AMKASYN

VARIABLE SPEED DRIVES

AMKASYN

Digital Inverters in Modular Construction

Programmable Controller PS

PS Command Set

Version PSx V02.14

4200.E Part No.: 25795

Arnold Müller, Antriebs- und Steuerungstechnik GmbH & Co. KG, D-73230 Kirchheim/Teck,
Tel.: 07021/50 05-0, Telefax: 07021/50 05-176

Contents

1 Overview of the architecture of the PS processor

1.1 Introduction

The PS (AMKASYN programmable control) works through a program developed for the special control task and influences the process to be controlled depending upon external (e.g. process inputs) and internal influences (e.g. program, data).

There are so-called blocks in the PS

- for filing the control program (OB organization blocks,
  PB program blocks and FB function blocks) or
- for holding data (data blocks).

The control instruction is designated as the smallest independent unit of the control program. A control instruction, by reference to the STEP®5 programming language, consists in any event of an operation part which an operand part can follow.

The operand part in turn is divided into the operand identifier(s) and parameters.

```
e.g.:   U   E  5.7
        │   │  │ │
        │   │  │ │
        │   │  │ Parameter 2 1)
        │   │  Parameter 1 1)
        │   Operand identifier 1)
        Operation part
```

1) Operand part

Instructions can contain one, two or no parameters. There are also instructions which manage without operand identifiers.

1.2 Operand areas of the PS

Inputs (E):	Access is made to the PS-internal process image with the operand identifier "E".
Outputs (A):	Access is made to the PS-internal process image of the outputs with the operand identifier "A".
Flags (M):	With the "M" identifier it is possible to temporarily store binary results.
Periphery (P):	The identifier "P" allows direct access to the process (i.e. input and output modules) bypassing the process image.
Counters (Z):	Access to counting functions.

Timers (T):	Access to time functions.
Data (D):	With the "D" identifier it is possible to temporarily store digital results in data blocks.
Constants (K):	Values permanently set by the program are identified by K.
Blocks (OB,PB,FB,DB):	These identifiers facilitate access to the different program modules with subroutine technique.

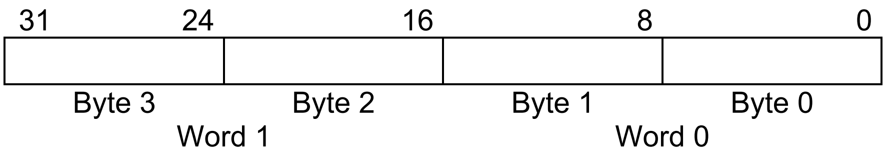
1.3 Registers of the PS processor

For executing the different control instructions, the PS has internal memory cells (registers, flags) which can be influenced directly or indirectly by the user. These registers are loaded, compared, operated on logically or transferred corresponding to the control instruction to be executed. Two registers appear mainly for the user:

(Bit) accumulator
This accumulator, with a processing width of 1 bit, is usually designated as the result of logic operation RLO. It is influenced by a large number of operations. Its main significance results from binary logic operations and in storage operations. A few conditional functions are executed or overridden depending upon the RLO.

(Double word) accumulator 1
This accumulator has a processing width of 32 bits.

Accumulator 1

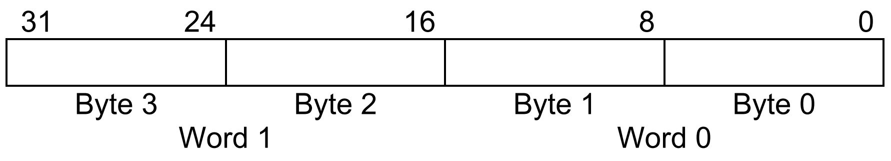


The accumulator 1 (ACCU1) is influenced essentially by the digital operations (e.g. loading functions, arithmetic functions, conversion functions, logic operation functions and shift functions). Transfer operations transfer the contents without changing the accumulator itself. Comparison operations do not influence the contents of the accumulator.

Further registers
A number of further memory cells which are not available to the user at the first glance are explained below. However, they can be observed and manipulated with APROS (cf. documentation: PS programming and test system APROS).

(Double word) accumulator 2
The accumulator 2 (ACCU2) has like accumulator 1 a processing width of 32 bits. However, it cannot be loaded directly and selectively with a value. A transfer of its contents directly is not possible.

Accumulator



It gains its significance in arithmetic and comparison operations which take account of its contents. Before loading operations the current contents of accumulator 1 are transferred into accumulator 2. The previous contents of accumulator 2 are lost in this case. Conversion operations and logic operations change only accumulator 1.

Erab
is used for managing the first input bit scan by the system software.

Erab has the following meaning:

High	next operation is first input bit scan;
LOW	next operation is no first input bit scan.

The Erab variable must be seen in combination with RLO-limiting (Erab is set "High") and NOT RLO-limiting operations. In RLO-limiting operations (e.g. an assignment "=") the RLO must be newly formed if necessary in a following operation, i.e. "first scanned".

Status
Represents the value of a current query (e.g. at U E 1.0 status represents the value of E 1.0). The current RLO is formed from the old RLO and the query value according to the logic operation.

OrStack
This variable is used as temporary storage for the RLO in the separate OR operation.

Ov
Overflow is a noting cell (display flag) of the PS with the values "1" or "0". This noting cell is influenced corresponding to the operation and the result (cf. e.g. comparison operations or arithmetic operations).

Anz0, Anz1
Anz0 and Anz1 are noting cells (display flags) which are influenced by different functions. Different conditional branch functions can be executed depending upon the contents of the flags.

NestingStack
The nesting stack is required for processing bracket expressions. It enables a nesting depth of 5 bracket levels.

2 Overview of the operation set

All realized functions can be assigned to one of the three operation groups listed below.

Operations

- **Binary operations**
  - Binary logic functions
  - Memory functions
  - Time functions
  - Counting functions
  - Bit test functions [1)]
- **Digital operations**
  - Loading functions
  - Transfer functions
  - Comparison functions
  - Digital logic operations
  - Arithmetic functions
  - Digital system functions [1)]
- **Organizational operations**
  - Block functions
  - Branch functions
  - Shift functions
  - Conversion functions
  - Processing functions [2)]
  - Other org. functions [2)]

1) These functions are not implemented completely currently.
2) These functions are partially implemented currently.

3 Command description

3.1 Binary operations

The following functions belong to the binary operations:

- Binary logic functions
  without brackets
  with brackets
- Memory functions
- Time functions
- Counting functions

These functions permit the logic operation of binary operands, such as inputs, outputs and flags, as well as the counter and timer outputs Q.

3.1.1 Binary logic functions

3.1.1.1 Binary logic functions without brackets

The AND and the OR operation as well as their negation belong as basic elements to the binary logic functions. Furthermore one includes the OR operation of AND functions, the AND operation of bracket expressions, the OR operation of bracket expressions as well as the brackets-closed instruction. Widely differing combinations are possible.

In the binary logic operations the operands

- inputs
- outputs
- flags
- timers and
- counters can be used.

The inputs/outputs/flags operand areas are defined by means of 2 addresses which are separated by a period. The address can be displayed in the form x.y. In this case x is the address of the byte (byte address) and y the number of the relevant bit (bit address) in the selected byte. Theoretically the byte address can assume values from 0 to 255. (However product-related there are restrictions of the highest value by the number of the configured input/output/flag bytes.) The bit address can assume values from 0 to 7 independently of the configuration.

Timers and counters are addressed through an address. This address is formed from the number in the PS-internal field of the timers or counters. Possible values for addresses are between 0 and 255. (The highest value is according to the configuration of the PS; cf. Section 7.1.) In the binary logic operations of timers and counters, their outputs Q are operated on logically corresponding to the function. The outputs of the counters indicate whether the counter value is 0 or unequal to 0. The timer outputs report whether the programmed time has elapsed or not.

Binary logic functions influence the logic operation result (RLO), but are always executed independently of the value of the RLO.

The following table provides an overview of the possible binary functions.

0	1	2	3	4	5
U E x.y	n	y	n		AND-Operat. w. E-query at 1
U A x.y	n	y	n		A-query at 1
U M x.y	n	y	n		M-query at 1
U T x	n	y	n		T-query at 1
U Z x	n	y	n		Z-query at 1
UN E x.y	n	y	n		AND-Operat. w. E-query at 0
UN A x.y	n	y	n		A-query at 0
UN M x.y	n	y	n		M-query at 0
UN T x	n	y	n		T-query at 0
UN Z x	n	y	n		Z-query at 0
O E x.y	n	y	n		OR-Operat. w. E-query at 1
O A x.y	n	y	n		A-query at 1
O M x.y	n	y	n		M-query at 1
O T x	n	y	n		T-query at 1
O Z x	n	y	n		Z-query at 1
ON E x.y	n	y	n		OR-Operat. w. E-query at 0
ON A x.y	n	y	n		A-query at 0
ON M x.y	n	y	n		M-query at 0
ON T x	n	y	n		T-query at 0
ON Z x	n	y	n		Z-query at 0

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

U AND operation
UN AND operation with neg. input
O OR operation
ON OR operation with neg. input

The following value table shows the value of the RLO after executing the function with all possible input assignments (prerequisite: Erab = 0):

Function	Operand/RLO =0 =0	Operand/RLO =0 =1	Operand/RLO =1 =0	Operand/RLO =1 =1
U	0	0	0	1
UN	0	1	0	0
O	0	1	1	1
ON	1	1	0	1

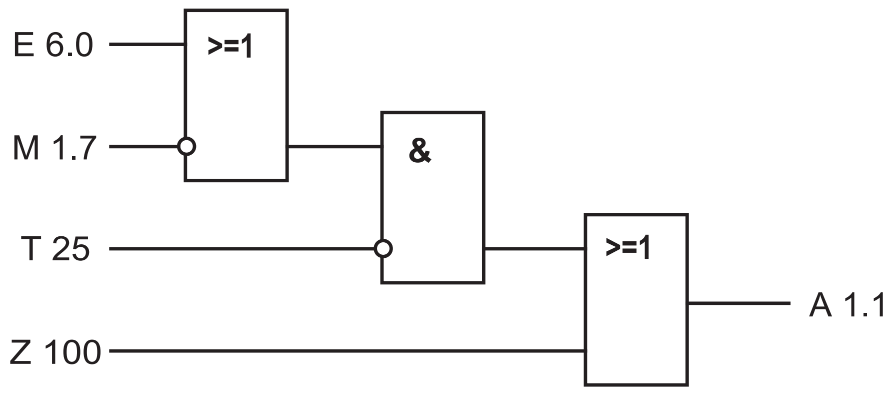
For instance, if the operand is 1 and the RLO before executing the function is also 1, then the RLO will contain the value 2 after executing an AND function.

If Erab = 1, the value of the operand (or the negated value) is assigned to the RLO.

Example:

```
                      ;Prerequisite: Erab = 1
:U     E 6.0          ;Transfer of the input bit 0 in the 6th input
                      ;byte into the RLO
:ON    M 1.7          ;OR operation of the negated 7th bit
                      ;of the 1st flag byte with the RLO
:UN    T 25           ;AND operation of the negated time
                      ;output 25 with the RLO
:O     Z 100          ;OR operation of the counter 100 with the RLO
:=     A 1.1          ;Current RLO is transferred to A 1.1,
                      ;Erab is set = 1
```

This sequence of AWL instructions corresponds to the following logic plan:



3.1.1.2 Binary logic functions with brackets

The PS processes the individual instructions in the normal case continuously corresponding to their sequence in the program, i.e. all logic operations are performed one after the other (sequentially). If it is necessary to deviate from sequential processing in the program, then the user can force the required processing sequence by means of bracket functions. The PS supports nesting a maximum of 5 bracket levels, i.e. a bracket-open instruction can be programmed up to 5 times without a bracket expression being ended previously. After processing the functions "O", "U(" or "O(", the current RLO is stored and a new value is formed from the now following functions. This is achieved by setting the first input bit scan in the PS. The function ")" represents the conclusion of a bracket-open instruction and implements the actual logic operation of the current RLO with the stored RLO. In contrast to the logic functions without brackets, logic functions with brackets limit the validity of the RLO.

The following table provides an overview of the possible functions:

0	**1**	**2**	**3**	**4**	**5**
O	n	y	y		OR operation before AND functions
U(	n	y	y		AND operation before bracket expressions
O(	n	y	y		OR operation before bracket expressions
)	n	y	y		Brackets closed

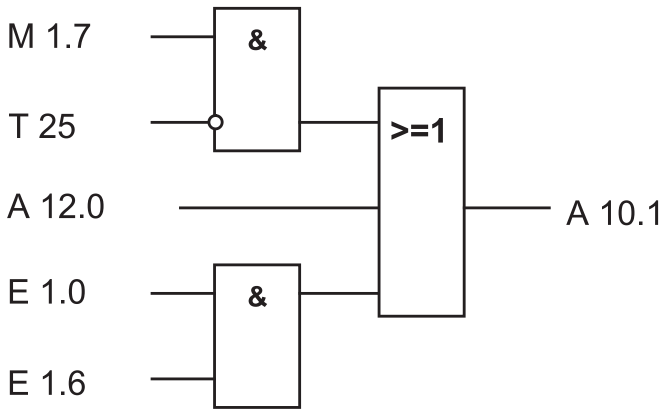
0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

O OR operation before AND functions
U(AND operation before bracket expressions
O(OR operation before bracket expressions
) Bracket

Example:

The following example shows how a logical function can be programmed with few instructions by using bracket commands. The program running time is also shortened by this.

The logic plan:



should be implemented in an AWL program.

There is a large number of possibilities for programming. The goal of programming must be to obtain a clear program with optimum program running time (cycle time). This requires compromises.

Version 1: A clear solution without brackets

```
:U      M 1.7
:UN     T 25
:=      M 0.1           ;Storing the result of the AND operation.
                        ;Next binary operation is
                        ;first input bit scan, i.e. RLO is newly formed.
:U      E 1.0
:U      E 1.6
:=      M 0.0           ;Storing the result of the AND operation.
                        ;Next binary operation is
                        ;first input bit scan, i.e. RLO is newly formed.

:O      A 12.0          ;OR operation of
:O      M 0.1           ;flags and
:O      M 0.0           ;A 12.0

:=      A 10.1          ;Assigning the total result to A 10.1
```

Version 2: Solution with brackets

```
:U     M 1.7       ;RLO is loaded with the flag bit 1.7
:UN    T 25        ;AND operation of the negated timer output 25
:O(                ;The current RLO is filed in the bracket memory
                   ;and with the next ")" command OR-operated with the
                   ;then valid RLO. Next command is the first input bit scan.
:U     E 1.0       ;The RLO is loaded with the E 1.0
:U     E 1.6       ;AND operation of the E 1.6 with current RLO
:)                 ;The current RLO is OR-operated with the
                   ;RLO standing in the nesting stack.
:O     A 12.0      ;OR operation of RLO and A 12.0

:=     A 10.1      ;Assigning the total result to A 10.1
```

Version 3: Solution without brackets with minimum number of instructions

```
:U     M 1.7
:UN    T 25
:O                 ;separate OR function
:U     E 1.0
:U     E 1.6
:O     A 12.0

:=     A 10.1
```

The PS is able to implement the AND-before-OR operation by means of "O(" function (version 2) or by means of separate OR function "O" (version 3).

In the example of version 3, the flag M 1.7 is AND-operated with the negated timer output 25. The formed RLO is stored at the "O" in the Or stack of the processor and the first input bit scan set. The processor then forms the RLO anew from the AND operation of the inputs 1.0 and 1.6. At the next "normal" OR or OR NOT function the RLO is OR'd with the OR stack before command execution and then the corresponding command is executed. Due to the following assignment the output A 10.1 receives the value of the RLO and thus of the entire operation.

Caution: In a "separated OR function" according to version 2, the Or stack is operated with the RLO by a following OR instruction with operand part, or a memory instruction ("S", "R", "="). The following example

```
U     M 1.7
UN    T 25
O                  ;separate OR function
U     E 1.0
U     E 1.6
BEB                ;conditional block end
```

is therefore not expedient, since with regard to the "Conditional block end" the Or stack is not yet operated on logically with the RLO for forming the "Conditional block end"! (The RLO is determined in the above example only by the two inputs E 1.0 and E 1.6!)

Furthermore the programmer is able with the aid of the "U(" function to program an OR-before-AND operation.

The following example illustrates this:

```
:U      E 0.1
:U(
:O      A 1.2
:ON     T 1
:)
:=      A 5.7
```

With the first AND instruction (first input bit scan), the RLO is loaded with input E 0.1. The current RLO moves due to the command "U(" into the nesting stack and the following operation is in turn a first input bit scan, i.e. the RLO is loaded with the output A 1.2. The OR operation then takes place with the negated timer output T1. The current RLO is operated on logically with the RLO temporarily stored in the nesting stack corresponding to the preceding bracket-open function with the now following bracket-closed function ")". If the last bracket-open function was the "U(" function, then the AND operation takes place, if it was the "O(" function, then the OR operation is performed.

3.1.2 Memory functions

Memory functions are used to influence bit operands. Approved operands are:

- Inputs
- Outputs and
- Flags

(Counter and timer outputs are not allowed)

A distinction is made between 3 types of memory functions:

Assignment: In the assignment (e.g. "=   E 1.0") the value of the currently valid RLO ("0" or "1") is assigned to the addressed operand, here the input bit 0 in the byte 1.

Setting: Setting (e.g. "S   A 2.5") is a conditional function. It depends upon the momentary value of the RLO. The output A 2.5 is set to "1" only if the momentarily valid RLO has the value "1". If the RLO has the value "0", then the output A 2.5 remains uninfluenced.

Resetting: Resetting also depends upon the value of the RLO. On resetting the addressed operand is set selectively to "0" (at RLO = "1") or not influenced (at RLO = "0").

The following table provides an overview of the possible instructions:

0			1	2	3	4	5
S	E	x.y	y	n	y		Setting an input (1) at RLO=1
S	A	x.y	y	n	y		Setting an output (1) at RLO=1
S	M	x.y	y	n	y		Setting a flag (1) at RLO=1
R	E	x.y	y	n	y		Resetting an input (0) at RLO=1
R	A	x.y	y	n	y		Resetting an output (0) at RLO=1
R	M	x.y	y	n	y		Resetting a flag (0) at RLO=1
=	E	x.y	n	n	y		Assigning an input with RLO
=	A	x.y	n	n	y		Assigning an output with RLO
=	M	x.y	n	n	y		Assigning a flag with RLO

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

S Setting
R Resetting
= Assigning

Example:

```
:O      E 0.0
:ON     E 0.0          ;RLO is now 1

                       ;Example of setting
:S      A 0.1          ;RLO = 1 -> Output 0.1 becomes 1

                       ;Example of resetting
:R      A 0.2          ;RLO = 1 -> Output 0.2 becomes 0

                       ;Example of an assignment
:=      A 1.1          ;Output 1.1 receives value of the RLO
```

3.1.2.1 Time functions

The operand area T of the PS is influenced by the time functions. The time functions permit temporal sequences to be controlled through the AWL program. Each timer is implemented in the PS by a time value (in the 1 ms grid) and a status variable (status).

The status is an internal system variable for identifying the current status. Only the binary output information of the status or the time value can be queried by the AWL program; however, further status information is also displayed by means of APROS (cf. documentation: PS programming and test system APROS).

The individual bits of the timer status have the following meaning:

Bit 0	Corresponds to the output of the timer. On querying the timer the new RLO is formed with this.	
Bit 1	Contains the RLO from the last processing at the start input of the timer.	
Bit 2	Identifies the status of the timer	
	= 0	Timer run down
	= 1	Time still running
Bit 3-7	Identifier for timer type	
	= 00000	Timer not started
	= 00001	Timer started as SI
	= 00010	Timer started as SV
	= 00100	Timer started as SE
	= 01000	Timer started as SS
	= 10000	Timer started as SA

A timer can be started in the form of the following types:
- SI Starting a time as pulse
- SV Starting a time as lengthened pulse
- SE Starting a time as switch-on delay
- SS Starting a time as storing switch-on delay
- SA Starting a time as switch-off delay

If a timer in the program was started e.g. as pulse (activated), and if in the further program sequence the same timer is started as another type, then the new type applies from now on for this timer.

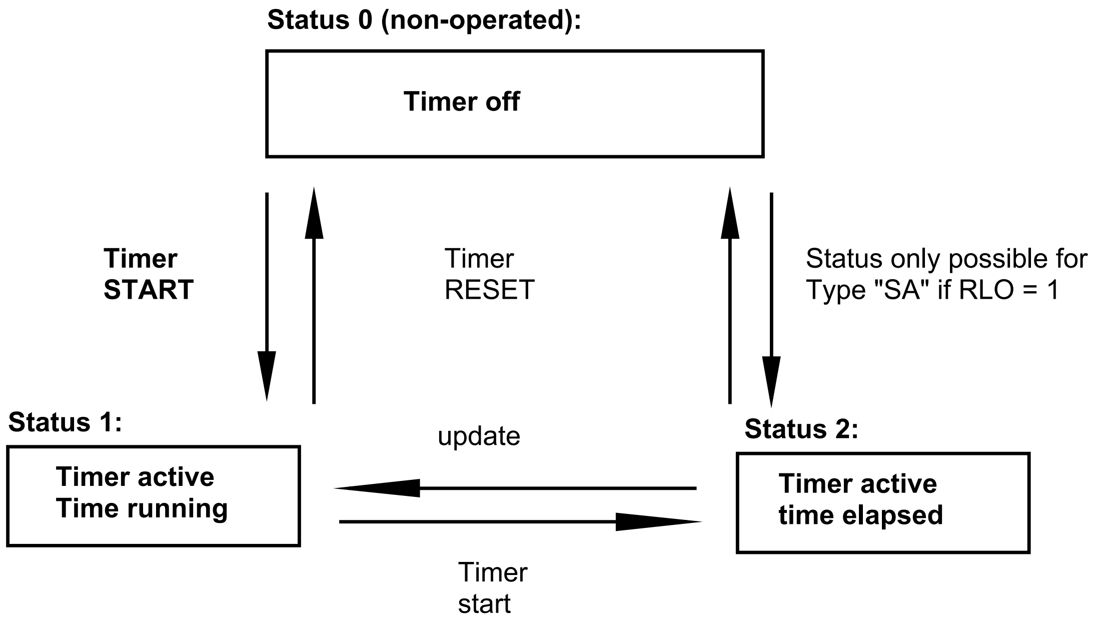
Each timer can assume internally 3 different statuses (cf. following figure):

Status 0 (basic status):
- Timer passive (timer not started)
  - Time value = 0
  - Timer status = 0
    - Output Q is 0 (Bit 0)
    - Old RLO is 0 (Bit 1)
    - Time not running (Bit 2 = 0)
    - Type flag (Bit 3-7 = 0)

Status 1:
- Timer active (timer started) / time runs
  - Time value unequal 0
  - Timer status unequal 0
    - Output Q corresponding to type set (Bit 0)
    - Old RLO corresponding to start input (Bit 1)
    - Time still running (Bit 2 = 1)
    - Type flag corresponding to type (Bit 3-7)

Status 2:

- Timer active (timer started) / time elapsed
  - Time value = 0
  - Timer status unequal 0
    - Output Q set corresponding to type (Bit 0)
    - Old RLO corresponding to start input (Bit 1)
    - Time no longer runs (Bit 2 = 0)
    - Type flag corresponding to type (Bit 3-7)


Status 0 (non-operated):
Timer off
Timer START
Timer RESET
Status only possible for Type "SA" if RLO = 1
Status 1:
Timer active Time running
update
Status 2:
Timer active time elapsed
Timer start

A timer is brought from the passive into the active status by a start command (SI, SV, SE, SS or SA). A timer can be brought only into the "Timer active / time elapsed" status by updating the process image of the inputs (PAE). A started timer can be "passivated" only by the reset command (R). An active timer the time of which has elapsed can also be restarted by a new start command (possibly also with changed type).

Values for status and time value of the different types at the start
(timer active / time runs):

				Status		
Command	**Explanation**	**Time value**	**Bit7-3**	**Bit2**	**Bit1**	**Bit0**
SI	Pulse	ACCU1	00001	1	1	1
SV	Lengthened pulse	ACCU1	00010	1	1	1
SE	Switch-on delay	ACCU1	00100	1	1	0
SS	Storing switch-on delay	ACCU1	01000	1	1	0
SA	Switch-off delay	ACCU1	10000	1	0	1

Values for status and time value of the different types with elapsed time **(timer active / time elapsed)** are generated on updating the process image of the inputs (PAE).

Command	Explanation	Time value	Bit7-3	Status Bit2	Status Bit1	Status Bit0
SI	Pulse	0	00001	0	x	0
SV	Lengthened pulse	0	00010	0	x	0
SE	Switch-on delay	0	00100	0	x	1
SS	Storing switch-on delay	0	01000	0	x	1
SA	Switch-off delay	0	10000	0	x	0

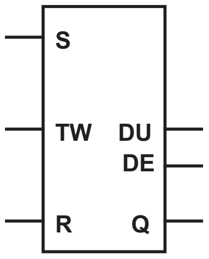
x Values are not influenced

On resetting a timer, both status and time value are set to 0. The command (R T x) is used exclusively for this.

Command	Explanation	Time value	Bit7-3	Status Bit2	Status Bit1	Status Bit0
R	Resetting	0	00000	0	x	0

x Values are not influenced

A timer can be displayed symbolically as follows:



Timer:

S	= Input START timer
TW	= Timer Preset value
Q	= Timer output
DU	= Time value output (dual)
DE	= Time value output (dec.)
R	= Input RESET timer

The timer has two binary inputs (S, R) and one binary output (Q). These act statically or dynamically according to timer type. The dual input TW serves for loading the 32-bit time value on starting the timer. The 32-bit value corresponds to the time in milliseconds. E.g., the value 32000 must be loaded for 32 seconds. The current time value of the timer can be loaded into the accumulator 1 by means of the load commands "L Tx" (DU time value dual) or "LC Tx" (DE time value BCD) (see Section 3.2.1). Thus functions are available e.g. for arithmetic operations or comparison functions.

The following table provides an overview of the possible instructions:

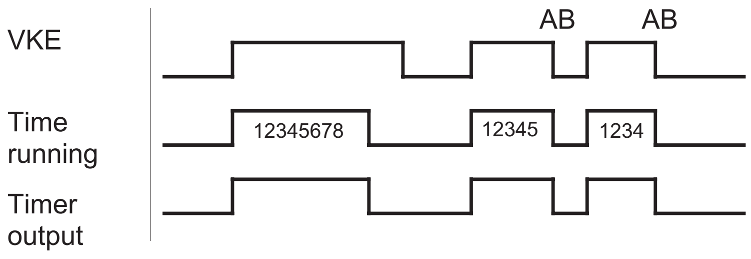
0			1	2	3	4	5
SI	T	x	y	n	y		Start of a time as pulse
SV	T	x	y	n	y		Start of a time as lengthened pulse
SE	T	x	y	n	y		Start of a time as switch-on delay
SS	T	x	y	n	y		Start of a time as storing switch-on delay
SA	T	x	y	n	y		Start of a time as switch-off delay
R	T	x	y	n	y		Resetting a time

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

SI Starting a time as pulse

The time is started with a status change of the starting RLO from "0" to "1". The binary output of the timer carries "1" during the running time. After the end of the time the output changes to "0". If the starting RLO drops back to "0" before the end of the time, the time is stopped and the output of the timer is set to "0".

Start condition RLO 0 --> 1
Output : = 1 with RLO, as longas RLO and time run



Set time = 8
AB: Abort

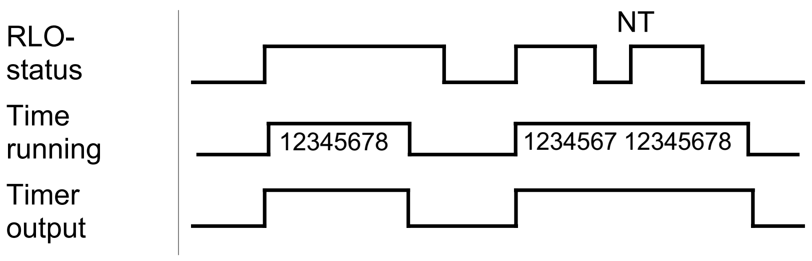
Example:

```
:U      E 1.0          ;Forming the RLO for start of the timer
:L      KT 1000        ;Providing the time value in Accu 1
:SI     T 5            ;If E 1.0 changes from 0->1, the timer 5 is started
                       ;as pulse through the ACCU1 with 1000x1 ms = 1 s
:U      T 5            ;Query, whether time has elapsed.
                       ;RLO=1 as long as time runs
```

SV Starting a time as lengthened pulse

The time is started with a status change of the starting RLO from "0" to "1". The binary output of the timer carries "1" during the running time. After the end of the time the output changes to "0". If the starting RLO drops back to "0" before the end of the time, the time runs on nevertheless. The output of the timer becomes "0" only if the time has completely elapsed. If further rising edges occur at the start input during the running time of the timer, the timer is retriggered.

