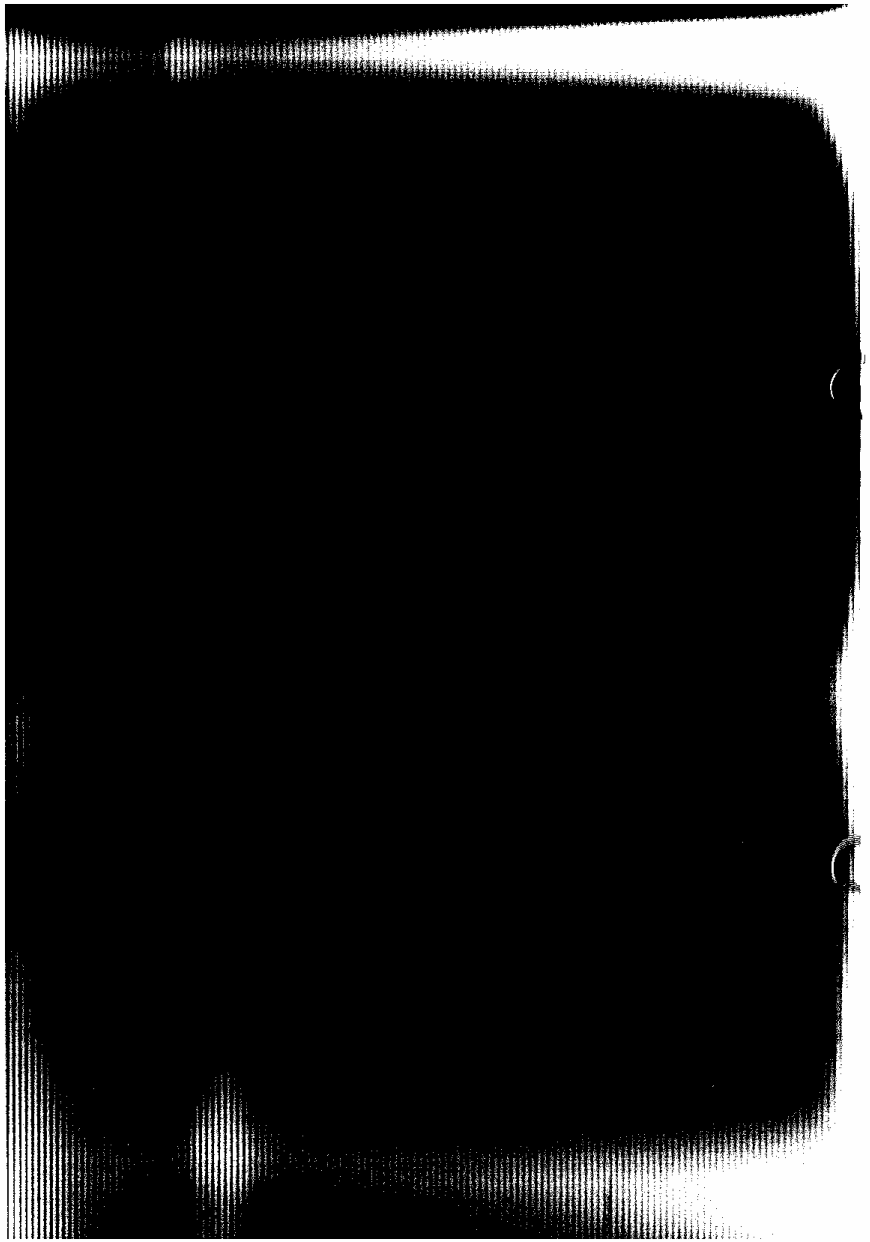


Klöckner MOELLER

Programming

SUCOcontrol PS 4 100 Series

SUCOS Automation

7/93 AWB 27-1158-GB

1st edition 7/93, printed 9/93

Authors: Peter Drath, Jörgen Prahl, Thomas Schumacher
Editor: Thomas Kracht
Translators: Terence Osborn, Katrin Schlemme

Printed on bleached cellulose.
100 % free from chlorine and acid.

Programming

Content

1 Programming Devices

The SUCOcontrol PS 4 100 series compact controller can be programmed by two devices – either with a hand-held programmer PRG 3 or with a personal computer (PC).

Programming with the PRG 3

The hand-held programmer PRG 3 is especially suitable for PLC beginners. Besides programming it also offers test functions which simplify commissioning.

Programming is done in instruction set IS, a programming language which is easy to comprehend. This can be carried out on the PRG 3 irrespective of whether it is connected to the PS 4 controller or not.

The programming software of the PRG 3 is divided into several menus. The different operations and functions are called up via multi-function keys and the operating software automatically activates only those keys and functions which apply to the initially selected function. This excludes the possibility of pressing the wrong keys.

For programming with the PRG 3 you need

- the hand-held programmer PRG 3
- the connecting cable KPG 2-PS 3 between PRG and PS 4 100 series
- documentation for operation and programming

All these elements are supplied with the PRG 3. The generation of user programs with the PRG 3 is described in detail in AWA 27-794.

Programming with the PC

The SUCOsoft S 30-S 3-GB programming software is required for programming with the PC. Like the PRG 3 programming can be carried out on the PC irrespective of whether it is connected to the PS 4 100 series or not.

The PC is connected to the PS 4 100 series either via the RS 232 interface or via the interface converter ZB4-501-UM1 or via the EPC 334.1 interface card.

Programming with the PC

SUCOsoft S 30-S 3-GB has a menu structure that enables the programmer to be guided interactively through the main and sub-menus.

SUCOsoft allows programming in

- instruction set (IS)
- ladder diagram (LD)
- function block language (FBL).

Test and commissioning functions, diagnostics, when running the PS 4 100 series, and extensive documentation functions simplify and help the use of the PS 4 100 series.

Programming with the PC requires the following prerequisites:

- MS-DOS operating system from version 3.3
- 640 kByte RAM working memory
- 5 $\frac{1}{4}$" or 3 $\frac{1}{2}$" floppy disk drive
- hard disk with at least a 10 MByte memory
- serial interface RS 232
- SUCOsoft S 30-S 3-GB, version V 2.3
- interface converter ZB4-501-UM1

or alternatively:

- SUCOsoft S 30-S 3-GB version V 2.3
- EPC 334.1 PC interface card

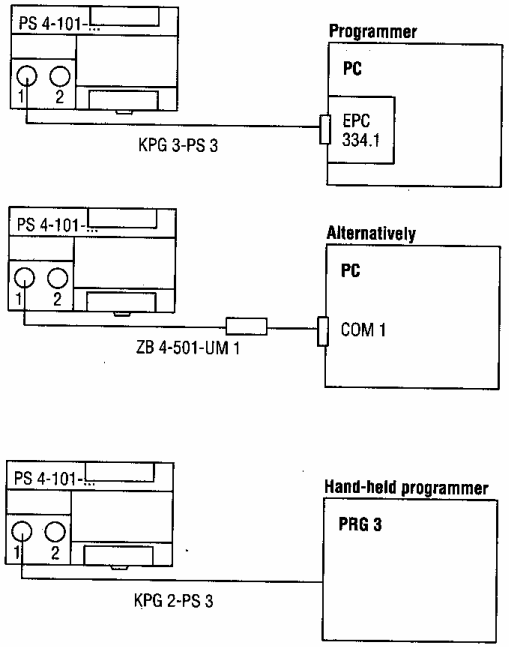


Fig. 1: Programming devices

Documentation for operation and programming

The generation of user programs with the PC is described in detail in the following manuals:

- AWB 27-1070-GB (IS)
- AWB 27-1071-GB (LD and FBL)

> **Note!**
> Earlier SUCOsoft versions than V 2.3 may be used for programming the PS 4 100 series, but they do not support the Hardware test and Status display functions. In this case it is also not possible to use the ZB4-501-UM1 interface converter.

2 Basic Programming Principles

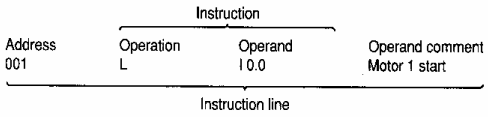
Program structure

A structured task definition is an essential prerequisite for the programming of a control system.

So from the beginning the user program needs a logical and clear structure. This starts by dividing the program structure into function blocks and ends with a sensible commentary. This does require some more work at the beginning, but it will pay out when troubleshooting, commissioning or when altering the program.

Instruction

The smallest part of the user program is the instruction, which is written in an instruction line.



An instruction line comprises several elements:

Address
All instructions are assigned an address in the form of a consecutive number. The controller carries out the instructions in the order of these addresses. After the last address, that is after one program cycle, the controller starts again with the first instruction.

Instruction
The instruction consists of operation and operand.

Operation
The operation indicates what is to be done.

Operand
The operand indicates with what an operation is to be carried out. The operand consists of an operand code (+ extension) and operand parameter.

