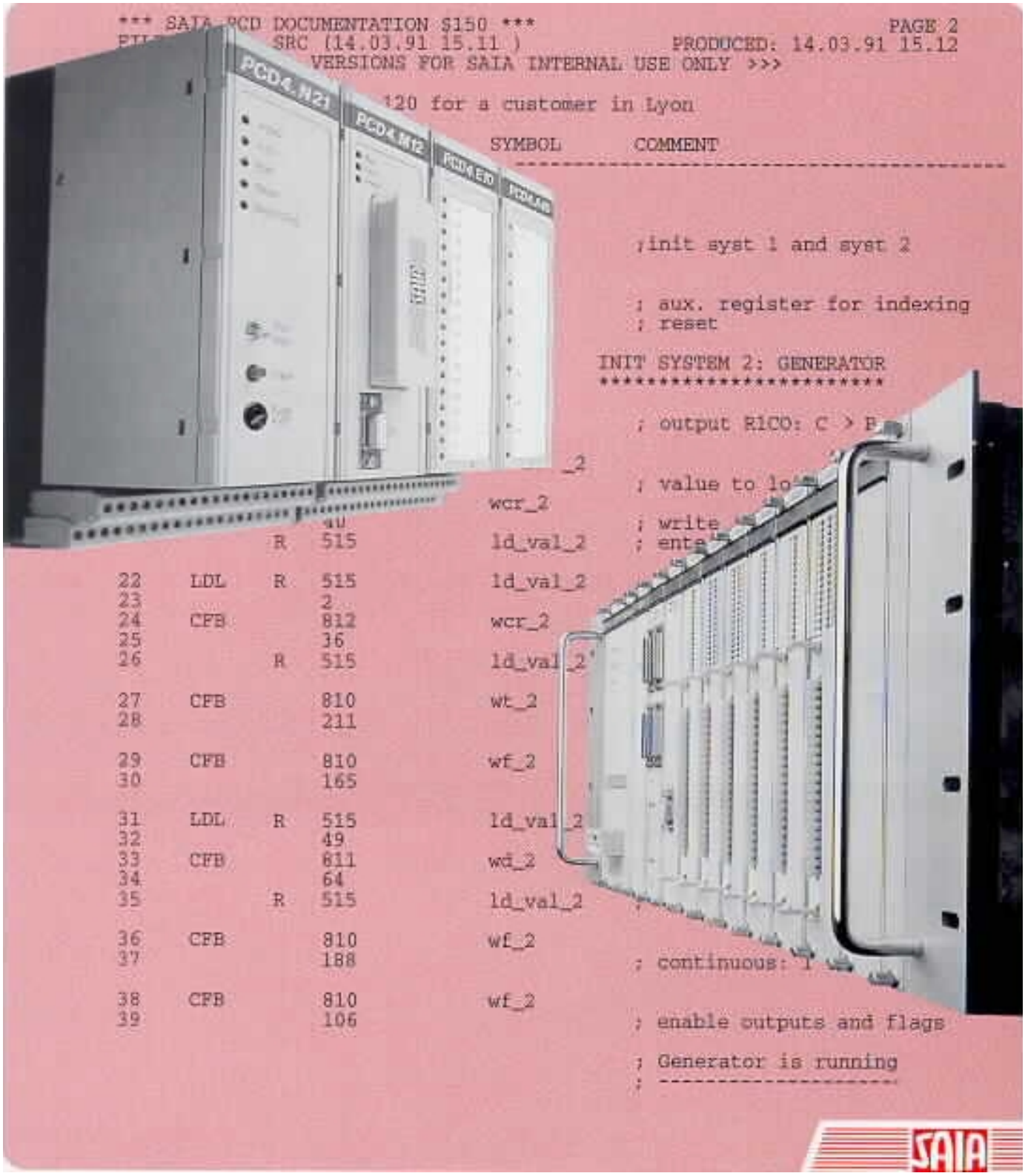
SAIA-Burgess Electronics

SWITCHES - MOTORS - CONTROLLERS

SAIA® PCD

Process Control Devices

Reference Guide SAIA® PCD



English edition 26/733 E6

SAIA-Burgess Electronics Ltd.
Bahnhofstrasse 18
CH-3280 Murten (Switzerland)
http;//www.saia-burgess.com

BA: Electronic Controllers Telephone 026 / 672 71 11
Telefax 026 / 670 44 43

SAIA-Burgess Companies

Switzerland SAIA-Burgess Electronics AG
Freiburgstrasse 33
CH-3280 Murten
☎ 026 672 77 77, Fax 026 670 19 83

Germany SAIA-Burgess Electronics GmbH
Daimlerstrasse 1k
D-63303 Dreieich
☎ 06103 89 060, Fax 06103 89 06 66

Austria SAIA-Burgess Electronics Ges.m.b.H.
Schallmooser Hauptstrasse 38
A-5020 Salzburg
☎ 0662 88 49 10, Fax 0662 88 49 10 11

Italy SAIA-Burgess Electronics S.r.l.
Via Cadamosto 3
I-20094 Corsico MI
☎ 02 48 69 21, Fax 02 48 60 06 92

France SAIA-Burgess Electronics Sàrl.
10, Bld. Louise Michel
F-92230 Gennevilliers
☎ 01 46 88 07 70, Fax 01 46 88 07 99

Nederlands SAIA-Burgess Electronics B.V.
Hanzeweg 12c
NL-2803 MC Gouda
☎ 0182 54 31 54, Fax 0182 54 31 51

Belgium SAIA-Burgess Electronics Belgium
Avenue Roi Albert 1er, 50
B-1780 Wemmel
☎ 02 456 06 20, Fax 02 460 50 44

Hungary SAIA-Burgess Electronics Automation Kft.
Liget utca 1.
H-2040 Budaörs
☎ 23 501 170, Fax 23 501 180

Representatives

Great Britain Canham Controls Ltd.
25 Fenlake Business Centre, Fengate
Peterborough PE1 5BQ UK
☎ 01733 89 44 89, Fax 01733 89 44 88

Denmark Malthe Winje Automation AB
Hovedgaden 60-62
DK-3630 Jaegerpris
☎ 70 20 52 01, Fax 70 20 52 02

Norway Malthe Winje Automasjon AS
Haukelivn 48
N-1415 Oppegård
☎ 66 99 61 00, Fax 66 99 61 01

Sweden Malthe Winje Automation AB
Truckvägen 14A
S-194 52 Upplands Våsby
☎ 08 795 59 10, Fax 08 795 59 20

Suomi/ Finland ENERGEL OY
Atomitie 1
FIN-00370 Helsinki
☎ 09 586 2066, Fax 09 586 2046

Portugal INFOCONTROL Electronica e Automatismo LDA.
Praceta Cesário Verde, No 10 s/cv, Massamá
P-2745 Queluz
☎ 21 430 08 24, Fax 21 430 08 04

Spain Tecnosistemas Medioambientales, S.L.
Poligono Industrial El Cabril, 9
E-28864 Ajalvir, Madrid
☎ 91 884 47 93, Fax 91 884 40 72

Czech Republic ICS Industrie Control Service, s.r.o.
Modranská 43
CZ-14700 Praha 4
☎ 2 44 06 22 79, Fax 2 44 46 08 57

Poland SABUR Ltd.
ul. Druzynowa 3A
PL-02-590 Warszawa
☎ 22 844 63 70, Fax 22 844 75 20

Australia Siemens Building Technologies Pty. Ltd.
Landis & Staefa Division
411 Ferntree Gully Road
AUS-Mount Waverley, 3149 Victoria
☎ 3 9544 2322, Fax 3 9550 9260

Argentina MURTEN S.r.l.
Av. del Libertador 184, 4° “A”
RA-1001 Buenos Aires
☎ 054 11 4312 0172, Fax 054 11 4312 0172

After sales service

USA SAIA-Burgess Electronics Inc.
1335 Barclay Boulevard
Buffalo Grove, IL 60089, USA
☎ 847 215 96 00, Fax 847 215 96 06

Issue : 01.02.2000

Subjet to change without notice

SAIA® Process Control Devices

Manual

Reference Guide

SAIA® PCD

Edition 26/733 E6 - updated: 04.2000

Subject to technical changes

Updates

Manual : Reference Guide SAIA PCD - Edition E6

Date	Chapter	Page	Description
04.2000	6	6-5 .. 6-8	XOB: div. corrections, XOB 6 added
04.2000	6	6-18 .. 6-19	SCOB: old and new
04.2000	8	8-18/19	SASI: $ is accepted
04.2000	8	8-48	SOCL: switch RS 485 and RS 422
04.2000	12	12-14 .. 12.17	SYSRD: div. corrections, read real time
04.2000	12	12-18 .. 12-22	SYSWR: div. corrections, write real time

Table of contents

Page

3. WORD Instructions

Page

4. INTEGER arithmetic

5. FLOATING POINT arithmetic

Page

6. BLOCTEC Instructions

7. GRAFTEC Instructions

Page

8. SERIAL COMMUNICATIONS Instructions

Page

9. LAN2 Instructions

10. CONTROL Instructions

11. DEFINITION Instructions

12. SPECIAL Instructions

13. History List

Alphabetical instruction list

Instr.	Ind	Pm	Page
ACC			2-11
ADD		=	4-3
ALGI	X	=	12-24
ALGO	X	=	12-25
AND	X	=	3-33
ANH	X	=	2-5
ANL	X	=	2-6
BITI	X	=	3-25
BITIR	X	=	3-26
BITO	X	=	3-27
BITOR	X	=	3-28
CCOB			6-20
CFB			6-16
CMP	X	=	4-8
COB			6-3
COM	X	=	2-16
COPY	X	=	3-15
CPB			6-12
CPBI			6-13
CSB			7-5
DEC	X	=	3-8
DEFTB			11-5
DEFTC			11-4
DEFTR			11-6
DEFVM			11-3
DEFWPH			11-8
DEFWPR			11-7
DEI		=	3-11
DIAG			12-13
DIGI	X	=	3-29
DIGIR	X	=	3-30
DIGO	X	=	3-31
DIGOR	X	=	3-32
DIV		=	4-6
DSP			3-6
DYN	X	=	2-12
ECOB			6-4
EFB			6-15
EPB			6-11
ESB			7-4
EST			7-9
ETR			7-11
EXOB			6-9
EXOR	X	=	3-35
FABS	X	=	5-16
FADD			5-5
FATAN			5-13
FB		=	6-14
FCMP	X		5-10
FCOS		=	5-12
FDIV		=	5-8
FEXP		=	5-14
FLN		=	5-15
FMUL		=	5-7

Instr	Ind	Pm	Page
FPI	X	=	5-4
FSIN	X	=	5-11
FSQR		=	5-9
FSUB		=	5-6
GET	X	=	3-16
HALT			10-6
IFP	X	=	5-3
INC	X	=	3-7
INI		=	3-10
IST			7-7
JPD			10-4
JPI			10-5
JR			10-3
LD	X		3-3
LDH	X	=	3-5
LDL	X	=	3-4
LOCK			10-7
LRXD	X	=	9-3
LRXS	X	=	9-5
LTXD	X	=	9-4
LTXS	X	=	9-6
MOV	X	=	3-14
MUL		=	4-5
NCOB			6-17
NOP			12-3
NOT	X	=	3-36
OR	X	=	3-34
ORH	X	=	2-7
ORL	X	=	2-9
OUT	X	=	2-13
OUTS	X	=	12-27
PB			6-10
PID			12-6
PUT	X		3-19
RCOB			6-21
RES	X	=	2-15
RESD	X		2-18
ROTD		=	3-40
ROTL	X	=	3-43
ROTR	X	=	3-44
ROTU		=	3-39
RSB			7-6
RSI		=	3-13
RTIME		=	12-4
SASI		=	8-8
SASII		=	8-20
SB			7-3
SCOB			6-19
SCON		=	8-49
SCONI		=	8-51
SEI		=	3-9
SET	X	=	2-14
SETD	X		2-17
SHID		=	3-38

Instr.	Ind	Pm	Page
SHIL	X	=	3-41
SHIR	X	=	3-42
SHIU		=	3-37
SICL		=	8-46
SOCL		=	8-47
SQR		=	4-7
SRXD	X	=	8-21
SRXM		=	8-32
SRXMI		=	8-37
ST			7-8
STH	X	=	2-3
STHS	X	=	12-26
STI		=	3-12
STL	X	=	2-4
STXD	X	=	8-22
STXM		=	8-39
STXMI		=	8-44
STXT	X	=	8-23
SUB		=	4-4
SYSCMP		=	12-23
SYSRD		=	12-14
SYSWR		=	12-18
TEST			12-10
TR			7-10
TFR	X	=	3-21
TFRI	X	=	3-23
UNLOCK			10-8
WTIME		=	12-5
XOB			6-5
XOR	X	=	2-10

Legend:

"Ind" column: a 'X' indicates that the instruction can be indexed.

"Pm" column: an '=' indicates that the instruction accept a FB parameter as operand

Notes

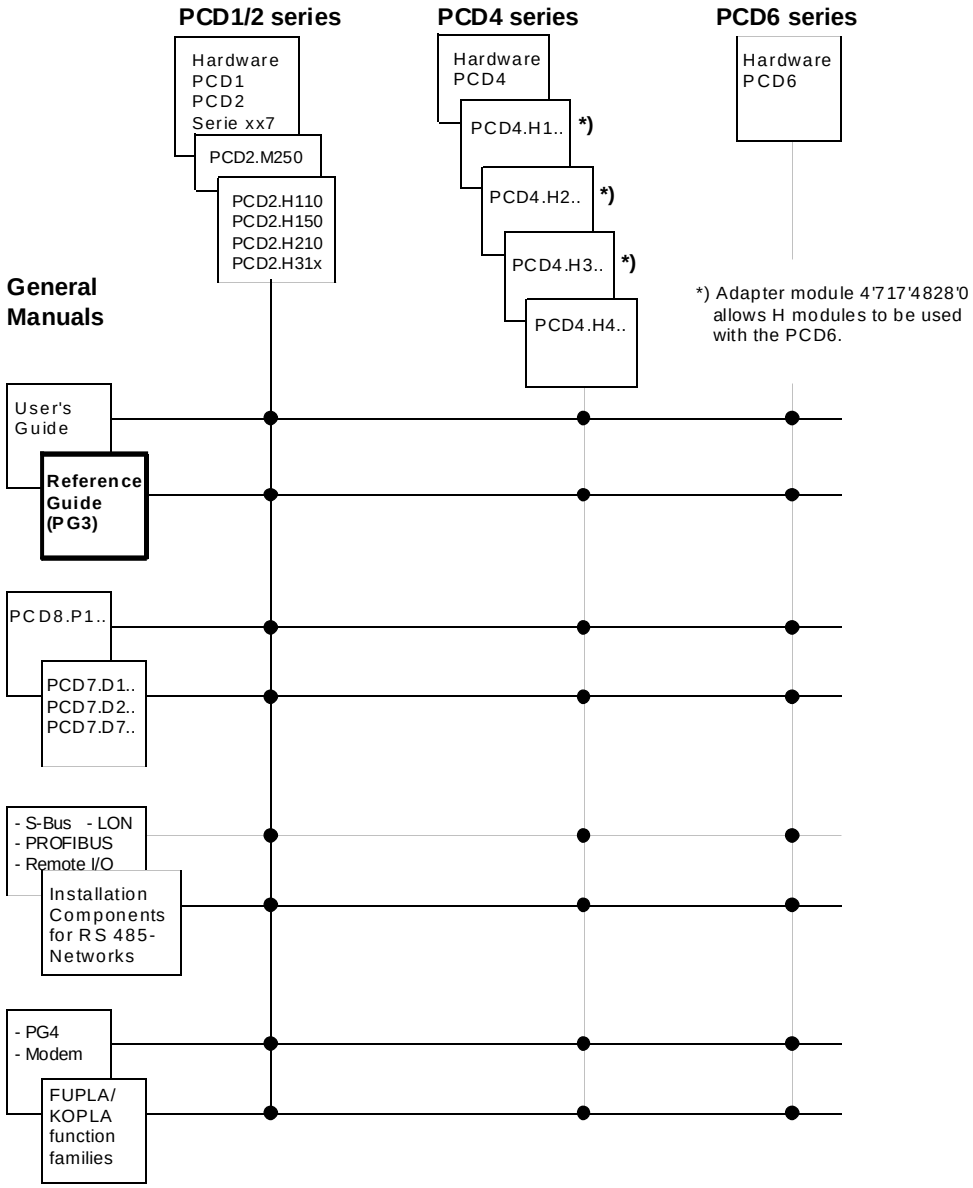
Please note:

A number of detailed manuals are available to aid installation and operation of the SAIA PCD. These are for use by technically qualified staff, who may also have successfully completed one of our "workshops".

To obtain the best performance from your SAIA PCD, closely follow the guidelines for assembly, wiring, programming and commissioning given in these manuals. In this way, you will also become one of the many enthusiastic SAIA PCD users.

If you have any technical suggestions or recommendations for improvements to the manuals, please let us know. A form is provided on the last page of this manual for your comments.

Summary



Reliability and safety of electronic controllers

SAIA-Burgess Electronics Ltd. is a company which devotes the greatest care to the design, development and manufacture of its products:

- state-of-the-art technology
- compliance with standards
- ISO 9001 certification
- international approvals: e.g. Germanischer Lloyd, United Laboratories (UL), Det Norske Veritas, CE mark ...
- choice of high-quality componentry
- quality control checks at various stages of production
- in-circuit tests
- run-in (burn-in at 85°C for 48h)

Despite every care, the excellent quality which results from this does have its limits. It is therefore necessary, for example, to reckon with the natural failure of components. For this reason SAIA-Burgess Electronics Ltd. provides a guarantee according to the "General terms and conditions of supply".

The plant engineer must in turn also contribute his share to the reliable operation of an installation. He is therefore responsible for ensuring that controller use conforms to the technical data and that no excessive stresses are placed on it, e.g. with regard to temperature ranges, overvoltages and noise fields or mechanical stresses.

In addition, the plant engineer is also responsible for ensuring that a faulty product in no case leads to personal injury or even death, nor to the damage or destruction of property. The relevant safety regulations should always be observed. Dangerous faults must be recognized by additional measures and any consequences prevented. For example, outputs which are important for safety should lead back to inputs and be monitored from software. Consistent use should be made of the diagnostic elements of the PCD, such as the watchdog, exception organization blocks (XOB) and test or diagnostic instructions.

If all these points are taken into consideration, the SAIA PCD will provide you with a modern, safe programmable controller to control, regulate and monitor your installation with reliability for many years.

1. Introduction

This Reference Guide describes in detail each instruction in the SAIA PCD family. The instructions are grouped together by type of instruction to facilitate their learning.

This guide is intented as a support to the "User's Guide" which gives a detailed description of the "PCD Utilities" and the structured programming methods used in the SAIA PCD.

Important
The instructions described in this guide are valid for:

PCD1 version V001 (and above)
PCD2 version V004 (and above)
PCD4.Mxx0 version V005 (and above)
PCD4.Mxx5 version V00C (and above)
PCD6.M5 version V004 (and above)
PCD6.M1/M2 version V009 (and above)
PCD Utilities version V2.0

If you have a PCD with an older version certain instructions can differ or be non-existant.

How to read this manual

One or more pages are provided for each instruction of the SAIA PCD.

Top Line For each instruction, the mnemonic(s) and the instruction name are shown on the top line.

Description Describes what the instruction does and its operand.

Usage Shows how the instruction is used and gives the type and range of each operand.
An "[X]" after the mnemonic means that indexed addressing is possible by adding the optional 'X' to the mnemonic (eg: STHX, INCX).
For indexed addressing, the indexed operand(s) is marked with a "(i)".

Example A typical example of the instruction.

Flags Shows which Status flags are affected (ACCU, N, P, Z, E)

See also A list of other instructions or topics which may be useful.

Practice A practical example which shows the use of the instruction in a suitable context.

Typographic Conventions:

[ ]	Square brackets in text enclose optional input or data. For example: [;comments] means that ";comments" is optional and need not be present.
[X]	An "[X]" after the mnemonic means that indexed addressing is possible by adding the optional 'X' to the mnemonic (eg: STHX, INCX).
(i)	When indexed addressing is possible (see [X]); the indexed operand(s) is marked with a "(i)"
.....	A series of "." in an example shows that this one can be continued by yourself.
LABEL:	In the examples, all the labels are represented with their name followed by a ':'; this is necessary if you use another editor than SEDIT. If you use SEDIT do not put this ':' after the label.
< >	Angle brackets enclose texts or expression which should not be typed verbatim, but replaced by any text or valid expression.

1.1 MEDIUM code (MC)

Medium control codes [mc] are used to select the element type

Code	Type	Range
I	Input	0..8191
O	Output	0..8191
F	Flag	0..8191
R	Register	0..4095
T	Timer	0..450
C	Counter	0..1599
K	Constant	0..16383
X	teXt	0..7999
DB	Data Block	0..7999

Inputs and Outputs

The inputs and outputs in the form of interface modules can be plugged into the PCD, the address range can be assigned as required with the position of the module in PCD1/2/4 or with DIP switches in PCD6.
Input states can only be interrogated.
Outputs can be set (switched on) and reset (switched off) and can also be interrogated as to the signal states.

Flags

Flags are 1 bit storage cells which can be treated like outputs, e.g. they can be set or reset, and can be interrogated as to their state. Accordingly flags are used for the storage of any suitable information.

Timers and Counters

Timers and Counters are programmable registers, they can hold a 31 bits value (0 - 2.147.483.647 in decimal). They share the same address range from 0-1599; the number of timers depends on the instruction DEFTC (the default value is 32 timers from addresses 0 to 31 and 1568 counters from addresses 32 to 1599)
The only difference between a timer and a counter is that a timer is decremented according to the time-base (defined by the instruction DEFTB, default value is 1/10 sec)
Timers and counters can only hold positive or nul values.

When a Timer or Counter contains a non-zero value its state is High (H), when its content is zero its state is Low (L)

Registers

A register is a 32 bits storage cell which can hold any information, in binary, decimal, hexadecimal, floating-point. You can perform arithmetical operations on them, transfer of information (from or to: inputs, outputs, flags, timers, counters, registers).
Registers can hold positive as well as negative values.

Constants

Several types of constant are used (see next page).

teXt

Texts are ASCII strings that can be memorized in the PCD for output on a serial line.

Data Block

A "Data Block" is an area of memory which is used for storing 32-bit data, which can be transfered to and from Registers, Timers and Counters

1.2 Constants

There are several types of constants used in PCD instructions, the valid range depends on the instruction.
K constants are NOT allowed for the LOAD instructions (LD, LDH and LDL).

Binary

End the value with a 'Q', eg. 1001Q, 11111111Q. Max 12 bits.

Decimal

Default, no special format.

Hexadecimal

End the hex value with an 'H', eg. ABCDH, 1234H, DEADH.

ASCII

Enclose the ASCII character in inverted commas, eg. 'A', 'z'.

Floating point

Include a '.' and/or an exponent 'E', eg. 1.2, 12E-1.
Range (FFP) is 2.710505E-20 to - 9.223371E+18.

K Constants

Used where a medium control code (mc) is required; this is never used for the LOAD instructions (LD, LDL, LDH)
To show it is a constant, the mc type 'K' is used.
Binary, decimal, hexadecimal or Ascii values can be given, eg. K 10, K 11Q, K FFH, K 'A',

Constant Type	Minimum value	Maximum value
Decimal	- 2.147.483.648	2.147.483.647
Hexadecimal	0 H	FFFFFFFF H
Binary	0 Q	1111111111111 Q
ASCII	'a', 'B', '%', etc.	
Floating point	- 9.22337177 E+18 - 2.71050535 E -20	+9.22337177 E+18 +5.42101070 E -20

1.3 The Condition Codes [cc]

Arithmetic Status Flags

The arithmetic status flags are affected mostly by the Integer and the floating point instructions.

The Error flag can be set High (and Low) by any instruction which is executed with invalid data.

P	Positive	High if result of arithmetic instruction is positive
N	Negative	High if result of arithmetic instruction is negative (the P flag is always the inverse of the N flag)
Z	Zero	High if the result of arithmetic instruction is 0
E	Error	High if an instruction cannot be executed (set High on Overflow, Underflow or Conversion)

Accumulator

Since the processor "reads" one instruction line after the other, it follows, that it can only interrogate one element after the other as to its signal state (H or L). In order to process a complete linkage (Linkage: section of program consisting of several instruction lines; it normally begins with a start instruction and ends with an action instruction. Each line in a linkage depends on the result of the previous one) up to an action instruction, the existing intermediate result of the linkage must be stored in the Accumulator (ACCU).
At the end of the linkage the end result is present in the ACCU (0 or 1). On the basis of this result the corresponding element (e.g. an output) is either not activated (ACCU = 0) or activated (ACCU = 1)

The ACCUmulator is set High/Low (1 or 0) mostly by the Bit instructions. It can be set to a specific state, or to the state of an arithmetic status flag, using the ACC instruction.

Conditions Codes [CCs]

Condition codes [cc] are used to select the Status flags condition for the execution of the instruction.
If the condition is false, the instruction is not executed.
Note: Certain instructions are Accumulator dependent, and are executed only in the Accumulator (ACCU) is High (1).

blank	No condition code
H	If Accumulator = H (1)
L	If Accumulator = L (0)
P	If Positive flag = H (Negative flag = L)
N	If Negative flag = H
Z	If Zero flag = H
E	If Error flag = H
(**C**)	Complement used with the ACC instruction only

1.4 Resource types and values

Type	Description		Range	Remark	
I	Input	Share same addresses	0..8191		Per System
O	Output		0..8191		
F	Flag		0..8191	Volatile/Non volatile	
T	Timer	Share same addresses	0..450	Volatile	
C	Counter		0..1599	Non Volatile	
R	Register		0..4095	Non Volatile	
K	K constant		0..16383		
COB	Cyclic Organisation Block		0..15		Per CPU
XOB	Exception Organisation Block		0..31		
PB	Program Block		0..299		
FB	Function Block		0..999		
SB	Sequential Block		0..31		
IST	Initial Step		0..1999		
ST	Step		0..1999		
TR	Transition		0..1999		
X	Text	Share same addresses	0..7999	4000..7999 in extended memory	
DB	Data Block		0..7999		
-	Semaphore		0..99		Per System

Notes

2. BIT Instructions

Bit instructions work with the Accumulator, Inputs, Outputs, Flags and the state of Timers or Counters.

STH	Start high
STL	Start low
ANH	AND high
ANL	AND low
ORH	OR high
ORL	OR low
XOR	Exclusive OR
ACC	Accumulator operations
DYN	Dynamic (edge detection)
OUT	Set element from ACCU
SET	Set element
RES	Reset element
COM	Complement element
SETD	Set element delayed
RESD	Reset element delayed

Notes

STH START HIGH

Description The ACCU is set to the logical state of the addressed element.
This is the start of a new linkage line.
The previous linkage results are cleared with the start instruction; simultaneously the signal state "H" of the addressed element I, O, F, T, C will be read and the result stored in the ACCU.

Usage

STH[X]	**element**	**(i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

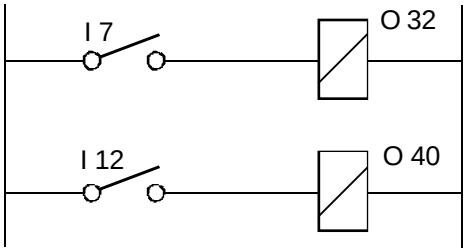
Example STH I 7 ; ACCU = State of I 7

Flags ACCU set to state of addressed element.

Note If a timer or counter contains 0 its state is defined L, otherwise its state is H

See Also STHS, STL.

Practice



; A minimum program in the PCD must consist of one COB

```
COB       0     ; COB header
          0
STH       I 7   ; If input 7 is H
OUT       O 32  ;    Then set output 32
                ; Else reset output 32
STH       I 12  ; If input 12 is H
OUT       O 40  ;    Then set output 40
                ; Else reset output 40
ECOB            ; END of COB
```


STL START LOW

Description The ACCU is set to the inverted logical state of the addressed element. This is the start of a new linkage line.
The previous linkage results are cleared with the start instruction; simultaneously the signal state "L" of the addressed element I, O, F, T, C will be read, inverted and the result stored in the ACCU.

Usage

STL[X]	**element**	**(i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

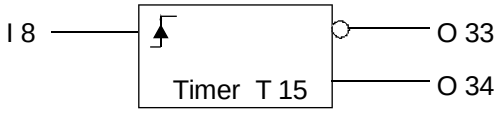
Example

```
STL     I 9     ; ACCU = inverted state of I 9
```

Flags **ACCU** set to the inverted state of the addressed element.

See Also STH.

Practice




```
COB         0       ; COB header
            0

STH       I 8       ; If Input 8 becomes high
DYN       F 10
LD        T 15      ;     Then load timer with 2 sec
            20

STL       T 15      ; If the time is elapsed
OUT       O 33      ;     Then set output 33
                    ;  Else reset output 33

STH       T 15      ; If the time is not elapsed
OUT       O 34      ;     Then set output 34
                    ;  Else reset output 34

ECOB                ; End of COB
```


ANH AND HIGH

Description The ACCU is AND linked with the logical state of the addressed element, the ACCU is set to the result.

Usage

ANH[X]	**element**	**(i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

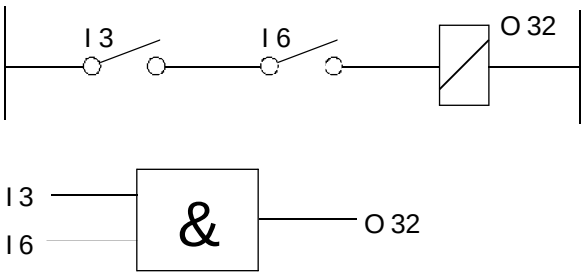
Example

```
ANH     I 3     ; ANDs ACCU with the state of Input 3
ANHX    I 128   ; ANDs ACCU with Input (128+Index)
```

Flags **ACCU** set according to result.

See Also ANL.

Practice




```
COB         0       ; COB header
            0

STH       I 3       ; If input 3 is H
ANH       I 6       ;         and input 6 is H
OUT       O 32      ;     Then set output 32
                    ; Else reset output 32

ECOB                ; END of COB
```


ANL AND LOW

Description The ACCU is AND linked with the inverted logical state of the addressed element, the ACCU is set to the result.

Usage

ANL[X]	**element**	**(i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

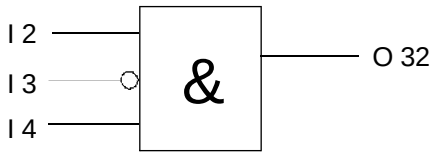
Example

```
ANL     I 4     ; ANDs ACCU with inverted Input 4 state.
ANHX    I 128   ; ANDs ACCU with inverted Input (128+Index)
```

Flags **ACCU** set according to result.

See Also ANH.

Practice




```
COB         0       ; COB header
            0

STH       I 2       ; If input 2 is H
ANL       I 3       ;     AND input 3 is L
ANH       I 4       ;     AND input 4 is H
OUT       O 32      ;  Then set output 32
                    ; Else reset output 32

ECOB                ; End of COB
```


ORH OR HIGH

Description The ACCU is OR linked with the logical state of the addressed element, and the ACCU is set to the result.
OR instructions are used for parallel linkages of elements.
The total linkage begins with a start (STH or STL); each additional parallel partial linkage begins with an ORH.
If a parallel partial linkage is detected as successful (ACCU=1) then the logical states of the following partial linkages no longer exercise any influence on the result of this total linkage.

Usage

ORH[X]	**element**	**(i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

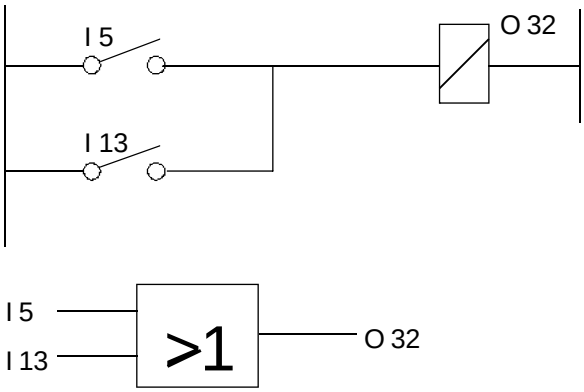
Example

```
STH     I 5     ; If I 5 is H
ORH     I 13    ;   OR I 13 is H
                ; Then ACCU = 1, Else ACCU = 0
```

Flags **ACCU** set to the result.

See also ORL.

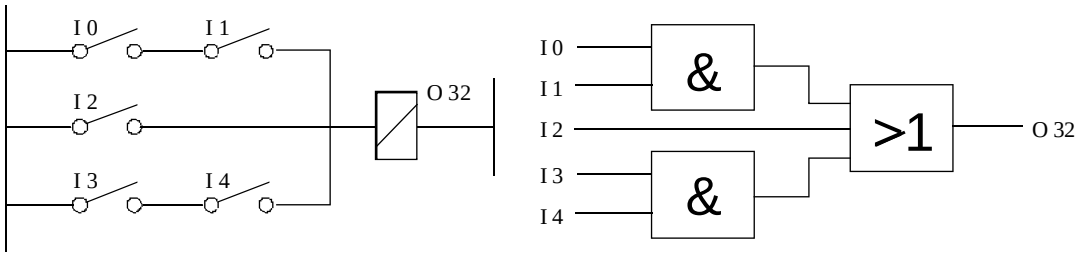
Practice




```
COB         0       ; COB header
            0

STH       I 5       ; If input 5 is H
ORH       I 13      ;          or input 13 is H
OUT       O 32      ;     Then set output 32
                    ;  Else reset output 32

ECOB                ; END of COB
```

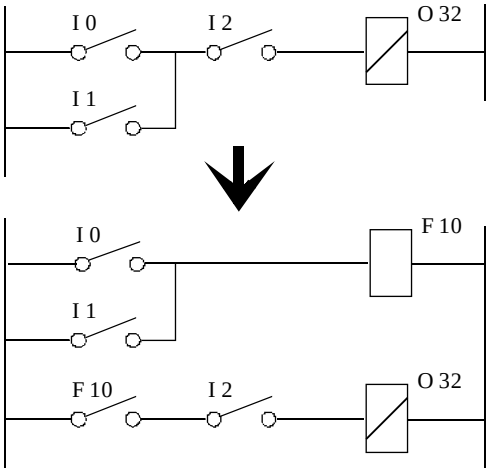




```
COB         0       ; COB Header
            0

STH       I 0       ; If input 0 is H
ANH       I 1       ;     and input 1 is H
ORH       I 2       ;     or input 2 is H
ORH       I 3       ;     or input 3 is H
ANH       I 4       ;     and input 4 is H
OUT       O 32      ;  Then set output 32
                    ; Else reset output 32

ECOB                ; END of COB
```

It can be seen from the above example that the OR- instruction has "priority" over AND.




```
COB         0       ; COB Header
            0

STH       I 0       ; If Input 0 is H
ORH       I 1       ;     or input 1 is H
OUT       F 10      ;  Then set flag 10
                    ; Else reset flag 10
STH       F 10      ; If Flag 10 is H
ANH       I 2       ;     and input 2 is H
OUT       O 32      ;  Then set output 32
                    ; Else reset output 32

ECOB                ; END of COB
```

ORL OR LOW

Description The ACCU is OR linked with the inverted logical state of the addressed element, and the ACCU is set to the result. OR instructions are used for parallel linkages of elements.

Usage

ORL[X]	**element**	**(i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

Example

```
STH     I 3     ; If I 3 is H
ORL     I 7     ;   OR I 7 is L
                ;  Then ACCU = 1, else ACCU = 0
```

Flags ACCU set to the result.

See Also ORH.

XOR EXCLUSIVE OR

Description The ACCU is XOR linked with the logical state of the addressed element, the ACCU is set to the result.
With the XOR instruction the signal states of two elements can be compared with one another. If they are identical, the ACCU content is 0; where they are different, it is 1.

Usage

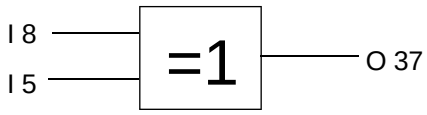
XOR[X]	**element**	**(i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

Example XORI 5; ACCU = ACCU XOR I 5

Flags The ACCU is set to the result.

See Also OR, AND.

Practice




```
COB       0      ; COB Header
          0

STH       I 8    ; If input 8 is H
XOR       I 5    ;          XOR input 5 is H
OUT       O 37   ;     Then set output 37
                 ; Else reset output 37

ECOB             ; END of COB
```

I 8	I 5	O 37
L	L	L
L	H	H
H	L	H
H	H	L

ACC ACCUMULATOR OPERATIONS

Description Modifies the state of the Accumulator according to the code:

C	Complement	Accu is complemented
H	High	Accu is set High (1)
L	Low	Accu is set Low (0)
P	Positive	Accu is set to Positive (P) flag state
N	Negative	Accu is set to Negative (N) flag state
Z	Zero	Accu is set to Zero (Z) flag state
E	Error	Accu is set to Error (E) flag state

The operand cannot be supplied as a Function Block parameter

Usage

```
ACC     code            ; code = C|H|L|P|N|Z|E
```

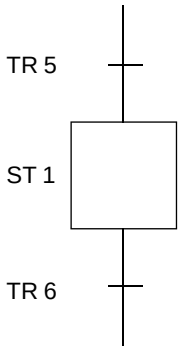
Example

```
ACC     H       ; Sets ACCU to 1
ACC     E       ; Sets ACCU to state of E flag
```

Flags ACCU is modified.

See Also OUT, Condition Codes.

Practice




```
CMP     R 99    ; Compare R 99
        'x'     ; with ASCII character 'x'
ACC     Z       ; If equal Then ACCU is set H
                ;    (copy of Z flag in ACCU)

STXT    1
        10

....
```


DYN DYNAMIC EDGE DETECTION

Description For rising or falling edge detection.

The result in the ACCU is High only when the ACCU goes from Low to High on consecutive executions of DYN (rising edge).

The Flag given in the operand stores the previous state of the ACCU. If the ACCU is Low, it remains Low, and the Flag is also set Low. The Flag need not be Low the first time DYN is executed. For rising edge detection, use STH to interrogate the element; for falling edge detection, use STL.

Usage

DYN[X]	**flag**	**(i)**	**; F 0-8191**

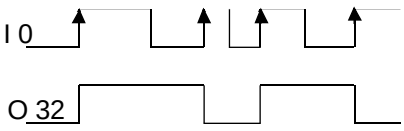
Example

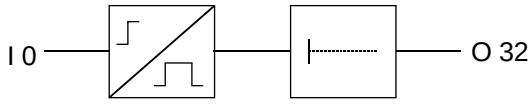
```
DYN     F 100   ; Flag 100 stores dynamic ACCU state
```

Flags The ACCU is set to the result.

See Also STH, STL, ANH, ANL, ORH, ORL.

Practice





; Solution with DYN instruction

```
COB       0     ; COB Header
          0
STH     I 0     ; If input 0 goes H
DYN     F 500   ;    (Edge detection)
COM     O 32    ;  Then complement output 32
                ; Else do nothing
ECOB            ; END of COB
```

; Solution without DYN instruction

```
COB       0     ; COB Header
          0
STH     I 0     ; If input 0 is H
ANL     F 500   ;    and flag 500 is L
SET     F 500   ;  Then  set flag 500 on H
COM     O 32    ;          complement output 32
                ; Else nothing
STL     I 0     ; If input 0 is L
RES     F 500   ;  Then reset flag 500 (state = L)
                ; Else nothing
ECOB            ; END of COB
```


OUT SET ELEMENT FROM ACCUMULATOR

Description Sets the Output or Flag to the state of the ACCU. If the ACCU is High then the Output or Flag is set High. If the ACCU is Low, then the Output or Flag is set Low.

.

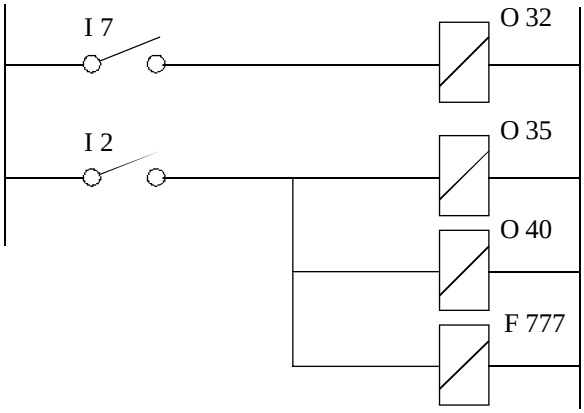
Usage	**OUT[X] element (i) ; O 0-8191, F 0-8191**

Example OUT O 32 ; Sets Output 32 to the state of the ACCU

Flags Unchanged.

See Also OUTS.