Start condition RLO 0 --> 1
Output: = 1, if RLO = 1 and timer is running



Programmed time = 8
NT: Retriggering

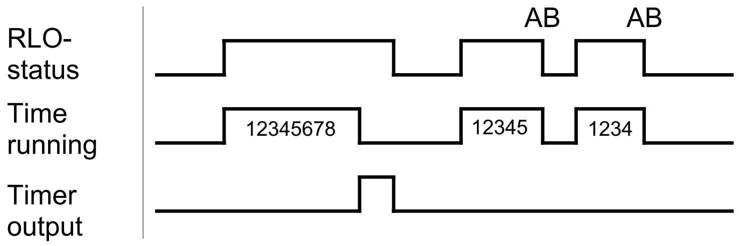
Example:

```
:U     E 1.0        ;Forming the RLO for start of the timer
:L     KT 1000      ;Providing the time value in Accu 1
:SV    T 5          ;If E 1.0 changes from 0->1, the timer 5 is started
                    ;as lengthened pulse through the ACCU 1
                    ;with 1000x1 ms = 1 s
:U     T 5          ;Query, whether time has elapsed.
                    ;RLO = 1 as long as time runs
```

SE Starting a time as switch-on delay

The time is started with a status change of the starting RLO from "0" to "1". The binary output of the timer carries "0" during the running time. The output is set to 1 only after the end of the time, provided the input signal (RLO) is still present with "1". The output is reset without a delay as soon as the signal at the start input (RLO) goes back to "0" or the reset function is activated. Start signals the duration of which are shorter than the programmed delay time result in no output signal of the timer.

Start condition RLO 0 --> 1
Output = 1, if RLO = 1 and timer is running



Programmed time = 8
AB: Abort

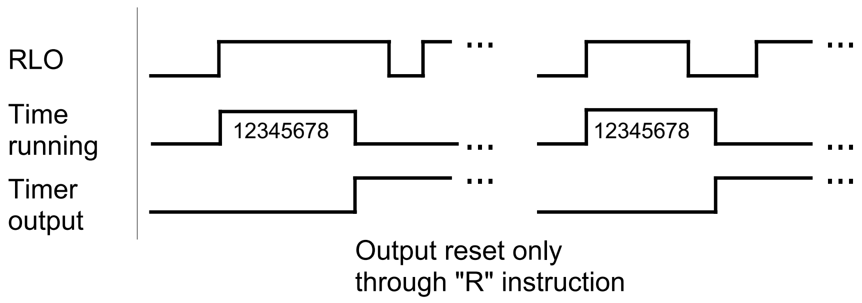
Example:

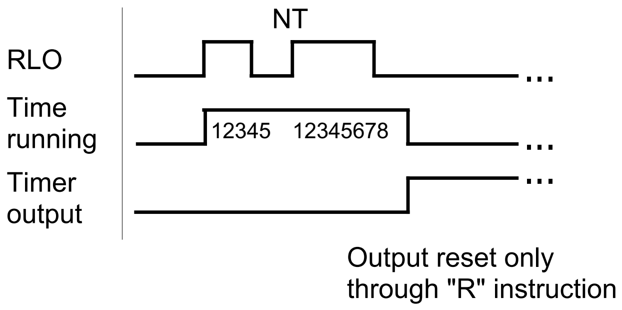
```
:U     E 1.0        ;Forming the RLO for start of the timer
:L     KT 1000      ;Providing the time value in Accu 1
:SE    T 5          ;If E 1.0 changes from 0->1, the timer 5 is started
                    ;through the ACCU 1 with 1000x1 ms = 1 s
                    ;as switch-on delay
:U     T 5          ;Query, whether time has elapsed.
                    ;RLO = 1 as soon as time elapsed
```

SS Starting a time as storing switch-on delay

The time is started with a status change of the starting RLO from 0 to 1. The binary output of the timer carries "0" during the running time. The output is set to "1" only after the end of the time, independent of the status of the signal at the start signal (RLO). The output of the timer must be switched off by a reset command. The started time can be retriggered with a renewed "0" - "1" transition of the signal at the start input.

Start condition: RLO 0 --> 1
Output: = 1, only if time has elapsed, independent of RLO





Programmed time = 8
NT: Retriggering

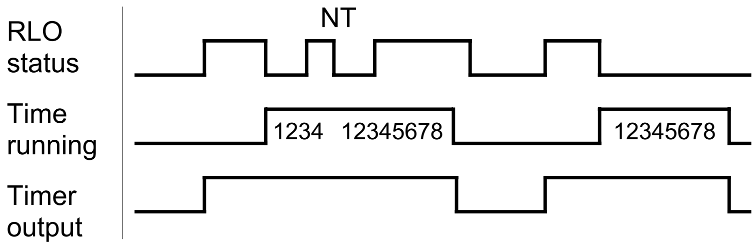
Example:

```
:U     E 1.0        ;Forming the RLO for start of the timer
:L     KT 1000      ;Providing the time value in Accu 1
:SS    T 5          ;If E 1.0 changes from 0->1, the timer 5 is started
                    ;through the ACCU 1 with 1000x1 ms = 1 s
                    ;as storing switch-on delay
:U     T 5          ;Query, whether time has elapsed.
                    ;RLO = 1 as soon as time elapsed
```

SA Starting a time as storing switch-off delay

The switch-on edge of the signal appears undelayed at the output of the timer. If the status of the input signal changes from "1" to "0", the delay time is started in the timer. The output of the timer assumes the value "0" only after the end of the programmed time. The timer can be retriggered if the input signal is switched on and off again during the running delay time. The output then remains at 1 until the time after the last switch-off edge has elapsed. Resetting the timer through "1" signal at the R input is possible only if the switch-off delay runs!

Start condition RLO 0 --> 1
Output = 1, if RLO = 1 and remains "1" until time elapsed



Programmed time = 8
NT: Retriggering

Example:
```
:U    E 1.0      ;Forming the RLO for start of the timer
:L    KT 1000    ;Providing the time value in Accu 1
:SA   T 5        ;If E 1.0 changes from 1->0, the timer 5 is started
                 ;through the ACCU 1 with 1000x1 ms = 1 s
                 ;as storing switch-off delay
:U    T 5        ;Query, whether time has elapsed.
                 ;RLO = 1 until time elapsed
```

R Resetting a time

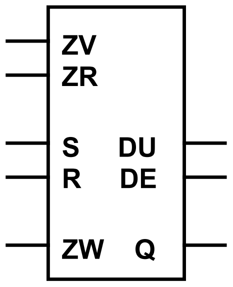
The stated timer is reset if the signal at the reset input R is "1" (RLO "1"). This operation acts statically, i.e. no edge change RLO "0"->"1" is required for resetting. The start of a timer remains blocked as long as RLO "1" acts at the reset input. On resetting the running time is interrupted and the output of the timer is set to "0".

Special case SA can be reset only if the delay time runs, i.e. after a "1"->"0" transition of the RLO at the start input.

Example:
```
:U    E 1.0      ;Forming the RLO for start of the timer
:L    KT 1000    ;Providing the time value in Accu 1
:SI   T 5        ;If E 1.0 = 1, the timer 5 is started through
                 ;the ACCU 1 with 1000x1 ms = 1 s as pulse
:U    E 0.0      ;Resetting only if E 0.0 = 1
:R    T 5
:U    T 5        ;Query, whether time has elapsed
                 ;RLO = 1 as long as time runs
```

3.1.3 Counting functions

Counting functions influence the counter operand area. A counter is displayed with the following symbol:



ZV	= Count up
ZR	= Count down
S	= Set counter
R	= Reset counter
Q	= Counter output
DU	= Actual counter value (binary)
DE	= Actual counter value (decimal)
ZW	= Counter Preset value (binary)

The counter has 5 inputs and 3 outputs. The binary inputs ZV, ZR and S act dynamically, i.e. only on edge change 0->1, whereas the reset input R influences the counter statically. If a counter is set (edge change 0->1 at the input S), then the accumulator 1 must be loaded previously with the specified count value. This value is taken over in the counter. The digital input ZW which enables inputting a 16-bit count value is used for this. If there is an edge change 0->1 at the ZV or ZR input, then the current count value of the counter is incremented or decremented by 1. If the reset input is effective, the count value and the binary counter output become zero. The current counter status can be loaded into the accumulator by means of the loading commands "L Z x" (DU counter value binary) or "LC Zx" (DE counter value BCD) (see Section 3.2.1). It is thus available for arithmetic operations or comparison functions. The binary output Q depends upon the counter status, and the following assignments apply:

- Counter status = 0 -> counter output Q = 0
- Counter status > 0 -> counter output Q = 1

The output Q of a counter thus forms the new logic operation result in the binary (qualitative) interrogation of the counter. The counting functions do not influence the RLO and do not restrict its validity.

The following table shows the possible commands for influencing the counter operand area:

0			1	2	3	4	5
ZV	Z	x	y	n	y		Forwards counting of a counter
ZR	Z	x	y	n	y		Backwards counting of a counter
S	Z	x	y	n	y		Setting a counter
R	Z	x	y	n	y		Resetting a counter

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

ZV Z x Forwards counting

A positive edge change of the RLO at the input ZV increases the counter status by 1 up to the count upper limit MAXCOUNT (e.g. 65535; cf. Section 7.1). If MAXCOUNT is reached, the count value is not further increased.

Example:

```
:L      KZ 150        ;Count value = 150
:U      E 1.0         ;Setting input is E 1.0
:S      Z 1           ;Setting the counter 1 to preset value 150
                      ;on changing E 1.0 from 0->1
:U      E 5.0         ;Input 5.0 is forwards counting input
:ZV     Z 1           ;If an edge change takes place at E 5.0, the
                      ;current count value is increased -> count value now 151
```

ZR Z x Backwards counting

A positive edge change of the RLO at the input ZR reduces the counter status by 1 down to a minimum value of 0. The transition into the negative range by further counting pulses is blocked.

Example:

```
:L      KZ 150        ;Count value = 150
:U      E 1.0         ;Setting input is E 1.0
:S      Z 1           ;Setting the counter 1 to preset value 150
                      ;on changing E 1.0 from 0->1
:U      E 5.0         ;Input 5.0 is backwards counting input
:ZR     Z 1           ;If an edge change takes place at E 5.0 the
                      ;current count value is reduced -> count value now 149
```

S Z x Setting counter

With a positive edge change of the RLO at the input S the counter is set to its starting value.

Example:

```
:L      KZ 150       ;Count value = 150
:U      E 1.0        ;Setting input is E 1.0
:S      Z 1          ;Setting the counter 1 to
                     ;preset value 150 on 0->1 edge
```

R Z x Resetting counter

With a statistical “1“ signal (RLO) at the input R (no edge evaluation!) the counter status and output Q are set to the value 0. The reset input is dominant compared with the other inputs.

Example:

```
:L      KZ 150       ;Count value = 150
:U      E 1.0        ;Setting input is E 1.0
:S      Z 1          ;Setting the counter 1 to
                     ;preset value 150 on 0 ->1 edge
        .
:U      E 5.0        ;Input 5.0 is reset input.
:R      Z 1          ;Counter is reset at RLO=1
```

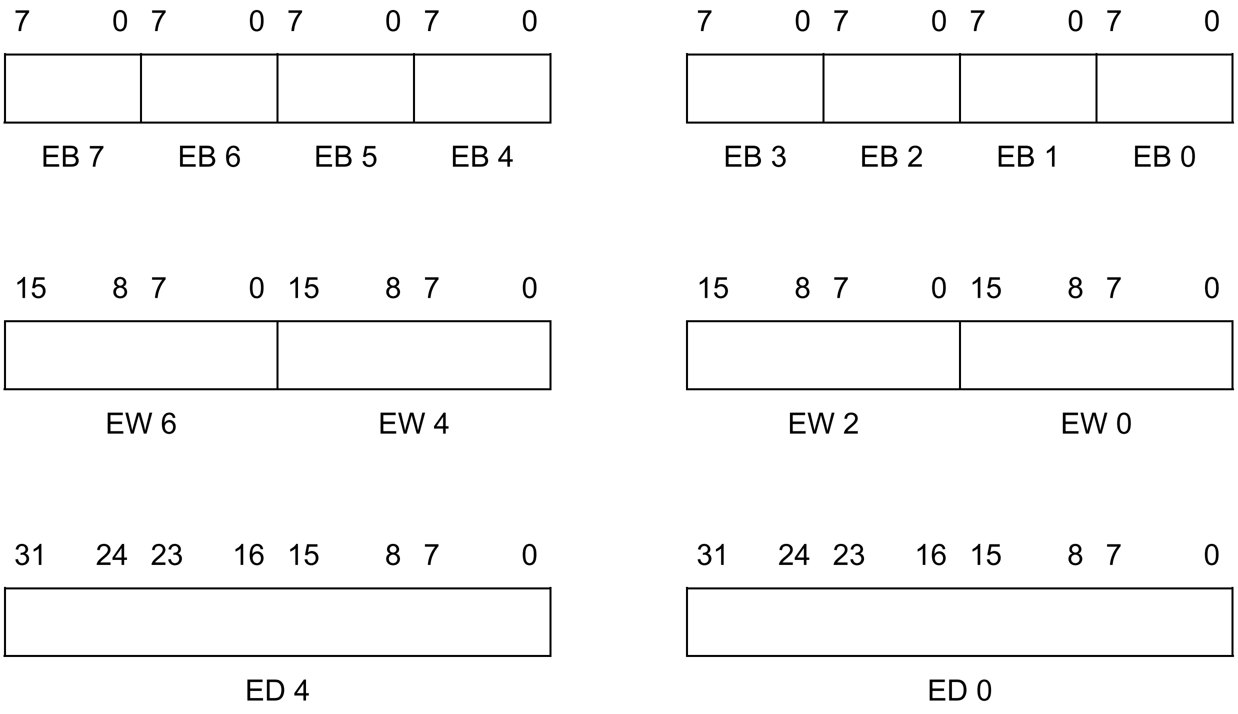
3.2 Digital operations

3.2.1 Loading functions

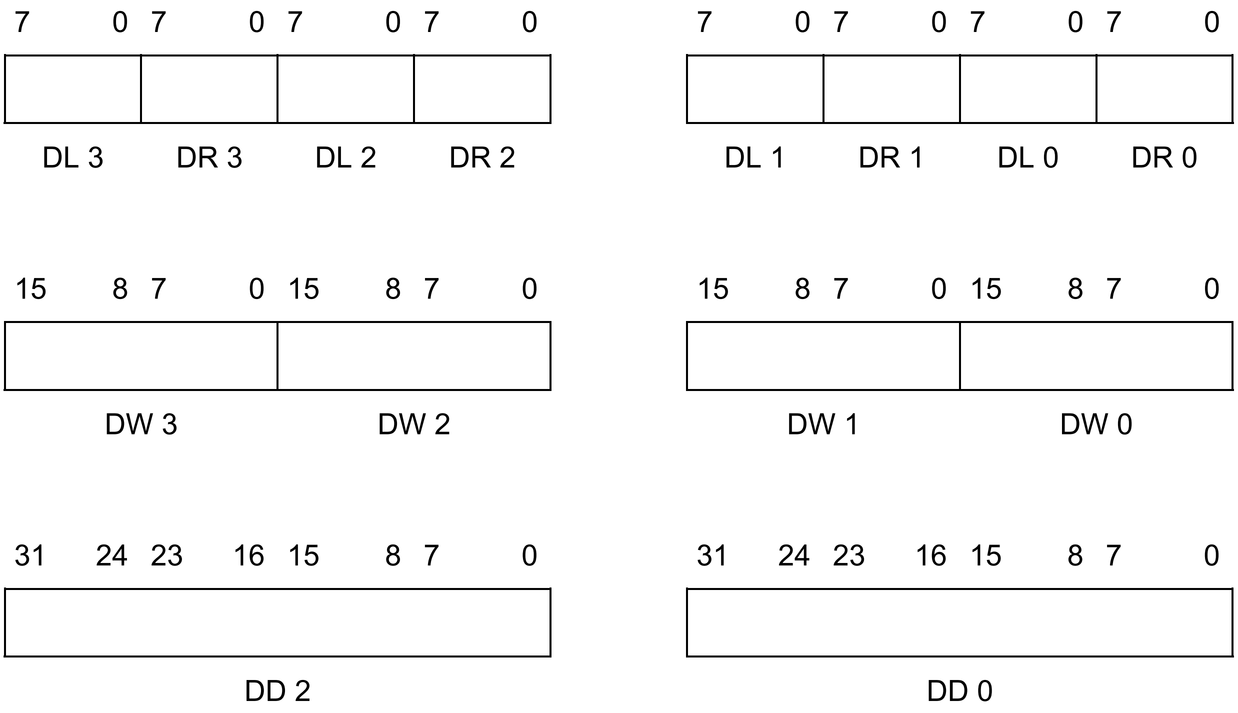
Loading functions are executed independently of the RLO and do not influence this. It is possible only to loaded directly into the accumulator 1. Before a loading command the contents of accumulator 1 are basically transferred into accumulator 2. The original value in Accu 2 is overwritten in this case. The loading functions influence most operand areas of the PS. The different functions enable the operand areas to be processed byte-wise, word-wise or double word-wise. The accumulator 1 has a processing width of 32 bits. The more significant part in the byte and word functions is filled with “0“. The operand is entered right justified. The operands themselves are not influenced by the loading functions. With negative byte or word values, the leading bit places of the accumulator 1 must be occupied with “1“ by AWL instruction.

Remarks:

The addresses (parameters) for E/A/M/P operands are basically byte addresses. In word commands modulo-2 addresses (0, 2, 4, ...) must be used exclusively. In double word commands modulo-4 addresses (0, 4, 8, ...) must be used exclusively.



The addresses (parameters) for data block operands are basically word addresses. In double word commands modulo-2 addresses (0, 2, 4, ...) must be used exclusively.



The significance of the operands rises basically from the lower to the higher address.

The following table shows the possible commands for the loading functions:

0			1	2	3	4	5
L	EB	x	n	n	n		Load input byte x in Accu 1
L	AB	x	n	n	n		Load output byte x in Accu 1
L	MB	x	n	n	n		Load flag byte x in Accu 1
L	EW	x	n	n	n		Load input word x in Accu 1
L	AW	x	n	n	n		Load output word x in Accu 1
L	MW	x	n	n	n		Load flag word x in Accu 1
L	ED	x	n	n	n		Load input double word x in Accu 1
L	AD	x	n	n	n		Load output double word x in Accu 1
L	MD	x	n	n	n		Load flag double word x in Accu 1
L	PY	x	n	n	n		Load periphery byte x in Accu 1
L	PW	x	n	n	n		Load periphery word x in Accu 1
L	PD	x	n	n	n		Load periphery double word x in Accu 1
L	DL	x	n	n	n		Load left data byte x in Accu 1
L	DR	x	n	n	n		Load right data byte x in Accu 1
L	DW	x	n	n	n		Load data word x in Accu 1
L	DD	x	n	n	n		Load data double word x in Accu 1
L	KB	x	n	n	n		Load const. (1 byte) x in Accu 1
L	KC	xy	n	n	n		Load const. (2 ASCII characters) x in Accu 1
L	KF	x	n	n	n		Load const. as fixed point number (word) x in Accu 1
L	KH	x	n	n	n		Load const. as hex number in Accu 1
L	KM	x	n	n	n		Load const. as bit pattern in Accu 1
L	KY	x,y	n	n	n		Load const. as 2 bytes in Accu 1
L	KT	x	n	n	n		Load const. time value (grid=1 ms) in Accu 1
L	KZ	x	n	n	n		Load const. count value in Accu 1
L	KD	x	n	n	n		Load const. as fixed point number (D word) in Accu 1
L	T	x	n	n	n		Load value of timer (1 ms) in Accu 1
LC	T	x	n	n	n		Load value of timer BCD-coded in Accu 1
L	Z	x	n	n	n		Load value of counter in Accu 1
LC	Z	x	n	n	n		Load value of counter BCD-coded in Accu 1

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

L EB x Load input byte x

Example:

```
:L      KD 1025        ;Accu 1 = 1025
:L      EB 0           ;Loading the 0th byte in the inputs operand area
                       ;in the accumulator 1
```

Before executing the instruction :L EB 0

Accumulator 1 = 1025

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Operand (e.g. EB 0) = 19

0 0 0 1 0 0 1 1

After executing the instruction :L EB 0

Accumulator 1 = 19

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 0 0 1 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 1025

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

The same principle is used in byte-wise loading of outputs and flags.

L AB x Load output byte x

Example:

```
:L      AB 0           ;Loading the 0th byte in the outputs operand area
                       ;in the accumulator 1
```

L MB x Load flag byte x

Example:

```
:L      MB 0           ;Loading the 0th byte in the flags operand area
                       ;in the accumulator 1
```

L EW x Load input word x

Example:

```
:L      KD 1025      ;Accu 1 = 1025
:L      EW 0         ;Loading the 0th word in the inputs operand area
```

Before executing the instruction :L EW 0

Accumulator 1 = 1025

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Operand (e.g. EW 0) = 1537

0 0 0 0 0 1 1 0	0 0 0 0 0 0 0 1
Byte 1	Byte 0

After executing the instruction :L EW 0

Accumulator 1 = 1537

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 1 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 1025

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

The same principle is used in the word-wise loading of outputs, flags, data and periphery.

L AW x Load output word

Example:

```
:L      AW 0         ;Loading the 0th value in the outputs operand area
```

L MW x Load flag word

Example:

```
:L      MW 0         ;Loading the 0th word in the flags operand area
```

L ED x Load input double word x

Example:

```
:L      KD 1025         ;Accu 1 = 1025
:L      ED 0            ;Loading the 0th double word
                        ;in the inputs operand area
```

Before executing the instruction :L ED 0

Accumulator 1 = 1025

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Operand (e.g. ED 0) = 2.000.000.000

24	16	8	0
0 1 1 1 0 1 1 1	0 0 1 1 0 1 0 1	1 0 0 1 0 1 0 0	0 0 0 0 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

After executing the instruction :L ED 0

Accumulator 1 = 2.000.000.000

24	16	8	0
0 1 1 1 0 1 1 1	0 0 1 1 0 1 0 1	1 0 0 1 0 1 0 0	0 0 0 0 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 1025

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

The same principle is used in double word-wise loading of outputs, flags, data and periphery.

L AD x Load output double word x

Example:

```
:L      KD 1025         ;Accu 1 = 1025
:L      AD 0            ;Loading the 0th double word
                        ;in the outputs operand area
```

L MD x Load flag double word x

Example:

```
:L      KD 1025      ;Accu 1 = 1025
:L      MD 0         ;Loading the 0th double word
                     ;in the flags operand area
```

L PY x Load input byte x directly from periphery

Example:

```
:L      PY 0         ;Loading the 0th byte in the periphery area
```

L PW x Load input word x directly from periphery

Example:

```
:L      PW 0         ;Loading the 0th word in the periphery area
```

L PD x Load input double word x directly from periphery

Example:

```
:L      PD 0         ;Loading the 0th double word in the periphery area
```

L DL x Load left byte (high byte) of the data word

Example:

```
:L      KD 1025      ;Accu 1 = 1025
:L      DL 0         ;Loading the HIGH byte in the 0th
                     ;word in the data operand area
```

Before executing the instruction

Accumulator 1 = 1025

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Operand (e.g. DW 0) = 1537

0 0 0 0 0 1 1 0	0 0 0 0 0 0 0 1
Byte 1	Byte 0

After executing the instruction

Accumulator 1 = 6

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 1 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 1025

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L DR x Load right byte (low byte) of the data word

Example:

```
:L      KD 1025      ;Accu 1 = 1025
:L      DR 0         ;Loading the LOW byte in the 0th
                     ;word in the data operand area
```

Before executing the instruction

Accumulator 1 = 1025

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Operand (e.g. DW 0) = 1537

0 0 0 0 0 1 1 0	0 0 0 0 0 0 0 1
Byte 1	Byte 0

After executing the instruction

Accumulator 1 = 1

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 1025

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L DW x Load whole data word

Example:

```
:L      DW 0         ;Loading the 0th word in the data operand area
```

L DD x Load whole data double word x

Example:

```
:L      DD 0         ;Loading the 0th double word
                     ;in the data operand area
```

L KB x Load byte constant x in the decimal format

Loading an 8-bit number as positive whole decimal number in the accumulator 1.

Example:

```
:L      KB 200       ;Loading the value 200 in the accumulator 1
                     ;(right justified) the previous Accu 1 value
                     ;now stands in Accu 2.
                     ;The value range comprises 0 <= x <= 255
```

Accumulator 1 = 200

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 1 0 0 1 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L KC XY Load ASCII constant XY

Loading two characters (without separation by comma!) in the accumulator 1. Only valid ASCII characters are allowed. After execution of the function the values corresponding to the ASCII characters stand in accumulator 1.

Example:

```
:L      KC AB        ;Loading the values A and B in the
                     ;accumulator 1 (right justified)
                     ;the previous Accu 1 value
                     ;now stands in Accu 2
```

Accumulator 1 41 H = "A“ 42 H = "B“

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 1	0 1 0 0 0 0 1 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L KF x Load word constant x in the fixed point number format

Loading an16-bit fixed point number in the accumulator 1. The number must be stated as a whole decimal number in the AWL program.

Example:

```
:L      KF 6000       ;Loading the value 6000 in the accumulator 1
                      ;(right justified) as 32-bit fixed point number. The
                      ;previous Accu 1 value now stands in Accu 2
                      ;The value range comprises -32768 <= x <= +32767
```

Accumulator 1 = 1770 H = 6000

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 0 1 1 1	0 1 1 1 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L KH x Load word constant in the hex format

Loading a 16-bit value in the accumulator 1. The number must be stated in the AWL program as a 4-digit hex number.

Example:

```
:L      KH FFFF       ;Loading the value 65535 in the accumulator 1
                      ;(right justified) as 32-bit fixed point number. The
                      ;previous Accu 1 value now stands in Accu 2
                      ;The value range comprises 0 <= x <= FFFF H
```

Accumulator 1 = FFFFH = 65535

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L KM x Load word constant in the binary format

Loading a 16-bit value in the accumulator 1. The number must be stated in the AWL program as a 16-digit bit pattern.

Example:

```
:L      KM 0000000010101010      ;Loading the value 170 in the Accu 1
                                 ;(right justified) as 32-bit fixed point number
                                 ;The previous Accu 1 value now stands in
                                 ;Accu 2. The value range comprises a 16-
                                 ;digit dual number
```

Accumulator 1 = 00AAH = 170

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 1 0 1 0 1 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L KY x,y Load 2 byte const. in the decimal format

Loading two 8-bit values which are separate by comma in the accumulator 1. The values must be stated as positive whole decimal values in the AWL program.

Example:

```
:L      KY 100,64     ;Loading the values 100 and 64 in the accumulator 1
                      ;(right justified). The previous Accu 1 value
                      ;now stands in Accu 2
                      ;The value range comprises  0 <= x <= 255
                          ;                       0 <= y <= 255
```

Accumulator 1		100	64
24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 1 1 0 0 1 0 0	0 1 0 0 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L KT x Load time value constant

Loading a time value in the accumulator 1. The value must be stated in the AWL program as positive whole decimal number and corresponds to a time value in the 1 millisecond grid. The largest time value is 2 147 483 647 ms. This corresponds to approx. 24 days.

Example:

```
:L      KT 100000     ;Loading the time value 100,000 ms = 100 seconds in
                      ;the accumulator 1 (right justified). The previous
                      ;Accu 1 value now stands in Accu 2
                      ;The value range comprises 0 <= x <= 2,147,483,647
```

Accumulator 1		186A0H = 100000	
24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	1 0 0 0 0 1 1 0	1 0 1 0 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L KZ x Load count value constant

Loading a count value in the accumulator 1. The value is stated in the AWL program as positive whole decimal number.

Example:

```
:L      KZ 30000      ;Loading the count value 30,000 in the Accu 1
                      ;(right justified). The previous Accu 1 value
                      ;now stands in Accu 2
                      ;The value range comprises 0 <= x <= 65535
```

Accumulator 1		7530H = 30000	
24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 1 1 1 0 1 0 1	0 0 1 1 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L KD x Load double word const. in the decimal format

Loading a 32-bit value in the accumulator 1. The value can be positive or negative and must be stated in the AWL program as whole number decimal value.

Example:

```
:L      KD -2147483647      ;Loading the double word
                            ;-2.147.483.647 in the accumulator 1
                            ;(right justified). The previous Accu 1 value
                            ;now stands in Accu 2
                            ;The value range comprises  -2147483647 <= x
                            ;                            2147483647 >= x
```

Accumulator 1 80000001 H = -2147483647

24	16	8	0
1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L T x Load current time value

The current time value of the timer x is loaded in the accumulator 1. After execution of the command the momentary time value of the counter in milliseconds stands in accumulator 1. It may contain only positive values.

Example:

```
:O      E 0.0
:ON     E 0.0       ;RLO always = 1
:L      KT 100000   ;100 s time value in Accu 1
:SI     T 1         ;Set timer 1 to pulse 100 s
:L      KB 0        ;Delete Accu 1
:L      T 1         ;Momentary value from T 1 in Accu 1
```

Accumulator 1 186A0 H = 100000

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	1 0 0 0 0 1 1 0	1 0 1 0 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

LC T x Load current time value BCD-coded

The current time value of the timer x is loaded BCD-coded in the accumulator 1.

Caution: The difference from the binary coded representation (cf. L T x) is stated in the BCD representation of the time value in seconds.

