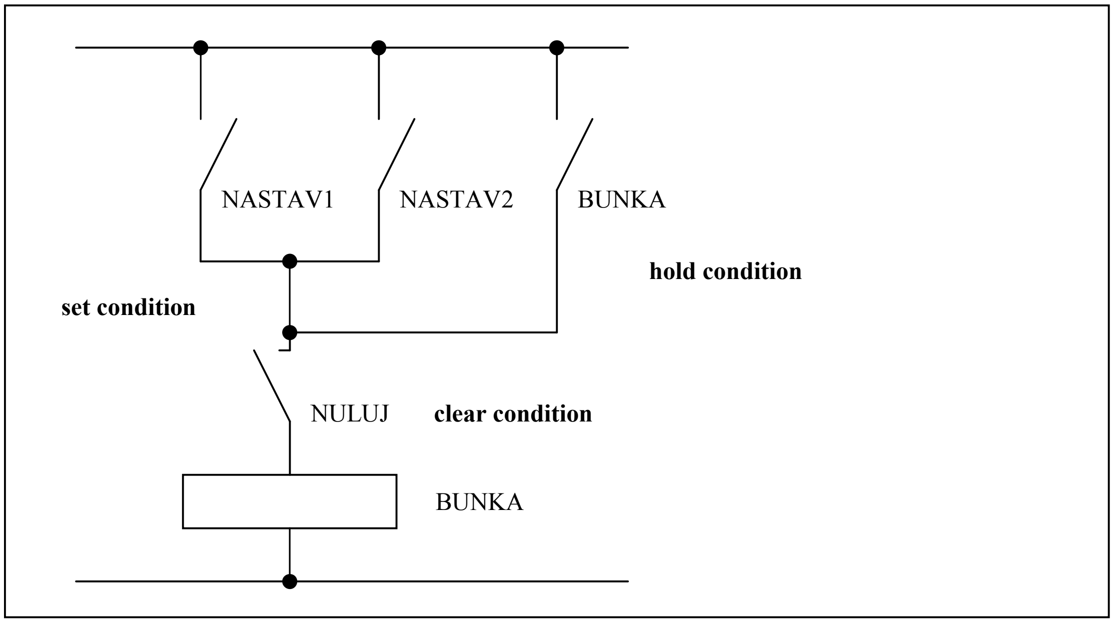
4

4. Sequential Logical Units

4.1 Architecture of PLC program

A PLC program can be created in many ways. A standard approach in PLC program writing is based on designing of logic (sequentially combinatorial) or translating relay logic into instructions of PLC836. Language For every relay its own inner bit cell is declared. Writing into this cell is controlled by instructions Set**, hold and clear** condition**.** Writing into the cell is in program at one place**.**

Example:
A standard approach to PLC program design.



