SIEMENS

SIMATIC S5

S5-155U
CPU 948

Programming Guide

This manual has the order number:
6ES5 998–3PR21

Contents

The List of Operations, order no. 6ES5 997-3UA22, is included with this manual.

10/98
C79000-G8576-C848
Release 04

Safety 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:

Danger

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

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.

Note

draws your attention to particularly important information on the product, handling the product, or to a particular part of the documentation.

Qualified Personnel

The device/system may only be set up and operated in conjunction with this manual.
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 circuits, equipment, and systems in accordance with established safety practices and standards.

Correct Usage

Note the following:

Warning

This device and its components may only be used for the applications described in the catalog or the technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored, set up, and installed correctly, and operated and maintained as recommended.

Trademarks

SIMATIC®, SIMATIC NET® and SIMATIC HMI® are registered trademarks of SIEMENS AG. Third parties using for their own purposes any other names in this document which refer to trademarks might infringe upon the rights of the trademark owners.

Siemens AG
Bereich Automatisierungs- und Antriebstechnik
Geschaeftsgebiet Industrie-Automatisierungssysteme
Postfach 4848, D-90327 Nuernberg

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.

Contents

Introduction 1

Contents of Chapter 1

Introduction 1

Aims of the manual

This manual is intended to provide specialized information about programming the CPU 948 for users who already have basic knowledge of programming PLCs and want to use the CPU 948 in the S5-155U programmable controller. If you do not yet have this basic knowledge, we strongly advise you read the documentation introducing the programming language STEP 5 /3/ or take part in a course at our training center. SIEMENS provides comprehensive training for SIMATIC S5. For more detailed information, contact your local SIEMENS office.

Contents of Chapter 1

Chapter 1 explains how to use the manual and deals with the areas of application of the S5-155U programmable controller with the CPU 948 and its structure.
The chapter explains the typical mode of operation of a CPU and the structure of the CPU program.
You will also find a few suggestions about how to tackle programming and will learn some of the features of the CPU 948 which are important for programming.
If you have already worked with the CPU 946/947 and would like to know the differences between these modules and the CPU 948, refer to Section 1.8.

Chapter 1 also informs you about differences between versions A01 and A02 of the CPU 948 and explains points you should remember when converting "928B" programs for the CPU 948.

1.1 Area of Application for the S5-155U with the CPU 948

SIMATIC S5 family

The S5-155U programmable controller belongs to the family of SIMATIC S5 programmable controllers. With the CPU 948, it is the most powerful multiprocessor unit for process automation (open and closed loop control, signalling, monitoring, logging).
Owing to its modularity and high performance, it can be used for medium to extremely large control systems as well as for complex automation tasks at the plant and process supervision level.

Suitability

The S5-155U with the CPU 948 is particularly suitable for the following:

- Tasks requiring fast bit and word-oriented processing and fast reaction times, i.e. with extremely fast open and closed loop controls. Examples of this are fast processes in mechanical engineering (bottling plant, packing machines or similar systems) and in the automobile industry.

- Tasks requiring an extremely high storage capacity and fast access times, e.g. in the automobile industry, process and plant engineering.

- Tasks requiring fast communication with other CPUs installed in the PLC and operating in the multiprocessor mode and with CP modules (e.g. when connected to bus systems, host computers, for visualization, operation and monitoring).

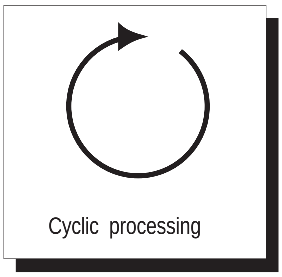
- Complex tasks which can be handled efficiently and clearly using the high level languages C and SCL.

1.2 Typical Mode of Operation of a CPU

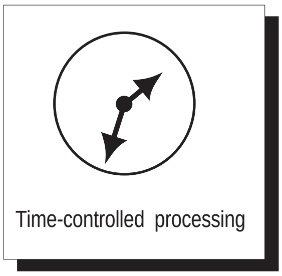
Mode of operation of a CPU

The following modes of operation are possible in a CPU:

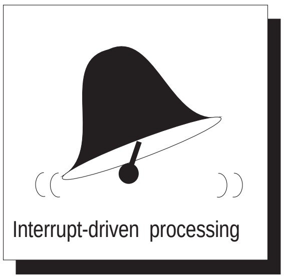
1.



2.



3.

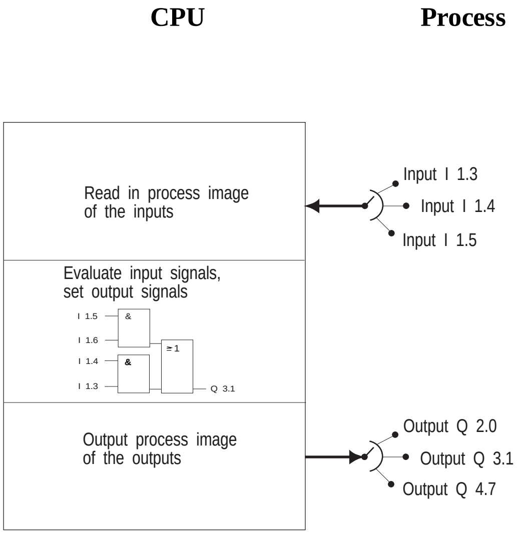


Cyclic processing

This is the main part of all activities in the CPU. As the name already says, the same operations are repeated in an endless cycle.

Cyclic processing can be divided into three main phases, as follows:

Phase	Sequence
1	All the input modules assigned to the CPU are scanned by the system program and the values read in are stored in the process image of the inputs (PII).
2	The values contained in the PII are processed by the user program and the values to be output are entered in the process image of the outputs (PIQ).
3	The values contained in the process image of the outputs are output by the system program to the output modules assigned to the CPU.


CPU
Process
Read in process image of the inputs
Input I 1.3
Input I 1.4
Input I 1.5
Evaluate input signals, set output signals
I 1.5
I 1.6
I 1.4
I 1.3
&
&
≥1
Q 3.1
Output process image of the outputs
Output Q 2.0
Output Q 3.1
Output Q 4.7

Time-controlled processing

In addition to the cyclic processing, **time-controlled** processing is also available for processes requiring control signals at constant intervals, e.g. non-time critical monitoring functions performed every second.

Interrupt-driven processing

If the reaction to a particular process signal must be particularly fast, this should be handled with **interrupt-driven** processing. With, for example, a system interrupt, triggered via an interrupt generating module, you can activate a special processing section within your program.

Processing according to priority

The types of processing listed above are handled by the CPU according to their **priority**.

Since a fast reaction is required to a time or interrupt event, the CPU interrupts cyclic processing to handle a time or interrupt event. Cyclic processing therefore has the lowest priority.

Whether or not the time-controlled processing is more important than the interrupt controlled processing depends, among other things, on the particular task. For this reason, the priority of time and interrupt-driven processing on the CPU 948 can be selected.

1.3 The Programs in a CPU

The program existing on every CPU is divided into the following:

- the **system program**

  and

- the **user program**.

System program

The system program organizes all the functions and sequences of the CPU which do not involve a specific control task (refer to Fig. 1-2).

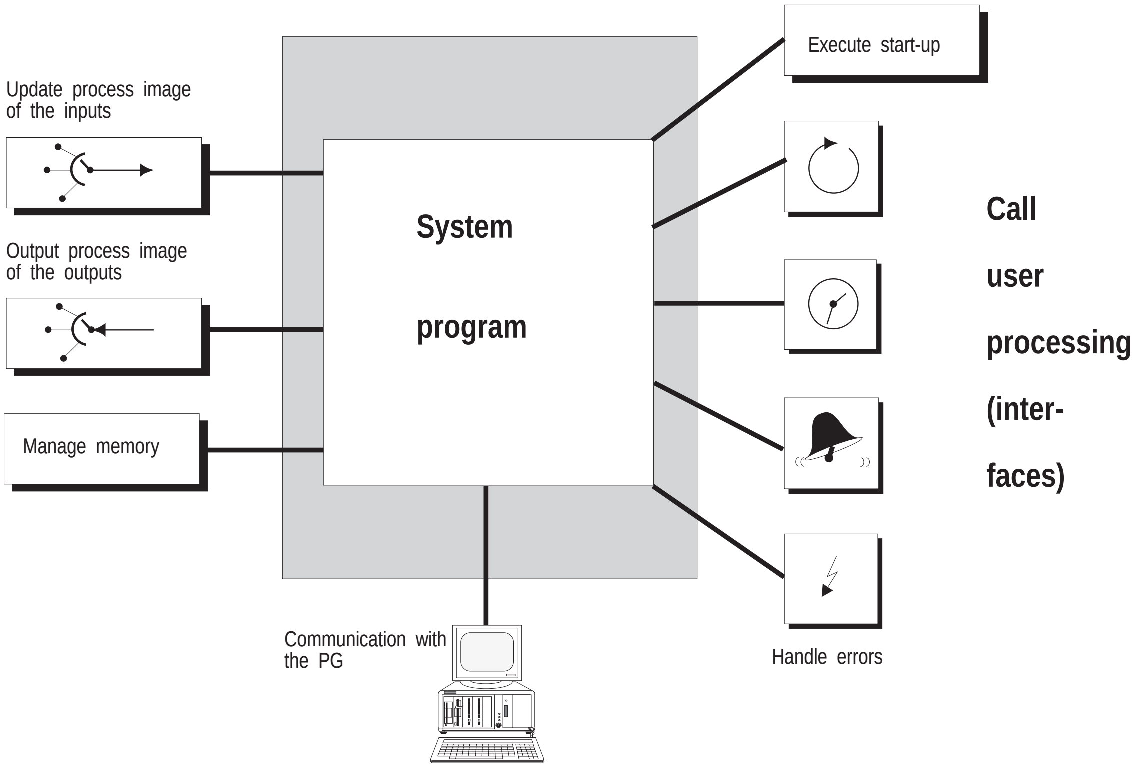

Execute start-up
Update process image of the inputs
Output process image of the outputs
Manage memory
System program
Call user processing (inter-faces)
Communication with the PG
Handle errors

Fig. 1-1 Tasks of the system program

Tasks

The tasks include the following: [1]

- cold and warm restart,
- updating the process image of the inputs and outputting the process image of the outputs,
- calling the cyclic, time-controlled and interrupt-driven programs,
- detection and handling of errors,
- memory management,
- communication with the programmer (PG).

User interfaces

As the user, you can influence the reaction of the CPU to particular situations and errors via special interfaces to the system program.

Storing the system program

After switching on the power supply to the PLC (POWER UP) the system program is read from the EPROM to the internal operating system RAM.

System program defaults

The following chapters, except for Chapter 7, describe the **default system reaction** to process events or errors. Depending on the defaults, the CPU changes to the stop mode if an operation code error occurs and the error organization block is not loaded.

Modifying the defaults

You can modify the system response by assigning parameters for the data block DX 0.
Chapter 7 describes the system response **following modification**.

[1] When operating with several CPUs (multiprocessing) further tasks are involved.

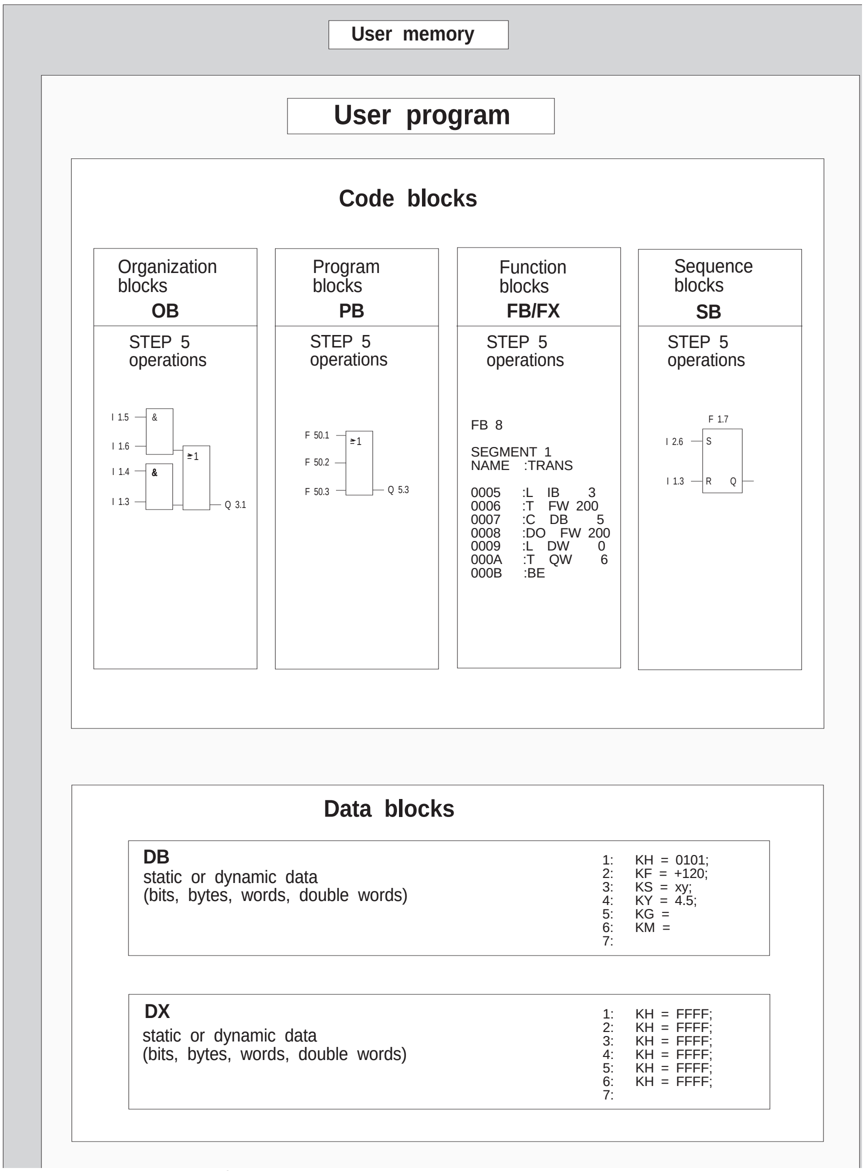
User program

Tasks

The user program contains all the functions required for processing a **specific control task**. In general terms, these functions can be assigned to the interface provided by the system program for the various types of processing, as follows:

Type of processing	Task
Cold and warm restart	To provide the conditions under which the other processing functions can start from a defined status following a cold or warm restart of the control system (e.g. assigning specific values to signals).
Cyclic processing	Constantly repeated signal processing (e.g. logic operations on binary signals, reading in and analyzing analog values, specifying binary signals for output, outputting analog values).
Time-controlled processing	Special, time-dependent processing with the following time conditions: - faster than the average cycle, - at a time interval greater than the average cycle time, - at a specified point in time.
Interrupt-driven processing	Special, fast reactions to certain process signals.
Error reaction	Handling problems within the normal sequence of the program.

Structure



```
FB 8

SEGMENT 1
NAME   :TRANS

0005   :L   IB     3
0006   :T   FW  200
0007   :C   DB     5
0008   :DO  FW  200
0009   :L   DW     0
000A   :T   QW     6
000B   :BE
```

```
1:   KH = 0101;
2:   KF = +120;
3:   KS = xy;
4:   KY = 4.5;
5:   KG =
6:   KM =
7:
```

```
1:   KH = FFFF;
2:   KH = FFFF;
3:   KH = FFFF;
4:   KH = FFFF;
5:   KH = FFFF;
6:   KH = FFFF;
7:
```

Fig. 1-2 Structure of a STEP 5 user program

Storing the user program

After programming the user program, you must load it in the user memory of the CPU 948 (directly from the PG) or via a memory card whose contents are copied to the user memory by an OVERALL RESET of the CPU.

Interfaces to the system program

Organization blocks are available as interfaces to the system program for the special types of processing.

1.4 Which Operands are available to the User Program?

The CPU 948 provides the following operand areas for programming:

- process image and I/Os
- flags (F flags and S flags)
- timers/counters
- data blocks

Process image of the inputs and outputs PII/PIQ

Characteristics	Size
The user program can access the following data types in the process image extremely quickly: - single bits, - bytes, - words, - double words	128 bytes each for inputs and outputs

I/O area (P area)

Characteristics	Size
The user program can access the I/O modules directly via the S5 bus. The following data types are possible: - bytes, - words.	256 bytes each for inputs and outputs

Extended I/O area (O area)

Characteristics	Size
The user program can access the I/O modules directly via the S5 bus. The following data types are possible: - bytes, - words.	256 bytes each for inputs and outputs

F flags

Characteristics	Size
The flag area is a memory area which the user program can access extremely quickly with certain operations. The flag area should be used ideally for working data required often. The following data types can be accessed: - single bits, - bytes, - words, - double words. Single flag bytes can be used as **interprocessor communication flags** (IPC flags) to exchange data between the CPUs in the multiprocessor mode (refer to Chapter 10). IPC flags are updated by the system program at the end of the cycle via a buffer in the coordinator or CP/IP.	2048 bits

S flags (extended flag area)

Characteristics	Size
The CPU 948 also contains an additional flag area, the S flag area. The user program can also access this area extremely quickly as with the F flags. S flags **cannot** however by used as **actual operands** with function block calls nor as **IPC flags** for data exchange between the CPUs. The bit test operations of the CPU 948 can also not be used with the S flags. These flags can only be used with the PG system software "S5-DOS" from version 3.0 upwards or "S5-DOS/MT" from version 1.0 upwards.	32 768 bits

Timers (T)

Characteristics	Size
The user program loads timer cells with a time value between 10 ms and 9990 s and by means of a start operation, decrements the timer from this value at the preselected intervals until it reaches the value zero.	256 timer cells

Counters (C)

Characteristics	Size
The user program loads counter cells with a start value (max. 999) and then increments or decrements them.	256 counters

Data words in the current data block

Characteristics	Size
A data block contains constants and/or variables in the byte, word or double word format. With STEP 5 operations, you can always access the "current" data block (refer to Section 2.4.2). The following data types can be accessed: - single bits, - bytes, - words, - double words.	256 words 1)

1) In data blocks with a length greater than 256 words, you can only access data words with the numbers > 255 with operations for absolute memory access (refer to Chapter 9).

1.5 How much Memory is available for the User Program?

For storing logic and data blocks, the CPU 948 only has the user memory in the internal RAM.

The CPU 948 is available with two versions of the user memory:

- Version 1: with 640 Kbytes,

- Version 2: with 1,664 Kbytes.

1.6 How to Tackle Programming

If you are an experienced user, you have probably found the most suitable method for creating programs for yourself and you can skip this section.

Less experienced readers will find tips for designing, programming, testing and starting up your STEP 5 program.

Implementation stages

The implementation of the STEP 5 control program can be divided into three stages:

Stage	Activity
1	Determining the technological task
2	Designing the program
3	Creating, testing and starting the program

Recursive procedure

In practice, you will recognize that certain steps must be repeated (recursive procedure), e.g. when you realize that more signals are required to improve the handling of the task.

Stage 1

Determining the technological task:

Stage	Activity
1	Create a general block diagram outlining the control tasks of your process.
2	Create a list of the input and output signals required for the task.
3	Improve the block diagram by assigning the signals and any particular time conditions and/or counter statuses to the individual blocks.

Stage 2

Designing the program

Stage	Activity
1	Based on the improved block diagram, decide on the types of processing required of your program (cyclic processing, time-controlled processing etc.) and select the OBs required for this.
2	Divide the types of processing into technological and/or functional units.
3	Check whether the units can be assigned to a program or function block and select the blocks you require (PB x, FB y etc.)
4	Find out which timers, counters and data or results memory you require.
5	Specify the tasks for each of the proposed logic blocks and the data for flags and data blocks which may be required. Create flow diagrams for the logic blocks.

Notes on the scope of cyclic processing

When deciding on the types of processing, keep the following conditions in mind:

- The cycle must run through quickly enough. The process statuses must not change more quickly than the CPU can react. Otherwise the process can get out of control.

- The maximum reaction time should be taken as twice the cycle time.
  The cycle time is determined by the cyclic processing of the system program and the type and scope of the user program. It is often not constant, since the cyclic user program may be interrupted when time and interrupt-driven program sections are called.

Stage 3

Creating, testing and starting up the program:

Stage	Activity
1	Decide on the type of representation for the logic blocks (LAD, CSF or STL, refer to Chapter 2). Remember that function blocks can only be created in the STL method of representation.
2	Program all logic and data blocks (please refer to your STEP 5 manual).
3	Start up the blocks one after the other (you may have to program a different OB for each individual step, to call the logic blocks): 1a: load the block(s) 1b: test the block(s) (For more detailed information please refer to your STEP 5 manual and Chapter 11).
4	When you are certain that all the logic blocks run correctly and all the data can be correctly calculated and stored, you can start up your whole program.

Note on test strategies

When you actually start up your program for the first time in genuine process operation, i.e. with real input and more importantly output signals, is a decision that must be left up to yourself or to a team of experts.
The more complex the process, the greater the risk and therefore the greater the care required when starting up.

1.7 Programming Tools

Suitable PGs

The following programmers are available for creating your user program, PG 685, PG 710, PG 730, PG 750 and PG 770. You can check on the performance and characteristics of these devices in the catalog ST 59 /9/.

> **Note**
> If you wish to use the **full range of performance** of the CPU 948 in your automation software, (particularly the DX 0 screen, the "Output ISTACK" screen, the display with the "memory configuration" function and the PG functions via the backplane bus) you require the PG system software **"STEP 5/ST"** from version 6.3 upwards or "STEP 5/MT" from version 6.0 upwards plus the "Delta diskette CPU 948" and a PG 7xx.

Suitable software

You can create **user programs for SIMATIC S5 programmable controllers** as follows:

- In the **STEP 5** programming language,

  Here you require the STEP 5 programming package along with the system software STEP 5/ST or STEP 5/MT (description, refer to /3/ in Further Reading),

  or

- In a higher programming language:

  If you are familiar with programming in higher programming languages, you can also formulate your STEP 5 program for the CPU 948 as follows:

  - **SCL** (refer to /12/ in Further Reading, the SCL compiler is contained in the PG software "S5-DOS/MT" from version 6 upwards.)
    or

  - **C with S5 C compiler** (refer to /13/ in Further Reading).

You can also create **programs for sequence control systems** in a graphic representation using the **GRAPH 5** programming package (description, refer to /4/ in Further Reading).

Depending on the task, you can also incorporate "off-the-peg" standard function blocks in your user program. The performance and characteristics of these blocks are described in the catalog ST 57 /11/.

1.8 Converting User Programs of the CPU 928B for the CPU 948

The following section informs you about the points you should remember when you convert user programs written for the CPU 928B for use on the CPU 948.

Operations

In the following operations, note the differences in the execution and handling (among other things the different memory utilization).

Operations	CPU 928B	CPU 948
IA/RA (disable/enable interrupts)	All process interrupts are disabled or enabled	Only the process interrupts via input byte IB 0 are disabled or enabled. Instead of these operations, use the special function OBs OB 122 or OB 142.
LIR/TIR	16 bit long addresses are used.	20 bit long addresses are used. Adaptation is necessary.
Block transfer operation TNB	16 bit long addresses are used.	The operation does not exist. Use TNW for block transfer from the 8-bit to the 8-bit area.
Block transfer operation TNW	- 16 bit long addresses are used. - Block transfers from the 8-bit to the 8-bit area and vice versa are possible.	- 20 bit long addresses are used. Adaptation is necessary. - only block transfers from the 8-bit to the 8-bit area and from the 16-bit to the 16-bit areas possible with TNW. - for the block transfer from the 8-bit to the 16-bit area use the operation **TXB**, - for the block transfer from the 16-bit to the 8-bit area use the operation **TXW.** (TXB and TXW do not exist on a CPU 928B)
All operations with the BR register	The BR register is 20 bits wide.	The BR register is 32 bits wide. Adaptation is necessary.

Timer processing

CPU 928B	CPU 948
The timers are updated during start-up.	The timers are only updated in the RUN mode (Reason: compatibility with CPU 946/947)

FB 0 as cycle block

CPU 928B	CPU 948
If no cycle block OB 1 exists, the system program calls FB 0 cyclically, provided it is loaded.	Only OB 1 can be used for cyclic processing. If you have programmed FB 0, create an OB 1 in which FB 0 is called.

Default priorities

CPU 928B	CPU 948
Process interrupts have higher priority than timed interrupts.	Timed interrupts have priority over process interrupts via IB 0 or system interrupts. You can change the priority with the parameters in DX 0.

Data block DB 0 (block address list)

CPU 928B	CPU 948
The block address list contains the **direct start addresses** of the blocks.	The block address list contains the segment addresses of the blocks. To obtain the start address of a block, its segment address must be shifted 4 bits to the left.

Data block DX 0

You must create a new DX 0 data block (see Chapter 7), since the DX 0 for the CPU 928B has a different structure and settings.

Using the RT area

With the CPU 928B, the RT area is not used by the system program, with the CPU 948 it is used to some extent by the handling blocks. You can only use the RT area for your user program when you do not use any standard FBs and any PG functions via SINEC H1 and the S5 bus.

Organization blocks

The number and function of the error and special function OBs are not the same on the CPU 928B and CPU 948:

Error OBs

The following error OBs of the CPU 948 respond differently from their namesakes on the CPU 928B:

OB	Function	Error IDs
OB 19 OB 26 OB 27	Same as on CPU 928	Different from CPU 928B
OB 28 OB 29 OB 30 OB 31	Function different from that on CPU 928B	–

Special function OBs

OB	Note
OB 110 OB 152 OB 160 to 163 OB 170 OB 190 to 193 OB 216 to 218 OB 220 and 221 OB 224 OB 226 to 228 OB 240 to 242 OB 250 and 251	These OBs do not exist on the CPU 948
 OB 111 OB 112 OB 113 OB 120 OB 121 OB 122 OB 123	On the CPU 948, the OB is replaced by : OB 131 OB 132 OB 133 OB 122 OB 141 OB 142 OB 143

OB	Note
OB 122	Parameter assignment is different from that of the CPU 928B OB 122 (reason: compatibility with CPU 946/947).
OB 180	In contrast to the CPU 928B, the access window of the CPU 948 can only be shifted by a multiple of 16.
OB 200 OB 202 to 205 (multiprocessor communication)	In contrast to the CPU 928B, these CPU 948 OBs change the content of **ACCU 4**.

R64 controller software

The R64 controller software cannot be run on the CPU 948.

Standard FBs

Generally, the standard function blocks used on the CPU 928B (e.g. for IPs) must be replaced by those for the CPU 948. The HDBs are an exception these can be taken from the CPU 928B (see Section 1.8.1).

User Program

2

Contents of Chapter 2

User Program 2

The following chapter explains the components that make up a STEP 5 user program for the CPU 948 and how it can be structured.

2.1 STEP 5 Programming Language

With the STEP 5 programming language, you convert automation tasks into programs that run on SIMATIC S5 programmable controllers. You can program simple binary functions, complex digital functions and arithmetic operations including floating point arithmetic using STEP 5.

Types of operation

The **operations** of the STEP 5 programming language are divided into the following groups:

Basic operations

- you can use these operations in all logic blocks
- methods of representation: ladder diagram (LAD), control system flowchart (CSF), statement list (STL).

Supplementary operations and system operations:

- can only be used in function blocks
- only statement list (STL) method of representation
- system operations: only experienced STEP 5 programmers should use system operations

2.1.1 The LAD, CSF, STL Methods of Representation

When programming in STEP 5, you can choose between the three methods of representation ladder diagram (LAD), control system flowchart (CSF) and statement list (STL) for each individual logic block. You can choose the method of representation that best suits your particular application.

The machine code MC5 that the programmers (PGs) generate is the same for all three methods of representation.

If you follow certain rules when programming in STEP 5 (see /3/), the programmer can translate your user program from one method of representation into any other.

Graphic representation or list of statements

While the ladder diagram (LAD) and control system flowchart (CSF) methods of representation represent your STEP 5 program graphically, statement list (STL) represents STEP 5 operations individually as mnemonic abbreviations.

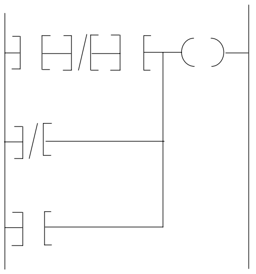
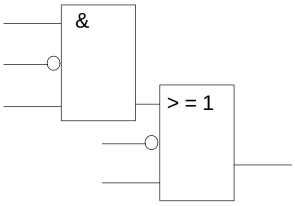
Ladder diagram	Statement list	Control system flowchart
Programming with graphic symbols like a circuit diagram	Programming with mnemonic abbreviations of function designations	Programming with graphic symbols
complies with DIN 19239	complies with DIN 19239	complies with IEC 117-15 DIN 40700 DIN 40719 DIN 19239
LAD 	STL A I AN I A I ON I O I = Q	CSF 

Fig. 2-1 Methods of representation in the STEP 5 programming language

Graphic representation of sequential controls

GRAPH 5 /4/ is a programming language for graphic representation of sequential controls. It is at a higher level than the LAD, CSF, STL methods of representation. A program written in GRAPH 5 as a graphic representation is automatically converted to a STEP 5 program by the PG.

2.1.2 Structured Programming

Using STEP 5, you can structure your program by dividing it into self-contained program sections (blocks). This division of your program clarifies the essential program structures making it easy to recognize the system parts that are related within the software.

Structured programming offers you the following advantages:

- simple and clear creation of programs, even large ones
- standardization of program parts
- simple program organization
- easy program changes
- simple, section by section program test
- simple system start-up

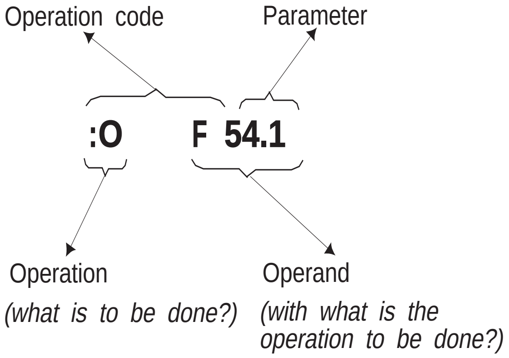
What is a block?

A block is a part of the user program that is distinguished by its function, structure or application. You can differentiate between blocks that contain **statements** (code) i.e. organization blocks, program blocks, function blocks or 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 the user program. It is the work specification for the CPU. A STEP 5 operation consists of an operation and an operand as shown in the following example:

Example



Absolute and symbolic operands

You can enter the operand **absolutely** or **symbolically** (using an assignment list) as shown in the following example:

Absolute representation:	:A I 1.4
Symbolic representation:	:A -Motor1

For more information on absolute and symbolic programming, refer to your STEP 5 manual.

Application of STEP 5 operations

The STEP 5 operation set enables you to do the following:

- set or reset and combine binary values logically
- load and transfer values
- compare values and process them arithmetically
- specify timer and counter values
- convert number representations
- call blocks and execute jumps within a block

  and

- influence program execution

Result of logic operation RLO

The central bit for controlling the program is the result of logic operation RLO. This is obtained as a result of binary logic operations and is influenced by some operations.
Section 3.5 describes the whole STEP 5 operation set and explains how the RLO is obtained. This section also includes programming examples for individual STEP 5 operations.

2.1.4 Number Representation

To allow the CPU to logically combine, modify or compare numerical values, these values must be located in the accumulators (working registers of the CPU) as binary numbers.
Depending on the operations to be carried out, the following number representations are permitted in STEP 5:

Binary numbers:	16-bit fixed point numbers
	32-bit fixed point numbers
	32-bit floating point numbers (with a 24-bit mantissa)
Decimal numbers:	BCD-coded numbers (sign and 3 digits)

Numerical input on the PG

When you use a programmer to input or display number values, you set the data format on the programmer (e.g. KF or fixed point) in which you intend to enter or display the values. The programmer converts the internal representation into the form you have requested.

Permitted operations

You can carry out **all arithmetic operations** with the 16-bit fixed point numbers and floating point numbers, including comparison, addition, subtraction, multiplication and division.

Note
Do not use BCD-coded numbers for arithmetical operations, since this leads to incorrect results.

Use 32-bit fixed point numbers to execute comparison operations. These are also necessary as an intermediate level when converting numbers in BCD code to floating point numbers. With the operations +D and -D they can also be used for addition and subtraction.
The STEP 5 programming language also has **conversion operations** that enable you to convert numbers directly to the most important of the other numerical representations.

16-bit and 32-bit fixed point numbers

Fixed point numbers are whole binary numbers with a sign.

Coding of fixed point numbers

Fixed point numbers are 16 bit (= 1 word) or 32 bit (= 2 words) in binary representation. Bit 15 or bit 31 contains the sign.

- ’0’ = positive number

- ’1’ = negative number

The two’s complement representation is used for negative numbers.

PG input

Input of 16-bit fixed point number data format at the PG:KF

Input of 32-bit fixed point number data format at the PG:DH

Permitted numerical range

16-bit fixed point number
-32768 to +32767 (16 bits)

32-bit fixed point number
-2147483648 to +2147483647 (32 bits)
(8000 0000H to 7FFF FFFFH)

Using fixed point numbers

Use fixed point numbers for simple calculations and for comparing number values. Since fixed point numbers are always whole numbers, remember that the result of dividing two fixed point numbers is also a fixed point number without decimal places.

Floating point numbers

Floating point numbers are positive and negative fractions. They always occupy a double word (32 bits). A floating point number is represented as an exponential number. The mantissa is 24 bits long and the exponent is 8 bits long.
In the CPU 948, the default mantissa is 24 bits long (bits 0 to 23) for adding, subtracting, multiplying and dividing.
The exponent indicates the order of magnitude of the floating point number. The sign of the exponent tells you whether the value of the floating point number is greater or less than 0.1.

Using floating point numbers

Use floating point numbers for solving extensive calculations, especially for multiplication and division or when you are working with very large or very small numbers!

Accuracy

The mantissa indicates the accuracy of the floating point number as follows:

- Accuracy with a 24-bit mantissa:

2^{-24} = 0.000000059604 (corresponds to 7 decimal places)

If the sign of the mantissa is "0" the number is positive; if the sign is "1" it is a negative number in its two’s complement representation. The **floating point value ’0’** is represented as the binary value **80000000H** (32 bits, see below).

Coding floating point numbers

Coding a floating point number:

31	30 24	23	22 0
V	2^6 ... . 2^0	V	2^{-1} .... 2^{-23}
Exponent		Mantissa	

Specification of the data format for floating point numbers at the PG: KG

Permissible numerical range

± 0.1469368 x 10^{-38} to ± 0.1701412 x 10^{39}

Input/output on PG

a) in a logic block:

You want to load the number N = 12.34567 as a floating point number.

Input:

:LKG1234567+2

b) in a data block:

PG display after you enter the line:

:L KG + 1234567 + 02

Mantissa with sign Exponent (base 10) with sign

Value of the number input: $+0.1234567 \times 10^{+2} = 12.34567$

You want to define the number N = - 0.005 as a floating point constant.

Input:

6: KG = - 5 - 2

PG display after you enter the line:

6: KG =- 5000000 - 02

Mantissa with sign Exponent (base 10) with sign

Value of the number input : $- 0.5 \times 10^{-2} = 0.005$

Numbers in BCD code

Decimal numbers are represented as numbers in BCD code. With their sign and three digits, they occupy 16 bits (1 word) in an accumulator as shown in the following example:

15 12	11 8	7 4	3 0
V V V V	hundreds	tens	ones

The individual digits are positive 4-bit binary numbers between 0000 and 1001 (0 and 9 decimal).
The left bits are reserved for the sign as follows:

Sign for a positive number:	0000
Sign for a negative number:	1111

Permissible numerical range -999 to +999

2.1.5 STEP 5 Blocks and Storing them in Memory

Identification

A block is identified as follows:

- the block type (OB, PB, SB, FB, FX, DB, DX)

  and

- the block number (number between 0 and 255).

Block types

The STEP 5 programming language differentiates between the following block types:

Organization blocks (OB)

Organization blocks are the interface between the system program and the user program. They can be divided into two groups as follows:

With OB 1 to OB 39, you can control program execution, the restart procedure of the CPU and the reaction in the event of an error. You program these blocks yourself according to your automation task. These OBs are called by the system program.

OBs 40 to 100 are blocks belonging to the operating system. You must not call these blocks.

OBs 121 to 255 contain special functions of the system program. You can call these blocks, if required, in your user program.

Program blocks (PB)

You require program blocks to structure your program. They contain program parts divided according to technological and functional criteria. Program blocks represent the heart of the user program.

Sequence blocks (SB)

Sequence blocks were originally special program blocks for step by step processing of sequencers. In the meantime, however, sequencers can be programmed with GRAPH 5/4/. Sequence blocks have therefore lost their original significance in STEP 5.
Sequence blocks now represent an extension of the program blocks and are used as program blocks.

Function blocks (FB/FX)

You use function blocks to program frequently recurring and/or complex functions (e.g. digital functions, sequence control systems, closed loop controls and signalling functions).

A function block can be called several times by higher order blocks and supplied with new operands (assigned parameters) at each call. Using block type FX doubles the maximum number of possible function blocks.

Data blocks (DB/DX)

Data blocks contain the (fixed or variable) data with which the user program works. This type of block contains no STEP 5 statements and has a distinctly different function from the other blocks. Using block type DX doubles the number of possible data blocks.

Formal structure of the blocks

All blocks consist of the following two parts:

- a block header

  and

- a block body

Block header

The **block header** is always 5 words long and contains information for block management in the PG and data for the system program.

Block body

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

- STEP-5 operations (in OB, PB, SB, FB, FX),

- variable or constant data (in DB, DX)

  and

- a formal operand list (in FB, FX).

Block preheader

The programmer also generates a **block preheader** (DV, DXV, FV, FXV) for block types DB, DX, FB and FX. These block preheaders contain information about the data format (for DB and DX) or the jump labels (for FB and FX). Only the PG can evaluate this information. Consequently the block preheaders are not transferred to the CPU memory. You cannot influence the contents of the block header directly.

Maximum length

A STEP-5 block can occupy a maximum of 32 767 words in the program memory of the CPU (1 word corresponds to 16 bits).

Available blocks

You can program the following block types:

OB	1 to 39	
FB	0 to 255	total 512
FX	0 to 255	
PB	0 to 255	
SB	0 to 255	
DB	2 to 255	total 508
DX	3 to 255	

Data blocks DB 0, DB 1, DB 2, DX 0, DX 1 and DX 2 contain parameters. These are reserved for specific functions and you cannot use them as normal data blocks.
Data block DX 2 is reserved for the 2nd serial interface and you should not use it.

Block storage

The programmer stores all programmed blocks in the program memory in the order in which they are transferred (Fig. 2-2). With the PG function "transfer data blocks A" the logic blocks are transferred first followed by the data blocks.

The start addresses of all stored blocks are placed in an address list in data block DB 0.

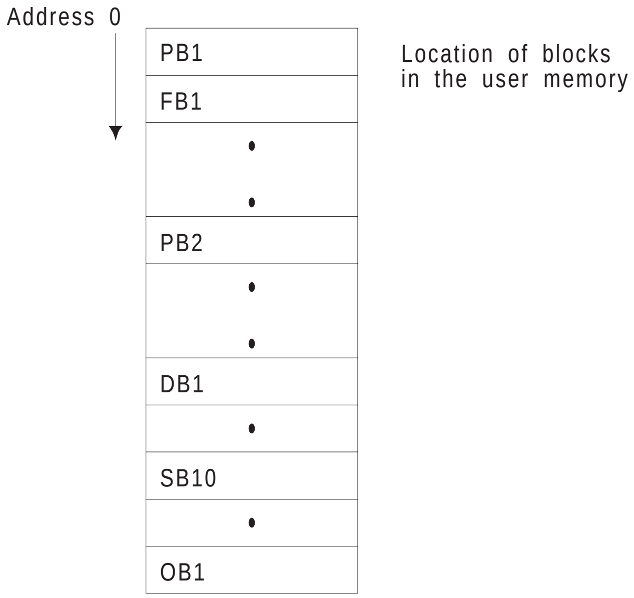


Fig. 2-2 Example of block storage in the user memory

Correcting and deleting blocks

When you **correct** blocks, the old block is declared invalid in the memory and a new block is entered.
Similarly, when blocks are **deleted**, they are not really deleted, instead they are declared invalid. The space they occupy is, however, not released and is not available for blocks loaded later.

Note
You can use the COMPRESS MEMORY online function to make space for new blocks. This function optimizes the utilization of the memory by deleting blocks marked as invalid and shifting valid blocks together.

2.2 Program, Organization and Sequence Blocks

Program blocks (PBs), organization blocks (OBs) and sequence blocks (SBs) are the same with respect to programming and calling. You can program all three types in the LAD, CSF and STL methods of representation.

Programming

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

Step	Action
1	First indicate the type of block and then the number of the block that you want to program. The following numbers are available for the type of block listed: - program blocks 0 to 255 - sequence blocks 0 to 255 - organization blocks 1 to 39
2	Enter your program in the STEP 5 programming language. When programming PBs, OBs and SBs, you can only use the STEP 5 **basic** operations! A STEP 5 block should always be a self-contained program section. Logic operations must always be completed within a block.
3	Complete your program input with the block end operation "BE".

Block calls

With the exception of OB 1 to OB 39 you must call the blocks to process them. Use the special STEP 5 block call operations to call the blocks.

You can program block calls inside an organization, program, function or sequence block. They can be compared with jumps to a subroutine. Each jump causes a block change. The return address within the calling block is buffered by the system.

Block calls can be unconditional or conditional as follows:

Unconditional call

The "JU" statement belongs to the unconditional operations. It has no effect on the RLO. The RLO is carried along with the jump to the new block. Within the new block, it can be evaluated but no longer combined logically.

The addressed block is processed **regardless** of the previous result of logic operation (RLO **-** see Section 3.4).

Example: **JU** PB 100

Conditional call

The JC statement belongs to the conditional operations. The addressed block is processed only if the previous **RLO = 1**. If the RLO = 0, the jump is not executed.

Example: **JC** PB 100

Note
After the conditional jump operation is executed, the RLO is set to "**1**" regardless of whether or not the jump to the block is executed.

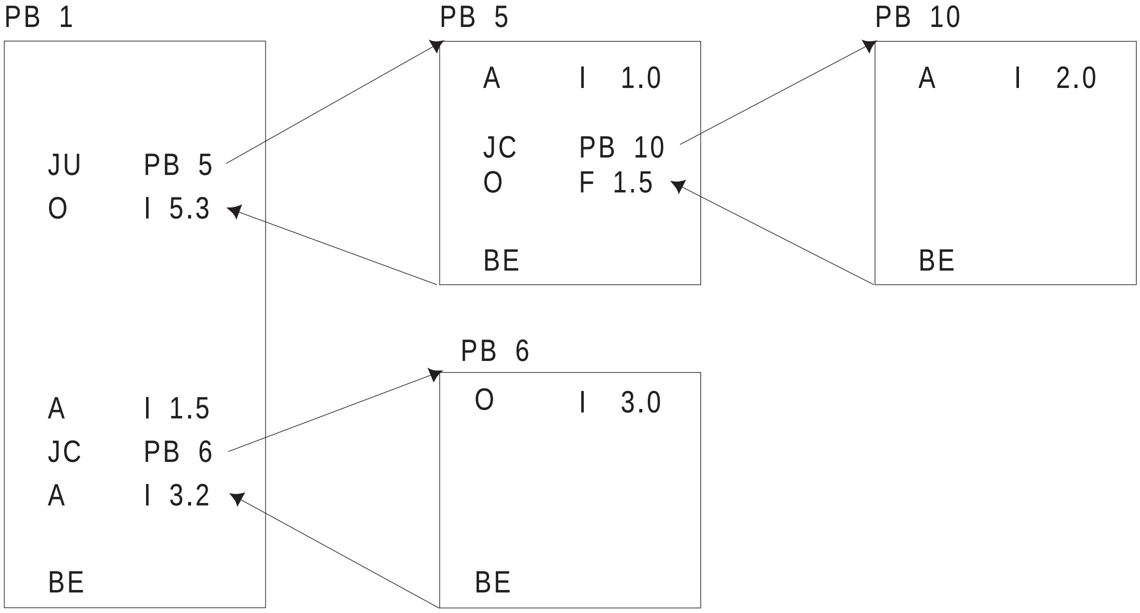


Fig. 2-3 Block calls that enable processing of a program block

Effect of the BE statement

After the "BE" statement (block end), the CPU continues the user program in the block in which the block call was programmed. Program execution continues at the STEP 5 statement following the block call.
The "BE" statement is executed regardless of the RLO. After "BE", the RLO can no longer be combined logically. However, the RLO or arithmetic result occurring directly before execution of the BE operation is transferred to the block where the call originated and can be evaluated there. When program execution returns from the block that has been called, the contents of ACCU 1, ACCU 2, ACCU 3 and ACCU 4, the condition codes CC 0 and CC 1 and the RLO are not changed. (Refer to Section 3.5 for more detailed information about the ACCUs, CC0/CC1 and RLO).

2.2.1 Organization Blocks as User Interfaces

Organization blocks form the interfaces between the system program and the user program. Organization blocks OB 1 to OB 39 belong to your user program just as program blocks. By programming these OBs, you can influence the behavior of the CPU during start-up, program execution and in the event of an error. The organization blocks are effective as soon as they are loaded in the PLC memory.
This is also possible while the PLC is in the run mode.
Once the system program has called a specific organization block, the user program it contains is executed.

Note
You can program blocks OB 1 to OB 39 as user interfaces and they are called automatically by the system program as a reaction to certain events.

For **test purposes**, you can also call these organization blocks from the user program (JC/JU OB xxx). It is, however, not possible to trigger a COLD RESTART, e.g. by calling OB 20.

The following table provides you with an overview of the user interfaces (OBs).

Table 2-1 Overview of the organization blocks of the CPU 948 for program execution

<table>
<tr><th colspan="4">Organization blocks for controlling program execution</th></tr>
<tr><th>Block</th><th colspan="3">Function and call criterion</th></tr>
<tr><td>OB 1</td><td colspan="3">Organization of cyclic program execution; first call after a start-up, then cyclic call.</td></tr>
<tr><td></td><td colspan="3"><b>With DX-0 setting "Process interrupt servicing via input byte IB 0 =on":</b>
(interruptability at block boundaries, can be set in DX 0)</td></tr>
<tr><td></td><td colspan="3">Call with signal state change in input byte IB 0 in bit:</td></tr>
<tr><td>OB 2</td><td colspan="3">I 0.0</td></tr>
<tr><td>OB 3</td><td colspan="3">I 0.1</td></tr>
<tr><td>OB 4</td><td colspan="3">I 0.2</td></tr>
<tr><td>OB 5</td><td colspan="3">I 0.3</td></tr>
<tr><td>OB 6</td><td colspan="3">I 0.4</td></tr>
<tr><td>OB 7</td><td colspan="3">I 0.5</td></tr>
<tr><td>OB 8</td><td colspan="3">I 0.6</td></tr>
<tr><td>OB 9</td><td colspan="3">I 0.7</td></tr>
<tr><td></td><td colspan="3"><b>With DX-0 setting "Process interrupt servicing via input byte IB 0 = off":</b>
(interruptability at operation or block boundaries, can be set in DX 0)</td></tr>
<tr><td></td><td colspan="3">Call via interrupt lines of the S5 bus:</td></tr>
<tr><td></td><td colspan="3">System interrupt INT X (INT A, B, C or D, depends on slot)</td></tr>
<tr><td>OB 2</td><td colspan="3">System interrupt INT E</td></tr>
<tr><td>OB 3</td><td colspan="3">System interrupt INT F</td></tr>
<tr><td>OB 4</td><td colspan="3">System interrupt INT G</td></tr>
<tr><td>OB 5</td><td colspan="3"></td></tr>
<tr><td></td><td colspan="3">Delayed interrupt</td></tr>
<tr><td>OB 6</td><td colspan="3">Clock-controlled interrupt</td></tr>
<tr><td>OB 9</td><td colspan="3"></td></tr>
<tr><td></td><td colspan="3">Organization of time-controlled program execution (timed interrupt) with selectable basic clock rate (default T = 100 ms) and clock distributor (default corresponds to 150U) in data block DX 0;
Calls as follows:</td></tr>
<tr><td></td><td>Default setting</td><td>150U clock distributor</td><td>2^n clock distributor</td></tr>
<tr><td>OB 10</td><td>0.1 s</td><td>T * 1</td><td>T * 1</td></tr>
<tr><td>OB 11</td><td>0.2 s</td><td>T * 2</td><td>T * 2</td></tr>
<tr><td>OB 12</td><td>0.5 s</td><td>T * 5</td><td>T * 4</td></tr>
<tr><td>OB 13</td><td>1.0 s</td><td>T * 10</td><td>T * 8</td></tr>
<tr><td>OB 14</td><td>2.0 s</td><td>T * 20</td><td>T * 16</td></tr>
<tr><td>OB 15</td><td>5.0 s</td><td>T * 50</td><td>T * 32</td></tr>
<tr><td>OB 16</td><td>10.0 s</td><td>T * 100</td><td>T * 64</td></tr>
<tr><td>OB 17</td><td>20.0 s</td><td>T * 200</td><td>T * 128</td></tr>
<tr><td>OB 18</td><td>50.0 s</td><td>T * 500</td><td>T * 256</td></tr>
</table>

Table 2-2 Overview of the organization blocks of the CPU 948 for start-up

Organization blocks to control the start-up procedure	
Block	**Function and call criterion**
OB 20	Call on request for COLD RESTART (manual and automatic)
OB 21	Call on request for MANUAL WARM RESTART/COLD RESTART WITH MEMORY
OB 22	Call on request for AUTOMATIC WARM RESTART/COLD RESTART WITH MEMORY

Table 2-3 Organization blocks of the CPU 948 for a SOFT STOP

Organization blocks to control the start-up procedure	
Block	**Function and call criterion**
OB 38	Organization of the start-up procedure for communication in the "soft stop" mode.
OB 39	Organization of the cyclic program for communication in the "soft stop" mode.

Table 2-4 Overview of the organization blocks of the CPU 948 for error handling

Organization blocks for reaction to device or program errors [1)]	
Block	**Function and call criterion**
OB 19	Runtime error (LZF): called block not loaded (PB, SB, FB, FX) or attempt to open a data block that is not loaded (DB, DX)
OB 23	Timeout (QVZ) in user program (during direct access to I/O modules)
OB 24	Timeout (QVZ) when updating the process image and transferring interprocessor communication flags.
OB 25	Addressing error (ADF)
OB 26	Cycle time exceeded (ZYK)
Table 2-4 continued:	
OB 27	Substitution error (SUF)

Organization blocks for reaction to device or program errors [1]	
Block	**Function and call criterion**
OB 28	Timeout input byte IB 0 (process interrupts)
OB 29	Timeout distributed I/Os, extended address volume
OB 30	Timeout and parity error (PARE) accessing the user memory
(OB 31)	(set cycle monitoring time) [2]
OB 32	Load and transfer error accessing data blocks (TRAF)
OB 33	Collision of timed interrupts (WEFES/WEFEH)
OB 34	Error setting up a data block (G DB/GX DX)
OB 36	Error in self test

[1] If the OB is not programmed, the CPU changes to the stop mode in the event of an error. EXCEPTION: if OB 19 (logic block not loaded), OB 23 or OB 24, OB 29 (timeout) or OB 33 (collision of timed interrupts) do not exist, there is no reaction!

[2] OB 31 only exists in the CPU 948 for the sake of compatibility.
To set the cycle monitoring time, you should use data block DX0 (refer to Chapter 7)
OB 31 is called once during the start-up, if loaded. You can also use it to set the cycle monitoring time by programming the following STEP 5 operations in it:

```
:L    KF  +nnn
:BE
```

nnn is a decimal number. The cycle monitoring time is obtained from "nnn * 10 ms".

Operating system organization blocks of the CPU 948	
Block	**Function**
OB 0	Internal block belonging to operating system

2.2.2 Organization Blocks for Special Functions

The following organization blocks contain special functions of the system program. You **cannot** program these blocks, but simply call them (this applies to all OBs with numbers between 121 and 255!). They do not contain a STEP 5 program. Special function OBs can be called in all logic blocks.

Table 2-5 Overview of the organization blocks of the CPU 948 for special functions

Integrated organization blocks with special functions	
Block:	**Block function:**
OB 121	Set/read time of day (compatible with CPU 946/947)
OB 122	"Disable interrupts" on/off
OB 124	Delete STEP 5 blocks
OB 125	Generate STEP 5 blocks
OB 126	Define/transfer process images
OB 129	Battery state
OB 131	Delete ACCU 1 to ACCU 4
OB 132	Roll up ACCU
OB 133	Roll down ACCU
OB 141	Enable/disable "disable individual timed interrupts"
OB 142	Enable/disable "delay all interrupts"
OB 143	Enable/disable "delay single timed interrupts"
OB 150	Set/read system time (compatible with CPU 928B)
OB 151	Set/read clock-controlled interrupt time
OB 153	Set/read time for delayed interrupt
OB 180	Variable data block access
OB181	Test data blocks (DB/DX)
OB 182	Copy data area
OB 200, 202 to 205	Functions for multiprocessor communication
OB 222	Restart cycle monitoring time
OB 223	Compare start-up types of CPUs in multiprocessor mode
OB 254, 255	Copy/duplicate DB and DX data blocks from memory card to user memory

These special functions are described in detail in Chapter 6.

2.3 Function Blocks

Function blocks (FB/FX) are also parts of the user program just like program blocks. FX function blocks have the same structure as FB function blocks and are programmed in the same way.
You use function blocks to implement frequently recurring or very complex functions. In the user program, each function block represents a complex complete function. You can obtain function blocks as follows:

- as a software product from SIEMENS (standard function blocks on diskette - see /11/); with these function blocks you can generate user programs for fast and simple open loop control, signalling, closed loop control and logging;

  or

- you can program function blocks yourself.

Compared with organization, program and sequence blocks, function blocks have the following four essential **differences**:

	OB, PB, SB	**FB/FX**
1.	**Range of operations**	
	only basic operations	- basic operations, - supplementary operations - system operations
2.	**Method of representation**	
	programming and call in STL, LAD, CSF	programming only in AWL
3.	**Name**	
	name environment not possible (only number)	in addition to the number a name with max. 8 chars. can be assigned
4.	**Operands**	
	none	formal operands (block parameters). When the block is called formal operands are assigned actual operands

2.3.1 Structure of Function Blocks

The **block header** (five words) of a function block has the same structure as the headers of the other STEP 5 block types.

The **block body** on the other hand, has a different structure from the bodies of the other block types. The block body contains the function to be executed in the form of a statement list in the STEP 5 programming language. Between the block header and the STEP 5 statements, the function block needs additional memory space for its name and for a list of formal operands. Since this list contains no statements for the CPU, it is skipped with an unconditional jump that the programmer generates automatically. This jump statement is not displayed when the function block is displayed on the PG!

When a function block is called, only the block body is processed.

Absolute or symbolic operands

You can enter operands in a function block in absolute form (e.g. F 2.5) or symbolically (e.g. MOTOR1). You must store the assignment of the symbolic operands in an **assignment list** before you enter the operands in a function block (see /3/).

Fig. 2-4 shows the structure of a function block in the memory of a programmable controller.

