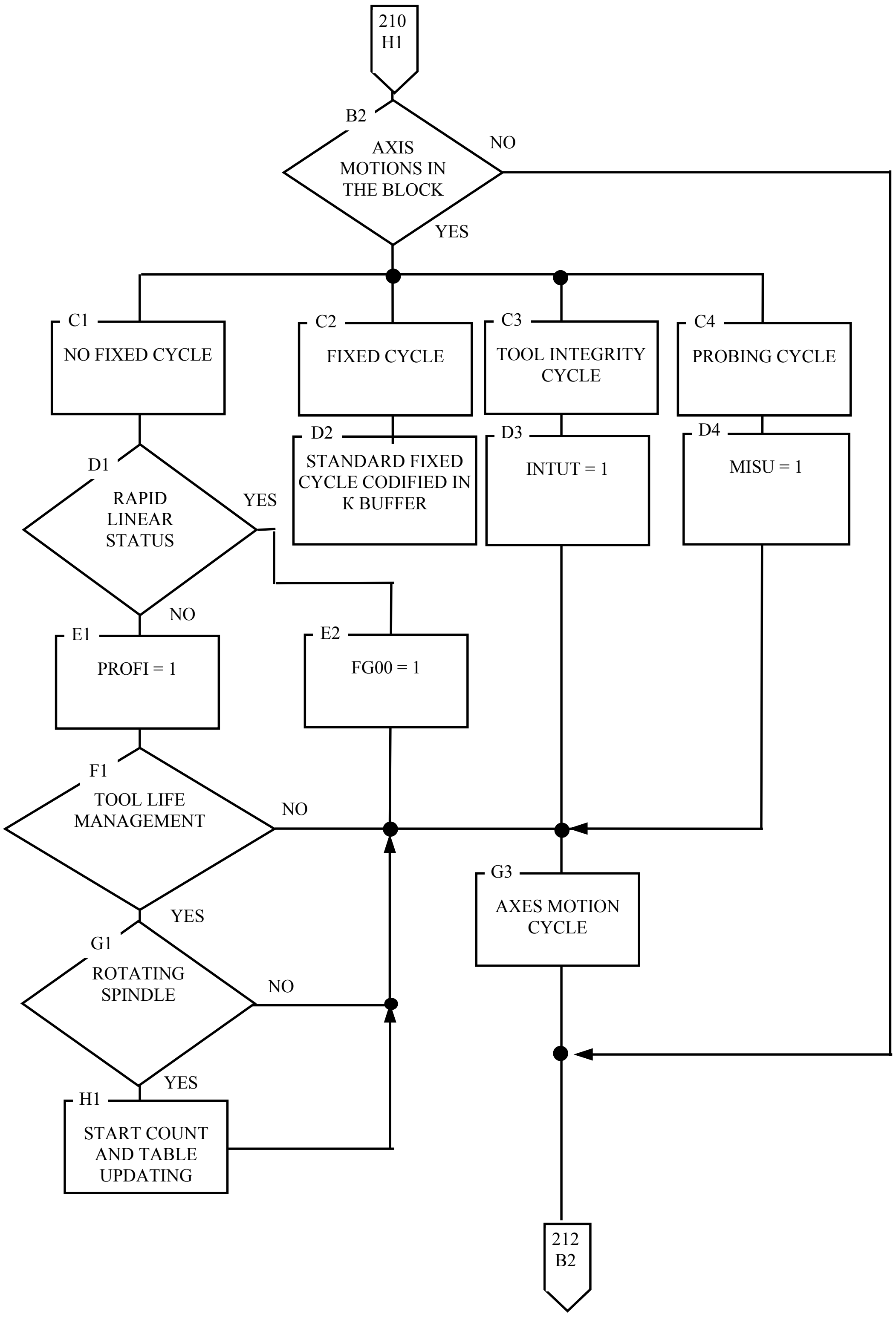
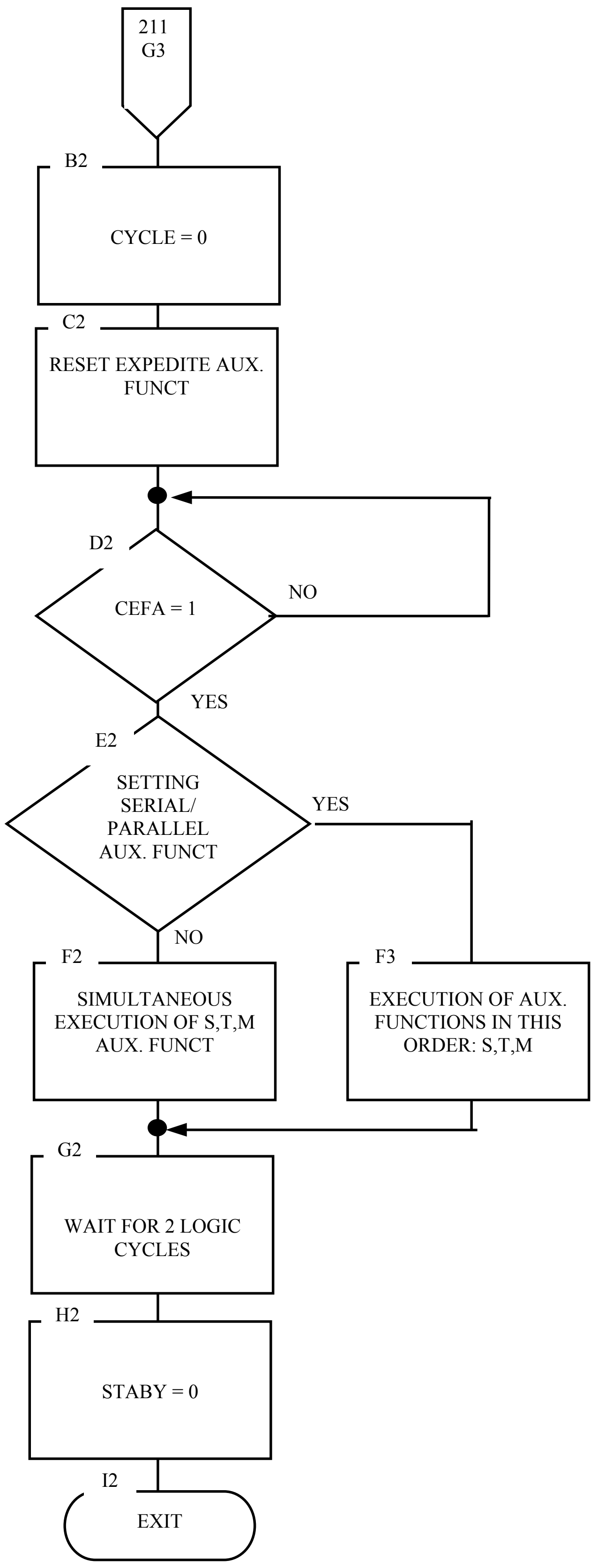

A2
INPUT
B2
MDI
YES
NO
C1
AUTO = 0
EMDI = 1
SEMI = 0
C2
AUTO
NO
YES
D2
AUTO = 1
EMDI = 0
SEMI = 0
D3
AUTO = 0
EMDI = 0
SEMI = 1
E2
BLOCK PROCESSING
F1
EXECUTE AUXILIARY INDEXED AXIS FUNCTION
F2
CYCLE = 1
G1
STABY = 0
G2
SERIAL PARALLEL AUX. FUNCTION MANAGEMENT
YES
NO
H1
EXECUTION OF EXPEDITE FUNCTIONS
H2
EXECUTE START AUXILIARY FUNCTION SEQUENCE IN THIS ORDER: S, T, M
H3
SIMULTANEOUSLY EXECUTE START AUXILIARY FUNCTIONS S, T, M
211
B2

Fig.14. AUTO-SEMIAUTO-MDI CYCLE


210
H1
B2
AXIS MOTIONS IN THE BLOCK
NO
YES
C1
NO FIXED CYCLE
C2
FIXED CYCLE
C3
TOOL INTEGRITY CYCLE
C4
PROBING CYCLE
D1
RAPID LINEAR STATUS
YES
D2
STANDARD FIXED CYCLE CODIFIED IN K BUFFER
D3
INTUT = 1
D4
MISU = 1
NO
E1
PROFI = 1
E2
FG00 = 1
F1
TOOL LIFE MANAGEMENT
NO
YES
G1
ROTATING SPINDLE
NO
YES
G3
AXES MOTION CYCLE
H1
START COUNT AND TABLE UPDATING
212
B2


211
G3
B2
CYCLE = 0
C2
RESET EXPEDITE AUX. FUNCT
D2
CEFA = 1
NO
YES
E2
SETTING SERIAL/ PARALLEL AUX. FUNCT
YES
NO
F2
SIMULTANEOUS EXECUTION OF S,T,M AUX. FUNCT
F3
EXECUTION OF AUX. FUNCTIONS IN THIS ORDER: S,T,M
G2
WAIT FOR 2 LOGIC CYCLES
H2
STABY = 0
I2
EXIT