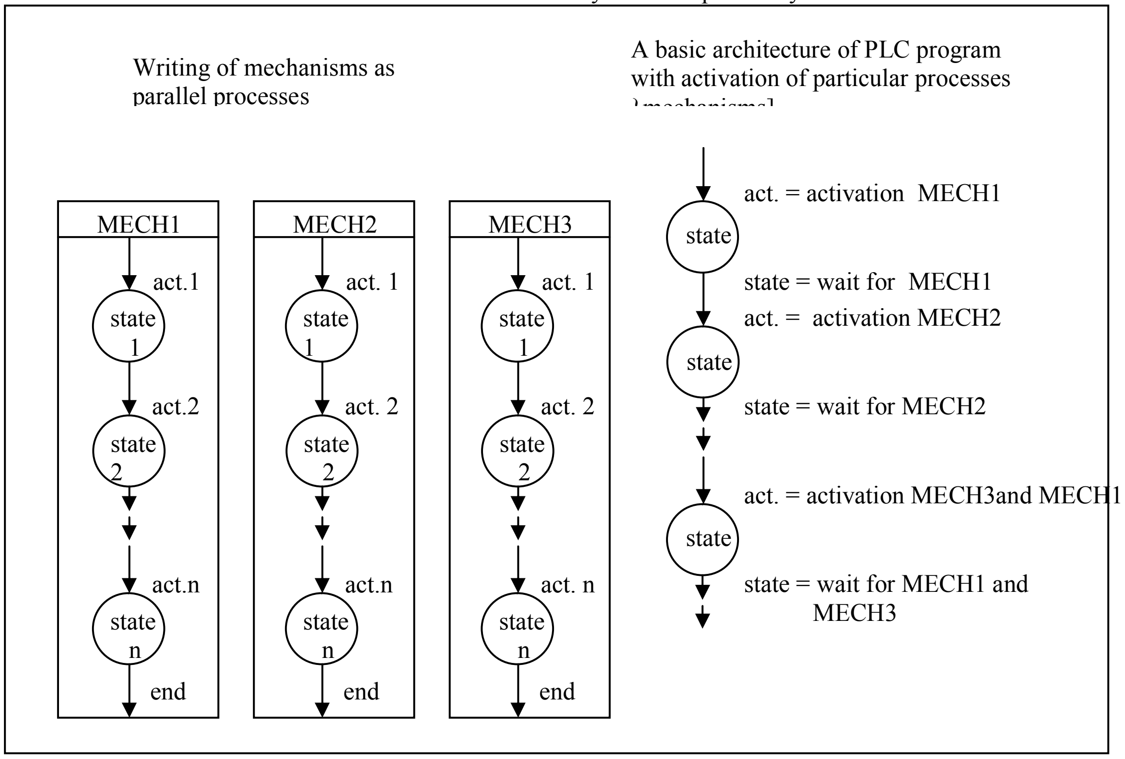
```
LDR     NASTAV1     ;set condition is logical OR of
LO      NASTAV2     ;bits NASTAV1 NASTAV2
LO      BUNKA       ;hold condition
LA      -NULUJ      ;bit NULUJ clears  flip flop
WR      BUNKA       ;bit memory variable
```

A modern approach to PLC program design is based on description of each process via flow-charts which create

Sequential Logical Units (mechanism).
The Sequential Logical Unit (next only MECHANISM) is a program part that consists of actions and states. Action means setting of appropriate bits which activate each part of mechanism for example switching on a hydraulic ,switching off ventilator's contactors e.g. A state means checking of requested input condition of the mechanism. For example „ wait until contactor of hydraulic is switched on". The mechanism can be described by flow chart or state diagrams.

Mechanism is activated by an activating variable (NAZEV). It is a bit variable that is declared automatically during mechanism's writing. This variable can be set , reset or tested in an interface program and thus a run control of mechanism is provided. The mechanism is activated by setting of the activating variable and after execution of mechanism function is this variable automatically cleared.

PLC program can content a great number of mechanisms each for particular machine function. (Spindle start, spindle reverse , spindle stop, clamp and releasing of axis, oriented stop, arm moving, start and stop of cooling, steps for tool changing, tool search, arrangement of revolution sequences e.g.). Any number of mechanism can be activated in the same time. The mechanisms run simultaneously and independently.



Using mechanisms simplifies PLC program design makes it more transparent. The PLC program is then more **structured** . A structuralization is given via describing of all machine processes by mechanisms and thus a program kernel is not strained by their particular problematic. The program kernel only activates appropriate process (mechanism) and tests its proper operation including error reporting. The mechanisms consist of actions and states. For each state only necessary condition for going to next state are tested.
In every time is possible to trace run of each mechanism: That means which mechanism is activated and its state. It simplifies program debugging and troubleshooting because every mechanism's state is given by so called **continuing condition** (condition analysis) . It is useful to trace this continuing conditions when mechanism stops. A big advantage of mechanisms is a possibility of time restriction of condition testing (time out) and thus to avoid mechanism freezing. The mechanism ends abnormally (with an error). In error description on the display can be in detailed text form expressed a continuing condition including number of inputs which given condition contents. This simplifies maintenance and troubleshooting works.

Except condition analysis is possible to perform analysis of mechanism timing. For instance how long and in which states mechanism stayed (time analysis).

4.2 Sequential Logical Unit Instructions

instruction	**MECH_BEGIN**	
	MECH_END	
	MECH_INIT	
operation	**MECH_BEGIN**	**beginning of logical system**
	MECH_END	**end of logical system**
	MECH_INIT	**initialization of logical system**
syntax	**MECH_BEGIN**	**mech**
	MECH_END	**mech**
	MECH_INIT	**mech**

MECH_BEGIN This instruction must be at the mechanism start. A variable "mech" is a name of mechanism and concurrently a name of its activating bit variable. **MECH_END** instruction must be at mechanism's end. Variable "mech" has the same meaning e.g. is identical to name in MECH_BEGIN.

