

SIEMENS

SIMATIC S5

S5-135U

CPU 928

Programming Guide

Contents	
CPU 928 Programming Guide C79000-B8576-C633-01	1
Multiprocessor Communication User's Guide C79000-B8576-C468-05	2

Order No.: 6ES5 998-1PR21
Release 01

Pocket Guide CPU 922/CPU 928/CPU 928B/CPU 948
Order No.: 6ES5 997-3UA22
is included in the manual

Copyright

Copyright © Siemens AG 1993 All Rights Reserved

The reproduction, transmission or use of this document or its contents is not permitted without express written authority.

Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

Disclaimer of Liability

We have checked the contents of this manual for agreement with the hardware and software described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcomed.

Technical data subject to change.

Safety-related guidelines

This manual contains notices which you should observe to ensure your own personal safety, as well as to protect the product and connected equipment. These notices are highlighted in the manual by a warning triangle and are marked as follows according to the level of danger:



Warning

indicates that death, severe personal injury or substantial property damage can result if proper precautions are not taken.



Caution

indicates that minor personal injury or property damage can result if proper precautions are not taken.

Only qualified personnel should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground and to tag equipment, systems and circuits in accordance with established safety practices and standards.

SIEMENS

SIMATIC S5 CPU 928

Programmer's Guide

C79000-B8576-C633-01

Contents

1	Introduction: S5 135U Mode of Operation and Application	1-1
1.1	New Features and Functions of the CPU 928.....	1-7
2	User Program.....	2-1
2.1	Programming Language STEP5.....	2-1
2.1.1	Methods of Representation: LAD, CSF, and STL.....	2-2
2.1.2	Structured Programming.....	2-3
2.1.3	STEP 5 Operations.....	2-4
2.1.4	Numeric Representation.....	2-5
2.1.5	STEP 5 Blocks.....	2-9
2.2	Organization, Program and Sequence Blocks.....	2-13
2.2.1	Programming	2-13
2.2.2	Calls.....	2-14
2.2.3	Special Organization Blocks.....	2-16
2.2.4	Special-function Organization Blocks	2-18
2.3	Function Blocks.....	2-19
2.3.1	Structure of Function Blocks.....	2-20
2.3.2	Programming Function Blocks.....	2-22
2.3.3	Calling and Assigning Parameter to Function Blocks.....	2-26
2.3.4	Special Function Blocks.....	2-29
2.4	Data Blocks.....	2-31
2.4.1	Structure of a Data Block.....	2-31
2.4.2	Programming Data Blocks.....	2-32
2.4.3	Opening Data Blocks.....	2-33
2.4.4	Special Data Blocks.....	2-35
3	Program Processing.....	3-1
3.1	Summary.....	3-1
3.1.1	Program Organization.....	3-1
3.1.2	Program Storage.....	3-5
3.1.3	Running the STEP5 User Program.....	3-6
3.1.4	Running the Program.....	3-9
3.2	STEP5 Operation Set with Examples of Programming.....	3-10
3.2.1	Basic Operation Set.....	3-13
3.2.2	Supplementary Operation Set.....	3-37

4	Operating States	4-1
4.1	Operating States and Program Levels.....	4-1
4.2	Operating State STOP.....	4-6
4.3	Operating State START-UP.....	4-9
4.3.1	Cold Restart and Manual Warm Restart.....	4-11
4.3.2	Automatic Warm Restart.....	4-13
4.3.3	Interruption during START-UP.....	4-14
4.4	Operating State RUN.....	4-16
4.4.1	CYCLE: Cyclic Program Execution.....	4-17
4.4.2	TIME INTERRUPT: Time-driven Program Execution.....	4-18
4.4.3	CONTROLLER INTERRUPT: Processing of Controllers.....	4-23
4.4.4	PROCESS INTERRUPT: Interrupt-driven Program Execution...	4-24
5	Handling Interrupts and errors	5-1
5.1	Frequent Errors in the User Program.....	5-1
5.2	Evaluation of Error information.....	5-2
5.3	Control Bits and Interrupt Stack (ISTACK).....	5-7
5.4	Error Handling Using Organization Blocks.....	5-18
5.5	Errors during START-UP.....	5-21
5.5.1	DBO-FE (Error in DB 0).....	5-22
5.5.2	DB1-FE (Error in DB 1).....	5-22
5.5.3	DB2-FE (Error in DB 2).....	5-24
5.5.4	DX0-FE (Error in DX 0).....	5-25
5.6	Errors in RUN and START-UP.....	5-26
5.6.1	BCF (Command Code Error).....	5-27
5.6.2	LZF (Execution Time Error).....	5-30
5.6.3	ADF (Addressing Error).....	5-34
5.6.4	QVZ (Acknowledgement Delay).....	5-35
5.6.5	ZYK-FE (Cycle Time Error).....	5-36
5.6.6	WECK-FE (Collision of Two Time Interrupts).....	5-37
5.6.7	REG-FE (Controller Error).....	5-37
5.6.8	ABBR (Abort).....	5-40
6	Integrated Special Functions	6-1
6.1	Handling of the Registers.....	6-5
6.1.1	Access to the Condition Code Byte (OB 110).....	6-5
6.1.2	Clear Accus 1, 2, 3 and 4 (OB 111).....	6-7
6.1.3	Roll Up Accu (OB 112) and Roll Down Accu (OB 113).....	6-8
6.2	Structure Commands.....	6-10
6.2.1	Counter Loops (OB 160 through 163).....	6-10
6.3	Read Block Stack (BSTACK) (OB 170).....	6-12

6.4	Block Handling.....	6-16
6.4.1	Variable Data Block Access (OB 180).....	6-16
6.4.2	Test Data Blocks (DB/DX) (OB 181).....	6-20
6.4.3	Transfer Flags to Data Block (OB 190, 192).....	6-22
6.4.4	Transfer Data Fields to Flag Area (OB 191 and OB 193)...	6-24
6.4.5	Transfer Data Blocks to DB-RAM (OB 254, OB 255).....	6-29
6.5	Multiprocessor Communication (OB 200 through OB 205)....	6-31
6.6	Page Access	6-32
6.6.1	Writing Data to a Page (OB 216).....	6-35
6.6.2	Reading Data from a Page (OB 217).....	6-37
6.6.3	Assigning a Page (OB 218).....	6-39
6.7	Sign Extension (OB 220).....	6-43
6.8	System Functions.....	6-44
6.8.1	Switch On/Off "Disable All Interrupts" (OB 120) and Switch On/Off "Delay All Interrupts" (OB 122).....	6-44
6.8.2	Switch On/Off "Disable Individual Time Interrupts" (OB 121) and Switch On/Off "Delay Individual Time Interrupts" (OB 122).....	6-46
6.8.3	Set Cycle Time (OB 221).....	6-49
6.8.4	Restart Cycle Time (OB 222).....	6-49
6.8.5	Compare Start-up Modes (OB 223).....	6-50
6.8.6	Transfer Interprocessor Communication IPC Flags as a Block (OB 224).....	6-50
6.8.7	Read Word from the System Program (OB 226).....	6-51
6.8.8	Read Check Sum of System Program (OB 227).....	6-52
6.8.9	Read Status Information of Program Level (OB 228).....	6-54
6.9	Functions for Standard Function Blocks (OB 230 through OB 237).....	6-56
6.10	Shift Register.....	6-57
6.10.1	Initialize Shift Register (OB 240).....	6-60
6.10.2	Process Shift Register (OB 241).....	6-63
6.10.3	Erase Shift Register (OB 242).....	6-64
6.11	Control: PID Algorithm.....	6-65
6.11.1	Initialize PID Algorithm (OB 250).....	6-72
6.11.2	Process PID Algorithm (OB 251).....	6-73
7	Extended Data Block DX 0.....	7-1
8	Memory Assignment and Memory Organization.....	8-1
8.1	Address Distribution in the CPU 928.....	8-2
8.1.1	Address Distribution - System-RAM.....	8-3
8.1.2	Address Distribution - I/O	8-4
8.2	Memory Organization in the CPU 928.....	8-7
8.2.1	Block Headers in the User Memory and the DB-RAM.....	8-7
8.2.2	Block Address Lists in Data Block DB 0.....	8-8
8.2.3	RI/RJ Area.....	8-12
8.2.4	RS/RT Area (System Data Assignment).....	8-12

9	Memory Access Using Absolute Addresses.....	9-1
9.1	Access to Registers and the Memory via an Address in Accu 1.....	9-5
9.2	Transfer of Memory Blocks.....	9-13
9.3	BR Register Operations.....	9-19
9.3.1	Loading of the BR Register.....	9-19
9.3.2	Shifting of the BR Register Contents.....	9-20
9.3.3	Access to the Local Memory.....	9-21
9.3.4	Access to the Global Memory.....	9-21
9.3.5	Access to the Page Frame.....	9-24
10	Multiprocessor Operation.....	10-1
10.1	Notes.....	10-1
10.2	Data Exchange between the Processors.....	10-3
10.2.1	Interprocessor Communication (IPC) Flags.....	10-4
10.2.2	Multiprocessor Communication.....	10-8
10.2.3	"Protected" Transfer of Connected Data Fields.....	10-8
10.3	I/O Assignment.....	10-9
10.3.1	Data Block DB 1.....	10-9
10.4	Start-up during Multiprocessor Operation.....	10-12
10.5	Test Operation.....	10-13
11	Testing Aids: Online Functions.....	11-1
11.1	Online Function 'STATUS VARIABLES'.....	11-3
11.2	Online Function 'STATUS'.....	11-4
11.3	Online Function 'PROCESSING CONTROL'.....	11-5
11.4	Online Function 'CONTROL'.....	11-9
11.5	Online Function 'CONTROL VARIABLES'.....	11-9
11.6	Online Function 'COMPRESS' memory.....	11-10
11.7	Online Function 'START'/'STOP'.....	11-10
11.8	Online Function 'PC OVERALL RESET'.....	11-11
11.9	Online Function 'OUTPUT ADDRESS'.....	11-11
11.10	Online Function 'MEMORY CONFIGURATION'.....	11-11
11.11	Table: Activities at the Checkpoints.....	11-12

ANNEX

A	Technical Data S5-135U.....	A-1
B	Summary of Error Identifications.....	B-1
C	STEP5 Command Summary.....	C-1
D	STEP5 Commands (arranged in alphabetical order).....	D-1
E	STEP5 Commands (arranged according to command code).....	E-1
F	STEP5 Commands Not Contained in the CPU 928.....	F-1
G	Summary of the Program Level Identifiers.....	G-1
H	Example: How to Evaluate the ISTACK.....	H-1

Index

List of Figures, Examples and Summaries

Important:

**This manual refers to the CPU 928 with
MLFB no. -3UA12 (12 MHz).**

Where to find what in the manual

Chapter 1 gives an introduction into a processor's mode of operation and internal structure. It describes the typical S5-135U system structure as well as the new features and functions of CPU 928.

Chapter 2 illustrates the user program structure and explains special features of the STEP 5 programming language. This is followed by a description of the various STEP 5 program blocks and how they are programmed.

Chapter 3 contains information on the cyclic program processing in CPU 928, the organization and storage of programs. A list of the entire STEP 5 operation set as well as many sample programs have also been included. (Additional information on STEP 5 operations can be found in the 'STEP 5 list of operations'. Please, also pay attention to the specifications made under 'Literature references'.)

Chapter 4 describes the operating states of CPU 928 (START-UP, RUN, STOP) and defines the term 'program level'. It explains possible program levels in the individual operating states of the processor. In addition, you will find important information on time-driven and interrupt-driven program processing.

Chapter 5 contains detailed information on error diagnostic and error handling. You will find a description of typical errors in START-UP and RUN and learn how to detect errors and handle them. The structure of the interrupt stack (ISTACK) and how to evaluate it is described by means of examples.

Chapter 6 deals with 'integrated special functions' and gives many application examples.

Chapter 7 illustrates the structure and the programming of data block DX 0 which can be used to easily adapt specific features and functions of CPU 928 to your requirements (incl. sample programs).

Chapter 8 gives detailed information on the memory areas of CPU 928. Experienced users will learn how system data is assigned.

Chapter 9 is dedicated to very experienced users who have a profound knowledge of the system. It contains all the STEP 5 commands used to access the entire memory via absolute addresses and describes the individual registers of CPU 928.

Chapter 10 gives additional information on multiprocessor operation and describes the structure and the programming of data block DB 1 which is necessary for multiprocessor operation. The particular features of test operation are explained at the end of this chapter.

Chapter 11 describes some online functions which you can call at the programmer to test your program. We will also turn your attention to peculiarities which may arise in connection with CPU 928.

Abbreviations

ABBR	Abort
Accu 1(2,3,4)-L	Low word in accumulator 1 (1,2,3), 16 bits
Accu 1(2,3,4)-H	High word in accumulator 1 (2,3,4), 16 bits
Accu 1(2,3,4)-LL	Low byte of low word in accu 1 (2,3,4), 8 bits
Accu 1(2,3,4)-LH	High byte of low word in accu 1 (2,3,4), 8 bits
ADL	Addressing error
BASP	Command output inhibit
BCD	Binary coded decimal number
BCF	Command code error
BSTACK	Block stack
C	Counter (counter locations)
CP	Communications processor
CPU	Central processing unit
COR	Coordinator
CSF	Control system flowchart
D, DL/DR, DW, DD	Data (1 bit), left-hand/right-hand data (8 bits), data word (16 bits), data double word (32 bits)
DB	Data block
DBA	Data block start address (in register 6)
DBL	Data block length (in register 8)
DSP 0, DSP 1	Condition codeword (often referred to as CC 0, CC 1 or ANZ 0, ANZ 1)
DX	Data block extended
EPROM	Erasable Programmable Read Only Memory
ERAB	First scan (bit indication)
F, FY, FW, FD	Flag bit, flag byte, flag word, flag double word
FB	Function block
FX	Extension function block
IP	Intelligent I/O module
ISTACK	Interrupt stack
LAD	Ladder diagram
LZF	Execution time error
OB	Organization block
OB, OW	Byte, word from "extension I/O" range
OR	Or (bit indication)
OS	Latching type overflow (word indication)
OV	Overflow (word indication)
PB	Program block
PB, PW	I/O byte (PG 675), I/O word
PC	Programmable controller
PG	Programmer
PI	Process image
PII	Process image of inputs
PIO	Process image of outputs
Proc.	Processor
PY	I/O byte (PG 685)
QVZ	Acknowledgement delay
RAM	Random Access Memory
RLO	Result of logic operation (bit indication)
SAC	STEP address counter (in register 15)
SB	Sequence block
SF	Special function
STA	Status (bit indication)
STL	Statement list
T	Timer (timer locations)
ZYK	Cycle error

Literature references

How to program in STEP 5 (introduction) and how to program the programmable controller SIMATIC S5-135U is explained in the following manuals:

S5-135U programmieren mit STEP 5 ¹⁾
Siemens AG, ISBN 3-8009-1461-1
(S/R processors)

Automatisieren mit SIMATIC S5-135U ²⁾
Siemens AG, ISBN 3-8009-1522-7
(S/R processors and CPU 928)

Also note the remaining chapters of your manual (Instructions, STEP 5 Operations List) dealing with CPU 928.

1) German version only

2) English version under preparation

1 Introduction: S5 135U Mode of Operation and Application

This chapter is for those users who have not yet worked with a programmable controller but who have had experience with other microcomputer systems.

Structure of the system

A programmable controller (PC) is a computer system that has been specifically developed for industrial use, e.g. for controlling manufacturing machines. PC's have a modular design and consist of a sub-rack with at least one processor module - from now on simply referred to as "processor" - and numerous peripheral units. The number and type of processors used depends on the particular automation task in question.

The S5 135U programmable controller belongs to the SIMATIC S5 family of stored program controllers. It is a high-performance multiprocessor device designed for process automation (open and closed-loop control, signalling, monitoring, logging) that can be used for constructing simple controls with binary signals as well as for solving extensive automation tasks.

The central unit of the S5-135U can be configured with

- a processor for single processor operation or
- a coordinator (COR) and up to 4 processors for multiprocessor operation,
- additional communications processors (CP's):
up to 7 CP's for single processor operation or 4 to 6 (7) CP's for multiprocessor operation.

The remaining slots in the central controller of the S5 135U are available for input and output modules. In order to expand the interface system you can attach extension racks (EU's) to the central unit.

Also refer to the catalog "Programmable Controller S5 135U", ST 54.1 Order no. E86010-K4654-A111-A3.

The following figure shows the typical construction of an S5-135U system. The modules highlighted by thicker lines are sufficient for single processor operation.

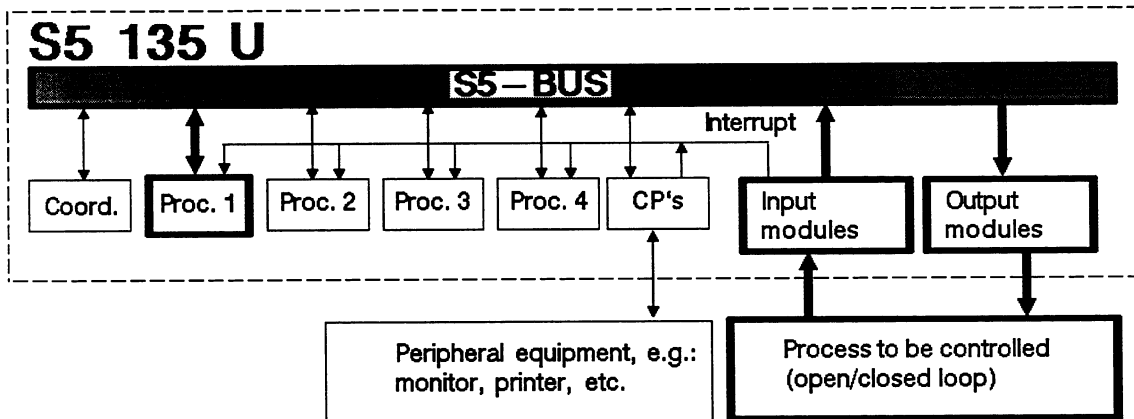


Fig. 1-1: Typical S5 135U system structure

Application

Depending on the particular situation one of the following processors can be used for simple automation tasks in single processor operation:

- S processor, particularly suitable for open-loop control tasks (fast bit processing)
- R processor, particularly suitable for closed-loop control tasks (fast byte processing)
- M processor, designed for measured-value processing; programmable in Assembler and in high-level programming languages (BASIC, C)
- CPU 928, universally applicable, fast bit and byte processing.

In contrast to many other PC's, the S5 135U central controller is able to operate with several processors simultaneously for more complex automation tasks, making it a multiprocessor device.

Multiprocessor operation is useful whenever the process to be controlled is too complex for one processor and when it can be divided into several more or less independent sub-tasks. Each sub-task can be assigned to the most suitable processor (see above). Each processor performs its individual task independent of the other processors.

The processors access the I/O modules one after the other using a common bus (= S5 bus). An additional module, the coordinator, allocates the S5 bus to the processors successively in fixed time periods. Only the processor the bus has been allocated to is capable of accessing the I/O's.

Processors can exchange data with each other via the S5 bus. This data exchange makes use of a mailbox on the coordinator.

Mode of operation

Within a processor, the following cycle is constantly repeated:

1. All the input modules assigned to a processor are scanned and the data read are temporarily stored in the process image of the inputs (PII).
2. Data contained in the PII is processed by the user program and the data to be output is entered in the process image of the outputs (PIO).
3. Data contained in the PIO is transferred to the output modules assigned to a particular processor.

The time which the processor needs in order to perform these tasks is called the cycle time.

The cycle must be fast enough, to ensure that process states do not change faster than the processor can react; otherwise the process may get out of control. Twice the cycle time has to be considered as the maximum reaction time. The cycle time is dependent on the type and complexity of the user program (see below) and is often not constant.

An additional time-driven program can be provided for processes needing control signals at regular time intervals. On completion of one of these intervals the cyclic program is interrupted to allow the time-driven program to be executed. Up to 9 time-driven programs are possible in the CPU 928! The cycle time increases by the amount which is needed for processing the time-driven program.

In the processor, an interrupt-driven program can be assigned to a process signal that has to be responded to very quickly. After such an interrupt, the processor breaks off the time-driven or cyclic program in order to process the interrupt-driven program. The cycle time increases by the amount which is needed for processing the interrupt-driven program.

At worst the cycle time is the sum of the time needed for the cyclic program and the possibly repeatedly called up time- and interrupt-driven programs.

Each processor monitors the cycle time. It will interrupt the program if a programmable limit values is exceeded and set itself and the others in stop status, then it cancels the output signals.

Program

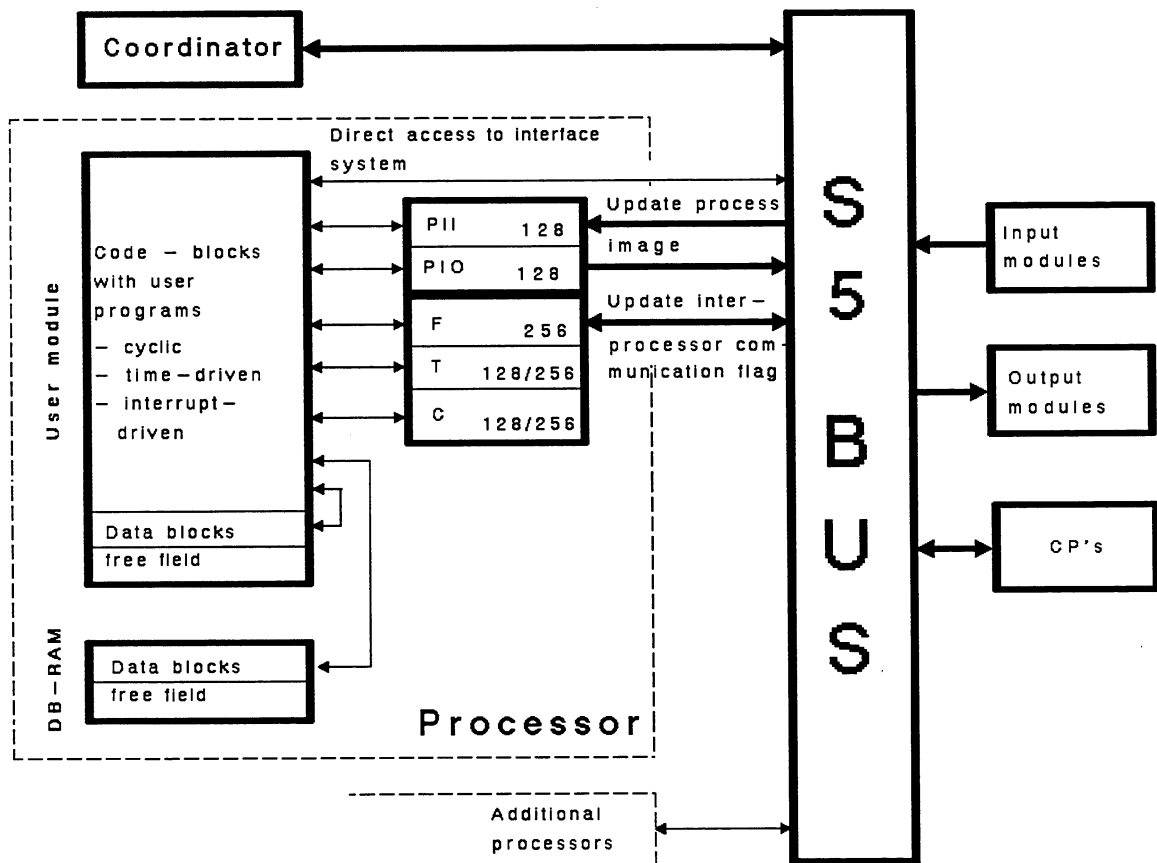
The program found in each processor is divided into two parts, an user program and a system program.

STEP 5 user programs for the S5 135U are written in the programming language STEP 5, which is especially designed for PC's (exception: M processor). The user program is modular and consists of at least one program module (block). A distinction must be made between two fundamental block types:

- a) Code blocks: blocks that contain STEP 5 commands.
- b) Data blocks: blocks containing the constants and variables for the STEP 5 program.

The user has no access to the system program. This program supports all the typical functions of a programmable controller. It includes:

- updating the process images (input, output, interprocessor communication flag)
- updating the timer locations
- calling the cyclic time-driven and interrupt-driven programs.



Block diagram of an S5 195 U processor (multiprocessor operation)

Internal design of a processor

A processor's memory is divided into several areas, the most important being:

- user memory (max. 32K words)

The user memory is located on a plug-in RAM or EPROM sub-module and contains code and data blocks.

- data block - RAM (= DB-RAM, max. 23.375 Kwords)

The DB-RAM is a memory area for data blocks. Data blocks whose contents the application program has to change, have to be copied from the EPROM to the DB-RAM.

- flag address area F (256 bytes)

This memory area can be accessed quickly by the user program. It should ideally be used for data, which are required frequently. The following types of data can be accessed: single bits, bytes, words, and double words. Single flag bytes can be used as interprocessor communication flags for data exchange between processors. Interprocessor communication flags are updated by the system program at the end of a cycle by way of a latch in the coordinator.

- process image of the inputs and outputs PII/PIO (each 128 bytes)

The user program can access the process image in the same way as the interprocessor communication flags. The process image is also updated by the system program at the end of the cycle.

- I/O area (512 bytes)

The user program can bypass the process image and access the peripheral units directly via the S5 bus. Possible types of data are: bytes and words.

- timers T (128 timer locations for the S and R processor, 256 timer locations for the CPU 928)

Timer locations are set at a value between 10 ms and 9990 s by the user program and are counted down in intervals of 10 ms by the system program.

- counter C (128 counters for the S and R processor, 256 counters for the CPU 928)

Counter locations are set at a starting value (max. 999) by the user program and decremented.

STEP 5 commands have access to the following operand areas:

- flag area
- process image of the inputs and outputs
- I/O area
- timers
- counters
- current data block

In order to access the above operand areas, STEP 5 commands employ 2 different mechanisms:

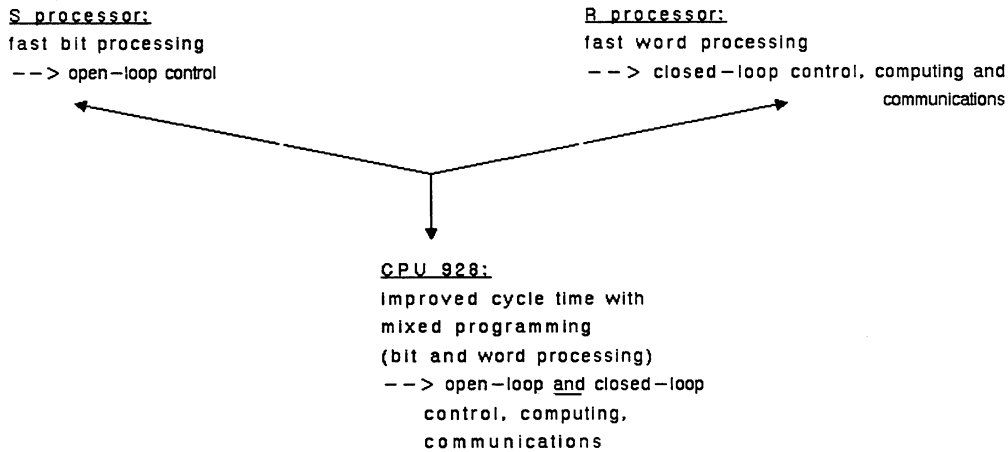
- Most of the STEP 5 commands address a storage location relative to the beginning of an operand area. As long as only these commands are being used, the program is separated from the operand areas and cannot overwrite itself if an error occurs.
- A few STEP 5 commands work with absolute addressing. These commands allow access to the entire memory area.

In comparison with other operand areas, the current data block does not have any fixed length or start address. It is the data block whose length and start address are entered in special registers (see below). The user program can only access the current data block unless commands for absolute addressing are used. Possible types of data are: single bits, bytes, words, and double words. Access to the current data block is slower than access to the flag area.

Besides those memory areas mentioned above, the processor has the following registers:

- 4 accumulators (32 bits) that serve as multipurpose registers, e.g. as auxiliary registers for memory-to-memory transfer, or as registers for operand and arithmetic results.
- 1 instruction counter (STEP address counter, SAC) containing the address of the next command.
- 1 block stack pointer (BSP) that organizes block stack input.
- 1 DBA register (DBA = data block start address) containing the start address of the current data block.
- 1 DBL register (DBL = data block length) containing the number of data words in the current data block.
- 1 condition code register.
- 1 BR register (BR = base address register) used for absolute addressing.

1.1 New Features and Functions of the CPU 928 (for Users of S and R Processors)



The CPU 928 is a 40 mm wide module that takes up 2 slots in the central controller 135U. It combines the advantages of the S processor (fast bit processing, designed for open-loop control tasks) and the R processor (fast byte processing, designed for closed-loop control tasks). More-over, the CPU 928 is particularly suitable not only for operation and for observation, but also for monitoring, signalling and for communication in multiprocessor operations. The CPU 928 is therefore a universal processor for handling a wide range of automation tasks.

If you are already familiar with the S or R processor in S5 135U, then the new features and functions of CPU 928 described in the following chapters should be of particular interest: *information given in italics applies to version 3UA12 only!*):

Chapter 3.1.1: Program organization

Maximum block nesting depth of the CPU 928 has been increased to '62' (CPU 928-3UA11: '30', R processor: '20').

Maximum cycle time permitted is now 6000 ms (CPU 928-3UA11 and R processor: 4000 ms).

Chapter 3.3: Supplementary operations

The STEP 5 operation set has been extended to include the following new commands:

- commands for adding and subtracting 32-bit fixed-point numbers: +D, -D, ADD DF (system operations) ¹⁾

- ¹⁾ Programming of these commands depends on the type of your PG as well as on the release of your PG system software
- commands for loading and transferring a word into the RJ or RT area: L RJ, T RJ, L RT, T RT (supplementary operations)

Chapter 4.4.2: Time interrupts

At present up to 9 time-driven programs may be executed. The individual programs are contained in the organization blocks OB 10 to OB 18. Each OB is called in another time base: OB 10, for instance, is processed every 10 ms, OB 15 every 500 ms. Time interrupt OBs with a shorter time base have a higher priority than time interrupts with a longer time base and, if required, are nested into the latter.

Chapter 5.6: Errors in START and RUN

Error identifiers in accumulators (accu) 1 and 2 have been added to.

Chapter 6: Special integrated functions

CPU 928 has new special functions available. These are:

- OB 110 : Access to the condition-code byte
- OB 111 : Clear accus 1, 2, 3 and 4
- OB 112 : Roll up accu
- OB 113 : Roll down accu
- OB 120 : Switch on/off "Inhibit all interrupts"
- OB 121 : Switch on/off "Inhibit individual time interrupts"
- OB 122 : Switch on/off "Delay all interrupts"
- OB 123 : Switch on/off "Delay individual time interrupts"
- OB 160 to OB 163 : Counter loop
- OB 170 : Read block stack (BSTACK)
- OB 180 : Random data block access
- OB 181 : Test data blocks
- OB 190 and OB 192: Transfer flag to data blocks
- OB 191 and OB 193: Transfer data blocks to flag area
- OB 228 : Read status information from a program level

Chapter 7: Data block DX 0

With the CPU 928 you can activate 256 counter and 256 timer locations (R processor: 128 counter, 128 timer locations).

The parameters for interrupt-driven program processing have been added to.

For floating-point arithmetics, DX 0 allows you to set whether the processor is to calculate with a 16-bit or a 24-bit mantissa.

Chapter 8.1: Memory address space distribution in the CPU 928

The data block RAM of the CPU 928 has been extended to 23.375 K words (R processor: 11.125 words). This allows you to work with more data blocks than before.

Furthermore, there are two new operand areas available, each 256 words long: the RJ and RT area. For access to these areas there are new STEP 5 commands.

Chapter 9: Memory access via absolute addresses

Those areas in the memory address space whose addressing by the STEP 5 commands LIR, TIR, TNB, and TNW is useful, have been enlarged and contain less gaps in the CPU 928.

Chapter 9.3: Operations using the BR register

To make absolute addressing easier the BR register (BR = base address register) has been introduced.

There are

- new commands used to load or modify the BR register (refer to 9.3.1), 1)*
- new commands to shift the contents of individual registers (refer to 9.3.2), 1)*
- new commands used to access local or global memory areas (refer to 9.3.3 and 9.3.4), 1)*
- new commands to access the page frame memory. 1)*

APPENDIX A and List of operations:

With version 3UA12 (12 MHz), the command execution and system run times of CPU 928 have been improved approximately by one third, compared with version 3UA11.

APPENDIX G:

The identifications for the individual program processing levels have been changed or added to.

1) Programming of these commands depends on the type of your PG as well as on the release of your PG system software

2 User Program

2.1 Programming Language STEP 5

Using the STEP 5 programming language you convert automation tasks into programs which run on SIMATIC S5 programmable controllers. Simple binary functions as well as complex digital functions and basic arithmetic operations can be programmed with STEP 5.

The **full range** of in the programming language STEP 5 is divided into three main groups:

Basic Operations:

- can be used in all blocks
- methods of representation:
 - ladder diagram (LAD)
 - control system flowchart (CSF)
 - statement list (STL)

Supplementary Operations:

- only for use in function blocks
- statement list (STL) only method of representation

System Operations:

- These belong to the supplementary operations group
- Can only be used in function blocks
- The statement list (STL) is the only method of representation
- Only for users with excellent knowledge of the system!

2.1.1 Methods of Representation: LAD, CSF, and STL

When programming with STEP 5, you can choose between the three methods of representation ladder diagram (LAD), control system flowchart (CSF), and statement list (STL) so that the programming method can be adapted to particular application in hand.

The machine code generated by the programmer (PG's) is identical in all three representation methods.

By keeping to certain rules while programming with STEP 5, the PG can translate your user program from one method of representation into any other method!

It is possible to represent your STEP 5 program graphically with the ladder diagram and control system flowchart while the statement list lists each STEP 5 command.

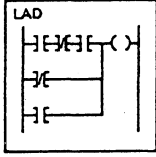
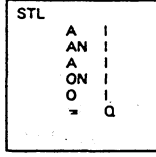
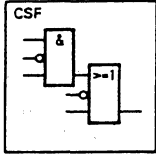
Ladder diagram	Statement list	Control system flowchart
Programming with graphic symbols as in circuit diagram to DIN 19239 	Programming with mnemonics of the function designation to DIN 19239 	Programming with graphic symbols to IEC 117-15 DIN 40700 DIN 40719 DIN 19239 

Fig 2.1: Programming language STEP 5 - methods of representation

The programming language GRAPH 5 is used for the graphic representation of sequence control systems. In the hierarchic order, it is above the methods of representation LAD, CSF and STL. The program written in GRAPH 5 with graphic representation is automatically converted by the programmer into a STEP 5 program.

2.1.2 Structured Programming

The total program of a processor consists of:

System Program: This contains the entire set of instructions and declarations necessary for implementing internal restart functions (e.g. saving data in case of power supply failure, prompting user reactions to interrupts etc.).

It is stored in EPROM's (erasable programmable read-only memory) and is a fixed component of the processor. You as user, *do not have access* to the system program.

User Program: This contains the entire set of instructions and declarations programmed by the user for signal processing, by means of which the plant (process) is controlled. The user program can be subdivided into blocks.

The entire user program can be divided into separate, self-contained program sections (blocks). The configuration of these blocks in the user program therefore makes the most important program structures clear or reflects the relationship between plant (process) and program.

This '**structured programming**' gives you the following advantages:

- clear and simple programming even with long programs
- standardizing program sections possible
- simple program organization
- program easy to modify
- simple program testing section by section
- simple commissioning

What is a block?

A block is a separate part of the user program distinguished by its function, structure, or application. A distinction must be made between blocks that contain instructions for signal processing (organization blocks, program blocks, function blocks, sequence blocks), and blocks that contain data (data blocks).

2.1.3 STEP 5 Operations

A STEP 5 operation is the smallest independent unit of a user program. It is a work instruction for the processor and is made up of an operation and an operand.

Example:

:O	F	54.1
/		\
Operation		Operand
(What to do?)		(With what?)

You can enter (via the assignment list) the operand either as **absolute** or **symbolic**.

Example of an absolute representation: :A I 1.4
Example of a symbolic representation: :A -Motor1

More information on absolute and symbolic programming can be found in the instruction manual "Programmable Controller PG 685", order number C79000-B8576-C373-xx.

STEP 5 operations allow you to:

- perform logic operations on binary data
- load, save, and transfer data
- compare values with each other and process them mathematically
- set time and count values
- convert numeric representations
- structure the user program
- influence program processing, etc.

Most STEP 5 operations use two registers either as the source or the destination for operands and only the destination for an operation result: accumulator 1 (accu 1) and accumulator 2 (accu 2). Each accumulator is 32 bits (1 double word) wide.

A detailed description of STEP 5's entire operation set can be found in Chapter 3.2. Here you will find programming examples of each STEP 5 command.

In Appendix C there is a list of all available STEP 5 operations and valid parameters.

2.1.4 Numeric Representation

Before the processor can change, compare or perform logic operations on numerical values, you must load these values in binary-coded form in the accumulators.

Depending on the type of operation to be done, STEP 5 allows the following numeric representations:

- binary numbers: a) 16-bit fixed-point numbers
 b) 32-bit fixed-point numbers
 c) floating-point numbers
- decimal numbers: d) BCD-coded numbers

The data format (e.g. KF for fixed point) in which values are to be entered or displayed is set on the PG. The PG then converts the internal method of representation into the one required by the user.

Using 16-bit fixed-point numbers and floating-point numbers you can carry out all arithmetic operations such as compare, add, subtract, multiply, and divide.

BCD-coded numbers are only used for input and output. You cannot perform any direct arithmetic operations with these codes.

Comparisons are possible with 32-bit fixed-point numbers. These are also necessary for converting BCD-coded numbers into floating-point numbers as an intermediate step. The new commands +D and -D can now also be used for adding and subtracting.

The STEP 5 language contains **conversion operations**, which let you convert numbers to the most important numeric representations directly.

16-bit and 32-bit fixed-point numbers

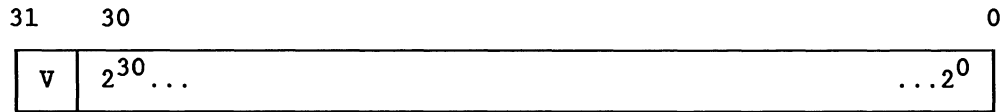
Fixed-point numbers are whole binary numbers with sign.

They are 16 bits (= 1 word), or 32 bits (= 2 words) long, where bit number 15 or bit number 31 is the sign: "0" = positive number, "1" = negative number.

Negative numbers are represented in their two's complement.

B8576633-01

32-bit fixed-point number:



Entry of the data format as 16-bit fixed-point on the PG: KF

Entry of the data format as 32-bit fixed-point: only KH

Allowed range of numbers: -32768 to +32767 (16 bits)

-2147483648 to +2147483647 (32 bits)

(To convert a 16-bit fixed-point number into a 32-bit fixed-point number see Chapter 3.2.2.)

Fixed-point numbers are used to solve simple arithmetic problems and to compare numerical values. Please note that, since fixed-point numbers are always integers, there can be no remainder following division.

Floating-point numbers

Floating-point numbers are positive and negative fractions. They always occupy a double word (32 bits), and are represented as exponentials. The mantissa is 24 bits in length, the exponent is 8 bits long.

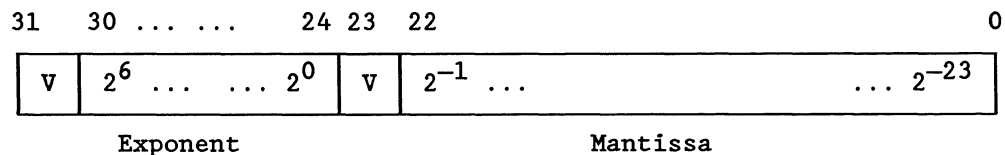
The exponent determines the magnitude of a floating-point number. The sign of the exponent will tell you whether or not the floating-point number is greater or less than 0.1.

The mantissa determines the accuracy of a floating-point number:

- 24-bit mantissa accuracy is: $2^{-24} = 0.000000059604$
(equivalent to 7 decimal places)
- 16-bit mantissa accuracy is: $2^{-16} = 0.000015258$
(equivalent to 4 decimal places)

If the mantissa sign is "0", then the number is positive; if it is "1", then the number is negative in two's complement representation.

Floating-point number:



The CPU 928 only computes with a 16 bit wide mantissa (bit 8 to 23) when adding, subtracting, multiplying, and dividing. Low order bits 0 to 7 (to the right) always have the value '0'!

If a higher accuracy is required for floating-point computing (accepting a small increase in runtime), you may select the following setting in DX 0: "Floating-point arithmetic with 24-bit mantissa" (see Chapter 7).

Entry of data format as floating-point on the PG: KG

Allowed range of numbers: $\pm 0.1469368 \times 10^{-38}$ to $\pm 0.1701412 \times 10^{39}$

Entering floating-point numbers Z with the PG:

$$Z = 12.34567$$

L KG + 1234567 + 02

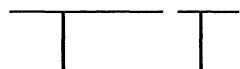


Mantissa Exponent (base 10) with sign

$$Z = +0.1234567 \times 10^{+2} = 12.34567$$

$$Z = -0.005$$

L KG - 50000000 - 02



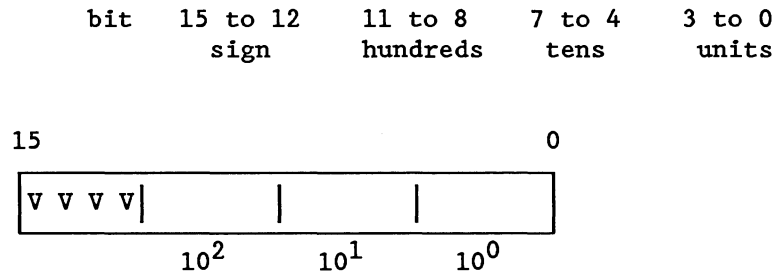
Mantissa Exponent (base 10) with sign

$$Z = -0.5 \times 10^{-2} = -0.005$$

Use floating-point numbers for solving more complex mathematical problems, particularly for multiplication and division and when you are working with very large or very small numbers!

BCD-coded numbers

Decimal numbers are represented as BCD-numbers. A three digit number with sign occupies 16 bits (1 word) in the accumulator:



The single digits are positive 4 bit binary numbers between 0000 and 1001 (0 and 9).

Allowed range of values: -999 to +999

The four left hand bits are reserved for the sign.
 sign for a positive number: "0000"
 sign for a negative number: "1111"

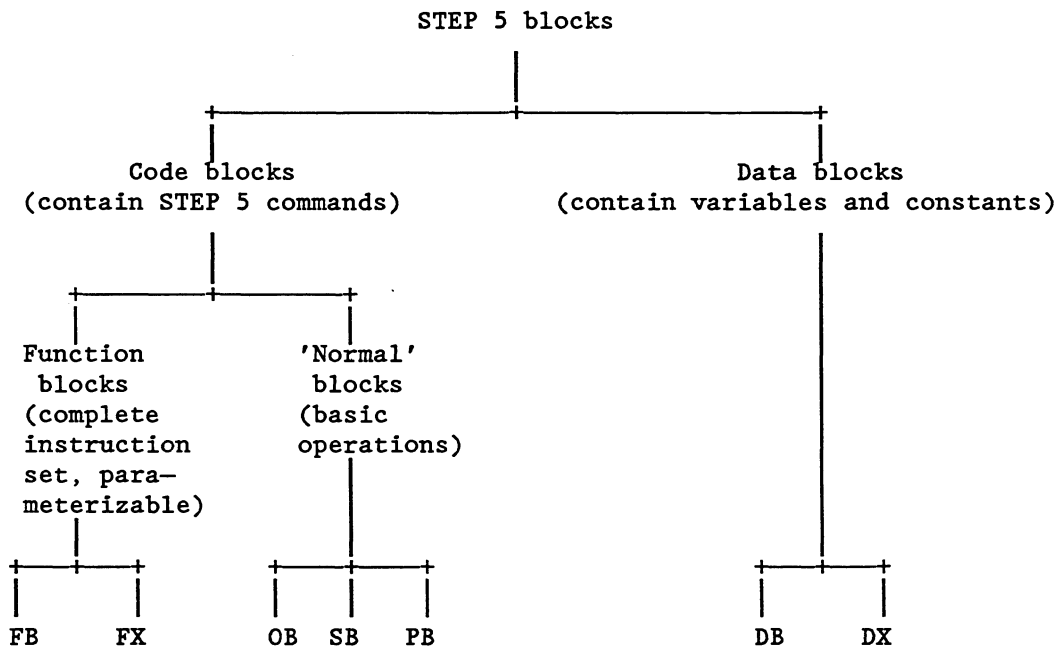
2.1.5 STEP 5 Blocks

A block is a separate part of the user program distinguished by its function, structure, or application.

It can be identified by its

- block type (OB, PB, SB, FB, FX, DB, DX) and
- block number (number between 0 and 255).

The STEP 5 programming language differentiates between the following types of blocks:



- organization blocks (OB)

These are the interface between the system program and the user program, and can be divided into two groups:

OB 1 to 39 are called up by the system program and control program processing, the startup routine of the processor, and reaction to faults. The user must program these OB's.

OB 40 to 255 contain special system program functions. The user calls these up as required.

- program blocks (PB)

These are used to structure the user program and contain structured subroutines that are process or function-oriented. Program blocks usually contain the greater part of the user program.

- function blocks (FB/FX)

These are used to program frequently recurring or complex functions (e.g. digital functions, sequential control, closed loop control, signalling functions). A function block can be called by primary blocks many times, each time with new operands assigned.

- sequence blocks (SB)

These are special program blocks that are used for step-by-step processing of sequence cascades.

- data blocks (DB/DX)

These blocks contain (fixed or variable) data with which the user program works. This type of block does not contain any STEP 5 instructions and its function is totally different from that of other blocks.

What does a block consist of?

- All blocks comprise
- a block header and
 - a block body

The **block header** always has a length of 5 data words. The programmer automatically enters the following in the header:

- the block start-identifier
- the block type (OB, FB ...)
- the block number
- the PG identifier
- the library number
- the block length (including block header)

Block header in program memory :

Start		Identifier
Block type		Block number
PG identifier		L i b r a r y
n u m b e r		
Block length incl. header (words)		

15

0

You can find a precise identification of block type and block number in Subsection 8.2.1.

Depending on the block type, the **block body** contains

- STEP5 commands (for OBs, PBs, SBs, FBs, FXs),
- variable or constant data (for DBs, DXs),
- list of formal operands (for FBs, FXs).

For the block types DB, DX, FB, and FX, the programmable controller produces a **block pre-header** (DH, DXH, FH, FXH) which contains information about data format (with DB and DX) or jump labels (with FB and FX). Only the programmable controller can evaluate this information, which is why block pre-headers are not transferred to the PC memory. As user, you have no direct influence on the contents of a block pre-header.

A STEP5 block can occupy max. 4096 words in the program memory of the processor. Do not forget the memory capacity of your PG when entering or transferring blocks with the programmer.

Of the possible block types, the following are available for programming:

OB 1	to	39
FB 0	to	255
FX 0	to	255
PB 0	to	255
SB 0	to	255
DB 3	to	255
DX 1	to	255

The data blocks DB 1, DB 2, and DX 0 contain parameters. They are reserved for specific functions and can therefore not be used for other purposes.

All programmed blocks are stored in an arbitrary order by the PG in the program memory (figure). This is a plug-in RAM or EPROM on the processor. The start addresses of the blocks stored are deposited in data block DB 0.

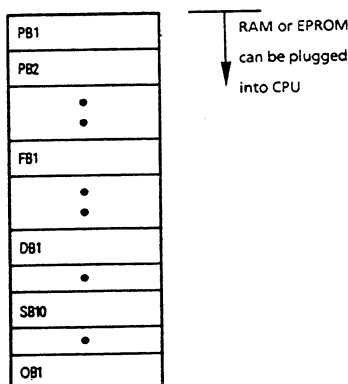


Fig.2-2: Block storage in the program memory

B8576633-01

When correcting a block, the 'old' block is declared invalid and the corrected or new block is entered in the memory. The same procedure applies to block deletion; instead of actually being deleted, the blocks are made invalid.

IMPORTANT!

Deleted and corrected blocks still occupy memory space!

By using the online function 'COMPRESS memory', you gain space in the memory for new blocks. This function erases all invalid blocks in the memory and moves the valid ones together (see Chapter 11.6).

2.2 Organization, Program and Sequence Blocks

These three types of block do not differ with respect to programming and calling. All three types can be programmed in LAD, CSF and STL.

2.2.1 Programming

When programming organization, program and sequence blocks, proceed as follows:

- First, specify the type of block and then the number of the block to be programmed.

The following numbers are available:

Program blocks	0 through 255
Sequence blocks	0 through 255
Organization blocks	1 through 39

- Enter your user program in STEP5.

IMPORTANT!

When programming PB's, SB's and OB's only STEP 5-basic operations are allowed!

- Terminate program input by entering "BE" (block end).

IMPORTANT!

A STEP5 block should always contain a complete program. Logic operations within a block must be complete.

Up to approx. 4000 words are possible within one block (depending on the type of programmable controller used).

The block header which is automatically generated by the PG requires 5 words in the program memory.

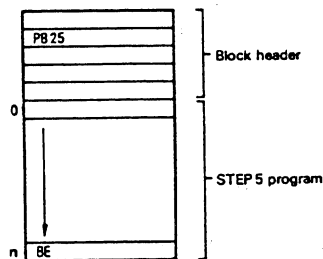


Fig. 2-3: Structure of an organization, program and sequence block

2.2.2 Calls

Blocks have to be enabled for processing. This is achieved by block calls (see fig.).

Programming of these block calls is possible within an organization, program, function or sequence block. They are comparable to the jumps to a subroutine. Each jump causes a block change.

It is possible to execute jumps either as conditional or unconditional jumps:

* Unconditional call: JU xx

The block called is processed *independent* of the previous result of logic operation (=RLO).

The RLO is the signal state within the processor which is used for further binary signal processing. It is possible to e.g. perform logic operations on the RLO and the signal state of the operands or to carry out operations dependent on the previous RLO: The "unconditional operations" are *always* executed, the "conditional operations" *only* if the RLO is = 1.

The jump instruction JU belongs to the category of unconditional operations. It has no influence on the RLO which is transferred to the new block if a jump is carried out. There, an evaluation of the RLO is possible, however, a further logic operation with it is not.

* Conditional call: JC xx

The jump instruction JC belongs to the category of conditional operations, i.e. the block called is *only* processed if the previous result of logic operation (RLO) is = 1. In the case of the RLO being = 0, the jump instruction is not executed. However, the RLO is set to "1"!

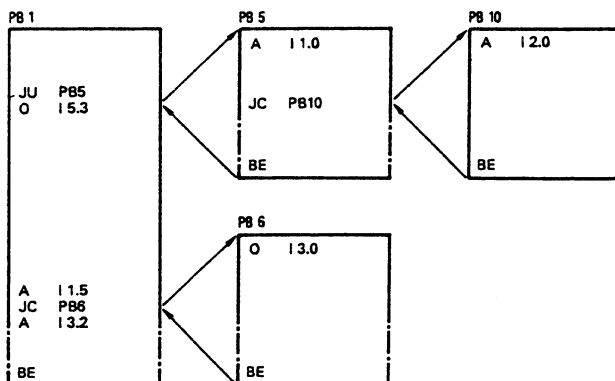


Fig. 2-4: Block calls that enable processing of a program block.

After the statement BE has been entered, a jump is made back to the block in which the block call had been programmed. Processing of the program is continued at the first STEP5 statement after the block call.

The block end statement BE is processed independent of the result of logic operation. After BE the result of logic operation can no longer be logically operated on. However, the result of logic operation/computed result present immediately before the execution of the BE statement is transferred to the block calling and an evaluation is possible there. When returning from the block called, the contents of accumulators 1, 2, 3 and 4, the condition codes CC 0 and CC 1 as well as the result of logic operation RLO are not altered.

2.2.3 Special Organization Blocks

The organization blocks are the interface between the system program and the user program. The organization blocks OB 1 through 39 are part of the user program which you program just like program, function, or sequence blocks. You thus have the possibility of influencing the reaction of the processor, during the start-up, program processing and in case of errors, by means of programming these OB's. The organization blocks become effective the moment they are loaded into the memory of the programmable controller. This is also possible while the system is running.

The important point to remember is that these OB's are called by the system program as a reaction to certain events.

Organization block	Function and criteria for block call
OB 1	organization of cyclic program processing called at the end of a start-up mode
OB 2	organization of interrupt-driven program processing called by signal via the S5 bus (process interrupt)
OB 10 - 18	organization of time-driven program processing (time interrupts)
OB 10	called every 10 msec
OB 11	called every 20 msec
OB 12	called every 50 msec
OB 13	called every 100 msec
OB 14	called every 200 msec
OB 15	called every 500 msec
OB 16	called every 1 sec
OB 17	called every 2 sec
OB 18	called every 5 sec
OB 20 - 22	organization of the start-up routine
OB 20	called with request "cold restart"
OB 21	called with request "manual warm restart"
OB 22	called after return of power ("auto. warm restart")
OB 19, 23-24	reaction to the following equipment faults or program errors: ¹⁾
OB 19	execution time error: call of an unprogrammed block
OB 23	timeout in the user program (for direct access to I/O modules or other S5 bus addresses)
OB 24	timeout when updating process image and transferring interprocessor communication flags
OB 25	addressing error
OB 26	cycle time exceeded
OB 27	command code error: substitution error

¹⁾ If the OB is not programmed, in the case of an error the processor will go over to the stop state. EXCEPTION: if OB 23 and 24 (acknowledgement delay) do not exist, there will be no reaction!

Organization block	Function and criteria for block call
OB 28	stop caused by programmer function/stop switch/S5 bus ¹⁾
OB 29	command code error: operation code illegal
OB 30	command code error: parameter illegal
OB 31	other execution time errors
OB 32	execution time error: transfer error in the data block
OB 33	time interrupts
OB 34	error during controller processing

¹⁾ OB 28 is called before a transition to the stop state. The processor will stop irrespective of whether and how OB 28 is programmed.

After the system program has called the respective organization block, the user program contained in it will be processed. Usually, the processor will then return to the program which has been interrupted by the error organization block (exception: OB 28). For the behaviour without error OB, refer to Chapter 5.4.

It is possible to have the user program call these organization blocks for test purposes (JU/JC OBxxx). However, it is not possible to cause a processor stop by calling OB 28 or to initiate an automatic warm restart by calling OB 22!

IMPORTANT!

The special organization blocks are programmed by the user and called automatically by the system program!

2.2.4 Organization Blocks with Special Functions

The following organization blocks contain special functions of the system program. They cannot be programmed by the user, only called (this applies to all OB's with numbers between 40 and 255!). They do not include a STEP5-program. Special function OBs may be called from any code block.

Overview 2-5: Organization blocks with special functions in the CPU 928

OB 110	access to the condition-code byte
OB 111	clear accus 1, 2, 3 and 4
OB 112	roll up accu
OB 113	roll down accu
OB 120	switch on/off "Inhibit all interrupts"
OB 121	switch on/off "Inhibit individual time interrupts"
OB 122	switch on/off "Delay all interrupts"
OB 123	switch on/off "Delay individual time interrupts"
OB 160 - 163	counter loop
OB 170	read block stack (BSTACK)
OB 180	variable data block access
OB 181	test data blocks
OB 190, 192	transfer flags to data blocks
OB 191, 193	transfer data fields to flag area
OB 200, 202 - 205	multiprocessor communication
OB 216 - 218	access to pages
OB 220	convert accumulator 1 from 16- to 32-bit fixed-point number by means of sign extension
OB 221	set and trigger new cycle time
OB 222	retrigger cycle time
OB 223	stop if start-up mode for multiprocessor operation is not uniform
OB 224	block transfer of interprocessor communication flags in multiprocessor operation
OB 226	read contents of a storage location of the system program in bytes
OB 227	read check sum of system program memory
OB 228	read status information of a program level
OB 230 - 237	functions for standard function blocks
OB 240	initialize shift register
OB 241	call shift register
OB 242	clear shift register
OB 250	initialize PID-controller
OB 251	process PID-controller
OB 254, 255	transfer data blocks to DB-RAM

For a detailed description of special functions refer to Chapter 6.

2.3 Function Blocks

Function blocks (FB/FX) are parts of the user program just as e.g. program blocks. The structure of FX-function blocks is similar to that of FB-function blocks and they are programmed in the same manner.

Frequently repeated or very complex functions are implemented by means of these function blocks.

In comparison with the organization, program and sequence blocks there are four important **differences**:

- **Parameter** assignment is possible for function blocks, i.e.: The formal operands of a function block can be substituted by other actual operands whenever they are called. This means that function blocks created for general use are very versatile.
- Programming of function blocks is possible using the complete operation range of the STEP5 programming language. In addition to the basic operations used in all types of blocks there are also **supplementary operations** as well as **system operations** available.

IMPORTANT!

Programming of supplementary operations and system operations is only possible in function blocks.

- It is **only** possible to program and document function blocks as a **statement list (STL)**.

However, *calling* of function blocks is also possible in control system flowchart or ladder diagram representation methods and is represented graphically as a box.

- Designation of function blocks may be carried out using **names** of up to 8 characters.

Within the user program, each of the function blocks represents a complex and self-contained function. The user can

- purchase function blocks directly from SIEMENS as a software product (standard function blocks on a mini-diskette); these standard function blocks allow fast and reliable generation of user programs for signalling, controlling and logging;

or

- program them himself.

2.3.1 Structure of Function Blocks

The **block header** (5 words) of a function block does not differ from that of the other STEP5 blocks.

The structure of its **body**, however, is fundamentally different from that of the other types of blocks. It contains the actual program of the function block. The function to be executed is written in the programming language STEP5 in the form of a statement list. The function block requires additional memory space for the data specifying its name and the list of formal operands *between* the header and the actual STEP5 user program. Since this list contains no instructions for the processor it will be skipped by means of an unconditional jump automatically generated by the PG. This jump instruction is not displayed on the PG!

Operands may be entered into a function block either in an absolute (e.g. F 2.5) or a symbolic form (e.g. -MOTOR1). Before doing this, you will have to enter the assignment of the symbolic operands into an *assignment list*.

If the function block is called, it is only the block body that is processed.

A function block in the PC memory has the following structure:

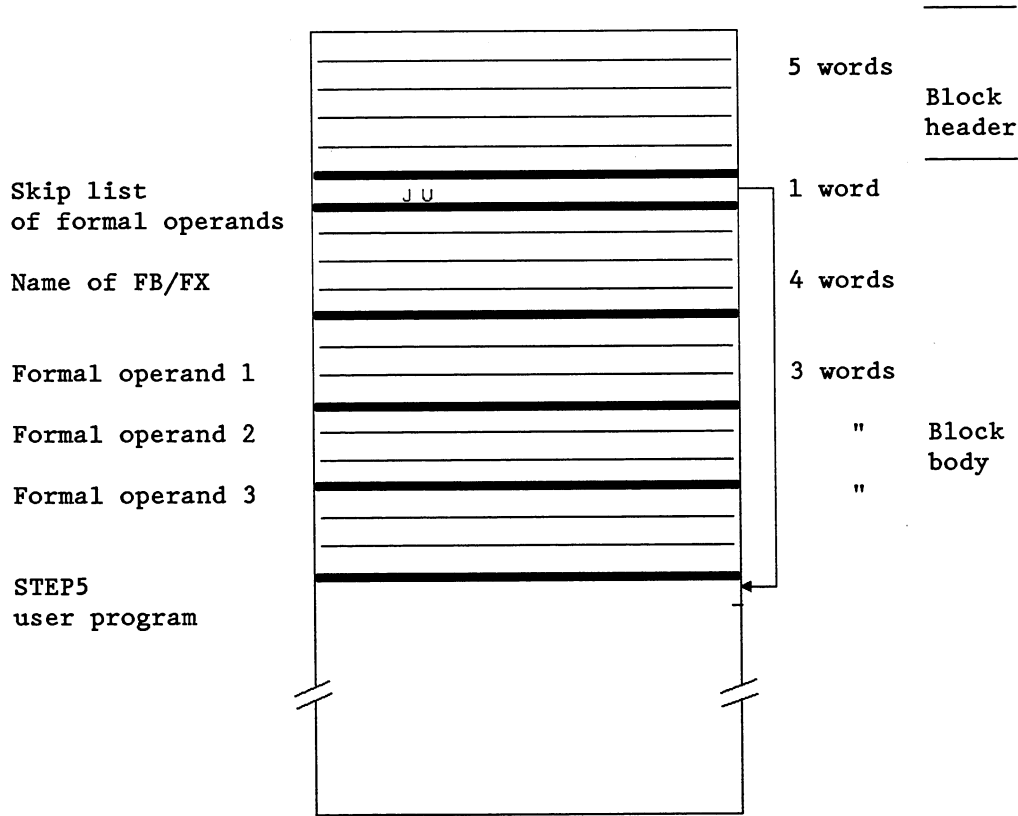


Fig. 2-6: Structure of a function block (FB/FX)

All data which the programmer requires to generate a graphic representation of the function block when called and all data required for testing the operands during parameter assignment and programming are therefore in the memory. An incorrect input will not be accepted by the programmer.

IMPORTANT!

When dealing with function blocks, make sure to differentiate between

- a) programming an FB/FX and
- b) calling an FB/FX and then assigning parameters.

When programming, you determine the function of a block. The operands entered are formal operands which function as a fill-in.

When calling a block via a primary block (OB, PB, SB, FB, FX) the formal operands are replaced by actual operands: parameters are assigned to the function block.

The following pages will help to illustrate these points.

2.3.2 Programming Function Blocks

To enter a function block at the PG proceed as follows:

- Input the **number** of the function block.

IMPORTANT!

It is advisable to assign numbers to the user function blocks in descending order starting with 255 to avoid interfering with the standard function blocks with numbers from FB 1 through FB 199.

- Entering a number from 0 through 99 999 as a **library number** is possible. This number will be assigned to the function block, independent of its block number or name.

It is advisable to assign a library number only once in order to ensure that function blocks are identified uniquely.

- Input the **name** of the function block. A maximum of 8 characters is permitted.
- Input the formal operands used in the block (40 formal operands max.)

Specify the following for each of the formal operands:

1. the name of the block parameter,
2. the class of block parameter,
3. the type of block parameter.

Up to 4 characters may be used for the **name**.

The programmer gives you the following choice for the input of the **class of block parameter**:

I = input parameter
Q = output parameter
D = data
B = command
T = timer
C = counter

The parameters marked I, D, B, T and C are shown on the left of the function symbol, whereas the parameters marked Q are on the right.

For the I, D and Q classes of parameter you must also specify the **type of parameter**:

BI/BY/W/D for parameter class I, Q
KM/KH/KY/KS/KF/KT/KC/KG for parameter class D

The parameter type indicates whether the I and Q parameters are bits, bytes, words or double words and which data format (e.g. bit pattern or hexadecimal pattern) is valid for D parameters.

Class of parameter	Type of parameter	Permissible actual operands
I, Q	BI for an operand with bit address	I n.m inputs Q n.m outputs F n.m flags
	BY for an operand with byte address	IB n input bytes QB n output bytes FY n flag bytes DL n left-hand data byte DR n right-hand data byte PY n peripheral bytes EB n peripheral bytes from the extended interface system
	W for an operand with word address	IW n input words QW n output words FW n flag words DW n data words PW n peripheral words EW n peripheral words from the extended interface system
	D for an operand with double word address	ID n input double words QD n output double words FD n flag double words DD n data double words
D	KM for a bit pattern (16 positions)	constants
	KY for two absolute values in bytes each from 0 through 255	
	KH for a hexadecimal pattern, max. 4 positions	
	KS for 2 alphanumeric characters	
	KT for a time value (BCD-coded) with time base .0 to .3 and time value 0 to 999	

Class of parameter	Type of parameter	Permissible actual operands
D	KC for a count value (BCD-coded) 0 to 999 KF for a fixed-point number -32768 to +32767 KG for a floating- point number	
B	No type specification permissible	DB n data blocks; command C DB n is executed. FB n function blocks (permissible only without parameters) are called unconditionally (JU ..n). PB n program blocks are called unconditionally (JU ..n). SB n sequence blocks are called unconditionally (JU ..n).
T	No type specification permissible	T 0 to 255 time ¹⁾
C	No type specification permissible	C 0 to 255 counter ¹⁾

1) The time value or count value must be assigned as a formal operand or is to be programmed as a constant in the function block.

- Then input your STEP5 program as a statement list.

The formal operands are marked by an equality sign placed in front of the operand (e.g. A =X1). These operands can be called several times at different locations in the function block.

IMPORTANT!

If the sequence or the number of the formal operands in the list of formal operands is altered then the substitution commands in the STEP5 program of the function block as well as the list of block parameters in the block initiating the call will have to be corrected accordingly!

IMPORTANT!

Make sure that you always program and alter the function blocks on either a floppy disk or a Winchester and then transfer them to the programmable controller.

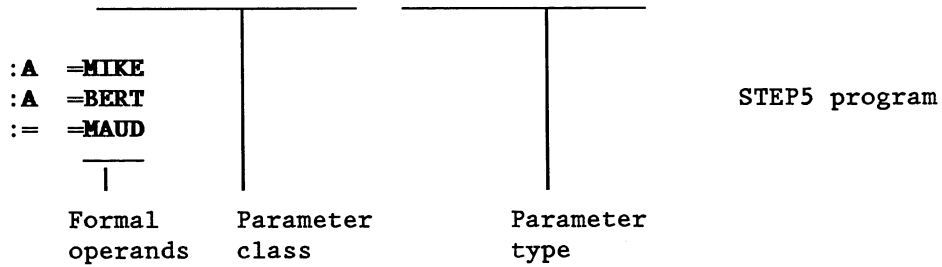
- Terminate the program input with 'BE' (block end).

Example 2-7: Programming a function block

FB 202

NAME: **EXAMPLE**

DECL.:	MIKE	I/Q/D/B/T/C:	I	BI/BY/W/D:	BI	
DECL.:	BERT	I/Q/D/B/T/C:	I	BI/BY/W/D:	BI	List of
DECL.:	MAUD	I/Q/D/B/T/C:	Q	BI/BY/W/D:	BI	formal operands



2.3.3 Calling and Assigning Parameters to Function Blocks

In the STEP5 user program each function blocks can be called wherever and as often as desired. The STEP 5 program is always written as a statement list, however, the function block calls can also be in a graphic representation CSF or LAD).

For calling and assigning parameters proceed as follows:

- Input the call statement for the function block in the block which is to initiate the call.

It is possible to program a function block call within an organization, program or sequence block or within another function block.

The call can be either conditional or unconditional:

- * Unconditional call (JU FBn for function blocks or DO FXn for extended function blocks):

The function block called is processed independent of the previous result of logic operation.

- * Conditional call (JC FBn for function blocks or DOC FXn for extended function blocks):

The function block called is processed only if the previous result of logic operation is RLO = 1. If RLO = 0, the jump statement is not executed, however, the RLO is set to 1.

No further logic operation using the RLO is possible after the unconditional and conditional call. However, it is transferred to the function block called when the jump is executed and evaluation is possible there.

After having input the call statement (e.g. JU FB200) the name as well as the list of formal operands of the respective function block will appear automatically:

- Now you assign the actual operand valid for this particular call to the individual formal operand, i.e. you assign parameters for the function block.

These actual operands may differ for the individual calls: e.g. for the first call of the FB200 inputs and outputs, for the second call flags.

Depending on the list of formal operands it is possible to assign a maximum of 40 actual operands for every function block call.

IMPORTANT!

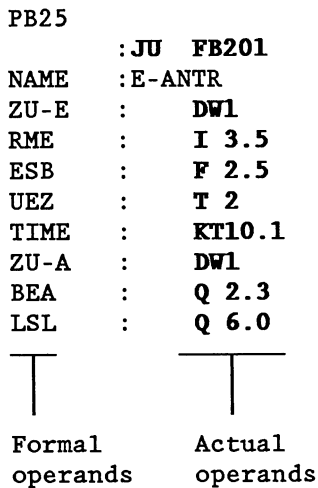
Before calling a function block and assigning parameters it is important to first program this particular function block and copy it onto the program disk and transfer it directly into the program memory of the programmable controller!

After the jump to the function block the actual operands of the block initiating the call are used for the processing of the function block program instead of the formal operands.

This particular feature of the function block (i.e. parameter assignment) allows for a variety of applications in your user program.

Example: Calling a function block and assignment of parameters with the representation methods STL and LAD/CSF in a program block.

- Representation method STL



- Representation method LAD/CSF

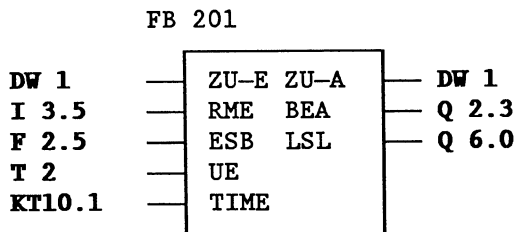


Fig. 2-8: Calling a function block and assignment of parameters

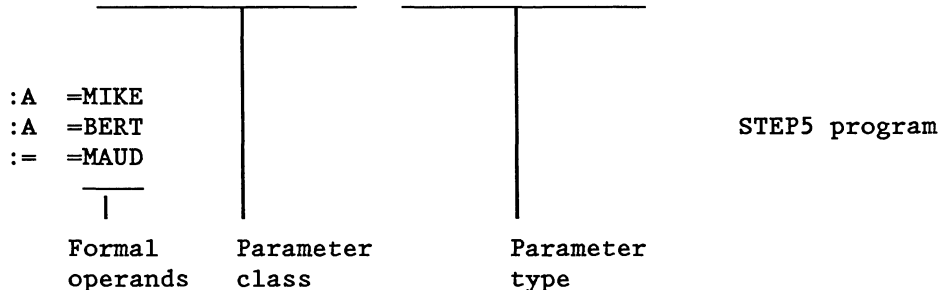
The following (complete) **example** should help to explain the programming, calling and assignment of parameters of a function block. You can easily execute these operations yourself.

The function block FB 202 is programmed:

FB 202

NAME: EXAMPLE

DECL.:MIKE	I/Q/D/B/T/C:	I	BI/BY/W/D:	BI	
DECL.:BERT	I/Q/D/B/T/C:	I	BI/BY/W/D:	BI	List of
DECL.:MAUD	I/Q/D/B/T/C:	Q	BI/BY/W/D:	BI	formal operands



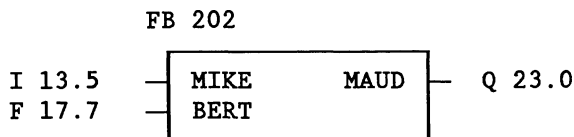
The function block FB 202 is called in the program block PB 25 and parameters are assigned:

- Representation method STL

```
PB25
: JU FB 202
NAME: EXAMPLE
MIKE: I 13.5
BERT: F 17.7
MAUD: Q 23.0
```

Formal operands	Actual operands

- Representation method LAD/GSF



The following program will be executed after the jump to FB 202:

```
: A I 13.5
: A F 17.7
: = Q 23.0
```

2.3.4 Special Function Blocks

- Standard function blocks

In addition to the function blocks which the user himself programs, standard function blocks are also available as an off-the-peg software product. These blocks contain standard functions for general applications (e.g. signalling functions, sequential controls etc.)

The numbers FB 1 through FB 199 are reserved for the standard function blocks.

When purchasing the standard function blocks make sure you follow the special instructions in the system description (areas occupied, conventions etc.).

The standard function blocks for the S5 135U, the execution time, the memory requirements as well as the variables assigned by the user are listed in the ST 57 catalogue "Software for U-Range Programmable Controllers and their Programmers".

Example of a standard function block

Floating point root extractor	RAD:GP	FB 6	for	S5 115U
		FB 6	for	S5 135U
		FB 19	for	S5 150U

The function block RAD:GP extracts the root of a floating-point number (8-bit exponent and 24-bit mantissa), i.e. it finds the square root. The result is also a floating-point number (8-bit exponent and 24-bit mantissa). The least significant bit of the mantissa is not rounded.

If necessary, the function block sets the identifier "radicand negative" for further processing.

Numerical range:

Radicand	-0.1469368 exp. -38 to +0.1701412 exp. +39
root	+0.3833434 exp. -19 to +0.1304384 exp. +20

Function: $Y = \sqrt{A}$
Y = SQRT; A = RAD I

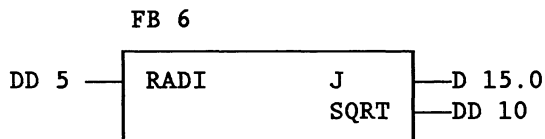
Calling the function block FB 6:

- Representation method STL - Representation method LAD

```

      : C DB 17
      :
      :
      : JU FB 6
NAME  : RAD : GP
RADI  : DD 5
J     : D 15.0
SQRT  : DD 10

```



DD = data double word

The above example shows how the root of a floating-point number, which is written in DD 5 with an 8-bit exponent and a 24-bit mantissa is extracted, the result, which is again a 32-bit floating-point number, is deposited in DD 10. Before this operation takes place, the respective data block has to be opened. The parameter J (parameter class: Q, parameter type: BI) indicates the sign of the radicand: J = 1 for a negative radicand. Flag words assigned: FW 238 through 254.

- Function block FB 0

If the organization block OB 1 is not programmed, then the system program cyclically calls FB 0 instead of OB 1.

IMPORTANT!

For this reason, FB 0 should only be used for programming the cyclic program! (Parameters are not permitted.)

Since the complete operation set of the STEP5 programming language is available in one function block, the programming of FB 0 instead of OB 1 is especially suitable if you want to process a short and time-critical program.

If OB 1 as well as FB 0 are programmed then only the organization block OB 1 is processed cyclically.

2.4 Data Blocks

The fixed and variable data employed by the user program are deposited in the data blocks (DB/DX). No STEP5 operations are processed in data blocks.

Data in a data block could be:

- any bit pattern, e.g. for system status,
- numbers (hexadecimal, binary, decimal) for time values, results of arithmetic operations
- alphanumeric characters, e.g. for messages.

2.4.1 Structure of a Data Block

A data block consists of the following components

- block preheader (DH, DXH)
- block header
- block body.

The **block preheader** is generated automatically. It contains the data formats of the data words entered in the block body. The user has no means of influencing the generation of the block preheader.

IMPORTANT!

If you transfer a data block from the programmable controller or the EPROM submodule to a floppy disk, the respective block preheader will be erased. Due to this you should never alter a data block with different data formats in the programmable controller and then transfer it back to the floppy disk, since all data words of this particular DB are automatically assigned the data format selected in the *presettings mask*.

The **header** is assigned 5 words in the memory and contains

- the block identifier
- the identifier of the programmer
- the block number
- the library number
- the block length (incl. the length of the header)

The **block body** contains, in ascending order and starting with data word DW 0, the data words used by the user program. Each of the data words is assigned 1 word in the memory (16 bits).

A data block can occupy max. 2000 words in the processor memory. When entering or transferring data blocks with the PG always take into consideration the storage capacity of your programmer!

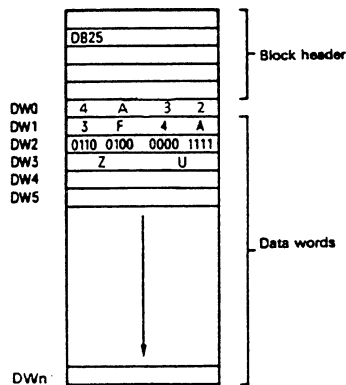


Fig. 2-9: Structure of a data block

2.4.2 Programming Data Blocks

This is how a data block is created:

- Input a data block **number** between 3 and 255 (for DB data blocks) or between 1 and 255 (for DX data blocks).

IMPORTANT!

The data blocks DB 0, DB 1, DB 2 and DX 0 are reserved for specific functions and thus not freely assignable!

- Input the individual **data words** in the desired data format.

Permissible data formats:

Examples:

KM = bit pattern	00100110 00111111
KH = hexadecimal number	263F
KY = byte	38,63
KF = fixed-point number	+9791
KG = floating-point number	+1356123+12
KS = character (ASCII)	?!ABCD123-+.,%
KT = time w. time base specif.(decimal)	055.2
KC = counter value	234
AL = assignment in assignment list (not with PG software S5DOS)	MOTOR1 = Q 12.5

IMPORTANT!

Input of data words is not terminated with block end statement 'BE'!

2.4.3 Opening Data Blocks

A data block (DB/DX) can only be opened unconditionally. This is possible within an organization, program, sequence or function block. One data block can be opened several times within the program.

This is how a data block is opened:

- DB data block with statement C DB..
- DX data block with statement CX DX..

Access to the data stored in the data block opened is possible during the program processing by means of the load and transfer commands:

The contents of the addressed data word are transferred to accumulator 1 and processed by the processor by means of a **load command**.

Load commands: L DW.. (word)
 L DR.. (right-hand byte)
 L DL.. (left-hand byte)
 L DD.. (double word)

The data contained in accumulator 1 are transferred to the addressed data word by means of a **transfer command**.

Transfer commands T DW..
 T DR..
 T DL..
 T DD..

The contents of a data word are not altered during the loading procedure.

The original contents of a data word are overwritten during the transfer procedure.

IMPORTANT!

* Before accessing a data word you will have to open the respective data block in the user program since this is the only means for the processor to find the correct data word! The addressed data word must be contained in the block opened, otherwise the system program will identify a transfer error for command T Dx or will load random values if command L Dx is entered.

* Using load and transfer commands, access is possible up to data word number 255 only.

Example: Transferring data words

The intention is to transfer the contents of data word DW 1 of data block DB 10 to data word DW 1 of data block DB 20 (cf. fig.).

Input the following statements:

```

C DB10    (open DB 10)
L DW1     (load DW 1 in the accumulator)
C DB20    (open DB 20)
T DW1     (transfer DW 1 from the accumulator to DW 1)
    
```

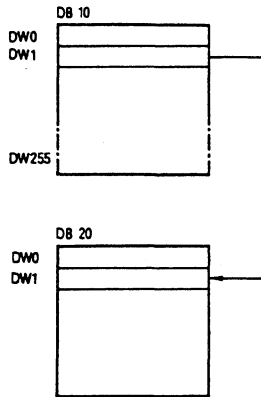


Fig. 2-10: Opening data blocks and accessing data words

After a data block has been opened all subsequent instructions with the operand range D refer to the block opened. The data block opened still remains valid, even if the program processing is continued in another block by means of a *block call* (e.g. JU/JC PB 20).

If another data block is opened in this block, it is only valid in the block opened (PB 20). After the jump has been made back to the block containing the call, the original data block is again valid.

IMPORTANT!

A data block opened thus remains valid until

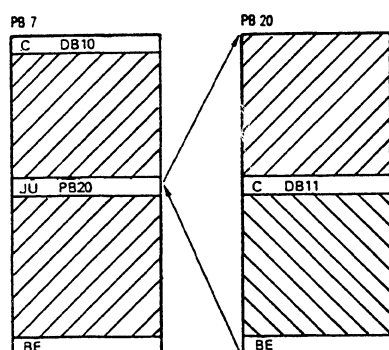
- a) another data block is opened
- or b) a jump back to the primary block is carried out
- or c) the block containing the call is terminated by means of 'BE'.

Example: Validity range of data blocks

The data block DB 10 is opened in the program block PB 7 (C DB10). The data contained in this data block are then processed in the subsequent program processing.

The program block PB 20 is processed after the call (JU PB20). However, the data block DB 10 is still valid. It is not until the data block DB 11 (C DB11) is opened that the data area changes. The data block DB 11 is now valid until the end of program block PB 20 (BE).

After a jump has been made back to program block PB 7, data block DB 10 is again valid.



/// Validity range of DB 10
 \\ \\ Validity range of DB 11

Fig. 2-11: Validity range of a data block after it has been called

2.4.4 Special Data Blocks

Data blocks DB 0, DB 1, DB 2 and DX 0 are reserved for specific functions. They are organized by the system program and are not freely assignable by the user.

- **Data block DB 0** (refer to Subsection 8.2.2)

Data block DB 0 contains the address list with the start addresses of all blocks in the user memory or the data block RAM of the processor. This address list is created by the system program during the initialization (every time the power is turned on and after an overall reset has taken place) and is updated automatically when blocks are input or altered by the PG.

- **Data block DB 1** (refer to Chapter 10.3)

Data block DB 1 contains the list of digital inputs and outputs (P-I/O's with relative byte addresses from 0 through 127) as well as that of interprocessor communication flag inputs and outputs assigned to the processor and, if required, a timer block length.

In the case of multiprocessor operation, the user will have to create the DB 1 for each of the processors used. The DB 1 is used for single-processor operation in order to reduce the cycle times, since only those inputs, outputs, interprocessor communication flag inputs and outputs or times that are contained in the DB 1, are updated.

- **Data block DB 2** (refer to Subsection 4.4.3)

Data block DB 2 serves for parameter assignment of the compact closed-loop control R64 by the user. The R64 is available as an off-the-peg software product. This function operates aided by the system program.

For more information, refer to description "Compact closed-loop control in the R processor of the S5 135U", order-no. C79000-B8576-C365-03.

- **Data block DX 0** (refer to Chapter 7)

By programming the DX 0 data block, you may alter the presettings of certain system program functions (e.g. when processing the start-up) and thus adapt the functions of the system program to your own requirements.

3 Program Processing

3.1 Summary

The STEP5 user program can be processed in various ways.

The cyclic program processing is normally prevalent:
With this type of processing the organization block OB 1 is run through cyclically and the user program organized in this block is processed continuously interspersed with various block calls.

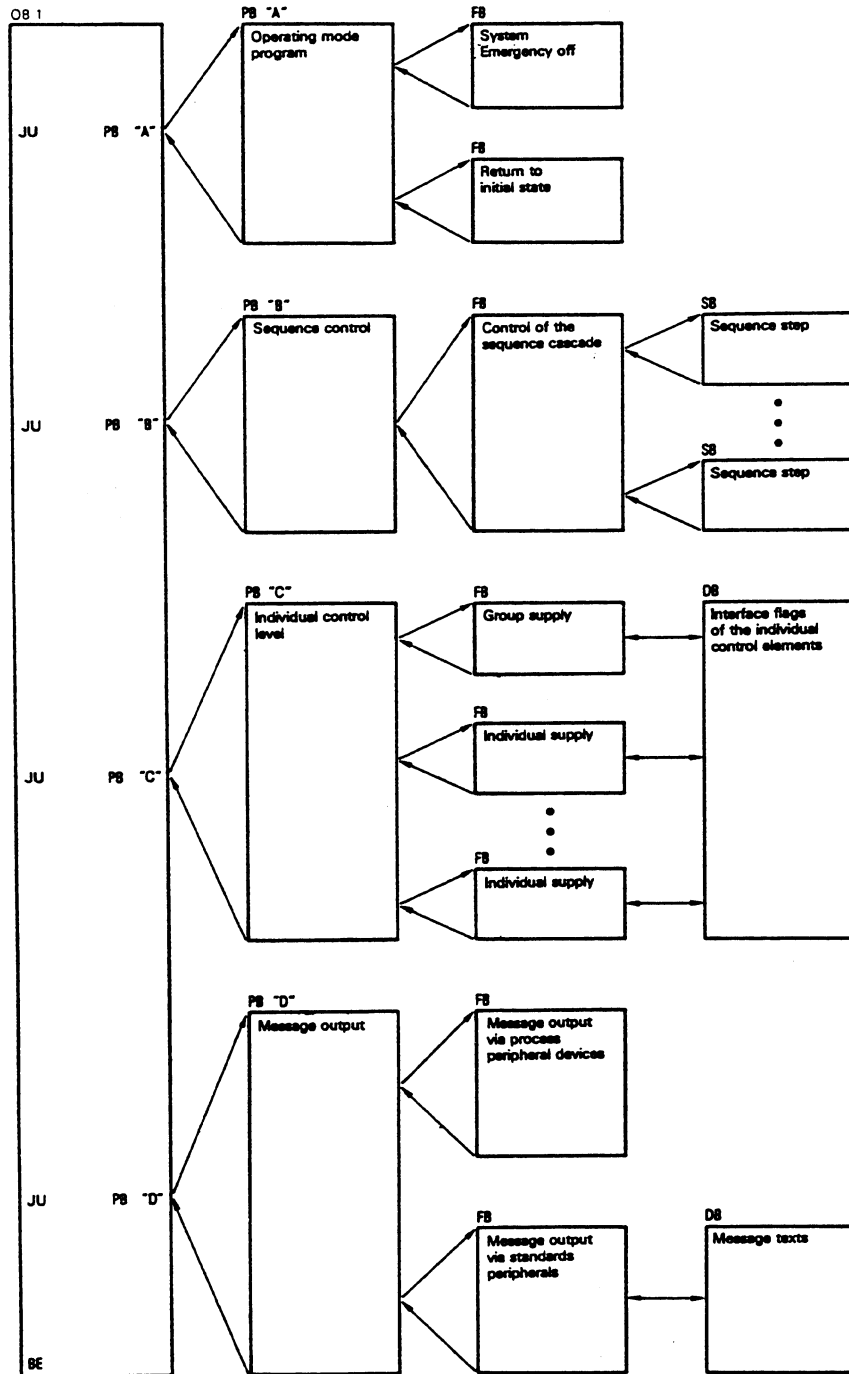
3.1.1 Program Organization

The program organization serves to determine if and in what sequence the blocks that have been created by you are to be processed. You therefore program conditional or unconditional calls for the blocks you require in the organization blocks.

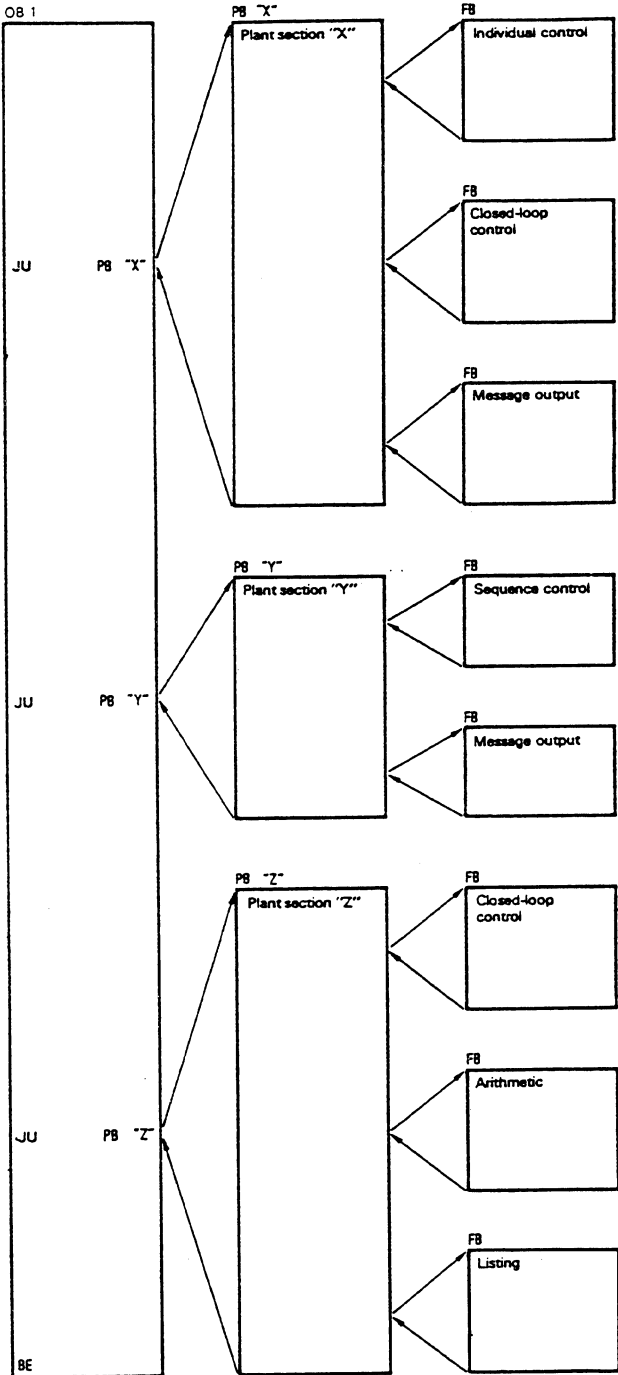
Calling further program, function or sequence blocks in any combination (successively or nested in one another) is possible in the program of the individual organization, program, function or sequence blocks.

It is advisable to have the user program organized so that important program structures or system components that are handled by a program are clearly identifiable.

Example 3-1: Organization of the user program according to the program structure



Example 3-2: Organization of the user program according to the system structure



IMPORTANT!

Nesting of a maximum of 62 blocks is possible. If more than 62 blocks are called the processor will output an error message.

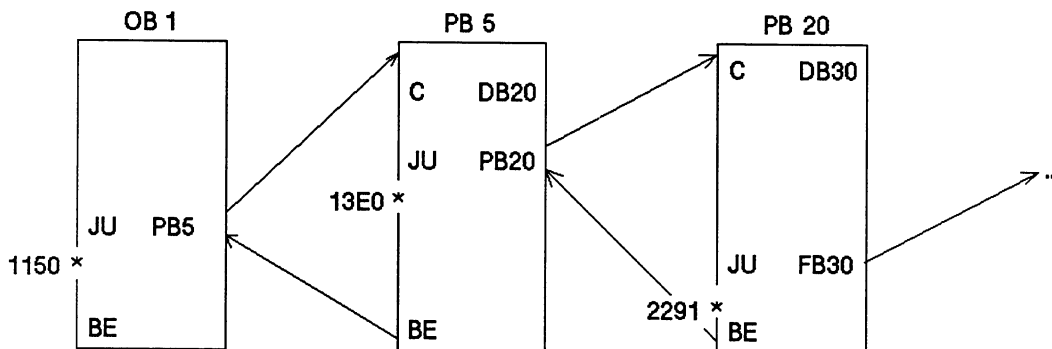
How to determine the nesting depth of your program:

- Add up all organization blocks that you have programmed (in the example on the following page: 4 OB's).
- Add up the nesting depth of the individual organization blocks (in the example: 2 + 2 + 1 + 0 = 5).
- Both values added together give you the program nesting depth (in the following example: 4 + 5 = nesting depth 9).
The value obtained should not exceed 62!

The position of a block in the user memory (or DB-RAM) is determined by its **block start address**: This is the address of the location in the memory where the first STEP5 command is found.

So that the processor can find the block called in the memory (JU/JC * xx, C DB), the start addresses of all programmed blocks are entered in the block address list in data block DB 0. DB 0 is organized by the system program and the user cannot call this block!

After processing a block which has been called the processor must be able to return to the block containing the original call. The processor therefore stores the **return address** whenever a new block is called. The return address is the address of that location in the memory where the STEP5 statement which follows the block call is found. The start address and the length of the data block valid at this particular point are also stored.

