5

5. STRUCTURE OF PLC PROGRAM

A structure of PLC program is designed to increase its effectiveness in matching CNC system to machine

5.1 Auxiliary Statements

Instructions	**DATA** **DATA_END (START)** **STOP**

operation	**DATA**	**Beginning of data area**
	DATA_END (START)	**end of data area and beginning of program**
	STOP	**end of program**
syntax	**DATA**	
	DATA_END (START)	
	STOP	

Each program written in PLC836 language must begin with obligatory statement **DATA**. Program written in PLC836 language must end with statement **STOP**. Using instructions DATA, DATA_END and STOP is described in following chapter in detail.
A module DATA can be used in all files of PLC program.

5.2 Modules of PLC836 Language

The modules of PLC836 language were created to simplify design of PLC program. Simplification recruits from two different approaches to PLC program design. First is a simplification of matching and synchronizing of PLC automaton to CNC system. For instance a module of introductory functions (see next) that is triggered only after block start has features similar as Sequential Logical Unit (mechanism) and is executed one times. This means that all statements located in this module are executed automatically in introductory functions after block start . Second approach lays in structuralization of PLC program, as was described earlier. The module of introductory and closing functions deals as activation module of each mechanism destined for particular machine process.
During writing a programmable interface routines is necessary to keep certain rules and recommendations. Program has fixed structure that must be kept by programmer. Program must begin with keyword **DATA**, after that variables in use with their length are defined.

The beginning of a program code starts with keyword **DATA_END (START)**

After the keyword **DATA_END** follow next modules (see pict.) The module names are obligatory The order

of names is random but it is recommended to keep an order shown in the picture. It is not necessary to use all modules shown but in the case of unused module at least its name (e.g. ZAVERECNE_FUNKCE) and its closing (e.g. ZAVERECNE_FUNKCE_END) must be given.

Last module must end with keyword **STOP**, which ends an interface program.

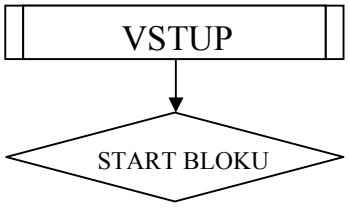
On the picture is shown a minimal program version which is translated by TECHNOL compiler with no error report . This program of course do not perform any PLC functions. A such a blank structure of PLC program is called ***zero PLC program***. All important variables in the interface between PLC and CNC system are declared in advance in such a way that CNC system even with a zero PLC program can run and execute all functions with are independent to machine technology. The zero PLC program hence can deal as a template for a new PLC program design..