Practice




```
COB         0       ; COB Header
            0

STH       I 7       ; If input 7 is H
OUT       O 32      ;   Then set output 32
                    ; Else  reset output 32
STH       I 2       ; If input 2 is H
OUT       O 35      ;   Then  set output 35, else reset output 35
OUT       O 40      ;         set output 40, else reset output 40
OUT       F 777     ;         set flag 777, else reset flag 777

ECOB                ; END of COB
```


SET SET ELEMENT

Description The Output or Flag is set High only if the ACCU is High. If the ACCU is Low, nothing is done.
An output or flag set with a SET-instruction remains set (H) until it is reset again by a RES-instruction.

SET and RES are generally used in sequentially programs (GRAFTEC)

Usage

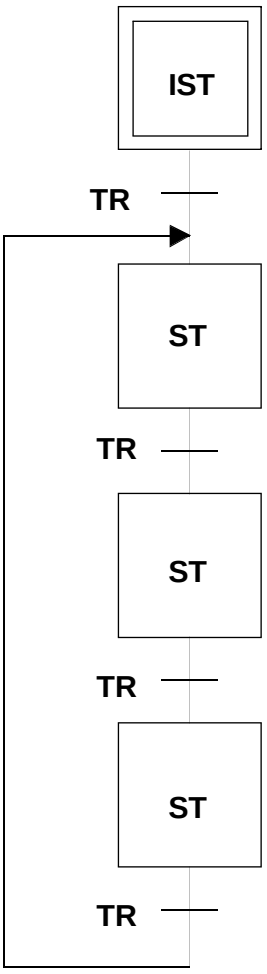
SET[X] element (i) ; O 0-8191, F 0-8191

Example SET O 32 ; If ACCU is 1 then O 32 = H

Flags Unchanged.

See Also RES, SETD, RESD.

Practice Outputs 36 and 37 must blink after input 7 has been switched on




```
STH  I  7    ; input = on ?

SET  O  36   ; set output 36
SET  O  37   ; set output 37
LD   T  1    ; load timer no. 1
           5 ;  with 0.5s

STL  T  1    ; timer elapsed ?

RES  O  36   ; reset output 36
RES  O  37   ; reset output 37
LD   T  1    ; load timer no. 1
           10;  with 1s

STL  T  10   ; timer elapsed ?
```


RES RESET ELEMENT

Description The Output or Flag is set Low only if the ACCU is High.
If the ACCU is Low, nothing is done.

Usage

RES[X] element (i) ; O 0-8191, F 0-8191

Example

```
RES     O 13    ; If ACCU is 1 then O 13 = L
```

Flags Unchanged.

See Also SET, SETD, RESD.

Practice see SET

COM COMPLEMENT ELEMENT

Description The state of the Output or Flag is complemented (inverted) only if the ACCU is High. If the ACCU is Low, nothing is done

Note:
This instruction is mainly used for the activation of the "WATCHDOG".
The "COM O 255" instruction must be placed in a cyclic program. Care must be taken that this instruction is only executed when the ACCU is H. (The "ACC H" instruction can be placed before the instruction or the instruction "COM" can be placed directly after the COB instruction)

Usage

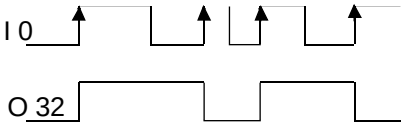
COM[X] element (i) ; O 0-8191, F 0-8191

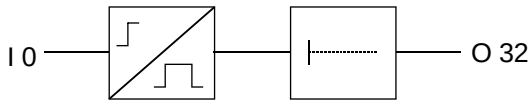
Example COM O 32 ; If ACCU is 1, inverts the state of Output 32

Flags Unchanged.

See Also OUT, SET, RES, DYN.

Practice






```
COB       0     ; COB Header
          0
STH       I 0   ; If input 0 goes H
DYN       F 500 ;       (Edge detection)
COM       O 32  ;   Then complement output 32
                ;   Else do nothing

ECOB            ; END of COB
```

SETD SET ELEMENT DELAYED

Description

The Output or Flag is set High after the delay given in the 2nd operand **only if the ACCU is High.** The delay is in timebase units, as set by the DEFTB instruction.

SETD and RESD are designed for the use in sequentially programs (GRAFTEC). The use of these instructions give simpler structures because the end of the delay must not be waited.

(If these instructions are used in BLOCTEC programs; they must always be combined with a DYN instruction. Without the DYN instruction, another timer will be activated on each program loop causing the ERROR flag to be set after the 16th loop when all the timers will be used.)

The operands cannot be supplied as Function Block parameters.

Usage

```
SETD[X]   element  (i)    ; O 0-8191, F 0-8191
          delay           ; Delay in timebase units (eg. 100ms)
```

Example

```
SETD  O 32     ; If ACCU is 1 then Output 32 = H
        100    ; after 100 x 100ms = 10 seconds.
```

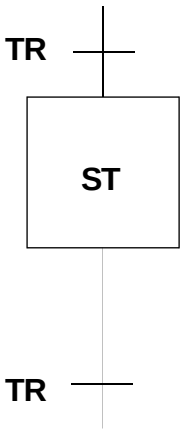
Flags

The **Error** (E) flag is set if more than 16 delayed actions are attempted.

See also

RESD, DEFTB.

Practice




```
STH ...        ; condition to continue

SET   O  35    ; O 35 is set immediately
SETD  O  40    ; O 40 is set after 120s
         1200  ;   not depending from
               ;   the continuation of
               ;   the GRAFTEC program


STH ...         ; condition to continue
```


RESD RESET ELEMENT DELAYED

Description

The Output or Flag is set Low after the delay given in the 2nd operand **only if the ACCU is High**. The delay is in timebase units, as set by the DEFTB instruction.

SETD and RESD are designed for the use in sequentially programs (GRAFTEC). The use of these instructions give simpler structures because the end of the delay must not be waited.

(If these instructions are used in BLOCTEC programs; they must always be combined with a DYN instruction. Without the DYN instruction, another timer will be activated on each program loop causing the ERROR flag to be set after the 16th loop when all the timers will be used.)

The operands cannot be supplied as Function Block parameters.

Usage

```
RESD[X]   element  (i)    ; O 0-8191, F 0-8191
          delay           ; Delay in timebase units (eg. 100ms)
```

Example

```
RESD   O 32     ; If ACCU is 1 then Output 32 = L
         100    ; after 100 x 100ms = 10 seconds
```

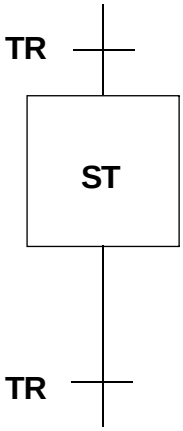
Flags

The **Error** (E) flag is set if more than 16 delayed actions are attempted.

See Also

SETD, DEFTB.

Practice




```
STH ...          ; condition to continue

SET   O  35      ; O 35 is set immediately
SETD  O  35      ; O 35 is reset after 5s
         50      ;   not depending from
                 ;   the continuation of
                 ;   the GRAFTEC program


STH ...           ; condition to continue
```

3. WORD Instructions

These instructions all work with Registers.
Registers contain binary, decimal, BCD or floating point values.
For floating point values, the floating point instructions must be used.

Loading Data	**LD**	Load (32-bit value)
	LDL	Load low word (lower 16 bits)
	LDH	Load high word (upper 16 bits)
	DSP	Load Display Register
Primary arithmetic	**INC**	Increment Register
	DEC	Decrement Register
Index Register	**SEI**	Set Index register
	INI	Increment Index register
	DEI	Decrement Index register
	STI	Store Index register
	RSI	Restore Index register
Moving Data	**MOV**	Move data
	COPY	Copy data } Specially useful
	GET	Get data } for indexed
	PUT	Put data } addressing.
	TFR	Transfer data
	TFRI	Transfer data indirect
Binary Input/Output	**BITI**	Bit in
	BITIR	Bit in reversed
	BITO	Bit out
	BITOR	Bit out reversed
BCD Digit Input /Output	**DIGI**	Digit in
	DIGIR	Digit in reversed
	DIGO	Digit out
	DIGOR	Digit out reversed
Logical	**AND**	AND Registers
	OR	OR Registers
	EXOR	Exclusive-OR Registers
	NOT	Complement Register
Rotates and Shifts	**SHIU**	Shift Registers up
	SHID	Shift Registers down
	ROTU	Rotate Registers up
	ROTD	Rotate Registers down
	SHIL	Shift Register left
	SHIR	Shift Register right
	ROTL	Rotate Register left
	ROTR	Rotate Register right

Notes

Notes

LD LOAD (32-BIT VALUE)

Description

The addressed Register, Timer or Counter is loaded with the given 32-bit value.

For Timers and Counters:

- the operation is done only if the ACCU is High.
- cannot be loaded with negative or floating point values. (only decimal, Hex, ASCII or binary values)
- if a timer is loaded, the timer starts immediately.
- The state of a timer or counter is H when it contains a non-zero value else its state is L

For Registers:

- the operation is independent of the ACCU state.
- the value can be a decimal, Hex, ASCII or floating point value.

Binary values are post-fixed with a 'Q' or 'Y'.
Hex values are post-fixed with an 'H'.
Floating point values must contain a decimal point '.' or an exponent "E6".
ASCII values are enclosed in single quotes 'a', 'A'.
Since the value is 32-bits, three program lines are used for the whole instruction.

The operands cannot be supplied as Function Block parameters.

Usage

```
LD[X]     element   (i)     ; R 0-4095, T 0-450, C 0-1599.
          value             ; Decimal: -2.147.483.648 to +2.147.483.647
                            ; Hex: 0H to FFFFFFFFH
                            ; Binary: 0Y to 111...111Y (32 bits)
                            ; Floating point: ± 5.42101E-20 to ± 9.22337E+18
                            ; ASCII: 'A'-'Z', '0'-'9', '!', '?' etc.
```

Example

```
LD    R 0         ; Loads R 0 with floating point value 321
        3.21E1    ; (Timers & Counters must have +ve values)
```

Flags

Unchanged.

See Also

LDH, LDL (16-bit loads), Constants.

Note

LD needs 3 program lines
LD T/C: only executed when the ACCU = H (1).
LDR: always executed.

LDL LOAD LOW WORD (LOWER 16 BITS)

Description

Loads the lower 16 bits (0-15) of a Register, Timer or Counter with a 16-bit value (0-65535); the upper 16 bits are always set to 0.

For Timers and Counters:
LDL is executed **only if the ACCU is 1**.
For Registers:
LDL is always executed.

LDL (and LDH, Load High) allow 16-bit constants to be passed as Function Block parameters, or loaded directly.
LDH loads the upper 16 bits.
Using these instructions, a 32-bit value can be loaded. To load all 32 bits, LDL must be executed first, because this sets the upper 16 bits to 0.
Values can be loaded in decimal, hex, binary or ASCII, NOT floating point.

Usage

```
LDL[X]    element (i)    ; R 0-4095, T 0-450, C 0-1599
          value          ; Decimal: 0-65535
                         ; Hexadecimal: 0H-FFFFH
                         ; Binary: 16 bits
```

Example

```
LDL    R 100      ; Loads Register 100 with FFFF Hex,
         0FFFFH   ; which is 65535 in decimal.
                  ; R100 = 0000FFFFH
```

Flags

Unchanged.

See Also

LDH, LD, Constants.

Note

When only values < 65535 are used, the LDL instruction can be standard used to load Counters, Timers or Registers.

LDH LOAD HIGH WORD (UPPER 16 BITS)

Description

Loads the upper 16 bits (16-31) of a Register, the lower 16 bits are not affected. LDH (and LDL, Load Low) allow 16-bit constants (0-65535) to be passed as Function Block parameters, or loaded directly. Using these instructions, a 32-bit value can be loaded. To load all 32 bits, LDL must be executed first, because this sets the upper 16 bits to zero (0).

Values can be loaded in decimal, hex, ASCII or binary, NOT floating point. LDH cannot be used to load Timers or Counters, where the upper 16 bits cannot be loaded separately.

Usage

```
LDH[X]    element (i)    ; R 0-4095
          value          ; Decimal: 0-65535
                         ; Hexadecimal:  0H-FFFFH
                         ; Binary: 16 bits
```

Example

```
LDH   R 100      ; Loads bits 31-16 of Register 100
        0FFFFH   ; with FFFF Hex.
                 ; R 100 = FFFFxxxx Hex
```

Flags

Unchanged.

See Also

LDL, LD, Constants

Practice

To load a Register in a Function Block with a 32-bit constant, you can't use directly the LD statement; in place, you must use LDL and LDH.
(The upper and lower 16 bits of a constant can be separated using the Assembler statements '&' (AND) and '/' (DIVIDE)). A constant (12345678) must be passed as parameter to a Function block where it is loaded in a Register. Remember that LDL must be done BEFORE LDH.

```
COB          0          ; COB Header
             0

CFB          0          ; Call Function Block 0
12345678 & 0FFFFH       ; Parameter 1 (lower 16 bits)
12345678 / 0FFFFH       ; Parameter 2 (upper 16 bits)

ECOB                    ; END of COB

FB           0          ; FB Header
LDL       R 0           ; Load the lower 16 bits of Register 0
          = 1           ;  with the 1st parameter (lower 16 bits)
LDH       R 0           ; Load the upper 16 bits of Register 0
          = 2           ;  with the 2nd parameter (upper 16 bits)
EFB
```

DSP LOAD DISPLAY REGISTER

Description The logical state of an Input, Output or Flag, or the contents of a Register, Timer or Counter, or a constant, can be loaded into the Display register.
This value is displayed on the PCD8.P100 Programming Unit in decimal.
It is useful as an error code or status display.

The operand can not be supplied as a Function Block parameter.

Usage

```
DSP     value     ; I 0-8191, O 0-8191, F 0-8191, T 0-450,
                  ; C 0-1599, R 0-4095, K 0-16383
```

Example

```
DSP   R 0       ; Display register = contents of R 0
DSP   K 1234    ; Display register = constant 1234
```

Flags Unchanged.

See Also PCD1/2 Hardware Manual

Practice (see the INC instruction)

INC INCREMENT REGISTER OR COUNTER

Description The Register or Counter is incremented by one.

Counters are incremented **only if the ACCU is 1**.
Registers are incremented **regardless of the ACCU state**.

Usage

INC[X] element (i) ; R 0-4095, C 0-1599

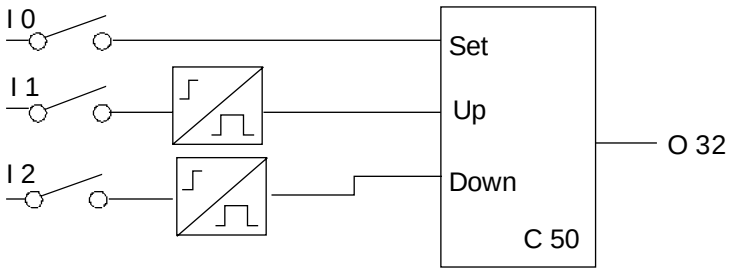
Example

```
INC    R 100     ; R 100 = R 100 + 1
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set if overflow occurs.

See Also DEC, ADD

Practice Up /down counter with preselection and display of the counter value




```
COB         0      ; COB Header
            0

STH       I 0      ; If input 0 is H
LD        C 50     ;      Then load counter 50 with 5
            5
                   ; Else do nothing
STH       I 1      ; If input 1 goes H
DYN       F 1      ;           (Edge detection)
INC       C 50     ;      Then counter 50 incremented by 1
                   ; Else do nothing
STH       I 2      ; If input 2 goes H
DYN       F 2      ;           (Edge detection)
DEC       C 50     ;      Then counter 50 decremented by  1
                   ; Else do nothing
STH       C 50     ; If counter 50 content >> 0
OUT       O 32     ;      Then set output 32
                   ; Else reset output 32
DSP       C 50     ; Display counter 50

ECOB
```


DEC DECREMENT REGISTER OR COUNTER

Description The Register or Counter is decremented by one.

Counters are decremented **only if the ACCU is 1**.
Registers are decremented **regardless of the ACCU state**.

Usage

DEC[X] element (i) ; R 0-4095, C 0-1599

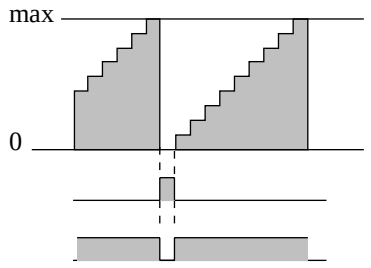
Example

```
DEC    R 100     ; R 100 = R 100 - 1
```

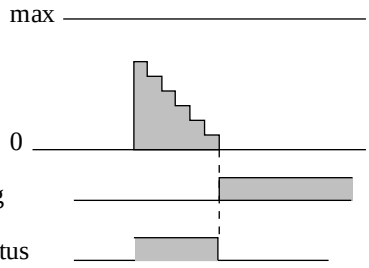
Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set if underflow occurs.

See Also INC, SUB

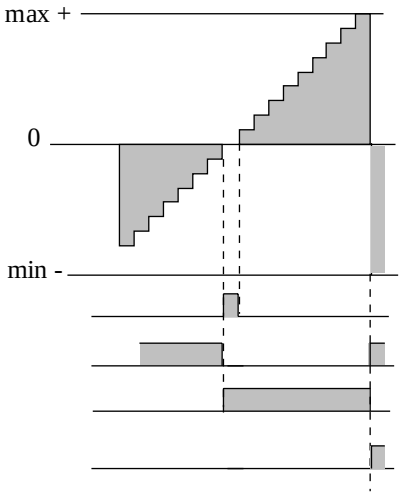
INC Counter



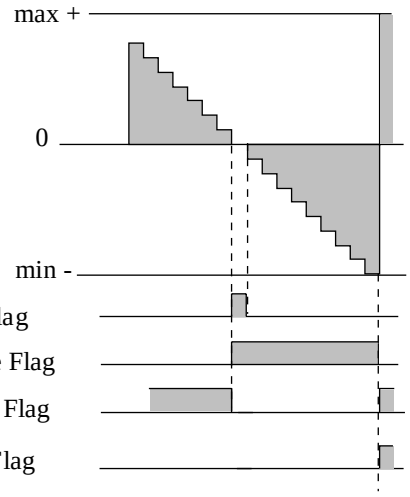
DEC Counter



INC Register



DEC Register



SEI SET INDEX REGISTER

Indexed Addressing

It is frequently necessary for series of inputs, outputs, flags,... to be dealt with in the same way (for example resetting of all retentive flags or registers). In cases like this, long programs can be drastically shortened with the help of address indexing.
Each COB or XOB has its own **Index register**.
This register is used for indexed addressing, where the contents of the Index register is added to the operand value to provide the actual address.
Indexing instructions are always ended with an 'X', for Example STHX, BITIX.
The Index register can be loaded or saved, incremented up to a given limit, or decremented down to a given limit.

Description The current Index register is loaded with the supplied constant (K 0-8191) or the contents of the indicated Register.

NOTE: The value range of the Index register is 0..8191 (13 bits).

If a value > 8191 is entered then the Index is set to 8191 and the XOB 12 is called.
If a value < 0 is entered then the Index register is set to 0 and the XOB 12 is called.

Usage

SEI	**value**	**; K 0-8191, R 0-4095**

Example

```
SEI   K 32     ; Loads Index register with the value 32
SEI   R 32     ; Loads Index register with the content of Register 32
```

Flags Unchanged.

See Also INI, DEI, STI, RSI

Practice The status of an input which address is given by a BCD encoder must be shown on output 32.

```
COB       0
          0

DIGI      2        ; Read 2 digits
          I 24     ;      from inputs 24 (to 31)
          R 500    ;      and store them in R 500

SEI       R 500    ; Load Index with the contents of R 500

STHX      I 0      ; If Input (0 + Index) is H
OUT       O 32     ;      Then set output 32
                   ;  Else reset output 32
ECOB
```


INI INCREMENT INDEX REGISTER

Description The current Index register is compared to the value of the operand (supplied K constant or the contents of a Register).
If the Index register is less than this value, the Index register is incremented and the ACCU is set 1.
If the Index register is equal or greater than the value of the operand, the Index register is NOT incremented and the ACCU is set 0.
If the value in the operand is > 8191 or < 0 then XOB 12 is called.

Usage

INI	**value**	**; K 0-8191, R 0..4095**

Example

```
INI    K 100     ; Increments Index register if it is lower than 100.
INI    R 333     ; Increments Index register
                   if it is lower than the contents of R 333
```

Flags **ACCU** = 1 if Index register is less than the value in the operand.
ACCU = 0 if Index register is greater or equal than the value in the operand.

See Also DEI, SEI

Practice At power on, the registers 1500 to 1999 must be reset (value 0)

```
          XOB       16       ; XOB executed at power on
          .....
          SEI       K 0      ; Set index register on 0
                             ; Repeat
Reset:    LDX       R 1500   ;    Load Register (1500 + Index reg)
                    0        ;     with 0
          INI       K 499    ;    Increment Index Register by 1
          JR        H Reset  ;    Until Index Register > 499
          (ACC      H)
          .....
          EXOB
```


DEI DECREMENT INDEX REGISTER

Description

The current Index register is compared to the value of the operand (K constant or the contents of a Register).
If the Index register is greater than this value, the Index register is decremented and the ACCU is set 1.
If the Index register is equal or less than the value of in the operand (constant or Register contents), the Index register is NOT decremented and the ACCU is set 0.
If the value in the operand is > 8191 or <0 then XOB 12 is called.

Usage

DEI	**value**	**; K 0-8191, R 0..4095,**

Example

```
DEI    K 100     ; Decrements Index register if it is greater
                 ;  than the constant 100.
DEI    R 444     ; Decrements Index register if it is greater
                 ;  than the contents of R 444.
```

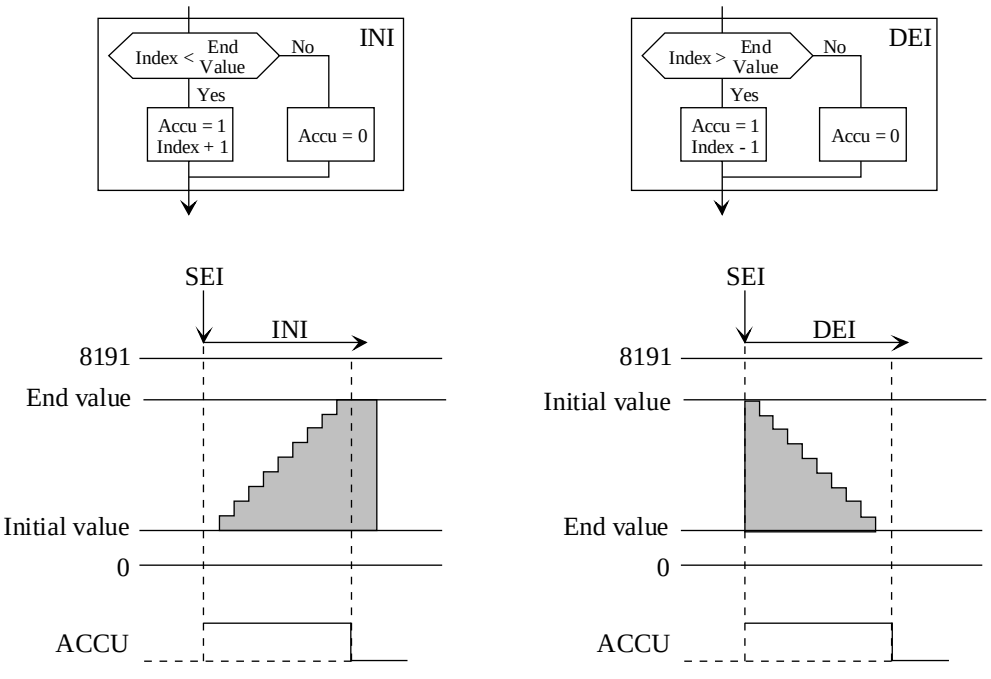
Flags

ACCU = 1 if Index register is greater than the value in the operand.

ACCU = 0 if Index register is less or equal than the value inthe operand.

See Also

INI, SEI



End value = value of the operand of the INI / DEI instruction

STI STORE INDEX REGISTER

Description The value in the current Index register is stored in the given Register.
It can be re-loaded into the Index register using the RSI instruction.
The Index register is not changed.

Usage

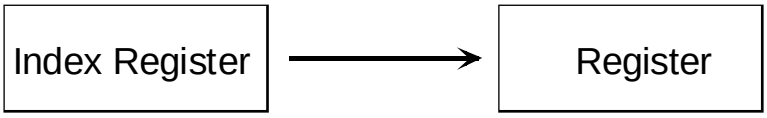
STI	**reg**	**; R 0-4095**

Example

```
STI    R 100     ; Stores the Index register contents in Register 100
```

Flags Unchanged.

See Also RSI



RSI RESTORE INDEX REGISTER

Description Loads the Index register with the contents of the given Register.
The value in the Register will typically be an Index register value saved by the STI instruction.
The maximum value the Register should contain is 8191, only the lower 13 bits are loaded.
If a parameter <0 or > 8191 is entered then the XOB 12 is called.

Usage

RSI	**reg**	**; R 0-4095**

Example

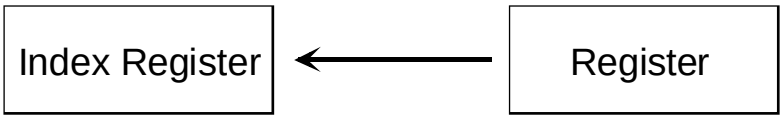
```
RSI     R 100   ; Loads the Index register with the contents of Register 100.

        This is the same as :   SEI   R 100
```

Flags Unchanged.

See Also STI, SEI

Practice



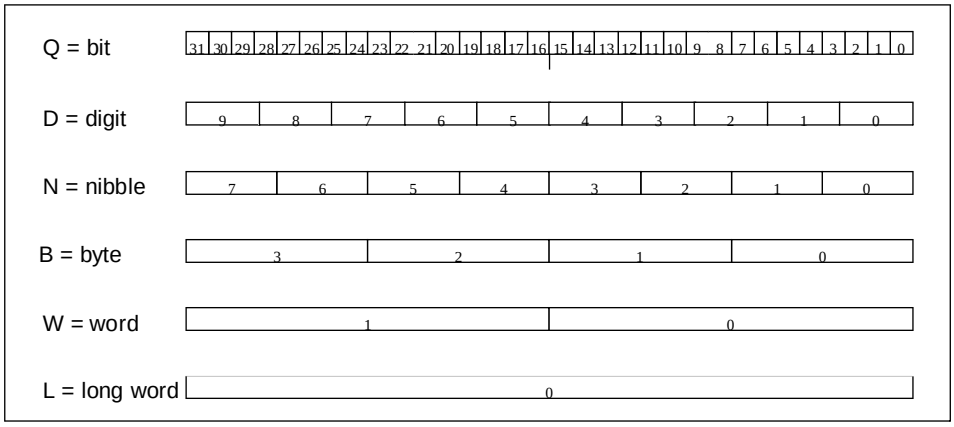
MOV MOVE DATA

Description Moves data from a Timer, Counter or Register into a Register.
This is a 4-line instruction:

- the 1st and 3rd operands are the source and destination.
- the 2nd and 4th operands are the data type and position:

Q = Bit (moves 1 bit)	0-31
D = Digit (4 Bits BCD)	0-9
N = Nibble (4 Bits Binary)	0-7
B = Byte (8 Bits)	0-3
W = Word (16 Bits)	0/1
L = Long word (32 Bits)	0

The data types (Q, D etc.) of the 2nd and 4th operands must be the same, but source and destination positions may differ.



Usage

```
MOV[X]   source    (i)     ; R 0-4095, T 0-450, C 0-1599
         type position     ; Q|D|N|B|W|L see above
         dest      (i)     ; R 0-4095
         type position     ; Q|D|N|B|W|L see above
```

Example Move the Highest nibble (N7) from Register 100 to the Lowest nibble (N0) of Register 101

```
; Before:  R100:  1111 1010 1010 1010 1010 1010 1010 1010
           R101:  0001 0001 0001 0001 0001 0001 0001 0001

           MOV    R 100
                  N 7
                  R 101
                  N 0

; After:   R100:  1111 1010 1010 1010 1010 1010 1010 1010
           R101:  0001 0001 0001 0001 0001 0001 0001 1111
```

Flags Unchanged.

See Also COPY, GET, PUT, LD, LDH, LDL

COPY COPY DATA

Description The COPY, GET and PUT instructions are all related.
They are used for the indexed transfer of data between Registers, Timers and Counters. In each case, an entire Register, Timer or Counter is copied.

For **COPYX**, the contents of the first operand is copied into the second, BOTH are indexed.

Usage

COPY[X]	**source**	**(i)**	**; R 0-4095, T0-450, C0-1599**
	destination	**(i)**	**; R 0-4095, T0-450, C0-1599**

Example

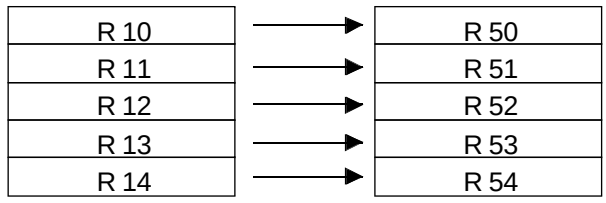
```
COPYX   R 10    ; Moves the contents of R (10+Index)
        R 50    ; into R (50+Index)
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the value copied.

See Also GET, PUT, MOV

Practice

```
        SEI     K 0
loop:   COPYX   R 10
                R 50
        ...
        INI     K 4
        JR      H loop
```



GET GET DATA

Description

The COPY, GET and PUT instructions are all related.
They are used for the indexed transfer of data between Registers, Timers and Counters. In each case, an entire Register, Timer or Counter is copied.

For **GETX**, the contents of the first operand is copied into the second, only the first operand is indexed.

In addition, the GET instruction allows the transfer from the Texts / Data Blocks to the Registers/Timers/Counters.

Usage

GET[X]	**source**	**(i)**	**; R 0-4095, T0-450, C0-1599, X0-3999, DB0-3999**
	destination		**; R 0-4095,T0-450, C0-1599**

Example

```
GETX    R 10    ; Moves the contents of R (10+index)
        R 50    ; into R 50
```

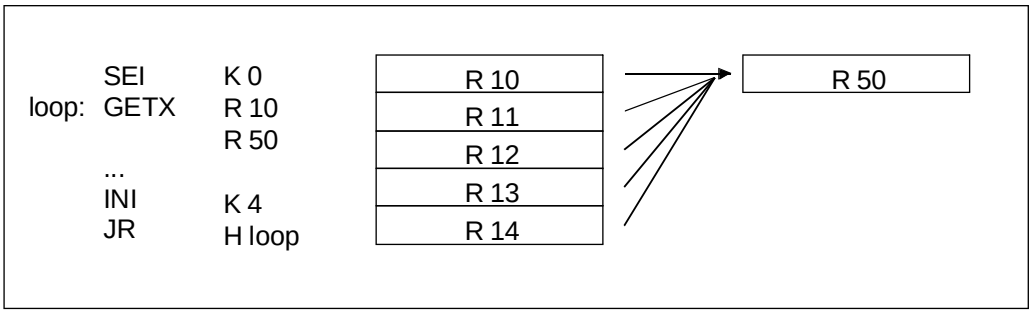
Flags

The **Zero** (Z) and **Sign** (P or N) flags are set according to the value copied.

See Also

PUT, COPY, MOV

Practice



Transfer between Text / Data Block and R/T/C

The instruction GET[X] can transfer the stated Text into the R/T/C's until the end of the Text (00 - NUL terminator). If the Text does not end on a R/T/C boundary then the remainder of the R/T/C will be left unchanged.

Similarly, GET[X] can transfer the data items present in a Data Block to the R/T/C's until the end of the Data Block.
A DATA BLOCK is an area of user memory where large numbers of Registers, Timers or Counters can be saved and read at run-time. Data Blocks can be used for storing values which are specific to a process to liberate R/T/C's for use by other processes.
If the instruction tries to read from a Text or Data Block which doesn't exist then XOB 13 (Error flag set) is called. If the indexed Text or Data Block number is greater than 3999 then XOB 12 is called (Index Register Overflow)

DATA BLOCK and TEXT Formats

All Data Blocks and Texts must be defined in the source program in the following way:

```
DB      number    '[' [length] ']' [value 1 [,value 2]...[,value n] ]

TEXT    number    '[' [length] ']'  "text line 1"   [;comment]
                                    ["text line 2"   [;comment]]
                                    ["text line n"   [;comment]]
```

where number, length can be	decimal symbol
and value 1, ... can be	decimal hexadecimal binary floating point ASCII symbol

Texts and Data Blocks are numbered 0..3999 but because they share the same area of memory, they cannot share the same number. For Example, if Data Block 10 is defined then Text 10 is unavailable.
A Text can be defined as normal with a maximum possible length of 3069 characters.
A Data Block can hold a maximum of 383 data items, where a data item is a register or timer or counter.

Note: It is forbidden to have <254> or <255> in the first byte of a normal Text. If the PCD contains EPROM memory then the Data Blocks and Texts can only be read (GET instruction).

Examples:

```
DB   100     [10]                ; Data Block number 100 of 10 data item
                                 ; all values initialised to zero.

DB   101     [] 1.0,konst,3.0,4  ; Data Block  number 101 of 4 data items
                                 ; two of which are floating point and one symbol

DB   102     [4] 1,2,3           ; Data Block number 102 of 4 items 1, 2, 3, 0

DB   num     [data_len]          ; Data Block with predefined symbols

TEXT 100     "NORMAL TEXT"
TEXT 103     [5] "123"           ; Text 103 which is "123  " (with two spaces)
```

GET Data Block /Text

Example 1:
Data Block as declared in the source program:

```
DB  100    [5] 0h,1h,2h,0a5a5a5a5h,720h
```

Instruction:

```
GET     DB  100     ; Transfer Data Block 100
        R   1000    ; into Regiters 1000 and consecutive.
```

Result:

Register	Value in Hex
1000	00000000
1001	00000001
1002	00000002
1003	a5a5a5a5
1004	00000720

Example 2:
Text as declared in the source program:

```
TEXT  100 "THIS IS A TEXT 123"
```

Instruction:

```
GET    X 100        ; Transfer Text 100
       R 1000       ; into Registers 1000 and consecutive.
```

Result:

Register	Value in ASCII	Value in Hex
1000	THIS	54484953
1001	IS	20495320
1002	A TE	41205445
1003	XT 1	58542031
1004	23	32332020

PUT PUT DATA

Description The COPY, GET and PUT instructions are all related.
They are used for the indexed transfer of data between Registers, Timers and Counters. In each case, an entire Register, Timer or Counter is copied.

For **PUTX**, the contents of first operand is copied to the second, only the destination (second) operand is indexed.

In addition, the PUT instruction allows the transfer from the Registers / Timers / Counters to the Texts / Data Blocks.

Usage

PUT[X]	**source**	**; R 0-4095, T 0-450, C 0-1599**
	destination (i)	**; R 0-4095, T 0-450, C 0-1599, X 0-3999, DB 0-3999**

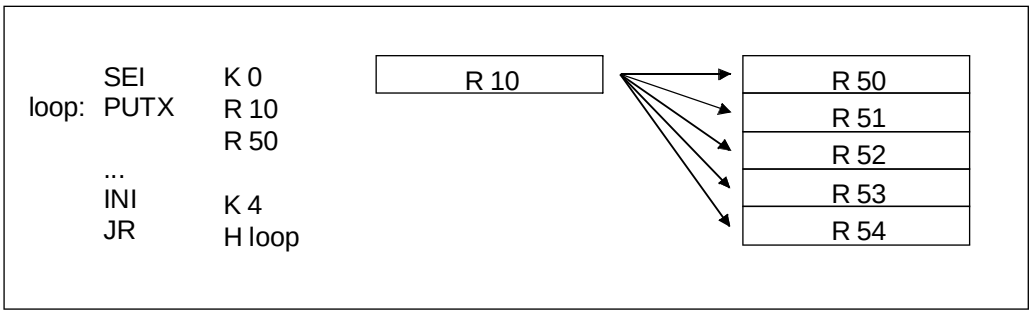
Example

```
PUTX    R 10    ; Moves the contents of Register 10
        R 50    ; into Register (50 + Index)
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the value copied.

See Also GET, COPY, MOV

Practice



Transfer between R/T/C and Text/Data Block

The instruction PUT[X] copies contents of the R/T/C starting with the stated R/T/C into the stated Text until the end of the Text. If there is a NUL (00h) char. in the R/T/C then the instruction PUT will change this value to a space. Similarly, PUT[X] copies the contents of the R/T/C's starting with the stated R/T/C into the stated Data Block until the Data Block is full.

A DATA BLOCK is an area of user memory where large numbers of Registers, Timers or Counters can be saved and read at run-time. Data Blocks can be used for storing values which are specific to a process to liberate R/T/C's for use by other processes.
For the Text/Data Block format declaration see the GET instruction.
If the instruction tries to copy from an R/T/C which doesn't exist then XOB 13 (Error flag set) is called. If the indexed Text or Data Block number is greater than 3999 then XOB 12 is called (Index Register Overflow)

PUT into Data Block /Text

NOTE:
PUT cannot extend the text or data block length
PUT cannot transfer values to a text or to a data block if eproms or rams with jumper in position "WP" are used.

Example 1:
Data Block as declared in the source program:

```
DB  100     [5]             ; Initial values are zero
```

Contents of Registers:

Register	Value in Dec
1000	00000001
1001	00000002
1002	00000003
1003	01234567
1004	00000720

Instruction:

```
PUT         R 1000          ;Transfer Register 1000 and consecutive
            DB 100          ; into Data Block 100.
```

Result as displayed with the Debugger in Decimal presentation:
DB 100 (0): 1 2 3 1234567 720

Example 2:
Text as declared in the source program:

```
TEXT  100 [17]          ; A text of 17 spaces
```

Contents of Registers:

Register	Value in ASCII	Value in Hex
1000	THIS	54484953
1001	IS	20495320
1002	A TE	41205445
1003	XT 1	58542031
1004	23	32332020

Instruction:

```
PUT         R 1000          ; Transfer Registers 1000 and consecutive
            X 100           ; into Text 100
```

Result as displayed with the Debugger:

```
TEXT 100:   "THIS IS A TEXT 12"
            (THE TEXT STOPS AFTER 17 CHARACTERS)
```


TFR TRANSFER DATA

Description This instruction enables the indexed data transfer of individual values from a Data Block or a Text to Registers, Timers or Counters; and vice versa.

Usage

TRANFER DATA BLOCK (or Text) ⟶ R, T, C

Copy an individual value (32 bit) from a Data Block or Text to a Register, Timer or Counter:

```
TFR[X]    source          ; DB  (or X) 0..3999, 4000..7999
          position        ; R 0..4095, K 0..382, 0..16383
          dest      (i)   ; R0..4095, T|C 0..1599
```

The 1st operand is the Data Block (or Text) containing the value to transfer.
The 2nd operand is the position of the value inside the Data Block (or Text); this position can be given as a constant or indirectly via a Register.
The 3rd operand is the destination Register, Timer or Counter

TRANSFER R, T, C ⟶ DATA BLOCK (or TEXT)

Copy a Register, Timer or Counter into a Data Block (or Text):

```
TFR[X]    source    (i)   ; R0..4095, T|C 0..1599
          destination     ; DB  (or X) 0..3999, 4000..7999
          position        ; R0..4095, K0..382, 0..16383
```

The 1st operand is the Register, Timer or Counter containing the value to transfer (source).
The 2nd operand is the destination Data Block (or Text).
The 3rd operand is the position inside the Data Block (or Text) where the value must be transferred, this position can be given as a constant or indirectly via a Register.

Remark:

The length of the Data Block is dependent of the PCD memory type:

Memory	Address	Maximum length of 1 DB
Standard	DB 0..3999	383 values
Extended	DB 4000..7999	16383 values

Example

```
TFR     DB  4010    ; Copy from the Data Block 4010
        K   13      ;   the value at position 13
        R   26      ;   to Register 26
TFR     R   120     ; Copy Register 120
        DB  4025    ;   to Data Block 4025
        K   6       ;   at position 6
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the value copied.

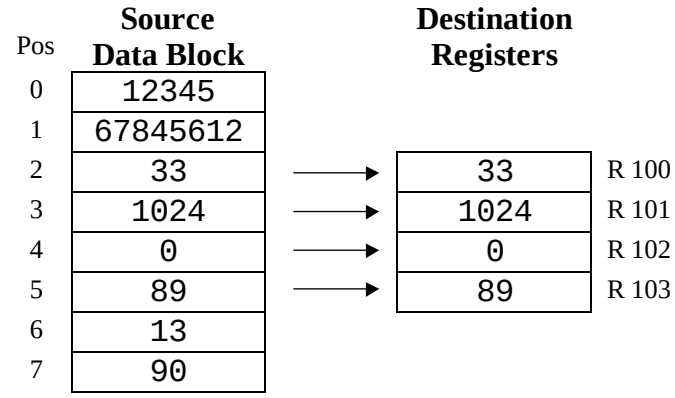
See Also PUT, GET

Notes For reasons of memory organisation, access to DBs from 4000..7999 is

significantly faster than to DBs from 0..3999. It is therefore recommended that this instruction should be used mainly on DBs from 4000..7999.