Instruction

The parameter is composed of station number and bit number.

Operand code
(Extension)
Parameter

Operand: I (B) 0.0

Bit number
Station number or word address (marker)

The extension of the operand code indicates the data type used:

No extension:	data type "bit"	e.g. I0.0
Extension B:	data type "byte"	e.g. IB0.0
Extension W:	data type "word"	e.g. IW0

Bit
A bit is the smallest unit of information. It contains the logical states "1" or "0" (on or off) and is used for the fast sequencing of inputs and outputs.

Byte
A byte is a combination of eight bits. This means that it can have the logical states "1" or "0" eight times. It is used for processing analogue values and for arithmetic operations. A byte represents the decimal values between 0 and 255 (2^8).

Word
A word is a combination of 16 bit (2 byte). It can have the logical state "1" or "0" 16 times. It is used in modules or in marker words. A word represents decimal values between 0 and 65535 (2^{16}).

Operand comment
The operand comment should relate to the operand itself and not to the function of the program line. The text that has been written after the operand's first call up appears again each time the operand is input somewhere else.

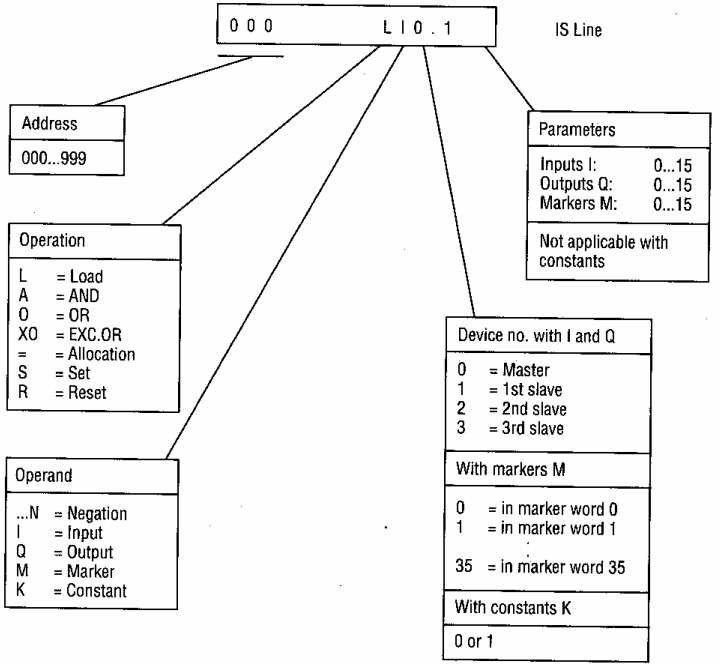


Fig. 2: Variations in a bit instruction

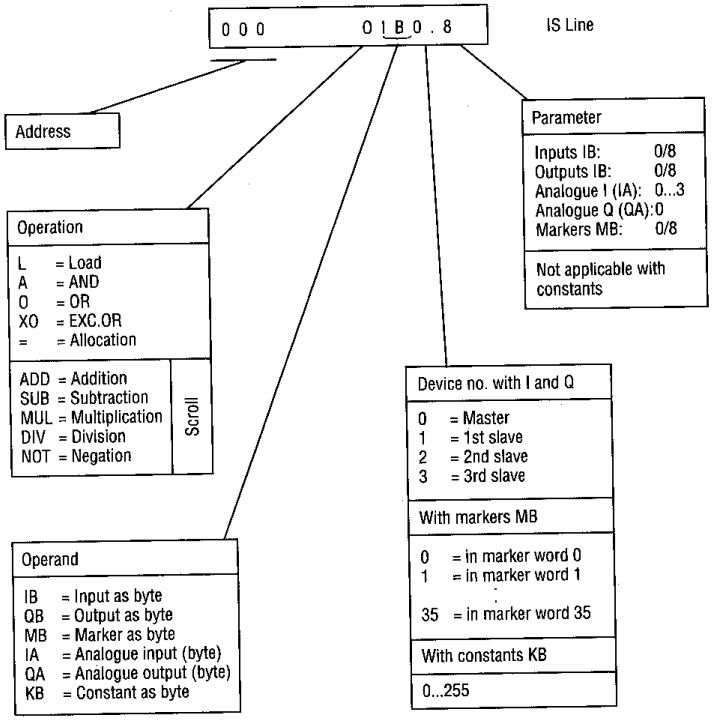


Fig. 3: Variations in a byte instruction

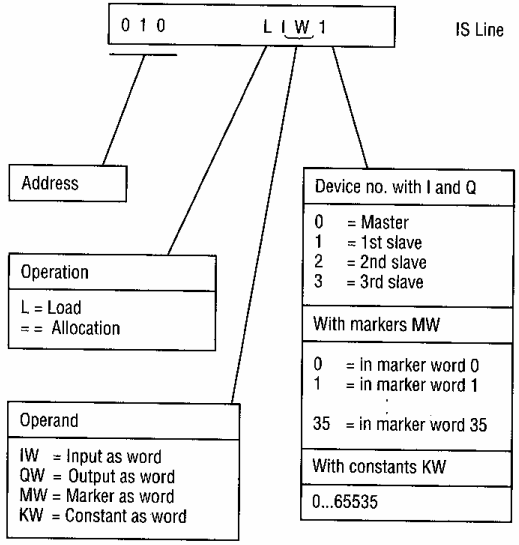


Fig. 4: Variations in a word instruction

Sequence

A sequence consists of several command lines, which must satisfy certain rules. The first command line of a sequence must contain a Load instruction (e.g. LI0.0).

The data type (bit, byte, word) of the operand of this Load instruction determines the data type of the entire sequence. The data type cannot be changed within a sequence.

The sequence can be terminated by one or several of the following instructions:

- Allocation (=),
- Set/reset (S/R),
- Jump instruction or
- Branches (JP, JC, JCN, BC, BNC)

Example of a sequence with the "bit" data type

LI0.0	Motor 1 start
ANI0.2	Limit switch
AM2.0	Display motor 1
=Q0.0	Motor 1 runs

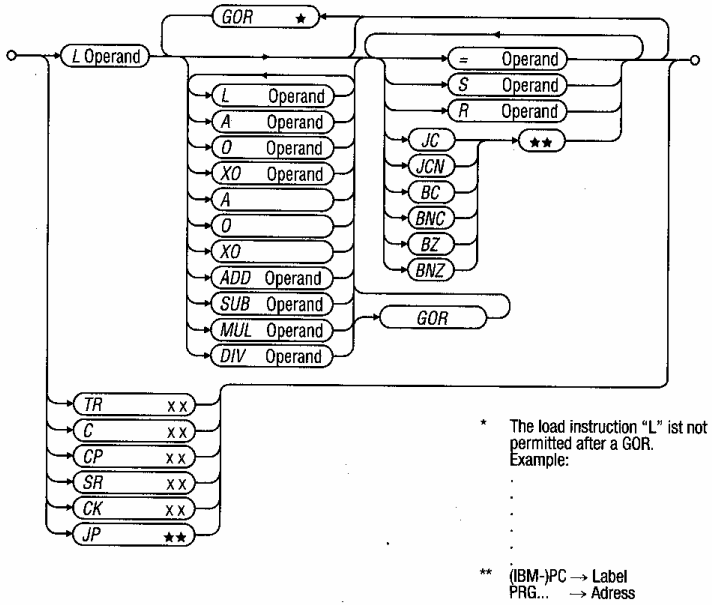


Fig. 5: Structure of a sequence

Block

If several sequences are taken together, e.g. because functionally they belong together, they are called a block.

When programming the PS 4 100 series controller, there are important differences between the block creation procedures for PRG 3 and those for the PC.

Programming

Programming with the PRG 3

If the user program is written with the hand-held programmer PRG 3, the entire user program is to be seen as one block.

Jumps are allowed within the block, but only to the beginning of a sequence.

If a program generated with the PRG 3 is loaded via the PS 4 into a PC, a translation program (discompiler) adapts it to the programming software SUCOsoft S 30-S 3-GB.

Example of a program generated with the PRG 3:

```
000   LI0.1
001   LI0.2
002   O10.3
003   A
004   AI0.4
005   LI0.5
006   O10.6
007   AI0.6
007   AI0.7
008   LI0.8
009   AI0.9
010   O
011   O
012   =Q0.0
```


Programming with the PC

The following example, programmed in instruction set (IS), shows a possible program structure:

```
00000 MOTOR1   "Activation Motor 1
001
002            LI0.0     Start Motor 1
003            OQ0.1     Motor 1
004            AI0.1     Stop Motor 1
005            AI0.2     Bimetal Motor 1
006            ANQ0.2    Motor 2
007            =Q0.1     Motor 1
008
00001 MOTOR2   "Activation Motor 2
001
002            LI0.4     Start Motor 2
003            OQ0.2     Motor 2
004            AI0.3     Stop Motor 2
005            AI0.5     Bimetal Motor 2
006            ANQ0.1    Motor 1
007            =Q0.2     Motor 2
008
00002 END      "
001            EP
```

The user program is divided into blocks. Where and how often a block is opened, as well as block label and block comment is optional.