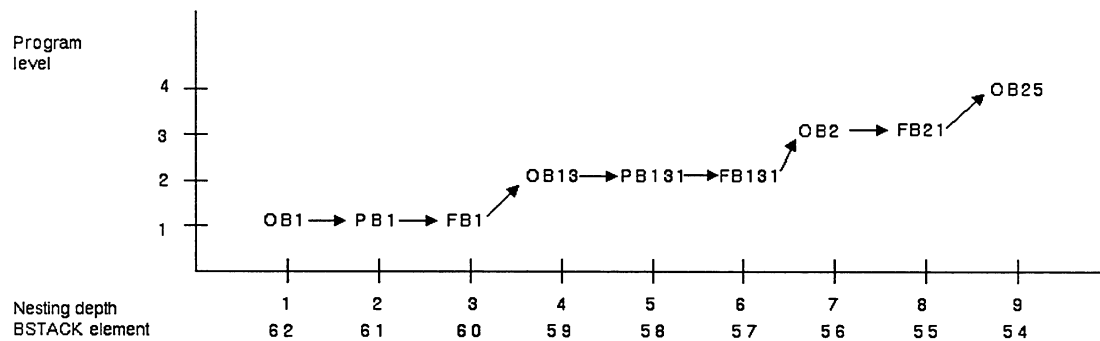


* Return addresses

This data is entered in the **block stack (BSTACK)**. The BSTACK is filled from the lower end: The first entry is equivalent to BSTACK-element 62, the second entry to BSTACK-element 61, etc.. If the block called has been processed completely and the processor has returned to the original block, the respective entries will be erased.

The block stack is full after 62 entries (BSTACK-element 1). If the permissible nesting depth is exceeded the processor will stop.

Example: Block nesting depth and block stack (BSTACK)



3.1.2 Program Storage

The processor can only process the user program if it has been loaded into the program memory. Here, there are two possibilities:

- a) If a plug-in **RAM-submodule** is used, you can transfer your user program directly from the programmer to the processor.

Fast and frequent alteration of the memory contents is possible if a RAM module is used. A back-up battery ensures that the user program is not lost in case of a power failure (refer to the operating instructions of the central controller S5 135U for information about the back-up battery).

All programmed blocks are stored in the RAM submodule in an arbitrary sequence. As soon as you alter a block the sequence of the blocks in the memory is also changed.

Data blocks DB and DX are deposited in the RAM submodule until it is full. Then they are deposited in the data block RAM of the processor.

- b) The complete user program is deposited permanently in a plug-in EPROM module. The user program is completely safe in the EPROM, even if there is a power failure and back-up battery.

The contents of an EPROM cannot be altered easily. Due to this, those data blocks that contain *variable* data and that will be altered during the processing of the user program, will have to be copied from the EPROM module to the data block RAM of the processor during a cold restart (refer to special function OB's 254 and 255, Subsection 6.4.5).

If the processor detects an error while searching the user memory it will request an overall reset and will go over to the stop status. After the overall reset, you will again have to load the user program into the memory.

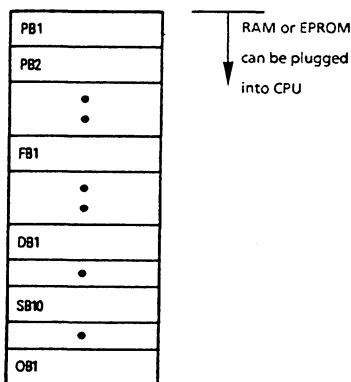


Fig. 3-4: Blocks in the program memory

3.1.3 Running the STEP5 User Program

The user program can be run in different ways. With PC's, the usual method is cyclic **program processing**.

At the end of the start-up the system program automatically calls the organization block OB 1 (or FB 0). There the processor starts with the first STEP5 instruction of the user program and then processes, one after the other, all instructions in the user program. Once the program end is reached, the processor starts the next cycle again at the program start.

The system program executes certain activities during every cycle:

- It starts the cycle time.
- It updates the process image of the inputs.
- It updates the interprocessor communication input flags.

- It calls the user interface.
- It updates the process image of the outputs.
- It updates the interprocessor communication output flags.

Cycle time

The system program monitors the time which the processor requires to run the user program. At the start of the program execution the cycle time to be monitored is started by the system program.

The standard setting of the maximum permissible value is 150 ms.

You can set the cycle time yourself or restart it while cyclic program is running (see DX 0, special function OBs 221 and 222).

The *total cycle time* is the execution time of the user program plus the execution time required for the cyclic part of the system program (see fig. on following page).

The execution time of the user program is the sum of the execution times of all blocks called in one program run (from the OB 1 or FB 0 call to the end). If, for example, you call a certain block n-times, then you will have to add its execution time n-times.

Process image of the inputs and outputs (PII and PIQ)

Before the STEP5 program execution starts the signal states of the input-I/O modules are read and transferred to the process image of the inputs (in the system data register of the processor). Based on the process image of the inputs the user program now computes the process image of the outputs. After the STEP5 program has been run, the signal states of the process image of the outputs are transferred to the output-I/O modules.

Thus, the process image is a memory area whose contents are only output to the I/O's or read in from the I/O's once per cycle.

IMPORTANT!

A process image only exists for input and output bytes of the P-I/O's with byte addresses from 0 through 127!

Interprocessor communication flags (IPC flags)

The IPC flags are used for the data exchange between the individual processors (for multiprocessor operation) as well as between the processor and the communications processors.

Before the STEP5 program is started the IPC-input flags of the processor are read-in. After the STEP5 program has been run, the IPC-output flags are transferred to the communications processors.

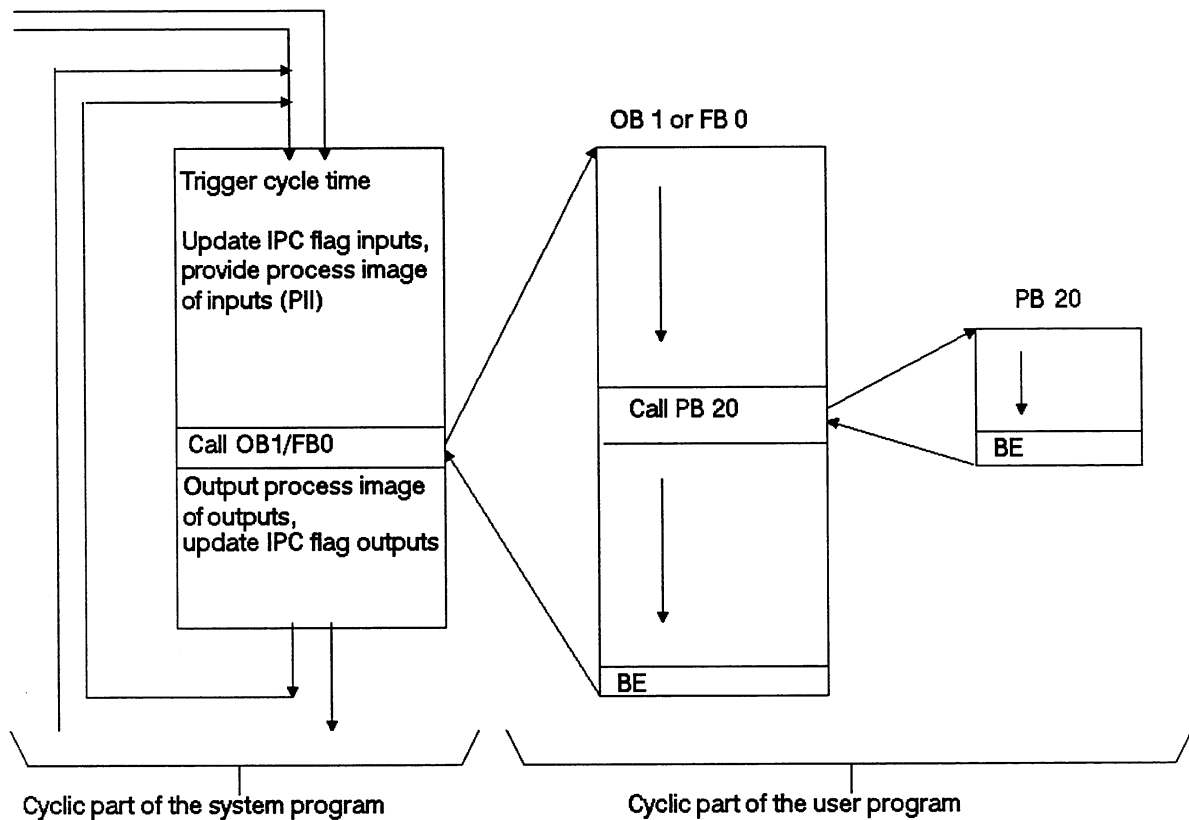


Fig. 3-5: Cyclic program execution

Breakpoints

Interruption of the cyclic program execution may be caused by

- interrupt-driven program execution
- time-driven program execution

The program can be interrupted or aborted

- if equipment defects or program errors occur
- by the operator (PG function, stop switch).

3.1.4 Running the Program

There are two ways of determining the reactions of the processor during the start-up, during the cyclic program and in case of a fault occurring:

- a) by programming the organization blocks OB 1 through OB 34 (interfaces between system and user program, see Subsection 2.2.3) and
- b) by programming the extended data block DX 0 (see chapter 7).

The **organization blocks OB 1 through OB 34** are the interfaces between the system and the user program since, on the one hand, they are called by the system program and, on the other hand, can be filled with STEP5 instructions just like 'normal' blocks. Calling further blocks is possible in these organization blocks. The STEP5 program contained in these blocks helps the user to determine the reaction of the processor to certain events.

The organization blocks OB 1 through OB 34 become effective as soon as they are loaded into the program memory (even during operation).

If they are not programmed by the user, there will either be no reaction from the processor at all or - with most faults - it will go over to the stop state (refer to Chapter 5.4).

Another way of influencing the reaction of the processor is to program **DX 0**.

The functions executed by the system program are preset standard functions. By specifying certain parameters in DX 0 it is possible to alter these standard preset values for certain system program functions.

Just as with the organization blocks, DX 0 can be loaded in the program memory during operation. **However, it will not become effective until the next cold restart is carried out.**

If DX 0 is not programmed by the user, the preset values are valid.

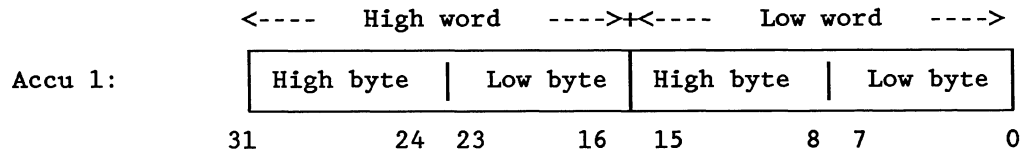
3.2 STEP5 Operation Set with Examples of Programming

STEP5 operations can be divided into different groups:

- The binary functions include binary logic operations, memory operations and timer as well as counter operations.
- The digital functions include loading and transfer operations, comparison operations as well as arithmetic operations.
- The organizational functions include the jump operations, stop and block end operations, instructions for the generation or calling of a data block etc..

The accumulators as auxiliary registers

The majority of STEP5 operations make use of two registers (32 bits) as the source for operands and the destination for results: accumulator 1 (accu 1) and accumulator 2 (accu 2).



The accumulators are changed, depending on the STEP5 instruction to be executed.

Examples:

- Accumulator 1 is always used as the destination for loading operations. The initial contents of accumulator 1 are shifted to accumulator 2 (stack lift). Accumulators 3 and 4 are not altered during any loading operation.
- Arithmetic instructions operate on the contents of accumulator 1 and accumulator 2, write the result in accumulator 1 and transfer the contents of accumulator 3 to accumulator 2 and the contents of accumulator 4 to accumulator 3 (stack drop).
- If a constant (ADD BN/KF) is added to the contents of accumulator 1 the accumulators 2, 3, and 4 are not altered.

Result bits

There are commands for processing information in bits and commands for processing information in words (8, 16, 32 bits).

For both groups, there are commands which set condition codes and commands which evaluate condition codes (see annex: operation list, influencing the condition codes). Corresponding to the two groups of instructions, bit condition codes (bit 0 through 3) and word condition codes (bit 4 through 7) exist. The condition code byte can be read out at the programmer and appears as follows:

Word condition codes				Bit condition codes			
DSP1	DSP0	OV	OS	OR	STA	RLO	$\overline{\text{ERAB}}$
Bit 7	6	5	4	3	2	1	0

Bit condition codes:

- $\overline{\text{ERAB}}$ First scan
This is where a logic operation starts. At the end of a sequence of logic operations (memory operations) the ERAB is set = 0. Commands which set ERAB = 0 (e.g. result allocation = Q2.4) have the effect of limiting the RLO (see annex), i.e. the result of logic operation will remain constant. Evaluation is possible (e.g. by means of RLO dependent instructions), however, no further logic operations with it are possible. It is not until the next logic statement (= first scan) occurs that the result of logic operation is again created and ERAB set = 1.
- RLO Result of logic operation
Result of bit wide logic operations. Truth statement for comparison instructions (see annex: operation list, binary logic operations and comparison operations).
- STA Status
States the logical status of the bit just scanned or set. The status is updated in the case of binary logic operations (except A(, 0(,), 0) and memory operations.
- OR Or
Informs the processor that the following AND logic operations are to be handled before an OR logic operation (AND before OR).

Word condition codes

- OV **Overflow**
States whether the permissible numeric range was exceeded by the arithmetic operation just completed.
- OS **Overflow, latching**
The over bit has been stored. This makes it possible to recognize if an error due to overflow (over) has occurred in the course of several arithmetic operations.

DSP1 and DSP0

Coded result bits. Their interpretation is illustrated in the following table:

Word result bits		Fixed point calculation, result	Logic operations, digital	Comparison contents of accu 1 and accu 2	Shift: last bit shifted
DSP1	DSP0				
0	0	result = 0	= 0	accu2 = accu 1	0
0	1	result < 0	-	accu2 < accu 1	-
1	0	result > 0	≠ 0	accu2 > accu 1	1

Jump operations are available for an immediate evaluation of the condition codes (see Subsection 3.2.2).

3.2.1 Basic Operation Set

o Binary logic operations

Operation	Parameter	Function
)		Close brackets
A (ANDing expressions in brackets
O (ORing expressions in brackets
O		ORing AND functions
A		AND operation
O		OR operation
I	0.0 to 127.7	with scanning of an input for signal status "1"
Q	0.0 to 127.7	with scanning of an output for signal status "1"
F	0.0 to 255.7	with scanning of a flag for signal status "1"
D	0.0 to 255.15	with scanning of a data word for signal status "1"
N I	0.0 to 127.7	with scanning of an input for signal status "0"
N Q	0.0 to 127.7	with scanning of an output for signal status "0"
N F	0.0 to 255.7	with scanning of a flag for signal status "0"
N D	0.0 to 255.15	with scanning of a data word for signal status "0"
T	0 to 255	with scanning of a timer for signal status "1"
N T	0 to 255	with scanning of a timer for signal status "0"
C	0 to 255	with scanning of a counter for signal status "1"
N C	0 to 255	with scanning of a counter for signal status "0"

Binary logic operations generate the result of logic operation (RLO) as their result.

At the start of a logic operation sequence the results from the first logic operation (first scan) are only dependent on the status of the scanned signal and whether or not it is negated (N = negation); they are not, however, dependent on the type of logic operation (O = OR, A = AND).

During a logic operation sequence, the RLO is formed from the type of logic operation, the previous RLO and the status of the scanned signal. A logic operation sequence is completed by an RLO limiting (ERAB = 0) command (e.g. memory operations).

The RLO remains unchanged until the next "first scan". It can be interpreted, but cannot be further operated on.

Example:

Program	Status	RLO	ERAB
:			
=Q 0.0	0	0	0 ← RLO limited
A I1.0	1	1	1 ← first bit scanned
A I1.1	1	1	1
A I1.2	0	0	1
=Q 0.1	0	0	0 ← RLO limited, end of logic operations sequence

● **Memory operations**

Operation	Parameter	Function
S		set
R		reset
=		assign
	I	0.0 to 127.7
	Q	0.0 to 127.7
	F	0.0 to 255.7
	D	0.0 to 255.15
		an input in the PII
		an output in the PIO
		a flag
		a data word bit

● **Loading, transfer and comparison operations**

Operation	Parameter	Function
L		load
T		transfer
	I B	0 to 127
	I W	0 to 126
	I D	0 to 124
	Q B	0 to 127
	Q W	0 to 126
	Q D	0 to 124
	F B	0 to 255
	F W	0 to 254
	F D	0 to 252
	D R	0 to 255
	D L	0 to 255
	D W	0 to 255
	D D	0 to 254
	P B/	0 to 127
	P Y	
	P B/	128 to 255
	P Y	
	O B	0 to 255
		an input byte from/to the PII
		an input word from/to the PII
		an input double word from/to the PII
		an output byte from/to the PIO
		an output word from/to the PIO
		an output double word from/to the PIO
		a flag byte
		a flag word
		a flag double word
		a data (right-hand byte) from DB, DX
		a data (left-hand byte) from DB, DX
		a data word
		a data double word
		a peripheral byte of the digital inputs or outputs (P area)
		a peripheral byte of the analog or digital inputs or outputs (P area)
		a byte of the extended I/O's (O area)

● Loading, transfer and comparison operations (continued)

Operation	Parameter	Function
L		load
T		transfer
P W	0 to 126	a peripheral word of the digital inputs or outputs (P area)
P W	128 to 254	a peripheral word of the analog or digital inputs or outputs (P area)
O W	0 to 254	a word of the extended I/O's (O area)
L		load
K M	16 bit pattern	a constant as bit pattern
K H	0 to FFFF	a constant in hexadecimal code
K F	-32 768 to +32 767	a constant as fixed point number
K Y	0 - 255 for each byte	a constant, 2 bytes
K B	0 to 255	a constant, 1 byte
K S	2 alphanum. character	a constant, 2 ASCII characters
K T	0.0 to 999.3	a time value (constant)
K C	0 to 999	a counter value (constant)
K G	¹⁾	a constant as floating point number (32 bit)
T	0 to 255	a timer value
C	0 to 255	a counter value
LC T	0 to 255	BCD coded loading of a timer value
LC C	0 to 255	BCD coded loading of a counter value
! =		compare for equal
> <		compare for not equal
>		compare for greater than
> =		compare for greater than or equal
<		compare for less than
< =		compare for less than or equal
F		two fixed point numbers (16 bits)
D		two fixed point numbers (32 bits)
G		two floating point numbers (32 bits)

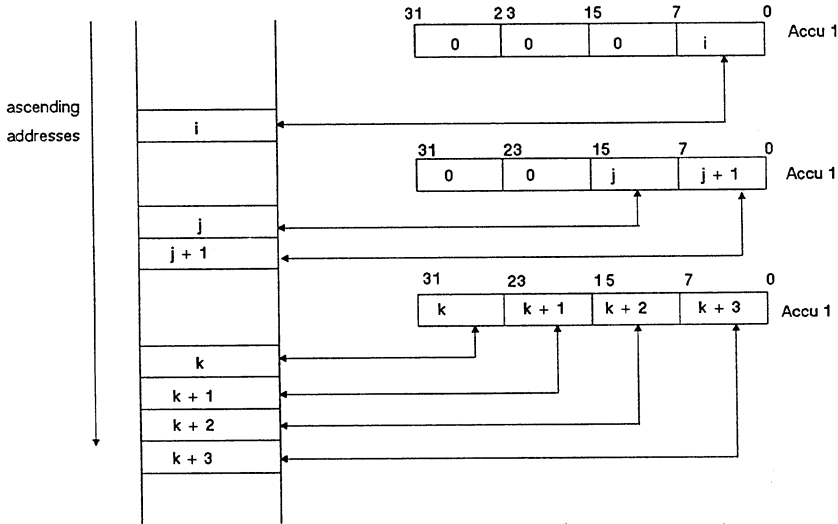
Loading operations write the value addressed in accumulator 1. The previous contents of accumulator 1 are saved in accumulator 2 (stack lift).

Transfer instructions write the contents of accumulator 1 in the memory location addressed.

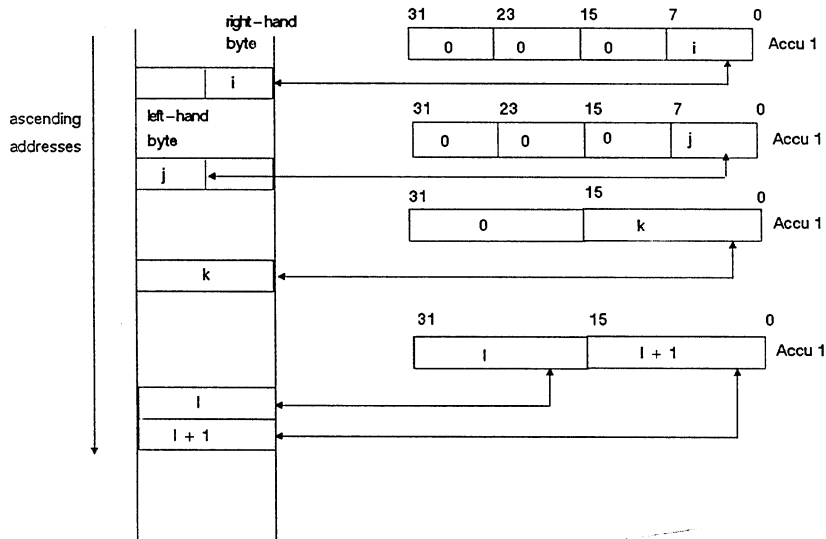
¹⁾ $\pm 0.1469368 \times 10^{-38}$ through $\pm 0.1701412 \times 10^{39}$

Example: Loading/transferring a byte, word or double word from/to a memory area organized in **bytes** (PII, PIO, flags, I/O's).

:L IW 5 Byte 5 and 6 of the PII are loaded in accumulator 1.
 :L FY 10 Flag bytes 10 through 13 are loaded.



Example: Loading/transferring a byte, word or double word from/to a memory area organized in **words**.



Words or double words are stored in the memory, beginning with the most significant byte or word in ascending order of addresses. The excess bits in accumulator 1 are erased when a byte or word is loaded.

The loading operations do not affect the condition codes. Transfer operations will in general clear the OS bit. The result of the compare commands are the RLO and the word condition codes DSP1 and DSP0. The contents of accumulator 1 and 2 are always compared (see program examples and operation list).

The I/O's can be called by loading and transfer operations:

1. directly:
with L/T PY, PW, OB, OW or
2. via the process image:
with L/T IB, IW, ID, QB, QW, QD and with logic operations.

The process image of the outputs is corrected at the same time when transfer operations T PY 0 through 127 and T PW 0 through 126 are executed.

The process image is a memory area the contents of which are output to the I/O (process output image, PIO) or read in from the I/O's (process input image, PII) only *once per cycle*. This prevents output "chattering" due to frequent alterations of the logic condition of a bit within a program cycle.

Please note the following points regarding the I/O's.

- A process image of the inputs and outputs exists for 128 input and 128 output bytes of the P I/O's with byte addresses from 0 through 127.
- No process image exists for the whole O I/O area and the P I/O area with relative byte addresses from 128 through 255! (see 8.2.1 for the I/O address distribution.)
- Input/output modules with addresses of the O I/O area are only permissible in expansion units (not in the central controller).
- Use of either P I/O's or O I/O's is possible in one expansion unit.
- If relative addresses of the P I/O's or O I/O's are used in an expansion unit, these addresses are no longer permissible for I/O modules in the central controller (double addressing!).

● **Timer and counter operations**

In order to load a time by means of a starting command or a counter by means of a setting command the value must first be loaded in accumulator 1.

The following loading operations are recommended:

For times: L KT, L IW, L QW, L FW, L DW.

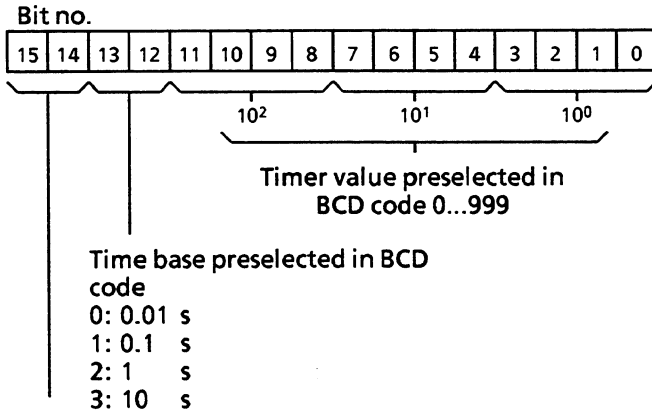
For counters: L KC, L IW, L QW, L FW, L DW.

Operation	Parameter	Function
S P T	0 to 255	starting a timer as a pulse
S E T	0 to 255	starting a timer as an extended pulse
S D T	0 to 255	starting a timer as ON delay
S S T	0 to 255	starting a timer as a latching ON delay
S F T	0 to 255	starting a timer as OFF delay
R T	0 to 255	resetting a timer
S C	0 to 255	setting a counter
R C	0 to 255	resetting a counter
C U C	0 to 255	incrementing a counter
C D C	0 to 255	decrementing a counter

When the timer or counter operations SP, SD, SE, SS, SF, and S are executed the value contained in accumulator 1 is transferred to the timer or counter location (corresponds to the transfer command) and the respective operation is triggered.

If the time or count value is loaded using IW, QW, FW or DW the corresponding word must have the following structure:

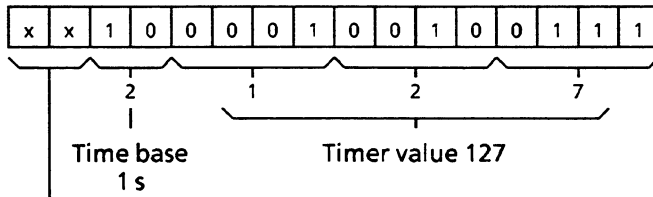
For the time value



These bits are irrelevant, i.e., they are not considered when the timer is started

Example: Setting a time of 127 s.

Bit assignment:



irrelevant

IMPORTANT!

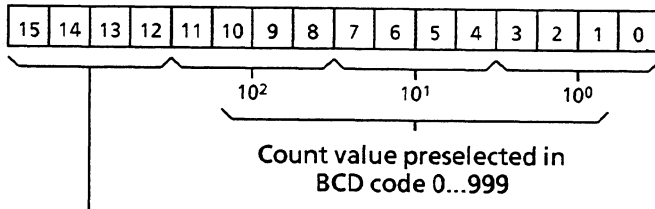
Whenever a timer is started there is an inaccuracy of one time base unit!

This means that if you start a timer location using the time base '1' (= 100 ms) n-times the inaccuracy will be n-times 100 ms.

If timer locations are to be used, select as small a time base as possible (time base << time value)!

Example: time 4 s not: 1 s x 4
 but: 10 ms x 400

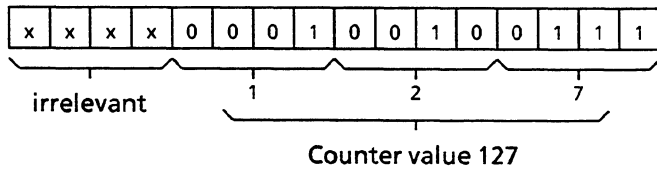
For the count value



These bits are irrelevant, i.e., they are not considered when the counter is set

Example: Setting a count value of 127

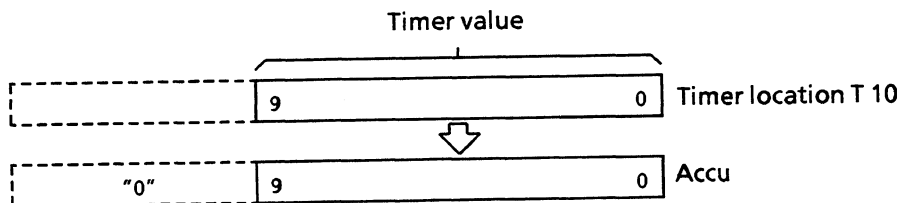
Bit assignment:



The time or count value is stored in the timer or counter location and is binary coded. In order to scan the timer or the counter the value of the timer or counter location can be loaded in accumulator 1 either directly or BCD coded.

Examples:

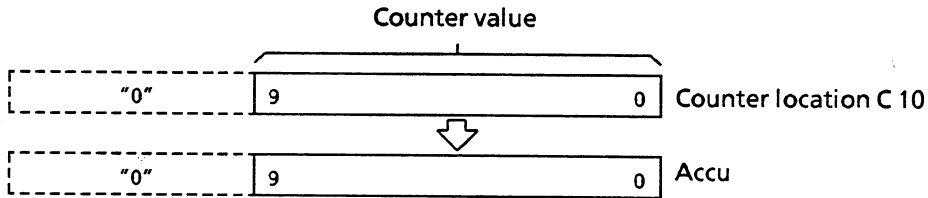
Direct loading of **time values**:



L T 10 Direct loading of the binary time value of timer T 10 in the accumulator

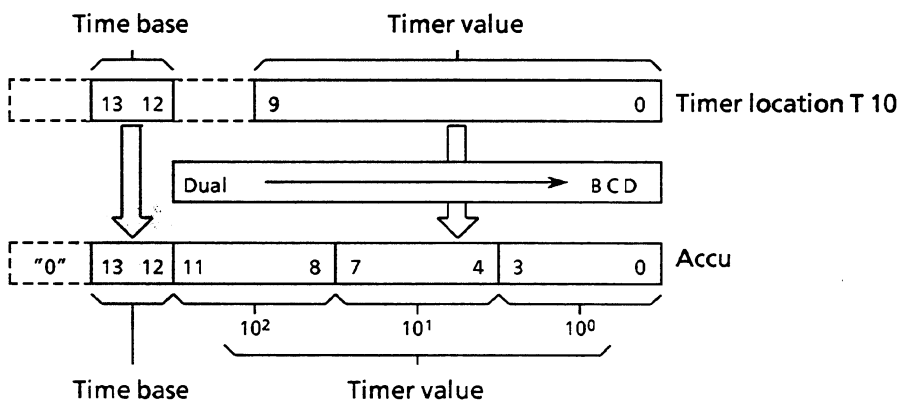
The time base is not loaded.

Direct loading of **count values**:



LC C 10 Direct loading of the count value of counter C 10 in the accumulator.

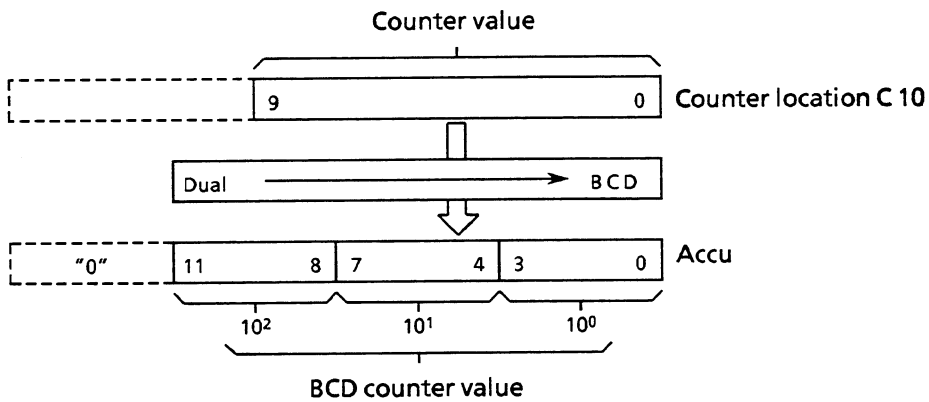
Coded loading of **time values**:



LC T 10 Coded loading of time value and time increment of time T 10 in the accumulator

The time increment is also loaded.

Coded loading of **count values**:



LC C 10 Coded loading of count value of counter C 10 in the accumulator

With coded loading the status bits 14 and 15 of the timer locations or 12 through 14 of the counter locations will not be loaded. Instead, accumulator 1 will contain the value 0. The value now in the accumulator can be processed.

● **Arithmetic operations**

Operation	Parameters	Function
+ F		addition of 2 fixed point numbers (16 bits)
- F		subtraction of 2 fixed point numbers (16 bits)
x F		multiplication of 2 fixed point numbers (16 bits)
: F		division of 2 fixed point numbers
+ G		addition of 2 floating point numbers
- G		subtraction of 2 floating point numbers
x G		multiplication of 2 floating point numbers
: G		division of 2 floating point numbers

The arithmetic operations use the contents of accumulators 1 and 2 (see operations list). The result is then available in accumulator 1. The arithmetic registers are changed by an arithmetic operation as follows:

before: <accu 1> <accu 2> <accu 3> <accu 4>
 after: <result> <accu 3> <accu 4> <accu 4>

The previous contents of accumulator 2 are lost.

Note that commands for subtracting and adding of double-word fixed-point numbers are available in the supplementary operations.

● **Block calls**

Operation	Parameters	Function
J U		jump unconditional
J C		jump conditional (only when RLO = 1)
O B	1 to 39	to an organization block
O B	40 to 255	to a system program special function
P B	0 to 255	to a program block
F B	0 to 255	to a function block
S B	0 to 255	to a sequence block
D O F X	0 to 255	unconditional jump to an extended function block
D O C F X	0 to 255	conditional jump to an extended function block
C D B	3 to 255	DB data block call
C X D X	1 to 255	DX extended data block call
B E		block end
B E C		conditional block end (only when RLO = 1)
B E U		unconditional block end

● **No operation**

Operation	Parameters	Function
N O P	0	no operation
N O P	1	no operation
B L D	0 to 255	display construction statement for the PG (is treated as a no operation by the CPU)

● **Stop statement**

Operation	Parameters	Function
S T P		CPU stops

Sample programs for logic, memory, timer, counter and compare operations

● **Logic operations**

AND operation

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 1.1 A I 1.3 A I 1.7 = Q3.5 </pre>		

A "1" signal appears at output Q 3.5 when all the inputs have "1" signals simultaneously
 A "0" signal appears at output Q 3.5 if at least one of the inputs has a "0" signal.
 There are no restrictions on the number of scans or on the programming sequence.

OR operation

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> O I 1.2 O I 1.7 O I 1.5 = Q3.2 </pre>		

A "1" signal appears at output Q 3.2 if at least one of the inputs has a "1" signal.
 A "0" signal appears at output Q 3.2 if all of the inputs have a "0" signal.
 There are no restrictions on the number of scans or on the programming sequence.

● Logic operations (continued)

AND before OR operation

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> A I 1.5 A I 1.6 O A I 1.4 A I 1.3 = Q3.1 </pre>		

A "1" signal appears at output Q 3.1 when the output of at least one of the ANDing operations is "1".
 A "0" signal appears at output Q 3.1 when neither of the ANDing operations results in "1".

OR before AND operation

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> O I 6.0 O I 6.1 A(O I 6.2 O I 6.3) = Q2.1 </pre>		

A "1" signal appears at output Q 2.1 when input I 6.0 or input I 6.1 and one of the inputs I 6.2 or I 6.3 have a "1" signal.
 A "0" signal appears at output Q 2.1 when input I 6.0 has a "0" signal and the AND condition is not met.

● Logic operations (continued)

OR before AND operation

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A(O I 1.4 O I 1.5) A(O I 2.0 O I 2.1) = Q3.0 </pre>		

A "1" signal appears at output Q 3.0 when both OR conditions are met.
 A "0" signal appears at output Q 3.0 when at least one of the OR conditions is not met.

Scanning for "0" signal status

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 1.5 AN I 1.6 = Q3.0 </pre>		

A "1" signal appears at output Q 3.0 only when input I 1.5 has a "1" signal (normally open contact actuated) and input I 1.6 has a "0" signal (normally closed contact not actuated).

● Memory operations

RS flip-flops for latching signal output

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 2.7 S Q3.5 A I 1.4 R Q3.5 </pre>		

A "1" signal at input I 2.7 sets the flip-flop, (signal "1" at output Q 3.5).
 If the signal at input I 2.7 changes to "0", the flip-flop status remains unchanged, i.e. the signal is latched.
 A "1" signal at input I 1.4 resets the flip-flop, (signal "0" at output Q 3.5).
 If the signal at input I 1.4 changes to "0", the flip-flop status remains unchanged.
 If the set signal (input I 2.7) and the reset signal (input I 1.4) appear simultaneously, the scan operation programmed last (in this case A I 1.4) is effective during the processing of the remaining program (reset has priority).

● Memory operations (continued)

RS flip-flop with flags

Original	STEP 5 operation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 2.6 S F 1.7 A I 1.3 R F 1.7 </pre>		

A "1" signal at input I 2.6 sets the flip-flop.
 If the signal at input I 2.6 changes to "0", the flip-flop status remains unchanged, i.e. the signal is latched.
 A "1" signal at input I 1.3 resets the flip-flop.
 If the signal at input I 1.3 changes to "0", the flip-flop status remains unchanged.
 If the set signal (input I 2.6) and the reset signal (input I 1.3) appear simultaneously, the scan operation programmed last (in this case A I 1.3) is effective during the processing of the remaining program (reset has priority).

● Memory operations (continued)

Simulation of a momentary contact relay

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 1.7 AN F 4.0 = F 2.0 A F 2.0 S F 4.0 AN I 1.7 R F 4.0 </pre>		

The AND logic condition (A I 1.7 and AN F 4.0) is fulfilled at each positive-going edge of the signal at input I 1.7 and flags F 4.0 ("pulse edge flag") and F 2.0 (pulse flag) are set if the RLO = "1".

The AND logic condition A I 1.7 and AN F 4.0 is no longer fulfilled during the next processing cycle since flag F 4.0 has been set. Flag F 2.0 is reset, i.e. it is "1" during a single program run.

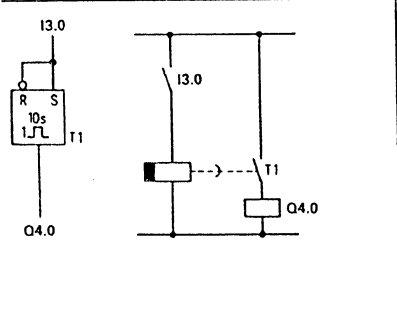
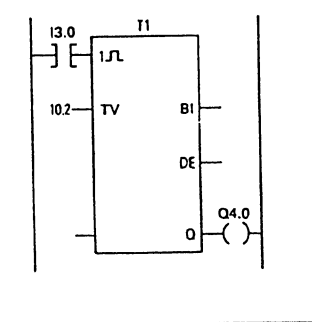
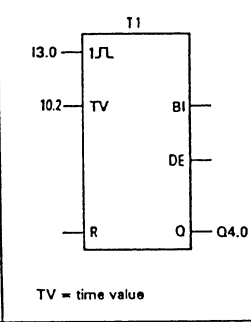
Binary scaler

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 1.0 AN F 1.0 = F 1.1 A F 1.1 S F 1.0 AN I 1.0 R F 1.0 A F 1.1 A Q 3.0 = F 2.0 A F 1.1 AN Q 3.0 S Q 3.0 A F 2.0 R Q 3.0 </pre>		

Output Q 3.0 of the binary scaler changes its state at each positive-going edge of the signal at input I 1.0, i.e. when input I 1.0 changes from "0" to "1". Consequently, half the input frequency appears at the binary scaler output.

● **Timer operations**

Pulse

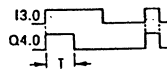
Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> A I 3.0 L KT 10.2 SI T 1 A T 1 = Q 4.0 </pre>		 <p>TV = time value</p>

The timer is started during the first processing cycle if the result of the logic operation is "1". The timer remains unaffected during subsequent processing if this results in a "1" signal. The timer is set to "0" (cleared) if the result of the logic operation is "0". The A T and O T scans result in a "1" signal as long as the timer is running.

KT 10.2:

The timer is loaded with the specified value (10). The number to the right of the decimal point indicates the time base:

0 = 0.01 s 2 = 1 s
 1 = 0.1 s 3 = 10 s



BI and DE are digital outputs of the timer location. The time value is binary at output BI and BCD with time base at output DE.

• **Timer operations (continued)**

Extended pulse

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 3.1 L IW15 SE T 2 A T 2 = Q 4.1 </pre>		

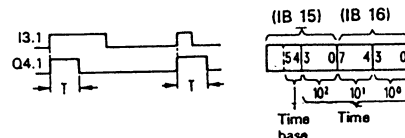
The timer is started during the first processing cycle if the result of the logic operation is "1".

The timer remains unaffected if the result of the logic operation is "0".

The A T or O T scans result in a "1" signal as long as the timer is running.

IW 15:

Setting the time value with the BCD value of the operands I, Q, F or D (input word value 15 in the example).



"On" delay

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 3.5 L KT9.2 SR T 3 A T 3 = Q 4.2 </pre>		

The timer is started during the first processing cycle if the result of the logic operation is "1". The timer remains unaffected during subsequent processing if the result of the logic operation remains "1".

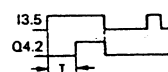
The timer is set to "0" (cleared) if the result of the logic operation is "0".

The A T or O T scans result in a "1" signal when the time has elapsed and the result of the logic operation is still present at the input.

KT 9.2:

The timer is loaded with the specified value (9). The number to the right of the point indicates the time base:

0 = 0.01 s 2 = 1 s
 1 = 0.1 s 3 = 10 s



● **Timer operations (continued)**

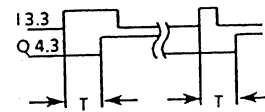
Latching "On" delay

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 3.3 L KT 10.2 SS T 4 A I 3.2 R T 4 A T 4 = Q 4.3 </pre>		

The timer is started during the first processing cycle if the result of the logic operation is "1".

The timer remains unaffected if the result of the logic operation is "0".

The AT or OT scans result in a "1" signal when of the time has elapsed. The signal status only changes to "0" when the timer is reset by the RT function.



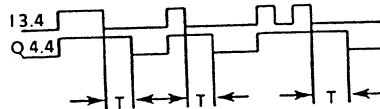
"Off" delay

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 3.4 L KT 10.2 SAT 5 A T 5 = Q 4.4 </pre>		

The timer is started when the result of the logic operation at the start input changes from "1" to "0". It runs for the time programmed.

The timer is set to zero (reset) if the result of the logic operation is "1".

The AT or OT scans result in a "1" signal if the timer is running or the result of the logic operation is still present at the input.



● Counter operations

Set counter

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 4.1 L IW20 S C 1 </pre>		

The counter is set during the first processing cycle if the result of the logic operation is "1". The counter remains unchanged during subsequent processing (irrespective of whether the result of the logic operation is "1" or "0"). The counter is set again (pulse edge evaluation) at the next processing cycle if the result of the logic operation is "1".

The flag necessary for pulse edge evaluation of the set input is included in the counter word.

BI and DE are digital outputs of the counter location. The count values are binary coded at output BI and BCD at output DE.

Reset counter

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 4.2 R C 1 A C 1 = Q 2.4 </pre>		

The counter is set to zero (reset) when the result of the logic operation is "1".

The counter remains unchanged even if the result of the logic operation becomes "0".

● Counter operations (continued)

Counting up

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<p>A I 4.1 CU C 1</p>		

The value of the addressed counter is incremented by 1 up to a maximum of 999. The CU function is effective only on a positive-going pulse edge (from "0" to "1") of the logic operation programmed before CU. The flags necessary for pulse edge evaluation of the counter inputs are included in the counter word.

A counter with two different inputs can be used as an up/down counter by means of the two separate pulse-edge flags for CU and CD.

Counting down

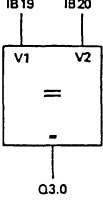
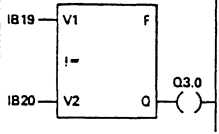
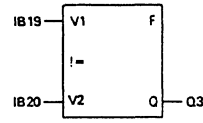
Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<p>A I 4.0 CD C 1</p>		

The value of the addressed counter is decremented by 1 to a minimum 0. The CD function is only effective with a positive-going edge (from "0" to "1") of the logic operation programmed before CD. The flags necessary for pulse edge evaluation of the counter inputs are included in the counter word.

A counter with two different inputs can be used as an up/down counter by means of the two separate pulse-edge flags for CU and CD.

● Compare operations

Comparing for equal to

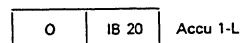
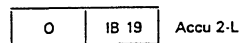
Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> L IB19 L IB20 I = F = Q3.0 </pre>		

The first operand specified is compared with the following operand according to the compare function.

The comparison produces a binary logic operation result:

RLO = "1": the condition is fulfilled, if accu 1-L = accu 2-L

RLO = "0": the condition is not fulfilled, if accu 1-L ≠ accu 2-L



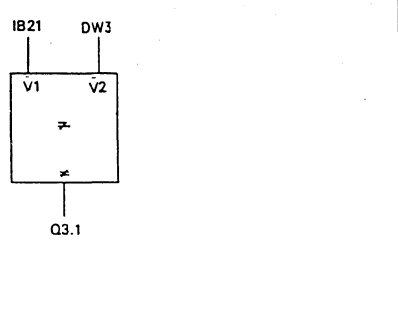
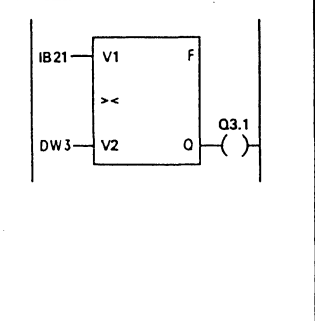
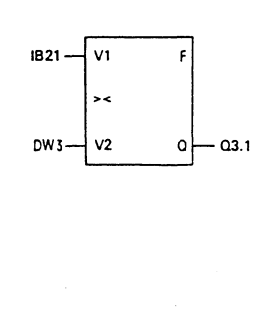
The condition codes DSP1 and DSP0 are set as explained in 4.1. Accu 2-H and accu 1-H remain unaffected during the 16-bit fixed point comparison.

During fixed point comparison (! = F) and floating point comparison (! = G) the total contents of accu 1 and accu 2 (32-bit) are compared with each other.

During the comparison the numerical representation of the operands is taken into account, i.e. the contents of accu 1-L and accu 2-L are interpreted as a fixed point number.

● Compare operations (continued)

Comparing for not equal to

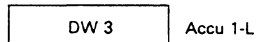
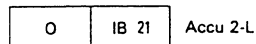
Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> L IB21 L IB >< F = Q3.1 </pre>		

The first operand specified is compared with the following operand according to the comparison function.

The comparison produces a binary logic operation result.

RLO = "1" the condition is fulfilled, if accu 1-L ≠ accu 2-L

RLO = "0" the condition is not fulfilled, if accu 1-L = accu 2-L



The condition codes DSP1 and DSP0 are set according to the table on page 26.

Accu 2-H and accu 1-H remain unaffected during the 16-bit fixed point comparison.

During the 32-bit fixed point comparison and the floating point comparison accu 2-H and accu 1-H are involved.

This also applies to comparing for greater than, greater than or equal to, less than and less than or equal to (see operations list).

With compare operations the numerical representation of the operands is taken into account, i.e. the contents of accu 1-L and accu 2-L are interpreted as a fixed point number.

3.2.2 Supplementary Operation Set

The supplementary operation set can only be used in the function blocks (FB and FX). The total operation set for function block therefore consists of the basic operations and the supplementary operations.

The supplementary operations include the system operations: The system operations allow for you to e.g. overwrite the memory at any position or alter the contents of the working register of the processor. *This means that you should be extremely careful when making use of the system operations (if at all).*

Refer to chapter 9 "memory access" for more information about the 'system functions'.

The operations are only represented in STL for the function blocks. This means that programming the function block programs is not possible graphically (LAD or CSF).

The supplementary operations described in the following may only be used in the function blocks.

The possible combinations of substitution commands with the actual operands are also listed.

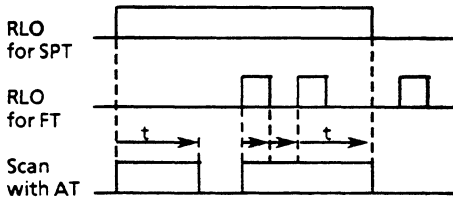
● Binary logic operations

Operations	Description
A = <input type="checkbox"/>	AND operation, scanning a formal operand for signal status "1".
AN = <input type="checkbox"/>	AND operation, scanning a formal operand for signal status "0"
O = <input type="checkbox"/>	OR operation, scanning a formal operand for signal status "1"
ON = <input type="checkbox"/>	OR operation, scanning a formal operand for signal status "0"
	Assign formal operand Inputs, outputs, data and flags addressed in binary code (parameter class I, Q; parameter type BI) and also timers and counters (parameter class T, C) are permitted as actual operands.

● Memory operations

Operation	Description
S = <input type="checkbox"/>	Set (binary) formal operand.
RB = <input type="checkbox"/>	Reset (binary) formal operand.
= = <input type="checkbox"/>	Assign result of logic operation to formal operand.
	Assign formal operand. Inputs, outputs, data and flags addressed in binary code (parameter class I, Q, parameter type BI) are permitted as actual operands.

● **Timer and counter functions**

Operation	Description
FR T 0 to 255	<p>Enabling a timer for restart The operation is only carried out on the leading edge of the result of the logic operation. The timer is restarted if the RLO is "1" at the time of the start operation.</p> 
FR C 0 to 255	<p>Enabling a counter The operation is only carried out on the leading edge of the result of the logic operation. The counter is set (counting up or down) if the result of the logic operation is "1" at the corresponding operation.</p>
FR = <input type="text"/>	<p>Enabling a formal operand for a restart (for description see FRT or FRC depending on formal operand; parameter class: T, C).</p>
RD = <input type="text"/>	<p>Resetting (digital) a formal operand (parameter class: T, C).</p>
SP = <input type="text"/>	<p>Starting a timer, specified as a formal operand, as a pulse with the value stored in the accumulator (parameter class: T).</p>
SR = <input type="text"/>	<p>Starting a timer, specified as a formal operand, as an on-delay with the value stored in the accumulator (parameter class: T).</p>
SEC = <input type="text"/>	<p>Starting a timer, specified as a formal operand, as an extended pulse with the value stored in the accumulator or setting a counter specified as a formal operand for the count value stored in accu 1 (parameter class: T, C)</p>
SSU = <input type="text"/>	<p>Starting a timer, specified as a formal operand, as a latching on-delay with the value stored in the accu or incrementing a counter specified as a formal operand (parameter class: T, C).</p>
SFD = <input type="text"/>	<p>Starting a timer, specified as a formal operand, as an off-delay with the value stored in the accu or decrementing a counter specified as a formal operand (parameter class: T, C).</p>
	<p>Enter formal operand</p> <p>Timers and counters are permitted as actual operand. Exceptions: SP and SR (only timers). The timer or counter value can be assigned as with basic operations: or as a formal operand it can be assigned as follows:</p> <p>Set the timer or counter value with the BCD value, of the IW, QW, FW, DW operands specified as formal operands (parameter class: I, parameter type: W) or as a constant (parameter class: D, parameter type: KT, KC).</p>

Examples

Function block call	Program in function block	Executed program
: JU FB203 NAME : EXAMPLE ANNE : I 10.3 BERT : T 17 FRED : Q 18.4	:A -ANNE :L KT 010.2 :SSU -BERT :A -BERT := -FRED	:A I 10.3 :L KT 010.2 :SS T 17 :A T 17 := Q 18.4
: JU FB204 NAME : EXAMPLE RUTH : I 10.5 PETE : I 10.6 MAUD : I 10.7 DORA : C 15 EMMA : F 58.3	:A -RUTH :SSU -DORA :A -PETE :SFD -DORA :A -MAUD :L KC100 :SEC -DORA :AN -DORA := -EMMA	:A I 10.5 :CU C 15 :A I 10.6 :CD C 15 :A I 10.7 :L KC 100 :S C 15 :AN C 15 := F 58.3
: JU FB205 NAME : EXAMPLE BILL : I 10.4 CARL : T 18 EGON : IW20 DAVE : F 100.7	:A -BILL :L -EGON :SEC -CARL :A -CARL := -DAVE	:A I 10.4 :L IW 20 :SF T 18 :A T 18 := F 100.7

● **Loading and transfer operations**

Operation	Description
= <input type="text"/>	Loading of a formal operand The value of the operand specified as a formal operand is loaded into the accumulator (parameter class: I, T, C, Q; parameter type: BY, W, D)
LD = <input type="text"/>	Coded loading of a formal operand. The value of the timer or counter location specified as a formal operand is loaded in BCD into the accumulator (parameters: T, C).
LW = <input type="text"/>	Loading the bit pattern of a formal operand. The bit pattern of the formal operand is loaded into the accumulator (parameter class: D; parameter type: KF, KH, KM, KY, KS, KT, KG).
LWD = <input type="text"/>	Loading the bit pattern of a formal operand. The bit pattern of the formal operand is loaded into the accumulator (parameter class: D; parameter type: KG).
T = <input type="text"/>	Transferring to a formal operand. The accumulator contents are transferred to the operand specified as a formal operand (parameter class: I, Q; parameter type: BY, W, D).
Enter formal operand	

Operands corresponding to the basic operations are permitted as actual operands. For LW, data is permitted in the form of a binary (KM) or hexadecimal (KH) pattern, 2 numbers in bytes (KY), characters (KS), fixed point number (KF), time values (KT) and count values (KC). For LD, a floating point number is permitted as data.

Operation	Parameter	Description
L RI	0 to 255	Loading a word in accumulator 1 from the area "interface data" (RI area)
L RJ	0 to 255	Loading a word in accumulator 1 from the area "interface data" (RJ area)
L RS	0 to 255	Loading a word in accumulator 1 from the area "system data" (RS area) (free: RS 60 through 63)
L RT	0 to 255	Loading a word in accumulator 1 from the area "system data" (RT area) (free: RT 0 through 255)
T RI	0 to 255	Transferring accumulator 1 to a word from the area "interface" (RI area)
T RJ	0 to 255	Transferring accumulator 1 to a word from the area "interface" (RJ area)
T RS ¹⁾	0 to 255	Transferring accumulator 1 to a word from the area "system" (RS area)
T RT ¹⁾	0 to 255	Transferring accumulator 1 to a word from the area "system" (RT area)

1) System operation

Contrary to areas RI, RJ, and RT, in the RS area only the words RS 60 through RS 63 are free for the user.

Refer to 8.2.4 "RS/RT area" for further information.

● **Arithmetic operations**

Operation	Description
ENT	Entry of data also used during arithmetic operations in accumulators 3 and 4: The contents of accumulators 2 and 3 are loaded in accumulators 3 and 4.

A stack lift is executed into accumulators 3 and 4:

```

<accu 4> := <accu 3>
<accu 3> := <accu 2>
<accu 2> := <accu 2>
<accu 1> := <accu 1>
    
```

Accumulators 1 and 2 are not altered. The previous contents of accumulator 4 are lost.

Example

The following fraction is to be calculated: $(30 + 3 \times 4) / 6 = 7$

	Accu 1	Accu 2	Accu 3	Accu 4
Contents of accumulators before the arithmetic oper.	a	b	c	d
L KF 30	30	a	c	d
L KF 3	3	30	c	d
ENT	3	30	30	c
L KF 4	4	3	30	c
xF	12	30	c	c
+ F	42	c	c	c
L KF 6	6	42	c	c
/ F	7	c	c	c

Operation	Parameters	Description
ADD BN ¹⁾	-128 to +127	Add byte constant (fixed point) to accu 1 ¹⁾
ADD KF ¹⁾	-32768 to +32767	Add fixed point constant (word) to accu 1 ¹⁾

¹⁾ System operation

Operation	Parameters	Description
ADD DF 1) 2)	-214743648 to +2147483647	Add fixed point constant (double word) to accu 1
+D 1)2)		Add two double word fixed-point constants (accu 1 + accu 2)
-D 1)2)		Subtract two double word fixed-point constants (accu 1 + accu 2)
TAK 1)		Swap contents of accu 1 and accu 2

- 1) System operation
- 2) Programming depends on the type of PG and its system software

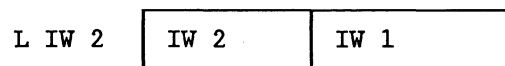
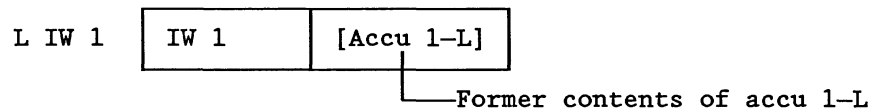
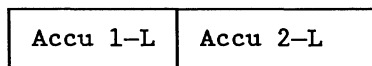
● Digital logic operations

Operation	Description
AW	ANDing of accu 1-L and 2-L
OW	ORing of accu 1-L and 2-L
XOW	Exclusive ORing of accu 1-L and 2-L

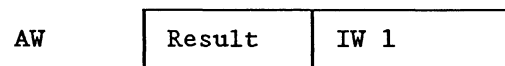
Accumulators 3 and 4 are not affected, but the condition codes DSP1 and DSP0 are (see word result bits).

By means of two loading operations, accumulators 1 and 2 can be loaded corresponding to the operands of the loading operation. Then, the contents of both accumulators can be operated on digitally.

Example



ANDing IW 2 and IW 1:



Organizational functions

● Jump operations

The destination of unconditional and conditional jumps is specified symbolically (a maximum of 4 characters beginning with a letter). The symbolic parameter of the jump instruction is identical to the symbolic address of the statement to be jumped to. When programming, it should be taken into account that the absolute jump distance does not cover more than +127 words and that a STEP 5 statement can consist of more than one word. Jumps can only be carried out within a block; jumps across segments are not permissible.

Note: jump statement and jump destination must be in one segment. Per segment only one symbolic address is permitted for jump destinations. These conditions do not apply to the JR jump, for which an absolute jump distance is specified as a parameter.

Operation	Description
JU = addr	Jump unconditional. An unconditional jump is carried out under all conditions.
JC = addr	Jump conditional. A conditional jump will be carried out if RLO = 1. If RLO = 0, the statement will not be carried out and the result of the logic operation will be set to RLO = 1.
JZ = addr	Jump condition: DSP1, DSP0. A jump will only be carried out if DSP1 = 0 and DSP0 = 0. The logic operation result is not changed.
JN = addr	Jump condition: DSP1, DSP0. A jump will only be carried out if DSP1 ≠ DSP0. The logic operation result is not changed.
JP = addr	Jump condition: DSP1, DSP0. A jump will only be carried out if DSP1 = 1 and DSP0 = 0. The logic operation result is not changed.
JM = addr	A jump will only be carried out if DSP1 = 0 and DSP0 = 1. The logic operation result is not changed.
JO = addr	Jump on overflow. A jump will be carried out if the condition code OV = 1. If there is no overflow, (OV = 0) the jump will not be carried out. The logic operation result is not changed. An overflow occurs if the permissible area for the numerical representation involved is exceeded by an arithmetic operation.

addr = symbolic address (a maximum of 4 characters)

Operation	Description
JS = addr	Jump if the condition code OS (latching overflow) is set (OS = 1).
JR -32768 to +32767	Jump via the system software; carried out under all conditions.

addr = symbolic address (a maximum of 4 characters)

● Shift operations

Operation	Description
SLW 0 to 15	Shifting to the left (zeros are filled in from the right).
SRW 0 to 15	Shifting to the right (zeros are filled in from the left).
SLD 0 to 32	Shifting a doubleword to the left (zeros are filled in from the right).
SSW 0 to 15	Shifting to the right with sign.
SSD 0 to 32	Shifting a doubleword to the right with sign (sign is filled in from the left).
RLD 0 to 32	Rotating to the left.
RRD 0 to 32	Rotating to the right.

With the shift functions only accu 1 is used. The parameter part of the commands specifies up to how many positions the accu contents are shifted or rotated. With SLW, SRW and SSW, only the lower order word is involved with the shift functions, with SLD, SSD, RLD and RRD the entire contents of accu 1 (32 bits) are used. Shift functions are carried out unconditionally.

The last bit shifted out can be scanned by means of jump functions. The DSP0 and DSP1 condition codes are affected.

With JZ, a jump can be carried out if the bit is 0. With JN, a jump can be carried out if the bit is 1.

DSP1	DSP0	Shift: last bit shifted
0	0	0
0	1	-
1	0	1

Examples

STEP5 program: contents of data words:

```
:L DW52      KH = 14AF
:SLW 4
:T DW53      KH = 4AF0
```

STEP5 program: contents of accumulator 1 (hexadecimal)

```
:L EDO      2348 ABCD
:SLW 4      2348 BCDO
:SRW 4      2348 OBCD
:SLD 4      3480 BCDO
:SSW 4      3480 FBCE
:SSD 4      0348 OFBC
:RLD 4      3480 FBCE
:RRD 4      0348 OFBC
:BE
```

Applications:

multiplication with the power 2,
e.g. new value = old value x 8

```
:L FW10
:SLW 3
:T FW10      Caution: Do not exceed the
              positive area limit!
```

division by the power 2,
e.g. new value = old value : 4

```
:C DB5
:L DW0
:SRW 2
:T DW0
```

● Conversion operations

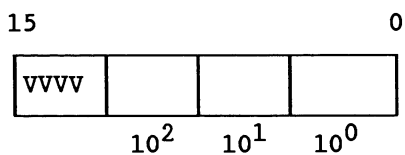
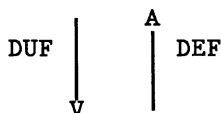
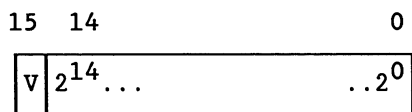
Operation	Meaning
CFW	Forming of one's complement of accu 1 (16 bit)
CSW	Forming of two's complement of accu 1 (16 bit)
CSD	Forming of two's complement of accu 1 (32 bit)
DEF	Fixed point conversion (16 bit) from BCD to binary
DUF	Fixed point conversion (16 bit) from binary to BCD
DED	Doubleword conversion (32 bit) from BCD to binary
DUD	Doubleword conversion (32 bit) from binary to BCD
FDG	Conversion of a fixed point number (32 bit) to a floating point number (32 bit): see OB 220; sign extension
GFD	Conversion of a floating point number to a fixed point number (32 bit)

DEF:

The value contained in accu 1-L (bit 0 through bit 15) is interpreted as a BCD-coded number. After the conversion a 16-bit fixed point number will be contained in accu 1-L.

DUF:

The value contained in accu 1-L (bit 0 through bit 15) is interpreted as a 16-bit fixed point number. After the conversion a BCD-coded number will be contained in accu 1-L.



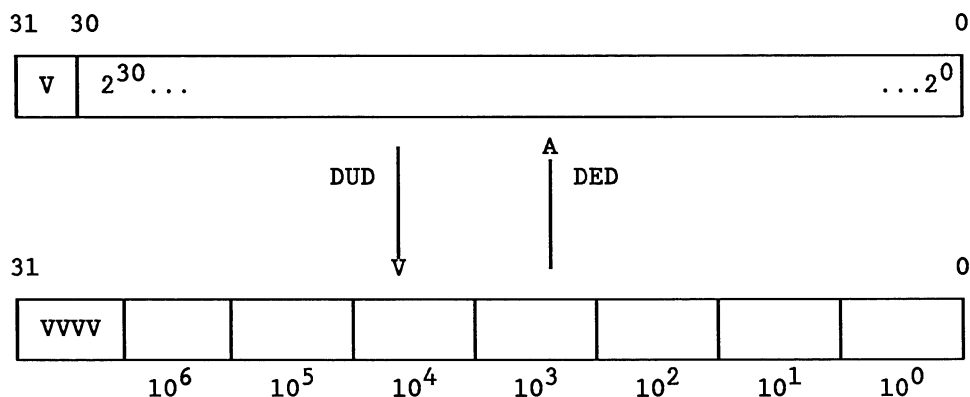
V (sign): 0 = positive
1 = negative

DED:

The value contained in accu 1 (bit 0 through bit 31) is interpreted as a BCD-coded number. After the conversion a 32-bit fixed point number will be contained in accu 1.

DUD:

The value contained in accu 1 (bit 0 through bit 31) is interpreted as a 32-bit fixed point number. After the conversion a BCD-coded number will be contained in accu 1.



V (sign): 0 = positive
 1 = negative

FDG:

The value contained in accu 1 (bit 0 through bit 31) is interpreted as a 32-bit fixed point number. After the conversion a floating point number (exponent and mantissa) will be contained in accu 1.

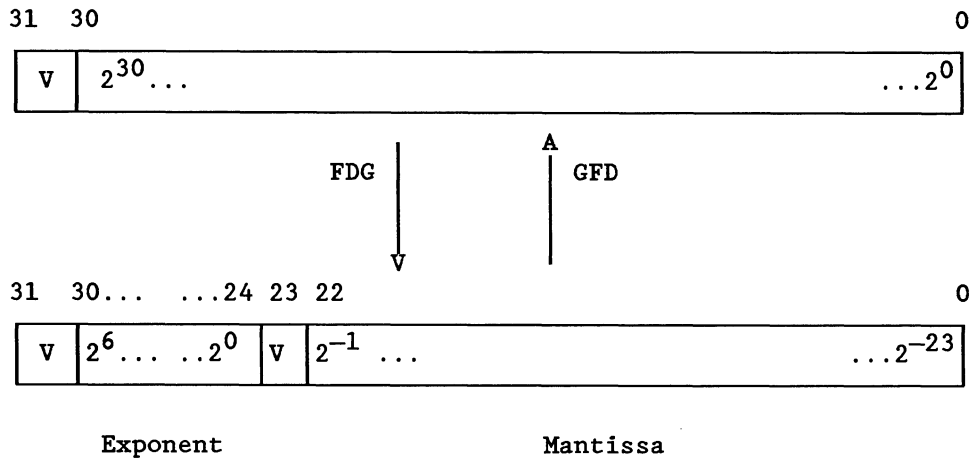
GFD:

The value contained in accu 1 (bit 0 through bit 31) is interpreted as a floating point number. After the conversion a 32-bit fixed point number will be contained in accu 1.

In this case, floating point numbers ≥ 0 or ≤ -1 are rounded down, if necessary, to the next smaller integer.

Floating point numbers < 0 and > -1 are rounded up to 0.

Examples: +5.7 --> 5
 -2.3 --> -3
 -0.6 --> 0
 +0.9 --> 0



CFW, CSW:

Examples

The contents of data word 64 are to be inverted bit by bit ('reversed') and deposited in data word 78.

STEP5 program: assignment of data words:

```

:L   DW64               KM = 0011111001011011
:CFW
:T   DW78               KM = 1100000110100100
    
```

The contents of data word 207 are to be interpreted as a fixed point number and deposited in data word 51 with the sign inverted.

STEP5 program assignment of data words:

```

:L   DW207              KF = + 51
:CSW
:T   DW51               KF = - 51
    
```

● **Decrementing/incrementing**

Operation	Description
D 1 to 255	decrement
I 1 to 255	increment
 parameter	

B8576633-01


The contents of accumulator 1 are decremented or incremented by the number stated as a parameter. The execution of this operation is independent of conditions. It is limited to the right-hand byte (without carry).

Example

STEP5 program: assignment of data words:

:L	DW7	KH = 1010
:I	16	
:T	DW8	KH = 1020
:D	33	
:T	DW9	KH = 10FF

● Processing operations

Operation	Description
DO DW 0 to 255 (operation)	process data word The following operation is combined and executed using the parameter stated in the data word.
DO FW 0 to 254 (operation)	process flag word The following operation is combined and executed using the parameter stated in the flag.
DO =  insert formal operands	process formal operands (parameter class: B): Only C DB, JU PB, JU FB, JU SB may be substituted.
DI 1) 2)	process via a formal operand (indirect) The number of the formal operand to be executed is contained in accumulator 1.
DO RS 1) 2) 60 through 63	instruction contained in the area of the system data (RS) is to be executed (free system data: RS 60 through 63).

- 1) System function
- 2) The value contained in the system data or in the formal operand is interpreted as the operation code of a STEP5 operation which will then be executed. Permissible operations as for DO FW and DO DW.

Combination of all operations is permissible with DO DW and DO FW, except the following:

- all two-word and three-word commands, see annex D, (permissible are G DB, GX DX, SED, SEE, CX DX, DO FX, and DOC FX)
- operations with formal operands in function blocks,
- JU/JU OB, JU/JC PB, FB, ...

The PG will not verify that the combination is permissible.

DO FW 14
L IB 0

120

—————> L IB 120 (= command executed)

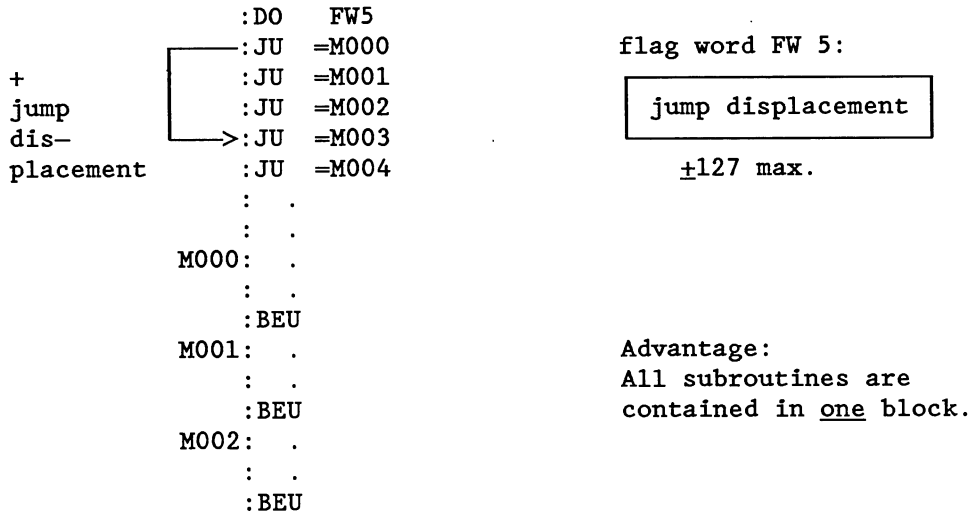
Example (process data word)

The contents of data words DW 20 through DW 100 are to be set to signal status "0". The index register for the parameter of the data words is DW 1.

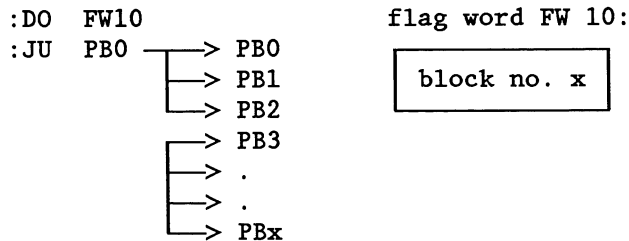
```

:L   KF 20      supply the index register
:T   DW1
M001 :L   KB 0      reset
      :DO  DW1
      :T   DW0
      :L   DW1      increment index register
      :L   KF1
      :+F
      :T   DW1
      :L   KF 100
      :<=F
      :JC  =M001    jump, if index is in the area
      ...          Further STEP5 program
    
```

Application: Jump distributor for subroutine method



Application: Jump distributor for block calls



- **Disable/enable process interrupts**

IA	disable interrupt driven processing
RA	enable interrupt driven processing

Use of "disable/enable interrupts" is possible, e.g. if interrupt driven processing is to be suppressed during time driven processing. This means that interrupt driven processing will no longer be possible in the program section between the IA and RA commands.

Also refer to Subsection 6.8.1, special function OB 120 "disable interrupts".

- **Other operations**

Operation	Parameter	Description
G DB	3 to 255	generation of a DB data block in the DB-RAM
GX DX	1 to 255	generation of a DX data block in the DB-RAM

G DB: Create data block

Command G DBxxx generates a DB data block with number xxx (between 3 and 255) in the internal data block RAM of the processor.

Before programming the instruction you must enter the number of data words which the new DB is to have in accumulator 1-L. The block header is generated by the G DB/GX DX instruction. One data block (including its header) must not occupy more than 4091 words in memory.

The system program will call **OB 31** if the data block exists already, if the length of the data block is illegal or if the space in the DB RAM is insufficient. The processor will stop due to an execution time error if OB 31 has not been programmed. The error identifiers are in accumulator 1.

Command GX DXxxx generates a DX data block in the DB RAM and is otherwise the same as G DBxxx (permissible parameters: 1 to 255).

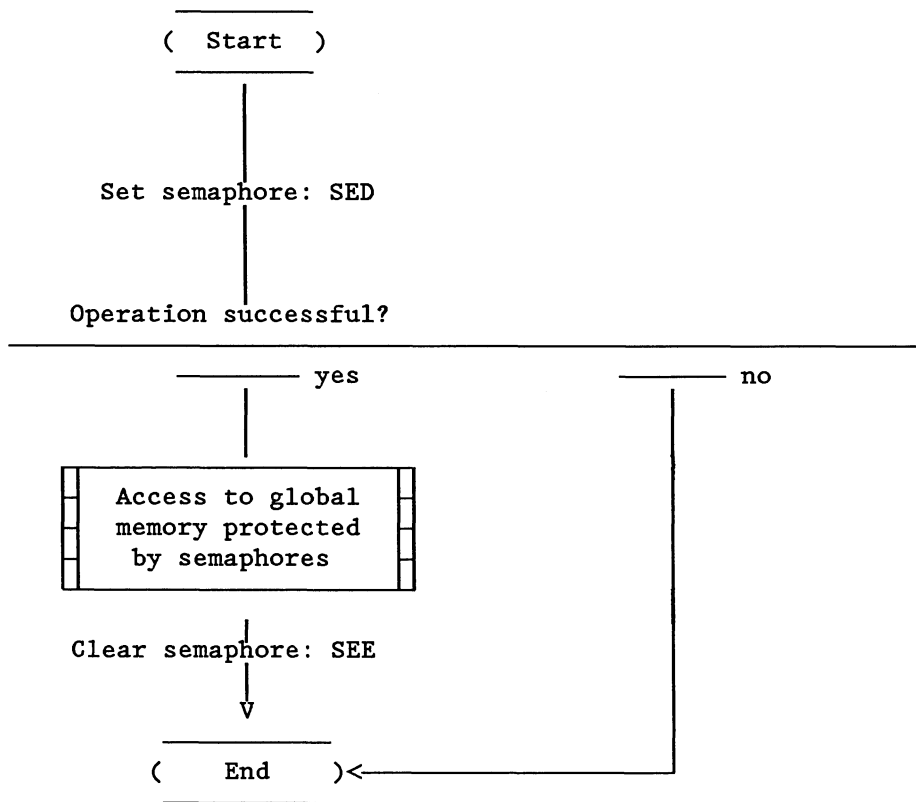
Operation	Parameter	Description
SED	0 to 31	setting a semaphore
SEE	0 to 31	clearing a semaphore

SED/SEE: Setting/clearing a semaphore

If two or more processors of a programmable controller use certain global memory areas (I/O's, CPs, IPs) together there is a danger of one processor overwriting the data of the other or invalid intermediate statuses of data being read out. Due to this coordination of the access of the individual processors to the common memory areas is required.

Coordination of the individual processors is possible by means of semaphores and the SED and SEE commands: Before the semaphores declared (SED) have been set the processors involved in multiprocessor operation will not access the common memory area. A semaphore xx can only be set by one single processor. If a processor fails to set the semaphore, access will be denied. Further access will also be denied if the processor has again cleared the semaphore (SEE).

All of the processors involved must contain a function block with the following program structure:



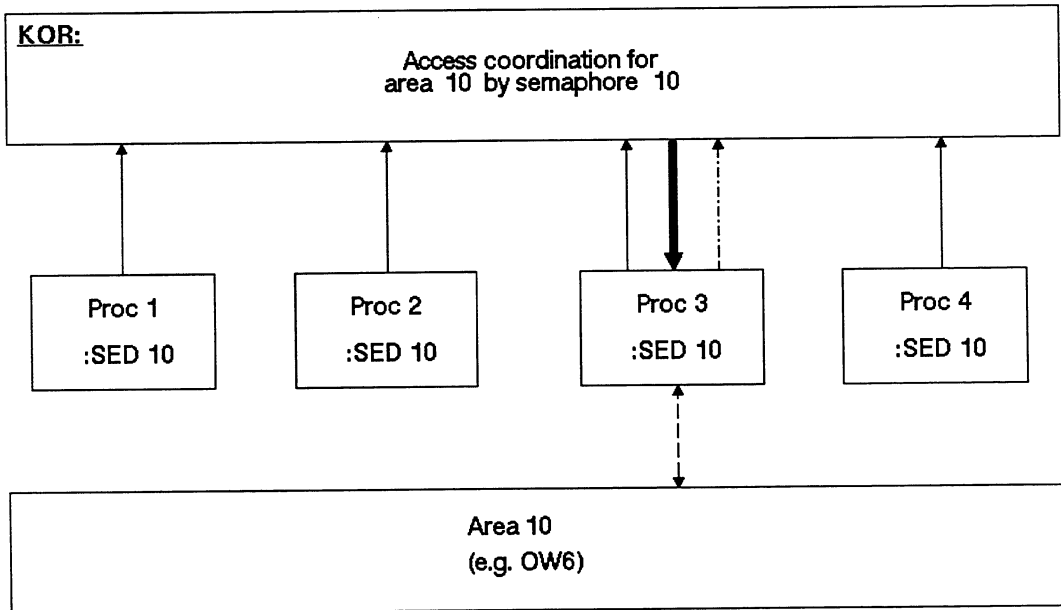
The use of SED and SEE instructions ensures that a processor can transfer information that belongs together to/from a certain memory area and is protected from another processor interrupting this operation.

IMPORTANT!

The instructions **SED xx** and **SEE xx** must be used by all processors requiring synchronized access to a common global memory area (addresses > F000H).

The SED xx (set semaphore) command occupies a certain byte in the coordinator for the processor executing the command (that is, if this byte has not been occupied by another processor). As long as the processor is entered there the remaining processors are denied reading or writing access to the memory area protected by the corresponding semaphore (numbers 0 to 31). This means that the area is blocked for all other processors.

The SEE xx (clear semaphore) instruction resets the byte in the coordinator. This means that the memory area protected is again available for the other processors, reading or writing is again possible. Only the processor responsible for setting of the semaphore can clear it.



- SED 10 Request of right to access area 10
- Access rights from KOR for processor 3
- - - - -→ Data access to area 10
-→ SEE 10 Access rights are returned

The SED and SEE commands scan the status of a semaphore before it is set or cleared. The DSP0 and DSP1 indications are affected in the following manner:

DSP1	DSP0	Significance	Evaluation
0	0	Semaphore was set by another processor and setting/clearing is not possible	JZ
1	0	Semaphore is set/cleared	JN, JP

IMPORTANT!

The process of scanning a semaphore (=reading) and the setting or clearing the semaphore (=writing) is one unit. During these processes access to the semaphore will be denied to all other processors!

Please note the points below when using the semaphores:

- A semaphore is a global variable, i.e the semaphore with number 16 only exists **once**, even if e.g. three processors are used.
- Use of the SED and SEE commands is required for **all** processors whose access to a common memory area is to be coordinated.
- All processors involved will have to run the **same** start-up mode. All semaphores will be erased if a cold restart is carried out, however, they will be retained if a manual or automatic restart is carried out.
- Start-up in multiprocessor operation must be synchronized. Due to this, the test mode is **not** permissible.

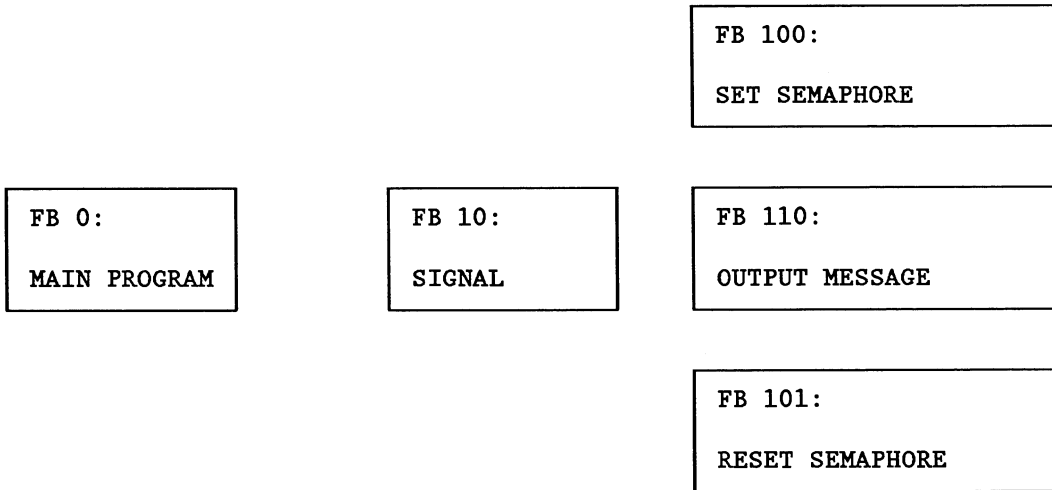
Example for semaphore application

In one S5 135U there are 4 processors which output status messages to a signalling unit via a common memory area of the 0 I/O's (OW 6). Each of the status messages has to be output for 10 seconds, the same or another processor cannot overwrite the initial message with a new message until the 10 seconds have elapsed.

Use of the I/O word OW 6 (extended I/O, no process image) is controlled by means of a semaphore. This means that only the processor which reserved this area with a semaphore can write its message in OW6. The semaphore will remain set for 10 seconds (TIMER T10). When the timer reaches "0" the processor will clear the semaphore and thus release the area for use by other processors. OW 6 may now receive a new message.

If a processor attempts to set a semaphore and this semaphore has already been set by another processor then, during the next cycle, the processor will again attempt to set the semaphore and output its message.

Execution of the following program is possible with a different message for each of the four processors. The following blocks are loaded:



5 flags are used:

- F 10.0 = 1: A message has been requested or is being processed.
- F 10.1 = 1: The semaphore has been set.
- F 10.2 = 1: The timer has been started.
- F 10.3 = 1: The message has been transferred.
- F 10.4 = 1: The semaphore has been reset.

B8576633-01

FBO

NAME :MAIN

```
:A F 10.0          if no message is active
:JC =M001
:
:AN E 0.0
:BEC
:
:L KH2222          create message and
:T FW12
:AN F 10.0
:S F 10.0          set flag 'MESSAGE'
:
MO01 :JU FB10      call FB 'SIGNAL'
NAME :SIGNAL
:
:BE
```

FB 10

NAME :SIGNAL

```
:AN F 10.1          if no semaphore is set,
:JC FB100           call FB 'set semaphore'
NAME :SEMASET
:
:A F 10.1           if semaphore is set
:AN F 10.2          and timer has not been started
:S F 10.2
:L KT10.2          start timer
:SE T 10
:
:A F 10.2           if timer has been started
:AN F 10.3          and no message is transferred,
:JC FB110          call FB 'output message'
NAME :MESS.OUT
:
:A F 10.2           if timer has been started
:AN F 10.4          and semaphore is not reset
:AN T 10           and timer has elapsed
:JC FB101          call FB 'reset semaphore'
NAME :SEMARESE
:
:AN F 10.4          if semaphore has been reset,
:BEC
:
:L KH0000
:T FY10            reset all flags
:BE
```

B8576633-01

FB100

NAME :SEMASET

```
:SED 10          set semaphore no. 10
:JZ =M001
:AN F 10.1       if semaphore has been set,
:S F 10.1        set flag 'SEMA SET'
M001 :BE
```

FB110

NAME :MESS.OUT

```
:L FW12          transfer message to
:T OW6           I/O's
:AN F 10.3
:S F 10.3        set flag 'TRANSFER MESSAGE'
:BE
```

FB101

NAME :SEMARESE

```
:SEE 10          clear semaphore no. 10
:JZ =M001
:AN F 10.4
:S F 10.4        set flag 'SEMAPHORE RESET'
M001 :BE
```

4 Operating States

4.1 Operating States and Program Levels

The processor has three operating states:

operating state STOP

operating state START-UP

operating state RUN

As described in chapter 2, the system program calls the appropriate organization blocks (OB 1 through OB 34) if certain events occur. In these blocks the user may determine the further reaction of the processor. If, for example, an acknowledgement delay occurs while the process image is being updated in the RUN state, the system program calls OB 24.

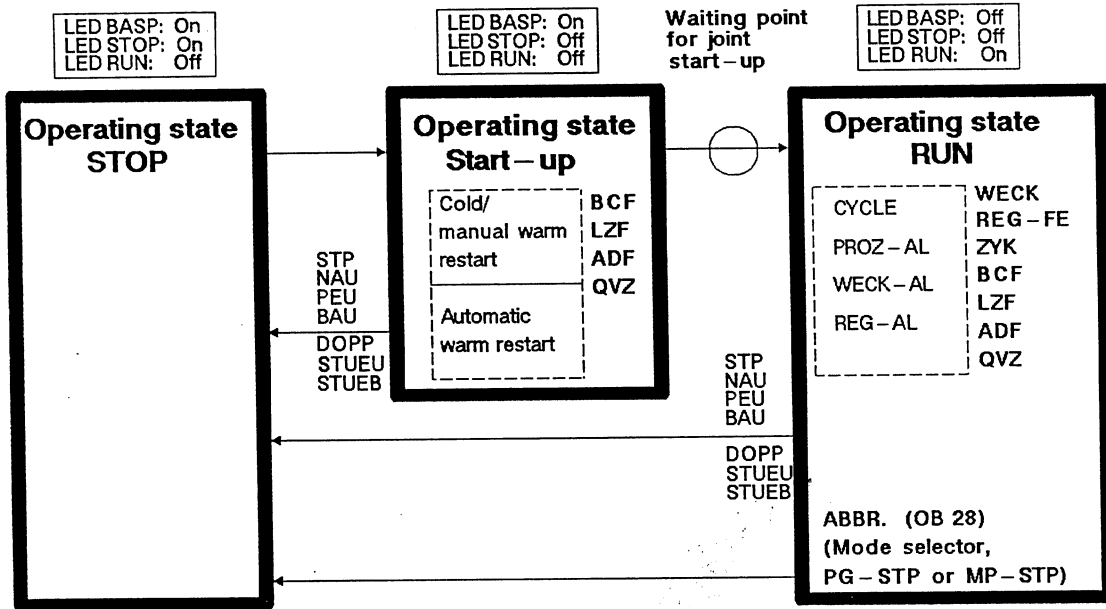
Some events are only possible in the operating state START-UP, some only in the operating state RUN and some in START-UP as well as RUN (see following page).

After an organization block has been called, the processor will execute the STEP5 user program contained in it. In doing so, a new register record is created (register: accumulator 1 through 4, block stack pointer, data block start address, STEP-address counter). If 'normal' program execution has been interrupted due to the occurrence of the event, then the processor will, after having processed the OB - including all of the blocks nested in it - continue the program at the breakpoint. There, the "old" register contents are again valid.

One or several of these organization blocks are each assigned a program level: if, e.g. OB 2 is called, this means that the program level 'PROCESS INTERRUPT' is activated. The program level 'COMMAND CODE ERROR' is activated if OB 27, OB 29 or OB 30 are called.

The following page provides a summary of the operating states and the program levels in the S5 135U, CPU 928.

Summary 4-1: Operating states and program levels:



Program levels in START-UP:

- COLD/MANUAL WARM RESTART
- AUTOMATIC WARM RESTART
- BCF** (command code error)
- LZF** (execution time error)
- ADF** (addressing error)
- QVZ** (acknowledgement delay)

Program levels in RUN:

- CYCLE** (cyclic program processing)
- TIME INTERRUPT (WECK-AL) 5sec (time-driven program execution)
- TIME INTERRUPT (WECK-AL) 2sec (time-driven program execution)
- TIME INTERRUPT (WECK-AL) 1sec (time-driven program execution)
- TIME INTERRUPT (WECK-AL) 500ms (time-driven program execution)
- TIME INTERRUPT (WECK-AL) 200ms (time-driven program execution)
- TIME INTERRUPT (WECK-AL) 100ms (time-driven program execution)
- TIME INTERRUPT (WECK-AL) 50ms (time-driven program execution)
- TIME INTERRUPT (WECK-AL) 20ms (time-driven program execution)
- TIME INTERRUPT (WECK-AL) 10ms (time-driven program execution)
- CONTROLLER INTERRUPT (REG-AL) (time-driven controller processing)
- PROCESS INTERRUPT (PROZ-AL) (interrupt-driven program execution)
- WECK** (collision of two time interrupts)
- REG** (controller error)
- ZYK** (cycle time error)
- BCF** (command code error)
- LZF** (execution time error)
- ADF** (addressing error)
- QVZ** (acknowledgement delay)
- ABBR** (abort)

A program level is further characterized by the following features:

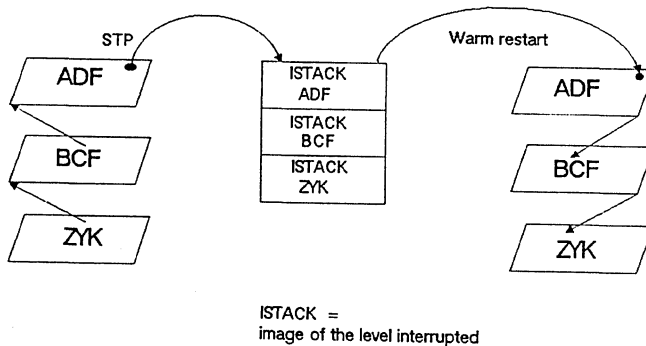
- Every program level has its own specific system program.

Example:

The system program updates the process image of the inputs and outputs, triggers the cycle time and calls the management of the PG-interface (system checkpoint) in the processing level CYCLE.

- If an interrupt occurs, the system program will create an interrupt stack (ISTACK) of its own and will enter, among other things, the level interrupted (see LEVEL) for every program level.

Example:



- The program levels have a fixed priority. Depending on this priority, they can interrupt each other and be nested in each other.

The "start-up levels and the error levels" differ from the "basic levels" insofar as they are nested *immediately* and at *command boundaries* as soon as the relevant event occurs. They may be nested into the basic levels as well as into each other. When errors occur, the last error that has occurred will have the highest priority.

In contrast to this, nesting of a "basic level" in one of a lower priority is only possible at the *block boundaries* unless this presetting has been changed by specific programming of DX 0 (see Chapter 7).

Priority of the "basic levels":

CYCLE	
TIME INTERRUPTS 1)	↓
CONTROLLER INTERRUPT	
PROCESS INTERRUPT	↑ ascending priority

1) The individual time interrupts are also assigned different priorities. Shorter time interrupts have a higher priority than longer ones.

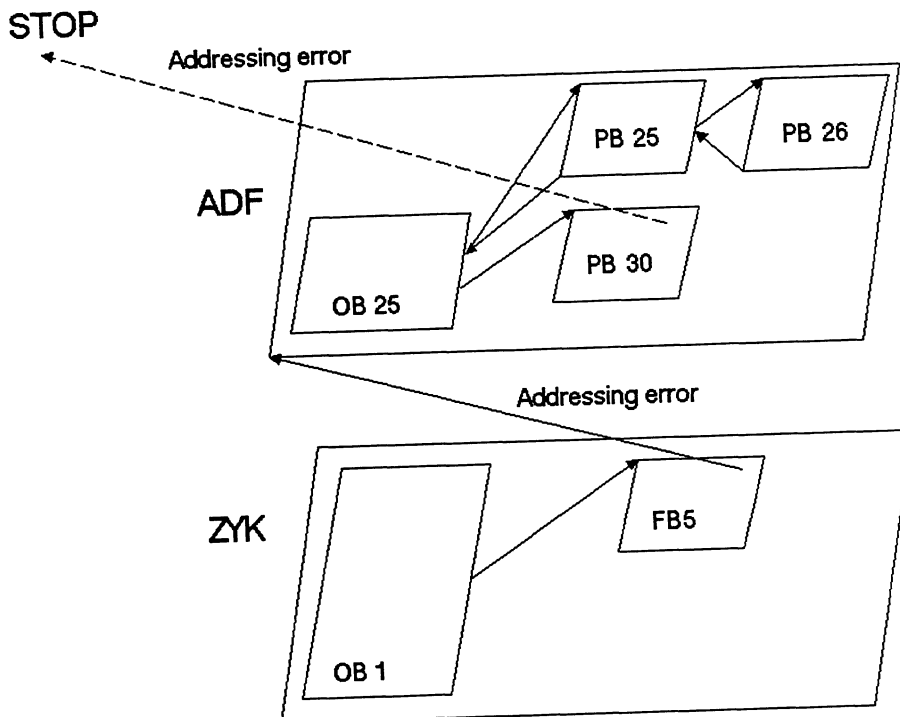
Example:

A process interrupt occurs while a time interrupt is being processed. Since the process interrupt has a higher priority, the processing of the time interrupt level is interrupted at the next block boundary and the PROCESS INTERRUPT program level is nested in. If, e.g., an incorrect address occurs while the process interrupt is being processed, the process interrupt is interrupted immediately at the next command limit in order to nest in the ADF-level.