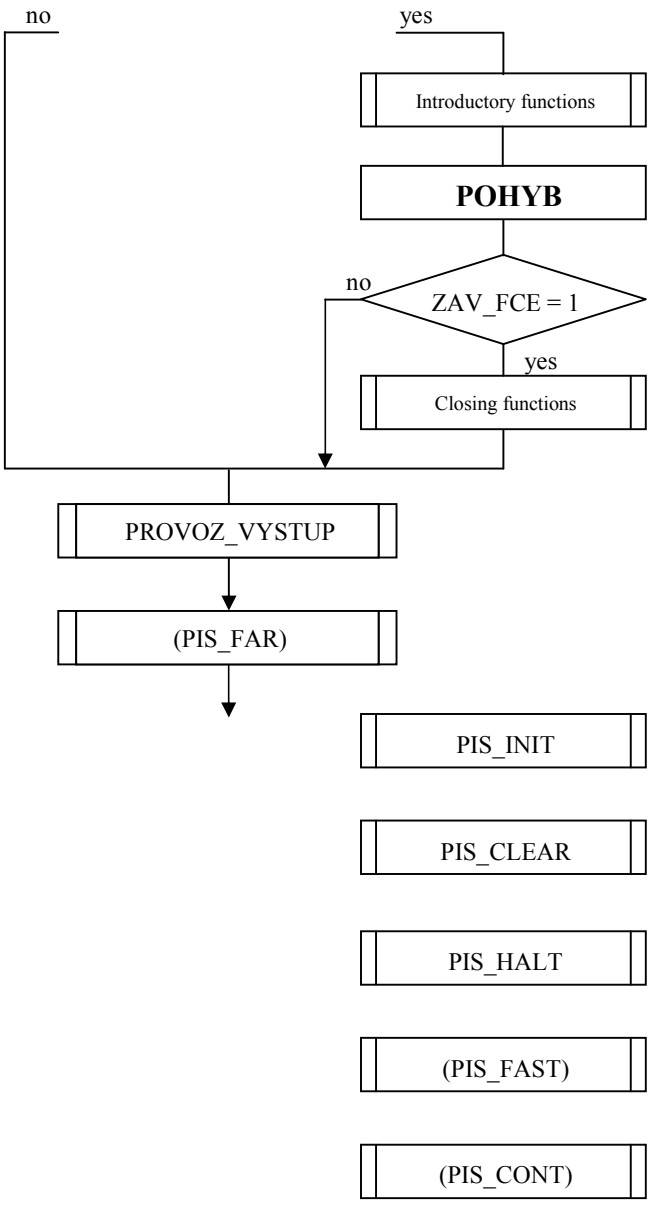
A PLC program goes to module VSTUP. After module MODULE_INPUT (VSTUP) program is branched depending on a system action. If a new block of partprogram is not started , program does not execute modules MODULE_BLOCK_INIT (Introductory functions) and MODULE_BLOCK_DONE (closing functions), but executes module MODULE_MAIN (PROVOZ_VYSTUP). If a new block is started program executes above mentioned modules in which an axis move occurs (if programmed). Modules MODULE_INIT (PIS_INIT), MODULE_CLEAR (PIS_CLEAR) and MODULE_HALT (PIS_HALT) are executed only when called. In the next chapter is a more detailed description of modules.

All modules are programmed according to function which represented and which is also noticeable from its name. As mentioned before the modules can be blank. Next are introduced the most frequent functions which are programmed in the modules .

```
DATA
    ;data definition
DATA_END  (START)

VSTUP
    ;module of inputs
VSTUP  END
```





The structure of PLC program

5.3 A Description of Modules

module	**DATA**

A module of global data starts with keyword **DATA** and ends with keyword **DATA_END (START)**.
All files of PLC program must start with keyword DATA, after it declaration of variables used in PLC must occur. Data which are declared in this module have a **global** character, this means that are known and reachable to all files of PLC program. The module DATA can be used in every file of PLC program only one times and on the very beginning of file only.

module	**DATA_LOCAL**

A module of local data starts with keyword **DATA_LOCAL** and ends with keyword **DATA_LOCAL_END** (Starting with version 6.028).

This is an optional module of PLC program which is destined for local variables declaration. Data declared in this module have a **local** character, this means that are known and reachable to the current module file. Module DATA_LOCAL can be multiple used in every file of PLC program and can be embedded into other modules except module DATA. The local data are used also to define "automatical" variables during advancement of some PLC836 language instructions .

Data defined in this module are not visible in this version even for debugging program WINTECHNOL. If it is necessary for debugging purposes to see local variables a temporary shift of module DATA_LOCAL to the body of module DATA must be done. If module DATA_LOCAL is placed within module DATA, which has a global character, the local data are visible also for WINTECHNOL.

module	**VSTUP (MODULE_INPUT)**

A module starts with keyword **VSTUP (MODULE_INPUT)** and ends with keyword **VSTUP_END (MODULE_INPUT_END)**.
The module is activated as first in every PLC cycle and has no restrictions.

In this module usually reading of input ports to declared PLC memory occurs. Those inputs which directly affects a block of feedback report (e.g. limit and reference switches) are rewritten in requested form to the block of feedback reports.

module	**INTRODUCTORY FUNCTIONS**

A module starts with keyword **PRIPRAVNE_FUNKCE (MODULE_BLOCK_INIT)** and ends with keyword **PRIPRAVNE_FUNKCE_END (MODULE_BLOCK_INIT_END)**.