Block number	Block label	Block comment
00000	Motor 1	Activation Motor 1

The block is introduced by a block number followed by a block label and a block comment. The block label consists of a maximum of eight characters, the first of which has to be a letter. The designation of the label is otherwise fully optional.

A block label must be used, if the block is the target address of a jump or a branch.
Block label and label of the jump instruction have to be the same.

Programming

```
00000 START  .....
001          LI0.5
002          JC HAND
.
.
.
00010 HAND   .....
001          LM1.0
002          0Q0.5
.
.
.
```

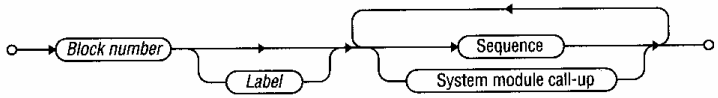


Fig. 6: Block structure

Inputs/outputs

Inputs

The inputs are the interface between the PLC environment and the PS 4 100 series. Via the inputs signals reach the PS 4 100 series controller, where they are processed further.

There are a maximum of eight inputs per PS 4 100 series available for the processing of digital signals. Their image registers can be addressed in bit, byte or word format.

Outputs

The outputs are the interface between the PS 4 100 series and the environment. Via the outputs signals generated in the controller are transmitted to the connected peripherals. The PS 4 100 series has six available outputs.

Unlike the input images, the output images have read-write characteristics. The values in the output images can be read back by the user program for further processing.

Markers

Markers (M) are used for the storage of intermediate results, which occur when processing signals in the PS 4 100 series.

The PS 4 100 series has 36 usable marker words (MW). They can be addressed in bit or byte format.

Of these 36 MW two have special functions and thus are not suitable for the user program (see next chapter).

Constants

When processing complex functions it is often necessary for the PS 4 100 series to process not only variable values, but also constants.

Constants can be sequenced logically, but not allocated. They are used as decimal numbers. Depending on the data type selected the values of the constants are within the following number ranges:

Bit:	K 0 and K 1
Byte:	KB 0 to KB 255
Word:	KW 0 to KW 65535

Registers

The SUCOcontrol PS 4 100 series features several registers which are available to the user and which allow, for example, intermediate values to be stored temporarily or to be sequenced.

These are the working register, the auxiliary register, the status register and the stack register.

Working register

The working register is the most frequently used register. All sequences and operations are carried out within it. Its size is variable. Depending on the operand data type of a sequence the register consists of:

- 1 bit in bit operations
- 8 bit (1 byte) in byte operations, with values from 0 to 255 as integers
- 16 bit (1 word) in word operations, with values from 0 to 65535

Auxiliary register

The auxiliary register is required for some arithmetic operations. The overflow after a multiplication or the remainder after a division is stored in the auxiliary register. These values can be loaded into the working register with the "GOR" instruction and can then be further processed.

Status register

The two-bit status register contains information on the content of the working register (zero bit) and on the result of the preceding arithmetic or logical operation (carry bit).

The individual status bits have the following meaning:

- Carry bit "C"
  It contains the carry of arithmetic operations and is virtually an extension of the working register by 1 bit.

- Zero bit "Z"
  This bit describes the content of the working register.
  It is set if the working register equals zero.
  It is reset (0) if the working register does not equal zero.

The status bits can be scanned by means of conditional jumps both individually and in combinations (with the branches BZ, BNZ).

Stack register

Since all instructions are processed sequentially, sometimes intermediate results of operations occur which are stored in a stack register. Stack registers are temporary memories. They enable multiple load instructions within a sequence.

The stack register functions as a LIFO (last in, first out) shift register. The contents of the stack register are sequenced with the contents of the working register by logical or arithmetic instructions. At the end of a sequence the stack register has to be empty.

Register

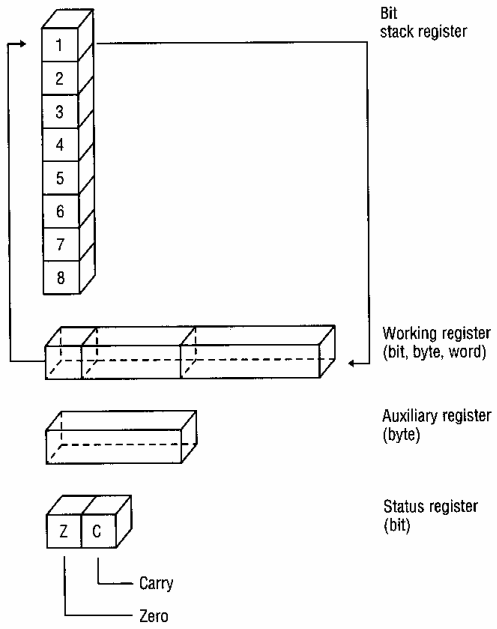


Fig. 7: Stack register

3 Special Features of Programming the PS 4 100 Series

A few features of the PS 4 100 series which influence the programming should be noted. Especially the inputs and outputs of the PS 4 100 series play an important role here.

Inputs/outputs

The user may choose between a bit, byte or word structure for processing inputs and outputs. It should be noted that the controller has only eight digital inputs and six digital outputs. This will mean that the most significant byte IB0.8 of IW0 will always have the value 0. When writing the output word QW0, the complete value is written to the output image register of the PS 4 100 series, and can be read back. Only the six least significant bits are physically output. This procedure applies, when byte processing, to byte IB0.8 and QB0.8. The analogue inputs and outputs are treated similarly as the digital inputs and outputs.

Special markers

The PS 4 100 series features two special marker words. Most functions of these special markers are described in the relevant chapters. But for greater simplification an overview of all functions of the special markers is given here.

Marker word 34 contains several different functions:

M34.0 – M34.3:	Error marker of the high-speed counter (see chapter "Diagnostics")
M34.4 – M34.7:	not specified
M34.8 – M34.11:	Output error marker (see chapter "Diagnostics")
M34.12 – M34.13:	Retentive marker (see chapter "Configuration" in AWB 27-1157-GB)
M34.14 – M34.15:	Clock marker

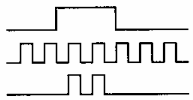
Special markers

Markers 34.0 to 34.13 are described in the chapters mentioned. The clock markers 34.14 and 34.15 are described below.

The PS 4 100 series has two clock markers that can be integrated directly into the user program. Clock marker 34.14 has an interval of 100 ms and clock marker 34.15 an interval of 2 s. The pulse ratio is 1:1

Marker 34.14 100 ms
Marker 34.15 2 s

Example:



Word marker 35 is described in chapter "Configuration" in AWB 27-1157-GB. It is responsible for the retentive features of the system modules.

Diagnostics

In addition to those errors described in chapter "Operation" in AWB 27-1157-GB, which disrupt the correct running of the user program, such as

- system errors
- programming errors
- cycle errors
- bus errors

there are also errors which are indicated. These errors occur when the program is running, but allow it to continue. They are sent to the user program in the form of indication markers. The indication markers show the user the internal states of the controller for further processing. There are two indication markers:

Output short-circuit

A short-circuit (overload) on a transistor output has been recognized. For a direct diagnosis in this case, the "Q-Fault" LED would flash. This message appears also in the PS status menu when using a PRG. The message disappears after correcting the error.
These are the relevant indication markers:

- M34. 8 Short-circuit in basic unit (master)
- M34. 9 Short-circuit in first expander rack (slave)
- M34. 10 Short-circuit in second expander rack (slave)
- M34. 11 Short-circuit in third expander rack (slave)

High-speed counter

High-speed counter means that the carry between input and output counter has not been detected by the cycle (IW scaler factor error). This message is only indicated by the relevant indication markers:

- M34. 0 Error in C0.0 (basic unit)
- M34. 1 Error in C1.0 (first expander rack)
- M34. 2 Error in C2.0 (second expander rack)
- M34. 3 Error in C3.0 (third expander rack)

Once an error has occurred, the message has to be reset by the user program.

4 Instructions

General

On the following pages you will find a complete list of all PS 4 instructions with descriptions. It includes the valid syntax for IS instructions and memory and time requirements for each instruction.

Cycle time and instruction length in byte are divided into two categories:

Category 1
Category 1 instructions relate to the basic unit and to the processor markers MW0 to MW3.

Category 2
Category 2 instructions relate to the expander racks and to the markers in RAM range MW4 to MW35.

Cycle time and byte length increase with IS instructions which relate to expander racks.

> **Note!**
> The PS 4 100 series supports the data types bit, byte and word. They are processed internally and made available at the output level with the appropriate hardware configuration.