Example:

```
:O      E 0.0
:ON     E 0.0          ;RLO always = 1
:L      KT 100000      ;100 s time value in Accu 1
:SI     T 1            ;Set timer 1 to pulse 100 s
:L      KB 0           ;Delete Accu 1
:LC     T 1            ;Momentary value from T 1 in Accu 1 BCD-coded
```

Accumulator 1 100 BCD

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

L Z x Load current count value

Loading the current count value from the counter x in accumulator 1. The count value is available after execution of the function as 32-bit value in accumulator 1. It can assume only positive values from 0 to 65535.

Example:

```
:O      E 0.0
:ON     E 0.0          ;RLO always = 1
:L      KZ 10000       ;Count value 10000 in Accu 1
:S      Z 1            ;Set counter to count value
:L      KB 0           ;Delete Accu 1
:L      Z 1            ;Momentary value from T 1 in Accu 1 DUAL-coded
```

Accumulator 1 2710 H = 10000

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 1 0 0 1 1 1	0 0 0 1 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

LC Z x Load current count value BCD-coded

Loading the current count value from the counter x in the accumulator 1. The count value is available after execution of the function as BCD-coded 32-bit value. A maximum of 5 places are used.

Example:

```
:O     E 0.0
:ON    E 0.0          ;RLO always = 1
:L     KZ 10000       ;Count value 10000 in Accu 1
:S     Z 1            ;Set counter to count value
:L     KB 0           ;Delete Accu 1
:LC    Z 1            ;Momentary value from T 1 in Accu 1 BCD-coded
```

Accumulator 1 10000 H = 65536

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

3.2.2 Transfer functions

Transfer functions are executed independently of the RLO and do not influence this. The transfer functions influence most operand areas of the PS. The different functions enable the operand areas to be processed byte-wise, word-wise or double word-wise. A value from the 32-bit accumulator 1 is always processed "right justified". If there is a transfer to an operand byte-wise or word-wise, then the more significant part of the accumulator is cut off. It is possible to transfer only directly from the accumulator 1. The two accumulators themselves are not influenced by the transfer functions.

A special feature must be observed in the transfer functions to the periphery. If a value (byte, word, double word) is written to the periphery, then the process image of the outputs is automatically updated.

The addresses (parameters) for E/A/M/P operands are basically byte addresses. In word commands modulo-2 addresses (0, 2, 4, ...) must be used exclusively. In double word commands modulo-4 addresses (0, 4, 8, ...) must be used exclusively (cf. Section 3.2.1).

The addresses (parameters) for data block operands are basically word addresses. In double word commands modulo-2 addresses (0, 2, 4, ...) must be used exclusively (cf. Section 3.2.1).

The significance rises basically from the low to the higher address.

The following table shows the possible commands for the transfer functions:

0			1	2	3	4	5
T	EB	x	n	n	n		Transf. byte from Accu 1 to the input byte x
T	AB	x	n	n	n		Transf. byte from Accu 1 to the output byte x
T	MB	x	n	n	n		Transf. byte from Accu 1 to the flag byte x
T	EW	x	n	n	n		Transf. word from Accu 1 to the input word x
T	AW	x	n	n	n		Transf. word from Accu 1 to the output word x
T	MW	x	n	n	n		Transf. word from Accu 1 to the flag word x
T	ED	x	n	n	n		Transf. D-word from Accu 1 to the input double word x
T	AD	x	n	n	n		Transf. D-word from Accu 1 to the output double word x
T	MD	x	n	n	n		Transf. D-word from Accu 1 to the flag double word x
T	PB	x	n	n	n		Transf. byte from Accu 1 to the periphery byte x
T	PW	x	n	n	n		Transf. word from Accu 1 to the periphery word x
T	PD	x	n	n	n		Transf. D-word from Accu 1 to the periphery double word x
T	DL	x	n	n	n		Transf. byte from Accu 1 to the left data byte x
T	DR		n	n	n		Transf. byte from Accu 1 to the right data byte x
T	DW	x	n	n	n		Transf. word from Accu 1 to the data word x
T	DD	x	n	n	n		Transf. D-word from Accu 1 to the data double word x

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

T EB x Transfer byte to the input byte x
T AB Transfer byte to the output byte x
T MB x Transfer byte to the flag byte x

Example:

```
:L      KD 16841217 ;corresponds 100FA01 H in Accu 1
:T      EB 0        ;The value standing in Accu 1 is written right justified
                    ;to the input byte 0
                    ; Result in EB 0 is 01 H = 1
:T      AB 1        ; Result in AB 1 is 01 H = 1
:T      MB 5        ; Result in MB 5 is 01 H = 1
```

T EW x Transfer word to the input word x
T AW x Transfer word to the output word x
T MW x Transfer word to the flag word x

Example:

```
:L      KD 16841217 ;corresponds to 100FA01 H in Accu 1
:T      EW 0        ;The value standing in Accu 1 is written right justified
                    ;to the input word 0
                    ; Result in EW 0 is FA01 H = 64001
:T      AW 2        ; Result in AW 2 is FA01 H = 64001
:T      MW 6        ; Result in MW 6 is FA01 H = 64001
```

T ED x Transfer double word to the input double word x
T AD x Transfer double word to the output double word x
T MD x Transfer double word to the flag double word x

Example:

```
:L      KD 16841217 ;corresponds to 100FA01 H in Accu 1
:T      ED 0        ;The value standing in Accu 1 is written
                    ;in the input double word 0.
                    ; Result in AD 4 is 100FA01 H = 16841217
:T      MD 8        ; Result in MD 8 is 100FA01 H = 16841217
```

T PB x Transfer byte directly to periphery byte x

Example:

```
:L      KD 16841217 ;corresponds to 100FA01 H in the Accu 1
:T      PB 0        ;The value standing in Accu 1 is written right justified
                    ;to the periphery byte 0.
                    ; Result in PB 0 is 01 H = 1
```

T PW x Transfer word directly to periphery word x

Example:

```
:L      KD 16841217 ;corresponds 100FA01 H in Accu 1
:T      PW 0        ;The value standing in Accu 1 is written
                    ;in the periphery word 0.
                    ; Result in PW 0 is FA01 H = 64.001
```

T PD x Transfer double word directly to periphery double word x

Example:

```
:L      KD 16841217 ;corresponds 100FA01 H in Accu 1
:T      PD 0        ;The value standing in Accu 1 is written right justified
                    ;to the periphery double word 0.
                    ; Result in PD 0 is 100FA01 H = 16841217
```

T DL x Transfer to the left byte (high-byte) of the data word x

Example:

```
:L      KB 100        ;Accu 1 = 100
:T      DL 0          ;High-byte data word 0 = 100
```

T DR x Transfer to the right byte (low-byte) of the data word x

Example:

```
:L      KB 100        ;Accu 1 = 100
:T      DR 0          ;Low-byte data word 0 = 100
```

T DW x Transfer in data word x

Example:

```
:L      KH 0064       ;Accu 1 = 64 H = 100
:T      DW 0          ;Data word 0 = 100
```

T DD x Transfer in data double word x

Example:

```
:L      ED 0          ;Input double word 0
:T      DD 0          ;Data double word 0 receives the contents of
                      ;input double word 0
```

3.2.3 Comparison functions

The comparison functions permit the comparison between two 32-bit fixed point numbers in the accumulators 1 and 2. The comparison is always performed independently of the RLO.

Comparison is always according to the following regulation:

Accumulator 2 Comparison operator Accumulator 1

The following comparison operators are allowed:

Comparison for equality	**!=FD**
Comparison for inequality	**><FD**
Comparison for greater	**>FD**
Comparison for greater or equal	**>=FD**
Comparison for less	**<FD**
Comparison for less or equal	**<=FD**

The accumulator contents are not changed by the functions. If the comparison is correct, the RLO is set to 1, otherwise to 0. In addition to the RLO, the display flags of the processor Anz0, Anz1 and Ov are set and can be evaluated by means of conditional branches (cf. Section 3.3.2).

Comparison result	Display flags		
	Anz0	Anz1	Ov
Accumulator 2 equal to accumulator 1	0	0	0
Accumulator 2 less than accumulator 1	1	0	0
Accumulator 2 greater than accumulator 1	0	1	0

The branch functions depending upon the display flags are described in Section 3.3.2.

The table provides an overview of the possible commands:

0	1	2	3	4	5
!=FD	n	y	n		<Accu 2> equal <Accu 1>
><FD	n	y	n		<Accu 2> unequal <Accu 1>
>FD	n	y	n		<Accu 2> greater <Accu 1>
>=FD	n	y	n		<Accu 2> greater equal <Accu 1>
<FD	n	y	n		<Accu 2> less <Accu 1>
<=FD	n	y	n		<Accu 2> less equal <Accu 1>

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

!=FD Comparison for "Equal"

Example:

```
:L      KB 1        ;Value should be in Accu 2
:L      KB 2        ;Value for Accu 1
:!=FD               ;Accu 2 = 1, Accu 1 = 2, comparison 1 != 2 ?
                    ;1 is not equal 2 -> RLO is 0
                    ;Anz0=1, Anz1=0, Ov=0
                    ;A conditional function can follow here
                    ;e.g. "=   M 0.0","SPB FB x"
```

><FD Comparison for "Unequal"

Example:

```
:L      KB 1        ;Value should be in Accu 2
:L      KB 2        ;Value for Accu 1
:><FD               ;Accu 2 = 1, Accu 1 = 2, comparison 1 >< 2 ?
                    ; 1 is unequal 2 -> RLO is 1
                    ;Anz0=1, Anz1=0, Ov=0
                    ;A conditional function can follow here
                    ;e.g. "=   M 0.0","SPB FB x"
```

>FD Comparison for "Greater"

Example:

```
:L      KB 2          ;Value should be in Accu 2
:L      KB 1          ;Value for Accu 1
:>FD                  ;Accu 2 = 2, Accu 1 = 1, comparison 2 > 1 ?
                      ; 2 is greater 1 -> RLO is 1
                      ;Anz0=0, Anz1=1, Ov=0
                      ;A conditional function can follow here
                      ;e.g. "=   M 0.0","SPB FB x"
```

>=FD Comparison for "Greater or Equal"

Example:

```
:L      KB 2          ;Value should be in Accu 2
:L      KB 2          ;Value for Accu 1
:>=FD                 ;Accu 2 = 2, Accu 1 = 2; comparison 2 >= 2 ?
                      ; 2 is greater equal 2 -> RLO is 1
                      ;Anz0=0, Anz1=0, Ov=0
                      ;A conditional function can follow here
                      ;e.g. "=   M 0.0","SPB FB x"
```

<FD Comparison for "Less"

Example:

```
:L      KB 1          ;Value should be in Accu 2
:L      KB 2          ;Value for Accu 1
:<FD                  ;Accu 2 = 1, Accu 1 = 2, comparison 1 < 2 ?
                      ; 1 is less 2 -> RLO is 1
                      ;Anz0=1, Anz1=0, Ov=0
                      ;A conditional function can follow here
                      ;e.g. "=   M 0.0","SPB FB x"
```

<=FD Comparison for "Less or Equal"

Example:

```
:L      KB 1          ;Value should be in Accu 2
:L      KB 2          ;Value for Accu 1
:<=FD                 ;Accu 2 = 1, Accu 1 = 2, comparison 1 <= 2 ?
                      ; 1 is less equal 2 -> RLO is 1
                      ;Anz0=1, Anz1=0, Ov=0
                      ;A conditional function can follow here
                      ;e.g. "=   M 0.0","SPB FB x"
```

3.2.4 Arithmetic functions

The four basic types of calculation are implemented in the processor of the PS with the "Arithmetic functions". They are performed independently of the RLO and do not influence this. The contents of the accumulators 1 and 2 are used for the arithmetic operations and operated on logically according to the following regulation:

Accumulator 2	**Arithmetic operation**	**Accumulator 1**
	Result	**-> Accumulator 1 (+,-,*,/)**
	Remainder in division	**-> Accumulator 2 (/)**

Before selecting the arithmetic function, the values of the operands to be operated on logically must be loaded in the accumulators (see Section 3.2.1).

The following arithmetic	+	(Addition),
operations are	-	(Subtraction),
possible:	*	(Multiplication)
	/	(Division).

The values of the accumulators are interpreted as dual-coded, signed, 32-bit fixed point numbers and operated on logically corresponding to the selected operation. The result then stands in accumulator 1.

In division the not rounded result (the quotient) is written in accumulator 1, the remainder of the division in accumulator 2. In the arithmetic functions (+,-,*) the accumulator 2 is not influenced.

Depending upon the result, the internal display flags Anz0, Anz1 and Ov of the processor, which can be used for program branches, are set (cf. Section 3.3.2).

Value of the result of the arithmetic operation	**Display flags**		
	Anz0	**Anz1**	**0V**
Result less than the allowed number range	0	1	1
Result is in the negative range	1	0	0
Result is zero	0	0	0
Result is in the positive range	0	1	0
Result exceeds the allowed number range	1	0	1

The branch functions depending upon the display flags are described in Section 3.3.2.

The following table provides an overview of the possible functions:

0	**1**	**2**	**3**	**4**	**5**
+FD	n	n	n		<Accu 1> = <Accu 2> + <Accu 1>
-FD	n	n	n		<Accu 1> = <Accu 2> - <Accu 1>
*FD	n	n	n		<Accu 1> = <Accu 2> * <Accu 1>
/FD	n	n	n		<Accu 1> = <Accu 2> / <Accu 1>
					<Accu 2> = Remainder from division

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

+FD Addition Accu 1 = Accu 2 + Accu 1

Example:

```
:L      KD 1000000000      ;Accu 1 in Accu 2, summand 2 in Accu 1
:L      KD 500000000       ;Summand 2 in Accu 2, summand 1 in Accu 1
:+FD
:T      DD 0               ;Data double word 0 has the value
                           ;1,500,000,000
                           ;Anz0=0, Anz1=1 and Ov=0
```

An overflow can be evaluated with SPO (cf. Section 3.3.2).

-FD Subtraction Accu 1 = Accu 2 - Accu 1

Example:

```
:L      KD 1000000000      ;Accu 1 in Accu 2, subtrahend in Accu 1
:L      KD -500000000      ;Minuend in Accu 1
:-FD
:T      DD 0               ;Data double word 0 has the value
                           ;1,500,000,000
                           ;Anz0=0, Anz1=1
```

An overflow can be evaluated with SPO (cf. Section 3.3.2).

*FD Multiplication Accu 1 = Accu 2 * Accu 1

Example:

```
:L      KD 2               ;Multiplicand in Accu 2
:L      KD -500000000      ;Multiplier in Accu 1
:*FD
:T      DD 0               ;Data double word 0 has the value
                           ; -1,000,000,000
                           ; Anz0=1, Anz1=0 and Ov=0
```

An overflow can be evaluated with SPO (cf. Section 3.3.2).

/FD Division Accu 1 = Accu 2 / Accu 1
Accu 2 = Remainder from the division

Example:

```
:L      KD 7        ;Dividend in Accu 2
:L      KD 2        ;Divisor in Accu 1
:/FD
:T      DD 0        ;Data double word 0 has the value 3 (quotient)
                    ;Accu 2 contains the remainder from the division
:TAK                ;Exchange Accu 1 and Accu 2 contents
:T      DD 2        ;Data double word 2 has the value 1
                    ;(remainder from division)
```

3.2.5 Digital logic operations

Accumulator 1 is influenced by the digital logic operations. The accumulator 1 is operated on logically with the accumulator 2 and the result assigned to accumulator 1. Accumulator 2 remains unchanged.

Accumulator 2	Digital logic operation	Accumulator 1 -> Accumulator 1

The logic operation is bit-wise. Three digital logic operations are possible:

AND
OR
EXCLUSIVE OR

Digital logic operations are independent of the RLO and do not influence this. However the display flags Anz0, Anz1 and Ov, which can be evaluated by means of branch functions, are set corresponding to the result (cf. Section 3.3.2).

Value of the result of the digital logic operation	Display flags		
	Anz0	Anz1	0V
Result is zero	0	0	0
Result is not zero	0	1	0

The dependent branch functions are described in Section 3.3.2

The following table provides an overview of the possible functions:

0	1	2	3	4	5
UD	n	n	n		AND operation of Accu 1 and 2
OD	n	n	n		OR operation of Accu 1 and 2
XOD	n	n	n		Excl.-OR operation of Accu 1 and 2

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

UD Bit-wise AND operation of the accumulators 1 and 2

Example:

```
:L      DD 4        ;Load data double word 4 (e.g. value = 25)
:L      KD 22       ;Accu 1 = 22, Accu 2 = 25
:UD                 ;Result is 16
```

Before executing the instruction UD

Accumulator 1 = 22

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 0 1 1 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 25

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 1 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

After executing the instruction UD

Accumulator 1 = 16

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 25

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 1 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

OD Bit-wise OR operation of the accumulators 1 and 2

Example:

```
:L      DD 4        ;Load data double word 4 (e.g. value = 25)
:L      KD 22       ;Accu 1 = 22, Accu 2 = 25
:OD                 ;Result is 31
```

Before executing the instruction OD

Accumulator 1 = 22

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 0 1 1 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 25

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 1 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

After executing the instruction OD

Accumulator 1 = 31

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 1 1 1 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 25

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 1 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

XOD Bit-wise exclusive-OR operation of the accumulators 1 and 2

Example:

```
:L      DD 4        ;Load data double word 4 (e.g. value = 25)
:L      KD 22       ;Accu 1 = 22, Accu 2 = 25
:XOD                ;Result is 15
```

Before executing the instruction XOD

Accumulator 1 = 22

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 0 1 1 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 25

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 1 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

After executing the instruction XOD

Accumulator 1 = 15

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 1 1 1 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 2 = 25

24	16	8	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 1 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

3.3 Organizational operations

3.3.1 Block functions

The block functions include:

- Absolute and conditional block selections (for organization blocks (OB), program blocks (PB) and function blocks (FB))
- Block end functions (absolute or conditional)
- The selection (activation) and the generation of data blocks (DB)

The absolute functions are executed independently of the RLO. The conditional functions are processed only for RLO = 1. With a block call the current program block is interrupted and the addressed block inserted in the processing. This is processed until its block end is reached and then the calling program is continued. A block call is also designated as "Branch with return intention". Since the processor notes internally the calling place in the block call, the program branches to the AWL command which follows the calling command on recognition of the next BE, BEA or BEB command. Each block must be concluded with a block end (AWL command BE, BEA or BEB). This command causes the ending of the current program as well as the return into the calling program. The previously valid data block is then activated again there. "BE", "BEB" or "BEA" command must be present in each program block (irrespective of whether OB, PB or FB). It is recommended that the instruction "BE" is programmed as last instruction in each block.

The function "A   DB x" is used for setting the current data block. x is here the number of the wanted data block. The processor processes for data loading or data transfer functions always only the current data block. The user must ensure that the current data block is set in the processor before he manipulates data. The data block 5 is now held as current data block by "A   DB 5". Data blocks can be generated both by the programming system and from the program. The function "E   DB x" is available for this. This function is used for creating and for deleting a data block.

During the processing of "E   DB x" the contents of accumulator 1 are taken into account. If accumulator 1 has the contents 0, then the data block x is deleted. If its contents are unequal to 0, then the data block x is created. The accumulator 1 determines the number of the data words which are reserved for the created data block. A generated data block is not automatically the current data block. Should there be access to it, then it must be called (selected) by "A   DB x". If it is attempted to delete a not created data block, to delete a data block created in APROS, or to create anew an already created DB, then this instruction is ignored without generating an error message.

As from software level AZ-PS4 V02.08 it is possible to generate data blocks remanently (battery buffered), by transferring the number of the data words as negative value into accumulator 1.

Caution: With DBs already not created remanently (generated with E DB x and "Accumulator 1 contents > 0" or through APROS) the command E DB x is not ignored if they are in addition generated remanently ("Accumulator 1 contents < 0"). Rather the DB becomes remanent in its already existing length, whereby the value of the data words is retained. (In this case only the sign of the "Accumulator 1 contents" is of significance, but not the amount!)

Data blocks already remanently created (generated with E DB x and "Accumulator 1 contents < 0") are **not** replaced by non-remanent DBs also delivered by APROS!

The following table provides an overview of the possible functions:

0			1	2	3	4	5
SPA	OB	x	n	n	y		Unconditional branch to organization block
SPA	PB	x	n	n	y		Unconditional branch to program block
SPA	FB	x	n	n	y		Unconditional branch to function block
SPB	OB	x	y	y	y		Conditional branch to organization block
SPB	PB	x	y	y	y		Conditional branch to program block
SPB	FB	x	y	y	y		Conditional branch to function block
BE			n	n	y		Block end
BEA			n	n	y		Unconditional return
BEB			y	y	y		Conditional return (for RLO=1)
A	DB	x	n	n	n		Call of a data block
E	DB	x	n	n	n		Generating/deleting a data block

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

SPA OB x Unconditional organization block call
SPA PB x Unconditional program block call
SPA FB x Unconditional function block call

When calling blocks the processor makes in principle no difference between OB, PB and FB. The blocks called in a program must be created and contain at least the command "BE". The block calls "SPA..." are always executed independently of the contents of the RLO.

Example:

```
:SPA   PB 1          ;Unconditional program block call
:SPA   OB 21         ;Unconditional organization block call
:SPA   FB 200        ;Unconditional function block call
```

SPB OB x Conditional organization block call
SPB PB x Conditional program block call
SPB FB x Conditional function block call

The block calls "SPB..." are executed only if the RLO contains the value 1. The RLO is taken along in the selected block. A binary logic operation following there is considered as first input bit scan. If the RLO is 0 at the block call, then the block call is not executed. The RLO and the first input bit scan are set to 1 and the following command is processed.

Example:

```
:U     E 0.1          ;Limit switch position x
:SPB   PB 1           ;Drive off
:UN    E 0.1          ;End position not yet reached
:SPB   PB 2           ;Drive on
:BE                   ;Block end
                      ;at place SPB PB x, SPB OB x or
                      ;SPB FB x can also stand.
```

BE Block end
BEA Unconditional block end
BEB Conditional block end

The block end functions can be subdivided like the block calls into absolute and conditional functions. BEA is an absolute command, which is executed independently of the contents of the RLO. BEB on the other hand is executed only if the RLO has the value 1. A following binary logic operation command in the user program is interpreted as first input bit scan. If the RLO is 0 at the block call, then the block call is not executed. The RLO and the first input bit scan are set to 1 and the following command processed. BE is processed like BEA. In contrast to BEA, which can be reached by means of going to different program branches of a block, BE always stands in the last instruction of a block.

Example:

```
          :UN    E       0.0    ;Driving switch to "HALT"
          :BEB                  ; yes
                 ; no
          :S     A       0.0    ;Signal light on
          :UN    E       1.7    ;Preliminary switching value not reached
          :SPB   = MARKE        ;
          :L     KD      1000   ;Command value for slow running
          :BEA
MARKE     :L     KD 20000              ;Command value for fast running
          :BE                          ;Also possible BEA
```

A DB x Activating a data block

Example:

```
:A      DB 3          ;Switching over to data block 3
                      ;The DB 3 must be allowed and available,
                      ;otherwise error message
```

E DB x Generating/deleting a data block

Example:

```
:L      KB 0
:E      DB 1          ;DB 1 is deleted if present and not
                      ;created by means of APROS in
                      ;the program storage area
:L      KB 20
:E      DB 1          ;DB 1 is created in the memory
                      ;with 20 data words
:A      DB 1          ;Switching over to DB 1
```

3.3.2 Branch functions

A distinction is made between unconditional and conditional branch functions. The unconditional branch "SPA = x" is always executed. All other branch functions are conditional (dependent) branches. They are used for branching within a program. If the branch condition is fulfilled, the normal processing sequence is interrupted and the program is continued at the stated entry point (branch address). The (AWL) line numbers (1-255) or symbolic marks are allowed as entrance points. If work is done with absolute line numbers, it must be observed that pure comment lines or blank lines may not be counted. The symbolic mark names may be up to 6 characters in length. It must be ensured that the called mark is defined in the program in the form "MARK1:U E 0.0 ;Comment". Marks must stand in the so-called mark field before the colon of an AWL line (see documentation: PS programming and test system APROS). If the branch condition is not fulfilled, the branch instruction is ignored and the program is continued with processing the next instruction. The execution of conditional branches depends upon the processor flags Anz0, Anz1, 0V or upon the RLO. The following table states the conditions under which conditional branches are executed depending upon the processor flags:

The following branches are performed	**Set displays**		
	Anz0	**Anz1**	**0V**
SPN =, SPP =, SP0 =	0	1	1
SPN =, SPM =	1	0	0
SPZ =	0	0	0
SPN =, SPP =	0	1	0
SPN =, SPM =, SP0 =	1	0	1

The branches named here are independent upon the contents of the RLO and are influenced only by the above named flags. The flags are set after performing the following functions:

- Arithmetic functions,
- Comparison functions,
- Shift functions and
- Conversion functions.

The branch "SPB =" is influenced only by the contents of the RLO. If the RLO=1, the branch is executed, with RLO=0 not. The flags Anz0, Anz1 and Ov have no influence.

The following table provides an overview of the possible functions:

0	1	2	3	4	5
SPA = x	n	n	n		Unconditional branch
SPB = x	y	y	y		Branch at RLO = 1
SPZ = x	n	n	n		Branch at (ANZ1 = 0) & (ANZ0 = 0)
SPN = x	n	n	n		Branch at (ANZ1) unequal (ANZ0)
SPP = x	n	n	n		Branch at (ANZ1 = 1) & (ANZ0 = 0)
SPM = x	n	n	n		Branch at (ANZ1 = 0) & (ANZ0 = 1)
SPO = x	n	n	n		Branch at OV

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

Remarks: At least one separator (blank, tabulator) must be provided both between the operation part and the equal sign and between the equal sign and the operand (branch destination)!

SPA = Unconditional branch

Branches "unconditionally" (always) to branch mark.

Example:

```
            :UN    E  0.0
            :S     A  0.0
            :UN    E  1.7
            :SPA   = MARKE
M1          :L     KD 1000
            :BEA
MARKE       :L     KD 20000
            :SPA   = M1
            :BE
```

SPB = Conditional branch

Branches only to branch mark if RLO = 1. Continuation of the program at RLO = 0.

Example:

```
            :U     E  0.0
            :SPB   = RECHTS
            :L     KD -1000   ;counterclockwise
            :BEA
RECHTS      :L     KD  1000   ;clockwise
            :BE
```

SPZ = Branch at "Zero"

Branches only to branch mark if Anz0 and Anz1 = 0, otherwise continuation of the program.

Example:

```
      :L     KD  100
      :L     ED  0
      :-FD                 ;Result of 100 - ED 0
      :SPZ   = MAR         ;No control deviation
      :SPA PB 1            ;Readjust
MAR   :BEA
```

SPN = Branch at "not zero"

Branches only to branch mark if Anz0 is unequal to Anz1, otherwise continuation of the program.

Example:

```
        :L      KD  100
        :L      ED  0
        :-FD                    ;Result of 100 - ED 0
        :SPN     = MAR          ;Control deviation
        :BEA
MAR     :SPA PB 1               ;Readjust
```

SPP = Branch at sign "Plus"

Branches only to branch mark if Anz0=0 and Anz1=1, otherwise continuation of the program.

Example:

```
        :L      MD  0
        :L      KD  0
        :><FD                   ;Flag double word unequal 0
        :SPP    = M1            ;MD 0 is greater than 0
        :SPA    PB  1           ;Heating
        :BEA
M1      :SPA    PB  2           ;Cooling
        :BE
```

SPM = Branch at sign "Minus"

Branches only to mark if Anz0=1 and Anz1=0. Otherwise continuation of the program.

Example:

```
                :L      ED  10          ;Actual position in accumulator 1
                :KZD                    ;corresponds to mult. with -1
                :SPM    = RECHTS        ;Result is negative
                :SPA    PB 1            ;by Accu1 value (forwards)
                :BEA
RECHTS          :KZD
                :SPA    PB 2            ;by Accu1 value (backwards)
                :BE
```

SPO = Branch at "Overflow"

Branches only to branch mark if Ov=1, otherwise continuation of the program.

Example:

```
                :L   ED 0
                :L   ED 2
                :+FD                    ;Ov is set if the value exceeds
                                        ;the permitted range
                :SPO    = FALSCH        ;No output of the wrong value
                :T      AD  0           ;Output of the correct result
                                        ;on the output card
FALSCH          :BE
```

3.3.3 Shift functions

The shift functions enable the contents of accumulator 1 to be shifted bit-wise to the right or left. Values from 0 to 31 are allowed as parameters. At the same time it is possible to evaluate the value of the last shifted out bit and depending upon this to execute conditional branch functions. The RLO is not influenced by the shift functions.

Influencing the display flags Anz0, Anz1 and Ov is apparent from the following table.

Value of the last shifted bit	Display flags		
	Anz0	**Anz1**	**0V**
Bit = 1	0	1	0
Bit = 0	0	0	0

Shifting by 0 bit to the left does not change the accumulator 1. However, after execution of this command the flags are set according to the above table depending upon bit 0 in accumulator 1. On shifting by 0 bit to the right the accumulator 1 is also not changed. However, after execution of this command the flags are set (see above) depending upon bit 31 in accumulator 1. Zeroes are pushed into the accumulator 1 both when shifting to the left and to the right.