The Module INTRODUCTORY FUNCTIONS is activated after block start only. The block start occurs when program runs in modes AUT end RUP, but also in manual modes MAN and JOG and in a central clearing. The module has features identical to Sequential Logical Unit (mechanism) and is executed one times. The module can deal as activating module of the mechanisms which are destined to executed particular machine processes .

In the module a typical introductory or initial functions of a partprogram blocks are executed., For instance spindle start switching on cooling releasing axis etc. The module is active only during block's start.

The module of introductory functions is logical sequential system and thus **all instructions type EX** can be used. (see chapter "Logical sequential systems"). A run control of module introductory function is described in chapter "A Run Control of interface supervisor".

Notice.:
Because during decoding of introductory functions also appearance of closing functions is recognized it is necessary to set index ZAV_FCE, that activates module ZAVERECNE_FUNKCE.

module CLOSING FUNCTIONS (MODULE_BLOCK_DONE)

A module starts with keyword **ZAVERECNE_FUNKCE (MODULE_BLOCK_DONE)** and ends with keyword **ZAVERECNE_FUNKCE_END (MODULE_BLOCK_DONE_END)**.

In this module actions which are typical. for closing functions of partprogram blocks are executed . For instance spindle stop switching off cooling etc. A condition for this module execution is setting of index ZAV_FCE (see notice in module INTRODUCTORY FUNCTIONS).

The module of introductory functions is logical sequential system and thus **all instructions type EX** can be used. (see chapter "Logical sequential systems").

module PROVOZ_VYSTUP (MODULE_MAIN)

A module starts with keyword **PROVOZ_VYSTUP (MODULE_MAIN)** and ends with keyword **PROVOZ_VYSTUP_END (MODULE_MAIN_END)**.
The module is activated in every PLC cycle and has no restrictions.

In this module the functions which must be permanently scanned are executed. for instance machine's feedback reports. It is advantageous to place the mechanisms here. **Instructions type EX** , valid for sequential systems (see chapter "Logical sequential systems") is possible to use in mechanisms only.
Here also setting of outputs is provided. In this module can be embedded PLC programs, which deal with state changing input signals .
This module can be used in all files of PLC program.

module PIS_INIT (MODULE_INIT)

A module starts with keyword **PIS_INIT (MODULE_INIT)** and ends with keyword **PIS_INIT_END (MODULE_INIT_END)**.

This module is destined for initialization of interface variables and other actions which are necessary during machine switch on.. The module is not called in every cycle but only during machine start up. In the nodule PIS_INIT is recommended to send all inputs in a defined form directly to output ports.

In the module PIS_INIT must be an initialization of all mechanisms via instruction MECH_INIT (see chapter "Logical sequential systems")**.**

The module PIS_INIT is not executed only if a first decade of machine constants 89 to value 0 which means "Stop PLC program after switching ON the system". This mode is used for PLC program debugging (see Chapter "Debugging of PLC program").
This module can be used in all files of PLC program.

module PIS_CLEAR (MODULE_CLEAR)

A module starts with keyword **PIS_CLEAR (MODULE_CLEAR)** and ends with keyword **PIS_CLEAR_END (MODULE_CLEAR_END)**.
This module is destined for clearing of variables. and setting the interface to its initial conditions . The module is activated by pushing an interface clear button .

In the module PIS_CLEAR must be an initialization of all mechanisms via instruction MECH_INIT (see chapter "Sequential logical units")**.**

PLC program often uses reconfigurable variables for instance for setting various times. These times can be changed by operator in machine constants which are saved in its BCD form. PLC program thus must often content a BCD to binary conversion. To avoid making this conversion in each PLC program cycle it is convenient to locate these conversions to the module PIS_CLEAR.
This module can be used in all files of PLC program.

module PIS_HALT (MODULE_HALT)

A module starts with keyword **PIS_HALT (MODULE_HALT)** and ends with keyword **PIS_HALT_END (MODULE_HALT_END)**.