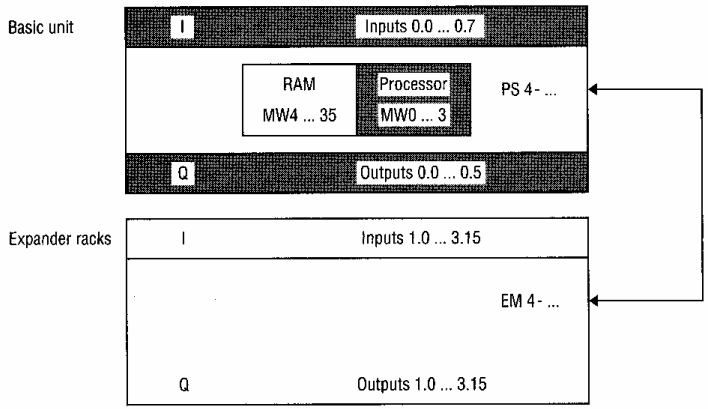


Fig. 8: Basic unit and expander racks

Allocation

Bit	Byte	Word

Description

The contents of the working register are allocated to the operand concerned.

The original value of the operand is overwritten. With allocations to a negative operand, the negated contents of the working register are allocated to the operand.

The operands have bit, byte or word format.
The following operands are permissible:

Outputs: Q, QB, QW
Markers: M, MB, MW

The working register and the auxiliary register remain unchanged by the allocation. The status registers are only changed if an allocation is made to negated operands.

Bit	Byte	Word

Syntax

Data type	Instruction	µs	Byte
Bit	= Q 0.0...0.15	2	2
	= Q 1.0...3.15	7	6
	= M 0.0...0.15	2	2
	= M 4.0...35.15	7	6
	= N Q 0...0.15	4	4
	= N Q 1.0...3.15	8	8
	= N M 0.0...3.15	4	4
	= N M 4.0...35.15	8	8
Byte	= M B 0.0...3.8	1	2
	= M B 4.0...35.8	3	3
	= Q B 0.0...3.8	1	2
	= Q B 0.0...0.8	3	3
	= Q A 0.0...3.0	3	3
Word	= M W 0.3	4	4
	= M W 4...35	8	7
	= Q W 0	4	4
	= Q W 1...3	8	7

Updated conditional bits

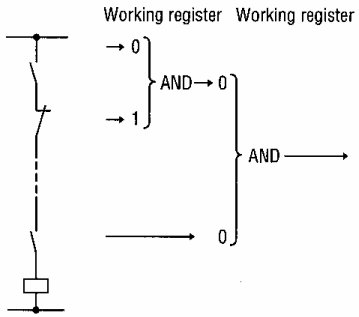
Carry bit (C) — is not changed

Zero bit (Z) — set, if the contents of the working register equal zero after the following negations;
otherwise deleted.

AND

Bit	Byte

Description



AND sequencing of the operand concerned with the contents the working register. The result is stored in the working register. The original contents of the working register are overwritten. The operand is not altered.

When ANDing byte operands, the corresponding bit of both operands involved are sequenced.

An AND sequence is added in the same way to the last value stored in the bit stack register. If a negation is entered then this will influence the contents of the working register, i.e. the last value stored in the stack register is ANDed with the negated contents of the working register.

The operands have bit or byte format. The following operands are permissible:

Inputs: I, IB, IA
Output: Q, QB, QA
Markers: M, MB
Constants: K, KB

The auxiliary register is not altered by the AND sequence.

Bit	Byte

Syntax

Data type	Instruction		µs	Byte
Bit	A I	0.0...0.15	2	2
	A I	1.0...3.15	5	5
	A Q	0.0...0.15	2	2
	A Q	1.0...3.15	5	5
	A M	0.0...3.15	2	2
	A M	4.0...35.15	5	5
	AN I	0.0...0.15	2	2
	AN I	1.0...3.15	5	5
	AN Q	0.0...0.15	2	2
	AN Q	1.0...3.15	5	5
	AN M	0.0...3.15	2	2
	AN M	4.0...35.15	5	5
	AK	0	2	2
	AK	1	1	2
	Sequencing the instruction in the stack register			
	A		2	
	AN		2	
Byte	A I B	0.0...0.8	1	2
	A I B	1.0...3.8	5	7
	A I A	0.0...3.3	5	7
	A M B	0.0...3.8	1	2
	A M B	4.0...35.8	5	7
	A Q B	0.0...0.8	1	2
	A Q B	1.0...3.8	5	7
	A Q A	0.0...3.0	5	7
	A K B	xx	1	2

Updated conditional bits

Carry bit (C)	No meaning for bit operations. Not altered by word operations.
Zero bit (Z)	Set, if the sequence result is zero; otherwise 0.

ADD

Addition

Byte

Description

The operand concerned is added to the contents of the working register, where the result is then stored. The original contents of the working register are overwritten. The operand is not altered.

The values that are added together are interpreted as integers (whole numbers).

The operands have byte format. The following operands are permissible:

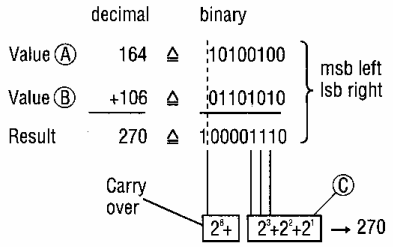
Inputs: IB, IA
Outputs: QB, QA
Markers: MB
Constants: KB

The auxiliary register is not affected by the addition.

Example

A + B = C + Carry over

Addition of 2 values. The result can exceed 255 (> 255).



Byte

Syntax

Data type	Instruction	µs	Byte
Byte	ADD IB 0.0...0.8	1	2
	ADD IB 1.0...3.8	5	7
	ADD IA 0.0...3.3	5	7
	ADD MB 0.0...3.8	1	2
	ADD MB 4.0...35.8	5	7
	ADD QB 0.0...0.8	1	2
	ADD QB 1.0...3.8	5	7
	ADD QA 0.0...3.0	5	7
	ADD KB xx	1	2

Updated conditional bits

Carry bit (C): Set, if there is a carry over, i.e. the result is larger than 8 bits in byte operations; otherwise 0.

Zero bit (Z): Set, if the sequence result is zero; otherwise 0.

B...

Conditional Branches

Byte

Description

BC, BNC, BZ, BNZ[1)]

The contents of the status register are compared with the branching condition. If they agree, the program is continued at the place indicated as the branch target. If the condition is not fulfilled, the branch is not executed. The target for a branch operation must always be:

a) with PRG 3 → Start of sequence

b) with (IBM-)PC → Start of block

Conditional branches are only permissible in byte sequences and only following arithmetic and test operations.

The working, auxiliary and status registers are not effected by branches.

Syntax

Instruction		µs	Byte	Branch condition	Note	
BC	Target	2...4	3	C = 1	Carry bit set	from PRG 3-
BNC	Target	2...4	5	C = 0	Carry bit not set	version 1.2
BZ	Target	2...4	5	Z = 1	RAw empty	
BNZ	Target	2...4	5	Z = 0	RAw not empty	

[1)] B : Branch
C : Carry
N . Not
Z : Zero

Division

Byte

Description

The contents of the working register (dividend) are divided by the operand (divisor) concerned and the result is stored in the working register. Any remainder is stored in the auxiliary register (GOR). The original contents of the working register are overwritten. The operand is not altered.

The values involved in the division are interpreted as positive integers with a data width of 8 bits (1 byte). The division operation can supply the following two results, depending on the dividend and the divisor.

1) If the quotient is within the range of 1...255, the quotient is stored in the working register and the remainder in the auxiliary register (GOR).

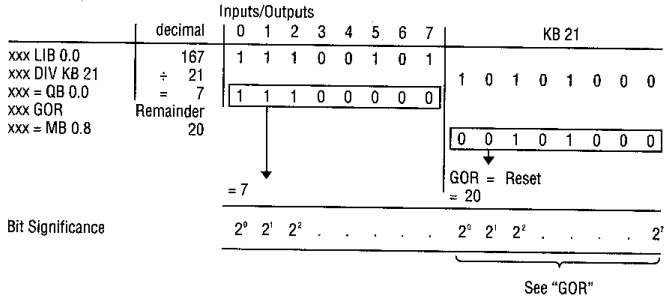
2) If the divisor = 0, the division is aborted. Invalid values are stored in the working register.

The operands have byte format:

Inputs: IB, IA
Outputs: QB, QA
Markers: MB
Constants: KB

Byte

Example:



Syntax

Data type	Instruction		µs	Byte
Byte	DIV IB	0.0...0.1	6	4
	DIV IB	1.0...3.8	6	9
	DIV IA	0.0...3.3	6	9
	DIV MB	0.0...3.8	6	4
	DIV MB	4.0...35.8	6	9
	DIV QB	0.0...0.8	6	4
	DIV QB	0.0...3.8	6	9
	DIV QA	0.0...3.0	6	9
	DIV KB	xx	8	4

Updated conditional bits

Carry bit (C): Low.