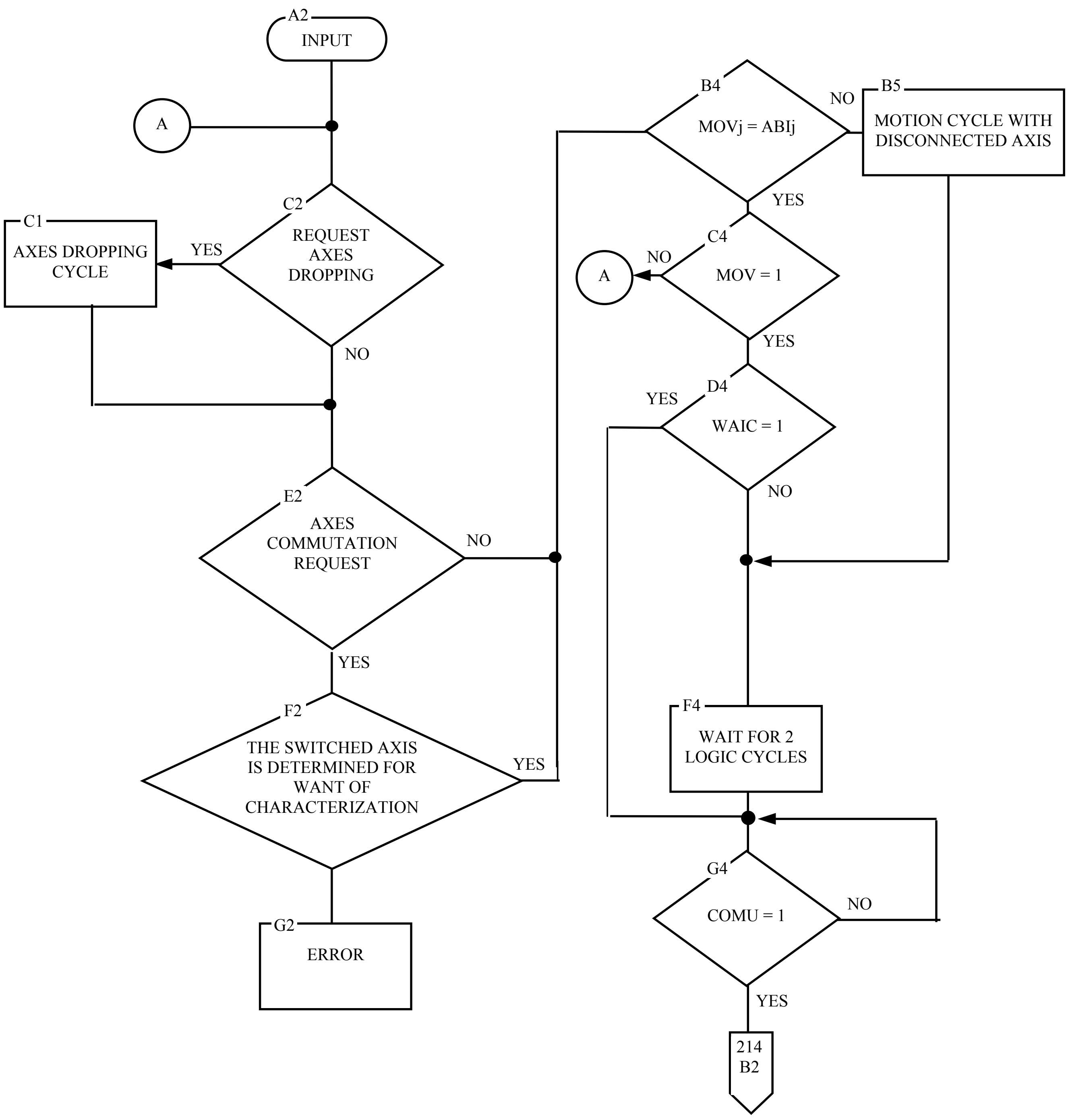
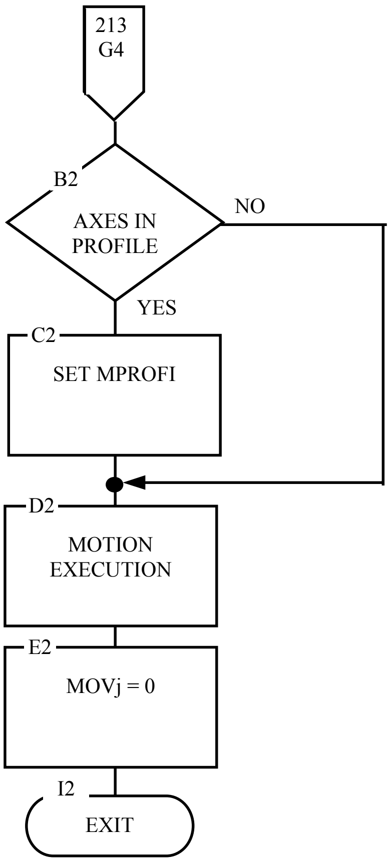


Fig.15. AXES MOTION CYCLE


213
G4
B2
AXES IN PROFILE
NO
YES
C2
SET MPROFI
D2
MOTION EXECUTION
E2
MOVj = 0
I2
EXIT

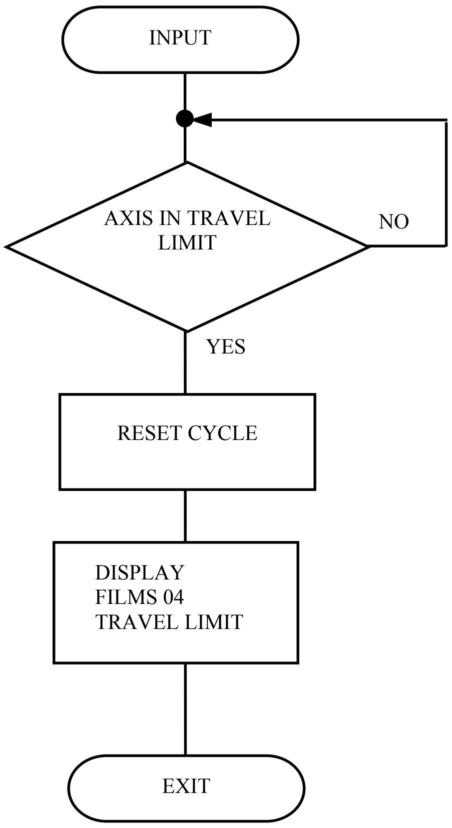


Fig.16. TRAVEL LIMIT ENTRANCE CYCLE

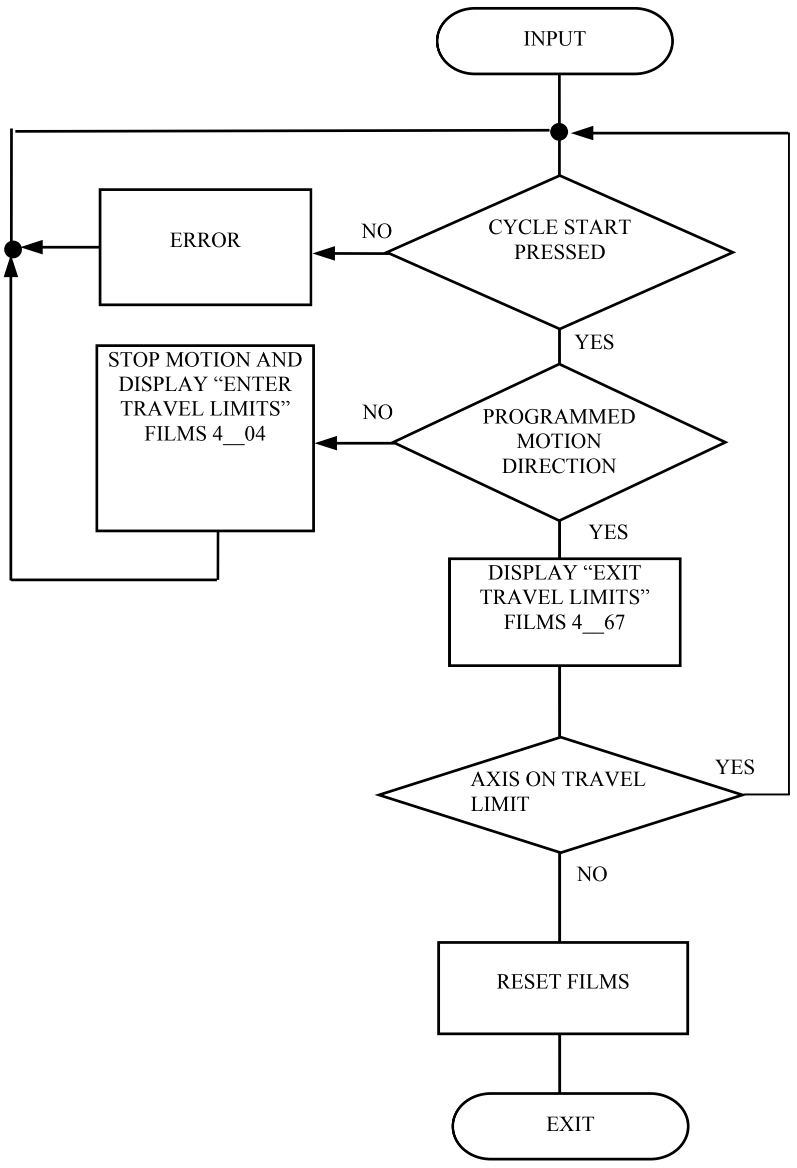

INPUT
CYCLE START PRESSED
NO
ERROR
YES
PROGRAMMED MOTION DIRECTION
NO
STOP MOTION AND DISPLAY "ENTER TRAVEL LIMITS" FILMS 4__04
YES
DISPLAY "EXIT TRAVEL LIMITS" FILMS 4__67
AXIS ON TRAVEL LIMIT
YES
NO
RESET FILMS
EXIT