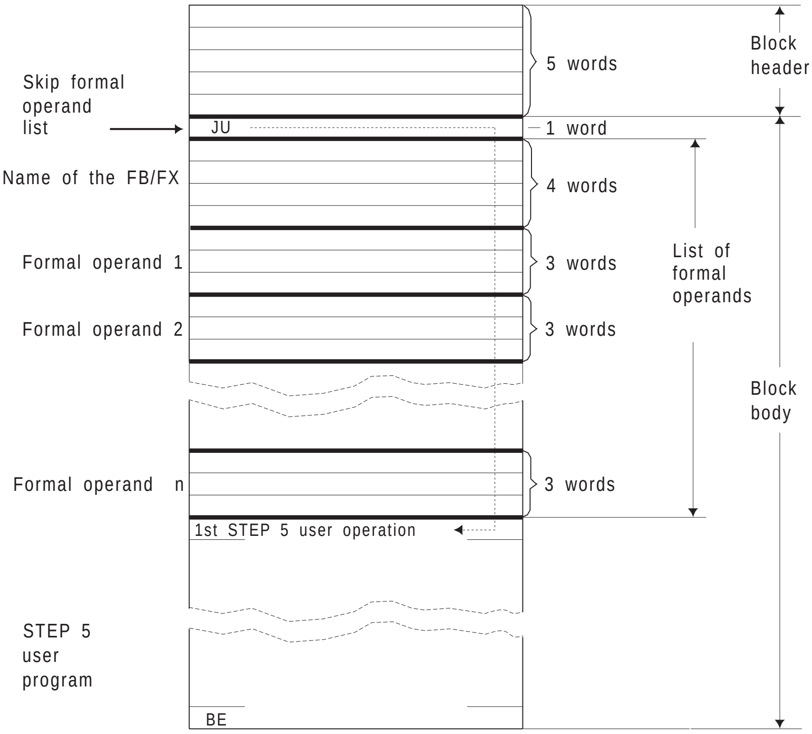


Fig. 2-4 Structure of a function block (FB/FX)

The memory contains all the information that the programmer needs to represent the function block graphically when it is called and to check the operands during parameter assignment and programming of the function block. The programmer rejects incorrect input.

When handling function blocks, distinguish between the following procedures:

- **programming** FB/FX

  and

- **calling** FB/FX and then **assigning actual values** to the parameters.

Distinction: "programming" – "calling and assigning parameters"

When **programming**, you specify the function of the block. You must decide which input operands the function requires and which output results it should transfer to the calling program. You define the input operands and output results as formal operands. These function as tokens.

When a block is **called** by a higher order block (OB, PB, SB, FB, FX), the formal operands (block parameters) are replaced by actual operands; i.e. **parameters** are assigned to the function block.

How to program

IF...	**THEN...**
You want to program a function block "directly", i.e. without formal operands.	Program it as you would a program or sequence block.
You want to use formal operands in a function block.	Proceed as explained on the following pages. Make sure you keep to the required order: First program the FB/FX with the formal operands and keep it on the PG (offline) or in the CPU memory (online) Then program the block(s) to be called with the actual operands.

2.3.2 Programming Function Blocks

You can program a function block only in the **"statement list"** method of representation. When entering a function block at a programmer, perform the following steps:

Step	Action
1	Enter the **block type** (FB/FX) and the **number** of the function block. Number your function blocks in descending order starting with FB 255, so that they do not collide with the standard function blocks. The standard function blocks are numbered from FB 1 to FB 199.
2	Enter the **name** of the function block. The name can have a maximum of eight characters and must start with a letter.
3	**If the function block is to process formal operands:** Enter the formal operands you require in the block as block parameters. Enter the following information for each formal operand: - the name of the block parameter (maximum 4 characters), - the type of block parameter and the data type of the block parameter (if applicable) You can define a maximum of 40 formal operands.
4	Enter your STEP 5 program in the form of a **statement list** (STL). The formal operands are preceded by an equality sign (e.g. A = X1). They can also be referenced more than once at various positions in the function block.
5	Terminate your program input with the block end operation "BE".

Note
If you change the **order** or the **number** of formal operands in the formal operand list, you must also update all STEP 5 statements in the function block that reference a **formal operand** and also the block parameter list in the calling block!

Program or change function blocks only on diskette or hard disk and then transfer them to your CPU!

Formal operands

The following parameter and data types are permitted as the formal operands of a function block (also known as **block parameters**):

Table 2-5 Permitted formal operands for function blocks

Parameter type	Data type

I = input parameter
Q = output parameter
	BI/BY/W/D

D = data	KM/KH/KY/KS/KF/ KT/KC/KG

B = block operation
T = timer
C = counter
	none
	(no type can be specified)

I, D, B, T or **C** are parameters that are indicated to the **left** of the function symbol in graphic representation.
Parameters labelled with **Q** are indicated on the **right** of the function symbol.

The data type indicates whether you are working with bits, bytes, words or double words for I and Q parameters and which data format applies to D parameters (e.g. bit pattern or hexadecimal pattern).

2.3.3 Calling Function Blocks and Assigning Parameters to them

You can call every function block as often as you want anywhere in your STEP 5 program. You can call function blocks in a statement list or in one of the graphic methods of representation (CSF or LAD).

To call a function block and assign parameters to it, perform the following steps:

Step	Action	Reaction on PG
1	Make sure that the called function block exists either in the PG memory (offline) or in the CPU memory (online).	none
2	Enter the call statement for the function block in the block where the call is to originate. You can program a function block call in an organization, program or sequence block or in another function block.	After you enter the call statement (e.g. JU FB200), the name of the relevant function block and the formal operand list appear automatically.
3	Assign the **actual** operand relevant to this call to each of the formal operands, i.e. you assign **parameters** to the function block. These actual operands can be different for separate calls (e.g. inputs and outputs for the first call of FB 200, flags for the second call). Using the formal operand list, you assign the required actual operands for each function block call.	none

Unconditional/conditional call

Unconditional call	Conditional call
"JU FBn" for FB function blocks or **"DOU FXn"** for FX extended function blocks: the referenced function block is processed regardless of the previous result of logic operation (RLO).	"JC FBn" for FB function blocks or **"DOC FXn"** for FX extended function blocks: the referenced function block is only processed when the result of logic operation RLO = 1. If RLO = 0 the block call is not executed. Regardless of whether the block call is executed or not, the RLO is alsways set to "1".
After the unconditional or conditional call, the RLO can no longer be combined logically. However, it is carried over to the called function block with the jump and can be evaluated there.	

Permitted actual operands

Which operands can be assigned as **actual operands** is shown in the following table.

Table 2-6 Permitted actual operands for function blocks

Parameter type	**Data type**		**Actual operands permitted**
I, Q	BI	for an operand with bit address	I n.m input Q n.m output F n.m flag
	BY	for an operand with byte address	IB n input byte QB n output byte FY n flag byte DL n data byte left DR n data byte right PY n peripheral byte OY n byte from extended periphery
	W	for an operand with word address	IW n input word QW n output word FW n flag word DW n data word PW n peripheral word OW n word from extended periphery
	D	for an operand with double word address	ID n input double word QD n output double word FD n flag double word DD n data double word
D	KM	for a binary pattern (16 bits)	Constants
	KY	for two absolute numbers, one byte each, each in the range from 0 to 255	
	KH	for a hexadecimal pattern with a maximum of four digits	
	KS	for two alphanumeric characters	
	KT	for timer value (BCD-coded) units .0 to .3 and values 0 to 999	
	KC	for a counter value 0 to 999	

Parameter type	Data type	Actual operands permitted
Table 2-6 continued:		
D (Cont.)	KF for a fixed point number -32768 to +32767 KG for a floating point number[1)]	Constants
B	Data type designation not possible	DB n Data block; the operation C DB n is executed FB n Function block (permitted only without parameters) called unconditionally (JU . .n) OB n Organization block called unconditionally (JU . .n) PB n Program blocks - called unconditionally (JU . .n) SB n Sequence blocks - called unconditionally (JU . .n)
T	Data type designation not possible	T 0 to 255 Timer
C	Data type designation not possible	Z 0 to 255 Counter

[1)] ±0.1469368 x 10^{-38} to ±0.1701412 x 10^{39}

Note
S flags are **not** permitted as actual operands for function blocks.

After the jump to a function block, the actual operands from the block then called are used in the function block program instead of the formal operands.
This feature of programmable function blocks allow them to be used for a wide variety of purposes in your user program.

When the program returns from the called function block, the list of actual operands in the calling block is skipped by a jump operation activated implicitly by STEP 5 in MC-5 code.

Examples

Example 1: the following (complete) example is intended to further clarify the programming and calling of a function block and the assignment of parameters to it. You yourself can easily try out the example.

Programming the function block FB 202:

```
FB 202

SEGMENT 1
NAME EXAMPLE
DECL :  INP1   I/Q/D/B/T/C:  I     BI/BY/W/D:  BI
DECL :  INP2   I/Q/D/B/T/C:  I     BI/BY/W/D:  BI
DECL :  OUT1   I/Q/D/B/T/C:  Q     BI/BY/W/D:  BI


     :A=  INP1
     :A=  INP2
     :==  OUT1

     :
     :  BE
```

Formal operand list

STEP 5 state- ments

Formal operands

Parameter type

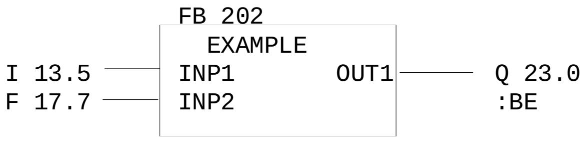
Data type

Function block FB 202 is called and has parameters assigned to it in program block PB 25:

STL method of representation

CSF/LAD method of representation

```
PB 25
SEGMENT 1

     :  JU FB 202
NAME :  EXAMPLE
INP1 :  I 13.5
INP2 :  F 17.7
OUT1 :  Q 23.0
     :  BE
```



Formal operands

Actual operands

The following operations are executed after the jump to FB 202

Example 2: calling a function block and assigning parameters to it with the STL and CSF/LAD methods of representation in a program block.

STL method of representation

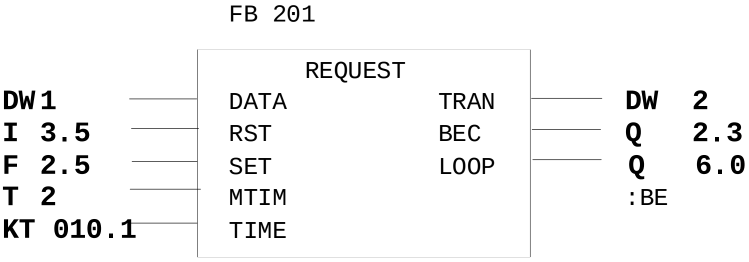
```
PB 25
SEGMENT 1
      :
      :  C  DB 5
      :
      :  JU   FB 201
NAME  :  REQUEST
DATA  :       DW   1
RST   :       I    3.5
SET   :       F    2.5
MTIM  :       T    2
TIME  :       KT   010.1
TRAN  :       DW   2
BEC   :       Q    2.3
LOOP  :       Q    6.0
      :  BE
```

Formal operands

Actual operands

CSF/LAD method of representation

PB 25
SEGMENT 1



2.3.4 Special Function Blocks

Apart from the function blocks that you program yourself, you can order **standard function blocks** as a finished software product. These contain standard functions for general use (e.g. signalling functions and sequence control).
Standard function blocks are assigned numbers FB 1 to FB 199.

If you order standard function blocks, remember the special instructions in the accompanying description (i.e. areas assigned and conventions etc.).

The standard function blocks for the S5-155U are listed in catalog ST 57 /11/.

Example

Floating point root extractor RAD:GP FB 6

The function block RAD:GP extracts the root of a floating point number (8-bit exponent and 24-bit mantissa). It forms 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 up or down.

If applicable, for the rest of the processing, the function block sets the "radicand negative" identifier.

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 = RADI

Calling the function block FB 6:

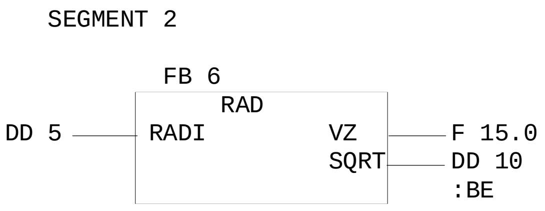
In the example, the root is extracted from a floating point number that is located in DD5 of DB 17 with an 8-bit exponent and a 24-bit mantissa. The result, another 32-bit floating point number, is written to DD 10. Prior to this, the appropriate data block must be opened. The parameter VZ (parameter type: Q, data type: BI) indicates the sign of the radicand: VZ = 1 for a negative radicand.

"Floating point root extractor" continued:

STL method of representation

LAD method of representation

```
Seg-         : C DB 17
ment         :
1            :***
             : JU FB 6
Seg-   NAME  : RAD : GP
ment   RADI  : DD 5
2      VZ    : F 15.0
 *)    SQRT  : DD 10
```



DD= data double word

*) Must be located in separate segments, since the operation "C DB 17" in segment 1 cannot be converted to LAD/CSF.

2.4 Data Blocks

Data blocks (DB) or extended data blocks (DX) are used to store the fixed or variable data with which the user program works. **No** STEP 5 operations are processed in data blocks.

The data of a data block includes the following:

- various bit patterns (e.g. for status of a controlled process)
- numbers (hexadecimal, binary, decimal) for timer values or arithmetic results
- alphanumeric characters, e.g. for message texts.

Structure of a data block

A data block (DB/DX) consists of the following parts:

- block preheader (DV, DXV),
- block header
- block body.

Block preheader

The **block preheader** is created automatically on the hard or floppy disk of the PG and not transferred to the CPU. It contains the data formats of the data words entered in the block body. You have no influence over the creation of the block preheader.

> **Note**
> When you transfer a data block from the PLC to diskette or hard disk, the corresponding block preheader can be deleted. For this reason, you must never modify a data block with different data formats in the PLC and then transfer it back to diskette, otherwise all the data words in the DB are automatically assigned the data format you selected in the presets screen form.

Block header

The **block header** occupies five words in the memory and contains the following:

- the block identifier
- the programmer identifier
- the block type and the block number
- the library number
- the block length (including the length of the block header).

Block body

The **block body** contains the data words with which the user program works. These data words are in ascending order in the block body, starting with data word DW 0. Each data word occupies one word (16 bits) in the memory.

Maximum length

A data block can occupy a total of maximum 32 767 words (including header) in the CPU memory. When you use your programmer to enter and transfer data blocks, remember the size of your CPU memory!

2.4.1 Creating Data Blocks

To create a data block, perform the following steps:

Step	Action
1	Enter the block type (DB/DX) and data block **number** (2 or 3 to 255).
2	Enter individual **data words** in the data format you require. (Do **not** complete your input of the data words with a BE statement!)

Note
Data blocks DB 0, DB 1, DX 0, DX 1 and DX 2 are reserved for specific functions and you cannot use them freely for other functions (see Section 2.4.3)!

Table 2-7 Data formats permitted in a data block

Type	Data format	Examples
KM	Bit pattern	00100110 00111111
KH	Hexadecimal	263F
KY	2 Bytes	038,063
KF	Fixed point number	+09791
KG	Floating point number	+1356123+12
KS	Character	?!ABCD123-+.,%
KT	Timer value	055.2
KC	Counter value	234

2.4.2 Opening Data Blocks

You can only open a data block (DB/DX) **unconditionally**. This is possible within an organization, program, sequence or function block. You can open a specific data block more than once in a program.

To open a data block, perform the following steps:

IF...	THEN...
You want to open a **DB** data block	Type in the STEP 5 operation **"C DB.."**
You want to open a **DX** data block	Type in the STEP 5 operation **"CX DX."**

Validity of a data block

After you open a data block, all statements that follow with the operand area **’D’** refer to the opened data block.

The opened data block also remains valid when the program is continued in a different block following a block call.

If a second data block is opened in this new block, the second data block is **only** valid in the newly called block from the point at which it is called. After program execution returns to the calling block, the old data block is once again valid.

Access

You can **access** the data stored in the opened data block during program execution using **binary logic operations, set/reset operations, load or transfer operations** (refer to Chapter 3 for more detailed information).

With a **binary operation**, the addressed data word bit is used to form the RLO. The content of the data word is not changed.

With a set/reset operation, the addressed data word bit is assigned the value of the RLO. The content of the data word may be changed.

A **load operation** transfers the contents of the referenced data word into ACCU 1. The contents of a data word are not changed.

A transfer operation transfers data from ACCU 1 to the referenced data word. The old contents of the data word are overwritten.

Note
Before accessing a data word, you must open the data block you require in your program. This is the only way that the CPU can find the correct data word.
The referenced data word must be contained in the opened block, otherwise the system program detects a load or transfer error.

With load and transfer operations, you can only access data word numbers up to 255!

An opened data block remains valid until one of the following events occur:

a) a second data block is opened

or

b) the block, in which the data block was opened, is completed with ’BE’, ’BEC’ or ’BEU’.

Examples

Example 1: transferring data words

You want to transfer the contents of data word DW 1 from data block DB 10 to data word DW 1 of data block DB 20.

Enter the following statements:

```
   :C   DB 10   (open DB 10)
   :L   DW 1    (load the contents of DW 1 into
   :            ACCU 1)
   :C   DB 20   (open DB 20)
   :T   DW 1    (transfer the contents of ACCU 1 to
   :            DW 1)
   :
```

Example 2: range of validity of data blocks (Fig. 2-5)

Data block DB 10 is opened in program block PB 7 (C DB 10). During the subsequent program execution, the data of this data block are processed.

After the call (JU PB 20) program block PB 20 is processed. Data block DB 10, however, remains valid. The data area only changes when data block DB 11 (C DB 11) is opened.
Data block DB 11 now remains valid until the end of program block PB 20 (BE).

After the jump back to program block PB 7, data block DB 10 is once again valid.

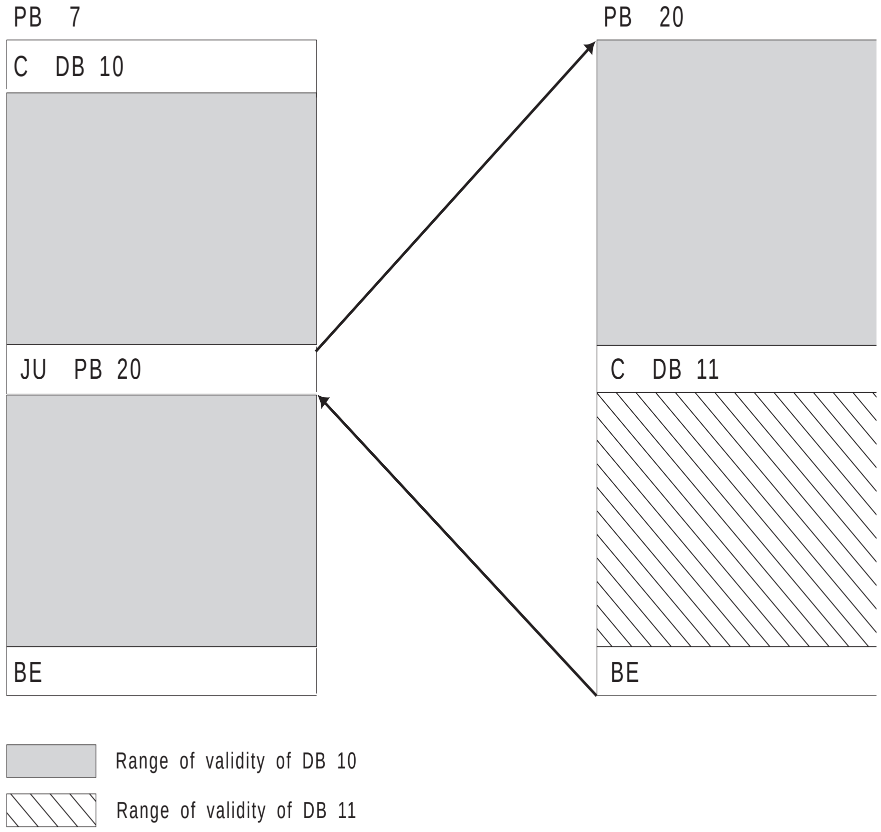


Fig. 2-5 Range of validity of an opened data block

2.4.3 Special Data Blocks

On the CPU 948 data blocks DB 0, DB 1, DX 0, DX 1 and DX 2 are reserved for special functions. They are managed by the system program and you cannot use them freely for other functions.

DB 0

- **Data block DB 0** (see Section 8.3.2)

  Data block DB 0 contains the address list with the start addresses of all blocks that are located in the data block RAM of the CPU. The system program generates this address list during initialization (following each POWER UP or OVERALL RESET) and it is updated automatically when you use a programmer to change data blocks or generate a new data block.

DB 1

- **Data block DB 1** (see Section 10.1.6)

  Data block DB 1 contains the list of digital inputs/outputs (P peripheral with relative byte addresses from 0 to 127) and the interprocessor communication (IPC) flag inputs and outputs that are assigned to the CPU. If applicable, the block may also contain a timer field length.

  DB 1 **can** have parameters assigned and be loaded as follows:
  to reduce the cycle time in single processor operation, since only the inputs, outputs or timers entered in DB1 are updated.

  DB 1 **must** be assigned parameters and loaded as follows:
  a) for multiprocessing
  b) when IPC flags exist with CPs

DX 0

- **Data block DX 0** (see Chapter 7)

  If you assign parameters to data block DX 0 and load it, you can change the defaults of certain system program functions (e.g. the start-up procedure) and adapt the performance of the system program to your particular application.

DX 1

- **Data block DX 1**

  Reserved.

DX 2

- **Data block DX 2**

  Reserved for the second serial interface.

Program Execution 3

Contents of Chapter 3

Program Execution 3

This chapter is intended for readers who do not yet have any great experience in using the programming language. The chapter therefore deals with the basics of STEP 5 programming and explains in detail (with examples) the STEP 5 operations for the CPU 948.

Experienced readers who require more information about a specific STEP 5 operation listed in the Pocket Guide can refer to the reference section in 3.5.

3.1 Principle of Program Execution

You can process your STEP 5 user program in various ways.

Cyclic program execution is most common with programmable controllers (PLCs). The system program runs through a program loop (the cycle, refer to Section 3.4) and calls organization block OB 1 cyclically in each loop (refer to Fig. 3-1).

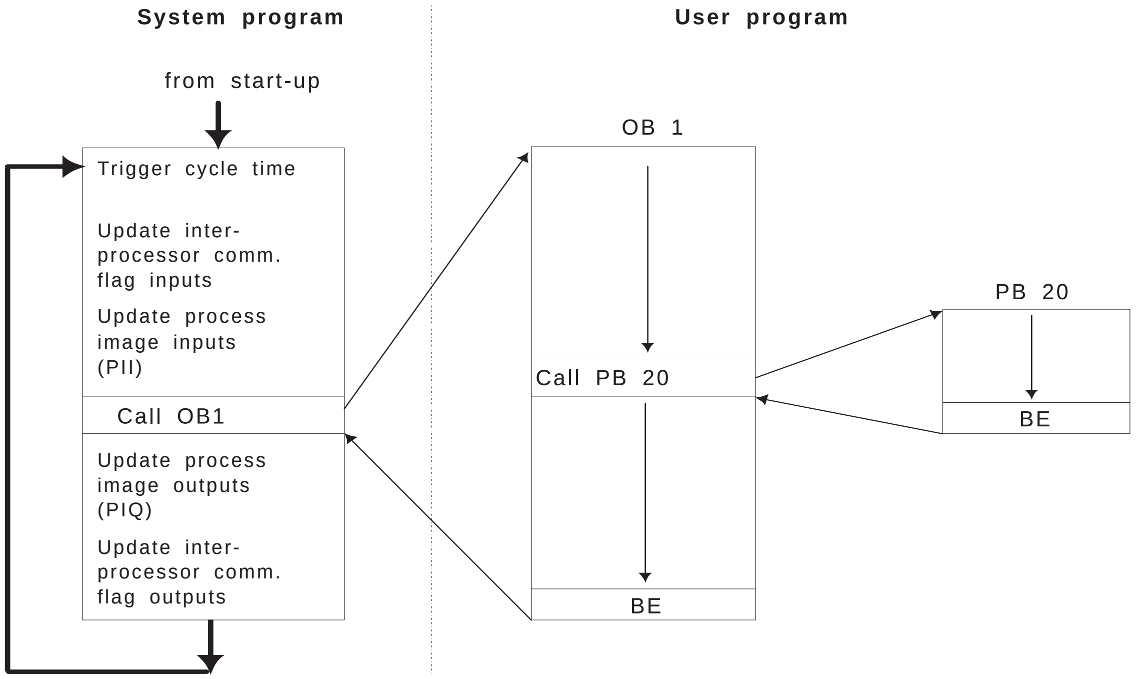

System program
User program
from start-up
Trigger cycle time
Update inter-processor comm. flag inputs
Update process image inputs (PII)
Call OB1
Update process image outputs (PIQ)
Update inter-processor comm. flag outputs
OB 1
Call PB 20
BE
PB 20
BE

Fig. 3-1 Principle of cyclic program execution

3.2 Program Organization

Program organization allows you to specify which conditions affect the processing of your blocks and the order in which they are processed. Organize your program by programming organization blocks with conditional or unconditional calls for the blocks you require.

You can call additional program, function and sequence blocks in any combination in the program of individual organization, program, function and sequence blocks. You can call these one after another or nested in one another.

For maximum efficiency, you should organize your program to emphasise the most important program structures and in such a way that you can clearly recognize parts of the controlled system which are related in the software.

Figs. 3-2 and 3-3 are examples of a program structure.

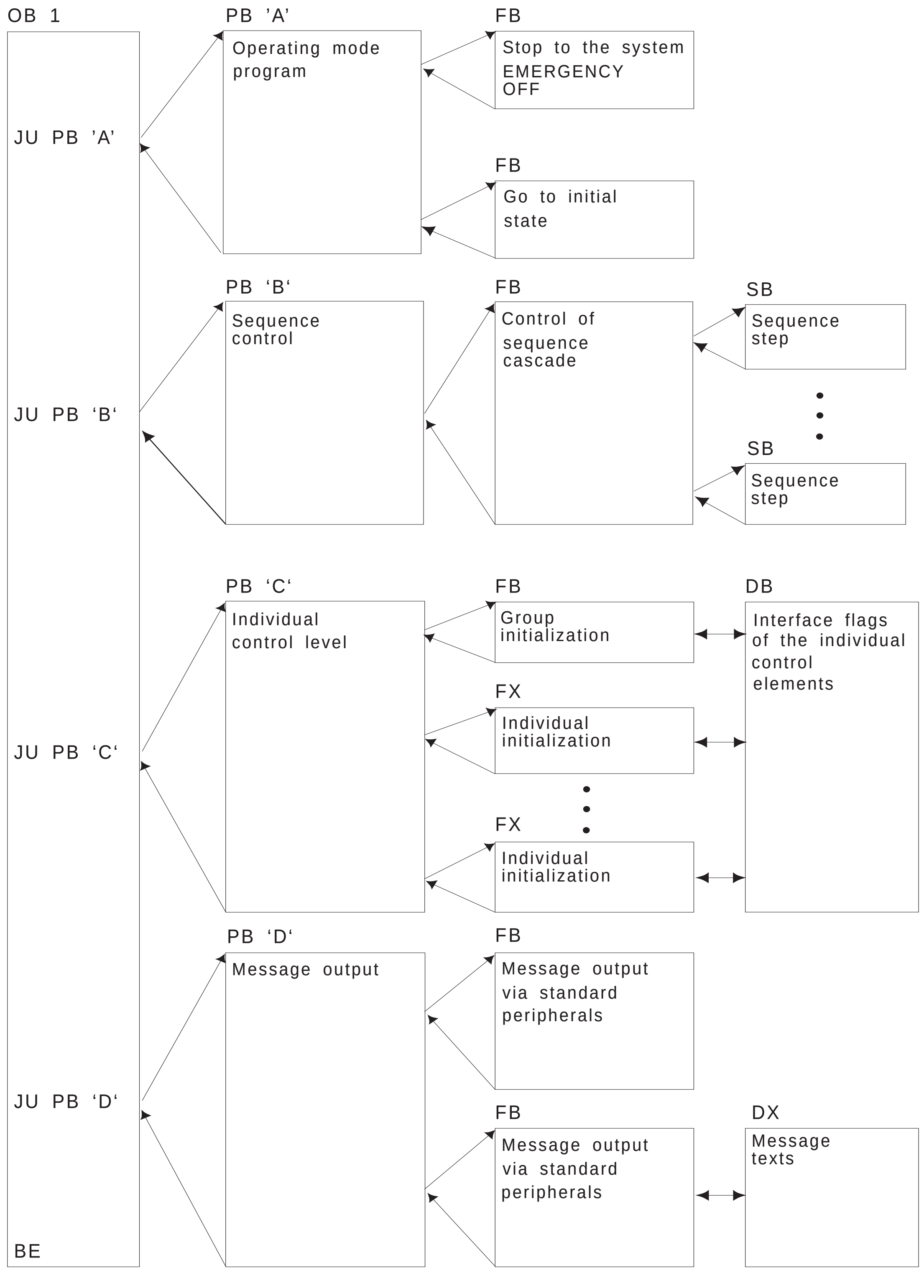


Fig. 3-2 Example of the organization of the user program according to the program structure

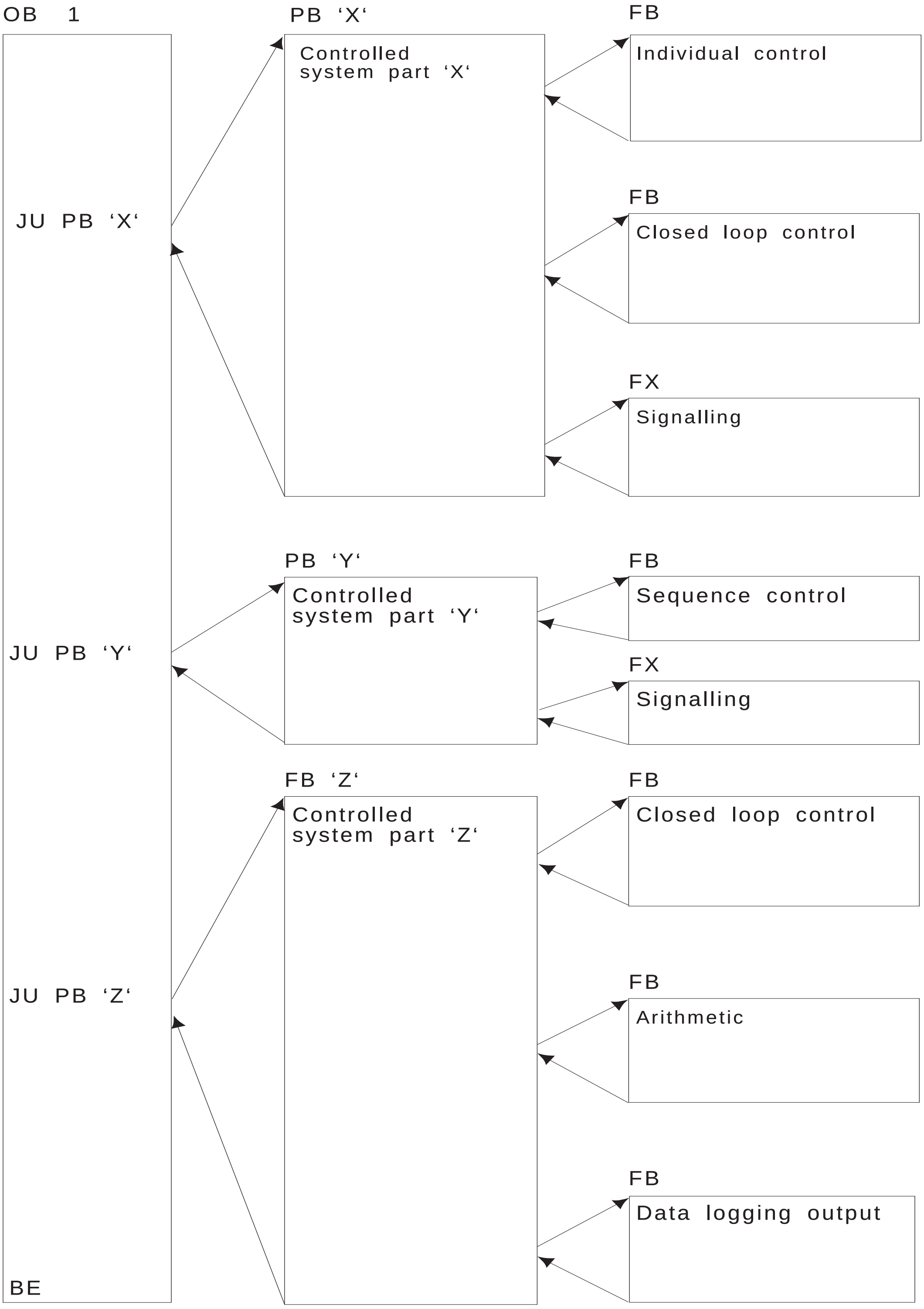


Fig. 3-3 Example of the organization of the user program according to the structure of the controlled system

Nesting blocks

Fig. 3-4 shows the principle of nested block calls.

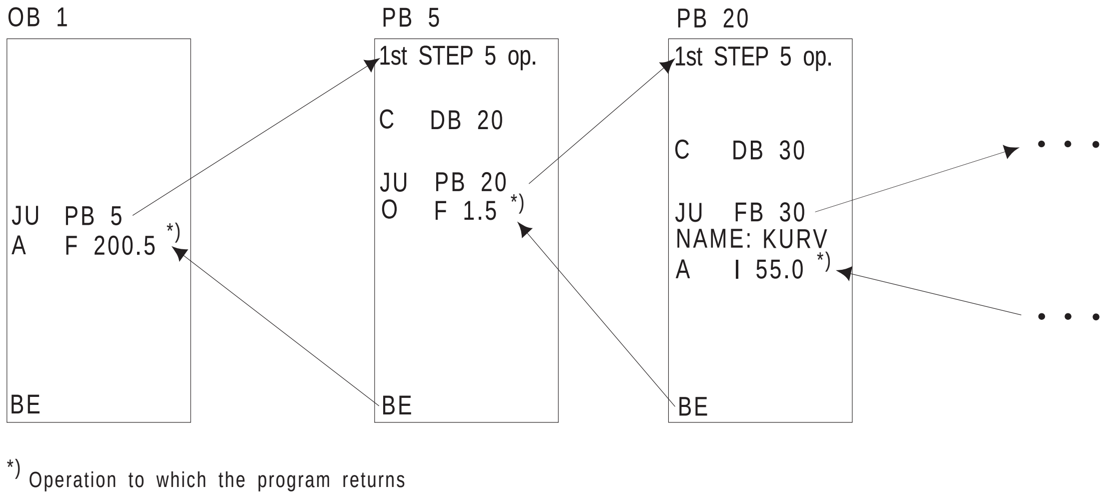


Fig. 3-4 Nested logic block calls

Block addresses

A block start address specifies the location of a block in the user memory. For logic blocks, this is the address of the memory location containing the first STEP 5 operation (with FB and FX, the JU operation via the formal operand list); with data blocks, it is the address of the first data word.

To enable the CPU to locate the called block in the memory, the start addresses of all valid blocks are entered in the block address list in data block DB 0. DB 0 is managed by the system program, you cannot call it yourself.

The CPU stores a **return address** every time a new block is called. After the new block has been processed, this return address enables the program to find the block from which the call originated. The return address is the address of the memory location containing the next STEP 5 statement after the block call. The CPU also stores the **start address and length of the data block** valid at this location.

Nesting depth

You can only nest 40 blocks within one another. If more than 40 blocks are called, the CPU signals an error and goes to the stop mode.

Example of nesting depth

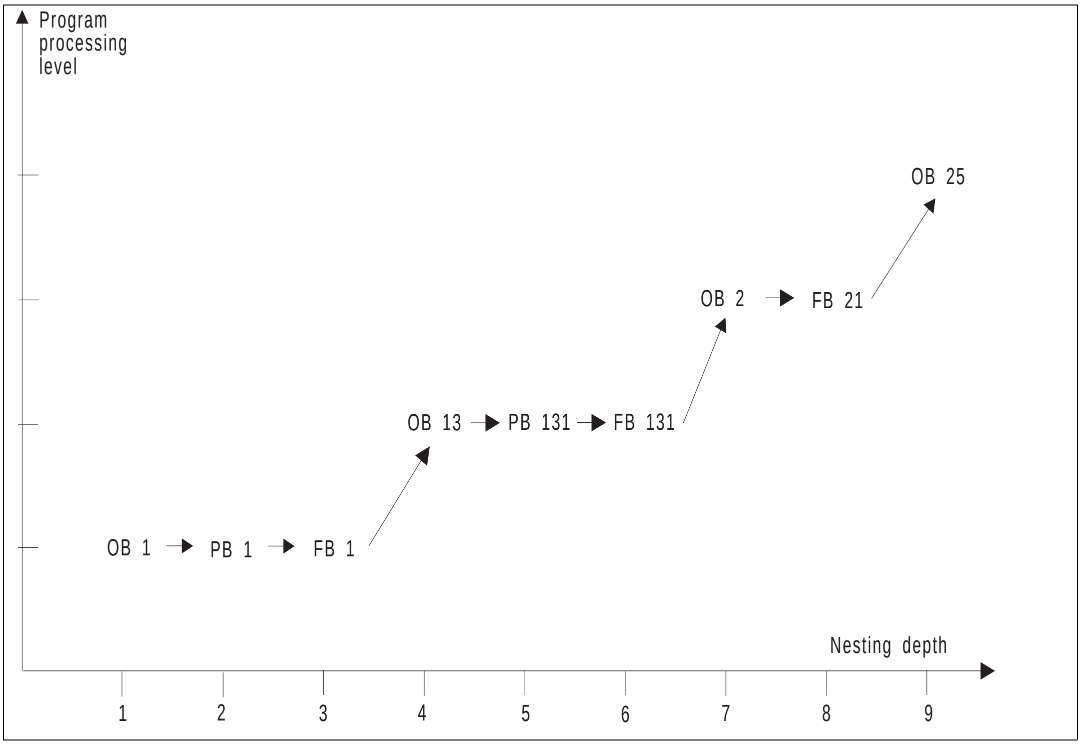


Fig. 3-5 Example of block nesting depth

You can determine the nesting depth of your program as follows:

- Add all the organization blocks you have programmed (in the example: 4 OBs).
- Add the nesting depth of the individual organization blocks (in the example: 2 + 2 + 1 + 0 = 5).
- Add the two amounts together to obtain the program nesting depth (in the example: 4 + 5 = nesting depth 9). It must not exceed a value of 40.

3.3 Storing Program and Data Blocks

On the CPU 948, the user program runs solely in the internal RAM. The user program including data blocks must, therefore, be loaded in the CPU 948 user memory.

How do I load programs and data blocks in the internal RAM?

You can use the following methods:

- You can load the individual logic and data blocks in the RAM using your PG.
- You can program a memory card (flash EPROM!) with your complete program including data blocks on the PG and then insert the card in the receptacle on the CPU.
  If you do an overall reset on the CPU (refer to Chapter 4) the complete contents of the memory card are loaded "1:1" in the internal RAM.
- You loaded your program in the internal RAM with the PG or from the memory card with an OVERALL RESET. You can then load additional blocks with the PG or replace existing blocks.

Note
You can only program the memory card on the PG. Use the PG software from version 6 upwards. When programming, the PG must be in the mode "WORD FIELD" (refer to the STEP 5 manual /3/).

Caution
If you have changed or added blocks using the PG after loading your program from the memory card, these changes are reversed by the next **OVERALL RESET**, since the memory is **overwritten** again with the contents of the memory card.

3.4 Processing the User Program

The complete software on the CPU (consisting of the system program and the STEP 5 user program) has the following tasks:

- CPU START-UP
- Controlling an automation process by continuously repeating operations (CYCLE).
- Controlling an automation process by reacting to events occurring sporadically or at certain times (interrupts) and reacting to errors.

For all three tasks, you can select special parts of your program to run on the CPU by programming user interfaces (organization blocks OB 1 to OB 35 - refer to Section 2.2.3).

START-UP

Before the CPU can start cyclic program execution, an initialization must be performed to establish a defined initial status for cyclic program execution and, for example, to specify a time base for the execution of certain functions. The way in which this initialization is performed depends on the event that led to a START-UP and on settings that you can make on your CPU. For more detailed information, refer to Chapter 4.

You can influence the START-UP procedure of your CPU by programming organization blocks OB 20, OB 21 and OB 22 or by assigning parameters in DX 0 (refer to Chapter 7).

CYCLE

Following the START-UP, the system program goes over to cyclic processing. It is responsible for background functions required for the automation tasks (refer to Fig. 3-1 at the beginning of this section). After the system functions have been executed at the beginning of a CYCLE, the system program calls organization block OB 1 or function block FB 0 as the cyclic user program. You program the STEP 5 operations for cyclic processing in this block.

Reactions to interrupts and errors

To allow you to specify the reactions to interrupts or errors, special organization blocks (OB 2 to OB 18 for interrupt servicing, OB 19 and OB 23 to OB 34 for reactions to errors) are available on the CPU 948. You can store an appropriate STEP 5 program in these blocks.

When interrupts or errors are to be processed, the system program activates the corresponding organization block during cyclic processing. This means that the cyclic processing is interrupted to service an interrupt or to react to an error. The nesting of the organization blocks has a fixed priority (for further information, refer to Chapters 4 and 5).

In addition to the organization blocks, you can also influence the reaction of the CPU to interrupt servicing by assigning parameters in data block DX 0.

Organization blocks OB 1 to OB 39 can be called by the system program as soon as they are loaded in the program memory (**also during operation**).
If the OBs are not loaded, there is either no reaction from the CPU or (in the event of errors) it goes to the stop mode (refer also to Section 5.4).

You can also load data block DX 0 into the program memory during operation like the organization blocks. **It is, however, only effective after the next COLD RESTART.** If DX 0 is not loaded, the standard settings apply (refer to Chapter 7).

3.4.1 Definition of Terms used in Program Execution

Cycle time

The cycle begins when the cycle monitoring time is triggered and ends with the next trigger. The time that the CPU requires to execute the program between two triggers is called the cycle time. The cycle time consists of the runtime of the system program and the runtime of the user program.

The cycle time therefore includes the following:

- the time required to process the cyclic program (system and user program),
- the time required to process interrupts (e.g. time-controlled interrupt),
- the time required to process interruptions (errors).

Cycle time monitoring

The CPU monitors the cycle time in case it exceeds a maximum value. The standard setting for this maximum value is 200 ms. You can set the cycle time monitoring yourself or restart it during user program execution (refer to DX 0/Chapter 7 and special function OB OB 222/Section 6.16).

Process input and output image (PII and PIQ)

The process image of the inputs and outputs is a memory area in the internal RAM.
Before cyclic execution of the user program begins, the system program reads the signal states of the input peripheral modules and transfers them to the process input image. The user program evaluates the signal states in the process input image and then sets the appropriate signal states for the outputs in the process output image. After the user program has been processed, the system program transfers the signal states of the process output image to the output peripheral modules.

Buffering the I/O signals in the process image of the inputs and outputs avoids a change in a bit within a program cycle from causing the corresponding output to "flutter".

The process image is therefore a memory area whose contents are output to the peripherals and read in from the peripherals **once per cycle**.

Note
The process image only exists for input and output bytes of the "P" peripherals with byte addresses from 0 to 127!
Apart from the process image integrated in the system, you can use OB 126 to define and transfer further process images (refer to Section 6.6)

Interprocessor communication (IPC) flags

IPC flags exchange data between individual CPUs (multiprocessing) or between the CPU and some communication processors.

The system program reads the input IPC flags of the CPU before cyclic execution of the user program begins. After the STEP 5 program is processed, the system program transfers the output IPC flags to the coordinator or to the communications processors.

You define the input and output IPC flags when you create data block DB 1 (refer to Section 10.1.6).

Interrupt events

Cyclic program execution can be interrupted by the following:

- time-controlled program execution (delayed interrupt, cyclic timed interrupts, clock-controlled interrupts),
- interrupt-driven program execution (process interrupt, system interrupt).

The cyclic program can be interrupted or even aborted completely by the following:

- a device hardware fault or program error
- operator intervention (using the PC stop function, or setting the mode selector to "stop", multiprocessor stop MP-STP),
- a stop operation

3.5 STEP 5 Operations with Examples

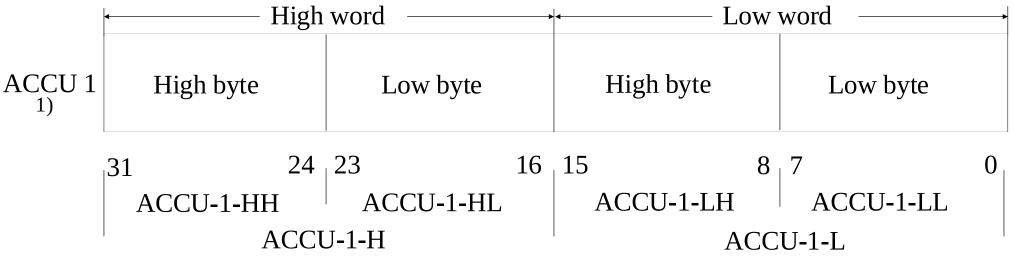
A STEP 5 operation consists of the operation and an operand. The operation specifies **what** the CPU is to do (operation). The operand specifies **with what** an operation is to be executed.

STEP 5 operations can be divided into the following groups:

- **basic operations** (can be used in **all** logic blocks),
- **supplementary operations,**
- **executive operations** (can only be used in FB/FX function blocks),
- semaphore operations (can only be used in FB/FX function blocks).

Accumulators as working registers

The CPU 948 has four accumulators, ACCU 1 to ACCU 4. Most STEP 5 operations use two 32-bit registers (ACCU 1 and ACCU 2) as the source of operands and the destination for results.



The STEP 5 operation to be carried out affects the accumulators, e.g.:

- ACCU 1 is always the destination in load operations. A load operation shifts the old contents of ACCU 1 to ACCU 2 (stack lift). Accumulators 3 and 4 are not changed by any load operations.

[1] analogous for ACCU 2 to ACCU 4

- Arithmetic operations combine the contents of ACCU 1 with those of ACCU 2, write the result to ACCU 1 and transfer the contents of ACCU 3 to ACCU 2 and the contents of ACCU 4 to ACCU 3 (stack drop). In 16-bit fixed point arithmetic, only the low word or ACCU 3 is transferred to the low word of ACCU 2 and the low word of ACCU 4 to the low word of ACCU 3.
- When a constant is added (ADD BF/KF/DH) to the contents of ACCU 1, the accumulators 2, 3 and 4 are not changed.

Condition codes

STEP 5 operations either set or evaluate condition codes. The condition codes are written to a condition code byte. Two groups of condition codes can be distinguished: condition codes of digital operations (word condition codes **-** bits 4 to 7 in the condition code byte) and condition codes from binary and executive operations (bit condition codes **-** bits 0 to 3 in the condition code byte). You can see how the various condition codes are influenced or evaluated by STEP 5 operations be referring to the operation list /1/.

You can display the condition code byte on a programmer using the "STATUS" online function (refer to Section 11.2.3). The byte has the following structure:

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

Bit condition codes

- **$\overline{\text{ERAB}}$** First bit scan

  A logic operation sequence containing binary operations always **begins** with the **first bit scan**, following which a **new** RLO is formed. The bit condition code $\overline{\text{ERAB}}$ = 1 is then set. While the remaining logic operations in the sequence are being performed, $\overline{\text{ERAB}}$ remains set to 1 and the RLO cannot be changed by these logic operations.

  The active sequence of logic operations is **terminated** by a binary set/reset operation (e.g. S Q 5.0). The set/reset operation sets $\overline{\text{ERAB}}$ to 0; the RLO can be evaluated (e.g. by RLO-dependent operations) but can no longer be combined logically. The next binary logic operation following a binary set/reset operation is once again a first bit scan.

Example of $\overline{\text{ERAB}}$

```
:A   I  1.0     ERAB is set to '1',
:               the new RLO is formed by
:               an AND operation
:O   I  6.3     The RLO is influenced by
:               an OR operation
:AN  I  2.1     The RLO is influenced by
:               an AND NOT operation.
:S   Q  2.4     ERAB is set to '0',
:               the sequence is now complete
:JC  FB 150     The function block is called
:               dependent on the RLO.
:
:
```

Other bit condition codes

- **RLO** Result of logic operation

  This is the result of bit logic operations. It is the truth statement for comparison operations (refer to operations list, binary logic operations or comparison operations).

- **STA** Status

  For bit operations, this indicates the logical status of the bit just scanned or set. The status is updated in binary logic operations - except for A(, O(,), O and for set/reset operations.

- **OR** Or

  Internal CPU bit for handling "AND before OR" logic operations.

Word condition codes

- **OV** Overflow

  This indicates whether the permissible number range was exceeded during the arithmetic operation just completed.

- **OS** Stored overflow

  The overflow bit is stored. It can be used in several arithmetic operations to indicate whether an overflow occurred at any point during the operations.

- **CC 1 and CC 0**

  These are the result condition codes that you can interpret from the following table:

> **Note**
> To evaluate the condition codes directly, comparison and jump operations are available (refer to Sections 3.5.1 and 3.5.3).

Table 3-1 Result condition codes of STEP 5 operations

Word condition codes		**Arith-metical operations**	**Digital logic operations**	**Com-parison operations**	**Shift operations**	**For SED, SEE**	**Jump operations executed**
CC 1	**CC 0**						
0	0	Result = 0	Result = 0	ACCU 2 = ACCU 1	Shifted bit = 0	Semaphore is set	JZ
0	1	Result < 0	–	ACCU 2 < ACCU 1	–	–	JM JN
1	0	Result > 0	Result ≠ 0	ACCU 2 > ACCU 1	Shifted bit = 1	Semaphore is set or enabled	JP JN
1	1	Division by 0	–	–	–	–	–

> **Note**
> When a change of level takes place, e.g. servicing a timed interrupt, all accumulators and the bit and word condition codes (RLO etc.) are saved and loaded again when the interrupted level is resumed.

3.5.1 Basic Operations

You can use the basic operations in **all** logic blocks and all methods of representation (STL, LAD, CSF).

Binary logic operations

Table 3-2 Binary logic operations

Operation	Operand	Function
A		AND logic operation after scanning for signal state "1"
O		OR logic operation after scanning for signal state "1"
	I 0.0 to 127.7	of an input in the PII
	Q 0.0 to 127.7	of an output in the PIQ
	F 0.0 to 255.7	of a flag bit
	S 0.0 to 4095.7	of an S flag bit
	D 0.0 to 255.15	of a data word bit
	T 0 to 255	of a timer
	C 0 to 255	of a counter
AN		AND logic operation after scanning for signal state "0"
ON		OR logic operation after scanning for signal state "0"
	I 0.0 to 127.7	of an input in the PII
	Q 0.0 to 127.7	of an output in the PIQ
	F 0.0 to 255.7	of a flag bit
	S 0.0 to 4095.7	of an S flag bit
	D 0.0 to 255.15	of a data word bit
	T 0 to 255	of a timer
	C 0 to 255	of a counter
O	–	Combine AND operations through logic OR
O U(O( )	–	ANDing of expressions in parentheses ORing of expressions in parentheses Close parenthesis (to complete the bracketed expression) Maximum of 8 levels are permitted, i.e. 7 opened brackets

RLO formation

The binary logic operations generate the result of logic operation (RLO).
At the beginning of a logic sequence, the RLO only depends on the signal state scanned (first scan) and not on the type of logic operation (O = OR, A = AND).

Within a sequence of logic operations, the RLO is formed from the type of operation, previous RLO and the scanned signal state. A sequence of logic operations is completed by an operation (e.g. set/reset operations) which retains the RLO (ERAB = 0). Following this, the RLO can be evaluated but cannot be further combined.

Example

Program	Status	RLO	$\overline{\text{ERAB}}$
:			
= Q 0.0	0	0	0 ← RLO retained
A I 1.0	1	1	1 ← first bit scan
A I 1.1	1	1	1
A I 1.2	0	0	1
= Q 0.1	0	0	0 ← RLO retained, end of the logic operations sequence

Set/reset operations

Table 3-3 Set/reset operations

Operation	Operand	Function
S R		Set if RLO = 1 Reset if RLO = 1
	I 0.0 to 127.7 Q 0.0 to 127.7 F 0.0 to 255.7 S 0.0 to 4095.7 D 0.0 to 255.15	an input in the PII an output in the PIQ a flag an S flag a bit in the data word
=		The RLO is assigned to
	I 0.0 to 127.7 Q 0.0 to 127.7 F 0.0 to 255.7 S 0.0 to 4095.7 D 0.0 to 255.15	an input in the PII an output in the PIQ a flag an S flag a bit in the data word

Load and transfer operations

Table 3-4 Load and transfer operations/part 1

Operation	Operand	Function
L		Load
T		Transfer
	IB 0 to 127	an input byte from/to the PII
	IW 0 to 126	an input word from/to the PII
	ID 0 to 124	an input double word from/to the PII
	QB 0 to 127	an output byte from/to the PIQ
	QW 0 to 126	an output word from/to the PIQ
	QD 0 to 124	an output double word from/to the PIQ
	FB 0 to 255	a flag byte
	FW 0 to 254	a flag word
	FD 0 to 252	a flag double word
	SY 0 to 4095	an S flag byte
	SW 0 to 4094	an S flag word
	SD 0 to 4092	an S flag double word
	DR 0 to 255	the right byte of a data word from/to DB,DX
	DL 0 to 255	the left byte of a data word from/to DB,DX
	DW 0 to 255	a data word from/to DB, DX
	DD 0 to 254	a data double word from/to DB, DX
	PY 0 to 127	a peripheral byte of the digital inputs/outputs (P area)
	PY 128 to 255	a peripheral byte of the analog or digital inputs/outputs (P area)
	PW 0 to 126	a peripheral word of the digital inputs/outputs (P area)
	PW 128 to 254	a peripheral word of the analog or digital inputs/outputs (P area)
	OY 0 to 255	a byte of the extended I/O area (O area)
	OW 0 to 254	a word of the extended I/O area (O area)

Table 3-5 Load and transfer operations/part 2

Operation	Operand		Function
L			Load
	KB	0 to 255	a constant, 1 byte
	KS	2 ASCII characters	a constant, 2 ASCII characters
	KF	-32768 to +32767	a constant as fixed point number
	KG	[1]	a constant as floating point number
	KH	0 to FFFF	a constant as hexadecimal number
	DH	0 to FFFF FFFF	a double word constant as a hexadecimal number
	KM	16-bit pattern	a constant as bit pattern
	KY	0 to 255 for each byte	a constant, 2 bytes
	KT	0.0 to 999.3	a constant timer value (in BCD)
	KC	0 to 999	a constant counter value
	T	0 to 255	a timer, binary coded
	C	0 to 255	a counter, binary coded
LC			Load
	T	0 to 255	a timer
	C	0 to 255	a counter
			in BCD

[1] ±0,1469368 x 10^{-38} to ±0,1701412 x 10^{39}

Load operations

Load operations write the addressed value into ACCU 1. The former contents of ACCU 1 are saved in ACCU 2 (stack lift).

Transfer operations

Transfer operations write the contents of ACCU 1 to the addressed memory location.