Zero bit (Z): Set, if the result or the divisor is zero; otherwise 0.

EP

End of Program

Bit	Byte	Word

The EP instruction tells the program counter that the user program is finished.

If the program is written with a PRG 3, the EP is always set automatically. It is invisible to the user.

If the program is created or recompiled with an (IBM-) PC, the EP instruction must always be programmed in.

Example:

PRG 3

```
000 L I 0.0
  1 A I 0.1
  2 O I 0.2
  3 = Q 0.3
```

(IBM-)PC

```
00001
    1 L I 0.0
    2 A I 0.1
    3 O I 0.2
    4 = Q 0.3
    5 EP
```


GOR

Load Auxiliary Register

Byte

Description

The contents of the auxiliary register are loaded into the working register.

The operation is permissible only in byte sequences and only after multiplication or division.

The contents of the auxiliary register are not altered. The status registers are not affected.

Syntax

Data type	Instruction	µs	Byte
Byte	GOR [1]	1	2

[1] GOR: Get on remainder

JC
JCN

Conditional Jumps

Bit

Description

The contents of the working register are tested and the result of this test is compared with the jump condition. If they agree, the program is continued at the location which is indicated as the jump target. If the condition is not fulfilled, no jump occurs. The jump target must always be

a) with PRG 3 → Sequence beginning

b) with (IBM-)PC → Block beginning

Conditional jumps are only permissible in bit sequences.

The working and the auxiliary registers are not affected by the conditional jumps.

Syntax

Instruction	µs	Byte
JC target	2...4	3
JCN target	2...4	5

Updated conditional bit

Carry bit (C)	Not altered
Zero bit (Z)	Set, if the test result is 0, otherwise low (0).

Unconditional Jump

JP

Bit | Byte | Word

Description

The program is continued at the point where the jump target is given.

a) with PRG 3 → Sequence beginning

b) with (IBM-)PC → Block beginning

The auxiliary and status registers are not affected by the unconditional jump.

Syntax

Instruction	µs	Byte	Jump Condition
JP target	2...4	3	None

Load

Bit	Byte	Word

Description

The value of the operand concerned is loaded into the working register. The original contens of the register are overwritten.

If the Load instruction is within a sequence, i.e. the contents of the working register have not yet been allocated to an operand, the original contents of the working register are stored in a stack register.

The operand is not altered.

The operands have bit, byte or word format.
The following operands are permissible:

Inputs: I, IB, IW, IA
Outputs: Q, QB, QW, QA
Markers: M, MB, MW
Constants: K, KB, KW

The auxiliary register is not altered by the Load instruction. The status registers have no meaning.

Bit	Byte	Word

Syntax

Data type	Instruction		µs	Byte
Bit	Bit sequence beginning			
	LI	0.0...0.15	1	2
	LI	1.0...3.15	4	5
	LQ	0.0...0.15	1	2
	LQ	1.0...3.15	4	5
	LM	0.0...3.15	1	2
	LM	4.0...35.15	4	5
	LNI	0.0...0.15	3	3
	LNI	1.0...3.15	5	7
	LNQ	0.0...0.15	3	3
	LNQ	1.0...3.15	5	7
	LNM	0.0...3.15	3	3
	LNM	4.0...35.15	5	7
	LK	0	1	2
	LK	1	1	2
	With stack operations			
	LI	0.0...0.15	3	4
	LI	1.0...3.15	6	7
	LQ	0.0...0.15	3	4
	LQ	1.0...3.15	6	7
	LM	0.0...3.15	3	4
	LM	4.0...35.15	6	7
	LNI	0.0...0.15	5	5
	LNI	1.0...3.15	7	9
	LNQ	0.0...0.15	5	5
	LNQ	1.0...3.15	7	9
	LNM	0.0...3.15	5	5
	LNM	4.0...35.15	7	9
	LK	0	3	4
	LK	1	3	4
Byte	LIB	0.0...0.8	1	2
	LIB	1.0...3.8	3	3
	LIA	0.0...3.3	3	3
	LMB	0.0...3.8	1	2
	LMB	4.0...35.8	3	3
	LQB	0.0...0.8	1	2
	LQB	1.0...3.8	3	3
	LQA	0.0...3.0	3	3
	LKB	xx	1	2
Word	LIW	0	2	4
	LIW	1...3	8	7
	LMW	0...3	2	4
	LMW	4...35	8	7
	LQW	0	2	4
	LQW	1...3	8	7
	LKW	xx	4	4

MUL

Multiplication

Byte

Description

The operand concerned is multiplied by the contents of the working register and the product is then stored there. The original contents of the working register are overwritten. The operand is not altered.

The values involved in the multiplication are interpreted as positive integers.

The multiplication of two 8-bit values can produce a 16-bit number as a result. The least significant byte of the result is stored in the working register whilst the most significant byte is stored in the auxiliary register. This so-called overflow can be processed with the "GOR" instruction.

The operands have byte format. The following operands are permissible:

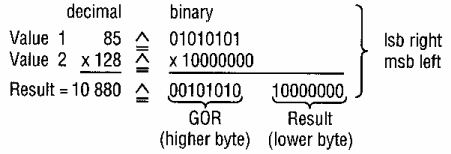
Inputs: IA, IB
Outputs: QA, QB
Markers: MB
Constants: KB

Byte

Example:

85 x 128 = 10880

The GOR instruction should be used if the result of the multiplication is greater or also less than 255.



The maximum possible result is 255 x 255 = 65 025.

Syntax

Data type	Instruction	µs	Byte
Byte	MUL IB 0.0...0.8	6	4
	MUL IB 1.0...3.8	8	6
	MUL IA 0.0...3.3	8	6
	MUL MB 0.0...3.8	6	4
	MUL MB 4.0...35.8	8	6
	MUL QB 0.0...0.8	6	4
	MUL QB 1.0...3.8	8	6
	MUL QA 0.0...3.0	8	6
	MUL KB xx	6	4

Updated conditional bit

Carry bit (C): Set, if the result is greater than 255, i.e. the result is greater than one byte: otherwise low (only for ADD and SUB). With MUL and DIV the carry bit is not affected.

Zero bit (Z): Set, if the test result is 0; otherwise low (0).

NOP

No Operation

Bit	Byte	Word

Description

NOP is a processor instruction. It adds one more cycle (1 µs) to the program. Each NOP lengthens the cycle time of the program.
NOP is permissible in bit, byte and word sequences.

Observe the following when using NOP:

```
Example:  L I 0.0   }
          O I 0.1   } Sequence
          = Q 0.5   }
          NOP
          NOP
          L I 0.4   }
          O Q 0.5   } Sequence
          = Q 0.6   }
```

NOP after a module is recognized as an error when compiling!

```
Example:  TR 0
          S:
          STP:
          EQ:
          NOP   }
          NOP   } not allowed!
          L I 0.4
          .
          .
          .
```


Negation

NOT

Byte

Description

The contents of the working register are negated, i.e. the one's complement is formed. The new content of the working register therefore consists of the inverted bits of the original contents.

The operation may be carried out in byte sequences.

The auxiliary register is not affected by the negation.

Syntax

Data type	Instruction	µs	Byte
Byte	NOT	1	1

Updated conditional bit

Carry bit (C)	Not altered.
Zero bit (Z)	Set, if the test result is 0, otherwise low (0).

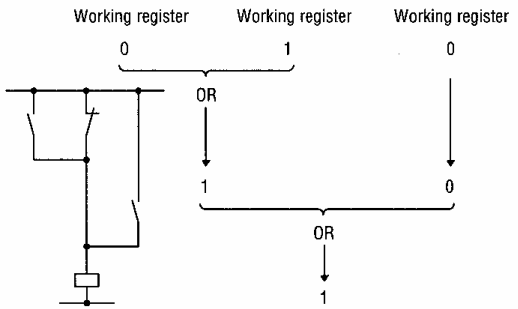
Example

00110110 → 11001001 (one's complement)

OR

Bit	Byte

Description



OR sequencing of the operand concerned with the contents of the working register. The result is stored in the working register. The original contents of the working register are overwritten. The operand is not altered.

In an OR sequence with byte operands, the corresponding bits of each operand involved are sequenced.

The OR sequencing of the last value stored in the stack register is carried out in the same way. If a negation is used, this affects the contents of the working register, i.e. the last value stored in the stack register is paralleled (ORed) with the negated contents of the working register.

The operands have bit or byte format. The following operands are permissible:

Inputs: I, IB, IA
Outputs: Q, QB, QA
Markers: M, MB
Constants: K, KB

The auxiliary register is not changed by the OR sequence.