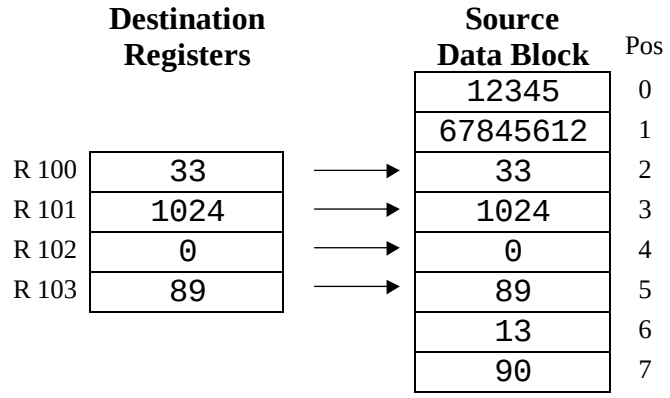
Practice From Data Block 4010, the 4 values from position 2..5 are copied to Registers 100..103.

```
        ….
        LD     R  999
               2
        SEI    K  0
LOOP:   TFRX   DB 4010
               R  999
               R  100
        INC    R  999
        INI    K  3
        JR     H  LOOP
        …
```



Registers 100..103 are copied to positions 2..5 of Data Block 4010:

```
        ….
        LD     R  999
               2
        SEI    K  0
LOOP:   TFRX   R  100
               DB 4010
               R  999
        INC    R  3
        JR     H  LOOP
        …
```



TFRI TRANSFER DATA INDIRECT

Description This instruction enables the indirect indexed data transfer of individual values from a Data Block or a Text to Registers, Timers or Counters; and vice versa.
This instruction does not work in parametrised mode

Usage **TRANSFER DATA BLOCK (or Text) ⟶ R, T, C**
Copy an individual value (32 bit) from a Data Block or Text to a Register, Timer or Counter:

```
TFRI    source        ; DB (or X) reg1
        position      ; R 0..4095, K 0..16383
        destination   ; R or T|C reg2
```

The 1st operand specifies that the source is a Data Block or a Text and the variable reg1 is a register number containing the address of the DB or Text.
The 2nd operand is the position of the value inside the Data Block (or Text); this position can be given as a constant or indirectly via a Register.
The 3rd operand specifies the type of the destination (R or T|C). The variable reg2 is a Register number which contains the destination number of the media.

TRANSFER R, T, C ⟶ DATA BLOCK (or TEXT)
Copy a Register, Timer or Counter into a Data Block (or Text):

```
TFRI    source        ; R or T|C reg1
        destination   ; DB ( or X) reg2
        position      ; R 0..4095, K 0..16383
```

The 1st operand specifies the type of the source (R or T|C). The variable reg1 is a Register number which content the destination address of the media.
The 2nd operand specifies the type of the destination (DB or Text) and the variable reg2 is a register number containing the address of the destination.
The 3rd operand is the position inside the Data Block (or Text) where the value must be transfered, this position can be given as a constant or indirectly via a Register.

Remark:

The length of the Data Block is dependent of the PCD memory type:

Memory	Address	Maximum length of 1 DB
Standard	DB 0..3999	383 values
Extended	DB 4000..7999	16383 values

Examples

Transfer the element at position 10 od Data Block 4000 to Register 4095.

```
LD      R  100  ; Initialisation of the DB address
           4000
LD      R  101  ; Initialisation of the Register address
           4095
TFRI    DB 100  ; Transfer DB
        K  10
        R  101
```

Transfer the Counter 1000 to the position 50 of Data Block 4000

```
LD      R  100  ; Initialisation of the DB address
           4000
LD      R  101  ; Initialisation of the position
           50
LD      R  102  ; Initialisation of the Counter address
           4095
TFRI    C  102  ; Transfer Counter
        DB 100
        R  101
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the value copied.

See Also TFR, PUT, GET

Notes For reasons of memory organisation, access to DBs from 4000..7999 is significantly faster than to DBs from 0..3999. It is therefore recommended that this instruction should be used mainly on DBs from 4000..7999.

BITI BIT IN

Description Moves a sequence of binary bits from Inputs, Outputs, Flags, a Timer or a Counter into a Register.

The 1st operand is the number of bits to be moved (1-32).
The 2nd operand is the source (I, O, F, T or C).
The 3rd operand is the destination Register.

If the source are Inputs, Outputs or Flags, the source address given is the lowest element of the sequence.
The LOWEST element becomes the LEAST SIGNIFICANT bit in the destination Register.
This is contrary to the SAIA PCA.

Usage

```
BITI[X]   bits          ; Number of bits to read 1-32
          source        ; Source data address I, O, F, T, C
          dest    (i)   ; Destination Register number R 0-4095
```

Example

```
BITI       16   ; Reads 16 bits
         I 32   ; from Inputs 32-47 and
         R 0    ; stores them in Register 0 bits 0-15
```

Flags **Zero** (Z) and **Sign** (P or N) flags set according to value read..

See Also BITIR, DIGI, DIGIR

Practice When input 8 goes High, an 8 bits binary value is read from inputs 0 to 7 and stored in Register 500

```
          COB        0     ; COB Header
                     0
          .....
          STH      I 8     ; If input 8 goes High
          DYN      F 100
          JR       L NEXT
          BITI       8     ;    Then read 8 bits binary
                   I 0     ;         from input 0 (to 7)
                   R 500   ;         and store them in R 500
NEXT:     ....
          ECOB
```


BITIR BIT IN REVERSED

Description Moves a sequence of binary bits from Inputs, Outputs, Flags, a Timer or a Counter into a Register.

The 1st operand is the number of bits to be moved (1-32).
The 2nd operand is the source (I, O, F, T or C).
The 3rd operand is the destination Register.

If the source are Inputs, Outputs or Flags, the source address given is the lowest element of the sequence.
The LOWEST element becomes the MOST SIGNIFICANT bit in the destination Register.
This is the same as the SAIA PCA.

Usage

```
BITIR[X]   bits          ; Number of bits to read 1-32
           source        ; Source data address I, O, F, T, C
           dest    (i)   ; Destination Register number R 0-4095
```

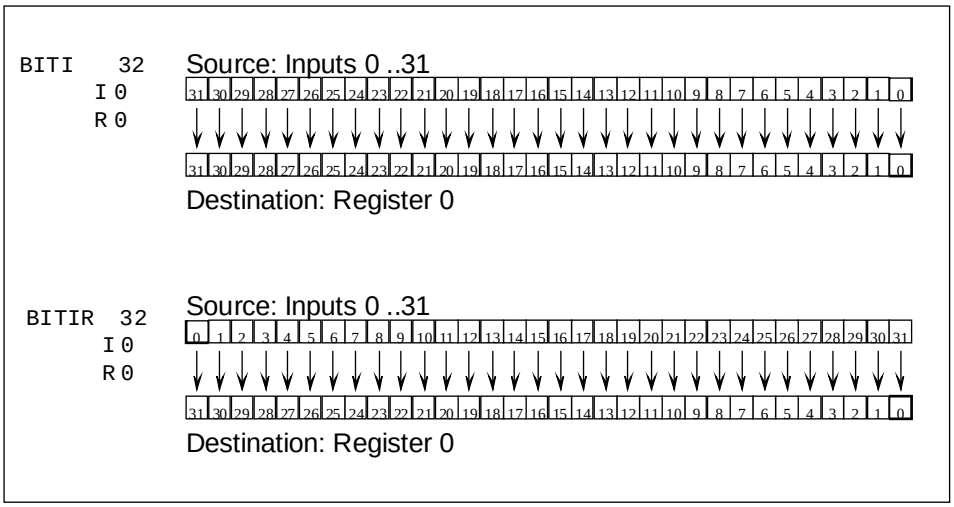
Example

```
BITIR      16   ; Reads 16 bits
         I 32   ; from Inputs 32-47 and
         R 0    ; stores them in Register 0 bits 15-0
```

Flags **Zero** (Z) and **Sign** (P or N) flags set according to value read.

See Also BITI, DIGI, DIGIR

Practice



BITO BIT OUT

Description Moves a sequence of binary bits from a Register to corresponding Outputs, Flags or bits in a Timer or Counter.

The 1st operand is the number of bits to transfer (1-32).
The 2nd operand is the source Register number.
The 3rd operand is the destination Outputs, Flags, Timer or Counter.

If the destination are Outputs or Flags, the address given is that of the lowest element in the sequence.
The LEAST SIGNIFICANT bit of the Register is moved to the LOWEST element. This is contrary to the SAIA PCA

Usage

BITO[X]	**bits**		**; Number of bits to move 1-32**
	source	**(i)**	**; Source Register R 0-4095**
	dest		**; Destination data address O, F, T, C**

Example

```
BITO    8       ; Move 8 bits
        R 10    ; from Register 10 bits 0-7
        O 48    ; and output to Outputs 48-55
```

Flags Unchanged.

See Also BITOR, DIGO, DIGOR

Practice Copy the status of the inputs 0 → 15 to the outputs 32 → 47

```
COB       0     ; COB Header
          0

BITI     16     ; Read 16 bits
       I 0      ;    from input 0 (to 15)
       R 0      ;    store them in R 0

BITO     16     ; Write 16 bits
       R 0      ;    from R 0
       O 32     ;    to outputs 32 (to 47)

ECOB
```


BITOR BIT OUT REVERSED

Description

Moves a sequence of binary bits from a Register to corresponding Outputs, Flags or bits in a Timer or Counter.

The 1st operand is the number of bits to transfer (1-32).
The 2nd operand is the source Register number.
The 3rd operand is the destination Outputs, Flags, Timer or Counter.

If the destination are Outputs or Flags, the address given is that of the lowest element in the sequence.
The LEAST SIGNIFICANT bit of the Register is moved to the HIGHEST element. This is the same as the SAIA PCA.

Usage

```
BITOR[X]  bits          ; Number of bits to move 1-32
          source  (i)   ; Source Register R 0-4095
          dest          ; Destination data address  O, F, T, C
```

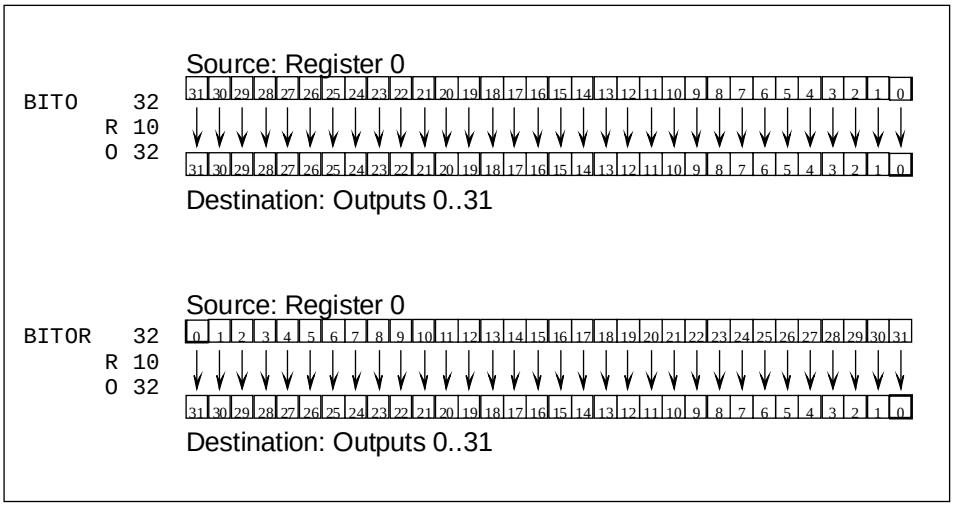
Example

```
BITOR     8     ; Move 8 bits
        R 10    ; from Register 10 bits 0-7
        O 48    ; and output to Outputs 55-48
```

Flags Unchanged.

See Also BITO, DIGO, DIGOR

Practice



DIGI DIGIT IN

Description Moves Binary Coded Decimal (BCD) digits from Inputs, Outputs or Flags into a Register.
A BCD digit is 4 bits (eg. 4 Inputs), which represents a decimal digit (0-9).
The 1st operand is the number of digits to move (1-10).
The 2nd is the base Input, Output or Flag.
The 3rd operand is the destination Register.
The lowest addressed Input, Output or Flag becomes the least significant bit of the least significant digit in the destination Register.
This is contrary to the PCA.

Usage

DIGI[X]	**digits**		**; Number of BCD digits 1-10**
	source		**; Source element I 0-8191, O 0-8191, F 0-8191**
	dest	**(i)**	**; Destination register R 0-4095**

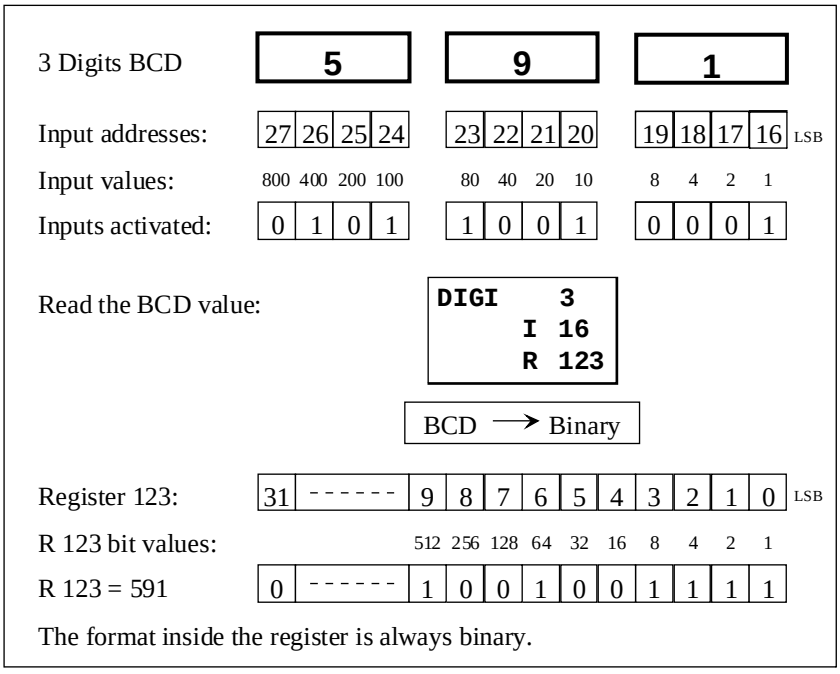
Example

```
DIGI     2     ; Reads 2 BCD digits
       I 32    ; from Inputs 39-36 and 35-32
       R 100   ; into Register 100
```

Flags **Zero** (Z) and **Sign** (P or N) flags set according to value read.

See Also DIGIR, DIGO, DIGOR, BITI, BITIR

Practice



DIGIR DIGIT IN REVERSED

Description Moves Binary Coded Decimal (BCD) digits from Inputs, Outputs or Flags into a Register.
A BCD digit is 4 bits (eg. 4 Inputs), which represents a decimal digit (0-9).
The 1st operand is the number of digits to move (1-10).
The 2nd is the base Input, Output or Flag.
The 3rd operand is the destination Register.
The lowest addressed Input, Output or Flag becomes the most significant bit of the most significant digit in the destination Register.
This is the same as the PCA.

Usage

```
DIGIR[X]  digits        ; Number of BCD digits 1-10
          source        ; Source element I 0-8191, O 0-8191, F 0-8191
          dest    (i)   ; Destination register R 0-4095
```

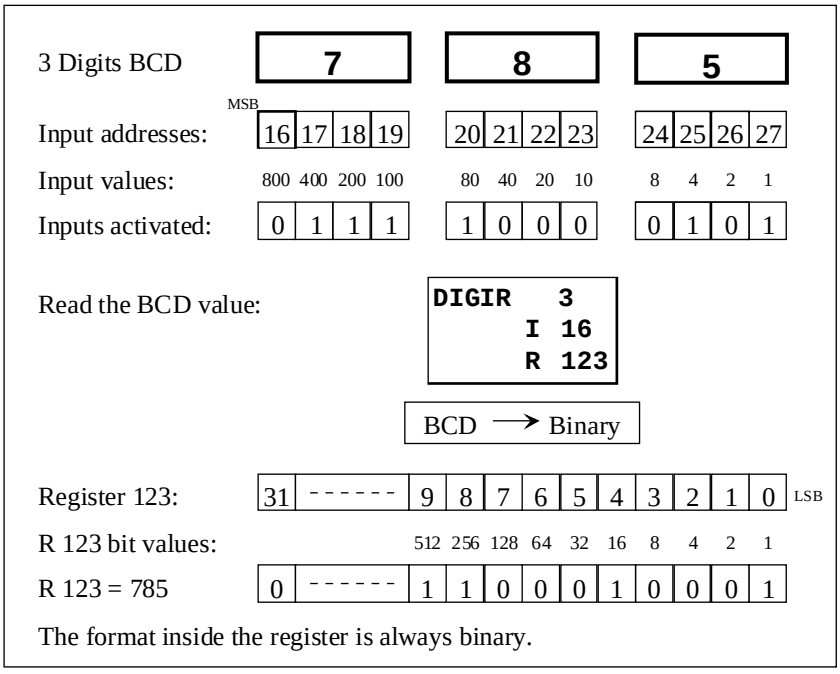
Example

```
DIGIR     2     ; Reads 2 BCD digits
        I 32    ; from Inputs 32-35 and 36-39
        R 100   ; into Register 100
```

Flags **Zero** (Z) and **Sign** (P or N) flags set according to value read.

See Also DIGI, DIGO, DIGOR, BITI, BITIR

Practice



DIGO DIGIT OUT

Description Moves BCD digits from a Register to a sequence of Outputs or Flags.
A BCD digit consists of 4 binary bits.
The 1st operand is the number of digits to move.
The 2nd is the source Register.
The 3rd operand is the base Output or Flag address.
The lowest addressed Output or Flag becomes the least significant bit of the least significant BCD digit.
This is contrary to the PCA.

Usage

```
DIGO[X]   digits          ; Number of BCD digits 1-10
          source   (i)    ; Source register R 0-4095
          dest            ; Destination element O 0-8191, F 0-8191
```

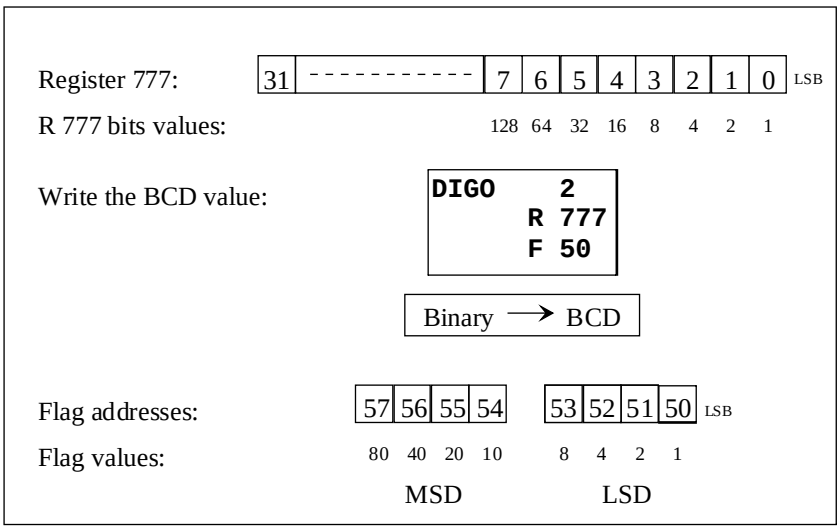
Example

```
DIGO      2     ; Outputs 2 BCD digits
        R 123   ; from Register 123
        O 40    ; to Outputs 47-44 and 43-40
```

Flags The **Error** (E) flag is set if a BCD digit is invalid (> 9).

See Also DIGOR, DIGI, DIGIR, BITO, BITOR

Practice



DIGOR DIGIT OUT REVERSED

Description Moves BCD digits from a Register to a sequence of Outputs or Flags.
A BCD digit consists of 4 binary bits.
The 1st operand is the number of digits to move.
The 2nd is the source Register.
The 3rd operand is the base Output or Flag address.
The lowest addressed Output or Flag becomes the most significant bit of the most significant BCD digit.
This is the same as the PCA.

Usage

```
DIGOR[X]  digits          ; Number of BCD digits 1-10
          source   (i)    ; Source register R 0-4095
          dest            ; Destination element O 0-8191, F 0-8191
```

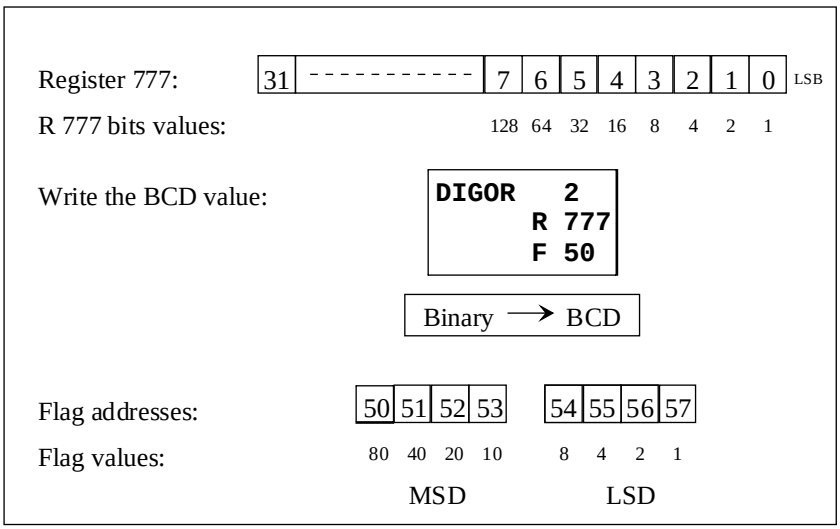
Example

```
DIGOR     2     ; Outputs 2 BCD digits
        R 123   ; from Register 123
        O 40    ; to Outputs 40-43 and 44-47
```

Flags The **Error** (E) flag is set if a BCD digit is invalid (> 9).

See Also DIGO, DIGI, DIGIR, BITO, BITOR

Practice



AND AND REGISTERS

Description The contents of the 1st Register is logically ANDed with the contents of the second register, and the result is placed in the 3rd Register.

Usage

AND [X]	**value1**	**(i)**	**; R 0-4095**
	value2		**; R 0-4095**
	result	**(i)**	**; R 0-4095**

Example

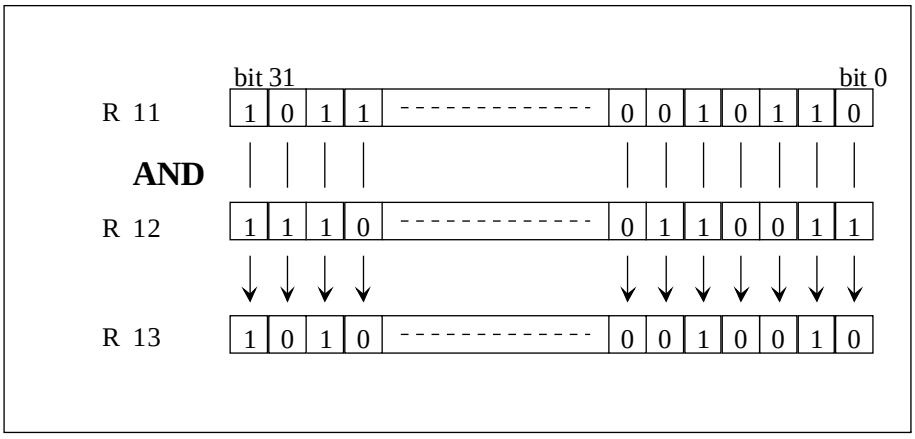
```
AND     R 11        ; ANDs R 11 and
        R 12        ; R 12 and places the result
        R 13        ;  in R 13.
```

R 13 contains a 1 bit for every 1 bit in both R 11 AND R 12

Flags The **Zero** (Z) and **Sign** (N or P) flags are set according to the result.
The **Error** (E) flag is always set Low..

See Also OR, NOT, EXOR

Practice



OR OR REGISTERS

Description The contents of the 1st Register is logically ORed with the contents of the 2nd Register, and the result is placed in the 3rd Register.

Usage

OR[X]	**value1**	**(i)**	**; R 0-4095**
	value2		**; R 0-4095**
	result	**(i)**	**; R 0-4095**

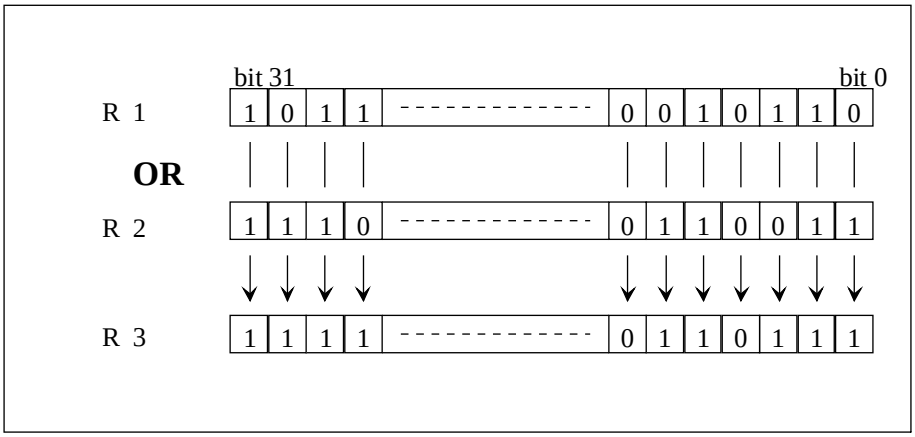
Example

```
OR      R 1     ; ORs Register 1
        R 2     ; with Register 2
        R 3     ; and puts the result in Register 3
```

Flags The **Zero** (Z) and **Sign** (P and N) flags are set according to the result.
The **Error** (E) flag is always set Low..

See Also EXOR

Practice



EXOR EXCLUSIVE-OR REGISTERS

Description The contents of the 1st Register is Exclusive ORed with the contents of the 2nd Register, and the result is placed in the 3rd Register.

Usage

EXOR[X]	**value1**	**(i)**	**; R 0-4095**
	value2		**; R 0-4095**
	result	**(i)**	**; R 0-4095**

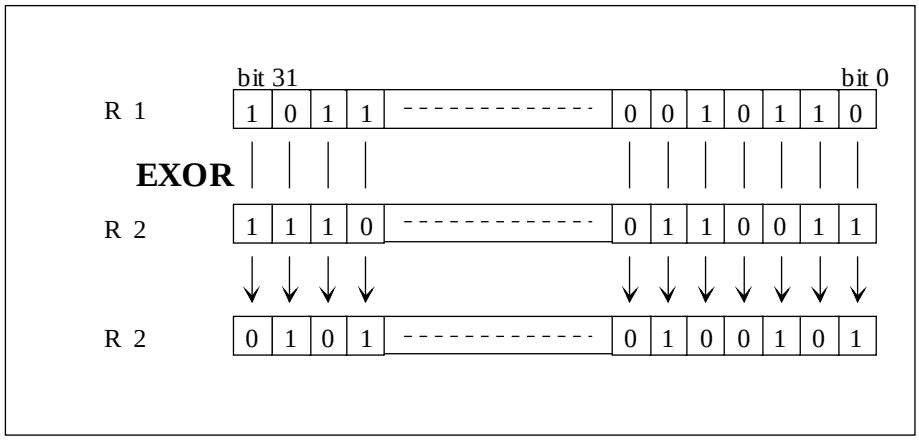
Example

```
EXOR    R 1     ; Register 1 is exclusive-Ored
        R 2     ; with Register 2 and
        R 2     ; the result is placed in Register 2
```

Flags The **Zero** (Z) and **Sign** flags (P and N) are set according to the result.
The **Error** (E) flag is always set Low..

See Also OR

Practice



NOT COMPLEMENT REGISTER

Description The contents of the 1st Register is inverted (1's complement) and stored in the 2nd Register.

Usage

NOT[X]	**value**	**(i)**	**; R 0-4095**
	result	**(i)**	**; R 0-4095**

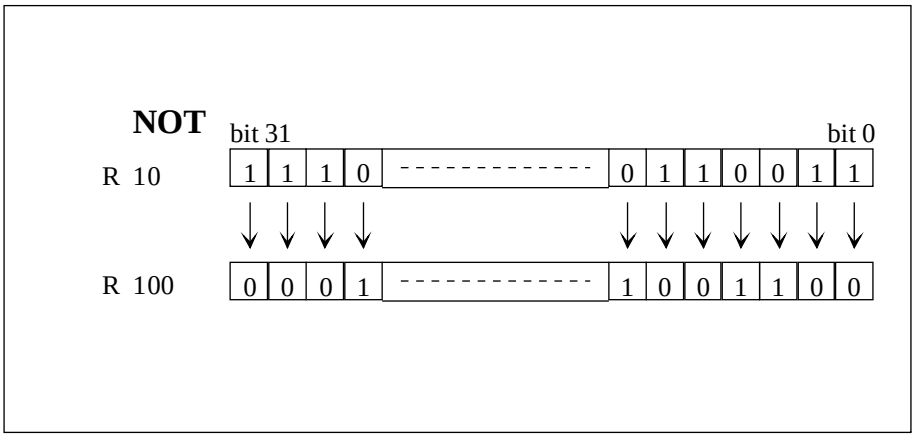
Example

```
NOT     R 10    ; Inverts the contents of Register 10
        R 100   ; and puts the result in Register 100
```

Flags The **Zero** (Z) and **Sign** (P and N) flags are set according to the result.
The **Error** (E) flag is always set Low..

See Also

Practice



SHIU SHIFT REGISTERS UP

Description Shifts the contents of a block of Registers up one place.
The 1st and 2nd operands are the start and end of the block of Registers to be shifted.
After the shift, the lowest Register contains zero, and the highest overwrites the Register above.
Either the upper or the lower Register can be specified first.

Usage

SHIU	**start**	**; R 0-4094**
	end	**; R 0-4094**

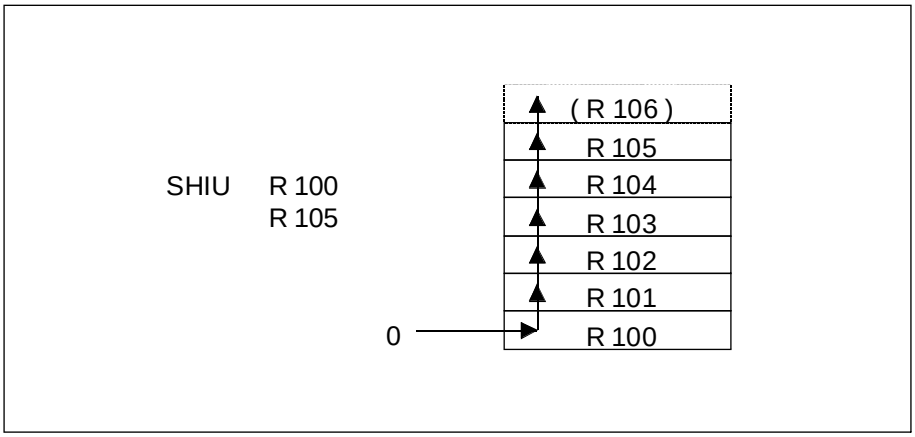
Example

```
SHIU    R 100   ; Shifts R 100 to R 105 up one address
        R 105
                ; R 100 = 0, R 101 = R 100 ... R 106 = R 105
```

Flags Unchanged.

See Also SHID, ROTU, ROTD

Practice



NOTE:
This instruction use one register more than those specified: the register which follows the end of the block is also used.

SHID SHIFT REGISTERS DOWN

Description Shifts the contents of a block of Registers down one place.
The 1st and 2nd operands are the start and end of the block of Registers to be shifted.
After the shift, the highest Register contains zero, and the lowest overwrites the Register below.
Either the upper or the lower Register can be specified first

Usage

SHID	**start**	**; R 0-4094**
	end	**; R 0-4094**

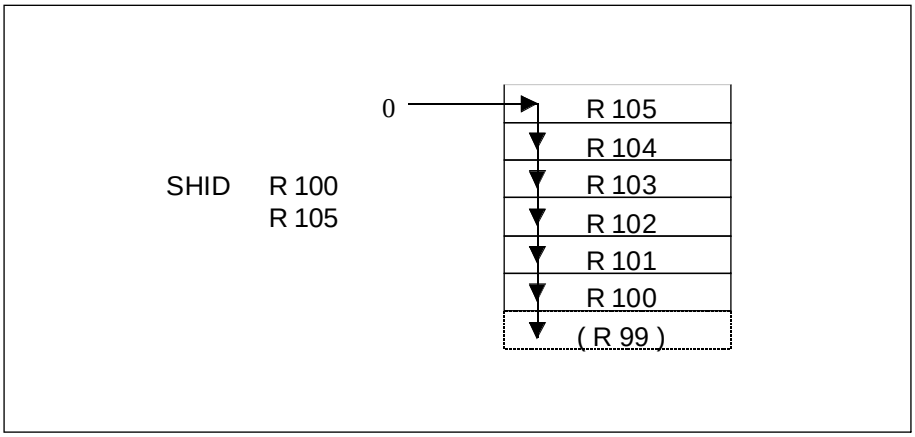
Example

```
SHID    R 100   ; Shifts R 100 to R 105 down one address
        R 105
                ; R 99 = R 100 ... R 104 = R 105, R 105 = 0
```

Flags Unchanged.

See Also SHIU, ROTU, ROTD

Practice



NOTE:
This instruction use one register more than those specified: the register which precedes the start of the block is also used.

ROTU ROTATE REGISTERS UP

Description Rotates the contents of a block of Registers up one place.
The 1st and 2nd operands indicate the start and end of the block of Registers to be rotated.
After the rotate, the lowest Register contains the value of the highest.
Either the higher or the lower Register can be specified first.

Usage

ROTU	**start**	**; R 0-4094**
	end	**; R 0-4095**

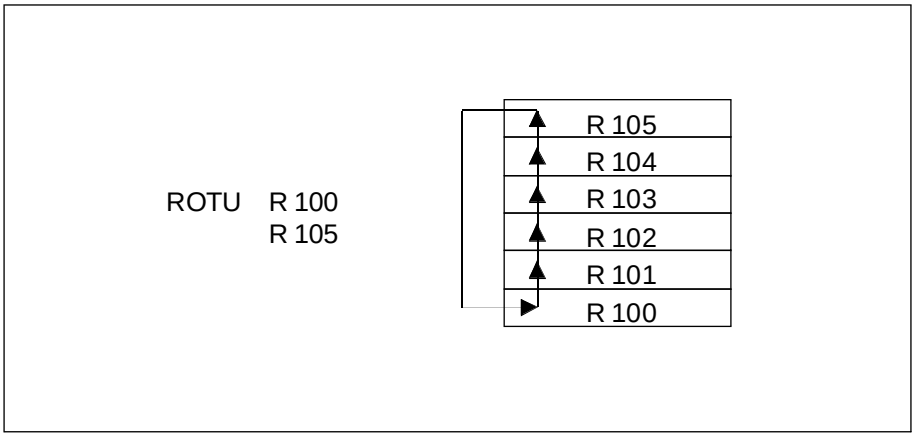
Example

```
ROTU    R 100   ; Rotates R 100 to R 105 up one address.
        R 105
                ; R 100 = R 105, R 101 = R100 ... R 105 = R 104
```

Flags Unchanged.

See Also ROTD, SHIU, SHID

Practice



ROTD ROTATE REGISTERS DOWN

Description Rotates the contents of a block of Registers down one place.
The 1st and 2nd operands indicate the start and end of the block of Registers to be rotated.
After the rotate, the highest Register contains the value of the lowest.
Either the higher or the lower Register can be specified first.

Usage

ROTD	**start**	**; R 0-4095**
	end	**; R 0-4095**

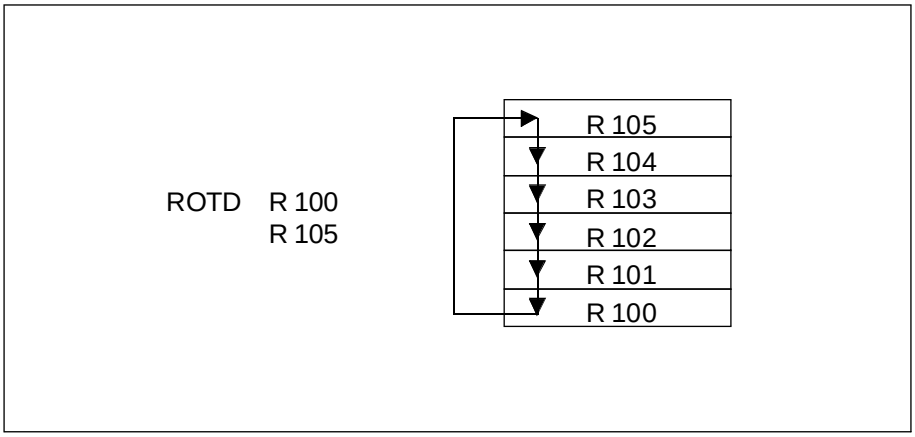
Example

```
ROTD    R 100   ; Rotates R 100 to R 105 down one address.
        R 105
                ; R 100 = R 101 ... R 104 = R 105, R 105 = R 100.
```

Flags Unchanged.

See Also ROTU, SHIU, SHID

Practice



SHIL SHIFT REGISTER LEFT

Description The contents of the addressed Register is shifted left by the number of bits given by the second operand.
The content of the ACCU (1 or 0) is shifted in from bit 0 (the least significant bit), n times.
At the end of the operation, the ACCU is set to the status of the last bit shifted out of the Register

Usage

SHIL[X]	**reg**	**(i)**	**; R 0-4095**
	n bits		**; Number of bits 1-32**

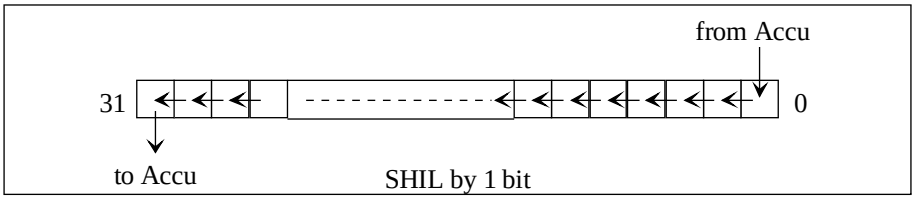
Example

```
SHIL    R 10    ; Register 10 is shifted left
          4     ; 4 bits
                ; (multiplied by 16 if ACCU was 0)
```

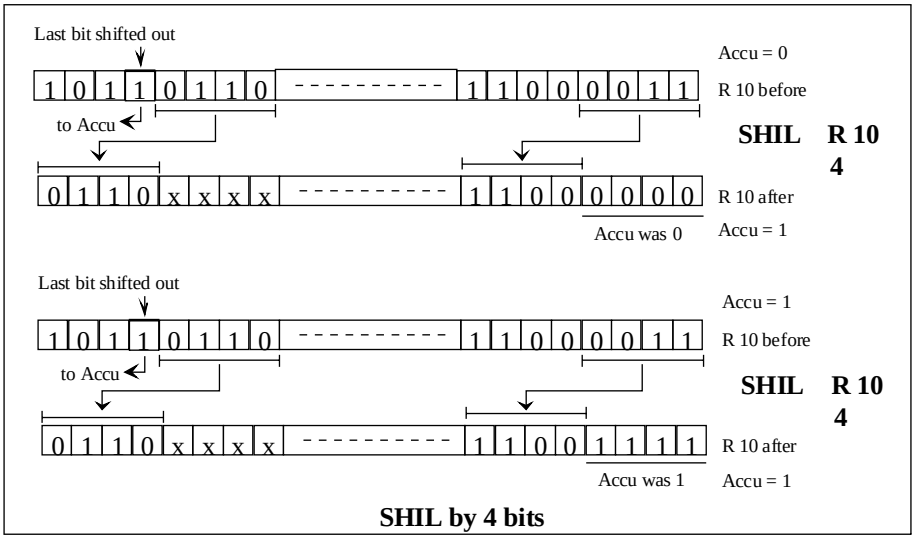
Flags The **ACCU** is set to the status of the last bit shifted out of the Register

See Also SHIR, ROTL, ROTR

Practice



SHIL by 1 bit



SHIL by 4 bits

SHIR SHIFT REGISTER RIGHT

Description The contents of the addressed Register is shifted right by the number of bits given by the second operand.
The contents of the ACCU (1 or 0) is shifted in from bit 31 (the most significant bit), n times.
At the end of the operation, the ACCU is set to the status of the last bit shifted out of the Register.

Usage