Examples of load and transfer operations

Example 1:

Fig. 3-6 illustrates loading/transferring a byte, word or double word from/to a memory area organized in **bytes** (PII, PIQ, flags, I/O).

```
   :L IB i    load byte i of the PII into ACCU-1-LL
   :L IW j    load bytes j and j+1 of the PII into ACCU-1-L
   :L FD k    load flag bytes k to k+3 in ACCU 1
```

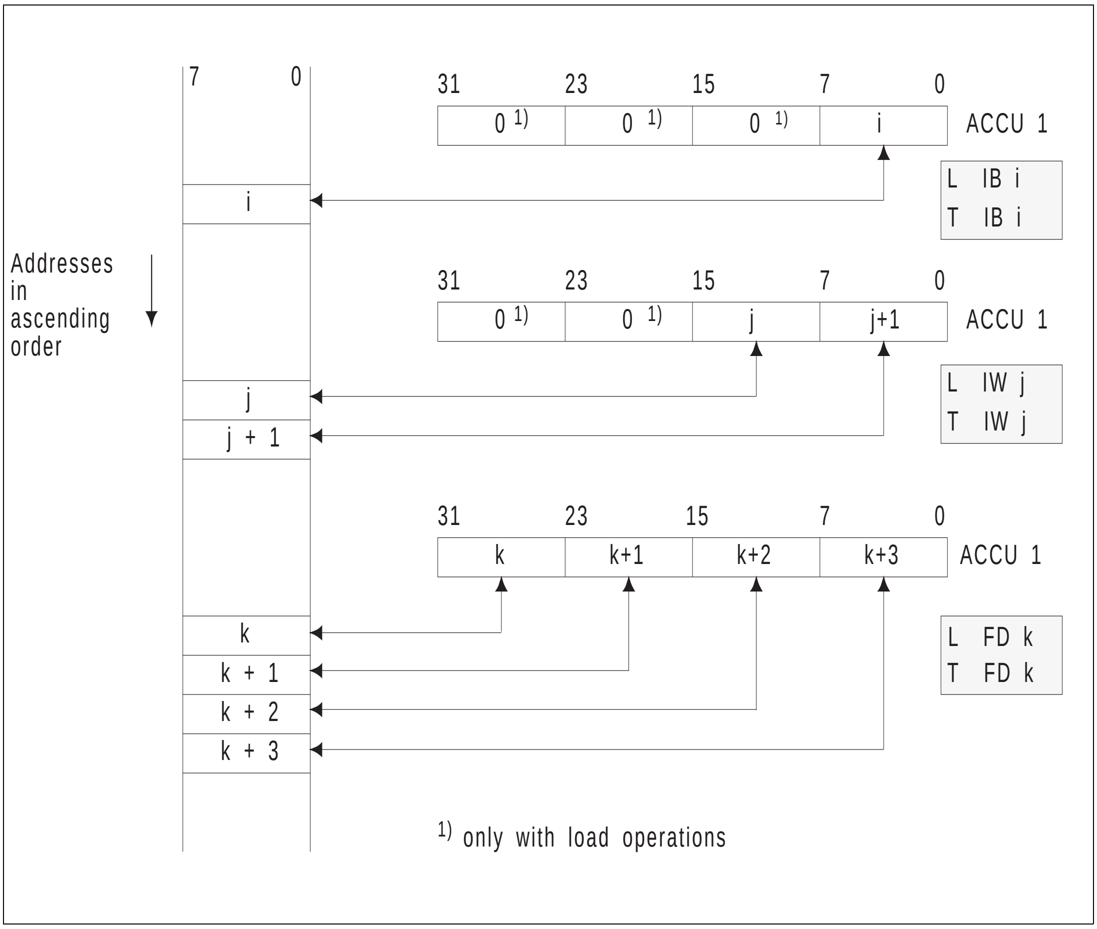


Fig. 3-6 Load and transfer operations in a byte-oriented memory area

Example 2:

Fig. 3-7 illustrates the loading/transfer of a byte, word or double word from/into a memory area organized in **words**.

```
   :L DR i load the right byte of data word i into ACCU-1-LL
   :L DL j load the left byte of data word j into ACCU-1-LL
   :L DW k load data word k into ACCU-1-L
   :L DD l load data words l and l+1 into ACCU 1
```

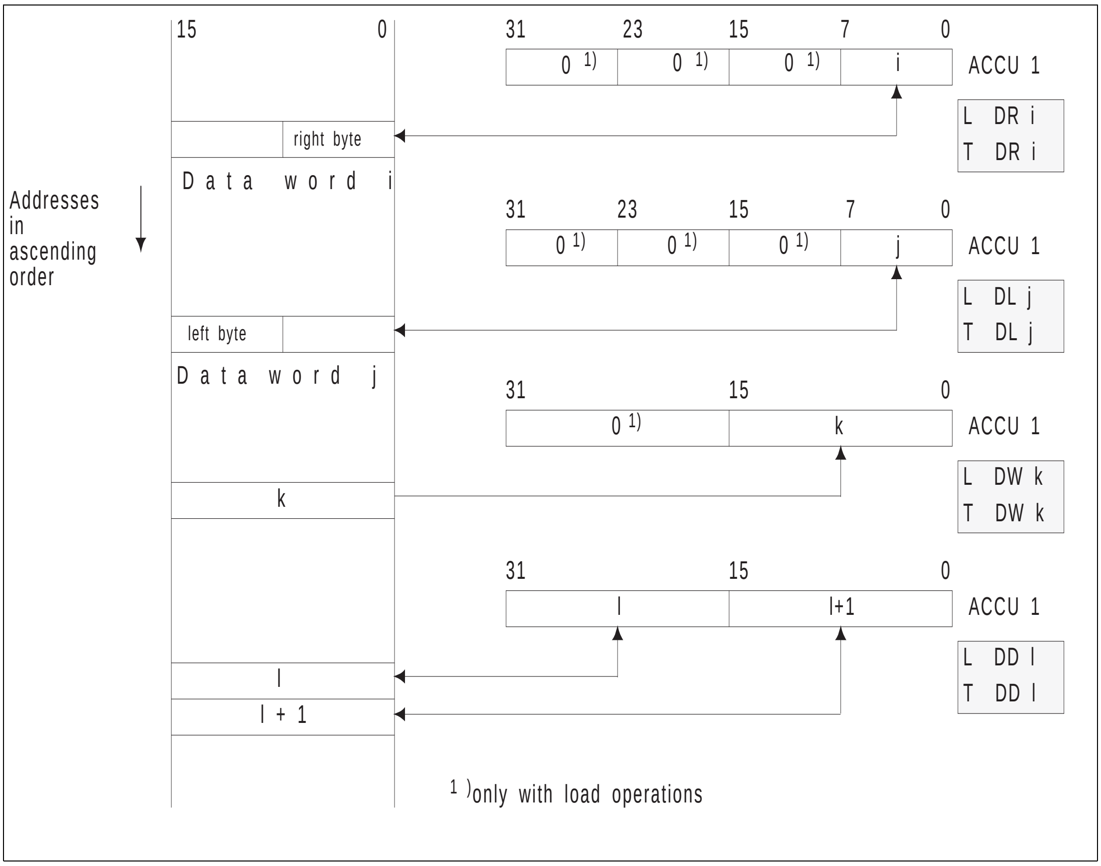


Fig. 3-7 Load and transfer operations in a word-oriented memory area

Note
Load operations do not affect the **condition codes**.
Transfer operations clear the **OS bit**.

When a **byte** or **word** is **loaded** the **extra bits** are **cleared** in ACCU 1.

Addressing I/Os

You can use load and transfer operations to address the I/O peripherals as follows:

- **directly using the following operations:**

  L../T.. ..PY, ..PW, ..OY, ..OW

or

- **using the process image with the following operations:**

  L../T.. ..IB, ..IW, ..ID, .QB, ..QW, ..QD

  and with logic and set/reset operations

Note
If you use the transfer operations T PY 0 to 127 and T PW 0 to 126, the process output image is updated at the same time. Exception: command output is disabled by the STEP 5 operation BAS (refer to Section 3.5.4).

Note the following points about I/O peripherals:

- A process input/output image exists for 128 input and 128 output bytes of the P peripherals with byte addresses from 0 to 127.
- No process image exists for the entire area of the O peripherals and the P peripherals with relative byte addresses from 128 to 256. (For more information on address space allocation see Section 8.2.2).
- I/O modules with addresses of the O peripherals can only be plugged into expansion units (not in the central controller).
- In **one** expansion unit, you can use either only P peripherals or only O peripherals.

Caution
If you use relative addresses of the O peripherals in an expansion unit, you can no longer use these addresses for I/O modules in the central controller (this would result in double addressing).

Timer and Counter operations

To load a timer using a start operation or a counter using a set operation, you must first load the value in ACCU 1.

The following load operations are preferable:

For timers: L KT, L IW, L QW, L FW, L DW, L SW.
For counters: L KC, L IW, L QW, L FW, L DW, L SW.

Starting a **timer** with the selected timer value requires an RLO signal change.

A **counter** is set or started with the selected counter value when a positive-going RLO signal edge is detected.

The following table indicates the signal edge change with corresponding arrows.

Table 3-6 Timer and counter operations

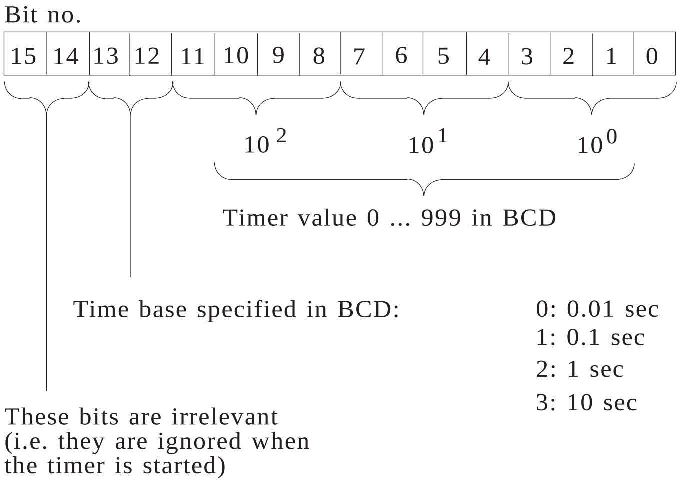
Operation	Operand	RLO 1)	Function
SP	T 0 to 255	↑	Start a timer as a pulse
SE	T 0 to 255	↑	Start a timer as extended pulse
SD	T 0 to 255	↑	Start a timer as ON delay
SS	T 0 to 255	↑	Start a timer as stored ON delay
SF	T 0 to 255	↓	Start a timer as OFF delay
R	T 0 to 255	1	Reset a timer
S	C 0 to 255	↑	Set a counter (BCD number from 0 to 999)
R	C 0 to 255	1	Reset a counter
CU	C 0 to 255	↑	Count up
CD	C 0 to 255	↑	Count down

[1] positive-going edge (↑): signal change from ’0’ to ’1’
negative-going edge (↓): signal change from ’1’ to ’0’

When executing the timer or counter operations SP T, SE T, SD T, SS T, SF T and S C the value in ACCU 1 is transferred to the timer or counter (as with the transfer operation) and the appropriate operation is started.

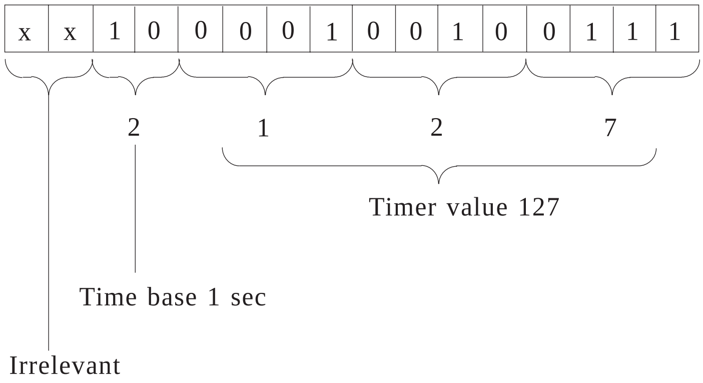
Timer value

With the operation L KT, you can load a **timer value** directly into ACCU 1 or indirectly from a flag or data word. The value must have the following structure (with L KT, you specify the time base after the period in the operand as shown below):



Example

```
You want to set a time of 127 sec.:

Bit assignment:
```



Note

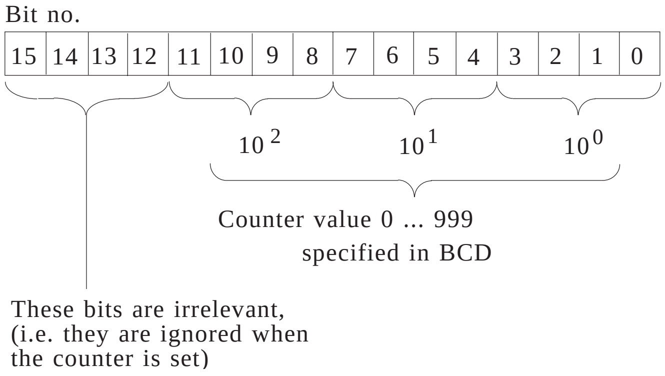
The start of each timer is liable to an inaccuracy of 1 time base! When using timers, you should therefore select the smallest possible time base (time base < timer value):

Example:

time value 4s	not:	1 s x 4	inaccuracy: 1 s
	but:	0.01 s x 400	inaccuracy: 0.01 s

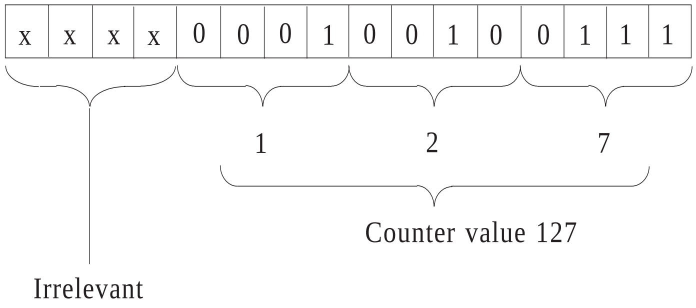
Counter value

With the operation L KC, you can load a **counter value** directly in ACCU 1 or indirectly from a flag or a data word. The value must have the following structure:



Example

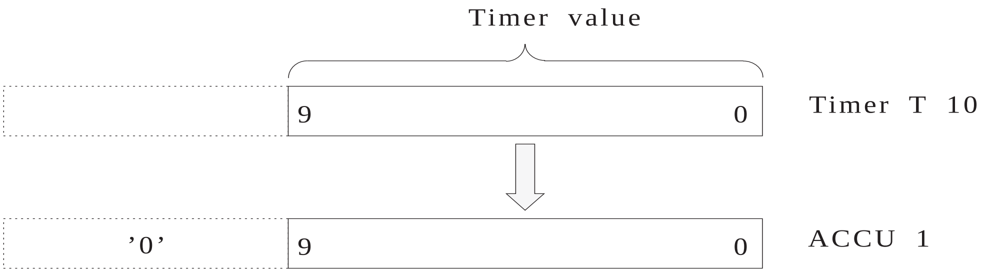
```
You want to specify a counter value of 127:

Bit assignment:
```



In the timer or counter itself, the value is in binary code. If you want to scan the timer or counter, you can load the actual timer or counter value into ACCU 1 **directly** or **in BCD code**.

Further examples of timer and counter values

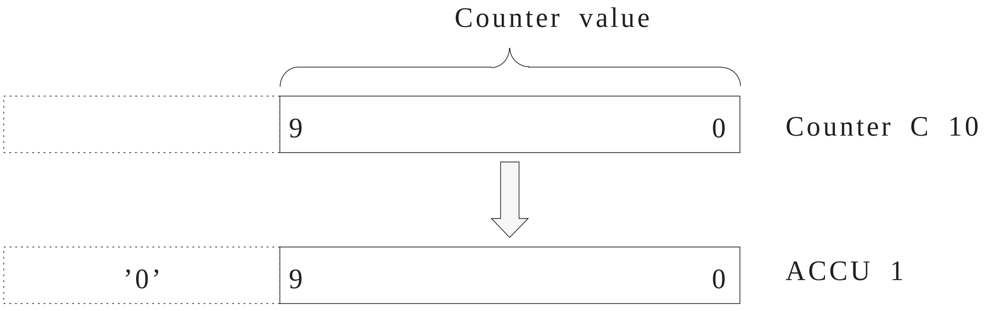
Loading **timer values** directly:



```
"L  T 10":   Loads the binary timer value of timer T 10
             directly into ACCU 1
```

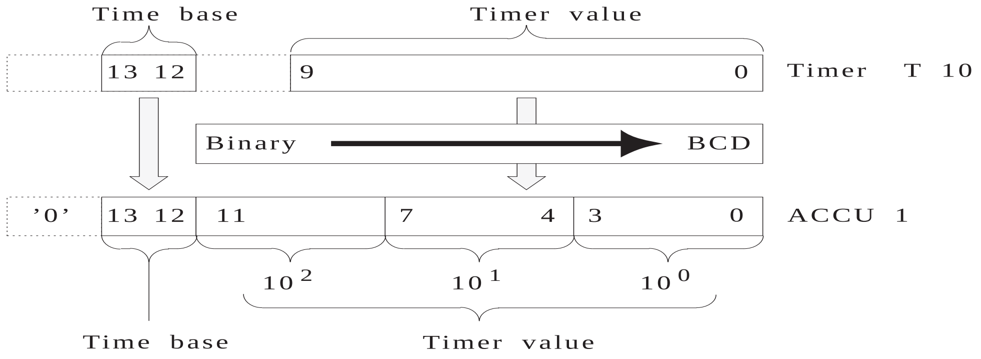
The time base is not loaded.

Loading **counter values** directly:



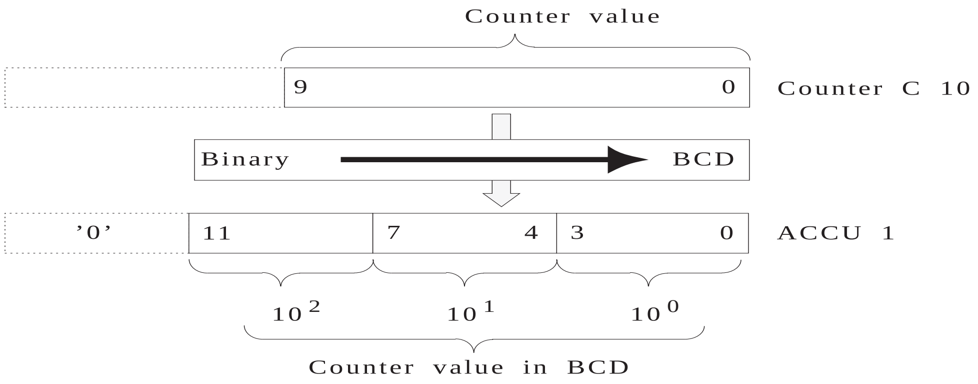
```
"L  C 10":   Loads the binary counter value of counter C 10
             directly into ACCU 1
```

Loading **timer values** in BCD code:



```
"LC T 10": Loads the timer value and time base of
           timer T 10 into ACCU 1 in BCD

The time base is also loaded.
```

Loading **counter values** in BCD code:



```
"LC  C 10":  Loads the counter value of counter C 10
             into ACCU 1 in BCD
```

If you load values in BCD, status bits 14 and 15 of the timer or 12 to 15 of the counter are not loaded. They have the value 0 in ACCU 1. The value in the ACCU can now be processed further.

Arithmetic operations

Table 3-7 Arithmetic operations

Operation	Operand	Function
+ F	–	Add two fixed point numbers (16 bits)
- F		Subtract one fixed point number from another (16 bits)
x F		Multiply two fixed point numbers (16 bits)
: F		Divide one fixed point number by another (16 bits): quotient in ACCU-1-L, remainder in ACCU-1-H
+ G		Add two floating point numbers (32 bits)
- G		Subtract one floating point number from another (32 bits)
x G		Multiply two floating point numbers (32 bits)
: G		Divide one floating point number by another (32 bits)

Arithmetic operations logically combine the contents of ACCU 1 and ACCU 2 (e.g. ACCU 2 - ACCU 1). The result is then contained in ACCU 1. An arithmetic operation changes the arithmetic registers as follows (in fixed point operations only the low word):

	ACCU 1	**ACCU 2**	**ACCU 3**	**ACCU 4**
before:	<ACCU 1>	<ACCU 2>	<ACCU 3>	<ACCU 4>
after:	<result>	<ACCU 3>	<ACCU 4>	<ACCU 4>

Note
Within the **supplementary operations**, there are operations for **subtraction** and **addition** of **double word fixed point numbers**.

Comparison operations

Table 3-8 Comparison operations

Operation	Operand	Function
! = >< F > D > = G < <=	–	Compare for equal to Compare for not equal to Compare for greater than Compare for greater than or equal to Compare for less than Compare for less than or equal to ...F: compare two fixed point numbers (16 bits) ...D: compare two fixed point numbers (32 bits) ...G: compare two floating point numbers (32 bits)

Block operations

Table 3-9 Block operations

Operation	Operand	Function
J U J C	 OB 1 to 39 [1)] OB 121 to 255 PB 0 to 255 FB 0 to 255 SB 0 to 255	Jump unconditionally Jump conditionally (only when RLO = 1) to an organization block to a system program special function to a program block to an FB function block to a sequence block
D O U D O C	 FX 0 to 255	Jump unconditionally Jump conditionally (only when RLO = 1) to an FX function block
B E B E C B E U	–	Block end Block end, conditional (only when RLO = 1) Block end, unconditional
C C X	DB 2 to 255 DX 3 to 255	Call a DB data block Call a DX data block
G GX	DB 2 to 255 DX 3 to 255	Generate data block DB Generate data block DX (ACCU 1 must contain the number of data words – maximum 4091 – that the new block is to have)

[1)] only for test purposes!

G DB/GX DX

Generating a data block

The operation G DBx generates a DB data block with the number x (2 ≤ x ≤ 255) in the user memory of the CPU. The content of the data block is **not** assigned the value 0, i.e. the data words can have any contents.
Before programming this statement, you must store the number of data words that the new DB is to have in ACCU-1-L. The operation "G DB" or "GX DX" creates the block header. A data block generated in this way (**without** block header) can occupy a maximum of 4091 words. You can generate longer data blocks using OB 125 (refer to Section 6.5).

If the data block already exists, the length of the DB is not permitted or there is not enough space in the DB-RAM, the system program calls **OB 34**. If this is not loaded, the CPU goes to the stop mode.

The GX DXx operation generates a DX data block in the DB-RAM and is otherwise the same as G DBx.

NOP/display/stop operations

Table 3-10 NOP/display/stop operations

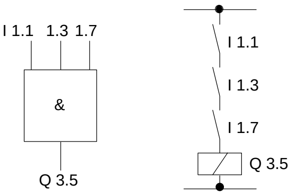
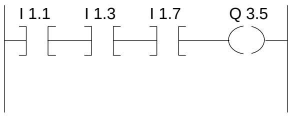
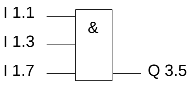
Operation	**Operand**	**Function**
N O P 0 N O P 1	–	No operation No operation
B L D	0 to 255 130 131/131/133 255	Display generation operation for the PG: the CPU handles the operation like a no operation Create blank line with carriage return Switch over between STL, CSF, LAD
S T P	–	At end of cycle or at end of OB 1, CPU changes to soft STOP.

Note
Since the operation STP is only effective at the end of the cycle, there is no ISTACK entry. The cause of the stoppage is then difficult to find afterwards.
To make diagnosis easier, you should set an identifier before calling the STP operation, e.g. a special bit pattern in a diagnostic DB or use the STEP 5 operation STS (refer to Section 3.5.4).

3.5.2 Programming Examples in the STL, LAD and CSF Methods of Representation

Logic operations

AND operation

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 1.1 1.3 1.7 & Q 3.5 I 1.1 I 1.3 I 1.7 Q 3.5	A I 1.1 A I 1.3 A I 1.7 = Q3.5		

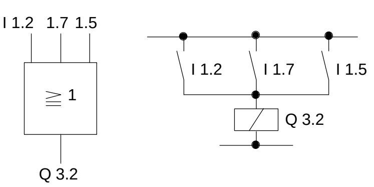
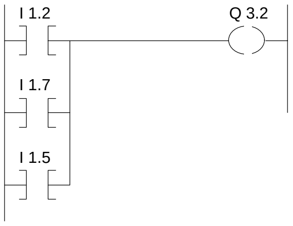
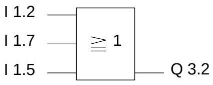
Output Q 3.5 is "1" when all inputs are "1" simultaneously

Output Q 3.5 is "0" if any of the inputs has signal state "0"

The number of scans and the sequence of the logic statements are optional

Logic operations (continued)

OR operation

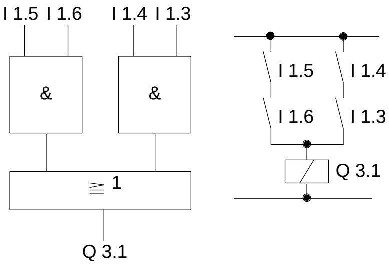
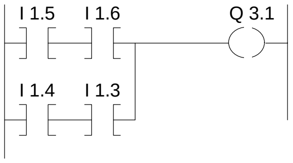
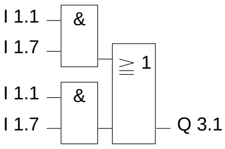
Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 1.2 1.7 1.5 ≧ 1 Q 3.2 I 1.2 I 1.7 I 1.5 Q 3.2	O I 1.2 O I 1.7 O I 1.5 = Q3.2		

Output Q 3.2 is "1" when at least one of the inputs is "1"

Output Q 3.2 is "0" when all inputs have the signal state
state "0" simultaneously

The number of scans and sequence of programming is optional

AND-before-OR operation

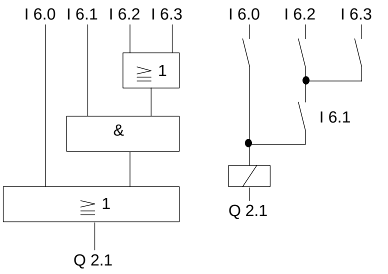
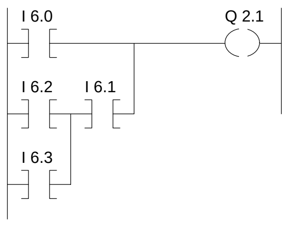
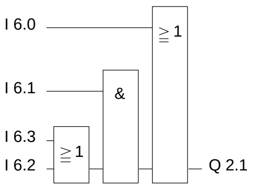
Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 1.5 I 1.6 I 1.4 I 1.3 & & ≧ 1 Q 3.1 I 1.5 I 1.4 I 1.6 I 1.3 Q 3.1	A I 1.5 A I 1.6 O A I 1.4 A I 1.3 = Q3.1		

Q 3.1 is "1" when at least one AND condition is satisfied

Q 3.1 is "0" when no AND condition is satisfied

Logic operations (continued)

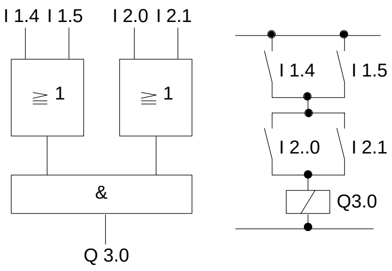
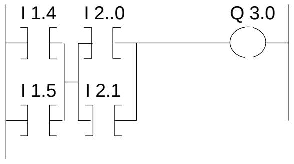
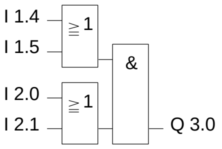
OR-before-AND operation /1st example

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 6.0 I 6.1 I 6.2 I 6.3 ≧ 1 & ≧ 1 Q 2.1 I 6.0 I 6.2 I 6.3 I 6.1 Q 2.1	A I 6.0 O A I 6.1 A (O I 6.2 O I 6.3) = Q2.1		 I 6.0 ≧ 1 I 6.1 & I 6.3 ≧ 1 I 6.2 Q 2.1

Output Q 2.1 is "1" when input I 6.0 or input I 6.1 and one of the inputs I 6.2 or I 6.3 has signal state "1"

Output Q 2.1 is "0" when input I 6.0 has signal state "0" and the AND condition is not satisfied

OR-before-AND operation /2nd example

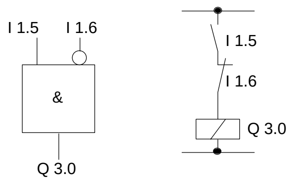
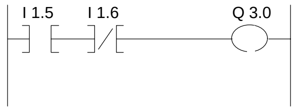
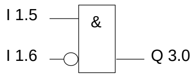
Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 1.4 I 1.5 I 2.0 I 2.1 ≧ 1 ≧ 1 & Q 3.0 I 1.4 I 1.5 I 2..0 I 2.1 Q3.0	A (O I 1.4 O I 1.5) A (O I 2.0 O I 2.1) = Q3.0		 I 1.4 ≧ 1 I 1.5 & I 2.0 ≧ 1 I 2.1 Q 3.0

Output Q 3.0 is "1" when both OR conditions are satisifed

Output Q 3.0 is "0" when at least one OR condition is not satisfied

Logic operations (continued)

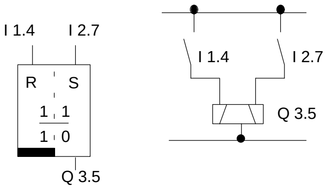
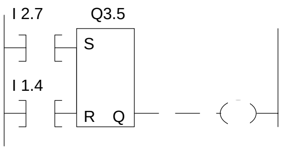
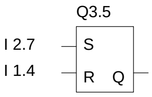
Scan for signal state "0"

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 1.5 AN I 1.6 = Q3.0		

Output Q 3.0 is "1" only when input I 1.5 has signal state "1" (normally open contact activated) and input I 1.6 has signal state "0" (normally closed contact activated)

Set/reset operations

RS flip-flop for a latching signal output

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 1.4 I 2.7 R S 1 1 1 0 Q 3.5 I 1.4 I 2.7 Q 3.5	A I 2.7 S Q 3.5 A I 1.4 R Q 3.5		

Signal state "1" at input I 2.7 sets the flip-flop (signal state "1" at output Q 3.5).
If the signal state at input I 2.7 changes to "0", the state of output Q 3.5 is retained (i.e. the signal is latched).

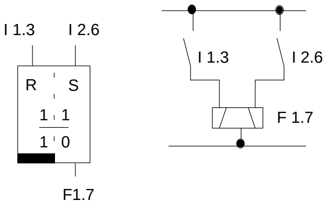
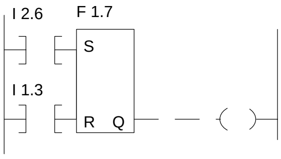
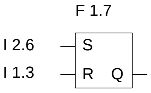
Signal state "1" at input I 1.4 resets the flip-flop (signal state "0" at output Q 3.5).

If the signal state at input I 1.4 changes to "0", the state of Q 3.5 is retained.

When the set signal (input I 2.7) and the reset signal (input I 1.4) are applied at the same time, the scan operation programmed last (in this case AI 1.4) remains in effect for the rest of the program (reset priority).

Set/reset operations (continued)

RS flip-flop with flags

<table>
<tr><th rowspan="2">Logical/circuit diagram</th><th colspan="3">STEP 5 representation</th></tr>
<tr><th>Statement list</th><th>Ladder diagram</th><th>Control system flowchart</th></tr>
<tr>
<td>
</td>
<td>A I 2.6
S F 1.7
A I 1.3
R F 1.7</td>
<td>
</td>
<td>
</td>
</tr>
</table>

Signal state "1" at input I 2.6 sets the flip-flop.

If the signal state at input I 2.6 changes to "0", the signal state of the flag is retained, i.e. the signal is latched.

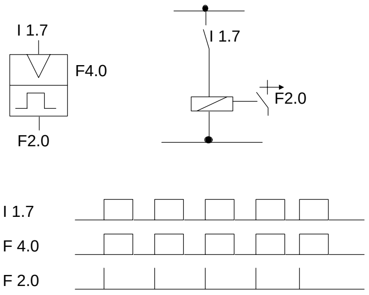
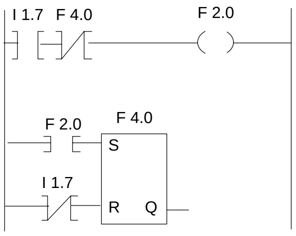
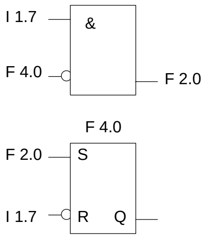
Signal state "1" at input I 1.3 resets the flip-flop.

If the signal state at input I 1.3 changes to "0", the signal state of the flag is retained.

When the set signal (input I 2.6) and the reset signal (input I 1.3) are applied at the same time, the scan operation last programmed (in this case AI 1.3) remains in effect for the rest of the program (reset priority).

Set/reset operations (continued)

Simulation of a momentary contact relay (one shot)

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	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		

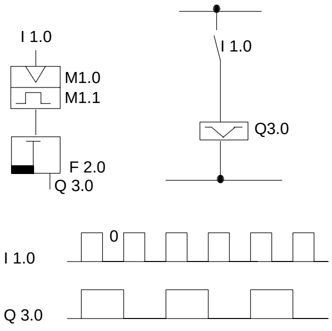
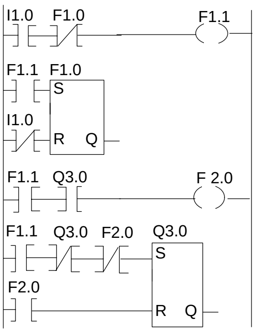
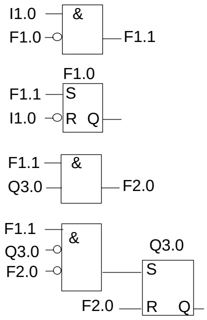
On each leading edge of the signal at input I 1.7, the AND condition (AI 1.7 and AN F 4.0) is satisfied; the RLO is "1". This sets flags F 4.0 (edge flag) and F 2.0 (pulse flag).

In the next processing cycle, the AND condition AI 1.7 and AN F 4.0 is not satisfied, since flag F 4.0 has already been set.

Flag F 2.0 is reset.

Flag F 2.0 therefore only remains "1" for one program run.

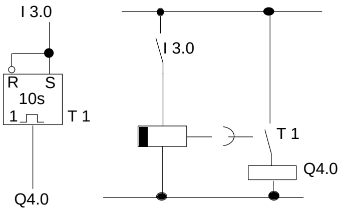
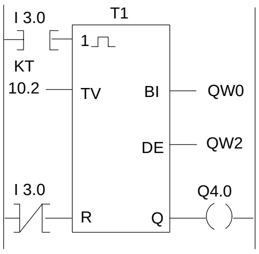
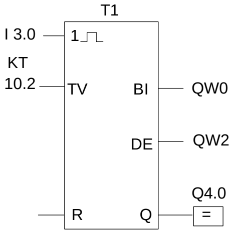
Binary scaler (binary divider)

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	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 Q3.0 = F 2.0 A F 1.1 AN Q3.0 AN F 2.0 S Q 3.0 A F 2.0 R Q 3.0		

The binary scaler (output Q 3.2) changes its state each time input I 1.0 changes its signal state from 0 to 1 (leading edge). Therefore, only half the input frequency appears at the output of the memory cell.

Timer operations

Pulse timer

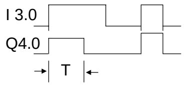
Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 3.0 R S 10s 1 T 1 Q4.0 I 3.0 T 1 Q4.0	A I 3.0 L KT 10.2 SP T 1 AN I 3.0 R T 1 L T 1 T QW 0 LC T 1 T QW 2 A T 1 = Q 4.0	 I 3.0 T1 1 KT 10.2 TV BI QW0 DE QW2 I 3.0 Q4.0 R Q	 T1 I 3.0 1 KT 10.2 TV BI QW0 DE QW2 Q4.0 R Q =

The timer is started during the first scan if the RLO is "1". Subsequent scans with an RLO of "1" do not affect the timer.

If the RLO is "0", the timer is reset (cleared).

The scan AT or OT produces the signal "1" as long as the timer is running.

KT 10.2:

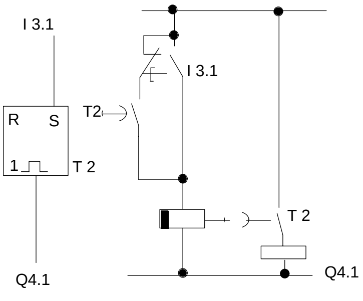
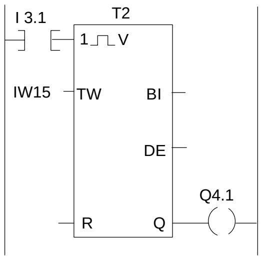
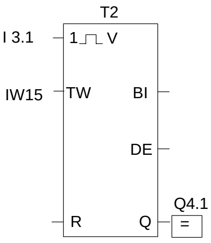


The timer is loaded with the specified value (10).
The number to the right of the decimal point indicates the time base:
0 = 0.1sec 2 = 1sec
1 = 0.1 sec 3 = 10 sec

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

Timer operations (continued)

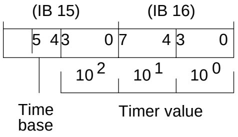
Extended pulse timer

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 3.1 R S 1 T 2 T2 I 3.1 T 2 Q4.1 Q4.1	A I 3.1 L IW 15 SE T 2 A T 2 = Q 4.1	 I 3.1 T2 1 V IW15 TW BI DE Q4.1 R Q	 T2 I 3.1 1 V IW15 TW BI DE Q4.1 R Q =

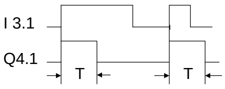
The timer is started during the first scan if the RLO is "1".

An RLO of "0" does not affect the timer.

The scan AT or OT produces a signal "1" as long as the timer is running.

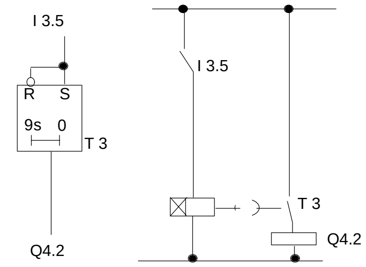
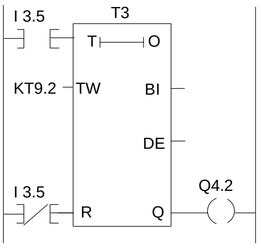
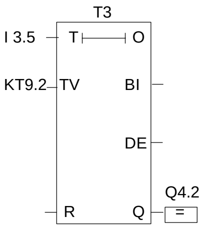


IW 15:
Set the timer with the value of the operand I, Q, F or D in BCD code (in this example, input word 15).



Timer operations (continued)

ON-delay timer

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 3.5 L KT 9.2 SD T 3 AN I 3.5 R T 3 A T 3 = Q 4.2		

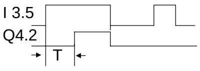
The timer is started during the first scan if the RLO is "1". An RLO of "1" during subsequent scans does not affect the timer.

When the RLO is "0", the timer is reset (cleared).

The scan AT or OT produces the signal "1" when the timer has elapsed and the RLO is still applied to the input.

KT 9.2:

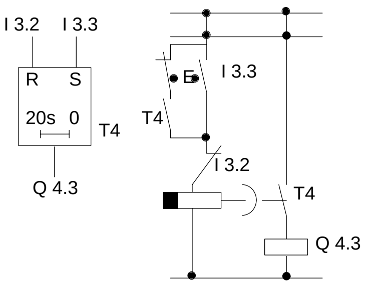
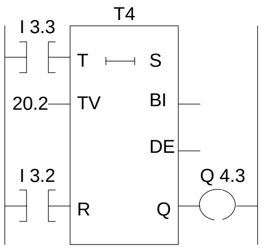
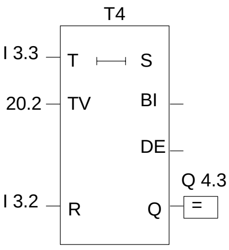
The timer is loaded with the specified value (9). The number to the right of the decimal point indicates the time base:



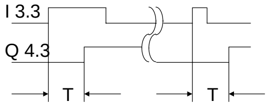
0 = 0.1sec	2 = 10 sec
1 = 0.1 sec	3 = 10 sec

Timer operations (continued)

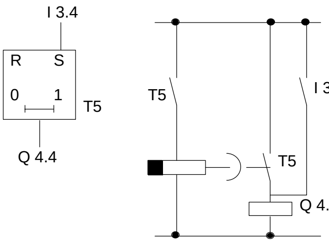
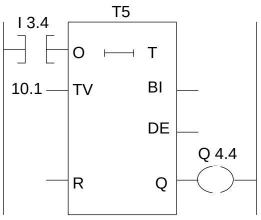
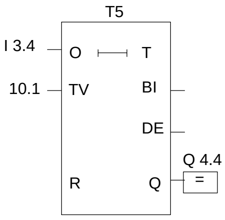
Stored ON-delay timer

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 3.2 I 3.3 R S 20s 0 T4 Q 4.3 E I 3.3 T4 I 3.2 T4 Q 4.3	A I 3.3 L KT 20.2 SS T 4 A I 3.2 R T 4 A T 4 = Q 4.3	 T4 I 3.3 T S 20.2 TV BI DE I 3.2 Q 4.3 R Q	 T4 I 3.3 T S 20.2 TV BI DE Q 4.3 I 3.2 R Q =

The timer is started during the first scan if the RLO is "1".
An RLO of "0" does not affect the timer.
The scan AT or OT produces the signal "1" when the timer has elapsed. The signal state does not change to "0" until the R T operation resets the timer.

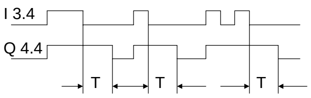


OFF-delay timer

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 3.4 R S 0 1 T5 Q 4.4 T5 I 3.4 T5 Q 4.4	A I 3.4 L KT 10.1 SF T 5 A T 5 = Q 4.3	 T5 I 3.4 O T 10.1 TV BI DE Q 4.4 R Q	 T5 I 3.4 O T 10.1 TV BI DE Q 4.4 R Q =

When the RLO at the start input changes from "1" to "0", the timer is started. It runs for the length of time programmed.

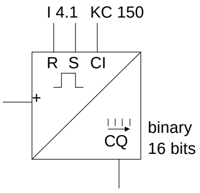
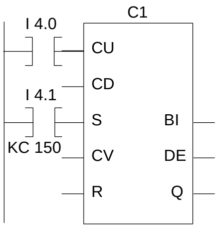
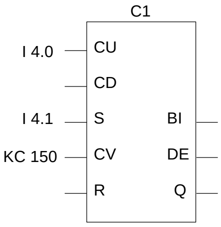
When the RLO is "1", the timer is reset (cleared).



The scan AT or OT produces signal state "1" if the timer is running <u>or</u> the RLO at the input is "1".

Counter operations

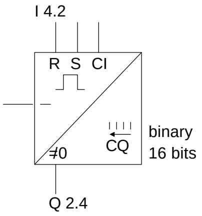
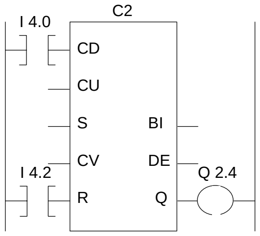
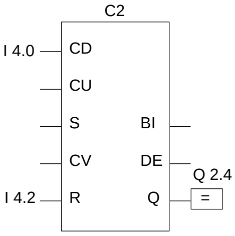
Set counter

Logical/circuit operation	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 4.0 CU C 1 A I 4.1 L KC 150 S C 1	 C1 I 4.0 CU CD I 4.1 S BI KC 150 CV DE R Q	 C1 I 4.0 CU CD I 4.1 S BI KC 150 CV DE R Q

When the result of logic operation changes at the start input (I 4.1) from "0" to "1", the counter is loaded with the specified value (150).

The flag necessary for edge evaluation of the set input is incorporated in the counter word.
BI and DE are digital outputs of the counter cell. The value at BI is in binary code and the value at DE is in BCD.

Reset counter

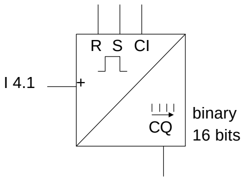
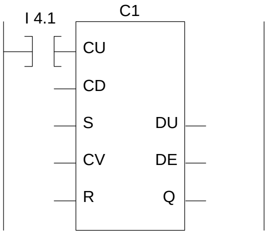
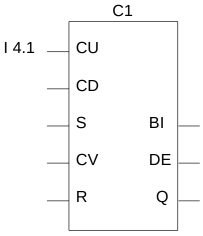
Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 4.2 R S CI – CQ ≠0 binary 16 bits Q 2.4	A I 4.0 CD C 2 A I 4.2 R C 2 A C 2 = Q 2.4	 C2 I 4.0 CD CU S BI CV DE I 4.2 R Q Q 2.4	 C2 I 4.0 CD CU S BI CV DE Q 2.4 I 4.2 R Q =

An RLO of "1" (I 4.2) resets the counter to zero.

An RLO of "0" does not affect the counter.

Counter operations (continued)

Count up

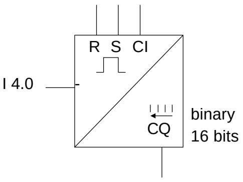
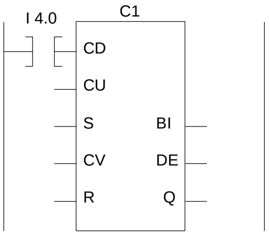
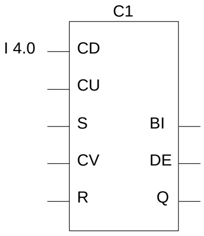
Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 4.1 CU C 1		

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

Owing to the two separate edge flags for CU and CD, a counter with two different inputs can be used as an up/down counter.

Counter operations
(continued)

Count down

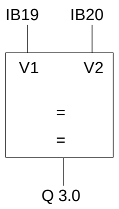
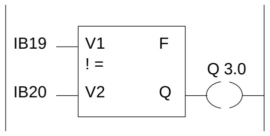
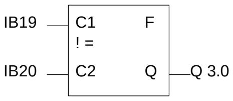
Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 4.0 CD C 1		

The value of the addressed counter is decremented by 1 to a maximum counter value of 0. The function is only executed on a positive edge (from "0" to "1") of the logic operation programmed before the CD. The flags necessary for edge evaluation of the counter inputs are incorporated in the counter word.

Owing to the two separate edge flags for CU and CD, a counter with two different inputs can be used as an up/down counter.

Comparison operations

Compare for equal to

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	L I B19 L IB20 ! = F = Q 3.0		

The first operand is compared with the second operand by the comparison operation. The RLO of the comparison is binary.

RLO = "1": comparison is satisfied if ACCU-1-L = ACCU-2-L
RLO = "0": comparison is not satisfied, when ACCU-1-L is not equal to ACCU-2-L.

The condition codes CC1 and CC0 are set as described in the list of operations.

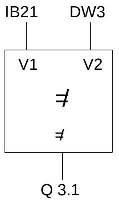
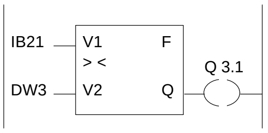
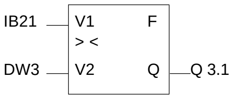
ACCU-2-H and ACCU-1-H are not involved in the operation for a 16-bit fixed point comparison.

In a 32-bit fixed point comparison (! = D) and floating point comparison (! = G) the entire contents of ACCU 1 and ACCU 2 (32 bits) 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 here as a fixed point number.

Comparison operations (continued)

Compare for not equal to

Logical/circuit diagram	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	L I B21 L DW3 > < F = Q 3.1		

The first operand is compared with the second operand by the comparison operation.
The RLO of the comparison is binary.
RLO = "1": comparison is satisfied if ACCU-1-L is not equal to ACCU-2-L.
RLO = "0": comparison is not satisfied if ACCU-1-L equals ACCU-2-L.
The condition codes CC1 and CC0 are set as described at the beginning of Section 3.5.
ACCU-2-H and ACCU-1-H are not involved in the operation for a 16-bit fixed point comparison.
ACCU-2-H and ACCU-1-H are involved in a 32-bit fixed point comparison and floating point comparison.
This information also applies to comparison operations for "greater than", "greater than or equal to", "less than" and "less than or equal to" (see the operations list). 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 here as a fixed point number.

3.5.3 Supplementary Operations

You can use the supplementary operations set on the programmer only in function blocks (FB and FX). This means that the total operations set for function blocks consists of the basic operations and the supplementary operations.

The system operations also belong to the supplementary functions. You can use the system operations, for example to overwrite the memory at optional locations or to change the contents of the working registers of the CPU. System operations can only be programmed if they have been enabled in the presets menu of the programmer (no longer necessary from S5-DOS Version 2.0 upwards).

If you intend to use system operations, you should be familiar with Chapter 9 "Memory access".

⚠ **Caution**

Only experienced system programmers should use the system operations and then only with extreme caution.

You can only write operations in function blocks in STL. You cannot program function blocks in graphic form (LAD and CSF methods of representation).
This section describes the supplementary operations and covers possible combinations of substitution operations with actual operands.

System operations

System operations are marked in the first column of the tables with **S**

Binary logic operations

Table 3-11 Binary logic operations with formal operands

Operation	Operand	Function
A =	□	AND operation, scan a formal operand for signal state ’1’
AN =	□	AND operation, scan a formal operand for signal state ’0’
O =	□	OR operation, scan a formal operand for signal state ’1’
ON =	□	OR operation, scan a formal operand for signal state ’0’
		Insert formal operand Inputs, outputs, data and flags addressed in binary (parameter types: I, Q; data type BI) and timers and counters (parameter type: T, C) are permitted as actual operands.

Digital logic operations

Table 3-12 Digital logic operations

Operation	Operand	Function
AW		AND operation on the contents of ACCU-1-L and ACCU-2-L
OW		OR operation on the contents of ACCU-1-L and ACCU-2-L
XOW		Exklusive OR operation on the contents of ACCU-1-L and ACCU-2-L

ACCUs 2, 3 and 4 are not affected, however, the condition codes CC 1 and CC 0 are affected (see word condition codes).

Bit test operations

Table 3-13 Bit test operations

Operation	Operand	Function
TB		Scan for signal state "1"
	I 0.0 to 127.7	of an input (PII)
	Q 0.0 to 127.7	of an output (PIQ)
	F 0.0 to 255.7	of a flag
	D 0.0 to 255.15	of a data word bit
	T 0.0 to 255.15	of a timer word bit
	Z 0.0 to 255.15	of a counter word bit
	RI 0.0 to 255.15	of a bit in RI area
	RJ 0.0 to 255.15	of a bit in RJ area
	RS 0.0 to 255.15	of a bit in RS area
	RT 0.0 to 255.15	of a bit in RT area
TBN		Scan for signal state "0"
	I 0.0 to 127.7	of an input (PII)
	Q 0.0 to 127.7	of an output (PIQ)
	F 0.0 to 255.7	of a flag
	D 0.0 to 255.15	of a data word bit
	T 0.0 to 255.15	of a timer word bit
	C 0.0 to 255.15	of a counter word bit
	RI 0.0 to 255.15	of a bit in RI area
	RJ 0.0 to 255.15	of a bit in RJ area
	RS 0.0 to 255.15	of a bit in RS area
	RT 0.0 to 255.15	of a bit in RT area

The bit test operations scan the state of the bit and indicate it via the RLO.

Set/reset operations

Table 3-14 Set/reset operations with formal operands

Operation	Operand	Function
S =	□	Set a formal operand (binary)
RB =	□	Reset a formal operand (binary)
RD=	□	Reset a formal operand (digital) for timers and counters
= =	□	Assign the value of the RLO to a formal operand
		Insert formal operand
		Inputs, outputs and F flags addressed in binary (parameter type: I, Q; data type BI) are permitted as actual operands.

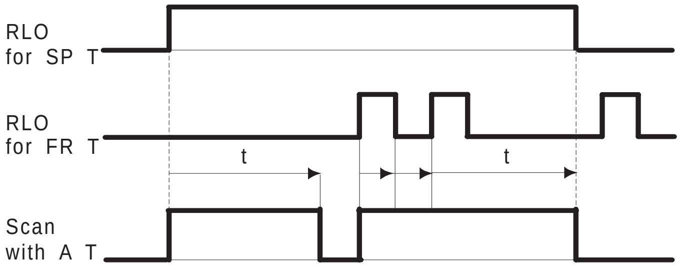
Table 3-15 Set and reset operations

Operation	Operand		Function
SU			Unconditional setting
	I	0.0 to 127.7	of an input (PII)
	Q	0.0 to 127.7	of an output (PIQ)
	F	0.0 to 255.7	of a flag
	D	0.0 to 255.15	of a data word bit
	T	0.0 to 255.15	of a timer word bit
	C	0.0 to 255.15	of a counter word bit
	RI	0.0 to 255.15	of a bit in RI area
	RJ	0.0 to 255.15	of a bit in RJ area
	RS	60.0 to 63.15	of a bit in RS area
	RT	0.0 to 255.15	of a bit in RT area
RU			Unconditional resetting
	I	0.0 to 127.7	of an input (PII)
	Q	0.0 to 127.7	of an output (PIQ)
	F	0.0 to 255.7	of a flag
	D	0.0 to 255.15	of a data word bit
	T	0.0 to 255.15	of a timer word bit
	C	0.0 to 255.15	of a counter word bit
	RI	0.0 to 255.15	of a bit in RI area
	RJ	0.0 to 255.15	of a bit in RJ area
	RS	60.0 to 63.15	of a bit in RS area
	RT	0.0 to 255.15	of a bit in RT area

Timer and counter operations

Table 3-16 Timer and counter operations with formal operands

Operation	Operand	Function
SP =	□	Start timer specified by the formal operand as a pulse with the value stored in ACCU-1-L (parameter type T).
SD =	□	Start timer specified by the formal operand as ON delay with the value stored in ACCU-1-L (parameter type T).
SEC =	□	Start timer specified by the formal operand as extended pulse with the value stored in ACCU-1-L or set counter specified as formal operand with the counter value stored in ACCU-1-L (parameter type: T, C).
SSU =	□	Start timer specified by the formal operand as stored ON delay with the value stored in ACCU-1-L or increment a counter specified as formal operand (parameter type: T, C).
SFD =	□	Start timer specified by the formal operand as stored OFF delay with the value stored in ACCU-1-L or decrement a counter specified as formal operand (parameter type: D, C).
FR =	□	Enable formal operand (timer/counter) for cold restart (see FR T or FR R); (parameter type: T, C). Insert formal operand
FR	T 0 to 255	Enable timer for cold restart: The operation is **only** executed on the leading edge of the RLO (change from 0 to 1). The timer is restarted if the RLO is 1 at the time of the start operation. (See timing diagram below the table).
	C 0 to 255	Enable a counter for setting or resetting: The operation is executed **only** on the leading edge of the RLO (change from 0 to 1). The counter is only started if the RLO = 1 at the time of the start operation.



Examples

Function block call	Program in the function block	Program executed
a)		
:JU FB 203		
NAME :EXAMPLE1		
ANNA : I 10.3	:A =ANNA	:A I 10.3
BERT : T 17	:L KT 010.2	:L KT 010.2
JOHN : Q 18.4	:SSU =BERT	:SS T 17
	:U =BERT	:U T 17
	:= =JOHN	:= Q 18.4
b)		
:JU FB 204		
NAME :EXAMPLE2		
MAXI : I 10.5	:A =MAXI	:A I 10.5
IRMA : I 10.6	:SSU =DORA	:CU C 15
EVA : I 10.7	:A =IRMA	:A I 10.6
DORA : C 15	:SFD =DORA	:CD C 15
EMMA : F 58.3	:A =EVA	:A I 10.7
	:L KC 100	:L KC 100
	:SEC =DORA	:S C 15
	:AN =DORA	:AN C 15
	:= =EMMA	:= F 58.3
c)		
:JU FB 205		
NAME :EXAMPLE3		
BILL : I 10.4	:A =BILL	:A I 10.4
JACK : T 18	:L =EGON	:L IW 20
EGON : IW 20	:SEC =JACK	:SE T 18
YOGI : F 100.7	:A =JACK	:A T 18
	:= =YOGI	:= F 100.7

Load and transfer operations

Table 3-17 Load and transfer operations with formal operands

Operation	Operand	Function
L =	□	Load a formal operand: The value of the operand specified as a formal operand is loaded into the ACCU (parameter type: I, T, C, Q; data type: BY, W, D).
LCD =	□	Load a formal operand in BCD code: The value of the timer or counter specified as a formal operand is loaded into the ACCU in BCD code (parameter type: T, C).
LW =	□	Load the bit pattern of a formal operand: The bit pattern of a formal operand is loaded into the ACCU (parameter type: D; data type: KF, KH, KM, KY, KS, KT, KC).
LWD =	□	Load the bit pattern of a formal operand: The bit pattern of a formal operand is loaded into the ACCU (parameter type: D; data type: KG).
T =	□	Transfer to a formal operand: The contents of the accumulator are transferred to the operand specified as a formal operand (parameter type: I, Q; data type: BY, W, D).
		Insert formal operand

Actual operands permitted include those of the corresponding basic operations **except for S flags**. For the "LW=" operation, permissible data types include a binary pattern (KM) or a hexadecimal pattern (KH), two absolute numbers of 1 byte each (KY), a character (KS), a fixed point number (KF), a timer value (KT) and a counter value (KC). For "LWD=" permissible data is a floating point number.