Fig.17. TRAVEL LIMIT EXIT CYCLE

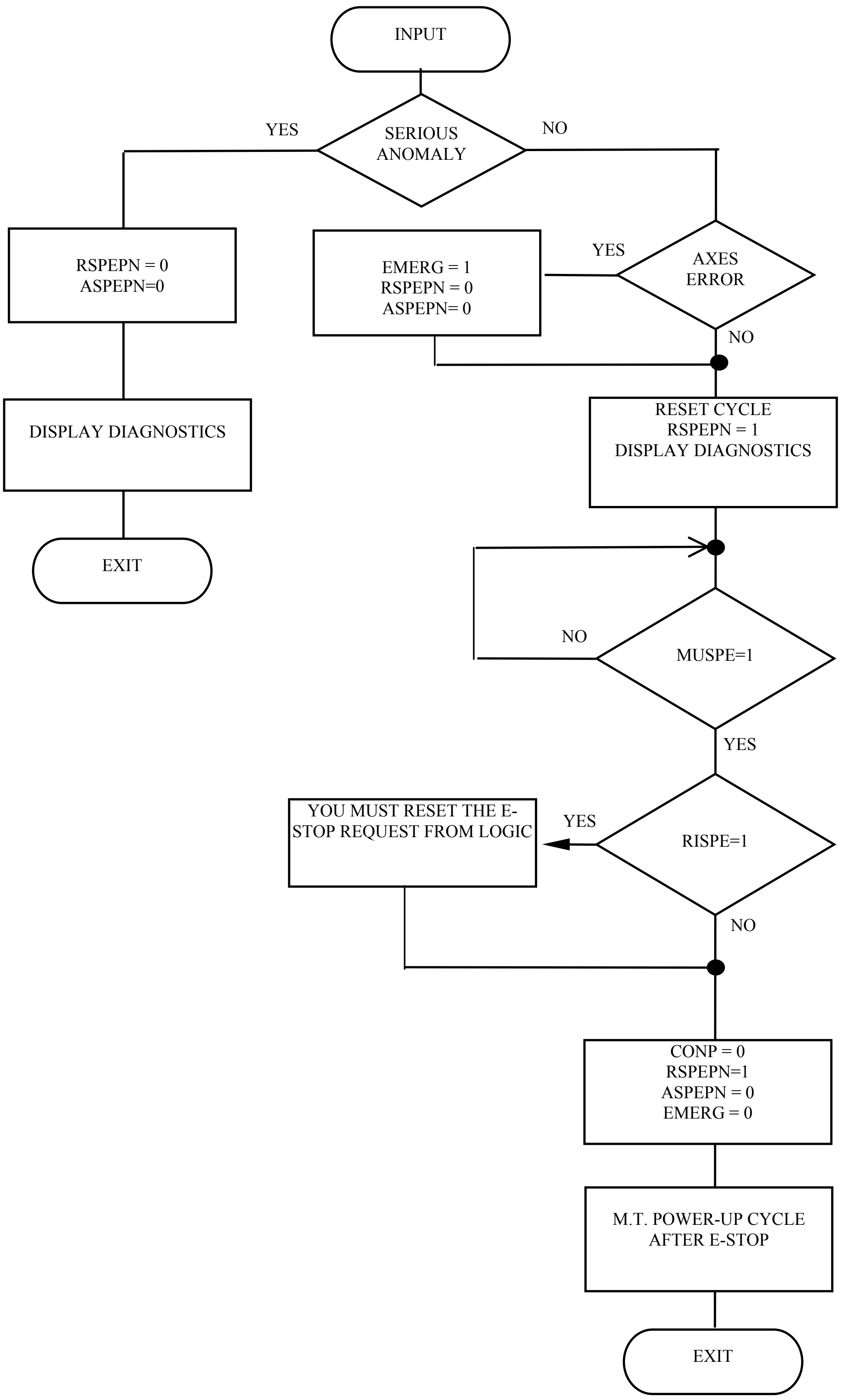


Fig.18. E-STOP CYCLE

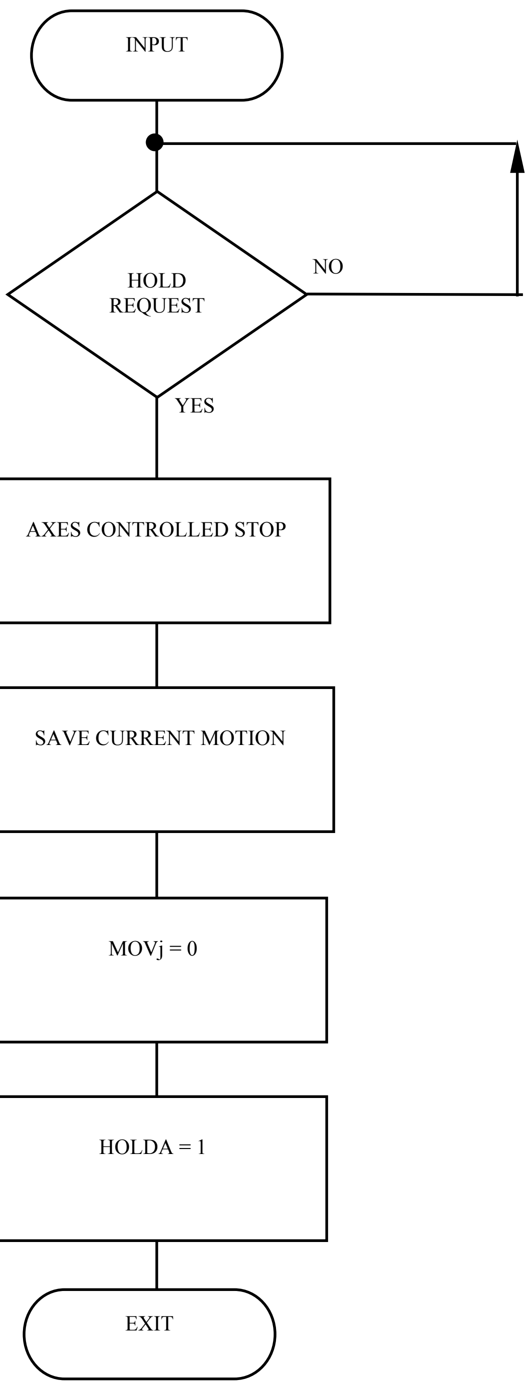


Fig.19. HOLD CYCLE

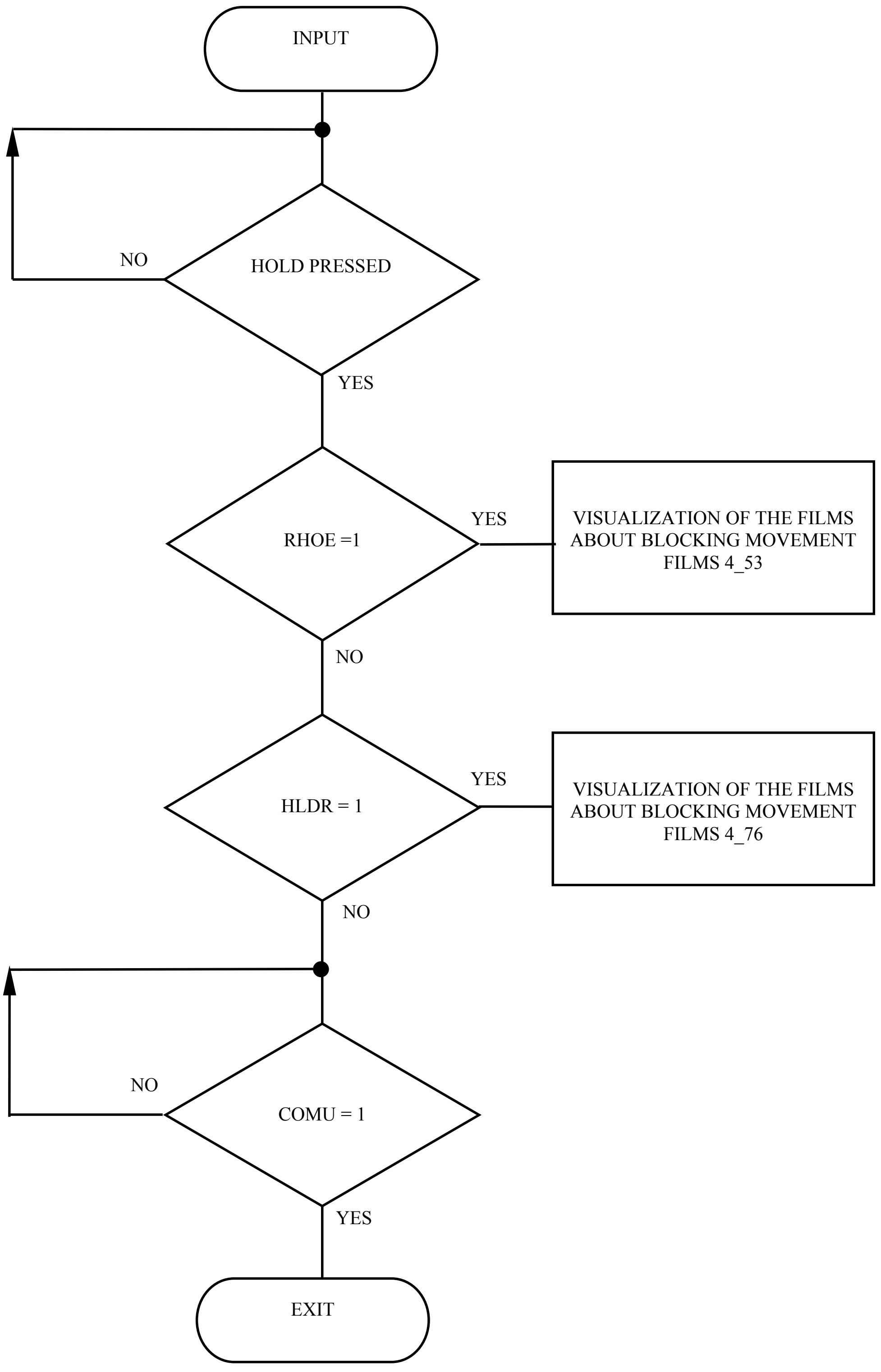

INPUT
HOLD PRESSED
NO
YES
RHOE =1
YES
VISUALIZATION OF THE FILMS ABOUT BLOCKING MOVEMENT FILMS 4_53
NO
HLDR = 1
YES
VISUALIZATION OF THE FILMS ABOUT BLOCKING MOVEMENT FILMS 4_76
NO
COMU = 1
NO
YES
EXIT