- Once an error level (ADF, BCF, LZP, OVZ, REG, ZYK) is activated which is still not completely processed, it cannot be reactivated even if another program level has been nested in between. **In this case the PC will immediately pass into the STOP state because a program level has been called twice (in the ISTACK: 'DOPP')** (for an exception of this rule see time interrupts). In depth '01' of the ISTACK, the identification 'DOPP' as well as the error level that has been called twice are marked.

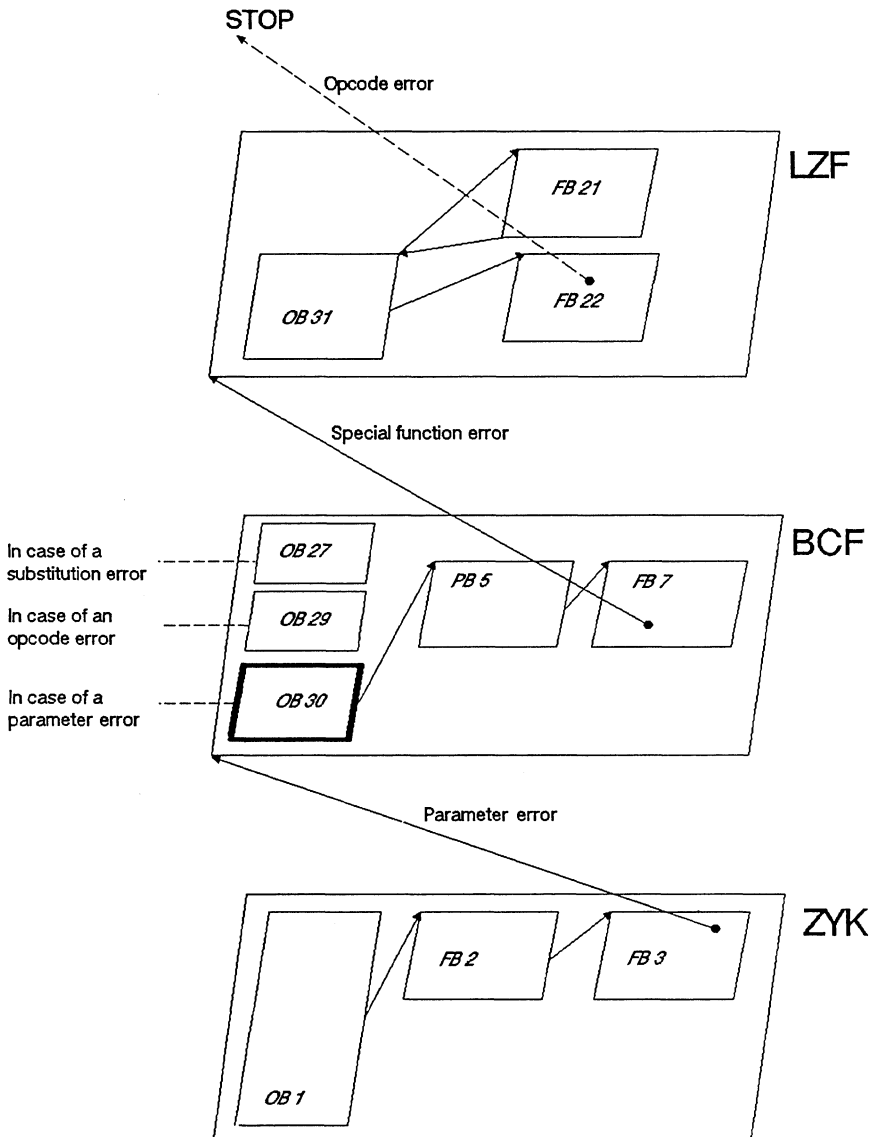
Example 1:

Another addressing error occurs while the ADF-level (user-interface OB 25) is being processed. Since the ADF-level is still activated, it cannot be called a second time: the processor stops.



Example 2:

When an opcode error occurs at the LZF-program level, the system program attempts to call the BCF-level (user-interface OB 29). However, this level has already been activated due to a parameter error (user-interface OB 30) and processing has not been completed yet. The BCF level cannot be called a second time in this situation. The processor stops.



The individual program levels with the corresponding user interfaces are described in detail in the following chapters:

Chapter 4.3 describes the "basic levels" in START-UP.

Chapter 4.4 describes the "basic levels" in RUN.

Chapters 5.5 and 5.6 describe the "error levels" in START-UP and RUN.

4.2 Operating State STOP

The operating state STOP is characterized by the following features:

- No user program is executed.
- If a program has been run, the values of counters, times, flags and process images are retained when the transition is made to the stop state.
- The BASP signal (command output inhibit) is output. This means that all digital outputs are inhibited.

Exception: The BASP signal is not output in the test mode!
(Refer to Chapter 10.5 for the test mode.)

- If a program has been run, an interrupt stack (ISTACK) is present in the stop state.

LED's on the front panel of the central processing unit:

RUN-LED:	off
STOP-LED:	on (permanent or flashing)
BASP-LED:	on (except in the test mode)

The STOP-LED may indicate the reasons for the present stop page. A permanently lit or flashing STOP-LED has a special significance which will be described briefly on the following pages.

The STOP-LED is lit permanently:

The operating state STOP was triggered

in single processor operation:

- a) by setting the mode selector from 'RUN' to 'STOP'
- b) with online function 'STOP'
- c) because of hardware faults (BAU, PEU, NAU)
- d) after an overall reset

in multiprocessor operation:

- a) when setting the mode selector on the coordinator to 'STOP'
- b) another processor has stopped because of an error (the processor not responsible for the error displays a permanently lit LED)

- c) with online function 'STOP' on one processor
- d) with online function 'PROCESSING CONTROL END' on another processor

The STOP-LED flashes slowly: (approx. once every 2 seconds)

A slowly flashing STOP-LED usually signals an error.

In multiprocessor operation a slowly flashing LED identifies the processor responsible for the stoppage (due to an error).

The STOP-LED flashes slowly:

- a) when programming a stop command in the user program
- b) in the case of operator error (DBI errors, selection of an inadmissible start-up mode etc.)
- c) in the case of programming errors or hardware faults (calling a block that has not been loaded, addressing errors, acknowledgement delay, command code error etc.);
in order to provide additional information about the cause of the error the following LED's will light up:
 - ADF-LED
 - QVZ-LED
 - ZYK-LED
- d) with online function 'PROCESSING CONTROL END' for this processor

The STOP-LED flashes quickly: (approx. twice per second)

A quickly flashing STOP-LED signals the warning message:
"Overall reset has been requested"!

Request overall reset

- a) The *system* requests overall reset:

The processor carries out an initialization routine every time the power is switched on and after an overall reset has been executed. If errors are detected during the initialization the processor goes over to the stop state and the LED flashes quickly.

- Possible errors:
- 1. RAM's are empty
Remedy: Overall reset of processor
 - 2. User EPROM empty or not plugged in
Remedy: plug in programmed EPROM and execute overall reset of processor

The cause of the problem will have to be eliminated and then an overall reset of the processor carried out (again).

b) The user requests overall reset:

The operator must take the following action to request an overall reset:

- Switch the mode selector from 'RUN' to 'STOP'.

Result: The processor is in the stop state.
The STOP-LED is permanently lit.

- Hold the selector switch in the 'OVERALL RESET' position; at the same time, turn the mode selector from 'STOP' to 'RUN' and back to 'STOP'.

Result: Overall reset is requested.
The STOP-LED flashes quickly.

IMPORTANT!

If the requested overall reset is not to be executed select a start-up mode.

Execute overall reset

Irrespective of the origin of the overall reset request, whether system or user, the overall reset is carried out as follows:

- Hold the selector switch in the 'OVERALL RESET' position; at the same time, switch the mode selector from 'STOP' to 'RUN' and back to 'STOP'.

Result: Overall reset is executed.
The STOP-LED is permanently lit.

- Or: By means of the online function 'ERASE'.
(When executing an overall reset with the PG, a manual reset request using the switch is not required! The position of the selector switch and the mode selector is not relevant.)

Result: Overall reset is executed.
The STOP-LED is permanently lit.

IMPORTANT!

If an overall reset has been carried out the only permissible start-up mode is a cold restart!

Leaving from the stop state:

- selection of a start-up mode (see Chapter 4.3)
- overall reset, then cold restart
- test mode (for multiprocessor operation, see Chapter 10.5)

4.3 Operating State START-UP

The operating state START-UP is characterized by the following features:

- The START-UP is the transition from the operating state STOP to the operating state RUN.
- The operator can select one of three different start-up modes: cold restart, manual warm restart and automatic warm restart. The cyclic user program is processed from the very beginning if a cold restart is carried out. If a manual or automatic warm restart is carried out, then the cyclic user program will be continued at the breakpoint.
- For all three types of start-up mode the system calls a different organization block in which the user can write a suitable start-up program. The length of the STEP5 start-up program in the OBs is not limited. There is no monitoring with respect to time. Calling of further blocks is possible in the start-up OB's.
- The values for counters, timers, flags and process images are handled differently in each start-up mode.
- The BASP signal (command output inhibit) is output. This means that all digital outputs are inhibited.
Exception: BASP is not output in the test mode!
(see Chapter 10.5 for the test mode)

LEDs on the front panel of the CPU:

RUN-LED: off
STOP-LED: off
BASP-LED: on (not in the test mode)

Remarks:

Refer to Chapter 10.4 for information about the "Start-up routine for multiprocessor operation".

This is how a cold restart is triggered:

- Hold the selector switch in the 'RESET' position; at the same time switch the mode selector from 'STOP' to 'RUN'.
- Or: By means of online function 'START' (--> cold restart).

A cold restart is required after

- stack overflow (ISTACK, 'STUEU, STUEB')
- double call of a program level (ISTACK: 'DOPP')
- overall reset (control bits: 'URGELOE')
- start-up abort (control bits: 'ANL-ABB')
- stop after online function "PROCESSING CONTROL END"

A cold restart is **always permissible** unless the system has requested "overall reset"!

This is how a manual warm restart is triggered:

- The selector switch is in the central position.
- Switch the mode selector from 'STOP' to 'RUN'.
- Or: With online function 'START' (--> manual warm restart)

A manual warm restart is always permissible, except in situations where only a cold restart is permitted (see above) or where the system has requested "overall reset".

Triggering the automatic warm restart:

The processor executes an initialization routine after power failure/power-off during START-UP or RUN followed by power recovery/power-on. It then automatically attempts to carry out a warm restart.

- Conditions:
- The switches on all processors and on the coordinator remain unchanged on 'RUN'.
 - No errors have occurred during the initialization.

If your processor is to carry out an *automatic cold restart* after a power failure and a following power recovery, you should change the presetting by programming the data block DX 0.

IMPORTANT!

A manual or automatic warm restart is only permissible if the user program has not been altered in the stop state.

4.3.1 Cold Restart and Manual Warm Restart

The following table contains a comparison between the start-up modes cold restart and manual warm restart.

Sequence of events	Cold restart	Manual warm restart
Triggering: ↓ ↓ ↓	stop switch from STOP position to RUN position and selector switch in RESET position online function START (cold restart) if "automatic cold restart" programmed in DX 0, switch on power.	stop switch from STOP position to RUN position and selector switch in central position online function START (manual warm restart)
System program functions: ↓ ↓ ↓ ↓ ↓ ↓	<ul style="list-style-type: none"> - block address list in DB 0 retained - erase process image of inputs - erase process image of outputs - erase flags, timers, counters - erase digital/analog I/O's (2x128 bytes each) - erase global interprocessor communication flags (256 bytes) - erase semaphores (all 32) - DB1 present: enter digital input/output and interprocessor communication flags input/output in PII/PIO lists - DB1 not present: enter actually existing modules (only digital input/output) in PII/PIO lists (interprocessor communic. flags are ignored) - evaluate controller parameter assign. blocks DB 2 	<ul style="list-style-type: none"> - block address list in DB 0 retained - flags, timers, counters retained - global interprocessor communication flags retained - semaphores retained
	<ul style="list-style-type: none"> - call user interface OB 20 (if present) 	<ul style="list-style-type: none"> - call user interface OB 21 (if present)
	<ul style="list-style-type: none"> - synchronize start-up for multiprocessor operation 	<ul style="list-style-type: none"> - synchronize start-up for multiprocessor operation

Cold restart: Programming the organization block OB 20

If the processor executes a cold restart OB 20 will be called automatically. You can write a STEP5 program in this OB which executes certain activities once before the cyclic program starts:

Such activities may include e.g.

- set flags
- start timers
- set outputs with direct peripheral access
- prepare the data exchange between the processor and the I/O modules
- carry out the synchronization with CP's
(see Chapter 6.9 for handling block SYNCHRON).

Complete OB 20 with 'BE' (block end)!

The cyclic program starts after OB 20 has been processed by calling OB 1 or FB 0.

If OB 20 is not programmed the processor immediately starts the cyclic program at the end of a cold restart (after the system functions).

If you have programmed an "automatic cold restart following power failure" in DX 0, OB 20 will also be called when the power returns.

Manual warm restart: Programming organization block OB 21

If the processor carries out a manual warm restart OB 21 will be called. You can write a STEP5 program in this OB which executes certain activities once before the cyclic program starts again.

Complete OB 21 with 'BE' (block end)!

After OB 21 has been processed the cyclic program is continued at the breakpoint with the next instruction:

- The BASP signal (command output inhibit) will remain in effect during the processing of the remaining cycle and will not be cancelled until the next (complete) cycle is started.
- The process image of the outputs remains reset at the end of the remaining cycle and will be updated only at the end of the next (complete) cycle.

If OB 21 is not programmed the processor will immediately start at the breakpoint following a manual warm restart.

4.3.2 Automatic Warm Restart

The *function* of the automatic warm restart is identical with that of the manual warm restart. The only difference is the way it is triggered.

Sequence of events	Automatic warm restart
Triggering ↓	Power return after power failure
System program functions: ↓	- block address lists retained in DB 0 - flags, timers, counters retained - interprocessor communication flags retained - semaphores retained
↓	- call user interface OB 22 (if present)
↓	- synchronize start-up for multiprocessor operation

Automatic warm restart: Programming organization block OB 22

The processor attempts to continue the interrupted program as soon as the power returns.

The first step is to call OB 22.

You can write a STEP5 program in this OB which executes certain activities once, before the cyclic program is started again.

If you wish to prevent the processor from ever carrying out an automatic warm restart then you will have to program a stop instruction in OB 22 and complete the block with 'BE' (block end)!

```

OB 22 : STP          (stop)
      : BE           (block end)
    
```

Result: The processor will go over to the stop state as soon as the power returns.

After OB 22 has been processed the cyclic program will be continued at the breakpoint with the next instruction. When the power returns following a power failure:

- The BASP signal (command output inhibit) will be retained during the processing of the remaining cycle. It will not be cancelled until the beginning of the next (complete) cycle.
- The process image of the outputs remains reset at the end of the remaining cycle and will be updated only at the end of the next (complete) cycle.

4.3.3 Interruption during START-UP

A start-up program can be interrupted by

- power failure
- mode selector to stop
- program error and hardware faults (see Chapter 5.6)

If the start-up that has been interrupted is to be continued using one of the three possible start-up modes, the following points should be noted:

If there is a **power failure during the start-up** followed by the return of power, three different situations may arise:

1. The processor is executing a cold restart (OB 20). When the power returns the organization block OB 22 (automatic warm restart) is nested in OB 20 at the breakpoint.
2. The processor is executing a manual warm restart (OB 21). When the power returns the organization block OB 22 (automatic warm restart) is nested in OB 21 at the breakpoint.
3. The processor is already executing an automatic warm restart (OB 22). When the power returns a second OB 22 will not be nested. The OB 22 that has been interrupted will not be continued when the power returns. This block will be aborted and the OB 22 that has just been called will be processed immediately.

Stop by means of the switch during the start-up and subsequent manual warm restart

If you abort any start-up with the stop-switch (or the online function 'STOP') and trigger a manual warm restart afterwards, the start-up that has been interrupted will be continued at the breakpoint and the OB 22 that has just been called will be processed immediately. No OB 21 is nested!

B8576633-01

Stop by means of the switch during the start-up and subsequent cold restart

If you abort any start-up with the stop-switch (or the online function 'STOP') and trigger a cold restart afterwards, the start-up that has been interrupted will be aborted and a cold restart will be carried out (if present, by means of OB 20).

4.4 Operating State RUN

The operating state RUN is characterized by the following features:

- The user program is processed cyclically.
- All counters and timers started in the program "run". The process images are updated cyclically.
- The BASP signal (command output inhibit) is cancelled. This means that the digital outputs are released.
- The interprocessor communication flags are updated cyclically.

LEDs on the front panel of the GPU:

RUN-LED: on

STOP-LED: off

BASP-LED: off

IMPORTANT!

If a manual or automatic warm restart has been carried out before the transition to the operating state RUN, the BASP-LED will remain lit until the remaining cycle has been processed and the process image updated.

IMPORTANT!

Reaching the operating state 'RUN' is only possible following the operating state 'START-UP'.

12 basic program levels exist in the operating condition RUN:

1. the cycle: (Zyklus) The user program is processed cyclically.
2. 9 time interrupts: (Weckalarm) The user program is processed time-driven.
3. the controller interrupt: (Regleralarm) A preset number of controllers is processed time-driven in addition to the user program.
4. the process interrupt: (Prozeßalarm) The user program is processed interrupt-driven.

They differ in the following respects:

- a) They are triggered by different events.
- b) The system program executes different functions for every program level.
- c) A different organization block or function block exists as the user interface for every program level.

4.4.1 CYCLE: Cyclic Program Execution

The **cyclic program execution** is the usual mode with programmable controllers.

Triggering:

If the processor has completed its start-up program without error it will start the cyclic program.

Functions of the system program:

- Sets the cycle time to be monitored at the start of the cycle.
- Updates the process image of the inputs (PII).
- Updates the interprocessor communication input flags.
- Calls the user interface: OB 1 is processed.
- Updates the process image of the outputs at the end of the cycle (PIO).
- Updates the interprocessor communication output flags.

User interface: OB 1 or FB 0

The organization block OB 1 or the function block FB 0 is called as the user interface for cyclic program execution. The STEP5 user program in OB 1 or FB 0 is processed continuously from the very beginning including different block calls. After the execution of the system functions the processor will again start at the very beginning with the first STEP5 instruction in OB 1 (or FB 0).

The program, function and sequence block calls that are to be processed in the cyclic program must be programmed in OB 1.

If you have a short and time-critical program which does not require structured programming then you should program FB 0: Since the complete STEP5 operation set is available you can save the block calls and thus reduce the execution time of the program.

IMPORTANT!

Program either OB 1 or FB 0.

If both OB 1 and FB 0 are programmed, only OB 1 will be called by the system program. If you intend to use FB 0 as the user interface no parameters are permissible in it!

Breakpoints

The cyclic program processing can be interrupted at the *block boundaries*. This is done by means of

- process interrupt driven processing
- controller processing
- time-driven processing

(Interruptions are also possible at the *command boundaries* by programming DX 0!)

The program can be interrupted or aborted at *command boundaries*

- if a programming error or a hardware fault occurs
- by the operator (online function, stop-switch).

Note:

Use of the arithmetic registers accumulator 1, 2, 3 and 4 as a data register is possible across the cycle boundaries - from the end of one program cycle to the start of the next.

4.4.2 TIME INTERRUPT: Time-Driven Program Processing

Time-driven execution means that a time signal (time interrupt) triggered by an "internal clock" causes the processor to interrupt the cyclic program and to process a specific program. After this program has been executed the processor will return to the breakpoint in the cyclic program and will continue the program from there.

This allows certain program sections to be automatically inserted in the cyclic program at preset time intervals.

The CPU 928 allows time-driven processing of up to 9 different programs. Each program is called up in a different time interval.

Triggering:

A time interrupt is triggered automatically at the intervals assigned to it provided that the appropriate OB has been programmed.

User interface: OB 10 to OB 18

When a specific time interrupt occurs, the appropriate organization block is called as the user interface at the next block boundary (or command boundary).

Assignment:	Organization block	Time interval
	OB 10	called every 10 ms
	OB 11	called every 20 ms
	OB 12	called every 50 ms
	OB 13	called every 100 ms
	OB 14	called every 200 ms
	OB 15	called every 500 ms
	OB 16	called every 1 sec
	OB 17	called every 2 sec
	OB 18	called every 5 sec

You should program e.g. in OB 13 the program section which is to be inserted into cyclic program processing at intervals of 100 ms.

You may program either all or any number or none of these 9 OBs. If none of them has been programmed, no time-driven program processing is executed.

Whenever a time interrupt OB (OB 10 to OB 18) is called, accu 1 gives information about how many time intervals have occurred since the last call of this OB. Note the following rule:

$$\text{Accu 1} := \text{Number of time intervals} - 1$$

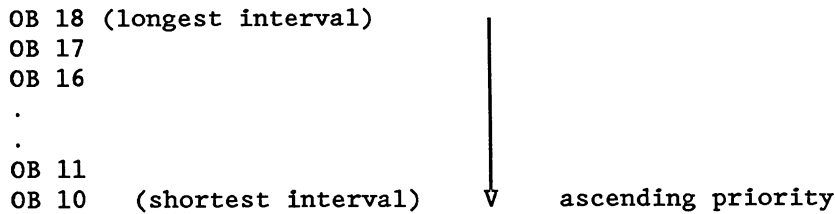
If, for instance, the number "5" is in accu 1 when OB 11 is called, that means that 120 ms (= 6 intervals) have passed since OB 11 has been called last. *As long as there is no collision of two time interrupts, "0" is transferred to accu 1.*

Priority of the time interrupts

Within the basic levels, the program levels TIME INTERRUPTS are arranged as follows:



The priority among the individual time interrupts is also fixed:



IMPORTANT!

Basically, OBs with shorter intervals have a higher priority and may interrupt OBs with longer intervals!

Breakpoints

Time-driven program execution can be interrupted either at the *block boundaries* (preset) or the *command boundaries* (programming DX 0) by means of

- renewed time-driven processing
- controller processing
- process interrupt-driven processing

and only at *command boundaries* by

- a program error or hardware fault.

Example:

OB 12 (intervals of 50 ms) is called while OB 14 (intervals of 200 ms) is processed. OB 14 is interrupted at the next block or command boundary and OB 12 is processed. Only when the latter has completely been processed (possibly interrupted by a controller interrupt, a process interrupt, error processing, or an OB 10 or 11), program execution in OB 14 is resumed and completed.

Collision of two time interrupts

IMPORTANT!

Interrupting of time-driven processing by the *same* time-driven process is not possible!

If a time interrupt OB that has not been completed is called a second time because the time interval has run out, this means that there is a collision of two time interrupts. This error also occurs when an OB is called a second time without the first call being processed. This may happen when "Interrupt time interrupts at block boundaries" has been selected as presetting and your STEP5 program contains large blocks.

When a collision of two time interrupts occurs, the error program level WECKFE is activated and the system program calls **OB 33** as the user interface. You can program the reaction to this condition in OB 33.

When OB 33 is called, a more detailed description of the first error occurred is written by the system program into accu 1 and 2:

Error identifier	Description
Accu 1 Accu 2	
1001H 0016H	Collision of two time interrupts at OB 10 (10ms)
1001H 0014H	Collision of two time interrupts at OB 11 (20ms)
1001H 0012H	Collision of two time interrupts at OB 12 (50ms)
1001H 0010H	Collision of two time interrupts at OB 13 (100ms)
1001H 000EH	Collision of two time interrupts at OB 14 (200ms)
1001H 000CH	Collision of two time interrupts at OB 15 (500ms)
1001H 000AH	Collision of two time interrupts at OB 16 (1sec)
1001H 0008H	Collision of two time interrupts at OB 17 (2sec)
1001H 0006H	Collision of two time interrupts at OB 18 (5sec)

Remark: The identifier contained in accu 2 identifies the level (EBENE) of the time interrupt that has caused the error.

If OB 33 is not programmed the processor will go over to the stop state. This means that 'WECKFE' is marked in the control bits at the programmer with 'output ISTACK', and the level (EBENE) of the corresponding time interrupt indicated in the ISTACK.

If the program is to be continued following a collision of two time interrupts you either have to program the block end statement 'BE' in OB 33 or alter the presetting in DX 0 so that the program should be *continued* if a collision of two time interrupts occurs and OB 33 is not programmed. After OB 33 has been executed the program will be continued from the breakpoint in OB 13.

Remarks:

- With regard to the time-driven program processing, please note the new special functions **OB 120**, **OB 121**, **OB 122** and **OB 123** that enable you to inhibit or delay the processing of time interrupts for a specified program section.
(This is possible either for all time interrupts programmed or for individual ones among them.)

- The 'faster' a time-driven program level is, the bigger is the risk of a collision of two time interrupts: time interrupts with short intervals such as the 10ms and 20ms-time interrupt normally require to be set to *interruption at command boundaries*. Then the controller interrupt and the process interrupt are to be set to interruption at the command boundaries, too (see Chapter 7, programming of DX 0).

4.4.3 CONTROLLER-INTERRUPT: Processing of Controllers

In addition to cyclic, time and interrupt driven program execution closed-loop controllers can also be processed in the CPU 928. The cyclic or time-driven program is interrupted and the respective controller processed at time intervals determined (= sampling time) by the user. The processor will then return to the breakpoint in the cyclic or time-driven program and will continue processing there.

Triggering:

A controller interrupt is triggered after the sampling time which the user has selected has elapsed.

Functions of the system program:

- manages the user interface for closed-loop controller processing.
- updates the closed-loop controller process image.

User interface: Standard Function Block "Controller Structure R 64"

The R64-standard function block is called for closed-loop controller processing as the user interface. With this block, up to 64 loops can be processed in connection with the controller parameter assignment block DB 2.

Parameters must be assigned to a specific data block for each of the controllers. In data block DB 2, the so-called 'controller list', you determine which controllers are to be processed and when by the system program. DB 2 is reserved for this particular task!

(You are assisted by a special program package: "COMREG" - order number for PG 685: 6ES5895-3SA11 - when assigning parameters, commissioning or testing the R64 standard FBs.)

Breakpoints

Interruption of controller processing is possible either at *block boundaries* (presetting) or *command boundaries* (programming DX 0). This is done by means of:

- process interrupt driven processing.

This can be interrupted at the *command boundaries* by

- a program error or a hardware fault.

4.4.4 PROCESS INTERRUPT: Interrupt-driven Program Execution

Interrupt-driven program execution involves an S5 bus signal from a digital input module capable of interrupts (e.g. 6ES5 432-4UA11) or an IP-module with a corresponding function causing the processor to interrupt program processing and to run a special program section. After the execution of this program the processor will return to the breakpoint and will continue processing there.

Triggering:

Process interrupts are triggered by an active state of an interrupt line on the S5 bus. Depending on the module slot, each of the processors is assigned one of the 7 interrupt lines (see Instructions CPU 928).

User interface: OB 2

OB 2 is called as the user interface if a process interrupt occurs. You have the option of programming a specific program in OB 2 which is to be processed in the case of a process interrupt.

If OB 2 is not programmed the program will not be interrupted. There will be no interrupt-driven program execution.

Breakpoints

A process interrupt-driven program can only be interrupted by

- a program error or hardware fault.

IMPORTANT!

An interrupt-driven program cannot be interrupted by time-interrupts or a further process interrupt.

If other process interrupts occur during the interrupt-driven program they are ignored until OB 2 has been fully executed (incl. all blocks called in the OB 2).

Then the processor will return to the breakpoint and will process the program to the next block boundary.

Only now will another process interrupt be accepted and OB 2 called again. This means that the cyclic program will be processed even if there is a permanent interrupt request. (This is not valid if in DX 0 the presetting "process interrupt effective at command boundaries" has been selected).

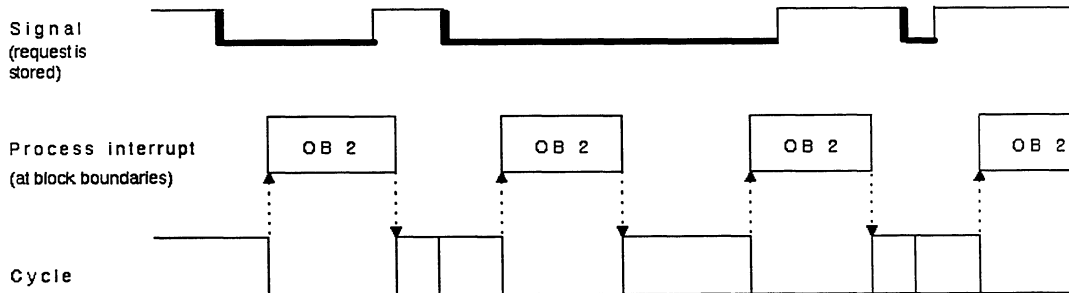
Process interrupt signal: level triggered

As a standard function, the process interrupt signal for the CPU 928 is level triggered, i.e.: the active state of the interrupt line triggers a request which causes OB 2 to be processed at the next block or command boundary. *This request is stored and only reset by the block-end command (BE) of OB 2.*

This leads to the following consequences:

- Multiple interrupts are not recognized.
- Those interrupts which occur during the processing of OB 2 and which are shorter than OB 2 are not recognized.
- OB 2 will be called even if the signal state of the interrupt line is passive again when reaching the block boundary (see figure).

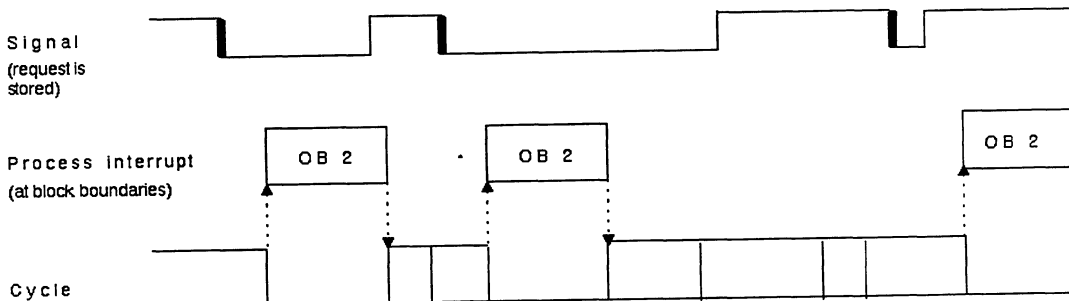
The called OB 2 is fully executed. If the signal level is still/again present at the end of OB 2, a block is processed in the cyclic program and then OB 2 is called again. If the signal level is no longer present OB 2 is not called again until the next signal change (from inactive to active).



Process interrupt signal: edge triggered

This setting is obtained by assigning parameters to DX 0. After the execution of OB 2 a new process interrupt can only be triggered by a change of the signal state (from inactive to active). With the edge-triggered process interrupt, the request to process an OB 2 is also stored until this OB is fully processed.

Any change of the signal state that may occur while OB 2 is processed will be ignored.



Inhibiting the process interrupt driven processing

An interrupt-driven program is inserted in the cyclic program at a block boundary or a STEP5 command boundary.

This interruption may have a negative effect if a cyclic program section has to be processed in a given time (in order to e.g. reach a certain response time) or interruption of a command sequence is not permissible (e.g. when reading or writing values that belong together).

If the interruption of a program section by interrupt-driven processing is not permissible, the following programming possibilities may be used:

- Program this program section so that it does not contain a block change and retain the presetting in DX 0 ("process interrupt at block boundaries"). Blocks that do not contain a block change cannot be interrupted.
- Write the program yourself in an interrupt-driven program. Interruption by another interrupt is then not possible.
- Program STEP5 command 'IA' (inhibit process interrupt). The command 'RA' (release process interrupt) serves to release the interrupt processing. No interrupt is serviced between these two commands. Thus, the program section between these two commands will not be interrupted by process interrupts occurring.

'IA' and 'RA' are only possible in function blocks (supplementary operation set)!
- Use the new special functions OB 120 and OB 122 that enable you to inhibit or delay the processing of process interrupts for a specific program section.

Priority between interrupt-driven and time-driven program execution

If a process interrupt is requested during time-driven program execution the program is interrupted at the next breakpoint (block or command boundary) and the process interrupt is serviced. Then the time driven program execution is continued.

If a time interrupt is requested during interrupt-driven program execution, the interrupt-driven program is completed first. Only then, is the time-driven program started.

If a process interrupt and a time interrupt occur simultaneously, then the process interrupt is serviced first at the next breakpoint. Only when this process interrupt has been executed as well as all the process interrupts that have occurred in the meantime, will the time interrupt be processed.

In this particular case, the time-driven processing has the **lower priority!**

Response time

The time required to respond to a time interrupt request is equivalent to the processing time of a block or a STEP5 command (depending on the selected presetting). If process interrupts exist at the time when the cyclic program is interrupted, only when all existing process interrupts have been fully executed, will the time-driven program be run.

In this case, the maximum response time between the occurrence and the servicing of a time interrupt is increased by the execution time of the process interrupts. If you wish to greatly reduce the possibility of a collision of two time interrupts for a specific time interrupt xy, note the following rule:

- A + B < C A = total of execution times of all program levels with a higher priority (process interrupt OBs, controller interrupt OBs, time interrupt OBs)
 B = Execution time of time interrupt OB xy
 C = Interval of time interrupt OB xy

The execution time for a time interrupt including any process interrupts may not exceed 100 ms!

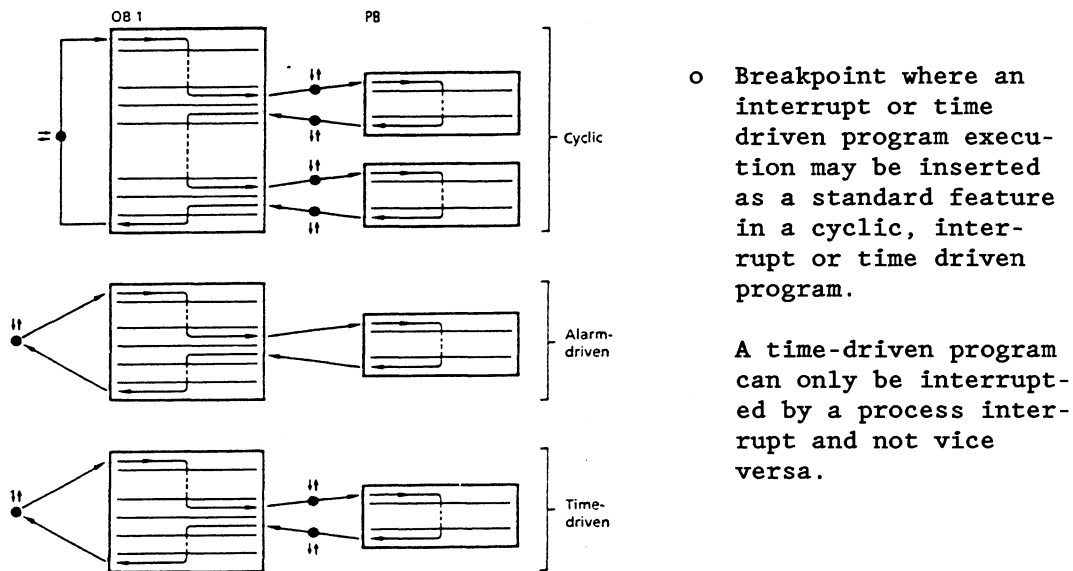


Fig. 4.2: Interrupt-driven program execution at block boundaries.

IMPORTANT!

If you run your user program not only cyclically but also time and/or interrupt driven, then there is a danger of e.g. the flags that are used as intermediate flags in the cyclic program being overwritten by time or interrupt driven processing being inserted in the cyclic program. For this reason it is important that you "save" the signal states of the flags in a data block before running a time or interrupt driven program and load them back into the flags at the end of the inserted program. For this purpose, there are four special function OBs available: OB 190 and OB 192 "transfer flags to data block" and OB 191 and 193 "transfer data blocks to flag area" (see corresponding chapters).

5 Handling Interrupts and Errors

The system program is in a position to detect faulty operation of the processor, errors in the system program processing or effects of incorrect programming by the user.

5.1 Frequent Errors in the User Program

The following list contains the errors which occur most frequently when the user program is first run. However, these errors may easily be avoided when creating the program.

For this reason, pay special attention to the following instructions when programming your STEP5 program:

- When specifying byte addresses for inputs and outputs the corresponding modules for these addresses must be plugged into the central controller or the expansion unit.
- Ensure that all operands are supplied the correct parameters.
- Outputs, flags, timers and counters should not be processed in different parts of the program with conflicting operations.
- Ensure that all data blocks called in the program exist and are long enough.
- Verify that all blocks to be called are actually in the memory.
- Be careful when making subsequent alterations in function blocks. Check that the FBs have been assigned the correct operands and that all actual operands have been specified.
- Timers should only be scanned once per cycle (e.g. A T1).

5.2 Evaluation of Error Information

If an error occurs during the start-up or the cyclic execution of the user program you have several "information sources" to trace the particular error.

a) LED's on the front panel of the processor

Make use of the LED's if the processor stops unexpectedly. These LED's may give you an indication of the causes of the error:

- STOP-LED is lit permanently
- STOP-LED flashes slowly
- STOP-LED flashes quickly

The different signals of the STOP-LED indicate certain causes of interruptions and errors.

Refer to Chapter 4.2 'Operating state STOP'.

The error LED's on the front panel are permanently lit.

- ADF (addressing error)
- QVZ (acknowledgement delay)
- ZYK (cycle time error)

b) Online function 'output ISTACK' (see Subsection 5.3)

The online function 'PC INFO' followed by 'output ISTACK' gives information on the statuses of the control bits and the contents of the interrupt stack (= ISTACK).

All information required for a warm restart is entered by the system program into the **ISTACK** during the transition to the stop state. These entries are a valuable help for error diagnosis. A complete output of the ISTACK is only possible in the stop condition.

Before the actual ISTACK is output, the statuses of the **control bits** are displayed first. These mark the current operating status and certain characteristics of the processor and the user program and give additional information about possible error causes.

Triggering of the online function 'output ISTACK' is not only possible in STOP but also in the operating states START-UP and RUN. In this case, however, the control bits are simply output.

c) System data RS 3 and 4 (see Chapter 5.5)

If your processor returns to the stop state during start-up, due to an error, the system data words RS 3 and RS 4 help to provide a more detailed definition of the cause of the error. These are errors which the system program detects when creating the address lists in DB 0 or DB 1 or when evaluating DB 2 or DX 0.

System data word RS 3: KH = EA03 (absolute memory address)

System data word RS 4: KH = EA04 (absolute memory address)

The error identifier in the system data word RS 3 helps you to identify the type of error.

The error identifier in the system data word RS 4 helps you to establish the location of the error.

The error identifiers are in data format KH.

Evaluation of system data word RS 3 and RS 4 with the programmer:

- o Using the online function 'OUTP. ADDRESS' (KH = EA03 or EA04) you can read the contents of the two system data words directly and determine the cause of the error.

d) Accumulator 1 and accumulator 2 (see Chapter 5.6)

If errors occur in the STEP5 program execution during the start-up or the cycle for which a specific organization block exists as the user interface, then the system program will automatically deposit additional error information in the accumulators 1 and 2 when the organization block is called to give a more precise explanation of the errors.

The error identifier in accumulator 1 helps you to determine the type of error.

The error identifier in accumulator 2 (if present) helps you to determine the location of the error.

The error identifiers are in data format KH.

Evaluation of accumulators 1 and 2 with the programmer:

- The online function 'output ISTACK' allows you to read the contents of both accumulators directly from the ISTACK and determine the exact cause of the error.

Evaluation of accumulators 1 and 2 with STEP5:

- Since the error identifiers are automatically deposited in the accumulators 1 and 2 when an error organization block is called you can take these identifiers into consideration when programming your error OBs.

Thus it is possible to program different reactions to different errors in an organization block, depending on the error identifier transferred to it.

e) Online function 'Output BSTACK'

The online functions 'PC INFO' and the subsequent 'BSTACK' allows you to have the contents of the block stack (= BSTACK) output in the stop state after an error has occurred (see Subsection 3.1.1).

Starting with OB 1 or FB 0 the BSTACK contains a list of all the blocks called before the processor stopped. Since the BSTACK is filled starting at the lower end, the block processed last and thus the block in which the error has occurred will be in the top line.

The following information is supplied for the evaluation of the top line:

BAUST.-NR. (block no.)	block type and number of the block processed before the stop state was reached
BAUST.-ADR. (block addr.)	absolute start address of this block in the program memory
RÜCKSPR.-ADR. (return addr.)	absolute address of the command to be processed next in this block. Using this command, the processor will continue the program after the 'manual restart'
REL.-ADR.	relative address (= RÜCKSPR.-ADR. -BAUST.-ADR.) of the command to be processed next in this block

(Display of relative addresses by the PG is possible in the operating mode 'input inhibit' (key operated switch))

DB-NR. number of the data block called last

DB-ADR. absolute start address of this data block
 (address of data word DW 0) in the program memory

Example: evaluate 'OUTPUT BSTACK'

BAUST.-NR. (block no.)	BAUST.-ADR. (block addr.)	RÜCKSPR.-ADR. (return addr.)	REL.-ADR.	DB-NR.	DB-ADR.
OB 23	0063	0064	0001	13	0078
FB 5	006A	0072	0008	13	0078
FB 6	008A	0091	0007	100	0098
OB 1	009D	009E	0001		

The above example shows that the stop occurred in OB 23 during the processing of the STEP5 instruction listed under the absolute address 'KH0064 - 1 = KH0063' in the memory.

OB 23 (QVZ error OB) has been called in FB 5 at the relative address 'KH0008 - 1 = KH0007'.

The data block DB 100 has been called in FB 6. Data block 13 was valid when the processor stopped. DB 13 was called in FB 5.

Summary

When searching for the cause of an error make use of all information available.

This could be:

1. LEDs on the front panel of the processor

Certain signals point to certain causes of errors or interruptions.

2. Online function 'output ISTACK'

The control bits are always output; in the stop state ISTACK is also output.

3. System data words RS 3 and RS 4:

You will find more exact information about the cause of the error in the system data words RS 3 and 4 for those errors that occur during the start-up.

4. Accumulator 1 and 2:

The system program deposits additional error information in accumulator 1 and accumulator 2 when error organization blocks are called.

5. Online function 'output BSTACK':

In the stop state you can identify the block and in it the address of the command being processed when the error occurred, by reading the top line of the BSTACK.

5.3 Control Bits and Interrupt Stack

By means of online functions 'PC INFO' (F7) followed by 'output ISTACK' (F5) you can analyze the operating status, the characteristics of the processor and the user program as well as possible causes of errors and interruptions.

IMPORTANT!

Output of control bits is possible in any operating state, output of the ISTACK only in stop.

- The **control bits** indicate the current or previous operating status as well as the cause of the error.
If several errors have occurred all errors that have occurred will be displayed in the control bits.
- The breakpoint (addresses) with the condition code words at that point and the contents of the accumulators as well as the cause of the error are entered in the ISTACK. If several errors have occurred a multi-layer interrupt stack is created:
depth 01 = last cause of error,
depth 02 = next to last cause of error etc..

In the case of an ISTACK overflow an immediate stop will be executed. A cold restart is required afterwards.

The significance of the abbreviations in the control bits and the interrupt stack is explained in the following pages. The abbreviations are output on the PG 685 programmer.

IMPORTANT

The mask displayed on your PG may differ from the one shown below. Irrespective of the terms used in your software version, the positions shown here are valid.

CONTROL BITS

>>STP<<	STP-6	FE-STP	BARBEND	PG-STP	STP-SCH	STP-BEF	MP-STP
>>ANL<<	ANL-6	NEUST	M W A	A W A	ANL-2	NEU-ZUL	MWA-ZUL
		X				X	X
>>RUN<<	RUN-6	EINPROZ	BARB	OB1GEL	FBOGEL	OBPROZA	OBWECKA
	X	X			X		
32KWRAM	16KWRAM	8KWRAM	EPROM	KM-AUS	KM-EIN	DIG-EIN	DIG-AUS
		X				X	X
URGLOE	URL-IA	STP-VER	ANL-ABB	UA-PC	UA-SYS	UA-PRFE	UA-SCH
DXO-FE	FE-22	MOD-FE	RAM-FE	DB0-FE	DB1-FE	DB2-FE	KOR-FE
N A U	P E U	B A U	STUE-FE	Z Y K	Q V Z	A D F	WECK-FE
B C F	FE-6	FE-5	FE-4	FE-3	L Z F	REG-FE	DOPP-FE

.pa

The statuses of the control bits are displayed on the first page of the screen when the ISTACK is output on the PG.

The following control bits mark the current or previous operating status of the processor and supply information about certain characteristics of the processor and the STEP5 user program.

Output of the control bits is possible in all operating conditions. This allows you to e.g. verify that the organization block OB 2 has been loaded and whether interrupt-driven program execution is possible or not.

STP Processor is in the operating state STOP; the following control bits explain why the processor is in this state

- STP-6 not used
- FE-STP Error-stop: stop state following NAU (power failure), PEU (I/Os not ready), BAU (battery not ready), STUEB (BSTACK overflow), STUEU (ISTACK overflow), DOPP (double error) or processor fault
- BARBEND Finish process check: stop condition after online function "process control end" (cold restart required)
- PG-STP PG stop: stop status due to command from PG
- STP-SCH Stop switch: stop status due to stop switch in the STOP position
- STP-BEF Stop command:
 - a) Stop status after the processing of STEP5 operation 'STP'
 - b) Stop status after stop command by the system program if error organization block has not been programmed.
- MP-STP Multiprocessor stop:
 - a) Selector switch on COR in the STOP position or
 - b) another processor has stopped during multiprocessor operation

ANL Processor is in operating state START-UP:

- ANL-6 not used
- NEUST Cold restart is requested or active or was executed as the last start-up.
- M W A Manual warm restart is requested or active or was executed as the last start-up.
- A W A Automatic warm restart after power failure is requested or active or was executed as the last start-up.

ANL-2 not used
NEU-ZUL Cold restart permissible as the next start-up mode
MWA-ZUL Manual warm restart permissible as the next start-up mode.

RUN Processor is in the operating status RUN (cyclic program execution is active):

RUN-6 not used
EINPROZ Single processor operation
BARB Online function "process control" is active
OB1GEL Organization block OB 1 has been loaded into the user memory.
Cyclic program execution is determined by OB 1.
FBOGEL Function block FB 0 has been loaded into the user memory.
Cyclic program execution is determined by FB 0 if no OB 1 has been loaded. If FB 0 and OB 1 have been loaded, then OB 1 is valid for cyclic program execution.
OBPROZA Process interrupt organization block OB 2 has been loaded, i.e. process interrupt driven program execution is possible
OBWECKA Time interrupt organization block has been loaded, i.e. time-driven program execution is possible.

32KWRAM User memory submodule is a RAM with 32×2^{10} words.
16KWRAM User memory submodule is a RAM with 16×2^{10} words.
8 KWRAM User memory submodule is a RAM with 8×2^{10} words.
EPROM User memory submodule is an EPROM.
KM-AUS Address list for interprocessor communication flag outputs from DB 1 present
KM-EIN Address list for interprocessor communication flag inputs from DB 1 present
DIG-EIN Address list for digital inputs present
DIG-AUS Address list for digital outputs present
URGELOE Overall reset of processor was carried out (cold restart required)
URL-IA Overall reset of the processor being carried out.
STP-VER Processor has caused stop status in the central

processing unit.

- ANL-ABB Abort during the start-up (cold restart required)
- UA-PG PG has requested overall reset.
- UA-SYS System program has requested overall reset (no start-up possible); overall reset must be carried out.
- UA-PRFE Overall reset request due to processor error
- UA-SCH Overall reset requested by operator (switch); carry out overall reset or select a start-up mode if requested overall reset is not to be carried out.

The following control bits mark errors which may occur in the operating states START-UP (e.g. in the case of the first cold restart) and RUN (e.g. in the case of time-driven program execution).

If several errors have occurred, then all the errors that have occurred up to this moment (and that are still to be processed) are indicated in the last three lines of the control bits mask. Note the system data RS 2: It contains the UAMK (interrupt condition codeword, collected) into which all the errors that have occurred and that are still to be processed have been entered as well (Subsection 8.2.4).

Errors during START-UP:

- DX0-FE Parameter assignment error in DX 0.
- FE-22 not used
- MOD-FE User module contains errors (overall reset required for RAM).
- RAM-FE Operating system RAM or the DB-RAM contains errors (overall reset requested).
- DB0-FE Structure of the block address list in DB 0 is incorrect
- DB1-FE Structure of the address list in DB 1 for updating of the process images is incorrect;
 - a) DB 1 not programmed with the coordinator plugged-in or in multiprocessor operation;
 - b) No acknowledgement from the byte addresses for inputs and outputs or interprocessor communication flags specified in DB 1 during a cold restart on the corresponding modules.
- DB2-FE Error during the evaluation of the parameter assignment data block DB 2 of controller structure R64

Error during START-UP or RUN:

- KOR-FE Error during data exchange with the coordinator
- NAU Power failure in the central controller
- PEU I/O not ready = power failure at an expansion unit
- BAU Battery defective = failure of the back-up battery (central controller)
- STUE-FE Interrupt or blockstack overflow (nesting depth too great; cold restart required)
- ZYK Cycle time exceeded
- QVZ Acknowledgement delay during data exchange with I/O's
- ADF Addressing error at inputs or outputs
- (error caused by access to process image with I/O modules addressed that were not plugged-in or defective or not specified in DB 1 during the last cold restart)
- WECK-FE Collision of two time interrupts
Prior to or during processing of a time interrupt OB, it has been called a second time.
- BCF Command code error:
- a) Substitution error: STEP5 command processed can not be substituted
 - b) Operation code error: STEP5 command processed is wrong
 - c) Parameter error: parameter of the STEP5 command processed is wrong
- FE-6 not used
- FE-5 not used
- FE-4 Power-down error:
- Processing of a previous power failure (NAU) has been terminated with error by the system program; warm restart is therefore inhibited.
- FE-3 not used
- LZF Execution time error:
- a) Block called has not been loaded
 - b) Transfer error with data blocks
 - c) Other execution time errors
- REG-FE Error during the processing of controller structure R64 in the cycle
- DOPP-FE Double error:
A program level (ADF, BCF, LZF, QVZ, REG, ZYK) which is still active has been activated a second time. (Cold restart required)

Press the return key to terminate the display of the control bits on your PG screen. The ISTACK is then displayed on the following page. During transition to the stop status, the system program uses this storage to enter all data necessary to perform a cold or warm restart.

IMPORTANT

The mask displayed on your PG may differ from the one shown below. Irrespective of the terms used in your software version, the positions shown here are valid.

ISTACK

```

DEPTH:      02

BEF-REG:    C70A      SAC:      00F3      DBADDR:    0000      BA-ADD:    0000
RST-STP:    0000      FB-NO.:   226      DB-NO.:    -NO.:
LEVEL:      0004      UAMK:     0100     UALW:      0000

ACCU1: 0000 C464   ACCU2: 0000 00FF   ACCU3: 0000 0000   ACCU4: 0000 0000

BRACKETS: KE1 111  KE2 100  KE3 111

RESULT BITS:      DSP1 DSPO OVFL OVFLS  OR  STATUS RLO ERAB
                  X                      X    X
CAUSE OF
INTERR.:          NAU  PEU  BAU  MPSTP ZYK  QVZ  ADF STP
                  X    X
                  BCF  S-6  LZF  REG   STUEB STUEU WECK DOPP
    
```

The ISTACK contains all the information required to find the instruction in the user program which was being executed when the processor stopped.

DEPTH Layer of the ISTACK for error nesting

DEPTH 01 = cause of last trouble
 DEPTH 02 = cause of second last trouble

BEF-REG Instruction register:
 Contains the machine code (first word) of the last command executed in an interrupted program level

RST-STP Block stack pointer:
 Contains the number of elements entered in the block stack (BSTACK)

LEVEL-Z States the level of program that has been interrupted

Z: 0002 = cold restart
0004 = cycle
0006 = time interrupt 5 sec (OB 18)
0008 = time interrupt 2 sec (OB 17)
000A = time interrupt 1 sec (OB 16)
000C = time interrupt 500 ms (OB 15)
000E = time interrupt 200 ms (OB 14)
0010 = time interrupt 100 ms (OB 13)
0012 = time interrupt 50 ms (OB 12)
0014 = time interrupt 20 ms (OB 11)
0016 = time interrupt 10 ms (OB 10)
0018 = not used
001A = not used
001C = controller processing
001E = not used
0020 = not used
0022 = not used
0024 = process interrupt
0026 = not used
0028 = not used
002A = not used
002C = transition to stop state in multiprocessor
operation, stop switch or PG stop
002E = not used
0030 = time interrupt
0032 = controller error
0034 = cycle time error
0036 = not used
0038 = command code error
003A = execution time error
003C = addressing error
003E = acknowledgement delay
0040 = not used
0042 = not used
0044 = manual warm restart
0046 = automatic warm restart

SAC STEP address counter:

contains the absolute address of the last command executed in an interrupted program level in the program memory. If an error occurs, the SAC will point directly to the command responsible for the error!

If the error is not at the STEP5 user program, the SAC will be '0', the contents of the BEF-REG are irrelevant.

..NO. Type and number of the block processed last.

REL-SAC Relative step address counter:
Contains the relative address (relative to the block start address) of the command executed last in the block processed last.

(Display of relative addresses is possible in the operating mode "input inhibited" (key operated switch) or when the block is output on the printer.)

UAMK Interrupt condition code word (collected):
The UAMK contains all the errors which have occurred and are still not fully processed (see "system data assignment", Subsection 8.2.4).

UALW Interrupt condition inhibit word
(see "system data assignment", Subsection 8.2.4)

DBADDR Absolute start address of the block in the program memory (DW 0) called last

(DB-ADR = 0000, if no DB has been called)

DB-NO Number of the data block called last

DBL-REG Length of the data block called last

BA-ADD Absolute address in the program memory of the command to be processed next in the block called last

...NO. Type and number of the block called last

ACCUL...4 Contents of the arithmetic register before transition to the stop state

In the case of certain errors the system program will deposit error identifiers in accumulators 1 and 2 during the transition to the stop state. These numbers supply a more detailed explanation of the causes of the interruption.

BRACKETS Number of bracketing levels: 'KEx abc'
x = 1 up to 7 levels

a = OR (see bit condition codes)
b = RLO (result of logic operation, see bit condition codes)
c = 1: A(
c = 0: 0(

RESULT

BITS: see Chapter 3.2

The following abbreviations are the most important causes of errors and interruptions. Only those causes are marked which have occurred in the program level being displayed (see LEVEL).

The information about the causes of interruptions is taken from the interrupt condition code word (UAMK, 16 bits; see Subsection 8.2.4). Some of this information is identical with that of the control bits.

NAU Power failure in the central controller

PEU I/O's not ready = power failure in expansion unit

BAU Battery not ready = failure of back-up battery (central controller)

MP-STP Multiprocessor stop:
a) Selector switch at KOR in STOP position or
b) another processor has stopped during multiprocessor operation

ZYK Cycle time exceeded

QVZ Acknowledgement delay during data exchange with I/O's

ADF Addressing error at inputs or outputs

STP Stop state due to stop switch in STOP position
Stop state due to instruction from the PG
Stop state after processing STEP5 operation 'STP'
Stop state after stop command by system program if error organization block has not been programmed.

BCF Command code error:
Errors recognized during command decoding

a) Substitution errors: STEP5 command processed can not be substituted

b) Operation code error: STEP5 command processed is wrong

c) Parameter error: Parameter of STEP5 command processed is wrong

S-6 not used

LZF Execution time error:
Errors recognized during command execution

a) Block called has not been loaded

b) Transfer error with data blocks

c) Other execution time errors

REG Error during processing of controller structure R64 in cycle

STUEB Block stack overflow (nesting depth too great; cold restart required)

STUEU Interrupt stack overflow (nesting depth too great; cold restart required)

WECK Collision of two time interrupts:
Before or during processing of a time interrupt OB, the same OB is called a second time.

DOPP Double error:
A program level (ADF, BCF, LZF, QVZ, REG, ZYK) which is still active has been activated a second time.
(Cold restart required)

Evaluating the ISTACK: Examples

The figure below shows the structure of the ISTACK in connection with the interruptions that have occurred.

1. The program level CYCLE (OB 1) is interrupted by the occurrence of a time interrupt (100-ms interval).
2. The program level TIME INTERRUPT is activated and OB 13 processed.
3. A process interrupt occurs. This results in the TIME INTERRUPT level being left, the PROCESS INTERRUPT level being activated and OB 2 processed.
4. Due to an illegal addressing instruction the ADF level is activated and OB 25 processed. The user's error routine contains a stop command (STP) so that the processor aborts the execution of the program.

Four different program levels have been interrupted before the processor passes into the stop status. Therefore, if you have the ISTACK output at the PG, what you receive is a 4-layer ISTACK. The uppermost layer represents depth 01 with the identification of the program level (= ADF) which was interrupted last. You may 'switch down' the ISTACK until you reach depth 04 representing the CYCLE level which was interrupted first.

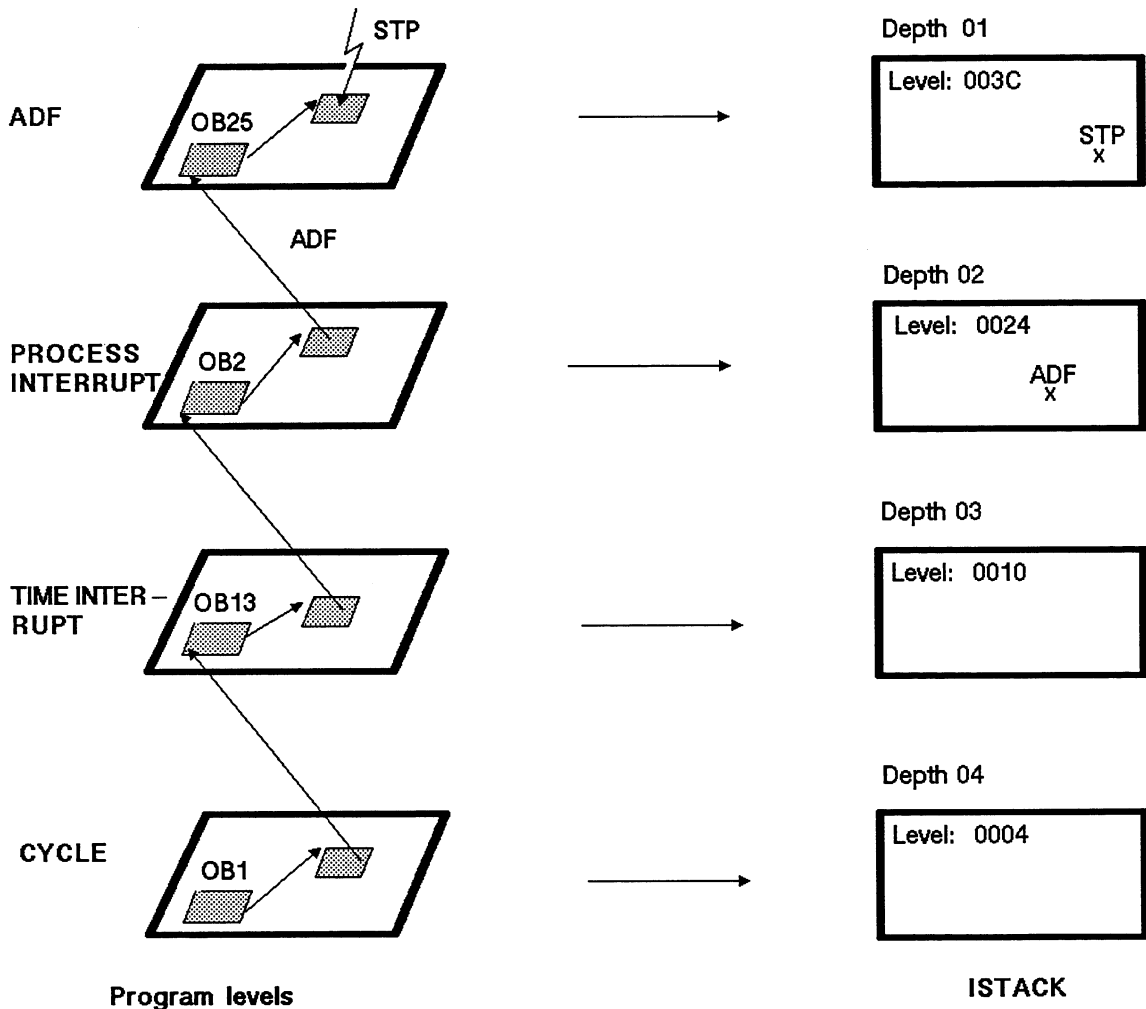
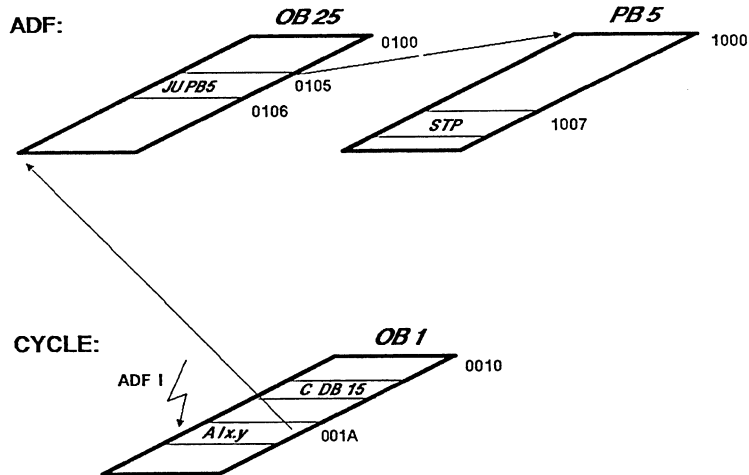


Fig. 5-1: The structure of the ISTACK in an example

In the following example, the processor detects an addressing error when executing the instruction 'A I x.y' in OB 1. This causes OB 25 to be processed. The processor goes into the stop status due to an STP instruction in PB 5.

Two interrupted program levels lead to a 2-layer ISTACK:



ISTACK			
DEPTH:	02		
BEF - REG:	AIxy	SAC: 001A	DB-ADDR: BA-ADDR: 0000
BST - STP:	1	OB-NO: 1	DB-NO: 16 -NO:
		REL-SAC: 000A	DBL - REG:
LEVEL:	0004	UAMK: 0200	UALW: 0000
ACCU1:			
RESULT BITS:		
CAUSE OF INTERRUPTION:			ADF x

ISTACK			
DEPTH:	01		
BEF - REG:	STP	SAC: 1007	DB-ADDR: BA-ADDR: 0106
BST - STP:	3	PB-NO: 5	DB-NO: 16 OB-NO: 25
		REL-SAC: 0007	DBL - REG:
LEVEL:	003C	UAMK: 0300	UALW: 0000
ACCU1:			
RESULT BITS:		
CAUSE OF INTERRUPTION:			STP x

5.4 Error Handling using Organization Blocks

As soon as the system program has recognized a certain error it will call the organization block programmed to handle this particular error. By programming this organization block you can decide how the processor should react.

You can program the organization block so that

- normal program execution is continued,
 - the processor stops
- or
- a special 'error program' is processed.

Organization blocks are available to handle the following causes of errors:

Cause of error	OB called	Reaction if OB is not programmed
Call of a block that has not been loaded (LZF)	OB 19	stop
Acknowledgement delay in the user program when accessing I/O modules (QVZ)	OB 23	none
Acknowledgement delay when updating the process image and for interprocessor communication flag transfer (QVZ)	OB 24	none
Addressing error (ADF)	OB 25	stop
Cycle time exceeded (ZYK)	OB 26	stop
Substitution error (BCF)	OB 27	stop
Mode selector on STOP, PG function 'PC-STOP', stop via S5 bus (multiprocessor operation)	OB 28	stop
Operation code error (BCF)	OB 29	stop
Parameter error (BCF)	OB 30	stop
Other execution time errors (LZF)	OB 31	stop
Transfer error with data blocks (LZF)	OB 32	stop
Collision of two time interrupts (WECK-FE)	OB 33	stop
Error during the processing of controller structure R64 (REG-FE)	OB 34	stop

If OB's are not programmed the reaction depends on the type of error:

a) **no interruption of cyclic program execution**

If an acknowledgement delay occurs and OB 23 and OB 24 have not been loaded the cyclic program execution is not interrupted. There will be no reaction from the processor.

If the processor is to go over to the stop state following QVZ, the organization block must include a stop instruction and be completed with BE.

Program for stop:

```
:  
:  
:STP  
:BE
```

b) **stop state**

With all other types of errors the processor will immediately go over to the stop state if the appropriate organization blocks have not been programmed by the user.

If it is necessary that one or the other error does not interrupt cyclic program processing in particular cases (e.g. during commissioning), all that is required is a block end instruction in the appropriate organization block.

Program for operation without interruption:

```
:  
:  
:BE
```

IMPORTANT!

Organization block OB 28 is an exception to this rule: A change to the stop state will always be made in this particular case, irrespective of how and whether OB 28 is programmed.

If you do not intend to program the corresponding organization block, you have the option of programming data block DX 0 in order to prevent the processor from going over to the stop state.

Interruptions during the processing of error organization blocks

After the system program has called the error organization block the user program contained in it will be processed.

If another error occurs during the processing of an organization block the cyclic program execution will be interrupted at the next command boundary and the corresponding organization block will be called.

The organization blocks are processed in the order in which they are called. The number of error organization blocks that may be nested into one another depends on

a) the type of errors that have occurred:

Nesting of organization blocks belonging to the same program level is not possible!
(Refer to the following chapter for the assignment of error-OBs to the program levels.)

When processing OB 27 (program level BCF) it is possible to nest in OB 32 (program processing level LZP), however, OB 29 or OB 30 (also BCF) may not be nested in.

The processor will stop immediately if a program level is called twice.

b) the number of program levels activated at the time:

When interrupts occur the system program requires extra memory space for each of the program levels activated in order to create the ISTACK. If there is no more memory space left, an ISTACK overflow is produced.

If this occurs the processor will immediately stop.

c) the number of blocks called at the time:

If an ISTACK overflow occurs the processor will immediately stop.

5.5 Errors during START-UP

Errors occurring during initialization or start-up may result in the start-up program being aborted. The processor goes into the stop status.

Errors occurring in the start-up program (organization blocks OB 20, 21 and 22) are handled in the same way as in cycle.

Exception: In case of a stop during start-up, OB 28 will not be called.

Possible causes of interruptions and errors without appropriate error organization block

STP:

Stop command from the system program (at FE-STP) or in the user program

BAU:

Failure of the back-up battery on the central controller

NAU:

Power failure on the central controller

PEU:

Power failure on an expansion unit

STOEU:

Stack overflow of the interrupt stack (ISTACK)

STUEB:

Stack overflow of the block stack because the nesting depth is too great

DOPP:

Double call of an error program level (for double errors, refer to the Examples on page 4-4)

RAM-FE:

Error during initialization: incorrect contents of the operating system RAM or the DB RAM

MOD-FE

Error during initialization: incorrect contents of the user submodule (RAM or EPROM submodule)

DB0-FE

Error during creation of the block address list (DB 0)

DB1-FE

Error during evaluation of DB 1 which is performed to create the address list for the updating of the process image

DB2-FE

Error during evaluation of DB 2 of controller structure R64

DX0-FE

Error during evaluation of DX 0. (A detailed description of DB0-FE, DB1-FE, DB2-FE and DX0-FE is given on the following pages.)

5.5.1 DBO-FE (Error in DB 0)

Error while setting up the block address list (data block DB 0)

DB 0 is created by the system program following power on.

If a DBO error occurs you will find error identifiers which provide a more detailed explanation of the error in the system data words RS 3 and RS 4.

Absolute memory address: RS 3 KH = EA03
 RS 4 KH = EA04

(Refer to Chapter 5.2 for details about the evaluation of error identifiers in RS 3 and RS 4.)

Error identifier RS3 RS4	Explanation
8001H yyyyH	Incorrect block length yyyy = address of block with incorrect length
8002H yyyyH	Calculated end address of block in the memory incorrect yyyy = block address
8003H yyyyH	Invalid block identifier yyyy = address of block with incorrect identifier
8004H yyyyH	Number of organization block too high (permissible: OB 1 through OB 39) yyyy = address of block with incorrect number
8005H yyyyH	Data block number 0 (permissible: DB 1 through DB 255) yyyy = address of block with incorrect number

5.5.2 DB1-FE (Error in DB1)

Errors while setting up the address list in DB 1 for updating of process images.

- DB 1 missing in multiprocessor operation or
- incorrect DB1-address list during cold restart

You will also find error identifiers which provide a more detailed explanation of DB1 errors in the system data words RS 3 and RS 4.

Absolute memory address: RS 3 KH = EA03
 RS 4 KH = EA04

(Refer to Chapter 5.2 for details about error evaluation in the system data words RS 3 and RS 4.)

Error identifier RS3	RS4	Explanation
0410H	yyyyH	Invalid identifier 1. Header identifier is missing or incorrect (permissible KS MASK01) 2. Invalid identifier (permissible KH DE00, DA00, CE00, CA00, BB00) 3. End delimiter missing or incorrect (permissible KH EEEE) yyyy = incorrect identifier
0411H	yyyyH	"Digital inputs" invalid number of addresses (permissible 0...128) yyyy = invalid number of addresses
0412H	yyyyH	"Digital outputs" invalid number of addresses (permissible 0...128) yyyy = invalid number of addresses
0413H	yyyyH	"Interprocessor communication flag input" invalid number of addresses (permissible 0...256) yyyy = invalid number of addresses
0414H	yyyyH	"Interprocessor communication flag output" invalid number of addresses (permissible 0...256) yyyy = invalid number of addresses
0415H	yyyyH	Invalid number of timer locations (permissible: 256) yyyy = incorrect number of timer locations
0419H	yyyyH	Acknowledgement delay at digital inputs yyyy = address of input bytes not acknowl.
041AH	yyyyH	Acknowledgement delay at digital outputs yyyy = address of output bytes not acknowl.
041BH	yyyyH	Acknowledgement delay at interprocessor communication flag inputs yyyy = address of flag byte not acknowl.
041CH	yyyyH	Acknowledgement delay at interprocessor communication flag outputs yyyy = address of flag byte not acknowl.

5.5.3 DB2-FE (Error in DB2)

Errors during the evaluation of the parameter assignment data block DB 2 of the controller structure R 64 (controller initialization).

You will find error identifiers which provide a more detailed explanation of DB2 errors in the system data words RS 3 and RS 4.

Absolute memory address:	RS 3	KH = EA03
	RS 4	KH = EA04

(Refer to Chapter 5.2 for details about the evaluation of error identifiers in RS 3 and RS 4.)

Error identifier RS3	RS4	Explanation
0421H	DByyH	Data block not loaded yy = number of data block not loaded
0422H	FByyH	Function block not loaded yy = number of function block not loaded
0423H	FByyH	Function block not identified yy = number of function block not identified
0424H	FByyH	Function block with incorrect PG software loaded yy = number of function block
0425H	DByyH	Invalid length of controller data block yy = number of data block
0426H	-	Not enough memory space in DB RAM to receive controller DBs coming in from the user EPROM

5.5.4 DX0-FE (Error in DX 0)

Errors that occur during the evaluation of data block DX 0.

You will find error identifiers which provide a more detailed explanation of DX 0 errors in the system data words RS 3 and RS 4.

Absolute memory address: RS 3 KH = EA03
 RS 4 KH = EA04

(Refer to Chapter 5.2 for details about the evaluation of error identifiers in RS 3 and RS 4.)

Error identifier		Explanation
RS3	RS4	
		Invalid identifier 1. Header identifier is missing or incorrect (permissible KS MASK01) 2. Invalid block identifier 3. End delimiter missing or incorrect (permissible KH EEEE) yyyy = incorrect identifier
0432H	yyyyH	Illegal parameter yyyy = incorrect parameter
0434H	yyyyH	Illegal number of timer locations (permissible: 0...256) yyyy = incorrect number of timer locations
0435H	yyyyH	Illegal cycle time (permissible: 1 ms to 6000 ms) yyyy = incorrect time value

5.6 Errors in RUN and START-UP

In the operating state RUN a cyclic, time or interrupt driven program or controller processing can be interrupted at command boundaries if certain errors/faults occur.

Causes of interruptions occurring during initialization and operating state START-UP also cause the processor to go into the stop state or call up the organization block provided for the handling of this error. Errors occurring in the start-up program are handled the same as in cycle.

A distinction is made between the causes of interruptions which directly cause the processor to STOP (e.g. STUEU) and those causes of interruptions that trigger the system program to call user-programmable organization blocks (e.g. ADF) before the transition is made to the stop state.

Possible causes of interruptions and errors *without* appropriate error organization block

STP:

Stop command from the system program (for machine errors) or in the user program

BAU:

Failure of the back-up battery on the central controller

NAU:

Power failure on the central controller

PEU:

Power failure on an expansion unit

STUEU:

Stack overflow of the interrupt stack (ISTACK)

STUEB:

Stack overflow of the block stack because the nesting depth is too great

DOPP:

Double call of an error program level (also refer to the examples on pages 4-4 ff)

An immediate transition to the stop state will be carried out for all of the errors/faults mentioned above. At the same time, an ISTACK is created in which the error that has occurred is indicated.

(Refer to Chapter 5.3 for information on how to evaluate the ISTACK)

Possible causes of interruptions and errors with appropriate error organization block in RUN and START-UP

BCF:

Command code errors	1. Substitution error	OB 27
	2. Operation code error	OB 29
	3. Parameter error	OB 30

LZF:

Execution time errors	1. Call of a block that has not been loaded	OB 19
	2. Transfer error	OB 32
	3. Other execution time errors	OB 31

ADF:

Addressing errors		OB 25
-------------------	--	-------

QVZ:

Acknowledgment delay	1. in the user program when accessing I/Os	OB 23
	2. during updating of the process image	OB 24

Possible causes of interruptions and errors with appropriate error organization block in RUN only

ZYK-FE:

Cycle error		OB 26
-------------	--	-------

WECK-FE:

Collision of two time interrupts		OB 33
----------------------------------	--	-------

REG-FE:

Controller error		OB 34
------------------	--	-------

ABBR:

Abort		OB 28
-------	--	-------

You will find more details about these errors in the following chapters.

5.6.1 BCF (Command Code Error)

A command code error occurs when the processor cannot interpret or execute a STEP5 command in the user program. All permissible command codes are listed in the appendix.

The command which is responsible for the command code error is not executed. If a BCF organization block has been programmed, it is called up and processed. Then the processor will continue with the next command in the user program that was interrupted. The processors stops if no BCF OB has been programmed.

A distinction is made between the following command code errors:

a) BCF = Substitution errors

If an operation using a formal operand is to be executed in a function block when processing the user program the processor will substitute this formal operand with the actual operand contained in the function block call.

The processor identifies an illegal substitution. The system program will now interrupt program processing and will call organization block **OB 27**.

Accumulator 1 contains additional information to explain the error.

Error identifier accum1 accum2	Explanation
1801H -	substitution error for command DO RS
1802H -	substitution error for command DO DW, DO FW
1803H -	substitution error for command DO=, DI=
1804H -	substitution error for command L=, T=
1805H -	substitution error for command A=, AN=, O=, ON=, ==, S= and RB=

b) BCF = Operation code errors

An illegal operation code occurs if a command has been programmed that is not part of the STEP5 range of commands of the processor (e.g. programming of RU and SU commands is possible with the PG, however, these commands cannot be interpreted by the R and S processors and the CPU 928 in the S5 135U).

If the processor identifies an illegal operation code, processing of the user program is interrupted at this point and organization block **OB 29** is called.

When OB 29 is called, additional information to provide a more detailed explanation of the error is provided in accumulator 1.

Operation code errors should not be acknowledged: The processor does not recognize whether the faulty command is a single or a multi-word instruction. If the processor has executed OB 29 it attempts to continue processing the program using the next instruction word. If this happens to be the second word of a multi-word instruction the processor will either recognize another command code error or will execute this particular word as a valid command.

Error identification accu1 accu2	Explanation
1811H -	Command with illegal operation code
1812H -	Illegal operation code for a command in which the high-byte of the 1st word contains the value 68H
1813H -	Illegal operation code for a command in which the high-byte of the 1st word contains the value 78H
1814H -	Illegal operation code for a command in which the high-byte of the 1st word contains the value 70H
1815H -	Illegal operation code for a command in which the high-byte of the 1st word contains the value 60H

c) BCF = Parameter errors

An illegal parameter occurs if an instruction has been programmed using a parameter which is illegal for the processor concerned (e.g. call of a reserved data block) or if a special function is called that does not exist.

If an illegal parameter is identified by the processor the system program will interrupt the user program and will call **OB 30**.

If OB 30 has not been programmed the processor will go over to the stop state.

When OB 30 is called, additional information to provide a more detailed explanation of the error is provided in accumulator 1.

Error identifier accu1 accu2	Explanation
	Illegal parameters with:
1821H -	C DB 0, 1, 2
182BH -	JU(B) OB 0
182CH -	JC(B) OB > 39: no special function exists
182DH -	AX DX0
182EH -	L FW/T FW/L PW/T PW/L OW/T OW/L DD/T DD/DO FW 255
182FH -	L IW/T IW/L QW/T QW 127

Error identifier accu1 accu2	Explanation
1830H -	L FD/T FD 253, 254, 255
1831H -	L ID/T ID/L QD/T QD 125, 126, 127
1832H -	RLD/RRD/SSD/SLD 33-255
1833H -	SLW/SRW/LIR/TIR 16-255
1834H -	SED/SEE 32-255
1835H -	A=/AN=/O=/ON=/S=/RB=/=/RD=/FR=/SP=/SR=/SEC= SSU=/SFD=/L=/LC=/LW=/T= 0, 127-255
1836H -	DO=/LC= 0, 126-255
1837H -	MBR with constant > 0FFFFH ($2^{16} - 2^{19} \# 0$)

5.6.2 LZF (Execution Time Error)

An execution time error occurs if the processor identifies an error during the processing of a STEP5 command.

The command responsible for the execution time error is not executed. If a LZF organization block has been programmed, it is called up. Then the processor will continue with the next command in the user program that was interrupted. The processor stops if no LZF OB has been programmed.

A distinction is made between the following execution time errors:

a) LZF = Calling a block that has not been loaded

If a block that does not exist is called in the user program the system program will identify an error. This applies to all kinds of blocks for both conditional and unconditional call statements.

If the processor identifies a call for a block that has not been loaded the system program will call **OB 19**. In OB 19 you can specify the reaction of your processor. If OB 19 contains only the BE command (block end), the processing of the STEP5 program that has been interrupted will be continued with the next command. If OB 19 is not programmed the processor will go over to the stop state as soon as a block that has not been loaded is called.

When OB 19 is called, additional information to provide a more detailed explanation of the error is provided in accumulator 1.

Error identifier accu1 accu2	Explanation
1A01H -	Data block not loaded (command CDB)
1A02H -	Data block not loaded (command CX DX)
1A03H -	Block not loaded (command JU(B) FB, OB, PB, SB)
1A04H -	Block not loaded (command DO(C)FX)
1A05H -	Data block for OB 254 or 255 not loaded

b) LZF = Transfer errors

When transferring data to a data block (DB, DX) the processor compares the length of the DB called with the parameters contained in the transfer command. If the length of the data block is exceeded owing to the parameters specified the transfer command will not be executed in order to prevent accidental overwriting of data in the memory.

A transfer error is also identified if an individual bit is to be scanned or altered within a data word that does not exist.

A transfer error is also recognized if a data word is to be accessed before a data block has been called (with C DBn or CX DXn).

If the system program identifies a transfer error it will call **OB 32**. The command responsible for the transfer error is no longer processed. If OB 32 has not been programmed the processor will go over to the stop state.

When OB 32 is called, additional information to provide a more detailed explanation of the error is provided in accumulator 1.

Error identifier accu1 accu2	Explanation
1A11h -	Access to a data word that has not been defined by means of A/AN D, O/ON D, S/R D, =D
1A12H -	Transfer error: TDR to a data word that has not been defined
1A13H -	Transfer error: TDL to a data word that has not been defined
1A14H -	Transfer error: TDW to a data word that has not been defined
1A15H -	Transfer error: TDD to a data word that has not been defined

c) Other execution time errors

This category includes all the execution time errors that do not belong to any of the groups of execution time errors already mentioned (transfer errors or calling a block that has not been loaded).

If the system program identifies a transfer error of this category it calls the organization block **OB 31**. The command (or special function) responsible for the transfer error is no longer processed. If OB 31 has not been programmed the processor will go over to the stop state. If processing is to be continued when one of the errors listed below occurs, then the block end statement (BE) in OB 31 is sufficient.

When OB 31 is called, additional information to provide a more detailed explanation of the error is provided in accumulator 1.

Error identifier accum1 accum2	Explanation
1A21H -	Error with GDB, GXDX: data block exists already
1A22H -	Error with GDB, GXDX: illegal data block length (< 5 words or > 4 x 2 ¹⁰ words)
1A23H -	Error with GDB, GXDX: memory location in RAM not sufficient
1A25H -	Error at DI=: illegal parameter in accumulator 1 (< 1 or > 125)
1A29H -	Bracket stack under or overflow after A(, O(,).
1A2AH -	Error with C DB or CX DX: block length in data header too short (length < 5 words)
1A2BH -	Function block has been loaded with wrong PG software
1A2CH -	Error with ACR: Invalid page frame number
1A31H -	Special function error with OB 254 or OB 255 (duplication) or OB 250: destination data block exists already in the DB-RAM
1A32H -	Special function error with OB 254 or OB 255 (copying): destination data block exists already in the DB-RAM
1A33H -	Special function error with OB 254 or OB 255: memory space in DB-RAM not sufficient
1A3AH -	Special function error with OB 221: illegal value for new cycle time (cycle time < 1ms or > 6000 ms)

Error identifier accu1 accu2	Explanation
1A3BH -	Special function error with OB 223: the processors participating in multiprocessor operation use different start-up modes
1A41H -	Special function error with OB 240, OB 241 or OB 242: illegal shift register or data block number (no. < 192)
1A42H -	Special function error with OB 241: shift register not initialized
1A43H -	Special function error with OB 240: memory space in the DB-RAM not sufficient
1A44H -	Special function error with OB 240: data word DW 0 of data block does not have content '0'
1A45H -	Special function error with OB 240: illegal shift register length in DW 1 (not between 2 and 256)
1A46H -	Special function error with OB 240: illegal pointer position or number of pointers
1A47H -	Special function error with OB 120
1A48H -	Special function error with OB 122
1A49H -	Special function error with OB 110
1A50H -	Error with LRW, TRW: The computed memory address <BR + constant> is outside 0 - EDFFH
1A51H -	Error with LRD, TRD: The computed memory address <BR + constant> is outside 0 - EDFEH
1A52H -	Error with TSG, LB GB, LW GW, TB GB, TW GW: the computed linear address <BR + constant> is outside 0 - EFFFH
1A53H -	Error with LB GW, LW GD, TB GW, TW GD: the computed linear address <BR + constant> is outside 0 - EFFEH
1A54H -	Error with LB GD, TB GD: the computed linear address <BR + constant> is outside 0 - EFFCH
1A55H -	Error with TSC, LB CB, LW CW, TB CB, TW CW: the computed page frame address <BR + constant> is outside F400H - FBFFH
1A56H -	Error with LB CW, LW CD, TB CW, TW CD: the computed page frame address <BR + constant> is outside F400H - FBFEH

Error identifier accu1 accu2	Explanation
1A57H -	Error with LB CD, TB CD: the computed page frame address <BR + constant> is outside F400H - FBFCH
1A58H -	Error with TNW, TNB: the source block is not fully contained in one of these areas: 0000-7FFF user memory 8000-DD7F data block RAM DD80-EDFF system RAM (DB0,RI,RJ,RS,RT,T,C) EE00-EFFF flags, process image F000-FFFF I/O
1A59H	Error with TNW, TNB: the destination block is not fully contained in one of these areas: 0000-7FFF user memory 8000-DD7F data block RAM DD80-EDFF system RAM (DB0,RI,RJ,RS,RT,T,C) EE00-EFFF flags, process image F000-FFFF I/O

5.6.3 ADF (Addressing Errors)

An addressing error occurs if an input or output in the process image is called by means of a STEP5 operation for which no I/O module was assigned at the time of the last cold restart (I/O module not plugged-in, defective or not specified in the data block DB 1 of the processor).

The system program will now interrupt the processing of the user program and will call the organization block **OB 25**. After the program in OB 25 has been processed the processor will continue with the next command of the interrupted program, i.e. the STEP5 command responsible for the ADF is not executed.

If OB 25 is not programmed the processor will go over to the stop state if an addressing error occurs unless you have specified a continuation of the program processing in data block DX0 for this particular situation.

Suppressing of the addressing error monitoring is also possible by programming DX0 accordingly.

No error identifiers are written into accu 1 or accu 2 when an addressing error occurs.

5.6.4 QVZ (Acknowledgement Delay)

An acknowledgement delay occurs if an input or output module does not transmit the RDY-signal (ready) within a given time after it has been addressed. Cause of this acknowledgement delay may be a defective module or the removal of the module during operation.

The following acknowledgement delay errors will interrupt the user program and will trigger an organization block call:

1. Acknowledgement delay in the user program when accessing the CP, IP, COR or one of the I/O modules directly via the S5 bus (e.g. with load or transfer commands L/T P...or O...):
The system program will call the organization block **OB 23**.

Accumulators 1 and 2 contain additional information to supply a more detailed explanation of the error.

Error identifier accu1 accu2	Explanation
1E23H yyyyH	Acknowledgement delay (QVZ) in the user program when accessing I/O's yyyy = QVZ-address

2. Acknowledgement delay during the updating of the process image of inputs and outputs and the transfer of interprocessor communication flags:
The system program will call the organization block **OB 24**.

Accumulators 1 and 2 contain additional information to supply a more detailed explanation of the error.

Error identifier accu1 accu2	Explanation
1E25H yyyyH	Acknowledgement delay during the updating of the digital outputs yyyy = address of the output byte which has not acknowledged
1E26H yyyyH	Acknowledgement delay during the updating of the digital inputs yyyy = address of the input byte which has not acknowledged
1E27H yyyyH	Acknowledgement delay during the updating of the interprocessor communication flag outputs yyyy = address of the flag byte which has not acknowledged
1E28H yyyyH	Acknowledgement delay during the updating of the interprocessor communication flag inputs yyyy = address of the flag byte which has not acknowledged

If the organization blocks called are not programmed the processing of the user program will be continued. However, an acknowledgement delay will increase the execution time of the STEP5 command responsible for the acknowledgement delay.

In the case of an acknowledgment delay, the processor reads the value "00H" in order to further process it as a 'substitute' if the QVZ is acknowledged by the user.

If an acknowledgement delay is to stop the processor the stop command STP will have to be programmed in OB 23 or 24.

Triggering of a system stop is possible if a QVZ occurs (acknowledgement delay) by programming DX0 accordingly, even if OB's 23/24 are not programmed.

5.6.5 ZYK-FE (Cycle Time Error)

The cycle time includes the total duration of the processing of the cyclic program. The cycle time preset in the processor may be exceeded e.g. due to incorrect programming, a program loop in a function block, a failure of the clock-pulse generator or system activities such as updating of the process image in connection with large programs.

If the cycle time is exceeded, the system program will interrupt the user program and will call the organization block **OB 26**, and the cycle time will be started again (triggered). If the cycle time expires once again before OB 26 is fully processed, the processor goes into the stop state signalling a double error.

The cycle time is variable (1 through 6000 msec) and can be re-triggered (see above). Irrespective of the cycle time, 150 msec after the cycle time has expired BASP will be signalled if OB 26 is still not fully processed at this moment.

If OB 26 is not programmed the processor will stop unless the pre-setting in DX0 has been altered by the user.

The user can preset the cycle time. This is done either by programming DX0 or by calling the special function organization block OB 221.

The cycle time monitoring in a cyclic program can be "retriggered" by calling SF-OB 222.

No error identifiers are written into accu 1 or accu 2 when a cycle time error occurs.

5.6.6 WECK-FE (Collision of Two Time Interrupts)

If there is a new request for a certain time interrupt OB without the last one being fully processed, the system program will identify a collision of two time interrupts and call the organization block **OB 33**. Pay also attention to Subsection 4.4.2 "TIME INTERRUPTS".

Additional information is contained in accumulators 1 and 2 to give a more detailed explanation of the error.

Error identifier accu1	accu2	Explanation
1001H	0016H	Collision of two time interrupts at OB 10 (10 ms)
1001H	0014H	Collision of two time interrupts at OB 11 (20 ms)
1001H	0012H	Collision of two time interrupts at OB 12 (50 ms)
1001H	0010H	Collision of two time interrupts at OB 13 (100 ms)
1001H	001EH	Collision of two time interrupts at OB 14 (200 ms)
1001H	001CH	Collision of two time interrupts at OB 15 (500 ms)
1001H	001AH	Collision of two time interrupts at OB 16 (1 sec)
1001H	0008H	Collision of two time interrupts at OB 17 (2 sec)
1001H	0006H	Collision of two time interrupts at OB 18 (5 sec)

Remark: The identifier contained in accu 2 represents the level (EBENE) identification of the time interrupt which has caused the error.

If OB 33 is not programmed the processor will stop. However, you have the option of having the program processing continued if a collision of two time interrupts occurs and OB 33 is not programmed, by programming DX0 accordingly.

Note that a renewed call of the error program level "Collision of two time interrupts" which has already been activated does not lead to a double error (DOPP) !

5.6.7 REG-FE (Controller Error)

An error during the processing of the standard function blocks of the controller structure R 64 which is supported by the system program is identified as a controller error.

IMPORTANT!

In contrast to e.g. a collision of two time interrupts which is identified by the system program when a specific time interrupt OB is not started and completely processed (see above) within its appropriate interval (e.g. 100 ms with OB 13), an incorrect processing of the controller program will not be identified and displayed in the **ISTACK** until the program level 'control' is called.

If a controller error occurs the program level 'control' will be exited and the level 'controller error' (level (EBENE): 001CH) with the organization block **OB 34** will be called.

Further reaction of the processor depends on the programming of OB 34:

- a) If OB 34 is not programmed the processor will stop.
Error identification is possible by outputting the ISTACK.
- b) If OB 34 has been programmed the STEP5 program contained in it (e.g. evaluation of accumulators 1 and 2 and the error handling dependent on it) will be processed. After that the controller processing will be continued at the point of interruption.

If controller errors are always to be ignored a block end command BE in OB 34 is sufficient.

If controller processing is to be continued if a controller error occurs and OB 34 has not been programmed you will have to alter the presetting in DX0 correspondingly.

If OB 34 is called additional information will be contained in accumulators 1 and 2 to give a more detailed explanation of the error.