Table 3-18 Load and transfer operations with special operands

Operation		Operand		Function
	L	RI	0 to 255	Load a word from the interface data area into ACCU 1 (RI area)
		RJ	0 to 255	Load a word from the extended interface area into ACCU 1 (RJ area)
	L	RS	0 to 255	Load a word from the system data area into ACCU 1 (RS area)
		RT	0 to 255	Load a word from the extended system data area into ACCU 1 (RT area)
	T	RI	0 to 255	Transfer the contents of ACCU 1 to a word in the interface data area (RI area)
		RJ	0 to 255	Transfer the contents of ACCU 1 to a word in the extended interface data area (RJ area)
	T	RS	60 to 63	Transfer the contents of ACCU 1 to a word in the system data area (RS area)
		RT	0 to 255	Transfer the contents of ACCU 1 to a word in the extended system data area (RT area)

In contrast to the RI, RJ and RT areas, you can only use words RS 60 to RS 63 of the RS area. Refer to Section 8.3.4 "RS/RT Area".

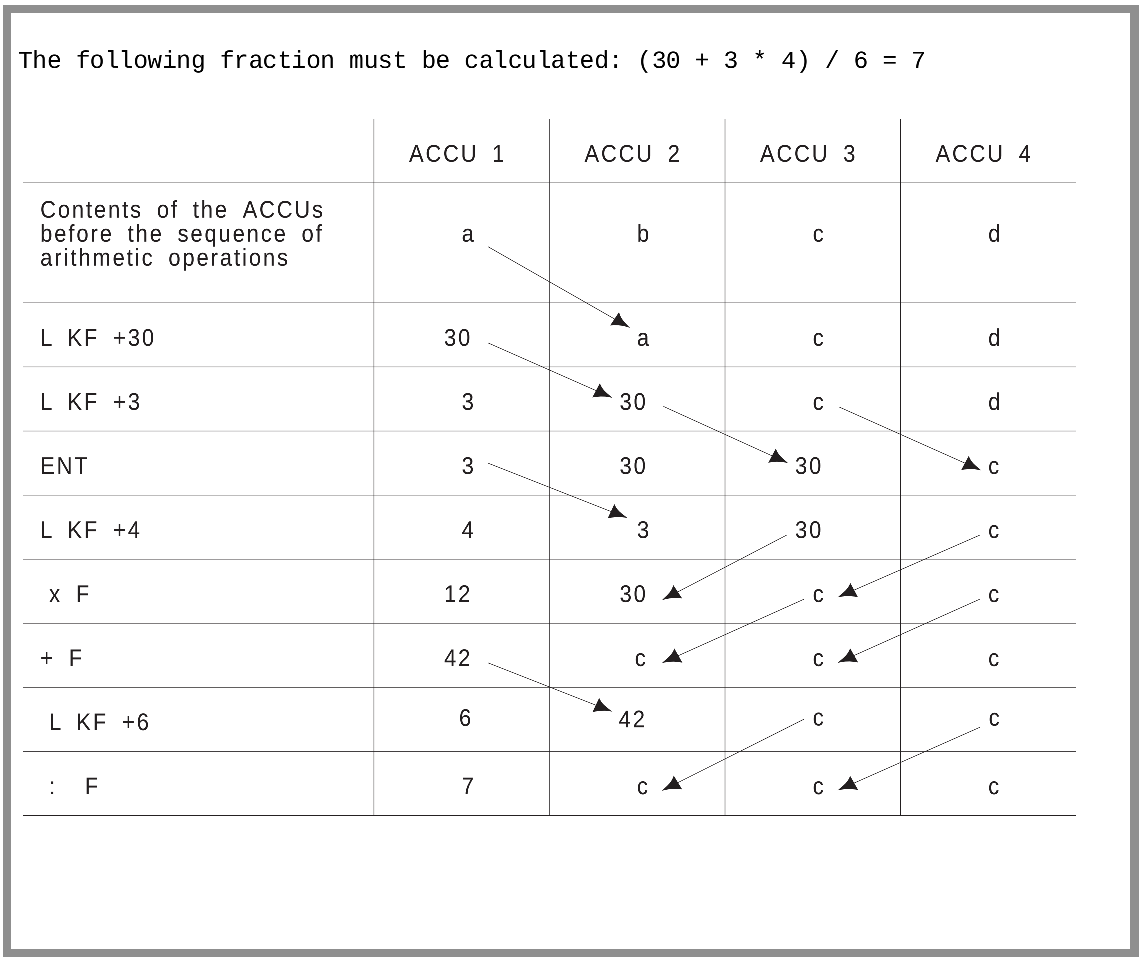
You can use the RT area in its complete length (RT 0 to RT 255) providing you do not use any standard function blocks.

Arithmetic operations

Table 3-19 Arithmetic operation ENT

Operation	Operand	Function
ENT	–	This causes a stack lift into ACCUs 3 and 4: <ACCU 4> := <ACCU 3> <ACCU 3> := <ACCU 2> <ACCU 2> := <ACCU 2> <ACCU 1> := <ACCU 1> ACCUs 1 and 2 are not changed. The old contents of ACCU 4 are lost.

Example



The following fraction must be calculated: (30 + 3 * 4) / 6 = 7

	ACCU 1	ACCU 2	ACCU 3	ACCU 4
Contents of the ACCUs before the sequence of arithmetic operations	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
x F	12	30	c	c
+ F	42	c	c	c
L KF +6	6	42	c	c
: F	7	c	c	c

Table 3-20 Supplementary arithmetic operations

Operation		**Operand**		**Function**
S	ADD	BN	-128 to +127	Add a byte constant (fixed point) to ACCU-1-L (includes sign change)/the condition code in CC 0, CC 1, OV and OS are not affected! – ACCU-1-H and ACCUs 2 to 4 remain unchanged.
S	ADD	KF	-32 768 to +32 767	Add a fixed point constant (word) to ACCU-1-L/ the condition codes in CC 0, CC 1, OV and OS are not affected! – ACCU-1-H and ACCUs 2 to 4 remain unchanged.
S	ADD [1]	DH	0000 0000 to FFFF FFFF	Add a double word fixed point constant to ACCU 1/the condition codes in CC 0, CC 1, OV and OS are not affected! – ACCUs 2 to 4 remain unchanged.
S	+D [1]			Add two double word fixed point constants (ACCU 2 + ACCU 1)/the result can be evaluated in CC 0/CC 1. [2]
S	-D [1]			Subtract two double word fixed point constants (ACCU 2 - ACCU 1)/the result can be evaluated in CC 0/CC 1. [2]
S	TAK			Swap the contents of ACCU 1 and ACCU 2

[1] Programming is dependent on the PG type and the release of the PG system software.

[2] For changes in ACCU 2 and ACCU 3: see Section 3.5.1 "Basic Operations/Arithmetic Operations".

3.5.4 Executive Operations

The executive operations also include system operations.

Caution
System operations should only be used with great care and then only by experienced programmers familiar with the system.

System operations are indicated in the table by S

Jump operations

When you use the supplementary jump operations, you indicate the jump destination for unconditional jumps symbolically. The symbolic parameter of the jump operation is identical to the symbolic address of the destination statement. When programming, remember that the absolute jump distance should not exceed ± 127 words and a STEP 5 statement can consist of more than one word. You can only execute these jumps within a block; jumps over segment boundaries are not permitted ("segment" = structural element in PBs, SBs, FBs, FXs and OBs; see PG description).

Note
The jump statement and jump destination (symbolic address) must be in the same segment. A symbolic address can only be used **once** per segment.
Exception: this does not apply to the JUR jump for which you specify an absolute jump distance as the parameter.

Table 3-21 Jump operations

Operation		Operand	Function
	JU =	addr (addr =symbolic address with maximum 4 characters)	Jump unconditionally: The jump is executed regardless of conditions
	JC =		Jump conditionally: the conditional jump is executed only if the RLO is 1. If the RLO is 0, the statement is not executed and the RLO is set to 1.
	JZ =		Jump if result is ’0’ : the jump is executed only if CC 1 is 0 and CC 0 is 0. The RLO is not changed.

	Operation	Operand	Function
Table 3-21 continued:			
	JN =	addr (addr = symbolic address with maximum 4 characters)	Jump if result is not 0 : the jump is executed only if CC1 is not equal to CC0. The RLO is not changed.
	JP =		Jump if result > ’0’ : the jump is only executed if CC 1 = 1 and CC 0 = O. The RLO is not changed.
	JM =		Jump if result < ’0’: the jump is only executed if CC 1 = 0 and CC 0 = 1. The RLO is not changed.
	JO =		Jump on overflow: the jump is executed when the OV condition code is 1. If there is no overflow (OV is 0), the jump is not executed. The RLO is not changed. An overflow occurs when an arithmetic operation exceeds the permissible range for a given numerical representation.
	JOS =		Jump when the OS (stored overflow) condition code is set: the jump is executed when the condition code OS is 1. If there is no overflow (OS is 0), the jump is not executed. The RLO is not changed. An overflow occurs when an arithmetic operation exceeds the permissible range for a given numerical representation.
S	JUR	-32 768 to +32 767	Relative jump within the user memory or within a function block (e.g. to arrive in a different segment). The operation is always executed regardless of conditions. The operand is the number of words difference between the address of the jump destination **-** the current destination. The jump is executed either to a higher (positive operand) or lower (negative operand) address than the current operation.

⚠

Caution

If you use **JUR** incorrectly, undefined statuses can occur in the system. It should only be used by extremely experienced programmers with detailed knowledge of the system.

Shift operations

Table 3-22 Shift operations

Operation	Operand	Function (operation with ACCU 1)
SLW	0 to 15	Shift a word to the left (vacant positions to the right are padded with zeros)
SRW	0 to 15	Shift a word to the right (vacant position to the left are padded with zeros)
SLD	0 to 32	Shift a double word to the left (vacant positions to the right are padded with zeros)
SSW	0 to 15	Shift a word with sign to the right (vacant positions to the left are padded with the sign - bit 15)
SSD	0 to 32	Shift a double word with sign to the right (vacant positions to the left are padded with the sign - bit 31)
RLD	0 to 32	Rotate to the left
RRD	0 to 32	Rotate to the right

Only ACCU 1 is involved in the execution of shift operations. The parameter part of these operations specifies the number of positions by which the accumulator contents should be shifted or rotated. For the SLW, SRW and SSW operations, only the low word of ACCU 1 is involved in the shift operations. For SLD, SSD, RLD and RRD operations, the entire contents of ACCU 1 (32 bits) are involved.

Shift operations are executed regardless of conditions.

You can use jump operations to scan the value of the last bits shifted out using CC 1/CC 0.

Shift: last bit shifted	CC 1	CC 0	Jump operation
0	0	0	JZ=
1	1	0	JN= JP=

Examples

1. You want to shift the contents of data word DW 52 four bits to the left and write them to data word DW 53.

```
STEP 5 program:   Contents of the data words:

:L   DW 52        KH = 14AF
:SLW 4
:T   DW 53        KH = 4AF0
```

2. You want to read the input double word ID 0, and shift the contents of ACCU 1 so that the bit positions of the input double word shown in bold face are retained and the remaining bit positions are set to defined values (0H or 0FH).

```
STEP 5 program:   Contents of ACCU 1 (hexadecimal)

                  ACCU-1-H:            ACCU-1-L:

:L   ID 0         2348                 ABCD
:SLW    4         2348                 BCD0
:SRW    4         2348                 0BCD
:SLD    4         3480                 BCD0
:SSW    4         3480                 FBCD
:SSD    4         0348                 0FBC
:RLD    4         3480                 FBC0
:RRD    4         0348                 0FBC
```

3. Application: Multiplication by the 3rd power, e.g. new value = old value x 8

```
:L   FW 10
:SLW     3
:T   FW 10        Caution: do not exceed the
                           positive area limit!
```

4. Application: Division by the 2nd power, e.g. new value = old value : 4

```
:C   DB 5
:L   DW 0
:SRW    2
:T   DW 0
```

Conversion operations

Table 3-23 Conversion operations

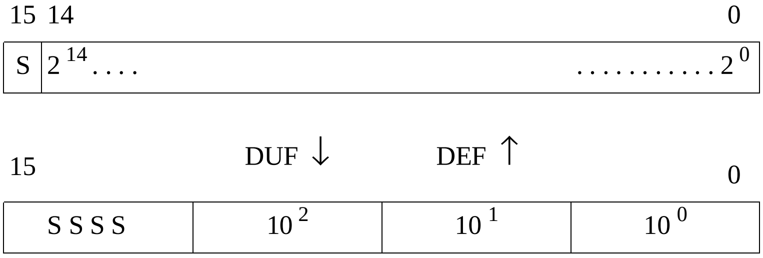
Operation	Function
CFW	Form the 1’s complement of ACCU-1-L (16 bits)
CSW	Form the 2’s complement of ACCU-1-L (16 bits)
CSD	Form the 2’s complement of ACCU 1 (32 bits)
DEF	Convert a fixed point number (16 bits) from BCD to binary
DUF	Convert a fixed point number (16 bits) from binary to BCD
DED	Convert a double word (32 bits) from BCD to binary
DUD	Convert a double word (32 bits) from binary to BCD
FDG	Convert a fixed point number (32 bits) to a floating point number (32 bits)
GFD	Convert a floating point number to a fixed point number (32 bits)

DEF

The value in ACCU-1-L (bits 0 to 15) is interpreted as a BCD number. After the conversion, ACCU-1-L contains a 16-bit fixed point number.

DUF

The value in ACCU-1-L (bits 0 to 15) is interpreted as a 16-bit fixed point number. After the conversion, ACCU-1-L contains a BCD number.



S (sign): 0 = positive
1 = negative

DED

The value in ACCU 1 (bits 0 to 31) is interpreted as a BCD number. After the conversion, ACCU 1 contains a 32-bit fixed point number.

DUD

The value in ACCU 1 (bits 0 to 31) is interpreted as a 32-bit fixed point number. After the conversion, ACCU 1 contains a BCD number.

31	30	0
S	2^{30}	. . . . . 2^{0}

DUD ↓ DED ↑

31							0
S S S S	10^{6}	10^{5}	10^{4}	10^{3}	10^{2}	10^{1}	10^{0}

S (sign):	0 = positive 1 = negative

FDG

The value in ACCU 1 (bits 0 to 31) is interpreted as a 32-bit fixed point number. After the conversion, ACCU 1 contains a floating point number (exponent and mantissa).

GFD

The value in ACCU 1 (bits 0 to 31) is interpreted as a floating point number. After the conversion, ACCU 1 contains a 32-bit fixed point number.

31	30	0
S	2^{30}	. . . . . 2^{0}

FDG ↓ GFD ↑

31	30 24	23	0
S	2^{6} . . . . 2^{0}	S	2^{-1} . . . . 2^{-23}
Exponent		Mantissa	

The conversion is made by multiplying the (binary) mantissa by the value of the (binary) exponent by shifting the mantissa value to more significant bits past an imaginary decimal point by the value of the exponent (base 2). After the multiplication, remnants of the original mantissa remain to the right of the imaginary decimal point. These bit places are cut off from the whole result.

This conversion algorithm produces the following result classes:

- Floating point numbers ≥ **0 or** ≤ **-1** result in the **next lower number**.
- Floating point numbers < **0 and** > **-1** result in the **value ’0’**.

Conversion examples

```
 Floating point number   32-bit fixed point number
                 GFD

    +5,7           →         5
    -2,3           →        -3
    -0,6           →         0
    +0,9           →         0
```

Examples of CFW, CSW

```
1. You want the contents of data word DW 64
   inverted bit for bit (reversed) and stored in
   data word DW 78.

   STEP 5 program:     Assignment of the data words:

   :L DW 64            KM = 0011111001011011
   :CFW
   :T DW 78            KM = 1100000110100100



2. The contents of data word DW 207 are
   interpreted as a fixed point number and stored
   in data word 51 with a reversed sign.

   STEP 5 program:     Assignment of the data words:

   :L DW 207           KF = +51
   :CSW
   :T DW 51            KF = -51
```

Decrement/ increment

Table 3-24 Decrement/increment operation

Operation	Operand	Function
D	1 to 255	Decrement the low byte (bits 0 to 7) of ACCU-1-L by the value of the operand [1]
I	1 to 255	Increment the low byte (bits 0 to 7) of ACCU-1-L by the value of the operand [1]

[1] The contents of the low byte of ACCU-1-L are decremented or incremented by the number specified as the operand without a carry. The operation is executed regardless of conditions.

Example

```
STEP 5 program:    Assignment of the data words:

   :L DW 7         KH = 1010
   :I   16
   :T DW 8         KH = 1020
   :D   33
   :T DW 9         KH = 10FF
```

Processing operations

Table 3-25 Processing operations

Operation	Operand	Function
DO	DW 0 to 255	Process data word: the following operation is combined with the parameter specified in the address data word and executed.
	FW 0 to 254	Process flag word: the following operation is combined with the parameter specified in the addressed F flag and executed.
DO =	□	Process formal operand (parameter type B): Only C DB, JU PB, JU OB, JU FB, JU SB can be substituted. Insert formal operand

Operation		Operand	Function
Table 3-25 continued:			
S	BI	[1]	Indirect processing of a formal operand: execute an operation whose operation code is stored in a formal operand. The number of the formal operand must be stored in ACCU 1.
	B	RS 60 to 63 [1]	Execute an operation whose operation code is stored in the system data area (RS = free system data: RS 60 to 63). In 2-word operations the 2nd word must be loaded in RS n + 1.

[1] The value in the formal operand or system data is interpreted as the operation code of a STEP 5 operation and is then executed.

Note
Only the following operations can be combined with **DO DW**, or **DO FW**, **DI** or **DO RS**:

- A.. , AN.. , O.. , ON.. , S.. , R.. , =..
  with areas I, Q, F, S,
- FR T, R T, SF T, SD T, SP T, SS T, SE T,
- FR C, R C, S C, CD C, CU C,
- L.., T.. with areas P, O, I, Q, F, S, D, RI, RJ, RS, RT,
- L T, L C,
- LC T, LC C,
- JU=, JC=, JZ=, JN=, JP=, JM=, JO=,
- SLW, SRW,
- D, I, SED, SEE,
- C DB, JU.. , JC.., G DB, GX DX, CX DX, DOC FX, DOU FX.

The PG does not check the legality of the combinations!

Examples of DO operations

DO DW/DO FW

Operand substitution

Using the statements "DO DW" and "DO FW" you can access data with a substitution, e.g. in a program loop. The substituted access consists of the statement DO DW/DO FW followed immediately by one of the STEP 5 operations listed above.
"Substituted" means that the operand for the operation is not programmed as a static value but is fixed during the course of the STEP 5 program.

Select the operand **type** from the range permitted for the operation when you write your program, e.g. **PB** for the operation "JU PB nn":

You must first load the operand **value** (**nn** in the example) in a data word or F flag word (parameter word) before the substituted access with DO DW/DO FW.

1. **Principle of substitution:**

```
          :L   KF +120
          :T   FW 14      load FW with the value "KF  +120"
          :DO  FW 14
          :L   IB 0
               |
               └─────────→ before the operation "L IB" is executed, the
                           operand value ’0’ is replaced by the value ’120’;
                           Operation executed: L IB 120
```

2. **Data word as index register:**

The contents of data words DW 20 to DW 100 are set to signal state ’0’. The index register for the parameter of the data words is DW 1.

```
          :L   KF +20     supply the index register
          :T   DW 1
  M001    :L   KF +0      reset
          :DO  DW 1
          :T   DW 0
          :L   DW 1       increment the index register
          :L   KF +1
          :+F
          :T   DW 1
          :L   KF +100
          :<=F
          :JC  =M001      jump if the index is within the range
          ...             remaining STEP 5 program
```

Continued on next page

Examples of operand substitution continued:

3. **Jump distributor for subroutine techniques:**

```
          :DO     FW 5
          :JU     =M001   Contents of flag word FW 5:
+         :JU     =M002
Jump      :JU     =M003          jump distance
distance →:JU     =M004         (maximum ±127)
          :JU     =M005
          : .
          : .
  M001    : .
          : .
          :BEU
  M002    : .             Advantage:
          : .             all program sections are
          :BEU            contained in one block.
  M003    : .
          : .
          :BEU
```

4. **Jump distributor for block calls:**

```
          :DO   FW 10                  Contents of flag word FW 10:
          :JU   PB 0 ──→PB 0
                     ──→PB 1            Block no. x
                     ──→PB 2
                     ──→PB 3
                     ──→.
                     ──→.
                     ──→PB x
```

Operand substitution with binary operations

For operand substitutions with binary operations you can use the following operand types: inputs, outputs, F flags, S flags, timers and counters.
In this substitution, the structure of the F flag word or data word (parameter word) depends on the operation you are using.

Parameter word for inputs and outputs

Bit no.	15 11	10 8	7	6 0
	no significance	Bit address from 0 to 7	0	Byte address from 0 to 127

Parameter word for F flags

Bit no.	15 11	10 8	7 0
	no significance	Bit address from 0 to 7	Byte address from 0 to 255

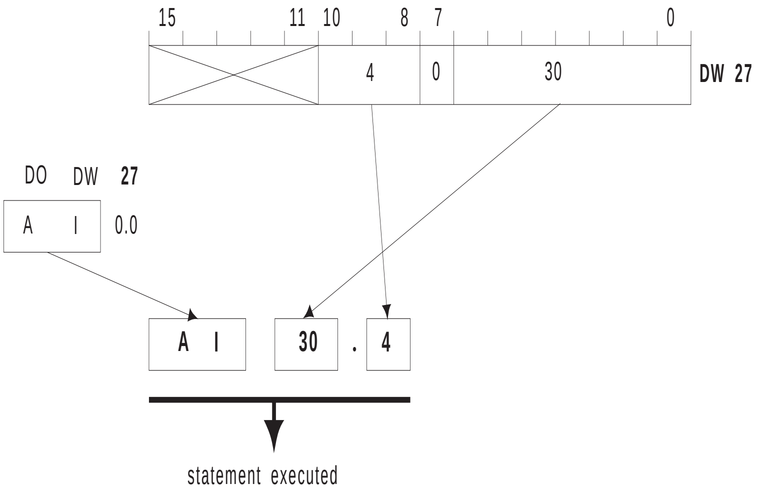
Parameter word for S flags

Bit no.	15	14 12	11 0
	0	Bit address from 0 to 7	Byte address from 0 to 4095

Parameter word for timers and counters

Bit no.	15 8	7 0
	no significance	Number of timer or counter cell from 0 to 255

Principle of the substitution with a binary operation



Example of DI operation

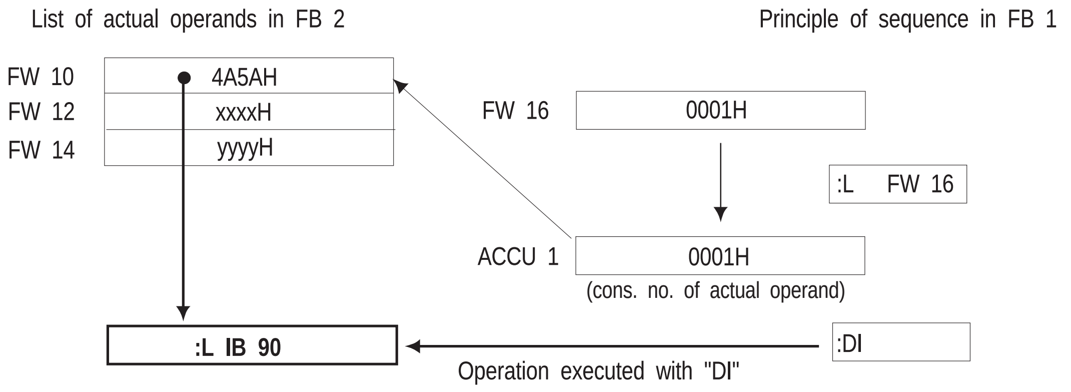
In function block FB 1, STEP 5 operations are executed whose operation codes were transferred
by a calling block as formal operands FW 10, FW 12 and FW 14.
Which of the operation codes is executed is written by the calling block as a consecutive number in flag word **FW 16**.
The result of the executed operation is then entered in ACCU 1 and is transferred to flag word FW 18.

FB 1

```
NAME :TEST

DECL :FW10     I/Q/D/B/T/C: D  KM/KH/KY/KS/KF/KT/KC/KG:  KH
DECL :FW12     I/Q/D/B/T/C: D  KM/KH/KY/KS/KF/KT/KC/KG:  KH
DECL :FW14     I/Q/D/B/T/C: D  KM/KH/KY/KS/KF/KT/KC/KG:  KH

     :L   FW 16           cons. number of formal operand
     :                    with required operation code
     :DI                  transferred operation code is executed
     :T   FW 16           result from ACCU 1
     :BE
```

FB 2

```
     :
     :L   KF +1
     :T   FW 16           cons. no. of formal operand with operation code
     :JU  =AUFR
     :
     :
AUFR :
     :JU  FB 1            call FB TEST
NAME :TEST
FW10 :    KH 4A5A         op. code "L IB 90",             formal operand 1
FW12 :    KH xxxx         other operation code,           formal operand 2
FW14 :    KH yyyy         other operation code,           formal operand 3
     :T   FW 18           ACCU 1 → FW 18
     :BE
```


List of actual operands in FB 2
Principle of sequence in FB 1
FW 10
4A5AH
FW 12
xxxxH
FW 14
yyyyH
FW 16
0001H
:L FW 16
ACCU 1
0001H
(cons. no. of actual operand)
:L IB 90
:DI
Operation executed with "DI"

I/O operations

Table 3-26 I/O operations

Operation		Operand	Function
	IA		Disable process interrupt servicing (via IB 0) (not for system interrupts!)
	RA		Enable process interrupt servicing (via IB 0) (not for system interrupts!)
	IAE		Disable addressing error
	RAE		Enable addressing error
	BAS		Disable command output: PIQ is no longer influenced by the operations S Q, R Q, = Q, T PY and T PW.
	BAF		Enable command output

"Enable/disable process interrupts" can, for example, be used when process interrupt driven processing is to be suppressed during time-controlled processing. In the program section between the IA and RA statements, no process interrupt driven processing is possible. Note the special function OB 122 "disable interrupts", Section 6.3.

Other operations

Table 3-27 Other operations

Operation		Operand	Function
S	STS		Stop command leading directly to a soft STOP.
	STW		Stop command leading directly to a hard STOP. (state can only be exited with POWER DOWN/UP).
	SIM		Set interrupt mask (UAMW) (32 bits): before calling the operation, the bit pattern for the mask must be loaded in ACCU 1 (32 bits)
	LIM		Read interrupt mask: bit pattern of the interrupt mask (32 bits) is loaded in ACCU 1

SIM/LIM – set/read interrupt condition code mask (UAMW)

The interrupt mask "masks" interrupts in the interrupt condition code word until the end of the cycle, i.e. all interrupts remain pending, but the program is not interrupted by them.

Bit in the interrupt condition code mask = 0: interrupt disabled
Bit in the interrupt condition code mask = 1: interrupt enabled

Meaning of the bits in UAMW-H or ACCU-1-H:

15							8	7							0
INTX	INTE	INTF	INTG	WEFE	WA	PA	BULE	PEU	HALT	ES	AV	INTAS	TAU	DARY	KZU

Meaning of the bits in UAMW-L or ACCU-1-L:

15							8	7							0
–	KB	KDB	STS	TLAF	SUF	STUEB	STUEU	NAU	ZA	QVZ	ADF	PARE	ZYK	STOP	HOLD

Table 3-28 Meaning of the abbreviations in UAMW

Abbrev.	**Meaning**
High word	
INTX	S5 bus/system interrupt A, B, C or D (slot-dependent)
INTE	S5 bus/system interrupt E
INTF	S5 bus/system interrupt F
INTG	S5 bus/system interrupt G
WEFE	Collision of timed interrupts
WA	Timed interrupt
PA	Process interrupt
BULE	Bus lock error
PEU	I/Os not ready
HALT	Stop instruction from coordinator COR
ES	Single step mode
AV	Address comparison active
INTAS	Interrupt from SPU processor
TAU	Clock failure of SPU processor
DARY	Continuous ready (access to faulty memory)
KZU	Bracket counter overflow

Abbrev.	**Meaning**
Table 3-28 continued:	
Low word	
KB	No block
KDB	No data block
STS	Soft stop
TLAF	Transfer/load error
SUF	Substitution error
STUEB	BSTACK overflow
STUEU	ISTACK overflow
NAU	Power failure
ZA	Timed interrupt (delayed interrupt, clock-controlled interrupt
QVZ	Timeout
ADF	Addressing error
PARE	Parity error
ZYK	Cycle time error
STOP	Mode selector switched to STOP
HOLD	DMA request from SPU processor

3.5.5 Semaphore Operations

If two or more CPUs in one programmable controller (see Chapter 10) require access to the same global memory area (peripherals, CPs, IPs), there is a danger that one CPU will overwrite the data of another CPU or that one CPU could read invalid intermediate data statuses of another CPU and misinterpret them. You must therefore coordinate CPU accesses to the common memory areas.

You can coordinate the individual CPUs using the SED and SEE operations.
You can, for example, program the following coordination between two CPUs: a CPU involved in multiprocessing can only access the common memory area after it has successfully set a declared semaphore (SES). A semaphore xx can only be set by a single CPU. If a CPU fails to set (i.e. disable) the semaphore, it cannot access the memory area. In the same way, a CPU can no longer access the memory once it has released the semaphore again (SEE).

SED/SEE disable/enable semaphore

(non-system operations)

Table 3-29 Disable/enable semaphore

Operation	Operand	Function
SED	0 to 31	Disable (set) a semaphore
SEE	0 to 31	Enable (release) a semaphore
		evaluation of the result of the operation via CC 0/CC 1

Note
The SED xx and SEE xx operations must be programmed **in all CPUs** that require synchronized access to a **common** global memory area.

Standard FBs, handling blocks and blocks for multiprocessor communication manage the coordination internally. If you use these blocks, you do not need to program the operations SEE xx and SED xx.

Effect of SED/SEE

The CPU that executes the operation SED xx (disable semaphore) accesses a specific byte in the coordinator (**provided** that no other CPU has access to that byte already). Once a CPU has reserved access, the other CPUs can no longer access the memory area protected by the semaphore (numbers 0 to 31). The area is therefore disabled for all other CPUs.
Make sure that the coordination functions correctly, all CPUs requiring access to the same area of global memory must use the same semaphore.

The SEE xx (enable semaphore) operation resets the byte on the coordinator. The protected memory area is then once again accessible to the other CPUs. A semaphore can only be enabled by the CPU that disabled it.

Use of SED/SEE

Fig. 3-8 illustrates the basic sequence of coordinated access using a semaphore.

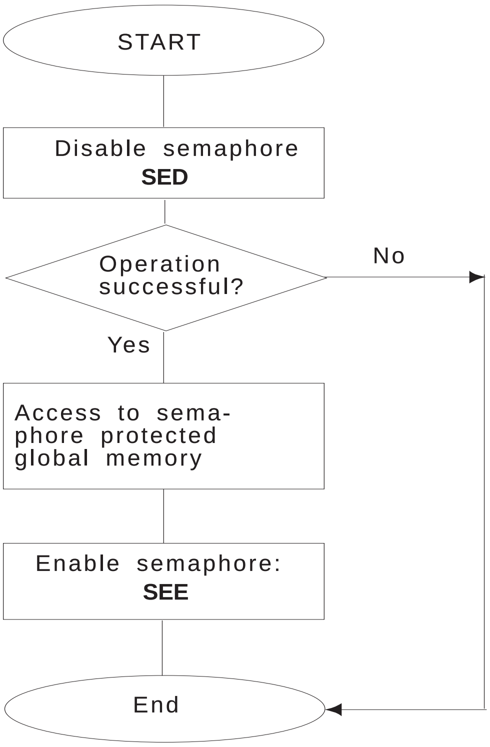


Fig. 3-8 Coordination of access to the global memory

Before disabling or enabling a particular semaphore, the SED and SEE operations scan the status of the semaphore. The condition codes CC 0 and CC 1 are affected as follows:

CC 1	CC 0	Evaluation	Significance
0	0	JZ	Semaphore was disabled by another CPU and cannot be disabled/enabled.
1	0	JN, JP	Semaphore was disabled/ enabled.

Note
The scanning of a particular semaphore (= read procedure) and the disabling or enabling of the semaphore (=write procedure) are **one unit**. No other CPU can access the semaphore during these procedures!

When using semaphores, remember the following points:

- A semaphore is a global variable, i.e. the semaphore with number 16 exists only **once** in the entire system, even if your controller is using three CPUs.
- **All** CPUs that require coordinated access to a common memory area must use the SED and SEE operations.
- All participating CPUs must execute the **same** start-up type. During a COLD RESTART, all the semaphores are cleared. During a manual or automatic warm restart, the semaphores are retained.
- Start-up in multiprocessor operation must be synchronized. For this reason, **no** test operation is allowed.

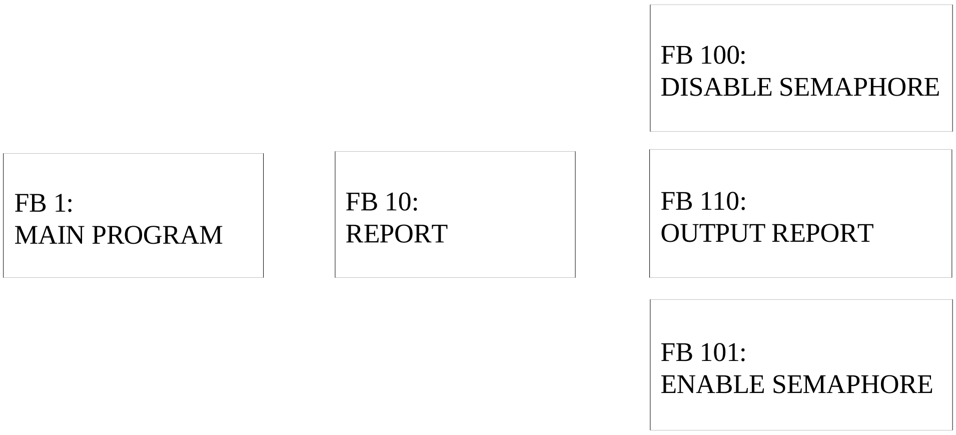
Application example for semaphores

Tasks:

Four CPUs are plugged into an S5-155U. They output status messages to a status signalling device via a common memory area of the O peripherals (OW 6). A CPU must output each status message for 10 seconds. Only after a 10 second output can a new message be output from the same CPU or a different CPU overwrite the first message. The use of peripheral word OW 6 (extended I/O area, no process image) is controlled by a semaphore. Only the CPU that was able to reserve this area for itself by disabling the assigned semaphore can write this message to OW 6. The semaphore remains disabled for 10 seconds at a time (TIMER T 10). The CPU re-enables the semaphore only after this timer has elapsed. After the semaphore has been re-enabled, the other CPUs can access the reserved area. The new message can then be written to OW 6.

Implementation:

The following program can run in all four CPUs, each with a different message. The blocks shown below are loaded.



5 flags are used as follows:

```
F 10.0 = 1:  a message was requested or is being processed

F 10.1 = 1:  the semaphore was disabled successfully

F 10.2 = 1:  the timer was started

F 10.3 = 1:  the message was transmitted

F 10.4 = 1:  the semaphore was re-enabled
```

Continued on next page

Semaphore application example continued:

FB 1

```
     :A   F 10.0
     :JC  =M001         If no message is active,
     :
     :AN  I  0.0
     :BEC
     :
     :L   KH 2222       generate message and
     :T   FW 12
     :AN  F 10.0
     :S   F 10.0        set "MESSAGE" flag.
     :
M001 :JU  FB10          Call "REPORT" FB
NAME :REPORT
     :
     :BE
```

FB 10

```
NAME :REPORT

     :AN  F 10.1        If no semaphore is disabled,
     :JC  FB 100        call "disable semaphore" FB.
NAME :SEMADIS
     :
     :A   F 10.1        If the semaphore is disabled
     :AN  F 10.2        and the timer has not started,
     :S   F 10.2
     :L   KT010.2       start the timer.
     :SE  T 10
     :
     :A   F 10.2        If the timer has started
     :AN  F 10.3        and no message is being transmitted,
     :JC  FB 110        call "output message" FB.
NAME :MSGOUT
     :
     :A   F 10.2        If the timer has started
     :AN  F 10.4        and the semaphore is not enabled
     :AN  T 10          and the timer has elapsed,
     :JC  FB 101        call "enable semaphore" FB.
NAME :SEMAENAB
     :
     :AN  F  10.4       If the semaphore is enabled,
     :BEC
     :
     :L   KH0000
     :T   FY10          reset all flags.
     :BE
```

Continued on next page

Semaphore application example continued:

FB 100

```
NAME :SEMADIS

     :SED 10           Disable semaphore no. 10
     :JZ  =M001
     :AN  F 10.1       If the semaphore is disabled successfully,
     :S   F 10.1       set "SEMAPHORE-DISABLED" flag.
M001 :BE
```

FB 110

```
NAME:MSGOUT

     :L   FW12         Transmit a message
     :T   OW 6         to the peripherals
     :AN  F 10.3
     :S   F 10.3       Set "TRANSFER MESSAGE"
     :                 flag
     :BE
```

FB 101

```
NAME :SEMAENAB

     :SEE 10           Enable semaphore no. 10
     :JZ  =M001
     :AN  F 10.4
     :S   F 10.4       Set "SEMAPHORE ENABLED"
     :                 flag
M001 :BE
```

Operating Statuses and Program Execution Levels

4

Contents of Chapter 4

4

Operating Statuses and Program Execution Levels

This chapter provides an overview of the operating statuses and program execution levels of the CPU 948. It informs you in detail about various types of start-up and the organization blocks associated with them, in which you can program your own sequences for various situations when restarting.

You will also learn the characteristics of the program execution modes "cyclic processing", "time-controlled processing" and "interrupt-driven processing" and will see which blocks are available for your user program.

4.1 Program Execution Levels

Fig. 4-2 provides you with an overview of the program processing levels in the various modes. The explanations of the abbreviations are on the following page.

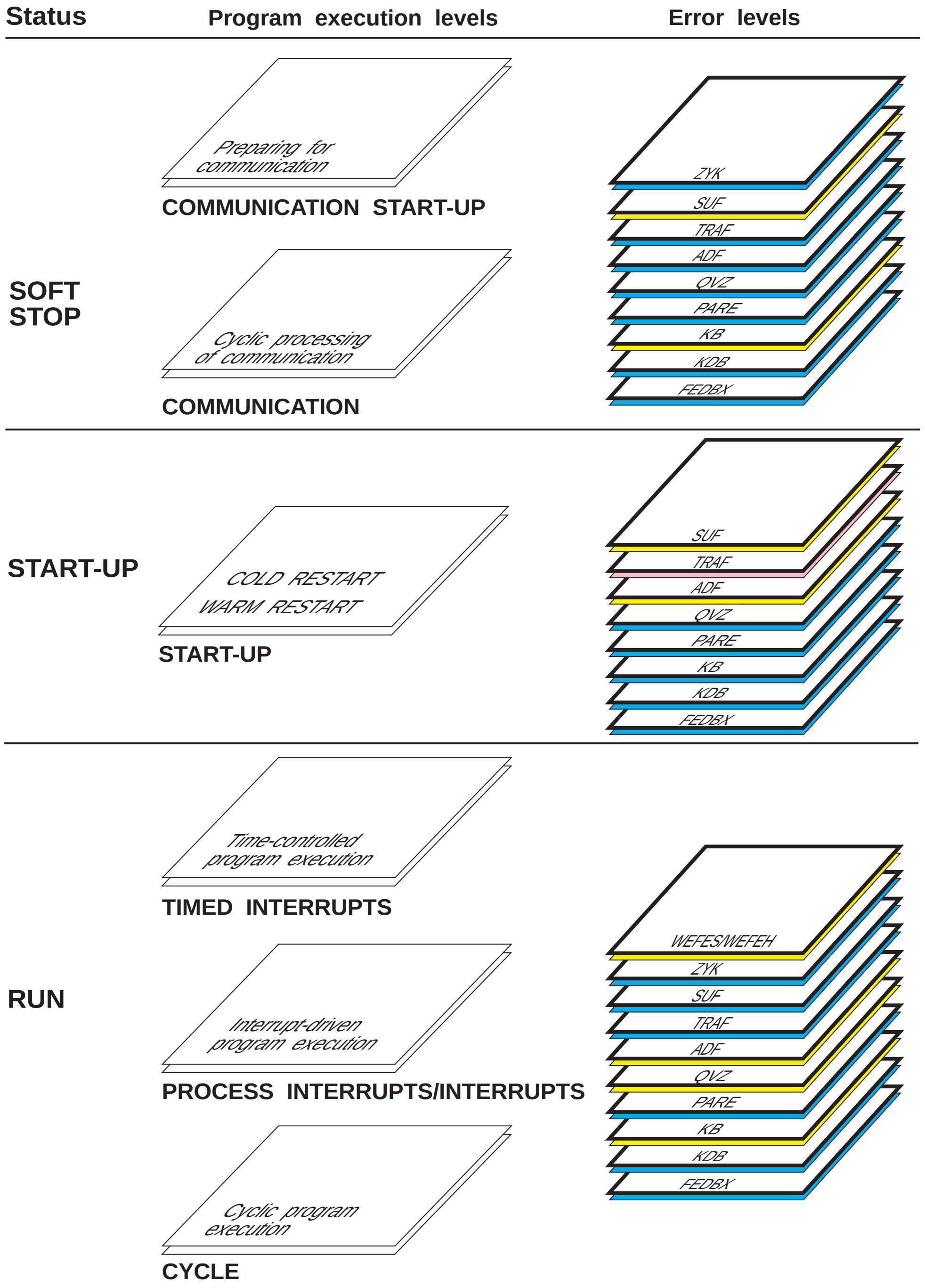


Fig. 4-1 Program execution levels

Table 4-1 Program execution levels

Level	Meaning	Priority
Error levels		
WEFES/WEFEH ZYK SUF TRAF ADF QVZ PARE KB KDB FEDBX	Collision of timed interrupts Cycle error Substitution error Transfer/load error Addressing error Timeout Parity error Called code block does not exist Called data block does not exist Error generating a data block, DB or DX	Each error handling routine has the highest priority. If an error occurs, the corresponding error level is nested in immediately.
Program execution levels in SOFT STOP		
COMMUNICATION START-UP COMMUNICATION	Preparation (start-up) for communication Cyclic processing of communication	–
Program processing levels in START-UP		
COLD RESTART WARM RESTART	Defined start of the user program Continuation of the user program at the point of interruption	–
Program execution levels in RUN		
TIMED INTERRUPTS PROCESS INTERRUPTS CYCLE	Time-controlled program execution Interrupt-driven program execution Cyclic program execution	Ascending priority (default) [1]

[1] The default can be changed by selecting parameters for DX 0 (refer to Chapter 7).

Processing via the system program

A specific system program is responsible for each level.

Interrupt stack (ISTACK)

If an interrupt occurs, the system program sets up an information field in the ISTACK for each level, to allow it to continue in the interrupted level after servicing the interrupt.

Nesting other levels

When an event occurs, which requires higher priority processing, the current level is interrupted by the system program and the higher priority level is activated.

This occurs in the following situations:

• at error levels:	always at operation boundaries,
• all other levels:	at block or operation boundaries (depending on the setting in DX 0 refer to Chapter 7)

Note
A maximum of 5 error organization blocks can be nested. If 5 error levels are activated simultaneously, this causes an ISTACK overflow and the CPU changes to the **HARD STOP**.

A specific **program execution level** is assigned to one or a group of organization blocks which are called by the system program after an event. If, for example, OB 9 is called to process a time-controlled interrupt, the program execution level TIMED INTERRUPTS is activated.

After the system program calls an organization block, the CPU executes the STEP 5 statements is contains. The current register record is saved in the ISTACK and a **new register record** is set up (register: ACCU 1 to 4, block stack pointer, block address register, data block start address, data block length, step address counter, base address register and the interrupt condition code words ICMK and ICRW).
If "normal" program execution is interrupted by the occurrence of an event, following the execution of the OB, the CPU continues the program execution at the point of interruption (including all the blocks nested there) as long as no stop is programmed in the OB.

Sub-levels

The TIMED INTERRUPTS level contains several sub-levels to which a specific program (OB) is assigned. Within the TIMED INTERRUPTS level, the sub-levels have their own priority (refer to the following table).

TIMED INTERRUPTS level	
Sub-level	**Priority**
Delayed interrupt cyclic timed interrupt, shortest period ... ... cyclic timed interrupt, longest period time-driven interrupt	ascending priority (default) ↑

Examples

Example of
"Execution by the system program":

At the CYCLE program execution level, the system program updates the process image of the inputs and outputs, triggers the cycle monitoring time and calls the PG interface management (system checkpoint).

Example of
"Interrupt stack":

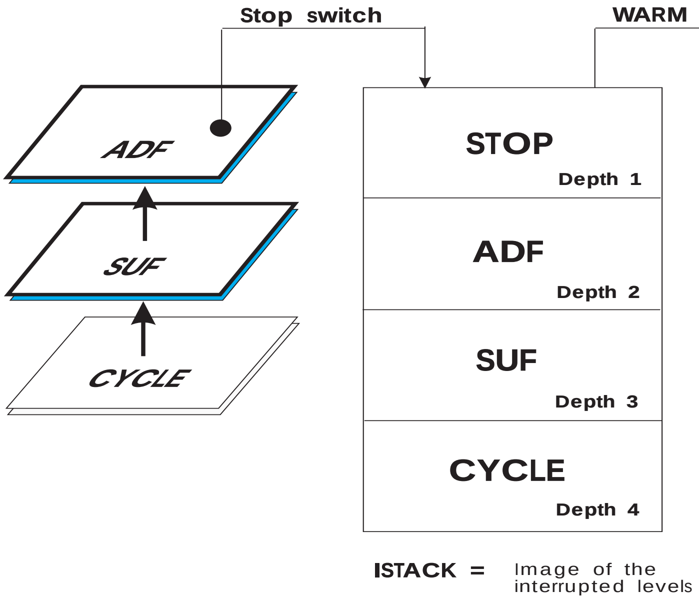


Fig. 4-2 Principle of changing level and the ISTACK

Example of
"Interrupting a basic level with interruptability at block boundaries":

A timed interrupt occurs while a process interrupt is being serviced. Since the timed interrupt has a higher priority, the servicing of the PROCESS INTERRUPT is interrupted at the next block boundary and the TIMED INTERRUPT level nested in. If an addressing error now occurs while servicing the timed interrrupt, the timed interrupt servicing is interrupted immediately at the next operation boundary to nest in the ADF level.

4.2 STOP Mode

The CPU 948 has two different STOP modes, the "hard" STOP and the "soft" STOP (= CPU capable of communication).

4.2.1 SOFT STOP

The SOFT STOP mode has the following features:

The CPU can communicate: the system program calls organization block OB 38 once after POWER UP (COMMUNICATION START-UP level in SOFT STOP) and then calls OB 39 (COMMUNICATION level in SOFT STOP).

Initialization of communication (OB 38)

To initialize communication, the system program calls **OB 38** as the user interface.

> OB 38 is only called after **POWER UP** . The call is not dependent on the type of start-up (AUTOMATIC COLD /AUTOMATIC WARM RESTART) set in data block DX 0.

Start-up monitoring of OB 38

The time required for the execution of OB 38 is not monitored by the system program. You can, however, abort execution by changing the mode selector to STOP.

If an error occurs in OB 38, its execution is aborted and OB 39 is called if it exists.

Cyclic communication (OB 39)

If the cyclic program is interrupted causing a change to the SOFT STOP mode, **OB 39** is called as the user interface.

If the interruption of the cyclic program means that a handling block (communication) is not completely executed, it is possible and permitted to re-call the same block or the same block type in OB 39 (e.g. SEND).

Monitoring the execution time of OB 39

The execution time of OB 39 is monitored by the system program. If execution takes longer than 2.55 seconds (fixed value), the system program detects a cycle time error; it then calls the error OB, OB 26, and then processes OB 39 again from the beginning. If a cycle time error occurs again, an ISTACK depth > 5 causes an ISTACK overflow (reaction: see following paragraph).

Reaction to an error in OB 39

If an error occurs in OB 39 or in a handling block called in OB 39 (e.g. QVZ), the system program calls the appropriate error organization block. After this has been executed, the program processing is continued in OB 39. If the error OB does not exist, OB 39 is executed again from the beginning. (Exception: with QVZ, KB and errors in the self test there is no reaction).
If further errors occur, an ISTACK depth > 5 causes an ISTACK overflow:
the system program aborts program execution (OB 39 is no longer called), the CPU however remains in the SOFT STOP mode.

Data are not reset

If cyclic execution has already taken place in the RUN mode, the values of counters, timers, flags and the process image are retained during the transition to the stop mode.

Real-time clock

The real-time clock continues to run. It is updated in the RS area at 10 ms intervals.

BASP signal

The BASP (disable output command) signal is active. This disables all the digital outputs (exception: in the test in multiprocessor operation and with the PG function "force outputs", BASP is not active - refer to Section 10.1.8).

ISTACK

If a user program was processed prior to the stop mode, information is entered in the interrupt stack (ISTACK) providing information about the cause of the interruption.

Timer processing in OB 38/39

While OB 38/39 is being executed, the processing of timers and counters is stopped. If necessary, timer information must be processed from system data area RS 96 to RS 99 or with OB 121 or OB 150.

OB 38/OB 39 call

Figures 4-3 and 4-4 illustrate the principle of the OB 38 and OB 39 calls.

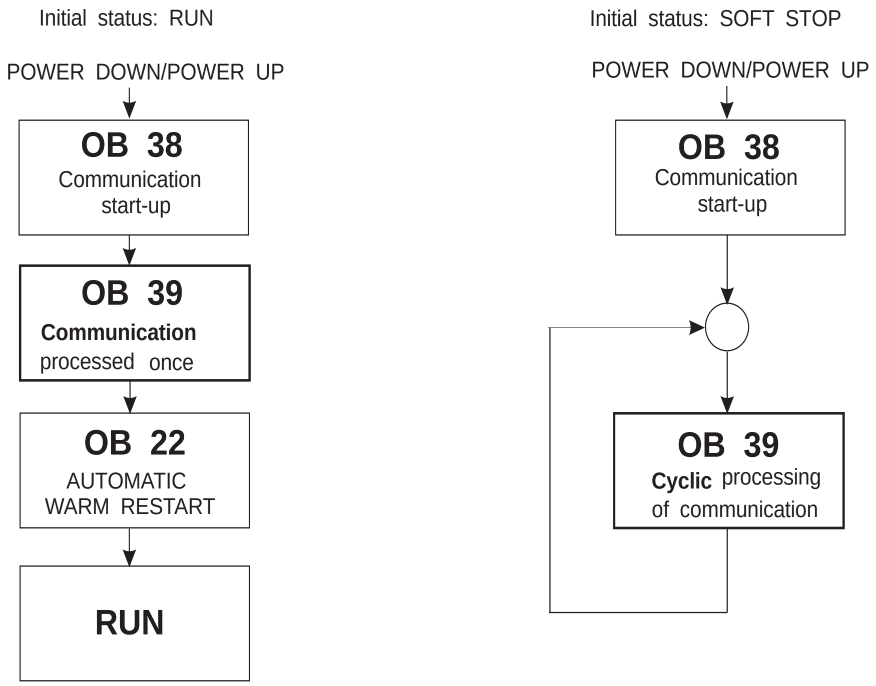


Fig. 4-3 Program execution after POWER UP

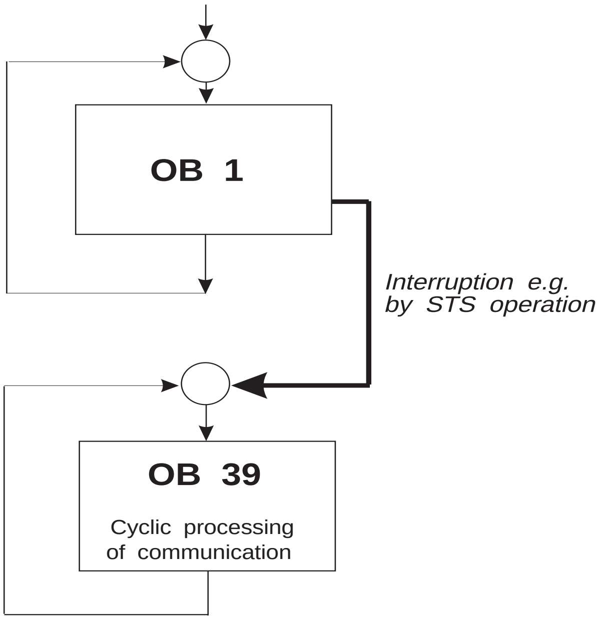

OB 1
Interruption e.g. by STS operation
OB 39
Cyclic processing of communication

Fig. 4-4 Program execution after a cycle interruption

LED displays

The SOFT STOP status can be recognized by the LEDs on the front panel of the CPU as follows:

LED	Status
RUN	off
STOP	on (continuous or flashing light)
SYSFAULT	off
BASP	on (except in test mode with multiprocessor mode or with PG function "force outputs")

The **STOP LED** signals the possible causes of the current stop status, as follows:

STOP LED lit continuously

The SOFT STOP mode was triggered by the following:

in single processor operation:

- by changing the mode selector from RUN to STOP,
- by the PG function PLC STOP,
- by a device fault (PEU),
- following an OVERALL RESET.

in multiprocessor operation:

- by changing the mode selector on the COR to STOP,
- a different CPU has changed to the STOP mode due to a problem (each CPU **not** causing the error has a constantly lit LED) or by the STOP switch,
- PG function PLC STOP
- PG function "program test end" on a different CPU

STOP LED flashes slowly (approx. once every 2 sec)

The SOFT STOP was triggered by the following:

- STP or STS statement in the user program,
- operator error (DB 1/DX 0 error, selection of an illegal start-up mode etc.),
- BSTACK overflow (STUEB) or bracket counter overflow (KZU),
- programming errors or device faults; the following LEDs provide further information:
  - "ADF" LED
  - "QVZ" LED
  - "ZYK" LED
- by the PG function "program test end" on this CPU.

Exiting the SOFT STOP status

The SOFT STOP status can be exited as follows:

a) by selecting a restart (refer to Section 4.4),

b) by an OVERALL RESET followed by a COLD RESTART.

4.2.2 HARD STOP

If the system program can no longer be executed properly, the CPU changes to the HARD STOP mode to ensure a safe mode in this situation.

A HARD STOP can be caused by the following:

- stop operation (STW) for the system program
- ISTACK overflow (STUEU),
- timeout (QVZ) or parity error (PARE) in the system RAM/EPROM,
- error in system program

Note
The CPU is stopped. You can only exit the HARD STOP mode by switching the power off and then on again!

LED displays

The HARD STOP status can be recognized by the following LEDs on the front panel of the CPU:

LED	Status
RUN	off
STOP	off
SYSFAULT	on
BASP	on

4.2.3 OVERALL RESET

With an OVERALL RESET the whole user memory and operand areas (flags, process images etc.) are deleted.

Requesting an OVERALL RESET

Before an OVERALL RESET can be performed, it must be requested. An OVERALL RESET is requested when the STOP LED flashes quickly.

Request by the system program

Each time you switch on the power, the CPU runs through an initialization routine. If an error is detected during the initialization (back-up voltage failure during POWER OFF), the system program requires an OVERALL RESET.

The cause of the problem must be eliminated (e.g. replacing the battery) following which an overall reset of the CPU is necessary.