Fig.20. CONSENT FOR HOLD EXIT

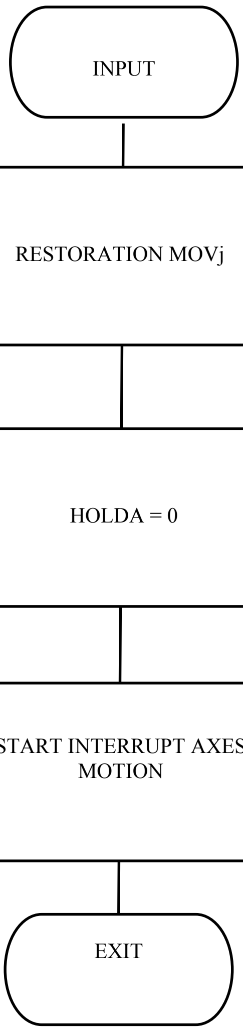


Fig.21. HOLD EXIT CYCLE

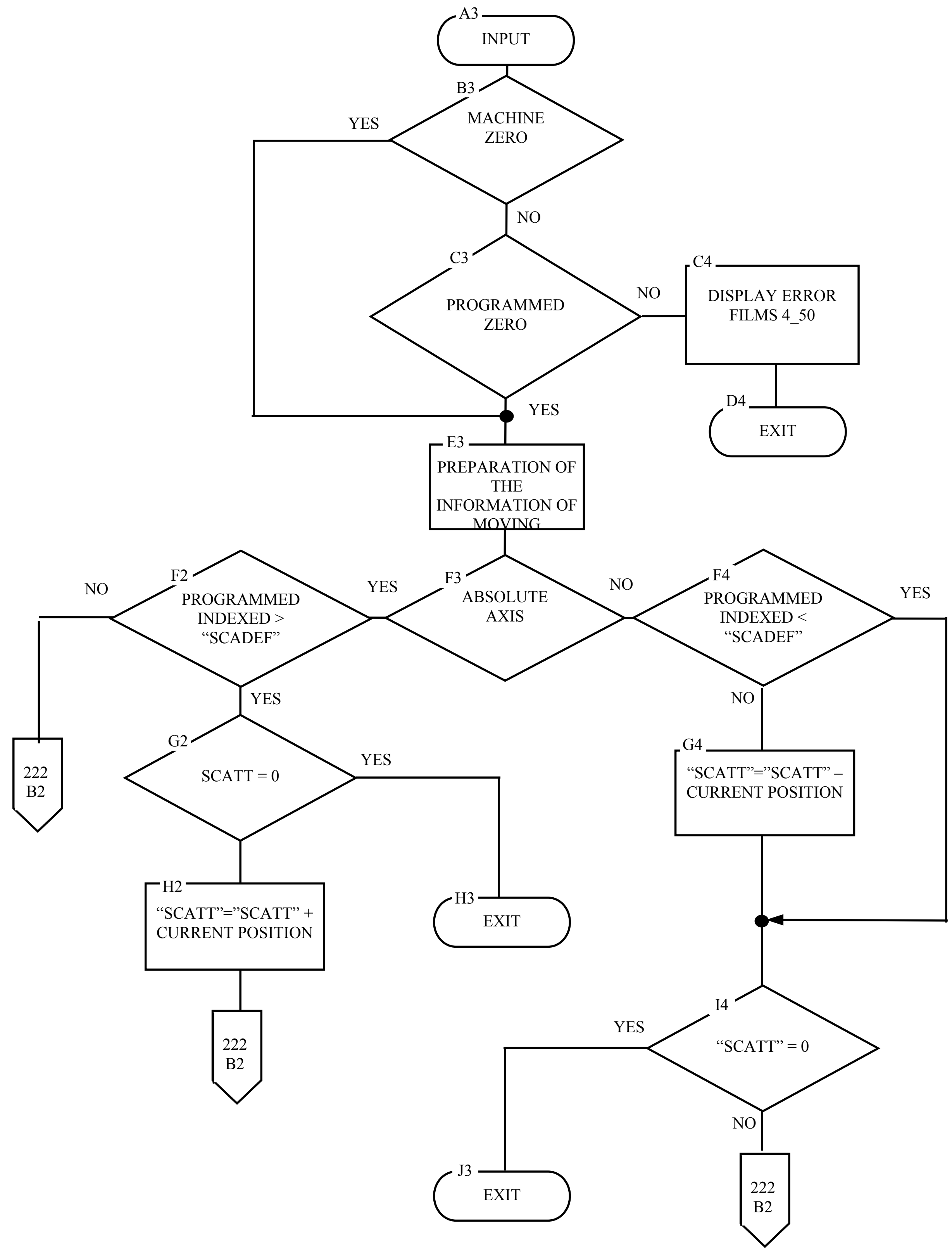
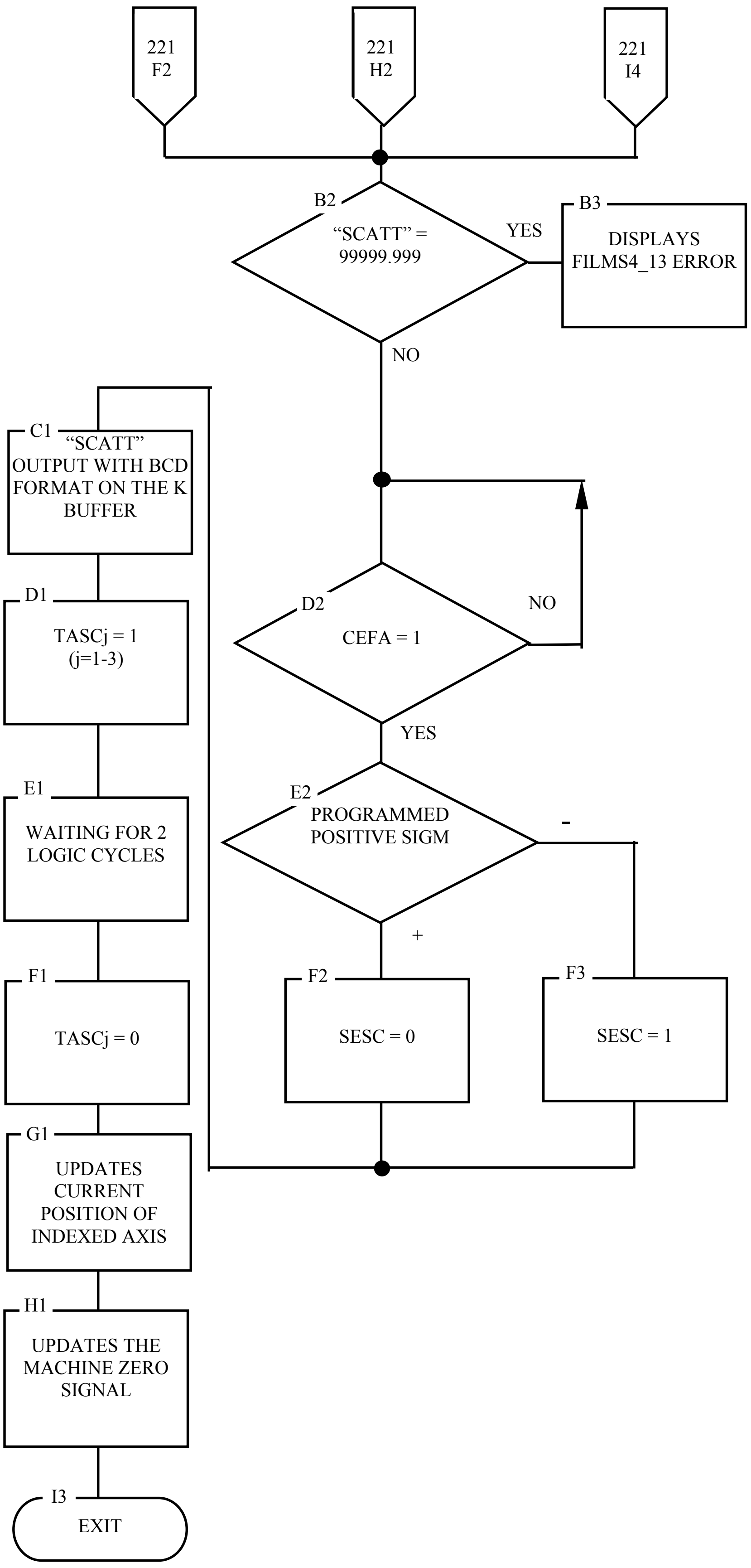


Fig.22. INDEXED AXES


221
F2
221
H2
221
I4
B2
"SCATT" =
99999.999
YES
B3
DISPLAYS
FILMS4_13 ERROR
NO
C1
"SCATT"
OUTPUT WITH BCD
FORMAT ON THE K
BUFFER
D1
TASCj = 1
(j=1-3)
D2
CEFA = 1
NO
YES
E1
WAITING FOR 2
LOGIC CYCLES
E2
PROGRAMMED
POSITIVE SIGM
-
+
F1
TASCj = 0
F2
SESC = 0
F3
SESC = 1
G1
UPDATES
CURRENT
POSITION OF
INDEXED AXIS
H1
UPDATES THE
MACHINE ZERO
SIGNAL
I3
EXIT

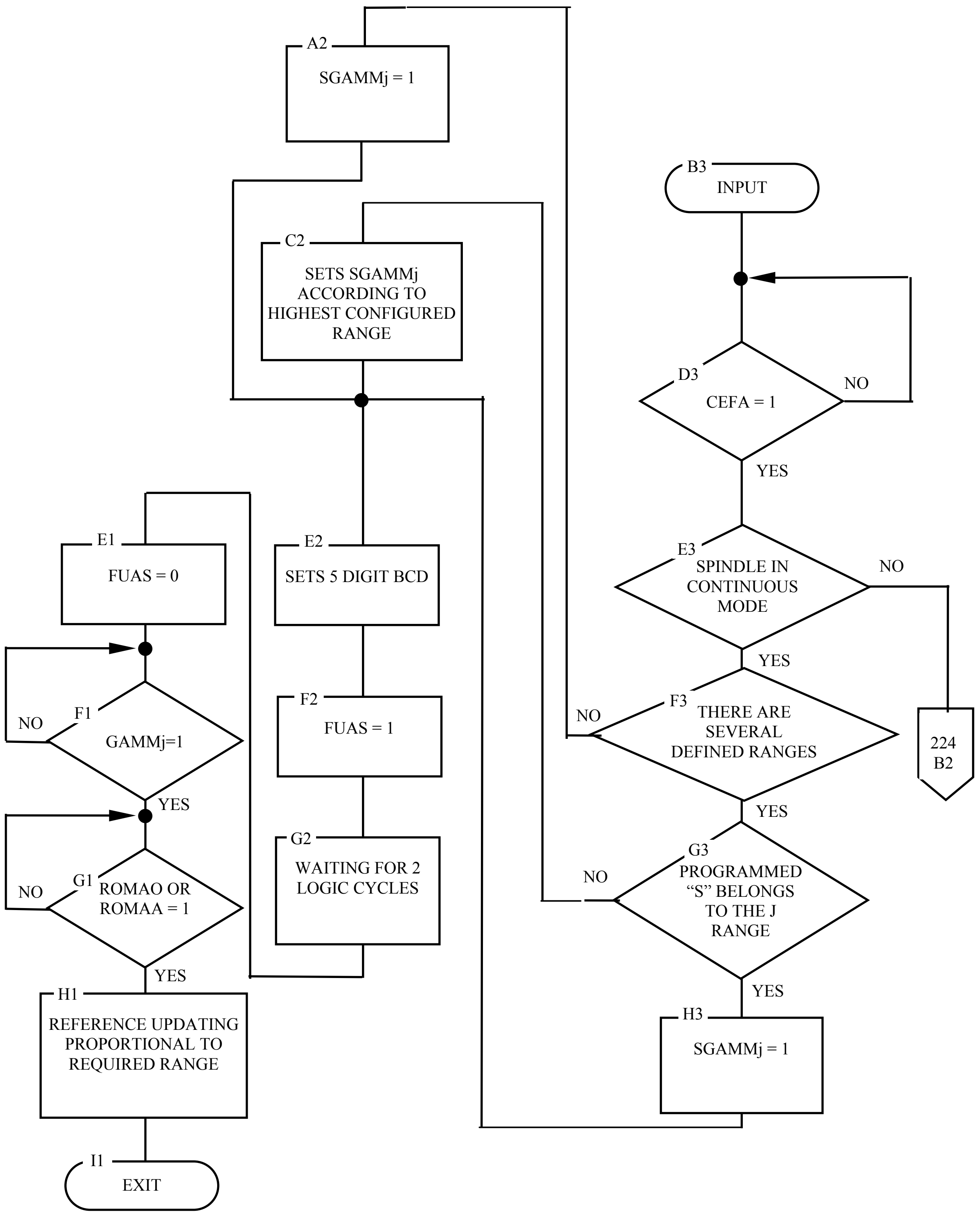
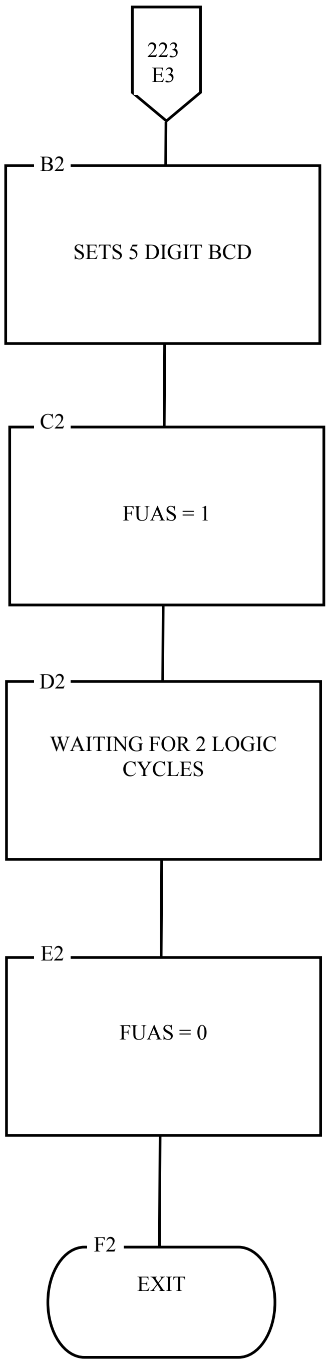