The following table provides an overview of the possible functions:

0		**1**	**2**	**3**	**4**	**5**
SLD	x	n	n	n		Shift Accu 1 x bit to the left
SRD	x	n	n	n		Shift Accu 1 x bit to the right with: x = 0..31

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

SLD x Shifting to the left

Example:

```
        :L      KD -2129587951
        :SLD    1                       ;Shifting by 1 bit to the left
        :SPZ    = MAR                   ;if bit 31 = 0 -> branch
        :  ...
        :  ...
        :  ...
MAR     :  ...
        :BE
```

Accumulator 1 before the function 81111111 H = - 2129587951

24	16	8	0
1 0 0 0 0 0 0 1	0 0 0 1 0 0 0 1	0 0 0 1 0 0 0 1	0 0 0 1 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator 1 after the function 02222222 H = 35791394

24	16	8	0
0 0 0 0 0 0 1 0	0 0 1 0 0 0 1 0	0 0 1 0 0 0 1 0	0 0 1 0 0 0 1 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

At the same time the flags are set as follows:

Anz0 = 0 Anz1 = 1 Ov = 0

SRD x Shifting to the right

Example:

```
        :L      KD -2129587951
        :SRD    1                       ;Shifting by 1 bit to the right
        :SPN    = MAR                   ;if bit 0 = 1 -> branch
        :  ...
        :  ...
        :  ...
MAR     :  ...
        :BE
```

Accumulator 1 before the function 81111111 H = -2129587951

24	16	8	0
1 0 0 0 0 0 0 1	0 0 0 1 0 0 0 1	0 0 0 1 0 0 0 1	0 0 0 1 0 0 0 1
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

Accumulator after the function 40888888 H = 1082689672

24	16	8	0
0 1 0 0 0 0 0 0	1 0 0 0 1 0 0 0	1 0 0 0 1 0 0 0	1 0 0 0 1 0 0 0
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	

At the same time the flags are set as follows:

Anz0 = 0 Anz1 = 1 Ov = 0

3.3.4 Conversion functions

The value standing in accumulator is converted with the conversion functions. The RLO is not influenced by these functions. They are executed independently of the RLO or other flags. The complement formation as well as the conversion of the numerical display belong to this group.

The following table provides an overview of the possible functions:

0	1	2	3	4	5
KED	n	n	n		One's complement of Accu1
KZD	n	n	n		Two's complement of Accu1
WDL	n	n	n		Converting Accu1 as BCD number into a dual number
WDZ	n	n	n		Converting Accu1 as dual number into a BCD number
WBD	n	n	n		Converting Accu1 as 8-bit integer into 32-bit integer
WWD	n	n	n		Converting Accu1 as 16-bit integer into 32-bit integer

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

KED One's complement formation

When forming the one's complement the accumulator 1 is negated bit-wise: This means ones are replaced by zeroes and zeroes by ones.

Example:
```
:L      KH  5555      ;corresponds to 5555 H
:KED
:T      DD  0         ;FFFFAAAA H is written into the data double word 0
                      ;of the current data block
:BE
```

KZD Two's complement formation

When forming the two's complement the accumulator 1 is negated bit-wise: Ones are replaced by zeroes and zeroes by ones. 1 is then added. This function corresponds to multiplication with -1. KZD influences as sole function of this group the flags (Anz0, Anz1, Ov) according to the following scheme:

Value of the result of the arithmetic operation	Display flags		
	Anz0	Anz1	0V
Result exceeds the allowed numerical range	0	0	1
Result lies in the negative range	1	0	0
Result is zero	0	0	0
Result lies in the positive range	0	1	0

After command execution the result can be evaluated by means of conditional branches (cf. Section 3.3.2).

Example:
```
:L      KD  2000      ;corresponds to 07D0 H
:KZD
:T      AD  0         ;FFFFF830 H is given at output double word 0
                      ;corresponds to -2000
:BE
```

WDL Convert BCD value into dual value

With this function a value standing in accumulator 1 is interpreted as BCD value (binary coded decimal value) and converted into a dual value. This can now be processed further by the processor, e.g. to transfer the BCD value present at an input card as preset values for a counter.

Example:
```
: .
: .
:L      ED  0         ;e.g. 1001100110011001 = 9999
:WDL                  ;9999 is converted to 270F H
:S      Z   0         ;Counter 0 is set with 9999
: .
: .
:BE
```

WDZ Convert dual value into BCD value

With this function a value standing in accumulator 1 is interpreted as dual value and converted into a BCD value (binary coded decimal value). This can be now assigned directly to an output module, which then activates a BCD display, for instance.

Example:

```
: .
:L      KD  2
:L      KD  12
:*FD                ;Value in Accu1 = 18 H = 24
:WDZ                ;Value in Accu1 = 24 BCD
:T      AD  0
: .
: .
: .
:BE
```

WBD Convert 8-bit integer value into 32-bit integer value

With this function a value standing in accumulator 1 is interpreted as 8-bit integer value and converted into a 32-bit integer value. The display flags Anz0, Anz1 and Ov are set depending upon the 32-bit integer value (with Ov=0: no overflow). The command is available as from version AZ-PS4 V02.09 and APROS V02.07.

Example:

```
: .
:L      KB  0       ;Value in Accu1 = 00000000 H = 0 Dec
:WBD                ;Value in Accu1 = 00000000 H = 0 Dec
:  .                ;Anz0=0, Anz1=0, Ov=0 (result is zero)
: .
:L      KB  128     ;Value in Accu1 = 00000080 H = 128 Dec
:WBD                ;Value in Accu1 = FFFFFF80 H = -128 Dec
:  .                ;Anz0=1, Anz1=0, Ov=0 (result lies in the negative range)
: .
:BE
```

WWD Convert 16-bit integer value into 32-bit integer value

With this function a value standing in accumulator 1 is interpreted as 16-bit integer value and converted into a 32-bit integer value. The display flags Anz0, Anz1 and Ov are set depending upon the 32-bit integer value (with Ov=0: no overflow). The command is available as from version AZ-PS4 V02.09 and APROS V02.07.

Example:

```
: .
:L      KH  0080    ;Value in Accu1 = 00000080 H = 128 Dec
:WWD                ;Value in Accu1 = 00000080 H = 128 Dec
:  .                ;Anz0=0, Anz1=1, Ov=0 (result lies in the positive range)
: .
:L      KH  8000    ;Value in Accu1 = 00008000 H = 32768 Dec
:WWD                ;Value in Accu1 = FFFF8000 H = -32768 Dec
:  .                ;Anz0=1, Anz1=0, Ov=0 (result lies in the negative range)
: .
:BE
```

3.3.5 Processing functions

Processing functions allow "indirect" addressing. The parameter of an operation (cf. Section 1.1) is formed in the AWL program by the current value of a settable flag or data word.

The following table provides an overview of the possible functions:

0	1	2	3	4	5
B MW x	n	n	n		Use flag word x for indirect addressing (with x = 0..254)
B DW x	n	n	n		Use data word x of the current DB for indirect addressing (with x = 0..255)

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

Processing commands always consist of 2 instructions. The processing command "B MW x" / "B DW x" is the 1st command. It determines the flag word/data word address, which contains the datum for forming the parameter value of the subsequent instruction. The required instruction stands in the 2nd command. The effective parameter for this operation is formed from the parameter of the 2nd command plus (addition) the current contents (numerical value) of the flag word/data word address of the 1st command.

Note: In indirectly addressed loading and transfer commands, the additive parameter value, which arises from the 1st command, must be interpreted as byte, word or double word value depending upon the operand identifier of the 2nd command (see explanation below).

B MW x Process indirectly with flag word x

This function determines the parameter value of the following instruction by addition of the numerical value in the flag word MW x.

Example:

```
       : .
       :L   KB  1
       :T   MW  10            ;Value of MW10 = 1
       : .
       : .
       :B   MW  10            ;MW10 determined for ind. processing
       :SPA = SpTab           ;Branch to SpTab+1
SpTab  :SPA = Mark0           ;Branch distributor table SpTab+0
                              ;Branch to mark0
       :SPA = Mark1           ;Branch distributor table SpTab+1
                              ;Branch to mark1
       : .                    ;...

Mark0  : .
       : .

Mark1  : .                    ;Branch destination
       : .

       :BE
```

B DW x Process indirectly with data word x

This function determines the parameter value of the following instruction by addition of the numerical value from data word DW x of the active data block.

Example:

```
:       .
:A      DB  8          ;DB 8 is already created and is
                       ;activated
:L      KB  3
:T      DW  0          ;Value of DB8, DW0 = 3
:       .
:       .
:L      KB  5          ;Accu1 = 5

:B      DW  0          ;DW0 determined for ind. processing, DB8
                       ;still active
:T      MB  8          ;Transfer Accu1 to MB11 (8 + (1*3))
:       .
:B      DW  0          ;DW0 determined for ind. processing, DB8
                       ;still active
:T      AW  0          ;Transfer Accu1 to AW6 (0 + (2*3))
:       .
:B      DW  0          ;DW0 determined for ind. processing, DB8
                       ;still active
:T      MD 20          ;Transfer Accu1 to MD32 (20 + (4*3))
:       .

:B      DW  0          ;DW0 determined for ind. processing, DB8
                       ;still active
:T      DR 10          ;Transfer Accu1 to DB8, DR13 (10 + (1*3))
:       .
:B      DW  0          ;DW0 determined for ind. processing, DB8
                       ;still active
:T      DD 20          ;Transfer Accu1 to DB8, DD26 (20 + (2*3))
:       .
:       .
:BE
```

In connection with the processing function, currently exclusively the following instructions are executed indirectly:

– Digital operations

Operation part	Operand identifier
L	EB, EW, ED; AB, AW, AD; MB, MW, MD [1] DR, DL, DW, DD 2) T Z
LC	T Z
T	EB, EW, ED; AB, AW, AD; MB, MW, MD [1] DL, DR, DW, DD [2]

1) Loading and transfer commands with E/A/M operands are based on a byte address. On indirect addressing the additive component of the parameter value (from the flag or data word which was identified by the processing command) is however weighted depending upon the operand identifier

– x 1 (in byte operands),
– x 2 (in word operands),
– x 4 (in double word operands)

(see above example).

2) Loading and transfer commands with D operands are based on a word address. On indirect addressing the additive component of the parameter value (from the flag or data word which was identified by the processing command) is however weighted depending upon the operand identifier

– x 1 (in word operands),
– x 2 (in double word operands)

It must be noted that DR and DL operands also refer to a word address
(see above example).

– Organizational operations

Operation part	Operand identifier
SPA	OB, PB, FB
SPB	OB, PB, FB
SPA	=
SPB	=
A	DB
E	DB

3.3.6 Other organizational functions

This function group comprises, apart from the TAK instruction (exchange accumulator contents), the so-called zero commands, which influence no operand areas of the PS and no registers.

As from APROS V02.10 and AZ-PS4 V02.10 the instructions TAB, TAW and TAD are also supported. These instructions facilitate rotating the Accu1 byte (cf. examples below).

NOP0, NOP1, "Blank line" and *** can be used to separate program parts optically from one another or to keep space free for "later commands". However, NOP0 and NOP1 consume processing time in the processor.

The commands STP and STS also influence no operand area and no registers/flags of the processor. However, they have wide reaching significance for the entire user program. When processing the command STP the processor "notes" this, still processes the cyclic organization block (OB 1) up to the end, sends the process image of the outputs and then goes into the PS stop status. This status of the PS can be reset only by RESET, HW-RESET or POWER-ON. The command STS has the same effect, only that the OB 1 is not firstly processed to the end. On recognizing this command, the processor immediately leaves the running program and goes into the STOP status.

The following table provides an overview of the possible functions:

0	1	2	3	4	5
TAK	n	n	n		Exchange Accu1 and Accu2 contents
TAB	n	n	n		Exchange Accu1: LwLb<->LwHb HwLb<->HwHb
TAW	n	n	n		Exchange Accu1: Lw<->Hw
TAD	n	n	n		Exchange Accu1: LwLb<->HwHb LwHb<->HwLb
NOP0	n	n	n		Zero operation
NOP1	n	n	n		Zero operation
	n	n	n		Blank line
***	n	n	n		Network end
STP	n	n	n		STOP after cycle end
STS	n	n	n		STOP immediately
Lw = Low Word Lb = Low byte Hw = High Word Hb = High byte					

0 = Operation, operands(s)
1 = Operation depends on the RLO
2 = Operation influences the RLO
3 = Operation limits the validity of the RLO
4 =
5 = Comment

TAK Exchange accumulator contents

Example:

```
:L      KD 17
:L      KD 3
:/FD
:T      DD 0          ;Transfer quotient = 5 from Accu1 to
                      ;data double word 0
:TAK                  ;Exchange accumulator contents 1 and 2
:T      DD 2          ;Transfer remainder = 2 from Accu1 to data double word 2
:BE
```

TAB Exchange accumulator1 contents (Low Word: Low byte with High byte; High Word: Low byte with High byte)

Example:

```
:L      KH 0403       ;Accu1 = 00000403
:SLD    16            ;Accu1 = 04030000
:L      KH 0201
:OD                   ;Accu1 = 04030201

:TAB                  ;Accu1 = 03040102

:BE
```

TAW Exchange accumulator1 contents (Low Word with High Word)

Example:

```
:L      KH 0403       ;Accu1 = 00000403
:SLD    16            ;Accu1 = 04030000
:L      KH 0201
:OD                   ;Accu1 = 04030201

:TAW                  ;Accu1 = 02010403

:BE
```

TAD Exchange accumulator1 contents ("Low Word, Low byte" with "High Word, High byte"; "Low Word, High byte" with "High Word, Low byte")

Example:

```
:L      KH 0403       ;Accu1 = 00000403
:SLD    16            ;Accu1 = 04030000
:L      KH 0201
:OD                   ;Accu1 = 04030201

:TAD                  ;Accu1 = 01020304

:BE
```

NOP0 Zero operation 0

Example:

```
        :L      ED 0
        :SRD   1            ;Shift by 1 bit to the left
        :SPN = MAR          ;If bit 0 was = 1 -> branch
        :NOP0
        :NOP0
        :BEA
MAR     :STP                ;If bit 0 was set in EB 0, the running
                            ;cycle is ended and no new cycle
                            ;started
        :BE
```

NOP1 Zero operation 1

Example:

```
        :L      ED 0
        :SRD    1           ;Shift by 1 bit to the left
        :SPN = MAR          ;If bit 0 was = 1 -> branch
        :NOP1
        :NOP1
        :BEA
MAR     :STP                ;If bit 0 was set in EB 0, the running
                            ;cycle is ended and no new cycle
                            ;started
        :BE
```

STP Stop after cycle end

Example:

```
        :L      ED 0
        :SRD   1            ;Shift by 1 bit to the left
        :SPN = MAR          ;If bit 0 = 1 -> branch
        :
        :
        :BEA
MAR     :STP                ;If bit 0 was set in EB 0, the running
                            ;cycle is ended and no new cycle
                            ;started
        :BE
```

STS Stop immediately

Example:

```
        :L      KD  100
        :L      ED  0
        :-FD                ;Result of 100 - ED 0
        :SPN   = MAR        ;Control deviation
        :BEA
MAR     :STS                ;On deviation of the input
                            ;double word 0 from 100 the
                            ;running program is interrupted immediately
```

Blank line

Example:

```
        :L      KD  100
        :L      ED  0
        :-FD                    ;Result of 100 - ED 0
        :SPN = MAR              ;Control deviation
        :BEA
        :
        :
MAR     :STS                    ;On deviation of the input
                                ;double word 0 from 100 the
                                ;running program is interrupted immediately
```

***   Network end

Example:

```
        :***
        :L      KD  100
        :L      ED  0
        :-FD                    ;Result of 100 - ED 0
        :SPN = MAR              ;Control deviation
        :BEA
        :
MAR     :STS                    ;On deviation of the input double word 0 from
                                ;100 the running program is interrupted immediately
        :***
```

4 Program structuring, block assignment

4.1 Program structuring, block assignment

The PS application program results from all AWL instructions programmed by the user. To achieve a good overview, the program is divided into individual sections. Function groups are formed and combined into "blocks". A distinction is made between block types. Type and functions of the blocks are described below.

4.1.1 Organization blocks (OBs)

Organization blocks are a type of interface between operating system of the PS and user program. OB1, OB17, OB18, OB19, OB20, OB21 and OB22 are implemented. OB17 to OB22 can be used optionally, OB1 must be present in every user program.

The following table provides an overview of the currently implemented OBs. Refer to the flow / status charts at the end of this section for the detailed interplay of the different OBs:

OB	Function
OB01	Cyclic OB
OB17	Error OB (group: PS system) FATAL_ERROR_OPT_MODUL [1)]
OB18	Error OB (group: PS user) ERROR_OPT_MODUL [1)]
OB19	Error OB (group: PS warning) WARNING_OPT_MODUL [1)]
OB20	Error OB (group: drive) [1)]
OB21	Start OB (AWL start)
OB22	Start OB (PS reset)

[1)] cf. Section 6.5.2, EW 218: Error bit bar

Cyclic OB01:

The organization block OB01 must be programmed by the user. Processing this OB starts after previous, one-time program run-through either of the OB22 or of the OB21. If these two blocks do not exist, the PS program starts with OB01 on switching on the operating voltage (power-on reset for real time run), after reset for "Test run" (triggering by SBUS command "PS reset") and after AWL start for "Test run" (triggering by SBUS command "AWL start"). The OB01 is processed cyclically. A cycle comprises acceptance of the process image of the inputs (PAE) before OB01 processing and transfer of the process image PAA to the outputs after OB01 processing. The OB01 contains the AWL total program (apart for presettings and initializations within the scope of OB21/OB22 as well as the error handling routines OB17/OB18/OB19/OB20), whereby program calls of other blocks can be programmed in all OBs.

Error OB17 ("PS system" group)
Error OB18 ("PS user" group)
Error OB19 ("PS warning" group)

These blocks are error OBs. They can be programmed by the user. The associated error OB is selected automatically depending upon the error type (error group, see documentation "PS error description"). The error OB contains the user program, which determines how the system should behave in the case of an error. A separate sequence can be programmed for each error group. Errors from the "PS system" and "PS user" groups generate in the PS the "ERROR" status even if OB17/OB18 are not used. The PS "ERROR" status must be deleted by "RESET". This is done by "Delete error" in the AMKASYN basic system or by power OFF/ON or through the programming unit. On a warning message ("PS warning" group) OB19 is activated automatically. Should the "ERROR" status also be generated in the PS with a warning, OB17 must be selected in the OB 19 by branch instruction.

Error OB20 ("Drive" group)

The OB20 is selected on errors in the AMKASYN basic system (cf. AMKASYN: Digital pulse converter in modular construction; description). If OB20 is not created, then the "ERROR" status is generated in the PS in the case of errors in the "Drive" group. If the OB20 is created, then the user can determine by his AWL program (e.g. by evaluating the group ready message in E 220.0) whether the system should transfer to the PS "ERROR" status. For this the error OB17 must be selected in the OB20. Without this selection the cyclic OB01 is activated anew after processing the OB20.

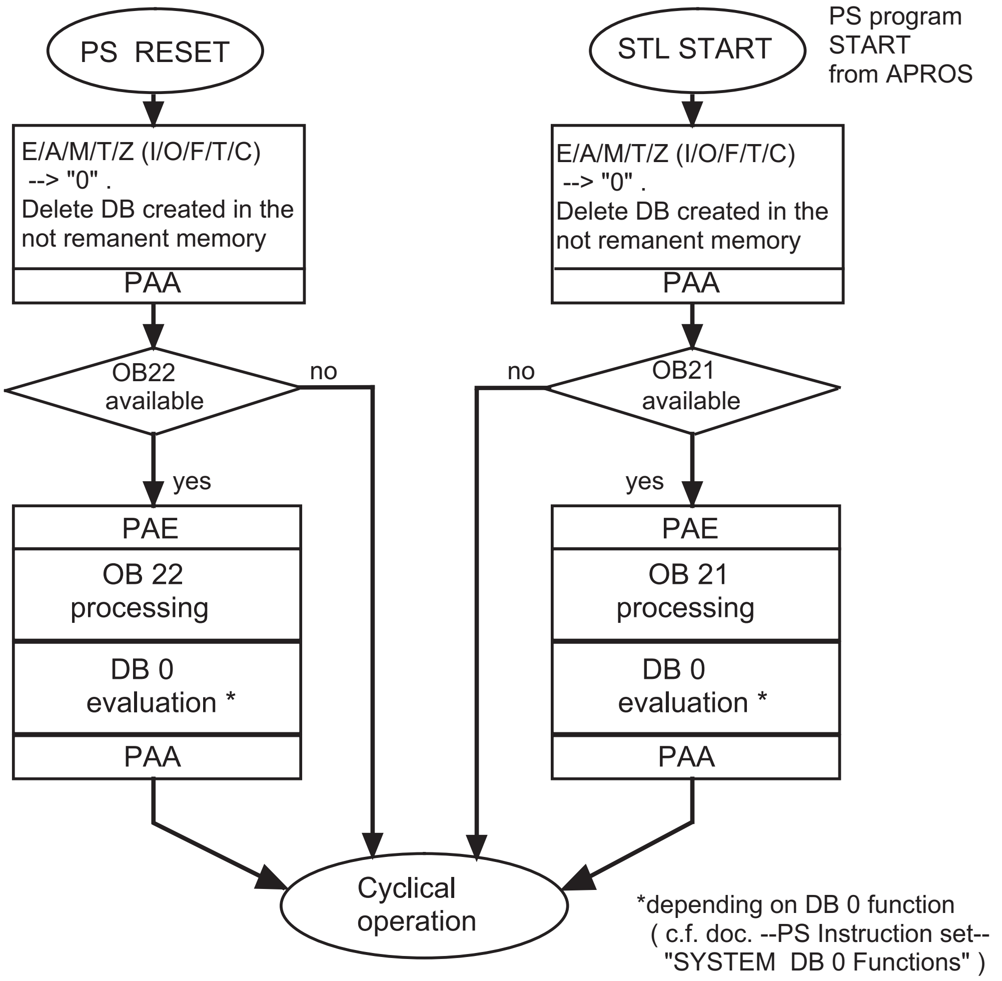
Start OB21:

On AWL start in the test run (triggering by means of PG by SBUS command "AWL start") the AWL program starts with this block. The process image of the inputs (PAE9) is still taken over previously. The OB21 is run through only once, so that one-time presettings or initializations in it can be implemented. After the program run-through of the OB21, the process image PAA is transferred to the outputs. Then OB01 determines the further processing of the AWL program.

Start OB22:

It determines the program start on switching on the operating voltage (power-on reset for real time running) and for "Reset for test run" (triggering by means of PG by SBUS command "PS reset"). Its AWL program must be written by the user. It is run through only once and can arrange for one-time presettings or initializations. The process image of the inputs (PAE) is taken over before processing the OB22. After processing the process image PAA is transferred to the outputs. OB01 then determines the further processing of the AWL program.

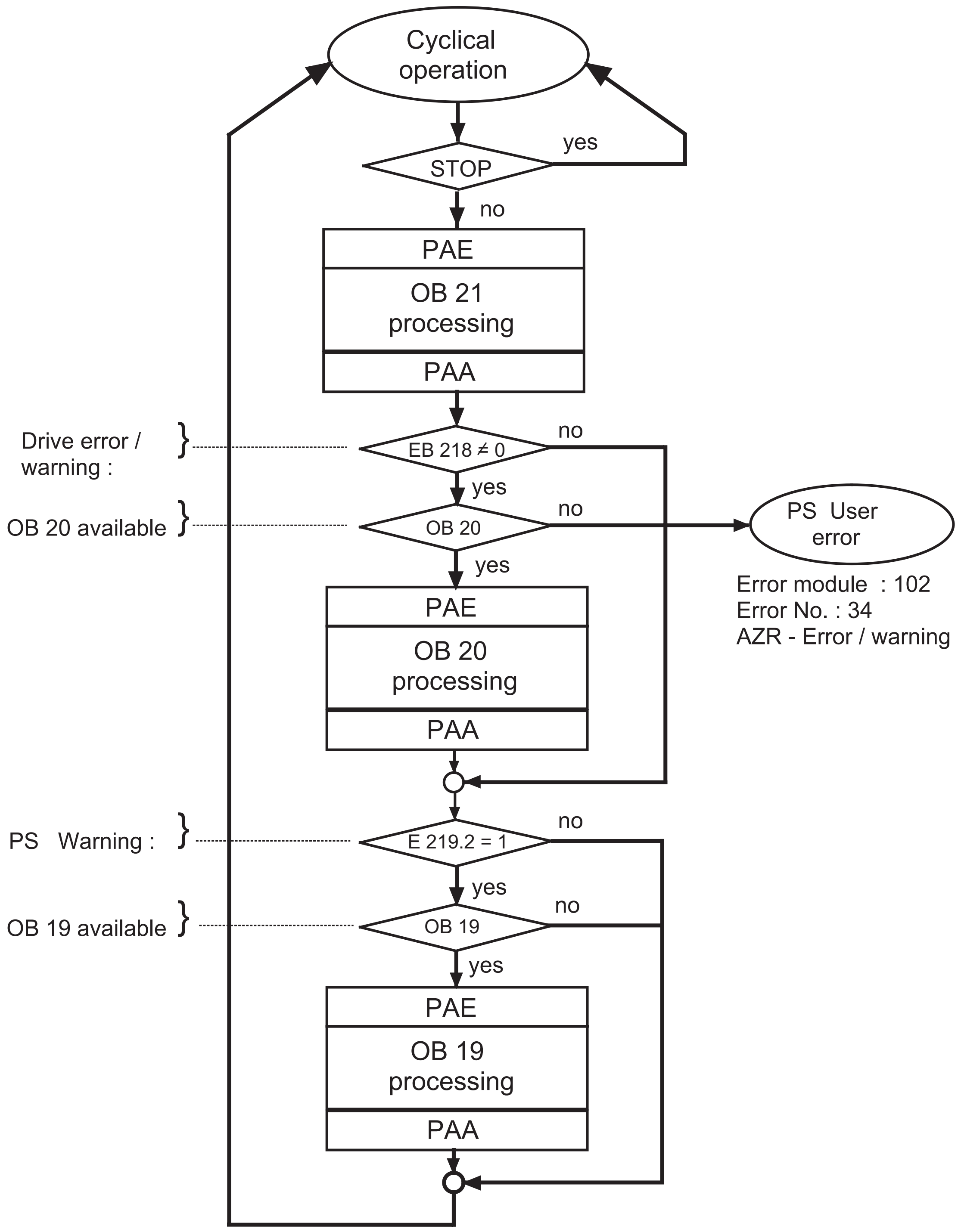
Flow chart: START OB processing“


PS RESET
STL START
PS program START from APROS
E/A/M/T/Z (I/O/F/T/C) --> "0" . Delete DB created in the not remanent memory
PAA
E/A/M/T/Z (I/O/F/T/C) --> "0" . Delete DB created in the not remanent memory
PAA
OB22 available
no
no
OB21 available
yes
yes
PAE
OB 22 processing
DB 0 evaluation *
PAA
PAE
OB 21 processing
DB 0 evaluation *
PAA
Cyclical operation

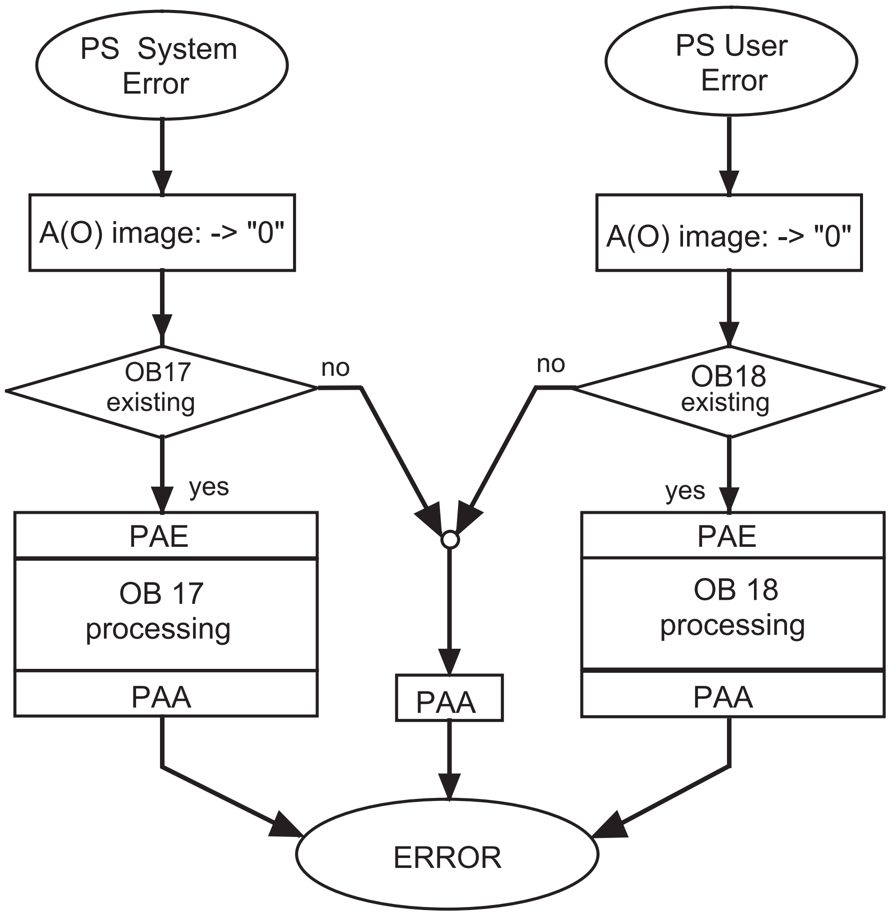
*depending on DB 0 function
(c.f. doc. --PS Instruction set-- "SYSTEM DB 0 Functions")
the according informations are
- either only with 1st PS RESET (after Power On),
- after every start
- or not at all at the start