MECH_INIT This instruction pushes mechanism to its steady state. It is used during interface initialization or during error treatment. This instruction must be used before first mechanism run and hence is recommended to put this instruction to initialization module of an interface (see next). Variable "mech" is identical as in MECH_BEGIN. See "Common Rules" (next).

instruction	**EX**	
	EX0	
	EX1	
	BEX	
operation	**EX**	**abortion for one interface cycle**
	EX0	**abortion if RLO = 0**
	EX1	**abortion if RLO = 1**

The instructions for state definition **EX, EX0, EX1** abort run of Sequential Logical Unit until dedicated condition is true. These instructions can be used inside of logical systems only. These abortions are called mechanism states.

The instruction EX causes unconditional abortion of sequence execution for one interface cycle.

The instruction BEX is similar to instruction EX, but do not cause sequence abortion. This instruction is used at the beginning of logical condition before instructions EX0, EX1, TEX0 a TEX1. It is recommended to use this instruction in time consuming processes when a lose of one interface cycle could be fatal.

The Instruction EX0, resp. EX1 abort run of Sequential Logical Unit when RLO=0, resp. RLO=1. Before instructions EX0 and EX1 can be logical condition for continuing in current state. For given state in every cycle conditions written between one before last and last instruction EX are evalua ted.

Actually instructions EX0 a EX1 do not wait on given place but jump to mechanism end.. During next mechanism execution jump from beginning of mechanism to one before last instruction EX and again evaluation of conditions for instruction EX0 or EX1 takes place. See Chapter " Common Rules for Mechanism" (next).

The instructions EX, EX0, EX1, TEX0, TEX1 a BEX are the ***end instructions*** for logical expressions. The Instructions do not preserve registers RLO and DR .

instructions	**TEX0**
	TEX1

operation	**TEX0**	**abortion execution when RLO = 0 with Time Out**
	TEX1	**abortion execution when RLO = 1 with Time Out**
syntax	**TEX0(TEX1)**	**counter, time, error**
	TEX0(TEX1)	**counter, time, error [, chyba]**
	TEX0(TEX1)	**[TYPE.]counter, time, error [, chyba]**
	TEX0(TEX1)	**TYPE. (counter+n), (time+m), error [, chyba]**
	TEX0(TEX1)	**- , time, error [, chyba]**
		TYPE = BYTE. WORD.

The instruction **TEX0**, resp. **TEX1** aborts sequence execution of mechanism when RLO=0, resp. RLO=1, but maximally for preset time "time" (BYTE, WORD, constant). If a logical condition for continuing , that is written before instructions TEX0 a TEX1, is not true for preset time the program continues from label which is given in parameter "error".

Parameter "counter" can be of type BYTE or WORD. For systems type CNC8x9 – DUAL starting with version 6.028 is a first parameter for "counter" optional. In this case a variable in a local data is automatically declared. Instead of 1st parameter is used character ”dash” or “NIL”.

Instructions mentioned above belong to “strong” instructions used in mechanisms, because they give us a possibility of error treatment and thus to exclude mechanism freezing . In a comment of possible error can be detailed description of **continuing condition** for a given state and thus perform machine diagnostic .

For every state in each cycle conditions written in program part between two last reached instruction type EX are evaluated. This instruction has three obligatory parameters. Designation of time element "counter", requested time out "time" and label "error", from which program continues , when condition is not true within time out. "time" can be either direct time value or an address of variable.. see "Common Rules" (next).

Forth optional parameter "error" can be a number (mostly an error number), which remains in data DR register when jump to label "error" occurs. Thus is possible from various instructions TEX0 and TEX1 jumped to one location "error" with common error stage treatment.

Instructions TEX0 a TEX1 are ***end instructions*** for logical expressions. Instructions do not preserve RLO and DR register, except jump to label "error", when the fourth parameter "error" is used.

The possibility of redefining operands "counter" and "time" is described in chapter "....... Redefining of variable type relates to both operands "counter" and "time", and thus is not possible to declare operand “time” as a constant.

instruction	**TIM**	
operation	**TIM**	**time delay**
syntax	**TIM**	**citac, doba**
	TIM	**[TYPE.]citac, doba**
	TIM	**TYPE.(citac+n), (doba+m)**
	TIM	**- , doba**
		TYPE = BYTE. WORD.