An OVERALL RESET is also requested if a CPU or system error occurs. You can recognize this error because the request occurs again following the OVERALL RESET. In this case, contact your local Siemens representative.

Operator request

With the steps outlined in the table, you can also request an OVERALL RESET (the operating elements are on the front panel of the CPU - Fig. 4-1):

Step	Action	Result
1	Change the mode selector switch from RUN to STOP.	The CPU is in the STOP mode. The STOP LED is lit continuously.
2	Hold the reset switch in the OVERALL RESET position; at the same time, change the mode selector from STOP to RUN and then back to STOP.	An OVERALL RESET is requested. The STOP LED flashes quickly.

Note
If you do not want to execute the OVERALL RESET you requested, carry out a MANUAL COLD RESTART or MANUAL WARM RESTART.

Performing an OVERALL RESET

Regardless of whether you or the system program requested the OVERALL RESET, you perform the OVERALL RESET as follows (initial status: STOP LED flashing quickly):

OVERALL RESET using control elements on the CPU

Action	Result
Hold the reset switch in the OVERALL RESET position. At the same time, change the mode selector from STOP to RUN and back to STOP again (refer to Fig. 4-1).	An OVERALL RESET is performed. The STOP LED is lit continuously.

OVERALL RESET from the PG

Action	Result
Activate the PG function "delete blocks".	An OVERALL RESET is performed. The STOP LED is lit continuously.

Note
In contrast to the CPU 946/947, you can also initiate an OVERALL RESET on the CPU 948 in the RUN mode. In this case, the CPU automatically changes to the STOP mode and the OVERALL RESET is then performed.

During the OVERALL RESET, all the LEDs are unlit apart from the INIT and BASP LEDs.

Once the OVERALL RESET is completed, the only permitted start-up mode is then a COLD RESTART.

Loading the memory card and calling OB 39

After performing the OVERALL RESET, the memory card is loaded (provided it is plugged in) following which OB 39 is called.

4.3 START-UP Mode

The START-UP mode has the following features:

Mode change

START-UP is the transition from the STOP mode to the RUN mode.

Start-up types

The CPU 948 has the following start-up modes:

- COLD RESTART (manual or automatic)
- WARM RESTART (manual or automatic)

(You can select the type of start-up with operating elements and by assigning parameters in DX 0)

COLD RESTART

The cyclic user program is processed from the beginning.

The system program calls **OB 20** as the user interface.

WARM RESTART

The execution of the cyclic user program is continued from the point at which it was interrupted.

The system program calls the following OBs as the user interface:

- **OB 21** for a MANUAL WARM RESTART,
- **OB 22** for an AUTOMATIC WARM RESTART.

No time monitoring of the start-up OBs

The execution time of the start-up organization blocks is **not** monitored. You can call other blocks within these OBs. Make sure you avoid endless loops!

Counters, timers, flags, process images

The values of counters, timers, flags and process images are handled differently in the various start-up types (refer to Section 4.4.3).

BASP

The BASP signal (disable output command) is active. This disables all the digital outputs (exception: in the test mode in multiprocessor operation, the BASP is not activated **-** refer to Section 10.1.8).

Interrupts

The interrupts (normal interrupts, process interrupts, timed interrupts) are disabled.

Start-up in the multiprocessor mode

Starting up in the multiprocessor mode is described in Section 10.1.7.

4.3.1 MANUAL and AUTOMATIC COLD RESTART

When is a COLD RESTART permitted?

A COLD RESTART is **always permitted** provided the system is not requesting an OVERALL RESET.

When is a COLD RESTART necessary?

A COLD RESTART is **necessary** after the following:

- OVERALL RESET,
- loading the user memory with the user program while the CPU is in the stop mode
- ISTACK/BSTACK overflow,
- COLD RESTART aborted (by POWER OFF or changing the mode selector to "STOP"),
- stop after PG function "program test end"

MANUAL COLD RESTART

You can **trigger** a MANUAL COLD RESTART as follows:

- Using the operating elements of the CPU:

  Hold the reset switch in the RESET position; at the same time change the mode selector from STOP to RUN (refer to Fig. 4-1).

- From the PG:

  Select the PG function PLC START/COLD RESTART.

AUTOMATIC COLD RESTART

An AUTOMATIC COLD RESTART is **triggered** as follows:

At POWER UP, when

- the default "AUTOMATIC WARM RESTART after POWER UP" in DX 0 has been changed to "AUTOMATIC COLD RESTART after POWER UP",
- the mode selector on all CPUs and on the coordinator must remain set to RUN
  and
  the CPU was not in the STOP mode when the power was switched off.

> **Note**
> If the CPU was in the STOP mode when the power was switched off (for example, following an addressing error), an **AUTOMATIC** COLD RESTART is **not** permitted. The STOP mode can only be exited in this case with a **MANUAL** COLD RESTART.

Aborting a cold restart

You can abort an active COLD RESTART only by changing the mode selector to STOP or by switching off the power. If you abort a COLD RESTART you must repeat it.

4.3.2 MANUAL and AUTOMATIC WARM RESTART

When is a WARM RESTART possible and permitted?

A MANUAL WARM RESTART is **only possible** after stoppages caused by the following:

- the mode selector was changed from the RUN position to the STOP position,
- a stoppage in multiprocessor operation caused by the HALT signal from the coordinator,
- POWER OFF, with the appropriate setting in data block DX 0,
- PG function PLC STOP.

> **Note**
> If the stoppage was caused by an event other than those listed above, then **no warm restart** is possible. The system program will only permit a COLD RESTART.

A MANUAL or AUTOMATIC WARM RESTART is **only permitted** when

- the user program was not modified during the stop mode

  and

- a COLD RESTART is not necessary for other reasons (refer to Section 4.4.1).

MANUAL WARM RESTART

You **trigger** a MANUAL WARM RESTART as follows:

- using the control elements of the CPU:

  initial state: the reset switch is in the mid setting

  change the mode selector from STOP to RUN (refer to Fig. 4-1)

- from the PG:

  select the PG function PLC START/WARM RESTART.

AUTOMATIC WARM RESTART

An AUTOMATIC WARM RESTART **is triggered** as follows:

With POWER UP, when

- the default "AUTOMATIC WARM RESTART after POWER UP" is set in data block DX 0 or DX 0 does not exist,

- the mode selector on all CPUs and on the coordinator remain unchanged and set to RUN
  and
  the CPU was not in the STOP mode when the power was switched off,

- no further errors have occurred during the initialization nor before the power was switched off,

- no COLD RESTART is required due to the reasons listed above.

Following power failure or switching the power off in the RUN mode followed by the return of power/POWER UP, the CPU runs through an initialization routine and then automatically performs a WARM RESTART.

If there is a power failure on an expansion unit, (PEU signal), the CPU changes to the STOP mode. It remains in this mode until the PEU signal is switched inactive and then performs an AUTOMATIC WARM RESTART or an AUTOMATIC COLD RESTART.

Note
With a WARM RESTART note the following special situation:

The CPU is currently processing an error OB (e.g. due to an addressing error ADF) and then changes to the STOP mode owing to POWER OFF, HALT, stop switch or PG-STP. Following this, a MANUAL or AUTOMATIC WARM RESTART is executed.

Reaction of the CPU:

- **Before** OB 21/22 is called, the interrupted processing of the error OB is completed.

- If the error OB does not lead to a stop operation, then following the processing of the remainder of the error OB, a **WARM RESTART** is executed.
- If the error OB sets the CPU to the stop mode, then only a **COLD RESTART** is possible.

Aborting a WARM RESTART

You can only abort a WARM RESTART after it has started by changing the mode selector to STOP or by POWER OFF. If you abort the warm restart in this way, both a COLD RESTART or WARM RESTART is then possible.

4.3.3 Comparison between COLD RESTART and WARM RESTART

The following table contains a comparison of the start-up types COLD RESTART and WARM RESTART..

Table 4-2 Characteristics of COLD RESTART and WARM RESTART

	COLD RESTART	WARM RESTART
Manual triggering:	Mode selector from position STOP to RUN and reset switch set to RESET position or PG function PLC START (COLD RESTART)	Mode selector from position STOP to position RUN or PG function PLC START (WARM RESTART)
Automatic triggering:	Switching on the power supply, when "AUTOMATIC COLD RESTART after POWER UP" is entered in DX 0	Switching on the power supply when the default is entered in DX 0 or no DX 0 exists
System program activities:	Set up block address list in DB 0 Delete process image of the inputs Delete process image of the outputs	Block address list retained in DB 0 Process image of the inputs retained Process image of the outputs retained
	Delete flags, timers and counters Delete digital/analog I/Os (each 2 x 128 bytes) Delete IPC flags (256 bytes) Delete delayed interrupts and timed jobs Delete ISTACK/BSTACK Delete semaphore	Flags, timers and counters retained IPC flags retained Delete delayed interrupts, timed jobs retained ISTACK/BSTACK retained Semaphore retained
	If DB 1 exists: write the digital I/Os entered in it into the PI lists If DB 1 does not exist: enter the modules which actually exist (only digital I/O) into the PI lists IPC flags are ignored	No entries made from DB 1

	COLD RESTART	**WARM RESTART**
	Table 4-3 continued:	
System program activities (continued):	Set system parameters according to the settings in DX 0	DX 0 not evaluated
	Call user interface **OB 20** (if it exists) [1]	Call user interface **OB 21/22** (if they exist) [1]
	Synchronize start-up in multiprocessor operation	Synchronize start-up in multiprocessor operation
	Transition to the cycle: - switch BASP inactive, - call OB 1	Transition to the cycle: - BASP remains active - delete process image of the outputs - process remaining cycle - switch BASP inactive - call OB 1

[1] Following POWER UP, the user interfaces are called in the following order during the START-UP: OB 38, OB 39, OB 20/OB 22.

4.3.4 RETENTIVE COLD RESTART

If the parameter for a "cold restart with memory" is stored in the loaded DX 0 block, the system program goes through a RETENTIVE COLD RESTART instead of a WARM RESTART. How this differs from a "normal" COLD RESTART can be seen in the following section.

4.3.5 Comparison of COLD RESTART and RETENTIVE COLD RESTART

The following table shows the differences between a COLD RESTART and RETENTIVE COLD RESTART.

Table 4-3 Differences between a cold restart and a RETENTIVE COLD RESTART

	COLD RESTART	RETENTIVE COLD RESTART
Manual triggering:	Mode selector from position STOP to position RUN and reset switch set to RESET or PG function PLC START (COLD RESTART)	Mode selector from position STOP to position RUN or PG function PLC START (WARM RESTART)
Automatic triggering:	Switching on the power supply, when AUTOMATIC COLD RESTART after POWER UP is entered in DX 0	Switching on the power supply when AUTOMATIC WARM RESTART after POWER UP and COLD RESTART WITH MEMORY is entered in DX 0
System program activities:	Set up block address in DB 0 Delete process image of the inputs Delete process image of the outputs Delete delayed interrupts and timed jobs Delete flags, timers and counters Delete digital/analog I/Os (each 2 x 128 bytes)	Block address list retained in DB 0 Process image of the inputs retained Process image of the outputs retained Delete delayed interrupts and timer jobs Flags, timers and counters retained Delete digital I/O (128 bytes) Analog I/Os retained (128 bytes)
	Delete IPC flags (256 bytes) Delete ISTACK/BSTACK Delete semaphore	IPC flags retained Delete ISTACK/BSTACK Semaphore retained
	If DB 1 exists: enter the digital I/Os and IPC flags it contains in the PI lists If DB 1 does not exist: enter the existing modules (only digital I/Os) in the PI lists IPC flags are ignored	No entries from DB 1

	COLD RESTART	RETENTIVE COLD RESTART
System program activities (continued)	Table 4-4 continued:	
	Set system parameters according to default in DX 0	No evaluation of DX 0
	Call user interface **OB 20** (if it exists) [1]	Call user interface **OB 21/22** (if it exists) [1]
	Synchronize start-up in multiprocessor operation	Synchronize start-up in multiprocessor operation
	Transition to the cycle: - switch BASP inactive - call OB 1	Transition to the cycle: - switch BASP inactive - call OB 1

[1] After POWER ON, the user interfaces are called in the following order during START-UP: OB 38, OB 39, OB 20/OB 22.

4.3.6 User Interfaces for Start-Up

The organization blocks OB 20, OB 21 and OB 22 serve as user interfaces for the various types of start-up. You can store your STEP 5 program for the type of start-up in these blocks.

OB 20

When the CPU executes a MANUAL or AUTOMATIC COLD RESTART, the system program calls OB 20 **once**. In OB 20, you can store a STEP 5 program which is responsible for preliminary steps for a **cold restart** of cyclic processing prior to the execution of the cyclic program.

You can, for example:

- set flags
- start timers (the start is executed by the system program when it enters the RUN mode)
- set default values for data to be output to I/O modules
- synchronize CPs.

After processing OB 20, the cyclic program begins by calling OB 1. If OB 20 is not loaded, the CPU begins the cyclic program execution immediately after the COLD RESTART is completed (following the system activities).

OB 21

If the CPU performs a MANUAL COLD RESTART or RETENTIVE MANUAL COLD RESTART, the system program calls OB 21 once. Here, you can store a STEP 5 program which executes preliminary steps for a **warm restart** of the cyclic program.

MANUAL WARM RESTART

With a MANUAL WARM RESTART, the cyclic program is continued with the next statement following the point at which it was interrupted after processing OB 21, the sequence is as follows:

- The BASP signal (disable command output) remains active during the processing of the remaining cycle and is only switched inactive at the beginning of the next (complete) cycle.
- The process image of the outputs is reset at the end of the remaining cycle.
- If OB 21 is not loaded, the CPU begins again at the point at which the program was interrupted on completion of the MANUAL WARM RESTART and the system activities.

MANUAL RETENTIVE COLD RESTART

If the parameter "COLD RESTART WITH MEMORY" is entered in the data block DX 0, after processing OB 21, the system program then goes through a **COLD RESTART** (the CPU resumes program execution with the **first STEP 5 statement in OB 1**). The signal states of the flags, IPC flags, semaphore and the block address list (DB 0) **are retained**.

OB 22

When the CPU executes an AUTOMATIC WARM RESTART or AUTOMATIC RETENTIVE COLD RESTART, the system program calls OB 22 once. Here, you can store a STEP 5 program which executes preliminary steps (generally following a power failure) for a warm restart of cyclic program execution.

AUTOMATIC WARM RESTART

Following POWER UP, the CPU executes the system activities listed in Section 4.4.4 for a warm restart and attempts to resume the program at the point at which it was interrupted.

OB 22 is called first.

After executing OB 22, the interrupted program execution is resumed with the next statement following the point at which the program was interrupted.

Following a power failure and the return of power:

- The BASP signal (disable command output) remains active during the remaining cycle and is only switched inactive at the beginning of the first complete cycle.

- The process image of the outputs is reset at the end of the remaining cycle.

- If OB 22 is not loaded, the CPU resumes program processing immediately at the point of interruption at the end of the AUTOMATIC WARM RESTART.

AUTOMATIC RETENTIVE COLD RESTART

If the parameter "COLD RESTART WITH MEMORY" is entered in the data block DX 0, after processing OB 22, the system program then goes through a **COLD RESTART** (the CPU resumes program execution with the **first STEP 5 statement in OB 1**). The signal states of the flags, IPC flags, semaphore and the block address list (DB 0) **are retained**.

4.3.7 Extended AUTOMATIC WARM RESTART with the CPU 948 (HOT RESTART)

The "HOT RESTART" mode specified in the IEC 1131 standard, part 1 is also possible in the CPU 948. The "HOT RESTART" is a warm restart controlled by a battery-backed clock (according to IEC 1131). The clock monitors the time between switching off and switching on the power supply for the CPU. Whether or not a warm restart is then permitted depends on the time elapsed.

The automatic "HOT RESTART" of the CPU 948 is not supported directly by the system program, which means that you must program this yourself.

The basic functions available are the AUTOMATIC WARM RESTART (OB 22) and the real time clock which although internal is backed up by an external battery.

A "HOT RESTART" is programmed as shown below:

<table>
<tr><th>Mode</th><th colspan="2">Activity/block</th></tr>
<tr><td>RUN</td><td colspan="2">Save time (time of day and date) regularly from the external clock in defined memory cells (e.g. in data words):
- at the end of the cycle in OB 1
or
- time-controlled by timed interrupts (e.g. OB 10), if higher accuracy is necessary</td></tr>
<tr><td rowspan="5">AUTO-MATIC WARM RESTART</td><td colspan="2">OB 22</td></tr>
<tr><td colspan="2">Calculate down time:
T_{down} = first time value after return of power - last time value saved before power failure</td></tr>
<tr><td><b>IF ...</b></td><td><b>THEN...</b></td></tr>
<tr><td>Down time > default maximum value</td><td>Stop warm restart
(STP operation)
or
execute modified warm restart</td></tr>
<tr><td>Down time ≤ default maximum value</td><td>Continue warm restart</td></tr>
</table>

4.3.8 Interruptions during START-UP

A start-up program can be interrupted by the following:

- power failure in the central controller (NAU) or in the expansion unit (PEU),
- stop switch, stop command, HALT or PG-STPor
- program errors or device faults (refer to Section 5.5).

Basic rules for an interrupted START-UP

The following basic rules apply to the start-up response of the CPU 948:

If the START-UP is interrupted, the subsequent START-UP is always restarted **from the beginning**.
The last selected type of start-up is selected. Example: 1. POWER DOWN (NAU) in the cycle 2. Switch set to STOP 3. POWER UP 4. Switch set to RUN Reaction: the CPU executes a MANUAL WARM RESTART
An interrupted COLD RESTART cannot be continued with a WARM RESTART, but must be repeated.
Following an interrupted WARM RESTART, both a COLD RESTART and a new WARM RESTART are possible.

Response of the CPU on the return of power after power failure or PEU signal

If the start-up execution is interrupted by a power failure or the PEU signal, the response of the CPU when power returns depends on the **set** and **interrupted** mode. The following table provides an overview.

Mode set in DX 0	**Interrupted mode**	**Interrupted start-up OB**	**Reaction of the CPU**
AUTOMATIC COLD RESTART	MANUAL COLD RESTART	OB 20	Depending on the prior events, an **AUTOMATIC COLD RESTART** is executed (OB 20 is called and processed from the beginning). No "**remaining start-up**" is processed, (OB 20 or OB 21 is not resumed).
	MANUAL WARM RESTART	OB 21	
	AUTOMATIC COLD RESTART	OB 20	
AUTOMATIC WARM RESTART	MANUAL COLD RESTART	OB 20	STOP
	MANUAL WARM RESTART	OB 21	An AUTOMATIC WARM RESTART is executed (OB 22 is called and processed from the beginning); no "**remaining start-up**" is processed (OB 21 or OB 22 is not resumed).
	AUTOMATIC WARM RESTART	OB 22	

4.4 RUN Mode

When the CPU has executed a START-UP (and only then) it changes to the **RUN** mode. This mode is characterized by the following features:

Execution of the user program

The user program in OB 1 is executed cyclically and additional interrupt-driven program sections can be nested in it.

Timers, counters, process image

All the timers and counters started in the program continue to run, the **process image is updated cyclically**.

BASP

The BASP signal (disable command output) is switched inactive. This enables all the digital outputs.

IPC flags

The IPC flags (if programmed in DB 1) are updated cyclically.

In the RUN mode, the following program execution levels exist:

CYCLE

The user program in OB 1 is processed cyclically.

PROCESS INTERRUPTS/ INTERRUPTS

The execution of the user program is interrupt-driven (4 interrupt levels or 1 process interrupt level with 8 sub-levels).

TIMED INTERRUPTS

The user program is processed time-controlled (9 cyclic timed interrupts, 1 delayed interrupt, 1 clock-controlled interrupt).

The execution levels differ from each other as follows:

- they are triggered by different events
- there are one or more organization blocks serving as user interfaces for each level of program execution.

All the processing levels in a CPU 948 can be programmed simultaneously. The levels are called by the system program according to the events that occur and the preset priority (refer to Section 4.2).

4.4.1 Cyclic Program Execution

With programmable controllers, **cyclic program execution** (program execution level **CYCLE**) is the main mode.

Triggering

If the CPU has completed the start-up program without errors, it then begins cyclic program execution.

Principle

The principle of cyclic program execution (system activities):

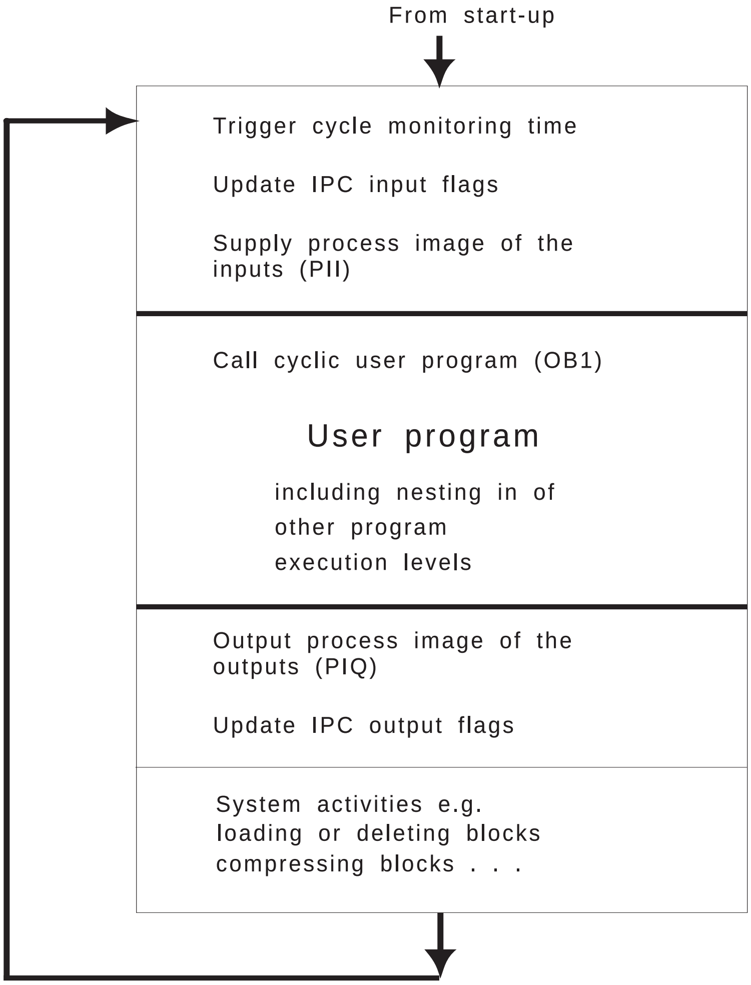

From start-up
Trigger cycle monitoring time
Update IPC input flags
Supply process image of the inputs (PII)
Call cyclic user program (OB1)
User program
including nesting in of other program execution levels
Output process image of the outputs (PIQ)
Update IPC output flags
System activities e.g. loading or deleting blocks compressing blocks . . .

Fig. 4-6 Cyclic program execution

User interface OB 1

During cyclic program execution, organization block OB 1 is called regularly as the user interface. The STEP 5 user program in OB 1 is processed from the beginning with the block calls you have programmed. After the system activities, the CPU starts again from the beginning with the first STEP 5 statement in OB 1.

In OB 1, you program the calls for program, function and sequence blocks to be executed within the cyclic program.

Interrupt points

Cyclic program execution can be interrupted by the following:

- time-controlled program execution at block or operation boundaries,
- process interrupt-driven program execution via input byte IB 0 at block boundaries,
- interrupt-driven program execution (interrupts INT A/B/C/D, E, F, G) at block or operation boundaries..

You specify the type of interrupt (at block or operation boundaries) in data block DX 0 (refer to Chapter 7).

Cyclic program execution can be interrupted or **aborted** regardless of the parameter setting in DX 0 as follows:

- when a device fault or program error occurs (at operation boundaries,
- by operator intervention:
  - stop switch, HALT (at operation boundaries),
  - PG function (at checkpoints - refer to Chapter 11),
- by the stop command STS (at operation boundaries),
- by a power failure on the central controller or in the expansion unit (at operation boundaries).

4.4.2 Specifying Time and Interrupt-Driven Program Execution

With time and interrupt-driven program execution, various types are available which can at present only be used as alternatives (i.e. not mixed). You decide which of the types of processing you want to use by setting the parameter in data block DX 0 (refer to Chapter 7).

"Process interrupts via IB 0"

The execution mode "process interrupts via IB 0" has the following features:

- only possible in **single processor operation**,
- the nesting of higher priority program levels is only possible at **block boundaries**,
- the delayed interrupt (processed by OB 6) **cannot** be used,
- the time-controlled interrupt (processing by OB 9) **cannot** be used.

"System interrupts" mode

The "system interrupts" mode is characterized by the following features:

- **single or multiprocessor mode** possible,
- higher priority program levels are nested at **block or command boundaries**,
- delayed interrupts are processed by OB 6,
- time-controlled interrupts are processed by OB 9.

Interrupting time and interrupt-driven program execution

Time and interrupt-driven program execution can be interrupted or **aborted** regardless of the parameter setting in DX 0 as follows:

- when a device fault or program error occurs (at operation boundaries,
- by operator intervention:
  - stop switch, HALT (at operation boundaries),
  - PG function (at checkpoints - refer to Chapter 11),
- by the stop command STS (at operation boundaries),
- by a power failure on the central controller or in the expansion unit (at operation boundaries).

4.4.3 Time-Controlled Program Execution

This type of program execution includes the delayed interrupt, the time-controlled interrupt and cyclic timed interrupts.

All these interrupts are **time-controlled**.

Time-controlled program execution uses the **TIMED INTERRUPTS** level.

Delayed interrupts

Triggered once after a selected delay time in the millisecond range. Organization block OB 6 is called with this interrupt.

Clock-controlled interrupt

Triggered at a selected interval or once at an absolute point in time. Organization block OB 9 is called with this interrupt.

Cyclic timed interrupt

Triggered at 9 different intervals. Each timed interrupt is assigned an organization block (OB 10 to OB 18). This involves fixed cycles, i.e. the interval between two program stops is fixed.

Priorities

Within time-controlled program execution, the following priorities are set:

Delayed interrupt OB 6	OB 6	ascending priority
cyclic timed int., period 1	OB 10, shortest period	
cyclic timed int., period 2	OB 11	
cyclic timed int., period 3	OB 12	
cyclic timed int., period 4	OB 13	
cyclic timed int., period 5	OB 14	
cyclic timed int., period 6	OB 15	
cyclic timed int., period 7	OB 16	
cyclic timed int., period 8	OB 17	
cyclic timed int., period 9	OB 18, longest period	
time-controlled interrupt	OB 9	

Note
Time-controlled interrupt servicing in OB 6 and in OB 9 is only possible when the parameter "process interrupts via IB 0 = off" is set in DX 0. With the default setting in DX 0 ("process interrupts via IB 0 = on") the corresponding process interrupts of IB 0 are processed using OB 6 and OB 9 (refer to Section 4.5.4).

Delayed interrupt

With the delayed interrupt of the CPU 948, small time intervals with a resolution of 1 ms can be set. Once the selected time has elapsed, the system program calls OB 6 **once**.

Resolution

The delayed interrupt is generated by calling the special function organization block OB 153 (refer to Section 6.14). As soon as the delay time assigned with OB 153 has elapsed, the system program interrupts the current program execution and calls OB 6 (program execution level TIMED INTERRUPTS). Following this, program execution is resumed at the point at which it was interrupted.

The use of the delayed interrupt is, however, only possible when "process interrupts via IB 0 = off" is set in the data block DX 0.

User interface OB 6

OB 6 is called as the user interface for a delayed interrupt. In OB 6, you write a STEP 5 program to be executed in this situation. If OB 6 is not loaded, program execution is not interrupted.

Interruptions

With the default setting, the TIMED INTERRUPTS level has the highest priority of the basic levels (can be modified by changing the parameter assignment in DX 0).

In timed-controlled program execution, the servicing of the delayed interrupt has highest priority.

Owing to the distribution of priorities, the processing of the delayed interrupt cannot be interrupted by any other user program.

Special features

- A delayed interrupt is only processed in the RUN mode. Delayed interrupts owing in the STOP mode, during **power down** or START-UP are **discarded**.
- A generated delayed alarm (= OB 153 call was processed) is **not** retained in the transition to the STOP mode and during POWER OFF.

- If you generate a new delayed interrupt, i.e. call OB 153 with new parameters, a previously set delayed interrupt is cancelled. A delayed interrupt currently being processed is continued. This means that only **one** delayed interrupt is valid at any one time.
- If a delayed interrupt occurs without the previous one being completely processed, the new interrupt is discarded. **Delayed interrupts are not checked for collisions!**
- Note the special functions OB 122 and OB 142 with which you can disable or delay the servicing of delayed interrupts.

Clock-controlled interrupt

The CPU 948 has a battery-backed clock (central back-up via the power supply of the central controller), which you can set and read out using the STEP 5 program. This clock allows time-controlled execution of a program section.

While the delayed interrupt is used for fast events, time-controlled interrupts are particularly suitable for processing events which occur **once** or which occur at **longer intervals**, e.g. hourly, daily or monthly. Once the point in time is reached, the system program calls OB 9.

Triggering

A clock-controlled interrupt (timed job) is generated by calling the special function organization block OB 151 (refer to Section 6.13). Once the time set in OB 151 is reached (a time, a date) the timed job is executed. The system program interrupts the current program execution and calls OB 9 (program execution level TIMED INTERRUPTS). Following this, program execution is resumed at the point at which it was interrupted.

To use the clock-controlled interrupt, "process interrupts via IB 0 = off" must be set in data block DX 0.

User interface OB 9

For a clock-controlled interrupt, OB 9 is called as the user interface. In OB 9, you write a STEP 5 program to be processed when the OB is called. If OB 9 is not loaded, program execution is not interrupted.

Interruptions

Owing to the default, the TIMED INTERRUPTS layer has the highest priority of the basic layers (can be modified in DX 0).

In time-controlled program execution, the execution of clock-controlled interrupts has the lowest priority. This can therefore be interrupted by the processing of a delayed interrupt or a cyclic timed interrupt.

Special features

- A clock-controlled interrupt is only processed in the RUN mode. Clock-controlled interrupts occurring in the STOP mode, when there is a **power failure** or during START-UP are **discarded**.
- Once a clock-controlled interrupt has been generated (= OB 151 call has been processed) it is retained in a WARM RESTART and following POWER DOWN/POWER UP; it is, however, deleted during a COLD RESTART.
- If you generate a new clock-controlled interrupt, i.e. call OB 151 with new time values, a previously set clock-controlled interrupt is cancelled. A clock-controlled interrupt currently being processed is continued. This means that only **one** clock-controlled interrupt is valid at any one time.
- If a time-controlled interrupt occurs without the previous one being completely processed, the clock time-controlled interrupt is discarded. **Clock-controlled interrupts are not checked for collisions!**
- Note the special functions OB 122 and OB 142 with which you can disable or delay the servicing of clock-controlled interrupts.

Cyclic timed interrupts

On the CPU 948, you can process 9 different time-controlled programs, each being called at a different cyclic interval.

Triggering

The basic clock pulse for timed interrupt processing is set to 100 ms. Using a special parameter in data block DX 0, you can adjust this in steps of 10 ms (basic clock pulse = yy ∗ 10 ms where: 01H ≤ yy ≤ FFH).

You can base the setting of the clock pulse on the shortest time required by your application for cyclic processing.

Time interval

Cyclic timed interrupts are processed at fixed intervals with 9 possible intervals (periods). Each interval is assigned to a specific organization block. You can select between two sets of intervals. You select the sets of intervals using a special parameter in data block DX 0.
the following table illustrates the two sets of intervals with the assignment of the different intervals to organization blocks.

Table 4-5 Sets of intervals and intervals of the TIMED INTERRUPTS

Interval set 1 (default)	Interval set 2	OB called
1x basic clock pulse	1 x basic clock pulse	OB 10
2 x basic clock pulse	2 x basic clock pulse	OB 11
5 x basic clock pulse	4 x basic clock pulse	OB 12
10 x basic clock pulse	8 x basic clock pulse	OB 13
20 x basic clock pulse	16 x basic clock pulse	OB 14
50 x basic clock pulse	32 x basic clock pulse	OB 15
100 x basic clock pulse	64 x basic clock pulse	OB 16
200 x basic clock pulse	128 x basic clock pulse	OB 17
500 x basic clock pulse	256 x basic clock pulse	OB 18

Note
The **first** TIMED INTERRUPT OB call following the start-up takes place within the time assigned to the OB.

If, for example, the interrupt time "500 s" is set for OB 18 (basic clock pulse setting in DX 0 = 1 s and time base = 1), then the first OB 18 call takes place after approximately 20 s following a COLD RESTART. All further calls are then at intervals of 500 s.

User interfaces OB 10 to OB 18

When a timed interrupt occurs, the corresponding organization block is called as the user interface at the next block boundary (or operation boundary).

For example, you would program the routine to be inserted in cyclic program execution every 100 ms in OB 10 (default).

The timed interrupt is only processed if the **assigned organization block is loaded**. If none of the organization blocks OB 10 to OB 18 are loaded, there is no time-controlled program execution and the cyclic program is not interrupted.

You can disable the execution of timed interrupts by setting a parameter in data block DX 0, e.g. for testing your program.

Interruptions

As default, the TIMED INTERRUPTS level has the highest priority of the basic levels (can be modified in DX 0).

Owing to the distribution of priority within time-controlled program execution, the following interruptions in the processing of a cyclic timed interrupt are possible:

- the processing of a cyclic timed interrupt can be interrupted by the processing of a delayed interrupt
- organization blocks with shorter time bases have higher priority and can interrupt organization blocks with longer time bases (e.g. OB 12 can interrupt OB 17).

Note
With the **three shortest time bases (OB 10 to 12)** multiple processing without interruption is possible. If, for example, while OB 10 is being processed, a further timed interrupt for OB 10 occurs, the currently active processing of OB 10 is first completed. Following this, OB 10 is called immediately again. If, however, there are more than **three timed interrupts** pending for one of the time bases, a **collision of timed interrupts** error occurs.

Collision of timed interrupts

In the CPU 948 there are two different types of collisions of timed interrupts:

Type of error/cause	ISTACK ID	Reaction of the CPU
Timed interrupt queue overflow: - there are more than three timed interrupts pending for one of the three shortest time bases (OB 10 to 12), - one of the other OBs (OB 13 to 18) is called again before it has been completely processed.	In the "ISTACK output" of the programmer, the error ID WEFES is marked in the control bits.	The system program calls OB 33 as the user interface. If this is not loaded, the CPU changes to the **stop mode**.
If the program can be interrupted at block boundaries, the timed processing is blocked by the run time of a block in the cyclic user program; the run time of the block is longer than the basic clock rate set in DX 0.	In the "ISTACK output" on the programmer, the error ID WEFEH is marked in the control bits.	The system program calls OB 33 as the user interface. If this is not loaded, the system program **continues program execution**.

Error reaction OB 33

In OB 33, you can program the required reaction to the interrupt collisions listed above. When OB 33 is called, the system program enters a collision ID in ACCU-1-L (bit nos. 0 to 9). You can see the meaning of these bits (bit = ’1’) in the following table.

Table 4-6 Timed interrupt collision IDs: meaning of the bits in ACCU-1-L

Bit number	Meaning
0	Queue overflow in timed interrupt period 1 (more than three timed interrupts are pending for OB 10).
1	Queue overflow in timed interrupt period 2 (more than three timed interrupts are pending for OB 11).
2	Queue overflow in timed interrupt period 3 (more than three timed interrupts are pending for OB 12).
3	Queue overflow in timed interrupt period 4 (OB 13 has been called again before the prior call was completely executed).
4	Queue overflow in timed interrupt period 5 (OB 14 has been called again before the prior call was completely executed).

Bit number	Meaning
5	Queue overflow in timed interrupt period 6 (OB 15 has been called again before the prior call was completely executed).
6	Queue overflow in timed interrupt period 7 (OB 16 has been called again before the prior call was completely executed).
7	Queue overflow in timed interrupt period 8 (OB 17 has been called again before the prior call was completely executed).
8	Queue overflow in timed interrupt period 9 (OB 18 has been called again before the prior call was completely executed).
9	Timed interrupt clock pulse masked for too long.

After processing OB 33, the program is resumed at the interrupted timed interrupt OB.

Note
If "interruptability at block boundaries" is set, following a collision of timed interrupts, the SAC does not point to the block at whose boundary the collision of timed interrupts took place (BE statement) but rather to the block which called the block that caused the error (the return address). You can set the following parameters in data block DX 0 (refer to Chapter 7) for time-controlled program execution:

- setting the basic clock rate
- setting the clock distributor,
- setting priorities relative to interrupt-driven program execution,
- enabling/disabling timed interrupt processing.

4.4.4 Interrupt-Driven Program Execution

Depending on the selected mode, two different types of interrupt-driven program execution are possible with the CPU 948:

- **PROCESS INTERRUPTS**
  via input byte IB 0 (max. 8 interrupts),
- **INTERRUPTS**
  via signal lines of the S5 bus (max. 4 interrupts)**.**

PROCESS INTERRUPTS via input byte IB 0

To service process interrupts, the default "process interrupts via IB 0 = on" must not be changed in the data block DX 0.

Program execution controlled by process interrupts means that a signal change in the input byte IB 0 causes the current program execution to be interrupted and a special program section to be executed.

Note
If you enable "servicing process interrupts via IB 0" you **cannot** use the delayed interrupt, the time-controlled interrupt and the system interrupt.

Triggering

The signal state change of a bit in input byte IB 0 triggers the process interrupt.

User interfaces OB 2 to OB 9

If a process interrupt occurs, one of the OBs listed in the following table is called as the user interface.

Table 4-7 User interfaces for process interrupts

Signal state change in IB 0 with bit	OB called
I 0.0	OB 2
I 0.1	OB 3
I 0.2	OB 4
I 0.3	OB 5
I 0.4	OB 6
I 0.5	OB 7
I 0.6	OB 8
I 0.7	OB 9

In the organization blocks OB 2 to OB 9, you program the part of your STEP 5 program to be executed when one of the process interrupts occurs indicated by a bit in input byte IB 0.
If the corresponding OB is **not loaded**, program execution is not interrupted. No interrupt-driven program processing takes place.

> **Note**
> If the I/O module no longer acknowledges when the CPU accesses IB 0, the system program recognizes a timeout and calls the user interface OB 28. If OB 28 is not loaded, the CPU changes to the stop mode.

Priority of process interrupts

The default setting means that PROCESS INTERRUPTS have a lower priority than the TIMED INTERRUPTS level.
A parameter in data block DX 0 allows you to change the default so that the PROCESS INTERRUPTS level has a higher priority than the TIMED INTERRUPTS level.

The following priorities are set for process interrupt processing:

I 0.0	OB 2	↑
I 0.1	OB 3	
I 0.2	OB 4	ascending priority
I 0.3	OB 5	
I 0.4	OB 6	
I 0.5	OB 7	
I 0.6	OB 8	
I 0.7	OB 9	

With process interrupts, **no** nested execution is possible. When a process interrupt OB has been completely processed and there are further process interrupts pending, the system program calls the OB with the next lower priority and processes it.
The PROCESS INTERRUPTS level is only exited when each signal change in input byte IB 0 has been dealt with and the corresponding OB completely processed.

> **Note**
> Process interrupt-driven program execution **cannot** be interrupted by further process interrupt-driven program execution.

INTERRUPTS via signal lines of the S5 bus

Interrupt-driven program execution means that an S5 bus signal from an I/O module with interrupt capability (e.g. digital inputs, IPs, CPs) causes the CPU to interrupt program execution and to process a specific section of program.

> **Note**
> If you want to use interrupt-driven program execution via S5 bus signal lines on your CPU, you must set "process interrupts via IB 0 = off" in DX 0 and activate the individual interrupts by means of DX 0 parameters. The interrupts must also be enabled with jumpers on the module (refer to the Appendix and /2/). In contrast to the CPU 946/947, you can set "interrupt at block boundaries" or "interrupt at operation boundaries" as the mode in DX 0.

Jumper settings for system interrupts

For interrupt-driven program execution with the CPU 948, there are four system interrupts available to you:

- INT A/B/C/D (dependent on the slot in the CPU, see the System Manual (Further Reading /2/)),
- INT E
- INT F
  and
- INT G.

The interrupts you want to use must be enabled with the supplied jumpers. The jumper plug is located on the basic board above the receptacle for the memory card. The exact position can be seen in Appendix 1.

Triggering

The active state of an interrupt line on the S5 bus triggers the interrupt. The interrupt signal is level-triggered (low level). To acknowledge the interrupt, please refer to the operating instructions for the module which triggers the interrupt.

User interfaces OB 2 to OB 5

If an interrupt occurs, one of the OBs listed in the following table is called as the user interface.

Table 4-8 User interfaces for interrupts

Interrupt triggered by	OB called
Interrupt signal X (A, B, C or D, slot-dependent)	OB 2
Interrupt signal E	OB 3
Interrupt signal F	OB 4
Interrupt signal G	OB 5

If, for example, the interrupt signal ’F’ occurs, the system program calls OB 4.

If the relevant OB is **not loaded**, program execution is not interrupted. There is no interrupt-driven program execution.

Priority of process interrupts

The default setting means that PROCESS INTERRUPTS have a lower priority than the TIMED INTERRUPTS level.
A parameter in data block DX 0 allows you to change the default so that the process interrupts level has a higher priority than the TIMED INTERRUPTS level.

Within interrupt servicing, the priorities of the individual interrupts are specified as follows:

- If there are several interrupts pending, the corresponding organization blocks are called according to the order of priority you specify in DX 0 (single priority).

  You can specify priority levels 1 to 5 for the four interrupts.

When an interrupt OB has been completely executed and there are further interrupts pending, the system program calls and processes the OB with the next lowest priority.
The INTERRUPTS processing level is only exited when every active signal state (low level) of an interrupt line on the S5 bus has been dealt with and the corresponding OB has been completely processed.

Note
Interrupt-driven program execution **cannot** be interrupted by the same interrupt occurring again.

Disabling interrupt-driven processing

An interrupt-driven program is called at a block or STEP 5 operation boundary in the cyclic program. This interrupt can cause problems when a cyclic section of program must be processed within a certain time (e.g. to achieve a certain reaction time) or when a series of operations must not be interrupted (e.g. when reading or writing interdependent values).

If a program section must **not** be interrupted by interrupt-driven processing, the following strategies are possible:

Interrupts at block boundaries

- Program this program section so that it does not contain a block change. Program sections that do not contain a block change can then not be interrupted.

- Use OB 122 with which you can disable the processing of process interrupts for a specific program section. Remember, however, that the timed interrupts are also disabled (refer to Section 6.3).

- Program the STEP 5 operation ’IA’ (disable process interrupts). With the operation ’RA’ (enable process interrupts) you can enable interrupt processing again.

  Between these two operations, no process interrupt-driven program execution is permitted. The program section between these two operations **cannot** be interrupted.

- ’IA’ and ’RA’ are only possible in function blocks (extended operation set - refer to Section 3.5.4) and only apply to process interrupts via IB 0.

Note
If a process interrupt is disabled using OB 22 or delayed using OB 142, the RA operation is not effective.

Interrupts at operation boundaries

- Program the section that must not be interrupted in an interrupt OB and assign the highest priority to it.
- Use the special function OB 122. With this, you can disable interrupts and timed interrupts (refer to Section 6.3).
- Using the operation LIM and SIM (system operations - refer to Section 3.5.4) read or set the 32-bit interrupt mask.

Interrupt servicing can be disabled completely or separately for individual interrupts in data block DX 0. This is, however, only possible following a COLD RESTART (refer to Chapter 7), since DX 0 is only evaluated in a COLD RESTART.

Reaction time

The time required to react to a process interrupt/interrupt request corresponds to the processing time of a block (for interrupts at block boundaries) or a STEP 5 operation (for interrupts at operation boundaries). If, however, at the time the cyclic program is interrupted there are higher priority timed interrupts pending, the interrupt-driven program is only executed when all the pending timed interrupts have been completely processed.

The maximum reaction time between the occurrence and execution of a process interrupt/hardware interrupt is increased in this case by the processing time of the higher priority timed interrupts.

Program execution levels and flags

If you run your user program not only cyclically but also time and interrupt-driven, you run the risk of overwriting flags.
This is the case when the same flag areas are accessed at different program processing levels.

It is therefore advisable to assign flags to individual program processing levels or at the beginning of time or interrupt-driven program execution, to "save" the signal states of multiply assigned flags in a data block and to write the values back at the end of the interrupt servicing. The same applies for a warm restart.

To avoid double use of flags, you can also use the S flags for most applications. Special "saving" strategies for flags are then no longer necessary, providing the S flags are assigned exclusively to individual program processing levels (there are enough S flags available).

Interrupt and Error Diagnostics 5

Contents of Chapter 5

5

Interrupt and Error Diagnostics

This chapter explains how to avoid errors when planning and programming your STEP 5 programs.
You will see what help you can get from the system program for diagnosing and reacting to errors and which blocks you can use to program reactions to errors.
At the end of the chapter, you will learn how to activate integrated system functions for a self-test of the CPU 948.

5.1 Frequent Errors in the User Program

The system program can detect effects of errors in the user program, faulty operation of the CPU, or errors in the system program processing.

The following list describes errors that occur most frequently during the start-up of the user program. You can avoid these errors by doing the following when you write you STEP 5 program.

- When specifying byte addresses for I/Os, make sure that the corresponding modules are plugged into the central controller or the expansion unit.
- Make sure that you have provided correct parameters for all operands.
- Be careful when changing function blocks. Check to see that the FBs/FXs are assigned the correct operands and that the actual operands are specified.
- Make sure that outputs, flags, timers, and counters are not processed in several locations in the program with operations that counteract each other.
- Make sure that timers are scanned only once per cycle (e.g., A T1).
- Make sure that all data blocks called in the program exist and are long enough.
- Check to see if all blocks called are actually in the memory.
- If required by other blocks (e.g. standard function blocks), scratchpad flags should be saved by interrupt-driven and time-controlled programs and loaded again when these program sections have been completed.

5.2 Error Information

If an error occurs during system start-up or during cyclic processing of your program, the sources of information described in this section can help you to find the problem. This includes:

- LEDs on the front panel of the CPU
- interrupt stack ISTACK and control bits
- block stack BSTACK

The following sections explain the ways of evaluating this information and how to use the information to analyze problems.

LEDs on the front panel of the CPU

If the CPU goes into the STOP mode when you do not want it to, check the LEDs on the front panel. They can indicate the cause of the problem.

LED display	Meaning
STOP LED lit continuously	The various states of the STOP LED indicate specific causes of interruptions and errors (see section 4.1).
STOP LED flashes slowly	
STOP LED flashes quickly	
SYS FAULT LED lit continuously	
ADF LED lit continuously	Addressing error
QVZ LED lit continuously	Timeout error
ZYK LED lit continuously	Cycle time exceeded error

PG online function OUTPUT ISTACK

You can get information about the status of the control bits and the contents of the interrupt stack (ISTACK) by using die PG online function OUTPUT ISTACK.

When the CPU goes into the STOP mode, the system program enters all the information it requires for a warm restart in the **ISTACK**. This information includes:

- contents of registers,
- contents of accumulators,
- STEP address counter SAC

  and

- results codes

Depending on where the interruption leading to the STOP occurred, the displayed information refers to user blocks or blocks of the system program (OB 0).

These entries are a valuable aid in error diagnosis.

Before the actual ISTACK is output on the programmer, the status of the **control bits** is displayed. The control bits mark the current operating status and specific characteristics of the CPU and the user program and provide additional information on the cause of an error.

You can call the ISTACK programmer function not only when the CPU is in the STOP mode, but also when it is in the RESTART or RUN mode. However, in the RESTART and RUN modes, you can only display the control bits (i.e., the first page of ISTACK information).

The meaning of the control bits and structure of the interrupt stack are described in detail in Section 5.4.

Online function OUTPUT BSTACK

You can display the contents of the block stack (BSTACK - see section 3.2 "Nesting blocks") using the PG online function OUTPUT BSTACK.

The BSTACK contains a listing of all blocks (blocks of the user program and organization block OB 0 of the system program) **called in sequence** and not completely processed when the CPU went into the STOP mode. Since the BSTACK is filled from the bottom, the top line of the BSTACK display contains the block that **called** the block containing an error.

In the **first** line, the information shown below is available:

Information	Meaning
BLOCK NO	Type and number of the block that called the faulty block
BLOCK ADDR	Absolute start address of the calling block in the program memory
RETURN ADDR	Absolute address of the first STEP 5 operation of this block in the user memory.
REL ADDR	Relative address (= difference "RETURN ADDR - BLOCK ADDR") of the next operation to be processed in the calling block. (You can display relative addresses on a programmer in the mode "disable input"/key switch and with S5-DOS from Stage IV upwards using the function key "addresses").
DB NO	Number of the last data block opened in the calling block
DB ADDR	Absolute start address in the program memory of the last data block opened in the calling block (address of data word DW 0)

Example

Evaluating the BSTACK function:

BLOCK NO	BLOCK ADDR	RETURN ADDR	REL ADDR	DB NO	DB ADDR
PB 3	00090	98	00008		0
PB 2	00050	51	00001		0
PB 1	00040	41	00001		0
OB 1	00010	11	00001		0
OB 66 [1)]	E2B10	E2C40	00130		0
OB 63	E0FC0	E12FA	0033A		0
OB 62	E0490	E0CBE	0082E		0
OB 61	E0010	E0273	00263		0

[1)] The blocks executed before OB 1 are internal blocks belonging to the system program (the BSTACK is structured chronologically).

```
In the example, PB 3 called the faulty block at relative address
"00008 - 1 = 00007".
During the jump to this faulty block, no data block was open.
```

5.3 Procedure for Error Analysis

If the CPU is in an abnormal stop mode, make use of all the information available to analyze the error, as follows:

Step	Action
1	Check the status of the STOP and SYS FAULT LEDs and the error LEDs on the front panel. These indicate causes of certain errors and interruptions.
2	Using the PG online function OUTPUT ISTACK, analyze the status of the control bits and the content of the ISTACK. This will provide you with further information about the location of the error and its cause.
3	Select the PG online function OUTPUT BSTACK: in the top line of the BSTACK display you will find information about the block which **called** the block causing the error.
4	The system data **RS 75** (refer to Section 8.3.4) also contains further detailed error information.

5.4 Control Bits and Interrupt Stack

You can use the online functions PLC INFO and OUTPUT ISTACK to analyze the following: operating status, characteristics of the CPU, characteristics of the user program, possible causes of errors and interruptions.

> **Note**
> You can display the **control bits** in **any** mode. You can display the **ISTACK** only in the **STOP** mode.

- The **control bits** indicate the current and previous operating status and the cause of the problem.
  If several errors occurred, the control bits indicate **all of them**.

- The **ISTACK** indicates the location of the interruption in question (addresses) with the current condition codes, the accumulator contents, and the cause of the problem.
  If several interruptions occurred, a multiple level ISTACK is constructed as follows (maximum 5 levels):

  DEPTH (level) 01 = last cause of interruption

  DEPTH (level) 02 = next to last cause of interruption, etc.

  When an ISTACK overflow occurs, the CPU goes into the STOP mode immediately (HARD STOP!). You must then turn the power off and on again and perform a cold restart.

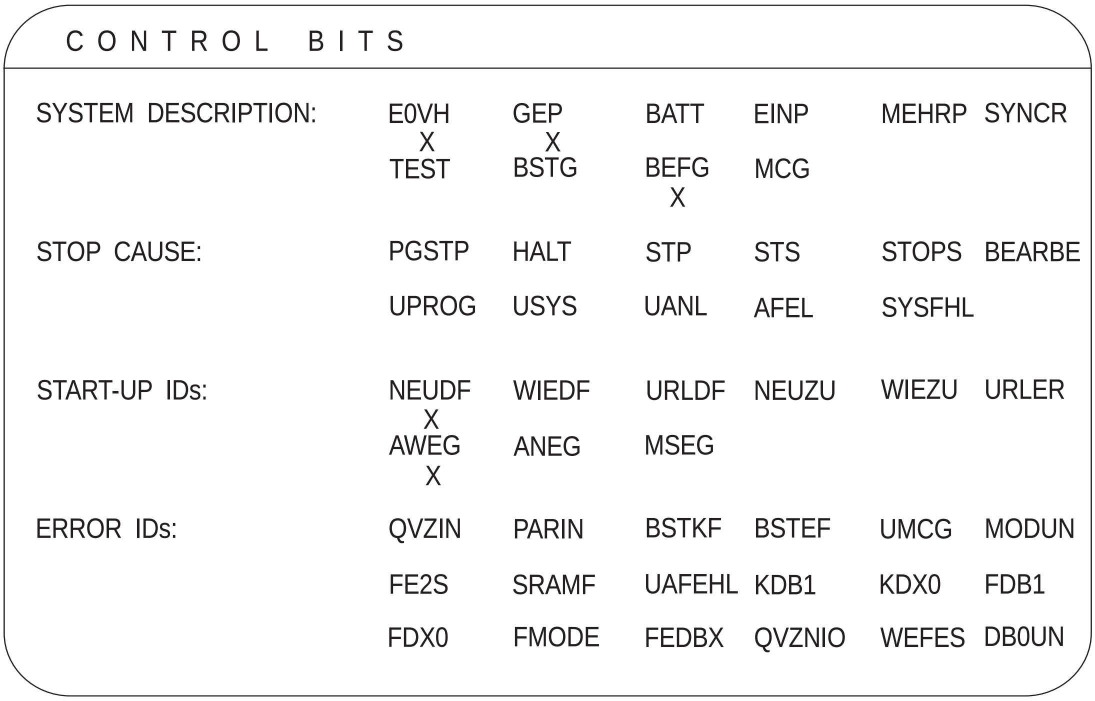
The meanings of the individual abbreviations in the control bits and in the ISTACK are described below.

> **Note**
> The text on the screen of your programmer depends on the PG software you are using. It may therefore differ from the display shown here. The description of the screen information is nevertheless relevant.

5.4.1 Control Bits

When you display the ISTACK on your programmer, the status of the control bits is indicated on the first page (see Fig. 5-1).

> **Note**
> The ISTACK screen form shown in Fig. 5-1 reflects the PG software STEP 5/ST, Version 6.3 or STEP 5/MT Version 6.0 with the "Delta diskette CPU 948". In older versions of the PG software, the abbreviations of the control bits may be different. The meaning of the control bits, however, is as described in the following tables.



```
  CONTROL BITS

SYSTEM DESCRIPTION:    E0VH    GEP     BATT    EINP    MEHRP  SYNCR
                         X       X
                       TEST    BSTG    BEFG    MCG
                                         X

STOP CAUSE:            PGSTP   HALT    STP     STS     STOPS  BEARBE

                       UPROG   USYS    UANL    AFEL    SYSFHL


START-UP IDs:          NEUDF   WIEDF   URLDF   NEUZU   WIEZU  URLER
                         X
                       AWEG    ANEG    MSEG
                         X

ERROR IDs:             QVZIN   PARIN   BSTKF   BSTEF   UMCG   MODUN

                       FE2S    SRAMF   UAFEHL  KDB1    KDX0   FDB1

                       FDX0    FMODE   FEDBX   QVZNIO  WEFES  DB0UN
```

Fig. 5-1 Example of the first screen form page "OUTPUT ISTACK": control bits

You can output the control bits in every mode. They mark the current or previous status of the CPU and provide information on specific features of the CPU and your STEP 5 program.

The control bits listed under ERROR IDS mark errors that can occur in the RESTART (e.g., during an initial cold restart) and RUN (e.g., during time-controlled program processing) modes. If several errors occur, **all** errors are displayed in the control bits.

The following tables explain the meaning of the individual bits.