PAE : Update process image INPUTS
(INPUT information is copied into process image INPUTS)
PAA : Update process image OUTPUTS
(The status of the process image OUTPUTS is transfered to the physical outputs)

Flow chart: „Cyclic mode

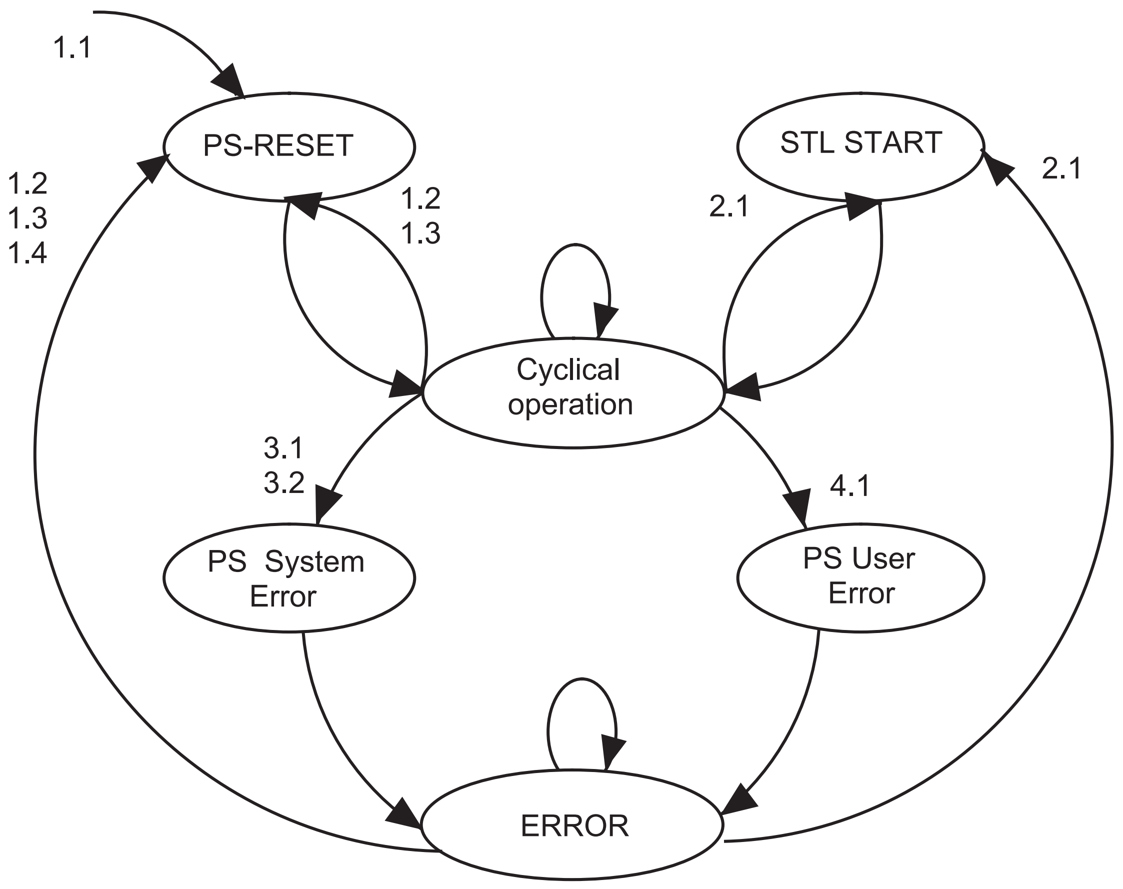


AMKASYN Programmable Controller PS Flowchart : " ERROR OB Processing "



PAE : Update process image INPUTS
(INPUT information is copied into process image INPUTS)
PAA : Update process image OUTPUTS
(The status of the process image OUTPUTS is transfered to the physical outputs)

Status chart



Event dependend changes to following states:

PS RESET : 1.1 After PS start-up during switching on of the AMKASYN system and output of System Ready SBM / SBT through the basic system.
1.2 By PS RESET through APROS
1.3 By operation of the RESET switch on the PS card.
1.4 By " ERROR RESET " if the PS is in " ERROR " status.
STL START : 2.1 By STL START through APROS

PS System Error : 3.1 If a PS system error is detected (error category " S " , c.f. PS Diagnosis messages / " K : 1 " AZB diagnosis).
3.2 After a call for OB 17 by a JUMP instruction.
PS User Error : 4.1 If a PS error is generated by the user (error category " A " c.f. PS diagnosis messages / " K : 2 " AZB diagnosis).

The state **"ERROR"** is only leaved by a change to the state "PS RESET" or "STL START"
In the state **"Cyclical operation"** the OB 01 (OB 20 and / or OB 19) are working.

4.1.2 Program blocks (PBs)

These program blocks contain normally the main part of the user program. They are selected by conditional or absolute call commands of organization or other program blocks. The number of the PBs is configured by AMK, depending upon the RAM memory required for them. Currently a maximum of 32 program blocks (PB 0...PB 31) are available. The permissible nesting depth, which results from selecting further blocks is determined at a maximum of 10 nestings (without return by BE, BEB or BEA).

4.1.3 Function blocks (FBs)

The function blocks are used by the user for frequently recurring or complex functions. They are divided into two groups:
The first group includes function blocks which the user programs himself. The number of these FBs is configured by AMK depending upon the RAM memory required for them. These are FB0 to FB63 (cf. Section 7). The second group comprises the AMK-specific FBs. They contain special functions programmed in the system software (not in AWL). They are used by the user and influenced in the scope of the permissible parameterization. The block numbers FB200 to FB255 are reserved for this second group. Further details are also given in the separate description "AMK-specific function blocks" (see Section 5). Function blocks are selected by conditional or absolute call commands from organization, program or function blocks.

4.1.4 Data blocks (DBs)

The data blocks represent areas in the user memory in which data can be made available from the process and for the process sequence for further processing in the AWL program. Data blocks can be created with the programming unit or they are generated in the AWL program with the instruction "E Dbxx". Before access to the data the required data block must be activated in the AWL program with the call "A Dbxx". The number of the DBs is configured by the AMK, depending upon the RAM memory required for them. Currently 64 data blocks (DB 0...DB 63) are available (cf. Section 7).

Remarks: Data blocks which are created with a programming unit (APROS) take up space in the EEPROM (program memory). They cannot be deleted or newly created in the AWL program. The instruction "E DB x" is ineffective for these DB numbers. It generates no error message! A change of the data contents in the program sequence is possible ("A DB x"; "T Dy z"). However, on PS reset the data set filed in the program memory is always loaded.

Note: DB0, the system DB, is reserved for AMK-specific applications. It may be used only in the course of the description shown in Section 4.2.
The data blocks DB01..DB15 are reserved for the AMK digital parallel interface (ADPS) and should not be used for other purposes! (Cf. documentation: "AMK digital parallel interface" or "AMK-specific function blocks", FB220.)

4.2 System DB functions

The system data block (DB0) allows a selection or configuration of the following system functions. DB addresses not listed are reserved for future PS versions and must remain preallocated with the value 0 for reasons of compatibility (see the following DB0 overview)!

System function / DB0 variable(s)	**NC-PS required**	**DB0 address**	**DB0 variable evaluation**
Cycle time monitoring:			
- Cycle monitoring time		DW 00	After every PS cycle
Process output image			
- PAA bit mask		DD 02	After every OB21/22 processing because of PS reset or AWL start
InterBus-S configuration			
- Option place		DR 04	PS3: After every OB21/22 processing because of PS reset or AWL start PS4: After first OB22 processing because of "Power on"
- Word length		DL 04	PS3: After every OB21/22 processing because of PS reset or AWL start PS4: After first OB22 processing because of "Power on"
- Mode		DR 05	PS3: After every OB21/22 processing because of PS reset or AWL start PS4: After first OB22 processing because of "Power on"
- Element number 1)		DL 05	PS4: After first OB22 processing because of "Power on"
Global status configuration	Yes		
- Global status DB number		DR 06	After every OB21/22 processing because of PS reset or AWL start
- Global status subscriber bit mask		DD 08	After every OB21/22 processing because of PS reset or AWL start
Operator panel communication	Yes		
- Operator panel send compartment DB number		DR 10	After every OB21/22 processing because of PS reset or AWL start
- Operator panel receive compartment DB number		DL 10	After every OB21/22 processing because of PS reset or AWL start
Following error display			
- AW number		DR 12	After every PS cycle
- AA number		DL 12	After every PS cycle
- Analog output final value		DD 14	After every OB21/22 processing because of PS reset or AWL start

System function / DB0 variable(s)	**NC-PS required**	**DB0 address**	**DB0 variable evaluation**
- Command value multiplier		DD 16	After every OB21/22 processing because of PS reset or AWL start
- Command value divider		DD 18	After every OB21/22 processing because of PS reset or AWL start
Config. AW message 32 averaging			
- AW mask		DR 20	After every OB21/22 processing because of PS reset or AWL start
- Sampling intervals		DL 20	After every OB21/22 processing because of PS reset or AWL start
Communication adapter subscriber address			
- Adapter1 address		DR 24	After first OB22 processing because of "Power on"
- Adapter2 address		DL 24	After first OB22 processing because of "Power on"
- Default module number		DR 25	After first OB22 processing because of "Power on"
- Updating mode [2)]		DL 25	After first OB22 processing because of "Power on"
NC parameter transfer	Yes		After every OB21/22 processing because of PS reset or AWL start
- DBNo.Channel1		DR 26	
- DBNo.Channel2		DL 26	
- Number of parameters		DW 27	
Activation mask[3)]		DD 30	After every OB21/22 processing because of PS reset or AWL start
Trace[3)]			After every OB21/22 processing because of PS reset or AWL start
- Mode byte (Trace1)		DR 32	
- DB number (Trace1)		DL 32	
- Info pointer (Trace1)		DD 34	
- Mode byte (Trace2)		DR 36	
- DB number (Trace2)		DL 36	
- Info pointer (Trace2)		DD 38	

[1)] as from version AZ-PS4 V0210 / 4397
[2)] as from version AZ-PS4 V0211 / 3098
[3)] as from version AZ-PS4 V0212 / 1999

DB0 overview:

Data	High-word		Low-word	
double word	H-byte	L-byte	H-byte	L-byte
0	Reserve = 0		Cycle monitoring time	
2	PAA bit mask			
4	Element number	Mode	Word length	Option place
6	Reserve = 0		Reserve = 0	Global status DB number
8	Global status subscriber bit mask			
10	Reserve = 0		Operator panel receive mailbox DB number	Operator panel send mailbox DB number
12	Reserve = 0		AA number	AW number
14	Final value analog channel			
16	Command value multiplier			
18	Command value divider			
20	Reserve = 0		Sampling intervals	AW mask
22	Reserve = 0		Reserve = 0	Reserve = 0
24	Updating mode	Default module number	Adapter2Adr	Adapter1Adr
26	Number of parameters		DBNo. Channel2	DBNo. Channel1
28	Spare = 0			
30	Activation mask			
32	Spare = 0		DB number (Trace1)	Mode byte (Trace1)
34	Info pointer (Trace1)			
36	Spare = 0		DB number (Trace2)	Mode byte (Trace2)
38	Info pointer (Trace2)			

4.2.1 Cycle time monitoring

The cycle time comprises the entire duration of processing the cyclic program, including the take-over of the "Process image inputs" (PAE) at the program start and the output of the "Process image outputs" (PAA) at the program end. It contains the call and the processing of the organization block OB01 and the program and function blocks called up in it with nesting. The cycle time is monitored in the PS. If the cycle monitoring time is exceeded, the PS generates the "ERROR" status and outputs an error message. The PS has a fixed standard cycle monitoring time. The user / programmer can define his own cycle monitoring time depending on the program. For this purpose he must create the data block DB0 and enter in it in the data value DW0 his cycle monitoring time (in milliseconds):

The following must apply: 0 < value in DW0, DB0 <= 32767

In the real time run the cycle time is now monitored for this value. In the test run monitoring is for 10 times this value (with restrictions, see table "Applies for test run").

Applies for real time run:

Standard cycle monitoring time: (for DB 0, DW 0 = 0)	2000 ms
Maximum cycle monitoring time : (for DB 0, DW 0 = 32767)	32767 ms

Applies for test run:

Standard cycle monitoring time: (for DB 0, DW 0 = 0)	20000 ms
Maximum monitoring time: for 1 <= value in DW 0, DB 0 <= 6553	10 ms x (value in DW 0, DB 0)
Maximum monitoring time: for 6554 <= value in DW 0, DB 0 <= 32767	65535 ms

4.2.2 Configuration of the process output image

To avoid conflicts when updating the process output image (PAA), e.g. if binary outputs of the AZ-R basic system are addressed directly due to the drive parameterization, binary outputs can be masked byte-wise from the PAA of the PS by means of the data double word 2 (DD 2) of the DB 0. These masked output bytes are then not updated by the PS:

The bit information of the bits 0..31 of the DD 2 correspond in this case with the PAA byte addresses 0..31 (or AB 0 .. AB 31):

- Bit Info = 0: byte is updated in the course of the PAA
- Bit Info = 1: byte is not updated in the course of the PAA

If DB0 is not created or at DD 2 = 0 in DB0, all output bytes are updated at the cycle end on assignment to the "Outputs" process image.

Note: DD 2 of the DB 0 must be defined already before processing the "Cyclic OB" (OB 01)! DB0 can be created with the programming device or during processing of the sequence OBs (OB21/OB22). After selection of the OB01 a change of the PAA configuration is no longer possible.

With PAE the corresponding output byte information is read back in the masked output byte of the process image (PA). Thus the bit information of the AZR basic system selected by drive parameterization for instance can be read out in the masked output byte (e.g. AB 7). (E.g. L AB 7, U A 7.0, ..; cf. Section 9.4.6).

4.2.3 InterBus-S configuration

As from "Software version AZ-PS4 V02.09 / 1197" there is with an AZ-PS4-I module the possibility of InterBus-S communication in connection with an AZ-PS4 functionality (AZ-PS4-I = AZ-PS4 module + AZ-IB1 communication adapter). The communication based on an AZ-PS4-I is upwards compatible with the communication based on the AZ-PS3 module. Therefore the AZ-PS3 module is described initially below. The special features or the extended possibilities of the AZ-PS4-I module are then described in Section 4.2.3.2, based on Section 4.2.3.1.

4.2.3.1 InterBus-S configuration in an AZ-PS3 module

The AZ-PS3 option card is an extension of the AZ-PS1 card by an InterBus-S interface. This is a remote bus-slave connection (InterBus-S-ID code/Bit0..7 = 03 without PCP; or = F3 with PCP; PCP = Peripherals Communication Protocol). The access to the InterBus-S, from the viewpoint of the AWL programmer, is facilitated in the form of a direct mapping of the InterBus-S data in the E/A address space of the PS (cf. Section 6.3). The PCP mode (see below) facilitates the writing and reading access to data blocks.

For the InterBus-S data, 8 bytes E and 8 bytes A information are held in the process image of the AZ-PS3 module. The option place 4 for the AZ-PS3 card as well as an InterBus-S data width of 8 bytes (4 words) are basic setting values (cf. following E/A assignment table). The error status of the InterBus-S is displayed in MB 227 (cf. Section 6.5.1 or 6.5.4).

Remarks: It must be noted that in the process image of the PS the significance rises from lower significant to higher significant addresses; which does not necessarily also have to be the case in InterBus-S master modules!

The meaning of the individual bit information is defined as follows:

– MB 227: InterBus-S error flag:

MB 227	InterBus-S error flag	
.0	x	1: IBS remote bus check (e.g. remote bus cable connection defective, IBS master in the reset state,)
.1	x	1: IBS inactive (no data transmission cycle within 640 ms)
.2	x	1: IBS transmission error (more than two consecutive, faulty data transmission cycles)
.3	x	1: IBS reset (the subscriber was put into the reset state by the InterBus-S master because of a fatal fault)
.4	0	Currently not used
.5	0	Currently not used
.6	0	Currently not used
.7	0	Currently not used

Remarks: MB 227 = 0 ⇒ no error.

InterBus-S word length	**Most significant byte address**	**Least significant byte address**
Option place = 1		
1	1	0
2	3	0
3	5	0
4	7	0
Option place = 2		
1	9	8
2	11	8
3	13	8
4	15	8
Option place = 3		
1	17	16
2	19	16
3	21	16
4	23	16
Option place = 4 [1)]		
1	25	24
2	27	24
3	29	24
4 [1)]	31	24
[1)] Basic setting (default setting)		

Table: E/A assignment table

As from PS version AZ-PS3V02.07 the Peripherals Communication Protocol (PCP) can be selected through DR5.

Caution: The least significant word of the InterBus-S data is required for PCP and in the PCP mode is no longer accessible directly in the process image of the PS! It must be further considered that the least significant word of the InterBus-S data is assigned to the highest word in the E/A image. Moreover it must be observed that the highest word in the E/A image depends upon the configured InterBus-S word length (see table above).

A change of the EA addresses (on selection of another option place for the PS3) can be achieved by means of the system data block DB0. A change of the InterBus-S word length (1..4 words) as well as selection of PCP by means of DB0 can also be performed:

- Option place (DR 4):

0 (or DB 0 not created)	-> Option place 4 (default setting)
1..4	-> Option place 1..4

Caution: Option place 1 is permitted only in connection with an AZ system software ≥ V02.08!

- Word length (DL 4):

0 (or DB 0 not created)	-> Word length = 4 (default setting)
1..4	-> Word length = 1..4

- Mode (DR 5):

0 (or DB 0 not created)	->PCP deselected (default setting)
1	->PCP mode selected

Peripherals Communication Protocol "PCP"

General

For the AZ-PS3 option card of the AMKASYN system, the PCP slave protocol is implemented in the version 1.5 of the Phönix Contact company as from the software level AZ-PS3V02.07 and the hardware version AZ-PS3-V1.02. The following services are available:

- Reading/writing of access organization data (data block number, data word offset)
- Reading/writing a data block contents

The PCP protocol is initialized through the system data block DB0 (see above). For this at the start of the PS (after reset) the PCP mode, the word length and the slot in which the PS3 card is located in the AMKASYN must be preset in the DB0. After the PS has recognized the PCP mode, the received messages or the messages to be transmitted are processed.

Initializing the PCP protocol

The PCP protocol is initialized through the system data block DB0 on start of the PS (after reset). For this purpose this DB must be created with a length of at least 6 words and the data bytes relevant with regard to the InterBus-S configuration must be filled with initial values (see above):

<u>InterBus-S configuration-relevant data bytes:</u>

DR4	Option place	=	1..4 (slot of the PS3 card)
DL4	Word length	=	1..4
DR5	Mode	=	0: not PCP mode (default) 1: PCP mode

Caution: The least significant word of the InterBus-S data is required for PCP and in the PCP mode is no longer accessible directly in the process image of the PS! It must be further considered that the least significant word of the InterBus-S data is assigned to the highest word in the E/A image. Moreover it must be observed that the highest word in the E/A image depends upon the configured InterBus-S word length (see table above).

All other data bytes of the DB0 must be set by reference to the application and are not relevant for the use of the Interbus-S with PCP protocol.

Example 1: Initializing by creating the DB0 by means of APROS (2 words cyclic data + PCP mode)

```
;DB00 (system data block; e.g. created in APROS)
        :KF     0       ;DW00   Cycle monitoring time
        :KF     0       ;DW01   Reserved
        :KD     0       ;DD02   PAA bit mask
        :KB     4       ;DR04   Option place = 4     (E/A starting from EW/AW 24)
        :KB     1       ;DL04   Word length = 3      (EW/AW 24..26: 2 words cyclic data
                        ;                             EW/AW 28: 1 word PCP)
        :KB     1       ;DR05   Mode                 (PCP selected)
        :KB     0       ;DL05   Reserved
```

Example 2: Initialization in the course of the start OB22 (PCP mode without cyclic data)

```
;Start OB22
:L      KB   6
:E      DB   0              ;Setting up the DB0
:A      DB   0              ;Activating the DB0

:L      KB   4              ;e.g. Option place = 4 (E/A starting from EW/AW 24)
:T      DR   4
:L      KB   1              ;e.g. Word length = 1 ( EW/AW 24: 1 word PCP)
:T      DL   4
:L      KB   1              ; PCP mode
:T      DR   5
:BE
```

Communication relation list

The communication relation list was initialized with the following values, it must be observed apart from the "InterBus-S-ID code/Bit0..7 = F3" within the scope of the PCP master configuration:

Communication reference		2
Remote address		0
Connection type		MMAZ
Maximum transmittable "confirmed request" messages		1
Maximum receivable "confirmed request" messages		1
Maximum transmittable "acknowledged request" messages		1
Maximum receivable "acknowledged request" messages		1
Asynchronous control interval		0
Connection type		‘D’
Maximum PDU length high priority for "request/response"		0
Maximum PDU length low priority for "request/response"		240 [1)]
Maximum PDU length high priority for "indication/confirmation"		0
Maximum PDU length low priority for "indication/confirmation"		240 [1)]
Services for client functionality	Byte 1	0
	Byte 2	0
	Byte 3	0
Services for server functionality	Byte 1	128
	Byte 2	48
	Byte 3	0
Maximum number of outstanding client services		1
Maximum number of outstanding server services		1
Communication type		connection-oriented
Symbol name		‘AMK ‘

The maximum symbol length may be 11 bytes.

1) see Section 4.2.3.2: DB00, DL5; element number

Object directory

In the object directory the manufacturer-specific static object directory as from index 0x2000 is supported exclusively on the part of the AMK company. This directory has 2 entries which are defined as follows :

- Index 0x2000 "Access organization information"

Object type	Field
Data type	unsigned integer
Length of an element	16 bits
Number of the elements	2
Password	0
Group association	0
Access rights	read/write
Symbolic name	"ZugrOrgInfo"
Extension length	0
Field variable 1:	Data block number
Field variable 2:	Data word offset

- Index 0x2001 "Data block"

Object type	Field
Data type	signed integer
Length of an element	16 bits
Number of the elements	100 (see Section 4.2.3.2: DB00, DL5; element number)
Password	0
Group association	0
Access rights	read/write
Symbolic name	"DB"
Extension length	0
Field variable 1 .. 100:	Data field

Reading/writing a data block

Before a data block can be read/written, the data block number and the data word offset from which there should be reading/writing must firstly be sent to the slave by a write indication message with the index 0x2000. In this case the data block number can assume a value corresponding to the application between 0 and 63 (255). Since the agreed PCP-PDU can send only a maximum of 100 data words, but a data block can have a greater length (e.g. 256 words), one is able due to the data word offset to read/write the message starting from an arbitrary place of the data block. The subindex acts additively to the data word offset. It must be observed here that the subindex "1" always designates the first element of the field or the contents of the destination/source pointer. With subindex "0" the entire object is always treated.

Now the actual data of the block can be read/written through a read/write indication message with the index 0x2001.

If the AMK-PCP software detects a message which it cannot accept (e.g. data are longer than data block length), then an error acknowledgement is transmitted with the error code 0x68 (definition E_IBS_ACC_TYPE_CONFLICT in the PCP basic software of the Phönix Contact company).

Cyclic data

For transmission of cyclic data in connection with PCP, the word length in the DB0 must be entered greater than 1. The cyclic data can be maximum 3 words in length in the PCP mode, since the highest word in the E/A image (depending upon the configured InterBus-S word length) is reserved for the PCP communication and maximum 4 words can be transmitted (word length = 4). The cyclic data words are filed in the internal process E/A image depending upon the slot of the PS3 card or transmitted from it (see above: E/A assignment table).

4.2.3.2 InterBus-S configuration with an AZ-PS4-I module

The following description within the scope of this section is based on the preceding Section 4.2.3.1 and must be considered as addition regarding an AZ-PS4-I module (cf. Section 4.2.3).

As from "Software version AZ-PS4 V02.09 / 1197" the direct data exchange of maximum 8 bytes E and 8 bytes A information is supported.

As from "Software version AZ-PS4 V02.10 / 4397" PCP (Peripherals Communication Protocol) is supported in addition.

Caution: The InterBus-S communication is available only if a start OB (OB22 / or OB21) is created!

InterBus-S configuration-relevant data bytes:

Byte	Meaning	Value range	Default value [1)]
DR4	Option place	1..4	4
DL4	Word length	1..4	4
DR5	Mode	0: not PCP mode 1: PCP mode	0
DL5	Element number	1..100	100
[1)] If DW is not created; or if an inadmissible numerical value is specified			

With the **"Element number (DB0, DL5)"** a variable length of the PCP object "Data block (index 0x2001)" can be specified if wanted.
In contrast to this compare Section 4.2.3.1, object directory, index 0x2001:
- Number of the elements = 100, constant

A variable length of the "maximum PDU length" results due to this at the same time with:
"Maximum PDU length" = element number * 2 + 6 (default value = 240).
In contrast to this compare Section 4.2.3.1, communication relation list:
- Maximum PDU length low priority for "request/response" = 240, constant
- Maximum PDU length low priority for "indication/confirmation" = 240, constant

Caution: If the DB0 is not created, or an InterBus-S configuration-relevant data byte lies outside the above stated range, the default value is used!

4.2.4 Global status configuration (only for AZ-MC1 module)

In combination with the multi-control module "AZ-MC1" (CNC) it is possible to read the global status of all SBUS subscribers (cf. documentation: NC-PS interface) through a data block.

The provision of the global status is possible within the scope of the system data block configuration. DB0, DR6 and DD8 are provided for this in the system data block (cf. Section 4.2).

Remarks: DR6 and DD8 of DB0 are evaluated only in the course of the OB22/OB21 processing (on start). A change outside the start OBs remains without effect!

If the start OB22 is not created, the DB0 is not created or the DB0 was created with a length less than 10 words, then the global status display is not selected.

Otherwise it applies that:

– **Global status DB number (DR6):**
  1..63: Global status display in DB x (with x = GlobSt-DBNr)
  otherwise: no global status display

– **Global status subscriber bit mask (DD8):**
  0x00000000: The global status of all 32 SBUS subscribers is updated
  0x00000001: The global status of subscriber 1 is not updated
  0x00000002: The global status of subscriber 2 is not updated
  0x00000003: The global status of subscriber 1 and subscriber 2 is not updated
  .
  .
  0xFFFFFFFF: The global status of all subscribers is not updated

On selection of a permissible global status DB number (1..63), the corresponding DB (DB1..DB63) is generated with a length of 96 words.

The assignment of the global status bytes (G-Byte0..G-Byte5; cf. documentation: NC-PS interface) to the data block bytes is as follows:

Data word	Left-byte (DL)	Right-byte (DR)
0	G-Byte1 (Subscriber 1)	G-Byte0 (Subscriber 1)
1	G-Byte3 (Subscriber 1)	G-Byte2 (Subscriber 1)
2	G-Byte5 (Subscriber 1)	G-Byte4 (Subscriber 1)
3	G-Byte1 (Subscriber 2)	G-Byte0 (Subscriber 2)
4	G-Byte3 (Subscriber 2)	G-Byte2 (Subscriber 2)
.	.	.
94	G-Byte3 (Subscriber 32)	G-Byte2 (Subscriber 32)
95	G-Byte5 (Subscriber 32)	G-Byte4 (Subscriber 32)

Example for selection of the global status display for all SBUS subscribers within the scope of the start OB22:

;OB22 (Start OB at PS reset)

```
:L     KF     10
:E     DB     0       ;Generate DB 0 with 10 words

:A     DB     0       ;Select DB 0

:L     KB     16
:T     DR     6       ;Global status DB number = 16 (DB 16)
:L     KH     0
:T     DW     8       ;Subscribers 1..16 active
:L     KH     0
:T     DW     9       ;Subscribers 17..32 active

:BE
```

4.2.5 Operator panel communication (only for AZ-MC1 module)

In connection with the multi-control module "AZ-MC1" (CNC) and a NC operator panel, it is possible to display display texts as well as to evaluate an entry key status.

The provision of the entry key status as well as the selection of display texts is possible within the scope of the system data block configuration. DR10 and DL10 are provided for this in the system data block DB0 (cf. Section 4.1.4). Further the start OB22 or OB21 (cf. Section 4.1.1) is required.

If the start OB22 is not created, the DB0 is not created or the DB0 was created with a length less than 12 words, the operator field communication is not selected.

Otherwise it applies that:

- Operator panel send mailbox DB number (DR10):
  1..63: Operator panel send mailbox data block number
  otherwise: no display text selection

- Operator panel receive mailbox DB number (DL10):
  1..63: Operator panel receive mailbox data block number
  otherwise: no entry key status selection

Remarks: DR10 and DL10 of DB0 are evaluated only in the scope of OB22/OB21 processing (at start). A change outside the start OBs remains without effect!