In this module are programmed such a functions which must be executed when a serious error occurs before is system halted. It is recommended to place here setting of outputs which cause supports stop or an output which is destined for treatment of global error (switching OFF the machine power). It is necessary to send outputs to the outputs ports.

module PIS_FAST (MODULE_FAST)

A module starts with keyword **PIS_FAST (MODULE_FAST)** and ends with keyword **PIS_FAST_END (MODULE_FAST_END)**. This module is optional.
This module should contain functions which ought to be performed in a time scale faster than 20 ms. This module **PIS_FAST** is activated in identical time intervals as a software for position feedback . The time scale for position feedback is set in sixth decade of machine constant 97 (see Machine constants of system CNC836). Using of **PIS_FAST** module attempts that the system CNC836 contents a fast input card **IN03** with order number of input card 3 (address switches: 2=on, 3=off, 4=on). This is a card of multiplexed inputs which in this case doesn't work in multiplexed mode. For reading fast inputs (maximally 8) a special instruction **IN_FAST** is used. (see chapter "Controlling binary inputs and outputs in system crate "). To use module **PIS_FAST** properly is necessary to send requested fast outputs via standard output instruction **OUTP**. The Logical sequential systems can be programmed in this module.

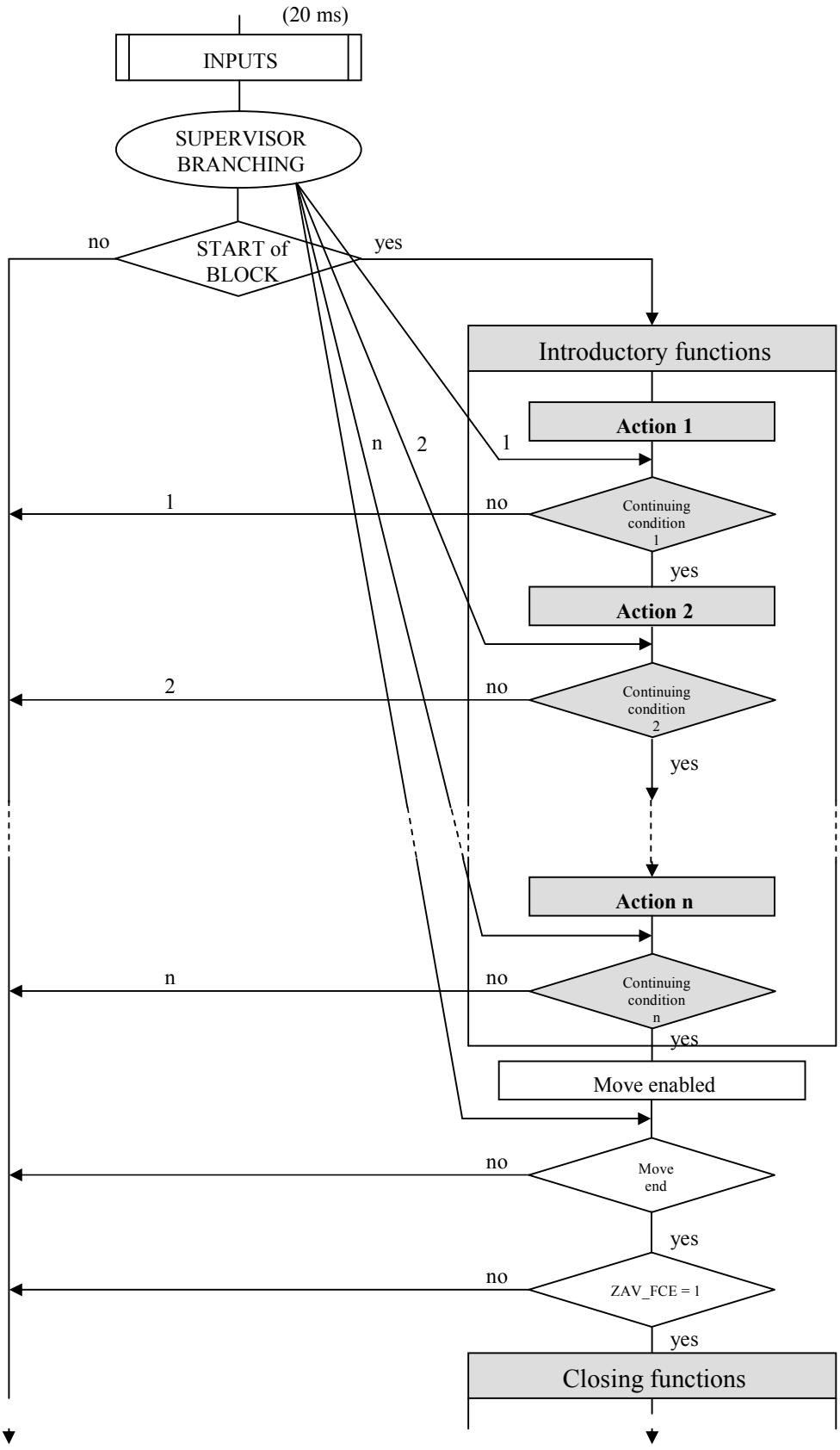
module PIS_CONT (MODULE_CONT)