```
SHIR[X]   reg      (i)     ; R 0-4095
          n bits           ; Number of bits 1-32
```

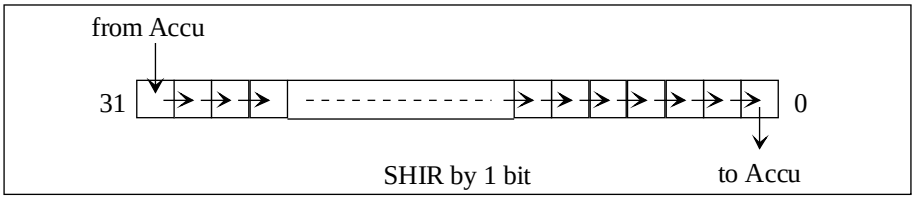
Example

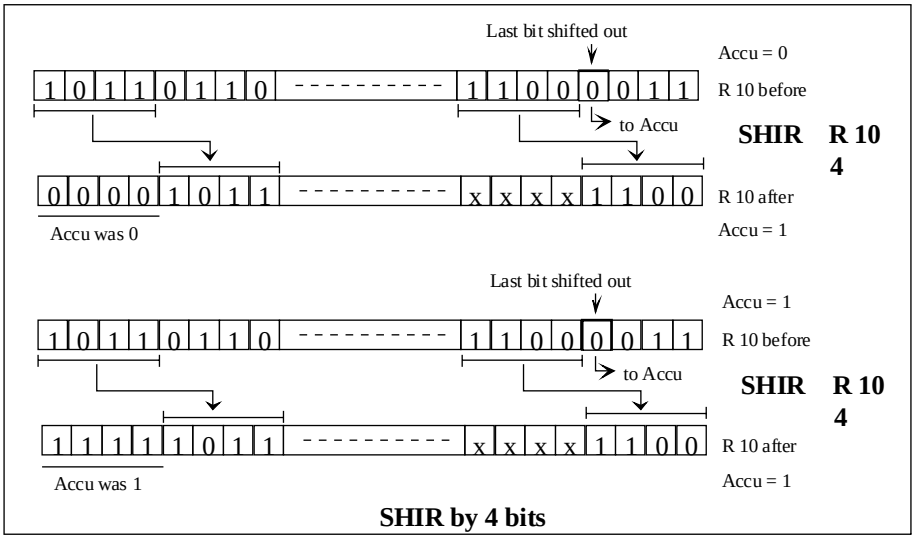
```
SHIR    R 10    ; Register 10 is shifted right
          16    ; 16 bits
                ; (divided by 65536 if ACCU was 0)
```

Flags The **ACCU** is set to the status of the last bit shifted out of the Register.

See Also SHIL, ROTL, ROTR

Practice





ROTL ROTATE REGISTER LEFT

Description The contents of the addressed Register is rotated left by the number of bits given in the 2nd operand.
The most significant bit 31 is copied into the least significant bit 0.
The ACCU is set to status of the last bit that was rotated.

Usage

```
ROTL[X]   reg      (i)     ; R 0-4095
          n bits           ; Number of bits 1-32
```

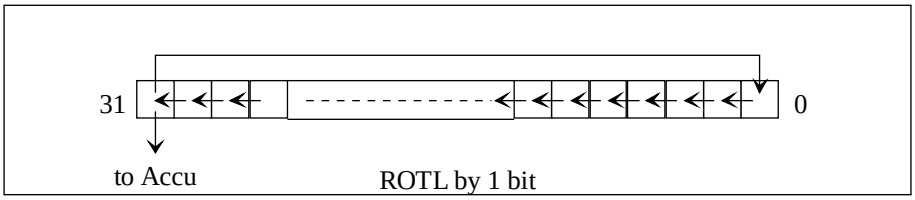
Example

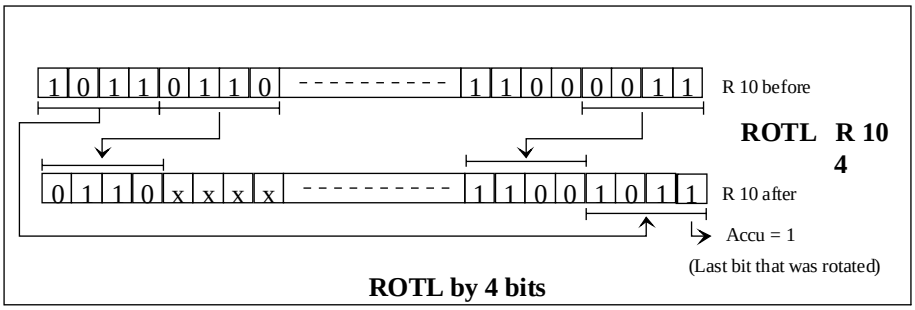
```
ROTL      R 10     ; Register 10 is rotated left 4 bits
            4
```

Flags The **ACCU** is set to status of the last bit that was rotated.

See Also ROTR, SHIL, SHIR

Practice





ROTR ROTATE REGISTER RIGHT

Description The contents of the addressed Register is rotated right by the number of bits given in the 2nd operand.
The least significant bit 0 is copied into the most significant bit 31.
The ACCU is set to status of the last bit that was rotated.

Usage

ROTR[X]	**reg**	**(i)**	**; R 0-4095**
	n bits		**; Number of bits 1-32**

Example

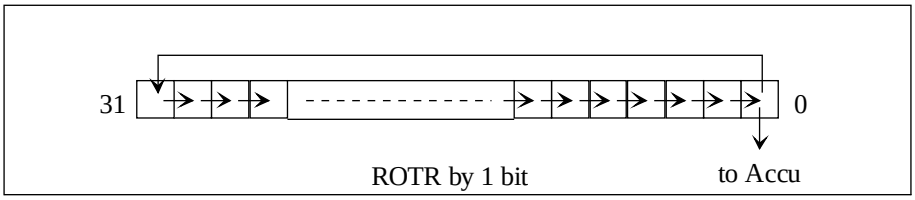
```
ROTR    R 10
        4
```

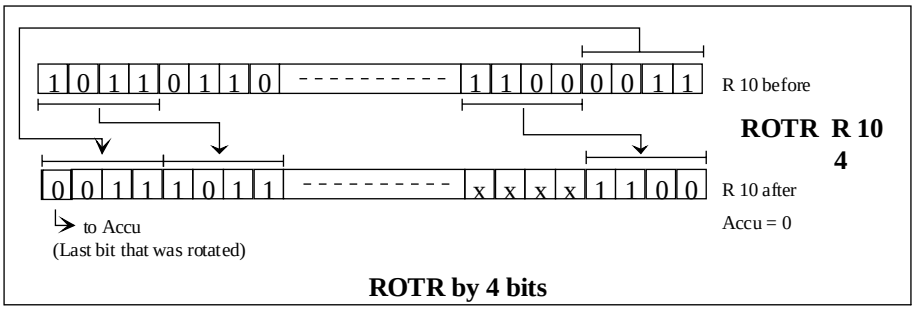
; Register 10 is rotated right 4 bits

Flags The **ACCU** is set to status of the last bit that was rotated..

See Also ROTR, SHIL, SHIR

Practice





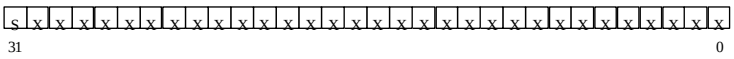
4. INTEGER arithmetic

The integer arithmetic instructions work only with registers.

ADD	Add Registers
SUB	Subtract Registers
MUL	Multiply Registers
DIV	Divide Registers
SQR	Square Root
CMP	Compare Registers

For floating point values, the Floating point instructions must be used.

The integer format is based on 32 bits with the following format:



where X: bit value (0 or 1)
S: sign

Bits 0 to 30 are the integer value in binary format.
Bit 31 is the sign bit (0 = positive value, 1 = negative value)

The range allowed by this format is as follows:

Decimal	2.147.483.647	to	-2.147.483.648
Binary (hexadecimal)	7FFF'FFFF	to	8000'0000

INTEGER format

Notes

Notes

ADD ADD REGISTERS

Description Integer addition.
Adds the contents of the 1st Register or constant to the contents of the 2nd Register or constant, and stores the result in the 3rd Register.

Usage

ADD	**value1**	**; R 0-4095, K 0-16383**
	value2	**; R 0-4095, K 0-16383**
	result	**; R 0-4095**

Example

```
ADD     R 20    ; Adds 123 to Register 20
        K 123
        R 20
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set on overflow.

See Also FADD (floating point add)

Practice Read two numbers, add them and put the result in another register.
The two numbers come from BCD encoders (2 digits) on inputs 16 to 23, respectively 24 to 31

```
COB       0     ; COB Header
          0

DIGI      2     ; Read 2 digits
        I 16    ;    from input 16 (to 23)
        R 100   ;    and store them in R 100

DIGI      2     ; Read 2 digits
        I 24    ;    from input 24 (to 31)
        R 200   ;    and store them in R 200

ADD     R 100   ; R 0 = R 100 + R 200
        R 200
        R 0

ECOB
```


SUB SUBSTRACT REGISTERS

Description Integer subtraction.
Subtracts the contents of the 2nd Register or K constant from the contents of the 1st Register or K constant, and stores the result in the 3rd Register.

Usage

SUB	**value1**	**; R 0-4095, K 0-16383**
	value2	**; R 0-4095, K 0-16383**
	result	**; R 0-4095**

Example

```
SUB     R 1     ; Register 3 = Register 1 minus Register 2
        R 2
        R 3
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set on underflow.

See Also ADD, FSUB

Practice Read two numbers, subtract them and put the result in another register.
The two numbers come from BCD encoders (2 digits) on inputs 16 to 23, respectively 24 to 31

```
COB       0     ; COB Header
          0

DIGI      2     ; Read 2 digits
        I 16    ;     from input 16 (to 23)
        R 10    ;     and store them in R 10

DIGI      2     ; Read 2 digits
        I 24    ;     from input 24 (to 31)
        R 11    ;     and store them in R 11

SUB     R 10    ; R 12 = R 10 - R 11
        R 11
        R 12

ECOB
```


MUL MULTIPLY REGISTERS

Description Integer multiplication.
Multiplies the contents of the 1st Register or K constant by the contents of the 2nd Register or K constant, and stores the result in the 3rd Register.

Usage

MUL	**value1**	**; R 0-4095, K 0-16383**
	value2	**; R 0-4095, K 0-16383**
	result	**; R 0-4095**

Example

```
MUL     R 0     ; Multiplies Register 0 by 10
        K 10
        R 0
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set on overflow.

See Also DIV, FMUL

Practice Read two numbers, multiply them and put the result in another register.
The two numbers come from BCD encoders (2 digits) on inputs 16 to 23, respectively 24 to 31

```
COB         0       ; COB Header
            0

DIGI        2       ; Read 2 digits
          I 16      ;     from input 16 (to 23)
          R 50      ;     and store them in R 50

DIGI        2       ; Read 2 digits
          I 24      ;     from input 24 (to 31)
          R 55      ;     and store them in R 55

MUL       R 50      ; R 4000 = R 50  * R 55
          R 55
          R 4000

ECOB
```


DIV DIVIDE REGISTERS

Description Integer division.
Divides the contents of the 1st Register or K constant by the contents of the 2nd Register or constant, and stores the result in the 3rd Register.
The remainder is placed in the 4th Register.

Usage

DIV	**value 1**	**; R 0-4095, K 0-16383**
	value 2	**; R 0-4095, K 0-16383**
	result	**; R 0-4095**
	remainder	**; R 0-4095**

Example

```
DIV     R 20    ; R 20 is
        K 1000  ;  divided by 1000
        R 21    ; the result is placed in Register 21
        R 1     ; and the remainder in Register 1
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set on divide by zero.

See Also FDIV

Practice Read two numbers, divide them and put the result in another register.
The two numbers come from BCD encoders (2 digits) on inputs 16 to 23, respectively 24 to 31

```
COB       0     ; COB Header
          0

DIGI      2     ; Read 2 digits
        I 16    ;     from input 16 (to 23)
        R 1     ;     and store them in R 1
DIGI      2     ; Read 2 digits
        I 24    ;     from input 24 (to 31)
        R 2     ;     and store them in R 2
DIV     R 1     ; R 100 = R 1 / R 2
        R 2
        R 100   ;     result
        R 101   ;     remainder
CPB     E 99    ; If Error Flag is set,
                ;     then call program block 99
ECOB

PB        99
SET     O 47    ; Alarm if division by zero (output 47)
EPB
```

SQR SQUARE ROOT

Description Integer Square Root.
The integer square root of the contents of the 1st Register is stored in the 2nd Register.
If the 1st Register contains a negative value, the Error flag is set and the operation is not performed.

Usage

SQR	**value**	**; R 0-4095**
	result	**; R 0-4095**

Example

```
SQR     R 0     ; The square root of Register 0 is
        R 100   ; placed in Register 100
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set on an attempt to obtain the square root of a negative value

See Also FSQR

Practice Extract the square rooot of a number read on a BCD encoders (4 digits) on inputs 16 to 31.

```
COB       0       ; COB Header
          0

DIGI      4       ; Read 4 digits
        I 16      ;    from input 16 (to 31)
        R 100     ;    and store them in R 100

SQR     R 100     ; R 101 = √ R 100
        R 101

ECOB
```


CMP COMPARE REGISTERS

Description

Compares the contents of the 1st Register or constant with the contents of the 2nd Register or constant. This is done by subtracting the 2nd value from the 1st value, the Status flags are set according to the result.
The contents of the Registers are unchanged

Usage

```
CMP[X]    value 1   (i)    ; R 0-4095, K 0-16383
          value 2          ; R 0-4095, K 0-16383
```

Example

```
CMP       R 0      ; Compares Register 0 with Register 1
          R 1      ; setting the status flag according to the result
```

Flags

The Zero, Positive and negative Flags are set according the following table

	Z	P	N
Value 1 = Value 2	High	High	Low
Value 1 > Value 2	Low	High	Low
Value 1 < Value 2	Low	Low	High

See Also

AND, OR, EXOR, FCMP

Practice

Read two numbers; if the first number is greater, equal or lower than the second one the output 32 (respectively 33 or 34) must be activated.
The two numbers come from BCD encoders (2 digits) on inputs 16 to 23, respectively 24 to 31

```
COB         0      ; COB Header
            0
DIGI        2      ; Read 2 digits
          I 16     ;    from input 16 (to 23)
          R 1      ;    and store them in R 1
DIGI        2      ; Read 2 digits
          I 24     ;    from input 24 (to 31)
          R 2      ;    and store them in R 2
CMP       R 1      ; Compare R 1 with R 2
          R 2
ACC       Z        ; If R1 = R2
OUT       O 33     ;    Then set output 33 and flag 0
OUT       F 0      ;         Else reset output 33 and flag 0
ACC       N        ; If R 1 < R2
OUT       O 34     ;    Then set output 34, else reset output 34
ACC       P        ; If R1 > R2
ANL       F 0      ;         (and not equal)
OUT       O 32     ;    Then set output 32, else reset output 32
ECOB
```


5. FLOATING POINT arithmetic

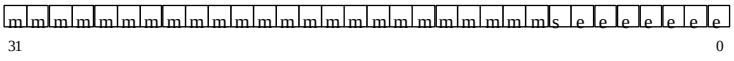
Floating point values can only be stored in Registers.
They can be loaded into Registers using the LD instruction.
To specify a floating point number, the number must include a decimal point '.' or an exponent 'E'. For example: 1.2 1E3, -4.656E-2.
The range for floating point numbers is:

+ 5.42101E-20 ... + 9.22337E+18 (accurate to 5 significant digits)
- 2.71056E-20 ... - 9.22337E+18

IFP	Integer to floating point
FPI	Floating point to integer
FADD	Floating point add
FSUB	Floating point subtract
FMUL	Floating point multiply
FDIV	Floating point divide
FSQR	Square root
FCMP	Floating point compare
FSIN	Sine function
FCOS	Cosine function
FATAN	Arc tangent
FEXP	Exponential function
FLN	Logarithm function
FABS	Absolute value

NOTE:
Floating point values are stored in Registers in a special binary format, using the value as an integer will yield incorrect results.
Mixing integer and floating point values in arithmetic operations gives invalid results. The integer values must first be converted to floating point with the IFP instruction.
Floating point values can be converted to integer with the FPI instruction.

The floating point format is based on 32 bits with the following format:



where m: 24-bit mantissa
s: sign of the number
e: 7-bit exponent in excess 64 notation

The sign bit is a 0 for a positive value, and a 1 for a negative value.
The mantissa is considered to be a binary fixed point fraction and except for 0 it is always normalized (has one bit in its highest position).
The exponent is the power of two needed to correctly position the mantissa to reflect th number's true arithmetic value. It is held in excess 64 notation which means that the two's complement values are adjusted upward by 64.

Floating Point Format

Notes

Notes

IFP INTEGER TO FLOATING POINT

Description Converts the integer value in the specified Register to floating point format. The 2nd operand indicates the power of ten to which the integer is to be raised, this controls the position of the decimal point.

For example, if the power of ten is +3, the contents of the Register is multiplied by 1000 (10^3), and the result is stored in the Register in floating point format.
If the Register contained 12, the result would be 12000.00.

If the conversion is not possible (number too big or too small), the Error flag is set and no conversion is done.

Usage

IFP[X]	**reg**	**(i)**	**; R 0-4095**
	power		**; Power of ten -20 to +18**

Example

```
IFP     R 0     ; R 0 = Floating point value of R 0 * 10^3
        3
```

Flags The **Error** (E) flag is set if conversion is not possible.

See Also FPI

Practice

R 500 Before	Instruction	Conversion	R 500 After
123	IFP R 500 0	R 500 * 10^0	1.23E+2
	IFP R 500 -2	R 500 * 10^{-2}	1.23E+0
	IFP R 500 3	R 500 * 10^3	1.23E+5

FPI FLOATING POINT TO INTEGER

Description Converts the floating point value in the specified Register to integer format.
The 2nd operand indicates the power of ten to be used in the conversion.
The result is the integer of the result of the Register contents multiplied by 10 to the power of the 2nd operand.

For example, if the Register contains 1234.56 and the power of ten is -2, the integer result will be 12.
If the conversion is not possible, the Error flag is set and nothing is done.

Usage

FPI[X]	**reg**	**(i)**	**; R 0-4095**
	power		**; Power of ten -20 to +18**

Example

```
FPI     R 0     ; If R 0 contains 1234.56, it is converted
        0       ; to the integer value 1234 (power of ten is zero)
```

Flags The **Error** (E) flag is set if conversion is not possible.

See Also IFP

Practice

R 500 Before	Instruction	Conversion	R 500 After
123.456	FPI R 500 0	R 500 * 10^{0}	123
	FPI R 500 -2	R 500 * 10^{-2}	1
	FPI R 500 3	R 500 * 10^{3}	123456

FADD FLOATING POINT ADD

Description Adds the contents of the 1st Register to the contents of the 2nd Register, and stores the result in the 3rd Register.
The Registers must contain valid floating point format values.

Usage

```
FADD    reg1      ; R 0-4095
        reg2      ; R 0-4095
        result    ; R 0-4095
```

Example

```
FADD    R 100     ; R 500 = R 100 + R 101
        R 101
        R 500
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set on overflow

See Also ADD

Practice

FSUB FLOATING POINT SUBTRACT

Description Subtracts the contents of the 2nd Register from the contents of the 1st Register, and stores the result in the 3rd Register.
Both Registers must contain valid floating point format values.

Usage

```
FSUB    reg1        ; R 0-4095
        reg2        ; R 0-4095
        result      ; R 0-4095
```

Example

```
FSUB    R 0     ; R 0 = R 0 - R 1
        R 1
        R 0
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set on overflow.

See Also SUB

FMUL FLOATING POINT MULTIPLY

Description Multiplies the contents of the 1st Register by the contents of the 2nd Register, and stores the result in the 3rd Register.
Both Registers must contain valid floating point format values.

Usage

```
FMUL    reg1        ; R 0-4095
        reg2        ; R 0-4095
        result      ; R 0-4095
```

Example

```
FMUL    R 20    ; R 0 = R 20 * R 30
        R 30
        R 0
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set on overflow, e.g. if the result is greater than 4.611686 E+18

See Also MUL

FDIV FLOATING POINT DIVIDE

Description Divides the contents of the 1st Register by the contents of the 2nd Register, and stores the result in the 3rd Register.
Divide by zero sets the Error flag, and the operation is not performed.
NOTE: Since Floating Point arithmetic is more exact, there is no remainder

Usage

```
FDIV    reg       ; R 0-4095
        divisor   ; R 0-4095
        result    ; R 0-4095
```

Example

```
FDIV    R 1     ; R 3 = R 1 / R 2
        R 2
        R 3
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set on divide-by-zero.

See Also DIV

FSQR FLOATING POINT SQUARE ROOT

Description Stores the square root of the contents of the 1st Register into the 2nd Register. If the 1st Register contains a negative value, the Error flag is set, and the square root of the absolute (+ve) value is taken.

Usage

```
FSQR    reg       ; R 0-4095
        result    ; R 0-4095
```

Example

```
FSQR    R 0     ; R 0 = Square root of R 0
        R 0
```

Flags The **Zero** (Z) flag is set according to the result.
The result is always positive, P is always set High, N is always set Low.
If the value was negative, the **Error** (E) flag is set.

See Also SQR

FCMP FLOATING POINT COMPARE

Description Compares the contents of the 1st Register with the contents of the 2nd Register and sets the Status flags according to the result.
Neither of the Registers are altered.
Both Registers must contain valid floating point format values.

Usage

```
FCMP[X]   reg1    (i)    ; R 0-4095
          reg2           ; R 0-4095
```

Example

```
FCMP    R 0     ; Compares R 0 and R 1, setting the Status
        R 1     ;  flags according to the result
```

Flags The Status flags are set as follows:

	Z	P	N
Value 1 = Value 2	High	High	Low
Value 1 > Value 2	Low	High	Low
Value 1 < Value 2	Low	Low	High

The **Error** (E) flag is set low

See Also CMP

Note **NEVER compare Floating Point values for equality, use > or < to avoid accuracy errors !**

FSIN SINE FUNCTION

Description The sine of the contents of the 1st Register is stored in the 2nd Register. The 1st Register must contain a floating point value in RADIANS in the range of $\pm 10^6$

Usage

FSIN[X]	**reg**	**(i)**	**; R 0-4095**
	result	**(i)**	**; R 0-4095**

Example

```
FSIN    R 0     ; R 100 = Sine of R 0
        R 100
```

Flags The **Sign** (Z) and **Status** (N or P) flags are set according to the result

See Also FCOS, FATAN

FCOS COSINE FUNCTION

Description The cosine of the contents of the 1st Register is stored in the 2nd Register. The 1st Register must contain a floating point value in RADIANS in the range of $\pm 10^6$.

Usage

FCOS[X]	**reg**	**(i)**	**; R 0-4095**
	result	**(i)**	**; R 0-4095**

Example

```
FCOS    R 100    ; R 20 = Cosine of R 100
        R 20
```

Flags The **Sign** (Z) and **Status** (N or P) flags are set according to the result.

See Also FSIN, FATAN

FATAN ARC TANGENT

Description The arc tangent of the contents of the 1st Register is stored in the 2nd Register.
The 1st Register must contain a valid floating point value in RADIANS.
The result is in the second Register will range from $-\pi/2$ to $+\pi/2$

Usage

FATAN[X]	**reg**	**(i)**	**; R 0-4095**
	result	**(i)**	**; R 0-4095**

Example

```
FATAN   R 1     ; R 0 = Arc tangent of R 1
        R 0
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.

See Also FSIN, FCOS

FEXP EXPONENTIAL FUNCTION

Description 'e' to the power of the contents of the 1st Register is stored in the 2nd Register. The Register must contain a valid floating point format value.

Usage

FEXP[X]	**reg**	**(i)**	**; R 0-4095**
	result	**(i)**	**; R 0-4095**

Example

```
FEXP    R 0     ; R 1 = e^R0
        R 1
```

Flags The **Zero** (Z) flag is set according to the result, the **Negative** (N) flag is always set Low (P = 1).
The **Error** (E) flag is set on overflow .

See Also FPI, IFP

FLN FLOATING POINT LOGARITHM FUNCTION

Description The natural log of the contents of the 1st Register is stored in the 2nd Register.
The 1st Register must contain a valid floating point format value.
If the natural log of a negative value is taken, the Error flag is set and the log of the absolute (+ve) value is taken.

Usage

FLN[X]	**reg**	**(i)**	**; R 0-4095**
	result	**(i)**	**; R 0-4095**

Example

```
FLN     R 1     ; R 2 = ln R 1
        R 2
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the result.
The **Error** (E) flag is set if the "ln" of zero or a negative value is taken

See Also FEXP

FABS FLOATING POINT ABSOLUTE VALUE

Description The absolute value (converted to positive if it is negative) of the 1st Register is stored in the 2nd Register. The 1st Register must contain a valid floating point format value.

Usage

FABS[X]	**reg**	**(i)**	**; R 0-4095**
	result	**(i)**	**; R 0-4095**

Example

```
FABS    R 1     ; R 2 = absolute value of R 1
        R 2     ; If R 1 contains -7.5 then R 2 = 7.5
```

Flags The **Zero** (Z) flag is set according to the result.

6. BLOCTEC Instructions

Bloctec is a structured programming method which breaks a program down into separate blocks of code.
A **Cyclic Organisation Block** (COB) is a main block of code which will typically call **Program Blocks** (PB), which in turn will call **Function Blocks** (FB).
At least one COB (COB 0) must be present in the program.
COBs can call PBs or FBs (with optional parameters).
PBs and FBs can themselves call any other PB or FB up to a nesting depth of 7.
The operand(s) of these instructions cannot be supplied as Function Block parameter(s).

For more information about the structured programming methods using BLOCTEC refer to "The Structured Programming" in the User's Guide.

COB	Cyclic Organisation Block
ECOB	End of Cyclic Org'n Block
XOB	Exception Organisation Block
EXOB	End of Exception Org'n Block
PB	Program Block
EPB	End of Program Block
CPB	Call Program Block
CPBI	Call Program Block Indirect
FB	Function Block
EFB	End of Function Block
CFB	Call Function Block
NCOB	Next Cyclic Org'n Block
SCOB	Stop Cyclic Org'n Block
CCOB	Continue Cyclic Org'n Block
RCOB	Restart Cyclic Org'n Block

The following BLOCTEC instructions are potentially highly dangerous:

RCOB, NCOB, SCOB, CCOB and the COB supervision time.

If these instructions are used in a program which uses GRAFCET then serious problems can arise.
If they are not used with the utmost care: these instructions can cause at best, the user program to become slow, or at worst, cause a complete desynchronisation of the GRAFTEC and a CRASH.
Avoid to use these instructions within a GRAFTEC structure.

Notes

Notes

COB CYCLIC ORGANISATION BLOCK

Description

Starts the specified Cyclic Organisation Block.
The 2nd operand is the COB supervision time, in 10 millisecond increments.
If the supervision time elapses before the COB has finished execution (ECOB reached), the Exception XOB 11 is executed if it is present; if not present, the next COB is started.
If the supervision time is 0, XOB 11 is never executed, the next COB is started only when this COB has ended (the ECOB is reached).
If several COBs are programmed, they run one after the other in numerical order.

The ACCU is always set High (1) at the start of each COB.

Note: The COB instruction needs 3 program lines.

Usage

```
COB     number      ; COB number 0-15
        time        ; Supervision time in 10ms increments
0-100000
```

Example

```
COB     0       ; Start of COB 0
        0       ;   Supervision time = 0
        ....    ;   Body of COB 0
        ....
        ....
ECOB            ; End of COB 0
```

Flags

The **ACCU** is set High (1) at the start of the COB.

See Also

ECOB, NCOB, RCOB, SCOB, XOB, User's Guide

ECOB END OF ORGANISATION BLOCK

Description Ends the current COB. The next COB (if present) will begin execution.
A COB body must always be terminated by an ECOB instruction.

Usage

ECOB	**; No operand required**

Example

```
COB     0       ; Start of COB 0
        0
        ...     ;       body of COB
        ...
        ...
ECOB            ; End of COB
```

Flags Unchanged.

See Also COB, User's Guide

XOB EXCEPTION ORGANISATION BLOCK

Description Marks the beginning of an Exception Organisation Block (XOB).

Usage

XOB	**number**	**; XOB number 0-30**

Example

```
XOB     16      ; Cold start XOB
        ...     ;  body of XOB
EXOB            ; End of XOB
```

Flags The **ACCU** is set High (1) at the start of the XOB

See Also EXOB, User's Guide

An Exception Organisation Block (XOB) is called when an error or another important event occurs. If during the execution of a XOB another XOB with the same or a lower priority is called, it is ignored.
The XOB can contain program code to handle these events. If no associated XOB is present, no action is taken (the event is ignored) and the ERROR lamp will be switched on.
At the end of the XOB, the exception routine will return to the location from where it was called.

Each XOB has a specific function:

XOB	Description	Priority
0	Power down	4
8	Invalid Opcode	4
7	System Overload	3
11	COB supervision time exceeded	3
14	Cyclic XOB	3
15	Cyclic XOB	3
17	S-Bus XOB Interrupt Request	3
18	S-Bus XOB Interrupt Request	3
19	S-Bus XOB Interrupt Request	3
20	Interrupt input INB1	3
25	Interrupt input INB2	3
1	Power down in extension rack	2
2	Low battery	2
4	Parity error on main bus (PCD6 only)	1
5	No response from I/O module	1
6	External error	1
9	Too many active tasks (GRAFTEC)	1
10	PB / FB nesting depth overflow	1
12	Index register overflow	1
13	Error flag set	1
16	Executed on PCD start-up	1
30	RIO connection master ↔ slaves	1

Level 4 exceptions:

Priority level 4 is the highest priority, only XOB 0 and 8 can interrupt execution of another XOB.

XOB 0 Power Down

There can be up to 10ms between the call of XOB 0 and the final loss of power to the CPU to give the user time to perform some urgent saves of values.
If the XOB 0 is programmed then the message "XOB 0 START EXEC" is written into the history list at the start of the XOB and "XOB 0 EXECUTED" upon completion of the XOB, this indicates to the user that the XOB completed before power was lost.
If the XOB is not programmed then a restart cold is immediately performed upon detection of the power down. If the XOB is programmed then a restart cold is performed upon completion of the XOB if there is still power.

XOB 8 Invalid Opcode

XOB 8 is called when the firmware detects an invalid instruction in the user program.

Level 3 exceptions:

If a level 2 or 3 exception occurs during execution of a lower priority XOB, then it will be treated directly after execution of the current level XOB.
XOB 20/25/11 have been given a higher priority level so that if the XOB is provoked during execution of a lower or equal priority then it will be executed directly after completion of the current XOB.

XOB 7 System Overload

The queuing mechanism for the level 3 XOB's has overloaded.

XOB 11 COB Supervision Time exceeded

If the second line of the COB instruction indicates a monitoring time (in 1/100 seconds) and if COB processing time exceeds this defined duration, XOB 11 is called. COB processing time is the time which can elapse between the COB and ECOB instructions.

XOB 14 Cyclic XOB
XOB 15

XOB 14 and 15 are called periodicaly with a frequency ranging from 5 ms to 1000s. This frequency can be set by using the SYSWR instruction.

XOB 17 S-Bus XOB Interrupt Request
XOB 18
XOB 19

These three XOB can be used as interrupt routines. Their execution can be started via the S-Bus network; it is also possible to start their execution with the SYSWR instruction.

XOB 20 Input Interrupt
XOB 25
XOB 20 (resp 25) is called when interrupt input INB1 (resp INB2) of the PCD1/2 has detected a rising edge (see PCD1/2 hardware manual for further details).

Level 2 exceptions:

XOB 1 Power down in extension rack
The voltage monitor in the supply module of an extension rack (PCD 2 or PCD6) detected an excessive drop in voltage.
In this case all Outputs of the extension rack are set low within 2ms and XOB 1 is invoked.
If Outputs from this "dead" extension rack continue to be handled (set, reset or polled) by the user program in any CPU, XOB 4 and/or XOB 5 are also invoked. (Only PCD4). XOB 1 will be called once up to 250 ms after detection of the error.

XOB 2 Battery failure or low battery
The battery is low, has failed or is missing.
Information in non-volatile Flags, Registers or the user program in RAM as well as the hardware clock may be altered.
XOB 2 is only called by CPU 0 every 250 ms in the evenet of this error.

Level 1 exceptions:

Any level 1 exception which occurs during another exception will never be treated.

XOB 4 Parity Failure
XOB 4 can only be invoked with PCD having extension racks (PCD6 only).
The monitor circuit of the address bus has noticed a parity error. This can either arise from a faulty extension cable, a defective extension rack or from a bus extension module, or else it is simply because the extension rack addressed is not present.

XOB 5 No response from I/O module (I/O Quit Failure)
The PCD's Input and Output modules return a signal to the CPU which has addressed them. If this signal is not returned, XOB 5 is called.
Generally, this invocation occurs if the module is not present, but it can also happen in case of faulty address decoding on the module.
For a PCD4 module with only 8 elements, XOB 5 is not called if one of the absent elements is addressed, since this address is still decoded and the signal is sent.

On the PCD1 and 2, this mechanism is not implemented.

XOB 6 External error

Not used. (Foreseen for intelligent modules of the PCD6)

XOB 9 Too many Graftec tasks
More than 32 GRAFTEC branches were simultaneously activated in a Sequential Block (SB).

XOB 10 More than 7 nested PB/FB calls
PBs and FBs can be nested to a depth of 7 levels. An additional call (calling the 8th level) results in XOB 10 executing.
The 8th level call is not executed.

XOB 12 Index Register overflow
If a program contains an indexed element which falls outside its address range (0 to 8191), then XOB 12 is called.

XOB 13 Error Flag
XOB 13 is always called when the Error flag is set, irrespective of whether the cause is a calculation, data transfer or communications error.

XOB 16 Cold Start
XOB 16 is the start-up XOB (Cold Start XOB), and is executed when the PCD is switched on, or is given a cold restart. XOB 16 can initialise any elements before the program begins. If during the execution of the XOB 16 an error occurs, the XOB 13 is not called.

XOB 30 RIO connection master ↔ slaves
After every message sent from the master to a slave, the connection is tested. If the test is not answered positively by the slave, the master CPU calls XOB 30. This is essentially the case when, online, a station is removed from the network or closed down.

EXOB END OF EXCEPTION ORGANISATION BLOCK

Description Ends the current XOB.
At the EXOB instruction, the XOB returns to the location from where it was called.

Usage

EXOB ; No operand required

Example

```
XOB     16      ; Start of XOB 16
        ...     ;       Body of XOB 16
        ...
        ...
EXOB            ; End of XOB 16
```

Flags Unchanged.

See Also XOB, User's Guide

PB PROGRAM BLOCK

Description Marks the beginning of a Program Block (PB), a subroutine without parameters.

Usage

PB	**number**	**; PB number 0-299**

Example

```
PB      26      ; Start of PB 26
        ...     ;     Body of PB 26
        ...
        ...
EPB             ; End of PB 26
```

Flags The **ACCU** is set High (1) at the start of the PB

See Also EPB, CPB, FB, User's Guide

EPB END OF PROGRAM BLOCK

Description Ends the current Program Block (PB).
A return is made to the instruction after the Call Program Block (CPB) instruction.

Usage

EPB	**; No operand required**

Example

```
PB      0       ; Start of PB 0
        ...     ;     Body of PB 0
        ...
        ...
EPB             ; End of PB 0
```

Flags The **ACCU** is restored to the state it had before the PB was called

See Also PB, CPB, User's Guide

CPB CALL PROGRAM BLOCK

Description Conditionally or unconditionally calls a Program Block.
If the condition is not satisfied, the PB is not called.

Condition	Program Block is called:
blank	Always (no condition code)
H	If Accumulator = H (1)
L	If Accumulator = L (0)
P	If Positive flag = H (Negative flag = L)
N	If Negative flag = H
Z	If Zero flag = H
E	If Error flag = H

Usage

```
CPB     [cc] number     ; PB number 0-299
                        ; cc = condition code: H|L|P|N|Z|E
```

Example CPB 10 ; unconditionally calls PB 10

Flags The ACCU is set High (1) at the start of the PB.
In the program from where the PB is called, the ACCU is restored to the state it has before the PB was called.

See Also PB, EPB, CFB, User's Guide

Practice IF .. THEN .. ELSE structure

```
COB         0
            0
...
STH       I 15    ; IF Input 15 is High
CPB       H 20    ;    THEN call PB 20
CPB       L 25    ;    ELSE call PB 25
...
ECOB

PB          20
.....
EPB

PB          25
.....
EPB
```


CPBI CALL PROGRAM BLOCK INDIRECT

Description Conditionally or unconditionally calls a Program Block whose number is contained in the given Register.
Since this instruction uses a condition code, the 'R' data type is not required.
If the given Register contains an invalid PB number (> 299), or the PB does not exist, the Error flag is set and XOB 13 is called (if present).
If the condition is not satisfied, the PB is not called

Condition	Program Block is called:
blank	Always (no condition code)
H	If Accumulator = H (1)
L	If Accumulator = L (0)
P	If Positive flag = H (Negative flag = L)
N	If Negative flag = H
Z	If Zero flag = H
E	If Error flag = H

Usage

```
CPBI    [cc] reg    ; reg = Register number containing the number
                    ;       of the PB to be called
                    ; cc  = condition code: H|L|P|N|Z|E
```

Example

```
CPBI    L 10    ; If the ACCU is Low (0), then the PB whose number
                ;  is contained in R 10 is called
```

Flags The Error flag is set if the given Register contains an invalid PB number or if the PB does not exist.
The ACCU is set High (1) at the start of the PB.
In the program from where the PB is called, the ACCU is restored to the state it has before the PB was called

See Also PB, EPB, CFB, User's Guide

FB FUNCTION BLOCK

Description Begins a Function Block (FB). An FB is a subroutine with optional parameters. A list of FB parameters can be defined, this list is supplied when the FB is called.

Usage

FB	**number**	**; FB number 0-999**

Example

```
FB      0       ; Start of FB 0
        ...
STH     = 1     ; FB Parameter reference
        ...
EFB             ; End of FB 0
```

Flags The **ACCU** is set high (1) at the start of the FB

See Also EFB, CFB, User's Guide

Practice Computation of the formula: Z = X * (X+Y)

```
FB        25    ; Function Block  X * (X+Y)
ADD     = 1     ; Z = X + Y
        = 2
        = 3
MUL     = 3     ; Z = Z  *  X
        = 1
        = 3
EFB

COB       7
          0
...
STH     I 1     ; If Input 1 goes H
DYN     F 1
CFB     H 25    ;     Then R107 = R100 * (R100+330)
        R 100   ; Parameter 1  (X)
        K 330   ; Parameter 2  (Y)
        R 107   ; Parameter 3  (Z)

STH     I 2     ; If Input 2 goes H
DYN     F 2
CFB     H 25    ;     Then R107 = R200 * (R200+R201)
        R 200   ; Parameter 1  (X)
        R 201   ; Parameter 2  (Y)
        R 107   ; Parameter 3  (Z)
...
ECOB
```


EFB END OF FUNCTION BLOCK

Description Ends the current Function Block (FB). Returns to the instruction following the Call Function Block (CFB) instruction.

Usage

EFB	**; No operand required**

Example

```
FB      0       ; Start of FB 0
        ...     ;    Body of FB 0
        ...
        ...
EFB             ; End of FB 0
```

Flags The **ACCU** is restored to the state it had before the FB was called.

See Also FB, CFB, User's Guide

CFB CALL FUNCTION BLOCK

Description Conditionally or unconditionally calls a Function Block.
If the condition is not satisfied, the FB is not called.
An optional parameter list can follow the CFB instruction.
The parameters are used by instructions within the Function Block.
Parameters are referenced by using '= n' as the operand, where 'n' is the parameter number to use (1-128).
The value of this parameter is substituted as the operand.

Type	Description	Value range
I	Input	0..8191
O	Output	0..8191
F	Flag	0..8191
C	Counter	0..1599
T	Timer	0..450
R	Register	0..4095
K	K constant	0..16383
X	teXt	0..7999
DB	Data Block	0..7999
S	Semaphore	0..99
W	Word	0..65535 Used for any untyped constant LDL, LDH
M	MOV data type	Q 0..31 D 0..9 N 0..7 B 0..3 W 0..1 L 0

Usage

```
CFB     [cc] number     ; FB number 0-999
                        ; cc = condition code: H|L|P|N|Z|E
        [param 1]       ; optional parameter list
        [param 2]
        [param n]
```

Example

```
CFB     H 10    ; Calls FB 10 if the ACCU is High
        I 32    ; Parameter 1
        R 10    ; Parameter 2
```

Flags The **ACCU** is set High (1) at the start of the FB.