On selection of a permissible operator panel send mailbox DB number or operator panel receive mailbox DB number (1..63) the corresponding DB (DB1..DB63) is generated with a length of 5 words.

The assignment of the display selection bytes (A-Byte0..A-Byte7) or of the entry key status bytes (E-Byte0..E-Byte7) to the data block bytes of the send or receive mailbox is as follows:

Send mailbox:

Data word	Left-byte (DL)	Right-byte (DR)
0	Reserve = 0	Send mailbox status
1	A-Byte1	A-Byte0
2	A-Byte3	A-Byte2
3	A-Byte5	A-Byte4
4	A-Byte7	A-Byte6

with: send mailbox status = 0:	transfer A-byte to operator panel
(send mailbox status = 1:	is set by AWL instruction!)

Receive mailbox:

Data word	Left-byte (DL)	Right-byte (DR)
0	Reserve = 0	Receive mailbox status
1	E-Byte1	E-Byte0
2	E-Byte3	E-Byte2
3	E-Byte5	E-Byte4
4	E-Byte7	E-Byte6

with: receive mailbox status = 0: no E byte status change
otherwise: Bit0 = 1 for E-Byte0 status change
.
.
Bit7 = 1 for E-Byte7 status change

Example for selection of the operator panel communication in the scope of the start OB22:

```
;OB22 (Start OB at PS reset)

        :L      KF      12
        :E      DB      0       ;Generate DB 0 with 12 words

        :A      DB      0       ;Select DB 0

        :L      KB      17
        :T      DR      10      ;Operator panel send mailbox DB number = 17 (DB 17)
        :L      KB      18
        :T      DL      10      ;Operator panel receive mailbox DB number = 18 (DB 18)

        :BE
```

4.2.6 Following error display

The fast functions "FIPW" and "FIPZ" comprise a following error compensation. As from software version AZ-PS3 V02.07 it is possible to display in the 2ms grid the following error of the position controller adjusted by the following error compensation component through an analog output of the AMKASYN basic system.

This display is possible within the scope of the system data block configuration. DR12, DL12 and DD14..DD18 are provided for this in the system data block DB0 (cf. Section 4.1.4). Further the start OB22 or OB21 (cf. Section 4.1.1) is required. Furthermore the position control difference (code 32824) of the corresponding AW **must** be selected by means of the config. AW message 32 (ID 32786).

If the start OB22 is not created, the DB0 is not created or the DB0 was created with a length less than 20 words, the display is not selected.

Otherwise it applies that:

- AW number (DR 12); can be overwritten at the program running time:

1..8:	Analog output of the corrected following error for AW1..AW8
otherwise:	no analog output

- AA number (DL12); can be overwritten at the program running time:

1..4:	Analog output AA1..AA4
otherwise:	Analog output AA1

- Analog output final value (DD14):

1..2^{31}-1:	Value for 10V output voltage
otherwise:	the value "2048" results in 10V output voltage

- Command value multiplier (DD16):

-2^{31}..2^{31}-1:	corresponding to ID 32893

- Command value divider (DD18):

2^{16}..2^{31}-1:	corresponding to ID 32892

Example for selecting the following error display within the scope of the start OB22:

Prerequisite is: ID 32786 = 32824 (with regard to AW1 for instance).

;OB22 (Start OB at PS reset)

```
        :L      KF      20
        :E      DB      0       ;Generate DB 0 with 20 words

        :A      DB      0       ;Select DB 0

        :L      KB      1
        :T      DR      12      ;Following error display AW1 selected
        :L      KB      1
        :T      DL      12      ;Display through analog output AA1
        :L      KD      1000
        :T      DD      14      ;1000 increments = 10V
        :L      KD      655360
        :T      DD      16      ;Command value multiplier (ID 32893)
        :L      KD      655360
        :T      DD      18      ;Command value divider (ID 32892)

        :BE
```

4.2.7 Config. AW message 32 averaging

As from software version AZ-PS3 V02.07 it is possible to smooth in the form of averaging the config. AW message 32 (e.g. feedback velocity values) held in the input double words ED160 .. ED190.

Averaging is possible within the scope of the system data block configuration. DR20 and DL20 are provided for this in the system data block DB0 (cf. Section 4.1.4). Further the start OB22 or OB21 (cf. Section 4.1.1) is required.

If the start OB22 is not created, the DB0 is not created or the DB0 was created with a length less than 22 words, averaging is not selected.

Otherwise it applies that:

- AW mask (DR 20):
  bit-coded
  - Bit 0 = 1: Averaging config. AW message 32 of AW1 (ED 160)
    0: no averaging
  - Bit 1 = 1: Averaging config. AW message 32 of AW2 (ED 164)
    0: no averaging
  - ....
  - Bit 7 = 1: Averaging config. AW message 32 of AW8 (ED 190)
    0: no averaging

- Sampling intervals per averaged value (DL 20);
  the statement is made as exponent to the base 2:
  1..6: 2..64 sampling intervals
  otherwise: 8 sampling intervals (default setting)

It must be observed that a new value is formed in the scope of the input image updating at the earliest after the time T = n * T_0 for the config. AW message 32 channels (input double words) selected by the AW mask, whereby it applies that:

- T_0 = 10ms
- n = Number of the sampling intervals per averaged value

At a default setting of 8 sampling intervals (corresponds to DL20 = 3) a time T = 80ms results.

Example for selecting averaging of the config. AW message 32 within the scope of the start OB22:

```
;OB22 (Start OB at PS reset)

        :L      KF      22
        :E      DB      0       ;Generate DB 0 with 22 words

        :A      DB      0       ;Select DB 0

        :L      KH      0003
        :T      DR      20      ;AW1 and AW2 selected regarding averaging
        :L      KB      4
        :T      DL      20      ; 2^4 = 16 sampling intervals (averaging time T = 160ms)

        :BE
```

4.2.8 Profibus-DP subscriber address (only for AZ-PS4 module)

The AZ-PS4 module allows from "Software version > AZ-PS4 V02.07/0796" the use of communication adapters. The communication adapter AZ-PB1 for instance facilitates a Profibus-DP slave connection according to DIN 19245, Part 3 (AZ-PS4-P module = AZ-PS4 module + AZ-PB1 communication module). The slave is designed as modular slave with a maximum module number of 6, as well as 8 bytes E/A per module (cf. device master data; GSD file on APROS installation floppy in the file "\APROS\ AMKGLOBL\PDP").

From the viewpoint of the AWL programmer the access to the Profibus is facilitated in the form of mapping the Profibus-DP modules in the E/A address space of the PS (e.g. read least significant Profibus-DP byte: L EB 16; write least significant Profibus-DP byte: T AB 16). 8 bytes E and 8 bytes A information are held in the process image of the AZ-PS4 module per module configured in the Profibus master. The first module occupies the E/A addresses 16..23. Thus a maximum of 48 E/A bytes (EB/AB16..63) are available with a maximum of 6 modules.

Remarks: It must be observed that no EA modules are configured in the option places 3 and 4 in the process image of the PS. (Option places 3 and 4 are reserved for the PS and the communication adapter AZ-PB.)
Further it must be observed that the significance rises from low significant to higher significant addresses in the process image of the PS; which does not necessarily have to be the case also in Profibus-DP master modules!

Caution: **For operating the Profibus-DP it is necessary that a start OB (OB22 or OB21) is created!**

It is possible to differentiate between two adapter addresses on the communication adapter by means of a jumper (BR1):

- Jumper not plugged in: Communication adapter = Adapter1.
- Jumper plugged in: Communication adapter = Adapter2.

If the AZ-PS4 is operated simultaneously with two communication adapters, these adapters must be differentiated as Adapter1 and Adapter2. If only one communication adapter is used, this can be configured optionally as Adapter1 or Adapter2.

The error status of the Profibus-DP slave is displayed in MB 227 (Adapter1) or MB 226 (Adapter2) depending upon the adapter address.

The meaning of the individual bit information is defined as follows:

– **MB 226: AZ-PS4 Adapter2 error flag (Profibus-DP)**

MB 226	Adapter2 error flag (Profibus-DP)	
.0	x	1: "BAUD_SEARCH" (determining the baud rate; e.g. if the communication connection is interrupted.)
.1	x	1: "NOT DATA_EX" (data exchange status not yet assumed, e.g. if the slave is not yet configured by the master.)
.2	0	Currently not used
.3	0	Currently not used
.4	0	Currently not used
.5	0	Currently not used
.6	0	Currently not used
.7	x	1: Error in the initialization of the communication adapter

Remarks: MB 226 = 0 ⇒ no error.

– **MB 227: AZ-PS4 Adapter1 error flag (Profibus-DP)**

MB 227	Adapter1 error flag (Profibus-DP)	
.0	x	1: "BAUD_SEARCH" (determining the baud rate; e.g. if the communication connection is interrupted.)
.1	x	1: "unequal DATA_EX" (data exchange status not yet assumed, e.g. if the slave is not yet configured by the master.)
.2	0	Currently not used
.3	0	Currently not used
.4	0	Currently not used
.5	0	Currently not used
.6	0	Currently not used
.7	x	1: Error in the initialization of the communication adapter

Remarks: MB 227 = 0 ⇒ no error.

In the course of the system data block configuration Adapter1 and Adapter2 can be assigned in each case to a Profibus-DP subscriber address. The data bytes DR24 and DL24 are provided for this in the system data block DB0. Further as from a "Software version ≥ AZ-PS4 V02.09" the default module number can be specified by means of DR25. As from "Software version ≥ AZ-PS4 V02.11e" the updating mode can be set by means of DL25.

If the DB0 is not created, the DB0 was created with a length less than 26 words or "DR24=0 and DL24=0", the following default assignment applies:

- Adapter1 (jumper not plugged in): Profibus-DP subscriber address = 2.
- Adapter2 (jumper plugged in): Profibus-DP subscriber address = 3.

If DR25=0, then: default module number = 2 (corresponding to 16 E/A bytes).

If DL25=0, then default updating mode = updating the PS input image in the OB01 cycle.

Otherwise it applies that:

– Adapter1Adr (DR 24):
  0..126: Profibus-DP subscriber address = 0..126; if Adapter1 selected

- Adapter2Adr (DL 24):

0..126:	Profibus-DP subscriber address = 0..126; if Adapter2 selected (jumper plugged in)

- Default module number (DR 25):

1..6:	Default module number = 1..6 (corresponding to 8..48 E/A bytes).

- Updating mode (DL 25):

0:	Updating the PS input image in the OB01 cycle
1:	Updating the PS input image in the OB01 cycle and Updating the SF input image in the SF cycle (set by ID 2)

Example of allocating a Profibus-DP subscriber address deviating from the default assignment:

<u>Task:</u> Independently of the adapter selection (jumper setting), the Profibus-DP communication adapter should assume the Profibus-DP subscriber address "5". 6 modules (48 E/A bytes) should be configured by default setting. The access of fast functions to Profibus-DB input information shall be facilitated in the SF cycle grid with updating mode "1". All other DB0 functions are not used (value = 0).

```
;DB00 (System data block; e.g. created in APROS)
      :KF     0       ;DW00  Cycle monitoring time
      :KF     0       ;DW01  Reserved
      :KD     0       ;DD02  PAA bit mask
      :KB     0       ;DR04  Option place
      :KB     0       ;DL04  Word length
      :KB     0       ;DR05  Mode
      :KB     0       ;DL05  Reserved
      :KB     0       ;DR06  Global status DB number
      :KB     0       ;DL06  Reserved
      :KF     0       ;DW07  Reserved
      :KD     0       ;DD08  Global status subscriber bit mask
      :KB     0       ;DR10  Operator panel send mailbox DB number
      :KB     0       ;DL10  Operator panel receive mailbox DB number
      :KF     0       ;DW11  Reserved
      :KB     0       ;DR12  AW number
      :KB     0       ;DL12  AA number
      :KF     0       ;DW13  Reserved
      :KD     0       ;DD14  Final value analog channel
      :KD     0       ;DD16  Command value multiplier
      :KD     0       ;DD18  Command value divider
      :KB     0       ;DR20  AW mask
      :KB     0       ;DL20  Sampling intervals
      :KF     0       ;DW21  Reserved
      :KB     0       ;DR22  Reserved
      :KB     0       ;DL22  Reserved
      :KF     0       ;DW23  Reserved
      :KB     5       ;DR24  Adapter1Adr
      :KB     5       ;DL24  Adapter2Adr
      :KB     6       ;DR25  Default module number
      :KB     1       ;DL25  Updating mode
```

4.2.9 NC parameter transfer (only in connection with AZ-MC1 module as NC)

A transfer parameter field for "R parameters" of the NC program can be provided in the form of a data block (cf. documentation: NC-PS interface description).

The data block number (per NC channel) in which the transfer parameter field is held, as well as the number of the parameters transferable there is defined in the system data block DB0 (cf. Section 4.2). (A NC parameter is held in the DB as data double word).

On system initialization (after reset) these data blocks are generated automatically with a length of "Number of parameters * 2" words. In 2-channel systems data blocks with the same number of parameters are generated for both channels if a "DB number ≠ 0" is allocated for both channels. If "DBNo. Channel1=0" or "DBNo. Channel2=0", then no transfer parameter field is created for the corresponding NC channel.

– DBNo. Channel1:	0	No transfer parameter field created for NC channel 1
	1..63	DB 1.. DB 63 is transfer parameter field for NC channel 1
– DBNo. Channel2:	0	No transfer parameter field created for NC channel 2
	1..63	DB 1.. DB 63 is transfer parameter field for NC channel 2
– Number of parameters:	1.. 100	Parameter number (one data double word is assigned per NC parameter.
	-1..-100	The DB is created as remanent DB due to a negative parameter number (cf. Section 3.3.1).

4.2.10 Activation mask

The activation mask, DD 30 in DB0, permits the selection (activation) of different functionality. The DB 0, DD 30 is evaluated only at the end of a start OB.

The following functions can be activated:

As from version V02.12/xx99:

DD30, Bit0	= 0:	Write modes inactive (only the read modes 1, 2, 3 are permitted; see remarks below).
	= 1:	Write modes active (read modes 1, 2, 3 and write modes 4, 5, 6 are permitted; see remarks below).

Remarks:
Arbitrary addresses can be read or written by means of MD 212, MD 216, MB 220 for the AMK-internal use (monitor function).

High Word	Low Word		
Monitor value			MD 212
Monitor address			MD 216
Reserve = 0	Reserve = 0	Monitor mode	MB 220

The following apply:

Monitor mode =	0:	Monitor function inactive
	1:	In MD 212 the byte value of the memory address is displayed according to MD216.
	2:	In MD 212 the word value of the memory address is displayed according to MD216.
	3:	In MD 212 the double word value of the memory address is displayed according to MD216.
	4:	The value in MD 212 is written as byte value to the memory address according to MD216.
	5:	The value in MD 212 is written as word value to the memory address according to MD216.
	6:	The value in MD 212 is written as double word value to the memory address according to MD216.

Caution:
The write modes 4, 5, 6 are active only on write enable (DB 0, DD 30, Bit0=1). A start OB (OB 21 or OB 22) **must** be created for this!

4.2.11 Trace

As from version V02.12/xx99 two trace functions independent of one another can be configured by means of DB0, DD32 .. DD38 for the AMK-internal use (Trace1: DD32..DD34; Trace2: DD36..DD38).

- Each trace stores its information in a data block to be assigned to the trace.
- A trace entry is made on value change of the configurable information.
- Value changes are acquired in the 1ms grid (time level 1).
- In addition to the entry of the information there is an entry of the current time (in milliseconds).
- Storing the trace entries in the DB is organized in the form of a ring memory. An index cell refers to the most current entry of the ring memory (see below: Layout of the trace DB).
- The trace is controlled through a configurable flag byte (MB x).

Caution:
A start OB (OB 21 or OB 22) **must** be created for selecting the trace functionality! Further the trace DB must already by generated at the end of the start OB (e.g. by APROS or by command "E DB x"). The length of the DB determines the number of trace points.

The information of the system DB (DB 0) for the organization of the trace is specified as follows (cf. Section 4.2):

Mode byte, Trace1 (DR 32)

– Meaning:	Selection of the flag byte address for controlling Trace1.	
– Value range:	0:	Trace1 not activated.
	1..255:	MB 1..MB 255
– Example:	Value = 1 ⇒ MB 1 determines the mode of the trace function;	
		MB 1 = 0: Trace stops
		MB 1 = 1: Byte trace active
		MB 1 = 2: Word trace active
		MB 1 = 3: Double word trace active

DB number, Trace1 (DL 32)

– Meaning:	DB number of the data block which is provided for holding the trace data.	
– Value range:	0:	Trace1 not activated.
	1..63:	DB 1..DB 63
– Example:	Value = 16 ⇒ The trace data are filed in DB 16.	
– Remarks:	The DB must be generated at the end of the start OB (e.g. by APROS or command "E DB x" in the start OB). The length of the DB determines the number of trace points.	

Info pointer, Trace1 (DD 34)

– Meaning:	Start address of the information to be recorded in trace 1.	
– Value range:	0:	Trace 1 not activated.
	0x00000001.. 0xFFFFFFFF:	Trace address
– Example:	Value = 0x040001E8 ⇒	The data are filed as from address 0x040001E8 in the Trace1 DB (e.g. DB 16). (E.g. the EB 0 is updated on address 0x040001E8.)

Mode byte, Trace2 (DR 36)

– Meaning:	Selection of the flag byte address for controlling the Trace2.	
– Value range:	0:	Trace2 not activated.
	1..255:	MB 1..MB 255

– Example:	Value = 2 ⇒ MB 2 determines the mode of the trace function; MB 2 = 0: Trace stops MB 2 = 1: Byte trace active MB 2 = 2: Word trace active MB 2 = 3: Double word trace active

DB number, Trace2 (DL 36)

– Meaning:	DB number of the data block which is provided for holding the trace data.	
– Value range:	0:	Trace2 not activated.
	1..63:	DB 1..DB 63
– Example:	Value = 17 ⇒ The trace data are filed in DB 17.	
– Remarks:	The DB must already be generated at the end of the start OB (e.g. by APROS or by command "E DB x" in the start OB). The length of the DB determines the number of trace points.	

Info pointer, Trace2 (DD 38)

– Meaning:	Start address of the information to be recorded in Trace 2.	
– Value range:	0:	Trace2 not activated.
	0x00000001.. 0xFFFFFFFF:	Trace address
– Example:	Value = 0x040001F8	⇒ The data are filed from address 0x040001F8 in the Trace2 DB (e.g. DB 17). (E.g. the AB 0 is updated on address 0x040001F8.)

Layout of the trace DB :

High Word	Low Word	
Reserve = 0	Index n	DD 0
Reserve = 0		DD 2
Trace value 1		DD 4
Trace time 1		DD 6
....		
Trace value n		DD n*4
Trace time n		DD n*4 + 2
....		
Trace value y		DD y*4
Trace time y		DD y*4 + 2

With:

Index n (DW 0)

– Meaning:	Current trace position n.		
– Value range:	1..y:	with y = (DB length in words / 4) - 1	
– Example:	n = 10	⇒ The 10th trace entry is the most current entry. (Corresponding to the trace value 10 in DD 40 and the trace time 10 in DD42.)	

Trace value n (DD n*4)

– Meaning: Trace value in the format according to mode byte (byte / word / double word) sampled at trace time n.

Trace time n (DD n*4 +2)

– Meaning: Trace time in milliseconds.

4.3 CAN support

The AZ-PS5 module supports as from a "Software version ≥ AZ-PS5 V02.14/3500" the use of a CAN master communication adapter (AZ-CNS). The AZ-PS5 module with AZ-CNS is designated as **AZ-PS5-C**.

The AZ-CNS facilitates a CAN open master connection and moreover generates a hardware clock for clock-synchronous command value supply of decentralized slave drives (cf. documentation KU: KU-PSC option module). The communication relations are determined by means of a description file (*.ccb file), which is stored non-volatilely on the AZ-CNS module.

From the viewpoint of the AWL programmer, the access to the CAN bus is achieved in the form of the following two mechanisms:

- Mapping communication relations in the E/A address space of the PS. Byte, word or double word information can be exchanged between bus subscribers with this (e.g. read least significant CAN byte: L EB 16; write least significant CAN byte: T AB 16). The E/A addresses 16..63 (EB/AB16..63) are available for access to the CAN bus in the process image of the AZ-PS5-C module. Consistency over several bytes is not given.
- Mapping communication relations in the source and sink of "Fast functions (SF)". Word and double word information can be used clock-synchronously and consistently with this by means of "Fast functions" CAN-wide (cf. documentation: AMK-specific function blocks; FB 207 - SF initialization).

Remarks: It must be observed that no EA modules are configured in the option places 3 and 4 in the process image of the PS. (Option places 3 and 4 are reserved for the PS and the communication adapter AZ-CNS.)
Further it must be observed that the significance rises from low significant to higher significant addresses in the process image of the PS!

The status of the CAN nodes is displayed in MB226.

The meaning of the individual bit information is defined as follows:

– **MB 226: AZ-PS5-C status flag (CAN status)**

MB 226	AZ-PS5-C status flag	
.0	x	1: OPTBOARD_READY
.1	x	1: NETWORK_READY
.2	0	1: ERROR
.3	0	1: WARNING
.4	0	1: OPERATIONAL_MODE
.5	0	Currently not used
.6	0	Currently not used
.7	x	Currently not used

OPTBOARD_READY	Acknowledges start of CANopen (after basic initialization at system booting)
NETWORK_READY	Network initialization successful (only AZ-PS5-C: CAN master). Is set after starting by NMT command[1] "Switch into operational mode".
ERROR	Is set by CAN error.
WARNING	Is set by CAN warnings.
OPERATIONAL_MODE	Nodes are located in the operational mode.

[1] NMT = network management

5 AMK-specific function blocks

A general description of the function blocks is given in Section 4.1.3. The following table provides a summarizing overview of the current status of AMK-specific FBs:

AMK-specific function blocks (in machine code)	
Block	**Description**
FB200	Updating drive status
FB201	Commanding drive
FB202	Write/read drive parameter
FB203	Write/read user list 1
FB204	Initialize serial interface
FB205	Send through serial interface
FB206	Receive through serial interface
FB207	Initializing fast function (SF)
FB208	Commanding fast function (SF)
FB209	Synchronization after BA change
FB210	Table value calculation
FB211	Floating point arithmetic
FB212	Diagnosis
FB220	Initialize AMK digital parallel interface (ADPS)

A detailed description of the AMK-specific function blocks is provided in the documentation "AMK-specific function blocks" (Chapter 3).

6 Command overview

6.1 Operation set

Operation	Operands	Function description
Binary logic functions		
U	E/A/M/T/Z	AND operation
UN	E/A/M/T/Z	AND operation with negated operand
O	E/A/M/T/Z	OR operation
ON	E/A/M/T/Z	OR operation with negated operand
O		OR operation of AND functions
U(		AND operation before bracket expressions (max. nesting depth = 5)
O(		OR operation before bracket expressions (max. nesting depth = 5)
)		Close bracket
Memory functions		
S	E/A/M	Setting to the value "1" at RLO = 1
R	E/A/M	Resetting to the value "0" at RLO = 1
=	E/A/M	Allocating the RLO value

Operation	Operands	Function description
Time functions (and associated loading functions)		
SI	T	Starting a time with pulse at RLO 0->1
SV	T	Starting a time as lengthened pulse at RLO 0->1
SE	T	Starting a time as switch-on delay at RLO 0->1
SS	T	Starting a time as storing switch-on delay at RLO 0->1
SA	T	Starting a time as storing switch-off delay at RLO 1->0
R	T	Resetting a time at RLO = 1
L	KT	Load constant for time value (dec. in ms)
L	T	Load current time value (dec. in ms)
LC	T	Load current time value (BCD in s)
Counting functions (and associated loading functions)		
ZV	Z	Forwards counting at RLO 0->1
ZR	Z	Backwards counting at RLO 0->1
S	Z	Setting to value in ACCU1 at RLO 0->1
R	Z	Resetting to the value "0" at RLO = 1
L	KZ	Load constant for count value (dec.)
L	Z	Load current count value (dual)
LC	Z	Load current count value (BCD)

Operation	Operands	Function description
Loading functions (loading the corresponding operand in ACCU1)		
L	EB/AB/MB/PY	Load byte
L	EW/AW/MW/PW	Load word
L	ED/AD/MD/PD	Load double word
L	DL/DR	Load left/right data byte in the current DB
L	DW	Load data word of the current DB
L	DD	Load data double word in the current DB
L	KB	Load byte constant in decimal format
L	KC	Load ASCII constant (2 characters)
L	KF/KH/KM	Load word constant in the fixed point number/hex/binary format
L	KY	Load 2 byte constants in the decimal format
L	KD	Load double word constant in the decimal format
Transfer functions (transfer from ACCU1 into the corresponding operand)		
T	EB/AB/MB/PB	Transfer byte
T	EW/AW/MW/PW	Transfer word
T	ED/AD/MD/PD	Transfer double word
T	DL/DR	Transfer in left/right data byte in the current DB
T	DW	Transfer in data word in the current DB
T	DD	Transfer in data double word in the current DB

Operation	Operands	Function description
Comparison functions (ACCU2 with ACCU1)		
!=FD		Compare for "Equality"
><FD		Compare for "Inequality"
>FD		Compare for "Greater"
>=FD		Compare for "Greater or Equal"
<FD		Compare for "Less"
<=FD		Compare for "Less or Equal"
Arithmetic functions (ACCU2 operation ACCU1 -> ACCU1) (-> ACCU2)		
+FD		Addition
-FD		Subtraction
*FD		Multiplication
/FD		Division (result in Accu1, remainder in Accu2)
Digital logic operations (ACCU2 operation ACCU1 -> ACCU1)		
UD		Bit-wise AND operation
OD		Bit-wise OR operation
XOD		Bit-wise Exclusive-OR operation

Operation	Operands	Function description
Block functions		
SPA	OB/PB/FB	Unconditional block call (max. nesting depth = 10)
SPB	OB/PB/FB	Conditional block call at RLO = 1 (max. nesting depth =10)
BE		Block end
BEA		Unconditional block end
BEB		Conditional block at RLO = 1
A	DB	Activating (selecting) of a data block
E	DB	Generating/deleting a data block
Branch functions		
SPA =		Unconditional branch
SPB =		Conditional branch at RLO = 1
SPZ =		Branch at "Zero"
SPN =		Branch at "Not zero"
SPP =		Branch at "Sign plus"
SPM =		Branch at "Sign minus"
SPO =		Branch at "Overflow"
Shift functions		
SLD		Shifting the Accu1 contents to the left
SRD		Shifting the Accu1 contents to the right
Conversion functions		
KED		Convert Accu1 contents into one's complement
KZD		Convert Accu1 contents into two's complement
WDL		Convert BCD contents of Accu 1 into dual value
WDZ		Convert Dual contents of Accu 1 into BCD value
WBD		Convert 8-bit integer contents of Accu 1 into 32-bit integer
WWD		Convert 16-bit integer contents of Accu 1 into 32-bit integer

Operation	Operands	Function description
Processing functions		
B	MW	Process following command indirectly with flag word
B	DW	Process following command indirectly with data word
Organizational functions		
TAK		Exchange accumulator contents
TAB		Exchange Accu1: LwLb<->LwHb HwLb<->HwHb
TAW		Exchange Accu1: Lw<->Hw
TAD		Exchange Accu1: LwLb<->HwHb LwHb<->HwLb
NOP0		Zero operation 0 (all bit positions "0")
NOP1		Zero operation 1 (all bit positions "1")
STP		Stop after cycle end
STS		Stop immediately
		Blank line
***		Network end

6.2 Block set

Block	Description
System-supporting organization blocks	
OB01	Cyclic OB
OB17 .. OB19	PS error OBs
OB20	AZ error OB
OB21, OB22	Program start OBs
System-supporting data blocks	
DB00	System data block
DB01 .. DB15	Default blocks for AMKASYN digital parallel interface ADPS
AMK-specific function blocks (in machine code)	
FB200	Updating drive status
FB201	Commanding drive
FB202	Write/read drive parameter
FB203	Write/read user list 1
FB204	Initialize serial interface
FB205	Send through serial interface
FB206	Receive through serial interface
FB207	Initializing fast function (SF)
FB208	Commanding fast function (SF)
FB209	Synchronization after BA change
FB210	Table value calculation
FB211	Floating point arithmetic
FB212	Diagnosis
FB213	Reserved
FB220	Initialize AMK digital parallel interface (ADPS)
FB247	Reserved
FB248	Reserved
FB249	Reserved
FB250	Reserved

6.3 Process EA image

EA address	Periphery assignment
Inputs (E)	
EB 0 .. EB 7	Option place 1 [1)2)4)]
EB 8 .. EB 15	Option place 2 [1)2)4)]
EB 16 .. EB 23	Option place 3 [1)2)4)5)]
EB 24 .. EB 31	Option place 4 [1)2)4)5)]
EB 32 .. EB 61	[5)]
EB 62.. EB 127	E image from the AZ-MC1
Outputs (A)	
AB 0 .. AB 7	Option place [1 1)3)4)]
AB 8 .. AB 15	Option place 2 [1)3)4)]
AB 16 .. AB 23	Option place 3 [1)3)4)5)]
AB 24 .. AB 31	Option place 4 [1)3)4)5)]
AB 32 .. AB 61	[5)]
AB 62.. AB 127	A image to the AZ-MC1
[1)] AZEA8	Implements in each case byte 0, 8, 16 or 24
[2)] AZEA24/16	Implements in each case byte 0..2, 8..10, 16..18 or 24..26
[3)] AZEA24/16	Implements in each case byte 0..1, 8..9, 16..17 or 24..25
[4)] AZ-PS3 AZ-PS4-I	Implements alternatively byte 0..7 or 8..15 or 16..23 or 24..31 (The byte significance rises from the less significant byte address to the more significant address; cf. Section 4.2.3 IBS configuration)
[5)] AZ-PS4-P	Implements byte 16..61 (depending upon the configured module number; 8 bytes per module, always starting from E/A address 16)

6.4 AZ-MC1-specific E/A image

The E/A bytes 64..127 were defined in connection with the AZ-MC1 as NC (cf. documentation AZ-MC1, NC-PS interface).