Table 5-1 Meaning of the control bits SYSTEM DESCRIPTION

SYSTEM DESCRIPTION	
Bit	**Meaning**
E0VH	Input byte IB 0 (process interrupts) exists, i.e. the digital input module addressed with ’0’ was plugged in during the last cold restart and the module acknowledged.
GEP	Programmable controller has a central back-up battery.
BATT	Battery failure in the central controller (BAU)
EINP	Single processor operation
MEHRP	Multiprocessing operation
SYNCR	Start-up of the CPUs in multiprocessing operation is synchronized
TEST	Test operation
BSTG	DX-0 setting "interrupts at block boundaries"
BEFG	DX-0 setting "interrupts at operation boundaries
MCG	Memory card inserted

Table 5-2 Meaning of the control bits STOP CAUSE

STOP CAUSE (see RS 7)	
Bit	**Meaning**
PGSTP	STOP mode set from programmer
HALT	Multiprocessor STOP mode: a) Selector switch on the coordinator (COR) is in the STOP position or b) Stop status caused by command STOP operation from system program when the corresponding error OB is not loaded and an error occurs
STS	STOP mode caused by STEP 5 operation ’STS’ (after executing an operation)
STOPS	STOP mode caused by setting the mode selector to the STOP position
BEARBE	STOP mode after the PROGRAM TEST END programmer function

STOP CAUSE (see RS 7)	
Bit	**Meaning**
Table 5-2 continued:	
UPROG	STOP mode caused by user program
USYS	STOP mode caused by system program (warm restart possible)
UANL	STOP mode caused by illegal start-up type
AFEL	STOP mode caused by errors in the start-up block
SYSFHL	STOP mode caused by system error (may be caused by user error, e.g. overwriting system RAM with a block transfer or similar (when a system error is marked, a four-digit hexadecimal number/error code appears at the bottom edge of the screen - refer to RS 75 in Chapter 8).

Table 5-3 Meaning of the control bits START-UP IDS

START-UP IDs (see RS 8)	
Bit	**Meaning**
NEUDEF	COLD RESTART was executed as last start-up type.
WIEDF	WARM RESTART was executed as last start-up type.
URLDF	Overall reset was executed or is active.
NEUZU	COLD RESTART permitted as next start-up type.
WIEZU	WARM RESTART permitted as next start-up type.
URLER	Overall reset required.
AWEG	AUTOMATIC WARM RESTART is preset.
ANEG	AUTOMATIC COLD RESTART is preset.
MSEG	Manual start is preset.

Table 5-4 Meaning of the control bits ERROR IDS

ERROR IDs	
Bit	**Meaning**
QVZIN	Timeout error in initialization
PARIN	Parity error in initialization
BSTKF	Wrong block ID
BSTEF	Wrong block delimiter
UMCG	Illegal memory card inserted
MODUN	Content of the memory card too large for the available internal user memory
FE2S	Error on the second interface
SRAMF	System RAM error
UAFEHL	Error in the interrupt condition code word (UAW)
KDB1	No DB 1 in multiprocessing operation
KDX0	No DX 0 in multiprocessing operation
FDB1	Error in DB 1
FDX0	Error in DX 0
FMODE	No IB 0 process interrupts allowed in multiprocessor mode
FEDBX	Error in the STEP 5 operations G DB, GX DX
QVZNIO	QVZ test faulty
WEFES	Collision of software-driven timed interrupts: queue overflow
DB0UN	DB 0 has been changed since the last COLD RESTART. Therefore, no WARM RESTART possible.

5.4.2 ISTACK Content

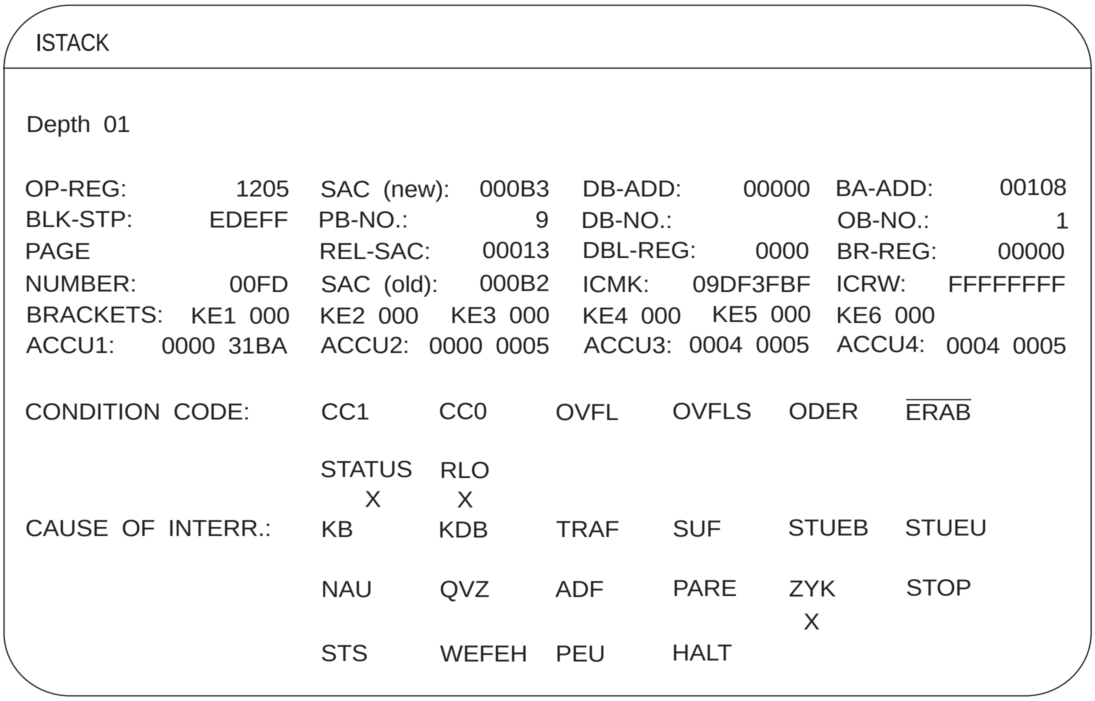
If the CPU is in the stop state, you can display the content of the ISTACK on the screen after the control bit display by pressing the enter key. When the CPU goes into the STOP mode, the system program enters all the information it needs in this ISTACK for a warm restart.

You can use the entries in this ISTACK to see what kind of error occurred and where it occurred in the program.

If the stop state was caused by **a single** error, only **one** level of the ISTACK information is displayed. With **several** errors, the **corresponding number** of ISTACK levels are output (DEPTH 01, DEPTH 02, etc.). At all levels, only one error is marked as the CAUSE OF INTERRUPT.

If several errors have occurred DEPTH 01 marks the error detected immediately before the change to the stop state.

Fig 5-2 is an example of a PG display of the ISTACK content.



```
ISTACK

Depth  01

OP-REG:          1205   SAC  (new):   000B3   DB-ADD:       00000   BA-ADD:        00108
BLK-STP:        EDEFF   PB-NO.:           9   DB-NO.:               OB-NO.:            1
PAGE                    REL-SAC:      00013   DBL-REG:       0000   BR-REG:        00000
NUMBER:          00FD   SAC  (old):   000B2   ICMK:      09DF3FBF   ICRW:       FFFFFFFF
BRACKETS:  KE1  000     KE2  000   KE3  000   KE4  000   KE5  000   KE6  000
ACCU1:     0000  31BA   ACCU2:   0000  0005   ACCU3:   0004  0005   ACCU4:   0004  0005

CONDITION  CODE:        CC1       CC0       OVFL      OVFLS     ODER      ERAB

                        STATUS    RLO
                          X        X
CAUSE  OF  INTERR.:     KB        KDB       TRAF      SUF       STUEB     STUEU

                        NAU       QVZ       ADF       PARE      ZYK       STOP
                                                                  X
                        STS       WEFEH     PEU       HALT
```

Fig. 5-2 Example of a screen page "OUTPUT ISTACK"

Explanation of the ISTACK screen

DEPTH

Information level of the ISTACK when more than one error has occurred:

DEPTH 01 = last cause of stop to occur
DEPTH 02 = next to last cause of stop to occur
......
DEPTH05 = ... (maximum depth)

Information about the error

The following table contains information about the ISTACK IDs with which the statement in the user program can be found which caused the CPU to change to the STOP mode.

Table 5-5 Meaning of the ISTACK IDs for errors

Information about the error	
ISTACK ID	**Meaning**
OP-REG	Operation register: Contains machine code (first word of the instruction processed last in an interrupted program processing level.
SAC (new)	STEP address counter (new): Contains the **absolute** address of the next operation in the **program memory** to be processed. When a warm restart occurs, the CPU continues the program with this operation.
DB-ADD	Absolute start address (DW 0) in the program memory of the data block currently opened (= 0000 if no data block was opened)
BA-ADD	Absolute address in the program memory for the operation to be processed next in the block where the **last** block call was made
BLK-STP	Block stack (BSTACK) pointer: Contains the 20-bit offset address of the last BSTACK entry (always **E**xxxx).
PB-NO (depending on type PB, OB ...)	Block type and number of the most recently **processed** block

Information about the error	
ISTACK ID	**Meaning**
Table 5-5 continued	
DB-NO.	Number of the data block currently opened
OB-NO. (depending on type OB, PB ...)	Block type and number of the last **calling** block
REL-SAC	Relative STEP address counter: contains the **relative** address (related to the block start address) of the next operation to be processed in the last block processed.
DBL-REG	Length of the data block currently opened
BS-REG	Content of the base address register prior to the transition to the stop mode
PAGE NUMBER	Number of the current dual-port RAM page selected (dual-port RAM access refers to this dual-port RAM page, you will find information about page access in Chapter 9)
SAC (old)	STEP address counter (old): contains the **absolute** address of the last operation processed in the **program memory** of an interrupted program level; if an error occurs, SAC (old) indicates the operation which caused the error.
ICMK	Interrupt condition code masking word: ICMK contains all the causes of interrupts which have occurred and not yet been completely processed.
ICRW	Interrupt condition code reset word
BRACKETS	Number of bracket levels: "KEx abc" where x = 1 to 7 levels, a = ’OR’ (refer to bit codes), b =’ RLO’ (result of logic operation, refer to bit codes), c = 1: ’A(’, c = 0: ’O(’.
ACCU1 to ACCU4	Contents of the calculation registers (accumulators) at the time of the interruption

CONDITION CODE

see Section 3.5

CAUSE OF INTERR.

The following abbreviations (ISTACK IDs) indicate the most important causes of interruptions.

Table 5-6 ISTACK IDs CAUSE OF INTERRUPTION

CAUSE OF INTERR.	
ISTACK ID	**Meaning (called error OB)**
KB	Called block not loaded (OB 19)
KDB	Opened data block not loaded (OB 19)
TRAF	Load or transfer error (OB 32)
SUF	Substitution error (OB 27): Processed STEP 5 operation cannot be substituted
STUEB	Block stack overflow: Nesting depth too great; required action: COLD RESTART
STUEU	Interrupt stack overflow: Nesting depth too great; required action: POWER DOWN, POWER UP, then COLD RESTART
NAU	Power failure in the central controller
QVZ	Timeout (OB 23/OB 24/OB 28/OB 29)
ADF	Addressing error for digital inputs and outputs with process image (OB 25)
PARE	Parity error (OB 30)
ZYK	Cycle monitoring time exceeded (OB 26)
STOP	STOP mode caused by setting the mode selector to the STOP position
STS	STOP mode caused by STEP 5 operation ’STS’ (after executing an operation)

CAUSE OF INTERR.	
ISTACK ID	**Meaning (called error OB)**
Table 5-6 continued:	
WEFEH	Collision of timed interrupts caused by the hardware clock (OB 33): timed interrupt clock was masked (ignored) for too long
PEU	I/Os not ready = power failure in expansion unit: After a statically pending PEU signal is removed (expansion unit is switched on), the system program always calls OB 22 (AUTOMATIC WARM RESTART).
HALT	Multiprocessor STOP mode: a) selector switch on the coordinator (COR) is in the STOP position b) another CPU entered the STOP mode in multiprocessing.

5.4.3 Example of Error Diagnosis using the ISTACK

Fig. 5-3 illustrates the structure of the ISTACK in conjunction with the interruptions that have occurred.

- The program execution level CYCLE (**OB 1**) is interrupted by an interrupt.
- Following this, the program processing level interrupt is activated and **OB 3** called.
- The occurrence of a timed interrupt means that the INTERRUPT level is exited and the TIMED INTERRUPT level activated and **OB 13** processed.
- An incorrect addressing operation leads to the activaton of the ADF level where **OB 25** is processed. In the error handling program, the user has programmed a stop operation (STS) the CPU aborts program execution.

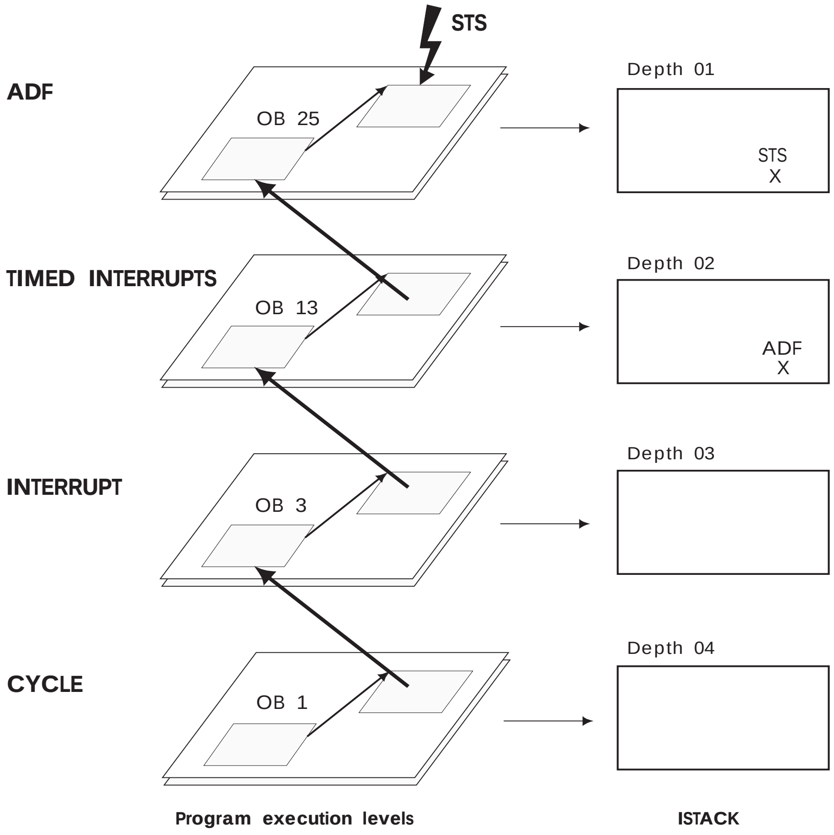


Fig. 5-3 Example of evaluating the ISTACK

Before the final transition to the stop mode, a total of four different program execution levels were interrupted. If you now display the ISTACK on the PG, you will obtain a **four-level** ISTACK, at the top the ISTACK with depth 01, with the ID of the **last** interrupted program execution level (= ADF). You can page down through the ISTACK until you reach depth 04, representing the CYCLE program execution level, which was interrupted **first**.

5.5 Error Handling Using Organization Blocks

When the system program detects an error, it calls the appropriate organization block to handle it. You can determine further operation of the CPU by programming the appropriate organization block. Therefore, the CPU can do one of the following:

- continue normal program processing
- go into the STOP mode

  and/or
- process a special "error handling program"

For the following causes of error, OBs are available:

Table 5-7 The organization blocks called in case of errors

Cause of error	Organization block called	Reaction of CPU if OB is not programmed
Call of a block that is not loaded (KB)	OB 19	none
Attempt to open a data block DB/DX that is not loaded (KDB)	OB 19	STOP
Timeout in the user program during access to I/O peripherals (QVZ)	OB 23	none
Timeout during update of the process image table and during interprocessor communication flag transfer (QVZ)	OB 24	none
Addressing error (ADF)	OB 25	STOP [1)]
Cycle time exceeded (ZYK)	OB 26	STOP
Substitution error (SUF)	OB 27	STOP
Timeout by reading input byte IB 0 (process interrupts – QVZ)	OB 28	STOP
Timeout during access to the distributed I/O peripherals (extended address area — QVZ)	OB 29	none
Parity error and timeout in the user memory (PARE)	OB 30	STOP

Cause of error	Organization block called	Reaction of CPU if OB is not programmed
Table 5-7 continued:		
Load and transfer error (TRAF)	OB 32	STOP
Collision of timed interrupts: a) queue overflow (control bit WEFES) b) timed interrupt clock was masked (ignored) for too long (WEFEH)	OB 33	 STOP none
Error during STEP 5 operation "G DB/GX DX" (control bit FEDBX)	OB 34	STOP
Error in self-test (refer to Section 5.7)	OB 36	none

[1] The CPU changes to the STOP mode only if the addressing error is not disabled by the STEP 5 operation "IAE".

Examples of reactions to organization blocks which are ***not loaded***

a) **No** reaction; cyclic program processing is **not** interrupted.

If a timeout error occurs and neither OB 23 nor OB 25 is loaded, cyclic program processing is **not** interrupted according to the table above. The CPU does not react.
If you want the CPU to go into the STOP mode when a timeout error occurs, you must enter a stop statement (STP for STOP at cycle end) in the appropriate organization block (e.g. OB 23 with QVZ) and terminate it with the block end statement ’BE’.

Example of OB 23:

```
:         QVZ has occurred
:
:STP      Cyclic processing is aborted
:BE       CPU changes to the stop mode
```

b) Reaction : the CPU changes to the **STOP mode**.

The CPU changes to the STOP mode immediately when a corresponding error (e.g. cycle or load/transfer error) occurs **-** if you did not load the appropriate organization blocks.

If, as an exception, you do not want one of these errors to interrupt cyclic program processing (e.g. while putting the system into operation), a block end statement in the appropriate organization block is sufficient.

Example of OB 25:

```
:         ADF occurred
:
:BE       Cyclic processing is continued, no CPU STOP
```

Interruptions during processing of error organization blocks

After the system program calls the appropriate organization block, the user program in that block is processed.
If another error occurs while that organization block is being processed, the program is interrupted at the next operation boundary and the appropriate organization block is called, just as in cyclic program processing.

The system program processes organization blocks in the order in which they are called.

Note
You can nest a maximum of five error organization blocks. With more than 5 errors, the CPU goes into the HARD STOP mode because of ISTACK overflow.

5.6 Causes of Error and Reactions of the CPU

Specific events can interrupt cyclic, time-controlled, or interrupt-driven program processing at operation boundaries when the CPU is in the RUN mode.

During initialization and also in the RESTART mode, interruptions can stop the start-up program and put the CPU into the STOP mode. The CPU then changes to the stop mode and calls the organization block for this particular error. Interruptions during the start-up program are handled like those in the RUN mode.

The reaction depends on the cause of the interruption:

- immediate change to the STOP mode, without calling the error OB (e.g. NAU → hard stop, STUEU → hard stop, PEU → soft stop),
- before changing to the STOP mode, the system program calls an error OB which you can program and (depending on the cause of the error) avoid a change to the stop mode (e.g. QVZ/IB 0 → OB 28, ADF → OB 25).

If an error occurs, note the entries in the control bits under "Error IDs" and the entries in the ISTACK under CAUSE OF INTERR.

The following sections explain possible causes of error in greater detail.

5.6.1 OB 19: Calling a Logic Block That Is Not Loaded (KB)

If your program jumps to a block that does not exist, the system program detects an error. This applies to all logic blocks and also for conditional and unconditional calls.

When the system program detects the call of a logic block that is not loaded, it calls **OB 19**, if this is loaded. In OB 19, you can specify how the CPU should proceed.

If OB 19 does not exist, the system program continues executing the interrupted STEP 5 program at the next operation.

5.6.2 OB 19: Calling a Data Block That Is Not Loaded (KDB)

If you call a data block or an extended data block in your program that does not exist in the memory or is marked as invalid, the CPU detects an error and the system program calls **OB 19**, if this is loaded. If OB 19 is not loaded, the CPU changes to the STOP mode. A zero is entered in the DBA and DBL registers.

> **Note**
> OB 19 is called both when a logic or data block is not loaded. You can read system data register RS 75 to determine (via the STEP 5 program) which type of error occurred. The contents of RS 75 are as follows:
> - for a KB error: 0101H,
> - for a KDB error: 0904H.

5.6.3 OB 23/24, OB 28/29: Timeout Error (QVZ)

A timeout error occurs when an addressable memory area does not respond to write or read accesses with the ready signal ("RDY") within a specific time after being addressed. This time is monitored by the hardware. A defective module or the removal of a module during operation of the programmable controller can cause a timeout error.

The following timeout errors interrupt the user program, jump to system program error handling, and call the appropriate blocks if they are loaded:

OB 23

QVZ with direct I/O access:

Cause of error	Reaction to error
Timeout error in the user program during direct access via the S5 bus to an IP, COR, or to a peripheral module (e.g., with load and transfer operations "L/T P..." or "L/T Q...").	If OB 23 is not loaded, the system program continues the processing of the user program.

OB 24

Cause of error	Reaction to error
Timeout error during update of the process image input/output tables or during transfer of interprocessor communication flags.	If OB 24 is not loaded, the system program continues processing of the user program.

Extension of the execution time

With calling OB 23 or OB 24 a timeout error increases the execution time of the STEP 5 operation which caused the timeout when the program is resumed:
extension = "acknowledgement monitoring time + time of error handling in the system program + processing time if error OB is called".

OB 28

Cause of error	Reaction to error
Timeout error at input byte IB 0 (process interrupts)	If OB 28 is not loaded, the CPU changes to the STOP mode.

OB 29

Cause of error	Reaction to error
Timeout error of the distributed peripherals in the following address areas: - F 0000H to F EFFFH - F F200H to F FFFFH	If OB 29 is not loaded, the system program continues processing of the user program.

Error address

When a timeout error occurs, you can read the error address in the system data area (see Chapter 8):

RS	Contents	Address
68	QVZ error address high	E F044H
69	QVZ error address low	E F045H

5.6.4 OB 25: Addressing Error (ADF)

An addressing error occurs when a STEP 5 operation references a process image input or output to which no I/O module was assigned at the time of the last COLD RESTART (the module is not plugged in, it is defective, or it is not defined in data block DB 1 of the CPU.

The STEP 5 operation at which the addressing error occurred is processed completely: For bit operations, the bit in the process image is scanned and combined logically or set/reset. Load and transfer operations are also executed. Continued processing can, however, result in incorrect or unwanted reactions.

When an addressing error occurs, the system program interrupts further processing of the user program and calls organization block **OB 25**. After running the program contained in OB 25, the program is resumed at the next operation.
If OB 25 is not loaded, the CPU changes to the STOP mode with an addressing error.

The STEP 5 IAE operation disables addressing error monitoring for individual program parts or for the entire program. You can enable it again using the RAE operation (see Section 3.5.4 and List of Operations).

5.6.5 OB 26: Cycle Time Exceeded Error (ZYK)

The cycle time is the time between the start of one OB 1 and the next. It includes the entire duration of cyclic program processing including interrupts, interrupt servicing and system program activities. The cycle monitoring time set on the CPU can, for example, be exceeded by incorrect programming (program loop).

Note
Hardware faults as the cause of cycle time errors are **extremely** rare. Normally, the error is in the user program or the programs and cycle monitoring time are incompatible.

When a cycle time exceeded error (ZYK) occurs, the system program interrupts the user program and calls **OB 26** if this is loaded. The monitoring time is then restarted (triggered). If the monitoring time is exceeded again, before OB 26 is completed, the CPU changes to the stop mode.

If OB 26 is not loaded, the CPU changes to the STOP mode.

The cycle monitoring time is variable (10 to 2550 msec) and is retriggerable (see above).
You can specify the cycle monitoring time individually by making an entry in DX 0 (refer to Chapter 7) or by programming OB 31. The default monitoring time is 200 ms.

In the cyclic program, the cycle monitoring time can be retriggered by calling the special function OB 222.

5.6.6 OB 27: (Substitution Error SUF)

If an operation with a formal operand is to be carried out in a function block, the CPU replaces (substitutes) this formal operand with the actual operand in the block when the block is called during user program processing.

If the CPU detects an illegal substitution, it interrupts the user program and calls **OB 27**, if this is loaded. If OB 27 is not loaded, the CPU changes to the STOP mode.

Apart from an illegal substitution, SUF is also indicated in the following situations:

- illegal operation code,
- special situation:
  you cannot open data blocks DB 0 and DB 1. The CPU handles the operations "C DB 0" and "C DB 1" like substitution errors. A zero is entered in the DBA and DBL registers.

5.6.7 OB 30: Parity Error and Timeout Error in the User Memory (PARE)

The user memory is protected by a parity bit. The system program checks whether the parity bit is correct each time the user memory is accessed. If the parity bit is incorrectly set, a parity error is indicated.

The system program calls **OB 30**. If OB 30 is not loaded, the CPU changes to the STOP mode.

The same reaction takes place if a timeout error occurs in the user memory.

PARE accessing the operating system RAM

If a parity error occurs when accessing the operating system RAM, the system program does **not** call OB 30, but **changes to a HARD STOP**.

Error address

If a parity error or timeout occurs, the address that caused the error can be read out of the system data area (refer to Chapter 8):

RS	Contents	Address
70	PARE error address high	E F046H
71	PARE error address low	E F047H

5.6.8 OB 32: Load and Transfer Error (TRAF)

A load and transfer error is indicated in the following situations:

- When accessing data in data blocks or extended data blocks, the CPU compares the length of the opened block to the parameter in the load or transfer operation.
  If the specified parameter exceeds the actual data block length, the CPU does not execute the load or transfer operation. This prevents data in the memory from being overwritten by mistake during transfer operations. With load errors, the contents of the accumulator are retained.

- A load or transfer error is also detected if a single bit within a non-existent data word is to be scanned or changed.

- If no data block has yet been opened (with "C DBn" or "CX DXn") prior to access to a data word, this also causes a load/transfer error.

- Accessing the memory using incorrect absolute addresses via the BR register or incorrect area boundaries with the "TNW", "TXW" and "TXB" operations can cause a load or transfer error.

When the system program detects a load or transfer error, it calls **OB 32**, if this is loaded. The operation that caused the load or transfer error is not processed.
If OB 32 is not loaded, the CPU changes to the STOP mode.

5.6.9 OB 33: Collision of Timed Interrupts Error (WEFES/WEFEH)

Time-controlled program processing (timed interrupts) is handled by organization blocks OB 6, OB 9 and OB 10 to OB 18.

The following types of timed errors can occur on the CPU 948:

Queue overflow

Cause:

Queue overflow servicing timed interrupts:

- there are more than three timed interrupts pending for one of the three shortest periods (OBs 10 to 12)

  or

- one of the other OBs (OBs 13 to 18) has been called before the first call was completely processed.

Reaction:

The system program calls **OB 33** as the user interface, if this is loaded. You can program the reaction to this state in this OB.

If OB 33 is not loaded, the CPU changes to the STOP mode.

PG display in "OUTPUT ISTACK":

The bit WEFES is marked in the control bits.

Masking the timed interrupt clock

Cause:

The internal timed interrupt clock is masked (ignored) too long (applies to interruptions at block boundaries/process interrupts).

This situation is related to the basic clock rate of the internal timed interrupts and the scan time of a block in the cyclic user program. If the scan of a cyclic block runs longer than the basic clock rate, a collision of timed interrupts occurs.

Reaction:

The system program calls **OB 33** as user interface, if this is loaded. Here, you can program the reaction to this state.

If OB 33 is not loaded, the CPU continues processing the program.

PG display with "OUTPUT ISTACK":

The bit WEFEH is marked in the control bits.

When the system program calls OB 33, a code for the collision of timed interrupts is transferred to ACCU-1-L (see Section 4.5.3).

Note
In the "process interrupt via IB 0" mode **and** "interruptability at block boundaries" the step address counter (SAC) does not point to the block **at whose boundary (BE statement) the collision of timed interrupts** took place. It points to the block that **called the block that caused the error** (return address).

As long as an error is pending or reoccurs every time the STEP 5 operation in question is processed in each scan, the appropriate error organization block is always called.

This can increase the cycle time considerably, depending on the duration of error handling by the system program and of processing time of the organization block.

5.6.10 OB 34: Error with G DB/GX DX (FEDBX)

Causes:

- with the operation G DB/GX DX, an illegal block number was specified (number of a reserved block, number > 255),
- with the operation G DB/GX DX, an illegal block length was specified (> 4091),
- there is no longer enough space in the user memory for the operation GDB/GXDX.

Reaction:

The system program calls **OB 34** is this is loaded. If OB 34 is not loaded, the CPU changes to the stop mode.

5.6.11 OB 35: Communication Errors

If problems occur on the second serial interface with an RK 512 computer link, data transfer with procedure 3964/3964R, data transfer with an open driver or data transfer with SINEC L1, the system program calls organization block OB 35 and enters additional information about the error in ACCU 1.

Reaction if OB 35 is not loaded

If you have not programmed OB 35, the system program does **not** react and the CPU does **not** change to the STOP mode.

Error information in ACCU 1

The system program checks whether errors have occurred on the serial interface every 100 ms. If an error has occurred, the system program enters error information in ACCU 1.

Error numbers for up to a maximum of three errors can be transferred when OB 35 is called. If there are more than three errors, this is indicated by an overflow identifier.

Structure of the error information in ACCU 1

	31							24	23 18	15 8	7 0
ACCU 1	0	0	0	0	F	U	B	0	Error number 1	Error number 2	Error number 3

F = ’0’, means no error entry
  = ’1’, means one or more errors entered

U = ’0’, means no error overflow (maximum three entries)
  = ‘1‘, means error overflow (more than three entries)

B = ’0’, means no BREAK on the interface
  = ’1’, means BREAK on the interface

BREAK

If there is a BREAK on the interface, OB 35 is only called at the beginning of the BREAK state.

Error number 1 to error number 3

Here, a maximum of three error numbers for errors detected on the interface are entered in the order in which they were detected by the system.

Meaning of the error numbers

The meaning of the error numbers and further information about dealing with interface errors can be found in the communication manual (Further Reading /14/).

5.6.12 OB 36: Error in Self-test

OB 36 is called if one of the self-test routines detects an error when it is run (for detailed information, refer to Section 5.7).

5.7 Self-Test

5.7.1 Overview

The CPU 948 contains integrated self-test routines in the system program.

Activating/deactivating

You can activate or deactivate the functions of the self-test using bits in system data RS 137.

Time slice

To reduce the cycle load caused by the self-test in the RUN mode, only part of the self-test is carried out within a cycle (time slice). The time available for the self-test can be set in RS 136 (refer to Section 5.7.3).

What can be tested?

The self-test routines can carry out the following tests:

WHAT IS TESTED?	WHEN?
The user memory	During OVERALL RESET
The BASP signal (disable command output)	In the STOP mode
The hardware clock	During COLD RESTART
The cycle time monitoring	During START-UP
The address lines	Cyclically in the RUN mode
The code of the system program (checksum)	Cyclically in the RUN mode
The code of the STEP 5 logic blocks in the user memory (checksum)	Cyclically in the RUN mode

5.7.2 Description of the Test Functions

Testing the user memory

(During OVERALL RESET, without time slice)

The user memory is tested during an OVERALL RESET. This test checks the user memory, the byte areas, the flags and process images.

During the test, the whole area (including the byte areas) are written with a test pattern and then checked to make sure that they match. At the end of the test, the area is written with zeros.

Note
The user memory test takes time to complete

- CPU 948-1 (640 Kbytes)	approx. 5 seconds
- CPU 948-2 (1 664 Kbytes)	approx. 22 seconds

Testing the BASP signal

(In the STOP mode, without time slice)

This test checks whether a BASP signal is output by the CPU. The test function runs in the stop loop. The BASP signal is then read cyclically. If an error is detected, an entry is made in the error buffer. At the next START-UP, OB 36 (error in self-test) is called, if it exists. If OB 36 is loaded and contains an STP operation, the START-UP is aborted. Otherwise, the CPU changes to the cyclic mode.

Testing the hardware clock

(During START-UP, however, only in CPU **COLD RESTART**; without time slice)

This test is made before OB 20 is called and takes one second.

The current time is retained; existing timed jobs (clock-controlled interrupts - OB 9) on the other hand are deleted.

Testing cycle time monitoring

(During START-UP, without time slice)

With this function, the cycle time monitoring is checked during the start-up phase. The cycle monitoring time is set to the minimum value (20 ms) and then a program loop started until the cycle error occurs.

Testing the address lines

(Cyclically in RUN mode, with time slice)

In this test, wire breaks and short circuits on the address lines are detected by writing test patterns via the lines, reading them back and comparing them.

Testing the system program code

(Cyclically in the RUN mode, with time slice)

In the operating code test, the content of the CPU operating system in the internal RAM is checked (test area D 0000H to E 7FFDH).

The test is made by adding the content of the test area and then comparing this with the checksum in the EPROM.

Testing the block code of STEP 5 logic blocks

(Cyclically in the RUN mode, with time slice)

The checksum of each valid STEP 5 logic block is checked.
If a memory card is inserted, the checksum of the logic blocks of the CPU 948 is created following an OVERALL RESET and after copying the memory card contents to the internal user memory. Logic blocks added at a later point in time are also checked.

Note
In the code test of the STEP 5 blocks, an error is detected if one or more logic blocks were modified dynamically.
It is possible to modify blocks with the PG. If this is the case, the checksum is created by the system program of the CPU 948.

5.7.3 Settings

Calculating and setting the number of time slices

The processing time for the self-functions is distributed on time slices which are called once per cycle. The number of time slices can be selected. This means that you can increase the time required for the self-test functions per cycle.

Calculating the number

First, you must estimate the time you can leave for the self-test within the cycle: the length of a time slice is approximately 500 µs, i.e. the self-test requires 500 µs in each cycle.

Once you have estimated how much time is available, you can calculate the number of time slices each taking approximately 500 µs.

Setting the number

You can set the number of time slices in system data word **RS 136** (16-bits wide). The default is **1** time slice (minimum value). You can set up to a maximum of 10 time slices (5 ms required in the cycle).

The number of time slices is derived from the value of system data word RS 136 as follows:

RS 136 = 0 or 1:	1 time slice
RS 136 = 2:	2 time slices
RS 136 = 3:	3 time slices
etc.	

Activating/deactivating the tests

You can activate the individual tests, e.g. in a start-up block, by setting the corresponding bits in RS 137 to ’1’ and deactivate the tests by setting the bits to ’0’.

Note
After an OVERALL RESET on a newly inserted CPU, **all** the test functions are **switched off**.
The next time you make an OVERALL RESET, only the test functions to be run in the **OVERALL RESET** are **activated**. All other test functions are deactivated.
This means that you can only check the user memory of a newly inserted CPU by activating the test function in RS 137 following an OVERALL RESET and then repeating the OVERALL RESET.

Assignment of system data word RS 137

Test function	Bit no.
Check the code of the system program	2
Check the code of the STEP 5 logic blocks in the user memory	5
Check the address lines	7
Check the clock	10
Check the BASP signal	11
Check the cycle time monitoring	13
Test the user memory	15

The bit numbers not listed in the table are not used.

5.7.4 Error Handling

Providing the test function is not running within an OVERALL RESET, the test function calls error OB 36 (refer to Section 5.6.11) if an error is detected and transfers the content of RS 137 containing the bits of the activated test routines to ACCU 1.

All the test routines also enter information about the type of test and error detected in the system data words RS 75 to RS 78.

For test components which only run in an OVERALL RESET, the cause of the error is indicated in RS 75. **The STOP LED then flashes quickly, when the tests are completed with an error in an OVERALL RESET**.

Errors detected by the self-test component "BASP signal" in the STOP mode are also indicated in RS 75. The following START-UP only leads to cyclic operation if **no STP operation** is programmed in OB 36.

Error information

Testing the user memory

System data word	Error information
RS 75	error no. 640CH testing the word memory error no. 650CH testing the byte memory
RS 76	test pattern in which the error occurred
RS 77	incorrect address, high
RS 78	incorrect address, low

Testing the BASP signal

System data word	Error information
RS 75	error no. 6700H
RS 76	FFFFH
RS 77	FFFFH
RS 78	FFFFH

Testing the hardware clock

System data word	Error information
RS 75	error no. 6800H
RS 76	FFFFH
RS 77	FFFFH
RS 78	FFFFH

Testing cycle time monitoring

System data word	Error information.
RS 75	error no. 6600H
RS 76	FFFFH
RS 77	FFFFH
RS 78	FFFFH

Testing the address lines

System data word	Error information
RS 75	error no. 630BH
RS 76	FFFFH
RS 77	incorrect address, high
RS 78	incorrect address, low

Testing the system program code

System data word	Error information
RS 75	error no. 610BH
RS 76	FFFFH
RS 77	actual checksum, high
RS 78	actual checksum, low

Testing the block code of STEP 5 logic blocks

System data word	Error information
RS 75	error no. 620AH
RS 76	block type/block no. (IDs from block header)
RS 77	expected checksum
RS 78	actual checksum

Integrated Special Functions 6

Contents of Chapter 6

Integrated Special Functions 6

The following chapter describes the special functions integrated in the system program, where you can use these functions and how to call and assign parameters to the special function OBs.
You will also learn how to recognize errors in the execution of a special function and possible ways of handling them in the program.

6.1 Introduction

The operating system of the CPU 948 provides you with special functions which you can call if necessary with a conditional (JC OB x) or an unconditional (JU OB x) block call. Organization blocks OB 100 to 255 are reserved for these special functions.

These functions are known as integrated special functions, since they are a fixed part of the system program. As user, you can call these special functions but cannot read or modify the corresponding program.

The following table provides an overview of the existing special functions.

Table 6-1 Overview of the special functions available with the CPU 948

Block	Function	Refer to section/page
OB 121	Set/read system time (compatible with CPU 946/947)	6.2/6 - 8
OB 122	"Disable interrupts" on/off	6.3/6 - 12
OB 124 OB 125	Delete STEP 5 blocks Generate STEP 5 blocks	6.4/6 - 14 6.5/6 - 17
OB 126	Define/transfer process images	6.6/6 - 20
OB 129	Battery state	6.7/6 - 25
OB 131	Delete ACCU 1, 2, 3 and 4	6.8/6 - 26
OB 132 OB 133	Roll-up ACCU Roll-down ACCU	6.9/6- 27 6.9/6- 27
OB 141 OB 142 OB 143	"Disable single cyclic timed interrupts" on/off "Delay all interrupts" on/off "Delay single cyclic timed interrupts" on/off	6.10/6 - 29 6.11/6 - 32 6.12/6 - 35
OB 150 OB 151 OB 153	Set/read system time Set/read time for clock-controlled interrupt Set/read time for delayed interrupt	6.13/6 - 38 6.14/6 - 43 6.15/6 - 50
OB 181	Test data block	6.16/6 - 53
OB 182	Copy data area	6.17/6 - 55
OB 200, 202 203, 204, 205	Functions for multiprocessor communication	6.18/6 - 58
OB 222	Restart cycle monitoring time	6.19/6 - 59
OB 223	Compare start-up modes in multiprocessor mode	6.20/6 - 60
OB 254, 255	Copy/duplicate DB and DX data blocks	6.21/6 - 61

Interfaces

The following are available as interfaces to the special functions:

Block call

- Conditional/unconditional block call JC .. / JU ..

Parameters

- Parameters for defaults via ACCU 1 and possibly ACCU 2 and/or memory locations

  In the following description of the individual special functions, all the data required by the CPU to execute the special function correctly are listed under the term **parameters**. Before calling the special function in the STEP 5 program, you must load this data in the accumulators or in the specified memory locations.

Writing to ACCUs

When assigning parameters for the special function organization blocks, please note the following conventions for writing to the ACCUs:

ACCU 1:	ACCU 1,	**32 bit**s
ACCU-1-L:	ACCU 1, low word,	**16 bit**s
ACCU-1-LL:	ACCU 1, low word, low byte,	**8 bit**s
ACCU-1-LH:	ACCU 1, low word, high-byte,	**8 bit**s

High word		Low word	
High byte	Low byte	High byte	Low byte
31 24	23 16	15 8	7 0

Error handling

An error occurring in the execution of the active special function triggers a special error reaction in the system program.

In terms of this error reaction by the system program, two groups of special functions can be distinguished.

- **Group 1:**

  ACCU condition code bits

  Group 1 includes all the special functions with which IDs are transferred to ACCU 1 if an error occurs to further explain the error.

- **Group 2:**

  RLO ,
  CC0/CC1

  With some of the special functions, the RLO or the status bits CC0/CC1 are written to indicate errors for specific special functions.
  If an error occurs when using one of these special functions, the RLO is set to "1" in most cases. In your STEP 5 program, you can use a JC operation (conditional jump) to evaluate the RLO for these special functions and then react to an error.

  In some special functions, the results bits CC0 and CC1 are affected by the processing of a special function. You can scan these bits in your STEP 5 program using a compare operation and once again program a reaction to an error.

Which of the error reactions occurs for the individual special function OBs is explained in the following sections.

Note
Calling a special function OB using the "JU OB 131/132/133" or "JC OB 131/132/133" operation does not have the same effect as a "genuine" block change, but functions as a STEP 5 operation without block operand. **No interrupts** are nested (with the default "interrupts at block boundaries").

6.2 OB 121: Set/Read System Time

Function

With OB121 you can set or read the system time (date and time). This function is compatible with the CPU 946/947.

Parameters

1. Data field

Four words in the word-oriented memory area.
With the **"set system time"** function the area **before** the OB121 call must be loaded with the time values to be set. The system program checks these values to ensure they are logically correct.
With the **"read system time"** function, OB121 enters the current time values in this area.

Structure of the data field

Bit no.	15 12	11 9	8 4	3 0
1st word	Sec. x 10	Sec. x 1	10th Sec.	100th Sec.
2ndword	Hour x 10	Hour x 1	Min. x 10	Min x 1
3rd word	Day x 10	Day x 1	Weekday	0
4th word	Year x 10	Year x 1	Month x 10	Month x 1

> **Note**
> The structure of the data field corresponds to the structure of system data RS 96 to RS 99 (current time).

<u>Possible time values:</u>

100th seconds:	0 to 9
10th seconds:	0 to 9
Seconds x 1:	0 to 9
Seconds x 10:	0 to 5
Minutes x 1:	0 to 9
Minutes x 10:	0 to 5
Hours x 1:	0 to 9
Hours x 10:	
Bit no. 12 /13:	0 to 1 with 12 hour format 0 to 2 with 24 hour format
Bit no. 14:	0 = AM with 12 hour format 1 = PM " " 0 in 24 hour format
Bit no. 15:	0 = 12 hour format 1 = 24 hour format

Weekday:	0 to 6 for Mon to Sun
Days x 1:	0 to 9
Days x 10:	0 to 3
Month x 1:	0 to 9
Month x 10:	0 to 1
Year x 1:	0 to 9
Year x 10:	0 to 9

2. BR register (base address register)

Start address of a data field in an area of memory organized in words, from which the time values to be set are read or in which the current time values are to be stored. The BR register must be loaded with the address **before** the OB 121 call.

3. ACCU-1-L

Function no.,	
Permitted values:	1 = set system time
	2 = read system time

Result

After correct and error-free processing, the system program enters the value ’0’ in ACCU-1-L. After the "set" function, the system time is set to the values contained in the data field; after the "read" function, the data field contains the current time values.

Possible errors

The errors listed in the following table can occur. If one of these errors does occur, the system program enters the error ID shown in the table in ACCU-1-L.

Table 6-2 Error IDs of OB 121 in ACCU-1-L

ID	Meaning
F001H	Illegal function no.
F00FH	Multiple block call
F101H	Year specification illegal
F102H	Month specification illegal
F103H	Day specification illegal
F104H	Weekday specification illegal
F105H	Hour specification illegal
F106H	Minute specification illegal
F107H	Seconds specification illegal
F108H	100th to 10th seconds in the data field unequal to 0. Entries from the 100th to the 10th second must equal 0.
F109H	Hour format differs from setting in OB 151

Examples

Programming example for "set system time"

FB 13 is programmed for the "set system time" function. The new values are transferred in data block DB 10 (data word DW 0 to DW 3).

STEP 5 program:

```
   FB13
   NAME    :CLKWR
           :C   DB 10     Open DB 10
           :L   KH 1500   15 seconds (10th .. 100th sec. = 0!)
           :T   DW 0
           :L   KH 9555   24 hour mode, 15 hours 55 minutes
           :T   DW 1
           :L   KH 1010   the 10th, Tuesday
           :T   DW 2
           :L   KH 9308   1993, August
           :T   DW 3
           :MBR EEC00     Load start address of the DB list
           :              in the BR
           :LRW +10       Load the start address of DB 10 in memory
           :              (paragraph address) in ACCU 1
           :SLD 4         Absolute address of DB 10 (DW 0)
           :MAB           Load content of ACCU 1 into BR register
           :L   KB 1      Load function no. ’1’ in ACCU-1-L
           :JU  OB 121    Set system time
           :L   KB 0
           :><F           Scan error bits
           :JC  =ERRO     Jump to error handling
           :BEU
           :
   ERRO    :              Error handling
           :
           :BE
```

Data block DB 10 contains the following information when OB 121 is called:

```
   0: KH = 1500;
   1: KH = 9555;
   2: KH = 1010;
   3: KH = 9308;
   4:
```

OB 121 transfers the required time parameters from DB 10 to the system data area RS 96 to RS 99.

Programming example for "read system time"

FB 14 is programmed for the "read system time" function. The current values should be stored in data block DB 11 (data word DW 0 to DW 3).

STEP 5 program:

```
   FB14
   NAME    :CLKRD
           :MBR EEC00      Load start address of the DB list
           :               in BR
           :LRW +11        Load start address of DB 11 in memory
           :               (paragraph address) in ACCU 1
           :L   KB  0
           :!=F            Check that OB 11 is loaded
           :JC  =NIVO      Jump to error handling
           :               if DB start address = 0
           :TAK
           :SLD 4          Absolute address of DB 11 (DW 0)
           :MAB            Load contents of ACCU 1 into BR register
           :L   KB 2       Load function no. ’2’ in ACCU-1-L
           :JU  OB 121     Read system time
           :BEU
           :
   NIVO    :               Error handling
           :
           :BE
```

Data block DB 11 contains the following information after OB 121 is called (example):

```
   0: KH = 2994;           29 sec., 940 millisec.
   1: KH = 9555;           24 hour format, 15 hours, 55 minutes
   2: KH = 1010;           10 days, weekday ’1’(Tuesday), 0
   3: KH = 9308;           93 years, 8 months
   4:
```

It is Tuesday the 10th of August 1993, 15 hours, 57 minutes, 29 seconds and 940 milliseconds (9 tenths and 4 hundredths of a second).

6.3 OB 122: "Disable Interrupts" On/Off

A STEP 5 program can be interrupted at block boundaries or operation boundaries by programs at an execution level with a higher priority. The program execution levels with higher priority include the following:

- TIMED INTERRUPTS

  and

- PROCESS INTERRUPTS.

The time required to run the interrupted programs is expanded by the run time of the nested programs.

Function

Using OB 122, you can prevent interrupt servicing blocks being nested at one or more consecutive block or operation boundaries.

OB 122 influences the acceptance of interrupts:

- "Disable interrupts" on means the following:
  no interrupts will be registered from now on.

- "Disable interrupts" off means the following:
  all interrupts occurring will be registered from now on and the corresponding blocks will be nested and executed at the next operation or block boundary (depending on the mode set in DX 0).

Interrupts that have already been registered, are then no longer serviced if the mode "interruptability at block boundaries" is set in DX 0.

Parameters

ACCU-1-L

Function no.,	
Permitted values:	1 = disable all interrupts
	2 = enable all interrupts

Result

After correct and error-free processing, the system program enters the value ’0’ in ACCU-1-L.

> **Note**
> By calling OB 122, the RLO (undefined) is influenced. The BR register is not modified.
> To disable and enable processing interrupts, you can also use the STEP 5 operations IA and RA instead of OB 122. The blocking of interrupts is cleared at the next system checkpoint (refer to Chapter 11).

Possible errors

The errors listed in the following table can occur. If one of these error does occur, the system program writes the error ID shown in the table into ACCU-1-L.

Table 6-3 Error IDs of OB 122 in ACCU-1-L

ID	Meaning
F001H	Illegal function number

Example

```
Excerpt of a STEP 5 program in which all
interrupts are disabled using OB 122 immediately
before a critical program section following which
they are enabled again:

   :L   KB 1      Load function ID in ACCU-1-L
   :JU  OB 122    Disable all interrupts
   :            )
   :            )
   :            )
   :            } Critical program section
   :            )
   :            )
   :L   KB 2      Load function ID in ACCU-1-L
   :JU  OB 122    Enable all interrupts
   :
   :
```

6.4 OB 124: Delete STEP 5 Blocks

Function

With OB 24, you can delete any STEP 5 blocks (logic and data blocks) in the user memory. The deleted block is removed from the address list in DB 0.
The gap in memory resulting from deleting a block is used again when new blocks are loaded.

Parameters

1. ACCU-1-LH

Block **type** of the block to be deleted

2. ACCU-1-LL

Block **number** of the block to be deleted

Permitted block types and numbers:

ACCU-1-LH (block type)	ACCU-1-LL (block number)
1 = PB	0 to 255
2 = SB	0 to 255
3 = FB	0 to 255
4 = FX	0 to 255
5 = DB	3 to 255
6 = DX	3 to 255
7 = OB	1 to 39

Result

After correct and error-free processing, the system program sets the RLO to ’0’ and clears the condition codes CC 1 and CC 0.

Note
While the blocks are actually being deleted, user interrupts are disabled: no interrupts come through.

By calling OB 124, the contents of ACCU 1 to ACCU 4 are modified. The BR register is retained.

Possible errors and warnings

If an error or warning occurs, the system program stops processing OB 124 and continues program execution at the next STEP 5 operation. It also sets the RLO to ’1’ and writes an ID to ACCU-1-LL (refer to Table 6-5).

If the function is aborted with a warning, it may be possible to achieve correct execution of OB 124 by re-calling the special function (possibly several times).

In the following case, OB 124 is aborted with a **warning**:

During the last 10 ms OB 124, OB 125, OB 254 or OB 255 has been called. (Only one call for these special functions is allowed within 10 ms. This avoids multiple calls for the OBs listed above blocking the interface to the PG so that it can no longer be processed.)

Condition code bits

After calling OB 124, you can check whether the special function has been executed correctly or was aborted with an "error" or "warning" using the result of logic operation and the condition code bits CC 1 and CC 0. The result can be evaluated with conditional jump operations.

Results bits

Table 6-4 Results bits of OB 124

RLO	CC 1	CC 0	Meaning	Scan
0	0	0	Special function was processed correctly	JC JZ
1	1	0	Processing of special function aborted with "warning"	JC JP JN
1	0	1	Processing of special function aborted with "error"	JC JM JN

IDs in ACCU-1-LL

In ACCU-1-LL, the system program stores IDs about the processing result, with which the cause of a warning or error is specified in more detail.

Bit no.	7	6	5	0
	W	E	Cause of error/warning	

The following group bits are fixed:

Bit no. 7 (W) = 1:	warning
Bit no. 6 (E) = 1:	error

Table 6-5 Result IDs of OB 124 in ACCU-1-LL

ID	Meaning
01H	Function was correctly processed
 45H 47H 4DH	Error: Block type not permitted Block does not exist Online function COMPRESS MEMORY active
 8DH 8EH	Warning: Conflict with an online function (except for "compress memory") 10 ms waiting time not elapsed

Example

```
:L   KY 6,100   This sequence of operations deletes
:JU  OB 124     data block DX 100 in the user
:               memory
```

6.5 OB 125: Generate STEP 5 Blocks

Function

With OB 125, you can generate any STEP 5 blocks (logic and data blocks) in the user memory. Generating logic blocks should, however, be left to specialists.

The specified block is set up in the internal RAM with a block header and block body and entered in DB 0. The block body contains random data. For this reason, a newly generated block must first be written to before any useful data can be read out of it.

Parameters

1. ACCU-1-LH

Block **type** of the block to be generated

2. ACCU-1-LL

Block **number** of the block to be generated

Permitted block types and numbers

ACCU-1-LH (block type)	ACCU-1-LL (block number)
1 = PB	0 to 255
2 = SB	0 to 255
3 = FB	0 to 255
4 = FX	0 to 255
5 = DB	3 to 255
6 = DX	3 to 255
7 = OB	1 to 39

3. ACCU-2-L

Number of words (required block length without block header). The maximum assignable block length is 32762 data words. At present, approximately 2 K words can be edited with a PG.

Result

After correct and error-free processing, the system program sets the RLO to ’0’ and clears the condition codes CC 1 and CC 0.

Note
While the blocks are actually being generated, user interrupts are disabled: no interrupts come through.

By calling OB 125, the contents of ACCU 1 to ACCU 4 are modified. The BR register is retained.

Possible errors and warnings

If an **error** occurs, the system program stops processing OB 125 and continues program execution at the next STEP 5 operation. It also sets the RLO to ’1’ and writes an ID to ACCU-1-LL (refer to Table 6-7).

If the function is aborted with a warning, it may be possible to achieve correct execution of OB 125 by re-calling the special function (possibly several times).

In the following case, OB 125 is aborted with a **warning**:

During the last 10 ms OB 124, OB 125, OB 254 or OB 255 has been called. (Only one call for these special functions is allowed within 10 ms. This avoids multiple calls for the OBs listed above blocking the interface to the PG so that it can no longer be processed.)

Condition code bits

After calling OB 125, you can check whether the special function has been executed correctly or was aborted with an "error" or "warning" using the result of logic operation and the condition code bits CC 1 and CC 0. The result can be evaluated with conditional jump operations.

Result bits

Table 6-6 Result bits of OB 125

RLO	CC 1	CC 0	Meaning	Scan
0	0	0	Special function was processed correctly	JC JZ
1	1	0	Processing of special function aborted with "warning"	JC JP JN
1	0	1	Processing of special function aborted with "error"	JC JM JN

IDs in ACCU-1LL

In ACCU-1-LL, the system program stores IDs about the processing result, with which the cause of a warning or error is specified in more detail.

Bit no.	7	6	5	0
	W	E	Cause of error/warning	

The following group bits are fixed:

Bit no. 7 (W) = 1:	warning
Bit no. 6 (E) = 1:	error

Table 6-7 Result IDs of OB 125 in ACCU-1-LL

ID	Meaning
01H	Function correctly processed
	Errors:
42H	Block already exists
43H	Not enough memory
44H	Block length not permitted
45H	Block type not permitted
4DH	Online function COMPRESS MEMORY active
	Warnings:
8DH	Conflict with an online function (except for "compress memory")
8EH	10 ms waiting time not yet elapsed

Example

```
:L   KF +2000   This sequence of operations
:L   KY 5,24    generates DB 24 with a length of
:JU  OB 125     2000 data words
:               (total length including header:
:               2005 words)
```

6.6 OB 126: Define, Transfer Process Images

Each time the cycle is run through, the system program updates the process image of the digital inputs and outputs and IPC flags. The inputs, outputs and IPC flags included in the process image are stored in system data block DB 1 (refer to Chapter 10).
With OB 126, you can use additional process images.

Function

Using OB 126, you can program up to four further process images in addition to the process image in DB 1 during a **COLD RESTART**. These additional process images can be read in and output with the STEP 5 program at **any program execution level**.

Parameters

1. Data field

6 flags with the following structure:

Bit no.	7	0
FY n	Function no.	
FY n+1	Address list no.	
FY n+2	Block type	
FY n+3	Block number	
FY n+4	Data word no. of the first ID word	
FY n+5	in the address list	

Parameters of the data field:

Function no.

With the function number, you stipulate which job OB 126 is to perform (refer to the table).