A2
SGAMMj = 1
B3
INPUT
C2
SETS SGAMMj ACCORDING TO HIGHEST CONFIGURED RANGE
D3
CEFA = 1
NO
YES
E1
FUAS = 0
E2
SETS 5 DIGIT BCD
E3
SPINDLE IN CONTINUOUS MODE
NO
YES
F1
GAMMj=1
NO
YES
F2
FUAS = 1
F3
THERE ARE SEVERAL DEFINED RANGES
NO
YES
224
B2
G1
ROMAO OR ROMAA = 1
NO
YES
G2
WAITING FOR 2 LOGIC CYCLES
G3
PROGRAMMED "S" BELONGS TO THE J RANGE
NO
YES
H1
REFERENCE UPDATING PROPORTIONAL TO REQUIRED RANGE
H3
SGAMMj = 1
I1
EXIT

Fig.23. S FUNCTIONS EXECUTION


223
E3
B2
SETS 5 DIGIT BCD
C2
FUAS = 1
D2
WAITING FOR 2 LOGIC CYCLES
E2
FUAS = 0
F2
EXIT

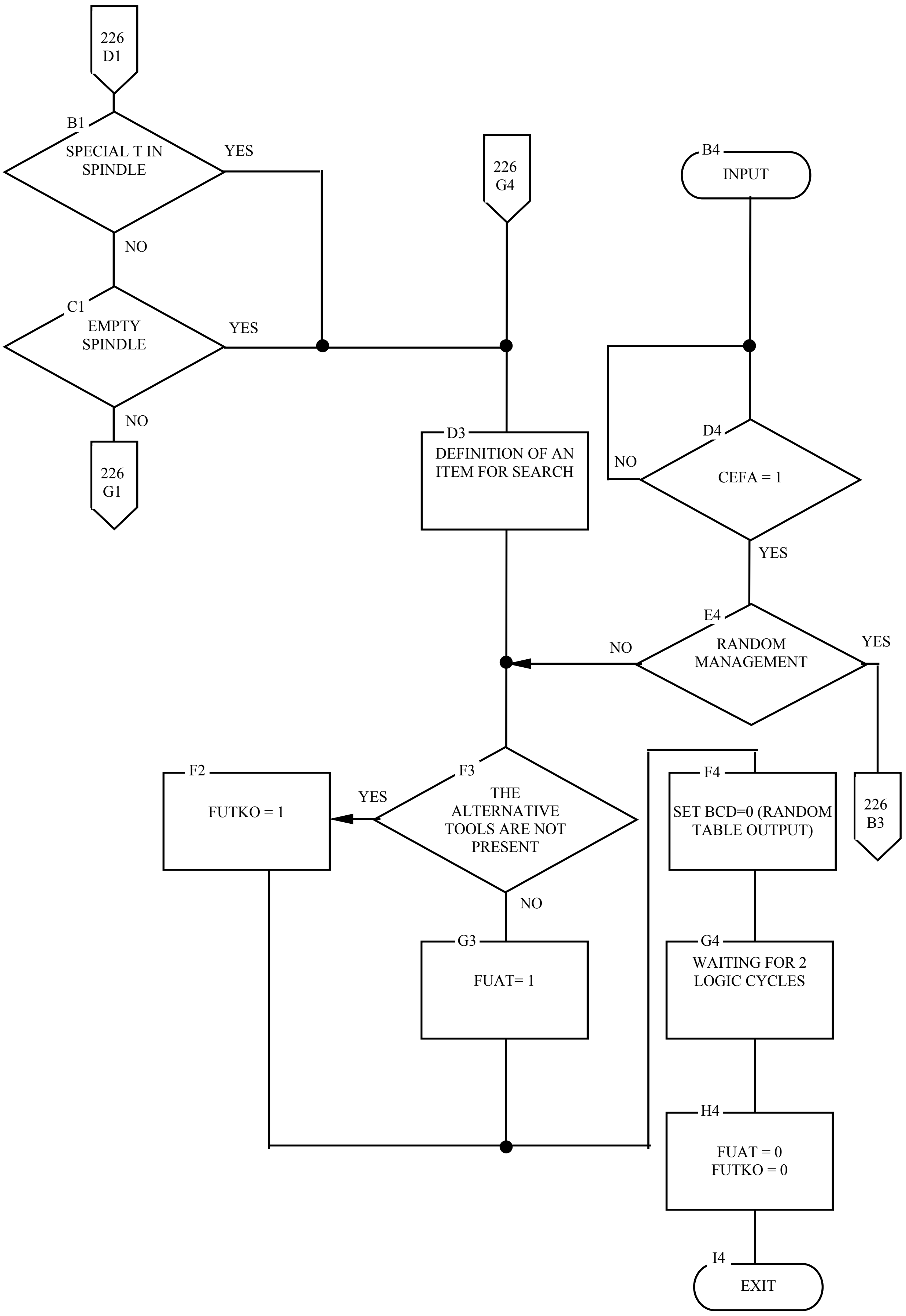
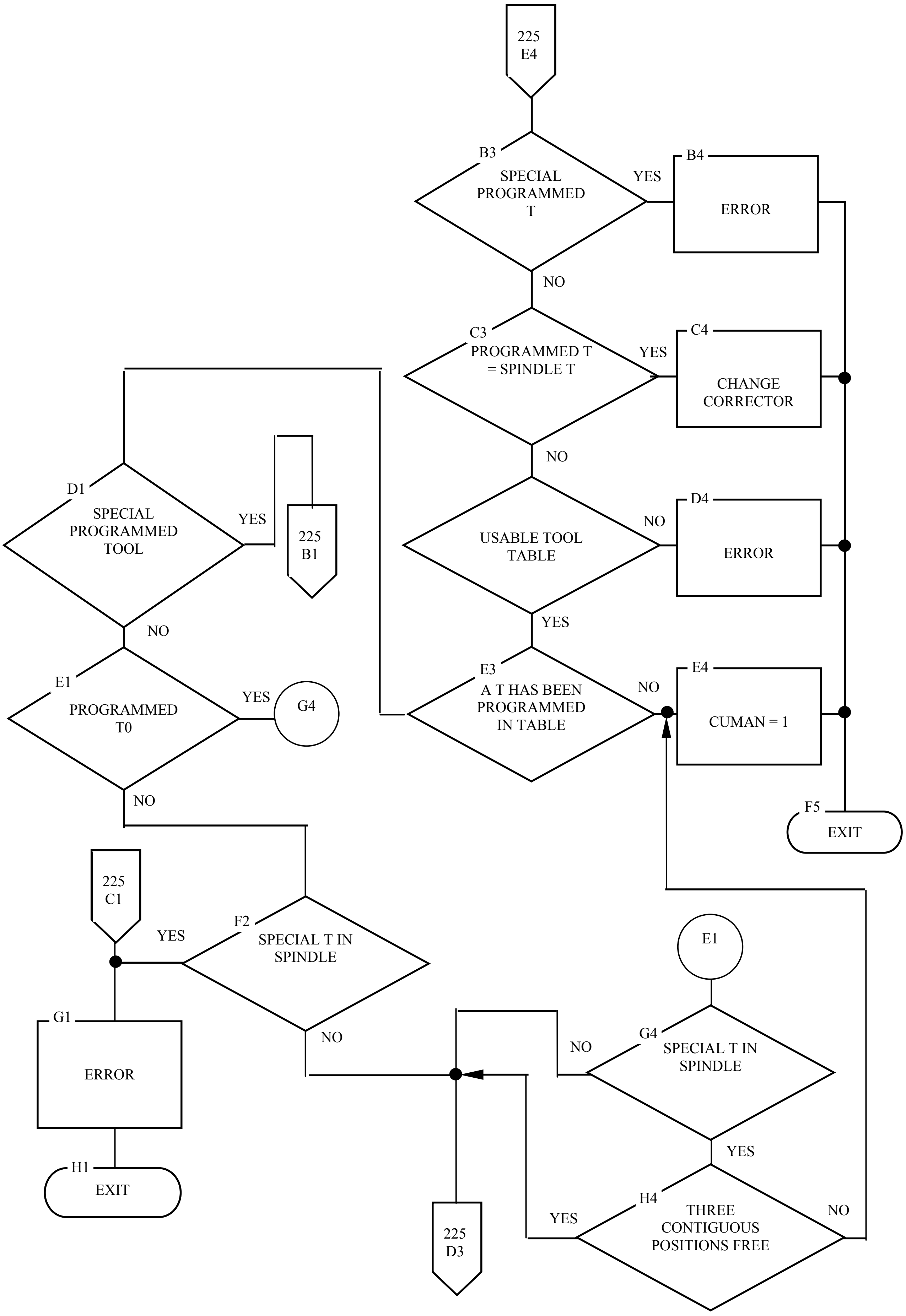


Fig.24. T FUNCTIONS EXECUTION


225
E4
B3
SPECIAL PROGRAMMED T
YES
B4
ERROR
NO
C3
PROGRAMMED T = SPINDLE T
YES
C4
CHANGE CORRECTOR
NO
USABLE TOOL TABLE
NO
D4
ERROR
YES
E3
A T HAS BEEN PROGRAMMED IN TABLE
NO
E4
CUMAN = 1
F5
EXIT
D1
SPECIAL PROGRAMMED TOOL
YES
225
B1
NO
E1
PROGRAMMED T0
YES
G4
NO
225
C1
YES
F2
SPECIAL T IN SPINDLE
NO
G1
ERROR
H1
EXIT
E1
NO
G4
SPECIAL T IN SPINDLE
YES
H4
THREE CONTIGUOUS POSITIONS FREE
YES
NO
225
D3