The instruction TIM aborts execution of Sequential Logical Unit for a dedicated time. This instruction has a two parameters. Designating of time element "citac" and requested time delay "doba". "Doba" can be either constant or a variable’s address. Both parameters can be of type BYTE or WORD. Instruction TIM works similarly instruction EX and can be used inside mechanism only.

For systems type CNC8x9 – DUAL starting with version 6.028 is the first parameter optional . In this case a variable in a local data is automatically declared . Instead of 1st parameter is used character ”dash” or “NIL”.

The possibility of redefining operands "citac" and "doba" is described in chapter "....... Redefining of variable type relates to both operands "citac" and "doba", and thus is not possible to declare operand “doba” as a constant.

4.3 Common Rules for Mechanism

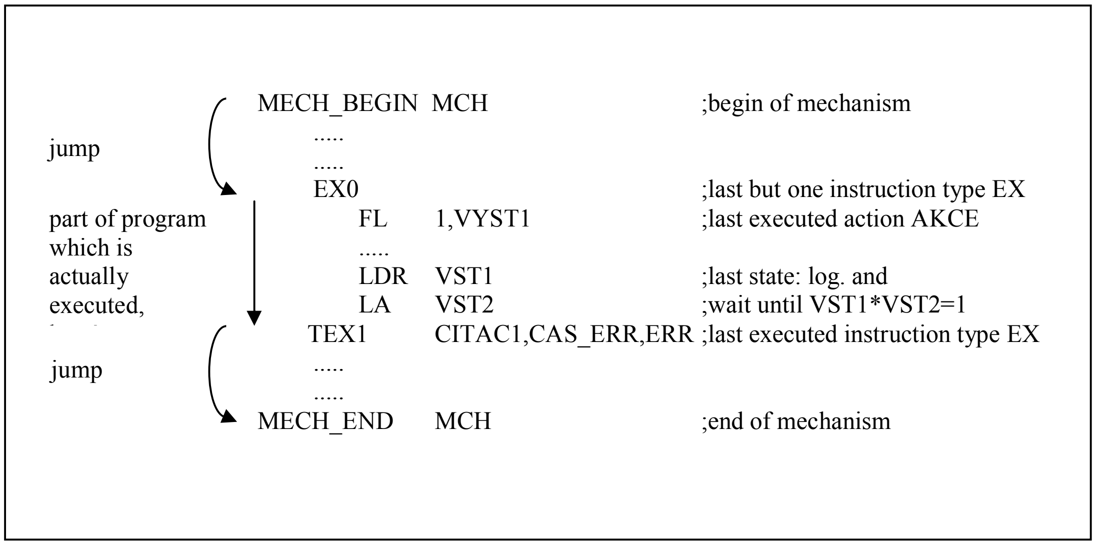
Instruction EX, EX0, EX1, TEX0, TEX1 and TIM can be used only inside of sequential logic systems circumscribed by instructions MECH_BEGIN a MECH_END. These instruction share common name: ***instructions type EX***.

Every mechanism must be initialized by instruction MECH_INIT in module PIS_INIT or in PIS_CLEAR (see chapter "Architecture of PLC program").

A module of introductory and closing functions is also a sequential logical module thus is possible to use in it all instructions type EX separately as well.. Using these instructions in those modules cause an abortion of introductory or closing functions until a tested condition comes true (see example).

Instructions EX0, EX1, TEX0, TEX1 and TIM abort sequence execution of the mechanism for time when appropriate condition is true (if RLO=0, or RLO=1) or for preset time (instruction TIM). Before instructions EX0, EX1, TEX0 and TEX1 can be written a logical condition for continuing in given state. For given state in ech cycle evaluation of conditions written between two last reached instructions type EX is provided.
Actually instructions type EX do not wait on given place but jump to mechanism end.. During next mechanism execution jump from beginning of mechanism to one before last instruction EX and again evaluation of conditions after last instruction type EX takes place.