Permitted values: 1 to 5

Function no.	Function
1 2 3 4	Read in the process image of the digital inputs Output the process image of the digital outputs Read in the process image of the IPC input flags Output the process image of the IPC output flags
5	Set up system internal address list (analogous to DB 1) **(only permitted in COLD RESTART OB 20)**

Address list number

Number of the address list for the additionally defined process image;

permitted values: 1 to 4

Block type

Type of data block containing the address list;

permitted values: 1 = DB
2 = DX

Block number

Number of the data block containing the address list;

permitted values: 3 to 255

DW no. 1st ID word

Here, you enter the number of the data word in which the **first** ID word of the address list is located (refer to structure of DB 1, Section 10.1.6).
The following ID words are possible:

KH = DE00 (digital inputs)
KH = DA00 (digital outputs)
KH = CE00 (IPC input flags)
KH = CA00 (IPC output flags)

The parameter occupies **2** flag bytes!

Note
The complete data field only needs to be set up when OB 26 is to generate the address list during a COLD RESTART (= function 5). To execute functions 1 to 4 it is adequate to simply enter the address list number alongside the function number in the data field. The remaining entries are then not needed.

You must structure the data block with which you want to set up the address list for an additional process image (= function 5) analogous to DB 1.
You can store the address list information for each of the additional process images in this data block by adding an end ID to each field of information as in DB 1.
To set up the address list, you must however call OB 126 for each additional process image using function "5" singly (only in COLD RESTART).

2. ACCU-1-L

No. of the flag byte **FY n**, at which the data field begins

permitted values: 0 to 250

Result

After correct and error-free processing, the system program sets the RLO to ’0’ and enters a ’1’ in ACCU-1-LL.

Note
When processing OB 126, user interrupts are disabled: no interrupts come through.

Calling OB 126 changes the contents of ACCU 1 to ACCU 4.
The BR register is retained.

During a WARM RESTART the remaining cycle is processed with BASP activated. All the digital outputs are disabled. At the end of the cycle, **all** outputs (also those in address lists 1 to 4) are reset.

Possible errors

If the special function cannot be executed, the system program interrupts processing of OB 126 and continues program execution with the next STEP 5 operation. It also sets the RLO to ’1’ and writes an ID to ACCU-1-LL (refer to the following table).

Special situation when handling errors:

If OB 126 is to execute function ’5’ (set up system internal address list), the system program checks the correct structure of the address list. It also checks whether the inputs and outputs or IPC flags contained in the address list acknowledge the corresponding modules. If an incorrect address list has been transferred, the CPU reacts in the same way as to a DB 1 error. It changes to the soft stop state and the STOP LED flashes slowly. A DB 1 error is indicated as the cause of the error.

IDs in ACCU-1-L

Table 6-8 Result IDs of OB 125 in ACCU-1-LL

ID	Meaning
01H	Function correctly processed
02H	Function number illegal
03H	Pointer in ACCU-1-L (flag number) illegal
04H	Block type/number illegal or DB/DX block does not exist
05H	The first ID word is not located in the specified data word of the data block (wrong DW number) or the address list contains an incorrect ID word
06H	Address list number illegal
07H	The call for the function is not permitted at the current program execution level

Examples

Creating the address list in DB 5

Using the function keys <input>, <scr form>, "block: DB 5" program a data block DB 5 on the PG with the following parameters:

```
Digital inputs:           1, 2,
Digital outputs:          3,
IPC input flags:          5, 6, 7,
IPC output flags:         20, 22,
```

If you create DB 5 manually, it must be structured like a DB 1 (with start ID, ID words and operand areas, end ID; refer to Section 10.1.6).

Entering the address list in the COLD RESTART/OB 20:

First, you must set up the data field in the flag area. This occupies flag bytes FY 20 to FY 25:

```
   :L   KB 5    Transfer function no. ’5’
   :T   FY 20   to FY 20
   :L   KB 1    Transfer address list no. ’1 ’
   :T   FY 21   to FY 21
   :L   KH 0105 Transfer block type DB (’1’) and
   :T   FW 22   number ’5’ to FY 22 and FY 23
   :L   KB 3    Trans. DW no. ’3’ (DW 3 in DB 5
   :T   FW 24   contains 1st ID word) to FY 24 & 25
```

Once the data field has been correctly set up, the number of the first flag byte in the data field must be transferred to ACCU-1-L. Following this, OB 126 is called which sets up the address list:

```
   :L   KB 20   Data field begins with FY 20
   :JU  OB 126  Call for address list generation
   :            poss. evaluation of status bits
```

Note:

The address lists with numbers 1 to 4 are only accepted by the CPU using an OB 126 call in OB 20 (COLD RESTART). To do this, OB 126 must be called with function number ’5’ in OB 20.

Output the process image of the outputs

The following STEP 5 program sequence can be located in any program execution level (in OB 1, in a timed interrupt OB or in a process interrupt OB etc.) and causes the process image of all outputs in address list 1 to be output.

```
   :L   KB 2    Transfer function no. ’2’
   :T   FY 50   to FY 50
   :L   KB 1    Transfer address list no. ’1’
   :T   FY 51   to FY 51
   :L   KB 50   Data field begins with FY 50
   :JU  OB 126  Call for outputting the PIQ
   :            possibly evaluation of status bits
   :            ...
```

6.7 OB 129: Battery State

Function

With OB 129, you can check the state of the back-up battery with a STEP 5 program (OB 129 scans the BAU signal). Depending on the result, you could, for example, set a fault indicator (lamp).

How the BAU signal is formed

The power supply contains two back-up batteries, a lithium cell (MB for main battery) and an accumulator (RB for reserve battery). The BAU signal is formed by ANDing the battery monitoring signals.

Jumper setting in the power supply

In the power supply of the PLC, there are two jumpers with which the monitoring of the batteries can be influenced. Jumper MB-NB determines when the BAU signal is generated (if this jumper is not inserted, BAU is generated once following POWER UP). Otherwise the signal is monitored cyclically during operation. The monitoring signal of the accumulator can be disabled with the jumper MA-NA. The possible combinations of jumper settings and the resulting battery monitoring by OB 129 are illustrated in the following table:

Jumper MB-NB	Jumper MA-NA	Time when BAU signal generated	BAU signal generated from monitoring signal of ...
open	open	after POWER UP	lithium cell
open	closed	after POWER UP	lithium cell and accu
closed	open	cyclic	lithium cell
closed	closed	cyclic	lithium cell and accu

Parameters

none

Result

RLO = ’0’: battery OK

RLO = ’1’: battery run down

Example

```
With the following sequence of operations, you
can check whether or not the battery is OK and if
it is not, you can energize a lamp:


     :
     :JU   OB 129
     :JC   =BATL      RLO = 1 -> battery run down
     :
     :
     :BEU
BATL :SU   Q  22.5    switch on warning lamp at
     :                output byte 22, bit 5
     :BE
```

6.8 OB 131: Delete ACCUs 1, 2, 3 and 4

Function

By calling special function organization block OB 131 once, you can delete the contents of ACCUs 1 to 4 extremely simply. OB 131 overwrites all four registers with ’0’.

Parameters

None

Result

The ACCUs 1 to 4 (each 32 bit) are deleted (’0’).

Possible errors

None

6.9 OB 132/133: Roll-Up ACCU/Roll-Down ACCU

Function

OB 132 and OB 133 roll the ACCU contents up or down:

- OB 132 (roll up) moves the contents of ACCU 4 to ACCU1, the contents of ACCU 1 to ACCU 2, the contents of ACCU 2 to ACCU 3 etc.
- OB 133 (roll down) moves the contents of the ACCUs in the opposite direction: the contents of ACCU 1 to ACCU 4, ACCU 4 to ACCU 3 etc.

Parameters

None

Result

Figs. 6-1 and 6-2 show the ACCU contents **before** and **after** calling OB 132 and OB 133.

> **Note**
> With the STEP 5 operations ENT (extended operation set) and TAK (system operation) the ACCU contents can also be shifted (refer to Section 3.4.3).

Possible errors

None

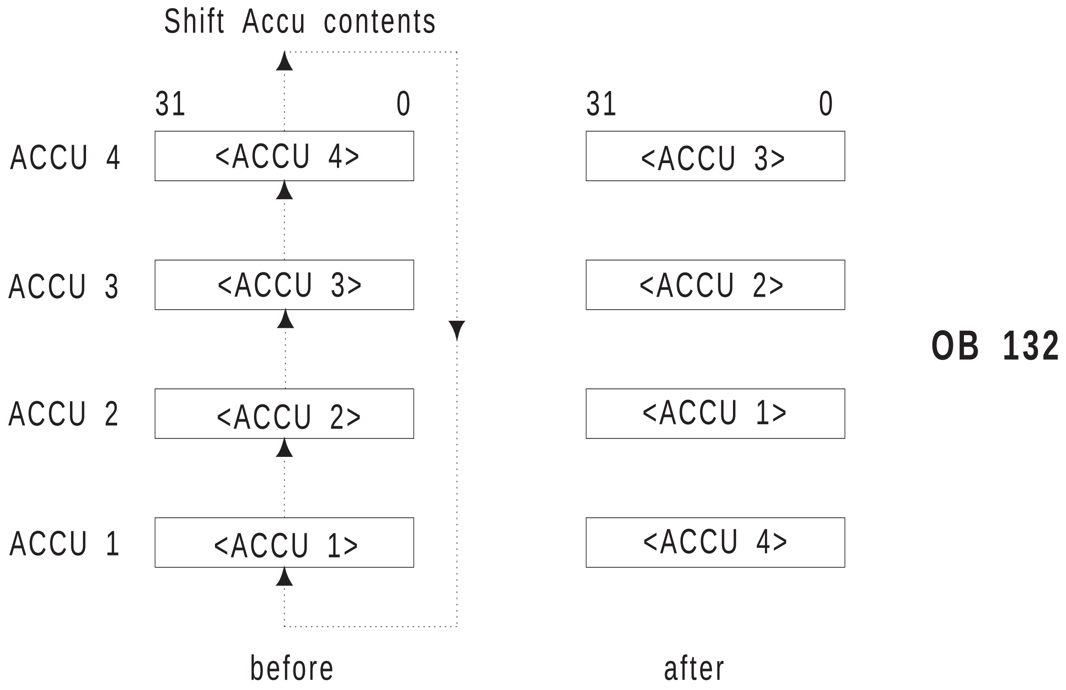


Fig. 6-1 Effect of the "roll-up" function

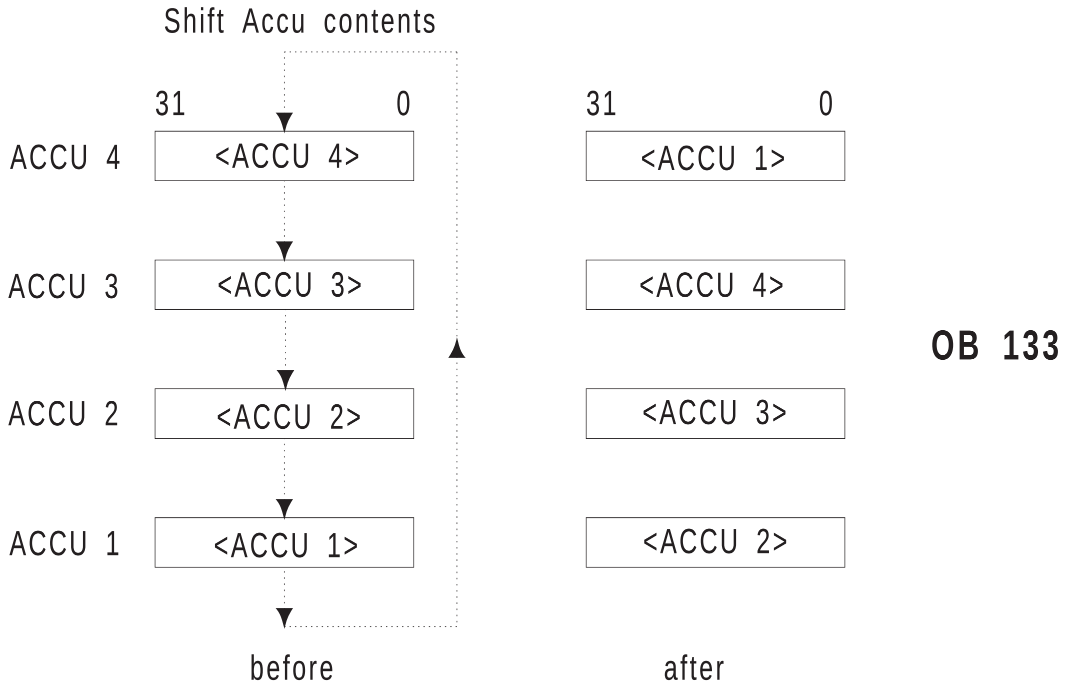


Fig. 6-2 Effect of the "roll-down" function

6.10 OB 141: "Disable Single Cyclic Timed Interrupts" On/Off

Using OB 141, you can prevent certain cyclic timed interrupt OBs (timed interrupts at **fixed intervals**) from being called at one or more consecutive block or operation boundaries. For example, you can prevent an OB 10 (period 1) and an OB 11 (period 2) from being called in a particular program section that must not be interrupted. On the other hand, all remaining programmed timed interrupts are processed as usual.

Function

OB 141 affects the reaction to cyclic timed interrupts.

"Disable single cyclic timed interrupts" **on** means that none of the specified cyclic timed interrupts are registered from this point onwards and the interrupts that were already registered (e.g. those waiting for a block boundary) are deleted. If a timed interrupt OB (for processing a timed interrupt with a fixed interval) has already been started, it is processed completely.

"Disable single cyclic timed interrupts" **off** means that all cyclic timed interrupts are registered again and are processed at the next block or operation boundary (depending on the setting in DX 0).

Parameters

1. Control word

OB 141 records the timed interrupts to be disabled in a system-internal control word:

Bit no.	15	0
	Control word	

The bits of the control word have the following significance:

Bit no.	**Interrupt**
0 to 2	Reserved, these bits must be ’0’
 3 = ’1’ 4 = ’1’ 5 = ’1’ 6 = ’1’ 7 = ’1’ 8 = ’1’ 9 = ’1’ 10 = ’1’ 11 = ’1’	Cyclic timed interrupts with fixed interval: period 1 (OB 10) period 2 (OB 11) period 3 (OB 12) period 4 (OB 13) period 5 (OB 14) period 6 (OB 15) period 7 (OB 16) period 8 (OB 17) period 9 (OB 18)
12 to 15	Reserved; these bits must be ’0’

As long as a bit is set to ’1’ the corresponding interrupt is disabled.

2. ACCUs

2a) ACCU-2-L

Function no.,
Permitted values: 1, 2 or 3 where:

1: The contents of ACCU 1 are loaded in the control word.

2: All the bits marked ’1’ in the mask in ACCU 1 are set to ’1’ in the control word. The new control word is loaded in ACCU 1

3: All the bits marked ’1’ in the mask in ACCU 1 are set to ’0’ in the control word. The new control word is loaded in ACCU 1

2b) ACCU 1

New control word or mask depending on the required function.

Result

After correct and error-free processing the system program sets the RLO to ’0’.
Calling OB 141 has the following results:

Funct. no. in ACCU-2-L	**Contents of ACCU 1**	
	before	**after**
1	control word	control word
2	mask	new control word
3	mask	new control word

Possible errors

If an error occurs, the system program sets the RLO to ’1’.
The errors listed in the following table can occur. If an error occurs, the system program enters the error ID listed below in ACCU-1-L.

Table 6-9 Error IDs of OB 141 in ACCU-1-L

ID	Meaning
8D01H 8D02H	illegal function no. in ACCU-2-L [1)] one of the reserved bits in ACCU 1 is ’1’ [1)]

[1)] the incorrect value is located in ACCU-2-L

Scan control word

- The status of the control word can be scanned with the following program sequence:

  1. load function no. ’2’ or ’3’ in ACCU-2-L,

  2. load value ’0’ in ACCU 1,

  3. call OB 141,

  4. read out ACCU 1.

6.11 OB 142: "Delay All Interrupts" On/Off

A STEP 5 program can be interrupted at block or operation boundaries by programs with a higher priority. The process interrupts and all timed interrupts belong to these higher priority program execution levels. The runtime of the interrupted program is extended by the runtime of the nested programs. Using OB 142, you can prevent the nesting of higher priority program execution levels at one or more consecutive block or operation boundaries (depending on the setting in DX 0).

Function

OB 142 affects the servicing of interrupts:

"Delay interrupts" **on** means that all interrupts occurring are registered and pending interrupts remain registered. The registered interrupts are, however, initially not serviced. The operation or block boundaries for servicing interrupts are temporarily made ineffective. If an OB for process interrupt servicing or an OB for timed interrupt servicing has already started, this is processed completely.

"Delay interrupts" **off** means that all registered interrupts are processed at the next block or operation boundary.

> **Note**
> The time in which the interrupts are delayed must be shorter than **three times the value of the shortest timed interrupt period**. If this is not the case, a collision of timed interrupts occurs.

Parameters

1. Control word

OB 142 enters the interrupts to be delayed in a system-internal control word, as follows:

Bit no.	15	0
	Control word	

The bits in the control word have the following meaning:

Bit no.	**Type of interrupt**
0 = ’1’	Cyclic timed interrupts, fixed period
1 = ’1’	Clock-controlled interrupt
2 = ’1’	Process interrupts
3 = ’1’	Delayed interrupt
4 to 15	Reserved: these bits must be ’0’

As long as a bit is set to ’1’, the corresponding interrupt is disabled.

2. ACCUs

2a) ACCU-2-L

Function no.,
Permitted values: 1, 2 or 3 where:

1: The contents of ACCU 1 are loaded in the control word.

2: All the bits marked ’1’ in the mask in ACCU 1 are set to ’**1**’ in the control word. The new control word is loaded in ACCU 1

3: All the bits marked ’1’ in the mask in ACCU 1 are set to ’**0**’ in the control word. The new control word is loaded in ACCU 1

2b) ACCU 1

New control word or mask depending on the required function.

Result

After correct and error-free processing the system program sets the RLO to ’0’.
Calling OB 142 has the following results:

Funct. no. in ACCU-2-L	Contents of ACCU 1	
	before	after
1	control word	control word
2	mask	new control word
3	mask	new control word

Possible errors

If an error occurs, the system program sets the RLO to ’1’.
The errors listed in the following table can occur. If an error occurs, the system program enters the error ID listed below in ACCU-1-L.

Table 6-10 Error IDs of OB 142 in ACCU-1-L

ID	Meaning
8E01H	Illegal function no. in ACCU-2-L [1]
8E02H	One of the reserved bits (no. 4 to 15) in ACCU 1 is ’1’ [1]
8EFFH	Incorrect mode (e.g. when the delayed interrupt is to be disabled and DX 0 contains the parameter "process interrupts via IB 0 = on")

[1] the incorrect value is located in ACCU-2-L

Scan control word

The status of the control word can be scanned with the following program sequence:

1. load function no. ’2’ or ’3’ in ACCU-2-L,

2. load value ’0’ in ACCU 1,

3. call OB 142,

4. read out ACCU 1.

6.12 OB 143: "Delay Single Cyclic Timed Interrupts" On/Off

Using OB 143, you can prevent certain cyclic timed interrupt OBs (timed interrupt OBs with a fixed period) from being called at one or more consecutive block or operation boundaries. For example, you can select a program section which cannot be interrupted by an OB 10 (period 1) and an OB 11 (period 2). On the other hand, all the remaining programmed timed interrupts are processed as usual.

Function

OB 143 affects the processing of cyclic timed interrupts:
"Delay single cyclic timed interrupts" **on** means that all interrupts are registered and pending timed interrupts remain registered. The timed interrupts specified in the control word are, however, not processed immediately. Temporarily, all the operation and block boundaries for processing these timed interrupts are made ineffective. If a timed interrupt OB (for processing a timed interrupt with fixed period) has already started, it is processed completely.

"Delay single cyclic timed interrupts" **off** means that all registered interrupts are serviced at the next block or operation boundary (depending on the setting in DX 0).

Parameters

1. Control word

OB 143 enters the timed interrupts to be delayed in a system-internal control word:

Bit no.	15	0
	Control word	

The bits in the control word have the following meaning:

Bit no.	**Interrupt**
0 to 2	Reserved, these bits must be ’0’
	Cyclic timed interrupts with fixed period
3 = ’1’	period 1 (OB 10)
4 = ’1’	period 2 (OB 11)
5 = ’1’	period 3 (OB 12)
6 = ’1’	period 4 (OB 13)
7 = ’1’	period 5 (OB 14)
8 = ’1’	period 6 (OB 15)
9 = ’1’	period 7 (OB 16)
10 = ’1’	period 8 (OB 17)
11 = ’1’	period 9 (OB 18)
12 to 15	Reserved, these bits must be ’0’

As long as a bit is set to ’1’, the corresponding interrupt is disabled.

2. ACCUs

2a) ACCU-2-L

Function no.,
Permitted values: 1, 2 or 3 where:

1: The contents of ACCU 1 are loaded in the control word.

2: All the bits marked ’1’ in the mask in ACCU 1 are set to ’1’ in the control word. The new control word is loaded in ACCU 1

3: All the bits marked ’1’ in the mask in ACCU 1 are set to ’0’ in the control word. The new control word is loaded in ACCU 1

2b) ACCU 1

New control word or mask depending on the required function.

Result

After correct and error-free processing the system program sets the RLO to ’0’.
Calling OB 143 has the following results:

Funct. no. in ACCU-2-L	**Contents of ACCU 1**	
	before	**after**
1	control word	control word
2	mask	new control word
3	mask	new control word

Possible errors

If an error occurs, the system program sets the RLO to ’1’.
The errors listed in the following table can occur. If an error occurs, the system program enters the error ID listed below in ACCU-1-L.

Table 6-11 Error IDs of OB 143 in ACCU-1-L

ID	Meaning
8F01H	Illegal function no. in ACCU-2-L [1]
8F02H	One of the reserved bits in ACCU 1 is ’1’ [1]

1) the incorrect value is located in ACCU-2-L

Scan control word

- The status of the control word can be scanned with the following program sequence:

  1. load function no. ’2’ or ’3’ in ACCU-2-L,

  2. load value ’0’ in ACCU 1,

  3. call OB 143,

  4. read out ACCU 1.

6.13 OB 150: Set/Read System Time

Features of the system time

- The system time is backed up by the battery in the PLC rack. If the time is set, it therefore remains correct even following a power failure.
- The resolution is 10 ms for reading and 1 s for setting.
- Leap years are taken into account.
- Hours can be represented either using the 24 hour clock or the 12 hour clock "am" and "pm".
- The weekday is specified.
- Input and output is BCD coded.

Function

Using OB 150, you can set or read out the date and time of the CPU 948 in your user program. The date and time are known as the "system time".

Note
The system time is started (initially with a default value) after the CPU is plugged in.

Parameters

1. Data field for the time parameters

When **setting** the system time, OB 150 reads in the values to be set from a data field, when **reading** the time, OB 150 transfers the current time to the data field. You can set up this data field in a **data block** or in one of the two **flag areas** (F or S flags).

The data field consists of four words.

1a) Format of the data field for **setting** the system time.

Bit no.	15			12	11			8	7			4	3			0
word 1	Seconds								0							
word 2	Format		Hours						Minutes							
word 3	Day of month								Weekday				0			
word 4	Year								Month							

1b) Format of the data field when **reading** the system time

Bit no.	15			12	11			8	7			4	3			0
word 1	Seconds								1/100 seconds							
word 2	Format		Hours						Minutes							
word 3	Day of month								Weekday				0			
word 4	Year								Month							

The time parameters have the following meaning, range of values and representation:

Parameter	Permitted range of values	Value in
Seconds 1/100 seconds Minutes Hours Weekday Day of month[1] Month Year	0 to 59 0 to 99 (with "set system time" = 0 0 to 59 0 to 23 or 1 to 12, depending on the "format" 0 to 6 for Mon to Sun 1 to 31 1 to 12 0 to 99	BCD format
Format	Format for the hour field with the following meaning: Bit 15 = 0: 12 hour format ("am" or "pm" selected in bit 14) Bit 15 = 1: 24 hour format (bit 14 = 0) Bit 14 = 0: "am" Bit 14 = 1: "pm"	–

[1] After OB 150 is called, the specified value is checked logically for the correct data taking into account leap years.

Data field in the flag area

If you set up the data field in a flag area, you must take into account the following assignment of the data field words to the flag bytes. ’x’ is the parameter "number of 1st data field word" which must be written to ACCU-1-L when OB 150 is called:

Bit no.	15 8	7 0
Data field word 1	Flag byte x	Flag byte x+1
Data field word 2	Flag byte x+2	Flag byte x+3
Data field word 3	Flag byte x+4	Flag byte x+5
Data field word 4	Flag byte x+6	Flag byte x+7

2. ACCUs

2a) ACCU-2-L

ACCU-2-L contains information about the required function and the data field used. It must have the following structure:

Bit no.	15			12	11			8	7			4	3			0
	Function no.				Address area type				Data block no.							

Parameters in ACCU-2-L

Function no.,
Permitted values: 1 = set system time
2 = read system time

Address area type,
Permitted values: 1 = DB data block
2 = DX data block
3 = F flag area
4 = S flag area

Data block no.,
Permitted values: 3 to 255 (only with address area type ’1’ or ’2’; with address area type ’3’ or ’4’ irrelevant)

2b) ACCU-1-L

Number of the 1st data field word
Permitted values (depending on the address area type):

DB, DX:	0 to 2039
F flags:	0 to 248 (= no. of flag byte ’x ’)
S flags	0 to 4088 (= no. of flag byte ’x ’)

Result

After correct processing of OB 150, the RLO, the condition code bits OR, ERAB and OS = 0.

Possible errors

The errors listed in the following table may occur. If an error occurs, the system program sets the RLO to ’1’ and stores the error IDs listed in the table in ACCU 1.

Table 6-12 Error IDs of OB 150 in ACCU-1-L

ID	Meaning
9601H	Data block not loaded
960FH	Multiple call for the block
9611H	Illegal function no.
9612H	Address area type illegal
9613H	Data block number illegal
9614H	"Number of first data field word" illegal
9615H	Data block length < 4 words
9621H	Year specification in the data field illegal
9622H	Month specification in the data field illegal
9623H	Day of month specification in the data field illegal
9624H	Weekday specification in the data field illegal
9625H	Hour specification in the data field illegal
9626H	Minute specification in the data field illegal
9627H	Second specification in the data field illegal
9628H	1/100 seconds in the data field not equal to 0
9629H	Hour format not equal to setting in OB 151

Note
If the parameters are incorrectly assigned for the "set system time" function and if the time has already been set correctly at least once, the error IDs listed are transferred; the previously set system time, however, continues to run.

Examples

"Set system time":

You want to set the system time to the following values:

"Thurs, 24.10.1993, 11:30 hours 0 seconds, 24 hour clock"

The time parameters are stored in data block DB 10 from data word DW 0 onwards.

```
   DB 10
         0: KH =  0 0 0 0    left byte = seconds (BCD), right byte = 0

         1: KH =  9 1 3 0    91 = Format (=80H) + hour (=11 BCD)
                             30 = Minutes (BCD)
         2: KH =  2 4 3 0    24 = Day of month (BCD)
                             30 = Weekday (3 = Thursday) + bit 0 to 3 = 0
         3: KH =  9 3 1 0    93 = Year (BCD)
                             10 = Month (BCD)
```

Continued on next page

"Set system time" *(continued)*

STEP 5 operations in OB 1 for calling OB 150:

```
        :
        :
        :L   KH  1 1 0 A  Values for ACCU-2-L:
        :        │ │  └───DB no. = 10
        :        │ └──────Address area type = 1 for "data field in DB"
        :        └────────Function no. = 1 for "set"
        :L   KF +0        ACCU-1-L:
        :                 No. of 1st data field word = 0
        :JU  OB 150       Call OB 150
        :
```

"Read out system time":

You want the current system time to be written to data block DB 10 from data word DW 4 onwards. You must therefore call OB 150 with the following parameters:

```
        :
        :
        :L   KH  2 1 0 A                Values for ACCU-2-L:
        :        │ │  └─── DB no. = 10
        :        │ └────── Address area type = 1 for "data field in DB"
        :        └──────── Function no. = 2 for "read"
        :L   KF +4        ACCU-1-L:
        :                 No. of 1st data field word = 4
        :JU  OB 150       Call OB 150
        :C   DB 10        Open DB 10
        :                 Evaluate DB 10
```

After OB 150 is called, the current system time is written to data block DB 10 in the following form ("Thurs, 24.10.93, 11:30 hours 20 seconds, 13/100 of a second, 24 hour clock"):

```
  DW 4:   KH =  2 0 1 3   Seconds = 20 (BCD)
                          1/100 seconds = 13 (BCD)
  DW 5:   KH =  9 1 3 0   Format = 24 hour (bit 15 = 1, bit 14 = 0),
                          Hour = 11(BCD), minutes = 30 (BCD)
  DW 6:   KH =  2 4 3 0   Day of month = 24 (BCD)
                          Weekday = 3 = Thursday
  DW 7:   KH =  9 3 1 0   Year = 93 (BCD)
                          Month = 10 (BCD)
```

6.14 OB 151: Set/Read Time for Clock-Controlled Interrupt

Function

By calling OB 151 you can do the following:

- cause the CPU 948 to activate the clock-controlled interrupt ("timed job" - OB 9, refer to Section 4.5.3) at a selected time:
  - every minute
  - every hour
  - every day
  - every week
  - every month
  - every year
  - once,
- read out the current status of a timed job,
- cancel a previously generated timed job.

OB 151 can be called in the START-UP and RUN modes. A generated clock-controlled interrupt is retained when a WARM RESTART (automatic or manual) is carried out. A COLD RESTART deletes an existing timed job.

If you generate a new timed job, an existing timed job is automatically cancelled. This means that only **one** clock-controlled interrupt can be active.

Parameters

1. Data field for job parameters

When **generating** or **cancelling** a timed job, OB 151 takes the required job parameters from a data field. When **reading out** the current status of the job management, OB 151 transfers the current job parameters to a data field.

You can set up this data field in a **data block** or in one of the two **flag areas** (F or S flags).

The data field consists of four words and has the following format when generating and reading out a timed job:

Bit no.	15			12	11			8	7			4	3			0
Word 1	Seconds								0							
Word 2	Format		Hours						Minutes							
Word 3	Day of month								Weekday				**Job type**			
Word 4	Year								Month							

The parameters have the following meaning, range of values and representation:

Parameter	Permitted range of values	Value in
Job type	0 to 7 where: 0: cancel job or no job active 1: every minute 2: every hour 3: every day 4: every week 5: every month 6: every year 7: once	BCD format
Seconds 1/100 seconds Minutes Hours Weekday Day of month [1] Month Year	0 to59 0 0 to 59 0 to 23 or 1 to 12, depending on the format 0 to 6 for Mon to Sun 1 to 31 1 to 12 0 to 99	BCD format
Format [2]	Format for the hour field with the following meaning: Bit 15 = 0: 12 hour clock Bit 15 = 1: 24 hour clock (bit 14 = 0) Bit 14 = 0: "am" Bit 14 = 1: "pm"	–

[1] After calling OB 151, the specified value is checked logically that the date is correct taking into account leap years.

[2] For the meaning of "am" and "pm": refer to OB 150 in the previous section: "format" must match the form specified when setting the system time with OB 150.

Data field in the flag area

If you set up the data field in a flag area, you must take into account the following assignment of the data field words to the flag bytes. "x" is the parameter ’number of the 1st data field word’ which must be written to ACCU-1-L when OB 151 is called.

Bit no.	15 8	7 0
Data field word 1	Flag byte x	Flag byte x+1
Data field word 2	Flag byte x+2	Flag byte x+3
Data field word 3	Flag byte x+4	Flag byte x+5
Data field word 4	Flag byte x+6	Flag byte x+7

2. ACCUs

2a) ACCU-2-L

ACCU-2-L contains information about the required function and the data field used. It must have the following structure:

Bit no.	15			12	11			8	7			4	3			0
	Function no.				Address area type				Data block no.							

Parameters in ACCU-2-L

Function no.,
Permitted values: 1 = generate job
2 = read job status

Address area type,
Permitted values: 1 = DB data block
2 = DX data block
3 = F flag area
4 = S flag area

Data block no.,
Permitted values: 3 to 255 (only with address area type ’1’ or ’2’; with address area type ’3’ or ’4’ irrelevant)

2b) ACCU-1-L

No. of the 1st data field word,
Permitted values (depending on the address area type):

DB, DX:	0 to 2039
F flags:	0 to 248 (= no. of flag byte ’x ’)
S flags	0 to 4088 (= no. of flag byte ’x ’)

Note
There is no point in generating a timed job cyclically (e.g. with an unconditional OB 151 call with function number ’1’ in OB 1).

Result

After correct processing of OB 151, the RLO, the condition code bits OR, $\overline{\mathrm{ERAB}}$ and OS = 0.

Note
If, when reading out the timed job, the job type is ’**0**’ and all the remaining parameters are ’**F**’ or ’**FF**’ (hex) in the data field, no timed job is active.
This status can occur in the following situations:

a) there was a COLD RESTART without a timed job being generated

b) when a "one-time" timed job was due

or

c) when you have cancelled a job.

Possible errors

The errors listed in the following table can occur. If one of these errors occurs, the system program sets the RLO to ’1’ and writes the error IDs listed in the table to ACCU 1.

Table 6-13 Error IDs of OB 151 in ACCU-1-L

ID	Meaning
9701H	Data block not loaded
970FH	Multiple call for the block
9710H	Wrong mode ("process interrupts via IB 0 = on")
9711H	Illegal function no.
9712H	Address area type illegal
9713H	Data block no. illegal
9714H	"Number of the 1st data field word" illegal
9715H	Data block length less than 4 words
9721H	Year specification in the data field illegal
9722H	Month specification in the data field illegal
9723H	Day of month specification in the data field illegal
9724H	Weekday specification in the data field illegal
9725H	Hour specification in the data field illegal
9726H	Minute specification in the data field illegal
9727H	Second specification in the data field illegal
9728H	1/100 seconds in the data field not equal to 0
9729H	Hour format not equal to setting in OB121/OB 150
972AH	Job type illegal

Note
If incorrect parameters are assigned **and** a valid timed job was previously generated, the error IDs listed above are transferred; **the previously generated timed job, however, remains active**.

Points to note with the time parameters

Regardless of when a clock-controlled interrupt (timed job) is to be triggered, the individual time parameters must be specified in certain combinations. Depending on the selected time for the clock-controlled interrupt, certain parameters must be specified, while others are not evaluated by the system program.

The following table shows which time parameters must be specified for which timed job (XXX = must be specified, --- = irrelevant).

Table 6-14 Assignment of "timed job - time parameters"

Interval	Seconds	Minu-tes	Hours	Week-day	Day of month	Month	Year
every minute	XXX	---	---	---	---	---	---
every hour	XXX	XXX	---	---	---	---	---
every day	XXX	XXX	XXX	---	---	---	---
every week	XXX	XXX	XXX	XXX	---	---	---
every month	XXX	XXX	XXX	---	XXX	---	---
every year	XXX	XXX	XXX	---	XXX	XXX	---
once	XXX	XXX	XXX	---	XXX	XXX	XXX

When **reading out** the time parameters, the irrelevant parameters are assigned the value FFH.

Special situations

- If the "29th of February" is selected with the job type "every year" (= 6), this means that OB 9 is only called every leap year.
- If the value "29", "30" or "31" is selected with the job type "every month" (= 5), OB 9 is only called in the months which have these dates.

Examples

Various timed jobs (in 24 hour format):

```
1. "Job at 29th second of every minute"
    (12:44:29, 12:45:29 etc.):

   You must specify:       Job type    =     1  (Function no. in ACCU-2-L = 1)
                           Seconds     =    29


2. "Job every hour at xx:14:15":

   You must specify:       Job type    =     2  (Function no. in ACCU-2-L = 1)
                           Seconds     =    15
                           Minutes     =    14


3. "Job daily at 5:32:47":

   You must specify:       Job type    =     3  (Function no. in ACCU-2-L = 1)
                           Seconds     =    47
                           Minutes     =    32
                           Format/hour =    85


4. "Job weekly, Tuesdays at 10:50:00":

   You must specify:       Job type    =     4  (Function no. in ACCU-2-L = 1)
                           Seconds     =    00
                           Minutes     =    50
                           Format/hour =    90
                           Weekday     =    01

5. "Job monthly, on the 14th at 7:30:15":

   You must specify:       Job type    =     5  (Function no. in ACCU-2-L = 1)
                           Seconds     =    15
                           Minutes     =    30
                           Format/hour =    87
                           Day of month=    14


6. "Job yearly, on the 1st of May at 00:01:45":

   You must specify:       Job type    =     6  (Function no. in ACCU-2-L = 1)
                           Seconds     =    45
                           Minutes     =    01
                           Format/hour =    80
                           Day of month=    01
                           Month       =    05
```

Continued on the next page

Various timed jobs (in 24 hour format), *continued* :

```
7. "Job once on the 31.12.1999 at 23:55:00":

   You must specify:       Job type    =     7  (Function no. in ACCU-2-L = 1)
                           Seconds     =    00
                           Minutes     =    55
                           Format/hour =    A3
                           Day of month=    31
                           Month       =    12
                           Year        =    99


8. "Cancel job":

   You must specify:       Job type    =     0   (Function no. in ACCU-2-L = 1)


9. "Read out time job":

   You must specify:       Function no. in ACCU-2-L = 2

   If no job is active, you obtain the following result in the data field:

                           Data field word 0:  FFFF H
                           Data field word 1:  FFFF H
                           Data field word 2:  FFF0 H
                           Data field word 3:  FFFF H
```

6.15 OB 153: Set/Read Time for Delayed Interrupt

Using OB 153, you can transfer so-called "delay jobs" to the system program. After a specified delay time "a delayed interrupt" is then processed (refer to OB 6, Section 4.5.3).

Function

By calling OB 153, you can do the following:

- define and start a delay time,
- stop an activated delay time (cancel delay job),
- read how long the delay time still has to run.

A delay job can be activated in the START UP and RUN modes.

Life of a delay job

The delayed interrupt triggered by a delay job is only activated by the system program in the **RUN** mode (OB 6 call).
Jobs which become due in a mode other than RUN are discarded by the system program **without any message**.
A currently active (but not yet due) job is also discarded if the CPU changes to the STOP mode or if the power is switched off.

Parameters

ACCUs

a) ACCU-2-L

ACCU-2-L only needs to be supplied with the function number ’**1**’ ("define delay time") when OB 153 is called, as follows:

Delay time in milliseconds (max. 65535)

Permitted values:	0001H to FFFFH

b) ACCU-1-L

Function no.

Permitted values:	1 = define and start delay time 2 = stop delay time (= cancel job) 3 = read remaining delay time

Note
If a previously defined delay time is not yet elapsed when a further delay time is defined, the previously defined time is lost and the new delay time started.

Result

After correct processing of OB 153, the RLO, the condition code bits OR, $\overline{\text{ERAB}}$ and OS = 0.

When OB 153 is called with the function no. ’2’ or ’3’, ACCU-1-L contains the remaining time to run in milliseconds.

If no delay job is active when OB 153 is called with function no. ’2’ or ’3’, ACCU-1-L contains the value ’0’.

Possible errors

The errors listed in the following table can occur. If one of the errors occurs, the system program sets the RLO to ’1’ and writes the error IDs listed in the table to ACCU 1.

Table 6-15 Error IDs of OB 153

ID	Meaning
990FH	Multiple call for the block
9910H	Wrong mode ("process interrupt via IB 0 = on")
9911H	Illegal function number
9921H	Delay time illegal

Examples

Define and start delay time:

When an AUTOMATIC WARM RESTART is performed, after 5 seconds a certain STEP 5 operation sequence must be run through once. To do this, the delay time is defined and started in start-up organization block OB 22.

The STEP 5 operations in OB 22 for calling OB 153:

```
        :
        :
        :L   KF  +5000     Value for ACCU-2-L: 5000 ms
        :L   KF  +1        Value for ACCU-1-L: function no. = 1 for
        :                  "define and start delay time"
        :JU  OB 153        Call OB 153
        :
```

Stop delay time (cancel job)

STEP 5 operations for calling OB 153:

```
        :
        :
        :L   KF  +2       Value for ACCU-1-L: function no. = 2 for
        :                 "stop delay time"
        :JU  OB 153       Call OB 153
        :
        :
```

Read out remaining time of a delay job:

STEP 5 operations for calling OB 153:

```
        :
        :
        :L   KF  +3       Value for ACCU-1-L: function no. = 3 for
        :                 "read out remaining time"
        :JU  OB 153       Call OB 153
        :
        :                 ACCU-1-L contains the time the delay job still
        :                 has to run.
        :
```

6.16 OB 180: Variable Data Block Access

Using OB 180

You can use OB 180 when working with data blocks that are longer than 261 words (incl. 5 words header).
Using OB 180, you can shift an "access window" of 256 data words in steps of 16 words over a data block (at paragraph addresses). Call OB 180 each time you want to shift the access window further.

In contrast to the CPU 928B, you cannot shift the access window continuously but only in steps that are **multiples of 16**.

Function

When you use OB 180, the start address of the current data block is shifted towards the end of the block by the specified value. This takes into account that the length of DB still available is reduced (DBA and DBL registers are loaded in keeping with the shift, see Sections 8.3 and 9.2.1).

> **Note**
> Before calling OB 180, a data block (DB or DX) with an **adequate length** must already be open.

Parameters

ACCU-1-L

Shift number S:	Number of data words, by which the data block start address will be shifted.
Permitted values:	0 ≤ V < DBL, **S = n * 16 (16, 32, 48 ...)**

Result

After you have called OB 180 **succesfully**:

- the relative address of DW 0 is shifted by the value contained in ACCU-1-L (the DBA and DBL registers are updated accordingly),
- the RLO is cleared (RLO = 0),
- all other bit and word codes are cleared,
- the content of ACCU-1-L = 0.

Possible errors

The errors listed in the following table can occur. If an error does occur, the system program sets the RLO to ’1’ and enters the error IDs listed in the table in ACCU 1. The other bit and word codes are cleared.
The values of the DBA and DBL remain unchanged.

Table 6-16 Error IDs of OB 180

ID	Meaning
B401H	No data block is open
B410H	The shift number S is not a multiple of 16
B411H	a) The shift number is too high; the block end is exceeded by the new window position. b) The shift number is negative.

Setting the access window back to the start value

Opening the data block again with the operations C DB or C DX returns the access window to its original position.

Reaction to nesting

If the access window is shifted by calling OB 180 in a logic block and a further logic block is then called, the position of the access window remains where it is in the called logic block **until OB 180 is called again** (the DBA/DBL values do not change).

If, on the other hand, the access window is shifted in a called logic block by calling OB 180, when program execution returns from the called block (block end operation), **it is returned to the position it had when the nested logic block was called.**

Example

You want the data block start address (DBA = 4152H in DB 17, length = 256 DW) to be shifted by 32 data words relative to the end of the block.

```
   :C   DB   17   open DB 17
   :L   KB   32   shift value as constant
   :JU  OB   180  call OB 180: shift access window
```

After OB 180 has been called, the data word, for example, at address 4 1543H can no longer be addressed with DW 35 but only with DW 3 etc. (see Fig. 6-3).

Due to the change made at the same time in the DBL register, error monitoring is not affected: the operation T DW 223 is permitted, while T DW 224/L DW 224 causes an error.

By calling OB 180 again, the DBA can be increased again (and the DBL reduced). The operation C DB 17 returns the block to its original settings (DBA = 4152H, length 256 DW).
If DB 17 had a length of 256 data words for example, you could then no longer access DW 256 and DW 257 using STEP 5 operations. By shifting the DBA resgister by 16, data words 256 and 257 can be addressed as "DW 240" and "DW 241".

Continued on next page

Example continued:

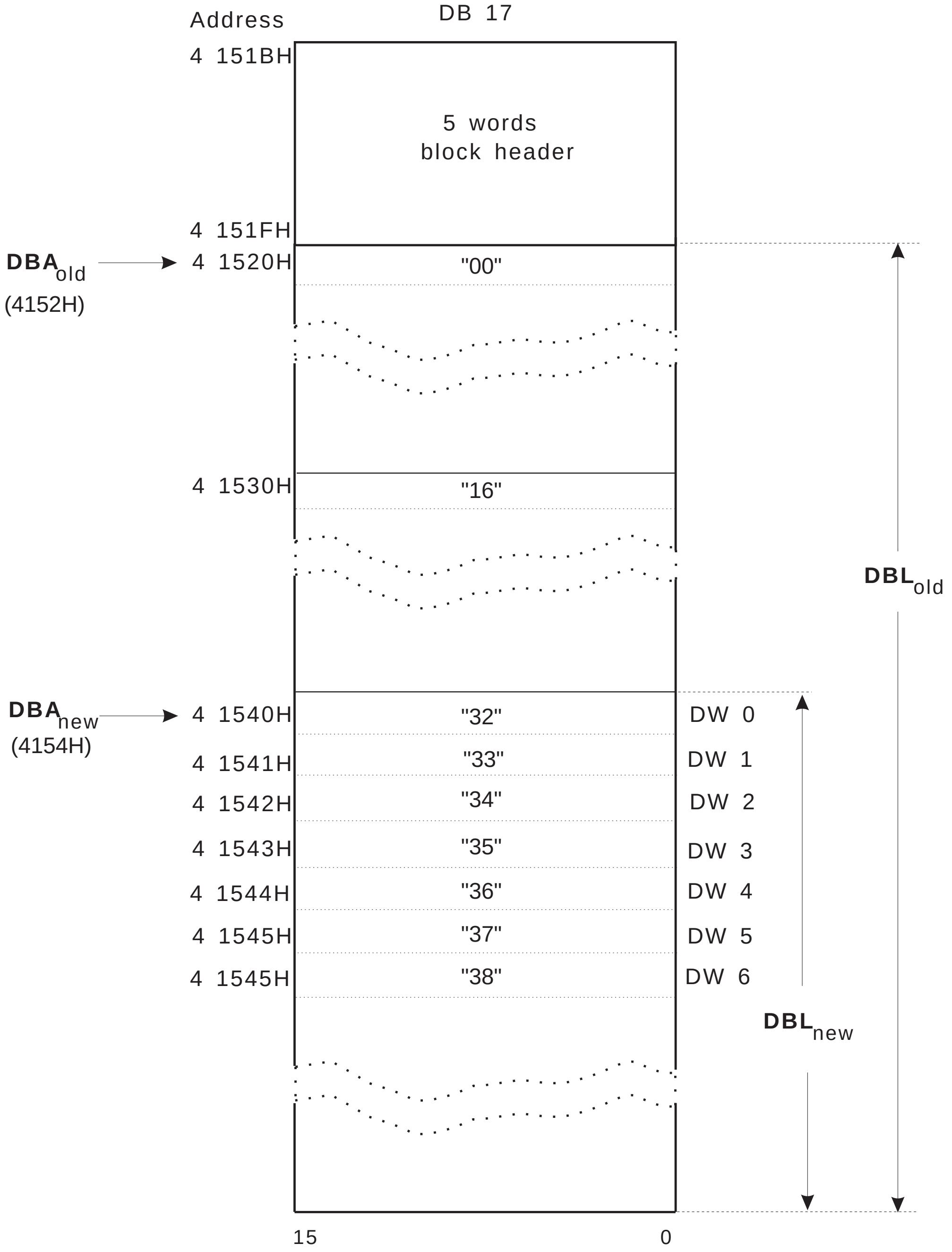


Fig. 6-3 Shifting the DB start address

6.17 OB 181: Test Data Blocks (DB/DX)

With the special function organization block OB 181, you can test data blocks as follows:

- whether or not a particular DB or DX data block exists,
- the address at which the first data word of the data block is stored,
- how many data words the data block contains.

Using OB 181

The function "test DB/DX" is useful before the operations TNB/TNW, G DB/GX DX and before calling special function organization blocks OB 182, OB 254 and OB 255.

Before transferring data words, for example, you can call OB 181 to check that the destination data block is valid and long enough for the transfer.

Function

OB 181 checks whether a specified data block exists and returns the characteristic parameters of a data block.

Parameters

ACCU-1-L

a) ACCU-1-LL

Block number,
Permitted values: 1 to 255

b) ACCU-1-LH

Block ID,
Permitted values: 1 = DB
2 = DX

Result

If the function is executed without any error and if the block exists on the CPU, the system program transfers the following values:

- **ACCU-1-L**: Address of the 1st data word (DW 0), 20-bit address,
- **ACCU-2-L**: Length of the data block in words (without block header) Example: ACCU-2-L contains the value'7': The data block consists of data words DW 0 to DW 6,
- **RLO**: = 0.

Possible errors

The errors listed in the table below can occur. If an error occurs, the system program sets the RLO to '1' and the following condition code bits as shown in the table. It also enters an error ID in ACCU-1-L.

Table 6-17 Error codes of OB 181 and their scans

RLO	CC 1	CC 0	ACCU-1-L	Meaning	Scan
1	0	1	B501H	Block does not exist	JC JM JN
1	1	0	B502H	Wrong block number	JC JP JN
1	1	0	B503H	Wrong block ID	JC JP JN

6.18 OB 182: Copy Data Area

Function

OB 182 copies a data area of variable length from one data block to another. The source and destination blocks can be DBs or DXs. The start of the area in the source and destination blocks can be freely selected. OB 182 can copy a maximum of 4091 data words.

> **Note**
> The source and destination block can be the same. The data areas of the source and destination can overlap. Even if they overlap, the **original** data of the source area are copied unchanged to the **destination** area. The **area of the overlap** in the **source** is overwritten after the copy function. You can use this to shift a data area within a block.

Parameters

1st data field with parameters for copy function

Before calling OB 182 make a data field available with the parameters for the copy function. This data field can be set up in a DB or DX data block or in the F or S flag area.

The data field identifies the source and destination blocks, the start address of the area in both blocks and the number of data words to be copied. It consists of five words:

Bit no.	15 … 8	7 … 0
1st word	Source DB type	Source DB no.
2nd word	No. of 1st transferred data word in source DB	
3rd word	Destination DB type	Destination DB no.
4th word	No. of 1st transferred data word in the destination DB	
5th word	No. of data words	

The parameters have the following significance and range of values :

Parameter	Permitted range of values
Data block type (source and destination)	1 = DB 2 = DX
Data block no. (source and destination)	3 to 255
No. of 1st data word (source and destination)	0 to 4090
Number of data words	1 to 4091

Data field in flag area

If you set up the data field in a flag area, you must take into account the following assignment of data field words to flag bytes. 'x' is the parameter "no. of the 1st data field word" which you must enter in ACCU-1-L when OB 182 is called:

Bit no.	15	78 0
1st data field word	Flag byte x	Flag byte x+1
2nd data field word	Flag byte x+2	Flag byte x+3
3rd data field word	Flag byte x+4	Flag byte x+5
4th data field word	Flag byte x+6	Flag byte x+7
5th data field word	Flag byte x+8	Flag byte x+9

2. Accumulators

2a) ACCU-2-L

ACCU-2-L contains information about the data field used. It must have the following structure:

Bit no.	15 8	7 0
	Address area type	Data block no.

Parameters in ACCU-2-L

Address area type,
Permitted values: 1 = DB data block
2 = DX data block
3 = F flag area
4 = S flag area

Data block no.,
Permitted values: 3 to 255 (only with address area type '1' or '2'; with address area type '3' or '4' irrelevant)

2b) ACCU-1-L

Number of the 1st data field word, possible values (depending on the address area type:

DB, DX:	0 to 2038
F flags:	0 to 246 (= No. of flag byte ’x ’)
S flags	0 to 4086 (= No. of flag byte ’x ’)

Result

After correctly processing OB 182: RLO, condition code bits OR, ERAB and OS = 0.

Possible errors

If an error occurs, an error ID is entered in ACCU 1 (see table below).

Table 6-18 Error IDs of OB 182 in ACCU-1-L

ACCU-1-L	Meaning
B601H	Data block not loaded
B60FH	Multiple block call
B611H	Incorrect content in the data field
B612H	Address area type illegal
B613H	Data block no. illegal
B621H	"Number of the 1st data field word" illegal
B622H	"Source data block type" illegal
B623H	"Source data block no." illegal
B624H	"No. of the 1st data word to be transmitted in source DB" illegal
B625H	Length of the source data block in the block header < 5 words
B626H	"Destination data block type" illegal
B627H	"Destination data block no." illegal
B628H	"No. of 1st data word to written in destination DB" illegal
B629H	Length of the destination block in block header < 5 words
B62AH	"Number of data words to be transmitted" illegal (= 0 or > 4091)
B62BH	Source data block too short
B62CH	Destination data block too short

6.19 OB 202 to 205: Multiprocessor Communication

A detailed description of these special function organization blocks can be found in Chapter 10.

The special function organization blocks OB 200 and OB 202 to OB 205 allow data transfer between the individual CPUs using the coordinator COR C in multiprocessor operation.

- **OB 200: initialize**

  This special function organization block sets up the memory in the COR C coordinator in which the blocks of data to be transferred are buffered.

- **OB 202: send**

  This function transfers a block of data to the buffer of the COR C and specifies how many blocks of data can still be sent.

- **OB 203: send test**

  The special function OB 203 checks the number of free memory fields in the buffer of the COR C.

- **OB 204: receive**

  This function accepts a block of data from the COR C buffer and indicates how many blocks of data can still be received.

- **OB 205: receive test**

  The special function organization block OB 205 checks the number of occupied memory fields in the COR C buffer.

6.20 OB 222: Restart Cycle Monitoring Time

The special function OB 222 causes the cycle monitoring time to be restarted, i.e. the timer for monitoring is started from the beginning. By calling this special function, the maximum permitted cycle time for the current cycle is extended by the value selected at the time of the call.

Parameters

none

Possible errors

none

6.21 OB 223: Compare Start-Up Modes

Function

By calling OB 223, you can check whether the start-up modes of **all** CPUs involved in multiprocessor operation are the same and able to execute a programmed reaction to errors.

Parameters

none

Result

After calling OB 223, the system program sets the RLO to ’0’ and writes the value 01H to ACCU-1-LL when the start-up modes are the same.

Possible errors

- Start-up modes are not the same
- Other errors, refer to condition code bits

Condition code bits

If an error occurs, the system program sets the RLO to ’1’ and transfers an error ID to ACCU-1-LL.

Table 6-19 Results IDs of OB 223 in ACCU-1-LL

ID	Meaning
01H	Start-up modes the same
02H	Internal system error
03H	Start-up modes not the same
04H	Single processor mode, comparison of start-up modes not possible

6.22 OB 254/255: Copy/Duplicate Data Blocks

With the special functions OB 254/255, you copy individual data blocks from a memory card to the user memory or duplicate individual data blocks within the user memory.
The special function OB 254 and OB 255 work identically, with OB 254 exclusively for DX data blocks and OB 255 for DB data blocks.

> **Note**
> During the copying or duplication, the user interrupts are blocked. No timed interrupts or process interrupts are accepted.

Application

Copying data blocks from the memory card or duplicating data blocks in the user memory and assigning a new block number.

Copying

Conditions

By using the "**copy**" function of the two special function OBs (OB 254 or OB 255 call), remember the following conditions:

- **The memory card must be plugged in before the OVERALL RESET and must not be removed afterwards**
- The destination data block must not yet exist
- The online function "compress memory" must not be active

Function

A data block is copied from the memory card to the user memory and retains its original block number. The start address is entered in the address list in DB 0.

Parameters

1. ACCU-1-LL

Number of the block to be copied.

The following block numbers are possible:

Block type	Block number
DB (OB 255) DX (OB 254)	3 to 255 3 to 255

2. ACCU-1-LH

ACCU-1-LH must be zero.

Duplicating

Function

A data block is duplicated within the user memory and it is assigned a different 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 is retained, i.e. the original data block remains valid.

The start address is entered in DB 0 only after the transfer is completed and all the IDs are correctly entered in the block header. The duplicated block is therefore only declared valid or existent by the system program after it has been completely transferred.

Parameters

1. ACCU-1-LL

Number of the block to be duplicated (source).

2. ACCU-1-LH

Number of the new block (destination).