Error identifier accu1 accu2	Explanation
0801H DByyH	Sampling time error yy = number of the corresponding controller data block
0802H DByyH	Controller data block not loaded yy = number of the data block not loaded
0803H FByyH	Controller function block not loaded yy = number of the function block not loaded
0804H FByyH	Controller function block not identified yy = number of function block not identified
0805H FByyH	Controller function block loaded with incorr. PG software yy = function block number
0806H DByyH	Incorrect length of controller data block yy = data block number
0880H yyyyH	Acknowledgement delay (QVZ) during controller processing

The error identifier **REG-FE** will be marked in the control bits at the programmer for all 7 types of errors. The position second to last in the lowest line of the control bit mask is not indicated if you are using a PG without S5-DOS, however, it will be marked as well. In the ISTACK mask of the 'control' level, **REG** has been marked as the cause of the error/fault.

Sampling time error

After the preset sampling time has elapsed the cyclic program will be interrupted at the next **block boundary** and controller processing will be inserted. It is now possible that "longer" blocks requiring too much time will be processed and as a consequence, the controller processing will "get out of step": there is a sampling time error.

A sampling time error can be treated just as the other controller errors (see a) and b)) or can be suppressed by means of a mask. Program processing will then not be interrupted if a sampling time error occurs.

Refer to the description "Compact closed-loop control in the R processor of the S5135U", C79000-B8576-C365-xx, page 5-2 onwards!

A sampling time error may be prevented if you alter the presetting in DX0 "processing of the controller and process interrupt at block boundaries" to "processing of the controller and process interrupt at command boundaries".

5.6.8 ABBR (Abort)

If a stop state is caused in the operating state RUN by means of

- a) Mode selector on the processor from RUN to STOP,
- b) Online function 'PC-STOP',
- c) Switch on the coordinator to STOP (in multiprocessor operation),

the system program will call **OB 28**. The processor will stop when OB 28 has been processed.

IMPORTANT!

The transition to the stop state will take place irrespective of how and whether OB 28 has been programmed.

No error identifiers are written into accu 1 or accu 2.

6 Integrated Special Functions

The operating system of the CPU 928 provides special functions which are called, if required, by means of a conditional (JC OB x) or an unconditional (JU OB x) block call. The organization blocks OB 40 through OB 255 are reserved for these special functions.

These functions are called *integrated* special functions since they are a permanent part of the system program. The user may call these special functions, however, reading or altering them is not possible.

IMPORTANT!

The command JU OB > 39 does not function like a 'real' block change. No interrupts are nested!

Special functions with pseudo command boundaries

Some of the special functions are long-running special functions which contain the so-called pseudo command boundaries, i.e.: the execution of the special function is carried out in several operating steps. If an error (e.g. ZYK) or an interrupt (e.g. time or process interrupt at command boundaries) occurs during the execution of one step then the corresponding organization block will be nested at the end of this step at a pseudo command boundary.

The special functions that contain pseudo command boundaries are marked in the following list.

Error during processing of special functions

A distinction can be made between two groups of special functions based on their individual reaction to errors.

Group 1:

The first group includes all the special functions that cause an error organization block to be called if error occurs. In this block you can program the processor's reaction.

If the processor detects e.g. an incorrect parameter assignment while processing a special function it will recognise an execution time error and call OB 31. If e.g. the special function called does not exist it will recognise a command code error and attempt to call OB 30.

If the error OB's 30 and 31 have not been programmed or contain an STP command then the processor will stop. 'LZF' or 'BCF' will be marked in the control bits and the ISTACK. Error identifiers that supply a more detailed explanation of the error are deposited in the accumulators of the error program level. If OB 30 and OB 31 have been programmed (and do not contain an STP-command) the user program will be continued with the next command after OB 30 or 31 has been processed. In this case the accumulators remain unchanged.

Group 2:

Some special functions use another method to handle special-function-specific errors: they influence the RLO and DSP0/DSP1.

If an error occurs during the processing of these special functions, the RLO is usually set (RLO = 1) and an evaluation is possible by means of JC-command (conditional jump).

Evaluation of the result of logic operation (RLO) is thus possible as a signal for "error" or "no error" and a special error program may be provided to handle any errors which occur.

For some special functions DSP0 and DSP1 are affected by the processing of the special functions and may also be scanned.

Summary of the integrated special functions in the CPU 928

OB 110	Access to the condition-code byte
OB 111	Clear accus 1, 2, 3 and 4
OB 112	Roll up accu
OB 113	Roll down accu
OB 120	Switch on/off "Disable all interrupts"
OB 121	Switch on/off "Disable individual time interrupts"
OB 122	Switch on/off "Delay all interrupts"
OB 123	Switch on/off "Delay individual time interrupts"
OB 160 - 163	counter loop
OB 170	read block stack (BSTACK)
OB 180	variable data block access
OB 181	test data blocks (DB/DX)
OB 190, 192	transfer flags to data blocks
OB 191, 193	transfer data blocks to flag area
OB 200 1), 202 1) 203, 204 1) and 205	functions for multiprocessor operation
OB 216 - 218	page access
OB 220	sign extension
OB 221 2)	set cycle time
OB 222	restart cycle time
OB 223	compare start-up modes in multiprocessor operation
OB 224 2)	transfer interprocessor communication flags in multiprocessor operation as a block
OB 226	read word from system program
OB 227	read check sum of system program
OB 228	read status information of a program level
OB 230 1) - 237 1)	functions for standard function blocks
OB 240	initialize shift register
OB 241	process shift register
OB 242	erase shift register
OB 250 1)	initialize PID-controller
OB 251 1)	process PID-controller
OB 254 1), 255 1)	transfer data blocks (DB/DX) to DB-RAM

1) Special functions with pseudo command boundaries (long-running)

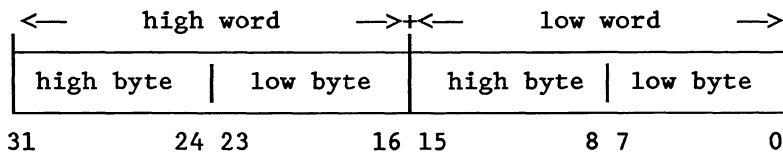
2) The special functions OB 221 and OB 224 were adopted from the S processor for reasons of compatibility and should not be programmed in the CPU 928. Instead, this particular system function should be programmed in the DX 0 (see Chapter 7).

Note:

When specifying the parameters for the individual special function organization blocks make sure you clarify the following notation:

Notation:

Accu 1:	Accu 1,	32 bits
Accu 1-L:	Accu 1, low word	16 bits
Accu 1-LL:	Accu 1, low word, low byte	8 bits
Accu 1-LH:	Accu 1, low word, high byte	8 bits



Note:

All data that the processor requires in order to be able to execute the special functions correctly are listed in the following description of the individual special functions under the heading parameters. Before calling the special function in the STEP5 program you will have to load these data in the accumulators or in the stipulated storage locations.

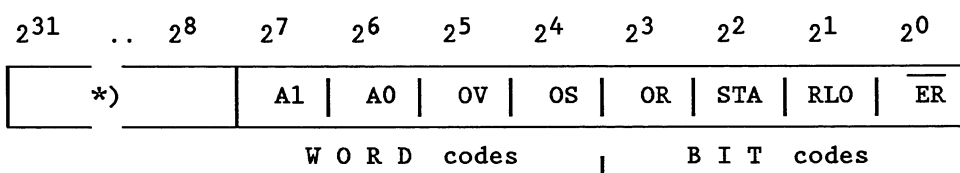
6.1 Handling of the Registers

6.1.1 Access to the Condition Code Byte (OB 110)

The condition code register contains information about the result of an arithmetic or logic operation and can be evaluated via specific commands dependent on the condition code.

Using the special-function organization block OB 110 you can load the condition code register into ACCU 1 or write the content of ACCU 1 into it. In addition, you can set individual condition code bits to "0" or "1".

Occupation of ACCU 1 when accessing the condition code register:



*) Bits 2^8 to 2^{31} are reserved for extensions and must be "0" when writing into the condition code register. They are to be ignored when reading from the condition code register.

Parameters:

1. ACCU 2-L: Function identifier
Possible values 1, 2 or 3
2. ACCU 1: New condition code byte or mask

ACCU 2-L	ACCU 1 before	ACCU 1 afterw.	Function:
1	new condition code byte	new condition code byte	The content of ACCU 1 is loaded into the condition code register
2	mask	new condition code byte 1)	Every bit marked with "1" in the mask of ACCU 1 is set to "1" in the condition code register. The new condition code byte is loaded into ACCU 1.
3	mask	new condition code byte 1)	Every bit marked with "1" in the mask of ACCU 1 is set to "0" in the condition code register. The new condition code byte is loaded into ACCU 1.

1) Restriction:

The bit codes OR, STA and ER cannot be read since they are constantly influenced by special function OB 110:
OR = 0, STA = 1 and ER = 0

Possible errors:

- Function identifier in ACCU 2-L does not equal 1, 2 or 3.
- In ACCU 1, one of the bits from 2^8 to 2^{31} is set.

In case of an error, OB 31 (other execution time errors) is called and the error identifier 1A49H is transferred to ACCU 1-L.

Example of application

The OB 110 can be used as an aid to test those commands which evaluate or influence the condition code register. Testing commands, however, is not the only application of OB 110. The following example shows further possible applications.

Distribution of calls

Depending on the contents of flag byte FY0, one out of four subprograms is to be called. These four subprograms are assigned bits F0.0 through F0.3. Only one of these bits may be set.

```

:L    FB0
:SLW  4           ;shift F0.0 through F0.3 by 4 to the left
:L    KB1         ;load function identifier
:TAK
:JU   OB110
:JOS  =F000       ;jump if OS = 1
:JO   =F001       ;jump if OV = 1
:JM   =F002       ;jump if DSP0 = 1
:JP   =F003       ;jump if DSP1 = 1
:.
:.               ;if no bit has been set
:.
:BEU
:
F000:..          ;if F0.0 = 1
:.
:BEU
F001:..          ;if F0.1 = 1
:.
:BEU
F002:..          ;if F0.2 = 1
:.
:BEU
F003:..          ;if F0.3 = 1
:.
:BEU

```

B8576633-01

6.1.2 Clear Accus 1, 2, 3 and 4 (OB 111)

By a single call of special-function organization block OB 111, you can erase the contents of accus 1 through 4: the four registers are overwritten with '0'.

Parameters: none

Possible errors: none

6.1.3 Roll Up Accu (OB 112) and Roll Down Accu (OB 113)

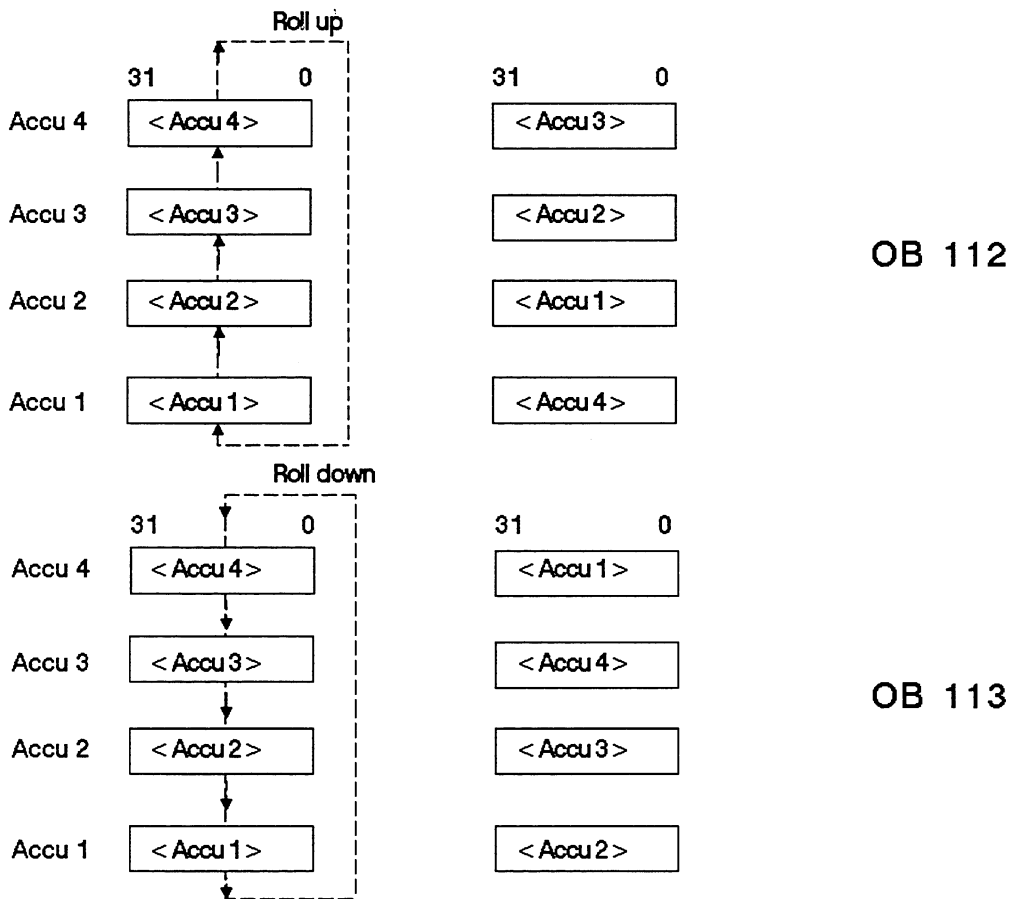
Using OB 112 and OB 113, you can have the contents of the accus scrolled up or down:

- OB 112 (Roll UP) moves the contents of accu 1 into accu 2, the contents of accu 2 into accu 3, etc.
- OB 113 (Roll Down) moves the contents of the accus into the opposite direction: the contents of accu 1 into accu 4, accu 4 into accu 3, etc.

Parameters: none

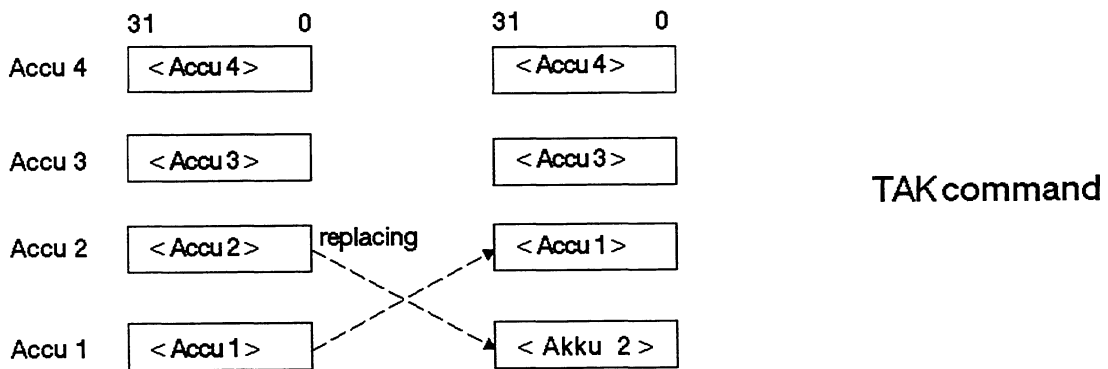
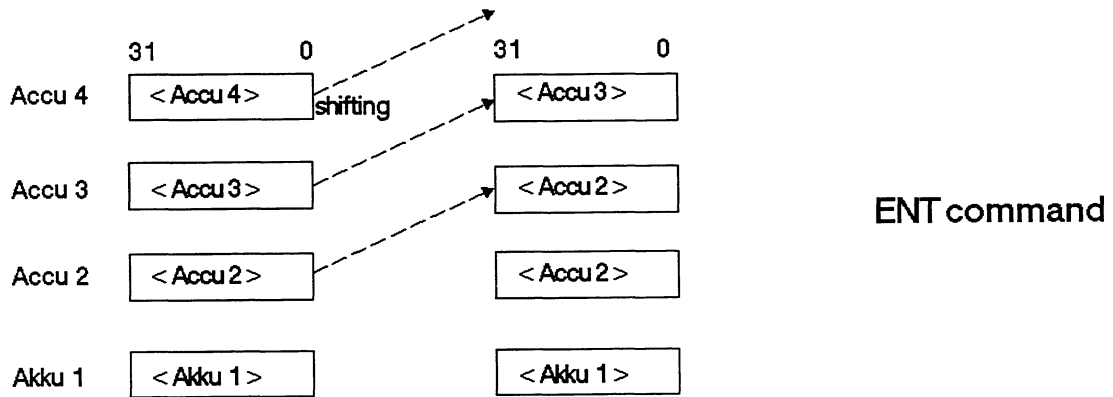
Possible errors: none

The following figure shows the contents of the accus before and after OB 112 and OB 113 are called.



Using the STEP5 commands ENT (Supplementary Operations) and TAK (System Operations), the contents of the accu can also be shifted.

Note the following different features:



6.2 Structure Commands

6.2.1 Counter Loops (OB 160 through 163)

Using these special function organization blocks, you can implement program loops with extremely favorable execution times.

Each of the 4 special function OB's is assigned a specific system data word:

RS 60	-->	OB 160
RS 61	-->	OB 161
RS 62	-->	OB 162
RS 63	-->	OB 163

You transfer the number of times the loop is to be run (iterations) into this system data word. If you now call the corresponding special function OB the loop counter in the system data word will be decremented by one. The loop is run until the loop counter reaches the value zero.

(If the loop counter contains the value zero before the special function OB is called it will still be decremented by one if a call is executed: the loop will be run 65,536 times!)

Loop counter in system data word > 0: RLO is set (RLO = 1)
 Loop counter in system data word = 0: RLO is cleared (RLO = 0)

The remaining bit and word condition codes are always cleared.

The accumulators are not altered and not evaluated. This means that they are still available at the beginning of the next iteration and will not have to be generated again.

The four organization blocks OB 160, 161, 162, and 163 allow a quadruple nesting of loops. This in turn means that you can have four different loop counters in the system data words RS 60 through 63.

If required, these special functions may be used in combination with command DO RS (process system data).

Execution time of the special function OB 160 through 163, if the loop counter was # 1 before the function was called: 12 - 16 μ s

Parameter:

1. System data word RS 60 - 63: loop counter
 possible values: 0 - 65,535 (FFFFH)

Possible errors: none

Example:

The desired number of iterations is contained in flag word x.

```
Initialize loop:          :L KBO
                        :L FWx          loop counter
                        :!=F
                        :JC =M002
                        :T RS62        transfer loop counter to
                                       system data word

'Loop program':         F001: .
                        : .
                        : .
                        : .
                        :
Manage loop:             :JU 0B162     counter loop
                        :JC =F001     with RLO = 1, the loop
                                       will be run again

Continue:               F002: .
                        : .
                        : .
                        : .
```

(Refer to Chapter 9.2 "TNW and TNB: transfer memory blocks" for a further example.)

6.3 Read Block Stack (BSTACK) (OB 170)

All the blocks are entered in the block stack, starting with OB 1 or FB 0, that have been called in succession and have not yet been completely processed.

You may read the entries contained in the BSTACK into a data block by means of the special function organization block OB 170. This allows you to determine the number of existing BSTACK entries and thus the space still available for further entries.

You will be supplied the return address (step address counter = SAC), the absolute start address of the data block (DBA) valid in this particular block as well as its length (number of data words = DBL) for every entry.

IMPORTANT!

Before calling OB 170 a sufficiently long data block (DB or DX) must be called first! Four data words will be required for every BSTACK entry.

Parameters:

1. Accumulator 2-L: Number of data word (DW n), starting with which the entries are to be written in the DB called (offset)

2. Accumulator 1-L: required number of BSTACK elements
possible values: 1 - 62

Ex.: If the accumulator 1-L contains the value '1', you will be supplied the last BSTACK entry; for '2', the last and second last, etc.

After successful call of OB 170

- the offset in the data block will remain in accumulator 2-L,
- the number of BSTACK elements actually displayed is in accumulator 1-L,

possible values: 0 - 62, with

displayed number \leq required number

0 = 'no BSTACK entry exists' or 'error'

(Contents of accumulator 1-L multiplied by 4 is the number of data words written in the DB called!)

- the RLO will be cleared.
- the result condition codes DSP0 and DSP1 can be evaluated (see below),
- all other bit and word condition codes are cleared.

Possible errors:

- no data block called
- data block called does not exist or length insufficient
- illegal parameters in accumulators 1 and 2

If an error occurs the RLO as well as the result condition codes DSP0 and DSP1 will be set (RLO, DSP0 and DSP1 = 1), the remaining bit and word condition codes are erased. The contents of accumulator 1 will be set to '0'.

Influencing the result condition codes RLO, DSP0 and DSP1

RLO	DSP0	DSP1	Scan with	Significance:
0	1	0	JM	actual number of BSTACK elements < desired number
0	0	0	JZ	actual number of BSTACK elements = desired number
0	0	1	JP	actual number of BSTACK elements > desired number
1	1	1	JC	error

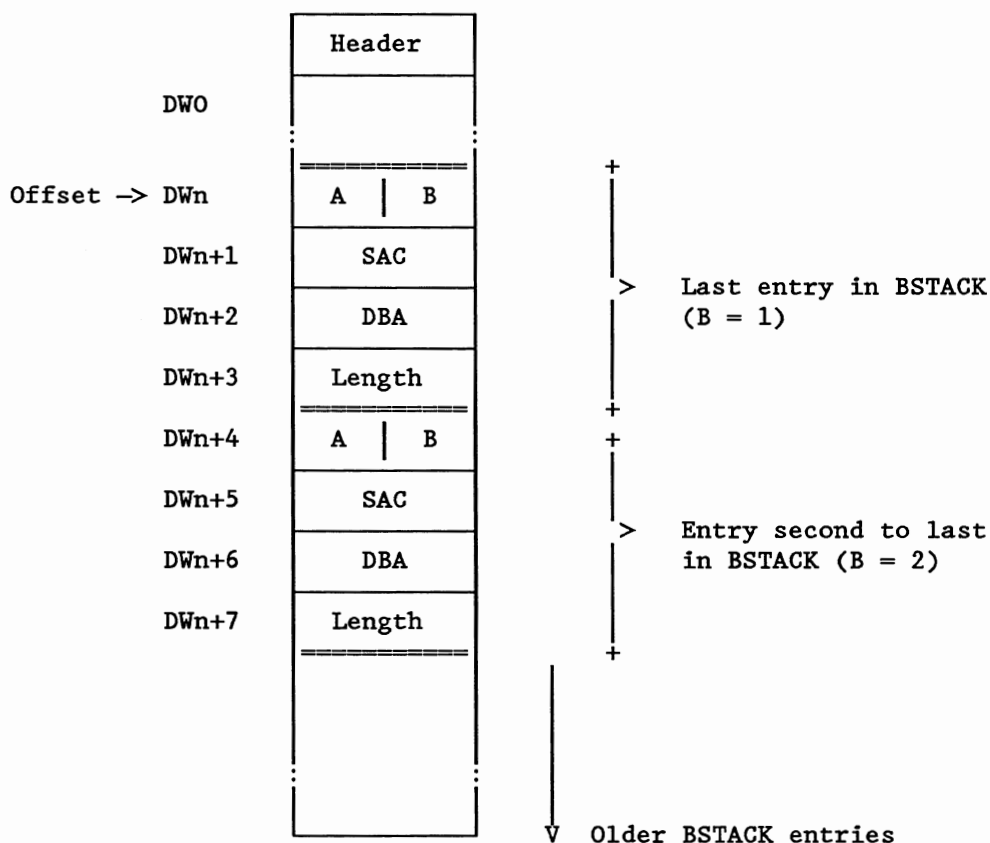
This is how the contents of the BSTACK, upon call of OB 170, are written into the data block called:

A = BSTACK element number (62 - 1)

(Already at the time of the output of the last BSTACK element, calculation of the reserve is possible:

A = 17 --> reserve = A - 1 = 16.)

B = depth of BSTACK element (1 - 62)



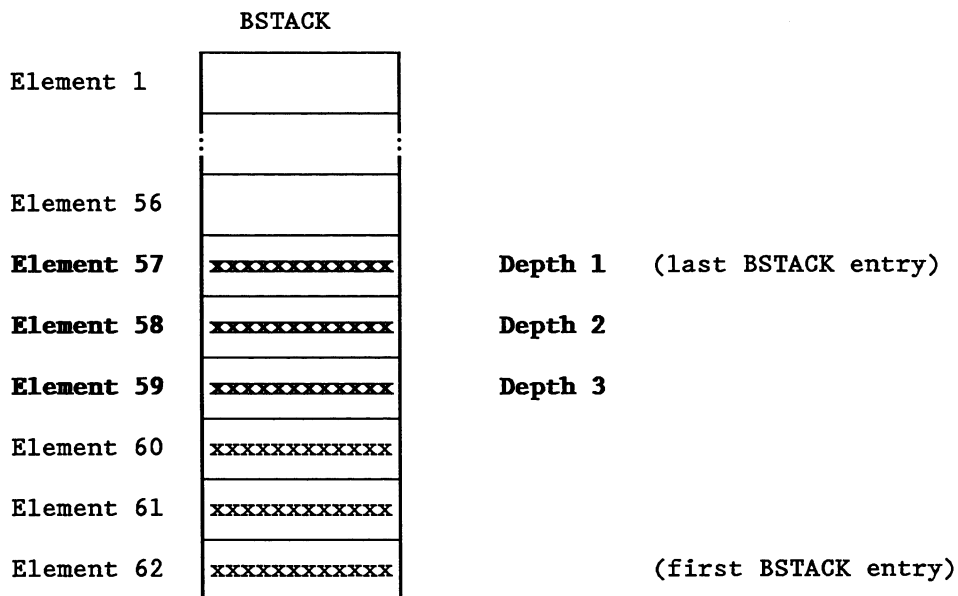
Example:

You want to read the last 3 BSTACK entries into data block DX 10. These entries should be written in from data word DW 16 in DX 10.

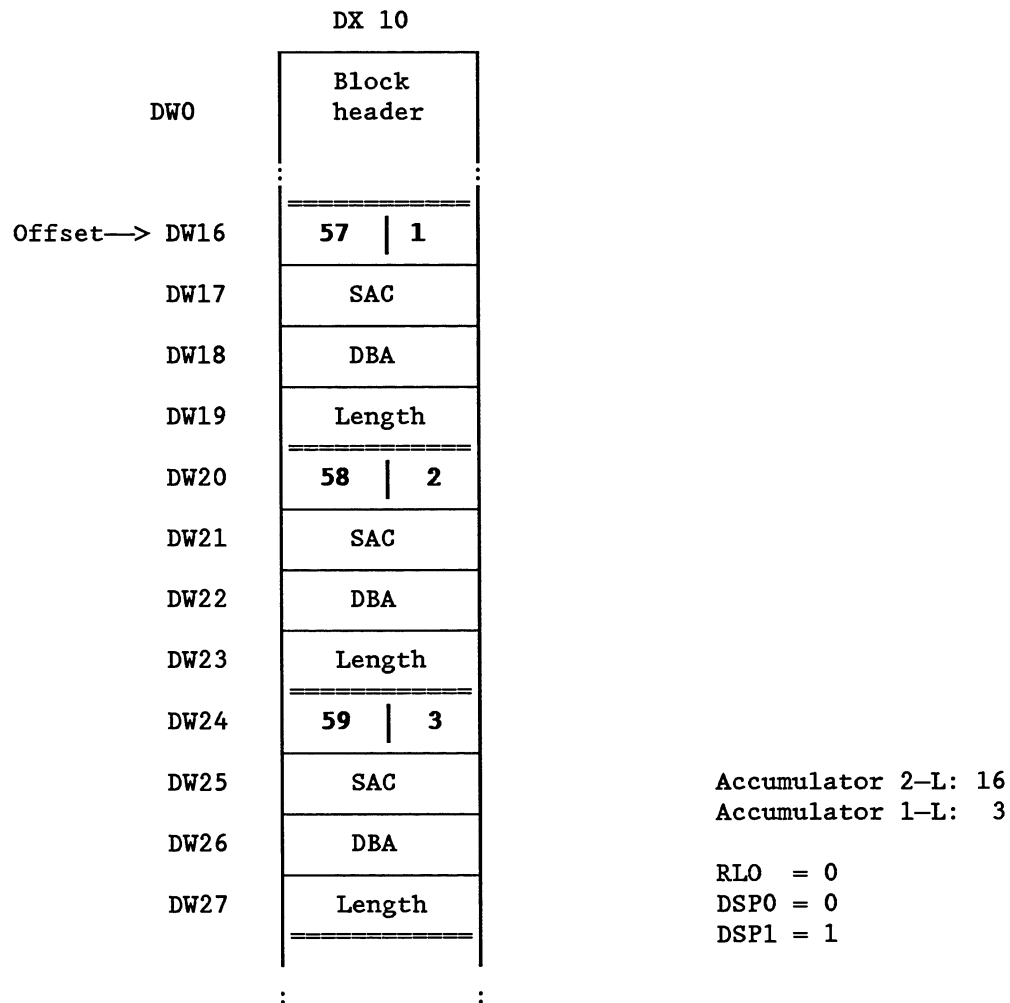
```

:CX DX 10 ;call DX 10
:L KY 16 ;BSTACK entries should be written starting with
          DW 16
:L KY 3 ;the last 3 BSTACK entries are required
:JU OB 170
    
```

6 blocks have been entered in the BSTACK:



After calling the special function OB, DX 10 is occupied as follows:



6.4 Block Handling

6.4.1 Variable Data Block Access (OB 180)

When calling a data block by means of C DB and CX DX commands, the 'DBA' register (data block start address, register no. 6) is loaded with the address of the data word DW 0 contained in DB0. The user can access the DBA register (16 bit) by means of the STEP5 program (LIR 6, TIR 6).

Access to data blocks by means of commands such as L DR 60 or D0 DW 240 etc. are always carried out relative to the data block start address.

IMPORTANT!

STEP5 access to data words is only permitted up to data word DW 255!

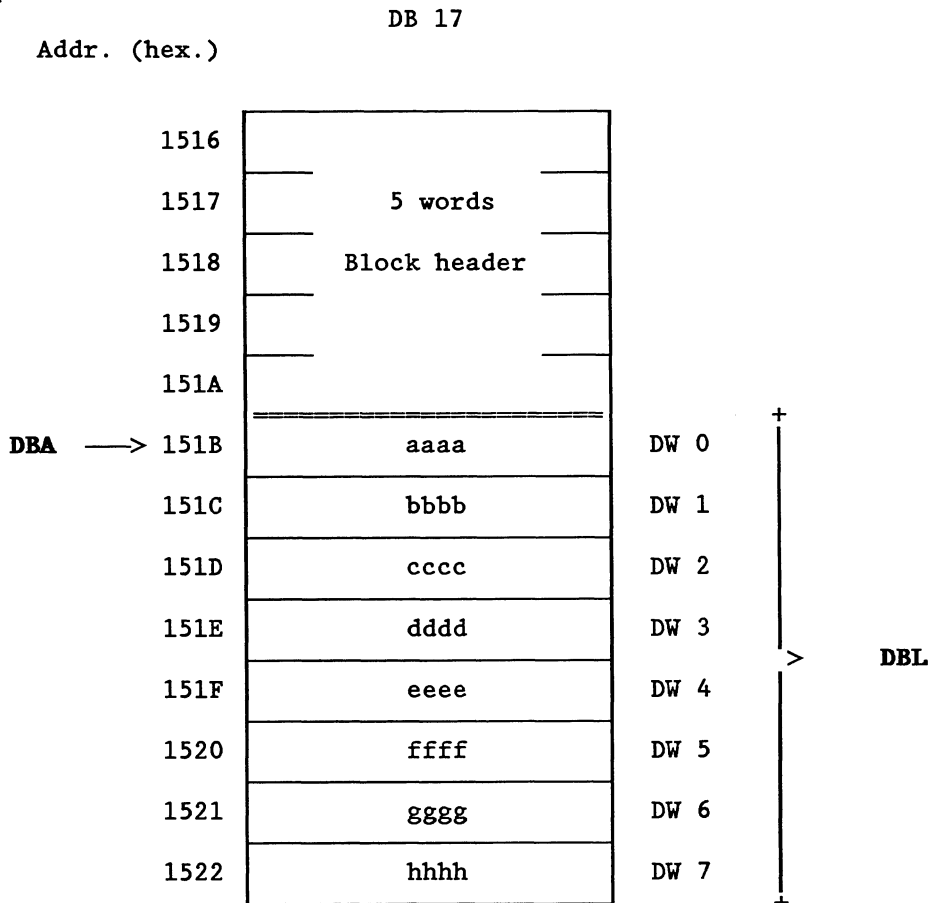
In addition to the DBA register, the 'DBL' register (data block length, register no. 8) is loaded every time a data block is called: it contains the length (in words) of the DB or DX data block opened without the block header.

The user can access the DBL register (16 bit) by means of the STEP5 program (LIR 8, TIR 8).

IMPORTANT!

A maximum length of 4091 data words may be entered in the DBL register - depending on the memory size of the PG used!

Example:



B8576633-01

The DBA register contains the address of the memory in which DW 0 is stored, for our example: DBA = 151B (hex.).

The DBL register contains the number of data words, for our example: DBL = 8 (DW 0 through DW 7).

Since access to data words by means of STEP5 commands L DW, A D, DO DW etc. is always carried out relative to the DBA, e.g. 3 is added to 151B in order to access DW 3. The address 151E contains DW 3. For writing access the DBL register is used to check whether a transfer error exists. E.g. T DW 7 is permissible, T DW 8, however, is incorrect.

The special function OB 180 shifts the data block start address by a preset number of data words. This means that you may also access data blocks longer than 256 data words by means of STEP5 commands. The STEP5 access range of 256 data words may be shifted as required within a data block by assigning suitable parameters and then calling OB 180.

IMPORTANT!

Before OB 180 is called a data block of sufficient length (DB or DX) must be called first.

Parameters:

1. Accumulator 1-L: offset (number of data words by which the data block start address is to be shifted)
- possible values: contents of accumulator
1-L < DBL !

After successful call of OB 180

- the value of the DBA register (= address of DW 0) will be increased by the value of accumulator 1-L,
- the value of the DBL register will be reduced by the value of accumulator 1-L,
- the RLO will be cleared (RLO = 0),
- all other bit and word condition codes will be cleared.

Errors:

- no data block opened
- contents of accumulator 1-L \geq DBL

If an error occurs (contents of accumulator 1-L \geq DBL) the DBA and DBL registers will not be affected. The RLO is set (RLO = 1). The remaining bit and word condition codes are erased.

If the DBL register contains the value '0' OB 180 recognizes that no data block has been opened. The RLO is set (RLO = 1) and an error is thus signalled.

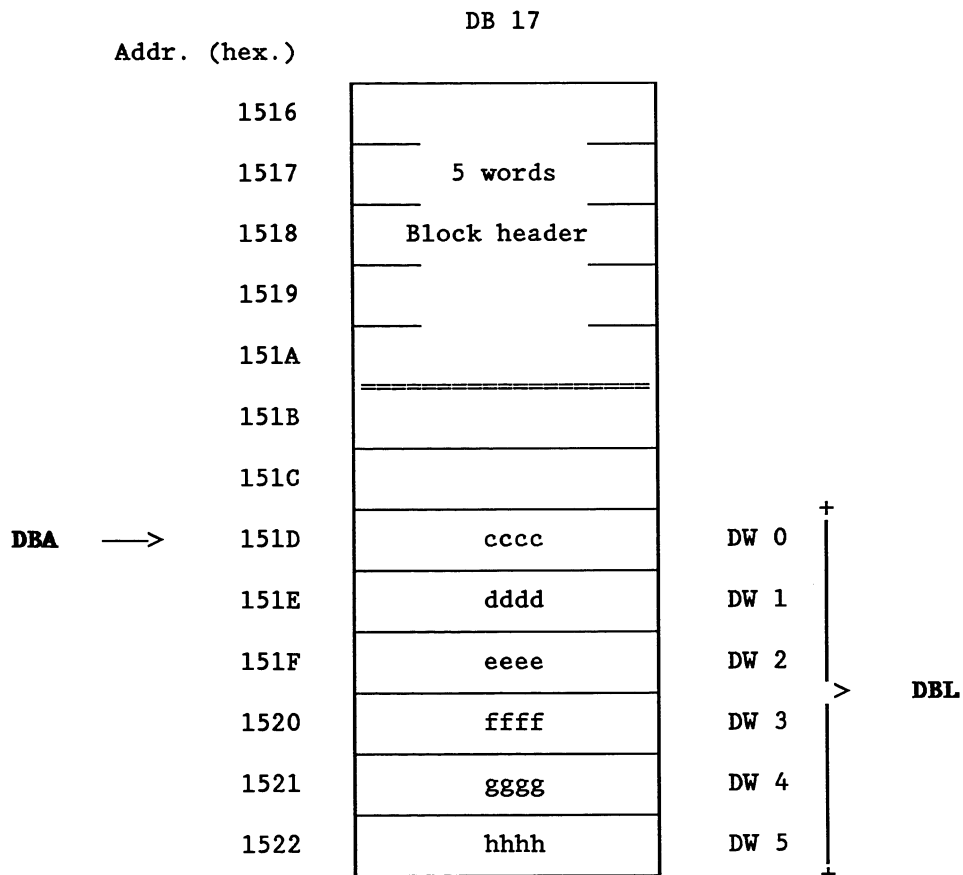
Reset DBA and DBL to the initial value

Calling the data block again by means of the commands C DB or CX DX will recreate the initial state.

Example:

The data block start address (DBA = 151B) in DB 17 (DBL = 8) is to be shifted by two data words.

C DB 17
L KB 2
JU OB 180



DBA after call of OB 180 = 151D (hex.)
 DBL after call of OB 180 = 6 (DW 0 through DW 5)

After OB 180 has been called, the data word stored under the address 1520 and with the contents 'ffff' can no longer be addressed with DW 5, but with DW 3 instead etc..

Since the DBL is altered at the same time, the **transfer error monitoring is guaranteed**: the command T DW 5 is permissible, however, the command T DW 6 is incorrect.

A further increase of the DBA (which means that the DBL will be again reduced further) is possible by calling OB 180; the C DB 17 command will recreate the initial state (DBA = 151B, DBL = 8).

If, DB 17 had a length of e.g. 258 data words, then access to DW 256 and DW 257 would no longer be possible by means of STEP5 commands. If the DBA is shifted by 2, this means that data words 256 and 257 will now be addressed with "DW 254" and "DW 255".

Possible applications of OB 180

- Accessing DB's which are too long, i.e. for DB's with a length of more than 261 words (5 words header, DW 0 - DW 255), see above.
- Handling data structures

If a record, which consists of several data words, exists more than once within a processor and the assignment (significance) is always the same, then this is referred to as a data structure.

E.g. the description of a (partial) process status could take 20 data words, where the 1st data word is temperature, the 2nd pressure etc.. If this process status is to be stored **more than once and within a DB**, e.g. because the partial process exists more than once and/or because historical values are to be stored, then the special function-OB, OB 180 allows access to each of the structures with the same commands L DD, S D, T DR etc. with the same parameters 0 through 19.

In contrast to other substitution mechanisms (substitution = indexed parameter assignment) the subroutines are simpler and have better execution times.

(Refer to Chapter 9 "memory access via absolute addresses", register 6 for the DBA register.)

6.4.2 Test Data Blocks (DB/DX) (OB 181)

Using the special function organization block OB 181 allows you to check,

- a) whether a certain DB or DX data block exists,
- b) the address of the first data word of the data block,
- c) how many data words this data block contains,
- d) the memory type and area (user memory: RAM or EPROM, DB-RAM).

Use of the function "test DB/DX" is advisable before the commands TNB/TNW, G DB/GX DX are input and before the special function organization blocks OB 254 and OB 255 are called.

You might, for example, call OB 181 before making a block transfer of data words in order to ensure that the destination data block is valid and long enough to accept all the data words to be copied.

Parameters:

1. Accumulator 1-LL: block number
possible values: 1 through 255
2. Accumulator 1-LH: block identifier
possible values: 1 = DB
2 = DX

If the block tested exists in the processor,

- accumulator 1-L will contain the address of the first data word (DW 0),
- accumulator 2-L will contain the length of the data block in words (without header),
example: The value '7' is entered in accumulator 2-L --> Data block consists of DW 0 through DW 6.
- the RLO will be cleared (RLO = 0),
- the word condition codes DSP0 and DSP1 are affected (see the following list),
- the remaining bit and word condition codes are cleared.

If the block tested does not exist in the memory or the parameter assignment is incorrect,

- the RLO will be set (RLO = 1),
- the word condition codes DSP0 and DSP1 are affected (see the following list),
- the remaining bit and word condition codes are cleared,
- the accumulators are not altered.

Errors:

- incorrect block number (illegal: 0)
- incorrect block identifier (illegal: 0, 3 through 255)
- storage error

Summary: Effect on the result condition codes RLO, DSP0 and DSP1

RLO = 0: DB exists
 RLO = 1: DB does not exist or error

DSP1 = 0: DB in the user module
 DSP1 = 1: DB in the DB-RAM

DSP0 = 0: DB in the read/write memory
 DSP0 = 1: DB in the read-only memory

RLO	DSP0	DSP1	Scan with	Significance
0	1	0	JM	\ } DB in the EPROM (read only) } > DB exists > in the user module
0	0	0	JZ	
0	0	1	JP	\ } DB in the RAM (read/write) } > > DB in the DB-RAM
1	1	1	JC	

Examples see
 Section 8.2.2 "block address lists in the DB-RAM"
 Section 9.1 "LIR and TIR: access to registers"
 Section 9.2 "TNW and TNB: transfer memory blocks"

6.4.3 Transfer Flags to Data Block (OB 190 and OB 192)

Organization blocks OB 190 and OB 192 transfer a number of flag bytes as specified by the user to a data block intended for this purpose.

This may be useful before block calls, in error organization blocks or when a cyclic program is interrupted by time or interrupt driven program execution.

You can have these flag bytes transferred back from the data block afterwards by OB 191 and 193.

IMPORTANT!

Before the actual call is executed a data block (DB/DX) must be opened first!

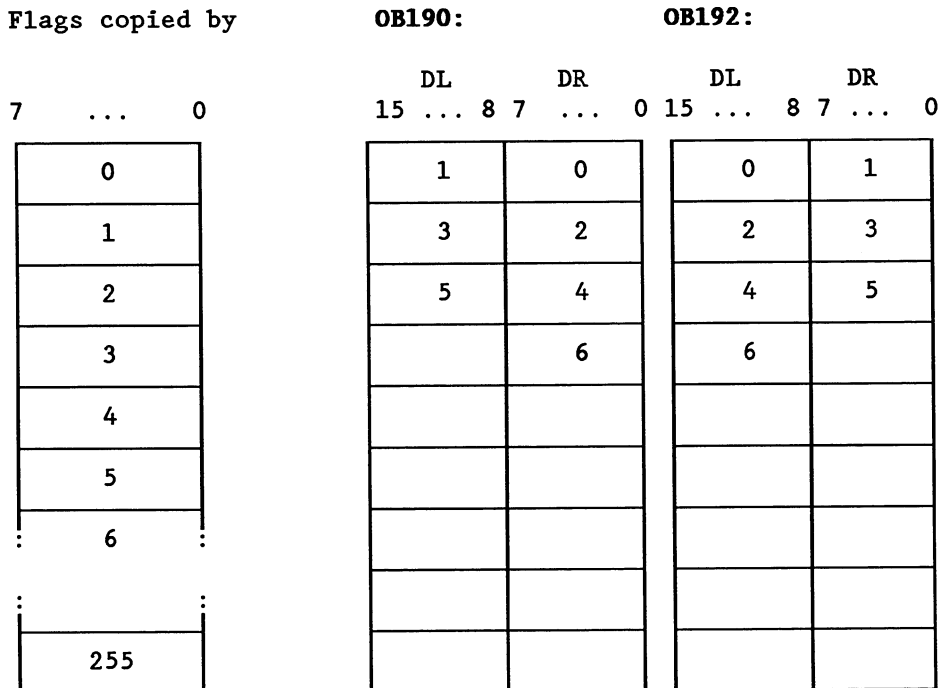
After OB 190/192 has been called, the flag bytes are buffered in the data block called beginning with the data word address specified. OB 190/192 will read the area of the flags to be saved from the accumulator.

The special function organization blocks OB 190 and OB 192 are identical, except for the manner in which they transfer the flag bytes:

- * OB 190 transfers the flags byte by byte.
- * OB 192 transfers the flags word by word.

This is important if the data that have been transferred to the data block are to be processed subsequently and the data block is not only used as a simple buffer.

The following figure will help to illustrate this difference:



Note: If an odd number of flag bytes is transferred then only half of the data word of the data block used last will be used. With OB 190 the data on the left-hand side will remain free, and with OB 192 the data on the right-hand side.

IMPORTANT!

Please note the following execution time table (in μ s):

(n = number of flag bytes)	OB 190	OB 192
Number of 1st flag byte is even	$25 + n * 0.32$	$40 + n * 0.57$
Number of 1st flag byte is uneven	$25 + n * 0.48$	$25 + n * 1.8$

Parameters:

Specifying the source:

1. Accumulator 2-LH: First flag byte to be transferred
possible values: 0 through 255
2. Accumulator 2-LL: Last flag byte to be transferred
possible values: 0 through 255

(Last flag byte \geq First flag byte !)

Specifying the destination:

3. Accumulator 1-L: Address of the first data word to be written into in the data block called

If the special function block OB 190/192 is processed correctly the RLO will be cleared (RLO = 0). The accumulators will not be altered.

The RLO will be set (RLO = 1) in the case of an error, the accumulators will not be altered.

Errors:

- no DB or DX data block has been called
- incorrect flag area (last flag byte < first flag byte)
- address of data word does not exist
- DB or DX data block not long enough

6.4.4 Transfer Data Fields in Flag Area (OB 191 and OB 193)

Using OB 191 and OB 193 you can transfer data from a data block to the flag area. Thus, it is possible to transfer the flag bytes previously 'saved' in a data block back to the flag area.

OBs 191/193 only differ from OBs 190/192 insofar as source and destination are reversed:

OB 190/192: flag area → flag → data block

OB 191/193: flag area ← data ← data block

IMPORTANT!

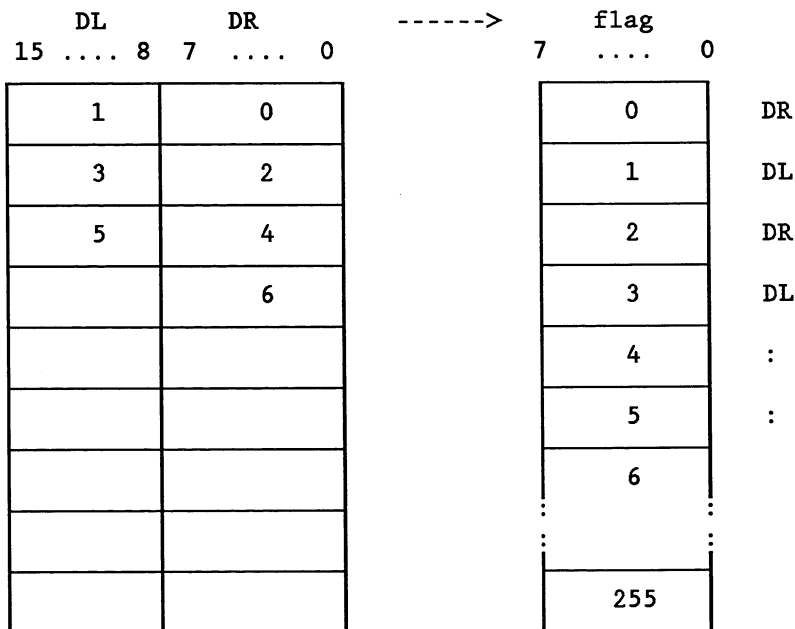
Before carrying out the actual call a data block (DB/DX) must be opened first!

The special function organization blocks OB 191 and OB 193 are identical, except for the manner in which they transfer data:

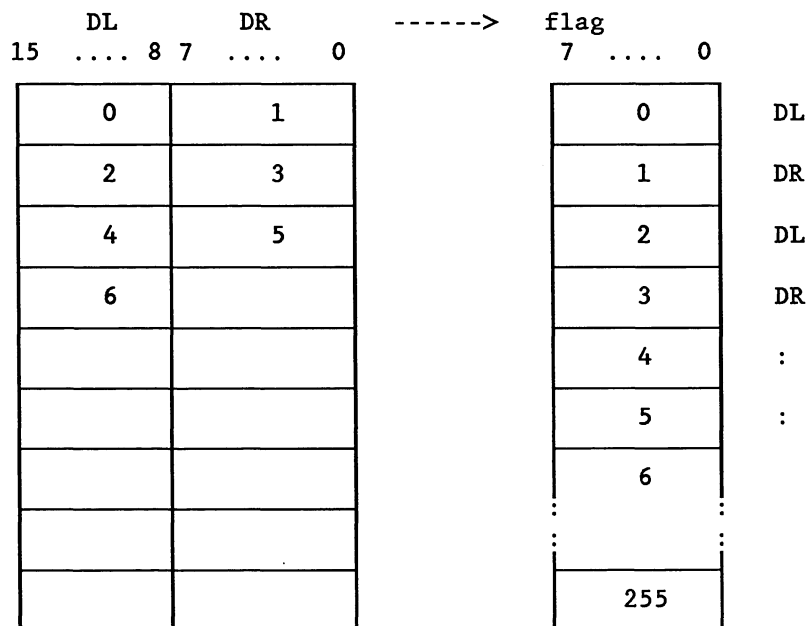
- * OB 191 transfers the data words byte by byte.
- * OB 193 transfers the data words word by word.

The following figure will help to illustrate this difference:

Data transferred by means of **OB191:**



Data transferred by means of **OB193:**



IMPORTANT!

Please note the following execution time table (in μ s):

(n = number of flag bytes)	OB 191	OB 193
Number of 1st flag byte is even	$25 + n * 0.32$	$40 + n * 0.57$
Number of 1st flag byte is uneven	$25 + n * 0.48$	$25 + n * 1.8$

Parameters:

Specifying the source:

1. Accumulator 2-L: Address of the first data word in the data block opened which is to be transferred

Specifying the destination:

2. Accumulator 1-LH: First flag byte to be written into possible values: 0 through 255
3. Accumulator 1-LL: Last flag byte to be written into possible values: 0 through 255

(last flag byte \geq first flag byte !)

If the special function block OB 191/193 is processed correctly the RLO will be cleared (RLO = 0). The accumulators will not be altered.

The RLO will be set (RLO = 1) in the case of an error, the accumulators will not be altered.

Errors: see OBs 190/192

Example:

Before the program block PB 12 is called all flags (FY0 through FY255) are to be saved in data block DX 37, starting at address 100. They are then to be written back again.

```
Save:           :CX DX37      call data block
                :L KYO,255    flag area FY0 through FY255
                :L KB100      address of 1st data word
                :JU OB190     save flags
```

```
Block change:  :JU PB12
```

```
Rewrite:       :           (data block called already)
                :L KB100      address of 1st data word
                :L KYO,255    flag area FY0 through FY255
                :JU OB191     rewrite flags
```

OB 190 / OB 191: Example of an application

Flags that are used in a cyclic user program may not be used by another time or interrupt-driven user program. Each program level must be assigned a certain section of the flag area.

```
Ex.: cyclic user program:      FY0...FY99
      time-driven user program: FY100...FY199
      interrupt-driven user program: FY200...FY255
```

However, if in the cyclic user program all the 256 flag bytes are already in use and if for instance the time-driven user program requires all the 256 flag bytes too, the flags must be exchanged and buffered when the program level is changed.

OB13

```
C DB100
FY 0...255  -> DW 0...127
DW 128...255 -> FY 0...255

time-driven
user program

C DB100
FY 0...255  -> DW 128...255
DW 0...127  -> FY 0...255
BE
```

DB100

```
DW0
:
:
:
DW127
DW128
:
:
:
DW255
```

Flag of the cyclic user program

Flag of the time-driven user program

The quickest way to save and load these flags is offered by the special functions OB 190 and OB 191:

STEP5 program in OB 13

```

:C  DB100
:L  KY0,255
:L  KB0
:JU OB190
:L  KB128
:L  KY0,255
:JU OB191
:.
:.
:.
:C  DB100
:L  KY0,255
:L  KB128
:JU OB190
:L  KB0
:L  KY0,255
:JU OB191
:BE
    
```

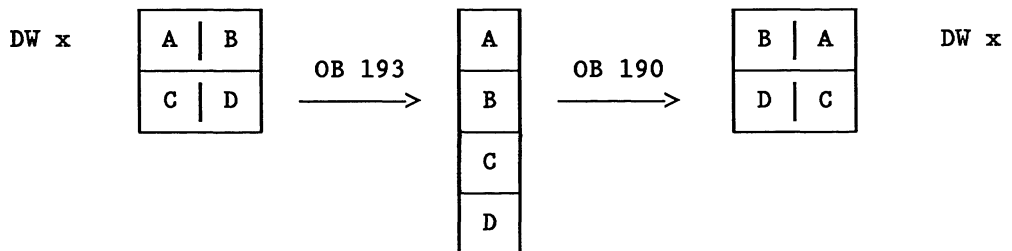
The time needed to exchange and buffer the flags is 704 μs for each call of OB 13.

Further use of organization blocks OB 190 through OB 193

- In the case of the CPU 928, the commands for single bit processing (A, O, ON, AN, S, R, =) which access the flag area are processed considerably quicker than comparable commands which access data blocks (see e.g. the 'A M' <-> 'A D' or 'S M' <-> 'S D'! commands).

Due to this you may improve the execution time if you copy data into the flag area, have them processed there and then return them to the data block.

- Reversing of high byte and low byte is possible without much trouble by transferring the data words to the flag area and back by means of the OB's:



B8576633-01

- 'Shifting' of data fields within a data block is possible if you specify the same DB number but a different data word when writing back from the flag area.
- It is just as simple to transfer data fields (max. 255 bytes) to other data blocks (it may be useful to use OB 191 'test data blocks (DB/DX)' first).

6.4.5 Transfer Data Blocks to DB-RAM (OB 254, OB 255)

The special function organization blocks OB 254 and OB 255 allow you to transfer data blocks from the user memory to the DB-RAM (data block storage) of the processor. The execution of OB 254 and OB 255 is identical, with OB 254 being responsible for DX blocks and OB 255 for DB blocks.

Use of these special functions is only permissible for a block length of max. 4K-words (incl. header).

When transferring, you can shift or duplicate data blocks:

- **Shifting** a data block from the user memory to the DB-RAM

A data block in the user memory is shifted to the DB-RAM, the original block number is retained. The new start address of the data block is entered in the address list in DB 0. The old address of the block is overwritten, i.e. the data block is declared invalid in the user memory.

Parameters:

1. Accumulator 1-L: number of data block to be shifted
2. Accumulator 1-H: 0

Errors:

- The data block to be shifted does not exist.
- The block exists already in the DB-RAM.
(Execute function only once - preferably during the start-up)
- The memory location in the DB-RAM is insufficient.

The function will not be executed if an error occurs. The system program will identify an execution time error and will call OB 31. Further reaction to the error depends on the programming of OB 31 (see 'Other execution time errors').

If OB 31 has not been programmed the processor will go over to the stop state. Error identifiers are written into accumulator 1 and provide a more detailed explanation of the error.

- **Duplication** of a data block in the DB-RAM

A data block in the user memory or in the DB-RAM is transferred to the DB-RAM, where it will be assigned a new block number. The start address of the new data block is entered in the address list in DB 0. The start address of the old block in DB 0 is retained, i.e. the original block is still valid.

The start address is not entered in DB 0 until the transfer has been completely executed and all identifiers have been entered correctly in the header. This means that the block duplicated is not identified as valid or existing by the system program until the transfer has been completely executed.

Parameters:

1. Accumulator 1-L: number of data block to be duplicated
2. Accumulator 1-H: number of new data block

Errors:

- The data block to be duplicated does not exist.
- The new data block exists already.
- Not enough space in the DB-RAM.

The function will not be executed if an error occurs. The system program identifies an execution time error and will call **OB 31**. Further reaction to the error depends on the programming of OB 31 (see 'Other execution time errors'). If OB 31 has not been programmed the processor will go over to the stop state. Error identifiers are written into accumulator 1 and provide a more detailed explanation of the error.

6.5 Multiprocessor Communication (OB 200 through OB 205)

The special function organization blocks OB 200 through OB 205 allow data transfer between the individual processors in multiprocessor operation, using coordinator KOR C.

OB 200: Initialize

This special function organization block initializes the memory in the coordinator KOR C which is used to buffer the data blocks to be transferred.

OB 202: Transmit

This function transfers a data block to the buffer of the KOR C and specifies the number of data blocks that can still be transmitted.

OB 203: Transmit-test

The special function OB 203 determines the number of free memory blocks in the buffer of the coordinator KOR C.

OB 204: Receive

This function receives a data block from the buffer of KOR C and displays the number of data blocks that can still be received.

OB 205: Receive-test

The special function OB 205 determines the number of memory blocks occupied in the buffer of the KOR C.

Detailed operating instructions for these particular special function organization blocks can be found in section 8 of this manual.

6.6 Page Access

Organization blocks OB 216 through 218 allow for access to so-called pages.

The organization blocks contain the following functions:

- OB 216 Writing a byte/word/double word to a page
- OB 217 Reading a byte/word/double word from a page
- OB 218 Assigning a page by the processor
(for coordination in multiprocessor operation)

On the one hand, these functions are used for test purposes; at the same time these basic functions allow handling blocks or similar functions to be programmed.

Use of these special function OB's is advisable for multiprocessor operation if information from different processors is to be written on one page or read from one page.

What are pages?

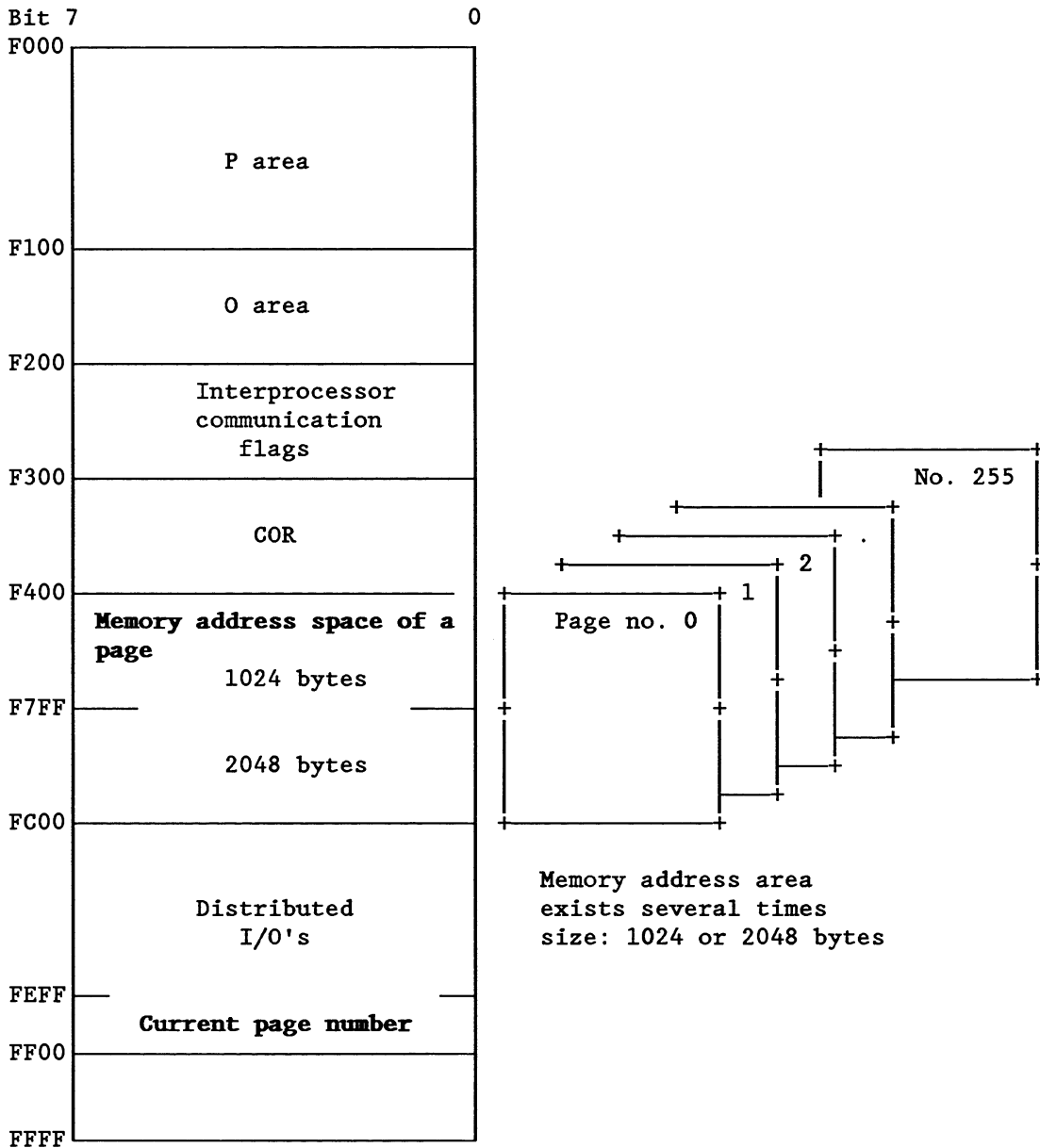
Pages are memory areas which exist once or several times on communications processors, certain intelligent I/O modules and certain coordinators for multiprocessor operation.

Pages are organized in bytes, i.e. each byte may be addressed **individually**.

A max. of 255 pages is possible in the programmable controller.

Page size:	Memory address space assigned
1024 bytes	F400H - F7FFH
2048 bytes	F400H - FBFFH

Memory address area for interface system on the S5 bus



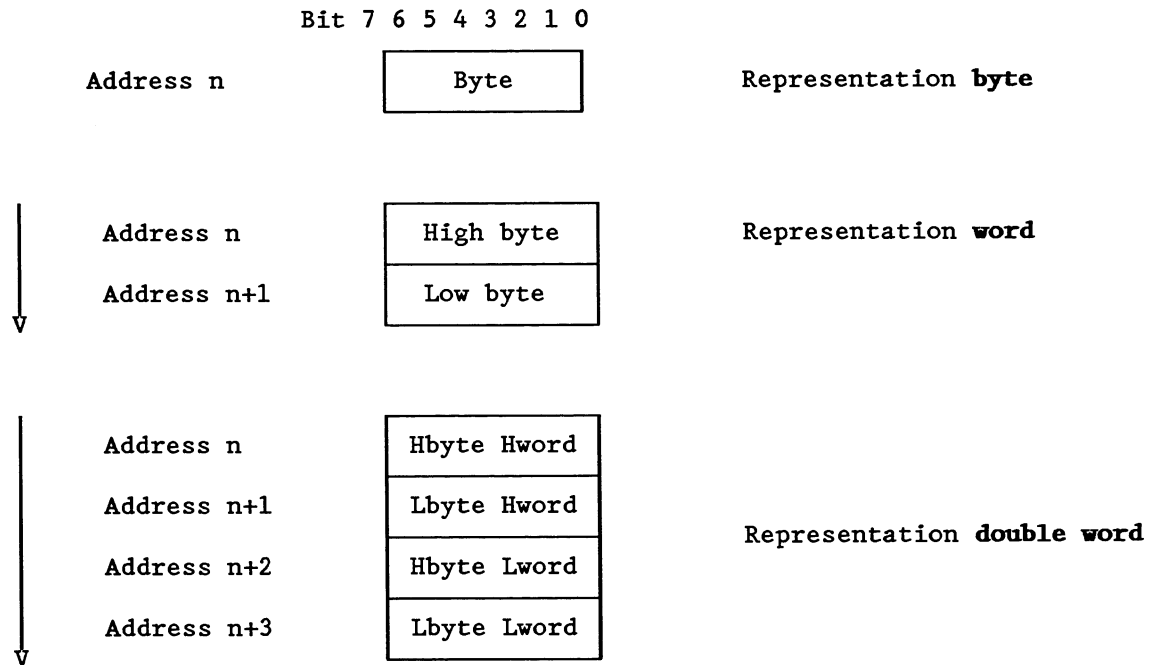
When programming the special function OB's 216, 217, and 218 you specify which of the 255 pages is to be used. The number of the "current" page will then be entered automatically in a location with address FEFF (see figure). All addresses then refer to the page whose number had been entered.

IMPORTANT!

The location with the address FEFF cannot be read.

Notes on assigning parameters

Writing (OB216) and reading (OB217) of a byte/word/double word is based on the following representation:



6.6.1 Writing Data to a Page (OB 216)

The special function organization block transfers a byte, word or double word from accumulator 1 (right-justified) to a certain page.

Addressing of the page and transfer of the complete data (1/2/4 bytes) form a program unit which cannot be interrupted.

Parameters:

1. Accumulator 2-L: destination address on page
possible values: 0 - 2047
2. Accumulator 3-LL: current page number
possible values: 0 - 255
3. Accumulator 3-LH: identification of data to be transferred
possible values: 0 = byte
1 = word
2 = double word

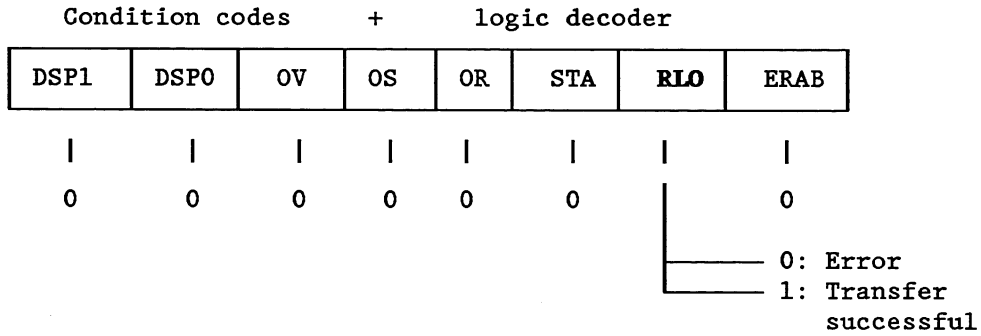
If writing to the page is successful,

- the contents of accumulator 1 and accumulator 3 are not altered,
- the accumulator 2-L will contain a value increased by 1/2/4 (depending on the length of the data transferred),
- the RLO is set (RLO = 1),
- the remaining bit and word condition codes are cleared (see result bits).

If writing on the page is not possible:

- the contents of all accumulators are not altered,
- the RLO is cleared,
- all remaining bit and word condition codes are also cleared.

Result bits:

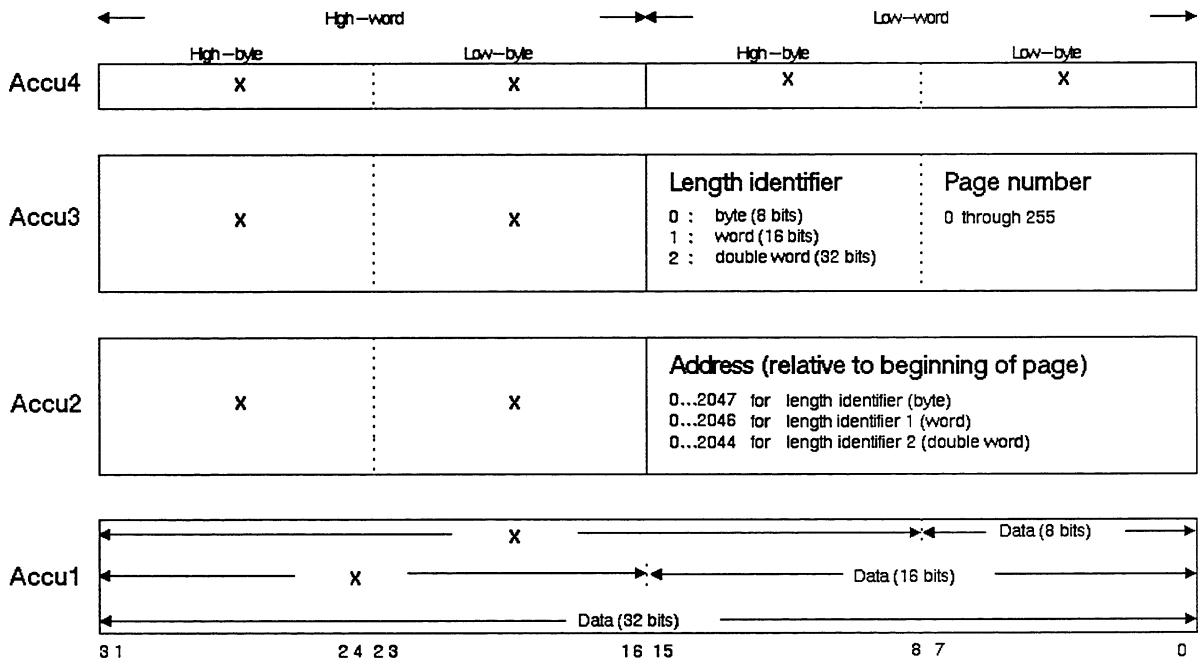


(see Chapter 3 for result condition codes)

Errors:

- incorrect length identifier in accumulator 3-LH
- destination address on page incorrect or non-existent
- page number stated does not exist
- page does not exist at all

Accumulator assignment b e f o r e writing



6.6.2 Reading Data from a Page (OB 217)

The special function organization block transfers a byte, word or double word from a certain page to accumulator 1 (right-justified).

Addressing of the page and transfer of the complete data (1/2/4 bytes) form an inseparable program unit which must not be interrupted.

Parameter:

1. Accumulator 2-L: source address on page
possible values: 0 - 2047
2. Accumulator 3-LL: current page number
possible values: 0 - 255
3. Accumulator 3-LH: identifier of data to be transferred
possible values: 0 = byte
1 = word
2 = double word

If reading from the page is successful,

- accumulator 1 (right-justified) will contain the value read (any of the 32 bits remaining are cleared)
- the contents of accumulator 3 are not altered,
- accumulator 2-L will contain a value increased by 1/2/4 (depending on the length of the data transferred),
- the RLO is set (RLO = 1),
- the remaining bit and word condition codes are cleared.

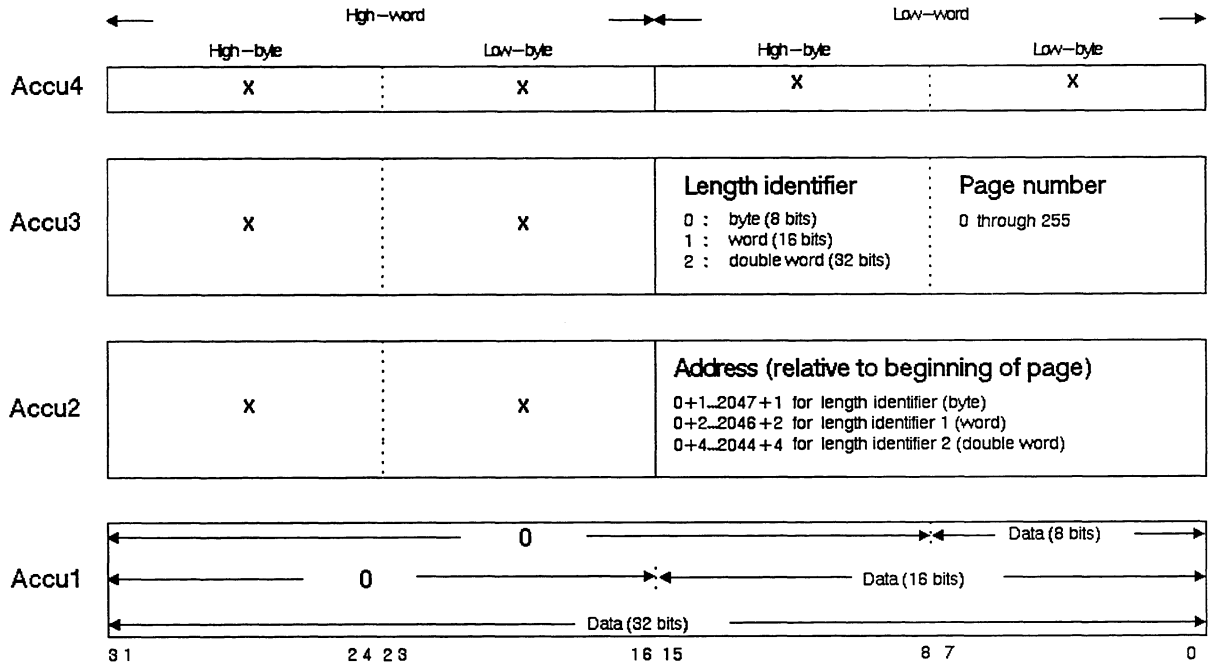
If reading from the page is not possible:

- the contents of all accumulators are not altered,
- the RLO is cleared (RLO = 0),
- all remaining bit and word condition codes are also cleared.

Errors:

- incorrect length identifier in accumulator 3-LH
- source address on page incorrect or non-existent
- page number stated does not exist
- page does not exist at all

Accumulator assignment a f t e r reading



6.6.3 Assigning a Page (OB 218)

The special function organization block transfers the slot identifier of 'its' processor to a certain page if the contents of the location addressed in this particular page are zero. As long as this slot identifier is entered in the location this particular page will be reserved for a particular processor and may **not** be used by other processors.

The organization block OB 218 is used to synchronize the data transfer and is extremely important if **larger data fields that belong together** are to be transmitted or transferred at once.

Addressing of the page, reading and, if applicable, writing of the slot identifier form a program unit which cannot be interrupted.

Parameter:

Accumulator 1-L: Destination address of location on page
possible values: 0 - 2047

Accumulator 1-LL: current page number
possible values: 0 - 255

(In this particular case the contents of accumulators 3 and 4 are not relevant.)

If assigning of the page is successful,
(contents of destination address = 0)

- the contents of the accumulators are not altered,
- the RLO is set (RLO = 1),
- the remaining bit and word condition codes are cleared.

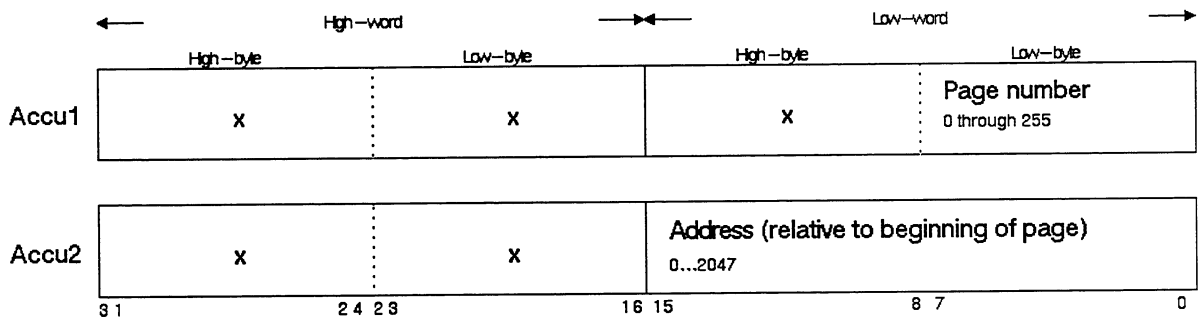
If assigning of the page is not possible:
(contents of the destination address # 0)

- the contents of the accumulators are not altered,
- the RLO is cleared (RLO = 0),
- all remaining bit and word condition codes are also cleared.

Errors:

- contents of destination address on page not zero
- destination address on page incorrect or non-existent
- page number stated does not exist
- page does not exist at all

Accumulator before / after reading



OB218.

Program example

The data words 4 through 11 of DB 45 of a CPU 928 are transferred to DX 45 (data words 0 through 7) of a second CPU 928 via COR C. Transmitter and receiver (multiprocessor operation) are synchronized by OB 218.

Current page on coordinator: no. 255
 Coordination location on page (assign): addr. 53
 Data transfer area on page (read and write): addr. 54-69

TRANSMITTER:

```

L   KB 255           page number
L   KB 53           address of coordination location
JU  OB 218         transfer slot identifier to location
                        on page
JC  =M001         if RLO = 1 (transfer successful) jump to mark
BEA                if not, block end

M001:C  DB 45       open source data block
L   KY 2,255       2=length id. double word, page no.
L   KB 54         start address on page
ENT                write to accumulator 3

L   DD 4           data words 4 and 5 (=4 bytes)
JU  OB 216         transfer 1st double word
                        increment address by 4 (Accu 2-L = 58)
TAK                save destination address

L   DD 6
JU  OB 216         transfer 2nd double word
TAK

L   DD 8
JU  OB 216         transfer 3rd double word
TAK

L   DD 10
JU  OB 216        transfer 4th double word
TAK

L   KY 0,255
L   KB 53           address with slot identifier
ENT
L   KB 0           accumulator 1 = 0
JU  OB 216        clear slot id. --> release data transfer area
BE
    
```

RECEIVER:

L	KB 255	page number
L	KB 53	coordination location
JU	OB 218	page reserved by 2nd CPU
JC	=M002	if RLO = 1, jump to mark
BEU		
M002: CX	DX 50	destination data block
L	KY 2,255	
L	KB 54	
ENT		write to accumulator 3
L	KB 0	write to accumulator 2
JU	OB 217	read 1st double word
		increment address by 4 (Accu 2-L = 58)
T	DD 0	transfer accu 1 to data word 0 and 1
JU	OB 217	read 2nd double word
T	DD 2	
JU	OB 217	read 3rd double word
T	DD 4	
JU	OB 217	read 4th double word
T	DD 6	
L	KY 0,255	
L	KB 53	address with slot identifier
ENT		
L	KB 0	accumulator 1 = 0
JU	OB 216	clear slot id. --> release data transfer area
BE		

6.7 Sign Extension (OB 220)

This special function extends the sign of a 16-bit fixed point number in accumulator 1 to the higher order word (accumulator 1-H):

The higher order word is loaded with KH = 0000 if bit $2^{15} = 0$ (positive number).

The higher order word is loaded with KH = FFFF if bit $2^{15} = 1$ (negative number).

This sign extension is necessary in order to be able to extend a negative 16-bit fixed point number to a 32-bit fixed point number before a fixed point-floating point conversion (32-bit, FDG command) is carried out.

Execution time of special function OB 220: 16 μ s

Parameters:

1. Accumulator 1-L: 16-bit fixed point number

Errors: none

6.8 System Functions

6.8.1 Switch on/off "Disable all interrupts" (OB 120) and switch on/off "Delay all interrupts" (OB 122)

A STEP5 program can be interrupted at the block or command boundaries by programs with a higher priority. Process and time interrupts are part of these high-priority program levels. The execution time of the program nested must be added to the execution time of the program interrupted.

Using special-function organization blocks OB 120 and OB 122 you can prevent the process and/or time interrupts from being nested at one or more subsequent block or command boundaries (according to the setting in DX 0).

OB 120: switch on/off "Disable all interrupts"

The special-function organization block OB 120 influences the *reception* of interrupts:

Switch on "Disable interrupts" means that from that point in time onwards, no more interrupts are recorded and those interrupts that have already been recorded (which are waiting e.g. for a block boundary) are erased. Only if OB 2 (process interrupt) or a time interrupt OB has already been started, will processing of this OB be brought to an end.

Switch off "Disable interrupts" means that all interrupts occurring are immediately recorded, nested at the next block or command boundary and processed again.

OB 122: switch on/off "Delay all interrupts"

This special-function organization block influences the processing of interrupts.

Switch on "Delay interrupts" means that all occurring interrupts are recorded and interrupts already present remain recorded. Processing of the recorded interrupts, however, is not started.¹⁾ The block and command boundaries are deactivated for interrupt processing. Only if OB 2 (process interrupt) or a time interrupt OB has already been started, will processing of this OB be brought to an end.

Switch off "Delay interrupts" means that all interrupts recorded are nested at the next block or command boundary and processed.

1) A collision of two time interrupts occurs if a specific time interrupt is called a second time within the "Delay interrupts" stage .

In a **control word** with the following assignment, bits are set by OB 120 and OB 122 to indicate interrupts to be disabled or delayed:

Bit 2^0 : time interrupts

Bit 2^2 : process interrupts

Bits 2^1 , 2^3 through 2^{31} : reserved; these bits must equal "0"!

Meaning:

As long as bit 2^0 equals 1, all the occurring time interrupts are disabled or delayed.

As long as bit 2^2 equals 1, all the occurring process interrupts are disabled or delayed.

If bit 2^0 as well as bit 2^2 are set to "1", no time interrupt nor process interrupt are recorded.

Parameters:

1. Accu 2-L: function identifier
possible values: 1, 2 or 3
2. Accu 1: new control word or mask

Accu 2-L	Accu 1		Function:
	before	after	
1	control word	control word	The contents of accu 1 are loaded into the control word
2	mask	new control word	Every bit marked in the mask in accu 1 with "1" is set to "1" in the control word. The new control word is loaded into accu 1.
3	mask	new control word	Every bit marked in the mask in accu 1 with "1" is set to "0" in the control word. The new control word is loaded into accu 1.

Errors:

- illegal function identifier in accu 2-L
- one of the reserved bits in accu 1 (2^1 , 2^3 through 2^{31}) equals "1"

In the case of an error, **OB 31** (other execution time errors) is called and an error identifier transferred to accu 1:

1A47H for OB 120
1A48H for OB 122

Remarks:

- The state of the control word can be scanned using the following program sequence:
 1. Load function identifier 2 or 3 into accu 2-L
 2. Load value "0" into accu 1
 3. Call up special-function OB 120/122
 4. Read accu 1
- The state of the interrupt processing can also be determined by reading the system data words RS 131 and RS 132.

RS 131 Condition code word "Disable all interrupts" (OB 120)
 RS 132 Condition code word "Delay all interrupts" (OB 122)

- To disable and enable the process interrupts you may use the commands IA and RA in lieu of OB 120:

IA	corresponds to	:L	KB2
		:L	KM00000000 00000100
		:JU	OB120
RA	corresponds to	:L	KB3
		:L	KM00000000 00000100
		:JU	OB120

6.8.2 Switch on/off "Disable individual time interrupts" (OB 121) and switch on/off "Delay individual time interrupts" (OB 123)

Using the special-function organization blocks OB 121 and OB 123, you can prevent *specific* time interrupt OBs from being nested at one or more subsequent block or command boundaries. You may specify, for example, that a specific program section cannot be interrupted by an OB 10 (10 ms) or OB 11 (20 ms). All other programmed time interrupts, however, are processed as usual.

OB 121: switch on/off "Disable individual time interrupts"

The special-function organization block OB 121 influences the *reception* of time interrupts:

Switch on "Disable individual time interrupts" means that from that point in time onwards no more time interrupts are recorded and those interrupts that have already been recorded (which are waiting e.g. for a block boundary) are erased. Only if a time interrupt OB has already been started, is processing of this OB brought to an end.

Switch off "Disable individual time interrupts" means that all time interrupts occurring are immediately recorded, nested at the next block or command boundary and processed again (according to setting in DX 0).

OB 123: switch on/off "Delay individual time interrupts"

This special-function organization block influences the processing of specific time interrupts.

Switch on "Delay individual time interrupts" means that all occurring interrupts are recorded and time interrupts being already present remain recorded. Processing of the time interrupts being marked in the control word, however, is not started.¹⁾

The block and command boundaries are deactivated for the processing of these interrupts. Only if one of these time interrupt OB has already been started, is processing of this OB brought to an end.

Switch off "Delay individual time interrupts" means that all interrupts recorded are nested at the next block or command boundary (according to setting in DX 0) and processed.

In a **control word** with the following assignment, bits are set by OB 121 and OB 123 to indicate interrupts to be disabled or delayed:

Bit 2⁰: must be "0"
Bit 2¹: must be "0"
Bit 2²: must be "0"
Bit 2³: time interrupt 10 ms (OB 10)
Bit 2⁴: time interrupt 20 ms (OB 11)
Bit 2⁵: time interrupt 50 ms (OB 12)
Bit 2⁶: time interrupt 100 ms (OB 13)
Bit 2⁷: time interrupt 200 ms (OB 14)
Bit 2⁸: time interrupt 500 ms (OB 15)
Bit 2⁹: time interrupt 1 sec (OB 16)
Bit 2¹⁰: time interrupt 2 sec (OB 17)
Bit 2¹¹: time interrupt 3 sec (OB 18)
Bit 2¹²: must be "0"
Bit 2¹³: must be "0"
Bit 2¹⁴: must be "0"
Bit 2¹⁵: must be "0"

As long as a bit is set to "1", will the corresponding time interrupt remain disabled or delayed.

Parameters:

1. Accu 2-L: function identifier
possible values: 1, 2 or 3
2. Accu 1: new control word or mask

Accu 2-L	Accu 1		Function:
	before	after	
1	control word	control word	The contents of accu 1 are loaded into the control word
2	mask	new control word	Every bit marked in the mask in accu 1 with "1" is set to "1" in the control word. The new control word is loaded into accu 1.
3	mask	new control word	Every bit marked in the mask in accu 1 with "1" is set to "0" in the control word. The new control word is loaded into accu 1.

Errors:

- illegal function identifier in accu 2-L
- one of the reserved bits in accu 1 equals "1"

In the case of an error, **OB 31** (other execution time errors) is called and an error identifier is transferred to accu 1:

1A4AH for OB 121
1A4BH for OB 123

Remarks:

- The state of the control word can be scanned using the following program sequence:
 1. Load function identifier 2 or 3 into accu 2-L
 2. Load value "0" into accu 1
 3. Call up special-function OB 121/123
 4. Read accu 1
- The state of the interrupt processing can also be determined by reading the system data words RS 135 and RS 137.

RS 135 Condition code word
"Disable individual time interrupts" (OB 121)

RS 137 Condition code word
"Delay individual time interrupts" (OB 123)

6.8.3 Set Cycle Time (OB 221)

By calling this special function you can alter the monitoring of the max. cycle time, (standard preset value 150 ms), to a new value. The timer for the monitoring is also restarted when the call is executed: The current cycle, i.e. the cycle in which OB 221 was called for the first time, is extended by the new value, beginning from the time when the special function is called. The cycle time of all subsequent cycles corresponds to the new value (= the time value you transfer to accumulator 1).

Parameters:

1. Accumulator 1: new cycle time (in milliseconds)
possible values: 1 ms - 6000 ms

Errors:

- cycle time specified not in the range between 1 ms - 6000 ms.

Note:

The special function OB 221 was adopted from the S processor for reasons of compatibility and should not be programmed in the CPU 928. Instead, it is advisable to program this system function in DX 0 (see Chapter 7).

6.8.4 Restart Cycle Time (OB 222)

The special function OB 222 is responsible for retriggering the cycle time monitoring, i.e. the timer for this monitoring is restarted. Calling this special function means that the max. permitted cycle time is extended by the value set (150 ms as a standard value or specified in DX 0) beginning from the time of the call was made.

Parameters: none

Errors: none

6.8.5 Compare Start-up Modes (OB 223)

If OB 223 is called - e.g. during the start-up or at the beginning of the cyclic program - a check is carried out in multiprocessor operation to establish whether or not the start-up modes of all processors involved are identical.

If this is not the case the processor in which OB 223 was called will detect an execution time error. OB 31 is then called. Accu 1 contains the error identification 1A3B (hex.). If OB 31 has not been programmed, the processor goes into the STOP state signalling an LZFL error. Its STOP LED flashes slowly whereas the remaining processors go into STOP with their LED's permanently lit.

Parameters: none

Errors: none

6.8.6 Transfer Interprocessor Communication (IPC) Flags as a Block (OB 224)

The IPC flags specified in DB 1 are transferred, in multiprocessor operation, if the processor receives the signal allowing access to the I/O bus.

If several processors attempt to access the bus simultaneously the coordinator will output the bus enable signal to one processor after the other. In this case the processor may only transfer one byte each time it is given access. This interleaved-transmission may cause IPC flag information which belongs together to be separated and in incorrect values being used.

By calling the organization block OB 224 you can transfer all IPC flags specified in DB 1 of the processor as a block: As long as a processor is transferring IPC flags an interruption by another processor will not be possible. Since the next processor will have to wait until transmission is possible, cyclic programm processing will be delayed by this time (cycle time!).

OB 224 ensures that the complete interprocessor communication flag information remains together. It must be called during the start-up program.

- a) for all processors involved in IPC flag transfer and
- b) for all types of start-up mode used.

Parameters: none

Errors: none

Note:

The special function OB 224 was adopted from the S processor for reasons of compatibility and should not be programmed in the CPU 928. Instead, it is advisable to program this system function in DX 0 (see Chapter 7).

6.8.7 Read Word from the System Program (OB 226)

The system program of the processor has a length of 64×2^{10} words and is located in a memory area which you cannot access with STEP5 commands. However, access is possible using OB 226.

Parameters:

1. Accumulator 1-L: address of system program storage location to be read

Errors: none

After calling OB 226

- the word read is in accumulator 1, right-justified,
- the remaining contents of accumulator 1 are erased,
- the previous contents of accumulator 1 (i.e. the word address) are now in accumulator 2,
- the previous contents of accumulator 2 are lost.

For more information about OB 226, please refer to the description of OB 227 and the program example.

6.8.8 Read Check Sum of System Program (OB 227)

The special function organization block OB 227 loads the cross-check sum from the system program memory area in accumulator 1.

Parameter: none

Errors: none

After calling OB 227

- the check sum read (1 word) is in accumulator 1, right-justified,
- the remaining contents of accumulator 1 are erased,
- the previous contents of accumulator 1 are in accumulator 2,
- the previous contents of accumulator 2 are lost.

Application:

You can check the contents of the system program during cyclic program processing by

- * reading the individual storage locations of the system program using OB 226,
- * adding all storage locations with fixed point addition (command +F),
- * reading the check sum using OB 227 and then
- * comparing the check sum read with the sum obtained by fixed point addition.

Program example**FB111**

NAME: CHECKSUM

```

      :L  KH0000
      :T  FW254      clear check sum flag
      :T  FW252      clear address counter

M001 :L  FW252      load address of storage location to be read
      :JU  OB226      read word
      :L  FW254      load check sum flag
      :+F              add
      :T  FW254      store check sum flag
      :
      :L  FW252      increment address counter
      :L  KF+1
      :+F
      :T  FW252
      :
      :L  KH0000      if address counter is not '0'
      :><  F
      :JC  =M001      jump to mark M001
      :
      :JU  OB227      if address counter is '0', read check sum
      :L  FW254      load check sum flag
      :!=F          if identical, block end
      :BEC
      :
      :STP          if not identical, stop command
      :BE

```

6.8.9 Read Status Information of Program Level (OB 228)

The system program will call the appropriate program level if a particular event occurs. This means that the program level is then 'activated'.

The organization block OB 228 helps you to specify, whether a certain program level is activated at a certain time or not. You transfer the number of the program level, the status of which is to be scanned in accumulator 1. (The numbers correspond to these numbers entered in the ISTACK under the heading 'LEVEL'.) When the special function is called, OB 228 will transfer the status information of the level specified to accumulator 1.