A module starts with keyword **PIS_CONT (MODULE_CONT)** and ends with keyword **PIS_CONT_END (MODULE_CONT_END)**. This module is optional.
In the module are programmed functions which are non-interrupted by debugging tools. This feature could be very advantageous during program debugging. A logic which is programmed in the module PIS_CONT apparently runs simultaneously with main modules of the PLC program.

module PIS_FAR (MODULE_FAR)

A module starts with keyword **PIS_FAR (MODULE_FAR)** and ends with keyword **PIS_FAR_END (MODULE_FAR_END)**. This module is optional.
Remote module of PLC program, which is logically connected to module PROVOZ_VYSTUP. It is used for longer PLC programs in which a code segment is greater than 64 kByte. The module has all features identical with module PROVOZ_VYSTUP, this means possibility of using all mechanisms and also using DEBUG instruction for break-points setting is allowed..
For systems type CNC8x9 DUAL is recommended to divide program into a few smaller parts instead of using module PIS_FAR.

Z


(20 ms)
INPUTS
SUPERVISOR BRANCHING
START of BLOCK
no
yes
Introductory functions
Action 1
1
n
2
1
no
Continuing condition 1
yes
Action 2
2
no
Continuing condition 2
yes
Action n
n
no
Continuing condition n
yes
Move enabled
no
Move end
yes
no
ZAV_FCE = 1
yes
Closing functions

5.4 A Run Control Of Interface Supervisor

A supervisor of a programmable interface controls a PLC program run in following way. After starting a block the control is handed to module **INTRODUCTORY FUNCTIONS** and the program is executed until a first instruction for the state definition occurs e.g. to fulfilling of certain condition. This is instruction **type EX** (see Chapter "Sequential Logical units"). In every next cycle only the continuing condition is tested . This means a program area between a two last instructions type EX.

When an continuing condition comes true program steps through module of introductory function to the next condition. The condition can be also executing of activated mechanisms. as is described in chapter "Logical sequential systems" -example of mechanism's starting :

```
FL      1,CW      ;activate variable setting
EX
LDR     CW        ;testing of mechanism execution
EX1
```

After executing of whole module of introductory functions the supervisor enables possible movement for an interpolator and waits in this state until condition for reaching programmed position is true.
After confirmation of requested position the supervisor hands the control to module **Closing functions** , but if and only if a bit variable **ZAV_FCE** is set**.** The module of closing functions is also the Sequential logical unit and the supervisor steps through this module similarly as in case of module of Introductory functions.
The speed of a block's execution depends on design of modules introductory and closing functions. The instructions EX for instance cause a one cycle delay in the PLC program. But these instructions are worth to use to establish a wait for condition function. This is programmed via instructions EX0, EX1, TEX0 or TEX1. In this case by using instruction EX is decreased a part of program which will be stepped through.. (see chapter "Sequential Logical units").