See Also FB, CPB, User's Guide

NCOB NEXT CYCLIC ORGANISATION BLOCK

Description Conditionally or unconditionally forces the program to switch to the next COB. If the condition code is not satisfied, the NCOB instruction is ignored.

Wait loops can be programmed using NCOB without interfering with the execution of any other COBs.
For every wait loop, an NCOB instruction should be inserted. This allows "parallel" execution of COBs.

Good BLOCTEC or GRAFTEC programs should NOT include wait loops, and hence should not need to use NCOB. Programs should normally use the ACCU status to control program execution.
Sequential processes can be easily programmed in GRAFTEC

Condition	
blank	Always (no condition code)
H	If Accumulator = H (1)
L	If Accumulator = L (0)
P	If Positive flag = H (Negative flag = L)
N	If Negative flag = H
Z	If Zero flag = H
E	If Error flag = H

Usage

```
NCOB    [cc]            ; cc = condition code: H|L|P|N|Z|E
```

Example

```
STH     I 15    ; Waits until I 15 = L
NCOB    L
JR      L -2
```

Flags Unchanged.

See Also RCOB, SCOB, CCOB, User's Guide

The following BLOCTEC instructions are potentially highly dangerous:

RCOB, NCOB, SCOB, CCOB and the COB supervision time.

If these instructions are used in a program which uses GRAFCET then serious problems can arise.
If they are not used with the utmost care: these instructions can cause at best, the user program to become slow, or at worst, cause a complete desynchronisation of the GRAFTEC and a CRASH.
Avoid to use these instructions within a GRAFTEC structure.

SCOB STOP CYCLIC ORGANISATION BLOCK (old)

Description Stops the given COB conditionally or unconditionally.
Execution continues with the next COB.
The COB is not executed again until the correct CCOB instruction is executed by another COB.
A COB can stop itself executing, but must be restarted by another COB containing a CCOB instruction.
If the condition is not satisfied, the SCOB instruction is ignored.

Good structured program should not need this instruction.
It should only be used in your application with the utmost care.

Condition	
blank	Always (no condition code)
H	If Accumulator = H (1)
L	If Accumulator = L (0)
P	If Positive flag = H (Negative flag = L)
N	If Negative flag = H
Z	If Zero flag = H
E	If Error flag = H

Usage

```
SCOB    [cc]  cob       ; COB number 0-15
                        ; cc = condition code: H|L|P|N|Z|E
```

Example

```
SCOB    L   10    ; Stops COB 10 if ACCU is Low (0)
```

Flags Unchanged.

See Also CCOB, NCOB, RCOB, User's Guide

Table: Firmware versions

Type of PCD	SCOB "old": FW ≤ V...	SCOB "new". FW ≥ V...
PCD1.M1xx	-	V001
PCD2.M110/M120	V003	V004
PCD2.M150	-	V0A0
PCD4.Mxx0	V005	-
PCD4.Mxx5	V00B	V00C
PCD4.M445	V001	V00C
PCD6.M540	V004	-
PCD6.M1/M2	V00A	-
PCD6.M3	-	V001

SCOB STOP CYCLIC ORGANISATION BLOCK (new)

Description Stops the given COB conditionally or unconditionally. If a SCOB is executed of the active COB, the execution continues with the next COB. If SCOB is executed of another COB, this COB will be ignored and the cycle contiues to the next instruction line. The COB is not executed again until the correct CCOB instruction is executed by another COB. If the condition is not satisfied, the SCOB instruction is ignored.

Good structured program should not need this instruction.
It should only be used in your application with the utmost care.

Condition	
blank	Always (no condition code)
H	If Accumulator = H (1)
L	If Accumulator = L (0)
P	If Positive flag = H (Negative flag = L)
N	If Negative flag = H
Z	If Zero flag = H
E	If Error flag = H

Usage

```
SCOB    [cc]  cob       ; COB number 0-15
                        ; cc = condition code: H|L|P|N|Z|E
```

Example

```
SCOB    L   10    ; Stops COB 10 if ACCU is Low (0)
```

Flags Unchanged.

See Also CCOB, NCOB, RCOB, User's Guide

The following BLOCTEC instructions are potentially highly dangerous:

RCOB, NCOB, SCOB, CCOB and the COB supervision time.

If these instructions are used in a program which uses GRAFCET then serious problems can arise.
If they are not used with the utmost care: these instructions can cause at best, the user program to become slow, or at worst, cause a complete desynchronisation of the GRAFTEC and a CRASH.
Avoid to use these instructions within a GRAFTEC structure.

CCOB CONTINUE CYCLIC ORGANISATION BLOCK

Description Conditionally or unconditionally allows a COB that was stopped by the SCOB instruction to resume execution.
If the condition is not satisfied, the COB is not resumed.
CCOB does not cause the COB to be executed immediately, but allows it to be executed the next time it is scheduled.

Good structured program should not need this instruction.
It should only be used in your application with the utmost care.

Condition	
blank	Always (no condition code)
H	If Accumulator = H (1)
L	If Accumulator = L (0)
P	If Positive flag = H (Negative flag = L)
N	If Negative flag = H
Z	If Zero flag = H
E	If Error flag = H

Usage

```
CCOB    [cc] number    ; COB number 0-15
                       ; cc = condition code: H|L|P|N|Z|
```

Example

```
CCOB    L 10    ; COB 10 is resumed if ACCU is Low (0)

CCOB    0       ; COB 0 is resumed unconditionally
```

Flags Unchanged.

See Also NCOB, RCOB, SCOB, User's Guide

The following BLOCTEC instructions are potentially highly dangerous:

RCOB, NCOB, SCOB, CCOB and the COB supervision time.

If these instructions are used in a program which uses GRAFCET then serious problems can arise.
If they are not used with the utmost care: these instructions can cause at best, the user program to become slow, or at worst, cause a complete desynchronisation of the GRAFTEC and a CRASH.
Avoid to use these instructions within a GRAFTEC structure.

RCOB RESTART CYCLIC ORGANISATION BLOCK

Description Restarts any COB, conditionally or unconditionally, from the given program line.
This instruction can be used within any COB or XOB.
If the condition is not satisfied, the RCOB instruction is ignored.
The 1st operand is the COB number to be restarted.
The 2nd operand is the program line number to restart from. The line number is an offset from the start of the COB, it is NOT an absolute program line number.

Good structured program should not need this instruction.
It should only be used in your application with the utmost care.

Condition	
blank	Always (no condition code)
H	If Accumulator = H (1)
L	If Accumulator = L (0)
P	If Positive flag = H (Negative flag = L)
N	If Negative flag = H
Z	If Zero flag = H
E	If Error flag = H

Usage

```
RCOB    [cc] cob    ; COB number 0-15
        line        ; Line number from start of COB (0-65535)
                    ; cc = condition code: H|L|P|N|Z|E
```

Example

```
RCOB    0       ; Restarts COB 0
        10      ; Execution begins from line 10 of COB 0
```

Flags Unchanged.

See Also NCOB, SCOB, CCOB, User's Guide

The following BLOCTEC instructions are potentially highly dangerous:

RCOB, NCOB, SCOB, CCOB and the COB supervision time.

If these instructions are used in a program which uses GRAFCET then serious problems can arise.
If they are not used with the utmost care: these instructions can cause at best, the user program to become slow, or at worst, cause a complete desynchronisation of the GRAFTEC and a CRASH.
Avoid to use these instructions within a GRAFTEC structure.

Notes:

7. GRAFTEC Instructions

SAIA® GRAFTEC is a self-documenting programming method for step-by-step processes.

A **GRAFTEC** program consists of a sequence of alternating STeps (ST) and TRansitions (TR). This sequence of STs and TRs forms the body of a Sequential Block (SB), which is called from a Cyclic Organisation Block (COB).

STEPS contain actions to be performed, instructions such as SET, RES, STXT, etc.

TRANSITIONS contain conditional linkages using instructions such as STH, ANL CMP, etc. A TR must always be followed by an ST. The ST is executed only if the preceding TR is satisfied (ACCU is High).

For easy GRAFTEC programming and maintenance, the **SAIA® GRAFTEC EDITOR** (SGRAF) is recommended.
This editor automatically handles the program structure, which you create graphically on the screen.
With this editor you don't need to use the low-level SAIA® GRAFTEC instructions listed below.

For more information about GRAFTEC programming refer to "The structured Programming" chapter in the User's Guide.

The operand(s) of these instructions can not be supplied as Function Block parameter(s).

SB	Sequential Block
ESB	End of Sequential Block
CSB	Call Sequential Block
RSB	Restart Sequential Block
IST	Initial Step
ST	Step
EST	End of Step
TR	Transition
ETR	End of Transition

Notes

Notes

SB SEQUENTIAL BLOCK

Description Starts a Sequential Block (SB).
A Sequential Block contains one independent GRAFTEC program.
The SB contains only GRAFTEC Instructions as IST, ST, TR, EST, ETR and ESB.

Usage

SB	**number**	**; SB number 0-31**

Example

```
SB      10      ; Start of SB 10
        ...     ;     Body of SB 10, will contain STs and TRs.
ESB             ; End of SB 10
```

Flags Unchanged.

See Also ESB, CSB, RSB, IST, ST, TR, User's Guide

ESB END OF SEQUENTIAL BLOCK

Description Ends the current Sequential Block (SB).

Usage

ESB	**; No operand required**

Example

```
SB      10      ; Start of SB 10
        ...     ;    Body of SB 10, contains STs and TRs.
ESB             ; End of SB 10
```

Flags The **ACCU** is restored to the state it had before the SB was called

See Also SB, ST, TR, User's Guide

CSB CALL SEQUENTIAL BLOCK

Description Conditionally or unconditionally calls a Sequential Block.
If the condition is not satisfied, the SB is not called.

A sequential block cannot be called from another SB.

Condition	Sequential Block is called:
blank	Always (no condition code)
H	If Accumulator = H (1)
L	If Accumulator = L (0)
P	If Positive flag = H (Negative flag = L)
N	If Negative flag = H
Z	If Zero flag = H
E	If Error flag = H

Usage

```
CSB     [cc] number     ; SB number 0-31
                        ; cc = condition code: H|L|P|N|Z|E
```

Example

```
CSB     L 10    ; Calls SB 10 if the ACCU is Low (0)
```

Flags The **ACCU** is set High (1) at the Start of the SB.
In the program from where the SB is called, the ACCU is restored to the state it had before the SB was called

See Also SB, CPB, User's Guide

RSB RESTART SEQUENTIAL BLOCK

Description

Conditionally or unconditionally restarts a Sequential Block (SB).
The 1st operand is the number of the SB to be restarted.
The 2nd operand is the STep number from where the SB is to be restarted.

If the restart must take place in simultaneous branches (parallel programs), the "RSB" instruction will contain as many additional lines as steps to be restarted.

Condition	:
blank	Always (no condition code)
H	If Accumulator = H (1)
L	If Accumulator = L (0)
P	If Positive flag = H (Negative flag = L)
N	If Negative flag = H
Z	If Zero flag = H
E	If Error flag = H

Usage

```
RSB     [cc] number     ; SB number 0-31
                        ; cc = condition code: H|L|P|N|Z|E
        step            ; STep number 0-1999
        [step]          ; [STep number 0-1999]
        [...]           ; [...]
        [step]          ; [STep number 0-1999]
```

Example

```
RSB     12      ; Restarts SB 12 at Step 1.
        1
```

Flags

The **ACCU** is set High (1) before restarting the SB

See Also

SB, CSB, ST, User's Guide

Practice

The RSB instruction is potentially very dangerous. There is no verification of the parameters specified in this instruction which, if badly chosen will, at best, cause a complete desynchronisation of the program or, at worst, a CRASH.

IST INITIAL STEP

Description The Initial Step defines the first Step to be executed when a Sequential Block (SB) is called.
Every SB must have at least one Initial Step.
In all other respects the Initial Step is the same as any other Step (see ST).
IST is followed by a list of incoming (I) and outgoing (O) Transitions.

Usage

IST	**number**	**; Initial Step number 0-1999**
	list	**; Incoming and outgoing transitions list**
		; (variable length)

Example

```
IST       1     ; Initial Step 1.
        I 900   ;       Incomming from Transition 900.
        O 1     ;       Outgoing to Transition 1
...             ;   Body of ST 1
EST             ; End of ST 1
```

Flags The **ACCU** is set High (1) at the start of the Initial Step

See Also EST, SB, ST

ST STEP

Description

Defines the start of a Step (ST).
Following the ST instruction must be a list of incoming (I) and outgoing (O) Transitions.
A Step should typically contain only action instructions such as SET, RES, OUT, LD, MOV, FADD, etc.
It must NEVER contain any wait loops.
Steps can call Program Blocks (PBs) and Function Blocks (FBs), providing these do NOT contain any wait loops.
In SAIA GRAFTEC, once a Step has been executed, the program pointer goes to the next Transition.
STeps can only appear inside SBs.

Usage

```
ST      number      ; Step number 0-1999
        list        ; Incoming and outgoing Transitions list
                    ; (variable length)
```

Example

```
ST          10   ; Step 10
          I 9    ;       Incoming from Transition 9
          O 10   ;       Outgoing to Transition 10
...              ;   Body of Step
EST              ; End of Step
```

Flags

The **ACCU** is set High (1) at the start of the ST

See Also

EST, IST, TR, SB, User's Guide

EST END OF STEP

Description Ends the current Step or Initial Step (ST or IST)

Usage

EST	**; No operand required**

Example

```
ST      0     ; Start of ST 0
      I 25    ;       Incoming from Transition 25
      O 47    ;       Outgoing to Transition 47
...           ;   Body of ST 0
EST           ; End of ST 0
```

Flags Unchanged.

See Also ST, TR, SB, User's Guide

TR TRANSITION

Description

Defines the start of a Transition (TR).
Following the TR instruction must be a list of all the incoming (I) and outgoing (O) Steps.
Typically a Transition should contain logical instructions forming a linkage whose final result indicates whether the following Step is to be executed.
If the final result of the Transition is false (ACCU = L (0)), then the next Step is NOT executed, execution continues with the next parallel branch or COB.
On the next program turn, the whole Transition will be processed again.
The next Step is executed only if the final result of the Transition is true (ACCU = H (1)).

With OR branching, the order of handling of the parallel TRs is set by the order of the outgoing Transitions defined in the preceding Step.
TRs can only appear inside SBs.

Usage

```
TR      number      ; Transition number 0-1999
        list        ; Incoming and outgoing Steps list
                    ; (variable length)
```

Example

```
TR          10   ; Transition number 10.
          I 900  ;       Incomming from Step 900
          O 1    ;       Outgoing to Step 1
          O 2    ;       Outgoing to Step 2
...              ;   Body of TR 10
ETR              ; End of Transition 10
```

Flags

The **ACCU** is set High (1) at the start of the TR.

See Also

ETR, SB, ST, User's Guide

ETR END OF TRANSITION

Description Ends the current Transition (TR).

Usage

ETR	**; No operand required**

Example

```
TR      0     ; Start of TR 0
      I 12    ;       Incomming from Step 12
      O 14    ;       Outgoing to Step 14
      ...     ;   Body of TR 0
      ETR     ; End of TR 0
```

Flags Unchanged.

See Also TR, ST, SB, User's Guide

Notes

8. SERIAL Communications Instructions

These instructions work only in CPU modules which contain serial ports.

Before any communications is done, the SASI instruction must be executed for each serial channel (up to 4).
This will configure the channels operating mode and baud rate.
Each channel can be run in a different mode and at different speeds.
Each channel also has its own reception and transmission buffers.

Modes	Function	Related instruction(s)
C	Send/Receive ASCII characters	STXD, SRXD
	Send entire texts	STXT
MD/SD	Send/Receive media	STXM, SRXM
MM4	Send/Receive registers on a LAC network	STXM, SRXM
SBUS	Send/Receive media on the SAIA BUS	STXM(I), SRXM(I)
PROFIBUS	Send/Receive media on the PROFIBUS	STXM(I), SRXM(I) SCON(I),
OFF	Deassign a serial interface	-

The serial channels control lines (CTS, RTS, DSR, DTR and DCD) can be read or written (SICL, SOCL).
SCON allows to open or close a virtual PROFIBUS channel. Communication via the LAN 1 is also possible after the SCON instruction has been executed.

SASI Assign serial interface
SASII Assign serial interface indirect

SRXD Serial receive character
STXD Serial transmit character
STXT Serial transmit text

SRXM Serial receive media
SRXMI Serial receive media indirect
STXM Serial transmit media
STXMI Serial transmit media indirect

SICL Serial input control line
SOCL Serial output control line

SCON Opens communications channel to LAN 1 and PROFIBUS
SCONI Opens communications channel to LAN 1 and PROFIBUS indirect

Notes

Notes

MODE C

Character or Text mode:

- Single characters from a Register or a Text are output.
- Single characters can be received and transferred into a Register.
- Often used to communicate with a terminal or printer.

MC0

Mode C without automatic handshaking
The user must control by himself the control signals with the SICL and SOCL instructions.

MC1

Mode C using RTS and CTS handshaking
The RTS control signal is automatically positionned by the PCD in function of the remaining space in the reception buffer.
The CTS signal influences the transmission of the PCD.

RTS	Low	Receive buffer contains more than 450 characters
	High	Receive buffer contains less than 300 characters
CTS	Low	Transmission is stopped
	High	Transmission is resumed

MC2

Mode C using Xon/Xoff protocol
This mode is similar to the RTS/CTS handshaking and is used when no control signals are present (eg. current loop).
Two special characters Xon (CTRL/Q) and Xoff (CTRL/S) are sent to control the transmission of the partner.

Receiver send		when
	Xoff	Receive buffer contains more than 450 characters
	Xon	Receive buffer contains less than 300 characters
Transmitter receives		then
	Xoff	Transmission is stopped
	Xon	Transmission is resumed

MC3

Mode C with Echo
This mode is used when communicating with a terminal; all received characters are sent back to the terminal screen.

MC4

Mode C for RS485 interface
The MC4 mode is a low level mode which will set the RS485 driver/receiver in drive mode only during the transmission of information (character/text) and will default to receive at any other time.

MODE D

Uses telegrams in accordance with ISO 1745, IBM BSC and DIN 66019.

SAIA® PCD specific data can be exchanged between two PCDs or between a PCD and another intelligent system (IBM PC, etc) connected directly or via the SAIA LAN 1.

The data can be the state of Inputs, Outputs or Flags; or the contents of Registers, Timers or Counters.

<mode>	**Description**
MD0	Mode D master
SD0	Mode D slave

The two modes are equivalent in functionality; the only difference is that when a conflict occurs in the full-duplex communication, the Master station always has priority over the Slave to repeat his request.

When communicating with a PC, the PCD must be set as Slave (SD0).

For a description of the complete protocol, consult the "Functional specification for the SAIA P8 Protocol".

SAIA LAN 1

The D mode can be used with the SAIA LAN 1: in this case, the connection between two stations can be achieved with the SCON instruction.

The status of the connection is automatically written by the LAN 1 in a Register, the address of this Register is given in the SASI instruction.

For more informations about the SAIA LAN 1 network, consult the "SAIA® LAN 1" manual.

MODE MM4

The MM4 mode allows the connection of the PCD on the COMPEX LAC/LAC2 Network. The LAC/LAC2 is an industrial local area network which enables the easy connection of heterogeneous intelligent machines. The PCD is connected to the network via a communicator which provides the required transmission services.

The MM4 mode consist in the exchange of Registers of 32 bits (4 ASCII characters). 64 Registers can be transferred with one telegram.

The error detection is assured by a CRC-16.

This mode also supports the point to point connection between 2 PCDs.

For more information, consult the "Description of the LAC MM4 Protocol".

MODE SBUS

S-Bus is the name of an efficient communication protocol for the SAIA® PCD generation of controllers. It can be used for both point-to-point communications and within a local master/slave network.

For point-to-point communications, any of the PCD's serial interfaces can be used. At the physical level, an S-Bus network uses the RS 485 standard, via two-core twisted and shielded cable.

S-Bus can be used as a simple, economic means of networking up to 255 PCD systems, connected to up to 8 segments, each containing up to 32 stations.

S-Bus has the following major characteristics:

- Ease of handling (installation, commissioning and user programming)
- Cost effective, since the S-Bus protocol is already built into every PCD processor. This means that no additional dedicated communications processor is required.
- Fail-safe data transfer, using CRC-16 error detection.
- High data transfer rate, due to the efficient binary protocol with transmission speed up to 38.4 kbps.
- Support for remote data access and diagnostics via a modem on leased or dial-up lines.
- Drivers are available for supervisory control systems such as Wizcon, InTouch, FactoryLink, Fix D-Macs and Genesis.
- With application level 2 (commissioning service) the programming unit has access to all slave stations on the network. This means that any slave station connected to the network can be controlled by the programming unit from a central point (e.g. by the debugger).
- Multi-master possibility by using the S-Bus Gateway

<mode>	**Description**
SM2	S-Bus master, with data mode
SM1	S-Bus master, with parity bit control
SM0	S-Bus master, no parity, with break character
SS2	S-Bus slave, with data mode
SS1	S-Bus slave, with parity bit control
SS0	S-Bus slave, no parity, with break character
GS2	S-Bus Gateway Slave, with data mode
GS1	S-Bus Gateway Slave, with parity bit control
GS0	S-Bus Gateway Slave, no parity, with break character
GM	S-Bus Gateway Master
OFF	De-initialize the serial line

For more information, consult the "Manual SAIA S-BUS" (ref 26/739).

PROFIBUS

PROFIBUS is the most successfull open Fiedlbus which can be used in wide range of applications.

PROFIBUS ensures that devices of different vendors can communicate together without the need to adapt interfaces.
PROFIBUS is standardized as an European standard (EN 50170).
PROFIBUS is vendor independant and devices are offered by a wide range of qualified vendors.
PROFIBUS consists of an assortment of compatible products. There are three main variations of PROFIBUS corresponding to the intended application:

- PROFIBUS-DP Decentral Periphery
- PROFIBUS-FMS Fieldbus Message Specification
- PROFIBUS-PA Process Automation

SAIA® PCD works with PROFIBUS-FMS

PROFIBUS-FMS is the universal solution for the communication between intelligent field devices and controllers and for information exchange between controllers.

The usage of PROFIBUS-FMS network with the SAIA® PCDs requires a dedicated processor: PCD4.M445 or a PCD7.F700 module.

PROFIBUS-DP is optimized for high speed and has been tailored for communication between automation systems and local peripherals. The usage of a PROFIBUS-DP network (master or slave) requires a module PCD7.F750 or. ..F770.

The definition and configuration (bus parameters, communication relationship list and object dictionnary) of a PROFIBUS network can be very extensive, depending on the size of the project. To make this task easier SAIA has developped a PROFIBUS configurator program under Windows. The configurator creates a file with the definition texts for all PROFIBUS channels in a station. This text file is used in the SASI instruction of the PROFIBUS channel.

For more information, consult the following manuals:

"Manual SAIA® PROFIBUS-FMS" (ref 26/742 E).
"Manual SAIA® PROFIBUS-DP" (ref 26/765 E).

SASI ASSIGN SERIAL INTERFACE

Description Initialises a serial channel.

The 1st operand is the serial channel number.
The 2nd operand is the number of a Text which contains the channel operating mode definition (see following pages).
This initialisation must be repeated for each serial channel to be used. Generally, the SASI instruction(s) will be placed in the XOB 16. All four channels can work in different modes and at different speeds. For the PROFIBUS channel assignation, consult the manual "SAIA® PROFIBUS" (ref 26/742).

Usage

SASI	**channel**	**; Serial channel number 0-3, [PROFIBUS: 10-99]**
	text nb	**; Definition text number 0-3999**

Example

```
SASI    0       ; Initialises serial channel 0
        100     ; using definitions in text 100
```

Flags The **Error** (E) flag is set if the definition text is missing or invalid

See Also SASI texts

Note For PROFIBUS, the channel number (10..99) is the virtual channel (also called CREF). The necessary information for initializing the channel are contained in a text which is automatically generated by the PROFIBUS configurator.

Practice Initialise the first serial channel (number 0) for TEXT mode with a speed of 4800 Bauds, 7 bits of data, Even parity and one stop bit
The SASI instruction is placed in XOB 16.

```
        XOB     16      ; Cold Start Exception Organisation Block

        SASI    0       ; Assign Serial channel 0
                10      ;      with parameters in text 10

        EXOB

TEXT 10 "UART:4800,7,E,1;"
        "MODE:MC0;"
        "DIAG:F1000,R4000;"
```

Flags 1000 .. 1007 are used as diagnostics flags and Register 4000 is used as diagnostic register.

The TEXT can also be written in one line:

```
TEXT 10 "UART:4800,7,E,1;MODE:MC0;DIAG:F1000,R4000;"
```


SASI Texts

A special definition text is needed for the Assign Serial Interface SASI instruction.

FORMAT:

```
TEXT xxxx     "<uart_def>;"
              "<mode_def>;"
              "<diag_def>;"
              ["<rx_buf>;"]
              ["<tx_buf>;"]
```

where xxxx is any valid text number (0..3999)

This text can also be written in one line.

The different parameters are:

<uart_def>	Defines the baud rate, data length, parity, ...
<mode_def>	Defines the Operating Mode (C, D, ...).
<diag_def>	Diagnostic Flags address and Diagnostic Register address.

The two last parameters are optional and only used in mode C:

<rx_buf>	Receive buffer length (default = 1).
<tx_buf>	Transmit buffer length.

MODE OFF

This mode differs from the standard SASI texts and is used when an interface which has already been assigned must be re-assigned.
To avoid contention over the interfaces a semaphore mechanism is implemented. When a serial interface is assigned a semahore is set so that if another assignation is tried on the same serial interface then the error led will be set and the instruction aborted. A serial interface can only be reassigned after it has been DESASI'd and this is done by executing the instruction SASI with the following text:

TEXT xxxx "MODE:OFF"

Likewise, if a CPU tries to deassign a serial interface which is already deassigned then the Error Flag Exception routine is also called.

<uart_def>

Defines the baud rate, data length, parity, number of stop bits, timeout
Format:
"UART:<baud-rate>,<char_len>,<parity>,<stop_bit>[,<timeout>];

<baud_rate>	<char_len>	<parity>	<stop_bit>	<time_out>	or by default
110	7	E (even)	1	10..15000 ms	15 s
150	8	O (odd)	2		9 s
300		L (low)			5 s
600		H (high)			3 s
1.200		N (none)			2 s
2.400					1 s
4.800					0,5 s
9.600					0,25 s
19.200					0,2 s
38.400					0,1 s

<time_out>:
The time_out is irrelevant for Mode C.

The default timeouts are given in seconds and are function of the baud rate
In the other modes, the timeout is the time after which a message is repeated if the partner receiving this messages does not give an acknowledgement.
If after 2 retries, the partner does not respond, he is declared "not responding".

Mode S-BUS:

The SBUS mode always use 11 bits (10 bits for data mode) to transmit one character: there is no definition for <char_len>, <parity>, <stop_bit>.

"UART:<Baudrate>[,<Timeout>][,<TS-Delay>][,<TN-Delay>][,<Break-Length>];"

<baud_rate>: 110 .. 38.400
<time_out>: 10 .. 15000 ms
<TS delay>: 10 .. 15000 ms
<Break-Length>: 4 .. 15 char

TimeOut, TS-Delay, TN-Delay and Break-Length are optional and normally only needed to be defined for special applications.

For more informations, consult the "Manual SAIA® S-BUS" (ref. 26/739).

Example of <uart_def> text:

```
"UART:9600,7,E,1;"
```

9600 bauds, 7 data bits, Even parity, 1 stop bits and default timeout

Note: All characters must be typed in Upper case. When the text is between the $SASI and $ENDSASI directives, the assembler tests the syntax of the text and all characters are converted in uppercases.

<mode_def>

Defines the operating mode of the serial channel.

Format: **"MODE: >[,<mode_opt>];"**
<mode_opt> is an optional series of parameters depending on the selected mode.

<mode>	**<mode_opt>**	**Description**
MC0	-	Mode C without automatic handshaking
MC1	-	Mode C using RTS and CTS handshaking
MC2	-	Mode C with Xon/Xoff protocol
MC3	-	Mode C with echo
MC4	-	Mode C for RS485 interface
MD0	-	Mode D Master
	R xxxx (1)	via LAN1; Register = SCON status
SD0	-	Mode D Slave
	R xxxx (1)	via LAN1; Register = SCON status
SM2	R xxxx (2)	Mode SBUS Master (client) Data mode
SM1	R xxxx (2)	Mode SBUS Master (client) Parity mode remote station
SM0	R xxxx (2)	Break mode
SS2	-	Mode SBUS Slave (server) Data mode
SS1	-	Mode SBUS Slave (server) Parity mode
SS0	-	Break mode
GM	-	Mode SBUS Gateway Master
GS2	-	Mode SBUS Gateway Slave Data mode
GS1	-	Mode SBUS Gateway Slave Parity mode
GS0	-	Break mode
MM4	(3)	Mode MM4
OFF	-	De-assignation of the serial line.

(1) D Mode with LAN 1
When using the SAIA LAN 1, the PCA2.T9x interface uses a Register to inform the PCD about the status of the connection. For more information, see the SCON instruction.

(2) SBUS Client
The address of the remote partner station is given in a Register.

(3) MM4 <mode_opt> consists of the following:
<BCS_opt>,<trpartner>,<trinfo>,<repartner>,<reinfo>,<rechar>

<mode_opt>	**Value**	**Description**
<BCS_opt>	0 or 1	Block Check Sum (0: no BCS, 1: CRC-16
<trpartner>	R xxxx	Transmission partner station number
<trinfo>	R xxxx	Remote ACK information
<repartner>	R xxxx	Reception partner station number
<reinfo>	R xxxx	Receive information
<rechar>	R xxxx	Number of received characters

<diag_def>

Defines the communications diagnostics media.

Format:
"DIAG:<dia_elem>,<dia_reg>;"

	Type	Description
<dia_elem>	O xxxx F xxxx	Base address of 8 consecutive Flags (or Outputs)
<dia_reg>	R xxxx	Address of a register for diagnostic

where xxxx is a valid address

The 8 Flags give informations about the status of the serial line. In case of error when executing an serial communication instruction, more informations can be obtained by examining the contents of the diagnostic register.

DIAGNOSTIC FLAGS

The Output or Flag address following the DIAG definition of the SASI texts is the base address of 8 consecutive Outputs or Flags, used as follows:

Address	Name	Description
xxxx	**RBSY**	Receiver busy
xxxx+1	**RFUL**	Receive buffer full
xxxx+2	**RDIA**	Receiver diagnostic
xxxx+3	**TBSY**	Transmitter busy
xxxx+4	**TFUL**	Transmit buffer full
xxxx+5	**TDIA**	Transmitter diagnostic
xxxx+6	**XBSY**	Text Busy
xxxx+7	**NEXE**	Not executed

RBSY

Receiver Busy

Mode	
C	RBSY is High when at least one character is available in the reception buffer. When all characters waiting in the reception buffer have been read with the SRXD instruction RBSY is cleared.
D **MM4**	RBSY is High when the receiver is busy.
SBUS	RBSY is High when a slave station receives a temegram. The flag is reset as soon as the reply telegram has been sent.
PROFIBUS	Not used

RFUL

Receive Buffer Full

Mode	
C	RFUL is set High when the number of incoming characters in the PCD Receive buffer is equal to or greater than the value of rx_buf (Receive buffer length). RFUL is Low when the number of characters remaining in the receive buffer is less than the vale of rx_buf. The internal reception buffer of the PCD always has room for 512 characters.
D **MM4**	RFUL is High when a correct data frame has been received.
SBUS	RFUL is High when elements in the slave station have been changed by the master station.
PROFIBUS	High indicates that a write telegram has been received.

RDIA

Receiver Diagnostic

Mode	
C **D** **SBUS** **MM4** **PROFIBUS**	RDIA is set High if the PCD detects an error during reception of a character; more information about the error can be obtained by examining the contents of the Communication diagnostic register. RDIA will be reset when all receiver diagnostic bits (0...15) in the diagnostic register are reset.

TBSY

Transmitter Busy

Mode	
C	TBSY is set High when the PCD transmits characters over the serial line. TBSY is set Low when all characters from the Transmission buffer have been transmitted
D **SBUS** **MM4** **PROFIBUS**	TBSY is set High when the PCD is transferring data. TBSY is set Low when the telegram has been acknowledged or when the number of retries is reach.

TFUL

Transmit Buffer Full

Mode	
C	TFUL is set High when the number of characters remaining in the PCD transmission buffer is greater than or equal to the value declared for tx_buf (Transmit buffer length). The TFUL is reset when the number of characters remaining in the Transmit buffer is less than the value of TBUF.
D	Not used
SBUS	
PROFIBUS	
MM4	TFUL is High when the acknowledgment has been received.

TDIA

Transmitter Diagnostic

Mode	
C **D** **SBUS** **MM4** **PROFIBUS**	TDIA is set High when the PCD detects an error during transmission of a character; more information about the error can be obtained by examining the contents of the Communication diagnostic register. TDIA will be reset when all transmitter diagnostic bits (16...31) in the diagnostic register are reset.

XBSY

Text busy

Mode	
C	XBSY is set High when the PCD transmits a text (STXT); when all the text has been transmitted XBSY is reset. Note: XBSY is reset at the beginning of the sending of the last character.
D	XBSY is High when a connection via the LAN1 is open.
SBUS	XBSY is low when the user has the permission to perform a SASI OFF.
MM4	XBSY is High when there is activity on the LAC network (STXM instruction)
PROFIBUS	Cross busy / channel open

NEXE

Not executed

Mode	
C	
D **SBUS** **MM4**	If the PCD is unable to perform the requested operation, NEXE is set High; further information about the error can be obtained by examining the contents of the Communication diagnostic register.
PROFIBUS	Set High when, after 3 attemps, it has not been possible to execute an instruction (STXM or RSXM). The flag is reset at the next instruction.

DIAGNOSTIC REGISTER

The address of a Diagnostic Register must also be supplied with the DIAG definition in the SASI text.
Normally all 32 bits of this Diagnostic Register are Low (0).
The register is to be reset to 0 by the user program.
If a bit is High (1) then its significance is the following (see relevant Mode):

Bit	Description	Cause	MODE				
			C	D	SBUS	MM4	PROFI BUS
0	Overrun error	Should never occur (notify SAIA)	●	●	●	●	
1	Parity error	Received a character with a parity error	●	●		●	
2	Framing error	Usually caused by an incorrect baud rate	●	●	●	●	
3	Break	Break in data line	●	●	●	●	
4	BCC error	Bad Block Check Code (or CRC-16)		●	●	●	
5	S-Bus PGU status	S-Bus PGU with Public Line modems			●		
6	End of transmit	Transmission ended SASI OFF		●	●		
7	Overflow error	Receive buffer overflow	●	●		●	
8	Length error	The telegram length is invalid			●		
9	Format error	Invalid telegram format		●		●	●
10	Address error	Adress of ACK is invalid			●	●	
11	Status error	PCD in false status			●		
12	Range error	Invalid element address		●	●		●
13	Value error	Error in the received value			●		
14	Missing media err	Address of media not defined or invalid			●		
15	Program error	Read from an empty receive buffer	●				
		LAN 1 not assigned or invalid station nb		●	●		
16	Retry count	Indicates the number of retries (in binary)			●		
17							
18	Transmission off	Sending is suspended (CTS = L or XOFF)	●				
19							
20	NAK response	NAK was received		●	●	●	●
21	No response	No response was received after timeout		●	●	●	
22	Multiple NAK	NAK received after retries		●	●	●	
23	TX buffer full	No more space in transmit buffer	●				
	TS Delay	No CTS after the TS Delay			●		
24	Enquiry error	No response to ENQ after retries		●			
25	Format error	Invalid definition text	●				
		Invalid command		●			●
26	Partner error	A problem has occured with the partner				●	
27	Network error	A problem has occured on the network				●	
28	Range error	Invalid element address		●	●	●	●
29							
30	Receive error	Error occured		●			●
31	Program error	Attempt to transmit when unauthorised		●	●	●	●

The <rx_buf> and <tx_buf> are only used for mode C

<rx_buf>

Defines the communication reception buffer limit.

Format:
"RBUF:<rbuf_len>;"

	Value	Description
<rbuf_len>	1.. 511	Receive buffer length

The Receive Buffer has always space for 512 x 8-bit characters.
For Mode C, the RBUF definition (1-511) indicates when to set the Receive Buffer Full status (RFUL).
For the other modes, RBUF is not used.

<tx_buf>

Defines the communication transmission buffer limit.

Format:
"TBUF:<tbuf_len>;"

	Value	Description
<tbuf_len>	1.. 511	Transmission buffer length

Similar to the Receive Buffer.
For Mode C, the TBUF definition (1-511) indicates when to set the Transmit Buffer Full status (TFUL).
For the other modes, RBUF is not used.

Examples of SASI Texts

Mode MC0 at 9600 Bds, 7 data bits, Even parity,1 stop bit, using F1000-F1007 as diagnostic flags and register R4000 as diagnostic register:

```
TEXT 10     "UART:9600,7,E,1;MODE:MC0;DIAG:F1000,R4000;"
```

Mode MC2 at 4800 Bds, 8 data bits, no parity,1 stop bit, using F0-F7 as diagnostic flags and register R100 as diagnostic register; a receive buffer length of 25 characters:

```
TEXT 20     "UART:4800,8,N,1;MODE;MC2;DIAG:F0,R100;"
            "RBUF:25;"
```

Mode SD0 (Slave) at 9600 Bds, 7 data bits, Even parity, 1 stop bit, F1000-F1007 as diagnostic flags and register R4000 as diagnostic register:

```
TEXT 30     "UART:9600,7,E,1;MODE:SD0;DIAG:F1000,R4000;"
```

Mode MD0 (Master) at 9600 Bds, 7 data bits, Even parity, 1 stop bit, F1000-F1007 as diagnostic flags and register R4000 as diagnostic register; the SAIA LAN1 is used and a timeout of 3 sec is needed:

```
TEXT 40     "UART:9600,7,E,1,3000;"
            "MODE:MD0,R1;"
            "DIAG:F1000,R4000;"
```

Register R 1 is used to store the connection state of the LAN1.

Mode MM4 at 9600 Bds, 8 data bits, no parity,1 stop bit, Timeout: 300ms, no BCS, Registers 100..104 are used for the remote partner number, ...
F1000-F1007 are used as diagnostic flags and register R1000 as diagnostic register:

```
TEXT 50     "UART:9600,8,N,1,300;"
            "MODE:MM4,0,R100,R101,R102,R103,R104;"
            "DIAG:F1000,R1000;"
```

Mode SBUS Parity mode (Master) at 9600 Bds, Register 555 is used to hold the partner number, F8000-F8007 as diagnostic flags and register R4005 as diagnostic register:

```
TEXT 60     "UART:9600;MODE:SM1,R555;DIAG:F8000,R4005;"
```

Mode SBUS Paritiy mode (Slave) at 9600 Bds, F8000-F8007 are used as diagnostic flags and register R4005 as diagnostic register:

```
TEXT 60     "UART:9600;MODE:SS1;DIAG:F8000,R4005;"
```

Mode SBUS Data mode(Slave) at 9600 Bds, Register 55 is used to hold the partner number, F8000-F8007 are used as diagnostic flags and register R4005 as diagnostic register:

```
TEXT 60     "UART:9600;MODE:SS2,R55;DIAG:F8000,R4005;"
```

```
Mode        SASI    10          ; channel 10
PROFIBUS            T_As_10     ; SASI text
```

The SASI text "T_As_10" is created by the PROFIBUS configurator

Using SYMBOLS in texts

Symbols can also be used in SASI texts.

The value and optionally the type of the symbol is inserted into the text. The symbol is written outside the ASCII text segment in double quotes, and must be separated from this or other symbols by a comma. After the symbol, an optional field width and prefix type can be given.

Format:

symbol [. [[-] [0] width] [t | T]]	
symbol	The symbol name. This can actually be any expression which includes a symbol, for example: MotorOn + 100, ... Symbols with floating point values are not permitted.
.	The dot immediately after the symbol indicates that a field width and/or a prefix is present.
Width	The field width: the number of digits or spaces required for the number. If the width begins with a 0, leading zeros are inserted.
t | T	Optional prefix type 't' or 'T'. If 't', the value is prefixed with the symbol's type in lower case (o, f, r, ...); if 'T', the symbol's type is in upper case (O, F, R,...)