Parameters:

1. Accumulator 1-L: number of program level
 (Cf. ISTACK, LEVEL)

possible values (hexadecimal):

02 = cold restart
04 = cycle
06 = time interrupt 5 sec
08 = time interrupt 2 sec
0A = time interrupt 1 sec
0C = time interrupt 500 ms
0E = time interrupt 200 ms
10 = time interrupt 100 ms
12 = time interrupt 50 ms
14 = time interrupt 20 ms
16 = time interrupt 10 ms
18 = not used
1A = not used
1C = controller
1E = not used
20 = not used
22 = not used
24 = process interrupt
26 = not used
28 = not used
2A = not used
2C = abort
2E = not used
30 = collision of two time interrupts
32 = controller error
34 = cycle error
36 = not used
38 = command code error
3A = execution time error
3C = addressing error
3E = acknowledgement delay
40 = not used
42 = not used
44 = manual warm restart
46 = automatic warm restart

Errors: none

After calling OB 228

- the status information is in accumulator 1:
 - contents of accumulator 1 = 0: program level has not been called
 - contents of accumulator 1 # 0 program level has been activated
- the previous contents of accumulator 1 are in accumulator 2,
- the previous contents of accumulator 2 are lost.

This means that your program can be executed dependent on the status of another program level.

Example:

You want an acknowledgement delay ignored during the cold restart, however, not in the other program levels.

At the beginning of OB 23 you call the special function organization block OB 228 to determine whether the program level COLD RESTART (number 02) was activated when acknowledgement delay occurred or not. Further error handling will now be dependent on the status information obtained:

Accu 1 = 0:	COLD RESTART passive	--->	acknowledgement delay did not occur during the cold restart
		--->	error program will have to be run
Accu 1 # 0:	COLD RESTART active	--->	acknowledgement delay occurred during the cold restart
		--->	acknowledgement delay can be ignored

OB 228 allows errors to be handled differently, depending on the situation.

6.9 Functions for Standard Function Blocks (OB 230 through OB 237)

The special function organization blocks OB 230 through OB 237 are reserved for handling functions and can only be called in the standard function blocks FB 120 through FB 127.

These standard function blocks (known as "handling blocks") control data transfer across the page area in multiprocessor operation: They are used if data or parameters as well as control information are to be transmitted from or to the communications processors.

Overview

Standard function block	Special-function organization block	Handling block
FB 120	SF-OB 230	SEND
FB 121	SF-OB 231	RECEIVE
FB 122	SF-OB 232	FETCH
FB 123	SF-OB 233	CONTROL
FB 124	SF-OB 234	RESET
FB 125	SF-OB 235	SYNCHRON
FB 126	SF-OB 236	SEND ALL
FB 127	SF-OB 237	RECEIVE ALL

The handling blocks are available as a software product on floppy disk and their use is described in detail in "Programmable controller S5-135U handling blocks for R processor and CPU 928" (Order number: C79000-B8576-C366-xx).

6.10 Shift Register

A software shift register consists of rows of 8-bit wide storage locations. The length of this register is between 1 and max. 256 storage locations.

The data of a shift register is in the data block RAM of the processor. Each shift register is invariably assigned to a specific data block: both of them have the same number (permissible: 192 through 255). When you have set up a shift register with for instance number 210, the appertaining data is contained in data block DB 210.

The DB-RAM contains approx. 23K words (address KH 8000 to KH DD7F). The data blocks copied by means of OB 254 and 255 (from KH 8000 onwards, ascending) and the shift registers created by the user (from KH DD7F onwards, descending) are located in this area. If the memory area of the DB-RAM is insufficient when copying DBs or creating shift registers, the processor identifies an execution time error and will call OB 31. Further reaction depends on how OB 31 is programmed (see 'Other execution time errors').

You can write data into the shift register or read data from it. This is done by means of the "pointers": pointers are flag bytes which contain the contents of individual locations of a shift register.

The following figures show the principle of the software shift register.

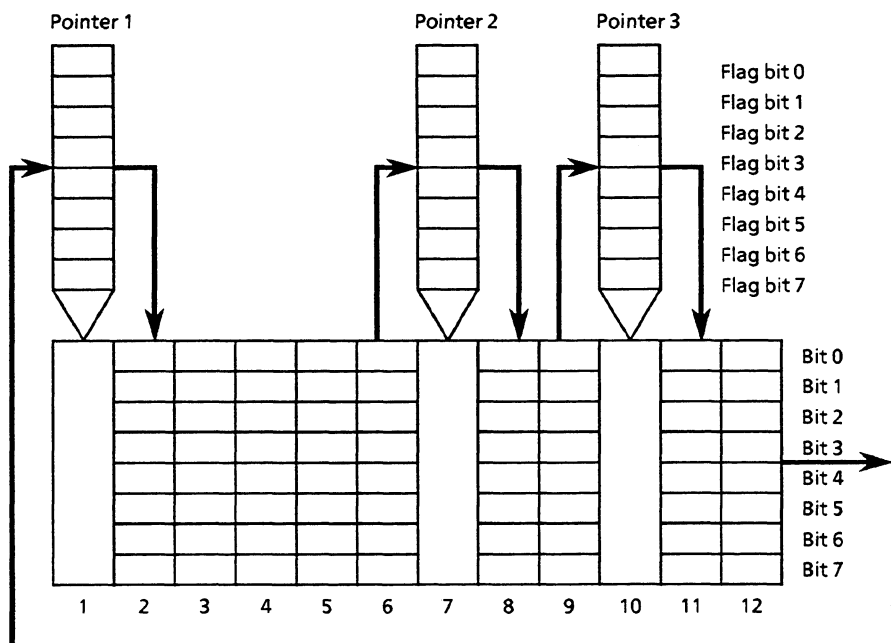


Fig. 6-1: Schematic diagram of the shift register with 3 pointers and 12 storage locations

The first pointer (= base pointer) is permanently set on the first storage location of the shift register. The number of this flag byte is specified by the user. All other pointers are then positioned relative to the base pointer. Between 1 and 6 pointers max. are possible per shift register.

When a shift register is processed the information is transferred from one storage location to the next byte per byte - just as in a hardware shift register (see fig.). This means that each time the shift register function is called information is shifted by exactly 1 storage location ($\hat{=}$ 1 clock pulse), and the pointers are supplied with new contents. As shown by the arrows the information is 'shifted through' the complete shift register to the last storage location. From there the information will again be transferred to storage location 1, (after 12 cycles for the shift register illustrated below).

Example:

The following illustrations show how information is shifted within a shift register.

The flag bits are set in the pointers before the special functions are called:

```

set flag bit 0 of pointer 1      :S F0.0
set flag bit 3 of pointer 2     :S F1.3
set flag bit 2 of pointer 3     :S F2.2
    
```

Then the shift register function is called :JU OB 241

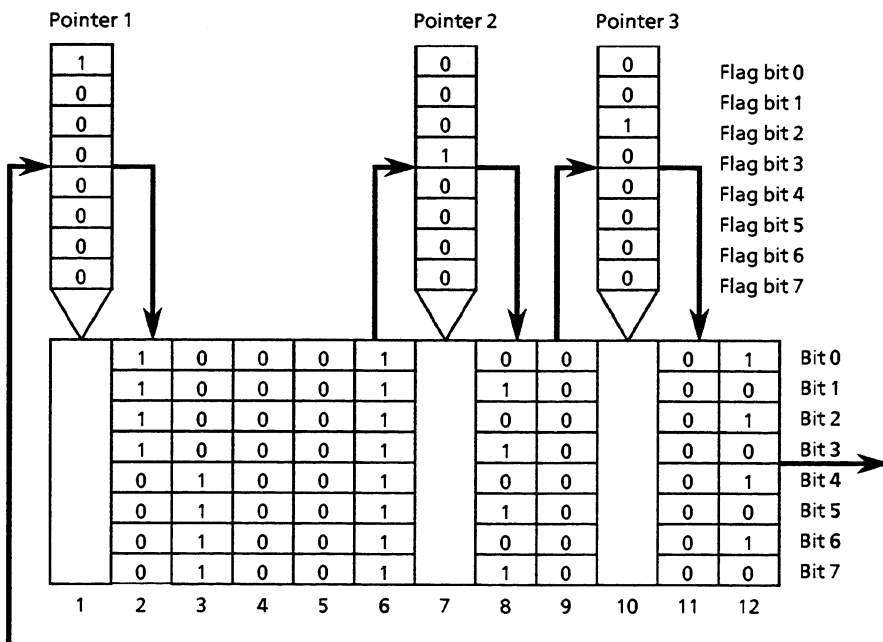


Fig. 6-2: Schematic diagram of the shift register with 3 pointers and 12 storage locations before the first clock pulse

After the special functions have been called the information in the storage locations 8-bit wide is shifted by one location:

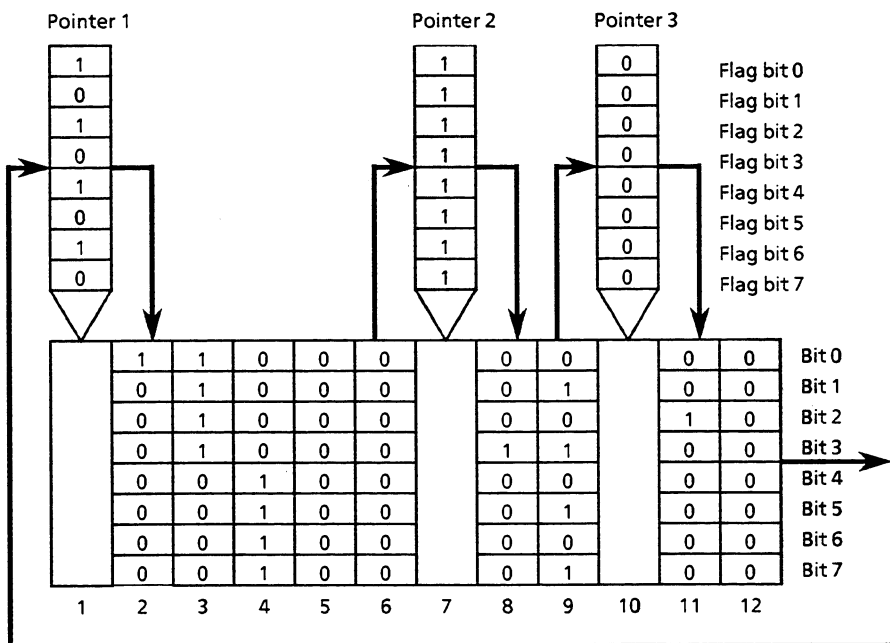


Fig. 6-3: Schematic diagram of the shift register with 3 pointers and 12 storage locations after the first clock pulse

Evaluation of the information now in the pointers is possible by means of :L FY 0 etc.

Flag bits 0, 3, and 2 may be scanned at the base pointer: This allows the complete information in the entries in all pointers to be evaluated at the base pointer (for our example this required 12 clock pulses).

If you want to use a shift register there are 3 special function organization blocks available:

OB 240: This function initializes a shift register.

OB 241: This function processes a shift register.

OB 242: This function erases a shift register.

6.10.1 Initialize Shift Register (OB 240)

Before processing a shift register it must be initialized first. This is done by calling OB 240 once (ideally in a start-up organization block).

The parameters which OB 240 requires in order to create a shift register are contained in a data block with the number of the shift register to be initialized. DB numbers are possible between 192 and 255.

The data block has a fixed structure which must not be changed. It can have a max. length of 9 data words (DW 0 through DW 8).

The individual data words must be assigned as follows:

0	DW 0
Shift register length (bytes)	DW 1
Number of 1st flag byte	DW 2
Spacing n_2	DW 3
Spacing n_3	DW 4
Spacing n_4	DW 5
Spacing n_5	DW 6
Spacing n_6	DW 7
0	DW 8

Fig. 6-4: Structure of the data block for the initialization of a shift register

Data word 0: Contents must always be 0.

Data word 1: The shift register length is the number (in bytes) of storage locations of the shift register. It can be in the range between $2 \leq L \leq 256$.

Data word 2: The number of the first flag byte determines the base pointer and thus the flag block assigned to the pointers. If, e.g., you assign two pointers this means that together with the base pointer you have three pointers. Then the flag byte specified in the data block as well as the two following flag bytes will be reserved.
Make sure that you still have a sufficient number of flags available for all pointers until the end of the flag block.

Data word 3
through 7 (max.):

Spacing (in bytes) of pointers to base pointer:

n_2 = spacing of pointer 2 to base pointer
 n_3 = spacing of pointer 3 to base pointer
 n_4 = spacing of pointer 4 to base pointer
etc.
(5 entries max.)

Data word
after last
pointer spacing:

(DW 8 for our example) Contents must always be 0.

If only two other pointers are required in addition to the base pointer the '0' will be contained in data word DW 5 etc..

All information will be available as fixed point numbers.

IMPORTANT!

- * The number of pointers (incl. the base pointer) must not exceed the length of the shift register!
- * The spacing of a pointer to the base pointer must not exceed the length of the shift register.
- * The contents of data word DW 0 and the data word after the last pointer spacing must always be '0'.
- * The data block must be programmed and called before OB 240 is called!

By calling OB 240 a certain memory area at the end of the data block RAM will be reserved and initialized using the information from this particular data block.

Memory requirements:

$n = \text{shift register length} / 2 + 8 \text{ data words}$

are required for every shift register, i.e. the length of the DB-RAM is reduced by n data words. The data block RAM end address is shifted to the lower addresses.

If a shift register which is to be initialized exists already, the area already assigned will be initialized again if the new and the already existing shift registers have identical lengths. If not, the old area is declared invalid and a new area will be opened.

Parameters:

1. data block called
possible values: DB no. 192 - 255

Errors:

- illegal data block number (<192, >255)
- existing memory location in the DB-RAM insufficient
- formal error in the structure of the data block
- illegal length stated for the shift register
- parameter assignment error at pointers

If an error occurs the processor identifies an execution time error and will call OB 31. Further reaction depends on the programming of OB 31 (see 'Other execution time errors').

If OB 31 has not been programmed the processor will stop. Error identifiers are written in accumulator 1 and supply a more detailed explanation of the errors.

6.10.2 Process Shift Register (OB 241)

The special function organization block OB 241 processes a shift register providing it has been initialized first by OB 240.

Max. 64 shift registers can be called in the CPU 928.

Parameters:

1. Accumulator 1-L: number of shift register to be processed
possible values: 192 - 255

Before OB 241 is called, certain flag bits are usually set/reset in the pointers.

The information is shifted byte by byte from one storage location to the next higher storage location every time OB 241 is called, and the pointers are supplied with new contents. Calling OB 241 repeatedly allows information to be 'shifted through' the complete shift register to the last storage location. From here it will again be transferred to storage location 1.

After OB 241 has been called the pointers (6 max. per shift register, positioned at any point required, except for the base pointer) will be supplied with the information of the neighbouring storage location. Evaluation of this information is now possible.

Errors:

- illegal shift register number in accumulator 1
- shift register has not been initialized

If an error occurs the processor identifies an execution time error and will call OB 31. Further reaction depends on how OB 31 is programmed (see 'Other execution time errors'). If OB 31 has not been programmed the processor will stop. Error identifiers are written in accumulator 1 and supply a more detailed explanation of the errors.

6.10.3 Erase Shift Register (OB 242)

This special function is used to 'erase' a shift register in the data block RAM: the entry in address list DB 0 is erased and the corresponding shift register in the DB-RAM is declared invalid. (Note: The shift registers that have been erased still take up memory space!)

Parameters:

1. accumulator 1-L: number of shift register to be erased
possible values: 192 - 255

After calling OB 242 the shift register will be erased and can no longer be used; if it is to be used again it will have to be re-initialized.

Errors:

- illegal shift register number in accumulator 1
- shift register has not been initialized

If an error occurs the processor identifies an execution time error and will call OB 31. Further reaction depends on how OB 31 is programmed (see 'Other execution time errors'). If OB 31 has not been programmed the processor will stop. Error identifiers are written in accumulator 1 and supply a more detailed explanation of the errors.

6.11 Control: PID Algorithm

This chapter is only for those users who wish to work with PID controllers!
 If you do not, the information contained in this chapter will not be required!

You can call one or more PID controllers in the CPU 928 of the S5 135U.

Each of the controllers must be initialized in the start-up organization block. A data block is used for the transfer of parameters.

The actual control algorithm is integrated in the system program and the user may only call it as an organization block. The data block again serves as the data interface between the control algorithm and the user program.

Functional description of the PID controller

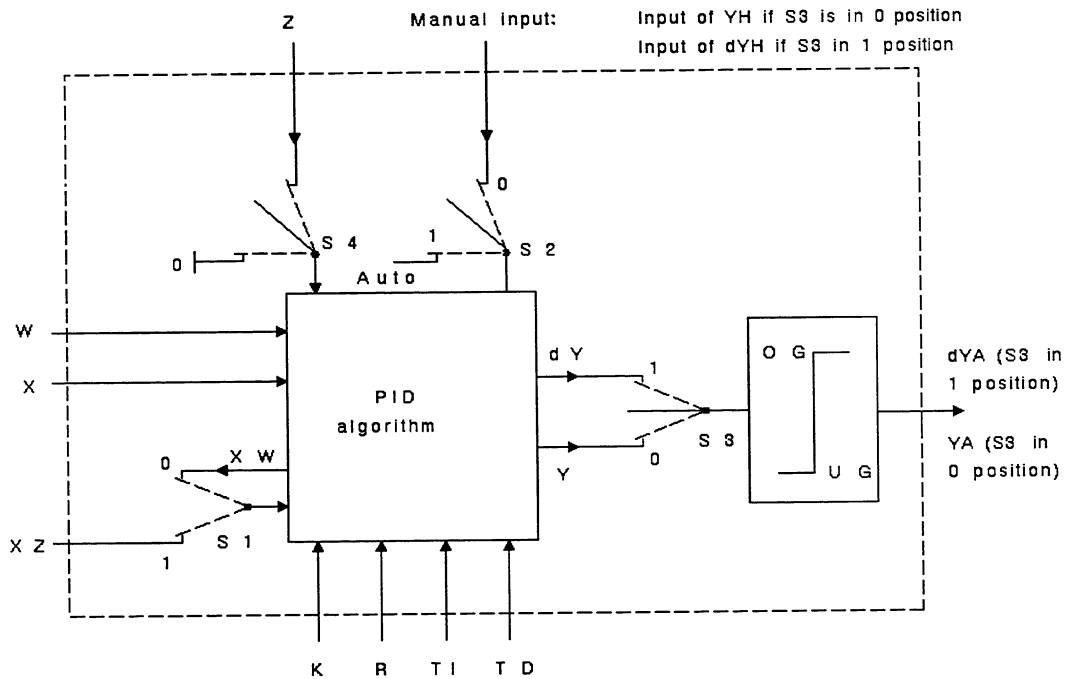


Fig. 6-5: Block diagram of the PID controller

Index k: k x sampling

Switch	Position	Effect
S1 STEU bit 1	1	The derivative unit is supplied with the error signal XW_k . The derivative unit can be supplied with another signal via XZ.
S2 STEU bit 0	0 1	Manual operation Automatic
S3 STEU bit 3	0 1	Position algorithm Velocity algorithm
S4 STEU bit 4	0 1	With feedforward control Without feedforward control

A function which corresponds to the switch positions of this block diagram is obtained by assigning parameters to the PID controller, i.e. setting the control bits in the control word, STEU.

The continuous controller is intended for fast controlled systems e.g. in process engineering for pressure, temperature or flow rate control.

The controller itself is based on a PID algorithm. The output signal can either be output as a manipulated variable (position algorithm) or as a change of the manipulated variable (velocity algorithm).

The individual P, I, and D actions may be disabled via their respective parameters R, TI, and TD if the corresponding locations are preset with zero. This allows simple implementation of all desired controller structures, e.g. PI, PID or PD controllers.

The derivative unit can be supplied either with the error signal XW or (via the XZ input) any disturbance or the inverted actual value -x.

If a precontrol of the actuator without dynamic behavior is required to compensate for the influence of a disturbance then a disturbance Z, which can be measured in the process, can be fed forward to the control algorithm. In manual operation this is replaced by the pre-selected manipulated variable YN.

If an inverted control direction is required a negative K value must be preset.

If the control information (dY or Y) reaches a limit the I action is automatically disabled in order to prevent a deterioration of the controller response.

The controller program may be supplied with preset fixed values or adaptive (dynamic) parameters (K, R, TI, TD). They are input via the storage locations assigned to the individual parameters.

The PID controller is based on a velocity algorithm according to which the respective positioning increment dY_k is calculated at a particular point in time $t = k*TA$ according to the following formula:

$$dY_k = K[(XW_k - XW_{k-1})R + \frac{TA}{2TN}(XW_k + XW_{k-1}) + \frac{1}{2} \frac{TV}{TA}(-(-XU_k - 2XU_{k-1} + XU_{k-2}) + dD_{k-1})]$$

$$= K(dPW_k R + dI_k + dD_k)$$

P action I action D action

$dXXX_k$: Change of variable XXX at time t.

U may either be W or Z, depending on, whether XW or XZ is supplied to the derivative unit. The following applies:

If XW_k is supplied:

If XZ is supplied:

$$PW_k = W_k - X_k$$

$$PW_k = XW_k - XW_{k-1}$$

$$QW_k = PW_k - PW_{k-1}$$

$$QW_k = XW_k - 2XW_{k-1} + XW_{k-2}$$

$$PZ_k = XZ_k - XZ_{k-1}$$

$$QZ_k = PZ_k - PZ_{k-1}$$

$$QZ_k = XZ_k - 2XZ_{k-1} + XZ_{k-2}$$

$$dPW_k = (XW_k - XW_{k-1})R$$

$$dI_k = TI * XW_k \quad TI = \frac{TA}{TN}$$

$$dD_k = \frac{1}{2} (TD * QU_k + dD_{k-1}) \quad TD = \frac{TV}{TA}$$

If the manipulated variable Y_k is required as the controller output at time t_k it is calculated according to the following formula:

$$Y_k = \sum_{m=0}^{m=k} dY_m$$

With most controller designs it is assumed that $R = 1$ if a P action is desired.

The variable R allows the adjustment of the proportional action of the PID controller.

Data blocks for the PID controller

Controller specific data are input using a transfer data block (see section 4.10.1 and 4.10.2 for initialization and processing of the PID controller).

These data must be preset in the transfer data block x:

K, R, TI, TD, W, STEU, YH, BGOG, BGUG

The structure of the transfer data block is described in detail in the remainder of this section. This data block must consist of 49 data words with numbers from 0 through 48.

Structure of the transfer data block

Addr. in DB	Name	I/O	Numerical format	PG format	Remarks
DW 0	-	-	-	-	reserve
DD 1	K	I	FLP	KG	proportional coefficient K > 0: positive control direction, i.e. change of actual value and manip. variable in same direction K < 0: negative control direction, floating point number range
DD3	R	I	FLP	KG	R parameter, usually = 1 for controllers with P action; floating point number range
DD 5	TI	I	FLP	KG	TI = TA/TN; floating point number range
DD 7	TD	I	FLP	KG	TD = TV/TA; floating point number range
DD 9	W _k	I	FLP	KG	setpoint input here, if STEU bit 6 = 1, if not in word no. 19 (-1 ≤ W _k < 1)
DW 11	STEU	I	BP	KM	control word
DD 12	YH _k	I	FLP	KG	manual value input here, if STEU bit 6 = 1, if not in word no. 18 (-1 ≤ YH _k < 1) value of manip. variable increments must be input here for velocity algorithms
DD 14	BGOG	I	FLP	KG	upper limit value -1 ≤ BGOG ≤ 1 (YA _{kmax}); !! BGUG < BGOG !!
DD 16	BGUG	I	FLP	KG	lower limit value -1 ≤ BGUG ≤ 1 (YA _{kmin});
DW 18	YH _k	I	NF	KF	manual value input here, if STEU bit 6 = 0, (-1 ≤ YH < 1) value of manip. variable increments must be input here for velocity algorithms
DW 19	W _k	I	NF	KF	setpoint input here, if STEU bit 6 = 0 (-1 ≤ W _k < 1)

Addr. in DB	Name	I/O	Numerical format	PG format	Remarks
DW 20	MERK		BP	KM	bit 0 = 1: positive limit exceeded; bit 1 = 1: negative limit undershot
DW 21	X_k	I	NF	KF	actual value input for STEU bit 7 = 0 ($-1 \leq X_k < 1$)
DD 22	X_k	I	FLP	KG	actual value input for STEU bit 7 = 1 ($-1 \leq X_k < 1$)
DW 24	Z_k	I	NF	KF	disturbance ($-1 \leq Z_k < 1$)
DD 25	Z_k	I	FLP	KG	disturbance input here if STEU bit 7 = 1 ($-1 \leq Z_k < 1$)
DD 27	Z_{k-1}		FLP	KG	historical value of disturbance
DW 29	XZ_k	I	NF	KF	value supplied to the derivative unit via input XZ ($-1 \leq XZ_k < 1$); input here if STEU bit 7=0
DD 30	XZ_k	I	FLP	KG	XZ input here, if STEU bit 7 = 1 ($-1 \leq XZ_k < 1$)
DD 32	XZ_{k-1}		FLP	KG	historical value of XZ_k
DD 34	PZ_{k-1}		FLP	KG	$XZ_{k-1} - XZ_{k-2}$
DD 36	dD_{k-1}		FLP	KG	derivative action
DD 38	XW_{k-1}		FLP	KG	historical value of error signal
DD 40	PW_{k-1}		FLP	KG	$XW_{k-1} - XW_{k-2}$
DW 42	-	-	-	-	reserve
DD 44	Y_{k-1}		FLP	KG	historical value of manipulated variable calculated Y_{k-1} or dY_{k-1} before the limiter
DD 46	YA_k	Q	FLP	KG	output variable
DW 48	YA_k	Q	NF	KF	output variable $BGUG \leq YA \leq BGOG$

+-----+ proposed format
 (KH, KM also permissible)
 +-----+ FLP = floating point number,
 NF = normalized fixed-point number
 +-----+ I = input, Q = output

Assignment of the control word STEU (data word DW 11 in the transfer DB)

DW 11 bit no.	Name	Significance
11.0	AUTO	= 1: automatic operation = 0: manual operation
11.1	XZ_ON	= 1: Another variable which must not be XW_k is supplied to the derivative unit via XZ input. = 0: XW_k is supplied to the derivative unit. The XZ input is ignored.
11.2	CTR_OFF	= 1: When the controller is called (OB 251) all variables (DW 20 through DW 48) except K, R, TI, TD, BGOG, BGUG, STEU, YH_k , W_k , Z_k and Z_{k-1} in the DB-RAM are erased once. The controller is disabled. The previous value of the disturbance is updated. = 0: Controller on
11.3	VELOC	= 1: Velocity algorithm = 0: Position algorithm
11.4 ¹⁾	MAN	= 1: The manipulated variable output last is retained if VELOC = 0 (position algorithm). The positioning increment dY_k is set = 0 if VELOC = 1 (velocity algorithm). = 0: If VELOC = 0 the value of manipulated variable YA output is brought exponentially in 4 sampling steps to the manually value set after switchover to manual operation. Then further manually set values are accepted directly at the controller output. If VELOC = 1 the manually set values are immediately enabled at the controller output. The limits are valid for manual operation. The following variables are updated during manual oper.: 1) X_k , XW_{k-1} , and PW_{k-1} 2) XZ_k , XZ_{k-1} , and PZ_{k-1} if STEU bit 1 = 1 3) Z_k and Z_{k-1} if STEU bit 5 = 0 The variable dD_{k-1} is set = 0. The algorithm is not calculated.
11.5	NO_Z	= 1: No feedforward control = 0: With feedforward control
11.6	PGDG	= 1: W_k -, YH_k input as floating point number = 0: input as left point number
11.7	VAR_FLP	= 1: The variables X_k , XZ_k , and Z_k are input as floating point number = 0: Input of variables as normal fixed-point number
11.8	BUMP	= 1: No bumpless changeover manual-automatic oper. = 1: Bumpless changeover manual-automatic oper.
11.9 to 11.15		Without significance

¹⁾ Only relevant for manual operation (AUTO = 0).

6.11.1 Initialize PID Algorithm (OB 250)

OB 250 initializes the PID algorithm and is called in the start-up OB's 20/21/22.

The parameters required for the initialization are contained in the transfer data block (DB x).

IMPORTANT!

The transfer data block must be called before OB 250 is called.

For data transfer each controller requires its own DBx ($x \leq 254$). The system program automatically generates a further DB $x + 1$ by copying the DB x into the data block RAM. This block is used by the controller as a data field in cyclic operation, the corresponding DB numbers must still be available. The data blocks DB $x + 1$ are the data interfaces between the controller and the user or I/O's.

Internally OB 250 uses OB 254 or OB 255 (duplication of data blocks). If an error occurs the processor recognizes an execution time error and will call OB 31. If OB 31 is not programmed the processor stops. The error identifiers in accumulator 1 then refer to OB 250.

Note! If DB $x + 1$ is not kept free during the initialization it will be used as a controller data field without any warning, that is, as long as its length is identical with that of a controller DB (48 data words); data words 20 through 48 will be erased. Otherwise the processor will stop.

Use of extended data blocks DX is possible instead of DB data blocks. Initialization is similar to that of DB data blocks.

6.11.2 Process PID Algorithm (OB 251)

OB 251 is called during cyclic program execution and processes the PID algorithm.

The controller should be called after the sampling time has elapsed. Keep to the following order:

- Call data block DB x + 1
- Load input data X_k , XZ_k , Z_k , and YH_k or a subset of these
- Convert input data to the correct format and transfer to DB x + 1
- Call OB 251 (process PID controller)
- Load output data YA_k from DB x + 1
- Convert data and transfer to process I/O.

Format of controller inputs and outputs

Internally the PID control algorithm uses the floating point format for numerical representation and may be supplied with floating point values. Supply of the PID controller algorithm is also possible using the normalized fixed-point format (see bit 6 and 7 in control word STEU). If this is used the controller will automatically convert the words to the floating point format with every call.

Adaptation of words from the input and output modules in the STEP5 program is faster if the left point format is used.

Inputs

W, YH, X, Z, and XZ may be input either as floating or left point numbers. Different memory locations have been reserved for each variable in the data transfer block.

Input as normalized fixed-point number

For explanations referring to the normalized fixed-point number: see the relevant paragraph.

Note! While keeping within the nominal input ranges of the analog input modules do not forget that the bit pattern for a certain input value differs from when the full input range is used. Taking this fact into consideration is extremely important when it comes to adjusting the setpoint. Otherwise it is possible that a setpoint input via the PG will not be reached although the actual value may far exceed the desired value.

If the analog-to-digital converter used supplies the negative numbers as number and sign it is important that the two's complement is formed from these values before transferring them to the controller DB. After that the binary digit 15 will have to be set = 1.

If the number -0 is possible as number and sign in the form of

1000000000000000

for the analog-digital converter used, this must not be converted to two's complement. The number must be transferred to the controller DB as +0:

0000000000000000

Output

The controller output YA exists in the DB as a normalized fixed-point and a floating point number. Taking into account the input and output modules used (ADC, DAC), the format must be converted for normalized fixed-point inputs and outputs before and after the controller is called in the STEP5 user program before they are transferred to or from the controller DB.

General notes

If STOS (STEU bit 8) is set to 0 the changeover from manual to automatic operation will be bumpless; i.e. an error signal of whatever value is corrected by the I action only. However, if $TI = TA/TN$ is selected = 0 (P or PD controller) the error signal will not cause a change of the manipulated variable when the changeover takes place.

This can be prevented by means of setting $STOS = 1$. This means that an error signal is corrected quickly when there is a manual-automatic changeover, irrespective of $TI = 0$. The manipulated variable jump thus created corresponds to the value of the error signal, which means that it is not arbitrary in the sense of a disturbance of controller operation.

Bit 0 and 1 of MERK can be displayed, if desired, in order to show that the manipulated variable (for velocity algorithm the positioning increment) lies between the upper and lower limits. Since these bits are evaluated by the algorithm for disabling the I action, overwriting is not permitted.

Reloading of controller data blocks $DB\ x + 1$ during cyclic operation is not permitted.

If two or more controllers are cascaded the following is to be taken into consideration:

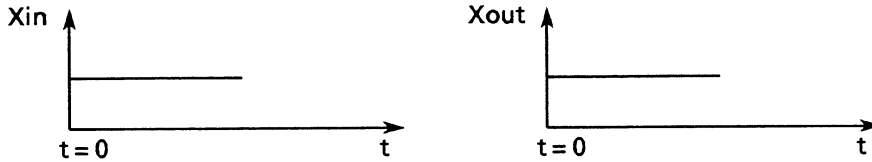
- If the cascade is split either all controllers will have to change to manual operation simultaneously in order to prevent any controller drift due to the I action or at least the controller of the outer loop must be operated manually in order to ensure that the last manipulated variable, which corresponds to the setpoint of the inner loop, is maintained or changed to a safe value.
- If the cascade is to be closed both loops should simultaneously go over to automatic or at least the inner circuit in order to ensure that the manipulated variable of the outer circuit is taken as the setpoint.

If the controlled system is disconnected from the controller and directly adjusted at the actuator following changeover to manual operation then the manipulated variable thus obtained must be supplied to the controller via the manual input. This in turn ensures that when the changeover from manual to automatic operation is made the controller output will correspond to the manipulated variable set during manual operation. In the case of the velocity algorithm this will be the change of the manipulated variable.

Controller parameters

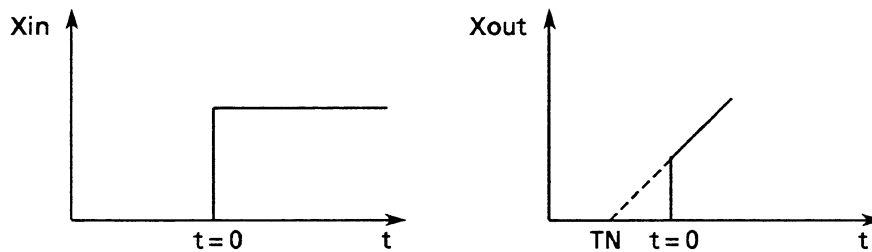
P controller

The parameter for a P controller is K. K is the quotient of output and input value: $K = X_{out}/X_{in}$.



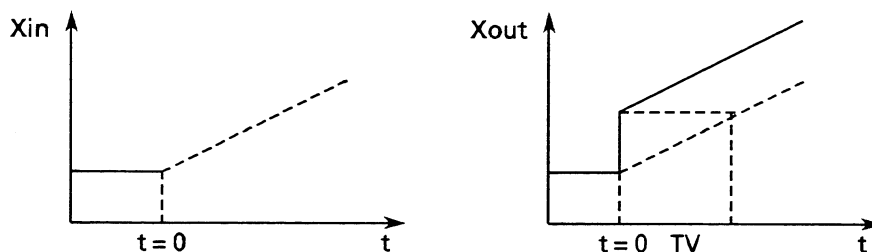
PI controller

The parameters for a PI controller are the proportional coefficient K and the reset time T_N . The proportional coefficient K is the quotient of output and input value and determines the P action. The reset time T_N is the time required to respond achieve the same change in the manipulated variable due to the I action as is brought about by the P action.



PD controller

The parameters for a PD controller are the proportional coefficient K (see above) and derivative time constant T_V . The derivative time constant is the time a P controller would require with a constant rate of change of the input variable in order to bring about the same change in the output variable that is brought about immediately by the D action of a PD controller. In order to determine the derivative time constant a linear change in the input variable is assumed and not a jump function.



PID controller

The parameters for a PID controller are the proportional coefficient K, the reset time TN and the derivative time constant TV. They in turn determine the P, I, and D actions.

Parameter change

The P action of the manipulated variable is obtained on the basis of the following formula:

$$P \text{ action} = KP \cdot R(XW_k - XW_{k-1})$$

If K or R are changed during automatic operation this will only have an effect on subsequent changes of the errors signal XW_k . The current value of the manipulated variable is not affected by the parameter change. This response allows for a bumpless parameter change. However, if this response is undesirable it may be eliminated by the following calculation (example of a KP change). This calculation is to be executed only once for every parameter change:

$$Y_{k-1} = Y_{k-1} + XW_{k-1}(KP_{\text{new}} - KP_{\text{old}})$$

If the following program is used in the case of a parameter change the controller will respond just as an analog controller:

```
:L  KPnew          load KPnew
:L  KPold          load KPold
:-G
:L  DD38            XWk-1
:xG
:L  DD44            Yk-1
:+G
:T  DD44            = Yk-1
```

Abbreviations for PID controllers

dY_k positioning increment calculated
dZ_k disturbing increment
FLP floating point representation
k k times sampling
K proportional coefficient
NF normalized fixed-point representation
OG upper limit (limiter)
R R parameter
TA sampling time
TD TV/TA
TI TA/TN
t sampling instant = k*TA
TN reset time
TV derivative time constant
UG lower limit (limiter)
W_k setpoint
X_k actual value
XW_k error signal
Y_k manipulated variable calculated
YA_k manipulated variable (positioning increment or man. variable)
Z_k disturbance

Normalized fixed-point number

One word is required for the representation of a normalized fixed-point number in a data block. The following example illustrates the difference between a fraction represented decimally, in binary and the representation at the programmer using the KF format.

Fraction in		Fixed point number
decimal repres.	binary repres.	
-0.999..	1000000000000001	-32767
-0.75	1010000000000000	-24576
-0.5	1100000000000000	-16384
-0.25	1110000000000000	- 8192
0	0000000000000000	0
+0.25	0010000000000000	+ 8192
+0.5	0100000000000000	+16384
+0.75	0110000000000000	+24576
+0.999...	0111111111111111	+32767

Negative normalized fixed-point numbers in a binary representation are obtained by creating the complement of two of positive normalized fixed-point numbers.

Normalized fixed-point numbers (NF) may be converted to the values represented at the programming unit (KF) on the basis of the following relation:

$$NF:32767 = KF$$

$$\text{with } -1 < NF < +1 \quad \text{and} \quad -32767 \leq KF \leq +32767$$

7 Extended Data Block DX 0

You have the option of adapting certain functions of the system program to your requirements by entering different settings in the DX 0 as an alternative to the standard presettings (marked "P" in the following table).

The standard presettings of the system program (P) are set automatically during each cold restart. DX 0 is evaluated after that. If DX 0 has not been programmed the standard presettings remain valid. If DX 0 has been programmed the settings entered by the user will be valid.

IMPORTANT!

Alteration of or input to DX 0 will be effective only if a cold restart is carried out.

Structure of DX 0

DX 0 is composed of three sections:

1. the start identifier for DX 0 (DW 0, 1 and 2)
2. several units of different length (depending on the number of parameters)
3. the end identifier EEEE.

The numerical values stated correspond to the hexadecimal format.

Formal structure:

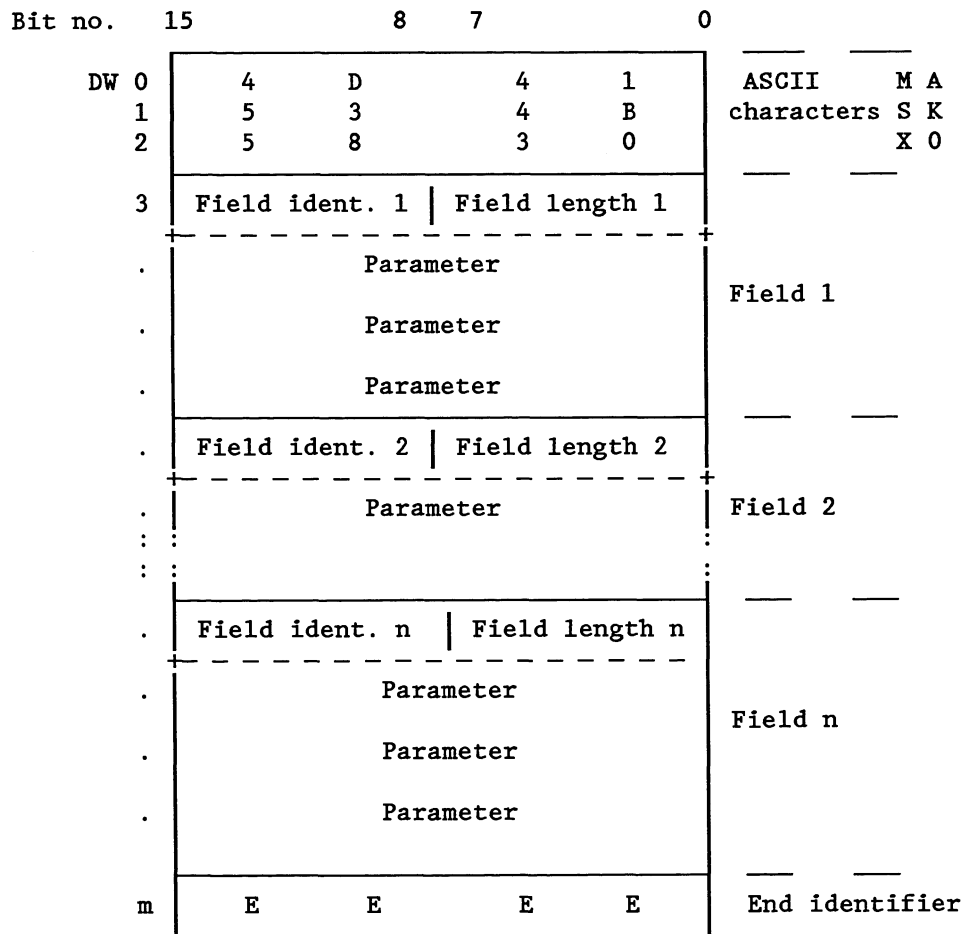


Fig. 7-1: Structure of DX 0

Examples for input of DX 0

Start identifier	DW 0:	KH= 4D41	
	DW 1:	KH= 534B	
	DW 2:	KH= 5830	
<hr/>			
Field identifier/length	DW 3:	KH= 0203	
Parameter (occupies 1 DW)	DW 4:	KH= 3001	Field 1
Parameter (occupies 2 DWs)	DW 5:	KH= BB00	
	DW 6:	KH: 0000	
<hr/>			
Field identifier/length	DW 7:	KH= 0402	
Parameter (occupies 2 DWs)	DW 8:	KH= 1000	Field 2
	DW 9:	KF= 4000	
<hr/>			
End identifier	DW10:	KH= EEEE	

One **field** in DX 0 consists of 1 to n data words.

These contain

- the field identifier
- the field length
- the field parameters.

The **field identifier** states the *significance* of the parameters following. Each block is assigned to a certain system program section or a certain system function (e.g. field identifier '04' --> cyclic program execution).

The **field length** states the *number of data words* occupied by the the parameters following.

The possible *parameters* are listed on the following pages.

Note the following points when assigning parameters to DX 0:

- The order in which you enter the individual fields is unimportant.
- There is no need to specify fields that are not required.
- If a field exists several times the field entered last is valid.
- The order in which you enter the individual parameters is unimportant.
- There is no need to specify that are not required.
- If a parameter is specified several times the last one is valid.

IMPORTANT!

After the last field is entered DX 0 must be completed with the end identifier **EEEE!**

Field identif. /length	Parameter	Significance 1)
Cyclic program execution:		
04xx 2)	1000 yyyy	Duration of cycle time in milliseconds Presetting: yyyy = 150 ms, permissible: $1 \leq yyyy \leq 1770$ (hex.) 1 ms to 6000 ms (decimal)
	4000	P Updating of process image of interprocessor communication flags without semaphore protection
	4001	Updating of process image of interprocessor communication flags semaphore protected (in the field)
Interrupt processing		
06xx 2)	1006	Processing of time, controller, and process interrupt at STEP5 command boundaries 3)
	1008	Processing of controller and process interrupt at STEP5 command boundaries, of time interrupt at block boundaries
	100A	Processing of process interrupt at STEP5 command boundaries, of controller and time interrupt at block boundaries
	100C	P Processing of time, controller, and process interrupt at block boundaries

The parameters listed above are valid for the CPU 928 version which works only with the 100-ms time interrupt. The new version of the CPU 928 allows up to 9 time interrupt OBs to be programmed. For this version new identifiers for DX 0 have been provided (see next page).

1) P = Presetting if no DX 0 has been loaded or if field is missing

2) xx = Field length (number of data words assigned to parameters)

3) **Note:**

If you require the processing of interrupts at STEP5 command boundaries in DX 0, do not forget that, in the case of interruptions, the commands 'TNB' and 'TNW' will not be processed completely since they contain pseudo command boundaries. This also applies to a small number of special function organization blocks, standard function blocks, and controller FBs.

Process interrupt = process-interrupt-driven program processing

Time interrupt = time-driven program processing

By programming the new parameters in DX 0, the following settings for the processing of interrupts are possible:

Parameters	Time interrupts									Controller	Process interrupt	Former parameters	
	5 s	2 s	1 s	500 ms	200 ms	100 ms	50 ms	20 ms	10 ms				
122C P	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	100C P
1224	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	100A
121C	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1008
1216	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-
1214	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-
1212	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-
1210	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-
120E	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-
120C	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-
120A	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-
1208	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-
1206	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1006

P = Preset value

- Interruption at block boundaries
- Interruption at command boundaries

Field identif.	Parameter	Significance 1)
06xx 2)	2000	P Process interrupt signal, level-triggered
	2001	Process interrupt signal, edge-triggered
Error handling:		
10xx 2)	1000	Collision of two time interrupts processing P System stop, if event occurs and OB 33 has not been loaded.
	1001	No system stop, if event occurs and OB 33 has not been loaded.
	1200	Controller error processing P System stop, if event occurs and OB 34 has not been loaded.
	1201	No system stop if event occurs and OB 34 has not been loaded.
	1400	Cycle error processing P System stop, if event occurs and OB 26 has not been loaded.
	1401	No system stop, if event occurs and OB 26 has not been loaded.
	1800	Command code error processing P System stop, if event occurs and OB 27/29/30 has not been loaded.
	1801	No system stop, if event occurs and OB 27/29/30 has not been loaded.

1) P = Presetting if no DX 0 has been loaded or if field is missing

2) xx = Field length (number of data words assigned to parameters)

Command code error = Substitution error, opcode error, parameter error

Execution time error = Call of a block that has not been loaded, transfer error or other execution time errors

Field identif.	Parameter	Significance 1)
10xx 2)	1A00	Execution time error processing P System stop, if event occurs and OB 19/31/32 has not been loaded.
	1A01	No system stop, if event occurs and OB 19/31/32 has not been loaded.
	1C00	Addressing error processing P System stop, if event occurs and OB 25 has not been loaded.
	1C01	No system stop, if event occurs and OB 25 has not been loaded.
	1E00	Acknowledgement delay error P System stop, if event occurs and OB 23/24 has not been loaded.
	1E01	No system stop, if event occurs and OB 23/24 has not been loaded.
EEEE		End identifier

1) P = Presetting if no DX 0 has been loaded or if field is missing

2) xx = Field length (number of data words assigned to parameters)

Example A: Parameter assignment of the DX 0:

You intend to use three processors for multiprocessor operation: processors A, B, and C. Processors A and B cooperate a lot, frequently exchange data and run an extensive start-up program. Processor C runs a short and time-critical program and is largely independent of the other processors.

As a standard feature, all processors start cyclic program processing together in multiprocessor operation, i.e. the processors wait until they have all completed their start-up procedures. Then they start cyclic program processing together.

Since processor C executes its program independent of the other processors and has a very short start-up program no start-up synchronization is required for it. By assigning parameters to DX 0 you can allow processor C to start cyclic program processing immediately after the start-up has been completed, without waiting for processors A and B.

This is how you should program DX 0:

<u>DX 0</u> Start identifier	DW 0:	KH= 4D41
	DW 1:	KH= 534B
	DW 2:	KH= 5830
1st field identifier/length	DW 3:	KH= 0201
Parameter 1	DW 4:	KH= 2001
End identifier	DW 5:	KH= EEEE

If you have loaded this DX 0 in the program memory it will be effective as soon as the next cold restart is executed. Since processor C has an extremely short start-up program and will not wait for processors A and B the green RUN LED will come on immediately. However, the BASP signal (command output inhibit) will not be cancelled until all three processors have completed their start-up. This means that processor C is denied access to the digital I/O's.

Example B: parameter assignment of the DX 0:

The following parameter assignment for DX 0 is used to

- a) disable address error monitoring,
- b) disable timer location updating,
- c) set the cycle time to 4 s.

<u>DX 0</u> Start identifier	DW 0:	KH= 4D41
	DW 1:	KH= 534B
	DW 2:	KH= 5830
1st field identifier/length	DW 3:	KH= 0203
Parameter	DW 4:	KH= 3001
Parameter *)	DW 5:	KH= BB00
	DW 6:	KH= 0000
2nd field identifier/length	DW 7:	KH= 0402
Parameter *)	DW 8:	KH= 1000
	DW 9:	KH= 4000
End identifier	DW10:	KH= EEEE

- *) Parameters which occupy two data words must be counted as '2' when the unit length is specified.

Assigning these parameters to DX 0 has the following effect on program execution:

- The section of the process image with no I/O modules assigned may be used as additional "flag area".
- The execution time of the system program is reduced since no timer locations are updated.
- A cycle error will not be identified unless the execution time of the cyclic user program together with that of the system program exceeds 4 s.

8 Memory Assignment and Memory Organization

The total memory area of the CPU 928 is basically divided into the following areas:

	Width:
1. User memory for OB, FB, FX, PB, SB, DB, DX	(16 bits)
2. DB-RAM for data blocks, shift registers	(16 bits)
3. - interface data area: RI, RJ	(16 bits)
- system area: RS, RT	(16 bits)
- counters: C	(16 bits)
- timers: T	(16 bits)
4. - flags: F	(8 bits)
- process image of inputs and outputs: PII, PIO	(8 bits)
5. I/O area:	(8 bits)
- P I/O's	
- O I/O's	
- interprocessor communication flags	
- COR	
- pages	
- distributed I/O's	

Refer to the memory assignment diagram on the following page for the exact addresses of the areas.

IMPORTANT!

STEP5 access to a memory location within an operand area (e.g. flags) should never be carried out via the absolute address of this memory location, but always relative to the base address of the operand area. The base addresses of all operand areas are stored in the area of the system data (RS area) (see "system data assignment").

8.1 Address Distribution in the CPU 928

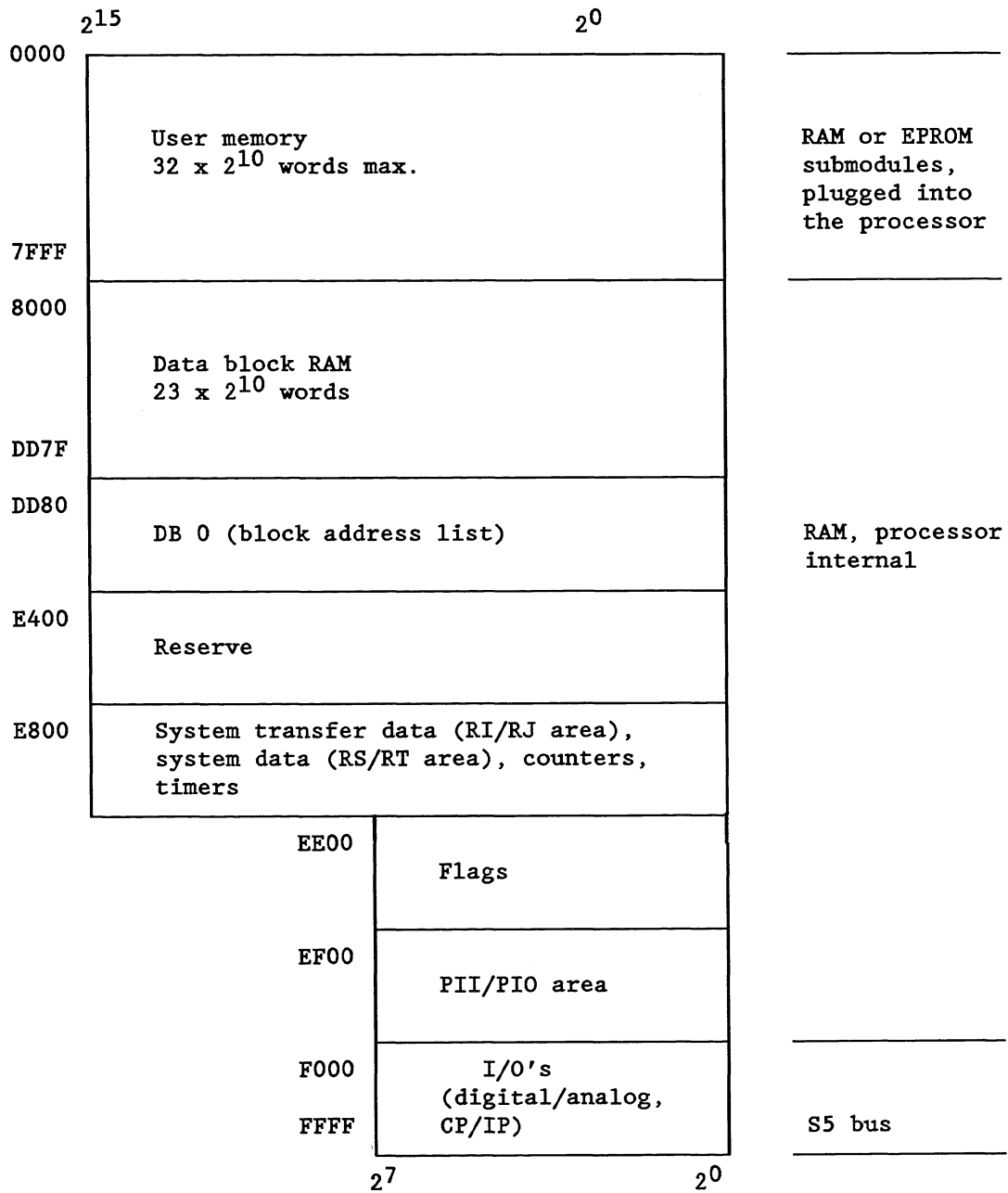


Fig. 8-1: Address distribution in the CPU 928

Address areas for I/O / programming

Area (absolute address)	Is addressed with	Parameters
EF00 PII (process input image) EF7F	L IB/T IB L IW/T IW L ID/T ID A I/AN I/O I/ON I S I/R I/=I	0 to 127 0 to 126 0 to 124 0.0 to 127.7
EF80 PIO (process output image) EFFF	L QB/T QB L QW/T QW L QD/T QD A Q/AN Q/O Q/ON Q S Q/R Q/=Q	0 to 127 0 to 126 0 to 124 0.0 to 127.7
F000 digital I/O inputs/outputs F07F	L PY/T PY L PW/T PW	0 to 127 0 to 126
P I/O with process image		
F080 digital or analog I/O inputs/outputs FOFF	T PY/T PY T PW/T PW	128 to 255 128 to 254
P I/O without process image		
F100 extended I/O inputs/outputs F1FF O I/O	L OB/T OB L OW/T OW	0 to 255 0 to 254

Fig. 8-4: Address distribution for I/O / programming

Access to the I/O

Using STEP5 commands you can either access the I/O directly or via the process image. Do not forget that a process image exists only for input and output bytes of the P I/O with byte addresses from 0 to 127!

Direct access to the I/O: L/T PY, L/T PW, L/T OB, L/T OW

The inputs and outputs are read or set at the time of command processing. The outputs in the process image are corrected to match digital I/O (0 to 127).

Access to the I/O
via the process image:

L/T IB, L/T IW, L/T ID,
L/T QB, L/T QW, L/T QD,
A/AN/O/ON I, =A etc.

The process image only will be altered at the time of command processing. It is not until the end of the cycle that the complete new status of the process image is output to the I/O.

8.2 Memory Organization in the CPU 928

The user memory comprises the memory area from 000H to 7FFFH. When the individual blocks of the user program are loaded they are deposited in the memory in any order (ascending addresses). If the user memory is full the data blocks will be deposited in the internal DB-RAM (8000H to DD7FH).

Using the online function 'MEMCON' (maximum memory capacity) you will be supplied the address (hexadecimal) of the memory location containing the block end instruction of the last block in the memory.

The 'old' block in the memory will be declared invalid (i.e. the start identifier is overwritten) if blocks are corrected and a new block will be entered in the memory. In a similar manner, the blocks in the memory are not actually erased when blocks are erased, instead they are simply declared invalid. This means that erased and corrected blocks still occupy memory area.

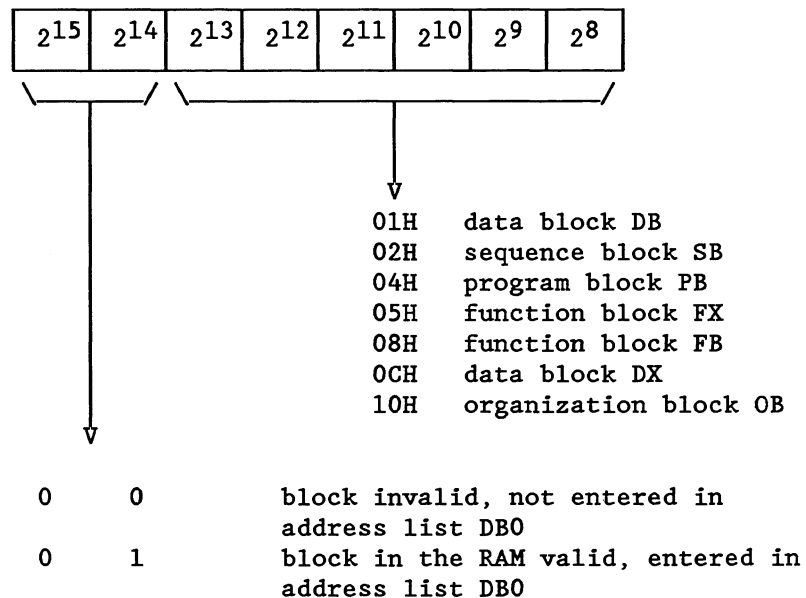
The online function 'COMPRESS' eliminates all invalid blocks in the memory and shifts the valid blocks together.

8.2.1 Block Headers in the User Memory and the DB-RAM

Each of the blocks in the memory starts with a header of 5 words.

1st word: block start identifier: 7070H

2nd word: high byte = block type



Low byte = block number

The block number (0 to 255) is contained in the low byte of the 2nd header word and is coded as a binary number: 00 to FFH.

3rd word: The identifiers for the programmer are entered in the high byte of the 3rd word; part of the library number is entered in the low byte.

4th word: The fourth word contains the remainder of the library number.

5th word: The length of the block including the header is entered in the 5th word (low and high byte). The information is in words.

8.2.2 Block Address Lists in Data Block DB 0

The data block DB 0 contains the address list with the start addresses of all blocks which exist in the user memory or the DB-RAM of the processor. This address list is created by the system program after the power is switched on and updated automatically when blocks are altered or input with the programmer.

An address list of 256 words exists for each block type in DB 0 which is reserved for the corresponding block type. Blocks that have not been loaded have the start address '0'.

The start addresses of the individual block address lists are also entered in the system data RS 32 through RS 38:

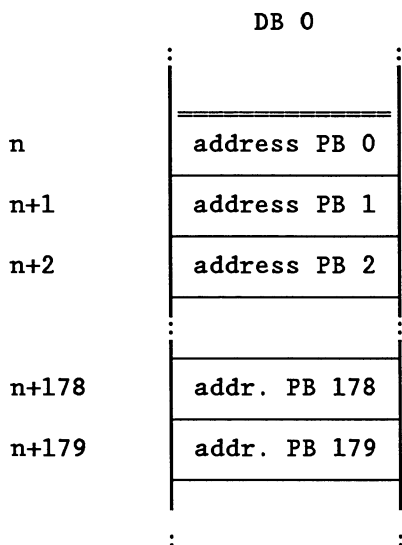
RS 32: Initial address of DX-address list
RS 33: Initial address of FX-address list
RS 34: Initial address of DB-address list
RS 35: Initial address of SB-address list
RS 36: Initial address of PB-address list
RS 37: Initial address of FB-address list
RS 38: Initial address of OB-address list (only 48 words long)

All of the start addresses always point to the first data word following the block header:

- in data blocks, to data word DW 0!
- in code blocks, to the first STEP5 instruction!
(in FBs, to the JU command)

Entering the block addresses in DB 0:

n = start address of PB address list (= contents of RS 36)



If the value '0' is entered as the address the block has not been loaded.

This is how you determine the address of any block in the memory:

Example: Start address of FB 40

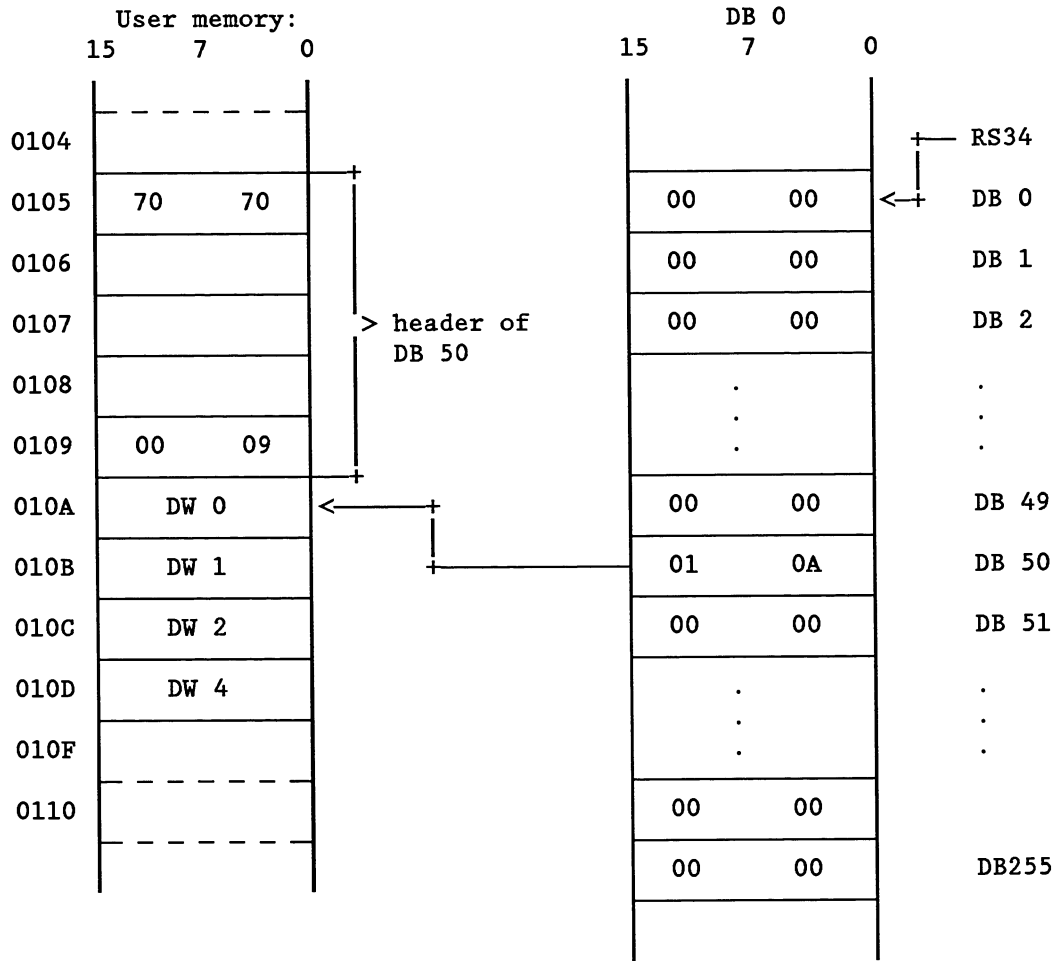
```

:L RS 37          base address FB address list
:L KB 40          + FB number
:+F              = address of the memory location that
:                contains the start address of FB 40
:
:LIR 1           load start address of FB 40 in accu 1
:                (block does not exist if start address = 0)
:
:
:

```

Example: Start address and length of data block DB 50

a) using indirect memory access



```

:L RS34      load base address of DB address list
:L KB50      calculate address of entry for DB 50
:+F          and load initial address
:LIR 1      in accu 1
:L KB0      if block does not exist,
:!=F        jump to NIVO marker
:JC =NIVO
:ENT        load start address of DB 50 in accu 3
:TAK        and accu 1
:L KF-1     reduce initial address by one,
:+F        load block length
:LIR 1      in accu 1
:
:
NIVO :.....
    
```

Result: accu 1-L: length of DB 50
 accu 2-L: start address of DB 50

b) 'test data blocks (DB/DX)' using the special function OB 181

OB 181 contains the same function as described in a). In addition, the OB will test whether the data block is located in the user memory (RAM or EPROM) or in the DB-RAM.

```

:L KY1,50          data block DB 50
:JU OB181          "test data block (DB/DX)"
:JC NIVO
:JM =PROM
:JZ =ANWE
:JP =DBRA

NIVO :              data block does not exist
:
:BEU

PROM :              data block located in the
:                   user memory (EPROM module)
:BEU

ANWE :              data block located in the
:                   user memory (RAM module)
:BEU

DBRA :              data block is located in the DB-RAM
:
:BE

```

Result: accu 1-L: start address of DB 50
 accu 2-L: length of DB 50
 RLO = 1: if DB 50 does not exist

8.2.3 RI/RJ Area

The RI area is an area of 256 words in the internal system RAM of the processor. It occupies addresses from E800H through E8FFH.

The RJ area is an area of 256 words in the internal system RAM of the processor. It occupies addresses from E900H through E9FFH.

The complete RI area (RI 0 to RI 255) and the complete RJ area (RJ 0 to RJ 255) is available for the user's own requirements.

The RI/RJ area is initialized only during overall reset.

8.2.4 RS/RT Area

The RS area is an area of 256 words in the internal system RAM of the processor. It occupies addresses from EA00H through EAFFH.

IMPORTANT!

Writing is only permissible to system data words RS 1, RS 60 to RS 63 and RS 133:

- RS 60 to RS 63 are available for user requirements.
- RS 1 and RS 133 have a fixed significance and affect program processing. Writing is only permissible if *valid identifiers* are used!

IMPORTANT!

All other system data can only be read:

- They contain information for the system programmer and non-public system variables.
- Writing to these system data may affect the operation of the programmable controller as well as the programmer connected!

The RT area is an area of 256 words in the internal system RAM of the processor. It occupies addresses from EB00H through EBFFH.

The complete RT area (RT 0 to RT 255) is available for the user's own requirements.

The RS/RT area is initialized only during overall reset.

System data assignment of RS areas

RS	Designation		Addr.
0	Interrupt condition code (CAUSE OF INTERR.)		EA00
1	Interrupt condition code erase word (UALW)		EA01
2	Interrupt condition codes (collected) (UAMK)		EA02
3	Start-up error identifier indication		EA03
4			EA04
5	Stop id's	Start-up id's	EA05
6	Cycle id's	Module id's	EA06
7	Overall reset.id's	Error id's (H)	EA07
8	Error id's (F)	Error id's (L)	EA08
9	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	Act. id number	EA09
10	Base addr. of input process interface module		EA0A
11	Base addr. of output process interface module		EA0B
12	Base address process image of inputs		EA0C
13	Base address process image of outputs		EA0D
14	Base address flag area		EA0E
15	Base address timer area		EA0F
16	Base address counter area		EA10
17	Base address interface data area		EA11
18	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	PC-SW issue	EA12
19	End address of user module memory		EA13
20	Base address of system data area		EA14
21	Length of DB address list		EA15
22	Length of SB address list		EA16
23	Length of PB address list		EA17

XXXXXXXXXXXXXXXX: assignment not enabled

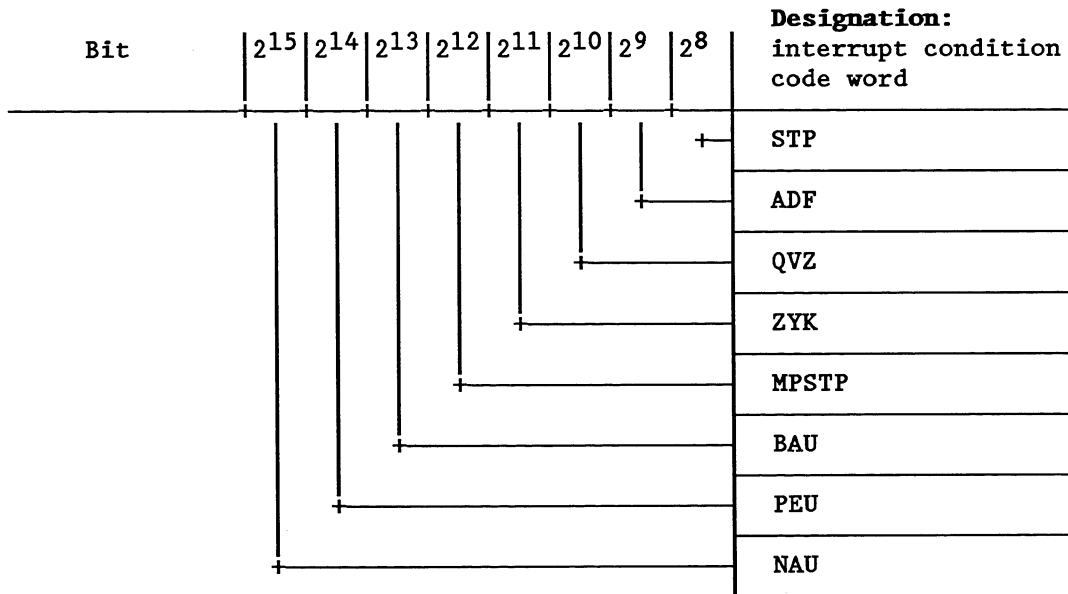
24	Length of FB address list	EA18.
25	Length of OB address list	EA19
26	Length of FX address list	EA1A
27	Length of DX address list	EA1B
28	Length of address list DB (DB 0)	EA1C
29	Slot id's	CPU id's 2 (type)
30	Length of block header information	EA1E
31	CPU id's 1	SW issue PG interface
32	Base address of DX address list	EA20
33	Base address of FX address list	EA21
34	Base address of DB address list	EA22
35	Base address of SB address list	EA23
36	Base address of PB address list	EA24
37	Base address of FB address list	EA25
38	Base address of OB address list	EA26
39	XX	EA27
40	XX	
41	XX	
42	XX	
43	XX	
44	XX	
45	XX	
46	XX	
47	XX	
48	XX	
49	XX	
50	XX	
51	XX	
52	XX	
53	XX	
54	XX	EA36
55	Counter for 1 hour (up to 3599 s ., hex.)	EA37
56	Reserved for handling blocks	EA38
57		
59		EA3B
60	Reserved for user purposes	EA3C
61		
63		EA3F
64	Reserved for system program	EA40
65		
66		
127		
128	XX	EA80
129	XX	EA81

130	Identifier 'control'	EA82
131	Condition code word 'Disable all interrupts'	EA83
132	Condition code word 'Delay all interrupts'	EA84
133	Identifier 'Process image updating'	EA85
134	XX	EA86
135	CC word 'Disable indiv. time interrupts'	EA87
136	XX	EA88
137	CC word 'Delay individual time interrupts'	EA89
138	XX	EA8A
:	:XX:	:
:	:XX:	:
:	:XX:	:
:	:XX:	:
:	:XX:	:
255	XX	EAFB

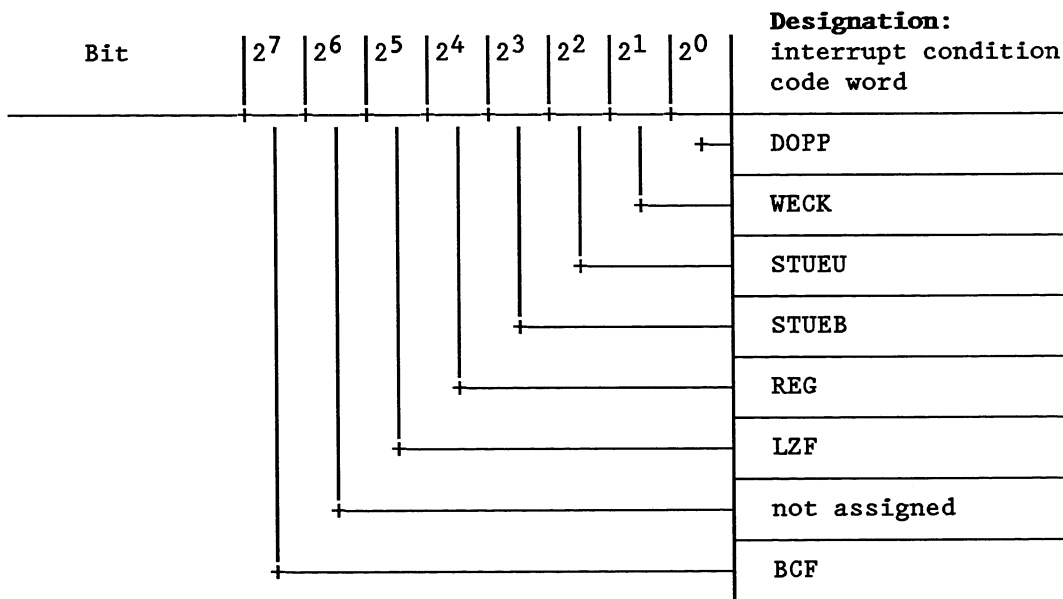
Information on some system data (on the internal structure of the processor, the issue of the software, CPU identifier etc.) are also available using the online function 'SYSTEM PARAMETERS'.

As a supplement to the above list the bit assignments of some system data are shown in the following. Evaluation of these is possible by means of STEP5 instructions or the PG (for abbreviations see 5.3).

System data: **RS 0** address: EA00 (HIGH)

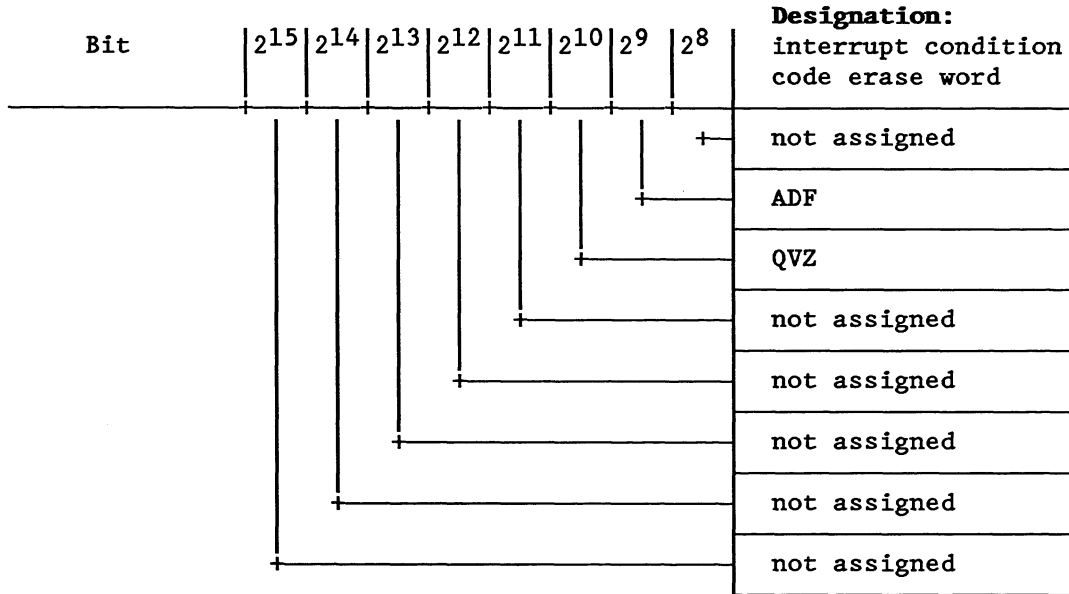


System data: **RS 0** address: EA00 (LOW)

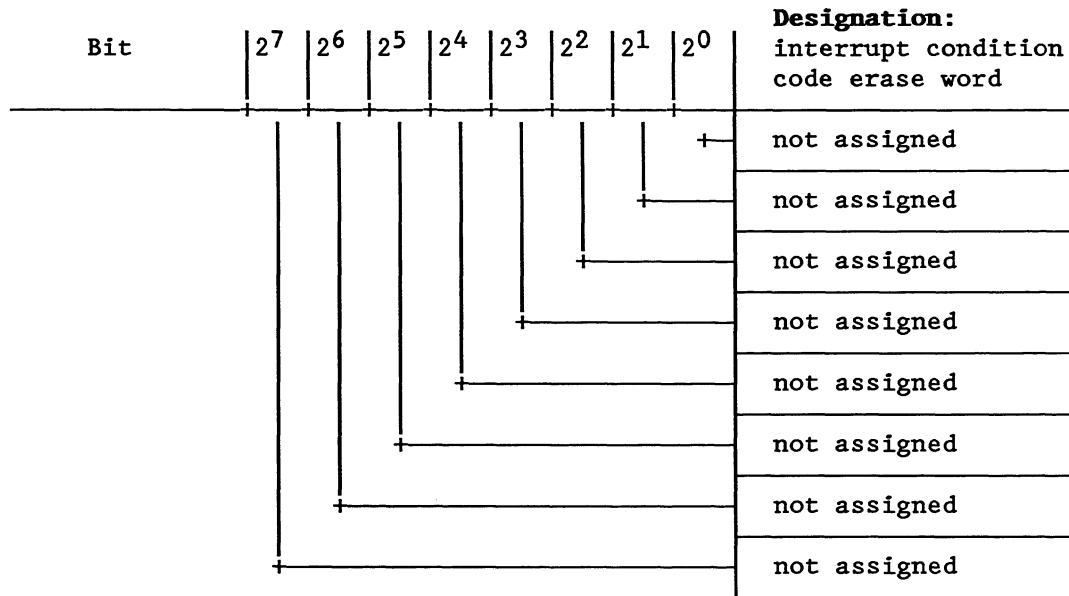


System data RS0 corresponds to the 'CAUSE OF INTERR.' in the ISTACK. If, e.g., an execution time error occurs during program execution, bit 2⁵ will be set. If program level LZF has been completely processed, bit 2⁵ will be reset.

System data: **RS 1** address: EA01 (HIGH)



System data: **RS 1** address: EA01 (LOW)



RS 1: Active interface, released for the user (see following page)!

Interrupt condition code erase word (system data RS 1):

Setting of bit 9 or bit 10 of the UALW (interrupt condition code erase word) will result in the QVZ or ADF following next being ignored and program execution therefore not being affected. The system program will reset the bit if a QVZ or an ADF occurs.

Each program level has a UALW of its own!

The following example shows a test to establish whether a module may be called using a certain I/O address. If the module does not exist an acknowledgement delay (QVZ) will be prevented by means of the UALW and a program intended for this particular event will be run. A test also establishes whether a certain I/O address is entered in DB 1. If not, an addressing error is prevented by means of the UALW and a special program will be run.

FB 10

NAME: PERITEST

DECL: PADR

I/Q/D/B/T/C: E BI/BY/W/D: BY

DECL: MASK

I/Q/D/B/T/C: E KM/KH/KY/KS/KF/KT/KC/KG: KM

```

:L RS1          load UALW
:T RS60         and save
:LW =MASK       set QVZ or ADF bit
:OW
:T RS1          rewrite UALW
:L =PADR        individual I/O access or access to
:L RS1          process image
:LW =MASK       mask QVZ or ADF bit
:AW
:L RS60         rewrite old UALW to allow next
:T RS1          QVZ or ADF to be identified
:TAK
:BE

```

FBO

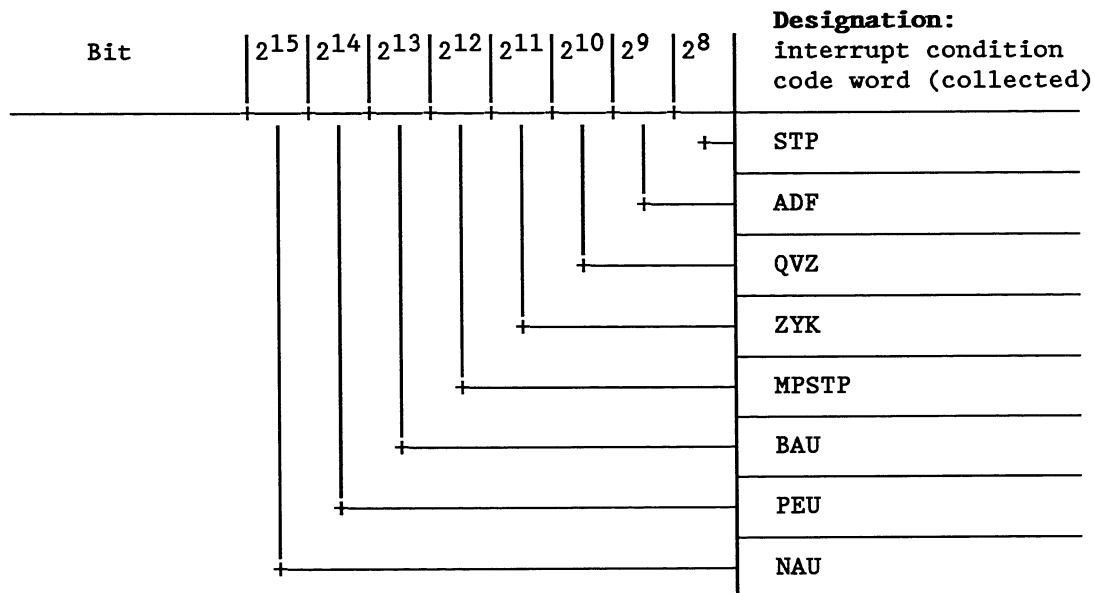
NAME:L

```

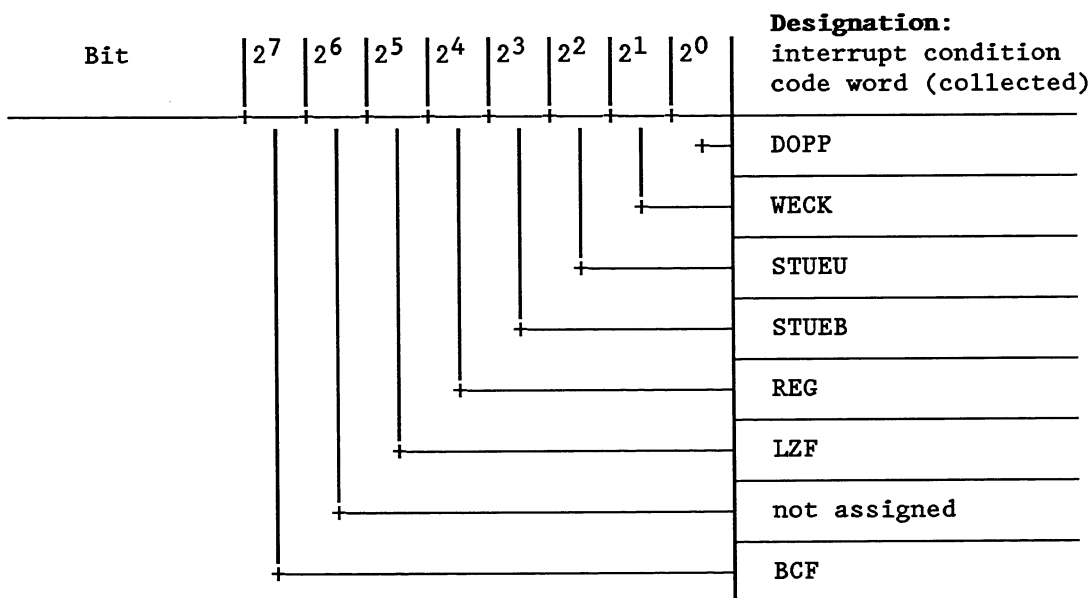
:JU FB10
NAME:PERITEST   test whether,
PADR:  PB128     using I/O address 128
MASK:  KM00000100 00000000 a module can be called
:JW  =M001
:..
:..            this program section is run if
:..            module cannot be
:..            called
M001:
:JU FB10
NAME:PERITEST   test whether
PADR:  QB4       a module is entered in DB 1
MASK:  KM00000010 00000000 with I/O address 4
:JW  =M002
:..
:..            this program section is run if
:..            I/O address is not
:..            entered
M002:
:BE

```


System data: **RS 2** address: EA02 (HIGH)



System data: **RS 2** address: EA02 (LOW)



The interrupt condition code word (collected) (UAMK in the ISTACK, see following page) may only be read!

Interrupt condition code word (collected) (system data RS 2)

The 16 bits of the interrupt condition code word (collected) correspond to the possible causes of errors listed under 'CAUSE OF INTERR.' in the ISTACK.

If an error occurs the corresponding bit will be set.

Example:

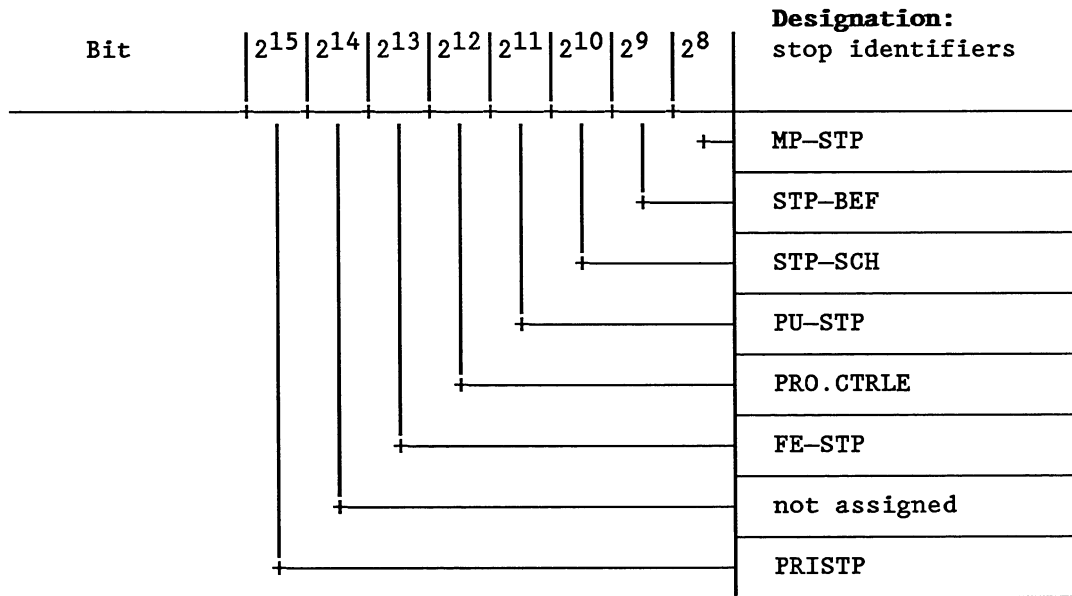
If the processor stops due to an addressing error (ADF) bit 9 in the UAMK will be set. If, during the processing of the ADF, a command code error (BCF) occurs bit 7 in the UAMK will also be set.

Contents of UAMK (binary):	00000010 10000000
Representation (hexadecimal)	
in the ISTACK:	0280

In contrast to the ISTACK, where only the last error that occurred is listed under 'CAUSE OF INTERR.', the UAMK contains all errors added up that have occurred up to then (ISTACK depth 05: 5 bits have been set in the UAMK). The contents of the UAMK can be evaluated if they are converted from hexadecimal code to binary code. This enables you to determine the error responsible for the stop page.

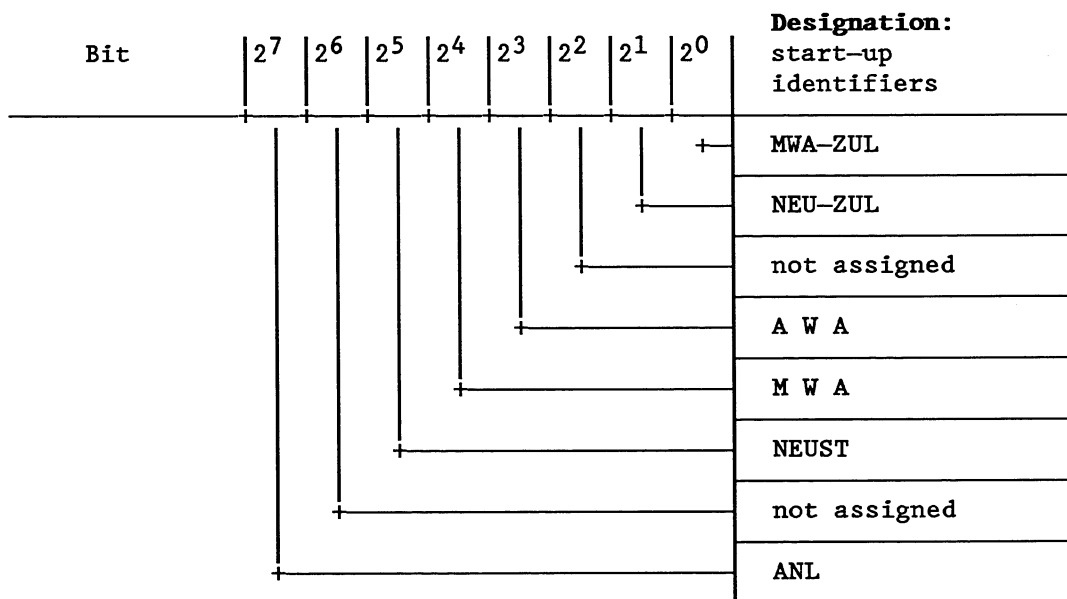
The error bits are reset as soon as the corresponding error program level has been processed completely and exited.