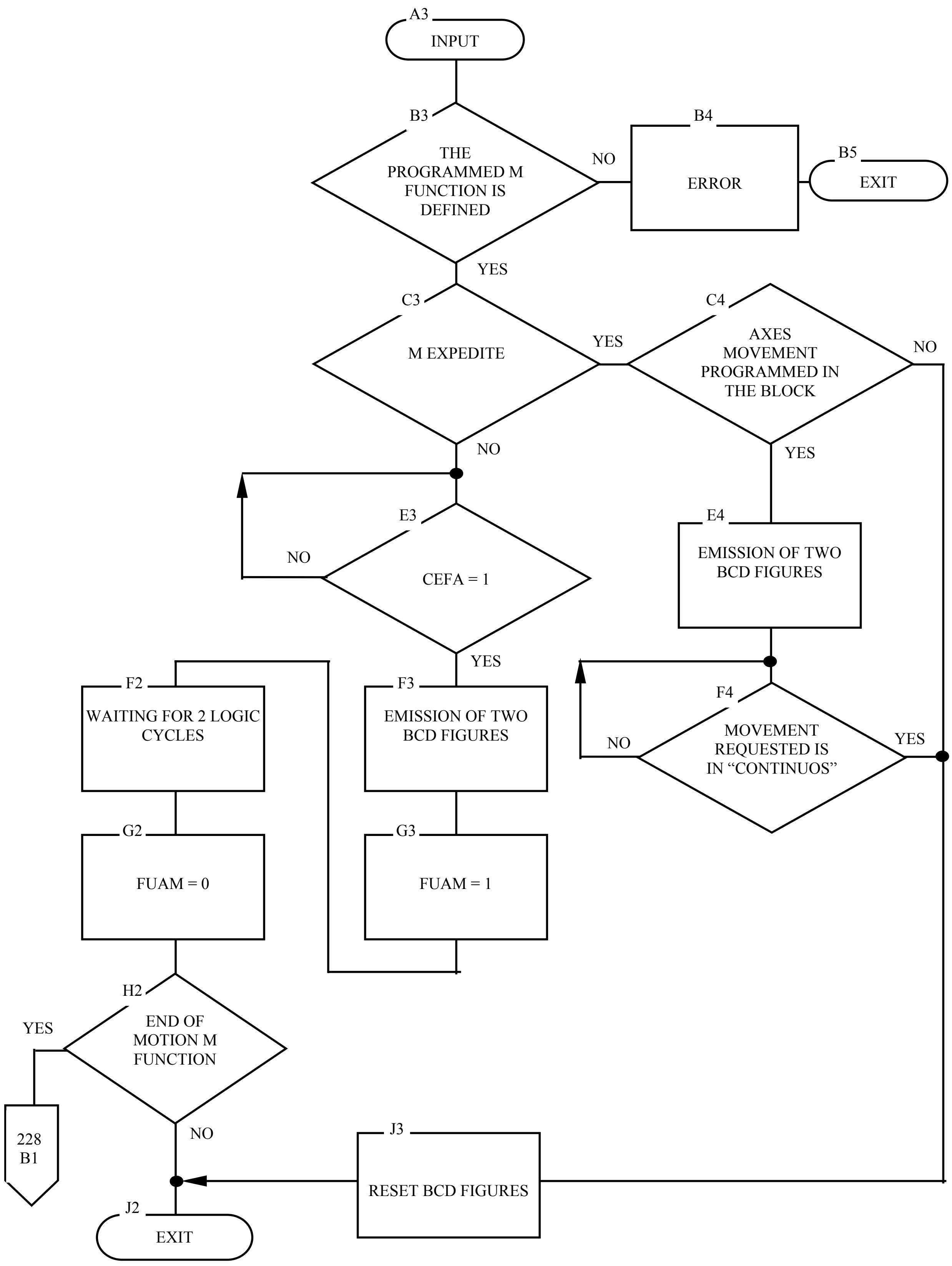
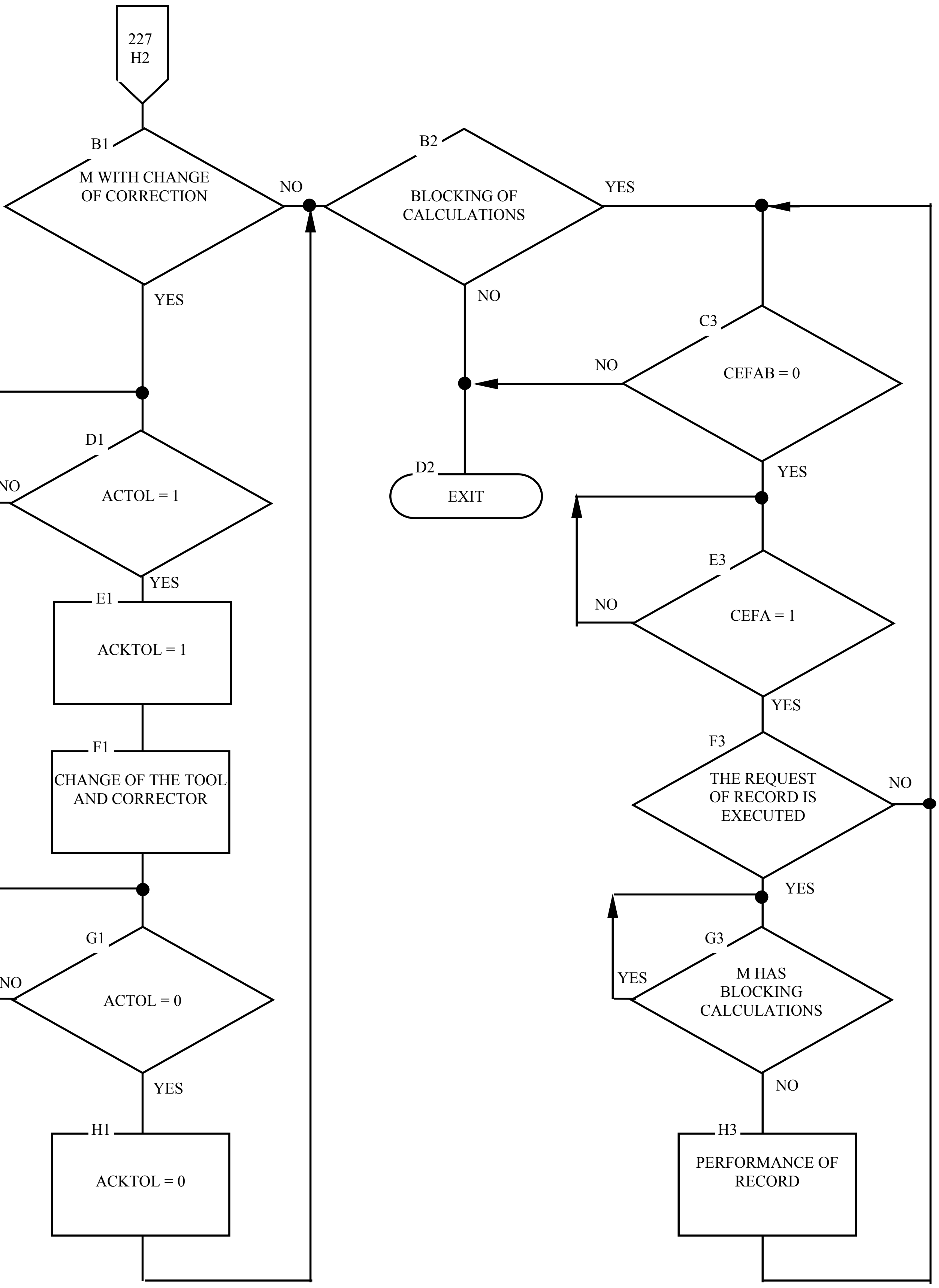

A3
INPUT
B3
THE PROGRAMMED M FUNCTION IS DEFINED
NO
B4
ERROR
B5
EXIT
YES
C3
M EXPEDITE
YES
C4
AXES MOVEMENT PROGRAMMED IN THE BLOCK
NO
NO
YES
E3
NO
CEFA = 1
E4
EMISSION OF TWO BCD FIGURES
YES
F2
WAITING FOR 2 LOGIC CYCLES
F3
EMISSION OF TWO BCD FIGURES
F4
NO
MOVEMENT REQUESTED IS IN "CONTINUOS"
YES
G2
FUAM = 0
G3
FUAM = 1
H2
YES
END OF MOTION M FUNCTION
228 B1
NO
J3
RESET BCD FIGURES
J2
EXIT

Fig.25. EXECUTION OF M FUNCTIONS


227
H2
B1
M WITH CHANGE OF CORRECTION
NO
YES
B2
BLOCKING OF CALCULATIONS
YES
NO
C3
CEFAB = 0
NO
YES
D1
ACTOL = 1
NO
YES
D2
EXIT
E1
ACKTOL = 1
E3
CEFA = 1
NO
YES
F1
CHANGE OF THE TOOL AND CORRECTOR
F3
THE REQUEST OF RECORD IS EXECUTED
NO
YES
G1
ACTOL = 0
NO
YES
G3
M HAS BLOCKING CALCULATIONS
YES
NO
H1
ACKTOL = 0
H3
PERFORMANCE OF RECORD