Examples:

```
BAUD      EQU       9600
D_FLAGS   EQU     F 500
D_REG     EQU     R 4095

          XOB       16
          SASI      1
                    3999

TEXT 3999"UART:",BAUD,",7,E,1;MODE:MC0;"
                  "DIAG:",D_FLAGS.T,",",D_REG.T,";"

          EXOB
```

The resulting text will be:

```
          "UART:9600,7,E,1;MODE:MC0;"
          "DIAG:F500,R4095;"
```

New SASI with '$' (SASI text accepts $) see next page
can be used from the following firmware versions only:

PCD1: V070 PCD4.Mxx5: V0E0 PCD6.M3: V030
PCD2: V080 PCD4.M445: V0E0

$SASI, $ENDSASI

These assembler directives can be used to delimit texts which are used by the SASI instruction. All texts enclosed within these directives will be checked by the assembler and any errors detected.

Format:

```
$SASI
<SASI text definition>
...
$ENDSASI
```

If $SASI .. $ENDSASI are not used, it is possible to enter an invalid text which may cause incorrect initilialisation of the serial port.

Example:

```
        XOB     16
        ...
        SASI    0       ; Initialize serial channel 0
                100     ; using text 100
        ...
        EXOB

$SASI               ; Text 100 is checked as SASI text by the Assembler
TEXT 100 "UART:9600,7,E,1;MODE:MC0;DIAG:F1000,R4000;"
$ENDSASI
```

New SASI with '$' (SASI text accepts $)

e.g.: `"UART:$Ra,$Rb,$Rc,$Rd;MODE:$Re,$Rf;DIAG:F$Rg,R$Rh;"`

Ra	Baudrate	110 .. 38400 (numeric)
Rb	Bits	7, 8 (numeric)
Rc	Parity	E, O, N (ASCII coded)
Rd	Stop	1 or 2 (numeric)
Re	Mode	'MC0', 'SM2' etc. (ASCII coded)
Rf	Station	Register with S-Bus station (numeric)
Rg	Diagnostic flags	Register with the base diagnostic flag number (0 .. 8191 numeric)
Rh	Diagnostic register	Register with the diagnostic register number (0 .. 4095 numeric)

Firmware to be used see page before.

SASII ASSIGN SERIAL INTERFACE INDIRECT

Description Initialises a serial channel or a PROFIBUS channel in indirect mode. This instruction works in the same way as the SASI instruction. The difference is that it works in indirect mode. Indirect mode means that the number of the channel and the definition text number can be given by the content of registers.

Usage

SASII	**channel**	**; Serial channel number or Register**
	text definition	**; Register**

Channel
Channel number to be initialised
This parameter can be given directly or indirectly:

0..3	Serial channel number
10..99	PROFIBUS channel number
R 0..4095	Register containing the channel number (0..3, 10..99)

Text_definition
This parameter is a register number (R 0..4095)
This register contents the address of a text containing where the interface parameters are defined. Valid addresses for text:

0..3999	in standard memory
4000..7999	in extension memory

Example

```
SASII    0     ; Initialises serial channel 0
         R 1   ; using definition text address contained in R 1
```

Flags The **Error** (E) flag is set if the definition text is missing or invalid

See Also SASI texts

SRXD SERIAL RECEIVE CHARACTER (Mode C)

Description Loads the next ASCII character present in the Receive Buffer of the channel given by the 1st operand into the Register given by the 2nd operand.

The instruction SRXD should be executed only if there is a character ready, indicated by RBSY = H otherwise the Error flag is set. After SRXD is executed, the least significant 8 bits of the Register contain the character, all other Register bits are set to 0.

Up to 512 characters can be in the Receive Buffer. Each time SRXD is executed, the next character is read.

If the Receive Buffer overruns (more than 512 characters), then there will be a receive error (the RDIA flag and the corresponding status bit in the channels Diagnostic Register are set).

Usage

```
SRXD[X]   channel        ; Serial channel number 0-3
          reg     (i)    ; Register to receive character R 0-4095
```

Example

```
SRXD      3     ; Reads a character from channel 3
        R 100   ; and stores it in Register 100
```

Flags The **Error** (E) is set if the SRXD instruction is executed with an empty reception buffer or if the channel has not been correctly initialised or does not exist.

See Also STXD, SRXM, Communications instructions, Diagnostic flags

Practice Typical application in Bloctec structured program:

```
....
STH      F RBSY          ; If there is a character waiting
CFB      H READ_CHAR     ;   Then read this character
....

FB         READ_CHAR     ; FB Read a character
[STH     H RDIA]         ; If there is a Receive Error
[CFB     H RCV_ERROR]    ;    Then handle the error
SRXD       0             ; Read the character on channel 0
         R 999           ;   and store it in R 999
....
EFB
```

Note: In simple applications, the error processing (above between brackets) can be omitted.

STXD SERIAL TRANSMIT CHARACTER (Mode C)

Description The character held in the least significant bits of the Register given in the 2nd operand is placed in the Transmit Buffer of the serial channel given by the 1st operand. It is then transmitted automatically.

The Transmit Buffer can hold up to 512 characters. If it is empty (all characters have been transmitted), the TBSY status flag is set Low. While there are characters waiting to be transmitted, TBSY remains High.

If the TDIA status is High after executing an STXD, this indicates a problem, and the Diagnostic Register shoud be examined.

Usage

STXD[X]	**channel**		**; Serial channel number 0-3**
	reg	**(i)**	**; Register containing character R 0-4095**

Example

```
STXD     1     ; Transmits the character held in
       R 100   ; Register 100 (bits 7-0) on channel 1
```

Flags The **Error** (E) flag is set if the channel has not been correctly initialised or does not exist.

See Also SRXD, STXT, Communications instructions, Diagnostic flags

Practice Typical application in Bloctec structured program:

```
....
STL      F TFUL           ; If there is room in the TX buffer
CFB      H SEND_CHAR      ;  Then send a character
....

FB         SEND_CHAR      ; FB Send a character
STXD       0              ; Send on channel 0
           R 900          ;   the character stored  in R 900
[STH     H TDIA]          ; If there is a Transmit Error
[CFB     H SND_ERROR]     ;   Then handle the error
....
EFB
```

Note: In simple applications, the error processing (above between brackets) can be omitted.

STXT SERIAL TRANSMIT TEXT (Mode C)

Description Transmits the Text indicated in the 2nd operand via the serial channel given by the 1st operand (0-3). Status XBSY is set High, and the CPU transmits the Text. XBSY is set Low when the Text has been transmitted. The text output can take several seconds for large texts. The normal execution of the program continues unaffected as the output of the text is executed in the background.

The XBSY flag indicates the completion of the background task. Whilst XBSY is High no other communications instruction should be performed on this serial channel. Texts can contain control strings to allow the formatted transmission of element values, see Text Control Strings. The NEXE diagnostic flag is set if the Text contains a bad control string.

Usage

```
STXT[X]   channel        ; Serial channel number 0-3
          text     (i)   ; Text number 0-3999
```

Example

```
STXT    0       ; Transmits Text 123 on serial channel 0
        123
```

Flags The **Error** (E) flag is set if the Text or channel does not exist, or the channel has not been correctly initialised.

See Also Texts, STXD, SRXD, Communications instructions, Diagnostic flags.

Practice When input 1 is High, the following text should be sent: "Hello world !"

```
        XOB     16
        SASI    1       ; Initialise serial channel 1
                0       ;     with parameters stored in Text 0
        EXOB
$SASI
TEXT  0  " UART:9600,7,E,1;MODE:MC0;DIAG:F1000,R1000;"
     ; 9600 Bds, 7 Data bits, Even Parity, 1 Stop bit
     ; Mode MC0, Diagnostic flags: F1000 .. F1007, Diagnostic register: R1000
$ENDSASI
        COB       0
                  0
        STH     I 1     ; If input 1 goes High
        DYN     F 0
        ANL     F 1006  ;  and not already transmitting (F1006 = XBSY)
        JR      L END
        STXT      1     ;     Then send Text 10 over serial channel 1
                  10
END:    ECOB
TEXT 10  "Hello world ! <10> <13>"
```


Texts

Texts can be defined anywhere in an Assembler source program, but are placed in an allocated area of Text memory in the PCD.
Texts can be written immediately following their referencing instructions; alternatively, all texts can be written in a separate Assembler source module.

The following rules should be observed:

- A text is defined with:
  TEXT n "The actual text"
  where 'n' is the text number (0...3999)

- The text can consist of several lines, each line must be enclosed in double inverted comas: " ..... ". Texts can be of any length.
- Control characters can be entered enclosed in angle brackets.
  For example: <LF>, <CR>, <FF>, <ESC>, …
- Control characters with decimal ASCII codes (1.. 31), or special characters with codes (127..255) can be entered as decimal values enclosed in angle brackets: <nnn>.
  For example: CR = <13>, LF = <10>, ESC = <27>, BELL = <7>, ...
- Standard ASCII characters (32..126) may be entered directly from the keyboard.

In the PCD memory, all texts are terminated with a NUL character (ASCII code 0), which is automatically appended to the end of the text by the Assembler. Therefore, a text cannot contain the character NUL.

Examples:
The two following texts give the same result:

```
TEXT 10    "The quick brown fox jumps over the lazy dog"
TEXT 11    "The quick brown fox"
           "jumps over the lazy dog"
```

If after the text a line-feed and a carriage return must be sent:

```
TEXT 12    "The quick brown fox jumps over the lazy dog<CR><LF>"
```

If you have an EPSON printer you can put a part of your text in bold, by sending special controls characters to the printer.
Assume that you will print the following text:

```
  There is no limit for the SAIA PCD4 programmable controller
TEXT 13    "There is no limit for the"
           "<ESC>E SAIA PCD4 <ESC>F"
           "programmable controller <LF><CR>
```

Note: The characters ESC E make your printer print in bold, ESC F returns it to normal.

Texts and variables

Texts can also contain variables such as the clock value, the status of an input, the content of a register, ...
There are two characters which have a special meaning for the PCD: $ and @

$ = DIRECT ADDRESSING

Absolute element number is provided.

$H	Time (Hour,Minute,Second): hh:mm:ss	
$HH	Time (Hour only): hh	
$HM	Time (Minute only): mm	
$HS	Time (Second only): ss	
$D	Date (Year, Month, Day): yy-mm-dd	
$d	Date (Day, Month, Year): dd.mm.yy	
$DD	Date (Day only): dd	
$DM	Date (Month only): mm	
$DY	Date (Year only): ss	
$W	Week (Week number, Day of week): ww-dd	
$WN	Week (Week number only): ww	
$WD	Week (Day number only): dd	
$innnn	Logical state of a single Input (0, 1)	nnnn : element number (must be 4 digits)
$onnnn	Logical state of a single Output (0, 1)	
$fnnnn	Logical state of a single Flag (0, 1)	
$Innnn	Logical state of 8 Inputs (nnnn to nnnn+7)	nnnn: first element number (must be 4 digits)
$Onnnn	Logical state of 8 Outputs (nnnn to nnnn+7)	
$Fnnnn	Logical state of 8 Flags (nnnn to nnnn+7)	
$Cnnnn	Counter contents	nnnn: element number (must be 4 digits)
$Rnnnn	Register contents	
$Tnnnn	Timer contents	
$Lnnnn	incLude another text (max 3 level)	nnnn:text number (must be 4 digits)
$xnn	Character 'x' is repeated 'nn' times The character must not be a data type: (H|h|D|d|W|i|o|f|I|O|F|C|R|T|L|x)	nn must be 2 digits
$Annnn	Output Register contents as ASCII char.	nnnn: Register number (must be 4 digits)

Example of $Annnn:

"$A0999" when R 999 = 00000000 hex 'NUL'
"$A0999" when R 999 = 00000061 hex 'a'
"$A0999" when R 999 = 00006162 hex 'ab'
"$A0999" when R 999 = 00616263 hex 'abc'
"$A0999" when R 999 = 61626364 hex 'abcd'

Preceding zeros are not output. An ASCII zero is only output if the lowest value byte is equal to 0.

@ = INDIRECT ADDRESSING

Element number is supplied in a Register

@innnn	Logical state of a single Input (0, 1)	nnnn : Register number (must be 4 digits)
@onnnn	Logical state of a single Output (0, 1)	
@fnnnn	Logical state of a single Flag (0, 1)	
@Innnn	Logical state of 8 Inputs (add to add+7)	
@Onnnn	Logical state of 8 Outputs (add to add+7)	
@Fnnnn	Logical state of 8 Flags (add to add+7)	
@Cnnnn	Counter contents	
@Rnnnn	Register contents	
@Lnnnn	incLude another text (max 3 level)	
@xnnnn	Character 'x' is repeated Register contents times. The character must not be a data type: (H|D|W|i|o|f|I|O|F|C|R|L|x)	

NOTE: To output a '$' use "$$", to output an '@' use "@@".

Example 1:

```
TEXT  10   "Date: $D Time: $H <CR><LF>"
"$-32 <CR><LF>"
"Input 0..7: $I0000 <CR><LF>"
"Register 100:  $R0100 <CR><LF>"
"$+20 <CR><LF>"
```

Assuming that this text is printed on the 16 August 90 at 09:00 am, that inputs 0 and 1 are High, and the contents of Register 100 is 12345: the following will be printed:

```
Date: 90-08-16   Time: 09:00:00
-----------------
Input 0..7: 11000000
Register  100: 12345
++++++++++++++++++++
```

Example 2: Practical use of "$A...."

Cursor position on a screen should be determined from 2 registers for the X and Y position:
X position from Register R 1 (1..80)
Y position from Register R 2 (1..25)

The escape sequence for cursor positionning is:
<27><17><value for X><value for Y>

It is possible to program: "...<27><17>$A0001$A0002..."

For example, to output a fixed position of X = 40 and Y = 12, the whole sequence of 4 characters can be written into a single register and output using $A....
Note that all values must be in hex format:

ESC = 1B hex; 17 = 11 hex; 40 = 28 hex (X value); 12 = 0C hex (Y value)

Load register R 1000:
```
LD        R 1000
          1B11280CH
```

Text output for cursor positionning: ". . . $A1000 . . ."

OUTPUT FORMATS

The format of transmitted Register and Counter data can also be specified in the Text. The field width and number of decimal places can be specified. Format definitions are introduced by the text "$%xxxx", where 'xxxx' is the required format, see below. If such a definition is output, all the following Register or Counter values are output using this format, until another format definition is encountered.

In the following format definitions, the 'd/D' means 'decimal', x/X = hexadecimal and b/B = binary. Other number base formats like o = Octal or f = floating point are not supported. If the value is too large to fit in the defined field, default formatting is used (no formatting).

Output format definitions:

Assume Registers 10, 11 and 12 contain respectively the following constant values: 123456, -7890 and 5

NO FORMATTING (DEFAULT):
The field width depends on the size of the number.

TEXT 0 "REGISTER 10: $R0010 <10><13>"
"REGISTER 11: $R0011 <10><13>"
"REGISTER 12: $R0012"

Output:

```
REGISTER 10: 123456
REGISTER 11: -7890
REGISTER 12: 5
```

FIXED WIDTH FIELD:
Use the format definition "$%xxd" or "$%xxD", where 'xx' (1 - 99) signifies the field width.

"$%xxd": The value is right-justified with **leading spaces**.

TEXT 1 "$%08dREGISTER 10: $R0010 <10><13>"
"REGISTER 11: $R0011 <10><13>"
"REGISTER 12: $R0012"

Output:

```
REGISTER 10:   123456
REGISTER 11:    -7890
REGISTER 12:        5
```

"$%xxD": The value is right-justified with **leading zeroes**.

TEXT 1 "$%08DREGISTER 10: $R0010 <10><13>"
"REGISTER 11: $R0011 <10><13>"
"REGISTER 12: $R0012"

Output:

```
REGISTER 10: 00123456
REGISTER 11: -0007890
REGISTER 12: 00000005
```


FIXED WIDTH FIELD and FIXED NUMBER OF DECIMAL PLACES:
The value is right-justified, but the number of decimal places is always displayed, and is padded on the right with zeros.
Use the format definition "$%xx.yd", where 'xx' is the total field width, and 'y' is the number of places to the right of the decimal point.

```
TEXT 2      "$%07.3dREGISTER 10: $R0010 <10><13>"
            "REGISTER 11: $R0011"<10><13>"
            "REGISTER 12: $R0012"
```

Output:

```
REGISTER 10: 123.456
REGISTER 11:  -7.890
REGISTER 12:   0.005
```

FIXED DECIMAL PLACES ONLY:
The number of decimal places is fixed but the field width is dependent on the size of the number.
Use the format definition "$%00.yd", where 'y' is the number of decimal places, padded on the right with zeros if required.

```
TEXT 2      "$%00.5dREGISTER 10: $R0010 <10><13>"
            "REGISTER 11: $R0011"<10><13>"
            "REGISTER 12: $R0012"
```

Output:

```
REGISTER 10: 1.23456
REGISTER 11:-0.07890
REGISTER 12: 0.00005
```

REMOVING FORMATTING:
"$%00d" sets the standard format (no formatting).

Saving / Restoring format definitions:
Format definitions may be saved using "$sn", where 'n' is a 'save' number.
Up to 10 format definitions can be saved (0-9).
Saved formats are restored using "$n", where 'n' is the 'save' number of the format definition to be restored.
Formats may be saved as part of the intialisation process, in XOB 16, the start-up XOB. To save a format, the text containing this format must be output to the serial line with the STXT instruction. If a format is restored which has not been saved, the default format (no formatting) is used.

Example:

```
          XOB       16
          ....
TEXT 991  "$%05.1d$s1"       ; Format 1 definition (nnn.n)
TEXT 992  "$%04.2d$s2"       ; Format 2 definition (n.nn)
TEXT 993  "$%08.3d$s3"       ; Format 3 definition (nnnn.nnn)

          SEI     K 0        ; Activation of the format definitions
DEF:      STH       XBSY
          JR      H DEF
          STXTX     0
                    991
          INI     K 2
          JR      H DEF
          ....
          EXOB

          COB       0
                    0
          ....
          STXT      1
                    10
TEXT 10   "Pump Litres     Price/L    Total <10><13>"
          " 1   $1$R0010   $2$R0011   $3$R0012 <10><13>"
          " 2   $1$R0013   $2$R0014   $3$R0015 <10><13>"
          ....
          ECOB
```

This has the same effect as formatting the text as:

```
          "1 $%05.1d$R0010 $%04.2d$R0011 $%8.3d$R0012"
          "2 $%05.1d$R0013 $%04.2d$R0014 $%8.3d$R0015"
```

Results:

```
          Pump   Litres   Price/l    Total
            1     13.8     0.86      11.868
            2    158.2     0.95     150.290
```


Including other texts:
The "$Lnnnn" sequence 'incLudes' another text which is processed as though it were part of the original text. If this included text contains a new format definition, the format is used until the end of the text. On return to the original text, the original format definition is restored.

Example:

```
         COB       0
                   0
         ....
         STXT      1    ; Send Text 10
                   10
TEXT 10  "$L0100 Motor speed too high<10><13>"
         ....
         STXT      1    ; Send Text 11
                   11
TEXT 11  "$L0100 Oil pressure too low<10><13>"
         ...
         ECOB
TEXT 100 "Diesel engine ALARM:"
```

Result:

```
         Diesel engine ALARM: Motor speed too high
         Diesel engine ALARM: Oil pressure too low
```


Using SYMBOLS in texts

With the PCD Utilities (V1.3 and after), SYMBOLS can be used inside texts. The value and optionally the type of the symbol is inserted into the text. The symbol is written outside the ASCII text segment in double quotes, and must be separated from this or other symbols by a comma. After the symbol, an optional field width and prefix type can be given.

Format:

symbol [. [[-] [0] width] [t | T]]	
symbol	The symbol name. This can actually be any expression which includes a symbol, for example: MotorOn + 100, ... Symbols with floating point values are not permitted.
.	The dot immediately after the symbol indicates that a field width and/or a prefix is present.
Width	The field width: the number of digits or spaces required for the number. If the width begins with a 0, leading zeros are inserted.
t | T	Optional prefix type 't' or 'T'. If 't', the value is prefixed with the symbol's type in lower case (o, f, r, ...); if 'T', the symbol's type is in upper case (O, F, R,...)

Examples:

```
Flag      EQU     F 123
Output    EQU     O 32
Reg       EQU     R 999
TEXT 0    "$",Flag.04T        ; Text 0 is "$F0123"
TEXT 1    "",Flag             ; Text  1 is "123" (the empty "" is needed)
TEXT 2    "DIAG:",Output.T,",",Reg.T
                              ; Text 2 is "DIAG:O32,R999"
TEXT 3    "55:",Flag.T,"-",Flag+7,":",Output.T,"-",Output+7
                              ; Text 3 is "55:F123-130:O32-39"
TEXT 4    "FLAG Number: *",Flag.-8,"*"
                              ; Text 4 is "FLAG Number: *123     *"
```

Symbols can also be used in SASI texts

```
D_FLAGS   EQU     F 500
D_REG     EQU     R 4095

          XOB       16
          SASI      1
                    3999
TEXT 3999"UART:9600,7,E,1;MODE:MC0;"
                  "DIAG:",D_FLAG.T,",",D_REG.T,";"
```

This creates the text "DIAG:F500,R4095;"

Note: The symbols (for texts) and texts must be defined in the same file.

SRXM SERIAL RECEIVE MEDIA (Mode D)

Description Reads elements from the Remote PCD, and copies them into destination elements in the Local PCD. Transfers can be I|O|F to O|F, R|T|C to R|T|C.
The 1st operand is the channel number.
The 2nd operand is the number of elements to be transferred.
The 3rd operand is the lowest address of the source element in the Remote PCD.
The 4th operand is the lowest address of the destination element in the Local PCD.
After executing SRXM, the TBSY flag is set High, it is set Low when the operation has completed.

Usage

```
SRXM[X]   channel          ; Serial channel number 0-3
          count            ; Number of elements to receive 1-16
          source    (i)    ; Source element I|O|F, R|T|C
          dest      (i)    ; Destination element O|F, R|T|C
```

Example

```
SRXM      0     ; Reads the contents of R 100-115
          16    ; into R 0-15 via serial channel 0
        R 100
        R 0
```

Flags The **Error** (E) flag is set if the channel is not correctly initialised, or SRXM is executed when already in communication.

See Also STXM, Communications instructions, Diagnostic flags

Practice The inputs 0 .. 15 of the Remote PCD must be copied into the Outputs 32..47 of the Local PCD. (The two PCDs are connected via the serial line).

Local PCD:

```
     XOB       16
     SASI      1
               0

TEXT 0   "UART:9600,7,E,1;MODE:MD0;"
         "DIAG:F1000,R1000;"

     EXOB

     COB       0
               0
     STH     F 1003   ; If not already busy (TBSY)
     JR      H next
     SRXM      1      ; Then receive on channel 1
               16     ; 16 elements
             I 0      ; from I 0 (to 15) of remote PCD
             O 32     ; to O 32 (to 47) of Local PCD
next:          ECOB
```

Remote PCD:
(only the serial line must be assigned)

```
     XOB       16
     SASI      1
               0

TEXT 0
"UART:9600,7,E,1;MODE:SD0;"
"DIAG:F1000,R1000;"

     EXOB
```


SRXM SERIAL RECEIVE MEDIA (Mode MM4)

Description

Copy the receive buffer (received frame) into consecutive registers of the PCD.
When a telegram has been received without errors: RFUL is set to 1; SRXM reset this flag to 0.
The 1st operand is the channel number.
The 2nd operand is always 0.
The 3rd operand is a Register or a Counter which will contain (after the execution of the instruction) the number of received characters.
The 4th operand is the address of the first Register where the received characters will be copied.
Each received character needs 8 bits of a Register: a Register can hold a maximum of 4 characters.
The characters are placed in the Registers as follow:

reg 1:	11111111	22222222	33333333	44444444	Characters 1 to 4
reg 2:	55555555	66666666	77777777	88888888	Characters 5 to 8

...
If the number of received characters is not a multiple of 4, the rest of the Register is set to 0.

The address of the partner which has sent the telegram is contained in the <repartner> Register defined in the SASI text.

Usage

```
SRXM    channel     ; Serial channel number 0-3
        0           ; Not used
        reg 1       ; Number of read characters R 0-4095
        reg 2       ; Destination register address R 0-4095
```

where:

- reg 1 Register contains (after execution) the number of read characters. A counter can also be used.
- reg 2 Address of the first register into where the information will be copied (a register holds up to 4 characters)

Example

```
SRXM    1       ;Transfers the telegram received on channel 1
        0       ; in the Register 20  and following
      C 100     ; C100: number of received characters
      R 20
```

Flags

The **Error** (E) flag is set if the channel is not correctly initialised, or if SRXM is executed when no telegram has been received.

See Also

For further information, consult the "Description of the LAC MM4 Protocol"

SRXM SERIAL RECEIVE MEDIA (Mode SBUS)

For more informations, consult the "S-Bus Manual"

Description Reads elements from a server PCD station, and copies them into destination elements in the client PCD. The address of the server station is passed via a register as defined in the SASI text (see page 8-11).
Transfers can be I|O|F to O|F, R|T|C to R|T|C.
The 1st operand is the channel number.
The 2nd operand is the number of elements to be transferred.
The 3rd operand is the lowest address of the source element in the server PCD.
The 4th operand is the lowest address of the destination element in the client PCD.
After executing SRXM, the TBSY flag is set High, it is set Low when the operation has complete.

SRXM can only be used in the client PCD

Usage

```
SRXM[X]   channel          ; Serial channel number 0-3
          count            ; Number of elements to receive
          source    (i)    ; Base address of source element
          dest      (i)    ; Base address of destination element
```

Where:

count	1..32	number of R/T/C to read	
	1..128	number of I/O/F to read	
	0	Special function code	
	R nnnn	Used for Data Block transfer	
source	I/O/F	0..8191	
	R	0..4095	Base address of elements
	T/C	0..1599	in the slave PCD
	DB	0..7999	
	K	0..6000	Special function code
destination	I/O/F	0..8191	
	R	0..4095	Base address of elements
	T/C	0..1599	in the master PCD
	DB	0..7999	

Example

```
LD      R 100   ; Register as defined in the SASI text for
          10    ;  the source station number
SRXM      0     ; Reads  via channel 0
          20    ;  20 elements
        R 100   :  from R100-119 of station 10
        R 0     ;  into R0-R19
```

Flags The **Error** (E) flag is set if the channel is not correctly initialised, or SRXM is executed when already in communication.

See Also STXM, Communications instructions, Diagnostic flags

The following table shows which elements can be copied from the source station to the appropriate elements in the destination station.

Master PCD (destination**)**

Slave PCD (source)	**O**	**F**	**R**	**C**	**T**	**DB**
I	•	•				
O	•	•				
F	•	•				
R			•	•	•	•
C			•	•	•	•
T			•	•	•	•
K			•			
DB			•	•	•	

Special functions

Code	Function description		Examples of result
K 0 ..7	**Read CPU status**: 0..6: CPU number of slave PCD 7: own CPU status		R, C, H, S, D
K 1000	**Read Clock**		same format as RTIME instruction
K 2000	**Read Display Register**		
K 3000	**Read Size of Data Block**		
K 5000	**Read Device type**	in ASCII	" D1" , " D2", " D4", …
K 5010		in decimal	1, 2, 4, …
K 5100	**Read Module type**	in ASCII	" M1_", " M11", " M12", " M14",..
K 5110		in decimal	10, 11, 12, 14, 24, …
K 5200	**Read Firmware version**	in ASCII	" $4C", " 004", " X41", …
K 5210		in dec	5, … or -1 dec for any '$', 'X', 'β'
K 5300	**Read CPU number**	in ASCII	" 0", " 1"
K 5310		in decimal	0 .. 6
K 6000	**Read S-Bus station number** in BROADCAST This will only work in point-to-point communication.		

Transfer of Data Blocks

The format of the SRXM instruction, when working with Data-Blocks, differs slightly from the conventional format. To address an element of a Data-Block, it is always necessary to specify the number of the Data-Block and then the position of the element in the Data-Block.

SRXM **Channel**
Count + Position
Source
Destination

Where Count + Position : Register number.

This register contains the "Count" or number of elements to transfer (range 1...32) and the "Position" in the Data-Block where to put or get the data. "Count" is given in the MS Word of the register and "Position" in the LS Word of the register.

SRXM SERIAL RECEIVE MEDIA (PROFIBUS)

For more informations, consult the "PROFIBUS Manual"

Description Reads data (objects) from the remote station and copies them in the local PCD.
The 1st operand is the channel number.
The 2nd operand is the sub-index from source and destination object.
The 3rd operand is the source object index (remote station).
The 4th operand is the destination object index (own station).

Usage

```
SRXM    channel     ; Channel number 10-99
        count       ; Sub-index 0-255
        source      ; Source object index K 0-16383
        dest        ; Destination object index K 100-499
```

Where:

channel	10..99	for PCD4.M445
	10..19	for PCD2 with PCD7.F700
count	0..255	Sub-index (0 = no sub-index)
source		Source object index (remote station)
	K 0..16383	
destination		Destination object index (own station)
	K 100..499	PCD4.M445
	K 100..199	PCD2 with PCD7.F700

Example

```
SRXM    13      ; Reads  via channel 13
        0       ;  no sub-index
      K 22      :  copies object 22 from the remote station
      K 150     ;  to the object 150 of the own station
```

Flags The **Error** (E) flag is set if the channel is not correctly initialised, or SRXM is executed when already in communication.

See Also STXM

SRXMI SERIAL RECEIVE MEDIA INDIRECT (Mode SBUS)

For more informations, consult the "S-Bus Manual" (Ref. 26/739)

Description This instruction works in the same way as the existing SRXM instruction. The difference is that it works in indirect mode. Indirect mode means that the number of the media for source and destination is given by the content of a register. SRXMI are only available for transfer of media. Transfer options like the Real Time clock, Display-Register,... are not allowed.

Usage

SRXMI	**channel**	**; Channel**
	count	**; Count or Count + Position**
	source	**; Source-type and Register number**
	dest	**; Destination-type and Register number**

Channel

This parameter is used to specify the channel number (range: 0...3).

Count or Count + Position

This parameter is a register number. This register contents the "Count" for standard medias or "Count" and "Position" for Data-Block.
For Data-Block, "Count" is given in the MS Word of the register and "Position" in the LS Word of the register and in that case, the initialisation of this register can be easily done with LDL and LDH instructions.

Source-type and Reg-number

Destination-type and Reg-number

These parameters specify the "Source" and "Destination " of the transfer. Each of these parameters is composed of a character giving the type of media (I/O/F/R/T/C/DB) and a register number (0...4095). The source and the destination must respect the source-destination validity described in the table for the SRXM/STXM instructions.

SRXMI does not work in indexed and parametrised mode.

Example

```
SRXMI     3     ; channel #3
        R 100   ; Number of elements in R 100
        O 101   ; Source: outputs with base address in R 101
        F 102   ; Destination: flags with base address in R 102
```

Flags The **Error** (E) flag is set if the channel is not correctly initialised, or SRXMI is executed when already in communication

See Also SRXM, Communications instructions, Diagnostic

SRXMI SERIAL RECEIVE MEDIA INDIRECT (PROFIBUS)

For more informations, consult the "PROFIBUS Manual"

Description Reads data (objects) in indirect mode from the remote station and copies them in the local PCD. It is possible to select direct or indirect addressing of channel operands.

SRXMI does not work in indexed and parametrised mode.

Usage

```
SRXM    channel     ; Channel number 10-99, R 0-4095
        count       ; Sub-index R 0-4095
        source      ; Source object index K [R 0-4095]
        dest        ; Destination object index K [R 0-4095]
```

Where:

channel	10..99		for PCD4.M445
	10..19		for PCD2 with PCD7.F700
	R 0..4095		for indirect addressing
count	R 0..4095		Register containing the Sub-index (0 = no sub-index)
source	Register containing the source object index (remote station)		
	K [R 0..4095]	0..16383	
destination	Register containing the destination object index		
	K [R 0..4095]	100..499	PCD4.M445
		100..199	PCD2 with PCD7.F700

Example

```
SRXMI   R 50    ; channel number in R 50
        R 51    ;  sub-index in R 51
        K 52    :  source object index in R 52
        K 53    ;  destination object index in R 53
```

Flags The **Error** (E) flag is set if the channel is not correctly initialised, or SRXMI is executed when already in communication.

See Also SRXM

STXM SERIAL TRANSMIT MEDIA (Mode D)

Description Transmits elements from the Local PCD to elements in the Remote PCD.
Transfers can be I|O|F to O|F, R|T|C to R|T|C.
The 1st operand is the channel number.
The 2nd operand is the number of elements to be transferred.
The 3rd operand is the lowest address of the source element in the Local PCD.
The 4th operand is the lowest address of the destination element in the Remote PCD.
During the execution of STXM, the TBSY flag is set High; when the operation is complete, it is set Low.

Usage

```
STXM[X]   channel          ; Serial channel number 0-3
          count            ; Number of elements to transmit  1-16
          source    (i)    ; Source element I|O|F, R|T|C
          dest      (i)    ; Destination element O|F, R|T|C
```

Example

```
STXM       0     ; Transmits the contents of R 100-115
           16    ; into R 0-15 via serial channel 0
         R 100
         R 0
```

Flags **The Error** (E) flag is set if the channel is not correctly initialised, or STXM is executed when already in communication.

See Also SRXM, Communications instructions, Diagnostic flags

Practice The inputs 0..15 of the Local PCD must be copied into the Outputs 32..47 of the Remote PCD.
Remote PCD: Only the serial channel must be assigned (see SRXM)
Local PCD:

```
          XOB       16
          SASI      1
                    15
TEXT 15   "UART:9600,7,E,1;MODE:MD0;DIAG:F1000,R1000;"
          EXOB

          COB       0
                    0
          STH     F 1003  ; If not already busy to communicate (TBSY)
          JR      H NEXT
          STXM      1     ;  Then transfer on serial channel 1
                    16    ;     16 Elements
                  I 0     ;     from Input 0 ( to 15) of local PCD
                  O 32    ;     to Output 32 ( to 47) of remote PCD
NEXT:     ECOB
```


STXM SERIAL TRANSMIT MEDIA (Mode MM4)

Description

Transfers registers over the LAC/LAC2 network using the MM4 protocol.
This transfer can occur via the LAC/LAC2 network or in point to point.
The 1st operand is the channel number.
The 2nd operand defined the transfer function.
The 3rd operand is a Register or a Counter which contains the number of characters to transfer.
The 4th operand is the address of the first Register containing the characters to transmit.
A Register can hold a maximum of 4 characters: each character needs 8 bits.
The characters must be placed in the Registers as follow:

reg 1:	11111111	22222222	33333333	44444444	Characters 1 to 4
reg 2:	55555555	66666666	77777777	88888888	Characters 5 to 8

...

The address of the partner is contained in the <trpartner> Register defined in the SASI text.

After the execution of STXM, the XBSY flag is set High; when the operation is completed (telegram acknowledged by the partner), XBSY is reset to Low.

Usage

```
STXM    channel     ; Serial channel number 0-3
        fct         ; Function to perform 0-4
        reg 1       ; Number of characters to transmit  R 0-4095
        reg 2       ; Source register address R 0-4095
```

where:

- fct — Function to perform
  - 0 /2 Transmission of data
  - 4 Diffusion
- reg 1 — Register containing the number of characters to be transmitted. A counter can also be used.
- reg 2 — Address of the first register from where the information is to be transferred (a register holds up to 4 characters)

Example

```
STXM    1       ;Transmit on channel 1
        0       ; indicates a Transmission
      C 100     ; number of characters to transmit
      R 20      ; 1st Register containing the data
```

Flags

The **Error** (E) flag is set if the channel is not correctly initialised, or if STXM is executed when already in communication.

See Also

For further information, consult the "Description of the LAC MM4 Protocol"

STXM SERIAL TRANSMIT MEDIA (Mode SBUS)

For more informations, consult the "S-Bus Manual"

Description Transmits elements from the client PCD to elements in a server PCD.
The address of the server station is passed via a register as defined in the SASI text (see page 8-11). Transfers can be I|O|F to O|F, R|T|C to R|T|C.
The 1st operand is the channel number.
The 2nd operand is the number of elements to be transferred.
The 3rd operand is the lowest address of the source element in the client PCD.
The 4th operand is the lowest address of the destination element in the server PCD.
During the execution of STXM, the TBSY flag is set High; when the operation is complete, it is set Low .

STXM can only be used in the client PCD

Usage

```
STXM[X]   channel          ; Serial channel number 0-3
          count            ; Number of elements to transmit
          source    (i)    ; Base address of Source element
          dest      (i)    ; Base address of Destination element
```

Where:

channel	0..3	Interface to be used	
count	1..32	number of R/T/C to transmit	
	1..128	number of I/O/F to transmit	
	0	Special function code	
source	I/O/F	0..8191	
	R	0..4095	Base address of elements
	T/C	0..1599	in the master PCD
	DB	0..7999	
	K	4000	Special function
destination	I/O/F	0..8191	
	R	0..4095	Base address of elements
	T/C	0..1599	in the slave PCD
	DB	0..7999	
	K	1000	Write clock in the slave PCD
	K	17, 18, 19	Special function

Example

```
LD      R 100   ; Register as defined in the SASI text for
          22    ; the destination station number

STXM      0     ; Transmits via channel 0
        100     ; 100 elements
      F 100     ; from F100-199
      O 32      ; to O32-131 of station 22
```

Flags The **Error** (E) flag is set if the channel is not correctly initialised, or STXM is executed when already in communication.

See Also SRXM, Communications instructions, Diagnostic flags.

The following table shows which elements can be copied from the source station to the appropriate elements in the destination station.

		Slave PCD (destination)						
		O	**F**	**R**	**C**	**T**	**DB**	**Clock**
Master PCD (source)	**I**	•	•					
	O	•	•					
	F	•	•					
	R			•	•	•	•	•
	C			•	•	•	•	
	T			•	•	•	•	
	DB			•	•	•		

When writing to the clock, two registers are sent. For the data format of registers, see the WTIME instruction.

Special function

It is possible to provoke the execution of an XOB in a slave station using the STXM instruction with the following arguments:

```
STXM        0..3          ; Serial channel number
            0             ; (must be 0)
          K 4000          ; Used to indicate XOB interrupt
          K 17|18|19      ; number of the XOB to execute
```

It is also possible to use this instruction in broadcast mode; this allows the synchronisation of events.

Transfer of Data Blocks (Write)

The format of the STXM instruction, when working with Data-Block, differs slightly from the conventional format. To address an element of a Data-Block, it is always necessary to specify the number of the Data-Block and then the position of the element in the Data-Block.

```
STXM      Channel
          Count Position
          Source
          Destination
```

Where Count + Position : Register number.

This register contains the "Count" or number of elements to transfer (range 1...32) and the "Position" in the Data-Block where to put or get the data. "Count" is given in the MS Word of the register and "Position" in the LS Word of the register.

STXM SERIAL TRANSMIT MEDIA (PROFIBUS)

For more informations, consult the "PROFIBUS Manual"

Description Transmits data (objects) from the remote station and copies them in the local PCD.
The 1st operand is the channel number.
The 2nd operand is the sub-index from source and destination object.
The 3rd operand is the source object index (own station).
The 4th operand is the destination object index (remote station).

Usage

```
STXM    channel     ; Channel number 10-99
        count       ; Sub-index 0-255
        source      ; Source object index K 100-499
        dest        ; Destination object index K 0-16383
```

Where:

channel	10..99	for PCD4.M445
	10..19	for PCD2 with PCD7.F700
count	0..255	Sub-index (0 = no sub-index)
source		Source object index (own station)
	K 100..499	PCD4.M445
	K 100..199	PCD2 with PCD7.F700
destination		Destination object index (remote station)
	K 0..16383	

Example

```
STXM    11      ; Transmits  via channel 11
        3       ;  sub-index (element) 3
      K 122     :  copies object 122 from the own station
      K 150     ;  to the object 150 of the remote station
```

Flags The **Error** (E) flag is set if the channel is not correctly initialised, or STXM is executed when already in communication.

See Also SRXM

Practice

STXMI SERIAL TRANSMIT MEDIA INDIRECT (Mode SBUS)

For more informations, consult the "S-Bus Manual"

Description This instruction works in the same way as the existing STXM instructions. The difference is that it works in indirect mode. Indirect mode means that the number of the media for source and destination is given by the content of a register. STXMI are only available for transfer of media. Transfer options like the Real Time clock, Display-Register,... are not allowed

Usage

STXMI	**channel**	**; Channel**
	count	**; Count or Count + Position**
	source	**; Source-type and Register number**
	dest	**; Destination-type and Register number**

Channel
This parameter is used to specify the channel number (range: 0...3).
Count or Count + Position
This parameter is a register number. This register contents the "Count" for standard medias or "Count" and "Position" for Data-Block.
For Data-Block, "Count" is given in the MS Word of the register and "Position" in the LS Word of the register and in that case, the initialisation of this register can be easily done with LDL and LDH instructions.
Source-type and Reg-number
Destination-type and Reg-number
These parameters specify the "Source" and "Destination" of the transfer Each of these parameter is composed of a character giving the type of media (I/O/F/R/T/C/DB) and a register number (0...4095). The source and the destination must respect the source-destination validity described in the table for the STXM instruction.