Tracing of Mechanism’s Program in a given State



```
              MECH_BEGIN  MCH                  ;begin of mechanism
                   .....
                   .....
                   EX0                         ;last but one instruction type EX
                       FL    1,VYST1           ;last executed action AKCE
                       .....
                       LDR   VST1              ;last state: log. and
                       LA    VST2              ;wait until VST1*VST2=1
                  TEX1       CITAC1,CAS_ERR,ERR ;last executed instruction type EX
                   .....
                   .....
              MECH_END    MCH                  ;end of mechanism
```

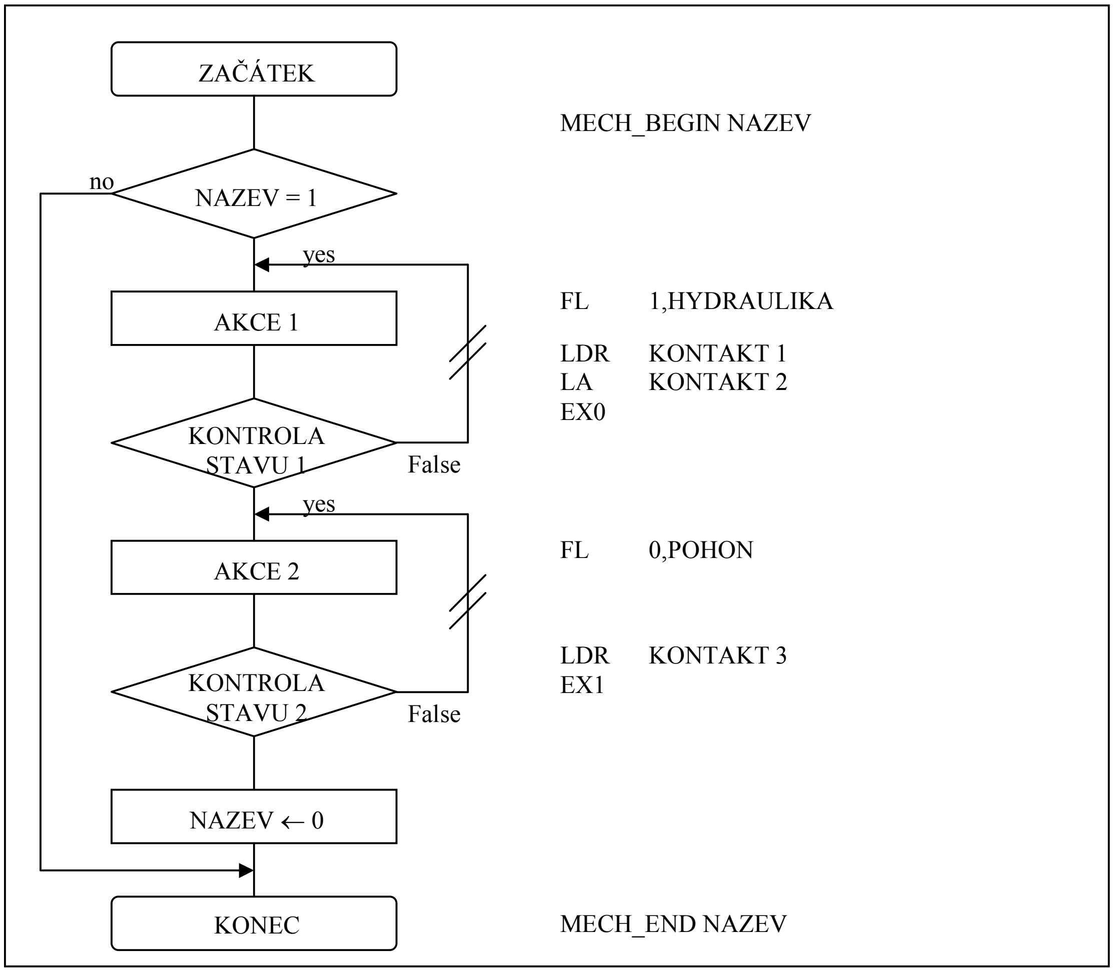
The mechanisms can be activated and deactivated also in other mechanisms. The ***mechanism deactivation*** is provided by instruction MECH_INIT, which clears a bit control variable of mechanism and set an address for continuing to beginning. Deactivation of mechanisms is advantageous in the case of ***contradictory mechanisms*** when mechanism with higher priority deactivates from the safety reasons its contradictory mechanisms. For instance a mechanism for spindle stop deactivates possibly developed mechanism for spindle start.