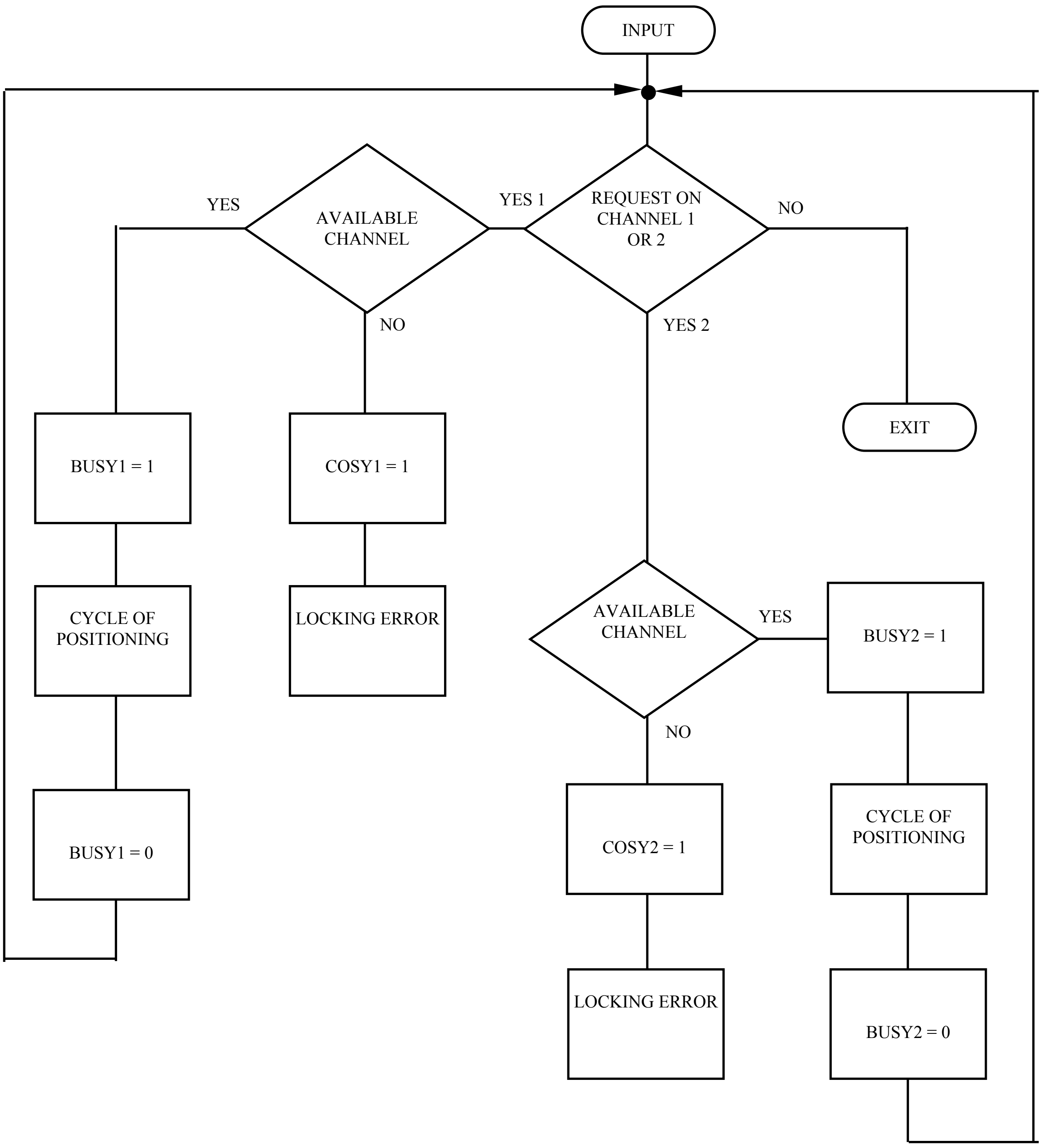


Fig.26. POINT-TO-POINT AXES POSITIONING CYCLE

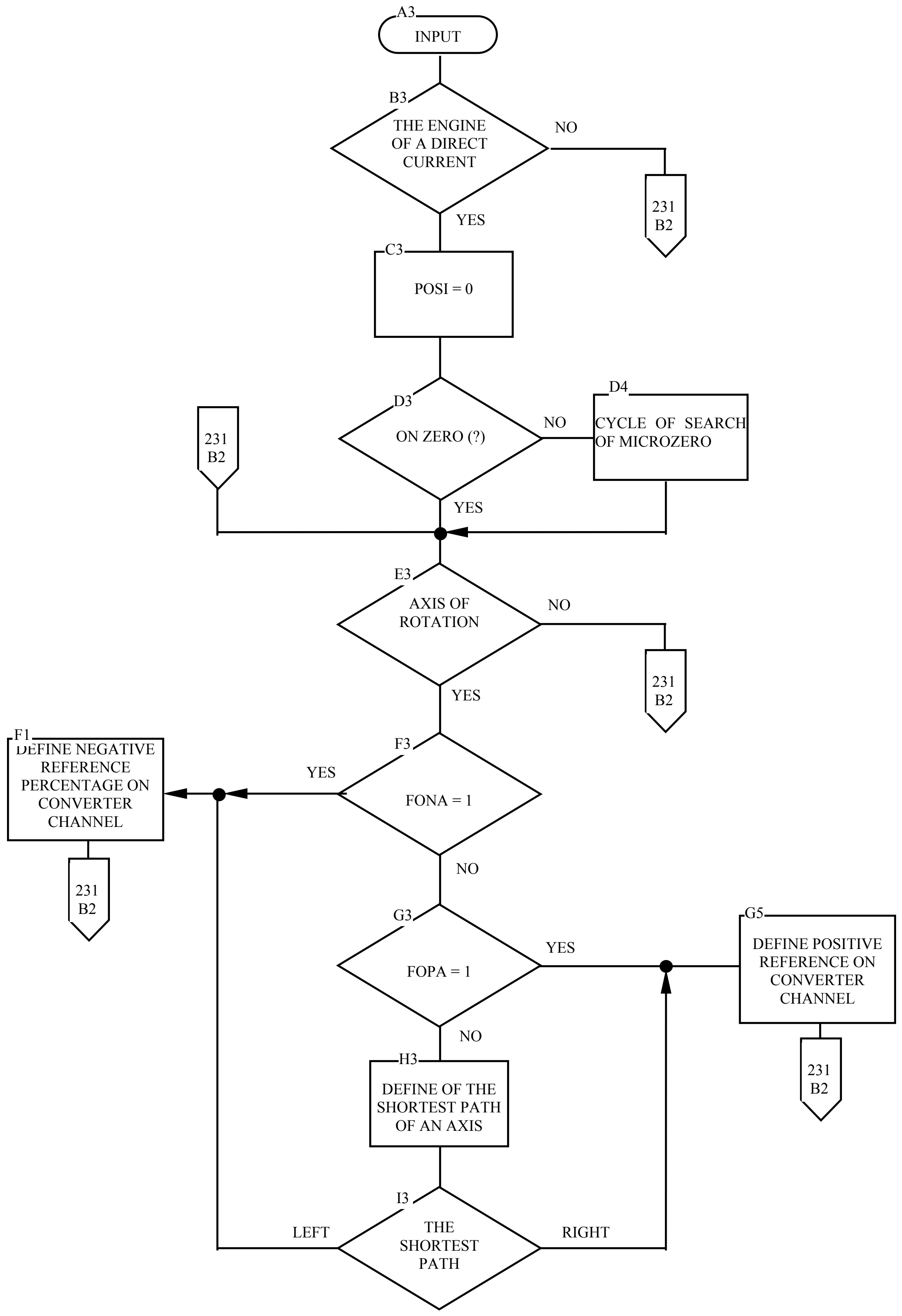
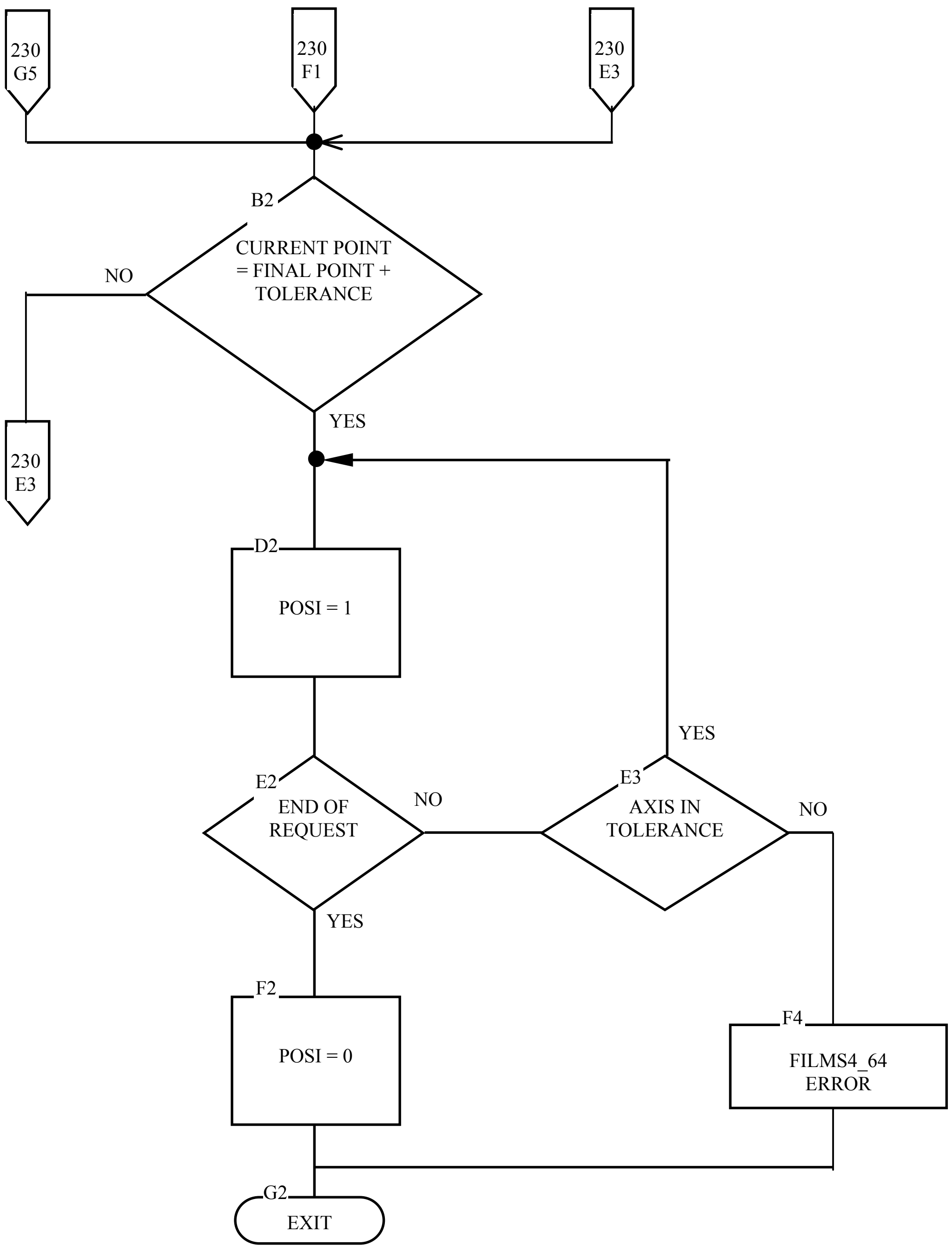

A3
INPUT
B3
THE ENGINE OF A DIRECT CURRENT
NO
231 B2
YES
C3
POSI = 0
D3
ON ZERO (?)
NO
D4
CYCLE OF SEARCH OF MICROZERO
231 B2
YES
E3
AXIS OF ROTATION
NO
231 B2
YES
F1
DEFINE NEGATIVE REFERENCE PERCENTAGE ON CONVERTER CHANNEL
YES
F3
FONA = 1
231 B2
NO
G3
FOPA = 1
YES
G5
DEFINE POSITIVE REFERENCE ON CONVERTER CHANNEL
231 B2
NO
H3
DEFINE OF THE SHORTEST PATH OF AN AXIS
I3
LEFT
THE SHORTEST PATH
RIGHT

Fig.27. AXIS POSITIONING CYCLE WITH A/D CONVERTER


230
G5
230
F1
230
E3
B2
CURRENT POINT = FINAL POINT + TOLERANCE
NO
YES
230
E3
D2
POSI = 1
E2
END OF REQUEST
NO
YES
E3
AXIS IN TOLERANCE
YES
NO
F2
POSI = 0
F4
FILMS4_64 ERROR
G2
EXIT