System data: RS 5 address: EA05 (HIGH)



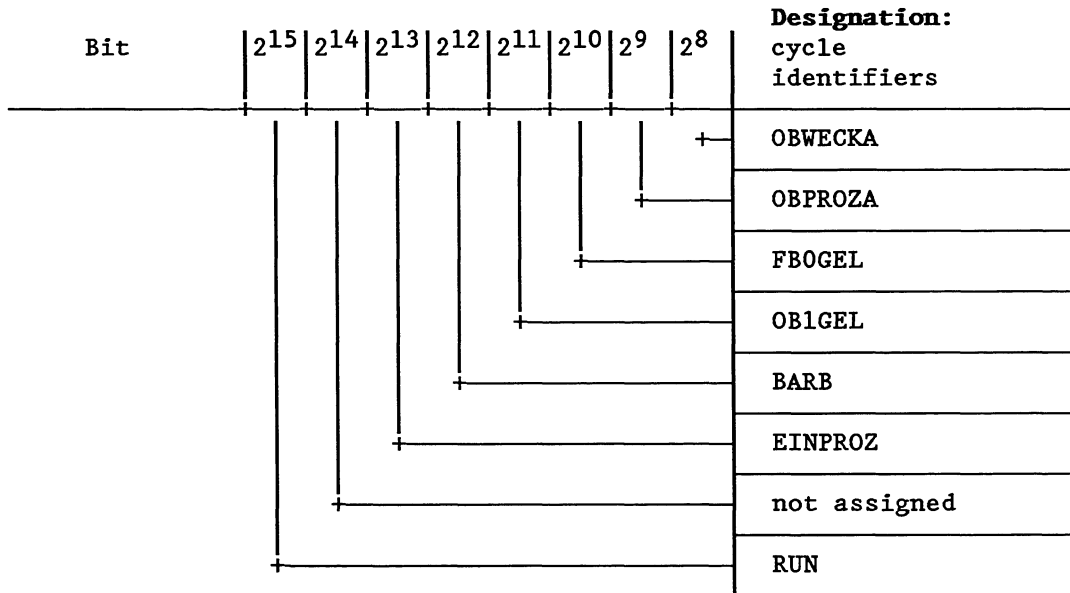
---> 1st line control bits

System data: RS 5 address: EA05 (LOW)



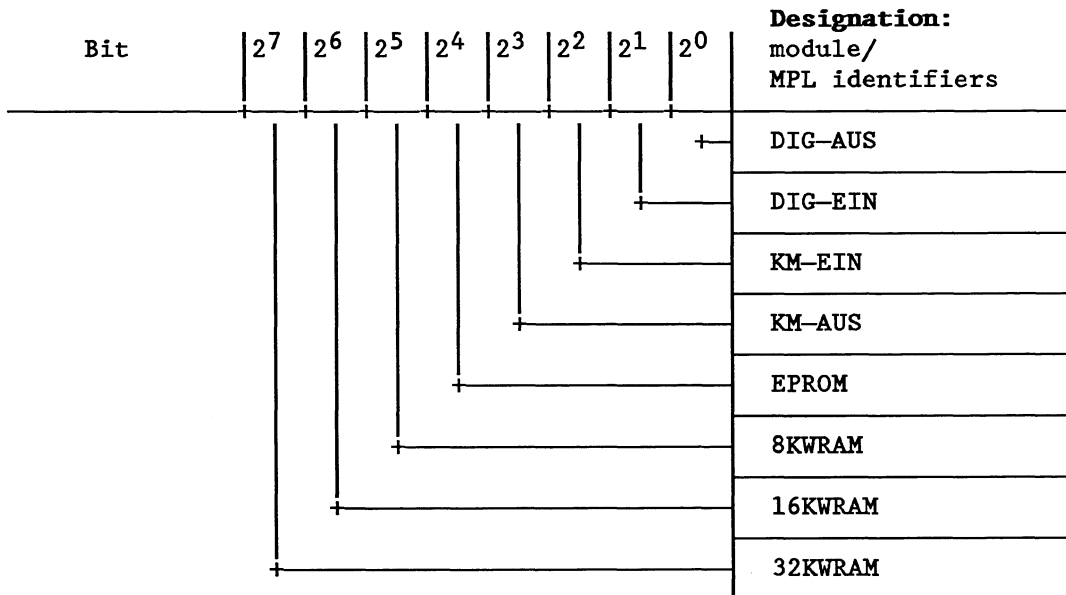
---> 2nd line control bits

System data: **RS 6** address: EA06 (HIGH)



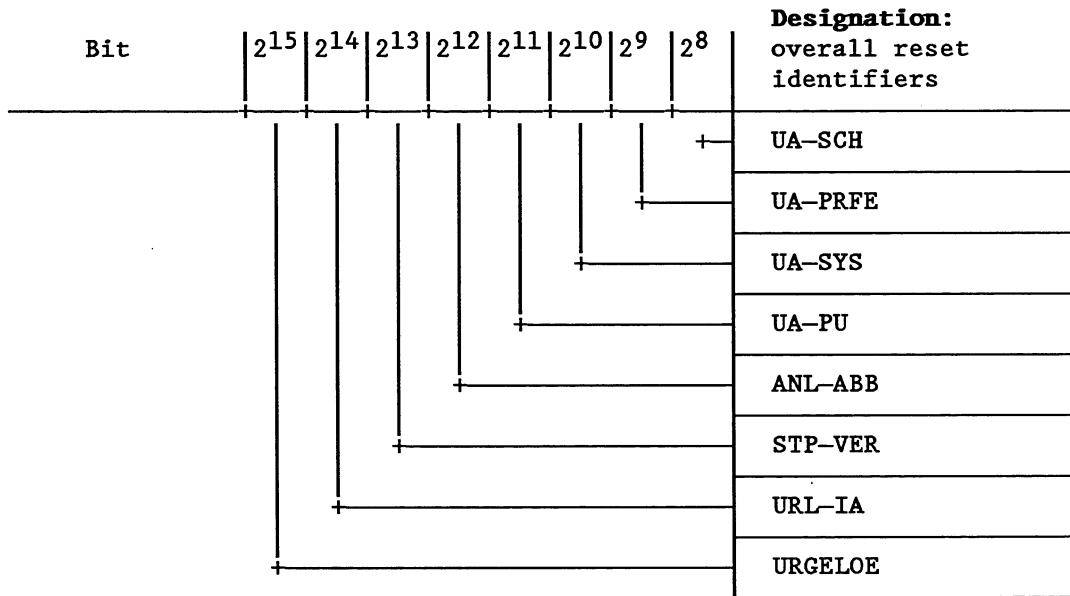
---> 3rd line control bits

System data: **RS 6** address: EA06 (LOW)



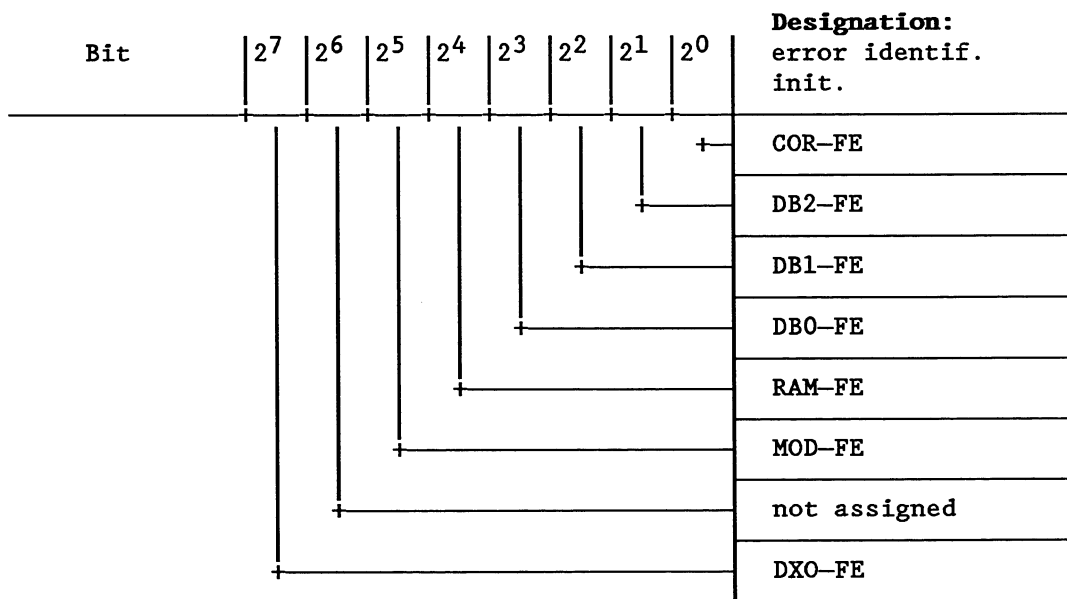
---> 4th line control bits

System data: **RS 7** address: EA07 (HIGH)



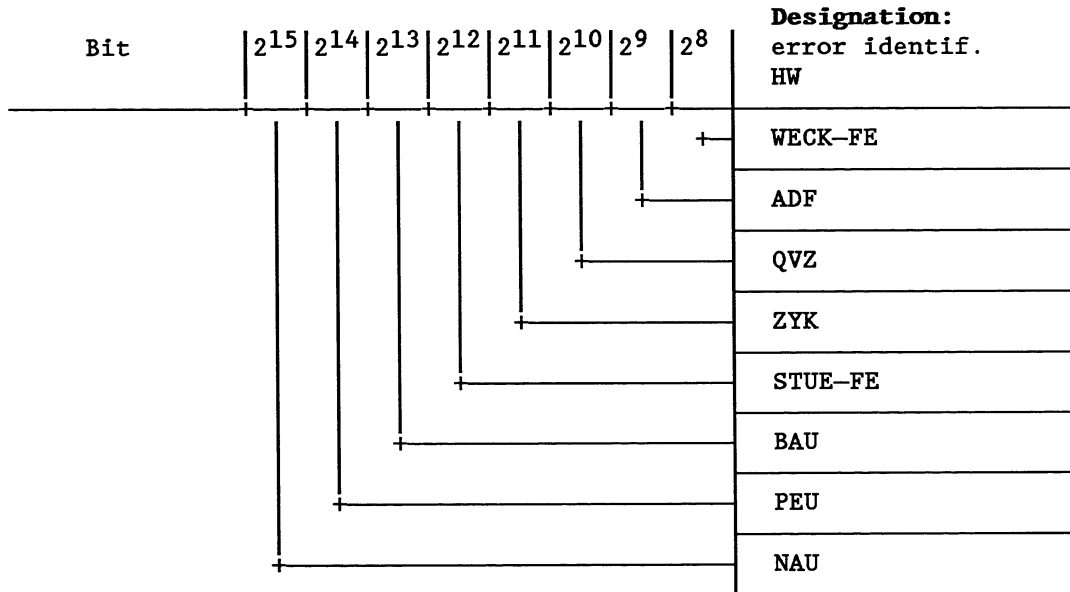
---> 5th line control bits

System data: **RS 7** address: EA07 (LOW)



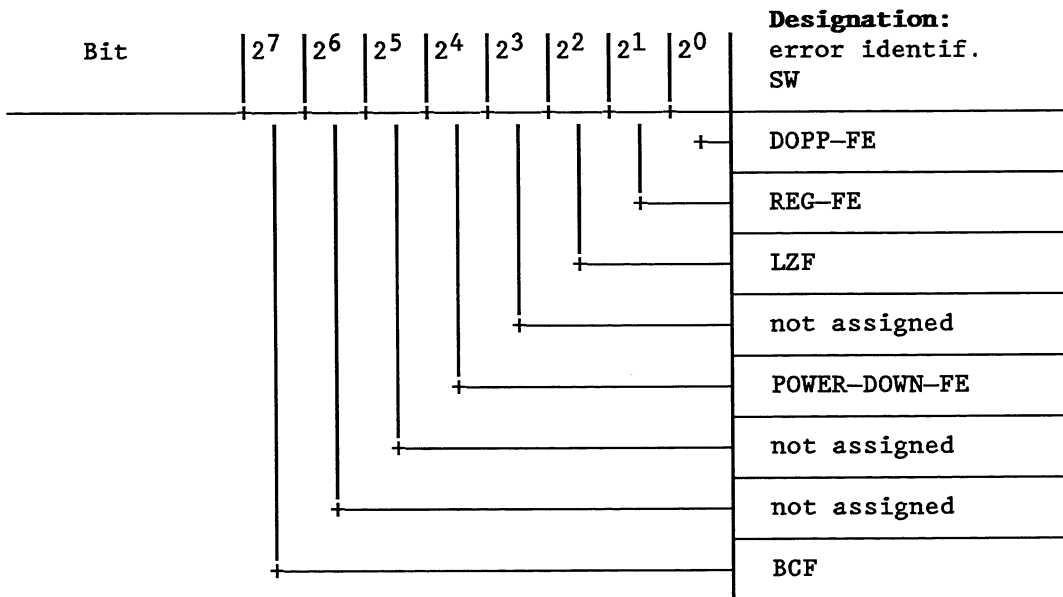
---> 6th line control bits

System data: **RS 8** address: EA08



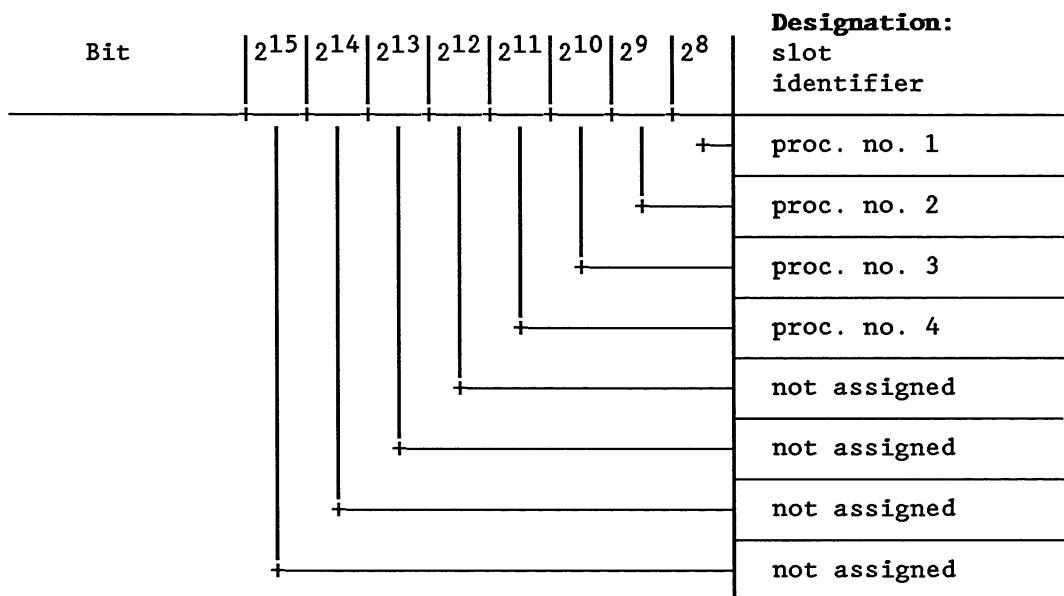
---> 7th line control bits

System data: **RS 8** address: EA08 (LOW)



---> 8th line control bits

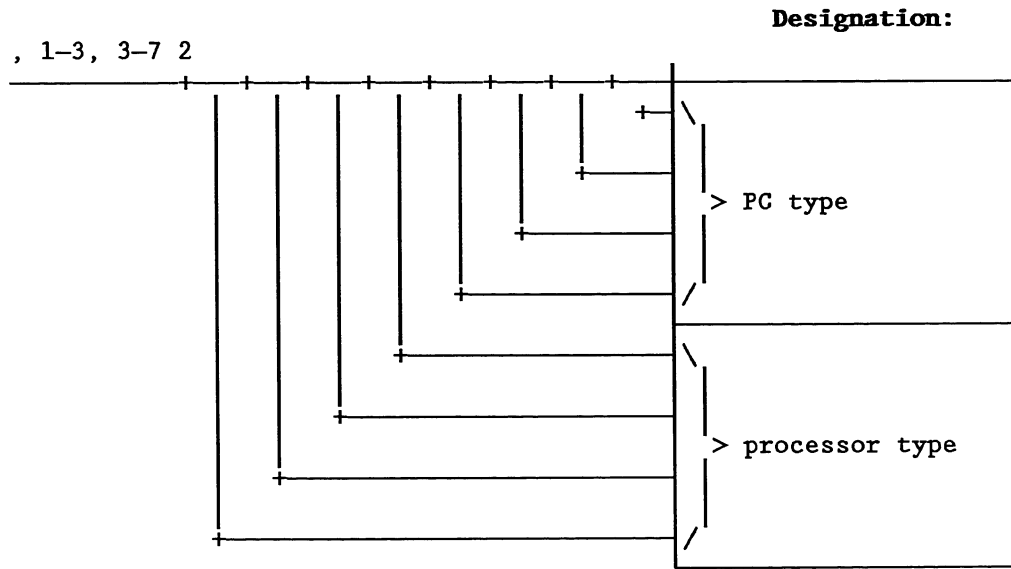
System data: **RS 29** address: EA1D (HIGH)



RS 29 (HIGH): Active interface, is used by the handling blocks and for multiprocessor communication as well as by OB 218 and the commands SED and SEE.

B8576633-01

System data: RS 29 address: EA1D (LOW)



PC type:

0 1 1 1 S5-135U

Processor type:

0 0 1 1 CPU 928

B8576633-01

System data: **RS 130** address: EA82 (LOW)

The system data RS 130 is used for displays only.

bit $2^0 = 0$: program level 'control' activated
bit $2^0 = 1$: program level 'control' suppressed

Before calling a start-up organization block (OB 20, 21 or 22), the system program evaluates the data block DB 2 (if present). Depending on the result of this evaluation, RS 130 will be set or reset by the system program. Then the system program will call a start-up OB.

If RS 130 (LOW) has been reset, controller processing in cyclic operation will be executed according to the controller list in DB 2.

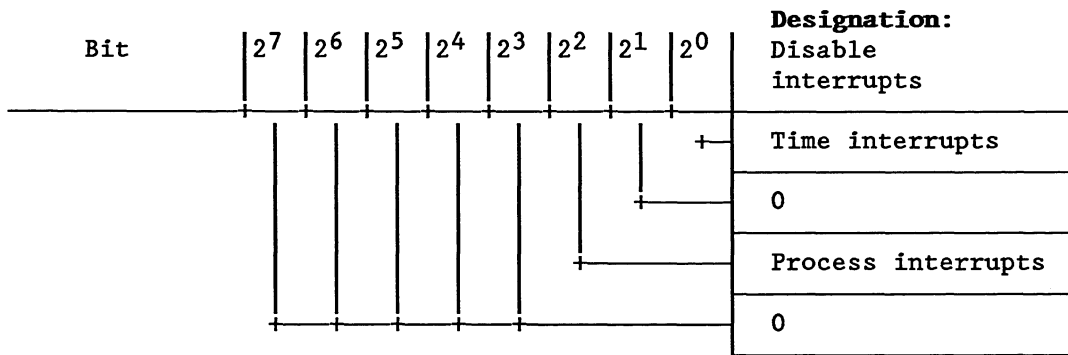
System data: **RS 131** address: EA83

The system data RS 131 is used for displays only.

For the **condition code word "Disable all interrupts"** see Subsection 6.8.1 (OB 120).

EA83 (HIGH) = 0

EA83 (LOW) :



Bit = 1 signifies: These interrupts are disabled.

B8576633-01

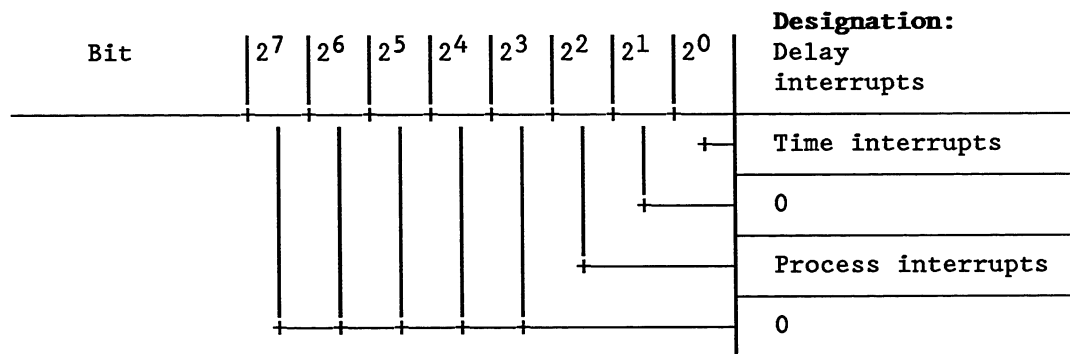
System data: **RS 132** address: EA84

The system data RS 132 is used for displays only.

For the **condition code word "Delay all interrupts"** see Subsection 6.8.1 (OB 120).

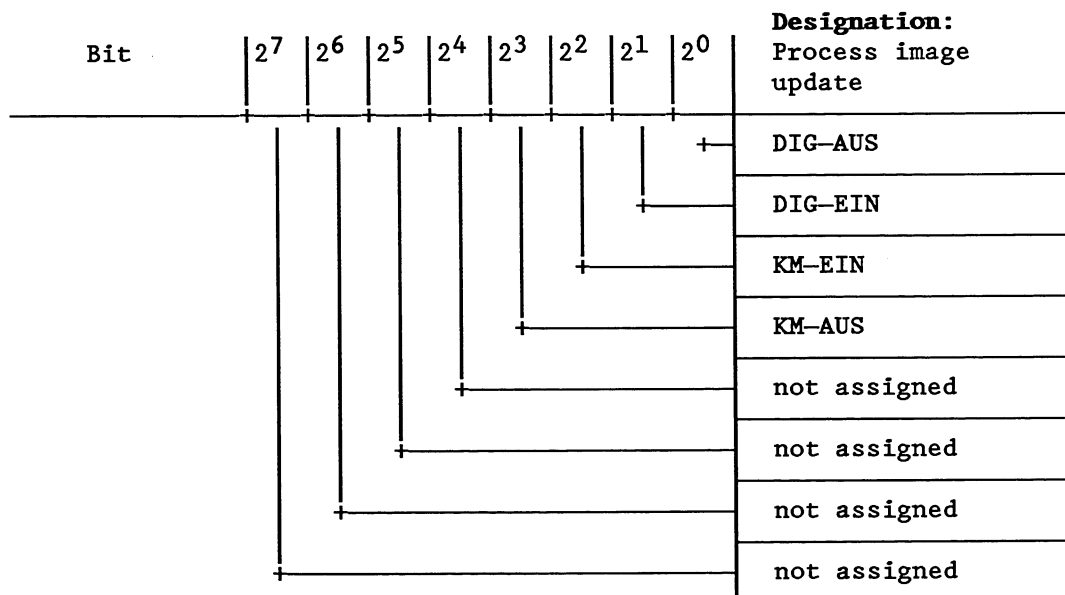
EA84 (HIGH) = 0

EA84 (LOW) :



Bit = 1 signifies: These interrupts are delayed.

System data: **RS 133** address: EA85 (LOW)



Bit 2⁰ = 0 : Output of the next process image of the digital outputs
 Bit 2⁰ = 1 : Suppression of the next process image update of the digital outputs

B8576633-01

- Bit 2¹ = 0 : Next process image of the digital inputs is read
Bit 2¹ = 1 : Suppression of the next process image update of the digital inputs
- Bit 2² = 0 : Next process image of the interprocessor communication (IPC) flag inputs is read
Bit 2² = 1 : Suppression of the next process image update of the IPC flag inputs
- Bit 2³ = 0 : Output of the next process image of the IPC flag outputs
Bit 2³ = 1 : Suppression of the next process image update of the IPC flag outputs

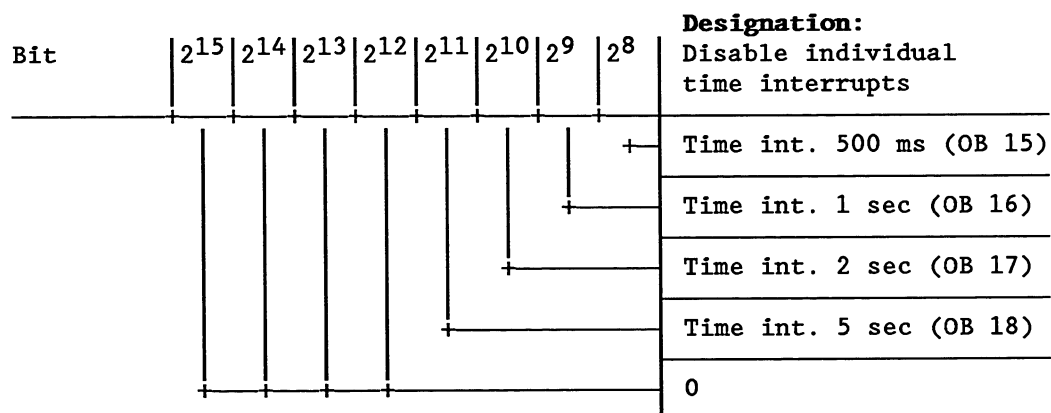
Note: Each bit, if set, prevents the update of the process image only once, afterwards it is immediately reset to '0' by the system program.

System data: **RS 135** address: EA87

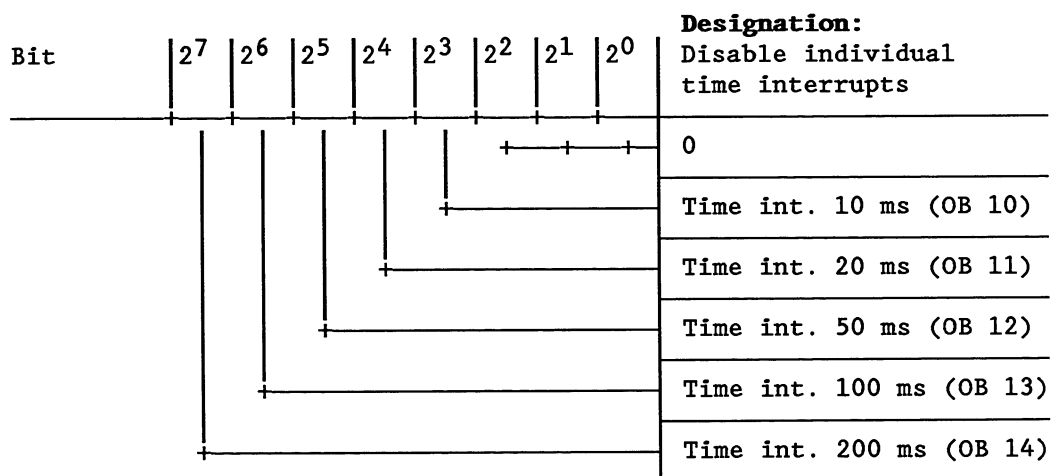
The system data RS 135 is used for displays only.

For the **condition code word "Disable individual time interrupts"** see Subsection 6.8.2 (OB 121).

EA87 (HIGH) :



EA87 (LOW) :



Bit = 1 signifies: This interrupt is disabled.

System data: **RS 137** address: EA89

The system data RS 137 is used for displays only.

For the **condition code word "Delay individual time interrupts"** see Subsection 6.8.2 (OB 123).

The structure of system data RS 137 does not differ from that of RS 135. If a bit is set to "1", processing of the time interrupt in question is delayed. If the bit is set to "0", the time interrupt is processed.

9 Memory Access Using Absolute Addresses

The STEP5 programming language contains commands that allow access to the whole memory area.

IMPORTANT!

If these commands are not used properly, STEP 5 blocks and system data may accidentally be overwritten. This can lead to unexpected and unwanted system statuses. Operations involving absolute addresses should therefore only be used by users with perfect knowledge about the system.

Local memory

The term 'local memory' is used to identify a memory area located on each processor (user submodule, DB RAM, RI/RJ/RS/RT area, counters, timers, flags, process image).

Global memory

There is only one global memory for all processors. It can be addressed via the S5 bus.

Memory structure

Memory areas can have a byte or a word structure.

Byte structure: each address points to a byte

Word structure: each address points to a word

The structure of the local memory is fixed (see Chapter 8 "Memory assignment and memory organization"). The structure of the global memory depends on the type of the modules inserted.

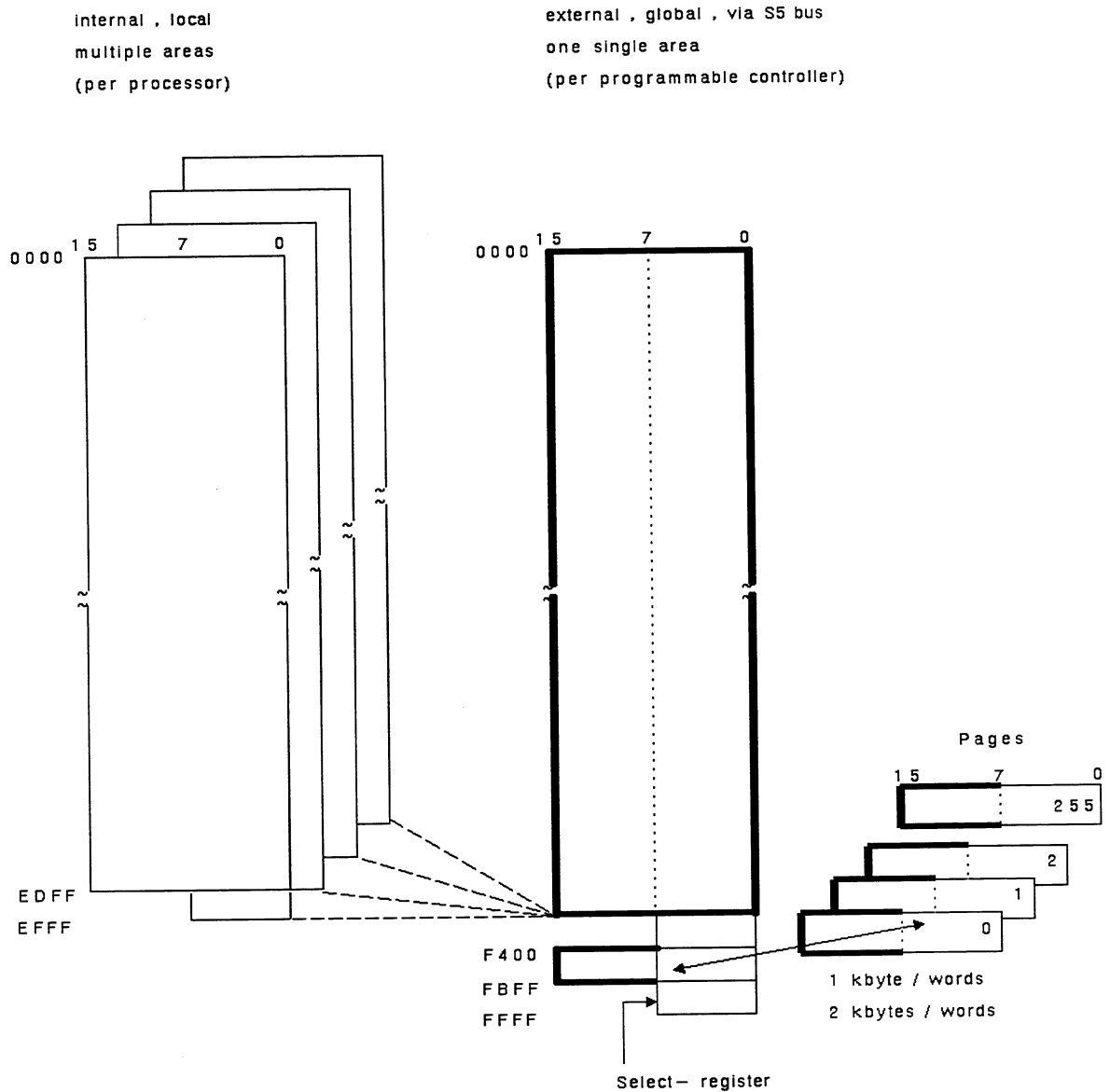


Fig. 9-1: Global memory and local memory

The following commands enable you to access local or global memory areas via *absolute addresses*.

Access to

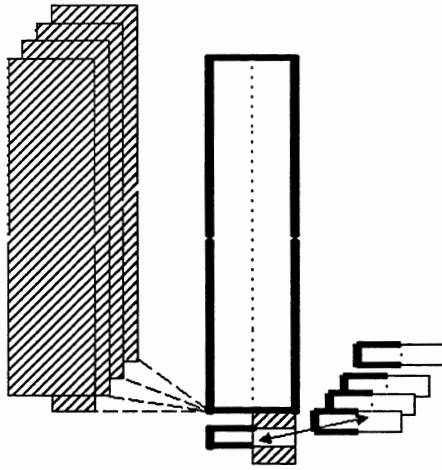
- a) the local area (0000 to EFFF) and the byte-organized section of the global area (F000 to F3FF, FC00 to FFFF):
TNB, TNW, LIR, TIR
- b) the word-organized section of the local area (0000 to EDFF):
LRW, TRW, LRD, TRD
- c) the byte-organized section of the global area (0000 to EFFF):
LB GB, LB GW, LB GD, TB GB, TB GW, TB GD, TSG

B8576633-01

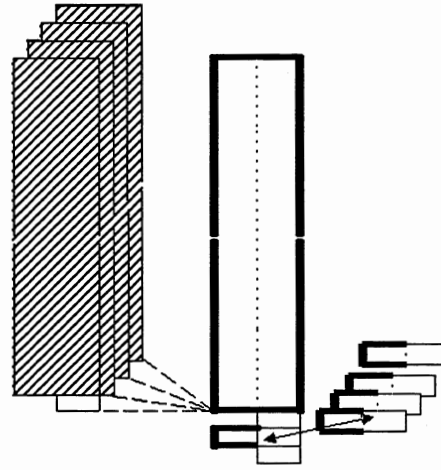
- d) the word-organized section of the global area (0000 to EDFF):
LW GW, LW GD, TW GW, TW GD, TSG
- e) the byte-organized section of the global area (F400 to FBFF, = page area):
LB CB, LB CW, LB CD, TB CB, TB CW, TB CD, TSC
- f) the word-organized section of the global area (F400 to FBFF, = page area):
LW CW, LW CD, TW CW, TW CD, TSC

Access to local/global memory areas via absolute addresses

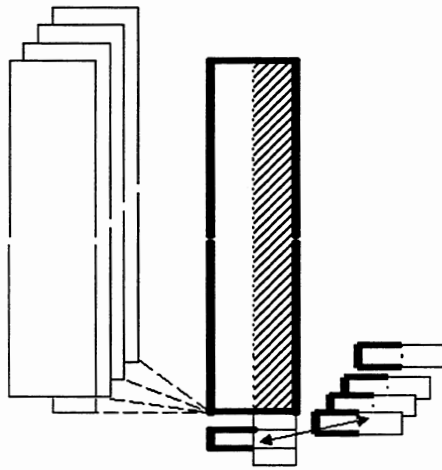
□ access is not possible ▨ access is possible



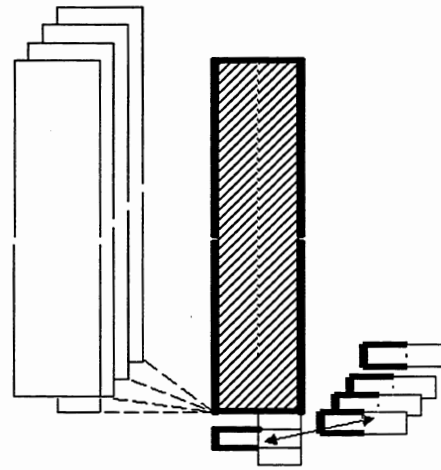
a) LIR, TIR, TNB, TNW



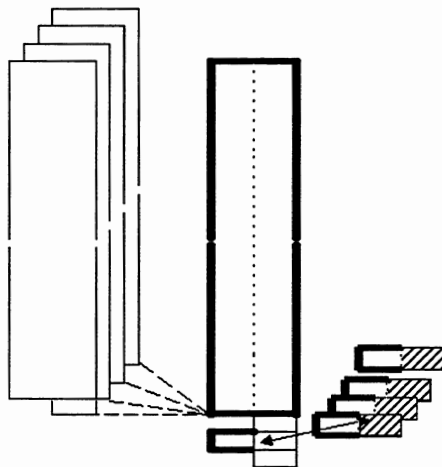
b) LRW, TRW, LRD, TRD



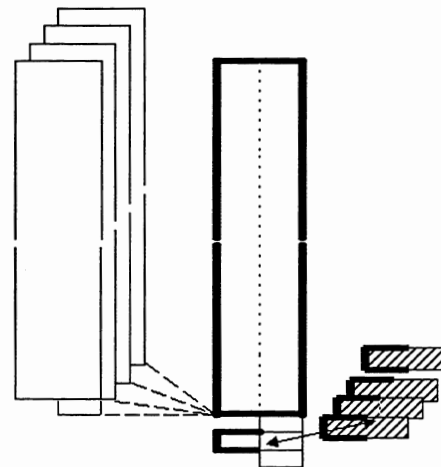
c) LB GB, LB GW, LB GD
TB GB, TB GW, TB GD, TSG



d) LW GW, LW GD
TW GW, TW GD, TSG



e) LB CB, LB CW, LB CD
TB CB, TB CW, TB CD, TSC



f) LW CW, LW CD,
TW CW, TW CD, TSC

9.1 Access to Registers and the Memory via an Address in Accu 1

The registers are special memory locations which the processor requires for processing the STEP5 program. Each register has a width of 16 bits. Access to the contents of the registers is possible by means of the system functions LIR (load indirectly register) and TIR (transfer indirectly register).

LIR 0...15 Loads the contents of the memory location which has been addressed via accu 1-L into register 0...15

TIR 0...15 Transfers the contents of register 0...15 into the memory location which has been addressed via accu 1-L

The memory location is situated either in the local memory area (0000 through EFFF) or in the byte-organized section of the global area (F000 through F3FF, FC00 through FFFF).

Access to the page area with LIR and TIR

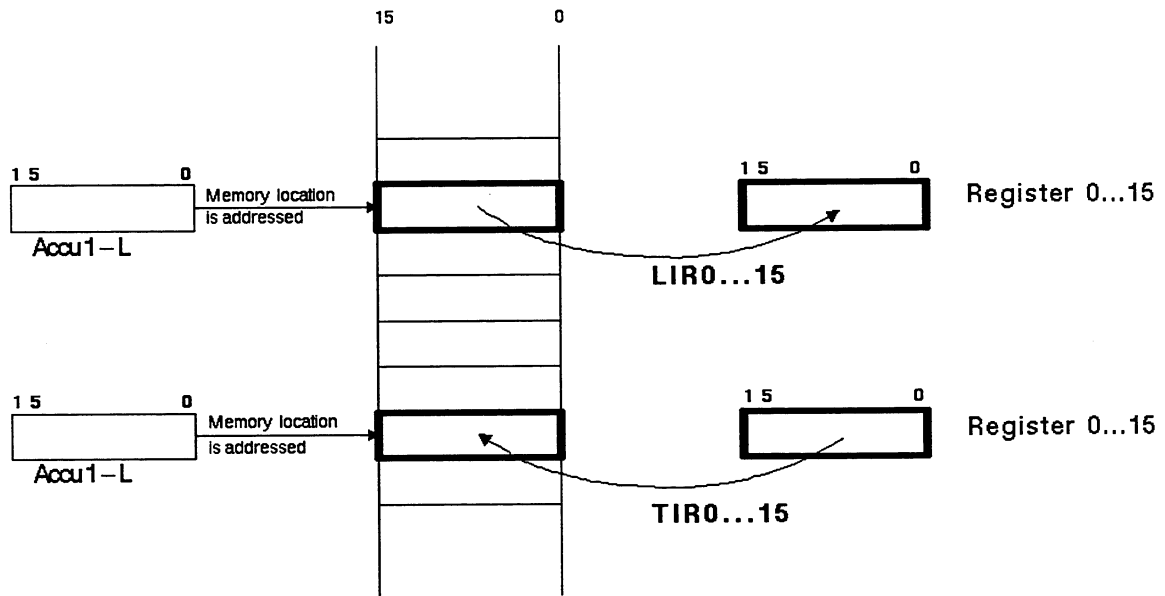
The commands LIR and TIR should not be used to access the page area (F400 - FBFF) in the multiprocessor programmable controller S5-135U. Use the commands from Subsection 9.3.5 "Access to the page frame" or the special functions from Chapter 6.6 "Page access".

The following table shows the register assignment for the CPU 928. It differs from the register assignment of the S or R processor!

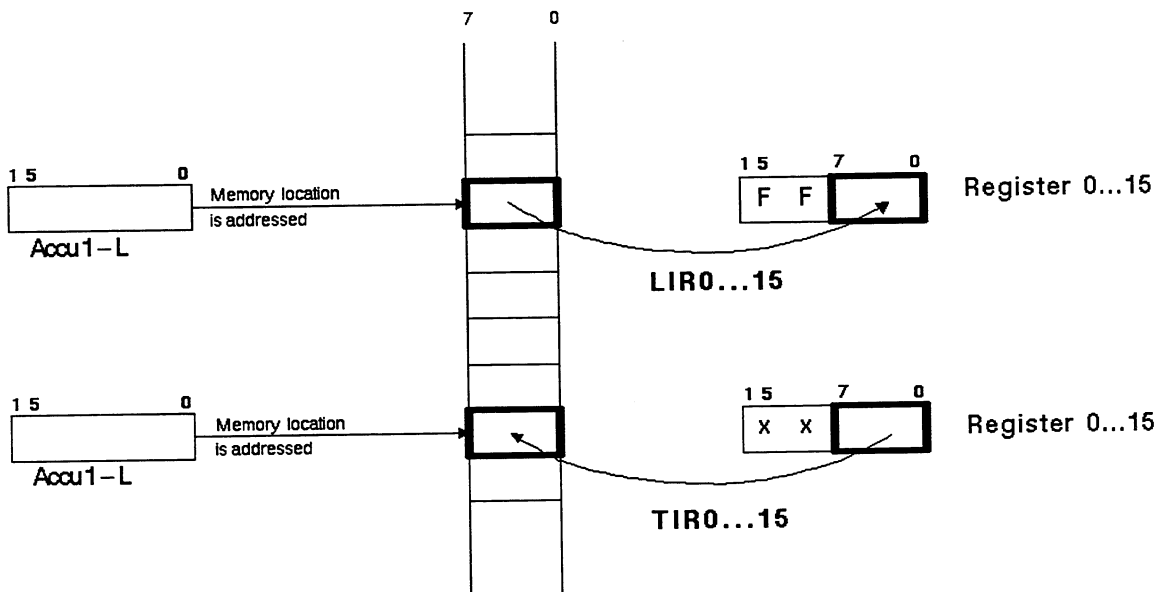
Register no.	Register assignment
register 0:	accu 1-H (left-hand word of accu 1, bits 16 - 31)
register 1:	accu 1-L (right-hand word of accu 1, bits 0 - 15)
register 2:	accu 2-H
register 3:	accu 2-L
register 6:	DBA (data block start address)
register 8:	DBL (data block length)
register 9:	accu 3-H
register 10:	accu 3-L
register 11:	accu 4-H
register 12:	accu 4-L
register 15:	SAC (STEP address counter)

Registers 4, 5, 7, 13, and 14 do not exist. LIR/TIR accesses to these register numbers are treated as 'null operations' (NOP).

LIR and TIR access to 16-bit memory areas



LIR and TIR access to 8-bit memory areas



If memory areas with a width of only 8 bits are accessed by means of LIR/TIR (for memory addresses \geq EE00H) do not forget that

- only the low byte of the register is transferred by means of TIR (the high byte of the register is lost) and
- FFH will be written into the high byte of the register if LIR is used.

● **Registers 0 to 3 and 9 to 12: accus 1, 2, 3, and 4**

The accumulators are used as buffers by the processor during program execution. By using the TIR/LIR instructions you can transfer the contents of the accumulators to memory locations addressed absolutely or load the contents of memory locations addressed absolutely into the accumulators. The absolute address of the memory location is entered in accumulator 1-L.

Example:

The contents of the memory location with address A000 are loaded into flag word FW 100.

```
:L  KHA000      load address A000 of memory location into accu 1
:LIR 1          load contents of memory location addressed by
:              accu 1 into register 1 = accu 1
:T  FW100      write contents of address A000 in FW 100
:BE
```

Example:

The contents of flag word 200 are transferred to the memory location with address A000.

```
:L  FW200      load flag word FW 200 into the accu 1
:L  KHA000      load address A000 to which the transfer is to be
:              made into accu 1 (flag word 200 to accu 2)
:TIR 3         transfer contents of register 3 = accu 2 to the
:              memory location addressed by accu 1
:BE
```

● **Register 6: DBA (data block start address)**

If a data block is called by means of the C DB and CX DX commands register 6 will be loaded with the address of the DW0 in the data block called. (This address is contained in the block address list in DB 0.)

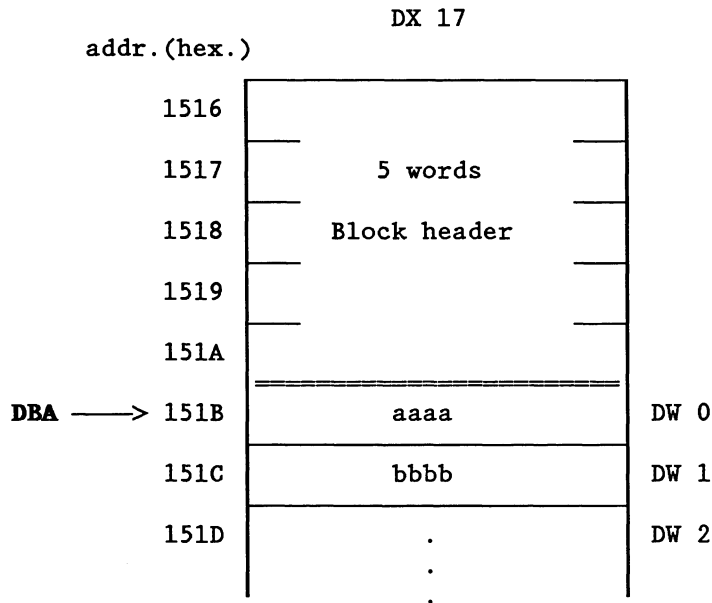
The DBA register will be retained if

- program execution is continued in another block by means of a jump instruction (JU/JC) or
- another program level is nested.

It will be altered if

- another data block is called or
- a jump back to a primary block is carried out after a new data block has been called from the block already called (also refer to Subsection 2.4.3, "Validity range of data blocks").

Example: CX DX17



The address of the memory word in which DW 0 has been stored is entered in the DBA register if DX 17 is called; in our example: DBA = 151B (hex.).

Note: The address entered in the DBA register is located in the ISTACK under 'DB-ADR'.

Access to data words is normally carried out by means of the STEP5 instructions L/T DW, L/T DR, L/T DL, L/T DD, A/O/AN/ON/=/S/R Dx.y. However, these are permitted only up to data word DW 255. By manipulating the DBA register you may also access data words > 255 by means of these commands.

Example:

Data word DW 300 of data block DB 100 is loaded by altering register 6.

FB 7

name: LIR/TIR6

```

:L   RS34           (start address of DB address list) + 100 =
:ADD BN+100         entry of DB 100 in address list
:LIR 1              start address of DB 100 (DW0) into accu 1
:ADD KF+200         enter address of DW 200 in DB 100 in
:T   RS62           system data word RS 62
:L   RS20           load base address of system data
:ADD KF+62         load address of RS 62 into accu 1
:LIR 6              load DBA register with contents of address of RS62
:                   i.e.
:                   the data block start is set to DW 200
:L   DW100          load DW (200 + 100) = DW 300
:T   FW100          deposit DW 300 in flag word FW100
:BE
    
```

Caution:

If you alter the DBA register as described above the DBL register (see below) will not be altered. This means that transfer error monitoring is no longer guaranteed!

By using the **special function OB 180** "variable data block access" you can also shift the DBA register by a preselected number of data words. Since OB 180 alters the DBL register at the same time transfer errors will still be monitored.

Example:

```
FB7
name :OB180
      :C  DB100          load DBA and DBL registers with the values of
      :L  KF200          DB 100 and, by means of OB180,
      :JU OB180          increase the DBA register by 200 and
      :                   reduce the DBL register by 200
      :JC =ERR          error output if DB 100 contains less than
      :                   200 data words
      :L  DW100          load DW 300 and
      :T  FW100          enter in FW 100
      :BEU
ERR  :                   program section for error handling
      :
      :BE
```

● **Register 8: DBL = data block length**

In addition to the DBA register the DBL register is loaded each time a data block is called. It contains the length (in words) of the data block called without the header.

Prior to every call of OBl or FB0, the DBL register is set to '0'.

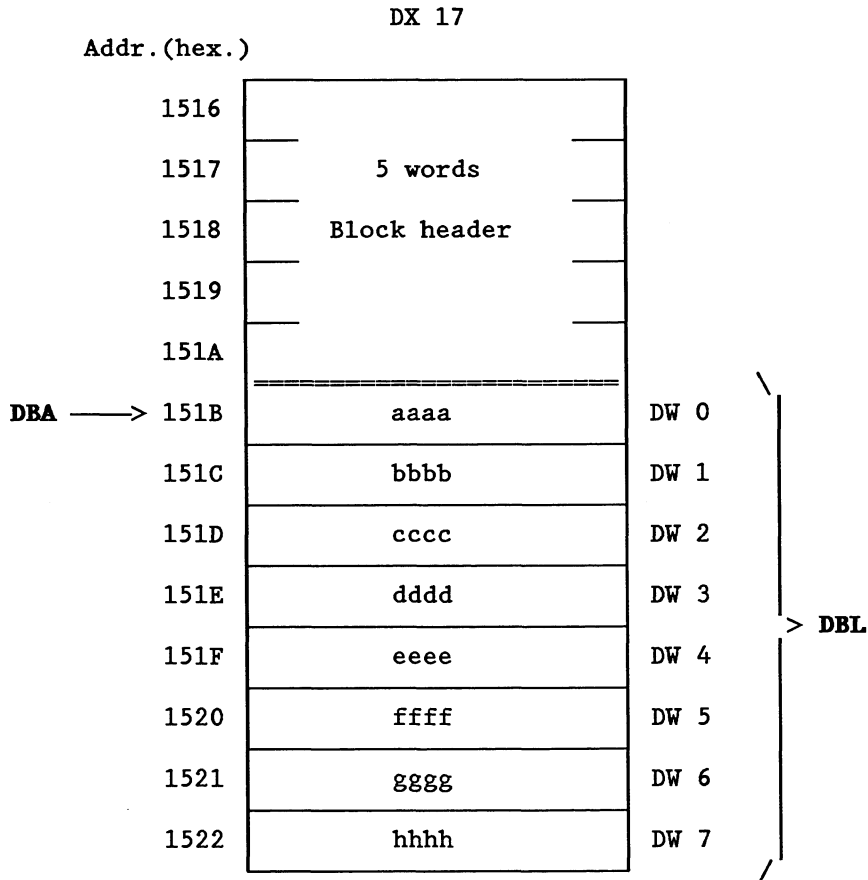
The DBL register will be retained if

- program execution is continued in another block by means of a jump instruction (JU/JC) or
- another program level is nested.

It will be altered if

- another data block is called or
- a jump back to a primary block is carried out after a new data block has been called from the block already called (also refer to Subsection 2.4.3, "Validity range of data blocks").

Example: CX DX17



The number of existing data words is entered in the DBL register if DX 17 is called, for our example: DBL = 8 (DW 0 to DW 7).

Note: The number entered in the DBL register is contained in the ISTACK under 'DBL-REG'.

● **Register 15: SAC = step address counter**

The absolute address of the instruction to be processed next in the program memory is entered in register 15 during STEP5 program execution.

Example: A constant is written into all data words of a data block.

The program represented below writes the constant KH A5A5 into all data words of DB 50. After the STEP5 commands in bold-face type are altered it may also be used to write any values required into other data blocks (DB or DX). Non-existent data blocks are identified and will cause a jump to the NIVO marker.

The start address (DBA) and length (DBL) of the data block are determined by means of the special function OB 181 'test data block (DB/DX)'.

The program makes use of all four accumulators. The figure shows the assignment of the accumulators during program execution up to the LOOP marker. Assignment of the accumulator is not altered within the loop. Initially, accu 1 contains the address of the last data word (DBA + DBL - 1) and is reduced by one during every loop cycle. Accu 2 contains the address of the first data word (DBA). The loop is interrupted as soon as the contents of accu 1 are less than the contents of accu 2.

The TIR 10 instruction is used for writing into data words. This instruction stores the contents of accu 3-L (the constant) under the address stated in accu 1-L.

```

      .
      .
      .
      :L      KHA5A5      constant to be written into all data words
      :
      :L      KY1,50      type and number of the data block
      :ENT
      :JU      OB181      special function OB 'test data blocks'
      :JC      =NIVO      abort if DB50 does not exist
      :TAK
      :ENT
      :+F
      :
      :          accu1 := address of last data word + 1
      :          accu2 := address of first data word
      :          accu3 := constant
      :
      LOOP :ADD      BN-1      overwrite all data words, beginning with last
      :>F          data word, using the constant contained in
      :JC          =CONT      accu 3-L
      :TIR      10
      :JU      =LOOP
      :
      :          program continued
      CONT : .      ... after all data words have been overwritten
      .
      :BEU
      NIVO : .      ... if DB50 does not exist or if it contains
      .          0 data words
      :BE
  
```

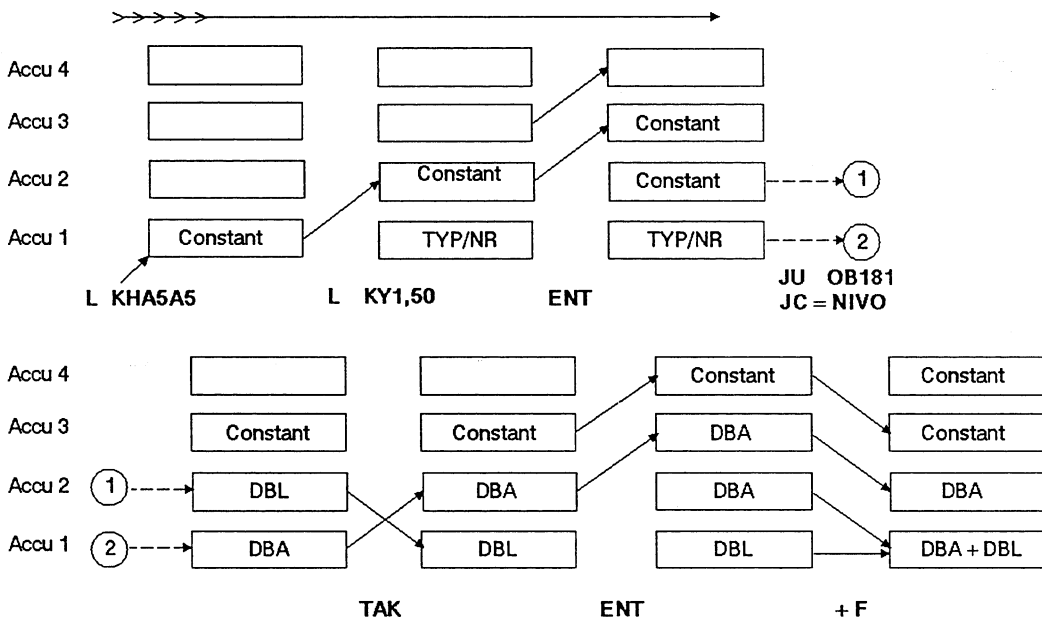


Fig. 9-2: Contents of the accumulators during program execution

Note:

The program section beginning at the LOOP marker can be used to overwrite any memory area (e.g.: flags, timers, counters) with a constant.

Example 9-3: Clearing of all flag bytes (FY0 to FY255)

```

:L   KB0          constant to be written into all flag bytes
:
:L   RS14         base address of flag area (= address of first
:                flag byte FY0)
:ENT
:L   KF256        + length of flag area
:ENT              = (address of last flag byte FY255) + 1
:+F
LOOP :ADD BN-1    all 256 flag bytes, starting with flag byte
:TIR 10          FY255 have the constant contained in accu 3-LL
:JU  =LOOP       written into them
:
:
:

```


9.2 Transfer of Memory Blocks

Using the system operations TNB and TNW you can transfer memory blocks (up to 255 bytes by means of TNB, up to 255 words by means of TNW).

Operation	Parameters	Description
TNB	0 to 255	Transfer memory block (1 to 255 bytes)
TNW	0 to 255	Transfer memory block (1 to 255 words)

By means of the commands TNB and TNW, you can access the local memory area as well as the byte-organized section of the global memory (F000 to F3FF, FC00 to FFFF).

Access to the page area with TNB and TNW

The commands TNB and TNW should not be used to access the page area (F400 - FBFF) in the multiprocessor programmable controller S5-135U. Instead, use the commands from Subsection 9.3.5 "Access to the page frame" or the special functions from Chapter 6.6 "Page access".

The parameter with TNW/TNB states the length (number of words/number of bytes) of the area to be transferred. Before this operation is carried out, the end address of the source area must be loaded in accu 2 and the end address of the destination area in accu 1. This means that the addresses stated are always the upper (higher) addresses of source and destination area. The transfer itself for the CPU 928 is 'decremental', i.e. the transfer is executed starting with the highest address of the source area and is completed with the lowest address.

Source and destination area must be completely located in one memory area and must not overlap. The following memory areas are differentiated by means of area limits:

Addresses (hexadecimal)

1. 0000 to 1FFF user memory (16 bit) 8K words
 0000 to 3FFF user memory (16 bit) 16K words
 0000 to 7FFF user memory (16 bit) 32K words
2. 8000 to DD7F DB-RAM (16 bit)
3. DD80 to EFFF system RAM (16 bit: DB0, RI/RS,
 timers, counters, etc.)
4. EE00 to EFFF RAM (8 bit: flags, process image)
5. F000 to FFFF interface system (8 bit)

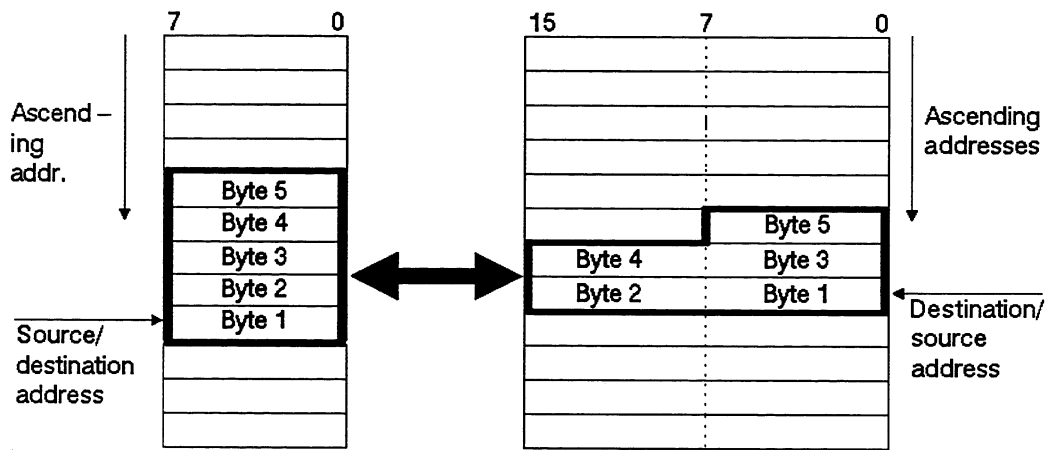
(see Chapter 8 "Memory assignment")

Pseudo command boundaries at TNB and TNW

The TNB and TNW instructions are long running STEP5 commands which contain so-called 'pseudo command boundaries': i.e.: data transfer is carried out in sub-units of different sizes, depending on the source and destination area. If an error (e.g. ZYK) or an interruption (e.g. due to a time or process interrupt) occurs during the transfer of a sub-unit, the corresponding organization block will be nested at the end of this sub-unit at the pseudo command boundary. Process interrupt OBs and time interrupt OBs can only be called if the setting "interruptibility at command boundaries" has been selected in DX 0.

Exception: If acknowledgement delays and/or addressing errors occur during the transfer, the complete data field will be transferred first and the organization block programmed for this particular event will be called once at the command boundary (only the QVZ-OB if QVZ and ADF occur simultaneously). The lowest address of the field will always be specified as the QVZ error address. Irrespective of this, OB 2, OB 10 through Ob 18 or an error organization block may be nested at the pseudo command boundaries.

TNB and TNW between 8 and 16 bit memory areas



Transfer of bytes 1 to 5:

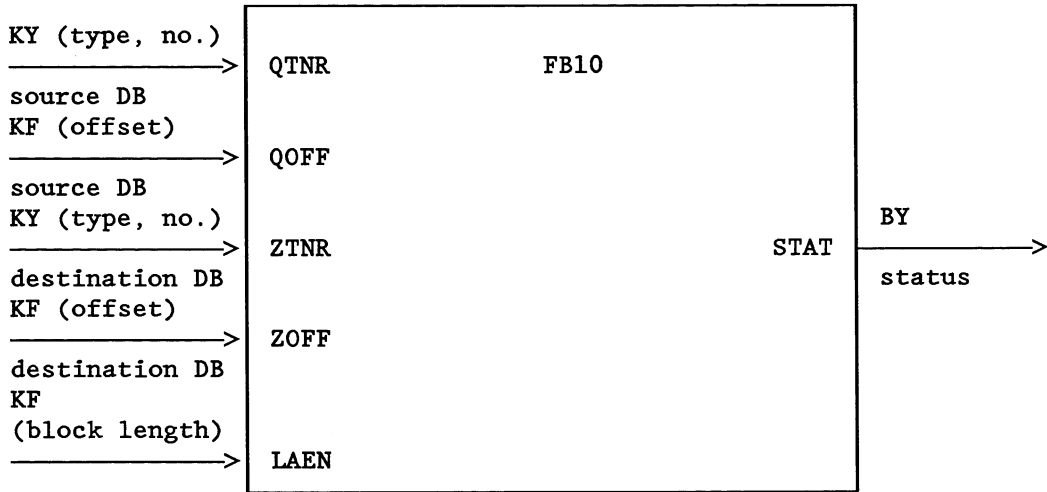
L <source address>
L <destination address>
TNB 5

Transfer of bytes 1 to 4:

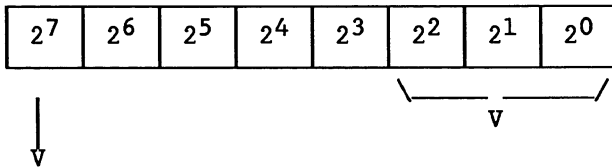
L <source address>
L <destination address>
TNW 2

Example:

Copying a field of 4095 data words max. from one DB or DX data block to another DB or DX data block. The start of the field within the source and destination data block is determined in each case by one offset value between 0 and 4095.



The input parameters are checked before copying is carried out. If an error is recognized bit 2^7 will be set to 1 in the output parameter STAT and the type of error will be stated in bits 2^0 to 2^2 :



0 = no error
1 = error

type of error:
 1 = source DB = destination DB
 2 = offset or length > 4095
 3 = source DB non-existent or illegal
 4 = source DB too short
 5 = destination DB non-existent or illegal
 6 = destination DB in the read only memory (EPROM module)
 7 = destination DB too short

FB 10 is divided into five program sections with the following functions:

1. Input parameters
 - Test whether source and destination data block have the same type and number or not.
 - Test whether input parameters 'source offset', destination offset', and 'block length' are less than 4096.

2. Source data block

- Test whether source data block exists and is long enough.
- Calculate the absolute address of the last data word in the destination block.

3. Destination data block

- Test whether destination data block exists and is long enough and whether located in the RAM submodules.
- Calculate the absolute address of the last data word in the destination block.

4. Transfer

- Copy by means of TNW instruction.
Fields containing more than 255 words are transferred in sub-units of 128 words each (instruction TNW 128).
Any remainders are transferred by an additional TNW instruction.

5. Condition code

- Supply of the output parameter 'status' corresponding to the result of the tests.

Memory locations occupied:

FW 242 end address of data destination
 FW 244 end address of data source
 FW 246 field length
 FW 248 offset in destination data block
 FW 250 type and number of destination data block
 FW 252 offset in source data block
 FW 254 type and number of source data block
 RS 60 sub-unit counter

Programming function block FB 10

Note: If copying is to be started beginning with data word DW 0 the program sections printed in italics are not valid. No offset value is stated.

FB10

SEGMENT 1

NAME :DB-DB-TR		DATA BLOCK-DATA BLOCK-TRANSFER
DECL :QTNR	I/Q/D/B/T/C: D	KM/KH/KY/KS/KF/KT/KC/KG: KY
DECL :QOFF	<i>I/Q/D/B/T/C: D</i>	<i>KM/KH/KY/KS/KF/KT/KC/KG: KF</i>
DECL :ZTNR	I/Q/D/B/T/C: D	KM/KH/KY/KS/KF/KT/KC/KG: KY
DECL :ZOFF	<i>I/Q/D/B/T/C: D</i>	<i>KM/KH/KY/KS/KF/KT/KC/KG: KF</i>
DECL :LAEN	I/Q/D/B/T/C: D	KM/KH/KY/KS/KF/KT/KC/KG: KF
DECL :STAT	I/Q/D/B/T/C: A	BI/BY/W/D: BY

```

:
: LW =QTNR      START INPUT PARAMETERS
: T  FW254     TYPE (DB/DX) AND NUMBER OF
: LW =ZTNR     SOURCE DATA BLOCK
: T  FW250     TYPE (DB/DX) AND NUMBER OF
: !=F         DESTINATION DATA BLOCK
: JC =F001     SOURCE DB = DESTINATION DB ?
:             JUMP, IF YES
:
: LW =QOFF     OFFSET IN SOURCE
: T  FW252     DATA BLOCK
: LW =ZOFF     OFFSET IN DESTINATION
: T  FW248     DATA BLOCK
: OW
: LW =LAEN     LENGTH (NUMBER OF DATA WORDS) OF
: T  FW246     BLOCK TO BE TRANSFERRED
: OW          (BLOCK LENGTH)
: L  KHf000    SOURCE OFFSET, DESTINATION OFFSET OR
: AW          LENGTH >= 4096 ?
: JP =F002     JUMP, IF YES
:             END INPUT PARAMETERS
:
:             START SOURCE DATA BLOCK
: L  FW254     TYPE AND NUMBER OF SOURCE DATA BLOCK
: JU OB181     TEST DATA BLOCK
: JC =F003     JUMP, IF BLOCK TEST NEGATIVE
: TAK         A1: NUMBER OF DW, A2: ADDRESS
: ENT         A3: ADDRESS
: L  FW252     OFFSET IN SOURCE DATA BLOCK
: ENT         A3: NUMBER OF DW, A4: ADDRESS
: L  FW246     BLOCK LENGTH
: +F         OFFSET + BLOCK LENGTH
: <F         NUMBER OF DW'S < OFFSET + BLOCK LENGTH ?
: JC =F004     JUMP, IF YES
: L  KB1       A2: OFFSET + BLOCK LENGTH, A3: ADDRESS
: -F         OFFSET + BLOCK LENGTH - 1
: +F         OFFSET + BLOCK LENGTH - 1 + ADDRESS
: T  FW244     END ADDRESS OF DATA SOURCE
:             END SOURCE DATA BLOCK
:
:             START DESTINATION DATA BLOCK
: L  FW250     TYPE AND NUMBER OF DESTINATION DATA BLOCK
: JU OB181     TEST DATA BLOCK
: JC =F005     JUMP, IF BLOCK TEST NEGATIVE
: JM =F006     JUMP, IF BLOCK IN EPROM
: TAK         A1: NUMBER OF DW, A2: ADDRESS
: ENT         A3: ADDRESS
: L  FW248     OFFSET IN DESTINATION DATA BLOCK
: ENT         A3: NUMBER OF DW, A4: ADDRESS
: L  FW246     BLOCK LENGTH
: +F         OFFSET + BLOCK LENGTH
: <F         NUMBER OF DW'S < OFFSET + BLOCK LENGTH
: JC F007     JUMP, IF YES
: L  KB1       A2: OFFSET + BLOCK LENGTH, A3: ADDRESS
: -F         OFFSET + BLOCK LENGTH -1
: +F         OFFSET + BLOCK LENGTH -1 + ADDRESS
: T  FW242     END ADDRESS OF DATA DESTINATION
:             END DESTINATION DATA BLOCK
:
:

```

```

:          START TRANSFER
:L   KBO   COMPARISON VALUE
:L   FY246 BLOCK LENGTH, HIGH BYTE
:!=F     BLOCK LENGTH >= 256 WORDS ?
:SLW 1    MULTIPLY WITH 2, NUMBER OF SUB-
:T   RS60     UNITS WITH 128 WORDS EACH
:L   FW244   END ADDRESS OF DATA SOURCE
:L   FW242   END ADDRESS OF DATA DESTINATION
:JC  =REST   JUMP, IF FIELD LENGTH < 256 WORDS
LOOP :TNW 128 TRANSFER OF A SUB-UNIT
      :ADD KF-128 REDUCE SOURCE END ADDRESS BY LENGTH OF
      :TAK           SUB-UNIT
      :ADD KF-128 REDUCE DESTINATION END ADDRESS BY LENGTH OF
      :TAK           SUB-UNIT
      :JU  OB160   COUNTING LOOP
      :JC  =LOOP   JUMP, IF NOT ALL
:          SUB-UNIT TRANSFERRED
REST :B   FW246  BLOCK LENGTH, LOW BYTE
      :TNW 0    TRANSFER REST OF FIELD
:          END OF TRANSFER
:
:          START OF CONDITION CODE
:L   KBO   IDENTIFIER 00 (HEX.): NO ERROR
ENDE :T   =STAT OUTPUT PARAMETER STATUS/ERROR
      :BEU
F001 :L   KB129  ERROR IDENTIFIER 81 (HEX.):
      :JU  =END   SOURCE DB = DESTINATION DB
F002 :L   KB130  ERROR IDENTIFIER 82 (HEX.):
      :JU  =END   OFFSET OR LENGTH >= 4096
F003 :L   KB131  ERROR IDENTIFIER 83 (HEX.):
      :JU  =END   SOURCE DB ILLEGAL
F004 :L   KB132  ERROR IDENTIFIER 84 (HEX.):
      :JU  =END   SOURCE DB TOO SHORT
F005 :L   KB133  ERROR IDENTIFIER 85 (HEX.):
      :JU  =END   DESTINATION DB ILLEGAL
F006 :L   KB134  ERROR IDENTIFIER 86 (HEX.):
      :JU  =END   DESTINATION DB IN THE READ/ONLY MEMORY
F007 :L   KB135  ERROR IDENTIFIER 87 (HEX.):
      :JU  =END   DESTINATION DB TOO SHORT
:          END OF CONDITION CODE
:BE

```

9.3 BR Register Operations

The BR register (base address register, 32 bits) is used by the load and transfer commands (described in Subsection 9.3.3 ff). A memory location is accessed if its absolute address equals the total of the the BR register contents and a constant:

$$\text{Absolute address} = \text{BR register contents} + \text{constant}$$

The BR register will be retained if

- program execution is continued in another block by means of a jump instruction (JU/JC) or
- another program level is nested.

The BR register is set to '0' before a program level is called.

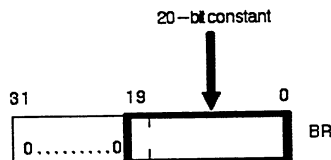
9.3.1 Loading of the BR Register

To load or alter the contents of the BR register use the following commands:

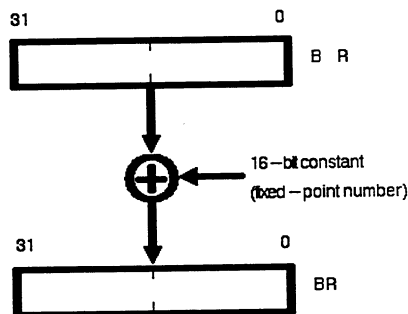
Operation	Parameters	Description
MBR	0 to FFFFF	Load a 20-bit constant into the BR register ¹⁾
ABR	-32768 to +32767	Add a 16-bit constant to the contents of the BR register

1) The bits 2²⁰ through 2³¹ are set to '0'.

MBR 0 through FFFFF



ABR -32767 through +32767



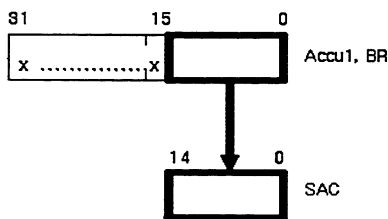
9.3.2 Shifting of the BR Register Contents

For a more flexible use of all the registers new commands have been introduced. They can be used to alter the contents of individual registers or to transfer them into other registers.

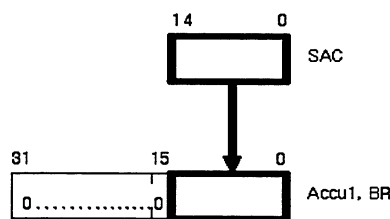
Operation	Parameters	Description
MAS	-	Transfer the contents of accu 1 (bit 2^0 to 2^{14}) into the SAC register (STEP address counter)
MBS	-	Transfer the contents of the BR register (bit 2^0 to 2^{14} , base address register) into the SAC register
MSA	-	Transfer the contents of the SAC register (STEP address counter) into accu 1 ¹⁾
MSB	-	Transfer the contents of the SAC register (STEP address counter) into the BR register (base address register) ¹⁾
MAB	-	Transfer the contents of accu 1 (bit 2^0 to 2^{31}) into the BR register (base address register)
MBA	-	Transfer the contents of the BR register (base address register) into accu 1

1) The bits 2^{15} through 2^{31} are set to '0'.

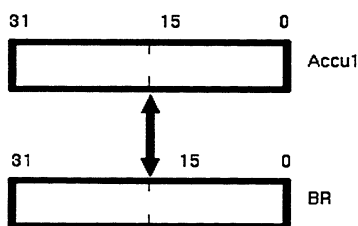
MAS, MBS



MSA, MSB



MAB, MBA



9.3.3 Access to the Local Memory

The following commands enable you to access the local, word-organized memory via absolute addresses. The absolute address equals the total of the BR register contents and the 16-bit constant contained in the command (-32768 through +32767).

Operation	Parameters	Description
LRW	-32768 to +32767	Load the word that has been addressed by means of BR register + constant into accu 1-L
LRD	-32768 to +32767	Load the double word that has been addressed by means of BR register + constant into accu 1
TRW	-32768 to +32767	Transfer the contents of accu 1-L into the word that has been addressed by means of BR register + constant
TRD	-32768 to +32767	Transfer the contents of accu 1 into the double word that has been addressed by means of BR register + constant

The absolute address must be within 0 and EDFFH (with LRW, TRW) or 0 and EDFEH (with LRD, TRD). Otherwise, the processor will recognize an execution time error and call the **OB 31**. Accu 1 contains error identifiers which describe the error occurred in more detail (see Subsection 5.6.2 "Other execution time errors").

9.3.4 Access to the Global Memory

Use the following commands to access the global *byte or word*-organized memory via an absolute address.

Testing and setting of a busy-condition location in the global area

The access of individual processors to common memory areas can be controlled via a busy-condition location. Each common memory area is assigned a busy-condition location which is to be checked (tested) by every processor which is going to access the common memory area. The busy-condition location contains either the value '0' or the slot identifier of the processor which is just using the memory area and which has to release it by overwriting the busy-condition location with '0'.

The command TSG supports the operations necessary to test and set a busy-condition location.

Operation	Parameters	Description
TSG	-32768 to +32767	Test and set the busy-condition location that has been addressed by means of BR register + constant

The absolute address must be within 0 and EFFFH. Otherwise, the processor will recognize an execution time error and call the **OB 31**. Accumulator 1 contains error identifiers which describe in more detail the error occurred (see Subsection 5.6.2 "Other execution time errors").

The Low-byte of the word addressed by means of BR register + constant is used as the busy-condition location. If the contents of the Low-byte equal '0', the TSG command enters the slot identifier (from RS 29) into the busy-condition location.

The testing (= reading) and the possible setting (= writing) form a program unit that cannot be interrupted.

The test result may be evaluated in the condition codes DSP0 and DSP1:

DSP0	DSP1	Description
0	0	Contents of the busy-condition location are '0'; the processor enters its slot identifier.
0	1	The own slot identifier has already been entered into the busy-condition location.
1	0	The busy-condition location contains a strange slot identifier.

IMPORTANT!

The TSG command must be used by all the processors which are to perform synchronized accesses to a common global memory area.

Also note the explanations on the SED and SEE commands (set/enable semaphore, Subsection 3.2.2) and on the special-function organization block OB 218 (Assigning a page, Subsection 6.6.3).

Load and transfer operations for the byte-organized global memory

Operation	Parameters	Description
LB GB	-32768 to +32767	Load the byte that has been addressed by means of BR register + constant into accu 1-LL
LB GW	-32768 to +32767	Load the word that has been addressed by means of BR register + constant into accu 1-L
LB GD	-32768 to +32767	Load the double word that has been addressed by means of BR register + constant into accu 1
TB GB	-32768 to +32767	Transfer the contents of accu 1-LL into the byte that has been addressed by means of BR register + constant
TB GW	-32768 to +32767	Transfer the contents of accu 1-L into the word that has been addressed by means of BR register + constant
TB GD	-32768 to +32767	Transfer the contents of accu 1-L into the double word that has been addressed by means of BR register + constant

The absolute address must be within

- 0 and EFFFH (for LB GB, TB GB)
- 0 and EFFEh (for LB GB, TB GW)
- 0 and EFFCh (for LB GD, TB GD).

If these limits are exceeded, the processor will recognize an execution time error and call the **OB 31**. Accu 1 contains error identifiers which describe the error occurred in more detail (see Subsection 5.6.2 "Other execution time errors").

Load and transfer operations for the word-organized global memory

Operation	Parameters	Description
LW GW	-32768 to +32767	Load the word that has been addressed by means of BR register + constant into accu 1-L
LW GD	-32768 to +32767	Load the double word that has been addressed by means of BR register + constant into accu 1-L
TW GW	-32768 to +32767	Transfer the contents of accu 1-L into the word that has been addressed by means of BR register + constant
TW GD	-32768 to +32767	Transfer the contents of accu 1 into the double word that has been addressed by means of BR register + constant

The absolute address must be between 0 and EFFFH (for LW GW, TW GW) or 0 and EFFEh (for LW GD, TW GD). Otherwise, the processor will recognize an execution time error and call the **OB 31**. Accu 1 contains error identifiers which describe the error occurred in more detail (see Subsection 5.6.2 "Other execution time errors").

9.3.5 Access to the Page Frame

The global area contains a 'window' located between the addresses F400H and FBFFH in order to insert one out of max. 256 memory areas (= pages). One page requires address space of up to 2k and may be byte or word-organized. Before the page area is accessed, one of the 256 pages must be selected by entering its page number into the Select register. The writing into the Select register as well as the subsequent access to the page area cannot be interrupted.

The following commands enable you to access *byte or word-organized* pages via an absolute memory address. The absolute address equals the total of the BR register contents and the constant contained in the command (-32768 through +32767).

Before the page area is accessed (load/transfer), one of the 256 pages is to be called. To do this, enter the number of the page to be called in accu 1-L; this number is entered in the Page register by means of the ACR command. All page operations that follow will write the contents of the Page register into the Select register before accessing a page.

The Page register is retained if

- program execution is continued in another block by means of a jump instruction (JU/JC) or
- another program level is nested.

The Page register is set to '0' before a program level is called.

Calling a page

Operation	Parameters	Description
ACR		Call the page whose number is indicated in accu 1-L

permissible values: 0 through 255

The page number must be between 0 and 255. Otherwise, the processor will recognize an execution time error and call the OB 31. Accu 1 contains error identifiers which describe the error occurred in more detail (see Subsection 5.6.2 "Other execution time errors").

Testing and setting of a busy-condition location in the page area

The access of individual processors to common memory areas can be controlled via a busy-condition location. Each common memory area is assigned a busy-condition location which is to be checked (tested) by every processor which is going to access the common memory area. The busy-condition location contains either the value '0' or the slot identifier of the processor which is just using the memory area and which has to release it by overwriting the busy-condition location with '0'.

The command TSC supports the operations necessary to test and set a busy-condition location.

Operation	Parameters	Description
TSC	-32768 to +32767	Test and set the busy-condition location of the called page, that has been addressed by means of BR register + constant

The absolute address must be within F400H and FBFFH. Otherwise, the processor will recognize an execution time error and call the OB 31. Accu 1 contains error identifiers which describe the error occurred in more detail (see Subsection 5.6.2 "Other execution time errors").

The Low-byte of the word addressed by means of BR register + constant is used as the busy-condition location. If the contents of the Low-byte equal '0', the TSC command enters the slot identifier (from RS 29) into the busy-condition location.

The testing (= reading) and the possible setting (= writing) form a program unit that cannot be interrupted.

The test result may be evaluated in the condition codes DSP0 and DSP1:

DSP0	DSP1	Description
0	0	Contents of the busy-condition location are '0'; the processor enters its slot identifier.
0	1	The processor's own slot identifier has already been entered into the busy-condition location.
1	0	The busy-condition location contains a strange slot identifier.

IMPORTANT!

The TSC command must be used by all the processors which are to perform synchronized accesses to a common global memory area.

Also note the explanations on the SED and SEE commands (set/enable semaphore, Subsection 3.2.2) and on the special-function organization block OB 218 (Assigning a page, Subsection 6.6.3).

Load and transfer operations for the byte-organized pages

Operation	Parameters	Description
LB CB	-32768 to +32767	Load the byte that has been addressed by means of BR register + constant from the page called into accu 1-LL
LB CW	-32768 to +32767	Load the word that has been addressed by means of BR register + constant from the page called into accu 1-L
LB CD	-32768 to +32767	Load the double word that has been addressed by means of BR register + constant from the page called into accu 1

Operation	Parameters	Description
TB CB	-32768 to +32767	Transfer the contents of accu 1-LL into the byte on the page called that has been addressed by means of BR register + constant
TB CW	-32768 to +32767	Transfer the contents of accu 1-L into the word on the page called that has been addressed by means of BR register + constant
TB CD	-32768 to +32767	Transfer the contents of accu 1 into the double word on the page called that has been addressed by means of BR register + constant

The absolute address must be within

- F400H and FBFFH (for LB CB, TB CB)
- F400H and FBFEH (for LB CW, TB CW)
- F400H and FBFCH (for LB CD, TB CD).

If these limits are exceeded, the processor will recognize an execution time error and call the **OB 31**. Accu 1 contains error identifiers which describe the error occurred in more detail (see Subsection 5.6.2 "Other execution time errors").

Load and transfer operations for the word-organized pages

Operation	Parameters	Description
LW CW	-32768 to +32767	Load the word that has been addressed by means of BR register + constant from the page called into accu 1-L
LW CD	-32768 to +32767	Load the double word that has been addressed by means of BR register + constant from the page called into accu 1
TW CW	-32768 to +32767	Transfer the contents of accu 1-L into the word on the page called that has been addressed by means of BR register + constant
TW CD	-32768 to +32767	Transfer the contents of accu 1 into the double word on the page called that has been addressed by means of BR register + constant

The absolute address must be within F400H and FBFFH (for LW CW, TW CW) or F400H and FBFEH (for LW CD, TW CD). Otherwise, the processor will recognize an execution time error and call the **OB 31**. Accu 1 contains error identifiers which describe the error occurred in more detail (see Subsection 5.6.2 "Other execution time errors").

10 Multiprocessor Operation

10.1 Notes

The S5 135 U is particularly suitable for complex control tasks. This is due to the fact that up to 4 processors (CPU's) can be installed in the central controller. These processors operate simultaneously, but independently. You may operate the following processors in different combinations if multiprocessor operation is required:

- S processor, especially suitable for open-loop control (binary control tasks), monitoring and signalling; occupies 1 slot.
- R processor, especially suitable for computing and closed-loop control (digital control tasks), for communication, monitoring and signalling; occupies 1 slot.
- M processor, suitable for a wide variety of applications, for programming in higher-level programming languages; occupies 1 slot.
- CPU 928, universal application, especially suitable for open and closed-loop control and computing (binary and digital control tasks); occupies 2 slots.

Multiprocessor operation - when and how ?

- If your user program is too extensive for a single processor and if the memory capacity is inadequate it is advisable to distribute your program on several processors.
- If a particular section of your plant requires fast processing the best solution is to separate the corresponding program section from the total program and have it processed by a 'fast' processor of its own.
- If your plant consists of sections with clear demarcation lines and which run relatively independently of each other then it is best to have section 1 of the plant processed by processor 1, section 2 by processor 2 etc..

For multiprocessor operation, refer to the description "Multiprocessor operation in the S5 135U" (C79000-B8576-C500-xx). It is contained in Section 7 of this manual and explains the individual steps to be taken when commissioning your processor, it contains operating instructions, describes several typical problems and points out possible causes of errors.

IMPORTANT!

If a *coordinator* (= KOR) is plugged into the central controller, it is in the multiprocessor mode, irrespective of whether you operate the coordinator with one or several processors!

The coordinator's task is to coordinate data exchange between the individual processors. In order to be able to accomplish this task the coordinator will have to know how many slots are to be "coordinated", i.e. how many processors will be assigned access to the S5 bus in succession.

IMPORTANT!

Do not forget to set the number of the processors used at the coordinator (see instructions of the coordinator)! Check whether single-width (e.g. R processors) or double-width processors (e.g. CPU 928) are used.

If, e.g. the number '3' is set on the KOR, the KOR will then assign bus enable signals to three processors, one after the other:
The first three slots to the right of the coordinator are supplied (no. 17, no. 19, and no. 27, see operating instructions of the CPU 928). In this particular case, the fourth slot (no. 35) will not be supplied by the KOR. Any processor plugged into this slot will not be able to access the S5 bus.

IMPORTANT!

Coordinator and processors have to be plugged-in without leaving gaps!

For multiprocessor operation you will have to program the data block DB 1 for each of the processors involved. This block contains a list of the digital inputs and outputs as well as the interprocessor communication flag inputs and outputs assigned to the corresponding processor (see 10.3.1)!

10.2 Data Exchange between the Processors

The '**interprocessor communication flags**' (see 10.2.1) are available for cyclic exchange of binary data between the processors or between the processors and the communications processors.

The '**special functions for multiprocessor operation**' OB 200 through OB 205 support the exchange of large data quantities (e.g. complete data blocks) between R and M processor and CPU 928 (see 10.2.2)

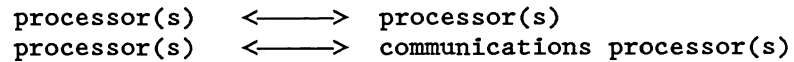
The '**handling blocks**' are available for communication with intelligent I/O's (IPs) and communications processors (CPs) (see 6.9).

If you wish to transfer long data fields and to ensure that the other processors will not interrupt this data transfer the software '**semaphores**' can be used (see 3.2.2).

10.2.1 Interprocessor Communication (IPC) Flags

The 'interprocessor communication flags' are available for the cyclic exchange of binary data. Primarily they are designed for byte by byte information transfer.

This data transfer is possible between



The system program transfers the IPC flags once per cycle. When data is transferred between processors the IPC flags are buffered on the coordinator.

IPC flags are flag bytes. They are defined in DB 1 either as input or output IPC flags for each processor. If, e.g. you have defined the flag byte 50 on processor 1 as the IPC flag **output** its signal state will be transferred cyclically via the coordinator to the processor, on which flag byte 50 has been defined as the IPC flag **input**.

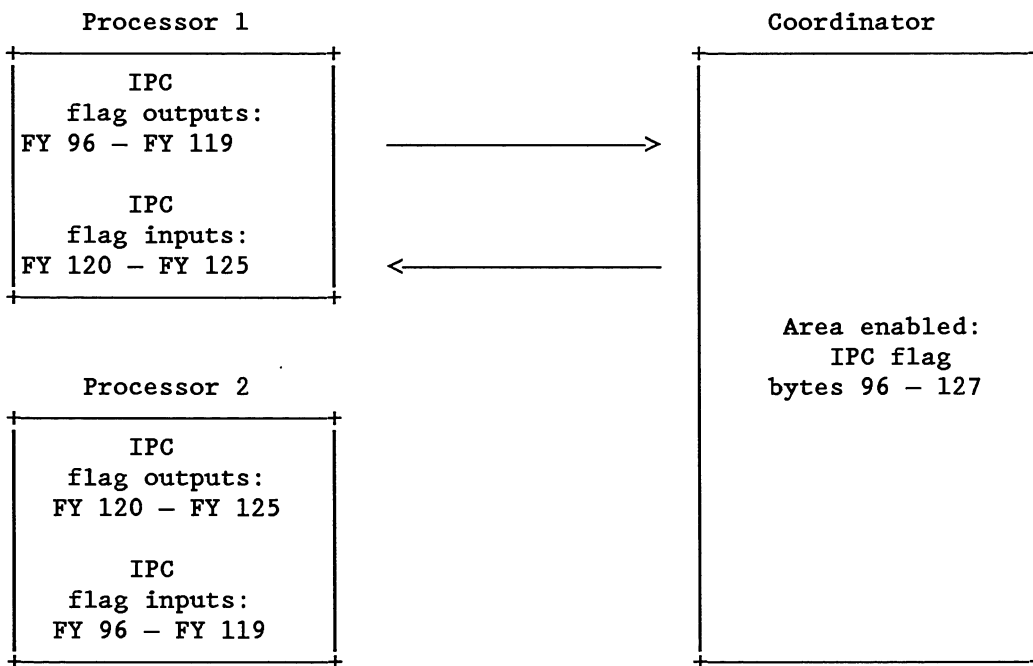
The memory area for the interprocessor communication flags on the coordinator and the communications processor covers addresses **F200H through F2FFH**. You have 256 IPC flag bytes available for each processor/communications processor.

Data exchange between processors

Data transfer between individual processors (CPU 928, R, S, and M processor) is executed via the coordinator. The processors read their flag bytes defined as IPC input flags in DB1 from the KOR or transfer their flag bytes defined as IPC output flags to the KOR.

You enable the number of IPC flags required on the coordinator: Division of the 256 flag bytes max. in areas of 32 bytes (8 areas) is possible by setting jumpers. Refer to the operating instructions of your coordinator.

Example:



IMPORTANT!

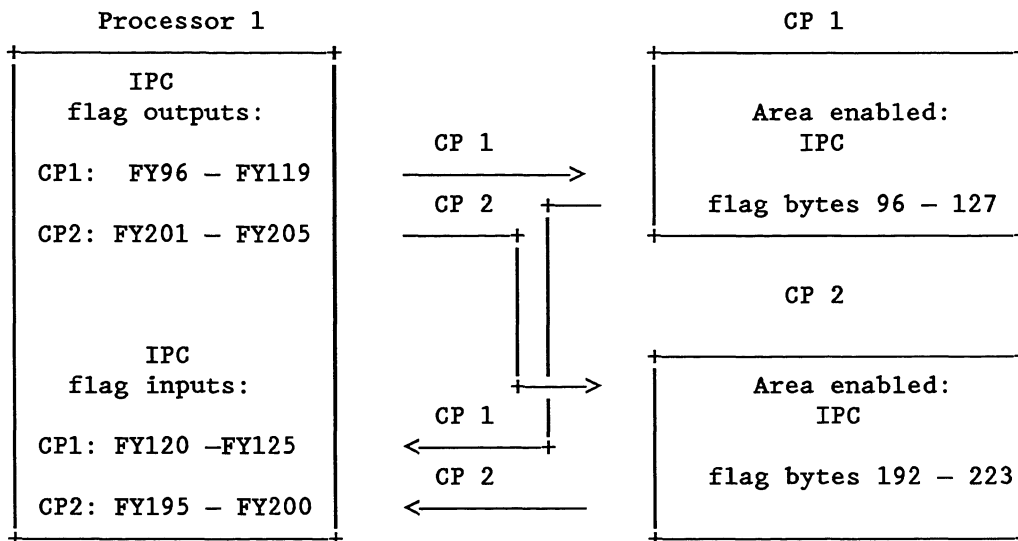
- * Only these flag bytes which have been enabled as IPC flags on the coordinator can be specified!
- * If a certain flag byte has been defined as an IPC flag input on one or more processors it will have to be defined as an IPC flag output on another processor. And: A flag byte may only be defined as an IPC flag output on one processor; however, it can be defined as an IPC flag input on e.g. three further processors!
- * The flag bytes not defined as IPC flags on a processor may be used just like "normal" flags!
- * Only specify the number of IPC flags actually required: the lower the number of IPC flags, the shorter the transfer time!

Data exchange between processors and communications processors

If data are to be transferred between *one* processor and *one* communications processor you will have to enable the number of IPC flags required on the communications processor (CP). There, you also have 256 bytes available which may be divided into areas of 32 bytes.

If data are to be transferred from *one* processor to *several* communications processors the areas enabled on the communications processors are not permitted to overlap, in order to prevent addresses from being assigned twice.

Example:



If your intention is to *simultaneously* use IPC flags on the coordinator and on one or several communications processors you will also have to avoid double addressing:

Divide the IPC flags on the KOR and the CPs into areas of 32 bytes; these IPC flag bytes that you use on the communications processor should be masked on the coordinator by removing the jumpers (see operating instructions of the coordinator used).

Once again a flag byte can only be defined as an IPC flag output on one processor. In contrast, a flag byte may be defined as an IPC input flag on several processors.

Transfer of interprocessor communication flags in multiprocessor operation

The IPC flags specified in DB 1 are transferred if the processor receives the signal from the coordinator which allows access to the I/O bus.

If several processors attempt to access the bus simultaneously the coordinator will output the bus enable signal to the processors, one after the other. Each processor is permitted to transfer only one byte at a time. This interleaved transfer may cause the IPC flag information that belongs together to be separated and subsequent processing may then be carried out using incorrect values.

If information consisting of more than 1 byte is to be transferred use the special function organization block OB 224: By calling OB 224 all IPC flags specified in DB 1 will be transferred as a block. As long as a processor is in the process of transferring IPC flags interruption by another processor is not possible. Since the next processor will have to wait, this means that cyclic program processing will be delayed.