STXMI does not work in indexed and parametrised mode.

Example

```
STXMI      1
        R  100
        DB 101
        R  102
```

Flags The **Error** (E) flag is set if the channel is not correctly initialised, or STXM is executed when already in communication.

See Also STXM, Communications instructions, Diagnostic flags

STXMI SERIAL TRANSMIT MEDIA INDIRECT (PROFIBUS)

For more informations, consult the "PROFIBUS Manual"

Description Transmits data (objects) in indirect mode from the remote station and copies them in the local PCD. It is possible to select direct or indirect addressing of channel operands.

STXMI does not work in indexed and parametrised mode.

Usage

```
STXMI   channel   ; Channel number 10-99, R 0-4095
        count     ; Sub-index R 0-4095
        source    ; Source object index K [R 0-4095]
        dest      ; Destination object index K [R 0-4095]
```

Where:

channel	10..99	for PCD4.M445	
	10..19	for PCD2 with PCD7.F700	
	R 0..4095	for indirect addressing	
count	R 0..4095	Register containing the Sub-index (0 = no sub-index)	
source	Register containing the source object index (own station)		
	K [R 0..4095]	100..499	PCD4.M445
		100..199	PCD2 with PCD7.F700
destination	Register containing the destination object index (remote)		
	K [R 0..4095]	0..16383	

Example

```
STXMI   R 100   ; channel number in R 100
        R 110   ;  sub-index in R 110
        K 200   :  source object index in R 200
        K 300   ;  destination object index in R 300
```

Flags The **Error** (E) flag is set if the channel is not correctly initialised, or STXMI is executed when already in communication.

See Also STXM

SICL SERIAL INPUT CONTROL LINE

Description Reads a control signal from the serial channel given in the 1st operand and stores its state in the ACCU.

The 2nd operand is the signal to be read:

0 = CTS Clear To Send
1 = DSR Data Set Ready
2 = DCD Data Carrier Detect

For the Port 0 (PGU) of the PCD1, PCD2, PCD4, PCD6.M3 and PCD6.M540, the instruction SICL is always allowed (independently, whether the port is assigned or configured).
For any other port of PCD1, PCD2, PCD4, PCD6.M3 or PCD6.M540, the instruction SICL is only allowed on a port configured for S-Bus PGU.
Otherwise, the instruction SICL is only allowed after execution of a SASI.

Usage

SICL	**channel**	**; Serial channel number 0-3**
	signal	**; Signal number CTS|DSR|DCD (0|1|2)**

Example

```
SICL    0     ; If DSR of channel 0 is High
        1
CPB     H 25  ;  Then Call PB 25
```

Flags The **ACCU** is set to the state of the addressed input control line.
The **Error** (E) flag is set if the channel does not exist or has not been correctly initialised.

See Also SOCL, Communications instructions

SOCL SERIAL OUTPUT CONTROL LINE

Description Sets a selected control signal of the serial channel given in the 1st operand to the state of the ACCU (H or L).

The 2nd operand is the signal to be set:

0 = RTS	Request To Send (RS485: Accu high = transmit, Accu low = receive)
1 = DTR	Data Terminal Ready
2 =	Special functions

For the Port 0 (PGU) of the PCD1, PCD2, PCD4, PCD6.M3 and PCD6.M540, the instruction SOCL is always allowed (independently, whether the port is assigned or configured).
For any other port of PCD1, PCD2, PCD4, PCD6.M3 or PCD6.M540, the instruction SOCL is only allowed on a port configured for S-Bus PGU.
Otherwise, the instruction SOCL is only allowed after execution of a SASI.

Usage

```
SOCL    channel     ; Serial channel number 0-3
        signal      ; Signal number RTS|DTR (0|1|2)
```

Example

```
SOCL    0       ; Sets DTR signal of channel 0 according to
        1       ;  the ACCU
```

Flags The **Error** (E) flag is set if the channel does not exist or has not been correctly initialised.

See Also SICL, Communications instructions

Special functions:

Port 0 on PCD2
A SASI for SM1/SS1 in the user program will configure the port 0 to RS485. If the user wishes to use RS232 on the port 0 then he must perform the following instructions *after the SASI* instruction:

```
ACC     L
SOCL      0
          2
```


Switch from RS485 to RS422
The serial interface RS422/RS485 on the PCD4.C130, PCD4.C340, PCD7.F110/F150, PCD7.F520/530 switches automatically to RS485 when certain modes are assigned.

Mode	**Type**
MC0 .. MC3, MD0 / SD0	RS422
MC4, S-Bus	RS485

It is sometimes needed to force the PCD to use S-Bus with RS422; in this case, the following instructions must be performed *after the SASI* instruction:

```
ACC     L
SOCL      Port_nb
          2
```

It is also possible to force the RS485 mode with MC0..MC3 or MD0/SD0 with:

```
ACC     H
SOCL      Port_nb
          2
```

Switch from receive to transmit in RS485:

To set the RS485 in the transmit mode
perform the following instructions after the SASI instruction:

```
ACC     H
SOCL      Port_nb
          0
```

To set the RS485 in the receive mode
perform the following instructions after the SASI instruction:

```
ACC     L
SOCL      Port_nb
          0
```


SCON SERIAL CONNECT TO LAN 1

Description Opens or closes a virtual connection to other stations on the SAIA LAN 1.
The 1st operand is channel number.
The 2nd operand is the station number (1-250).
The 3rd operand is the connection state (0 = Close, 1 = Open).

The connection state is written into the Register defined in the SASI instruction by the MODE assignment (Example: "MODE:SD0,R4000;").

Connection state returned in the register:

Value	Description
0	Disconnected
1	Connected
2	Queued
3	Destination busy
4	Destination unknown
6	Remote PLC not connected
10..2500	If the connection was made by a remote PLC the register contains the number of the PLC multiplied by 10

Consult the SAIA LAN 1 manual for more information.

Usage

```
SCON    channel    ; Serial channel number 0-3
        station    ; LAN 1 Station number 1-250
        state      ; Connection state (0 = close, 1 = open)
```

Example

```
SCON    0       ; Open connection to Station 100 on
        100     ; channel 0
        1
```

Flags The **Error** (E) flag is set if the channel doesn't exist or has not been correctly initialised

See Also SASI, Communications instructions

SCON OPEN COMMUNICATION CHANNEL (PROFIBUS)

For more informations, consult the "PROFIBUS Manual"

Description Opens or closes a virtual PROFIBUS channel.
Before exchanging information, it is necessary to initialize (open) the virtual communication channel with the SCON instruction.
The 1st operand is channel number.
The 2nd operand is always 0 (not used).
The 3rd operand is the connection state (0 = Close, 1 = Open).

Usage

```
SCON    channel     ; PROFIBUS channel number 10-99
        0           ;
        state       ; Connection state (0 = close, 1 = open)
```

Example

```
SCON    10      ; Open channel 10
        0       ;
        1
```

Flags The **Error** (E) flag is set if the channel doesn't exist.

See Also

SCONI SERIAL CONNECT TO LAN 1 INDIRECT

Description Opens or closes a virtual connection to other stations on the SAIA LAN 1 in indirect mode.
The 1st operand is channel number or a register containing the channel number.
The 2nd operand is a register containing the station number (1-250).
The 3rd operand is a register containing the connection state (0 = Close, 1 = Open)

SCONI does not work in indexed and parametrised mode.

The connection state is written into the Register defined in the SASI instruction by the MODE assignment (Example: "MODE:SD0,R4000;").

Connection state returned in the register:

Value	Description
0	Disconnected
1	Connected
2	Queued
3	Destination busy
4	Destination unknown
6	Remote PLC not connected
10..2500	If the connection was made by a remote PLC the register contains the number of the PLC multiplied by 10

Consult the SAIA LAN 1 manual for more information.

Usage

```
SCONI    channel    ; Serial channel number 0-3 or R 0-4095
         station    ; LAN 1 Station number R 0-4095
         state      ; Connection state R 0-4095
```

Example

```
SCONI      0      ; Open channel 0
         R 100    ; LAN 1 station number in R 100
         R 101    ; Connection state in R 101
```

Flags The **Error** (E) flag is set if the channel doesn't exist or has not been correctly initialised

See Also SCON, SASI, Communications instructions

SCONI OPEN COMMUNICATION CHANNEL INDIRECT (PROFIBUS)

For more informations, consult the "PROFIBUS Manual"

Description Opens or closes a virtual PROFIBUS channel in indirect mode.
Before exchanging information, it is necessary to initialize (open) the virtual communication channel with the SCON instruction.
The 1st operand is channel number or a register containing the channel number.
The 2nd operand is a register and must always be 0.
The 3rd operand is a register containing the connection state (0 = Close, 1 = Open).

SCONI does not work in indexed and parametrised mode.

Usage

SCONI	**channel**	**; PROFIBUS channel nb 10-99 or R 0- 4095**
	station	**; R 0- 4095 (always 0)**
	state	**; Connection state R 0- 4095**

Example

```
SCONI   R 10    ; Open PROFIBUS channel  on R 10
        R 11    ; Station on R 11 (must be 0)
        R 12    ; Connection state on R 12
```

Flags The **Error** (E) flag is set if the channel doesn't exist.

See Also SCON

9. LAN 2 Instructions

The SAIA LAN2 is a Local Area Network working on the token passing principle which can interconnect up to 255 stations.

A LAN2 module (PCD6.T1.. or PCD4.M340) is required for each station.

The states of any Inputs, Outputs or Flags, and the values in any Registers, Timers or Counters, or the status of any CPU can be sent or received via the LAN2.

The LAN2 commands are defined in LAN2 command texts. It is possible to transfer from any bit element to any other bit element (eg. Inputs to Flags), or from Timer/Counter to Timer/Counter, or from Register to Register.

IMPORTANT:
The instructions and functionnality described here are valid for version 004 (and above) of the LAN2.

LAN 2 instructions:

LRXD	Receive data via LAN2
LTXD	Transmit data via LAN2
LRXS	Receive status via LAN2
LTXS	Transmit status via LAN2

Notes

Notes

LRXD RECEIVE DATA VIA LAN2

Description

Receives data over the LAN2 from a remote PCD.
The 1st operand is the priority (0 = high, 1 = low). To benefit of the high priority, the length of the data to transfer must be less than 32 bytes.
The 2nd is the number of a Text containing the address of the remote partner, the elements of the partner to be read and where they must be copied (own station).
The 3rd operand is the base address of 10 Diagnostic Flags (or Outputs).
The first Diagnostic element is the EXEC flag. It is initially set Low by LRXD, and remains Low on subsequent executions of the same LRXD instruction, until the transfer is complete.

When the transfer is complete the EXEC flag is set High. If the LRXD instruction is executed again, the data transfer is repeated. The state of the EXEC flag is altered only when the LRXD instruction is executed.

Usage

```
LRXD[X]   priority          ; 0 = high,   1 = low
          text      (i)     ; Number of text containing command 0-3999
          diag              ; Diagnostic flags base address F|O 0-8182
```

Example

```
LRXD        1     ; Low priority transfer
            900   ; Text 900 contains the command
          F 200   ; Diagnostic Flags F 200-209
```

Flags

Unchanged.

See Also

LTXD, LRXS, LTXS, LAN2 Diagnostic Flags, LAN2 Command Texts

Practice

After switching on I 1 on the local PCD, inputs 0-7 from the remote PCD station number 3 are read and transferred to the local PCD to the output 32-39

```
            COB       5
                      0
            STH     I 1     ; If Input 1 goes High
            DYN     F 1
            ORL     F 100   ;        EXEC Flag
            CPB     H 50    ;    Then Call PB 50
            ECOB

            PB        50
            LRXD      1     ; Priority
                      15    ; Text nb.
                    F 100   ; EXEC Flag
$LAN
TEXT 15     "3:I0-7:O32-39"
$ENDLAN
            EPB
```


LTXD TRANSMIT DATA VIA LAN2

Description

Transfers data over the LAN2 to a remote PCD.
The 1st operand is the priority (0 = high, 1 = low). To benefit of the high priority, the length of the data to transfer must be less than 32 bytes.
The 2nd is the number of a Text containing the address of the remote partner, the elements to be transfered and where they must be copied (remote station).
The 3rd operand is the base address of 10 Diagnostic Flags (or Outputs).
The first Diagnostic element is the EXEC flag. It is initially set Low by LTXD, and remains Low on subsequent executions of the same LTXD instruction, until the transfer is complete.

When the transfer is complete the EXEC flag is set High. If the LTXD instruction is executed again, the data transfer is repeated. The state of the EXEC flag is altered only when the LTXD instruction is executed.

Usage

```
LTXD[X]   priority        ; 0 = high,  1 = low
          text      (i)   ; Number of text containing command 0-3999
          diag            ; Diagnostic flags base address F|O 0-8182
```

Example

```
LTXD      1     ; Low priority transfer
          900   ; Text 900 contains the command
        F 200   ; Diagnostic Flags F 200-209
```

Flags

Unchanged.

See Also

LRXD, LRXS, LTXS, LAN2 Diagnostic Flags, LAN2 Command Texts.

Practice

After switching on I 8 on the local PCD, inputs 0-7 are read once and transferred to the remote PCD station number 3 to the output 40-47

```
          COB       5
                    0
          STH     I 8     ; If Input 8 goes High
          DYN     F 8
          ORL     F 100   ;        EXEC Flag
          CPB     H 51    ;    Then Call PB 51
          ECOB

          PB        51
          LTXD      1     ; Priority
                    16    ; Text nb.
                  F 100   ; EXEC Flag
$LAN
TEXT 16   "3:I0-7:O40-47"
$ENDLAN
          EPB
```


LRXS RECEIVE STATUS VIA LAN2

Description

Reads the status of a remote PCD into the defined status flags or read the statistics (traffic control) for its own station.
The status can be:

HALT	CPU is halted
RUN	CPU is rnning
DIS	The station is disconnected from the LAN2
CON	The station is conected to the LAN2

The 1st operand is the number of a Text containing the address of the station from which the status must be read.
The 2nd operand is the base address of 10 Diagnostic Flags (or Outputs).

The first Diagnostic element is the EXEC flag. It is initially set Low by LRXS, and remains Low on subsequent executions of the same LRXS instruction, until the transfer is complete.

When the transfer is complete the EXEC flag is set High.

The state of the EXEC flag is altered only when the LRXS instruction is executed.

The LRXS has always high priority.

If the status returned is "disconnected", then only the "Disconnected" Flag is set, since there was no error.

Usage

```
LRXS[X]   text    (i)    ; Number of Text containing command 0-3999
          diag           ; Diagnostic flags base address F|O 0-8182
```

Example

```
LRXS       100   ; Text 100 contains the command
         O 32    ; Diagnostic Outputs O 32-41
TEXT 100 "020"   ; Read status of station 20
```

Flags

Unchanged.

See Also

LTXD, LRXD, LTXS, LAN2 Diagnostic Flags, LAN2 Command Texts

LTXS TRANSMIT STATUS VIA LAN2

Description Changes the status of a remote PCD.

The status can be changed to:

HALT	Halt CPU
RUN	Run CPU
DIS	Disconnect Station from LAN 2
CON	Connect Station on LAN 2
TOUT:nnn	Set Time Out value where nnn = Value in 1/10 sec (10..250).

The 1st operand is the number of a Text containing the address of the station where the status must be changed and the new status.
The 2nd operand is the base address of 10 Diagnostic Flags (or Outputs).

The first Diagnostic element is the EXEC flag. It is initially set Low by LTXS, and remains Low on subsequent executions of the same LTXS instruction, until the transfer is complete.

When the transfer is complete the EXEC flag is set High.

The state of the EXEC flag is altered only when the LTXS instruction is executed.

The LTXS has always high priority.

Usage

```
LTXS[X]   text    (i)    ; Number of Text containing command 0-3999
          diag           ; Diagnostic flags base address F|O 0-8182
```

Example

```
LTXS       1000        ; Text 1000 contains the command
         F 100         ; Diagnostic Flags F 100-109
TEXT 1000 "035:DIS"    ; Disconnects station 35
```

Flags Unchanged.

See Also LRXS, LRXD, LTXD, LAN2 Diagnostic Flags, LAN2 Command Texts

LAN2 Diagnostic flags

For each SAIA LAN2 instruction, one operand gives the base address for the 10 diagnostic elements (Outputs or Flags)

Flag		High Status	Low Status
0	EXECuted	Command executed	Command in progress
+1	FAILure	Failure	No Failure
+2	Local PCD Status	Invalid Command Text	Command Text Valid
+3		Disconnected	Connected
+4		Time Out (Transmission error)	No Timeout
+5	Remote PCD Status	Disconnected	Connected
+6		Not used	Not used
+7		Write protected	Write Ok
+8		Halted (CPU 0)	Running (CPU 0)
+9	Watchdog (1)	LAN2 reconfigurated	

(1) from version V002 (PCD4) and V006 (PCD6)

EXEC Flag:

The first Diagnostic element is the EXECuted flag. While EXEC is Low, it indicates that the LAN2 instruction is still executing (receiving or transmitting data). The EXEC flag is initially set Low when a LAN2 instruction is first executed, and remains Low on subsequent executions of the same instruction, until the transfer is complete. When the transfer is complete the EXEC flag is set High.

Programs must be structured so that LAN2 instructions are continuously executed while the EXEC flag is Low.

If the LAN2 instruction is executed again, the data transfer is repeated. The state of the EXEC flag is altered ONLY when the LAN2 instruction is executed.

PRIORITY:

The transfer of information in the LAN2 is frame oriented; each frame is 32 bytes long and therefore can contain 8 R|C|T or 256 I|O|F.

When the station receives the "Token", only one frame is transmitted and the token goes to the following station. A transfer which is longer than one frame will be separated in different frames and will always have a low priority (1); if the high priority is requested flag +2 (Invalid command text) is set and the telegram is not sent.

A short telegram which is less then a frame can be sent with high priority (0) or low priority (1). If the priority is high, the frame can be inserted between successive frames of a long telegram (low priority).

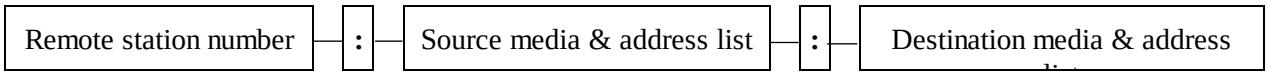
The status telegrams always have high priority (0).

LAN2 Command texts

Data transfers (LRXS/LTXS)

The LTXD and LRXD instructions need a command text which defines the data to be sent or received over the LAN2.
Always use CAPITAL LETTERS in these texts.

FORMAT:



Each command text contains the remote station number, the source media type and address list, and the destination media type and address list. The station number and source and destination address lists are separated by colons ':'.

The address list can be:

- a **range**: two addresses separated with a '-'
  eg: "I100-200" (Input addresses 100 to 200)
- a **list** of up to 8 single elements separated by commas
  eg: "R100,110,120".

The source and destination lists must match.

NOTE:
Not all combination of source and destination elements are valid (e.g. Flags cannot be transferred to Registers).

Source	Destination					Address
	O	F	T	C	R	Range
I	●	●				0..8191
O	●	●				0..8191
F	●	●				0..8191
T			●	●		0..450
C			●	●		0..1599
R					●	0..4095

Examples for RECEIVE DATA command (LRXD):

TEXT 100 "015:O64-127:F1000-1063"

This transfers the value of Outputs 64 to 127 from remote PCD station number 15 into Flags 1000 - 1063 of the local PCD.

TEXT 101 "008:T11,13,25:C55,125,1231"

This transfers the contents of Timers 11, 13 and 25 from the remote PCD station number 8 into Counters 55, 125 and 1231 of the local PCD.

Example for TRANSMIT DATA command (LTXD):

TEXT 75 "000:R11-22:R33-44"

This transfers the contents of Registers 11 to 22 from local PCD into Registers 33 to 44 of the remote PCD station number 0.

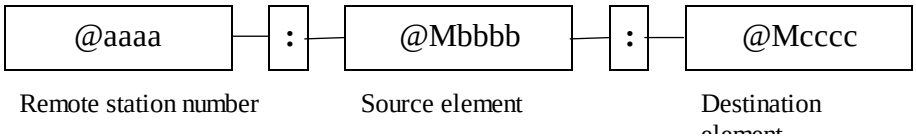
Indirect addressing

The number of texts used to define transfers via the LAN2 can be reduced by using indirect addressing: the effective address is determined by the contents of a register.

Each part of the LAN2 command text (station number, source and destination) can be indirected with the '@' character.

Indirect addressing of a single element

FORMAT:



where: @ indicates indirect addressing.
M Media type (I|O|F|T|C|R)
aaaa Register containing the remote station number
bbbb Register containing the address of the source element
cccc Register containing the address of the destination element

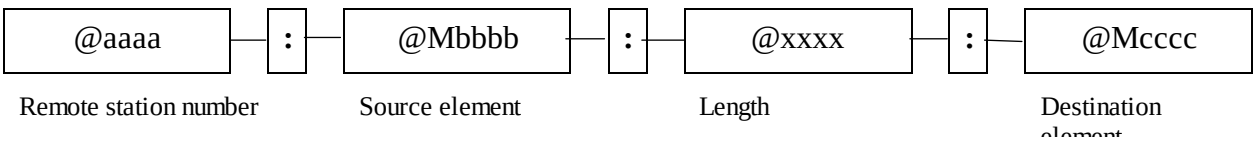
Example:

"@100:@I400:@F600"

The remote station number is in register 100. The Input which address is given by the register 400 will be transfered to the Flag which address is given by register 600

Indirect addressing of multiple elements:

FORMAT:



where: @ indicates indirect addressing.
M Media type (I|O|F|T|C|R)
aaaa Register containing the remote station number
bbbb Register containing the address of the first source element
xxxx Register containing the number of elements to transfer
cccc Register containing the address of the first destination element

Example:

"@200:@C100-@101:@C500"

The remote station number is in register 200. The first counter address is given by the contents of register 100, the number of counters to be transfered is in register 101. These counters will be copied to counters starting with the contents of register 500.

Mixed addressing

The direct and indirect addressing can be mixed in the same command text.

In the mixed addressing: the syntax of the indirect addressing must be followed.

Example:

"@5:R100-50:@99"

The remote station number is in register 5. 50 Registers are transferred beginning with Register 100 and are copied starting at the address contained in Register 99.

"25:I0-@55:F1000"

The remote station number is station 25. Inputs between I 0 and the contents of Register 55 are transferred onto Flags beginning at address 1000.

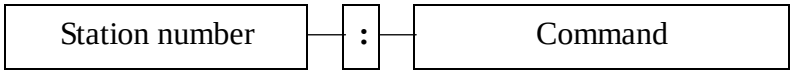
Remarks about indirect and mixed addressing

As the contents of the registers used in the indirect or mixed addressing can not be tested by the "Assembler", the user must take care to not go beyond the range of the addressed elements.

Going beyond the authorized range can have unpredictable results.

Transmit Status (LTXS)

The status of a station can be transferred via the LAN2.
FORMAT:



Station number	
0-254	The command is sent to the specified station
255	The command is sent to **all** stations connected to the network except the own station
Command	
HALT	Set all processors for the specified station in HALT User program is stopped Diagnostic flag 8 is set
RUN	Set all processors for the specified station in RUN
CON	Set the LAN2 station in connected state. Diagnostic flag 3 or 5 is set low.
DIS	Set the LAN2 station in disconnected state. In this case no instructions are executed and no data is transferred. Diagnostic flag 3 or 5 is set high
TOUT:NNN	This sets the timeout for the specified station NNN is the number of stations on the network (2-255)

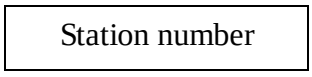
Example of TRANSMIT STATUS text (LTXS):

```
TEXT 213    "018:HALT"
```

The status of the Remote station 18 is changed to HALT.

Read Status (LRXS)

The status of a station can be read :
FORMAT:



The status is returned in the diagnostic flags (Flag 3, 5 or 8)

Example for RECEIVE STATUS text (LRXS):

```
TEXT 137    "015"
```

The status of station 15 is read into the status flag:

```
"Disconnected"      EXEC Flag + 5
"Halted"            EXEC Flag + 8
```

The "Failure" flag (EXEC Flag + 1) is not affected.

If the status returned is "disconnected", then only the "Disconnected" Flag is set, since there was no error.

Note: LAN 2 communication is only complete when the EXEC flag = H again after executing LRXS

Line traffic control (LRXS)

It is possible to check what happens on the SAIA LAN2 network with the LRXS instruction and a special status text :

FORMAT:

where xxxx is a register number (0-4095)

This command text is used to read the LAN2 transmission/reception statistics into 4 registers starting at register xxxx. This command is useful to diagnose hardware errors on the network.

Values returned are as follow:

Register	Description
xxxx	Number of received frames
xxxx + 1	Number of transmitted frames
xxxx + 2	Number of retries (= number of timeouts * 3)
xxxx + 3	Number of rejected messages (= corrupted messages)

Note: During the first communication with a station, the number of retries is incremented: it means that every communicating station has at least 1 retry. Values in the statistic registers are stored in 16 bits format (max value is 65535)

Example :

```
TEXT 123  "100:STAT:R20"
```

Read statistics of own station 100 and store the values in Registers 20 to 23.

Syntax of LAN2 Texts in the source file:

The syntax of LAN2 texts written between the $LAN ($LLAN) and $ENDLAN ($NOLAN, $NOLLAN) directives is checked during assembly. Lower case characters are also converted to the required upper case.

Futhermore, the Assembler converts the LAN2 texts into binary format, which makes LAN2 communication faster. The time gained is about 15 ms per command; this is particulary effective for short telegrams.

Examples:

```
$LAN
TEXT 0   "2:I0-255:F1000-1255"
TEXT 1   "5:r501-510:r101-110"
TEXT 2   "7:f0-31:o96-127"
TEXT 15  "2:con"
TEXT 37  "15"
$ENDLAN
```

Note: Binary LAN Texts cannot be displayed or edited from the debugger because they have a different format.

Using Symbols in LAN2 texts:

Symbols can also be used in LAN 2 texts.

The value and optionally the type of the symbol is inserted into the text. The symbol is written outside the ASCII text segment in double quotes, and must be separated from this or other symbols by a comma. After the symbol, an optional field width and prefix type can be given.

Format:

symbol [. [[-] [0] width] [t | T]]	
symbol	The symbol name. This can actually be any expression which includes a symbol, for example: MotorOn + 100, ... Symbols with floating point values are not permitted.
.	The dot immediately after the symbol indicates that a field width and/or a prefix is present.
Width	The field width: the number of digits or spaces required for the number. If the width begins with a 0, leading zeros are inserted.
t | T	Optional prefix type 't' or 'T'. If 't', the value is prefixed with the symbol's type in lower case (o, f, r, ...); if 'T', the symbol's type is in upper case (O, F, R,...)

Examples:

```
SOURCE     EQU  R 55
DEST       EQU  R 66

$LAN
TEXT  25   "8:" , SOURCE.T , ":" , DEST.T
$ENDLAN
```

This creates the text : `"008:R55:R66"`

10. CONTROL Instructions

These instructions control the execution of the program.

Jumps instructions are causes of errors (infinite loops, ...); these instructions must therefore be used with care. Jumps will be preferabily used in program blocks or function blocks rather than in the main program.

GRAFTEC allows you to program without using "jumps".

The operand of these instructions cannot be supplied as a Function Block parameter.

JR	Jump relative
JPD	Jump direct
JPI	Jump indirect
HALT	Halts the CPU
LOCK	Lock semaphore
UNLOCK	Unlock semaphore

Notes

Notes

JR JUMP RELATIVE

Description Conditionally or unconditionally jumps a specified number of program lines forwards or backwards from the current program line number.

The number of lines that can be jumped is -4095 (backwards) to +4095 (forwards), the program line jumped to is calculated by adding this value to the number of the program line containing the JR instruction. It is illegal to jump out of the current block (COB, PB, FB, ST, TR or SB): the destination MUST be in the current block.
The following **conditions codes** are valid:

-	Unconditionnal jump (condition code blank)
H	Jump if Accumulator = H (1)
L	Jump if Accumulator = L (0)
P	Jump if Positive flag = H (Negative flag = L)
N	Jump if Negative flag = H
Z	Jump if Zero flag = H
E	Jump if Error flag = H

If the condition is not true, the jump is not made; execution continues with the instruction following JR.

When programming using the Assembler, it is usual to use labels (symbolic names) for jump destinations. Labels can be any length, but only the first 8 characters are significant; the labels must always begin with a letter (A..Z).
When using an editor other than SEDIT, it is necessary to put a ':' after each label.

Usage

```
JR      [cc] offset      ; cc = condition code (H|L|P|N|Z|E)
                         ; offset is the relative number of lines
                         ; to be jumped (-4095.. +4095)
```

Example

```
JR      H -2        ; Jump 2 line above
JR      H REPEAT    ; Jump to label "REPEAT"
```

Flags Unchanged

See Also JPD, JPI, LD

Note Well structured programs have no jumps

```
        STH   I 15
        ANL     XBSY
        DYN   F 15
        JR    H next
        STXT    1
                57
        ….
Next:   ….
```

→

```
        STH   I 15
        ANL     XBSY
        CPB   H 25
        ….
        PB      25
        STXT    1
                57
        EPB
```


JPD JUMP DIRECT

Description Jumps conditionally or unconditionally to a program line number relative to the start of the current block (COB, XOB, PB, FB, ST or TR). The destination line number is always positive, between 0 and the number of lines in the current block (max 8191 lines). Labels can be used.

The following **conditions codes** are valid:

-	Unconditionnal jump (condition code blank)
H	Jump if Accumulator = H (1)
L	Jump if Accumulator = L (0)
P	Jump if Positive flag = H (Negative flag = L)
N	Jump if Negative flag = H
Z	Jump if Zero flag = H
E	Jump if Error flag = H

If the condition is not true, the jump is not made; execution continues with the instruction following JPD.

Usage

```
JPD     [cc] offset     ; cc = condition code (H|L|P|N|Z|E)
                        ; offset  is offset from start of block (0..8191)
```

Example

```
JPD     L 10    ; If the ACCU is Low, a jump is made
                ;   to 10th line of the current block.
```

Flags Unchanged

See Also JR, JPI

JPI JUMP INDIRECT

Description

Similar to JPD: jumps conditionally or unconditionally to a program line number relative to the start of the current block (COB, XOB, PB, FB, ST or TR).
The program line number is read from the given Register number (only the least 13 bits are significant). Since this instruction utilises a condition code, the 'R' data type code is omitted.

The following **conditions codes** are valid:

-	Unconditionnal jump (condition code blank)
H	Jump if Accumulator = H
L	Jump if Accumulator = L
P	Jump if Positive flag = H (Negative flag = L)
N	Jump if Negative flag = H
Z	Jump if Zero flag = H
E	Jump if Error flag = H

If the condition is not true, the jump is not made; execution continues with the instruction following JPI.

The value of a label can be loaded into a Register using the LD instruction.

Usage

```
JPI     [cc] reg     ; cc = condition code (H|L|P|N|Z|E)
                     ; reg is the Register number containing
                     ; the offset from the start of the block (0..8191)
```

Example

```
JPI     H 300    ; If the ACCU is High, a jump is made
                 ;  to the line of the current block stored in
                 ;  Register 300
```

Flags Unchanged

See Also JR, JPI, LD

Note

Care must be taken that the destination of the jump is not outside of the current block.

HALT HALTS THE CPU

Description Conditionally or unconditionally Halts the CPU. The Halt state is not the same as the Stop state. After a HALT, the CPU can only be set to Run by a Restart operation, or by powering the PCD off and on.

If the condition is not true, the HALT is not made; execution continues with the following instruction.

NOTE: After a HALT of CPU 0, a Restart Cold can be executed only on all CPUs. The status of the outputs after the HALT is defined by the hardware configuration (jumpers).

The following **conditions codes** are valid:

-	Unconditionnal jump (condition code blank)
H	Jump if Accumulator = H (1)
L	Jump if Accumulator = L (0)
P	Jump if Positive flag = H (Negative flag = L)
N	Jump if Negative flag = H
Z	Jump if Zero flag = H
E	Jump if Error flag = H

Usage

HALT	**[cc]**	**; cc = condition code: H|L|P|N|Z|E**

Example HALT E ; Halts if the Error (E) flag is set

Flags Unchanged

See Also User's Guide

Practice In case of Error, stops the PCD and memorise in register some diagnostic informations

```
XOB       13
DIAG    R 1000
HALT
EXOB
```

WARNING

The HALT instruction should mainly be used during the commissioning phase. It should only be used in a completed application program with the utmost care !

LOCK LOCK SEMAPHORE

Description LOCK in conjunction with UNLOCK, is used to prevent access conflicts when several CPUs read or write the same elements. 100 Semaphores (special flags) are available (0-99). The LOCK instruction checks the Semaphore. If it is High (another CPU has executed a LOCK), then the ACCU is set Low. If it is Low, the ACCU and the Semaphore are set High.

It is the programmers responsibility to ensure that the CPU does not reference an element if the associated Semaphore is High (ACCU = L (0) after LOCK).

The UNLOCK instruction clears the Semaphore. An UNLOCK instruction MUST quickly follow a LOCK instruction so that no CPU is blocked from accessing an element for too long.

Usage

LOCK	**semaphore**	**; Semaphore 0-99**

Example

```
LOCK      1     ; If Semaphore 1 is low (data is not being
CFB     H 100   ; accessed by another CPU), then call FB 100.
                ; Semaphore 1 is used to protect data accessed
```

Flags **ACCU** set High/Low

See Also UNLOCK

Practice see UNLOCK

UNLOCK UNLOCK SEMAPHORE

Description UNLOCK in conjunction with LOCK, is used to prevent access conflicts when several CPUs read or write the same elements. 100 Semaphores (special flags) are available (0-99). The UNLOCK instruction clears the Semaphore.

Usage

```
UNLOCK    semaphore        ; Semaphore 0-99
```

Example UNLOCK 1 ; Semaphore 1 is set Low

Flags Unchanged.

See Also LOCK

Practice A PCD is equipped with two CPUs. CPU 0 compares the contents of 2 registers while CPU1 transfers BCD information into one of these two registers.

```
CPU 0

...
LOCK       1
CFB      H 10
….
FB         10
CMP      R 88
         R 89
UNLOCK     1
EFB
```

```
CPU 1

...
LOCK       1
CFB      H 100
….
FB         100
DIGI       2
         I 16
         R 88
DIGI       2
         I 24
         R 89
UNLOCK     1
EFB
```

The use of semaphore 1 ensures that CPU 0 never compares the two registers while CPU 1 is executing the DIGI instructions, and is altering the contents of the registers. If CPU 0 were to compare the registers at the same moment that CPU 1 was updating them, it might compare a new value with an old one. Semaphore 1 also prevents CPU1 executing the DIGI instructions until CPU 0 has finished the CMP instruction.

11. DEFINITION Instructions

These instructions are executed on power up, and are executed ONCE only.
If an instruction is executed again it is ignored.

Normally these instructions will be placed in the start-up XOB 16.

The operand of these instructions cannot be supplied as a Function Block parameter.

DEFVM	Define volatile memory (Flags)
DEFTC	Define Timers/Counters
DEFTB	Define timebase
DEFTR	Define Timer Resolution
DEFWPH	Define write protected area (Halt)
DEFWPR	Define write protected area (Run)

Notes

Notes

DEFVM DEFINE VOLATILE MEMORY (Flags)

Description Defines the area of Flags which are to be non-volatile (battery backed-up). Non-volatile Flags retain their values even after power to the PCD is lost. Volatile Flags are all set to 0 on power-up of the PCD. All Flags **ABOVE** the Flag indicated in the operand are defined as being non-volatile.

If the instruction is not used, ALL Flags are non-volatile by default.

This instruction is executed by CPU 0 only.

Usage

DEFVM	**flag**	**; 0-8190, vol/non-vol Flag partition**

Example

```
DEFVM   200     ; Flags   0 - 199 are volatile,
                ;         200 - 8191 are non-volatile
```

Flags Unchanged.

See Also DEFTC, DEFTB, DEFWPR, DEFWPH

Practice Assume that Flags 0..200 must be declared to be volatile

```
XOB         16      ; Cold start XOB

DEFVM       200     ; Flags 200..8191 are non-volatile
....
EXOB
```


DEFTC DEFINE TIMER/COUNTERS

Description Defines the number of Timers for the PCD. Timers and Counters occupy the same addressing space. All elements **BELOW** the operand value are Timers, all the others are Counters.

If the instruction is not used, the default is:

Timers: 0 - 31
Counters: 32 - 1599.

NOTE: Do not define more Timers than are actually required by the program. The handling of each Timer affects the program execution speed. The maximum number of Timers allowed is 450.

This instruction is executed by CPU 0 only.

Usage

DEFTC	**ctr**	**; Lower limit for Counters (0-450)**

Example

```
DEFTC   64      ; Timers 0-63, Counters 64-1599
```

Flags Unchanged.

See Also DEFTB, DEFTR, DEFVM, DEFWPR, DEFWPH

Practice Assume that 100 Timers are necessary for an application

```
XOB     16      ; Cold start XOB

DEFTC   100     ; 0..99 are Timers
                ; 100..1599 are counters
....
EXOB
```


DEFTB DEFINE TIMEBASE

Description Defines the timebase for the decrementing of the Timers. The operand indicates the timebase in 10's of milliseconds. Values of 1 to 1000 are valid (10 ms to 10 sec).

If the timebase is not defined (no DEFTB), the default is 100 ms (1/10 sec).

DEFTB is processed by CPU 0 only.

For the other CPUs, DEFTB defines the timebase for the internal timers used by the delayed instructions SETD and RESD. Therefore, the internal timers of CPU 0 always have the same timebase as the user Timers; the timebase of the internal timers of the other CPUs may differ from that of the user Timers.

Note: care should be taken when defining a low timebase (eg. 10 ms) and a large number of timers (the handling of a large amount of Timers can slow down the program execution speed).

Usage

DEFTB	**timebase**	**; time base in 10's of milliseconds (1- 1000)**

Example

```
DEFTB     100  ; Timebase = 1 sec (100 * 10ms)
```

Flags Unchanged.

See Also DEFVM, DEFTC, DEFWPR, DEFWPH,SETD, RESD

Practice Assume that for a slow process, the minimal temporisation is 1 sec.

```
XOB       16    ; Cold start XOB

DEFTB     100   ; Time base is 100 * 10 ms = 1000 ms
....
EXOB
```


DEFTR DEFINE TIMER RESOLUTION

Description Defines the resolution of decrementation of the Timers in **milliseconds**. For example, if a "DEFTR 100" is specified, all non-zero Timers will be decremented by 100 every 100 ms. A "DEFTR 1000" will decrement all Timers by 1000 every 1000ms and so on. If DEFTR and DEFTB are used in the same program , the message "**DOUBLE TIME BASE**" will appear in the History List and the CPU will automatically put itself in "HALT" upon a restart cold or on powerup.

The advantage of the DEFTR instruction (over the DEFTB) is that the values you specify when using timers are independant of the timebase or resolution and always introduced in multiple of 10 ms. For the DEFTR instrcution to have an influence on the Timers it must be programmed in CPU0. The DEFTR instruction allows a maximum timer resolution of 10ms which means that the value specified in the instructions is rounded if necessary .

Example: DEFTR 25: a time base of 20 ms will be set (25 rounded down to 20). The DEFTR instruction, as with the DEFTB instruction, also acts on the instructions SETD, RESD and OUTD. If the DEFTR instruction is present in the user program then the time base of these instructions is fixed to **10ms** independent of the specified value by DEFTR

Usage

DEFTR resolution ; resolution ≥ 10 ms

Example DEFTR 100 ; Timer resolution = 100 msec

Flags Unchanged.

See Also DEFTB

Practice The output 20 will be set 150ms (15 * 10ms) after the instruction has been executed.

```
XOB       16
DEFTR     200
…
EXOB

COB       0
          0
….
SETD    O 20
          15
…
ECO
```


DEFWPR DEFINE WRITE PROTECTED AREA (RUN)

Description Defines which elements are to be protected from being overwritten by the LAN2.

DEFWPR defines elements to be write protected when the CPU is in Run (DEFWPH defines elements to be write protected when the CPU has Halted or stopped).

In both instructions, the operand defines the element type and the top end of the range to be protected. Elements addressed from 0 up to this value are write protected. The instructions must be executed once for each element type to be protected: O, F, T, C, R. If all element types are to be protected, DEFWPR must be executed five times.

If the instructions are not present, NO elements are write protected in the RUN state.

This instruction is executed by CPU 0 only.

Usage

DEFWPR adds ; O 0-8191, F 0-8191, T 0-450, C 0-1599, R 0-4095

Example

```
DEFWPR  F 999  ; Flags 0-999 are write protected (RUN)
               ; and therefore can not be overwritten by
               ; another LAN2 station
```

Flags Unchanged.