The mechanism may be looped and thus is advantageous to use also ***permanently looped mechanisms.*** These mechanisms have features of sequential drivers. The permanently looped mechanism must have in its body at least one instruction type EX !

By definition of mechanism *"mech"* is provided an automatic declaration of bit variable with a name "*mech*", declaration of memory cell type WORD for continuing address of mechanism and a memory cell type WORD with a name "mech_LINE" for actual program line in which is mechanism located. During compilation of PLC program by compiler TECHNOL (see next) is created an program listing with extension ".LS1" where program rows are numbered. In every state definition via instructions EX, EX0, EX1, TEX0, TEX1 and TIM is for inspection purposes a number of row written to memory cell *"mech*_LINE". It helps easy tracing of current mechanism state. If mechanism is not activated a memory cell "*mech*_LINE" is cleared.

Example:
Activating of mechanism named "CW" in introductory functions and waiting for mechanism execution:

```
FL      1,CW    ; Set activation variable.
EX
LDR     CW      ; Checking mechanism execution.
EX1             ;Wait until CW = 1
```



Example:
Activating of mechanism named "CW" in introductory functions and waiting 10 sec for mechanism execution. If not executed go to error treatment ERR1:

```
                EQUI        D500,500
                EQUI        CHYBA1,12H          ;ERROR 4.12

                FL          1,CW                ;Set activation variable.
                EX
                LDR         CW                  ;Checking mechanism execution.
                TEX1        CITAC,D500,ERR1     ;Wait when CW = 1, with time out 10 sec
                ....                             else go to ERR1
                ....
ERR1:           LOD         CHYBA1              ; Error treatment. Mechanism time out overflow
                STO         BZH11               ;
                ....
                ....
                ....
```

Example:

Write mechanism for spindle reversation from CW to CCW. Name mechanism CWCCW.

```
            MECH_BEGIN        CWCCW       ;beginning of mechanism.
            FL                0,SMP       ;clear contactor motor positive.
            LDR               KSMP
            EX1                           ;Wait when contact motor positive is 1.
            TIM               CAS1,D02    ;Delay 0,2 sec.
            FL                1,SMN       ;switch on contac. motor negative.
            LDR               KSMN
            EX0                           ;wait when contact motor negative is 0.
            MECH_END          CWCCW       ;end.
```

Notice:
in a actual case is better in mechanism CWCCW instead of instructions EX1 a EX0 use instructions TEX1 a TEX0, as is shown in following example.

Example:
Write mechanism for reversation of spindle revolution from CW to CCW. name mechanism CWCCW. If a contact of motor -positive contactor KSMP is not released in 1sec. go to error 4.04. If contact of motor-negative contactor KSMN is not switched on within 1. sec switch off motor contactor and go to error 4.05.

```
            EQUI      D1,50                   ;Time 1 sec.
            EQUI      D02,10                  ;time 0,2 sec.

MECH_BEGIN            CWCCW                   ;Beginning.
            FL        0,SMP                   ;switch off cont. motor-positive.
            LDR       KSMP
            TEX1      CITAC1,D1,CW_ERR,04     ;Wait when cont. motor- positive is 1.
            TIM       CITAC,D02               ;Delay 0,2 sec.
            FL        1,SMN                   ;switch on cont. motor-negative.
            LDR       KSMN
            TEX0      CITAC,D1,CW_ERR,05      ;wait when cont. motor-negative is 0 with.
                                              ;Time Out 1 sec
            JUM       CW_END                  ;Jump to end.
                                              ;Error treatment.
CW_ERR:     FL        0,SMN                   ;Switch off motor contactor.
            STO       BZH11                   ;Set error 4.05, 4.04
CW_END:
MECH_END              CWCCW                   ;End.
```

Example:
Ways of writing for instructions TEX0,TEX1 and TIM with a type redefining:

```
CITAC:      DS        100       ;array of counters
REKONFIG:   DS        100       ;array of redefinable variables

            TEX0      WORD.(CITAC+30),(REKONFIG+14),ERROR
            TEX1      BYTE.(CITAC+13),(REKONFIG+21),ERROR,23h
            TIM       WORD.(CITAC+20),(REKONFIG+10)
```