Bit	Byte

Syntax

Data type	Instruction		µs	Byte
Bit	OI	0.0...0.15	2	2
	OI	1.0...3.15	5	5
	OQ	0.0...0.15	2	2
	OQ	1.0...3.15	5	5
	OM	0.0...3.15	2	2
	OM	4.0...35.15	5	5
	ONI	0.0...0.15	2	2
	ONI	1.0...3.15	5	5
	ONQ	0.0...0.15	2	2
	ONQ	1.0...3.15	5	5
	ONM	0.0...3.15	2	2
	ONM	4.0...35.15	5	5
	OK	0	1	2
	OK	1	1	1
	Sequencing the instruction in stack register			
	O		2	2
	ON		3	3
Byte	OIB	0.0...0.8	1	2
	OIB	1.0...3.8	5	7
	OIA	0.0...3.3	5	7
	OMB	0.0...3.8	1	2
	OMB	44.0...35.8	5	7
	OQB	0.0...0.8	1	2
	OQB	1.0...3.8	5	7
	OQA	0.0...3.0	5	7
	OKB	xx	1	2

Updated conditional bit

Carry bit (C)	No meaning for bit operations; is not changed by byte operations
Zero bit (Z)	Set, if the sequence result is 0; otherwise low (0).

R

Reset

Bit

Description

The bit of the operand concerned is reset, if the contents of the working register equal "1". If this resetting condition is not fulfilled, the operand is not altered.

The operation is only permissible for bit sequences.

The operands have bit format. The following operands are permissible:

Outputs: Q

Markers: M

The working and auxiliary registers are not altered. The status register is altered by the reset condition.

Syntax

Data type	Instruction	µs	Byte
Bit	RM 0.0...3.15	2...3	4
	RM 4.0...35.15	2...8	8
	RQ 0.0...0.15	2...8	4
	RQ 1.0...3.15	2...8	8

Updated conditional bit

Carry bit (C): Not altered.

Zero bit (Z): Set, if the working register result is 0; otherwise low (0).

Set

S

Bit

Description

The bit of the operand concerned is set if the contents of the working register equal "1". If the setting condition is not fulfilled, the operand is not altered.

The operands have bit format. The following operands are permissible:

Outputs: Q

Markers: M

The working and auxiliary registers are not altered. The status register is altered by the setting condition.

Syntax

Data type	Instruction	µs	Byte
Bit	SM 0.0...3.15	2...3	4
	SM 4.0...35.15	2...8	8
	SQ 0.0...0.15	2...8	4
	SQ 1.0...3.15	2...8	8

Updated conditional bit

Carry bit (C): Not altered.

Zero bit (Z): Set, if the working register result is 0; otherwise low (0).

SUB

Subtraction

Byte

Description

The operand (subtrahend) concerned is subtracted from the contents of the working register (minuend) and the result is stored in the working register. The original contents of the working register are overwritten. The operand is not altered.

The values involved in the subtraction are interpreted as positive integers.

The operands have byte format. The following operands are permissible:

Inputs:	IB, IA
Outputs:	QB, QA
Markers:	MB
Constants:	KB

The auxiliary register is not influenced by the subtraction.

Example:

A – B = C

Subtraction of two values. The result is not less than 0.

	decimal		binary	
Value Ⓐ	7	≙	00000111	lsb right
Value Ⓑ	-5	≙	-00000101	
Result Ⓒ	2	≙	00000010	msb left

$2^1 \rightarrow 2$

Byte

Syntax

Data type	Instruction		µs	Byte
Byte	SUB IB	0.0...0.8	2	3
	SUB IB	1.0...3.8	7	10
	SUB IA	0.0...3.3	7	10
	SUB MB	0.0...3.8	2	3
	SUB MB	4.0...35.8	7	10
	SUB QB	0.0...0.8	2	3
	SUB QB	1.0...3.8	7	10
	SUB QA	0.0...3.0	7	10
	SUB KB	xx	2	3

Updated conditional bit

Carry bit (C)
Set, if a so-called borrow bit is needed, i.e. the minuend is smaller than the subtrahend;
low (0), if a carry from the most significant bit of the result has occurred.

Zero bit (Z)
Set, if the working register result is 0;
otherwise low (0).

XO

Exclusive OR

Bit	Byte

Description

Exclusive OR sequencing of the operand concerned with the contents of the working register, where the result is then stored. The original contents of the working register are overwritten. The operand is not altered.

In exclusive OR sequencing of byte operands the relevant bits of every operand involved are sequenced.

The exclusive OR sequencing of the operand with the last value stored in the stack register is done in the same manner. If a negation is inserted here, it will affect the contents of the working register, i.e. the last value stored in the stack register is ORed with the negated contents of the working register.

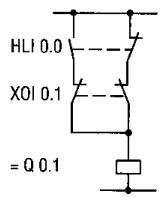
The operands have bit, or byte format. The following operands are permissible:

Inputs: I, IB, IA
Outputs: Q, QB, QA
Markers: M, MB
Constants: K, KB

The auxiliary register is not altered by the Exclusive OR sequence.

Bit	Byte

Example:



Syntax

Data type	Instruction		µs	Byte
Bit	XOI	0.0...0.15	3...4	4
	XOI	1.0...3.15		7
	XOQ	0.0...0.15	3...4	4
	XOQ	1.0...3.15		7
	XOM	0.0...3.15	3...4	4
	XOM	4.0...35.15		7
	XONI	0.0...0.15	3...4	4
	XONI	1.0...3.15		7
	XONQ	0.0...0.15	3...4	4
	XONQ	1.0...3.15		7
	XONM	0.0...3.15	3...4	4
	XONM	4.0...35.15		7
	XOK	0	1	2
	XOK	1	1	2
	Sequence the instruction in the stack register			
	XO		3...4	4
	XON		3...4	4
Byte	XOIB	0.0...0.8	1	2
	XOIB	0.0...3.8	1	7
	XOIA	0.0...3.8	5	7
	XOMB	1.0...3.8	1	2
	XOMB	4.0...35.8	5	7
	XOQB	0.0...0.8	1	2
	XOQB	1.0...3.8	5	7
	XOQA	0.0...3.0	5	7
	XOKB	xx	1	2

Updated conditional bit

Carry bit (C)	No meaning for bit operations; is not altered by byte operations.
Zero bit (Z)	Set, if the working register result is 0; otherwise low (0).

5 System Modules

The PS 4 100 series offers 32 counters, 32 shift registers, 32 timers and 32 comparators as system modules.

- All modules are self-contained units (sequences).
- No further sequences are permitted within the modules. If with each module step just the Return key is pressed, the input or the output is automatically set to 0.

It is possible to scroll up and down within a module. Only downward scrolling is possible outside the module. When scrolling up, the display always jumps to the beginning of the module.

Modules		Module No.
C	Counter	0-31
TR	Timer	0-31
SR	Shift register	0-31
CP	Comparator	0-31

Alls modules can be connected in series (are cascadable).

The setting or modification of a module is only then complete, when all input and output parameters have been set and/or have simply been skipped by pressing the Return key and when the next memory address appears on the display.

The retentive features can be set for a maximum of four modules:

MB 35.0 contains the retentive parameters.

Default = 00 00 00 00 (no retentive features).

Code:

00 no retentive system modules
01 retentive timers (TR)
10 retentive counters (C)
11 retentive shift registers (SR)

Example: 11 01 01 01 = 3 x TR + 1 x SR = 213

L KB 213
= MB 35.0

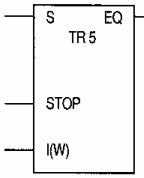
The order of the retentive parameters is optional.

Syntax

Call-up
TR <Module No.>

Number ranges:	0.1...6553.5 s
Module No.:	0...31
Cascadable:	Yes
Retentive:	Yes
Time base:	0.1 s

Display

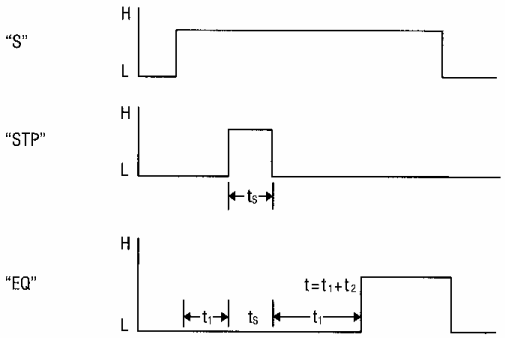
LD (IBM)-PC	**IS** PRG 3	**IS** (IBM)-PC
	xxx TR5 TR 5 ⋆ S:_ TR 5 ⋆ STP:_ TR 5 ⋆ IW:_ TR 5 ⋆ EQ:_	TR 5 [ ] S: [ ] STOP: [W] I: [W] EQ:

Key to inputs and outputs

		Data format
S	Start and set	Bit
STOP	Time interruption	Bit
I	Reference time value	Word
EQ	Control output	Bit

PRG...: Actual time value can be displayed in the "SHOW" menu
(IBM-)PC: Actual value **cannot** be displayed

Time diagram



Description

TR 0...31 are on delayed timers with a stop input. When there is a rising edge at the "S" input, the delay factor at "I" is transfered. The "EQ" output is set to 1 after the time "t".
The delay times results from:

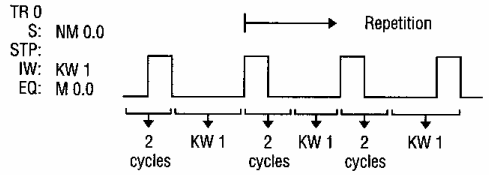
t = delay factor x 0.1 s

The delay factor can be between 1 and 65535. When the "Stop" input is set to 1, the time can be interrupted; i.e. the delay time "t" is extended by the time "t_s" for which the stop input is set to 1. If the signal level at the "S" input is switched from 1 to 0, the module is reset.