See Also DEFWPH, DEFTC, DEFTB, DEFVM, LAN2

Practice In an application using a LAN 2, 1000 local flags and 500 registers must be protected from being written by another station when the CPU is in RUN.

```
XOB          16    ; Cold start XOB

DEFWPR   F 999     ; Protected Flags 0..999
DEFWPR   R 499     ; Protected Registers 0..499
....
EXOB
```


DEFWPH DEFINE WRITE PROTECTED AREA (HALT)

Description Defines which elements are to be protected from being written to by the LAN2.

DEFWPH defines elements to be write protected when the CPU has Halted (DEFWPR defines elements to be write protected if the CPU is in Run).

In both instructions, the operand defines the element type and the top end of the range to be protected. Elements addressed from 0 up to this value are write protected. The instructions must be executed once for each element type to be protected: O, F, T, C, R.

If all element types are to be protected, DEFWPH must be executed five times. If the instructions are not present, NO elements are write protected in the Halt state.

This instruction is executed by CPU 0 only.

Usage

DEFWPH adds ; O 0-8191, F 0-8191, T 0-450, C 0-1599, R 0-4095

Example

```
DEFWPH  C 79   ; Timers and Counters 0-79 are      write protected (in Halt state)
```

Flags Unchanged.

See Also DEFWPR, DEFTC, DEFTB, DEFVM, LAN2

Practice In an application using a LAN 2, 1000 local flags and 500 registers must be protected from being written by another station when the CPU is in RUN and in HALT.

```
XOB         16    ; Cold start XOB

; Define protection when CPU is in RUN
DEFWPR  F 999   ; Protected Flags 0..999
DEFWPR  R 499   ; Protected Registers 0..499

; Define protection when CPU is in HALT
DEFWPH  F 999   ; Protected Flags 0..999
DEFWPH  R 499   ; Protected Registers 0..499
....
EXOB
```


12. SPECIAL Instructions

NOP	No operation
RTIME	Read time
WTIME	Write time
PID	P.I.D. control
TEST	Test hardware
DIAG	Read XOB diagnostic
SYSRD	System Read
SYSWR	System Write
SYSCMP	System Compare

The following instructions must no longer be used but are maintained for compatibility reason:

ALGI	Analogue input
ALGO	Analogue output

These two instructions works only with the analogue card PCA2.W1x. To read or write values to analogue cards PCD2, PCD4 and PCD6, consult the appropriate hardware manual.

STHS	Start high slow
OUTS	Out slow

These instructions were used for accessing slow I/O modules such as the PCA2.W2x / W3x.

Notes

Notes

NOP NO OPERATION

Description Do-nothing instruction.
Used for patching out other instructions, or for leaving space in the code for future additions or modifications.

Usage

NOP	**; Has no operand**

Example

```
NOP         ; Does nothing
```

Flags Unchanged.

RTIME READ TIME

Description Reads the contents of the internal hardware clock into two Registers. The first Register is specified in the instruction. After the RTIME instruction, the Registers are set as follows:

Digit number	9	8	7	6	5	4	3	2	1	0
Register	0	0	0	0	Hour	Hour	Min	Min	Sec	Sec
Register + 1	0	Week	Week	Wday	Year	Year	Month	Month	Day	Day

Week	Week number	1..53
Wday	Day of week number	1..7 (Monday = 1, Sunday = 7)
Year	Year	0..99
Month	Month of year	1..12
Day	Day of month	1..28/29/30/31 (month dependent)
Hour	Hour	0..23
Min	Minute	0..59
Sec	Second	0..59

The register data is stored in binary, NOT in BCD, but can be moved or output in BCD using the DIGO instruction.

Usage

```
RTIME     reg                ; Register number R 0-4095
```

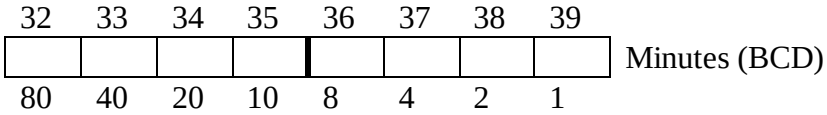
Example

```
RTIME    R 10    ; Clock is copied into Registers 10 and 11
```

Flags Unchanged.

See Also WTIME, DIGO

Practice After switching on Input 3, the actual minutes of the clock should be displayed in BCD-format on outputs 32-39




```
COB          0
             0
STH       I 3
DYN       F 3
CPB       H 25
...
ECOB
```

```
PB           25
RTIME     R 20
MOV       R 20
          D 2
          R 99
          D 0
MOV       R 20
          D 3
          R 99
          D 1
DIGOR       2
          R 99
          O 32
EPB
```


WTIME WRITE TIME

Description Writes the contents of two Registers to the internal hardware clock. The first of the two Registers is specified in the instruction. The format of the Register contents is as for the RTIME instruction:

BCD values can be loaded into the Registers from Flags etc. using the DIGI instruction.

Usage

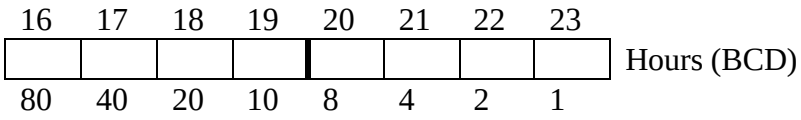
WTIME reg ; Source Register R 0-4095

Example `WTIME R 500` ; Loads the clock from Registers 500 and 501

Flags Unchanged.

See Also RTIME, DIGI

Practice After switching on Input 4, the hours of the clock should be set on a new value. The new value is to be read from the BCD switches on inputs 16-23.




```
COB       0
          0
STH     I 4
DYN     F 4
CPB     H 26
...
ECOB
```

```
PB        26
RTIME   R 200
DIGIR     2
        I 16
        R 199
MOV     R 199
        D 0
        R 200
        D 4
MOV     R 199
        D 1
        R 200
        D 5
WTIME   R 200
EPB
```


PID PID CONTROL ALGORITHM

Description Implements a PID algorithm, using data defined in a 13-Register block.

Register	Usage	Symbol		
+0	New Result	Y_n	*	size is 'm' bits
+1	Previous Result	Y_{n-1}	*	
+2	New Controlled Variable	X_n	w	size is 'm' bits
+3	Prev. Controlled Variable	X_{n-1}	*	
+4	Reference Variable	W_n	w	size is 'm' bits
+5	Prev. Set point Variable	W_{n-1}	*	
+6	Proportional Factor	F_p	w	* 256
+7	Integral Factor	F_i	w	* 256
+8	Derivative Factor	F_d	w	* 256
+9	Dead Range	D_r	w	
+10	Cold Start Y	Y_s	w	Starting value for Yn
+11	Precision in bits	m	w	m = 8, 12 or 16 bits
+12	Workspace	Z_s	*	

* These values are handled by the PID instruction.
w These values must be written into the register by the user program.

Usage

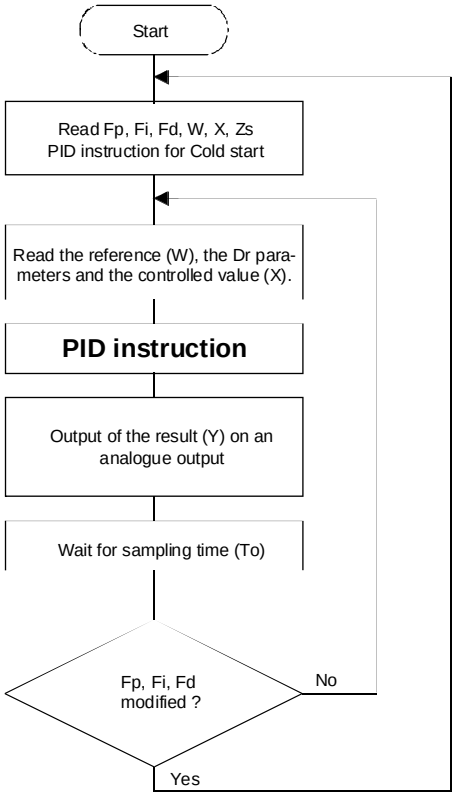
```
PID     reg     ; reg is the lowest address of 13 Registers
                ; (R 0-4083)
```

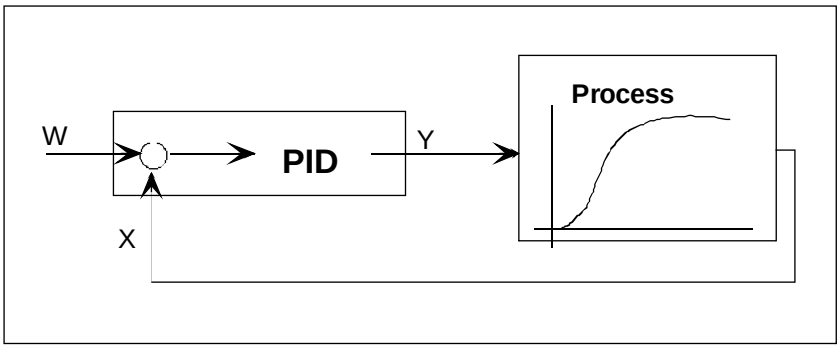
Example

```
PID     R 1000  ; Uses R 1000-1012 for the PID control data
```

Flags Unchanged

Practice A typical PID control loop must consist of the following:





New Result Y_n:

This is the actual result to control the process determined by the system program from the following equation with $Z_s = Z_s + (W_n - X_n)$:

$$Y_n = \frac{F_p}{256} * \{ \ (W_n - X_n) + \frac{F_i}{256} * Z_s + \frac{F_d}{256} * [(W_n - W_{n-1}) - (X_n - X_{n-1})] \ \}$$

If the result exceeds the declared precision in bits, it will be limited to its maximum value (m bits) or, in case of a negative result, it will be set to 0.

Previous Result Y_{n-1}:

This is the old result determined in the previous operation.

Controlled Variable X_n:

The controlled variable X_n is read from the process and written to the register (R+2) by the user program.
The controlled variable should be maximum 'm' bits

Previous Controlled Variable X_{n-1}:

This is the old controlled variable used in the previous arithmetic operation.

Reference W_n:

The reference (setpoint) is written to the register (R+4) by the user program. The reference should be maximum 'm' bits.

Previous Reference W_{n-1}:

This is the old reference used in the previous arithmetic operation.

Proportional Factor F_P:

This factor determines the proportional (amplification) characteristic of the regulator and is written to the register (R+6) by the user program.
When calculating, only the 16 lower bits are used (0..65535)
The Proportional factor is determined as follows:

$$F_p = \frac{1}{X_p} * 256 \qquad \text{with } X_p\text{: Proportional band}$$

Note: To enter a proportional band of 5 %, the F_p factor must be set to:

(1 / 0.05) * 256 = 5120

A cold start of the PID must be executed after a modification of F_p or F_i

Integral Factor F_i:

This factor determines the integral characteristic of the regulator and is written to the register (R+7) by the user program.
When calculating, only the 16 lower bits are used (0..65535)
The Integral factor is determined as follow:

$$F_i = \frac{T_0}{T_i} * 256$$ with T_0: sampling time of the PID instruction
T_i : integral time

A cold start of the PID must be executed after a modification of F_p or F_i

Derivative Factor F_d:

This factor determines the derivative characteristic of the regulator and is written to the register (R+8) by the user program.
When calculating, only the 16 lower bits are used (0..65535)
The Derivative factor is determined as follow:

$$F_d = \frac{T_d}{T_0} * 256$$ with T_0: sampling time of the PID instruction
T_d : derivative time

Dead Range D_r:

The dead range defines the range in which the variations of the controlled variable may occur without causing a modification of the Result variable (Y_n).

Cold Start Y_S:

This value is used as starting value for Y_n by the system program. As soon as the user program writes a value other than 0 to the cold start register, a cold start calculation is made:

$Y_n = Y_s$
$Y_{n-1} = Y_s$
$Z_s = [(Y_s * 256/F_p) - (W_n - X_n)] * 256/F_i$
$W_{n-1} = W_n$
$X_{n-1} = X_n$

The value of Y_s is automatically reset to 0 by the system program after being used once and will not be used again.

For a Cold Start with an output value of 0, the Y_s register must be set to -1.

When $F_i = 0$, the Y_n value can not be initialised with a Cold Start. A Cold Start is however recommended to initialise the workspace register. In this case, the Y_s value is ignored, the Z_s register is set to 0 and Y_n take the value of the proportional part of the algorythm.

Note: Changing from manual to automatic control is a typical application of a cold start calculation. In order to achieve a smooth transition, Y_s may be set equal to the currently output variable (Y_n).

Resolution m:

The maximum values of X, W, Y_n and Y_s are determinated by the resolution.
If m= 8: 8 bits are used (0..255)
If m= 12: 12 bits are used (0..4095)
If m= 16: 16 bits are used (0..65535)
The resolution is mostly defined by the analog module used for the Result variable output. If the resolution for the input and output are not the same, the Y_n value must be adapted after the PID instruction.

Sampling Time:

The sampling time T_o must be done outside the PID instruction with a timer.
In practice: $T_o \approx 0{,}1$ time constant of the process (T_o must be at least 80 ms)

Calculation capacity:

The workspace register Z_s has a maximum capacity of 2^{31}.
When using 16 bits values (m = 16), an overflow can occur; in this case the PID will not work properly.
To avoid this problem, the factor F_p must be ≥ 2 if m = 16 (There is no problem when m = 8 or 12).

TEST TEST HARDWARE

Description Conditionally or unconditionally tests selected hardware of the PCD.

If any test fails, the test is aborted, and the ACCU is set Low (0).
If all the selected tests pass, the ACCU is set High (1).

Individual tests are selected as follows:

Value	Bit number	Test Description
	11	
400	10	Public Memory Loss
200	9	Memory Extention Corruption
100	8	System RAM Memory
	7	
40	6	System Firmware Checksum
20	5	Serial Channels
10	4	Real Time Clock
	3	
4	2	User Program/Text Checksum
2	1	User Program/Text RAM
1	0	Public RAM (F, T, C, R, Mailbox)

For every bit set, the corresponding test is done. Tests 0 and 5 are executed if the tested CPU is the only one in Run; if any other CPUs are running, these tests are NOT performed.

Note: Some of the tests are very slow, and should not be done during normal operation of the PCD, run the tests on startup or during an idling period.

The operand cannot be supplied as a Function Block parameter.

Usage

```
TEST    [cc] number    ; cc = condition code (H|L|P|N|Z|E)
                       ; number = number which defines the tests
```

Example

```
TEST      50    ; Unconditionally tests System Firmware Checksum (40)
                ;   + Real Time Clock (10)
TEST    L 4     ; If ACCU = L (0), then verify checksum of user program/text
```

Flags **ACCU** set High (1) if all tests pass, Low (0) if any test fails.

Public RAM Test (Value = 1)

Tests the RAM which contains the F/R/T/C's with a save-write-read-compare-restore operation.
This test is not performed if another CPU is in RUN in a multiprocessor environment.

ACCU	Error Flag	Description
0	1	Another CPU was in RUN
0	0	A Public RAM error was detected
1	x	Public RAM is OK

User Program/Text RAM Test (Value = 2)

Tests the RAM which contains the program/text with a save-write-read-compare-restore operation.
If the memory is not RAM or if the RAM is write-protected, then the User Program/Text Checksum test is performed.

ACCU	Error Flag	Description
0	1	Program Header invalid
0	0	RAM is faulty
1	0	RAM is OK

User Program/Text ChecksumTest (Value = 4)

Calculates the sum of the whole Program + Text area and compares it with the checksum disposed in the header.

ACCU	Error Flag	Description
0	1	Program Header invalid
0	0	Checksum is not OK
1	0	Checksum is OK

Real Time Clock Test (Value = 10)

Checks the existence of the RTC and tests if it is incrementing correctly.
Any other CPU accessing the RTC at the same time as this test is performing will be blocked by up to 15 ms.

ACCU	Error Flag	Description
0	1	RTC does not exist
0	0	RTC is faulty
1	0	RTC is OK

Serial Channels Test (Value = 20)

Test the serial channels by initialising the port to local loopback mode and then transmitting a test pattern and verifying the reception of the same pattern.
If any of the serial channels is assigned, this test is not performed.

ACCU	Error Flag	Description
0	1	One of the serial port is assigned
0	0	Port is not OK
1	0	Serial channels are OK

System Firmware Checksum Test (Value = 40)

The firmware system EPROMs are checked

ACCU	Error Flag	Description
0	0	Checksum is invalid
1	0	Cheksum is OK

System RAM Memory Test (Value = 100)

The system RAMs are checked

ACCU	Error Flag	Description
0	0	Checksum is invalid
1	0	Cheksum is OK

Memory Extension Corruption Test (Value = 200)

The memory extension is tested during startup and if there is a corruption it is indicated here by using an internal flag.

ACCU	Error Flag	Description
0	0	Memory extension was corrupted
1	0	No corruption has occured

Public Memory Loss Test (Value = 400)

If the test pattern stored in the mailbox is not valid when it is tested during the startup routine then it is assumed that the Public RAM has been corrupted during power down due to the battery discharching..

ACCU	Error Flag	Description
0	0	Public Memory was corrupted
1	0	No corruption has occured

DIAG READ XOB DIAGNOSTIC

Description Fills a 12-Register block with diagnostic information relating to the last or the present Exception Organisation Block (XOB) executed. The operand is the lowest Register number of the block of 12 Registers. DIAG is normally used within an XOB.

Register block usage:

Register.		
0	XOB Number	Number of last or present XOB
+1	Program Line	Program Line when XOB was called
+2	Index Register	Value of Index Register when called
+3	COB Program Line	Program Line of Level 0 call
+4	Nesting Level 1 Program Line	Program Line of Level 1 call
+5	Nesting Level 2 Program Line	Program Line of Level 2 call
+6	Nesting Level 3 Program Line	Program Line of Level 3 call
+7	Nesting Level 4 Program Line	Program Line of Level 4 call
+8	Nesting Level 5 Program Line	Program Line of Level 5 call
+9	Nesting Level 6 Program Line	Program Line of Level 6 call
+10	Nesting Level 7 Program Line	Program Line of Level 7 call
+11	Not Used	Reserved

The program line numbers of the block calls (Nesting level information) give the program line where the previous call (CFB, CPB, etc) took place. From these, it can be established exactly where the program was when the XOB was executed.

Note: The most important information are provided by register R and R+1

The operand cannot be supplied as a Function Block parameter.

Usage

DIAG	**reg**	**; R 0-4084, lowest address of 12 Registers**

Example

```
DIAG    R 1000   ; Stores diagnostic information in  Registers 1000-1011
```

Flags

See Also User's Guide

Practice The address of the line where an error occurs must be printed.

```
          XOB       13
          DIAG    R 1000
          STXT      1
                    100
TEXT 100  "$D $ H :"
          " Error Flag set at address $R1001 <CR><LF>"
          EXOB
```


SYSRD SYSTEM READ

Description Read the PCD system parameters like: PCD Device type, CPU type, Firmware version, User program name, S-Bus parameters, …

Usage

SYSRD	**Function**	**; Function code, K code, R 0..4095**
	Result	**; Result of the read, R 0..4095**

Function

K x or R x: Constant or register containing a function code. This instruction can either be direct, by using a constant for the function code or indirect by using a register. It permits the user to have access to useful system information via the user program.

Result

R 0..4095 Register containing the result or first of 2 registers (see code 5400) or first of a set of registers (see codes 65xx)

Example

```
SYSRD   K 5000  ; Read the PCD type in ASCII
        R 20    ;  and put the result in R 20
```

Flags If the function code does not exist, the Error flag is set.

See Also SYSWR

Function codes

Code	Function description	Result
2000 2001 2002 2003 2004 2005 ¦ 2049	**Read User EEPROM** Register 0 Register 1 Register 2 Register 3 Register 4 Register 5 Register nn Register 49 PCD1 other PCD	Value contained in the EEPROM
5000 5010	**Read Device type** in ASCII in decimal	ASCII / Dec / Type " D1" / 1 / PCD1 " D2" / 2 / PCD2 " D4" / 4 / PCD4 " D6" / 6 / PCD6
5100 5110	**Read own CPU type** in ASCII in decimal	ASCII / Dec / Type " M1_" / 10 / PCD1.M1 " M1_" / 10 / PCD2.M12 " M15" / 15 / PCD2.M15 " M11" / 11 / PCD4.M11 " M12" / 12 / PCD4.M12 " M14" / 14 / PCD4.M14 " M24" / 24 / PCD4.M24 " M34" / 34 / PCD4.M34 " M44" / 44 / PCD4.M44 " M1_" / 10 / PCD6.M1 " M2_" / 20 / PCD6.M2 " M3_" / 30 / PCD6.M3 " M54" / 54 / PCD6.M5
5200 5210	**Read own Firmware version** in ASCII in dec	Examples of valid responses: " $4C", " 004", " X41" Ex: 5 dec for Version 005 -1 dec for any '$', 'X', 'β'
5400	**Read User program name** in ASCII The user program name always contains 8 ASCII characters	**R x** contains the upper 4 bytes of the program name in ASCII **R x+1** contains the lower 4 bytes of the program name in ASCII
6000	**Read S-Bus station number**	Example of result: 2 station number = 2 -1 station number not configured
6010	**Read S-Bus PGU TN delay**	Example of result: 10 Delay in mS -1 S-Bus not configured
6020 6030	**Read S-Bus PGU TS delay** **Read S-Bus PGU timeout**	

Code	Function description	Result	
6040	**Read S-Bus PGU baudrate**	Example of result: 9600 bps -1 S-Bus not configured	
6050	**Read S-Bus PGU mode**	Status	Dec
		BREAK without modems PARITY without modems DATA without modem BREAK with modems PARITY with modems DATA with modem S-Bus not configured	0 1 2 10 11 12 -1
6060	**Read S-Bus PGU port number**	Example of result: 1 S-Bus PGU port configured on port 1 -1 S-Bus not configured	
6070	**Read S-Bus level**	Status	Dec
		S-Bus Level 1 (reduced) S-Bus Level 2 (full) S-Bus not configured	1 2 -1
6080	**Read current PGU owner** (S-Bus or P8 protocol)	CPU 0 CPU 1	0 1
6100	**Read modem status byte** Reads the current status of the modem connection. This information tells the user at what stage the modem is at in the initialisation procedure. Results: 2 PCD waiting for modem connection. 6 .. 39 PCD initialising the modem. 40 Reassign serial port for mode SS1/SS0. 45..49 Connection to modem has been lost. This is an intermediate status before the modem in reinitialised. 50 Everything is OK and PCD is online in mode SS0/1.		
6500 6510 6520	**Read modem type string** . **Read modem reset string** **Read modem initialisation string** Read the specified modem string from the user program extended header into the block of registers starting with base address **R x**		
7000	**Read system counter**	0.. 2 147 483 647	
	A internal System Counter is incremented every millisecond. This System Counter is reset to 0 at power up, so a "Restart Cold" , for instance, doesn't affect it. The period of the System Counter is exactly: **24 days 20 hours 31 min 23 sec 647 ms** For an example see the SYSCMP instruction.		

Read real time

It is possible to read each value separately depending on function code. The return value is in decimal format. The function codes are between 7050 and 7090. The function code 7090 allows to know the number of seconds elapsed since midnight (00:00:00), 01/01/1970, co-ordinated universal time, according to the system clock.

Function code table:

Function code:	For:
7050	Seconds
7051	Minutes
7052	Hours
7053	Minutes and seconds
7054	Hours and minutes
7055	Hours, minutes and seconds
7060	Day
7061	Month
7062	Year (< 100)
7063	Month and day
7064	Year and month
7065	Year, month and day
7070	Day of week
7071	Week of year
7072	Week of year and day of week
7081	Time and date (on two registers)
7090	Seconds elaped since 1970

Examples:

```
1)  SYSRD    7055     ; Read hours, minutes and seconds
          R  0

    Result:  R 0:  120203


2)  SYSRD    7081     ; Read time and date
          R  0

    Result:  R 0:  120203    R 1: 991130
```

Notice:
When the user takes a function code between 7050 and 7090 which is not in this table, XOB 13 is called and the error flag is set.

SYSWR SYSTEM WRITE

Description This is the complement to SYSRD and it allows modification of system information or initialisation of system functions via the user program

Usage

```
SYSWR    Function    ; Function code, K code, R 0..4095
         Value       ; Value to write
```

Function

K x or R x:	Constant or register containing a function code. This instruction can either be direct, by using a constant for the function code or indirect by using a register. It permits the user to have access to useful system information via the user program.

Value

K y	Value to be written
R 0..4095	Register containing the value to be written

Example

```
SYSWD    K 4014  ; Initialize the XOB 14 with a frequency
         K 10    ;  of 10 ms
```

Flags If the function code does not exist, the Error flag is set.

See Also SYSRD

Code	Function description
1000	**System Watchdog (PCD1 and PCD2 only)** Permitted values of **K y** or **R y**: 0 Disactivate WDOG 1 Activate WDOG and make a restart cold if not refreshed within 200 ms 2 Activate WDOG and call XOB 0 before making a restart cold if not refreshed within 200 ms. Once the watchdog is activated the instruction must be repeated continually within 200 ms intervals. A watchdog XOB 0 is distinguished from the power down XOB 0 from the initial error massage written into the history list. When the WDOG is provoked the message "XOB0 WDOG START" is written into the history list, for a powerdown XOB 0 the message is "XOB 0 START EXEC".
2000 2001 2002 2003 2004 2005 ¦ ¦ ¦ ¦ 2049	**Write EEPROM (not on all PCD)** The PCD is equipped with an EEPROM of max. 49 user registers which are written in the following way : Function code (2000 - 2049) indicates EEPROM register 0 .. 49. Permitted values of **K y** or **R y**: PCD1: max. 5 registers (0 .. 5) Other PCD: max. 49 registers (0 .. 49) R y: Source register containing value to be written into EEPROM. Warning: A maximum of 100,000 user writes is permitted on the EEPROM so do not execute this instruction frequently in your user program. The SYSWR instruction takes 20mS to execute so it is should not be used in XOB 0. This instruction should be used to configure values for initialisation of systems.
4000	**Set XOB overflow limit** The XOBs 14/15/17/18/19/20/25 all work using a queuing mechanism. If an XOB is active then the pending XOB is placed in a queue which has a maximum size of 127 entries *per* XOB. If this limit is surpassed then XOB 7 is called and the queue is cleared. The error message 'SYSTEM OVERLOAD' is written into the History list. This limit of 127 entries can sometimes be too large for real time applications so it is now possible to define a user limit with this instruction. This limit is common to all XOBs which can be queued. Permitted values of **R y** or **K y**: 0 .. 127

Code	Function description
4005 4013	**Enable/disable XOB 5 / 13** Enable or disable the XOBs 5 or 13. In some cases, execution of these XOBs immediatley after they have been provoked, complicates the execution of the user program. For this reason, it is now possible to disable these XOBs. If an XOB is provoked one or more times whilst it is disabled then it will be called once upon being reenabled. Function code: 4005 XOB 5 4013 XOB 13 Permitted values of **R y** or **K y**: 0 Disable the XOB 1 Enable the XOB 2 Clears the Error Flag in the current COB and in the active XOB (For K 4013 only)
4014 4015	**Install XOB 14 /15** Configure periodic XOB with the frequency defined in **Ky** or **Ry**. It is possible to configure two periodic XOBs with a frequency from 5 ms to 1000s. The value in **Ky** or **Ry** is given in ms, if it is zero then the XOB is deactivated. This instruction can be executed at any time. If an XOB is already being executed when an XOB becomes pending then it will be queued until a time when there is no XOB active and it can be executed. The XOBs are only executed if the CPU is in RUN or CONDITIONAL RUN. Function code 4014 Configure XOB 14 4015 Configure XOB 15 Permitted values of **R y** or **K y**: 5 .. 1 000 000
4017 4018 4019	**Execute XOB 17 /18 / 19** Execute the XOB specified in **R x** or **K x** on the CPU specified in **K y** or **R y**. The XOBs 17/18/19 are user XOBs which can be provoked via S-BUS or the user program. The XOBs are only executed if the CPU is in RUN or CONDITIONAL RUN. Function code : 4017 Execute XOB 17 4018 Execute XOB 18 4019 Execute XOB 19 Permitted values of **R y** or **K y**: 0 .. 6 CPU on which XOB will be provoked 7 Provoke XOB on own CPU 8 Provoke XOB on all CPUs.

Code	Function description
6000	**Write S-Bus station number** Change the S-Bus station number to the value held in **K y** or **R y.** This instruction will work for user program in EPROM and in RAM. Permitted values of **K y** or **R y**: 0 .. 254
7000	**FFP-IEEE Conversion** Convert between FFP (Fast Floating Point format) and IEEE format for floating point values. The FFP standard is used by ALL Floating Point instructions in SAIA PCDs. Once a value is converted to IEEE format no floating point operations can be carried out on the value. Function code 7000 FFP to IEEE format 7001 IEEE to FFP format Permitted values of **R y**: **R y** contains the value to be converted. The result is stored in the same register.

Write real time

It is possible to write each value separately depending on function code and each value is on 2 digits, for example: 12h, 2 min and 3 sec, it will be written 120203. The function codes are between 7050 and 7081 as showing the following table.

Function code table:

Function code:	For:
7050	Seconds
7051	Minutes
7052	Hours
7053	Minutes and seconds
7054	Hours and minutes
7055	Hours, minutes and seconds
7060	Day
7061	Month
7062	Year (< 100)
7063	Month and day
7064	Year and month
7065	Year, month and day
7081	Time and date (on two registers)

Examples:

```
1) LD     R  0
             120203

    SYSWR     7055     ; Write hours, minutes and seconds
          R  0


2) LD     R  0
             120203
   LD     R  1
             991130

   SYSWR     7081     ; Write time and date
          R  0
```

Notice:
When the user takes a function code between 7050 and 7081 which is not in this table, XOB 13 is called and the error flag set.

SYSCMP SYSTEM COMPARE

Description

The SYSCMP instruction is able to transform any register into a pseudo Timer. It's task is to compare the sum of the first and second operands to the System Counter and set the ACCU according to the result.

If the result of the addition is greater than the System Counter, the ACCU is set High (1). If the result of the addition is smaller than or equal to the System Counter, then the ACCU is set Low (0).

The advantage of this instruction coupled to the instruction SYSRD K 7000 is that it is now possible to have Timers with a resolution of 1 ms. We also can measure the time between two events to a resolution of 1ms

Usage

SYSCMP	**R x**	**; R 0..4095**
	K y or R y	**; K 0..16383 or R 0..4095**

Example

```
SYSCMP  R 100   ; Compare contents of Register 100 + 1500
        K 1500  ;   to System Counter and set ACCU accordingly

SYSCMP  R 100   ; Compare content of R100 + R101
        R 101   ;   to System Counter and set ACCU accordingly
```

Flags **ACCU**

See Also SYSRD

Practice

Programming a high resolution Timer (1ms) with SYSRD and SYSCMP
This example shows how to program a high resolution Timer (1ms) with the instruction SYSRD and SYSCMP.

```
        COB     0
                0
        …
        LD      R 100   ; Load the time to wait in ms (1500)
                K 1500  ; in R 100

        SYSRD   K 7000  ; Read the System Counter in R 101
                R 101

wait:   SYSCMP  R 101   ; Compare System Counter to R100 + R101
                R 100   ; and set ACCU accordingly
        JR      H wait  ; If ACCU = High (1) then loop
        …
        ECOB
```


ALGI ANALOGUE INPUT

Description Reads a 12-bit value from a **PCA2.W1x** analogue module, and stores it in the specified Register.
The 1st operand contains both the A/D channel number (0-7) and the base address of the module.
The 2nd operand is the destination Register number.

If the first operand is supplied as an FB parameter, both the A/D channel number and the base address must be supplied on the same line.

Usage

```
ALGI[X]   c   base          ; c = channel 0-7, base = 0-8176
          register  (i)     ; Destination register R 0-4095
```

Example

```
ALGI    2   64  ; Reads analogue value from channel 2, at
        R   10  ; module base address 64 and saves in R 10
```

Flags The **Zero** (Z) and **Sign** (P or N) flags are set according to the value read.
The **Error** (E) flag is always set Low.

See Also ALGO

Note This instruction cannot be used for PCD4.Wxxx and PCD6.Wxxx modules (see the respective hardware manuals).

ALGO ANALOGUE OUTPUT

Description Outputs a 12-bit binary value from the specified Register to a **PCA2.W1x** analogue module.
The 1st operand is the Register to be output.
The 2nd operand contains both the D/A channel number, and the base address of the module.

If the second operand is supplied as an FB parameter, both the D/A channel number and the base address must be supplied on the same line.

Usage

```
ALGO[X]   register   (i)      ; Source register R 0-4095
          c  base             ; c = channel 0-3, base = 0-8176
```

Example

```
ALGO      R  100    ; Outputs the value in R 100
          3  128    ; to channel 3 of module at base address 128
```

Flags Unchanged

See Also ALGI

Note This instruction cannot be used for PCD4.Wxxx and PCD6.Wxxx modules (see the respective hardware manuals).

STHS START HIGH SLOW

Description The ACCU is set to the logical state of the addressed element, usually an Input. This is the same as the STH instruction, except that the timing on the PCD I/O bus is slightly slower, and it is therefore suitable for slow I/O modules. Program execution speed is not significantly affected.

Use this instruction to access Analogue modules PCA2.W2x/W3x.

Usage

STHS[X] element (i) ; I 0-8191, O 0-8191, F 0-8191

Example

```
STHS    I 25
```

Flags The **ACCU** is set to the logical state of the specified element

See Also OUTS, STH, Bit instructions

OUTS SET ELEMENT FROM ACCUMULATOR SLOW

Description The specified element, usually an Output, is set to the state of the ACCU. This is the same as the OUT instruction, except that the timing on the PCD I/O bus is slightly slower, and it is therefore suitable for slow I/O modules. The program execution speed is not significantly affected.

Use this instruction to access Analogue modules PCA2.W2x/W3x

Usage

OUTS[X] element (i) ; I 0-8191, O 0-8191, F 0-8191

Example

```
OUTS    O 32
```

Flags The **ACCU** is set to the logical state of the specified element.

See Also OUT, OUTD, STHS

Practice The analogue value of channel 0 from a PCA2.W2x (base address 96) must be read and stored in Register 100.

After the conversion is made with the OUTS instruction, 8 binary bits can be read starting from the module base address + 8 (=104)

```
COB         0
            0
...
ACC       H             ; Be sure that ACCU is High
OUTS        96          ; Select analog channel
...                       ; Wait ± 100 ms *)
CPB         RD_VAL      ; Call RD_VAL program block
....
ECOB

PB          RD_VAL
BITIR       8           ; Read 8 bits binary in reversed order
          I 104         ;    from address 104 .. 111
          R 100         ;    into  Register 100
EPB
```

*) The analogue module PCA2.W2x has a conversion time of ≤ 100 ms. This wait function can be done by inserting a number of consecutive NOP instructions. (The number of NOPs is depending from the CPU type).

Notes:

13. History list

The following is a detailed description of all the errors that can be reported in the HISTORY LIST or the HALT REASON REGISTER. Whenever an error is detected in the CPU, a message is stored in the HISTORY LIST which can be viewed from the PG3 debugger with the command "Display History".

Errors which provokes an XOB (if programmed)

Message	HALT	XOB	System *)	Meaning
XOB START EXEC	N	0	All	XOB 0 has been started
XOB 0 EXECUTED	N	0	All	XOB 0 has been completed during a power down
XOB 0 WDOG START	N	0	1	The system watchdog has been activated
EXTERN PWR FAIL	N	1	2, 5, 6	Extension rack power failure
PARITY FAILURE	N	4	5, 6	PCD6 Backplane error
SYSTEM OVERLOAD	N	7	All	The queuing mechanism for the level 3 XOBs has overloaded.
ILLEGAL OPCODE	N	8	All	Executes XOB 8 then makes a restart cold after an invalid instruction has been detected
>32 ST/TR ACTIVE	N	9	All	Too many active GRAFTEC tasks
>7 CALL LEVELS	N	10	All	PB/FB nesting depth overflow

The following errors have a fixed entry in the history table with an error counter

Message		HALT	XOB	System	Meaning
BATT FAIL	000	N	2	2, 4, 5, 6	The Battery has discharged
IO QUIT FAIL	000	N	5	4, 5, 6	An I/O location has been accessed which is not equipped
IR OVERFLOW	000	N	12	2, 4, 5, 6	Index Register incremented beyond 8191
ERROR FLAG	000	N	13	2, 4, 5, 6	Error flag set

*) System:
1: PCD1
2: PCD2
4: PCD4
5: PCD6.M540
6: PCD6.M1.., M2.., M3..

System Startup Errors

All the following errors are detected on POWER-UP of the PCD

Message	HALT	System	Meaning
RTC FAILURE	Y	All	Real Time Clock is detected but is not working correctly
DUART HW ERROR	Y	All	One of the DUARTs is defective
CHECKSUM FAIL	Y	All	User Program EPROM checksum fail
BAD TXT/DB TABLE	Y	All	Caused by an unsuccessful 'make text table' in the startup
TXT/DB HW ERROR	Y	All	Caused by an unsuccessful 'make text table' in the startup
BAD MEM EXT INIT	Y	All	Caused by an unsuccessful 'make text table' for memory extension in the startup
USR MEM HW ERROR	Y	6	Caused by unsuccessful 'user program test' in the startup
CPU SYNCH ERROR	Y	6	Caused if the time-out of the 2nd CPU
CPU FIRMWARE MIX	Y	4, 6	Multiple CPU system equipped with incompatible versions

Serious System Errors

These errors are written into the HALT REASON register which is read by the PG3 Debugger when a HALT is detected. They can occur on POWER-UP or when the PCD is in RUN.

Message	HALT	System	Meaning
BUS QUIT FAILURE	Y	All	FW has attempted to access non-existent address
68K INVALID OPC	Y	All	A invalid 68000 assembly instruction has been executed
68K ADDR ERROR	Y	All	Attempted to access an odd address
ZERO DIVIDE	Y	All	Internal system FW error
68K CHK INSTR	Y	All	…
68K TRAPV INSTR	Y	All	…
PRIVILEGE VIOL	Y	All	…
TRACE	Y	All	…
ILLEGAL AUTO VEC	Y	All	…
INTERRUPT ERROR	Y	All	…
RESERVE INT	Y	All	…

Programming or Configuration Error Messages

The following errors will be detected on POWER-UP of the PCD

Message	HALT	System	Meaning
EVERYTHING IS OK	N	All	Normal power up message
MODIFIED PROGRAM	N	All	User Program has been modified by PG3 Debugger. Only indicated when user program is write protected.
CPU NUMBER > 6	Y	6	The CPU number set at the DIL switch is invalid
CPU 0 START FAIL	Y	4, 5, 6	CPU 1-6 only : No CPU can be put in run without a program in CPU 0
INIT-FAILURE	Y	All	More than 32 GRAFTEC Initial steps have been defined
HEADER FAIL	Y	All	USER PROGRAM header is corrupted
NO PROGRAM	Y	All	CPU has no program to execute
MEM-EXT CORRUPT	Y	All	The memory extension in RAM has been corrupted
INVALID OPCODE	Y	All	Invalid IL (AWL) instruction has been downloaded into the CPU
MEDIA CORRUPTION	Y/N	All	Caused by battery failure
DOUBLE TIME BASE	Y	All	DEFTB and DEFTR instructions in same program
BAD MODEM STRING	Y	1	Modem string in EEPROM too long

The following errors will be detected whilst the PCD is in RUN, those which put the PCD in HALT will also write the message in the HALT REASON

Message	HALT	System	Meaning
BLOC NONEXISTENT	Y	All	Call to missing PB, FB, SB, ST, TR executed
HALTED BY LAN-2	Y	4, 6	The LAN-2 coprocessor has put the PCD in HALT
LAN-2 WATCHDOG	N	4, 6	The LAN-2 FW watchdog has been activated
HALT INSTRUCTION	Y	All	A HALT user instruction has been executed
MANUAL HALT	Y	4, 5, 6	CPU has been halted by the HALT switch
HALTED BY CPU 0	Y	4, 6	The Coprocessor(s) have been halted by CPU 0
SBUS-PGU ERROR	N	All	Invalid assignation of S-BUS PGU on a port

Notes

Main menu

Performance overview from PCD1 up to PCD6

PCD1: compact, small, for demanding situations

PCD2: for small but demanding control tasks

Series xx7: the SIMATIC® S7 compatible programmable controller

SAIA®PCD for factory automation

PCD4: for tasks with a broad requirement profile

PCD6: up to 6 processor modules and 30 communications interfaces

PCD2.M250: PLC + PC in <u>one</u> compact industrial unit

From small, low cost text terminals up to touch screen terminals

SAIA®PCD for building automation

Programming tools

Overview manuals

General informations