Do not forget that the cycle time may increase considerably if OB 224 is called! (see 6.8.6)

10.2.2 Multiprocessor Communication

The special function organization blocks OB 200 through OB 205 allow large quantities of data to be transferred (e.g. complete data blocks) between R and M processors and CPU 928 in multiprocessor operation, with the data being buffered on the coordinator KOR C.

OB 200 initializes the mailbox in the coordinator KOR C, where the data blocks to be transferred are buffered.

OB 202 transfers a data field to the mailbox of KOR C and states how many data blocks may still be transferred.

OB 203 determines the number of free memory blocks in the mailbox of the coordinator KOR C.

OB 204 receives a data block from the mailbox of KOR C and indicates the number of data blocks that may still be received.

OB 205 determines the number of memory blocks occupied in the mailbox of KOR C.

A detailed user's guide for this special-function organization block titled "SIMATIC S5, S5-135U programmable controller, multiprocessing" can be found in Section 8 of this manual.

10.2.3 "Protected" Transfer of Connected Data Fields

If you wish to transfer long data fields and to ensure that the other processors will not interrupt this data transfer then you can use the software '**semaphores**' (see 3.2.2).

10.3 I/O Assignment

The I/O area of each processor covers the addresses **F000H through FFFFH**. The I/O modules are addressed in this area, the IPC flags and the page area are located there. Access to this I/O area is possible for all processors, both reading or writing is permitted. The coordinator's task is to coordinate access of the individual processors to the I/O area.

10.3.1 Data Block DB 1

In multiprocessor operation DB 1 must be programmed for each processor. This means that you specify the inputs and outputs, the IPC flag inputs and outputs used by the corresponding processor.

IMPORTANT!

Only the inputs and outputs that have been defined in DB 1 are considered when the process image is updated!

Input/alteration of DB 1

1. On-line via the programmer while the processor is in the stop state, if the processor is equipped with a user RAM.
2. By programming the user EPROM.

IMPORTANT!

The input or altered DB 1 will be accepted by the processor only in the start-up mode "cold restart"!

This is how DB 1 is programmed via masks and softkeys:

DB 1:

```

DIGITAL INPUTS      , 0, 1, 2, 3, 7, 10, , , , ,
                    , , , , , , , , , ,
                    , , , , , , , , , ,
DIGITAL OUTPUTS     , 0, 2, 4, 12, , , , , ,
                    , , , , , , , , , ,
IPC FLAG INPUTS     , 50, 51, 60, , , , , ,
                    , , , , , , , , , ,
IPC FLAG OUTPUTS    , 70, 72, 100, , , , , ,
                    , , , , , , , , , ,
TIMER BLOCK LENGTH  , 128,

```

(Note that a DB 1 entered with "S5-DOS" via the mask may produce an error when being read with the PG software "Studos"!)

This is how DB 1 is programmed manually:

- The data words 0, 1, and 2 must have the DB 1 start identifier. This means that they must be permanently assigned with

DW 0: KH = 4D41
DW 1: KH = 534B
DW 2: KH = 3031

- The individual operand areas are specified from data word 3 onwards.

Enter an identifier for each of the operands:
Possible code words are:

Code word for digital inputs	KH = DE00
Code word for digital outputs	KH = DA00
Code word for input IPC flags	KH = CE00
Code word for output IPC flags	KH = CA00
Code word for timer block length	KH = BB00

After the code word list the numbers of the inputs and outputs used and the required timer block length in fixed point format.

- The last data word will receive the value KH = EE00 as the DB 1 end identifier.

IMPORTANT!

- * The sequence of the entries is arbitrary. Note that the process image of the inputs and outputs is updated in the same order in which they have been entered in DB 1.
- * Multiple entries of identical bytes, e.g. for test purposes are possible. Once again note that the process image of bytes entered several times is updated several times.
- * Input KH = EE00 after the last entry in DB 1 as the end identifier!

An example of how to create DB 1 is given on the next page.

Example of DB 1

```

0:      KH = 4D41;          DW 0-2:
1:      KH = 534B;          start identifier
2:      KH = 3031;          for DB 1
3:      KH = DE00;          code word for digital inputs
4:      KF = +00000;        input byte 0
5:      KF = +00001;        input byte 1
6:      KF = +00002;        input byte 2
7:      KF = +00003;        input byte 3
8:      KF = +00007;        .
9:      KF = +00010;        .
10:     KH = DA00           code word for digital outputs
11:     KF = +00000;        output byte 0
12:     KF = +00002;        output byte 2
13:     KF = +00004;        .
14:     KF = +00012;        .
15:     KH = CE00;          code word for IPC flag inputs
16:     KF = +00050;        flag byte 50
17:     KF = +00051;        .
18:     KF = +00060;        .
19:     KH = CA00;          code word for IPC flag outputs
20:     KF = +00070;        flag byte 70
21:     KF = +00072;        .
22:     KF = +00100;        .
23:     KH = BB00;          code word for timer block length *
24:     KF = +00127;        timer 0 through timer 127
25:     KH = EE00;          end identifier

```

* By entering a timer block length in DB 1 you can specify the number of timer locations to be updated cyclically by the system program. This system function should be programmed in DX 0, see "Extended data block DX 0".

During a cold restart DB 1 is adopted by the system program. It checks whether the inputs and outputs or IPC flags acknowledge on the appropriate modules. If they do not, the processor will stop with a DB 1 error and the STOP led will flash slowly. Your user program will not be run.

As soon as you have programmed a DB 1 and it has been transmitted to the processor the start-up mode 'cold restart' the following applies:

- Access to I/O modules *via the process image* is permissible only for the inputs specified in DB 1 (commands L/T, IB, IW, ID, QB, QW, QD and logic operations with inputs and outputs).
- Direct loading of I/O bytes *by passing the process image* using the instructions L PB/PY, PW, OB, OW is possible for all acknowledging inputs - irrespective of the entries in DB 1.
- Direct transfer (T PB/PY, PW, OB, OW) to the bytes 0 through 127, however, is possible only for the outputs specified in DB1 since the process image is also written to with direct transfer. Direct transfer to byte addresses > 127 is possible whether an entry has been made in DB 1 or not.

10.4 Start-up during Multiprocessor Operation

This is how the coordinator is started during multiprocessor operation:

- The mode selectors of all processors plugged-in are set to 'RUN'. The mode selector of the coordinator is set to 'STOP'.
- Switch the mode selector on the coordinator from 'STOP' to 'RUN'.

(Starting up the programmable controller during multiprocessor operation simply by starting the coordinator is possible only if the stop state was actually caused by the coordinator.)

or:

- The mode selector of all processors plugged-in and of the coordinator are on 'RUN'.
- The processor responsible for the stop page is started by means of the online function "PC start", in the required start-up mode.

The start-up mode of the individual processors now depends on how and whether their settings have been changed in the meantime, while in the stop state. This means that it is possible that some processors carry out a manual warm restart, others, however, a cold restart.

If the processor settings have *not* been changed in the meantime they will carry out a manual warm restart (CPU 928 and R processor) or a manual cold restart without reset (S processor).

IMPORTANT!

Due to the differing start-up modes it is possible that, if the programmable controller was in the cycle before the stoppage, incorrect signal states are transferred from one processor to the other via the IPC flags. This can be prevented by programming the start-up OBs 20, 21, and 22.

The coordinator is started automatically following a power failure and a subsequent power return. In this case all S processors will carry out an automatic cold restart without reset, all CPUs 928 and R processors an automatic restart or an automatic cold restart, depending on the presetting in DX 0.

The start-up of the individual processors is **synchronized** during multiprocessor operation, i.e. the individual processors wait until all others have completed their start-up and then simultaneously start cyclic operation. In the case of the CPU 928 and the R processor you have the option of disabling this start-up synchronization by programming DX 0.

10.5 Test Operation

This is how test operation is triggered:

- The "test operation" function must be enabled at the coordinator.
- Switch the mode selector at the KOR from 'STOP' to 'TEST'. The BASP led will no longer be lit.
- Select the start-up mode for the processors that are to RUN.

Special features during test operation

In the test mode you have the option of starting up the processors individually or in various combinations. The processors that are in the stop state can no longer block the whole programmable controller.

The start-up of the individual processors is *not* synchronized in the test mode. The processors will start their cyclic program processing at different times, depending on the length of the start-up OBs 20, 21 or 22.

If an error occurs on one of the processors only this processor will stop during test operation. The other processors are not affected by the error.

Exception: During overall reset, with online functions "stop" PC , PRO.CTRL, PRO.CTRLE and if there is a DB 1 error at a processor the complete programmable controller will stop.

Output of the BASP signal is suppressed for all processors in the test mode. The digital I/O outputs will *not* be disabled if an error occurs (exceptions see above).

IMPORTANT!

In test operation, output of the BASP signal is suppressed for every processor. If an error occurs the digital I/O outputs are *not* disabled (for exceptions see above).

IMPORTANT!

Make sure to deactivate the test operation mode by means of the respective setting at the coordinator after the commissioning has been completed! You will thus avoid incorrect operation which may lead to dangerous situations in the plant/process.

Summary: This is how to start your multiprocessor controller

- Set the number of processors used at the coordinator.
Enable interprocessor communication flags on the coordinator.
- Plug-in the processors in the central controller; do not leave gaps!
- Switch on the power supply.
- Switch the mode selector at the coordinator to 'STOP'.
- Execute overall reset for all processors plugged-in.
- Load user program in the processors.
- Execute cold restart for all processors.
- Switch the mode selector at the coordinator to 'RUN' or 'TEST'.

11 Testing Aids: Online Functions

The online functions are an important aid when it comes to testing your user program. Refer to your programmer manual for detailed information on the operation of the programmer and the application of these functions. The following chapter will describe some special features of online functions in connection with the CPU 928.

The online functions are executed at defined points in the programmable controller. For this purpose, points exist in the system program that are system checkpoints and points in the user program that are user checkpoints.

System checkpoints

The system checkpoint stop exists in the STOP mode. This checkpoint is called regularly.

The system checkpoint cycle is called at the end of program processing level CYCLE during the RUN mode and before the process image is updated.

If the processor is in the WAIT MODE the system checkpoint wait state will be called regularly.

In addition to that there is a time dependent system checkpoint timeout. This checkpoint is called if none of the other system checkpoints has been reached within 250 ms. This means that nesting of this system checkpoint is possible during program execution. This may occur e.g. in the case of a permanent loop in the user program or in cycles which continue for more than 250 ms.

Processing of a online function at a system checkpoint will be interrupted after 5 ms max. and is continued at the next system checkpoint (see table 11.11).

User checkpoints

A user checkpoint is used for the test functions 'STATUS' and 'PROCESSING CONTROL'. This checkpoint is called if a command is executed which has been marked by the PG.

WAIT MODE

Up to now, you are familiar with the STOP, START-UP, and RUN mode. The processor will adopt another mode if the online function 'PROCESSING CONTROL' is executed: the WAIT MODE. If the processor is in this mode, calling of further online functions is possible.

Characteristics of the wait mode

- No user program processing is carried out in the wait mode.
- LEDs on the front panel:

RUN LED:	off
STOP LED:	off
BASP LED:	on
- All timers are 'frozen', i.e., no timers run (the timer locations are not altered). Furthermore all system timers are stopped, just as during closed-loop control and time-driven processing.

The timers will continue to run after the wait mode has been left (see Chapter 10.3).

- Causes of interruptions such as PEU, MP-STP or STP-SCH are recorded in the wait mode, however, no reaction takes place.

If causes of interruptions were recorded in the wait mode, the corresponding program levels are called immediately after the wait mode has been left.

If NAU occurs, the wait mode is left and the online function 'PROCESSING CONTROL' aborted. Following power on 'BARBEND' is marked in the control bits. STOP can only be exited by means of a cold restart.

11.1 Online Function 'STATUS VARIABLES'

The online function 'STATUS VARIABLES' allows you to have the current signal statuses of certain operands (process variables) output. This function activates system checkpoints in the cycle, in the stop mode and the wait mode. If the system checkpoint is reached the current signal status of the desired process variable is output. You can specify all process variables. No ADF is triggered in the area of the process image.

Running of the function during program execution:

If the function is processed in the START-UP mode or RUN program execution will be continued until the system checkpoint 'cycle' is reached. Then the signal status of the operands at the cycle end will be scanned and output. Inputs are read from the **process image**. The signal statuses will be updated while the program is running until the function is aborted. The signal statuses are *not* scanned at *every* system checkpoint.

If the system checkpoint 'cycle' is *not* reached *no output* of the signal status will be carried out (e.g. in the case of a permanent loop in the user program)!

Running of the function in the stop mode:

If the function 'STATUS VARIABLES' is processed in STOP the signal statuses of the operands will be output as they are at the system checkpoint 'stop status'. In doing so it is important that the inputs are scanned and output **directly** by the I/O module. Thus it is possible to e.g. test whether an I/O input signal is actually transferred to the processor. For multi-processor operation you may specify *all* inputs irrespective of the assignment made in DB 1. The outputs are read from the process image.

Running of the function in the wait mode:

Calling of the function 'STATUS VARIABLES' is also possible if the processor is in the wait mode due to the 'PROCESSING CONTROL' function. The signal statuses of the operands are scanned and output at the system checkpoint 'wait mode'. Just as in the stop mode, the inputs are read directly, the outputs from the **process image**.

If the processor changes from one operating mode to the other (e.g. RUN -> STOP -> MANUAL WARM RESTART) the function will remain called. 'STATUS VARIABLES' is terminated by pressing the abort key on the programmer.

Note: The variables are *not* output in *every cycle following*.

11.2 Online Function 'STATUS'

The online function 'STATUS' allows you to test connected command sequences in a block at any point in the user program. The current signal statuses of the operands, the contents of the accumulators, the RLO etc. are output at the programmer for every instruction executed in the block. In the same way the parameters assigned to function blocks can also be tested: The current values of the actual operands will be displayed.

Calling the function and preselecting the stop points

If you call the 'STATUS' function at the PG and input the block type and block number (possibly with nesting sequence and search term) of the block to be tested this means that you preselect a so-called stop point.

If the function is called during program execution in the START-UP or RUN mode then program processing will be continued until the command marked by the preselected stop point is reached in the correct nesting sequence. Then the commands monitored are executed up to the command boundary and the results of command execution are output at the PG.

Calling of the 'STATUS' function is also possible in the stop status. After this has been done, a cold as well as a manual warm restart is possible. The processor will then process the user program until the preselected stop point is reached. The data for the desired command sequence are the output. Thus, the 'STATUS' function is also suitable for e.g. testing the user program during start-up or in the first cycle.

Note: The results of command execution are *not* output in every cycle following.

Nesting and interruptions

A command sequence marked by a stop point will be executed completely, even if another program level (e.g. an error OB, a process or time interrupt) is nested and processed in the meantime.

If an interrupt stops the processor in a nested program level the data will be output in the stop mode up to the instruction executed last before the nesting. Data of the remaining instructions are filled with '0' (also SAC = 0).

If the processor changes from one operating mode to another (e.g. RUN --> STOP --> manual warm restart) the function will remain called. 'STATUS' is terminated by pressing the abort key on the programmer.

11.3 Online Function 'PROCESSING CONTROL'

The online function 'PROCESSING CONTROL' allows you to check individual program steps at any location in the user program. This is done by stopping program execution and having the processor execute one command after the other. The current signal statuses of the operands, the contents of the accumulators, the RLO etc. are output at the programmer for every command executed.

Calling the function and preselecting the 1st stop point

In order to call the 'PROCESSING CONTROL' function you specify the block type and block number (possibly with nesting sequence) of the block to be tested and mark the first command at the PG whose data are to be output. Thus you preselect the first stop point. 'PRO.CTRL' will be marked in the control bits. Command output is inhibited (BASP LED = on).

Note: If test operation has been set at the coordinator, the command output is not inhibited (BASP LED = off). When commands which alter the digital peripherals are now being processed or when the processor is updating the process image, the process interface modules output the corresponding signals.

If you preselect the first stop point during program execution in the START-UP or RUN mode the processor will continue program execution until the command is reached that has been marked by the preselected stop point. The command is carried out until the command boundary is reached. (The D0 FW and D0 DW commands will be processed including the command substituted.)

Then the processor will transfer to the wait mode. There the data of the instruction executed will be output.

Calling the function in the stop mode:

Calling of the 'PROCESSING CONTROL' function and preselection of the first stop point is also possible in the stop mode. The processor will remain in the stop state. You may now execute either a cold restart or a manual warm restart. The processor will run the program until the command marked is reached and will then proceed as above.

Continue function and preselect another stop point

Situation: The processor is in the wait mode.

There are two ways of continuing the 'PROCESSING CONTROL' function.

1. You shift the existing stop point:

The preselected stop point is shifted by one command. The processor will leave the wait mode and will continue program execution at this particular instruction. When the instruction has been processed up to the command boundaries the processor will again transfer to the wait mode and will output data there. However, if the subsequent command is reached in a nested program level, the processor will continue the program. The shifted stop point will remain set.

Important: A stop point cannot be shifted in the stop status!

2. You preselect a new stop point:

You preselect any command at the PG either in the same or in another block. The processor will continue the program until the new stop point is reached. The command will be executed up to the command boundary. Then the processor will go over to the wait mode and will output data.

Using 'PROCESSING CONTROL' you may let the processor continue through another complete cycle (test cycle-by-cycle). To do this, set the stop point in wait mode to the same command as before. Make sure that the command is not part of a program loop. Otherwise, the loop is executed once; the program processing stops at the cycle boundary.

Note: You can call other functions such as 'OUTPUT DIR', 'STATUS VARIABLES' or 'CONTROL VARIABLE' in the wait mode.

As soon as the wait mode is left and the program is continued, the timer and system timers will continue until the stop point is reached.

Cancelling the stop point:

If a preselected stop point has not yet been reached you can still cancel it by pressing the abort key on the PG. The processor will then go over to the wait mode. You can then preselect a new stop point or call 'PROCESSING CONTROL END'.

Aborting the function:

You may abort the function during the program, in the wait mode or in the stop status by calling 'PROCESSING CONTROL END'. The processor will stop (or remain in the stop state). The STOP LED will flash slowly. 'PRO.CTRLE' will be marked in the control bits. A cold restart is required afterwards.

If an interface error or NAU occurs during the 'PROCESSING CONTROL' (interruption at the PG cable) the function will be aborted as above.

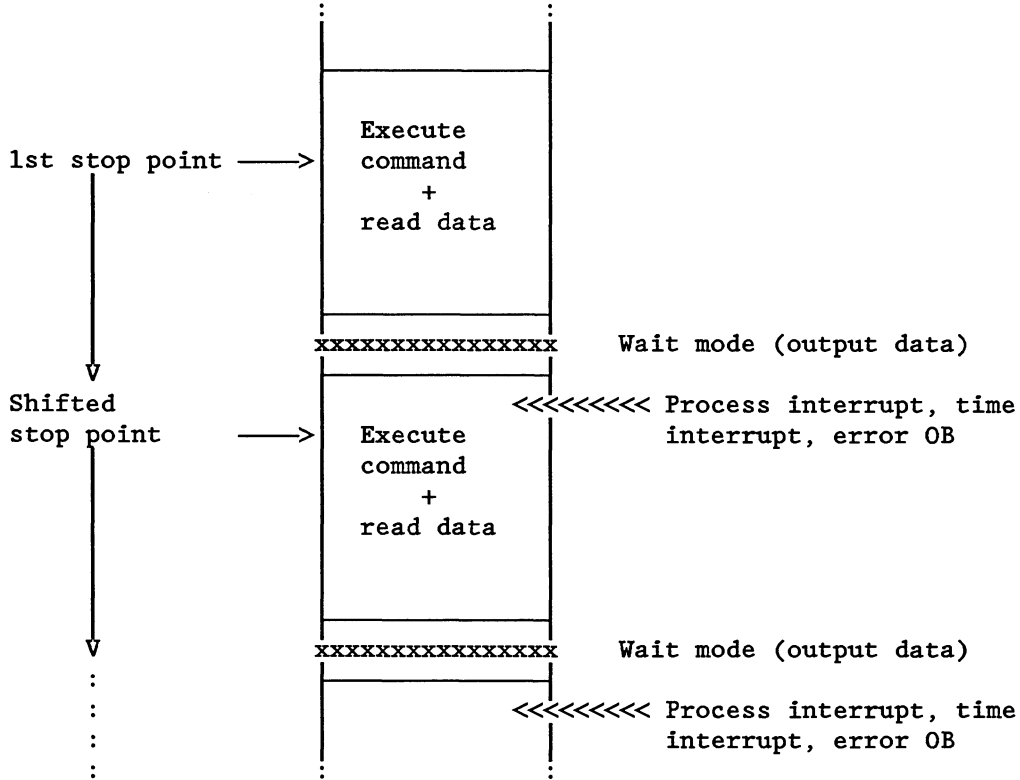
Nesting

Nesting of other program levels is possible if 'PROCESSING CONTROL' is called and before the processor goes over to the wait mode as well as during the transition from the wait mode.

If the command at the stop point is processed and another program level is called at this point (e.g. an error OB, a process or time interrupt), this level will only be nested and processed completely when the wait mode is left again.

IMPORTANT!

Data are read and output at the command boundaries. All corresponding nestings have still not been processed.



If requests such as PEU, MP-STP, STP-SCH etc. have occurred during the wait mode they will only be recorded. They will become effective immediately after the wait mode has been left: A program level is nested or an interrupt will cause the processor to stop. Valid is the order of events. Simultaneous requests are prioritized.

Note: When the processor is in the wait mode and there is a nesting request, you can set a stop point at one of the nesting commands. At a command that triggers a timeout (QVZ) you may thus observe the QVZ error OB.

Interruptions

- Program execution (start-up/run) --> stop status

If causes of interruptions occur during program execution (e.g. MP-STP, PEU, STP-SCH, error OB not programmed etc.) and the preselected stop point has not yet been reached the processor will immediately stop. If a start-up (cold restart or manual warm restart) is carried out the 'PROCESSING CONTROL' function will remain called and the stop point is still preselected.

- Command execution at the stop point (start-up/run) --> stop status

If, during command processing, stop conditions such as a stop switch or STEP5 command 'STP' occur at the stop point or shifted stop point, the processor goes into the stop mode immediately after command processing has been concluded and transfers the data. If in stop mode no new stop point is preselected, the processor goes into the *wait mode* following start-up. 'PROCESSING CONTROL' remains called up.

- Stop mode --> stop status

Causes of interruption that occur in the wait mode (e.g. MP-STP, PEU or STP-SCH) or that have been produced by previous commands (errors that lead to STOP) are recorded, however, the processor remains in the *wait mode*. Only when a new stop point is preselected in wait mode and the processor leaves this mode, will the causes of interruption occurred make the processor pass into the *stop state*. The preselected stop point is not reached. If a subsequent start-up (cold restart or manual warm restart) is carried out the stop point will still be preselected.

IMPORTANT!

If the mode selector is switched to stop in the wait mode the processor will not go over to the stop status until the wait mode is left.

IMPORTANT!

If causes of interruption during 'PROCESSING CONTROL' are responsible for the processor stopping the 'PROCESSING CONTROL' function (and any preselected stop point) will remain active after a subsequent start-up.

11.4 Online Function 'CONTROL'

By means of the 'CONTROL' function you can set the *output* bytes of the programmable controller to the desired signal status manually and directly by skipping the process image, or detect process interface modules that do not acknowledge (digital I/Os 0 to 127) (message to PG). You are offered the option to directly check and control the process units (motor, valves) supplied by the outputs.

IMPORTANT!

The 'CONTROL' function is permissible only in the stop mode!

Calling the function

The command output inhibit (BASP = off) is cancelled if the function is called in the stop mode. All the digital I/O's (F000H to F07FH) are cleared by overwriting each address with value '0'. The function may not be interrupted during the clearing of the I/O's.

The I/O outputs are controlled byte by byte, directly and without influencing the process image of outputs! Controlling of all I/O outputs is possible in multiprocessor operation (irrespective of an I/O assignment in DB1).

Execution of a cold restart or a manual warm restart is possible while the function is active (message "control ready" at the PG). Controlling will again be possible after the processor has stopped. The process interface output modules will not be erased in this particular case.

Aborting the function

The function is aborted by means of pressing the abort key at the PG. The command output inhibit will again be output (BASP = on).

11.5 Online Function 'CONTROL VARIABLES'

The online function 'CONTROL VARIABLES' allows you to alter the values of operands (process variables) once. This is permissible in every operating mode of the processor. All process variables can be specified. No ADF is triggered in the area of the process image.

The alteration will become effective at a system checkpoint.

Do not forget that subsequent overwriting of the values controlled is possible (e.g. by the user program or the process image updating)!

Note: The PG controls the process variables I, Q, F byte by byte and DW, T, and C word by word.

IMPORTANT!

If you control several operands the bytes altered (for DW, T, C the words) will be altered one after the other distributed over several system checkpoints.

11.6 Online Function 'COMPRESS' Memory

This function is used to shift all valid blocks of the user program together. This is carried out individually in the RAM module and the DB-RAM. Gaps that are created when blocks are erased or corrected will disappear. This is done by shifting a complete block to the beginning of the memory area. Execution of this operation is possible at system checkpoints 'cycle' and 'stop'.

The function will be aborted and an error message output if the BSTACK is not empty at the system checkpoint 'stop'. This is the case if an interrupt causes a stop during program execution. If this is the case, further compressing will only be possible in the cycle.

No further block will be shifted if a power failure occurs during compressing. If the function 'COMPRESS' memory is called again, shifting will be continued.

IMPORTANT!

The 'COMPRESS' memory function will identify the following errors in the block memory:

- incorrect block length
- invalidated pattern '7070' in the header
- invalid block type (invalid block number in the case of OBs).

The function is aborted. A message is output at the PG. An overall reset is then required. The function can be called up again only after the overall reset.

Note: 'COMPRESS memory' is not permitted as long as 'PROCESSING CONTROL' is active.

11.7 Online Function 'START'/'STOP'

PG operation corresponds to manual operation. Calling the 'STOP' function allows you to stop the programmable controller. The following will be displayed at the processor connected to the PG:

STOP-LED: on
BASP-LED: on

'PG-STP' is marked in the control bits. The control bit 'MP-STP' is set on the other processors in multiprocessor operation.

A processor can be started by a cold restart or a manual warm restart. The processor will leave the stop state in the case of single-processor operation. In the case of multiprocessor operation the start-up mode will be recorded first (control bit 'NEUST' or 'M W A' is set), however, the processor will remain in the stop state. You can start the programmable controller by means of 'start system'. This corresponds to the operation of the coordinator (switch set to RUN).

The PG 'START' function offers another alternative: You can select the start-up mode for all processors one after the other and start the PC with the last processor.

11.8 Online Function 'PC OVERALL RESET'

Overall reset of a processor is possible from the PG (corresponds to 'Delete all blocks'). The overall reset will be executed unconditionally (see Chapter 4.2).

If the processor is in the operating mode START-UP or RUN when the 'PC OVERALL RESET' is called a transition to the stop state will be executed first. In this case the organization block OB 28 - if loaded - will be called.

Note: 'PC OVERALL RESET' is not permissible as long as 'PROCESSING CONTROL' is active.

11.9 Online Function 'OUTPUT ADDRESS'

The 'OUTPUT ADDRESS' function allows you to output the contents of memory and I/O addresses in words hexadecimally at the PG. All addresses may be called. No ADF is triggered in the area of the process image, no QVZ is caused in the I/O area.

The high byte is represented as 'FF' in the byte addressable area (flags, process image).

The high byte is output as '00' in the I/O area when addresses acknowledge. If an I/O address does not acknowledge the high byte will be output as 'FF'.

11.10 Online Function 'MEMORY CONFIGURATION'

The 'MEMORY CONFIGURATION' function shows you the highest usable address of the RAM module ('0' will be displayed for EPROM) and the last address in the memory submodule occupied by blocks of the user program.

11.11 Table: Activities at the Checkpoints

Activities with online functions	System checkpoint				User checkpoint
	Stop	Cycle	Wait mode	Time- out	
Input address: write data 1)	*	*	*	*	
Block input: declare block valid	*	*	*	*	
Erase block	*	*	*	*	
Compress memory: shift block 2)	*3)	*			
Start/stop	*	*	*	*	
Overall reset	*	*		*	
STATUS VAR: read and outp. data	*	*	*		
STATUS: read and output data					*
Processing control: preselecting the stop point	*	*	*	*	
read data					*
output data	*		*		
Control process int. mod. 1)	*				
Control VAR 1)	*	*	*	*	

- 1) Activities that may be distributed over several checkpoints
- 2) One block max. per system checkpoint
- 3) Only if there is no BSTACK entry

Table: Activities carried out at system and user checkpoints

ANNEX A: Technical Data S5 135U

	S processor	R processor	CPU 928
typic. command execution times for bit commands with F, I, Q D formal operand	1.1 μ s 100 μ s 94 μ s	22 μ s 37 μ s 46 μ s	1 μ s 34 μ s 24 μ s
typic. command execution times for word commands loading operations L FY (byte) L FW (word) L FD (double word) fixed & float.p.arithm.	31 μ s 33 μ s 70 μ s <900 μ s	15 μ s 15 μ s 20 μ s <86 μ s	12 μ s 12 μ s 26 μ s <69 μ s
cyclic program processing (single process. oper.) - basic load when OB1/FB0 are called: - Added time for process image updating depend. on the number of I/O bytes (n) with $0 < n \leq 256$ - Added time for transfer of IPC flags depending on the number of IPC flags (n) with $0 < n \leq 256$	221/268 μ s 23 μ s + n x 1.63 μ s 25 μ s + n x 1.93 μ s	107/119 μ s 33 μ s + n x 6 μ s 35 μ s + n x 6.5 μ s	147/149 μ s 18 μ s + n x 1.58 μ s 19 μ s + n x 1.84 μ s
- Added time for timer location proc. depend. on time unit length (ZBL) ZBL=0 ZBL#0, n = number of timers <u>running</u> (increment 10ms) - interrupt driven program execution extension of cycle time by nesting of an empty OB2 (w/o STEP5 command) at a block boundary Response time	every 10 ms 20 μ s 50 μ s + ZBL x 9.3 μ s + n x 17.9 μ s 439 μ s 425 μ s	every 2.5 ms 50 μ s 60 μ s + ZBL x 1.56 μ s + n x 1.24 μ s 367 μ s 300 μ s	every 10 ms 5 μ s 200 μ s + n x 0.35 μ s (for $0 < n \leq 128$) 400 μ s + n x 0.35 μ s (for $128 < n \leq 256$) 330 μ s 280 μ s

	S processor	R processor	CPU 928
- time-driven program execution extension of cycle time by nesting of an empty OB13(w/o STEP5 command) at a command boundary	327,us	375,us	340,us for the 1st time interrupt OB, 180,us for any other time interrupt OB being due at the same time
time cycle for calling time driven program	100 ms	100 ms	10, 20, 50, 100, 200, 500 ms, 1, 2, 5 sec
cycle time monitoring preset adjustable between triggering poss.	100 ms - x	150 ms 1...4000 ms x	150 ms 1...6000 ms x
size of user memory (in K words)	≤ 32	≤ 32	≤ 32
size of memory for data blocks (DB-RAM in kwords)	approx. 3.7	approx. 11.1	approx. 23.3
number of timer and counter locations	128 each	128 each	256 each
number of flag bytes	256 each	256 each	256 each

Terms and definitions:

Basic load: Basic load is the part of the cyclic system execution time which can be measured without update of the process image and transfer of the IPC flags.

Response time: Response time is the time interval from activation of the program processing level PROCESS-INTERRUPT up to processing of the first command in OB 2, provided that OB 2 is called immediately after the detection of the process interrupt. However, the response time is increased when the next command or block boundary is to be waited for.

ANNEX B: Summary of Error Identifications

System data 3 and 4

SD 3	SD 4		
Construction of block address lists:			
8001H	yyyyH	Incorrect block length	yyyy=addr. of block w. incorrect length
8002H	yyyyH	Computed end addr. of block in memory wrong	yyyy=block address
8003H	yyyyH	Invalid block id	yyyy=addr. of block with invalid id
8004H	yyyyH	OB number too high (perm.:OB 1 to 39)	yyyy=addr. of block with incorrect number
8005H	yyyyH	Data block number 0 (permitted: 1 to 255)	yyyy=addr. of block with
Structure of address list for process image updating:			
0410H	yyyyH	Invalid identifier	yyyy=incorrect identifier
0411H	yyyyH	Incorrect parameter in address list "digital inputs" (permitted: 128)	yyyy=addr. of input-byte specified incorrectly
0412H	yyyyH	Incorrect parameter in address list "digital outputs" (permitted: 128)	yyyy=addr. of output-byte specified incorrectly
0413H	yyyyH	Incorrect parameter in address list "IPC flag input" (permitted: 256)	yyyy=addr. of flag byte specified incorrectly
0414H	yyyyH	Incorrect parameter in address list "IPC flag output" (permitted: 256)	yyyy=addr. of flag byte specified incorrectly
0415H	yyyyH	Invalid no. of timer locations (perm.:256)	yyyy=incorr. number of timer locations
0419H	yyyyH	Acknowledgement delay at digital inputs	yyyy=address of input byte not acknowl.
041AH	yyyyH	Acknowledgement delay at digital outputs	yyyy=address of output byte not acknowl.
041BH	yyyyH	Acknowledgement delay at IPC flag input	yyyy=address of flag byte not acknowl.
041CH	yyyyH	Acknowledgement delay at IPC flag output	yyyy=address of flag byte not acknowl.
Evaluation of DB 2:			
0421H	DByyH	Data block not loaded	yy = number of DB not loaded
0422H	FByyH	Function block not loaded	yy = number of FB not loaded
0423H	FByyH	Function block not recognized	yy = number of FB not recog.
0424H	FByyH	Function block loaded with incorr. PG software	yy = number of function block
0425H	DByyH	Incorrect length of controller DB	yy = number of data block
0426H	-	Not enough memory capacity in DB-RAM	
Evaluation of DX 0:			
0431H	yyyyH	Invalid identifier	yyyy = incorrect identifier
0432H	yyyyH	Unknown parameter	yyyy = incorrect parameter
0434H	yyyyH	Illegal number of timer locations (perm.:256)	yyyy = incorrect number of timer locations
0435H	yyyyH	Illegal cycle time (perm.: 1 ms to 4 s)	yyyy = incorrect time value

Accu 1 and accu 2

Accu1	Accu2		
Controller processing			
0801H	DByyH	Sampling time error	yy = no. of resp. DB call OB 34
0802H	DByyH	Controller DB not loaded	yy = no. of DB not loaded call OB 34
0803H	FByyH	Controller FB not loaded	yy = no. of FB not loaded call OB 34
0804H	FByyH	Controller FB not recog.	yy = no. of FB call OB 34
0805H	FByyH	Controller FB loaded with incorrect PG software	call OB 34
0806H	DByyH	Incorrect length of controller DB	yy = no. of DB
0880H	yyyyH	Acknowledgement delay (QVZ) during controller processing	call OB 34
STEP5 command code errors			
1801H	-	Substitution error at DO RS command	call OB 27
1802H	-	Substitution error at DO DW/DO FW command	call OB 27
1803H	-	Substitution error at DO=/DI= command	call OB 27
1804H	-	Substitution error at L=/T= command	call OB 27
1805H	-	Substitution error at A=-/AN=-/O=-/ON=-/=-/S=-and RB=- command	call OB 27
1811H	-	Command with illegal operation code	call OB 29
1812H	-	Comm. with illegal opcode (h-byte of 1 comm.word=68H)	call OB 29
1813H	-	Comm. with illegal opcode (h-byte of 1 comm.word=78H)	call OB 29
1814H	-	Comm. with illegal opcode (h-byte of 1 comm.word=70H)	call OB 29
1815H	-	Comm. with illegal opcode (h-byte of 1 comm.word=60H)	call OB 29
1821H	-	Illegal parameter at C DB0, C DB1, C DB2	call OB 30
182BH	-	Illegal parameter at JU(DO) OB 0	call OB 30
182CH	-	Illegal param. at JU(DO) OB > 39: spec. fct. not pres.	call OB 30
182DH	-	Illegal parameter at CX DX0	call OB 30
182EH	-	Illegal par.at LFW/TFW/LPW/TPW/LOW/TOW/LDD/TDD/DOFW255	call OB 30
182FH	-	Illegal par. at LIW/TIW/LQW/TQW 127	call OB 30
1830H	-	Illegal par. at LFD/TFD 253, 254,255	call OB 30
1831H	-	Illegal par. at LID/TID/LQD/TQD 125, 126, 127	call OB 30
1832H	-	Illegal par. at RLD/RRD/SSD/SLD/ 33-255	call OB 30
1833H	-	Illegal par. at SLW/SRW/LIR/TIR 16-255	call OB 30
1834H	-	Illegal par. at SED/SEE 32-255	call OB 30
1835H	-	Illegal par. at A=/AN= O=/ON=/S=/RB=/=-/RD=/FR=/SP=/ SR=/SEC=/SSU=/SFD=/L=/LD= LW=/T=0, 127-255	call OB 30
1836H	-	Illegal par. at DO=/LWD=0, 126-255	call OB 30
STEP5 execution time errors			
1A01H	-	Data block not loaded at CDB	call OB 19
1A02H	-	Data block not loaded at CXDX	call OB 19
1A03H	-	Block not loaded at JC(B) FB, OB, PB, and SB	call OB 19
1A04H	-	Block not loaded at DOC(B) FX	call OB 19
1A05H	-	Data block not loaded at OB 254 or 255	call OB 19
1A11H	-	Access to a data word not defined	call OB 32
1A12H	-	Transfer error to a data word not defined (T DR)	call OB 32
1A13H	-	Transfer error to a data word not defined (T DL)	call OB 32
1A14H	-	Transfer error to a data word not defined (T DW)	call OB 32
1A15H	-	Transfer error to a data word not defined (T DD)	call OB 32
1A21H	-	Error at GDB, GXDX: DB exists already	call OB 31
1A22H	-	Error at GDB, GXDX: Illegal DB length	call OB 31
1A23H	-	Error at GDB, GXDX: Main memory loc. in RAM insuffic.	call OB 31
1A25H	-	Error at DI=: Illegal parameter in accu 1	call OB 31
1A29H	-	Bracketstack under- or overflow after A(, O(,)	call OB 31
1A2AH	-	Error at CDB or CXDX: DB header length insuff.<5 word)	call OB 31
1A2BH	-	Function block loaded with wrong PG software	call OB 31

1A2CH	-	Error in ACR: page no. in accu 1-L > 255	call OB 31
1A31H	-	SF error at OB 254 or OB 255: DB exists already in RAM	call OB 31
1A32H	-	SF error at OB 254 or OB 255: New DB exists already	call OB 31
1A33H	-	SF error at OB 254 or OB 255: Main mem.loc.in RAM insuff.	call OB 31
1A3AH	-	SF error at OB 221: Illegal value for new cycle time	call OB 31
1A3BH	-	SF error at OB 223: Different start-up modes in multipro- cessor operation	call OB 31
1A41H	-	SF error at OB 240, 241 or 242: Illegal shift reg. or DB no.	call OB 31
1A42H	-	SF error at OB 241: Shift register not initialized	call OB 31
1A43H	-	SF error at OB 240: Main mem. loc. in DB-RAM insuff.	call OB 31
1A44H	-	SF error at OB 240: DWO of data block contents ≠ '0'	call OB 31
1A45H	-	SF error at OB 240: Illegal shift reg.length in DW1	call OB 31
1A46H	-	SF error at OB 240: Illegal pointer position or no.	call OB 31
1A47H	-	SF error at OB 120: Illegal values in accu 1 or accu 2-L	call OB 31
1A48H	-	SF error at OB 122: Illegal values in accu 1	call OB 31
1A49H	-	SF error at OB 110: Illegal values in accu 1	call OB 31
1A4AH	-	SF error at OB 121: Illegal values in accu 1 or accu 2-L	call OB 31
1A4BH	-	SF error at OB 123: Illegal values in accu 1	call OB 31
1A50H	-	Error at LRW, TRW: Illegal address	call OB 31
1A51H	-	Error at LRD, TRD: Illegal address	call OB 31
1A52H	-	Error at TSG, LBGB, LWGW, TBGB, TWGW: Illegal address	call OB 31
1A53H	-	Error at LBGW, LWGD, TBGW, TWGD: Illegal address	call OB 31
1A54H	-	Error at LBGD, TBGD: Illegal address	call OB 31
1A55H	-	Error at TSC, LBCB, LWCD, TBCW, TWCD: Illegal address	call OB 31
1A56H	-	Error at LBCW, LWCD, TBCW, TWCD: Illegal address	call OB 31
1A57H	-	Error at LBCD, TBCD: Illegal address	call OB 31
1A58H	-	Error at TNW/TNB: Source field is not fully contained in area	call OB 31
1A59H	-	Error at TNW/TNB: Target field is not fully contained in area	call OB 31
Acknowledgement delay:			
1E23H	yyyyH	Acknowledgement delay (QVZ) in the user program yyyy = QVZ address	call OB 23
1E25H	yyyyH	Acknowledgement delay at process image of dig. outputs yyyy = address of output byte not acknowledged	call OB 24
1E26H	yyyyH	Acknowledgement delay at process image of dig. inputs yyyy = address of input byte not acknowledged	call OB 24
1E27H	yyyyH	Acknowl. delay at proc. im. of IPC flag outputs yyyy = address of flag byte not acknowledged	call OB 24
1E28H	yyyyH	Acknowl. delay at proc. im. of IPC flag inputs yyyy = address of flag byte not acknowledged	call OB 24

ANNEX C: Overview of the STEP5 Operations Set

Basic operations

Operation	Parameter
• Boolean logic operations	
A I	0.0 to 127.7
A Q	0.0 to 127.7
A F	0.0 to 255.7
A D	0.0 to 255.15
A T	0 to 255
A C	0 to 255
AN I	0.0 to 127.7
AN Q	0.0 to 127.7
AN F	0.0 to 255.7
AN D	0.0 to 255.15
AN T	0 to 255
AN C	0 to 255
O I	0.0 to 127.7
O Q	0.0 to 127.7
O F	0.0 to 255.7
O D	0.0 to 255.15
O T	0 to 255
O C	0 to 255
ON I	0.0 to 127.7
ON Q	0.0 to 127.7
ON F	0.0 to 255.7
ON D	0.0 to 255.15
ON T	0 to 255
ON C	0 to 255
)	
A(
O(
O	
• Comparison operations:	
!=F	
><F	
>F	
>=F	
<F	
<=F	
!=D	
><D	
>D	
>=D	
<D	
<=D	
!=G	
><G	
>G	
>=G	
<G	
<=G	

Operation	Parameter
• Set/reset operations:	
S I	0.0 to 127.7
S Q	0.0 to 127.7
S F	0.0 to 255.7
S D	0.0 to 255.5
R I	0.0 to 127.7
R Q	0.0 to 127.7
R F	0.0 to 255.7
R D	0.0 to 255.15
= I	0.0 to 127.7
= Q	0.0 to 127.7
= F	0.0 to 255.7
= D	0.0 to 255.15
• Load operations:	
L IB	0 to 127
L IW	0 to 126
L ID	0 to 124
L QB	0 to 127
L QW	0 to 126
L QD	0 to 124
L FY	0 to 255
L FW	0 to 254
L FD	0 to 252
L DL	0 to 255
L DR	0 to 255
L DW	0 to 255
L DD	0 to 254
L T	0 to 255
L C	0 to 255
L PY	0 to 127 128 to 255
L PW	0 to 126 128 to 254
L OY	0 to 255
L OW	0 to 254
LC T	0 to 255
LC C	0 to 255
L KB	0 to 255
L KS	2 alphanumeric char.
L KM	16-bit pattern
L KH	0 to FFFF
L KF	-32 768 to +32 767
L KY	0 to 255 for each byte
L KT	0.0 to 999.3
L KC	0 to 999
L KG	1)

1) $\pm 0.1469368 \times 10^{-38}$ to
 $\pm 0.1701412 \times 10^{39}$

Operation	Parameter
• Timer and counter operations	
SP T	0 to 255
SE T	0 to 255
SD T	0 to 255
SS T	0 to 255
SF T	0 to 255
R T	0 to 255
S C	0 to 255
R C	0 to 255
CU C	0 to 255
CD C	0 to 255

• Transfer operations:

T IB	0 to 127
T IW	0 to 126
T ID	0 to 124
T QB	0 to 127
T QW	0 to 126
T QD	0 to 124
T FY	0 to 255
T FW	0 to 254
T FD	0 to 252
T DR	0 to 255
T DL	0 to 255
T DW	0 to 255
T DD	0 to 254
T PY	0 to 127
	128 to 255
T PW	0 to 126
	128 to 254
T OY	0 to 255
T OW	0 to 254

• Block call operations:

JU PB	0 to 255
JU FB	0 to 255
DOU FX	0 to 255
JU SB	0 to 255
JU OB	1 to 39
JU OB ¹⁾	40 to 255
JC PB	0 to 255
JC FB	0 to 255
DOC FX	0 to 255
JC SB	0 to 255
JC OB	1 to 39
JC OB ¹⁾	40 to 255
C DB	3 to 255
CX DX	1 to 255
BE	
BEC	
BEU	

1) Call of a special function

2) System operation

Operation	Parameter
• Arithmetic operations:	
+F	
-F	
xF	
:F	
+G	
-G	
xG	
:G	
• Other operations:	
NOP 0	
NOP 1	
STP	
BLD	0 to 255

Supplementary operations

Operation	Parameter
• Digital operations, (word operations:	
AW	
OW	
XOW	
• Timer and counter operations	
FR T	0 to 255
FR C	0 to 255
FR =	Formal operand
SP =	Formal operand
SD =	Formal operand
SEC =	Formal operand
SSU =	Formal operand
SFD =	Formal operand
RD =	Formal operand
• Load operations:	
L =	Formal operand
LD =	Formal operand
LW =	Formal operand
LWD =	Formal operand
L RS	0 to 255
L RT	0 to 255
L RI	0 to 255
L RJ	0 to 255
LIR ²⁾	0 to 15

Operation	Parameter
● Load operations (continued):	
LRW	-32768 ... +32767
LRD	-32768 ... +32767
LB GB	-32768 ... +32767
LB GW	-32768 ... +32767
LB GD	-32768 ... +32767
LW GW	-32768 ... +32767
LW GD	-32768 ... +32767
LB CB	-32768 ... +32767
LB CW	-32768 ... +32767
LB CD	-32768 ... +32767
LW CW	-32768 ... +32767
LW CD	-32768 ... +32767
● Transfer operations:	
T =	= Formal operand
T RI	0 to 255
T RJ	0 to 255
T RS ²⁾	0 to 255
T RT	0 to 255
TIR ²⁾	0 to 15
TNB ²⁾	0 to 255
TNW ²⁾	0 to 255
TRW	-32768 ... +32767
TRD	-32768 ... +32767
TSG	-32768 ... +32767
TB GB	-32768 ... +32767
TB GW	-32768 ... +32767
TB GD	-32768 ... +32767
TW GW	-32768 ... +32767
TW GD	-32768 ... +32767
TSC	-32768 ... +32767
TB CB	-32768 ... +32767
TB CW	-32768 ... +32767
TB CD	-32768 ... +32767
TW CW	-32768 ... +32767
TW CD	-32768 ... +32767
● Boolean logic operations:	
A =	Formal operand
AN =	Formal operand
O =	Formal operand
ON =	Formal operand
● Conversion operations:	
CFW	
CSW	
CSD	
DEF	
DUF	

Operation	Parameter
● Conversion operations: (continued):	
DED	
DUD	
FDG	
GFD	
● Shift operations:	
SLW	0 to 15
SRW	0 to 15
SLD	0 to 32
SSD	0 to 32
RLD	0 to 32
RRD	0 to 32
SSW	0 to 15
● Jump operations:	
JU =	Symbolic address
JC =	Symbolic address
JY =	Symbolic address
JN =	Symbolic address
JP =	Symbolic address
JM =	Symbolic address
JO =	Symbolic address
JOS =	Symbolic address
JUR	-32768 ... +32767
● BR register operations:	
MBR	0 ... FFFFF
ABR	-32768 ... +32767
MAS	
MAB	
MSA	
MSB	
MBA	
MBS	
● Set/reset operations:	
S =	Formal operand
RB =	Formal operand
= =	Formal operand
● Other operations:	
ACR	
RA	
IA	
ENT	
D	0 to 255
I	0 to 255
DO =	Formal operand

Operation	Parameter
-----------	-----------

● Other operations:
(continued):

DO DW	0 to 255
DO FW	0 to 255
DI 2)	
DO RS	0 to 255
TAK	
BLD	0 to 255
G DB	0 to 255
GX DX	0 to 255
SED	0 to 31
SEE	0 to 31

● Arithmetic operations:

ADD BN	-128 to +127
ADD KF	-32 768 to +32 767
ADD DF 2)	-2147483648 to +2147483647
+D 2)	
-D 2)	

2) System operations

ANNEX D: STEP5 Command Summary (arranged in alphabetical order)

The commands marked with * belong to the category of supplementary operations and are valid only in function blocks (FB/FX)!

STEP 5 command	Command group	STEP 5 command	Command group
!=D	comparison oper.	AN C	arithmetic oper.
!=F	"	AN D	"
!=G	"	AN F	"
)	logic oper.binary	AN I	"
+D *	system operation!	AN Q	"
+F	arithmetic oper.	AN T	"
+G	"	BE	block end
-D *	system operation!	BEC	"
-F	arithmetic oper.	BEU	"
-G	"	BLD	other operation
:F	"	C DB	block call
:G	"	CD C	counter operation
xF	"	CFW *	convert operation
xG	"	CSD *	"
<=D	comparison oper.	CSW *	"
<=F	"	CU C	counter operation
<=G	"	CX DX	block call
⊖	"	D *	decrement
⊖F	"	DED *	convert operation
⊖G	"	DEF *	"
=	setting operation	DI *	system operation!
=D *	memory operation	DO FW *	other operation
=F	"	DOC FX	block call
=I	"	DO = *	other operation
=Q	"	DO RS *	system operation!
⊗	comparison oper.	DO DW *	other operation
⊗F	"	DOU FX	block call
⊗G	"	DUD *	convert operation
⊗D	"	DUF *	"
⊗F	"	ENT *	other operation
⊗G	"	FDG *	convert operation
>D	"	FR = *	time/counter oper.
>F	"	FR C *	counter operation
>G	"	FR T *	time operation
A(logic oper.binary	G DB	generate DB
A = *	"	GFD *	convert operation
A C	"	GX DX	generate DX
A D	"	I *	increment
A F	"	IA *	other operation
A I	"	JC = *	jump operation
A Q	"	JC FB	block call
A T	logic oper.binary	JC OB	block call + jump
AW *	logic oper.digit.	JC PB	block call
ABR *	system operation!	JC SB	"
ACR *	"	JM = *	jump operation
ADD BN *	arithmetic oper.	JN = *	"
ADD DF *	"	JO = *	"
ADD KF *	"	JP = *	"
AN = *	"	JUR *	system operation!

STEP 5 command	Command group
JOS = *	jump operation
JZ = *	"
JU = *	"
JU FB	block call
JU OB	block call + jump
JU PB	block call
JU SB	"
L = *	load operation
L C	"
L DD	"
L DL	"
L DR	"
L DW	"
L FD	"
L FW	"
L FY	"
L IB	"
L ID	"
L IW	"
L KB	"
L KC	"
L KF	"
L KG	"
L KH	"
L KM	"
L KT	"
L KY	"
L OB	"
L OW	"
L PW	"
L PY	"
L QB	"
L QW	"
L RI *	"
L RJ *	"
L RS *	"
L RT *	"
L T	"
LB CB *	system operation!
LB CD *	"
LB CW *	"
LB GB *	"
LB GD *	"
LB GW *	"
LC C	load operation
LC T	"
LD = *	"
LIR *	system operation!
LRD *	"
LRW *	"
LW = *	load operation
LW CD *	system operation!
LW CW *	"
LW GD *	"

STEP 5 command	Command group
LW CW *	system operation!
LWD = *	load operation
MAB *	system operation!
MAS *	"
MBA *	"
MBR *	"
MBS *	"
MSA *	"
MSB *	"
NOP 0	zero operation
NOP 1	"
0	logic oper.binary
O("
O = *	"
O C	"
O D	"
O F	"
O I	"
O Q	"
O T	"
ON = *	"
ON C	"
ON D	"
ON F	"
ON I	"
ON Q	"
ON T	"
OW *	logic oper.digit.
R D	memory operation
R F	"
R I	"
R Q	"
RA *	other operation
RB = *	setting operation
RC	counter operation
RD = *	timer/counter op.
RLD *	shift function
RRD *	"
RT	timer operation
S = *	setting operation
S C	counter operation
S D	memory operation
S F	"
S I	"
S Q	"
SD T	timer operation
SE T	"
SEC= *	timer/counter op.
SEE *	other operation
SED *	"
SF T	timer operation
SFD= *	timer/counter op.
SLD *	shift function
SLW *	"

STEP 5 command	Command group
SP =	* shift function
SP T	"
SR =	* timer operation
SRW	* shift function
SS T	timer operation
SSD	* shift function
SSU=	* timer/counter op.
SSW	* shift function
STP	stop instruction
T =	* transfer operation
T DD	"
T DL	"
T DR	"
T DW	"
T ID	"
T IW	"
T FD	"
T FW	"
T FY	"
T OW	"
T OY	"
T PY	"
T PW	"
T QB	"
T QD	"
T QW	"
T RI	"
T RJ	"
T RS	* "
T RT	"
TAK	* other operation
TB CB	* system operation!
TB CD	* "
TB CW	* "
TB GB	* "
TB GD	* "
TB GW	* "
TIR	* "
TNB	* "
TNW	* "
TRD	* "
TRW	* "
TSC	* "
TSG	* "
TW CD	* "
TW CW	* "
TW GD	* "
TW GW	* "
XOW	* logic oper.digit.

ANNEX E: STEP5 Commands of the CPU 928
(arranged according to command code)

Explanations:

- Column 'command code':

The command code consists of three words max. that are either represented as a hexadecimal number or - in a few cases - as a bit pattern.

The bit positions containing the parameter are marked with the letters 'p' (1st parameter, e.g. byte address) and 'q' (2nd parameter, e.g. bit address).

Bit positions that are not decoded are marked with the letter 'x'.

- Column 'parameter range':

Contains the permitted range of values of the bit positions marked with the letters 'p' or 'q' in the command code.
All values stated are decimal.

- Column 'STEP5':

Contains the abbreviations for programming in STL (STEP5 mnemonic).

- Column 'remarks':

Changes with respect to the R processor:

P = parameter range increased by 128 counters/timers
N = new command

Command code	Parameter range	STEP5	Remark
<i>Word</i>			
---1 ---2 ---3			
00xx		NOP 0	
0100		CFW	
02pp	0-255	LT	P
03pp	0-255	TNB	
04pp	0-255	FRT	P
0500		BEC	
06pp	1-126	FR=	
07pp	1-126	A=	
0800		IA	
0880		RA	
0900		CSW	
0App	0-255	LFY	
0Bpp	0-255	TFY	
0Cpp	0-255	LCT	P
0Dpp	-128,+127	JO =	
0Epp	1-126	LD=	
0Fpp	1-126	O=	
10xx		BLD	
11pp	0-255	I	
12pp	0-254	LFW	
13pp	0-254	TFW	
14pp	0-255	SFT	P
15pp	-128,+127	JP =	
16pp	1-126	SFD=	
17pp	1-126	S=	
18pp	0-255	DO RS	
19pp	0-255	D	
1App	0-252	LFD	
1Bpp	0-252	TFD	
1Cpp	0-255	SET	P
1Dpp	0-255	JCFB	
1Epp	1-126	SEC=	P
1Fpp	1-126	=	
20pp	3-255	CDB	
2120		>F	
2140		<F	
2160		><F	
2180		!=F	
21A0		>=F	
21C0		<=F	
22pp	0-255	LDL	
23pp	0-255	TDL	
24pp	0-255	SDT	P
25pp	-128,+127	JM=	
26pp	1-126	SR=	
27pp	1-126	AN=	
28pp	0-255	LKB	
29pp	0-32	SLD	
2App	0-255	LDR	
2Bpp	0-255	TDR	
2Cpp	0-255	SST	P
2Dpp	-128,+127	JU=	
2Epp	1-126	SSU=	
2Fpp	1-126	ON=	

Command code	Parameter range	STEP5	Remark
<i>Word</i>			
---1 ---2 ---3			
3001 pppp	0-65535	LKC	
3002 pppp	"	LKT	
3004 pppp	"	LKF	
3010 pppp	"	LKS	
3020 pppp	"	LKY	
3040 pppp	"	LKH	
3080 pppp	"	LKM	
3120		>G	
3140		<G	
3160		><G	
3180		!=G	
31A0		>=G	
31C0		<=G	
32pp	0-255	LDW	
33pp	0-255	TDW	
34pp	0-255	SPT	P
35pp	-128,+127	JN=	
36pp	1-126	SP=	
37pp	1-126	RB=	
3800 pppp pppp	0-4294967295	LKG	
3920		>D	
3940		<D	
3960		><D	
3980		!=D	
39A0		>=D	
39C0		<=D	
3App	0-254	LDD	
3Bpp	0-254	TDD	
3Cpp	0-255	RT	P
3Dpp	0-255	JUFB	
3Epp	1-126	RD=	
3Fpp	1-126	LW=	
40pp	0-15	LIR	
4100		AW	
42pp	0-255	LC	P
43pp	0-255	TNW	
44pp	0-255	FRC	P
4500	-128,+127	JZ=	
46pp	1-126	L=	
47pp	0-255	LRJ	N
48pp	0-15	TIR	
4900		OW	
<i>Bit pattern (word1)</i>			
5432 1098 7654 3210			
0100 1010 0ppp pppp	0-127	LIB	
0100 1010 1ppp pppp	0-127	LQB	
0100 1011 0ppp pppp	0-127	TIB	
0100 1011 1ppp pppp	0-127	TQB	
<i>Word</i>			
---1 ---2 ---3			
4Cpp	0-255	LCC	P
4Dpp	1-255	JCOB	
4Epp	0-254	DO FW	
4Fpp	0-255	LRT	N

B8576633-01

Command code	Parameter range	STEP5	Remark
<i>Word</i>			
---1 ---2 ---3			
50pp	-128,+127	ADDBN	
5100		XOW	
<i>Bit pattern (word1)</i>			
5432 1098 7654 3210			
0101 0010 0ppp pppp	0-126	LIW	
0101 0010 1ppp pppp	0-126	LQW	
0101 0011 0ppp pppp	0-126	TIW	
0101 0011 1ppp pppp	0-126	TQW	
<i>Word</i>			
---1 ---2 ---3			
54pp	0-255	GDC	
55pp	0-255	JCPB	
56pp	1-125	LWD=	
57pp	0-254	LOW	
5800 pppp	-32768,+32767	ADDKF	
5900		-F	
<i>Bit pattern (word1)</i>			
5432 1098 7654 3210			
0101 1010 0ppp pppp	0-124	LID	
0101 1010 1ppp pppp	0-124	LQD	
0101 1011 0ppp pppp	0-124	TID	
0101 1011 1ppp pppp	0-124	TQD	
<i>Word</i>			
---1 ---2 ---3			
5Cpp	0-255	SC	P
5Dpp	0-255	JCSB	
5Fpp	0-255	LOB	
6000		:F	
6003		:G	
6004		xF	
6005 pppp pppp	-2147483648, +2147483647	ADDDF	N
6007		xG	
6008		ENT	
6009		-D	N
600B		-G	
600C xxpp	-128,+127	JOS=	
600D		+D	N
600F		+G	
61pp	0-15	SLW	
62pp	0-255	LRs	
63pp	0-255	TRs	
64pp	0-32	RLD	
6500		BE	
6501		BEU	
66pp	1-126	T=	
67pp	0-255	TRJ	N
6800 pppp	-32768,+32767	LRW	N
68p1	0-15	SSW	
6802		GFD	
6803 pppp	-32768,+32767	TRW	N

Command code	Parameter range	STEP5	Remark
<i>Word</i>			
---1 ---2 ---3			
6804 pppp	-32768,+32767	LRD	N
6805 pppp	-32768,+32767	TRD	N
6806		FDG	
6807		CSD	
6808		DUF	
680A		DUD	
680C		DEF	
680E		DED	
6819		MAS	N
6829		MAB	N
6849		MSA	N
6869		MSB	N
6889		MBA	N
6899		MBS	N
69pp	0-15	SRW	
6App	0-255	LRI	
6Bpp	0-255	TRI	
6Cpp	0-255	CUC	P
6Dpp	1-255	JUOB	
6Epp	0-255	DO DW	
6Fpp	0-255	TRT	P
7002		TAK	
7003		STP	
700B pppp	-32768,+32767	JUR	
71pp	0-32	SSD	
72pp	0-255	LPY	
73pp	0-255	TPY	
74pp	0-32	RRD	
75pp	0-255	JUPB	
76pp	1-125	DO=	
77pp	0-254	TOW	
7801 xxpp	0-255	DO FX	
7802 xxpp	0-255	DOCFX	
7803 xxpp	1-255	CXDX	
7804 xxpp	0-255	GXDX	
7805 xxpp	0-255	GDB	
7806 xxpp	0-31	SED	
7807 xxpp	0-31	SEE	
78p9 pppp	-32768,+32767	MBR	N
780A pppp	-32768,+32767	ABR	N
780D pppp	-32768,+32767	LBCB	N
780E pppp	-32768,+32767	LBGB	N
781D pppp	-32768,+32767	LBCW	N
781E pppp	-32768,+32767	LBGW	N
782D pppp	-32768,+32767	LBCD	N
782E pppp	-32768,+32767	LBGD	N
783D		ACR	N
783F 0qpp	0.0-255.15	AD	
783F 1qpp	0.0-255.15	OD	
783F 2qpp	0.0-255.15	AND	
783F 3qpp	0.0-255.15	OND	
783F 4qpp	0.0-255.15	SD	
783F 5qpp	0.0-255.15	RD	
783F 6qpp	0.0-255.15	=D	

Command code	Parameter range	STEP5	Remark
<i>Word</i>			
---1 ---2 ---3			
785D pppp	-32768,+32767	LWCW	N
785E pppp	-32768,+32767	LWGW	N
786D pppp	-32768,+32767	LWCD	N
786E pppp	-32768,+32767	LWGD	N
788D pppp	-32768,+32767	TBCB	N
788E pppp	-32768,+32767	TBGB	N
789D pppp	-32768,+32767	TBCW	N
789E pppp	-32768,+32767	TBGW	N
78AD pppp	-32768,+32767	TBCD	N
78AE pppp	-32768,+32767	TBGD	N
78CD pppp	-32768,+32767	TSC	N
78CE pppp	-32768,+32767	TSG	N
78DD pppp	-32768,+32767	TWCW	N
78DE pppp	-32768,+32767	TWGW	N
78ED pppp	-32768,+32767	TWCD	N
78EE pppp	-32768,+32767	TWGD	N
7900		+F	
7App	0-254	LPW	
7Bpp	0-254	TPW	
7Cpp	0-255	RC	P
7Dpp	0-255	JUSB	
7Epp		DI	
7Fpp	0-255	TOB	
<i>Bit pattern (word1)</i>			
5432 1098 7654 3210			
1000 0qqq pppp pppp	0.0-255.7	AF	
1000 1qqq pppp pppp	0.0-255.7	OF	
1001 0qqq pppp pppp	0.0-255.7	SF	
1001 1qqq pppp pppp	0.0-255.7	=M	
1010 0qqq pppp pppp	0.0-255.7	ANF	
1010 1qqq pppp pppp	0.0-255.7	ONF	
1011 0qqq pppp pppp	0.0-255.7	RF	
<i>Word</i>			
---1 ---2 ---3			
B8pp	0-255	AC	P
B9pp	0-255	OC	P
BAxx		A(
BBxx		O(
BCpp	0-255	ANC	P
BDpp	0-255	ONC	P
BFxx)	
<i>Bit pattern (word1)</i>			
5432 1098 7654 3210			
1100 0qqq 0ppp pppp	0.0-127.7	AI	
1100 0qqq 1ppp pppp	0.0-127.7	AQ	
1100 1qqq 0ppp pppp	0.0-127.7	OI	
1100 1qqq 1ppp pppp	0.0-127.7	OQ	
1101 0qqq 0ppp pppp	0.0-127.7	SI	
1101 0qqq 1ppp pppp	0.0-127.7	SQ	
1101 1qqq 0ppp pppp	0.0-127.7	=I	
1101 1qqq 1ppp pppp	0.0-127.7	=Q	

B8576633-01

Command code	Parameter range	STEP5	Remark
<i>Word</i>			
---1 ---2 ---3			
1110 0qqq 0ppp pppp	0.0-127.7	ANI	
1110 0qqq 1ppp pppp	0.0-127.7	ANQ	
1110 1qqq 0ppp pppp	0.0-127.7	ONI	
1110 1qqq 1ppp pppp	0.0-127.7	ONQ	
1111 0qqq 0ppp pppp	0.0-127.7	RI	
1111 0qqq 1ppp pppp	0.0-127.7	RQ	
<i>Word</i>			
---1 ---2 ---3			
F8pp	0-255	AT	P
F9pp	0-255	OT	P
FApp	-128,+127	JG=	
FBxx		0	
FCpp	0-255	ANT	P
FDpp	0-255	ONT	P
FFxx		NOP 1	

ANNEX F: STEP5 Commands Not Contained in the CPU 928

Please note that the following STEP5 commands of the S5-150U may not be processed in the CPU 928:

BAS		Command output inhibit
BAF		Command output enable
TB	I,Q,F,C,T,D,RI,RJ,RS,BT	Check bit for '1'
PN	I,Q,F,C,T,D,RI,RJ,RS,BT	Check bit for '0'
SU	I,Q,F,C,T,D,RI,RJ,RS,BT	Set bit unconditional
RU	I,Q,F,C,T,D,RI,RJ,RS,BT	Reset bit unconditional
LIM		Load command mask
SIM		Set interrupt mask
UBE		Interrupt block end
STW		Stop command for time interrupt processing
IAE		Disable addressing error interrupt
RAE		Enable addressing error interrupt
RAI		Enable user interrupt processing
IAI		Disable user interrupt processing

ANNEX G: Summary of the Program Level Identifiers

The identifiers correspond to those (hexadecimal) entered in the ISTACK under 'LEVEL' (EBENE).

- 0002H - Cold restart
- 0004H - Cycle
- 0006H - Time interrupt 5sec
- 0008H - Time interrupt 2sec
- 000AH - Time interrupt 1sec
- 000CH - Time interrupt 500ms
- 000EH - Time interrupt 200ms
- 0010H - Time interrupt 100ms
- 0012H - Time interrupt 50ms
- 0014H - Time interrupt 20ms
- 0016H - Time interrupt 10ms
- 0018H - not used
- 001AH - not used
- 001CH - Closed-loop control
- 001EH - not used
- 0020H - not used
- 0022H - not used
- 0024H - Process interrupt
- 0026H - not used
- 0028H - not used
- 002AH - not used
- 002CH - Abort
- 002EH - not used
- 0030H - Collision of two time interrupts
- 0032H - Controller fault
- 0034H - Cycle error
- 0036H - not used
- 0038H - Command code error
- 003AH - Execution time error
- 003CH - Addressing error
- 003EH - Acknowledgement delay (timeout)
- 0040H - not used
- 0042H - not used
- 0044H - Manual warm restart
- 0046H - Automatic warm restart

ANNEX H: Example: How to Evaluate the ISTACK

This (very simplified) example shows a possible method of how to evaluate the ISTACK. Also refer to Chapter 5.3 "Control bits and interrupt stack".

Starting point: The processor has aborted cyclic program execution and gone into the stop state.
 To find out the reason, we have to select the online function "Output ISTACK" at our programmer.

First the control bits are output by the PG:

```

>>STP<<  STP-6    FE-STP    BARBEND   PG-STP    STP-SCH    STP-BEF    MP-STP
          X
>>ANL<<  ANL-6    NEUST     M W A     A W A     ANL-2      NEU-ZUL    MWA-ZUL
          X                      X
>>RUN<<  RUN-6    EINPROZ   BARB      OB1GEL    FBOGEL     OBPROZA    OBWECKA
          X                      X
32KWRAM  16KWRAM   8KWRAM    EPROM     KM-AUS     KM-EIN     DIG-EIN    DIG-AUS
          X
URGELOE  URL-IA    STP-VER   ANL-ABB   UA-PG      UA-SYS     UA-PRFE    UA-SCH
DXO-FE   FE-22    MOD-FE    RAM-FE    DB0-FE     DB1-FE     DB2-FE     KOR-FE
N A U    P E U    B A U     STUE-FE   Z Y K      Q V Z      A D F      WECK-FE
B C F    FE-6     FE-5      FE-4      FE-3       L Z F      REG-FE     DOPP-FE
          X
    
```

The control bits give information about the current operating state of the processor (>>STP<<) as well as about other features such as OB 1 loaded, single-processor operation, 16KW user memory and so forth. In the upper line 'STP-BEF' is marked as the cause of the stop state. Since there is no STP command in our STEP5 user programs, the stop command may have been activated only by the system program, due to a missing error OB. In the lower line, the 'LZF' identification is marked. The system program might have recognized that there is no corresponding error organization block programmed when an execution time error occurs. Since there are various possible execution time errors and we do not know which of them it is, the information given by the control bits is not sufficient.

B8576633-01

We request the display of the ISTACK:

I N T E R R U P T S T A C K

DEPTH: 01

INS-REG: 0000	SAC: 0000	DB-ADD: 0000	BA-ADD: 0000
BLK-STP: 0001	-NO.:	DB-NO.:	-NR.:
LEVEL: 003A	REL-SAC:	DBL-REG: 0000	
	UAMK: 0120	UALW: 0000	
ACCU1: 0000 1A01	ACCU2: 0000 0000	ACCU3: 0000 0000	ACCU4: 0000 0000

RESULT BITS: DSP1 DSP0 OVFL OVFLS OR STATUS RLO ERAB ¹⁾

CAUSE OF INTERR.:	NAU	PEU	BAU	MPSTP	ZYK	QVZ	ADF	STP
								X
	BCF	S-6	LZF	REG	STUEB	STUEU	WECK	DOPP

Depth 01 of the ISTACK represents the program level which has been activated last before the processor went into the stop state. The identification '3A'H (next to LEVEL) indicates that this is the ISTACK of the program level EXECUTION TIME ERROR. Accu 1 contains the error identification '1A01'H. We may thus assume that an execution time error has occurred due to the fact that no data block was loaded after the command 'C DB'. Since the corresponding error OB 19 is not available in our user program, the system program has aborted any further program execution (STP). The cause of interruption is stored in the 'interrupt condition-code mask word' UAMK: the identification '0120'H corresponds to the bit pattern '0000 0001 0010 0000'. Bit 2⁵ (LZF) and bit 2⁸ (STP) are set.

Now we have to find out in which block and by which command the execution time error has been caused.

¹⁾ DSP1 and DSP0 are referred to as CC1 and CC0 in the S5-135U list of operations

B8576633-01

We switch the output of the ISTACK down to depth 02:

DEPTH: 02

INS-REG: 2006 SAC: 0037 DB-ADD: 0000 BA-ADD: 0000
BLK-STP: 0001 OB-NO.: 1 DB-NO.: -NO.:
LEVEL: 0004 REL-SAZ: 0004 DBL-REG: 0000 UALW: 0000
ACC1: 0000 0001 ACCU2: 0000 0000 ACCU3: 0000 0000 ACCU4: 0000 0000

RESULT BITS: DSP1 DSP0 OVFL OVFLS OR STATUS RLO ERAB

CAUSE OF INTERR.: NAU PEU BAU MPSTP ZYK QVZ ADF STP
BCF S-6 LZF REG STUEB STUEU WECK DOPP
X

The identification '04'H (next to LEVEL) indicates that this is the ISTACK of the interrupted program level CYCLE. The STEP address counter (SAC) points to the address '37'H. This is the absolute address (in the user memory) of the command that caused the error. The interruption occurred in organization block OB 1. In OB 1, the command that caused the error can be found under the relative address '04'H (REL-SAC). As we have already found out this was the command that led to an execution time error (see UAMK, bit 2⁵, and CAUSE OF INTERR.).

The incorrect command can now be displayed at the programmer using the online function "SEARCH". Therefore we need to enter the block in question (OB 1) and the relative address of the command.

F 1 ! F 2 ! F 3 ! F 4 ! F 5 ! F 6 ! F 7 ! F 8
SYMB.DISP! LIB.NO. ! SEARCH ! ! ! ! !

OUTPUT DEVICE: PC BLOCK: OB1 SEARCH: 4H

When the search has been performed, the PG displays the command "C DB 6". This is the command that has caused the interruption since there is no data block with number 6 in the user memory.

OB 1
SEGMENT 1 0000
0004 :C DB 6
0005 :
0006 :
0007 :
0008 :BE

Index

1

16-bit fixed point numbers; 2-5

3

32-bit fixed point numbers; 2-5; 2-6

A

Accumulators; 2-4; 3-10

 erase; 6-7

 scroll; 6-8

Accu 1 and accu 2

 evaluate; 5-4

Acknowledgement delay (QVZ); 5-35

Actual operands; 2-23; 2-24; 2-26; 3-40

Address assignment

 CPU 928; 8-2

 system RAM; 8-3

Addressing error; 5-34

Arithmetic operations; 3-22; 3-41

Automatic warm restart; 4-13

B

BASP (command output inhibit); 5-36; 10-13

BCD coded number; 2-8

Basic operations; 2-1

Binary numbers; 2-5

Bit condition codes; 3-11

 access to; 6-5

Block; 2-3; 2-9

 start address; 3-4

 call error; 5-30

Block address lists (DBO); 3-4

Block start addresses (DBO); 8-8

Block body; 2-11

Block call; 3-23

 conditional; 2-14; 2-26

 error; 5-30

 unconditional; 2-14; 2-26

Block header; 2-10; 8-7

Block type; 2-9

Block boundaries; 4-3

Block number; 2-11

Block parameter; 2-23; 2-24

Block pre-header; 2-11

Block stack (BSTACK); 3-5

 read; 6-12

 evaluate; 5-4

Block transfer; 6-22; 6-24

Blocks

 correct; 2-12

 delete; 2-12

BR register; 9-19

 load; 9-19

Byte (representation); 6-34

C

- COMPRESS memory; 11-10
- CONTROL VARIABLES; 11-9
- CONTROL; 11-9
- Code blocks; 1-4; 2-9
- Cold restart; 4-12
 - trigger; 4-9
- Collision of two time interrupts; 4-20; 4-21; 5-37
- Command code; E-1
- Command code error; 5-27
- Command boundaries; 4-3
- Communications processor (CP); 10-6
- Comparison operations; 3-14
- Complement
 - forming of; 3-46
- Configuration; 1-1
- Control bits
 - abbreviations; 5-8
 - evaluate; 5-7
- Controller
 - errors; 5-37
 - inputs and outputs; 6-73
 - interrupts; 4-23
 - parameters; 6-76
 - structure R64; 4-23
- Control system Flow chart; 2-2
- Conversion operations; 3-46
- Coordinator; 410-2; 10-5
 - start; 10-12
- Count value; 3-19
- Counter loops; 6-10
- Counter operations; 3-18; 3-38
- CPU identification; 8-26
- Cycle; 4-17
 - interrupts; 4-18
- Cycle time; 1-3; 3-7; 5-36
 - error; 5-36
 - restart; 6-49
 - set; 6-49

D

- DB0 error; 5-22
- DB1 error; 5-22
- DB2 error; 5-24
- DBA register; 9-7
- DBL register; 9-10
- DX0 structure; 7-2
- DX0 error; 5-25
- Data
 - format; 2-32
 - technical; 11-1
- Data block RAM (DB-RAM); 6-57
- Data block DB 0; 2-35
- Data block DB 1; 2-35; 10-2; 10-9
- Data block DB 2; 2-36
- Data block DX 0; 2-36; 7-1
- Data blocks; 1-4; 2-9; 2-10; 2-31
 - access to; 2-33; 6-16
 - alter; 2-31
 - structure; 2-31

- create; 3-52
- duplicate; 6-30
- open; 2-33
- program; 2-32
- shift; 6-29
- test; 6-20
- transfer; 6-29
- validity range; 2-34
- Data transfer; 10-4; 10-5; 10-6
 - protected; 10-8
- Decimal numbers; 2-5
- Decrement; 3-48
- Digital outputs; 10-10
- Digital inputs; 10-10
- DSP1 and DSP0; 3-12
- Double addressing; 10-6
- Double word (representation); 6-34
- Double word conversion; 3-46

E

- EPROM submodule; 2-3; 3-6
- ERAB (first scan); 3-11; 3-13
- Error causes;
 - search; 5-6
- Error diagnosis; 5-2
- Error handling
 - using organization blocks; 5-18
- Error identifiers
 - summary; B-1
- Error LEDs; 5-2
- Error nesting; 5-12
- Error organization blocks
 - interruptions; 5-20
- Execution time; 3-7
- Execution time error
 - others; 5-32
- Exponent; 2-6
- Exponential numbers; 2-6
- Expansion units; 3-17

F

- Fixed point conversion; 3-46
- Fixed point-floating point conversion; 3-47
- Flag bytes
 - rewrite; 6-24
 - transfer; 6-22
- Floating point numbers; 2-6
 - input; 2-7
- Formal operands; 2-22
- Function block FB 0, 2-30
- Function blocks; 2-10; 2-19
 - alter; 2-25
 - call; 2-26
 - structure; 2-20
 - parameter assignment; 2-26
 - program; 2-22
- Functions, new; 1-7

B8576633-01

G

Graph 5; 2-2

H

Handling blocks; 6-56

I

I/O; 3-17; 8-5

access; 8-6

addresses; 6-33; 8-4

bus; 10-7

modules; 10-9; 10-11

Increment; 3-48

Initialization; 5-21

errors; 5-21

Intermediate flags; 4-27

Interprocessor communication flags; 3-7; 10-4

transfer; 6-50; 10-7

Interrupt condition code erase word (UALW); 8-18

Interrupt condition code word (collected) (UAMW); 8-19; 8-20

Interrupt lines; 4-24

Interrupt stack (ISTACK);

abbreviations; 5-12

evaluate; 5-12

evaluate (example); 5-16; H-1

Interruption causes; 5-21; 5-26; 5-27

Interruption location; 5-7

Interruptions of error OBs; 5-20

IPC

input flags; 3-8; 10-5; 10-10

output flags; 3-8; 10-5; 10-10

J

Jump operations; 3-43

L

Ladder diagram (LAD); 2-2

Load operations; 3-14; 3-39

Logic operations

binary; 3-13; 3-37

digital; 3-42

Loop counter; 6-10

M

MEMORY EXTENSION; 11-11

Mantissa; 2-6

Manual restart; 4-12

Memory access

absolute; 9-1

Memory areas; 1-5

Memory blocks

transfer; 9-13

Memory operations; 3-14; 3-37

Memory organization; 9-1

Multiprocessor operation; 10-1; 10-9

start-up; 10-12

Multiprocessor communication; 10-8

B8576633-01

N

Nesting depth; 1-7; 3-4
No operations; 3-23
Normalized fixed-point number; 6-74; 6-78
Numerical representation; 2-5

O

O I/O's; 3-17
OR; 3-11
OS (overflow, latching); 3-12
OUTPUT ADDRESS; 11-11
OV (overflow); 3-12
Online functions; 11-1
Operand; 2-4
Operating modes
 Overview; 4-2
 RUN; 4-16
 START-UP; 4-9
 STOP; 4-6
 WAIT MODE; 11-1
Operation; 2-4
Operation code error; 5-28
Organization blocks; 2-9
 call; 2-14
 program; 2-13
 special functions; 2-18
 special; 2-16
 structure; 2-13
Output BSTACK; 5-4
Output ISTACK; 5-2
Output bytes
 control; 11-9
Overall reset
 execute; 4-8
 request; 4-7

P

P controller; 6-76
P I/O's; 3-17
PC OVERALL RESET, 11-11
PD controller; 6-76
PI controller; 6-76
PID algorithm; 6-67; 6-72
PID controller; 6-65
 abbreviations; 6-77
 transfer data blocks; 6-69
PROCESSING CONTROL; 11-5
Pages; 6-32
Page frame
 access; 9-24
Parameter error; 5-29
Power failure (NAU); 4-10; 4-13; 4-14; 10-12
 during start-up; 4-14
Power return; 4-10; 4-13; 10-12
Process image; 3-17; 5-34; 5-35; 8-6
 of inputs; 3-7
 of outputs; 3-7

Process interrupt
 delay; 6-44
 inhibit; 3-52; 4-26; 6-44
 edge triggered; 4-25
 release; 3-52; 4-26
 interruptions; 4-24
 level triggered; 4-25
Processing operations; 3-50
Program (processing)
 controller; 4-23
 cyclic; 3-6; 4-17
 manipulate; 3-9
 interrupt driven; 4-24
 time driven; 4-18
Program blocks; 2-9
 call; 2-14
 program; 2-13
 structure; 2-13
Program examples; 3-24
Program levels
 characteristics; 4-3
 identifiers; G-1
 priority; 4-3
 summary; 4-2
Program organization; 3-1
Pseudo command boundaries; 6-1; 9-14

R

RAM submodule; 3-5
RI area (interface data area); 8-12
RJ area; 8-12
RS 3 and RS 4
 evaluate; 5-3
RS area; 8-12
RT area; 8-12
RUN
 errors; 5-26
Register; 1-6
 access; 9-5
 shift; 9-20
Register assignment; 9-2
Remaining cycle; 4-12; 4-14; 4-16
Response time; 4-27
Result bits; 3-11
 access; 6-5
Result of logic operation (RLO); 2-14; 2-15; 3-11; 3-13
Return address; 3-4

S

SAC (step address counter); 9-11
START-UP
 errors; 5-21; 5-26
 interruptions; 4-14
 synchronization; 10-12
 trigger; 4-9
STATUS VARIABLES; 11-3
STEP5 operations; 2-4
 binary; 3-10
 digital; 3-10
 organizational; 3-10
STEP5 total operation set; C-1
STOP LED
 permanently lit; 4-6
 flashing quickly; 4-7
 flashing slowly; 4-7
Sampling time; 4-23
Sampling time error; 5-39
Scratchpad flags; 4-27
Semaphores
 application example; 3-56
 enable; 3-53
 set; 3-53
Sequence blocks; 2-10
 call; 2-14
 program; 2-13
 structure; 2-13
Shift operations; 3-44
Shift register; 6-57
 erase; 6-64
 initialize; 6-60
 process; 6-63
Sign; 2-6; 2-8
 extension; 6-43
Slots; 10-2
Slot identification; 8-25
Special functions; 6-1
 error; 6-1
 overview; 6-3
Standard function blocks; 2-29
Start-up modes
 compare; 6-50
Statement list (STL); 2-2
Status (STA); 3-11
Stop point; 11-4; 11-5
Stop state; 4-6
 leave; 4-8
Stop instruction; 3-23
Substitution error; 5-28
Supplementary operations; 2-1; 3-37
System checkpoint; 11-1
System data assignment; 8-13
System data words RS3 and RS4
 evaluate; 5-3
System operations; 2-1; 3-37
System program; 1-4; 2-3
 read checksum; 6-52

T

Technical data S5-135U; A-1
Test operation; 10-13
 special features; 10-13
 trigger; 10-13
Time base; 3-19
Time interrupt
 prioritize; 4-19
 inhibit; 6-44
 interruptions; 4-20
 delay; 6-43
Time value; 3-19
Timer block length; 10-10; 10-11
Timer operations; 3-18; 3-38;
Total cycle time; 3-7
Transfer error; 5-31
Transfer operations; 3-14; 3-39

U

User checkpoints; 11-1
User interface; 3-9
User memory; 8-7
User program; 1-4; 2-3

W

Wait state
 features; 11-2
Word (representation); 6-34
Word condition codes; 3-11; 3-12
 access; 6-5

List of Figures, Examples and Summaries

Chapter 1

Fig.: Typical S5 135U system structure.....1-2

Chapter 2

Fig.: Programming language STEP5 - methods of representation.....2-2
 Fig.: Block storage the in program memory.....2-11
 Fig.: Structure of an organization, program and sequence block...2-13
 Fig.: Block calls that enable processing of a program block.....2-14
 Summary: Special-function organization blocks in the CPU 928.....2-18
 Fig.: Structure of a function block (FB/FX).....2-20
 Example: Programming a function block.....2-25
 Fig.: Calling a function block and assignment of parameters.....2-27
 Fig.: Structure of a data block.....2-32
 Fig.: Opening data blocks and accessing data words.....2-34
 Fig.: Validity range of a data block after it has been called....2-35

Chapter 3

Example: Organization of the user program according to the
 program structure.....3-2
 Example: Organization of the user program according to the
 system structure.....3-3
 Example: Block nesting depth and block stack.....3-5
 Fig.: Blocks in the program memory.....3-6
 Fig.: Cyclic program execution.....3-8

Chapter 4

Summary: Operating states and program levels.....4-2
 Fig.: Interrupt-driven program execution at block boundaries....4-27

Chapter 5

Fig.: The structure of the ISTACK in an example.....5-16

Chapter 6

Fig.: Schematic diagram of the shift register with 3 pointers
 and 12 storage locations.....6-57
 Fig.: Schematic diagram of the shift register with 3 pointers
 and 12 storage locations before the first clock pulse.....6-58
 Fig.: Schematic diagram of the shift register with 3 pointers
 and 12 storage locations after the first clock pulse.....6-59
 Fig.: Structure of the data block for the initialization of a
 shift register.....6-60
 Fig.: Block diagram of the PID controller.....6-65

Chapter 7

Fig.: Structure of DX 0.....7-2

Chapter 8

Fig.: Address distribution in the CPU 928.....8-2
 Fig.: Address distribution - system RAM (16 bits).....8-3
 Fig.: Address distribution - I/O (8 bits).....8-4
 Fig.: Address distribution for I/O/programming.....8-5

Chapter 9

Fig.: Global memory and local memory.....9-2
 Fig.: Contents of the accumulators during program execution.....9-12
 Example: Clearing all flag bytes (FY0 to FY255).....9-12

SIEMENS

SIMATIC S5

Multiprocessor Communication
S5-135U, CPU 922 (R Processor),
CPU 928 and CPU 928B
S5-155U, CPU 946/947

User's Guide

C79000-B8576-C468-05

	Page
1	Introduction 3
1.1	Configuration..... 4
1.2	Principle 4
1.3	Sender/Receiver Identification..... 5
1.4	Buffering the Data 6
1.5	System Restart 9
1.6	Calling and Nesting the Special Function Organization Blocks OB 200 and OB 202 to OB 205 10
1.7	Parallel Processing in a Multiprocessor Programmable Controller..... 11
1.8	Required Memory Areas 11
1.9	Runtime..... 12
2	Parameter Assignment 14
2.1	Evaluating the Output Parameters 14
2.1.1	Condition Codes..... 15
2.1.2	Condition Code Byte: Initialization Conflict/Error/Warning 16
3	INITIALIZE Function (OB 200) 21
3.1	Input Parameters 23
3.1.1	Mode (Automatic / Manual) 23
3.1.2	Number of CPUs 24
3.1.3	Block ID and Number / Start Address of the Assignment List..... 24
3.2	Output Parameters 26
3.2.1	Condition Code Byte 26
3.2.2	Total Capacity 28
4	SEND Function (OB 202) 29
4.1	Input Parameters 29
4.1.1	Receiving CPU 29
4.1.2	Block ID and Number / Field Number 29
4.2	Output Parameters 31
4.2.1	Condition Code Byte 31
4.2.2	Transmitting Capacity 33

5	SEND TEST Function (OB 203)	34
5.1	Input Parameters	34
5.1.1	Receiving CPU.....	34
5.2	Output Parameters	34
5.2.1	Condition Code Byte	34
5.2.2	Transmitting Capacity	35
6	RECEIVE Function (OB 204)	36
6.1	Input Parameters	36
6.1.1	Transmitting CPU.....	36
6.2	Output Parameters	37
6.2.1	Condition Code Byte	37
6.2.2	Receiving Capacity	38
6.2.3	Block ID and Number.....	38
6.2.4	Address of the First/Last Received Data Word.....	39
7	RECEIVE TEST Function (OB 205)	40
7.1	Input Parameters	40
7.1.1	Transmitting CPU.....	40
7.2	Output Parameters	40
7.2.1	Condition Code Byte	40
7.2.2	Receiving Capacity	41
8	Applications	42
8.1	Calling the Special Function OB Using Function Blocks	42
8.1.1	Setting Up a Buffer (FB 200).....	43
8.1.2	Sending a Block of Data (FB 202)	45
8.1.3	Testing the Transmitting Capacity (FB 203).....	47
8.1.4	Receiving a Block of Data (FB 204).....	48
8.1.5	Testing the Receiving Capacity (FB 205)	50
8.2	Transferring Data Blocks	51
8.2.1	Functional Description.....	51
8.2.2	Transferring a Data Block (FB 110)	51
8.2.3	Application Example (for the S5-135U).....	54
8.3	Extending the IPC Flag Area	56
8.3.1	The Problem.....	56
8.3.2	The Solution	57
8.3.3	Data Structure	57
8.3.4	Program Structure.....	60
8.3.5	Sending Data Word Areas (FB 100)	62
8.3.6	Receive Data Word Areas (FB 101)	65
8.3.7	Application Example (for S5-135U).....	68

1 Introduction

You can operate the multiprocessor programmable controllers S5-135U and S5-155U with up to four CPUs. You can use the following "tools" individually or in combination to exchange data between the CPUs:

- F flags are transferred, if you define them as interprocessor communication (IPC) output flags in **one** CPU and as IPC input flags in one or more CPUs.
- To transfer data blocks, or to be more precise, blocks of data with a maximum length of 64 bytes (= 32 data words), you can use the following special functions that are integrated in the CPU:

INITIALIZE (OB 200) :	preassign
SEND (OB 202):	send a block of data
SEND TEST (OB 203):	test sending capacity
RECEIVE (OB 204):	receive a block of data
RECEIVE TEST (OB 205):	test receiving capacity

To use these functions, you only require basic knowledge of the STEP 5 programming language and the way in which SIMATIC S5 programmable controllers operate. You can obtain this basic information from the publications listed in the table of documentation.

Whereas the IPC flags are updated "automatically" by the system program, you must call the **INITIALIZE, SEND, SEND TEST, RECEIVE and RECEIVE TEST** functions as special function organization blocks using the **JU OB** or **JC OB** operations.

1.1 Configuration

S5-135U or S5-155U

These PLCs contain the S5 bus and in the multiprocessor mode they also have the following components:

- **1 coordinator 923C**

This module contains four pages. These are memory areas of 1024 bytes. They all occupy the address area F400H to F7FFH. You select (address) the "current" page using the select register (also known as the identification or page address register, similar to chip select). The select numbers 252, 253, 254 and 255 are **fixed** as the four pages of the 923C coordinator and are used for multiprocessor communication.

- **2 to 4 CPUs**

For the S5-135U: CPU 922 (R processor), CPU 928 , CPU 928B or CPU 920 (M processor)

For the S5-155U: CPU 946/947, CPU 922 (R processor), CPU 928 , CPU 928B or CPU 920 (M processor).

These CPUs can exchange data with each other in any combination, you can also use "handling blocks" which also work with page addressing without any restrictions.