The basic pulse of 100 ms used for the time generation means that the timers are accurate to 100 ms.

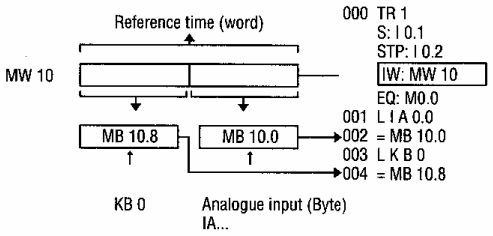
Examples

1) Pulse transmitter with a timer:

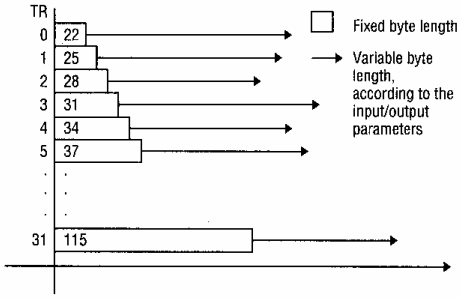


2) "Timing relay" with potentiometer:

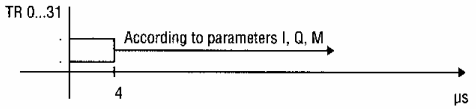
Time range 0...25.5s



Memory requirements



Processing time



Example: TR4

		Variable bytes	Fixed bytes	Processing time
TR4			34	4
TR4*S:	I0.2	2	+2	1
TR4*STP:	I3.5	5	+2	4
TR4*IW:	MW5	7	–	8
TR4*EQ:	Q0.8	2	+2	2

Σ 56 Bytes Σ 19 µs
+EP 4 "
60 "

Word

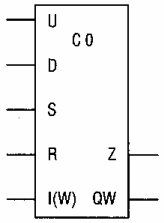
Syntax

Call-up
C <Module No.>

Number range:	0...65535
Max. counting frequency:	2 x cycle time
Module No.:	0...31
Cascadable:	Yes
Retentive:	Yes

Display

LD
(IBM)-PC



IS
PRG 3

```
xxx C_
C0 * U:_
C0 * C:_
C0 * S:_
C0 * R:_
C0 * IW:_
C0 * Z:_
C0 * QW:_
```

IS
(IBM)-PC

```
   C 0
[ ] U:
[ ] D:
[ ] S:
[ ] R:
[W] I:
[ ] Z:
[W] Q:
```

Key to inputs and outputs

		Data format
U	Up pulse	(Bit)
D	Down pulse	(Bit)
S	Set	(Bit)
R	Reset	(Bit)
I	Set value input	(Word)
Z	Count zero	(Bit)
Q	Count output	(Word)

Truth table

Input Data type Function	U Bit	D Bit	S Bit	R Bit	I Word
Count up	⌐¬	0	0	0	x
Count down	0	⌐¬	0	0	x
Set	x	x	⌐¬	0	Value
Delete	x	x	x	1	x

Word

Description

When there is a rising edge at the "S" input, the value at the "I" input is transfered to the counter. The counter moves forward when there is a rising edge at the "U" input and is reversed when there is a rising edge at the "D" input.

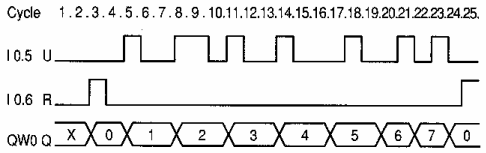
When the "R" input is set to 1, the counter is reset and the contents deleted. The "Z" output of the counter is 1, if the counter contents are 0. The "Q" output shows the actual count value.

Example

Counter 11 is to count one step further each time I 0.5 closes. I 0.6 resets the counter. The actual count is displayed via the QW 0 output word. Down counting and setting are not used.

The program:

```
C 11
   U:  I0.5        Up pulse
   D:
   S:
   R:  I0.6        Reset
   I:
   Z:
   Q:  QE 0        Actual value
```

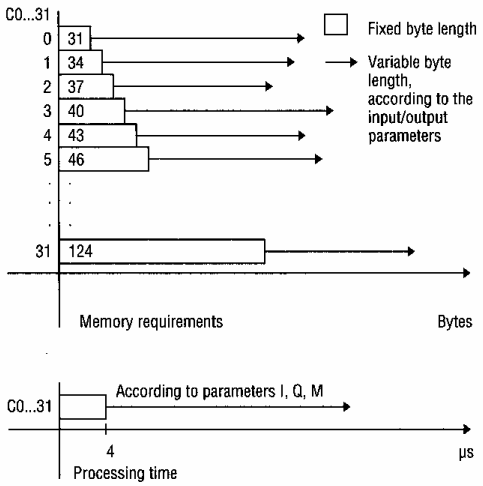


Word

Note!
The pulse must be at least the length of a cycle time, so that the counter can recognize every counting pulse. The counter also has to recognize, with certainty, a subsequent 0 phase of the signal in order to form the next rising edge automatically. The maximum counting frequency is therefore

$$f = \frac{1}{2 \cdot t_{cycle}} \quad \text{[Pulse duty cycle 1 : 1]}$$

Memory requirements and processing times



Word

Example

		Variable bytes	Fixed bytes	Processing time
010 C15			76	4
C15★U:	I0.1	2	+2	1
C15★D:	I0.2	5	+2	4
C15★S:	(NOP)		+1	1
C15★R:	M0.14	2	+2	2
C15★IW:	MW10	7	–	8
C15★Z:	Q0.4	2	+2	2
C15★QW:	QW1	7	+2	8
		Σ 112 Bytes		Σ 30 µs
		+ EP 4 "		
		116 "		

Word

Syntax

Call up:
C<Module No.>

Input number range:	0...65 535
Output number range:	0...65 535
Module No.:	0.0...3.0
Retentive:	Yes

Display

LD (IBM-)PC	**IS** PRG 3	**IS** (IBM)-PC
C...0	xxx C-.0	C-.0
	C-.0 ⋆ S:-	[ ] S:
S/R QW	C-.0 ⋆ IW:-	[W] I:
	C-.0 ⋆ QW:-	[W] Q:
I(W)		

Key to inputs and outputs

	Data type
S/R set/reset input	Bit
Reference value input	Word (up to PRG 3 Vers. 1.0 = constant)
Reference value output	Word

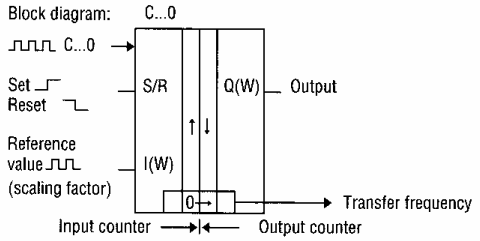
Truth table

Input	S/R	I	3 kHZ
Set	⌐	Value	x
Reset	¬	0	0

Word

Description

The high-speed counter input functions as pre-scaler. With the aid of this module, the PS 4 100 series are able to detect pulses which are faster than the cycle time. When there is a rising edge at the "S/R" input, the reference value (constant) is set at the "I(W)" input. This value is then decremented to "0" by the incoming signals. Then the counter is again loaded with the set value. This input counter is a ring counter which increments the output counter of the module by "1" every time it reaches "0". The actual value of the output can be called up via "Q(W)".



The actual value **cannot** be made visible.

Word

Example

Display	Input	Data type	Example
xxx C_	0.0...3.0	–	0.0
C0.0 ⋆ S:_	Set/reset input	Bit	I 0.1
C0.0 ⋆ IW:_	Reference value input	Constant	KW 5000
C0.0 ⋆ QW:_	Reference value output	Word	MW0

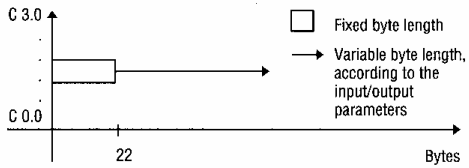
xxx+1_ Continuation of the IS program

Note:

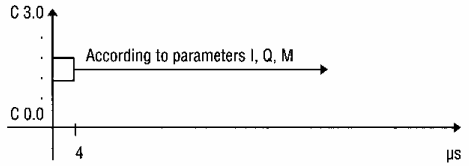
Input frequency ⎍⎍⎍⎍ = Transfer frequency
Scaling factor I(W)

The cycle frequency of the PS user program must always be greater than the transfer frequency between input counter and output counter.