6.5 Drive-specific E/A/M image

6.5.1 Overview

Drive-specific/ internal E/A/M	Drive-specific variable designation (or function description)
Inputs (E)	
ED 128 .. ED 156	32-bit position feedback value (AW1..AW8) 1)
ED 160 .. ED 188	By means of ID 32786 config. 32-bit value (AW1..AW8) 1)
EW 192 .. EW 206	By means of ID 32785 config. 16-bit value (AW1..AW8)
EB 208 .. EB 211	Reserved
ED 212	APSF: SF source (cf. Docu.: Fast functions SF)
EB 216	APSF: E byte (cf. Docu.: Fast functions SF)
EB 217	APSF:SF commanding code (cf. Docu.: Fast functions SF)
EW 218	Error bit bar (AZ/PS error)
EW 220	AZ ready and acknowledgement messages
EW 222	AZ warning messages
EW 224 .. EW 238	AW status messages (AW1..AW8)
EW 240 .. EW 254	AW warning messages (AW1..AW8)
Outputs (A)	
AB 128 .. AB 199	Reserved
AW 200	Arcnet diagnosis counter
AB 202 .. AB 211	Reserved
AD 212	APSF: SF sink (cf. Docu.: Fast functions SF)
AB 216	APSF: A byte (cf. Docu.: Fast functions SF)
AB 217 .. AB 239	Reserved
AW 240	By means of ID 32787, 32789, 32791, 32793 config. as AZ-analog output AA1..AA4
AW 242	By means of ID 32787, 32789, 32791, 32793 config. as AZ-analog output AA1..AA4
AW 244	By means of ID 32787, 32789, 32791, 32793 config. as AZ-analog output AA1..AA4
AW 246	By means of ID 32787, 32789, 32791, 32793 config. as AZ-analog output AA1..AA4
AW 254	AZ control bits (UE, RFx, FL)

1) Updating the 32-bit values depends upon the Sercos cycle time (ID 2; default = 10ms!).

Drive-specific/ internal E/A/M	Drive-specific variable designation (or function description)
Flag (M)	
MB 192 .. MB 209	Reserved
MW 210	1ms counter
MD 212	Reserved
MD 216	Reserved
MB 220	Reserved
MB 221	Reserved
MB 222.. MB 223	Reserved
MB 224.. MB 225	Reserved
MB 226	PS4 Adapter2 error flag (cf. Section 4.2.8)
MB 227	InterBus-S error flag (cf. Section 4.2.3) / PS4 Adapter1 error flag (cf. Section 4.2.8)
MB 228	Reserved
MB 229 .. MB 230	Reserved
MB 231	Current SF number regarding the SF status and error byte value (MB 232 / MB 233)
MB 232 / MB 233	Status / error byte (SF commanding)
MB 234	AW1..AW8 homed
MB 235	Send control byte (SBS) / error byte-RK512
MB 236	Receive control byte (SBE)
MB 237	Status and error byte (user list 1 / write and read remanent drive parameters; write/read diagnostic information)
MB 238 / MB 239	Status / error byte (change temp. drive parameters AW1 .. AW8)
MB 240 / MB 241	Status / error byte (commanding AW1)
MB 242 / MB 243	Status / error byte (commanding AW2)
MB 244 / MB 245	Status / error byte (commanding AW3)
MB 246 / MB 247	Status / error byte (commanding AW4)
MB 248 / MB 249	Status / error byte (commanding AW5)
MB 250 / MB 251	Status / error byte (commanding AW6)
MB 252 / MB 253	Status / error byte (commanding AW7)
MB 254 / MB 255	Status / error byte (commanding AW8)

6.5.2 Drive-specific input image

The meaning of the individual bit information is defined as follows:

– **ED 128,..,ED 156: 32-bit position feedback value AW1..AW8**

Caution: Updating the 32-bit position feedback values depends upon the Sercos cycle time (ID 2, default = 10ms).

	High Word / High Byte	High Word / Low Byte	Low Word / High Byte	Low Word / Low Byte
ED 128	Position feedback value xi (AW1)			
ED 132	Position feedback value xi (AW2)			
ED 136	Position feedback value xi (AW3)			
ED 140	Position feedback value xi (AW4)			
ED 144	Position feedback value xi (AW5)			
ED 148	Position feedback value xi (AW6)			
ED 152	Position feedback value xi (AW7)			
ED 156	Position feedback value xi (AW8)			

– **ED 160,..,ED 188: By means of 32786 config. 32-bit value AW1..AW8**

Caution: Updating the config. 32-bit values depends upon the Sercos cycle time (ID 2, default = 10ms).

	High Word / High Byte	High Word / Low Byte	Low Word / High Byte	Low Word / Low Byte
ED 160	AW message 32 (AW1) 1)			
ED 164	AW message 32 (AW2) 1)			
ED 168	AW message 32 (AW3) 1)			
ED 172	AW message 32 (AW4) 1)			
ED 176	AW message 32 (AW5) 1)			
ED 180	AW message 32 (AW6) 1)			
ED 184	AW message 32 (AW7) 1)			
ED 188	AW message 32 (AW8) 1)			

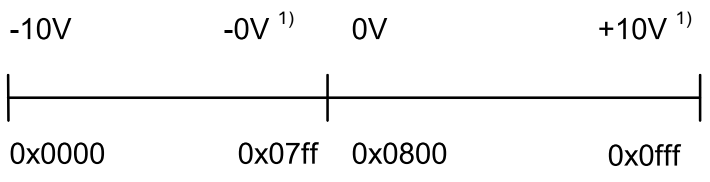
1) Configurable through ID 32786
(cf. documentation: AMKASYN digital pulse converter in modular construction; parameters):
(ID 32786) = 36 : Velocity command value
(ID 32786) = 40 : Velocity feedback value
(ID 32786) = 47 : Absolute position command value
(ID 32786) = 51 : Motor encoder position feedback value

– **EW 192,..,EW 206: By means of 32785 config. 16-bit value AW1..AW8**

	High Byte	Low Byte
EW 192	AW message 16 (AW1) 1)	
EW 194	AW message 16 (AW2) 1)	
EW 196	AW message 16 (AW3) 1)	
EW 198	AW message 16 (AW4) 1)	
EW 200	AW message 16 (AW5) 1)	
EW 202	AW message 16 (AW6) 1)	
EW 204	AW message 16 (AW7) 1)	
EW 206	AW message 16 (AW8) 1)	

1) Configurable through ID 32785
(cf. documentation: AMKASYN digital pulse converter in modular construction; parameters),
(ID 32785) = 32897 : Analog input A1
(ID 32785) = 32898 : Analog input A2

The data format of the 12-bit A/D converter of the AW modules is defined as follows:



1) -1 Increment

– **ED 212: SF source for SF APSF**

	High Word / High Byte	High Word / Low Byte	Low Word / High Byte	Low Word / Low Byte
ED 212	SF source for SF APSF (cf. documentation: Fast functions, SF APSF)			

– **EB 216: E byte for SF APSF**

	Byte
EB 216	E byte for SF APSF (cf. documentation: Fast functions, SF APSF)

– **EB 217: SF commanding code for SF APSF**

	Byte
EB 217	SF commanding code for SF APSF (cf. documentation: Fast functions, SF APSF)

– **EW 218: Error bit bar (AZ/PS errors)**

EB 218	Low-Byte	
.0	x	1: FATAL_ERROR
.1	x	1: WARNING_AW_MODUL
.2	x	1: ERROR_AW_MODUL
.3	x	1: WARNING_AZ_MODUL
.4	x	1: ERROR_AZ_MODUL
.5	x	1: KONFIG_FEHLER
.6	n	Currently not used
.7	n	Currently not used

EB 219	High-Byte	
.0	x	1: FATAL_ERROR_OPT_MODUL
.1	x	1: ERROR_OPT_MODUL
.2	x	1: WARNING_OPT_MODUL
.3	n	Currently not used
.4	n	Currently not used
.5	n	Currently not used
.6	n	Currently not used
.7	n	Currently not used

– **EW 220: AZ ready and acknowledgement messages**

EB 220	Low-Byte	
.0	x	1: Group ready message
.1	x	1: Inverter on acknowledgement
.2	x	1: Delete error acknowledgement
.3	n	Currently not used
.4	n	Currently not used
.5	n	Currently not used
.6	n	Currently not used
.7	n	Currently not used

EB 221	High-Byte	
.0	x	1: Q. controller enable for AW1
.1	x	1: Q. controller enable for AW2
.2	x	1: Q. controller enable for AW3
.3	x	1: Q. controller enable for AW4
.4	x	1: Q. controller enable for AW5
.5	x	1: Q. controller enable for AW6
.6	x	1: Q. controller enable for AW7
.7	x	1: Q. controller enable for AW8

– **EW 222: AZ warning messages**

EB 222	Low-Byte	
.0	x	1: Cooling air overtemp.
.1	x	1: Ext. component overtemp.
.2	x	1: Power supply unit overtemp.
.3	x	1: Feed/regeneration overtemp.
.4	x	1: Power supply overvoltage
.5	x	1: Power supply undervoltage
.6	n	Currently not used
.7	n	Currently not used

EB 223	High-Byte	
.0	n	Currently not used
.1	n	Currently not used
.2	n	Currently not used
.3	n	Currently not used
.4	n	Currently not used
.5	n	Currently not used
.6	n	Currently not used
.7	n	Currently not used

– **EW 224,..,EW 238: AW status messages AW1..AW8**

EB 224 EB 226 EB 228 EB 230 EB 232 EB 234 EB 236 EB 238	AW1 status message(Low-Byte) AW2 status message (Low Byte) AW3 status message (Low Byte) AW4 status message (Low Byte) AW5 status message (Low Byte) AW6 status message (Low Byte) AW7 status message (Low Byte) AW8 status message (Low Byte)	
.0	x	1: |Nsoll-Nist| < Velocity window (ID 157)
.1	x	1: |Nist| < Zero velocity window (ID 124)
.2	x	1: |Nist| < Velocity limit (ID 125)
.3	x	1: |Mist| > Torque limit (ID 126)
.4	x	1: |Mist| > Torque limit (ID 82 / ID 83)
.5	x	1: |Nsoll| > Limiting velocity (ID 38 / ID 39)
.6	x	1: |Xsoll-Xist| < Position window (ID 57)
.7	x	1: |Pist| >= Power limit (ID 158)

EB 225 EB 227 EB 229 EB 231 EB 233 EB 235 EB 237 EB 239		AW1 status message (High-Byte) AW2 status message (High Byte) AW3 status message (High Byte) AW4 status message (High Byte) AW5 status message (High Byte) AW6 status message (High Byte) AW7 status message (High Byte) AW8 status message (High Byte)
.0	x	1: Negative position limit reached (ID 50)
.1	x	1: Drive angle-synchronous (ID 228)
.2	x	1: Drive position-synchronous (ID 32952)
.3	x	1: Acknowledgement SWQ1 calibrated
.4	x	1: Acknowledgement IWQ1 calibrated
.5	x	1: Residual distance deleted (ID 32922)
.6	x	1: Overcurrent utilization > 50% load limit
.7	x	1: Positive position limit reached (ID 49)

– **EW 240,..,EW 254: AW warning messages AW1..AW8**

EB 240 EB 242 EB 244 EB 246 EB 248 EB 250 EB 252 EB 254		AW1 warning message (Low-Byte) AW2 warning message (Low-Byte) AW3 warning message (Low-Byte) AW4 warning message (Low-Byte) AW5 warning message (Low-Byte) AW6 warning message (Low-Byte) AW7 warning message (Low-Byte) AW8 warning message (Low-Byte)
.0	x	1: Nominal current excess (current limit integral)
.1	x	1: AW overtemperature
.2	x	1: Motor overtemperature
.3	n	Currently not used
.4	x	1: Supply voltage error (+/- 12V)
.5	x	1: Encoder error
.6	n	Currently not used
.7	x	1: Short circuit/ground fault output terminals

EB 241 EB 243 EB 245 EB 247 EB 249 EB 251 EB 253 EB 255		AW1 warning message (High-Byte) AW2 warning message (High-Byte) AW3 warning message (High-Byte) AW4 warning message (High-Byte) AW5 warning message (High-Byte) AW6 warning message (High-Byte) AW7 warning message (High-Byte) AW8 warning message (High-Byte)
.0	n	Currently not used
.1	x	1: Undervoltage error (DC bus not connected)
.2	n	Currently not used
.3	x	1: Excess error (ID 159)
.4	x	1: SERCOS communication error
.5	n	Currently not used
.6	n	Currently not used
.7	x	1: Manufacturer-specific error

6.5.3 6.5.5 Drive-specific output image

The meaning of the individual bit information is defined as follows:

– **AW 200: Arcnet diagnosis counter**

	High-Byte	Low-Byte
AW 200	Arcnet diagnosis counter (cf. documentation: Option module AZ-PS4-A)	

– **AD 212: SF sink for SF APSF**

	High Word / High Byte	High Word / Low Byte	Low Word / High Byte	Low Word / Low Byte
AD 212	SF sink for SF APSF (cf. documentation: Fast functions, SF APSF)			

– **AB 216: A byte for SF APSF**

	Byte
AB 216	A byte for SF APSF (cf. documentation: Fast functions, SF APSF)

– **AW 240,..,AW246: Configurable as analog output AA1..AA4 1)**

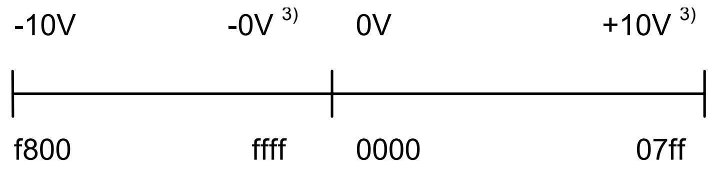
	High Byte	Low Byte
AW 240	Analog output AA1 2)	
AW 242	Analog output AA2 2)	
AW 244	Analog output AA3 2)	
AW 246	Analog output AA4 2)	

1) Configurable through ID 32787, .., ID 32793
(cf. documentation: AMKASYN "Parameters", e.g.:

ID 32787 = 32908.0:	Analog output AA1 = AW 240
ID 32788 = 2047:	Final value analog channel 1
ID 32789 = 32909.0:	Analog output AA2 = AW 242
ID 32790 = 2047:	Final value analog channel 2
ID 32791 = 32910.0:	Analog output AA3 = AW 244
ID 32792 = 2047:	Final value analog channel 3
ID 32793 = 32911.0:	Analog output AA4 = AW 246
ID 32794 = 2047:	Final value analog channel 4

2) see following data format description

The data format of the 12-bit D/A converter of the AZ module is defined as follows:



3) -1 Increment

– **AB 254: AZ control bits**

AB 254		
.0	x	1: Delete error
.1	x	1: Inverter On
.2	n	Currently not used
.3	n	Currently not used
.4	n	Currently not used
.5	n	Currently not used
.6	n	Currently not used
.7	n	Currently not used

Remarks: For error deletion by means of A 254.0 is must be assured initially that the acknowledgement of a previously executed error deletion was withdrawn (E 220.2 = 0 ?). After the start of error deletion (A 254.0 = 1) this signal must be kept active (= 1) until "Error deletion acknowledgement" is set by the AZ computer (E 220.2 = 1 ?). A 254.0 = 0 must then be set and the AZ computer then resets the acknowledgement (E220.2=0).

– **AB 255: Controller enables RFx for AW1 ... AW8**

AB 255		
.0	x	Controller enable for AW1
.1	x	Controller enable for AW2
.2	x	Controller enable for AW3
.3	x	Controller enable for AW4
.4	x	Controller enable for AW5
.5	x	Controller enable for AW6
.6	x	Controller enable for AW7
.7	x	Controller enable for AW8

6.5.4 Drive-specific flag image

The meaning of the individual bit information is defined as follows:

– **MW 210: 1ms counter (from version AZ-PS4 V02.10)**

Remarks: The 16-bit counter is incremented in the 1ms grid. At 0xffff it is counted on after 0x0000. Updating is independent of the process image updating or of the PS cycle time.

– **MB 221: PS Power-On status**

MB 221		
.0	x	1: Battery low at "Power On"
.1	x	1: Eprom boot up at "Power On"
.2	0	Currently not used
.3	0	Currently not used
.4	0	Currently not used
.5	0	Currently not used
.6	0	Currently not used
.7	0	Currently not used

Remarks: The power-on status is determined once on switching on the supply voltage and entered in the MB 221 at power-on reset (OB22) or AWL reset (OB21).

– **MB 226: AZ-PS4 Adapter2 error flag (Profibus-DP)**

(see: Section 4.2.8)

Remarks: MB 226 = 0 $\Rightarrow$ no error.

– **MB 227: InterBus-S error flag / AZ-PS4 Adapter1 error flag (Profibus-DP)**

(see: Section 4.2.3 or Section 4.2.8)

Remarks: MB 227 = 0 $\Rightarrow$ no error.

– **MB 231: SF number (SF commanding)**

Remarks: Number of the last commanded SF (SF number = 0.. 15 $\Rightarrow$ SF 0 .. SF 15).

– **MB 232: SF status (SF commanding)**

MB 232		
.0	0	Currently not used
.1	x	1: SF ACTIVE
.2	0	Currently not used
.3	0	Currently not used
.4	0	Currently not used
.5	0	Currently not used
.6	x	1: SF_FEHLER
.7	x	1: FB208 call not yet processed

Remarks: The SF status of the last commanded SF (SF number = 0 .. 7; cf. MB 231) is valid with M 232.7 = 0. (M 232.7 is set automatically to "1" by calling the FB208.)

– **MB 233: SF error (SF commanding)**

Remarks: The error number (SF error) of the last commanded SF (SF number = 0.. 7; cf. MB 231) is valid with M 232.7 = 0. (M 232.7 is set automatically to "1" by calling the FB208.)

SF error = 0:	no error.
Otherwise	cf. AMKASYN documentation, programmable control PS, error description; SF error.

– **MB 234: AW1..AW8 homed**

MB 234		
.0	x	1: AW1 homed
.1	x	1: AW2 homed
.2	x	1: AW3 homed
.3	x	1: AW4 homed
.4	x	1: AW5 homed
.5	x	1: AW6 homed
.6	x	1: AW7 homed
.7	x	1: AW8 homed

Remarks: The bit information of the individual drives is set automatically (=1) in the course of the homing run. Setting by the AWL is not permitted! Resetting by the AWL program on the other hand causes renewed homing, e.g. in the course of spindle positioning.

– **MB 235: Control byte transmission (SBS) / error byte RK512**

FB204: Mode = 0..2

MB 235	SBS	
.0	 0 1	Error in the last data transfer No error Error detected
.1	n	Currently not used
.2	n	Currently not used
.3	n	Currently not used
.4	n	Currently not used
.5	n	Currently not used
.6	n	Currently not used
.7	 0 1	Access right SF User program can access SF Operating system can access SF

FB204: Mode = 3..4

Error byte-RK512 = 0:	No error.
Otherwise:	cf. AMKASYN documentation, programmable control PS AMK-specific function blocks; FB204.

FB204: Mode = 5..6

MB 235	SBS	
.0 .. .6		Error number (cf. AMKASYN documentation, programmable control PS, AMK-specific function blocks; FB204)
.7	 0 1	Access right SF User program can access SF Operating system can access SF

– **MB 236: Control byte reception (SBE)**

FB204: Mode = 0..2

MB 236	SBE	
.0	 0 1	Error in the last data transfer No error Error detected
.1	n	Currently not used
.2	n	Currently not used
.3	n	Currently not used
.4	n	Currently not used
.5	n	Currently not used
.6	n	Currently not used
.7	 0 1	Access right EF User program can access EF Operating system can access EF

FB204: Mode = 5..6

MB 236		
.0 .. .6		Error number (cf. AMKASYN documentation, programmable control PS, AMK-specific function blocks; FB204)
.7	 0 1	Access right EF User program can access EF Operating system can access EF

MB 237: Status and error byte
(Write/read user list 1;
write/read remanent drive parameters;
write/read diagnostic information)

MB 237		
.0	 0 1	Error status FB202 or FB203 No error Error
.1	 0 1	Error status FB212 No error Error
.2	n	Currently not used
.3	n	Currently not used
.4	n	Currently not used
.5	n	Currently not used
.6	 0 1	Write/read status Starting status (FB 212 can be called) Execution FB 212 not yet completed
.7	 0 1	Write/read status Starting status (FB 202 or FB 203 can be called) Execution FB 202 or FB 203 not yet completed

– **MB 238: Status (changing temporary drive parameters)**

MB 238		
.0	x	1: FB202 re AW1 not yet processed
.1	x	1: FB202 re AW2 not yet processed
.2	x	1: FB202 re AW3 not yet processed
.3	x	1: FB202 re AW4 not yet processed
.4	x	1: FB202 re AW5 not yet processed
.5	x	1: FB202 re AW6 not yet processed
.6	x	1: FB202 re AW7 not yet processed
.7	x	1: FB202 re AW8 not yet processed

– **MB 239: Error (changing temporary drive parameters)**

MB 239		
.0	x	1: Error in FB202 re AW1
.1	x	1: Error in FB202 re AW2
.2	x	1: Error in FB202 re AW3
.3	x	1: Error in FB202 re AW4
.4	x	1: Error in FB202 re AW5
.5	x	1: Error in FB202 re AW6
.6	x	1: Error in FB202 re AW7
.7	x	1: Error in FB202 re AW8

– **MB 240, 242, .., 254: Status (commanding AW1, AW2, .., AW8)**

MB 240	(e.g. for commanding AW 1)	
.0	x	1: KMD set in the drive, order accepted
.1	x	1: KMD not interrupted
.2	x	1: KMD not ready
.3	x	1: KMD is in error status
.4	n	Currently not used
.5	n	Currently not used
.6	n	Currently not used
.7	x	1: FB201 call not yet processed, status still not valid

For the evaluation the less significant 4 bits as well as the most significant bit7 of the command status must be evaluated as follows:

0xxx0000: Basic status (after switching on) "RESET"
0xxx0011: KMD executed correctly in the drive "READY"
(e.g. command value output ended)
0xxx0101: KMD interrupted in the drive "HALT"
0xxx0111: KMD in execution in the drive "ACTIVE"
0xxx1111: Error status "ERROR"

x = arbitrary

– **MB 241, 243, .., 255: Error (commanding AW1, AW2, .., AW8)**

MB 241	(e.g. for commanding AW 1)	
.0	x	Error code 0: No error
.1	x	Error code 0: No error
.2	x	Error code 0: No error
.3	x	Error code 0: No error
.4	x	Error code 0: No error
.5	x	Error code 0: No error
.6	x	Error code 0: No error
.7	x	Error code 0: No error

7 Performance features

The performance features of the programmable control (PS) are determined by

- the drive system (AZ/AW system or KU system) within which the PS functionality is implemented and
- the option module which is used as PS module (AZ-PS3, AZ-PS4, AZ-PS5, KU-PSC).

AZ-PS3, AZ-PS4, AZ-PS5 shall be used exclusively within the scope of the AZ/AW system. The computing power increases with increasing PS number.

The KU-PSC on the other hand is a PS option module designed specially for the KU system with integrated CAN bus interface connection.

7.1 System limits in the scope of the AZ/AW system

System variable	Designator	Max. number	Other information
Process periphery (binary; external)			
Inputs	E	24 [1]	Bit inputs
		72 [2]	Bit inputs
		348 [4]	Decentralized inputs (CAN)
Outputs	A	24 [1]	Bit outputs
		48 [2]	Bit outputs
		348 [4]	Decentralized outputs (CAN)
Process periphery (binary; drive-internal)			
Inputs	E	304	Bit inputs
Outputs	A	16	Bit outputs
Process periphery (digital; drive-internal)			
Inputs	E	90	Byte inputs
Process periphery (analog)			
Inputs	E	16 [3]	12 Bit, 1ms, ± 10V, 15kOhm
		24 [4]	Decentralized EW (CAN)
Outputs	A	4	12 Bit, 1ms, ± 10V, 10mA
		24 [4]	Decentralized AW (CAN)
Temporary storage (volatile; user-specific)			
Flags	M	1024	Bit flag (MB 00 .. 127)
Temporary storage (volatile; drive-specific)			
Flags	M	656	reserved (MB 128 .. 209)
	M	368	Bit flag (MB 212 ...255)
Temporary storage (remanent; user-specific)			
User list 1	FB203	128	Word flag (up to AZ V02.04)
		254	Word flag (from AZ V02.05)
remanent data blocks	E DB x	15 kbyte	Distributed over several DBs (from AZ-PS4 V02.08)

[1] For AZEA8 periphery modules
[2] For AZEA24/16 periphery modules
[3] For 8 AW modules (2 inputs per AW module can be configured)
[4] From AZ-PS5-C V0214 (total E/A address space: E = 48 bytes / A = 48 bytes)

System variable	Designator	Max. number	Other information
Timers / counters			
Timers	T	32	Range: 0..2147483647 ms
Counters	Z	32	Range: 0..65535
Serial interface			
Init./send/ receive	FB204 - FB206 FB214 - FB216	1 1	RS422, RS485 [1]; 300 .. 76800 baud Protocols: transparent, 3964(R), RK512 (DB access), Modbus (DB access) RS232; 300 .. 76800 baud (from AZ-PS5 V0214)
Communication modules			
Extension	AZ-PB1	1	Profibus-DP: as from AZ-PS4-P V02.08
Extension	AZ-ARC	1	Arcnet: as from AZ-PS4-A V02.09
Extension	AZ-IB1	1	InterBus-S: as from AZ-PS4-I V02.10
Extension	AZ-CNS	1	CAN-Open: as from AZ-PS5-C V02.14
[1] only in connection with AZ-PS3 module			

System variable	Designator	Max. number	Other information
Blocks			
Organization blocks	OB	7	max. 255 operations per block
Program blocks	PB	32	max. 255 operations per block
Function blocks	FB	64	max. 255 operations per block
Data blocks	DB	64	max. 256 data words (16 bit) per block from AZPS4 V02.07 / APROS V02.09: max. 512 data words (16 bit), or max. 2001 data double words (32 bit) in connection with fast functions, SF)

Feature	Value	Other information
Nesting depth		
Bracket operations	5	O(, U(,)
Block calls	10	OB, PB, FB
Running time features		
AZ-PS3		
Cycle time	typ. 30ms	1000 operations
Min. reaction time	typ. 4ms	E -> A
AZ-PS4		
Cycle time	typ. 10ms	1000 operations
Min. reaction time	typ. 3ms	E -> A
AZ-PS5		
Cycle time	typ. 2ms	1000 operations
Min. reaction time	typ. 2ms	E -> A
Memory dependent features		
AZ-PS3		
Program and data memory	15kbyte	for OB, PB, FB and DB information (remanent)
AZ-PS4		

Program and data memory	48kbyte
	
	96kbyte
		for OB, PB, FB and DB information (remanent)
		from AZ-PS4 V02.13

Remanent data memory	15kbyte	DB (remanent): from AZ-PS4 V02.08
AZ-PS5		
Program and data memory	96kbyte	for OB, PB, FB and DB information (remanent)
Remanent data memory	15kbyte	DB (remanent)

7.1.1 Product-related limits (AZ-PS3 V02.08 / AZ-PS4 V02.13 / AZ-PS5 V02.13)

System variable	Designator	Max. number	Other information
Process periphery (binary; external)			
Inputs	E	24 [1]	Bit inputs
		72 [2]	Bit inputs
Outputs	A	24 [1]	Bit outputs
		48 [2]	Bit outputs
Process periphery (binary; drive-internal)			
Inputs	E	304	Bit inputs
Outputs	A	16	Bit outputs
Process periphery (digital; drive-internal)			
Inputs	E	90	Byte inputs
Process periphery (analog)			
Inputs	E	16 [3]	12 Bit, 1ms, ± 10V, 15kOhm
Outputs	A	4	12 Bit, 1ms, ± 10V, 10mA
Temporary storage (volatile; user-specific)			
Flags	M	1024	Bit flag (MB 00 .. 127)
Temporary storage (volatile; drive-specific)			
Flags	M	656	reserved (MB 128 .. 209)
	M	368	Bit flag (MB 212 ...255)
Temporary storage (remanent; user-specific)			
User list 1	FB203	128	Word flag (up to AZ V02.04)
		254	Word flag (from AZ V02.05)
Remanent data block	E DB x	15 kbyte	Distributed over several DBs (from AZ-PS4 V02.08)