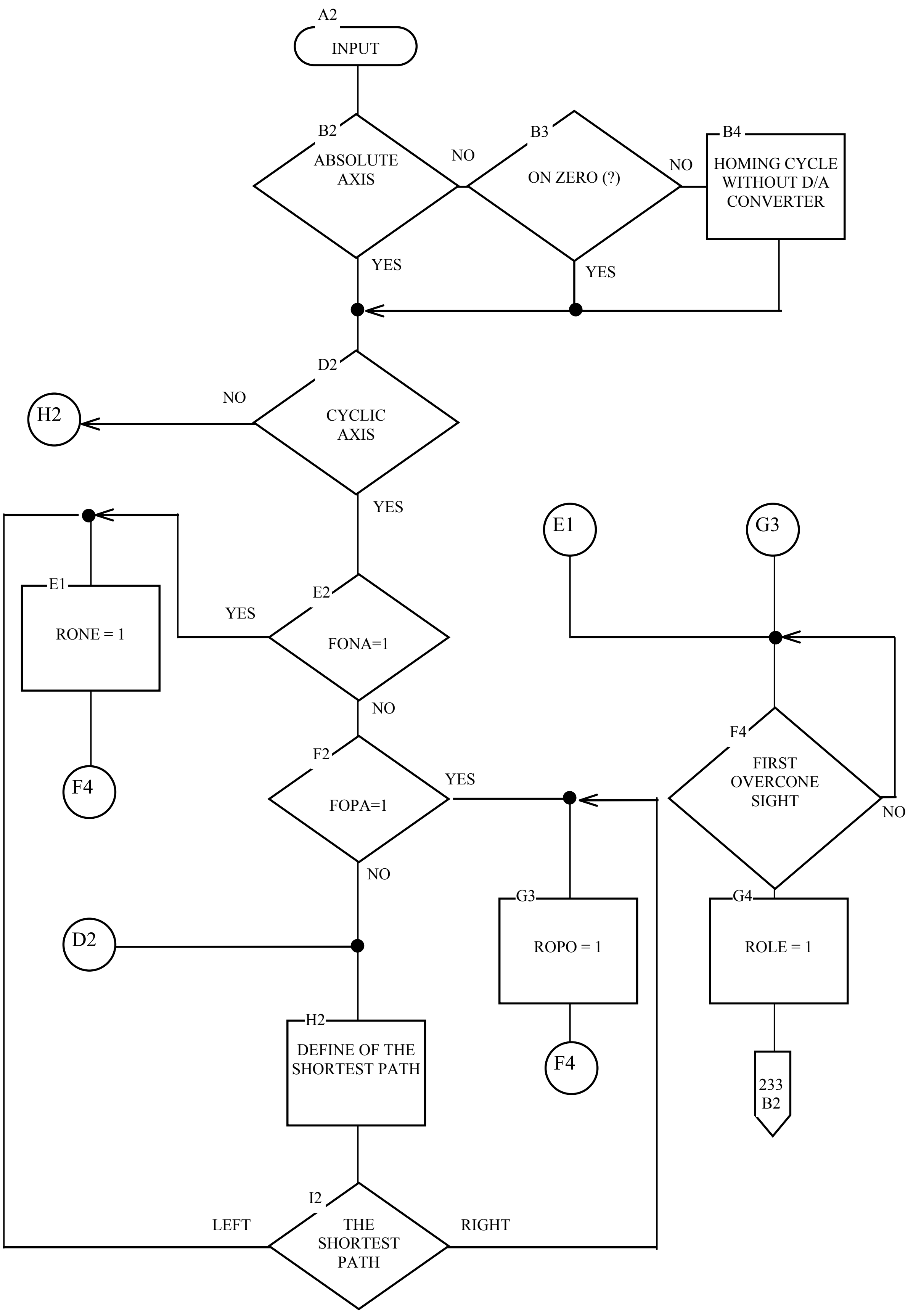
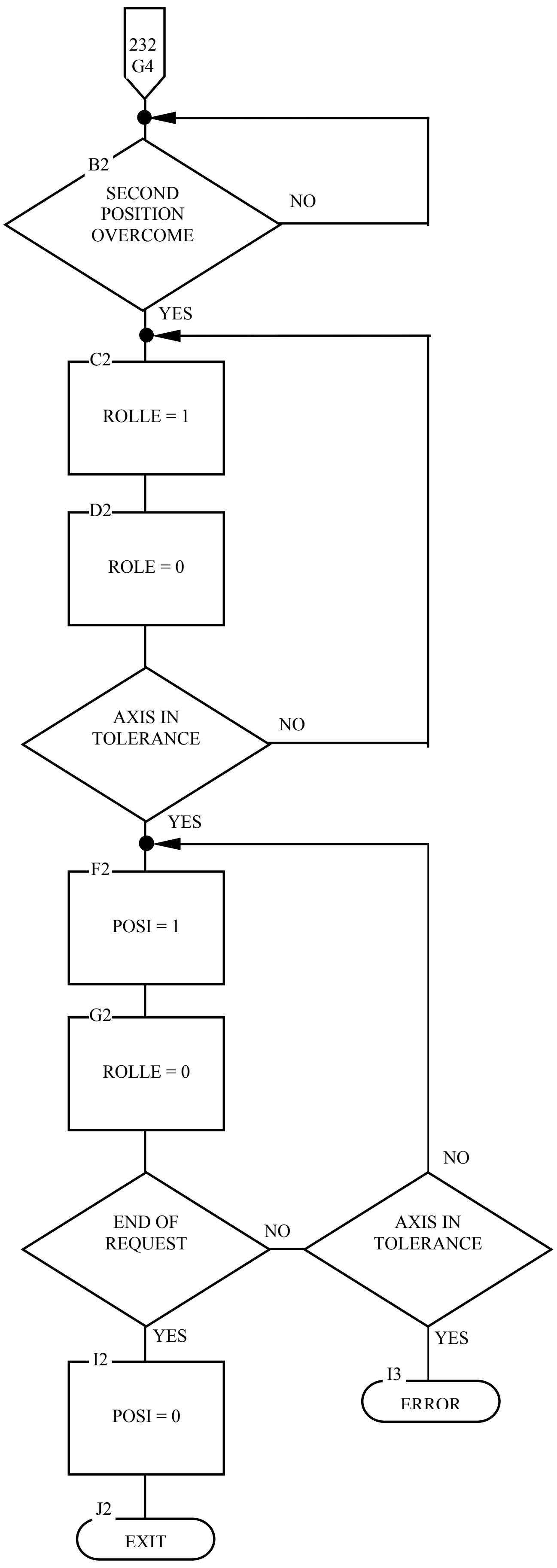


Fig.28. AXES POSITIONING CYCLE WITHOUT D/A CONVERTER


232
G4
B2
SECOND POSITION OVERCOME
NO
YES
C2
ROLLE = 1
D2
ROLE = 0
AXIS IN TOLERANCE
NO
YES
F2
POSI = 1
G2
ROLLE = 0
END OF REQUEST
NO
AXIS IN TOLERANCE
NO
YES
I3
ERROR
YES
I2
POSI = 0
J2
EXIT

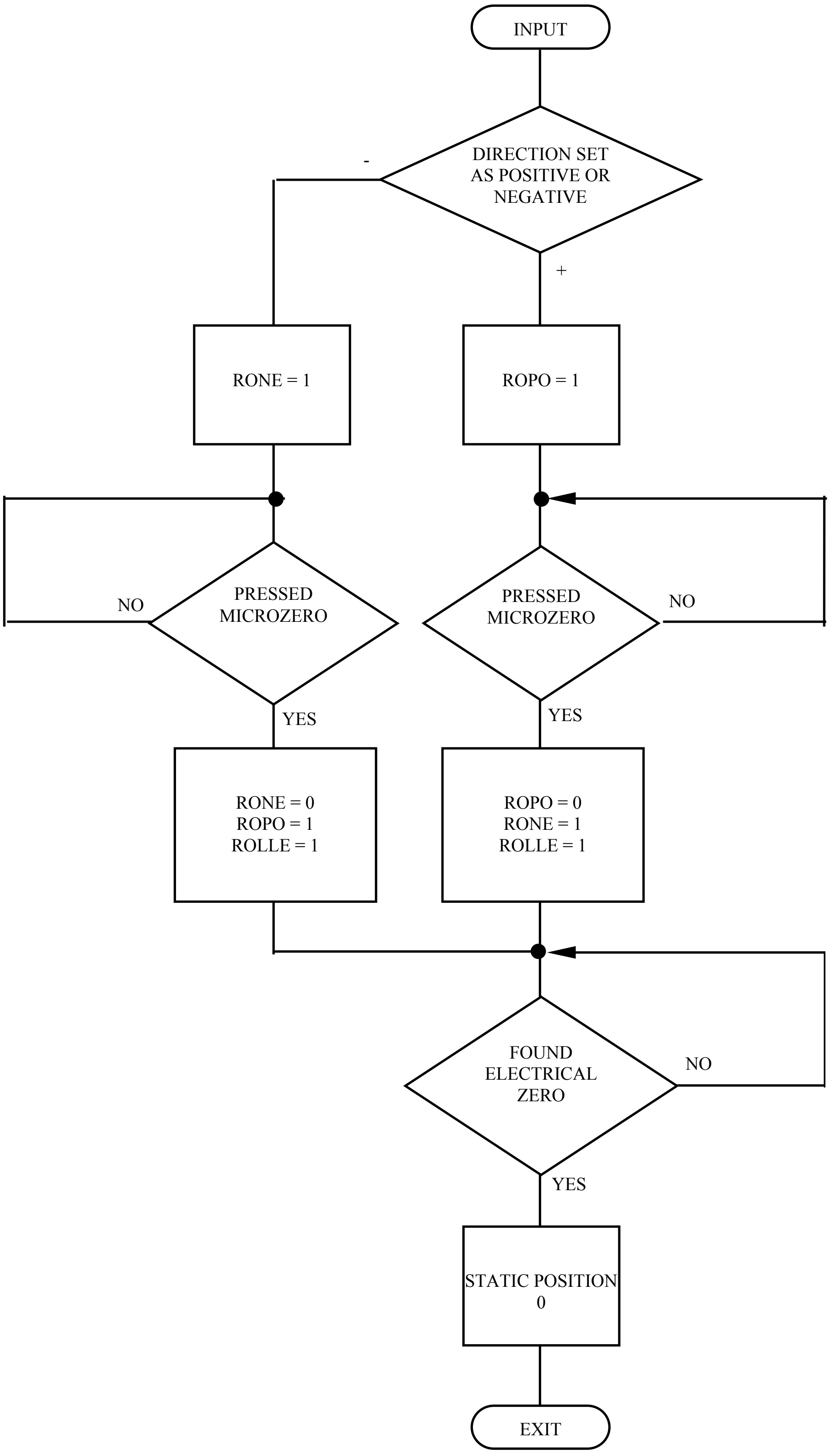


Fig.29. POINT-TO-POINT AXES ZERO MICRO CYCLE WITHOUT A/D CONVERTER