A module **PROVOZ** is activated in each interface cycle (20 ms) and thus this is not the Sequential Logical Unit. In this module are placed sequential systems (mechanisms) via statements **MECH_BEGIN** and **MECH_END**.

5.5 Writing PLC Programs in Multiple Files

A possibility of writing a PLC program into multiple files is allowed for systems type CNC8x9 –DUAL.

The PLC program consists of a main file that contents all obligatory program modules (VSTUP, PRIPRAVNE_FUNKCE, ZAVERECNE_FUNKCE, PROVOZ_VYSTUP, PIS_HALT, PIS_CLEAR and PIS_INIT). Except the main file the PLC program can be written other independent files (in the version 6.001 maximally 7). Other files can have declared data and are continuation of module PROVOZ_VYSTUP (MODULE_MAIN) from the main file.

The files of PLC program can be used for linking debugged library functions of PLC program .
The files of PLC program must fulfill following rules:

a) In configuration file TECH.KNF is a main file declared by keyword **FilePlc** and next files by keywords **FilePlcExt**. These keywords are optional.

```
Example   FilePlc      =   I_O_MAIN    ;main file
          FilePlcExt   =   I_O_MOD2    ;second file  -  naximally 7 files
          FilePlcExt   =   NO          ;not used
```

b) Every other file of PLC program must contents modules:

```
DATA                                ;Global data
        declaration of global data
DATA_END                            ;(START)

DATA_LOCAL                          ;Local data
        ;… declaration of local data
DATA_LOCAL_END


MODULE_MAIN                         ;(PROVOZ_VYSTUP)
        ;…basic logic , mechanisms
MODULE_MAIN_END                     ;(PROVOZ_VYSTUP_END)
```

These modules are optional:

```
MODULE_INIT                         ;(PIS_INIT)
        ;… data initialization
MODULE_INIT_END                     ;(PIS_INIT_END)


MODULE_CLEAR                        ;(PIS_CLEAR)
        ;… data clearing
MODULE_CLEAR_END                    ;(PIS_CLEAR_END)
```

All modules are continuation of the same modules from main file of PLC program.

c) Module **DATA** must be declared first . All date which are defined in any PLC file including main file have **global** character, this means that are reachable to all other files.

d) Module **DATA_LOCAL** is an optional module of PLC program which is destined for declaration of l**ocal** variables Data declared in this module are reachable only to current module file. Module DATA_LOCAL can be multiple used in all files with PLC program and can be embedded to other modules except module DATA. Local data are used for defining of "automatical" variables in range of development some instructions of PLC836 language. The data defined in this module are in this version invisible even for debugger WINTECHNOL. If is necessary from debugging purposes to see local variables shift temporarily module DATA_LOCAL into a body of module DATA. When a module DATA_LOCAL is located inside of module DATA, which has a global character will be the local data visible also for WINTECHNOL.

e) Module **MODULE_MAIN** (PROVOZ_VYSTUP) is continuation of the same module from previous files of PLC program. The file can content mechanisms and can call mechanisms which are defined in other files of PLC program. Also instruction MECH_INIT can be used in all files even if a mechanism is not there defined. Using instruction DEBUG in nodules is allowed.

f) Modules **MODULE_INIT** and **MODULE_CLEAR** (PIS_INIT a PIS_CLEAR) are in other files optional. If used they are continuation of the same modules in previous files.

g) All files can content any procedure definitions **PROC_BEGIN – PROC_END** and also any procedure calling even if they are defined in other files **PROC_CALL**.

h) In all files is allowed multiple using of instructions for defining time intervals **DFTM01**, **DFTM1**, **DFTM10**, **DFM100**.