If you have one or more additional CPU 921s (S processors) in the same rack, they **cannot** take part in the multiprocessor communication. You must not call the S processor handling blocks as long as R and M processors, CPU 928s, CPU 928Bs and CPU 946/947s are processing their handling blocks or are involved in multiprocessor communication. CPUs can, however, always communicate via IPC flags.

1.2 Principle

To transfer data, you must activate the SEND function on the transmitting CPU and the RECEIVE function on the receiving CPU.

The data words of a DB or DX data block located in the transmitting CPU are transported via the coordinator 923C to the receiving CPU one after the other and written to the DB or DX data block with the same number and under the same data word address; i.e. this represents a "1:1" copying.

Example

	Data to be sent in the transmitting CPU	Data received in the receiving CPU
Data block	DB 17	DB 17
Data word address	DW 32 to DW 63	DW 32 to DW 63

The amount of data that can be transferred with the SEND and RECEIVE functions is normally 32 words.

If the block length (without header) is not a multiple of 32 words, the last block of data to be transferred is an exception and is less than 32 words.

The data block in the receiving CPU can be longer or shorter than the data block to be sent. It is, however, important that the data words transferred by the SEND function exist in the receiving block; otherwise the RECEIVE function signals an error.

1.3 Sender/Receiver Identification

The CPUs are numbered so that the leftmost CPU has the number 1 and each subsequent CPU to the right has a number increased by 1.

Example

S5-135U/155U:

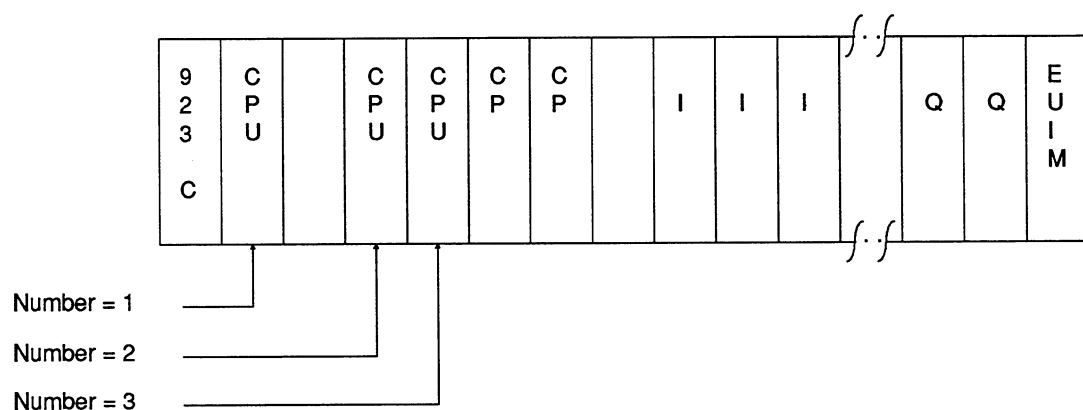


Fig. 1 Sender/receiver identification

Example

The following table indicates a possible data transfer sequence assuming that the connection "from CPU 3 to CPU 2" has seven memory fields assigned by the INITIALIZE function.

Transmitter: CPU 3		Receiver: CPU 2		
Sequence	Transmitting capacity (free memory fields)	Receiving capacity (occupied memory fields)	Sequence	
		Time ↓		
Sends a blocks of data (A)	7		0	Initialize
Sends four blocks of data (B,C,D,E)	6		1	
	2		5	Receives two blocks of data (A, B)
Sends four blocks of data (F,G,H,I)	4		3	
	0		7	Receives five blocks of data (C,D,E,F,G)
Sends two blocks of data (K,L)	5		2	Receives two blocks of data (H,I)
	5		2	

REMEMBER



Sending/receiving n data blocks means that the corresponding function is called n times.

To simplify the representation, at any one time, data can either be sent or received in this example.

It is, however, possible and useful to transmit (CPU 3) and receive (CPU 2) simultaneously (see "Parallel processing in a multiprocessor programmable controller"). In the example, blocks H and I are received while blocks K and L are sent.

The example illustrates the queue organization of the buffer; the blocks of data sent first (A,B,C...) are received first (A,B,C...).

Summary

Buffering data on the coordinator 923C allows the asynchronous operation of transmitting and receiving CPUs and compensates for their different processing speeds.

Since the capacity of the buffer is limited, the receiver should check "often" and "regularly" whether there are data in the buffer (RECEIVE TEST function, receiving capacity > 0) and should attempt to fetch stored data (RECEIVE function). Ideally, the RECEIVE function should be repeated until the receiving capacity is zero. This means that the transmitted data are not buffered for a longer period of time and that the receiver always has the current data. This also means that memory fields remain free (the transmitting capacity is increased) and prevents the sender from being blocked (i.e. when the transmitting capacity is zero).

A receiving capacity of zero represents the ideal state (i.e. all transmitted data have been fetched by the receiver), on the other hand a transmitting capacity of zero indicates incorrect planning, as follows:

- the SEND function is called too often.
- the RECEIVE function is not called often enough,
- there are not enough memory fields assigned to the connection. The capacity of the buffer is insufficient to compensate temporary imbalances in the frequency with which the CPUs transmit and receive data.

1.5 System Restart

If you require multiprocessor communication, then all the CPUs involved must go through the **same** STOP-RUN transition (= RESTART), i.e. all the CPUs go through a COLD RESTART or all CPUs go through a WARM RESTART.

You must make sure that the restart of at least all the CPUs involved in the communication is **uniform** (see Chapter 10), in the following ways:

- **direct operation** (front switch, programmer),
- **parameter assignment** (DX 0) and/or
- **programming** (using the special function organization block OB 223 "stop if non-uniform restarts occur in the multiprocessor mode").

COLD RESTART

In organization block OB 20 (COLD RESTART) **one** CPU must set up the buffer (in the 923C) using the INITIALIZE function. Any existing data is lost.

Following this, i.e. during the RESTART, you can call the SEND, SEND TEST, RECEIVE, RECEIVE TEST functions in the individual CPUs. With appropriate programming, you must make sure that this only occurs after the buffer in the coordinator has been correctly initialized.

On completion of the RESTART, i.e. in the RUN mode, the user program is processed from the beginning, i.e. from the first operation in OB 1 or FB 0.

WARM RESTART

You must **not** use the INITIALIZE function in the organization blocks OB 21 (MANUAL WARM RESTART) and OB 22 (AUTOMATIC WARM RESTART). Calling the SEND, SEND TEST, RECEIVE, RECEIVE TEST functions can cause problems (refer to the following section).

On completion of the WARM RESTART, i.e. in the RUN mode, the user program is not processed from the start, **but** from the point at which it was interrupted. The point of interruption can, for example, be within the SEND function.

1.6 Calling and Nesting the Special Function Organization Blocks OB 200 and OB 202 to OB 205

The simplest procedure is as follows:

- program the call for the INITIALIZE function only in the cold restart organization block OB 20;
- program the call for the SEND, SEND TEST, RECEIVE, RECEIVE TEST functions either **only** within the cyclic program **or only** within the time-driven program.

REMEMBER



Depending on the assignment of parameters in DX 0 ("interrupts at command boundaries" for the CPU 928B, CPU 928 and CPU 920, or "155U mode" for the CPU 946/947), and the type of program execution (WARM RESTART, interrupt handling, e.g. OB 26 for cycle time error) it is possible that one of the functions INITIALIZE, SEND, SEND TEST, RECEIVE and RECEIVE TEST can be interrupted. If a user interface inserted at the point of interruption (e.g. OB 13 when interrupts are possible at operation boundaries or OB 22 following power down) also contains one of the functions SEND, SEND TEST, RECEIVE and RECEIVE TEST an illegal call (double call) is recognized and an error is signalled (error number 67, Section 2.1.2).

1.7 Parallel Processing in a Multiprocessor Programmable Controller

Once you have completed the assignment of the buffer (INITIALIZE function), you can execute the functions SEND, SEND TEST, RECEIVE and RECEIVE TEST in any combination and with any parameter assignment in all the CPUs simultaneously and parallel to each other.

Taking a single connection (from CPU 'SE' to CPU 'RE') it is possible to execute the SEND function (CPU 'SE') and the RECEIVE function (CPU 'RE') simultaneously. While CPU 'SE' is sending blocks of data to the coordinator, CPU 'RE' can receive (fetch) buffered blocks of data from the coordinator.

1.8 Required Memory Areas

The special function organization blocks OB 200 and OB 202 to OB 205 do not require a working area (e.g. for buffering variables) and do not call data blocks. They do, of course, access areas containing parameters, although only the parameters marked as output parameters are modified. These OBs also affect the condition codes (CC1, RLO etc., see Section 2.1).

- CPU 922, CPU 928, CPU 928B: The contents of ACCU 1 to ACCU 4 and the contents of the registers are not affected by the special function OBs for multiprocessor communication.
- CPU 946/947: The contents of all registers and ACCU 1, 2 and 3 remain the same, only the contents of ACCU 4 are affected.

1.9 Runtime

Special function OB		Runtime			
Block / name	Block function	R processor	CPU 928	CPU 946/947	CPU 928B
OB 200 / initialize	Assign buffer	230 ms	130 ms	128 ms	130 ms
OB 202 / send	Send a block of data (32 data words)	806 μ s (294 μ s basic time + 16 μ s / word); 118 μ s if a warning occurs	666 μ s (250 μ s basic time + 13 μ s / word); 115 μ s if a warning occurs	762 μ s (426 μ s basic time + 21 μ s / double word); 243 μ s if a warning occurs	696 μ s (280 μ s) basic time + 31 μ s / word); 145 μ s if a warning occurs
OB 203 / send test	Test transmitting capacity	72 μ s	50 μ s	207 μ s	80 μ s
OB 204 / receive	Receive a block of data (32 data words)	825 μ s (281 μ s basic time + 17 μ s / word); 115 μ s if a warning occurs	660 μ s (244 μ s basic time + 13 μ s / word); 98 μ s if a warning occurs	772 μ s (421 μ s basic time + 22 μ s / double word); 243 μ s if a warning occurs	690 μ s (274 μ s basic time + 13 μ s / word); 128 μ s if a warning occurs
OB 205 / receive test	Test receiving capacity	70 μ s	48 μ s	223 μ s	78 μ s

The "runtime" is the processing time of the special function organization blocks; the time from calling a block to its termination can be much greater if it is interrupted by higher priority activities (e.g. updating timers, processing closed loop controllers etc.).

The runtimes listed above assume that of four CPUs inserted in a rack, only the CPU whose runtimes are being measured accesses the SIMATIC S5 bus. If other CPUs use the bus intensively, the runtime increases particularly for the send/receive functions.

An important factor of a connection (from CPU 'SE' to CPU 'RE') is the total data transfer time. This is made up of the following components:

- time required to send (see runtime)
- length of time the data are buffered (on the 923C coordinator)
- the time required to receive data (see runtime)

The length of time that the data are "in transit" is largely dependent on the length of time that the data is buffered and therefore on the structure of the user program (see "Buffering Data").

2 Parameter Assignment

The "actual" parameters are located in a maximum 10 byte long data field in the F flag area. The number of the first flag byte in the data field (= pointer to the data field) must be loaded in ACCU-1-L. Permitted values are 0 to 246.

The data field is divided into an area for **input parameters** and an area for **output parameters**.

- **Input parameters**

All or part of the input parameters are read and evaluated by the functions, the functions do not write to this area.

- **Output parameters**

Some or all of the output parameters are written to by the functions, the functions do not read this area.

REMEMBER



You can assign a **flag area with 10 flag bytes** for all communications functions. The functions themselves require different numbers of bytes. Refer to the description of the single functions (Chapters 3 to 7).

Example: data field with parameters for the RECEIVE function (OB 204)

FY x + 0:	transmitting CPU	input parameter
FY x + 1:	—	not used
FY x + 2:	condition code byte	output parameter
FY x + 3:	receiving capacity	output parameter
FY x + 4:	block ID	output parameter
FY x + 5:	block number	output parameter
FY x + 6:	address of the first	output parameter
FY x + 7:	received data word	output parameter
FY x + 8:	address of the last	output parameter
FY x + 9:	received data word	output parameter

This example illustrates that the number of the first F flag byte in the data field must not be higher than FY 246, since otherwise the parameter field of up to 10 bytes would exceed the limits of the flag area (FY 255).

2.1 Evaluating the Output Parameters

Output parameters are data made available to the user program for evaluation. Among other things, they indicate whether or not a function could be executed and if not they indicate the reason for the termination of the function.

2.1.1 Condition Codes

The INITIALIZE, SEND, SEND TEST, RECEIVE and RECEIVE TEST functions affect the condition codes (see programming instructions for your CPUs, general notes on the STEP 5 operations):

- the OV and OS bits (word condition codes) are always cleared,
- the OR, STA, ERAB bits (bit condition codes) are always cleared,
- RLO, CC 0 and CC 1 indicate whether a function has been executed correctly and completely.

RLO = 0: Function executed correctly and completely

RLO = 1: Function aborted: the pointer to the data field in the flag area may have an illegal value, i.e. the low word of the ACCU contains a value greater than 246.

In the following sections, it is assumed that the pointer to the data field contains a correct value. The first byte of the output parameter provides detailed information about the cause of termination.

CC 1 = 1: Additional warning information (warning number 1 or 2)

CC 0 = 1: Additional error indication (error number 1-9)

Situation	Condition codes			Evaluation with operation
	RLO	CC 1	CC 0	
Function executed completely and correctly	0	0	0	JC =
Function aborted, pointer to data field illegal	1	0	0	JC =
Function aborted owing to an initialization conflict	1	0	0	JC =
Function aborted owing to an error	1	0	1	JC = and JM =
Function aborted owing to a warning	1	1	0	JC = and JP =

2.1.2 Condition Code Byte: Initialization Conflict/Error/Warning

The first byte in the field of the output parameters (condition code byte) also indicates whether or not a function has been correctly and completely executed. This byte contains detailed information about the cause of termination of a function.

Assuming that at least the pointer to the data field contains a correct value, this byte is **always** relevant.

If the function has been executed correctly and completely, all the bits are cleared (= 0), and all other output parameters are relevant.

If the function is aborted with a warning (bit $2^7 = 1$), only the condition code for the transmitting/receiving capacity is relevant, other output parameters (if they exist) are unchanged.

If the function is aborted owing to an error (bit $2^6 = 1$) or an initialization conflict (bit $2^5 = 1$), all other output parameters remain unchanged.

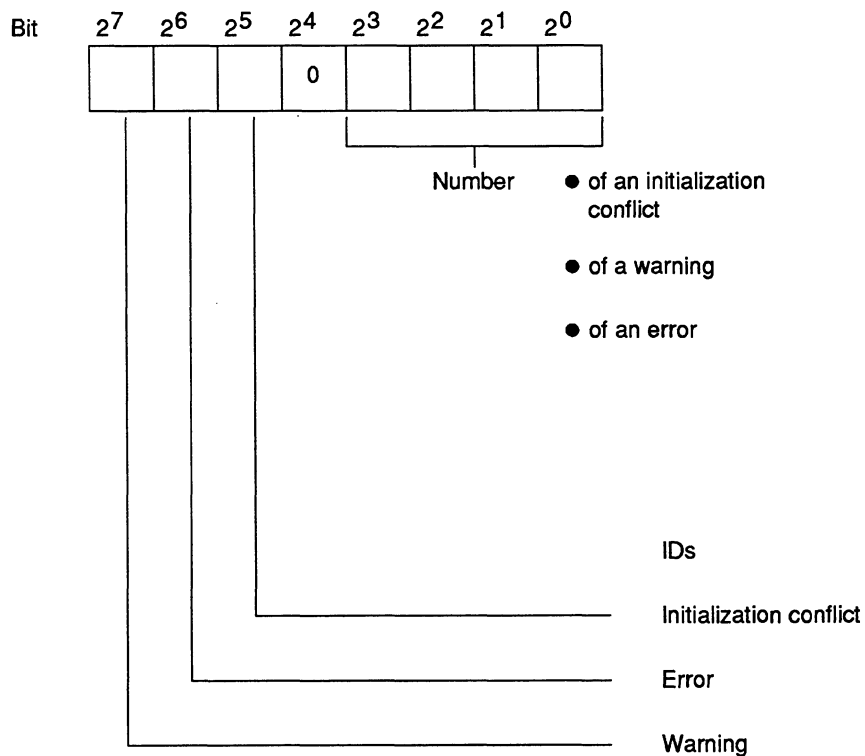


Fig. 2 Coding of the first byte

Evaluation

The identifiers in bit positions 2^5 to 2^7 indicate the significance of the numbers in bit positions 2^0 to 2^3 .

Apart from this bit-by-bit evaluation, it is also possible to interpret the whole condition code byte as a fixed point number without sign. If you interpret the condition code byte as a **byte**, the groups of numbers have the following significance:

Number group	Significance
0	Function executed correctly and completely
33 to 42	Function aborted owing to an initialization conflict
65 to 73	Function aborted owing to an error
129 to 130	Function aborted owing to a warning

The values of the following numbers **also** indicate the **order** in which errors or initialization conflicts were recognized and indicated by the functions.

Example

The SEND function indicates an error and is not executed. If you then make program and/or parameter modifications and the SEND function once again indicates an error with a higher number than previously, you can assume that you have corrected one of several errors.

Initialization conflict

An initialization conflict can only occur with the INITIALIZATION function. If a conflict occurs, you must modify the program or parameters.

Initialization conflict numbers (evaluation of the condition code byte as a byte)

- (33) The pages required for multiprocessor communication (numbers 252 to 255) are not or not all available.
- (34) The pages required for multiprocessor communication (numbers 252 to 255) are defective.
- (35) The parameter "automatic/manual" is illegal. The following errors are possible:
 - The "automatic/manual" ID is less than 1.
 - The "automatic/manual" ID is greater than 2.

- **(36)** The parameter "number of CPUs" is illegal. The following errors are possible:
 - The number of CPUs is less than 2.
 - The number of CPUs is greater than 4.

- **(37)** The parameter "block ID" is illegal. The following errors are possible:
 - The block ID is less than 1.
 - The block ID is greater than 2.

- **(38)** The parameter "block number" is illegal, since it is a data block with a special significance. The following errors are possible:
 - If block ID = 1 : DB 0, DB 1, DB 2
 - If block ID = 2 : DX 0, DX 1, DX 2

- **(39)** The parameter "block number" is incorrect, since the data block does not exist.
- **(40)** The parameter "start address of the assignment list" is too high or the data block is too short.
- **(41)** The assignment list in the data block is not correctly structured.
- **(42)** The sum of the assigned memory fields is greater than 48.

Errors

If an error occurs, you must change the program/parameters.

Error numbers (evaluation of the condition code byte as a byte)

- (65) The parameter "receiving CPU" (SEND, SEND TEST) is illegal, since it is a data block with a special significance. The following errors are possible:
 - The number of the receiving CPU is greater than 4.
 - The number of the receiving CPU is less than 1.
 - The number of the receiving CPU is the same as the CPU's own number.

- (66) The parameter "transmitting CPU" (RECEIVE, RECEIVE TEST) is illegal, since it is a data block with a special significance. The following errors are possible:
 - The number of the transmitting CPU is greater than 4.
 - The number of the transmitting CPU is less than 1.
 - The number of the transmitting CPU is the same as the CPU's own number.

- (67) The special function organization block call is wrong (SEND, RECEIVE, SEND TEST, RECEIVE TEST). The following errors are possible:
 - a) Secondary error, since the INITIALIZE function could not be called or was terminated by an initialization conflict.
 - b) Double call: the call for this function, SEND, SEND TEST, RECEIVE or RECEIVE TEST is illegal, since one of the functions INITIALIZE, SEND, SEND TEST, RECEIVE or RECEIVE TEST has already been called in this CPU in a lower processing level (e.g. cyclic program execution). (See "Calling and nesting the special function organization blocks".)
 - c) The CPU's own number is incorrect (system data corrupted); following power down/power up the CPU number is generated again by the system program.

- (68) The management data (queue management) of the selected connections are incorrect; set up the buffer in the coordinator 923C again using the INITIALIZE function (SEND, RECEIVE, SEND TEST, RECEIVE TEST).

- (69) The parameter "block ID" (SEND) or the block ID provided by the sender (RECEIVE) is illegal. The following errors are possible:
 - The block ID is less than 1.
 - The block ID is greater than 2.

- (70) The parameter "block number" (SEND) or the block number supplied by the sender (RECEIVE) is illegal, since it is a data block with a special significance. The following errors are possible:
 - If the block ID = 1 : DB 0, DB 1, DB 2
 - If the block ID = 2 : DX 0, DX 1, DX 2

- (71) The parameter "block number" (SEND) or the block number provided by the sender (RECEIVE) is incorrect. The specified data block does not exist.
- (72) The parameter "field number" (SEND) is incorrect. The data block is too short or the field number too high.
- (73) The data block is not large enough to receive the block of data transmitted by the sender (RECEIVE).

Warning

The function could not be executed; the function call must be repeated, e.g. in the next cycle.

Warning numbers (evaluation of the condition code byte as a byte)

- (129) The SEND function cannot transfer data, since the transmitting capacity was already zero when the function was called.
- (130) The RECEIVE function cannot accept data, since the receiving capacity was already zero when the function was called.

3 INITIALIZE Function (OB 200)

Call parameters

1st data field

Before calling OB 200, you must supply the input parameters in the data field. OB 200 requires eight F flag bytes in the data field for input and output parameters:

FY x + 0:	Mode (automatic/ manual)	input parameter
FY x + 1:	Number of CPUs	input parameter
FY x + 2:	Block ID	input parameter
FY x + 3:	Block number	input parameter
FY x + 4:	Start address of the assignment list	input parameter
FY x + 5:		input parameter
FY x + 6:	Condition code byte	output parameter
FY x + 7:	Total capacity	output parameter

2 ACCU-1-L:

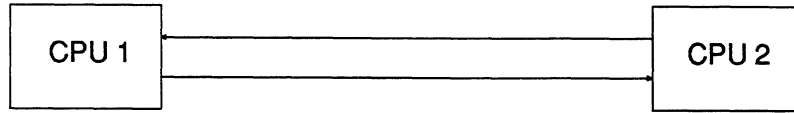
No. of the 1st flag byte "x" in the data field,
permitted values:

ACCU-1-LH: 0
ACCU-1-LL: 0 to 246

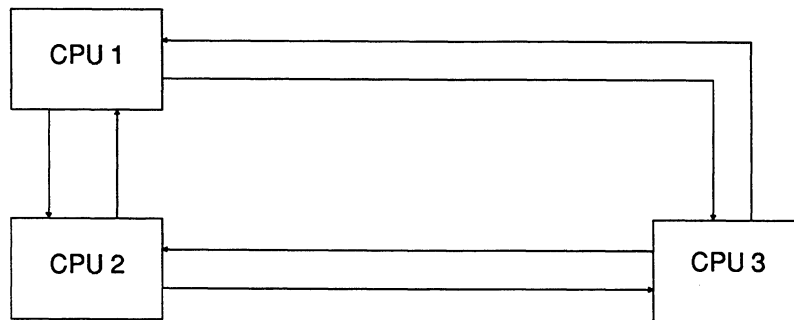
To transfer data from one CPU to another CPU, the data must be temporarily buffered. The INITIALIZE function sets up a buffer on the KOR 923C coordinator. The memory capacity is stipulated in fields (with a length of 32 words).

Each memory field (always with a length of 32 words) accepts one block of data (with a length between one data word and 32 data words). A block of data is entered in a memory field by a SEND block and read out by a RECEIVE block.

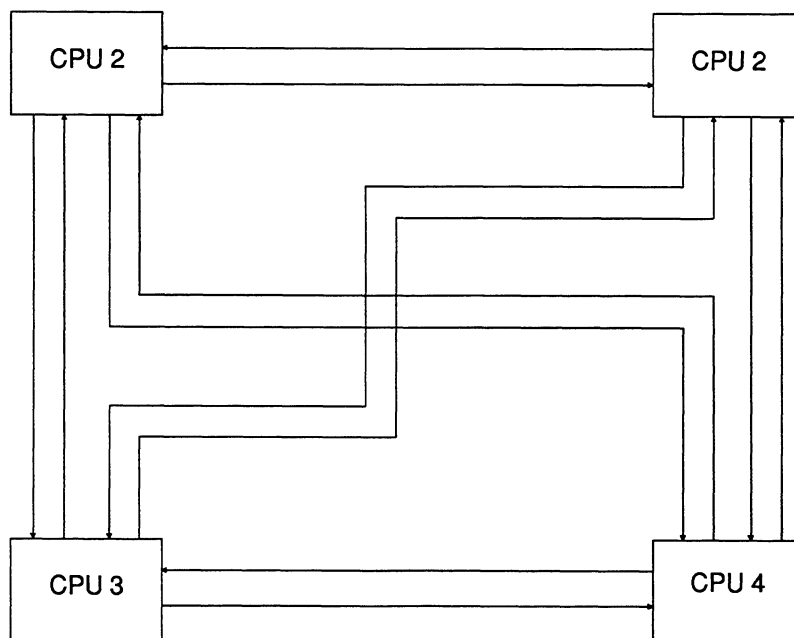
If you are using two CPUs, there are two connections (transfer directions, "channels"):



If you are using three CPUs, there are six connections:




If you are using four CPUs, there are twelve connections:



The INITIALIZE function specifies how the total of 48 available memory fields are assigned to the maximum twelve connections.

This means that each possible connection, specified by the parameters "transmitting CPU" and "receiving CPU" has a certain memory capacity available.

	<p>REMEMBER</p> <p>Before you can call the SEND / RECEIVE / SEND TEST / RECEIVE TEST functions, one CPU must have already called the INITIALIZE function and executed it completely and without errors.</p>
---	--

If the INITIALIZE function is called several times, one after the other, the last assignment made is valid. While a CPU is processing the INITIALIZATION function, no other functions including the INITIALIZE function can be called on other CPUs.

3.1 Input Parameters

3.1.1 Mode (Automatic / Manual)

Mode = 1 : automatic
 Mode = 2 : manual
 Mode = 0 or 3 to 255 : illegal, causes an initialization conflict

"Automatic" mode

If you select the "automatic" mode, the memory fields available are divided **equally** according to the number of CPUs:

Number of CPUs	Number of connections	Memory fields per connection
2	2	24
3	6	8
4	12	4
0; 1; 5 to 255	Illegal: causes an initialization conflict	

"Manual" mode

If you select the "manual" mode, you must create an assignment list in a data block in which the 48 (or less) available memory fields are assigned to the maximum 12 connections according to a fixed scheme. This function is particularly useful when some connections have far more data traffic than others. For example, CPUs 921 (S processors) cannot take part in the multiprocessor communication described here; the potential connections between this CPU and other CPUs do not therefore need memory fields and should not have memory fields assigned to them. The parameters

- block ID,
- block number and the
- start address of the assignment list

specify where the assignment list is stored. These three parameters are therefore only relevant for the "manual" mode.

3.1.2 Number of CPUs

This parameter is only relevant if you select the "automatic" mode; (see 3.1.1)

3.1.3 Block ID and Number / Start Address of the Assignment List

These parameters are only relevant if you select the "manual" mode.

Block ID and number

ID = 1:	DB data block
ID = 2:	DX data block
ID = 0 or 3 to 255 :	illegal, causes an initialization conflict

For the block number, you specify the number of the DB or DX data block in which the assignment list is stored.

Start address of the assignment list

Along with the block ID and number, this specifies the area (or more precisely, the start address of the area) in which the assignment list is stored.

The assignment list contains further input parameters for the INITIALIZE function, i.e. this area is only read (the contents are not changed). The assignment list has the structure shown on the following page:

Assignment list

Data word	:	Format	Value	:	Significance
DW n+ 0	:	KS	S1	:	Transmitter = CPU 1
DW n+ 1	:	KY	2, a	:	Receiver = CPU 2
DW n+ 2	:	KY	3, b	:	Receiver = CPU 3
DW n+ 3	:	KY	4, c	:	Receiver = CPU 4
DW n+ 4	:	KS	S2	:	Transmitter = CPU 2
DW n+ 5	:	KY	1, d	:	Receiver = CPU 1
DW n+ 6	:	KY	3, e	:	Receiver = CPU 3
DW n+ 7	:	KY	4, f	:	Receiver = CPU 4
DW n+ 8	:	KS	S3	:	Transmitter = CPU 3
DW n+ 9	:	KY	1, g	:	Receiver = CPU 1
DW n+ 10	:	KY	2, h	:	Receiver = CPU 2
DW n+ 11	:	KY	4, i	:	Receiver = CPU 4
DW n+ 12	:	KS	S4	:	Transmitter = CPU 4
DW n+ 13	:	KY	1, k	:	Receiver = CPU 1
DW n+ 14	:	KY	2, l	:	Receiver = CPU 2
DW n+ 15	:	KY	3, m	:	Receiver = CPU 3

REMEMBER

You must keep to this structure even if you have less than four CPUs.

The lower case letters a to m in bold face represent numbers between 0 and 48; **the sum of these numbers must not exceed 48.**

The next page shows an example of a completed assignment list.

Example

You have three CPUs in your rack, CPU 2 sends a lot of data to the other two CPUs. The other two CPUs, however, only send a small amount of data back to CPU 2 as acknowledgements in a logical handshake. There is no data exchange between CPU 1 and CPU 3.

Assignment list

Data word	:	Format	Value	:	Significance
DW n + 0	:	KS	S1	:	Transmitter = CPU 1
DW n + 1	:	KY	2, 2	:	Receiver = CPU 2
DW n + 2	:	KY	3, 0	:	Receiver = CPU 3
DW n + 3	:	KY	4, 0	:	Receiver = CPU 4
DW n + 4	:	KS	S2	:	Transmitter = CPU 2
DW n + 5	:	KY	1, 22	:	Receiver = CPU 1
DW n + 6	:	KY	3, 22	:	Receiver = CPU 3
DW n + 7	:	KY	4, 0	:	Receiver = CPU 4
DW n + 8	:	KS	S3	:	Transmitter = CPU 3
DW n + 9	:	KY	1, 0	:	Receiver = CPU 1
DW n + 10	:	KY	2, 2	:	Receiver = CPU 2
DW n + 11	:	KY	4, 0	:	Receiver = CPU 4
DW n + 12	:	KS	S4	:	Transmitter = CPU 4
DW n + 13	:	KY	1, 0	:	Receiver = CPU 1
DW n + 14	:	KY	2, 0	:	Receiver = CPU 2
DW n + 15	:	KY	3, 0	:	Receiver = CPU 3

3.2 Output Parameters

3.2.1 Condition Code Byte

This byte informs you whether the INITIALIZE function was executed correctly and completely.

Initialization conflict

The initialization conflicts listed are recognized and indicated by the function in the ascending order of their numbers.

If an initialization conflict occurs, you must change the program / parameters.

Initialization conflict numbers (evaluation of the condition code byte as a byte)

- (33) The pages required for multiprocessor communication (numbers 252 to 255) are not or not all available.
- (34) The pages required for multiprocessor communication (numbers 252 to 255) are defective.
- (35) The parameter "automatic/manual" is illegal. The following errors are possible:
 - The "automatic/manual" ID is less than 1.
 - The "automatic/manual" ID is greater than 2.
- (36) The parameter "number of CPUs" is illegal. The following errors are possible:
 - The number of CPUs is less than 2.
 - The number of CPUs is greater than 4.
- (37) The parameter "block ID" is illegal. The following errors are possible:
 - The block ID is less than 1.
 - The block ID is greater than 2.
- (38) The parameter "block number" is illegal, since it is a data block with a special significance. The following errors are possible:
 - If block ID = 1 : DB 0, DB 1, DB 2
 - If block ID = 2 : DX 0, DX 1, DX 2
- (39) The parameter "block number" is incorrect, since the data block does not exist.
- (40) The parameter "start address of the assignment list" is too high or the data block is too short.
- (41) The assignment list in the data block is not correctly structured.
- (42) The sum of the assigned memory fields is greater than 48.

Errors

The "error" number group cannot occur with the INITIALIZE function.

Warning

The "warning" number group cannot occur with the INITIALIZE function.

3.2.2 Total Capacity

This parameter specifies how many of the 48 available memory fields are assigned to connections.

In the "automatic" mode, this parameter always has the value 48. In the "manual" mode, it can have a value less than 48. This means that existing memory capacity is not used.

4 SEND Function (OB 202)

Call parameters

1st data field: Before calling OB 202 you must specify the input parameters in the data field. OB 202 requires six F flag bytes in the data field for input and output parameters:

FY x + 0:	receiving CPU	input parameter
FY x + 1:	block ID	input parameter
FY x + 2:	block number	input parameter
FY x + 3:	field number	input parameter
FY x + 4:	condition code byte	output parameter
FY x + 5:	transmitting capacity	output parameter

2. ACCU-1-L: No. of the first flag byte "x" in the data field:
 permitted values: ACCU-1-LH: 0
 ACCU-1-LL: 0 to 246

The SEND function transfers a data block to the buffer of the 923C coordinator. It also indicates how many blocks of data can still be sent and buffered.

4.1 Input Parameters

4.1.1 Receiving CPU

The data to be sent are intended for the receiving CPU; the permitted value is between 1 and 4 but must be different from the CPU's own number.

4.1.2 Block ID and Number / Field Number

Block ID

ID = 1:	DB data block
ID = 2:	DX data block
ID = 0 or 3 to 255:	illegal, causes an error message

Block number

The block number, along with the block ID (see above) and the field number (see below) specifies the area from which the data to be sent is taken (and where it is to be stored in the receiving CPU).

Remember that certain data blocks have a special significance, for example, DB 0, DB 1 or DX 0 (see programming instructions for your CPUs). These data blocks must therefore not be used for the data transfer described here.

If you attempt to use these block numbers, the function is aborted with an error message.

Field number

The field number indicates the area in which the data to be sent is located.

Field number	Data area	
	First data word	Last data word
0	DW 0	DW 31
1	DW 32	DW 63
2	DW 64	DW 95
3	DW 96	DW 127
4	DW 128	DW 159
5	DW 160	DW 191
6	DW 192	DW 223
7	DW 224	DW 255
8	DW 256	DW 287
9	DW 288	DW 319
:	:	:
:	:	:

The following situations are possible:

- If the data block is sufficiently long, you obtain a 32-word long area as shown in the table above.
- If the end of the data block is within the selected field, an area with a length between 1 and 32 words will be transferred.
- If the first data word address is not within the length of the data block, the SEND function detects and indicates an error.

Example

Data block with a length of 80 words: DW 0 to DW 74, 5 words are required for the block header.

Field number	Data area		Length
	First data word	Last data word	
0	DW 0	DW 31	32 words
1	DW 32	DW 63	32 words
2	DW 64	DW 74	11 words
3 and higher	Incorrect parameter assignment		

4.2 Output Parameters**4.2.1 Condition Code Byte**

This byte informs you whether the SEND function was executed correctly and completely.

Errors

If an error occurs, you must change the program/parameters.

Error numbers (evaluation of the condition code byte as a byte)

- (65) The parameter "receiving CPU" is illegal. The following errors are possible:
 - The number of the receiving CPU is greater than 4.
 - The number of the receiving CPU is less than 4
 - The number of the receiving CPU is the same as the CPU's own number.

- **(67)** The special function organization block call is wrong. The following errors are possible
 - a) Secondary error, since the INITIALIZE function could not be called or was terminated by an initialization conflict.
 - b) Double call: the call for this function, SEND, SEND TEST, RECEIVE or RECEIVE TEST is illegal, since one of the functions INITIALIZE, SEND, SEND TEST, RECEIVE or RECEIVE TEST has already been called in this CPU in a lower processing level (e.g. cyclic program execution). (See "Calling and nesting the special function organization blocks".)
 - c) The CPU's own number is incorrect (system data corrupted); following power down/power up the CPU number is generated again by the system program.

- **(68)** The management data (queue management) of the selected connections are incorrect; set up the buffer in the coordinator 923C again using the INITIALIZE function.

- **(69)** The parameter "block ID" is illegal. The following errors are possible:
 - The block ID is less than 1.
 - The block ID is greater than 2.

- **(70)** The parameter "block number" is illegal, since it is a data block with a special significance. The following errors are possible:
 - If the block ID = 1 : DB 0, DB 1, DB 2
 - If the block IF = 2 : DX 0, DX 1, DX 2

- **(71)** The parameter "block number" provided by the sender (RECEIVE) is incorrect. The specified data block does not exist.

- **(72)** The parameter "field number" is incorrect. The data block is too short or the field number too high.

Warning

The function could be executed; the function call must be repeated, e.g. in the next cycle.

Warning numbers (evaluation of the condition code byte as a byte)

- (129) The SEND function cannot transfer data, since the transmitting capacity was already zero when the function was called.

Initialization conflict

The "initialization conflict" number group cannot occur with the SEND function.

4.2.2 Transmitting Capacity

The "transmitting capacity" indicates how many blocks of data can still be sent and buffered.

5 SEND TEST Function (OB 203)

Call parameters

1. Data field: Before calling OB 203, you must specify the input parameters in the data field. OB 203 requires 4 F flag bytes in the data field for input and output parameters:

FY x + 0:	receiving CPU	input parameter
FY x + 1:	—	not used
FY x + 2:	condition code byte	output parameter
FY x + 3:	transmitting capacity	output parameter

2. ACCU-1-L: No. of the first flag byte "x" in the data field,
permitted values: ACCU-1-LH:0
ACCU-1-LL: 0 to 246

The SEND TEST function determines the number of free memory fields in the buffer of the 923C coordinator.

Depending on this number m, the SEND function can be called m times to transfer m blocks of data.

5.1 Input Parameters

5.1.1 Receiving CPU

The CPU's own number and the number of the receiving CPU identify the connection for which the transmitting capacity (see above) is determined.

5.2 Output Parameters

5.2.1 Condition Code Byte

This byte indicates whether the SEND TEST function was executed correctly and completely.

Errors

If an error occurs, you must change the program parameters.

Error numbers (evaluation of the condition code byte as a byte)

- (65) The parameter "receiving CPU" is illegal. The following errors are possible:
 - The number of the receiving CPU is greater than 4.
 - The number of the receiving CPU is less than 1.
 - The number of the receiving CPU is the same as the CPU's own number.

- (67) The special function organization block call is wrong. The following errors are possible:
 - a) Secondary error, since the INITIALIZE function could not be called or was terminated by an initialization conflict.
 - b) Double call: the call for this function, SEND, SEND TEST, RECEIVE or RECEIVE TEST is illegal, since one of the functions INITIALIZE, SEND, SEND TEST, RECEIVE or RECEIVE TEST has already been called in this CPU in a lower processing level (e.g. cyclic program execution). (See "Calling and nesting the special function organization blocks".)
 - c) The CPU's own number is incorrect (system data corrupted); following power down/power up the CPU number is generated again by the system program

- (68) The management data (queue management) of the selected connections are incorrect; set up the buffer in the coordinator 923C again using the INITIALIZE function.

Warning

The "warning" number group cannot occur with the SEND TEST function.

Initialization conflict

The "initialization conflict" number group cannot occur with the SEND TEST function.

5.2.2 Transmitting Capacity

The "transmitting capacity" parameter indicates how many blocks of data can be sent and buffered.

6 RECEIVE Function (OB 204)

Call parameters

1. Data field

Before calling OB 204, you must specify the input parameters in the data field. OB 204 requires 10 F flag bytes in the data field for input and output parameters:

FY x + 0:	transmitting CPU	input parameter
FY x + 1:	—	not used
FY x + 2:	condition code byte	output parameter
FY x + 3:	receiving capacity	output parameter
FY x + 4:	block ID	output parameter
FY x + 5:	block number	output parameter
FY x + 6:	address of the first	output parameter
FY x + 7:	received data word	output parameter
FY x + 8:	address of the last	output parameter
FY x + 9:	received data word	output parameter

2. ACCU-1-L:

No. of the first flag byte "x" in the data field,
permitted values: ACCU-1-LH: 0
ACCU-1-LL: 0 to 246

The RECEIVE function takes a block of data from the buffer of the 923C coordinator. It also indicates how many data blocks are still buffered and can still be received. The RECEIVE function should be called in a loop until all the buffered blocks of data have been received.

6.1 Input Parameters

6.1.1 Transmitting CPU

The receive block receives data supplied by the transmitting CPU; the permitted value is between 1 and 4, but must be different from the CPU's own number.

6.2 Output Parameters

6.2.1 Condition Code Byte

This byte informs you whether the RECEIVE function was executed correctly and completely.

Errors

If an error occurs, you must change the program/parameters.

Error numbers (evaluation of the condition code byte as a byte)

- (66) The parameter "transmitting CPU" is illegal. The following errors are possible:
 - The number of the transmitting CPU is greater than 4.
 - The number of the transmitting CPU is less than 1.
 - The number of the transmitting CPU is the same as the CPU's own number.
- (67) The special function organization block call is wrong. The following errors are possible
 - a) Secondary error, since the INITIALIZE function could not be called or was terminated by an initialization conflict.
 - b) Double call: the call for this function, SEND, SEND TEST, RECEIVE or RECEIVE TEST is illegal, since one of the functions INITIALIZE, SEND, SEND TEST, RECEIVE or RECEIVE TEST has already been called in this CPU in a lower processing level (e.g. cyclic program execution). (See "Calling and nesting the special function organization blocks".)
 - c) The CPU's own number is incorrect (system data corrupted); following power down/power up the CPU number is generated again by the system program.
- (68) The management data (queue management) of the selected connections are incorrect; set up the buffer in the coordinator 923C again using the INITIALIZE function.
- (69) The block identifiers supplied by the transmitter is illegal. The following errors are possible
 - The block ID is less than 1
 - The block ID is greater than 2.

- (70) The block number supplied by the transmitter is illegal, since it is a data block with a special significance. The following errors are possible:
 - If the block ID = 1 : DB 0, DB 1, DB 2
 - If the block ID = 2 : DX 0, DX 1, DX 2
- (71) The block number provided by the transmitter is incorrect. The specified data block does not exist.
- (73) The data block is too small to receive the block of data supplied by the transmitter.

Warning

The function could be executed; the function call must be repeated, e.g. in the next cycle.

Warning numbers (evaluation of the condition code byte as a byte)

- (130) The RECEIVE function cannot receive data, since the receiving capacity was already zero when the function was called.

Initialization conflict

The "initialization conflict" number group cannot occur with the RECEIVE function.

6.2.2 Receiving Capacity

The "receiving capacity" parameter indicates how many blocks of data are still buffered and can still be received.

6.2.3 Block ID and Number

Block ID

ID = 1: DB data block
ID = 2: DX data block

Block number

The block number along with the block ID (see above) and the addresses of the first and last data word (see below) specifies the area in which the received data were stored by the RECEIVE function (and the area from which they were taken in the transmitting CPU by the SEND function).

Remember that the receive data blocks should be in a random access memory (RAM); using read-only memories (EPROM) might possibly serve a practical purpose for transmit data blocks.

6.2.4 Address of the First/Last Received Data Word

The difference between the addresses of the first and last data word transferred is a maximum of 31, since a maximum of 32 data words can be transferred per function call.

7 RECEIVE TEST Function (OB 205)

Call parameters

1. Data field: Before calling OB 205, you must specify the input parameters in the data field. OB 205 requires 4 F flag bytes in the data field for input and output parameters:

FY x + 0:	transmitting CPU	input parameter
FY x + 1:	—	not used
FY x + 2:	condition code byte	output parameter
FY x + 3:	transmitting capacity	output parameter

2. ACCU-1-L: No. of the first flag byte "x" in the data field,
permitted values: ACCU-1-LH: 0
ACCU-1-LL: 0 to 246

The RECEIVE TEST function determines the number of occupied memory fields in the buffer of the 923C coordinator. Depending on this number m, the RECEIVE function can be called m times to receive m blocks of data.

7.1 Input Parameters

7.1.1 Transmitting CPU

The CPU's own number and the number of the transmitting CPU identify the connection for which the receiving capacity (see above) is determined.

7.2 Output Parameters

7.2.1 Condition Code Byte

This byte indicates whether the RECEIVE TEST function was executed correctly and completely.

Errors

If an error occurs, you must change the program parameters.

Error numbers (evaluation of the condition code byte as a byte)

- (66) The parameter "transmitting CPU" is illegal. The following errors are possible:
 - The number of the transmitting CPU is greater than 4.
 - The number of the transmitting CPU is less than 1
 - The number of the transmitting CPU is the same as the CPU's own number.

- (67) The special function organization block call is wrong. The following errors are possible:
 - a) Secondary error, since the INITIALIZE function could not be called or was terminated by an initialization conflict.
 - b) Double call: the call for this function, SEND, SEND TEST, RECEIVE or RECEIVE TEST is illegal, since one of the functions INITIALIZE, SEND, SEND TEST, RECEIVE or RECEIVE TEST has already been called in this CPU in a lower processing level (e.g. cyclic program execution). (See "Calling and nesting the special function organization blocks".)
 - c) The CPU's own number is incorrect (system data corrupted); following power down/power up the CPU number is generated again by the system program..

- (68) The management data (queue management) of the selected connections are incorrect; set up the buffer in the coordinator 923C again using the INITIALIZE function.

Warning

The "warning" number group cannot occur with the RECEIVE TEST function.

Initialization conflict


The "initialization conflict" number group cannot occur with the RECEIVE TEST function.

7.2.2 Receiving Capacity

The "receiving capacity" parameter indicates how many blocks of data can be received and buffered.

8 Applications

When using one of the function blocks listed below and using interrupts (e.g. OB 2), make sure that the scratchpad flags are saved at the beginning of the interrupt handling and are written back again at the end.

	<p>REMEMBER</p> <hr/> <p>This also applies to the setting "interrupts at block boundaries", since the call of the special function organization blocks represents a block boundary.</p>
---	--

8.1 Calling the Special Function OB Using Function Blocks


The following five function blocks (FB 200 and FB 202 to FB 205) contain the call for the corresponding special function organization block for multiprocessor communication (OB 200 and OB 202 to OB 205).

The numbers of the function blocks are not fixed and can be changed. The parameters of the special function OBs are transferred as actual parameters when the function blocks are called. The direct call of the special function organization blocks is faster, however, is more difficult to read owing to the absence of formal parameters.

FB no.	FB name	Function
FB 200	INITIAL	Set up buffer
FB 202	SEND	Send a block of data
FB 203	SEND-TST	Test the sending capacity
FB 204	RECEIVE	Receive a block of data
FB 205	RCV-TST	Test receiving capacity

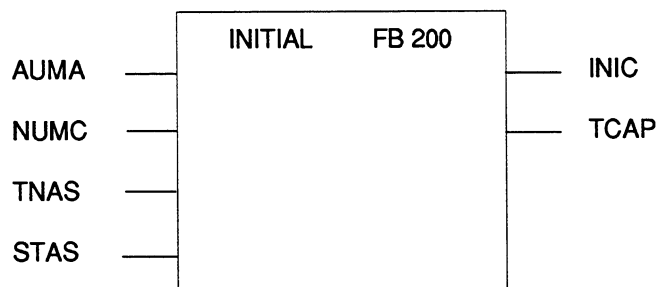
The flag area from FY 246 to maximum FY 255 is used by the function blocks as a parameter field for the special function organization blocks.

The exact significance of the input and output parameters is explained in the description of the special function organization blocks.

	<p>REMEMBER</p> <hr/> <p>The following examples of applications involve finished applications that you can program by copying them.</p>
---	--

8.1.1 Setting Up a Buffer (FB 200)

Parameter name	Significance	Parameter type	Data type	Parameter field
AUMA	Automatic/manual	I	BY	FY 246
NUMC	Number of CPUs	I	BY	FY 247
TNAS	Type (H byte) and number (L byte) of the data block containing the assignment list	I	W	FW 248
STAS	Start address of the assignment list	I	W	FW 250
INIC	Initialization conflict	Q	BY	FY 252
TCAP	Total capacity	Q	BY	FY 253



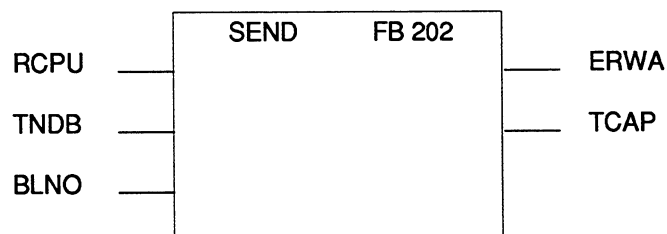
Applications

FB 200			LEN=45	ABS
SEGMENT 1				
NAME:INITIAL				
DECL :AUMA	I/Q/D/B/T/C:	I	BI/BY/W/D:	BY
DECL :NUMC	I/Q/D/B/T/C:	I	BI/BY/W/D:	BY
DECL :TNAS	I/Q/D/B/T/C:	I	BI/BY/W/D:	W
DECL :STAS	I/Q/D/B/T/C:	I	BI/BY/W/D:	W
DECL :INIC	I/Q/D/B/T/C:	Q	BI/BY/W/D:	BY
DECL :TCAP	I/Q/D/B/T/C:	Q	BI/BY/W/D:	BY

0017	:L	=AUMA	AUTOMATIC/MANUAL
0018	:T	FY 246	
0019	:L	=NUMC	NUMBER OF CPUs
001A	:T	FY 247	
001B	:L	=TNAS	DB TYPE, DB NO.
001C	:T	FW 248	
001D	:L	=STAS	START ADDRESS OF THE ASSIGNMENT LIST
001E	:T	FW 250	
001 F	:		
0020	:L	KB 246	SF OB:
0021	:JU	OB 200	INITIALIZE
0022	:		
0023	:L	FY 252	INITIALIZATION CONFLICT
0024	:T	=INIC	
0025	:L	FY 253	TOTAL CAPACITY
0026	:T	=TCAP	
0027	:BE		

8.1.2 Sending a Block of Data (FB 202)

Parameter name	Significance	Parameter type	Data type	Parameter field
RCPU	Receiving CPU	I	BY	FY 246
TNDB	Type (H byte) and number (L byte) of the source data block	I	W	FW 247
BLNO	Block number	I	BY	FY 249
ERWA	Error warning	Q	BY	FY 250
TCAP	Transmitting capacity	Q	BY	FY 251



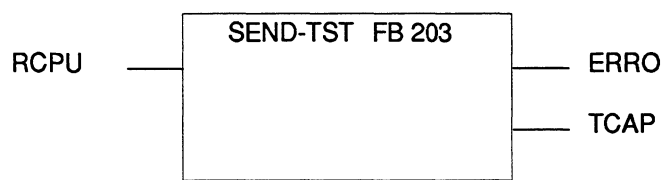
Applications

FB 202			LEN=40	ABS
SEGMENT 1				
NAME:SEND				
DECL :RCPU	I/Q/D/B/T/C:	I	BI/BY/W/D:	BY
DECL :TNDB	I/Q/D/B/T/C:	I	BI/BY/W/D:	W
DECL :BLNO	I/Q/D/B/T/C:	I	BI/BY/W/D:	BY
DECL :ERWA	I/Q/D/B/T/C:	Q	BI/BY/W/D:	BY
DECL :TCAP	I/Q/D/B/T/C:	Q	BI/BY/W/D:	BY

0014	:L	=RCPU	RECEIVING CPU
0015	:T	FY 246	
0016	:L	=TNDB	DB TYPE, DB NO.
0017	:T	FW 247	
0018	:L	=BLNO	BLOCK NUMBER
0019	:T	FY 249	
001A	:		
001B	:L	KB 246	SF OB:
001C	:JU	OB 202	SEND A BLOCK OF DATA
001D	:		
001E	:L	FY 250	ERROR/WARNING
001F	:T	=ERWA	
0020	:L	FY 251	TRANSMITTING CAPACITY
0021	:T	=TCAP	
0022	:BE		

8.1.3 Testing the Transmitting Capacity (FB 203)

Parameter name	Significance	Parameter type	Data type	Parameter field
RCPU	Receiving CPU	I	BY	FY 246
ERRO	Error	Q	BY	FY 248
TCAP	Transmitting capacity	Q	BY	FY 249



```

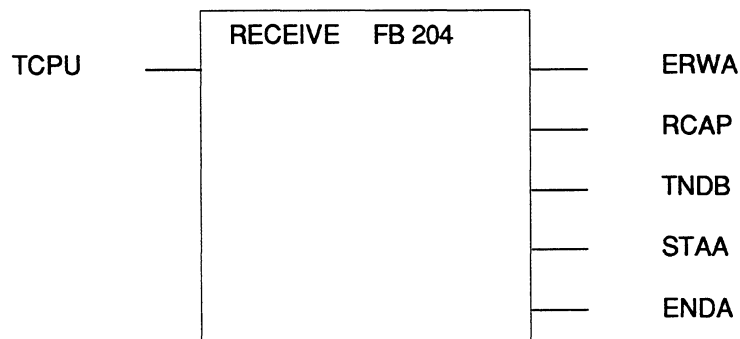
FB 203                               LEN=30  ABS
SEGMENT 1
NAME:SEND-TST
DECL :RCPU      I/Q/D/B/T/C:      I  BI/BY/W/D:      BY
DECL :ERRO      I/Q/D/B/T/C:      I  BI/BY/W/D:      BY
DECL :TCAP      I/Q/D/B/T/C:      Q  BI/BY/W/D:      BY
  
```

```

000E      :L      =RCPU              RECEIVING CPU
000F      :T      FY 246
0010      :
0011      :L      KB 246              SF OB:
0012      :JU     OB 203              TEST TRANSMITTING CAPACITY
0013      :
0014      :L      FY 248              ERROR
0015      :T      =ERRO
0016      :L      FY 249              TRANSMITTING CAPACITY
0017      :T      =TCAP
0018      :BE
  
```

8.1.4 Receiving a Block of Data (FB 204)

Parameter name	Significance	Parameter type	Data type	Parameter field
TCPU	Transmitting CPU	I	BY	FY 246
ERWA	Error warning	Q	BY	FY 248
RCAP	Receiving capacity	Q	BY	FY 249
TNDB	Type (H byte) and number (L byte) of the destination data block	Q	W	FW 250
STAA	Address of the first received data word (start address)	Q	W	FW 252
ENDA	Address of the last received data word (end address)	Q	W	FW 254



```

FB 204                                LEN=45  ABS
SEGMENT 1
NAME:RECEIVE
DECL :TCPU      I/Q/D/B/T/C:      I  BI/BY/W/D:      BY
DECL :ERWA      I/Q/D/B/T/C:      Q  BI/BY/W/D:      BY
DECL :RCAP      I/Q/D/B/T/C:      Q  BI/BY/W/D:      BY
DECL :TNDB      I/Q/D/B/T/C:      Q  BI/BY/W/D:      W
DECL :STAA      I/Q/D/B/T/C:      Q  BI/BY/W/D:      W
DECL :ENDA      I/Q/D/B/T/C:      Q  BI/BY/W/D:      W

```

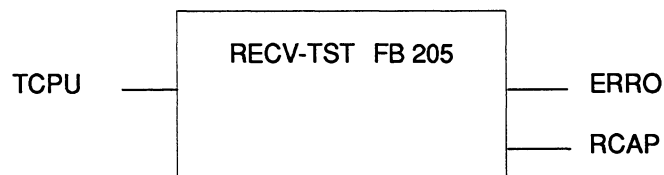
```

0017      :L      =TCPU              TRANSMITTING CPU
0018      :T      FY 246
0019      :
001A      :L      KB 246              SF OB:
001B      :JU      OB 204             RECEIVE A BLOCK
                                         OF DATA
001C      :
001D      :L      FY 248              ERROR/WARNING
001E      :T      =ERWA
001F      :L      FY 249              RECEIVING CAPACITY
0020      :T      =RCAP
0021      :L      FW 250              DB TYPE, DB NO.
0022      :T      =TNDB
0023      :L      FW 252              START ADDRESS
0024      :T      =STAA
0025      :L      FW 254              END ADDRESS
0026      :T      =ENDA
0027      :BE

```

8.1.5 Testing the Receiving Capacity (FB 205)

Parameter name	Significance	Parameter type	Data type	Parameter field
TCPU	Transmitting CPU	I	BY	FY 246
ERRO	Error	Q	BY	FY 248
RCAP	Receiving capacity	Q	BY	FY 249



```

FB 205                                LEN=30  ABS
SEGMENT 1
NAME: RECV-TST
DECL :TCPU      I/Q/D/B/T/C:      I  BI/BY/W/D:      BY
DECL :ERRO      I/Q/D/B/T/C:      Q  BI/BY/W/D:      BY
DECL :RCAP      I/Q/D/B/T/C:      Q  BI/BY/W/D:      BY
  
```

```

000E      :L      =TCPU                TRANSMITTING CPU
000F      :T      FY 246
0010      :
0011      :L      KB 246                SF OB:
0012      :JU     OB 205                TEST RECEIVING CAPACITY
0013      :
0014      :L      FY 248                ERROR
0015      :T      =ERRO
0016      :L      FY 249                RECEIVING CAPACITY
0017      :T      =RCAP
0018      :BE
  
```

8.2 Transferring Data Blocks

8.2.1 Functional Description

The function block TRAN DAT (FB 110) transfers a selectable number of blocks of data from a data block in one CPU to the data block of the same type and same number in a different CPU. (For a description of the parameter list, function block and STEP 5 program, see the following page.)

The FB number has been selected at random and you can use other numbers.

8.2.2 Transferring a Data Block (FB 110)

The data area to be transferred is stipulated by the input parameter FIRB (= number of the first block of data to be transferred) and NUMB (= number of blocks of data to be transferred). A block of data normally consists of 32 data words. Depending on the data block length, the last block of data may be less than 32 data words.

The transfer is triggered by a positive-going edge at the start input STAR. If the output parameter REST is zero after the transfer, this means that the function block TRANDAT was able to send all the blocks of data (according to the NUMB parameter).

If, however, the REST output parameter has a value greater than zero, this means that the function block must be called again, for example in the next cycle. This means that you or the user program can only change the set of parameters (i.e. the values of all parameters) when the REST parameter indicates zero showing that the data transfer is complete.

You can call the function block TRANDAT several times with different parameters. In this case, various data areas are transferred simultaneously (interleaved in each other). The special function organization blocks for multiprocessor communication OB 202 to OB 205 can also be used "directly". This possibility is illustrated in the application example.

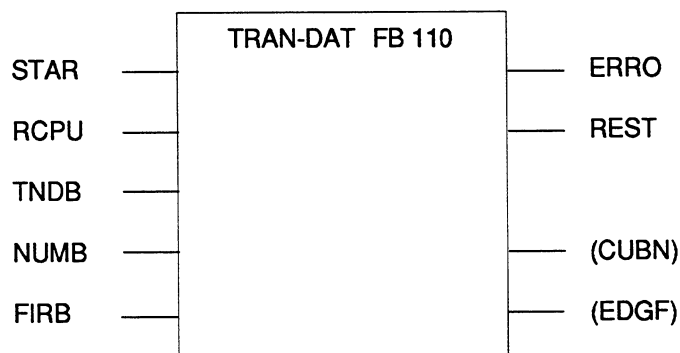
If the SEND function (OB 202) is not correctly executed within the TRANDAT function block, the error number is entered in the output parameter ERRO, the RLO = "1" and the output parameter REST is set to "0".

The TRANDAT function block uses flag bytes FY 246 to FY 251 as scratchpad flags. All other variables whose value is significant as long as the output parameter REST = "0" continue to have memory assigned to them using the mechanism of formal/actual parameters. This is necessary to allow various data blocks to be transferred simultaneously.

Note: data block and block of data are not synonymous. Data block is a DB or DX and a block of data is part of a DB from 1 to maximum 32 data words.

Parameter name	Significance	Parameter type	Data type
STAR	Start the transfer of the data block on a positive-going edge	I	BI
RCPU	Receiving CPU	I	BY
TNDB	Type (H byte) and number (L byte) of the data block to be transferred	I	W
NUMB	Number of blocks of data to be transferred	I	BY
FIRB	Number of the first block of data to be transferred	I	BY
ERRO	Error	Q	BY
REST	Number of blocks of data still to be transferred	Q	BY
CUBN ¹⁾	Current block number	Q	BY
EDGF ¹⁾	Edge flag	Q	BI

1) Internal scratchpad flag, not intended for evaluation.




```

FB 110                                LEN=89   ABS
SEGMENT 1
NAME:TRAN-DAT
DECL :STAR      I/Q/D/B/T/C:      I   BI/BY/W/D:      BI
DECL :RCPU      I/Q/D/B/T/C:      I   BI/BY/W/D:      BY
DECL :TNDB      I/Q/D/B/T/C:      I   BI/BY/W/D:      W
DECL :NUMB      I/Q/D/B/T/C:      I   BI/BY/W/D:      BY
DECL :FIRB      I/Q/D/B/T/C:      I   BI/BY/W/D:      BY
DECL :ERRO      I/Q/D/B/T/C:      Q   BI/BY/W/D:      BY
DECL :REST      I/Q/D/B/T/C:      Q   BI/BY/W/D:      BY
DECL :CUBN      I/Q/D/B/T/C:      Q   BI/BY/W/D:      BY
DECL :EDGF      I/Q/D/B/T/C:      Q   BI/BY/W/D:      BI

0020      :L      =RCPU                ASSIGN PARAMETER FIELD FOR
0021      :T      FY 246                SF OB 202
0022      :L      =TNDB
0023      :T      FW 247
0024      :
0025      :L      =REST                FIRST SEND ANY REMAINING
0026      :L      KB 0                 BLOCKS OF DATA
0027      :><F
0028      :JC     =TRAN
0029      :
002A      :AN     =STAR                POSITIVE EDGE AT START
002B      :RB     =EDGF                INPUT ?
002C      :ON     =STAR
002D      :O      =EDGF
002E      :JC     =GOOD
002F      :S      =EDGF
0030      :
0031      :L      =NUMB                INITIALIZE THE GLOBAL FLAGS
0032      :T      =REST                AFTER POSITIVE EDGE AT
0033      :L      =FIRB                START INPUT
0034      :T      =CUBN
0035      :
0036      :L      =REST                AS LONG AS REST ><0,
0038 LOOP :L      KF+0                 CONTINUE TO ATTEMPT
0039      :!=F                          TO SEND BLOCKS OF DATA
003A      :JC     =GOOD
003B TRAN :L      =CUBN
003C      :T      FY 249
003D      :L      KB 246                SF OB:
003E      :JU     OB 202                SEND A BLOCK OF DATA
003F      :L      FY 250
0040      :JM     =ERRO                ABORT IF ERROR
0041      :JP     =GOOD                ABORT IF TRANS-CAP = 0
0042      :L      =CUBN                INCREMENT BLOCK NUMBER
0043      :I      1
0044      :T      =CUBN
0045      :L      =REST                DECREMENT NUMBER OF
0046      :D      1                     REMAINING BLOCKS OF DATA

```

```

0047      :T  =REST
0048      :JU =LOOP
0049      :
004A GOOD:A  F 0.0          REGULAR END OF PROGRAM
004B      :AN  F 0.0
004C      :L   KB 0          RLO = 0, ERRO = 0
004D      :T  =ERRO
004E      :BEU
004F      :
0050 ERRO :T  =ERRO          PROGRAM END IF ERROR
0051      :L   KB 0
0052      :T  =REST          RLO = 1, ERROR CONTAINS
                                ERROR NUMBER
0053      :BE

```

8.2.3 Application Example (for the S5-135U)

You want CPU 1 to transfer data blocks DB 3 (blocks of data 2 to 5) and DB 4 (blocks of data 1 to 3) to CPU 2 during the cyclic user program. The RECEIVE function (OB 204) is also called in the cyclic user program.

The following blocks must be loaded in the individual CPUs:

Function	CPU 1	CPU 2
Cold restart block	OB 20	—
Cycle block ¹⁾	FB 0	FB 0
Send DB	DB 3; DB 4	—
Receive DB	—	DB 3; DB 4

¹⁾ Only OB 1 is permitted as the cycle block in the CPU 946/947.

OB 20 calls the INITIALIZE function (OB 200) and reserves several memory fields for the connection from CPU 1 to CPU 2.

The cyclic user program in function block FB 0 of CPU 1 contains two calls for the function block TRANDAT in each case with different sets of parameters. The transfer of the first data block DB 3 begins after a positive edge after input I 2.0. A positive edge at input I 2.1 starts the transfer of the second data block DB 4.

```

FB 0                               LEN=66   ABS
SEGMENT 1
NAME:DEMO

0005      :L   KB 2                TO CPU 2 ..
0006      :T   FY 0
0007      :L   KY 1,3              .. FROM DATA BLOCK DB 3
0009      :T   FW 1
000A      :L   KB 4                .. FOUR BLOCKS OF DATA
000B      :T   FY 3
000C      :L   KB 2                .. SEND FROM 2ND BLOCK OF DATA
000D      :T   FY 4
000E      :
000F      :JU  FB 110
0010 NAME :TRAN-DAT
0011 STAR :   I 2.0
0012 RCPU :   FY 0
0013 TNDB :   FW 1
0014 NUMB :   FY 3
0015 FIRB :   FY 4
0016 ERRO :   FY 5
0017 REST :   FY 6
0018 CUBN :   FY 7
0019 EDGF :   F 8.0
001A      :
001B      :
001C      :JC  =HALT              ABORT AFTER ERROR
001D      :
001E      :L   KB 2                TO CPU 2 ..
001F      :T   FY 10
0020      :L   KY 1,4              .. FROM DATA BLOCK DB 4
0022      :T   FW 11
0023      :L   KB 3                .. THREE BLOCKS OF DATA
0024      :T   FY 13
0025      :L   KB 1                .. SEND FROM 1ST BLOCK OF DATA
0026      :T   FY 14
0027      :
0028      :JU  FB 110
0029 NAME :TRAN-DAT
002A STAR :   I 2.1
002B RCPU :   FY 10
002C TNDB :   FW 11
002D NUMB :   FY 13
002E FIRB :   FY 14
002F ERRO :   FY 5
0030 REST :   FY 16
0031 CUBN :   FY 17
0032 EDGF :   F 8.1
0033      :
0034      :
0035      :JC  =HALT              ABORT AFTER ERROR
0036      :BEU

```

0037 :
0038 HALT :

The error handling takes place here (e.g stop, message output on the printer, ...)

0039 :BE

In CPU 2, the RECEIVE function (OB 204) called by FB 0 enters each transmitted block of data into the appropriate data block. It may take several cycles before a data block has been completely received.

FB 0
SEGMENT 1
NAME:RECV-DAT

LEN=26 ABS

0005	:L	KB 1	RECEIVE DATA FROM CPU 1
0006	:T	FY 246	
0007	:		
0008	LOOP	:L KB 246	SF OB:
0009	:JU	OB 204	RECEIVE
000A	:JM	=ERRO	ABORT IF ERROR
000B	:L	FY 249	THE RECEIVE FUNCTION IS
000C	:L	KB 0	CALLED UNTIL THERE ARE NO
000D	:><F		FURTHER BLOCKS OF DATA IN
000E	:JC	=LOOP	THE BUFFER, I.E. THE RECEIVING
000F	:		CAPACITY = 0.
0010	:BEU		
0011	ERRO	:	

The error handling takes place here (e.g. stop, message output on printer, ...)

0012 :BE

8.3 Extending the IPC Flag Area

8.3.1 The Problem

In the multiprocessor programmable controllers S5-135U and S5-155U, each of the 256 flag bytes of a CPU can become an input or output IPC flag by making an entry in data block DB 1. This, however, reduces the number of "normal" flag bytes. To transfer a data record (several bytes) other mechanisms are also required (semaphore variable or DX 0 parameter assignment "transfer IPC flags as a block") are necessary to prevent the receiver from receiving a fragmented data record.

8.3.2 The Solution

Consecutive data words of a DB or DX data block are defined from DW 0 onwards as "IPC data words". Each connection is assigned its own data block and is totally independent of the other connections.

At the beginning of the cycle block (CPU 946/947: OB 1, CPU 92x: OB 1 or FB 0), the IPC data words are received with the aid of the special function organization blocks for multiprocessor communication. This is followed by the "regular" cyclic program, that evaluates the received data and generates the data to be sent. At the end of the cycle, this data is then sent with the aid of the special organization blocks for multiprocessor communication. It can therefore be received by the other CPUs at the beginning of their cycles.

The following applies for each of the maximum 12 possible connections regardless of the other connections:

- The transmitting CPU is only active when the receiving CPU has read out all the "old" data from the 923C buffer.
- The receiving CPU is only active when the transmitting CPU has written all the "new" data in the 923C buffer.

This means that the receiving CPU can either receive a complete new data record or the old data record remains unchanged: **no mixing of "old" and "new" data**.

8.3.3 Data Structure

Which data words (for the data word area below) are to be transferred from which CPU to which CPU is described in the connection list (see table on the following page). This is located in an additional data block that must exist in all the CPUs involved.

The data word areas always begin from data word DW 0, and their lengths are specified in blocks of data. Remember the following points:

- A complete block of data consists of 32 data words.
- If the last block of a data block is "truncated", i.e. it contains between 1 and 31 data words, less data words are transferred.
- If a send data block is longer than the number of blocks of data specified in the connection list, the excess data words can be used in the corresponding CPU.
- If a receive data block is longer than the received data word area, the excess data words can be used in the corresponding CPU.

Structure of the connection list

	SUB-LIST 1			SUB-LIST 2		
Connection		DB type	DB number			No. of blocks of data
from CPU 1 to ...	DW 0	S 1		DW 16	S 1	
... CPU 2	DW 1	DW 17	2	...
... CPU 3	DW 2	DW 18	3	...
... CPU 4	DW 3	DW 19	4	...
from CPU 2 to ...	DW 4	S 2		DW 20	S 2	
... CPU 1	DW 5	DW 21	1	...
... CPU 3	DW 6	b	c	DW 22	3	a
... CPU 4	DW 7	DW 23	4	...
from CPU 3 to ...	DW 8	S 3		DW 24	S 3	
... CPU 1	DW 9	DW 25	1	...
... CPU 2	DW 10	DW 26	2	...
... CPU 4	DW 11	DW 27	4	...
from CPU 4 to ...	DW 12	S 4		DW 28	S 4	
... CPU 1	DW 13	DW 29	1	...
... CPU 2	DW 14	DW 30	2	...
... CPU 3	DW 15	DW 31	3	...

2^{15}

$2^0 \ 2^{15}$

2^0

The connection consists of two similarly structured sub-lists, each with 16 data words. For each of the four sender CPUs (S1, S2, S3, S4) three entries are required to describe a connection.

- **Number of blocks of data**

The number of blocks of data specifies the size (= the number of data words) of the data word area to be transferred. (If connections do not exist or you do not require them, enter 0 for the number of blocks of data, and for the DB type and DB number.)

- **DB type**

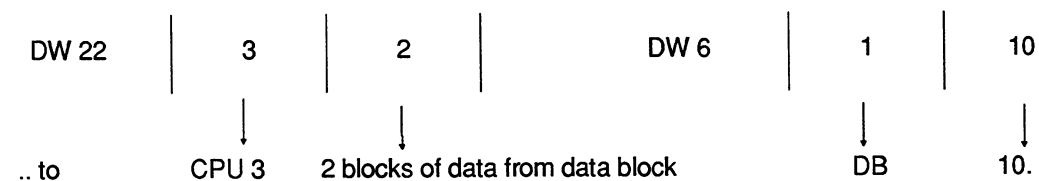
Type of data block containing the data word area to be transferred.

- **DB number**

Number of the data block containing the data word area to be transferred.

As shown in the table, these entries can be read in and completed in lines. If, for example, you want to transfer the first two blocks of data in data block DB 10 from CPU 2 (S2) to CPU 3, make the following entries:

CPU 2 (S 2) sends ..



Sub-list 2 is identical to the assignment ("manual" mode) required for the INITIALIZE function (OB 200). Within the data block, sub-list 1 must occupy data words 0 to 15 and sub-list 2 data words 16 to 31. You must not alter the entries shown in bold face.

8.3.4 Program Structure

During restart, one of the CPUs calls the INITIALIZE function (OB 200) to reserve exactly the same number of coordinator memory fields per connection as blocks of data to be transmitted on this connection.

To send and receive data word areas, each CPU uses two function blocks:

FB no.	Name	Function
FB 100	SEND-DAT	Send data word areas to the other CPUs
FB 101	RECV-DAT	Receive data word areas from the other CPUs

These FB numbers have been selected at random and you can use others.

The function blocks SEND-DAT and RECV-DAT read the connection list to determine which data word areas are to be sent from or received by which data blocks. The **whole** data word area is always sent or received. If this is not possible owing to insufficient transmitting or receiving capacity, the send or receive function is not executed.

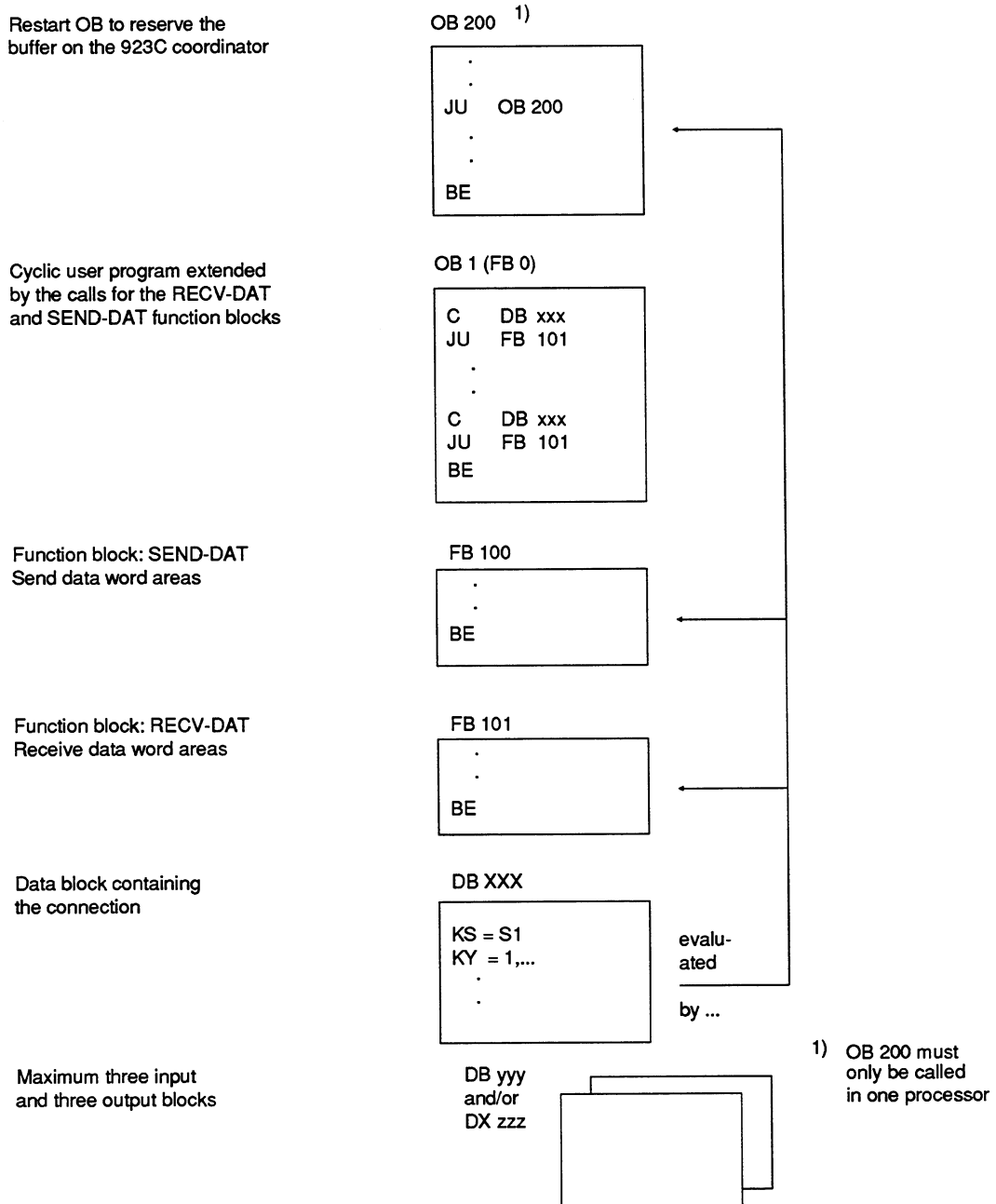


Fig. 3 Overview of the blocks required in each CPU

REMEMBER

The function blocks SEND-DAT and RECV-DAT contain the special function organization blocks for multiprocessor communication OB 202 to OB 205. You cannot call these organization blocks outside SEND-DAT / RECV-DAT.

8.3.5 Sending Data Word Areas (FB 100)

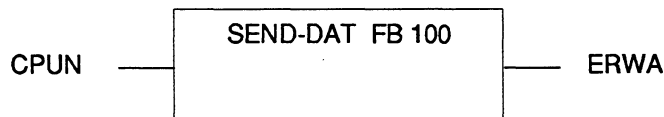
Before you call FB 100, the data block containing the connection list must be open. The function block SEND-DAT requires the number of the CPU on which it is called in order to evaluate the information contained in the connection list.

If the SEND function (OB 202) is not executed correctly in the function block, the error or warning number is transferred to the output parameter ERWA and RLO is set to 1.

If the input parameter CPUN (CPU number) is illegal, ERWA has the value 16 (bit $2^4 = 1$).

The function block SEND-DAT uses flag bytes FY 239 to FY 251 as scratchpad flags.

Parameter name	Significance	Parameter type	Data type
CPUN	Number of the CPU on which FB 100 is called. The numbers 1 to 4 are permitted.	D	KF
ERWA	Error/warning (see SEND function (OB 202))	Q	BY



```

FB 100                                LEN=90
SEGMENT 1                            0000
NAME:SEND-DAT
DECL :CPUN      I/Q/D/B/T/C:      D   KM/KH/KY/KS/KF/KT/KC/KG:  KF
DECL :ERWA      I/Q/D/B/T/C:      Q   BI/BY/W/D:      BY

000B      :LW  =CPUN                CPUN = CPUN -1
000C      :L   KB 1                ERROR IF:
000D      :-F
000E      :JM  =ERWA                CPU NO. <1
000F      :L   KB 3
0010      :>F
0011      :JC  =ERWA                CPU NO. >4
0012      :TAK
0013      :
0014      :SLW      2                CPUN = CPUN * 4
0015      :T   FY 245                BASE ADDRESS
0016      :
0017      :L   KB 1
0018      :T   FY 244                CONNECTION COUNTER
0019      :
001A LOOP :L   FY 245                BASE ADDRESS
001B      :L   FY 244                + COUNTER
001C      :+F
001D      :T   FW 240
001E      :ADD BN+16                + OFFSET
001F      :T   FW 242
0020      :
0021      :DO   FW 242
0022      :L   DR 0                NUMBER OF RESERVED
0023      :T   FY 239                FIELDS = 0 ?
0024      :L   KB 0
0025      :!=F
0026      :JC  =EMPT
0027      :
0028      :DO   FW 242
0029      :L   DL 0                NO. OF THE RECEIVING CPU
002A      :T   FY 246
002B      :L   KB 246
002C      :JU  OB 203                SF OB:
002D      :L   FY 248                TEST TRANSMITTING CAPACITY
002E      :JC  =OBER                ABORT IF ERROR
002F      :
0030      :L   FY 249                TRANSMITTING CAPACITY >< NO.
0031      :L   FY 239                OF RESERVED FIELDS ?
0032      :><F
0033      :JC  =EMPT
0034      :
0035      :L   KB 0                FIELD COUNTER
0036      :T   FY 249
0037      :
0038      :DO   FW 240
0039      :L   DW 0                TYPE AND NUMBER OF THE
003A      :T   FW 247                SOURCE DB

```

Applications

003B	:		
003C	TRAN	:L	KB 246
003D		:JU	OB 202
003E		:L	FY 250
003F		:JC	=OBER
0040	:		
0041		:L	FY 249
0042		:I	1
0043		:T	FY 249
0044		:L	FY 239
0045		:<F	
0046		:JC	=TRAN
0047	:		
0048	EMPT	:L	FY 244
0049		:I	1
004A		:T	FY 244
004B		:L	KB 4
004C		:<F	
004D		:JM	=LOOP
004E		:L	KB 0
004F		:T	=ERWA
0050		:BEU	
0051	:		
0052	ERWA	:L	KB 16
0053	OBER	:T	=ERWA
0054		:BE	

SF OB:
SEND A BLOCK OF DATA
ABORT IF ERROR/WARNING

BLOCK NO. = BLOCK NO. + 1

ALL BLOCKS OF DATA TRANSFERRED ?

INCREMENT
CONNECTION COUNTER

ALL THREE CONNECTIONS
PROCESSED ?

REGULAR PROGRAM END:
RLO = 0, ERWA = 0

PROGRAM END IF ERROR:
RLO = 1, ERWA CONTAINS
ERROR/WARNING NUMBER

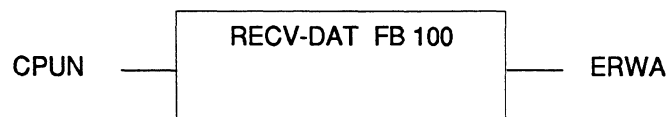
8.3.6 Receive Data Word Areas (FB 101)

Before you call FB 101, the data block containing the connection list must already be open. The function block RECV-DAT requires the number of the CPU in which it is called in order to evaluate the information contained in the connection list.

If the RECEIVE function (OB 204) is not correctly processed within the function block, the corresponding error or warning number is transferred to the output parameter ERWA and the RLO is set to 1. If the input parameter CPUN is illegal, ERWA has the value 16 (bit $2^4 = 1$).

The RECV-DAT function block uses flag bytes FY 242 to FY 255 as scratchpad flags.

Parameter name	Significance	Parameter type	Data type
CPUN	Number of the CPU on which FB 101 is called. The numbers 1 to 4 are permitted.	D	KF
ERWA	Error/warning (see RECEIVE function (OB 204)).	Q	BY



Applications

```

FB 101                               LEN=88
SEGMENT 1                            0000
NAME:RECV-DAT
DECL :CPUN      I/Q/D/B/T/C:      D   KM/KH/KY/KS/KF/KT/KC/KG:  KF
DECL :ERWA      I/Q/D/B/T/C:      Q   BI/BY/W/D:          BY

000B      :LW  =CPUN                ERROR IF:
000C      :L   KB 1
000D      :<F
000E      :JC  =ERWA                CPU NO.<1
000F      :LW  =CPUN
0010      :L   KB 4
0011      :>F
0012      :JC  =ERWA                CPU NO.>4
0013      :
0014      :L   KB 1                CONNECTION COUNTER
0015      :T   FY 242
0016      :
0017      :L   KB 16
0018      :T   FW 244                POINTER TO SUB-LIST 2
0019      :
001A SRCH :L   FW 244                SEARCH SUB-LIST 2 UNTIL THE
001B      :I   1                    NEXT ENTRY FOR THE RECEIVING
001C      :T   FW 244                CPU WITH THE NUMBER "CPUN"
001D      :DO  FW 244                IS FOUND
001E      :L   DL 0
001F      :LW  =CPUN
0020      :><F
0021      :JC  =SRCH
0022      :
0023      :DO  FW 244
0024      :L   DR 0                NUMBER OF RESERVED
0025      :T   FY 243                MEMORY FIELDS = 0 ?
0026      :L   KB 0
0027      :!=F
0028      :JC  =EMPT
0029      :
002A      :L   FW 244                DETERMINE THE NUMBER OF THE
002B      :L   KM 00000000 00001100 TRANSMITTING CPU FROM THE
002D      :AW
002E      :SRW 2                    POINTER TO SUB-LIST 2
002F      :I   1
0030      :T   FY 246
0031      :
0032      :L   KB 246                SF OB:
0033      :JU  OB 205                TEST RECEIVING CAPACITY
0034      :L   FY 248
0035      :JC  = OBER                ABORT IF ERROR
0036      :
0037      :L   FY 249                RECEIVING CAPACITY = NUMBER
0038      :L   FY 243                OF RESERVED MEMORY FIELDS?
0039      :><
003A      :JC  =EMPT
003B      :

```

003C RECV :L KB 246
 003D :JU OB 204
 003E :L FY 248
 003F :JM =OBFE
 0040 :
 0041 :L FY 249
 0042 :L KB 0
 0043 :><F
 0044 :JC =RECV
 0045 :
 0046 EMPT :L FY 242
 0047 :I 1
 0048 :T FY 242
 0049 :L KB 4
 004A :<F
 004B :JM =SRCH
 004C :L KB 0
 004D :T =ERWA
 004E :BEU
 004F :
 0050 ERWA :L KB 16
 0051 OBER :T =ERWA
 0052 :BE

SF OB:
 RECEIVE A BLOCK OF
 DATA
 ABORT IF ERROR/
 WARNING
 IF RECEIVING CAPACITY = 0,
 PROCESS NEXT
 CONNECTION

 INCREMENT
 CONNECTION COUNTER

 ALL CONNECTIONS
 PROCESSED ?

 REGULAR PROGRAM END:
 RLO = 0, ERWA = 0

 PROGRAM END IF ERROR:
 RLO = 1, ERWA CONTAINS
 ERROR/WARNING NUMBER

8.3.7 Application Example (for S5-135U)

You want to exchange data between three CPUs:

- From CPU 1 to CPU 2: data block DB 3, DW 0 to DW 127 (= 4 blocks of data)
- From CPU 1 to CPU 3: data block DX 4, DW 0 to DW 63 (= 2 blocks of data)
- From CPU 2 to CPU 1 and CPU 3: data block DB 5, DW 0 to DW 95 (= 3 blocks of data)

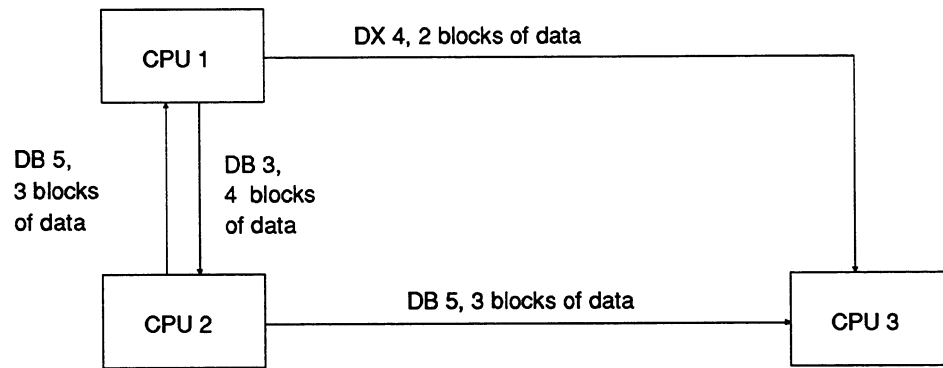


Fig. 4 Data exchange between 3 CPUs

Function block FB 0 is the interface for the cyclic user program on all three CPUs. CPU 1 calls the INITIALIZE function (OB 200) during the cold restart. The connection list is in data block DB 100.

The following blocks must be loaded in the individual CPUs:

Function	CPU 1	CPU 2	CPU 3
Restart OB	OB 20	—	—
User program	FB 0	FB 0	FB 0
FB: SEND-DAT	FB 100	FB 100	FB 100
FB: RECV-DAT	FB 101	FB 101	FB 101
Connection list	DB 100	DB 100	DB 100
Input DB	DB 5	DB 3	DB 5; DX 4
Output DB	DB 3; DX 4	DB 5	—

First of all the connection list (structure described in the section "Data structure") must be written and entered in DB 100:

DB100

LEN=37 ABS
PAGE 1

— Sub-list 1 —

0 :	KS	=S1	Send from CPU 1 to ..
1 :	KY	=001,003;	.. CPU 2 (DB 3)
2 :	KY	=002,004;	.. CPU 3 (DX 4)
3 :	KY	=000,000;	
4 :	KS	=S2	Send from CPU 2 to ..
5 :	KY	=001,005;	.. CPU 1 (DB 5)
6 :	KY	=001,005;	.. CPU 3 (DB 5)
7 :	KY	=000,000;	
8 :	KS	=S3	
9 :	KY	=000,000;	
10 :	KY	=000,000;	
11 :	KY	=000,000;	
12 :	KS	=S4	
13 :	KY	=000,000;	
14 :	KY	=000,000;	
15 :	KY	=000,000;	

— Sub-list 2 —

16 :	KS	=S1	Send from CPU 1 to ..
17 :	KY	=002,004;	.. CPU 2 (four blocks of data)
18 :	KY	=003,002;	.. CPU 3 (two blocks of data)
19 :	KY	=004,000;	
20 :	KS	=S2	Send from CPU 2 to ..
21 :	KY	=001,003;	.. CPU 1 (three blocks of data)
22 :	KY	=003,003;	.. CPU 3 (three blocks of data)
23 :	KY	=004,000;	
24 :	KS	=S3	
25 :	KY	=001,000;	
26 :	KY	=002,000;	
27 :	KY	=004,000;	
28 :	KS	=S4	
29 :	KY	=001,000;	
30 :	KY	=002,000;	
31 :	KY	=003,000;	

Applications

Data words DW 16 to DW 31 contain the assignment list required for the manual INITIALIZATION function (OB 200). OB 200 is called by the OB 20 shown below in CPU 1 during the restart.

OB 20
SEGMENT 1

LEN=23 ABS

0000	:L	KB 2	MANUAL INITIALIZATION OF
0001	:T	FY 246	THE PAGES
0002	:		
0003	:L	KY 1,100	THE ASSIGNMENT LIST IS ENTERED
0005	:T	FW 248	IN DB 100 FROM DATA WORD 16
0006	:L	KF+16	ONWARDS
0008	:T	FW 250	
0009	:		
000A	:L	KB 246	SF OB:
000B	:JU	OB 200	INITIALIZE
000C	:		
000D	:AN	F 252.5	BLOCK END IF THERE IS NO
000E	:BEC		INITIALIZATION CONFLICT
000F	:		

The error handling routine is inserted here if an initialization conflict occurs (e.g. stop, output message on printer etc.)

0010 :BE

The user program on each CPU is extended by the RECV-DAT and SEND-DAT call. Function block FB 0 shown below is for CPU 1. For the other CPUs, the input parameter CPUN (CPU number) must be modified.

```

FB0                               LEN=31   ABS
SEGMENT 1
NAME:PROG-1

0005      :C   DB100                CONNECTION LIST      DB 100
0006      :JU  FB101                RECEIVE THE INPUT
                                         DATA BLOCKS

0007 NAME :RECV-DAT
0008 CPUN :    KF+1
0009 ERWA :    FY0
000A      :JC  =ERWA                ABORT IF ERROR/WARNING
000B      :
000C      :

```

The cyclic user program is inserted here and reads data from the input data blocks and writes data to the output data blocks.

```

000D      :
000E      :
000F      :
0010      :C   DB100                CONNECTION LIST      DB 100
0011      :JU  FB100                SEND THE OUTPUT
                                         DATA BLOCKS

0012 NAME :SEND-DAT
0013 CPUN :    KF+1
0014 ERWA :    FY0
0015      :JC  =ERWA                ABORT IF ERROR/WARNING
0016      :BEU
0017      :
0018 ERWA :                AFTER ERROR/WARNING
0019      :BE                      EXECUTE ERROR HANDLING

```

The error handling is inserted here, (e.g. stop, output error message on printer or monitor, etc.)

REMEMBER



This example (IPC flag extension using function blocks SEND-DAT and RECV-DAT) can only be performed correctly if the special function organization blocks for multiprocessor communication OB 202 to OB 205 are not called outside these function blocks in any of the CPUs.