Memory requirements



Processing time



Word

Example

		Variable bytes	Fixed bytes	Processing time
C0.0			22	4
C0.0⋆S:	I1.0	5	+2	4
C0.0⋆MW:	MW 5	7	–	8
C0.0⋆QW:	MW 0	4	+2	4
		Σ	42 Bytes	Σ 20 µs
		+EP	4 "	
			46 "	

Note:
The high-speed counter offers in addition error indication. If the transfer between input counter and output counter is not detected by the cycle, the M 34.0...34.3 marker bits are set to "1".

C 0.0	→	M 34.0
C 1.0	→	M 34.1
C 2.0	→	M 34.2
C 3.0	→	M 34.3

These markers are reset via the "S/R" module input.

CP

Comparator

Word

Syntax

Call-up:
CP <Module No.>

Number ranges:	0...65535
Modules No.:	0...31
Retentive:	Yes

Display

LD (IBM)-PC	**IS** PRG 3	**IS** (IBM)-PC
WI1 GT CP 7 EQ WI2 LT	xxx CP 7 CP 7 * I1W:_ CP 7 * I2W:_ CP 7 * GT:_ CP 7 * EQ:_ CP 7 * LT:_	CP 7 [W] I1: [W] I2: [ ] GT: [ ] EQ: [ ] LT:

Key to inputs and outputs

	Data format		
I1	Value 1	Value	
I2	Value 2	Word	
GT	I1 > 12	Bit	GT = Greater than
EQ	I1 = 12	Bit	EQ = Equal
LT	I1 < 12	Bit	LT = Less than

Truth table

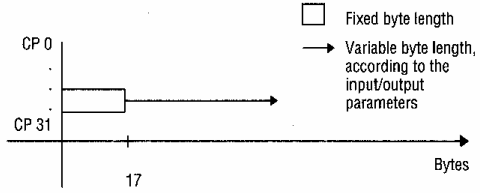
I1 < I2	GT = 0	EQ = 0	LT = 1
I1 = I2	GT = 0	EQ = 1	LT = 0
I1 > I2	GT = 1	EQ = 0	LT = 0

Description

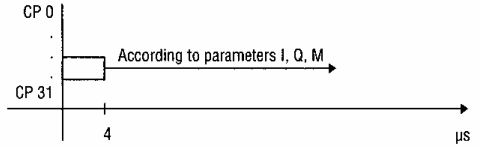
The module compares the values at the "I1" and "I2" word inputs and then sets the outputs according to the truth table.

Word

Memory requirements



Processing time



Example

		Variable bytes	Fixed bytes	Processing time
xxx CP 20			17	4
CP 20*I1W:	KW 2000	4	+5	4
CP 20*I2W:	IW 1	7	–	8
CP 20*GT:	Q 0.10	2	+2	2
CP 20*EQ:	M 20.6	6	+2	8
CP 20*LT:	Q 0.11	2	+2	2
		Σ 49 Bytes		Σ 28 µs
		+EP 4 "		
		53 "		

Bit

Syntax

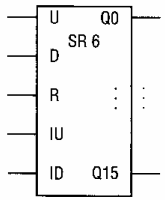
Call-up:

SR <Module No.>

Register length:	16
Module No.:	0...31
Cascadable:	Yes
Retentive:	Yes

Display

LD (IBM)-PC



IS PRG 3

```
xxx SR 6
SR 6 * U:_
SR 6 * U:_
SR 6 * U:_
SR 6 * U:_
SR 6 * U:_
SR 6 * U:_
.
.
.
SR 6 * Q15:_
```

IS (IBM)-PC

```
SR 6
[ ] U:
[ ] D:
[ ] R:
[ ] IU:
[ ] ID:
[ ] Q0:
.
.
.
[ ] Q15:
```

Key to inputs and outputs

		Data format
U	Pulse input forward	Bit
D	Pulse input reverse	Bit
R	Reset	Bit
IU	Data input forward	Bit
ID	Data input reverse	Bit
Q0	Output 0	Bit
Q15	Last output	Bit

Truth table

Inputs Type Function	U Bit	D Bit	R Bit	UI Bit	ID Bit
Shift forward	⌐¬	0	0	Value	x
Shift reverse	0	⌐¬	0	x	Value
Delete	x	x	1	x	x

Bit

Description

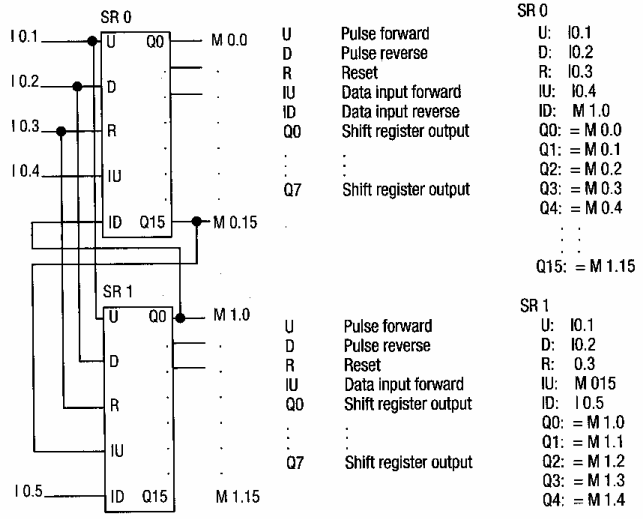
Shift register, 1-bit wide, with fixed register length. When there is a rising edge at the "U" input, the value of the "IU" input is transferred to the first register field, after all the other register fields have been shifted by one step in the positive direction. A rising edge at the "D" input causes the transfer of the "ID" value to the last register field, after the contents of all the other register fields have been shifted by one step in the negative direction.

The contents of all the register fields are externally displayed via the "Q" outputs. When the "R" input is set to 1, the shift register is reset and all the register fields are deleted.

The register length is limited to 16 register fields. Several shift registers can be cascaded if more than 16 shift steps are required.

Bit

Example



IS

U	Pulse forward
D	Pulse reverse
R	Reset
IU	Data input forward
ID	Data input reverse
Q0	Shift register output
.	.
.	.
.	.
Q7	Shift register output

SR 0

U: I0.1
D: I0.2
R: I0.3
IU: I0.4
ID: M 1.0
Q0: = M 0.0
Q1: = M 0.1
Q2: = M 0.2
Q3: = M 0.3
Q4: = M 0.4
. .
. .
. .
Q15: = M 1.15

U	Pulse forward
D	Pulse reverse
R	Reset
IU	Data input forward
Q0	Shift register output
.	.
.	.
.	.
Q7	Shift register output

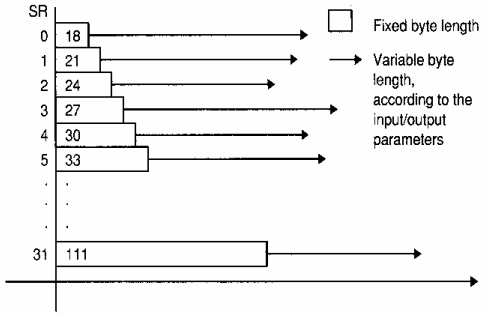
SR 1

U: I0.1
D: I0.2
R: 0.3
IU: M 015
ID: I 0.5
Q0: = M 1.0
Q1: = M 1.1
Q2: = M 1.2
Q3: = M 1.3
Q4: = M 1.4
. .
. .
. .
Q15: = M 1.15

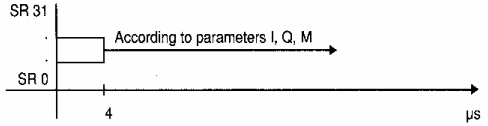
Cascading of two 16 step long bit shift registers to form one 32 step long bit shift register.

Bit

Memory requirements



Processing time



SR

Shift Register

Bit

Example: SR 8

Processing time			Variable bytes	Fixed bytes
xxx SR8			42	4
SR8*U:	I13.1	5	+2	4
SR8*D:	M0.10	2	+2	1
SR8*R:	M0.1	2	+2	1
SR8*IU:	M10.5	5	+2	4
SR8*ID:	M10.6	5	+2	4
SR8*Q0:	Q1.0	6	+2	7
SR8*Q1:	Q1.1	6	+2	7
SR8*Q2:	Q1.2	6	+2	7
SR8*Q3:	Q1.3	6	+2	7
SR8*Q4:	Q1.4	6	+2	7
SR8*Q5:	M0.1	2	+1	1
SR8*Q6:	– (NOP)	–	+1	1
.		.	.	.
.		.	.	.
.		.	.	.
.		.	.	.
SR8*Q15:	– (NOP)	–	+1	1

Σ 125 bytes Σ 64 µs
+EP 4 "
129 "

Note:
There is a "NOP" for each unassigned input or output.

Appendix

Index