[1] For AZEA8 periphery modules
[2] For AZEA24/16 periphery modules
[3] For 8 AW modules (2 inputs per AW module can be configured)

System variable	Designator	Max. number	Other information
Timers / counters			
Timers	T	32	Range: 0..2147483647 ms
Counters	Z	32	Range: 0..65535
Serial interface			
Init./send/ receive	FB204 - FB206	1	RS422, RS485 [1]; 9600 .. 76800 baud protocols: transparent, 3964(R), RK512 (DB-access), Modbus (DB-access)
Communication modules			
Extension	AZ-PB1	1	Profibus-DP: from AZ-PS4-P V02.08
Extension	AZ-ARC	1	Arcnet: from AZ-PS4-A V02.09
Extension	AZ-IB1	1	InterBus-S: from AZ-PS4-I V02.10

[1] only in connection with AZ-PS3 module

System variable	Designator	Max. number	Other information
Blocks			
Organization blocks	OB	7	max. 255 operations per block
Program blocks	PB	32	max. 255 operations per block
Function blocks	FB	64	max. 255 operations per block
Data blocks	DB	64	max. 256 data words (16 Bit) per block from AZPS4 V02.07 / APROS V02.09: max. 512 data words (16 Bit), or max. 2001 data double words (32 Bit) in connection with fast functions, SF)

7.2 System limits in the scope of the KU system

System variable	Designator	Max. number	Other information
Process periphery (binary; external)			
Inputs	E	9 [1]	Bit inputs
		348 [4]	Decentralized inputs (CAN)
Outputs	A	5 [1]	Bit outputs
		348 [4]	Decentralized outputs (CAN)
Process periphery (binary; drive-internal)			
Inputs	E	80	Bit inputs
Outputs	A	16	Bit outputs
Process periphery (digital; drive-internal)			
Inputs	E	10	Byte inputs
Process periphery (analog)			
Inputs	E	2	12 Bit, 1ms, ± 10V, 15kOhm
		24 [2]	Decentralized EW (CAN)
Outputs	A	3	8 Bit, 1ms, ± 10V, 10mA
		24 [2]	Decentralized AW (CAN)
Temporary storage (volatile; user-specific)			
Flags	M	1024	Bit flags (MB 00 .. 127)
Temporary storage (volatile; drive-specific)			
Flags	M	656	Reserved (MB 128 .. 209)
	M	368	Bit flags (MB 212 ...255)
Temporary storage (remanent; user-specific)			
User list 1	FB203	254	Word flags

[1] For KUEA1 periphery modules
[2] Total CAN-E/A address space: E = 48 bytes / A = 48 bytes

System variable	Designator	Max. number	Other information
Timers / counters			
Timers	T	32	Range: 0..2147483647 ms
Counters	Z	32	Range: 0..65535
Serial interface			
Init./send/ receive	FB204 - FB206	1	RS422, RS485 [1]; 300 .. 76800 baud Protocols: transparent, 3964(R), RK512 (DB access), Modbus (DB access)
Communication interface			
CAN interface (integrated)			

System variable	Designator	Max. number	Other information
Blocks			
Organization blocks	OB	7	max. 255 operations per block
Program blocks	PB	32	max. 255 operations block
Function blocks	FB	64	max. 255 operations pro block
Data blocks	DB	64	max. 512 data words (16 bit), or max. 2001 data double words (32 bit) in connection with fast functions, SF)

Feature	Value	Other information
Nesting depth		
Bracket operations	5	O(, U(,)
Block calls	10	OB, PB, FB
Running time features		
KU-PSC		
Cycle time	typ. 5ms	1000 operations
Min. reaction time	typ. 4ms	E -> A
Memory dependent features		
KU-PSC		
Program and data memory	15kbyte	for OB, PB, FB and DB information (remanent)
Dynamic data memory	4kbyte	DB information (volatile)

Functional restrictions or special features of the KU-PSC option module

The KU-PSC V0214 is based on the functional extent of AZ-PS5 V0214 with the following restrictions or special features:

- In the area of the process periphery (see above)
  - Maximum 9 binary inputs and 5 binary outputs can be implemented per KU system to the KU-EA1 periphery module. (E0.0,.., E0.7, E1.0; A0.0,.. ,A0.4).
  - 2 analog-to-digital converters (12bit) and 3 digital-to-analog converters (8bit) are available per KU system.
  - 48 Byte CAN E/A address space are available per KU system (EB16,.., EB63; AB16,.., AB63).
- In the area of the drive-specific E/A/M image (see above)
  - All drive-specific E/A/M addresses are relevant only for the AW1 address (KU drive).

- In the area of the command set
  - Generating remanent data blocks is not supported.
- In the area of system data block functionality (DB0).
  - The Interbus-S configuration by DR4, .. , DL5 is not required (without effect).
  - The global status configuration by DR6, .. , DD8 is not required (without effect).
  - The operator panel communication by DR10, DL10 is not required (without effect).
  - The communication adapter subscriber address by DR24, .. , DL25 is not required (without effect).
  - The NC parameter transfer by DR26, .. , DW27 is not required (without effect).
- In the area of the AMK-specific function blocks.
  - FB200 (drive status), FB201 (drive commanding) are limited to AW1 (KU drive).
  - FB202 (temporary parameter change) is limited to AW1 (KU drive).
  - FB202 (remanent parameter change) is limited to maximum 4 parameter sets (parameter set 0..3).
  - FB207 (initialization of fast functions) is limited to initializing maximum 8 SF. AW1 (KU drive) is permitted exclusively as drive sink. In addition special source and sink areas are provided for the synchronous CAN access (cf. documentation: AMK-specific function blocks; FB207).

    The following SF types are supported:
    - Type 0: COPY
    - Type 2: FIPZ
    - Type 3: STDFKT
    - Type 4: DFKT
    - Type 5: FGEN
    - Type 6: REGEL
    - Type 7: IMES
    - Type 9: XFIPW
    - Type 10: APSF
    - Type 11: EPOS (without "Replacing positioning" operating mode)
    - Type 12: NOCKE
  - FB211 (floating point arithmetic) is limited to the operator extent of the AZ-PS3 option module (+, -, *, /, sin, cos, root, modulus; cf. documentation: AMK-specific function blocks; FB211).
  - FB220 (initialize ADPS) is not supported.

- In the area of the KU-terminology different AZ/AW specific terms used in the course of the PS documentation shall be transferred analogous to the KU application. The following apply in particular:
  - Config AZ message → Config KU message (ID 32984)
  - AW message 16 → KU message 16 (ID 32785)
  - AW message 32 → KU message 32 (ID 32786)

8 Explanations of the terms and abbreviations used

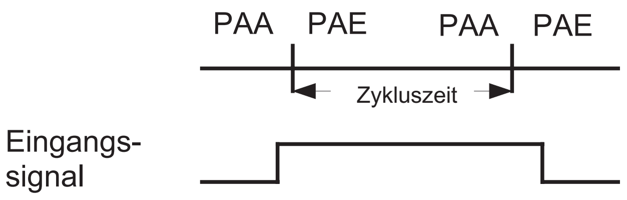
Term/abbreviation	Explanation
ADPS	AMK digital parallel interface
ACCU1	Accumulator 1 (on loading into ACCU1 the original value is shifted into ACCU2)
ACCU2	Accumulator 2
ARCNET	Arcnet communication
APROS	AMK-PS operating programming and test system
AWL	Instruction list
BCD	Binary coded decimal value
DL/DR	Data word (left/right byte)
DW	Data word (16 bits)
DD	Data double word (32 bits)
E/A/M/T/Z	Input/output/flag/timer/counter
EB/AB/MB/PB	Input/output/flag/periphery byte
EW/AW/MW/PW	Input/output/flag/periphery word
ED/AD/MD/PD	Input/output/flag/periphery double word
IBS	InterBus-S
OB/PB/FB/DB	Organization/program/function/data block
PDP	Profibus-DP (according to DIN 19245, Part3)
SF	Fast function(s)
RLO	Result of logic operation

9 Appendix

9.1 Signal characteristics

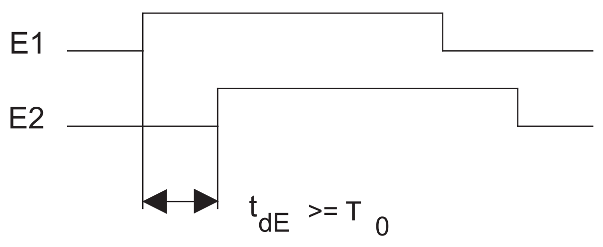
The time dependencies listed below must be taken into account for the PS program:

As is customary in programmable logic controls, the condition "Signal duration > cycle time" must be guaranteed. In addition, because of internal sampling of the E/A image in the T0 grid (time T0 = 1ms; from AZ-V02.04), the signal duration must be selected not less than T0.

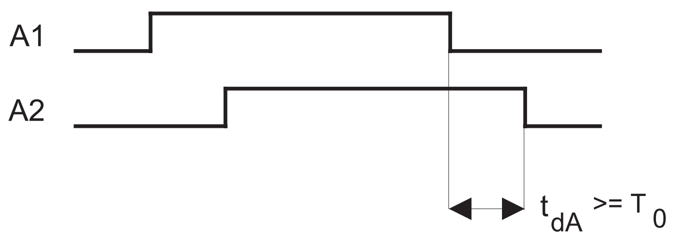


PAA: Prozeßabbild Ausgänge
PAE: Prozeßabbild Eingänge

– Because of the updating of external E/A signals in a fixed time grid T0 (time T0 = 1 ms from AZ-V02.04). The following conditions must also be taken into account:



Signal E2 wird garantiert nicht vor Signal E1 als "1" erkannt.



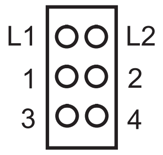
Signal A2 wird garantiert nicht vor Signal A1 zu "0" gesetzt.

9.2 Hardware features

9.2.1 LED assignment in AZ-PS1

Only the LEDs 1 and 2 have significance for the PS in the LED field on the front panel of the AZ-PS1 module (cf. following description):

- LED 1 is switched off with running PS program (status "RUN") after acquisition of the input image (PAE) and switched back on before output of the output image (PAA). In the case of an error ("ERROR" status) the LED 1 flashes with a constant flashing rate of 1 second. In the "HALT" status LED 1 is switched off.
- LED 2 is operated on logically with the card reset ("Off" on reset).

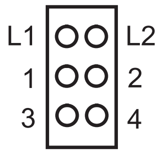


LED field (AZ-PS1 module)

9.2.2 LED and switch field assignment in AZ-PS3

Currently only the LEDs 1, 2 and 4 have significance with regard to the PS functionality in the LED field on the front panel of the AZ-PS3 module (cf. following description):

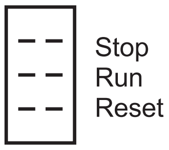
- LED 1: ERROR-LED
  In the case of error ("ERROR" status) LED 1 flashes with a constant flash rate of 1 second.
- LED 2: STOP-LED
  In the "HALT" status LED 2 is switched on.
- LED 4: RUN-LED
  The RUN-LED is switched off with running off with running PS program ("RUN" status) after acquisition of the input image (PAE) and switched back on before output of the output image (PAA). (-> LED 4 becomes darker with increasing cycle time). LED 4 is switched off in the „HALT“ or „ERROR“ status.



LED field (AZ-PS3 module)

A distinction is made between the following switch positions in the switch field on the front panel of the AZ-PS3 module (cf. following description):

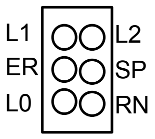
- Stop (stayput position):
  Cyclic/program processing is interrupted.
- Run (stayput position):
  Cyclic program processing is executed.
- Reset (touch position):
  A "PS reset", analogous to the APROS functionality (cf. documentation: APROS), is performed.



Switch field (AZ-PS3 module)

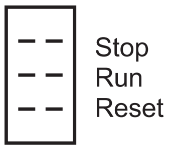
9.2.3 LED and switch field assignment in AZ-PS4

The LED field on the front panel of the AZ-PS4 module corresponds to that of the AZ-PS3 (cf. Section 9.2.2). The two additional LEDs (L1, L2) have currently no significance with regard of the PS functionality. The LED L0 is used in connection with the extended meaning of the switch field as from AZ-PS4 version 02.12/1999 (see below).



LED field (AZ-PS4 module)

The switch field assignment is identical to the AZ-PS3 module (cf. Section 9.2.2).



Switch field (AZ-PS4 module)

From the AZ-PS4 version 02.12/1999 the following addition functions were created by means of the switch field in the course of a PS reset:

- Function1: Delete all remanent data blocks.
- Function2: Delete the current PS project (and if a user program is filed in the system Eprom, load this; cf. Section 9.3).

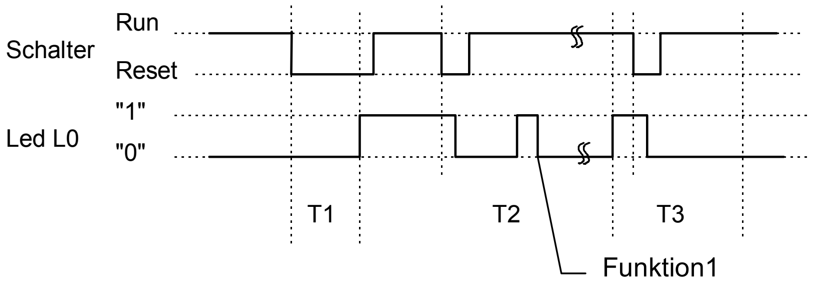
These functions are triggered by the sequence described below:

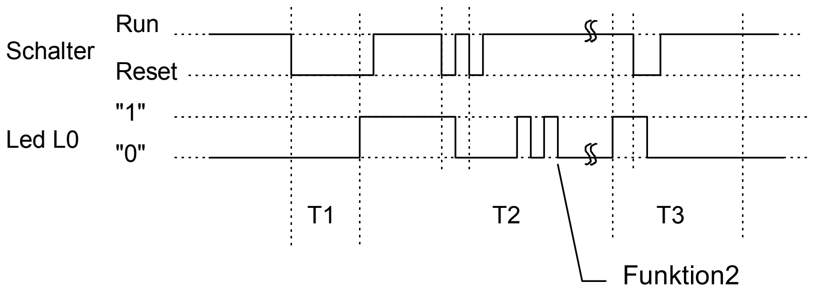
1. Function 1 or 2 is selected on pressing the reset key for at least 5 seconds (T1). After this time the LED0 (L0) is activated and the switch can be released (in the run position).

2. In the following 20 seconds (T2) function 1 can be selected by touching the switch once in the „Reset“ position. (By touching two-times function 2). Selection of the function is displayed by LED0: Flashing once for function 1; flashing twice for function 2.

3. After the selection time T2 has passed the LED0 remains active for 10 seconds. If the selection is confirmed in this time, (by touching in „Reset“ position), then the corresponding function is executed and the LED0 becomes inactive.

If the reset key is pressed shorter than T1 or if the time T2 or T3 passes without there being a selection or confirmation of the selection, then a previously customary PS reset is performed.





9.3 Load user program from system EPROM (EPROM boot-up)

The AZ-PS4 option module contains a battery-buffered file system for storing PS binary files. After switching on the PS loads the PS program contained in the file system. With withdrawn or empty battery the file system is empty and the PS goes into the error status.
With the "eprboot" software tool described here, the possibility is created of operating the PS (from version AZ-PS4-V02.11) also without or with empty battery. For this purpose the corresponding PS binary files, which have to be programmed in the system program EPROMs, must be generated from a PS project with the aid of „eprboot.bat“ PC program.

If the AZ-PS4 module contains system EPROMs with these PS binary files, then on booting the file "p01.sps" stored in the EPROM is firstly loading into the file system (boot-up) before reading the file system before the PS starts with the reading process of the data from the file system. A corresponding identifier prevents the boot-up process taking place with an existing battery-buffered file system.

9.3.1 Selecting the "eprboot.bat" program

"erpboot.bat" is a MS-DOS batch program for generating PS project binary files which can be programmed in AZ-PS4 system EPROMs. It is based on the "bootfile.exe" program and is controlled through the project name in the selection line.

Prerequisites are:

- APROS is installed.
- eprboot.bat is in the APROS folder (e.g.: c:\AMK\APROS; from APROS V02.10).
- bootfile.exe is in the APROS make folder (e.g.: c:\AMK\APROS\MAKE; from APROS V02.10).
- eprboot.bat is selected with statement of the project name from the APROS folder (e.g.: eprboot amkdemo; with amkdemo = project name of a demo project).

Result:
"eprboot.bat" generates in the EPRBOOT folder the two binary files "projectname.00" and "projectname.01" (with "projectname" = name of the APROS project in the batch selection; e.g.: amkdemo.00 or amkdemo.01). These binary files can be programmed as from AZ-PS4 system software V02.11 in the last 64k-bytes of the relevant system Eproms (e.g.: AZ-PS4 V02.11/0 or AZ-PS4 V02.11/1).

Caution:
Since the system Eprom can be deleted only in total, work must be done during the test phase with copies of the original system Eprom. Programming exclusively project binary files in the same Eprom several times is not possible!

Remarks:
In the **word-oriented** 4-Mbit system Eproms currently used by AMK, this corresponds to the **word address** 38000h to 3FFFFh (cf. Section 9.3.3). The EPRBOOT folder is generated automatically by "eprboot.bat".

9.3.2 Explanations for the boot-up

If a boot-up file (e.g. p01.sps) is available, then this is transferred to the file system during the switch-on process only in the case that the file system is unformated (if the battery was withdrawn previously or no battery is available).

As from the AZ-PS4 version 02.12/1999, a boot-up process can be initiated in the course of a PS reset by means of the switch field (cf. Section 9.2.3).

In the case of a boot-up having been made, **no battery error message** is generated. It is possible to check in M 221.1 whether a boot-up has taken place or not (cf. Section 6.5.4).

Thus the following operating modes are possible:

a) Operation without battery
   - Each boot-up starts with a boot-up process.
   - There is no battery error message.

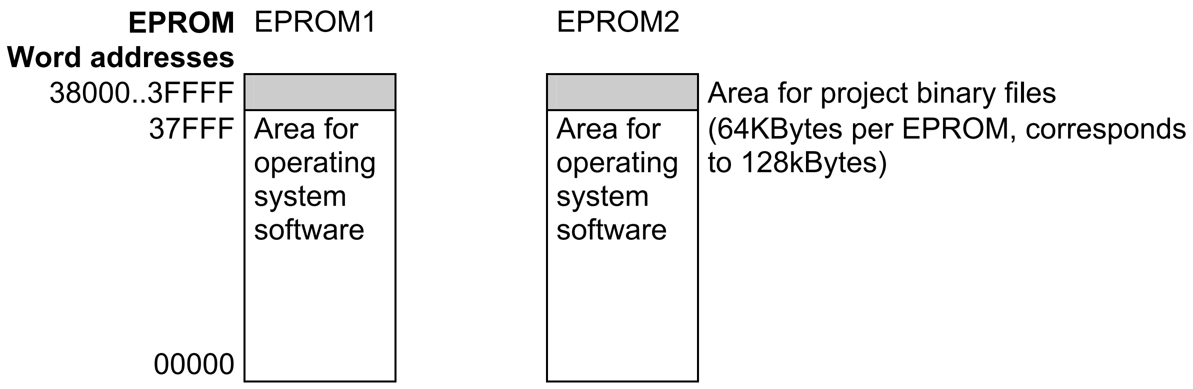
b) Operation with battery
   - One-time boot-up process at the 1st power up.
   - Then operation with battery, remanent data blocks possible, battery monitoring active.

Remarks for a):

- No operation with remanent data blocks is possible!
- Take care that sufficient time is waited between switching off the AMKASYN system and renewed switching back on until the power supply of the AZ-PS4 module goes out. Otherwise, because of the only partially deleted file system information, the following diagnostic message can arise when switching on: F:120 / M:1 (battery error), F:116 / M:11 (checksum error of the file system), M:104 / F:30 (PS program checksum error).

The PS4 module is then functional again only if it is free of voltage for a minimum time of some seconds (supply voltage = 0 **and** no battery plugged in).

9.3.3 Address subdivision


EPROM
Word addresses
EPROM1
EPROM2
38000..3FFFF
37FFF
00000
Area for operating system software
Area for operating system software
Area for project binary files (64KBytes per EPROM, corresponds to 128kBytes)

Programming example

Programming the project binary files (e.g. amkdemo.00 / amkdemo.01):
- Program amkdemo.00 e.g. in the system Eprom PS4 V02.11/0.
- Program amkdemo.01 e.g. in the system Eprom PS4 V02.11/1.

The following programming device settings must be used for both programming processes:

• RAM start address:	0x00000	
• RAM end address:	0x07FFF	(32K words)
• Device start address:	0x38000	

The RAM addresses describe for the programming device used the start and the end of the memory area in which the project binary files are loaded into the programming device.
The device start address identifies the Eprom address from which the data in the RAM area of the programming device are programmed in the Eprom.

Caution: The address statements are **word addresses** (corresponding to the used word-oriented system Eprom type).

9.4 Application notes

The following sections illustrate quite different system conditions and their possible effects on the characteristics of a PS application.

9.4.1 Temporary parameter change at the AZB (AZ operator panel)

Temporary parameter changes in the drive can be performed using the AZ operator panel, also while drive and PS are active.

If the query "Accept parameter? Yes / No" is answered with "No", there is no acceptance of the currently entered parameter in the remanent AZ data. However, the temporary parameter change remains effective. On entry of "Yes" on the other hand a change of the remanent AZ data is performed. The change is thus retained even on switching off the AMKASYN system.

A change of the remanent AZ data has the following effect among others:

After withdrawal and renewed activation of the controller enable

- the system is in the main operating mode (HBA),
- Messages such as "Reference point known" are deleted!

For further information cf. documentation: "Drive functions; Chapter 17, Effect of the controller enable signal".

9.4.2 Characteristics of fast functions (SF)

Fast functions work independently of the controller enable. They are not reset by the system software on withdrawal of the controller enable.

Fast functions which are started require a suitable drive operating mode, if they specify the command value for a drive. This operating mode must be set before starting the SF by a "Operating mode change" drive commanding (cf. documentation: "Fast functions", "AMK-specific functions", "Drive functions", "Parameters").

If several SF are cascaded, then the cascade must start at the lower SF number. Further it must be observed on starting the SF that the cascaded SF are started in the order of the cascade. This means beginning at the SF with the lowest SF number. (Remarks: A SF cascade arises if the output information of a SF is used as input information of a further SF.)

9.4.3 Changing the remanent AZ data, parallel to the PS mode with fast functions

If fast functions specify the reference input for a drive and if remanent AZ data are changed during the PS mode, the following must be observed after deactivating the controller enable in the PS user program (cf. Section 9.4.1 or 9.4.2):

- Before activating the controller enable of the drive, reset the SF (cf. documentation: "AMK-specific functions, FB208, commanding fast function"). The sink of the SF is released by this and thus the reference input specification for the drive is deselected.
- Before starting the SF, select the drive operating mode assigned to the SF anew (cf. documentation: "AMK-specific functions, FB201, commanding drive").

9.4.4 Position controller synchronization

The **position controller clock** of the AMKASYN inverters (AWs) can be adapted to the cycle time of "Fast functions" (SF). In this way the following error compensation (SAK) of the AWs can also be used in cycle times > 0.5ms.

The following possibilities arise:

- In the inputs "ED 160, .., ED 188" the mechanical angle offset of master and slave axis are displayed, also on coupling by a "Fast function". A prerequisite is that ID 32786 = 32824 (position control difference without SAK) is configured.
- In the input bits "E 225.2, .., E 239.2" it is displayed whether the corresponding axis lies within an adjustable tolerance range.

Compare the documentation: "Parameter set; ID 32786, config. AW message 32; ID 32952, LR velocity sync. window"; "PS command set; drive-specific E/A/M image".

Prerequisites

Software version

AZ software:	Special software	AZ V02.08 S 1897
AW software:		AW V02.12 3397

Settings

(1) NC cycle time (ID1) = position controller cycle (ID32958) = Sercos cycle time (ID2).

It must be observed that the position controller cycle (ID 32958) can be set per AW. ID 32958 can be input on the AZ operator panel (AZB) only through the "Sercos parameter" service menu item.

(2) Switch on the following error compensation in the operating mode selected for the SF (e.g. ID 32801 = 3C0804).

Reset the SF on withdrawal of the controller enable. The operating mode must always be set before starting the SF.

(3) For compensation of a SF-specific running time (dead time), specify the following value in the commanding DB, DW2 (SAK factor) in SF commanding:

SAK = 256,	if the input command value of the SF comes from the pulse encoder input (external drive).
SAK = 128 + 128 / ID2,	if the input command value of the SF originates from the position feedback value of another AW (internal drive). With statement of ID2 in ms.

(4) SFs may be cascaded only through the "Internal output value" (sink / source address = 255; cf. documentation: "AMK-specific function blocks, FB207, initialize fast functions").

It must be noted that the output value of the SF with a lower SF number always results in the input value of an SF with a higher SF number. The SFs must be started in the order of the SF numbers.

9.4.5 Communication: Siemens S7 and AMK AZ-PS4-P by means of Profibus DP

The GSD file required for the Profibus-DP master installation is also supplied on the APROS installation floppy in the "\AMKKLOBL\PDP" folder (GSD = device master data).

Up to APROS version V02.09, Allen-Bradley-specific instructions are contained in the last four lines of the GSD file. These instructions must be removed, or made ineffective by means of comment characters ";" (semicolon) for the integration of the GSD file by a S7 software version 3.1.

The S7 generates a type file with optionally a standard module and an AMK-specific module (PS4-8EA) of 8 bytes E/A length and consistency over the entire range. The S7 (e.g. CPU 315-2DP) sends the useful data only to this E/A if these are consistent over the entire range. This can be achieved with the aid of the integrated system functions SFC 14 and SFC 15.

9.4.6 Displaying bit information in the output image of the PS

The contents of the PS output bytes AB 0 to AB 31 are written by the PS at every OB01 run-through to the corresponding binary output bytes in the course of the output image updating (PAA).

The output image bytes AB 0 to AB 31 of the PS can be excluded from this output image updating by different mechanisms.

Exclusion mechanisms are:

- Masking out the output byte(s) by means of system DB0 (cf. Section 4.2.2), e.g. for outputting bit information of the basic system by means of 32846 and following (cf. AMKASYN documentation: Parameters).
- Masking out an output byte by initializing a fast function "SF" (cf. documentation: AMK-specific function blocks, FB 207), for outputting bit information of the SF per SF cycle.
- Masking out two output bytes by initializing ADPS (cf. documentation: AMK-specific function blocks, FB 220) for outputting the ADPS control and data bytes per ADPS cycle (cf. documentation: AMK digital parallel interface ADPS).

The information specified by another function is written back into the corresponding binary output byte (displayed) by the PS system software at the start of an OB01 run-through by input image updating (PAE).

The bit information of the basic system, or of the SF, for instance, can thus be evaluated in the correspondingly masked output byte of the PS.

Example
of use of the basic system bit information for the display of the selected operating mode (e.g. main operating mode 0 HBA0).

Prerequisites (AZ):
ID 32846 = 514 (Output port 1 address = AZ slot 1, 3rd byte; corresponds to AB 2)
ID 32847 = 33062.1 (Output port 1 source Bit0 = HBA active according to ID 32800; for AW1)
ID 32848 = 33062.2 (Output port 1 source Bit1 = HBA active according to ID 32800; for AW2)
....
ID 32854 = 33062.8 (Output port 1 source Bit7 = HBA active according to ID 32800; for AW8)
(Remarks: HBA = main operating mode)

Prerequisites (AZ-PS):
Masking the AB 2 from the output image update (PAA) by means of system DB0.

```
;System DB (DB00)
:KF 0    ;DW00:  Cycle monitoring:          Default time
:KF 0    ;DW01:  Spare
:KH 4    ;DD02:  PAA mask AB0 ..AB15:       AB2 masked
:KH 0    ;DD02:  PAA mask AB16..AB32:
```

Remarks:
No physical output is lost by using the AB 2. Access can be made to AB 2 by PS as to any other output (e.g. O A 2.0; L AB 2; ...).

10 Impressum

Title	**PS Command set**
Objective	**Description of the PS Command set**
Part-Number	**25795**

History

Date
2000/42

Service

Tel.: **+49/(0)7021 / 5005-191, Fax –193**

Business Hours:
Mo-Fr 7.30 - 16.30, On weekends and holidays calls are forwarded to an emergency response number by the automated answering system.

To assure a fast and accurate response to solve customer problems we ask for your cooperation in providing us with the following information:

- Nameplate data
- Software version
- System configuration and application
- Description of problem and presumed cause of failure
- Diagnostic message (error code)

Publisher

AMK Arnold Müller Antriebs- und Steuerungstechnik GmbH & Co. KG
Gaußstraße 37 – 39, 73230 Kirchheim/Teck

Tel.: 07021/5005-0, Fax: 07021/5005-176
E-Mail: info@amk-antriebe.de

For further information www.amk-antriebe.de