The following block numbers are possible:

Block type	Block number
DB (OB 255) DX (OB 254)	3 to 255 3 to 255

Result following copying and duplicating

After the function has been executed correctly and error-free, the system program sets the RLO to ’0’ and clears the condition code bits CC 1 and CC 0.

Calling OB 254/255 changes the contents of ACCU 1 to ACCU 4. The BR register is retained.

Possible errors and warnings when copying and duplicating

If an error or warning occurs, the system program stops processing OB 254/255 and continues program execution at the next STEP 5 operation. It also sets the RLO to ’1’ and writes an ID to ACCU-1-LL (refer to Table 6-18).

If the function is aborted due to a warning, it may be possible to run OB 254/255 correctly by calling the special function again (if necessary, repeated several times).

In the following situation, OB 254/255 is aborted with a **warning**:

an OB 124, OB 125, OB 254 or OB 255 has been called during the last 10 ms. (During a period of 10 ms, however, only one special function OB call is permitted. This prevents multiple calls for the OBs listed above preventing the interface to the PG from being processed.)

Condition code bits following copying and duplicating

After OB 254/255 is called, you can see whether the special function has been carried out correctly or whether it was stopped by an "error" or "warning" based on the result of logic operation and the condition code bits CC 1 and CC 0. The result can be evaluated by conditional jump operations.

Result codes

Table 6-20 Result codes for OB 254/255

RLO	CC 1	CC 0	Meaning	Scan
0	0	0	Special function correctly processed	JB JZ
1	1	0	Special function aborted with "warning"	JB JP JN
1	0	1	Special function aborted with "error"	JB JM JN

IDs in ACCU-1-LL

The system program sets IDs in ACCU-1-LL which specify the causes of a warning or error in more detail.

Bit no.	7	6	5	0
	W	E	Cause of error/warning	

The following group codes apply:

Bit no. 7 (W) = 1:warning
Bit no. 6 (E) = 1:error

Table 6-21 Result IDs for OB 254/255 in ACCU-1-LL

ID	Meaning
01H	Function correctly executed
	Error:
41H	Block header on memory card invalid
43H	Not enough memory space
48H	Source data block does not exist
4AH	Block number or type illegal/source DB
4BH	Block number or type illegal/destination DB
4CH	Destination data block already exists in the user memory
4DH	Online function COMPRESS MEMORY active
4EH	No memory card plugged in
	Warnings:
8DH	Conflict with an online function except "compress memory"
8EH	10 ms waiting time not yet elapsed

Examples

1. "Copy":

```
:L   KY 0,120  This sequence of operations copies
:JU  OB 254    the data block DX 120 from the
:              memory card to the user memory
:
```

2. "Duplicate":

```
:L   KY 80,85  This sequence of operations
:JU  OB 255    duplicates data block DB 85; the
:              new data block has the number 80.
:              The contents of DB 80 and DB 85
:              are identical.
:
```

Extended Data Block DX 0

7

Contents of Chapter 7

7 Extended Data Block DX 0

The following chapter explains how to use the data block DX 0 and how it is structured. You will find information about the meaning of the various DX 0 parameters and will learn how to create and how to assign parameters for a DX 0 data block based on examples.

7.1 Application

You can adapt certain system program functions to meet your own requirements by selecting alternative defaults in DX 0 compared to the standard defaults (marked in the parameter table by "D").

The defaults of the system program (D) are automatically set during each COLD RESTART and DX 0 is then evaluated. If you have not assigned parameters in DX 0 and loaded it, the defaults remain valid; otherwise the defaults you have selected in DX 0 become the valid settings.

You can make settings in DX 0 by programming the values just as in any normal data block (refer to Section 7.2 to 7.4.1) or using the PG system software S5-DOS from Version 3.0 onwards, you can enter the values as parameters in a special screen form on your PG (refer to Section 7.4.2).
You can make use of the full range of functions of the DX 0 screen form if the PG software STEP 5/ST, Version 6.3 or STEP 5/MT, Version 6.0 plus the corresponding "Delta diskette CPU 948" is installed on your PG.

> **Note**
> The settings or modifications made in DX 0 only become effective following a COLD RESTART.
> If a **modified DX 0** is read during a COLD RESTART, the **unmodified** parameter assignments are **retained**.

Differences compared with the CPU 946/947

Compared with DX 0 parameter assignment for the CPU 946/947, there are various differences when assigning DX 0 parameters for the CPU 948, as follows:

- Modes:
  There is no longer a 150U/155U mode, instead of this, there is now "**interruptability at block boundaries**" and "**interruptability at operation boundaries".**

- Processing system interrupts:
  With the CPU 948, you can now also combine "process interrupts via IB 0 = off" (= system interrupt processing) with "**interruptability at block boundaries**".
  This means that multiprocessor operation is now also possible in the "interruptability at block boundaries" mode.

- Additional timed interrupts:
  If you switch off the mode "process interrupts via IB 0", the additional timed interrupts the **delayed interrupt (OB 6)** and the **clock-controlled interrupt (OB 9)** are available.

7.2 Structure of DX 0

DX 0 consists of three parts:

- the start ID for DX 0 (DW 0, 1 and 2) ,
- several fields of different lengths (depending on the number of parameters)

  and

- the end ID.

Start ID

ASCII characters MASKX0 in DW 0 to DW 2

Field

A **field** in DX 0 consists of one to n data words. These contain the following:

- the field ID,
- the field length

  and

- the field parameters.

Field ID

The **field ID** specifies the meaning of the parameters following it. Each field is assigned to a specific system program section or to a specific system function (e.g. the field ID "04" identifies the parameter field for cyclic program execution).

Field length

The **field length** specifies how many data words are occupied by the parameters.

Parameters

The possible **parameters** are listed in Section 7.3. The specified numerical values are in hexadecimal format (KH).

End ID

This indicates the end of DX 0 with EEEEH in the last data word.

Formal structure

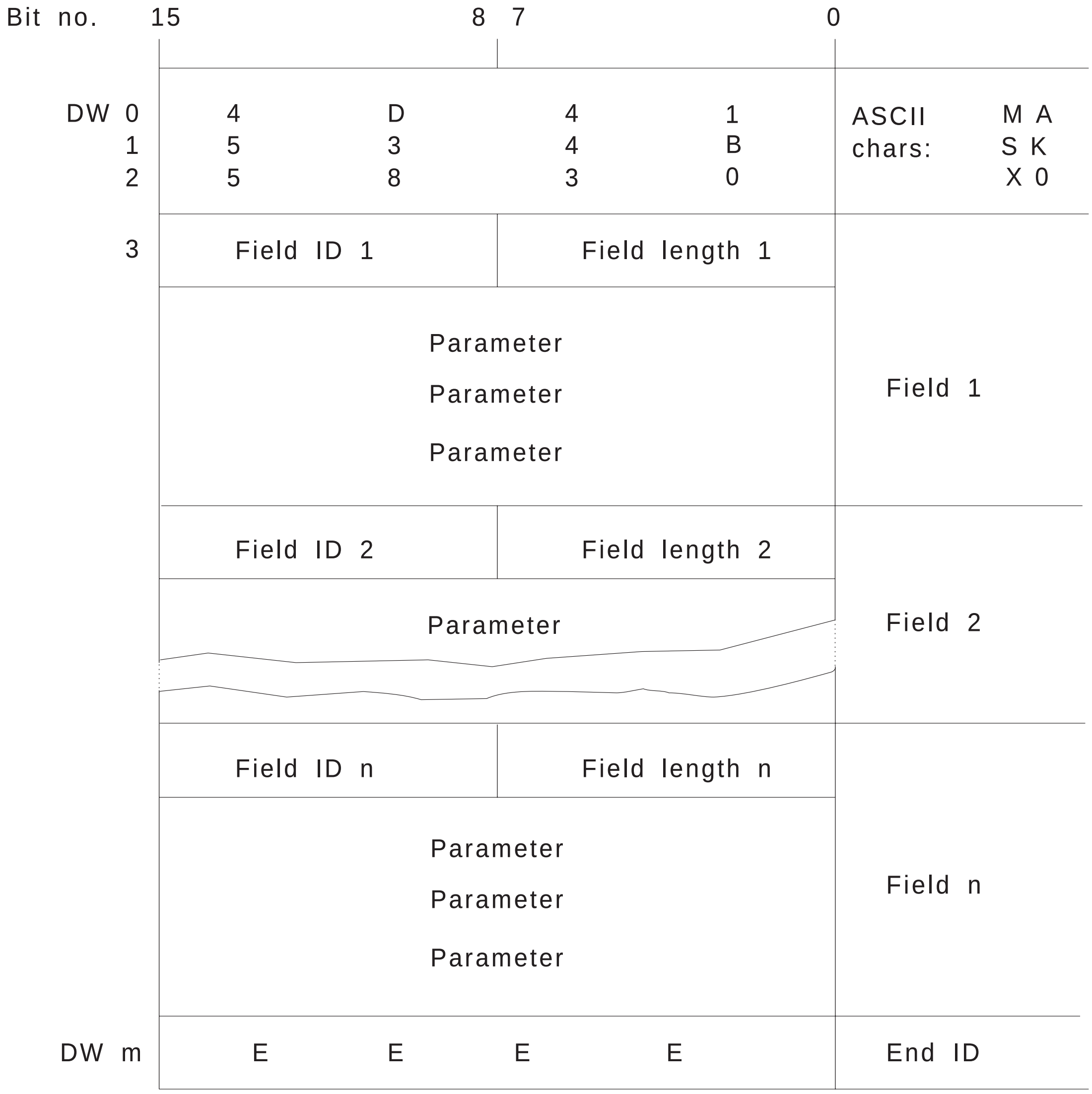


Fig. 7-1 Structure of DX 0

7.2.1 Example of Input in DX 0

```
Start ID                      DW 0:         KH = 4D41
                              DW 1:         KH = 534B
                              DW 2:         KH = 5830

Field ID/length               DW 3:         KH = 0101    Field 1
Parameters (occupies 1 DW)    DW 4:         KH = 1001

Field ID/length               DW 5:         KH = 0402
Parameters (occupies 2 DW)    DW 6:         KH = 1000    Field 2
                              DW 7:         KH = 0040

End ID                        DW10:         KH = EEEE
```

When assigning parameters for DX 0, note the following points:

- Unnecessary fields do not need to be specified.
- Maintain the order of the fields (e.g. specify the field with ID ’02’ **before** the field with ID ’05’).
- A specific field must only occur **once** in DX 0.
- The number of parameters must correspond to the field length specified at the beginning of the field.
- Maintain the order of parameters. Unnecessary parameters towards the **beginning of the field** must be assigned the default to ensure that the parameter order is maintained.
- Close DX 0 after entering the last field with the end identifier "KH=EEEE".

7.3 Parameters for DX 0

Table 7-1 DX 0 parameters and their meaning

Field ID/length	Parameters 1st/2nd word	Meaning [1]
Modes		
01xx [2]	1000 1001	D Interrupts at block boundaries [3] (CPU 946/947: 150U mode) Interrupts at operation boundaries [4] (CPU 946/947: 155U mode)
Start-up program execution		
02xx	1000 1001 1002	D AUTOMATIC WARM RESTART after POWER UP AUTOMATIC COLD RESTART after POWER UP MANUAL COLD/WARM RESTART after POWER UP
	2000 2001	D Synchronization of START-UP in the multiprocessor mode No synchronization of START-UP in the multiprocessor mode
	BB00 00yy [5]	Number of timers to be updated yy: [6] Permitted values: 00H to FFH D yy = FF (timer T 0 to T 255)
	4000 4001	D Restart type = WARM RESTART Restart type = RETENTIVE COLD RESTART
Cyclic program execution		
04xx	1000 00yy	Setting the cycle monitoring time: [7] Cycle monitoring time = (yy ∗ 10 ms) Permitted values: 01H to FFH D yy = 14H (200 ms)
	4000 4001	D Updating of the process image and IPC flags without semaphore protection Updating the process image and IPC flags with semaphore protection (in the field, refer to Section 10.1.3)
Interrupt servicing: timed interrupts		
05xx	1000 000c 1001 0000	D Timed interrupt servicing "on" Timed interrupt servicing "off" c = level priority, permitted values: 1 to 5 D c = 1 (highest level priority)

Field ID/length	Parameters 1st/2nd word		Meaning [1]	
Table 7-1 continued:				
05xx			**Interrupt servicing: timed interrupts (cont.)**	
	2000	00yy	Basic clock rate for timed interrupt servicing: Basic clock rate = (yy ∗ 10 ms) Permitted values: 01H to FFH D yy = 0A (100 ms)	
	3000 3001		D Clock rate distribution according to interval 1 (1, 2, 5, 10) Clock rate distribution according to interval 2 (2^n) (Refer to Section 4.5.2)	
			Interrupt servicing: process interrupts via S5 bus/system interrupts	
	4000 4001	000c 0002	System interrupt X "on" D System interrupt X "off"	c = level priority, Permitted values: 1 to 5 X = A, B, C or D V c = 2 (level priority 2) (The servicing of system interrupts can also be combined with "interruptability at block boundaries" with the CPU 948.)
	5000 5001	000c 0002	System interrupt E "on" D System interrupt E "off"	
	6000 6001	000c 0002	System interrupt F "on" D System interrupt F "off"	
	7000 7001	000c 0002	System interrupt G "on" D System interrupt G "off"	
	8000 8001	000c 0000	D Process interrupts via IB 0 "off" Process interrupts via IB 0 "on" c = level priority, permitted values 1 to 2 D c = 2 (level priority 2) When "process interrupts via IB 0 = on": - only single processor mode, - only "interruptability at block boundaries".	
EEEE			End ID	

[1] D = default with DX 0 not loaded or not present.

[2] xx = field length (number of data words occupied by the parameters).

3) With the CPU 948, can also be combined with a system interrupt.

[4] Must not be combined with process interrupts via IB 0.

[5] When specifying the field length, the value "2" must be taken into account for parameters occupying two data words.

[6] For updating the timers, please read the explanation on the following page.

[7] Set cycle monitoring time with OB 31 or DX 0: If the cycle monitoring time is set both with OB 31 and with DX 0, the system program uses the setting made in OB 31 since it has already evaluated DX 0. For this reason, you should only use one of these possibilities. We recommend setting the cycle monitoring time using DX 0.

Updating the timers

- As standard, the timers T 0 to T 255 are updated.
- If you enter the value "0" in DX 0, **no** timers are updated, even if they are included in the program. There is then also no error message output.
- Updating is as follows:

Entry	’0’	’1’	’2’	’3’	’4’	....
Updating	none	T0 to T1	T0 to T2	T0 to T3	T0 to T4	....

Level priorities

You can specify different priorities for the program execution levels. This is achieved either using the default or by specifying DX 0 parameters.

Priorities when "process interrupts via IB 0 = on" is selected (PROCESS INTERRUPTS level)

These priorities have the following default values in DX 0:

- timed interrupts: level priority 1 (higher priority)
- Process interrupts via input byte IB 0: level priority 2 (lower priority)

You can swap over the priorities in DX 0.

Priorities when "process interrupts via IB 0 = off" is selected (= processing of system interrupts level INTERRUPTS)

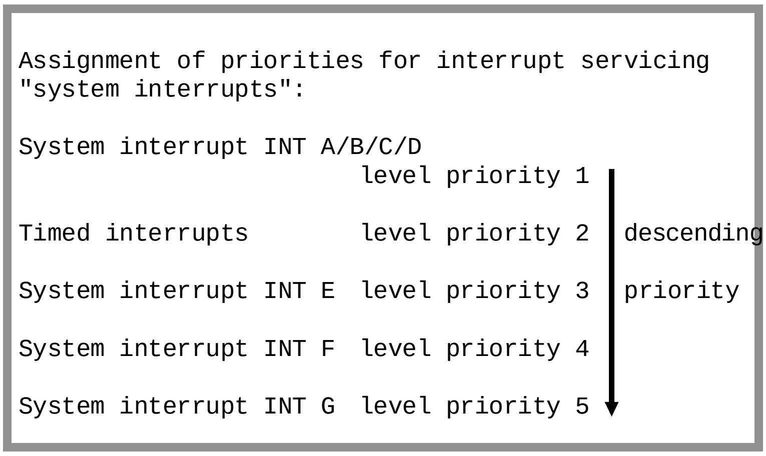
In this mode, the following default priorities are set:

- timed interrupts: level priority 1 (higher priority)
- system interrupts:level priority 2 (lower priority)

You can modify these priorities for the following program execution levels individually in DX 0, by specifying the priority value from ’1’ to ’5’ (the value ’1’ means highest priority):

- timed interrupts
- system interrupt INT X (X = A, B, C or D),
- system interrupt INT E,
- system interrupt INT F,
- system interrupt INT G.

Example



```
Assignment of priorities for interrupt servicing
"system interrupts":

System interrupt INT A/B/C/D
                        level priority 1

Timed interrupts        level priority 2   descending

System interrupt INT E  level priority 3   priority

System interrupt INT F  level priority 4

System interrupt INT G  level priority 5
```

7.4 Examples of Parameter Assignment

7.4.1 STEP 5 Programming

Example A:

You want to use three CPUs in the multiprocessor mode: CPU A, B and C. CPU A and B work closely with each other, often exchange data and execute a complicated start-up program.

CPU C executes a short, time-critical program largely independent of A and B.

As standard, all CPUs operating in the multiprocessor mode start cyclic program execution together, i.e. the CPUs wait for each other until they have all completed their start-up and then change to cyclic program execution together.

Since CPU C executes its program independently of the other CPUs and has a **very short start-up program**, there is no need to synchronize its start-up with the others. By assigning parameters in DX 0, CPU C can start its cyclic program immediately after completing the start-up without waiting for CPU A and B.

The parameter for synchronizing the CPUs in the multiprocessor mode is the second parameter in the first field. To maintain the order of the parameters, the first parameter for the start up must have the default value (AUTOMATIC WARM RESTART after POWER UP).

Program DX 0 for CPU C as follows:

```
DX 0      Start ID "MASKX0"               DW 0:          KH = 4D41
                                          DW 1:          KH = 534B
                                          DW 2:          KH = 5830
          1st field ID/length             DW 3:          KH = 0202
          Parameter 1                     DW 4:          KH = 1000
          Parameter 2                     DW 4:          KH = 2001
          End ID                          DW 5:          KH = EEEE
```

Once you have loaded this DX 0 in the program memory, it becomes effective at the next COLD RESTART. Since CPU C runs through a very short start-up program and does not wait for A and B, its green RUN LED lights up immediately following start-up. The BASP signal (disable command output) is, however, only deactivated when all three CPUs have completed their start-up. This means that CPU C cannot access the digital I/Os.

Example B:

The parameter assignment for DX 0 shown below achieves the following:

- the mode "interrupts at operation boundaries" is set,
- the timer updating is switched off,
- the cycle time is set to 2.5 seconds,
- the level priority of timed interrupts is set to ’2’

  and

- the system interrupt INT E is activated with level priority ’1’.

DX 0	Start ID "MASKX0"	DW 0:	KH = 4D41
		DW 1:	KH = 534B
		DW 2:	KH = 5830
	1st field ID/length	DW 3:	KH = 0101
	Parameters	DW 4:	KH = 1001
	2nd field ID/length	DW 5	KH = 0202
	Parameters [1]	DW 6:	KH = BB00
		DW 7:	KH = 0000
	3rd field ID/length	DW 8	KH = 0402
	Parameters [1]	DW 9:	KH = 1000
		DW10:	KH = 00FA
	4th field ID/length	DW11:	KH = 0504
	Parameters [1]	DW12:	KH = 1000
		DW13:	KH = 0002
	Parameters [1]	DW14:	KH = 5000
		DW15:	KH = 0001
	End ID	DW16:	KH = EEEE

This parameter assignment in DX 0 has the following effects on program execution:

Program execution is interrupted by higher priority levels at operation boundaries instead of at block boundaries.

The runtime of the system program is slightly reduced since no timers are updated.

A cycle error is only recognized when the runtime of the user program and the system program together exceeds 2.5 seconds.

No process interrupts from input byte IB 0 are processed, but rather system interrupt INT E. Owing to its higher priority, this interrupts timed interrupt servicing, the processing of the delayed interrupt and the processing of a timed job.

[1] Under "field length", specify the number of data words occupied by a parameter!

7.4.2 Parameter Assignment using the PG Screen Form

With the PG system software, screen forms are available for assigning parameters in DX 0 for the CPU 948. The PG software automatically generates data block DX 0 according to the default parameters (values in bold face)and parameters you have specified. To assign parameters to DX 0, two screen forms are required.

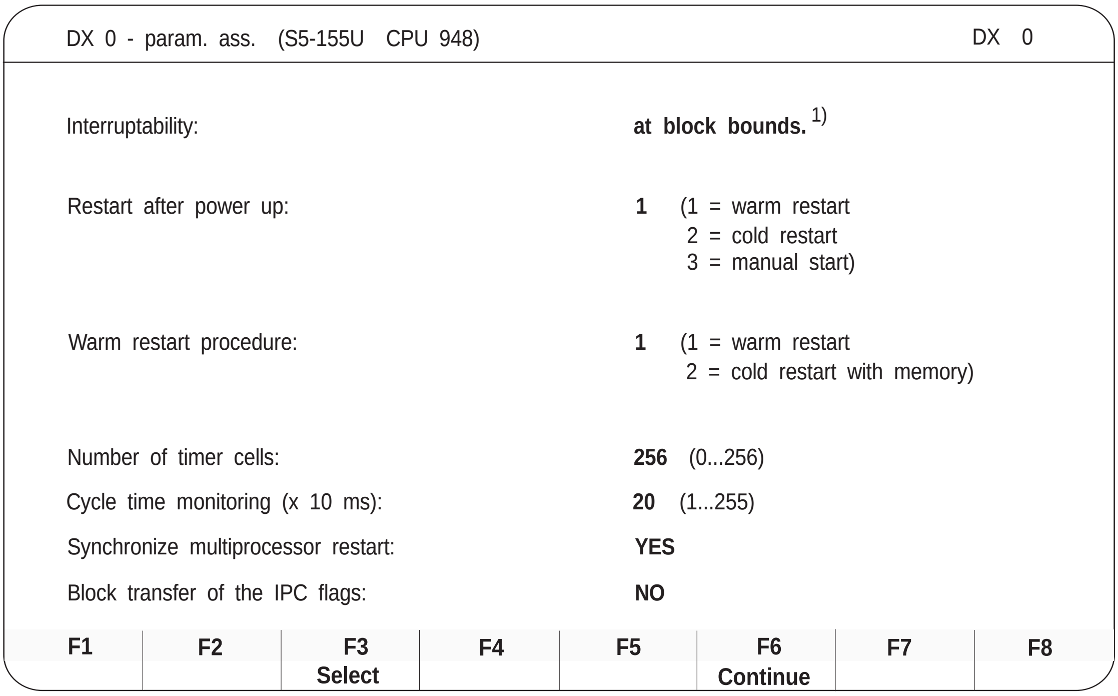
How to select and complete the PG screen forms is explained in your STEP 5 manual.

Structure of the screen forms

Two screen forms are required for completing parameter assignment to DX 0:

The first screen form (Fig. 7-2) contains the parameter groups

- Interruptability,
- Restart after power up,
- Warm restart procedure,
- Number of timer cells,
- Cycle time monitoring,
- Synchronize multiprocessor restart,
- Block transfer of the IPC flags.



DX 0 - param. ass. (S5-155U CPU 948) DX 0

Interruptability: **at block bounds.** [1)]

Restart after power up: **1** (1 = warm restart
2 = cold restart
3 = manual start)

Warm restart procedure: **1** (1 = warm restart
2 = cold restart with memory)

Number of timer cells: **256** (0...256)

Cycle time monitoring (x 10 ms): **20** (1...255)

Synchronize multiprocessor restart: **YES**

Block transfer of the IPC flags: **NO**

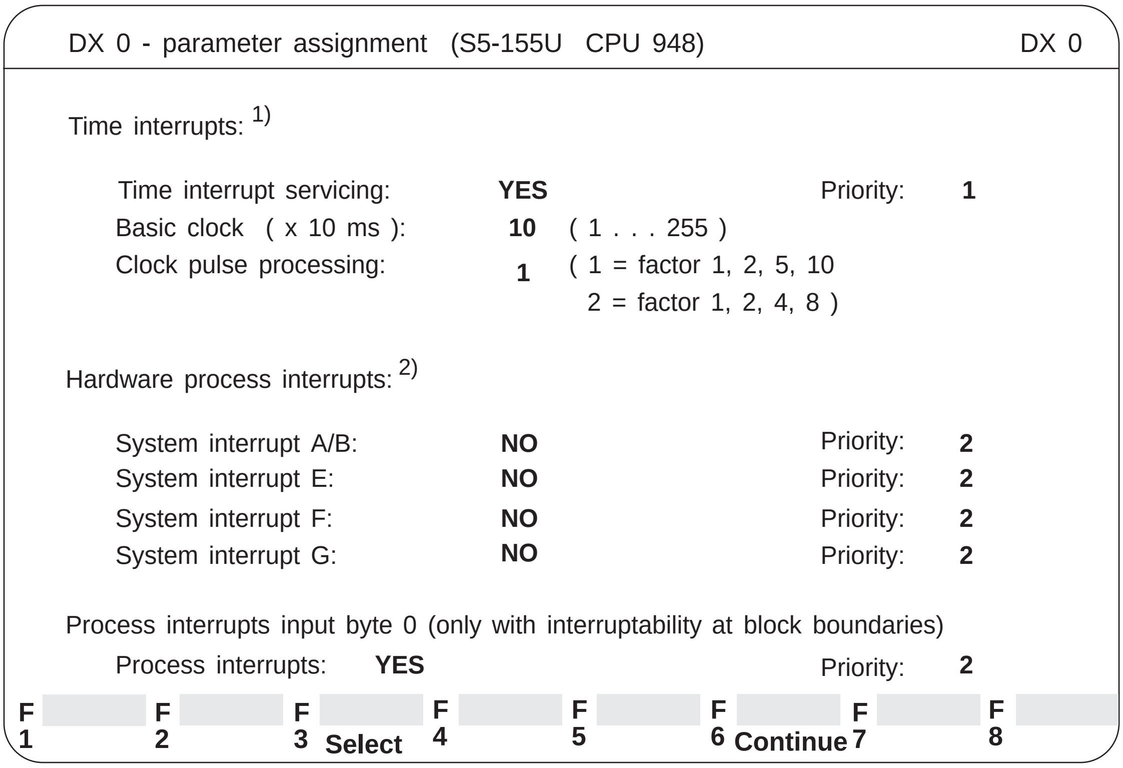
F1	F2	F3	F4	F5	F6	F7	F8
		Select			Continue		

[1)] with older PG-software versions the following is displayed:
Mode 150U for "interruptability at block boundaries"
Mode 155U for "interruptability at operation boundaries"

Fig. 7-2 PG screen form for assigning parameters to DX 0 / Part 1

If you move on to the second screen form (Fig. 7-3) you will find the following parameters:

- Time interrupts,
- Hardware process interrupts,
- Process interrupts input byte 0.



DX 0 - parameter assignment (S5-155U CPU 948) DX 0

Time interrupts: [1]

Time interrupt servicing: **YES** Priority: **1**
Basic clock (x 10 ms): **10** (1 . . . 255)
Clock pulse processing: **1** (1 = factor 1, 2, 5, 10
2 = factor 1, 2, 4, 8)

Hardware process interrupts: [2]

System interrupt A/B: **NO** Priority: **2**
System interrupt E: **NO** Priority: **2**
System interrupt F: **NO** Priority: **2**
System interrupt G: **NO** Priority: **2**

Process interrupts input byte 0 (only with interruptability at block boundaries)
Process interrupts: **YES** Priority: **2**

[1] The delayed interrupt and clock-controlled interrupt must, if necessary, be activated extra by switching off process interrupts (interrupt servicing on)

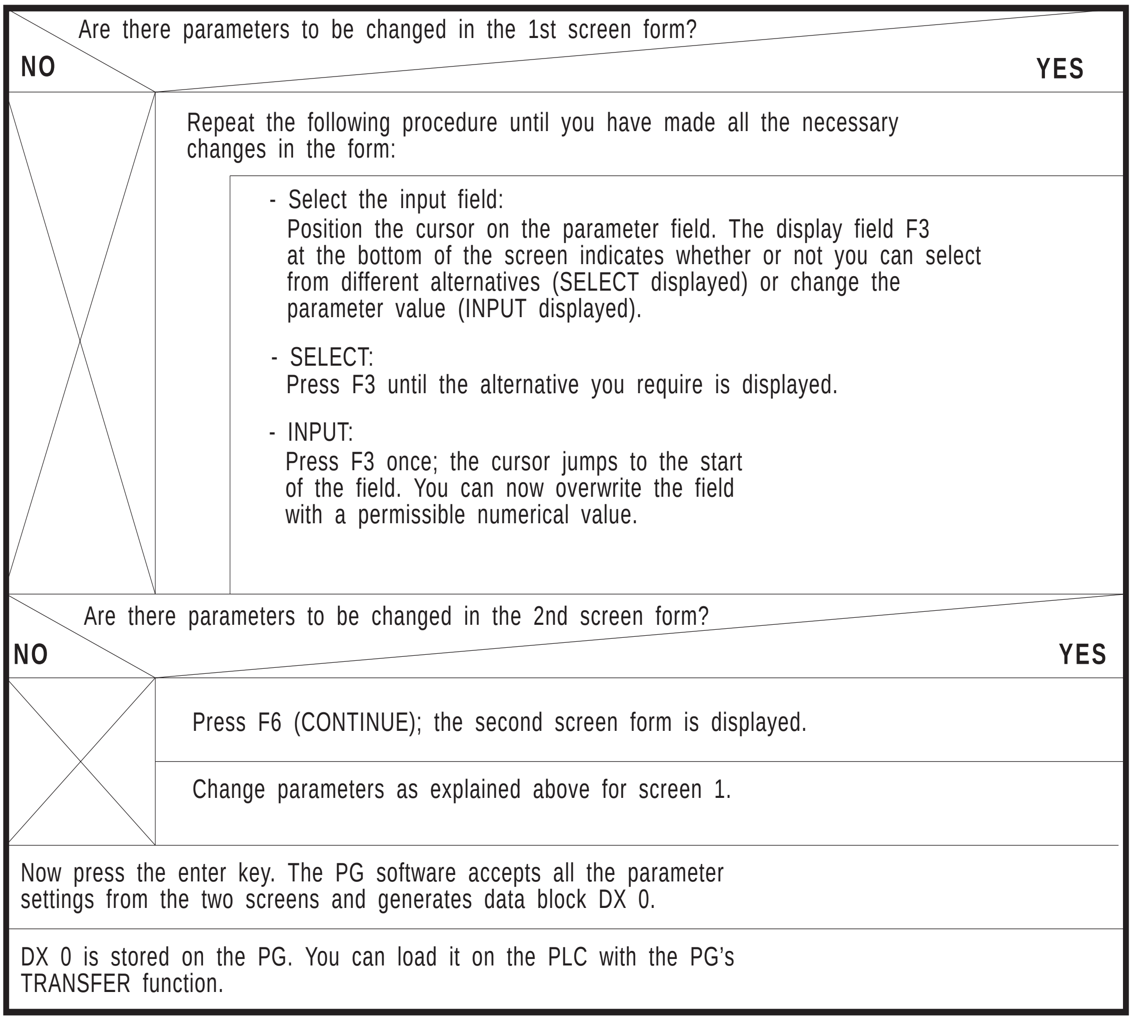
[2] CPU 948: System interrupts can be serviced with "interruptability at block boundaries" or "interruptability at operation boundaries"

Fig. 7-3 PG screen form for assigning parameters to DX 0 / Part 2

Using the screen forms

The following flow chart explains how to complete the screen forms and fields, how to save the parameters and load a generated DX 0 data block.

Flow chart for completing the DX 0 screen forms



Are there parameters to be changed in the 1st screen form?

NO

YES

Repeat the following procedure until you have made all the necessary changes in the form:

- Select the input field:
  Position the cursor on the parameter field. The display field F3 at the bottom of the screen indicates whether or not you can select from different alternatives (SELECT displayed) or change the parameter value (INPUT displayed).
- SELECT:
  Press F3 until the alternative you require is displayed.
- INPUT:
  Press F3 once; the cursor jumps to the start of the field. You can now overwrite the field with a permissible numerical value.

Are there parameters to be changed in the 2nd screen form?

NO

YES

Press F6 (CONTINUE); the second screen form is displayed.

Change parameters as explained above for screen 1.

Now press the enter key. The PG software accepts all the parameter settings from the two screens and generates data block DX 0.

DX 0 is stored on the PG. You can load it on the PLC with the PG's TRANSFER function.

Example

You want to assign parameters in DX 0 to achieve the following system program response (different from the defaults):

- mode "interrupts at operation boundaries",
- no timer updating,
- cycle monitoring time = 2.5 seconds,
- level priority for timed interrupts = 2
- system interrupt INT E with priority = 1 .

Continued on the next page

Continuation of the example

Complete the screen form as follows to obtain this response:

First DX 0 screen form:

- For the MODE parameter, select "interruptability at operation boundaries" with function key F3.
- For the parameter NUMBER OF TIMER CELLS first press function key F3 and then type in the number 0 (= **no** timer).
- For the CYCLE MONITORING parameter, first press function key F3 and then type in the number 250 (= 2.5 seconds).
- Press function key F6 (CONTINUE). The second DX 0 screen form is displayed.

Second DX 0 screen form:

- For the TIMED INTERRUPT/PRIORITY parameter, select the value ’2’ with function key F3.
- For the SYSTEM INTERRUPT E parameter, select the setting ’yes’ with function key F3.
- For the SYSTEM INTERRUPT E/PRIORITY parameter, select the value ’1’ with function key F3.
- For the PROCESS INTERRUPTS parameter, select the setting ’no’ with function key F3.
- Press the enter key to confirm your input. Data block DX 0 is then generated by the system software.

Memory Assignment and Memory Organization

8

Contents of Chapter 8

8 Memory Assignment and Memory Organization

You can use this chapter as a reference section to check on the organization of the CPU 948 memory. The chapter also includes important information contained in some of the system data words.

8.1 Structure of the Memory Area

The memory of the CPU 948 is essentially divided into the following areas:

Table 8-1 Structure of the memory area

Memory area		Data width	Location
User memory for:	OBs, FBs, FXs, PBs, SBs, DBs, DXs	16 bits	CPU internal
Serial communications interface area:	RI, RJ	16 bits	
System area:	RS, RT	16 bits	
Timers:	T	16 bits	
Counters:	C	16 bits	
Flags:	F	8 bits	
Flags:	S	8 bits [1]	
Process image (PI) inputs/ outputs:	PII, PIQ	8 bits	
Peripheral area, divided into:			On the S5 bus
	"P" peripherals	8 bits	
	"O" peripherals	8 bits	
	Interprocessor communication flags	8 bits	
	Coordinator (COR) (semaphore, ...)	8 bits	
	Dual-port RAM pages (CP, IP, COR 923C)	8/16 bits	
	Distributed peripherals	8 bits	
	Hardware registers	8/16 bits	

[1] S flags occupy 8 bits in the 16-bit area. The high byte is undefined.

The next section lists the addresses of the memory areas shown.

Note
When using STEP 5, you should not access a memory register within an operand area (e.g., flags) directly via the absolute address of the memory register. This can result in undesirable operating statuses. Access it only relative to the base address of its operand area.

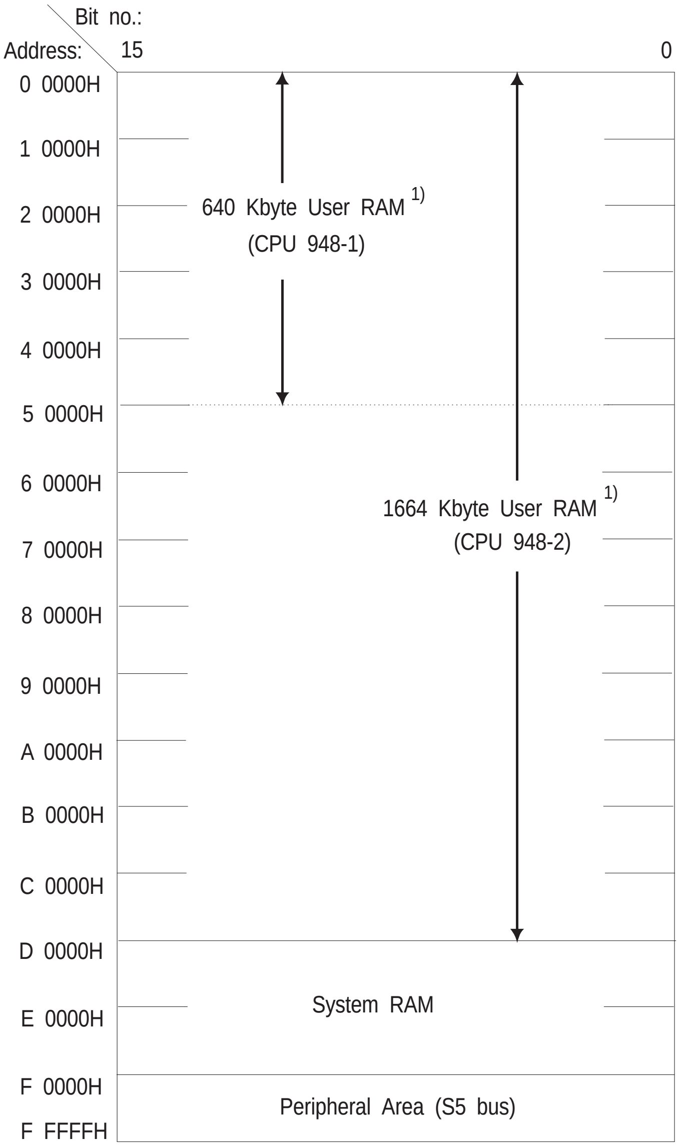
Direct access to the areas I, Q and F result in ’FFH’ in the high byte and the data in the low byte. For direct access to S flags, the **high byte is undefined**!

8.2 Memory Assignment in the CPU 948

With the CPU 948, there are two possible user memories (RAM) available:

- the CPU 948-1 with 640 Kbytes of user memory
- the CPU 948-2 with 1664 Kbytes of user memory.

Fig. 8-1 illustrates the distribution of the address area of the CPU 948 and the location of the user memory versions.



[1] The last 20 words of the user RAM cannot be used.

Fig. 8-1 Memory assignment in CPU 948/overview

8.2.1 Memory Assignment for the System RAM

Address:	Bit no.: 15 … 8	7 … 0	
D 0000H E 9FFFH	System program and system data		
E A000H E AFFFH	☒	S flags	
E B000H E DEAFH	System program data		
E DEB1H	BSTACK (60 entries)		
E DF6FH	Reserved		
E DF7CH	ISTACK entry - 1		
E DFA1H	ISTACK (16 entries)		
E E1F0H	Reserved		
E E1FBH	DB 0 header		DB 0 (contains the paragraph addresses of all blocks, i. e. address bit no. 4 to address bit no. 19)
E E200H	Address list OB 0 to OB 255		
	Reserved		
E E400H	Address list PB 0 to PB 255		
	Reserved		
E E600H	Address list SB 0 to SB 255		
	Reserved		
E E800H	Address list FB 0 to FB 255		
	Reserved		
E EA00H	Address list FX 0 to FX 255		
	Reserved		
E EC00H	Address list DB 0 to DB 255		
	Reserved		
E EE00H	Address list DX 0 to DX 255		
	Reserved		

Fig. 8-2 Memory assignment for the system RAM/Part 1

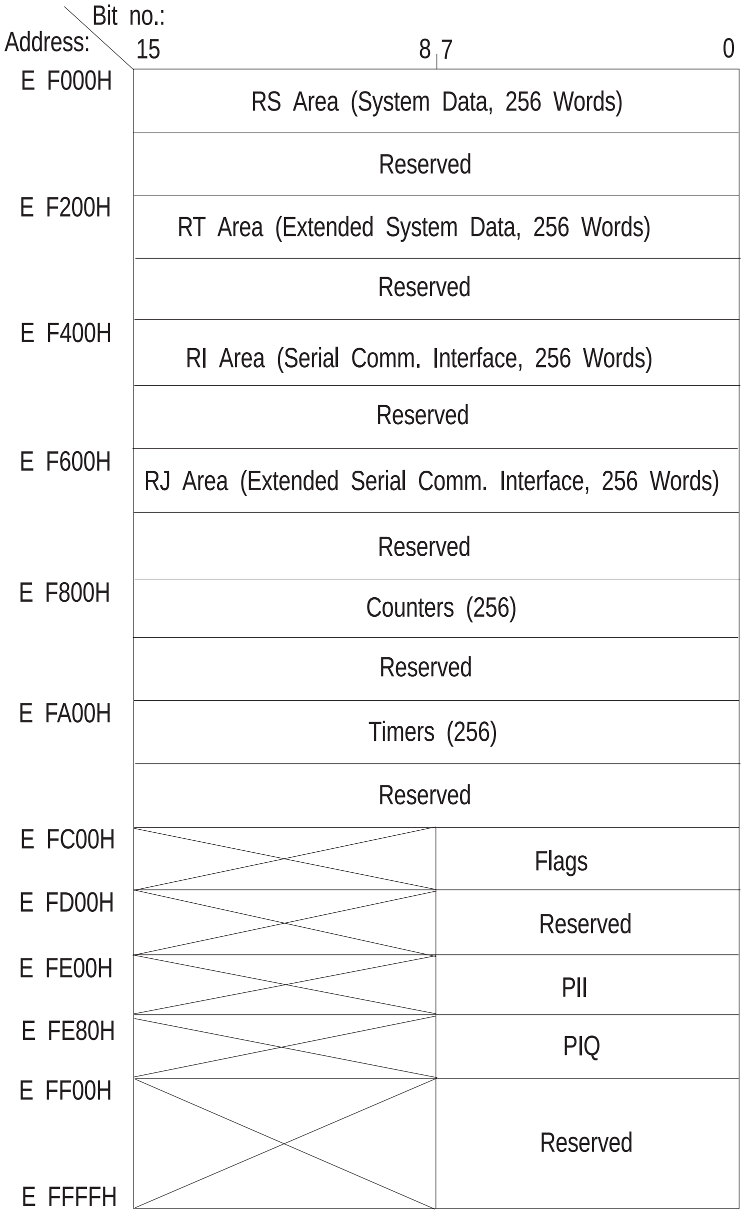


Fig. 8-3 Memory assignment of the system RAM/Part 2

8.2.2 Memory Assignment for the Peripherals

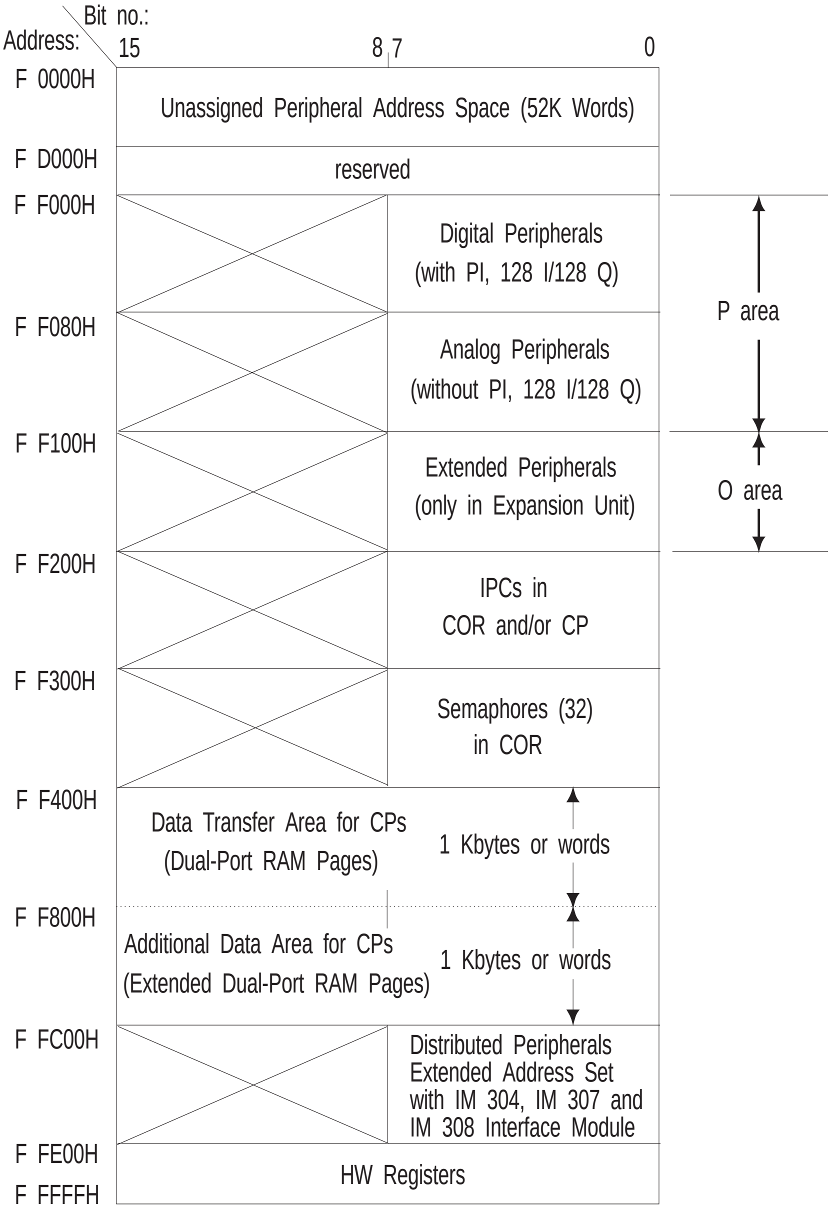

Bit no.:
Address:
15
8
7
0
F 0000H
Unassigned Peripheral Address Space (52K Words)
F D000H
reserved
F F000H
Digital Peripherals (with PI, 128 I/128 Q)
F F080H
Analog Peripherals (without PI, 128 I/128 Q)
P area
F F100H
Extended Peripherals (only in Expansion Unit)
O area
F F200H
IPCs in COR and/or CP
F F300H
Semaphores (32) in COR
F F400H
Data Transfer Area for CPs (Dual-Port RAM Pages)
1 Kbytes or words
F F800H
Additional Data Area for CPs (Extended Dual-Port RAM Pages)
1 Kbytes or words
F FC00H
Distributed Peripherals Extended Address Set with IM 304, IM 307 and IM 308 Interface Module
F FE00H
HW Registers
F FFFFH

Fig. 8-4 Address areas for peripherals (8 bits) on the S5 bus

Address Areas for Peripherals and Programming Them

Area (absolute address)		Referenced with	Parameter
E FE00 E FE7F	PII (Process image input)	L IB / T IB	0 to 127
		L IW / T IW	0 to 126
		L ID / T ID	0 to 124
		A I/ AN I / O I / ON I S I / R I / = I	0.0 to 127.7
E FE80 E FEFF	PIQ (Process image output)	L QB / T QB	0 to 127
		L QW / T QW	0 to 126
		L QD / T QD	0 to 124
		A Q / AN Q / O Q / ON Q S Q / R Q / = Q	0.0 to 127.7
"P" peripherals with process image		When the operation is processed, only the process image is changed. The new status of the process image of the outputs is only output to the I/Os at the end of the cycle.	
F F000 F F07F	Digital peripherals Inputs/outputs	L PY / T PY	0 to 127
		L PW / T PW	0 to 126
F F080 F F0FF	Digital or analog peripherals Inputs/outputs	L PY / T PY	128 to 255
		L PW / T PW	128 to 254
"P" peripherals		The inputs and outputs are addressed directly in bytes or words.	
F F100 F F1FF	Extended peripherals Inputs/outputs	L OY / T OY	0 to 255
		L OW / T OW	0 to 254
"O" peripherals		The inputs and outputs are addressed directly in bytes or words.	

Using STEP 5 operations, you can access peripherals either directly or via the process image (PI). Note that a process image exists only for input and output bytes of the "P" peripherals with byte addresses from 0 to 127!

Note
Using the interface modules IM 304, IM 307 and IM 308, you can access distributed address areas using your program. This allows access to two new address areas similar to the O area. In contrast to the O area, however, access to these areas is only possible using absolute addressing or using FB 196 of the "basic functions" software package (refer to Catalog ST59).

8.3 User Memory Organization in the CPU 948

Depending on the version of the CPU 948 used, the user memory occupies the memory area from 0 0000H to C FFFFH. When you load the individual blocks of your program, they are stored in the memory in random order (with addresses in ascending order).

Block management

When you correct a block, the old block in the memory is declared invalid (i.e., the start ID is overwritten) and a new block is entered in the memory and the address list. This also applies when you delete blocks. The blocks are not really deleted in the memory but simply declared invalid. Gaps created by deleting are managed as available memory locations and are used again when you load new blocks.

Compressing memory

The online COMPRESS MEMORY PG function pushes all valid blocks in the memory together. When you activate the COMPRESS MEMORY while the CPU is in the STOP mode, all blocks that are not directly next to each other are shifted. However, when you activate this function while the CPU is in the RUN mode, long data and extended data blocks (i.e., longer than 512 data words) are not shifted because of data length consistency. Compressing produces large available memory areas which you can use for loading new blocks.

If the online COMPRESS MEMORY PG function is interrupted (e.g., when the power is turned off), compressing is terminated and does not resume automatically when power is turned on.

Location of blocks in the user memory

In the CPU 948, blocks are stored so that data word DW 0 or the first STEP 5 statement of each block is located at a **paragraph address**. Paragraph addresses are at 16-word boundaries. Therefore, all blocks begin in the memory at the address xxxxBH (bit no. 0 to 3 = BH) and all block bodies at the address yyyy0H (bit no. 0 to 3 = 0H). The gaps that result between blocks are filled in by invalid data blocks, so that all blocks continue to exist in consecutive order.

Filler blocks

These invalid data blocks are known as "filler blocks". They are treated in the same way as the other blocks by the memory management and have the following structure:

Start ID:	7070H	;
Block type/block number:	01FBH	;DB 251 invalid
Programmer ID:	00FFH	;irrelevant
Library number:	FFFFH	;irrelevant
Block length:	00XXH	;length 5 to 20 words
Data:	FFFFH	;according
	:	;to
	:	;length;
	FFFFH	;can be left out entirely

Example

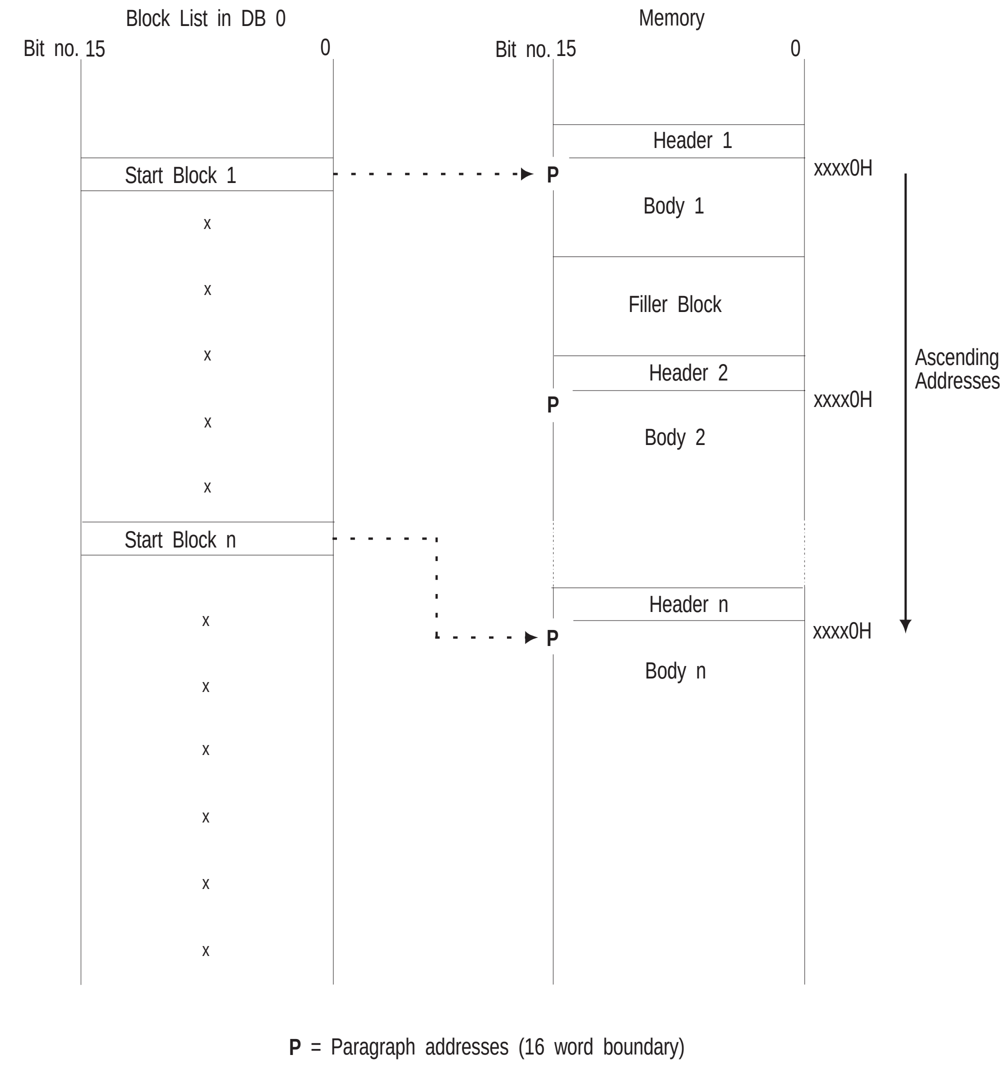


Fig. 8-5 Example: Location of blocks in memory

You can calculate the length of a filler block by finding the difference between the end address of the last block stored and the address before the next paragraph address.

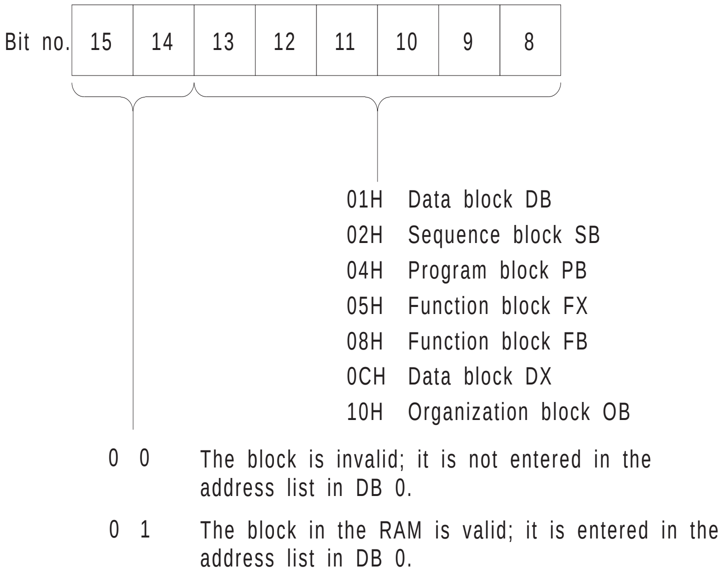
Difference	Calculation for length of filler block (including header)
0 to 5	Add 10 to the difference
6	No filler block is inserted
7 to 10	Add 10 to the difference
11 to 15	Subtract 6 from the difference

8.3.1 Block Headers in User Memory

Each block in the memory begins with a header that is five words long. The block header is divided as follows:

1st word: Block start ID: 7070H

2nd word: High byte = Block type



01H	Data block DB
02H	Sequence block SB
04H	Program block PB
05H	Function block FX
08H	Function block FB
0CH	Data block DX
10H	Organization block OB

0 0	The block is invalid; it is not entered in the address list in DB 0.
0 1	The block in the RAM is valid; it is entered in the address list in DB 0.

Low byte = Block number
The low byte of the second header word contains the block number (0 to 255). It is coded as a hexadecimal number: 00 to FFH.

3rd word: The high byte of the third word contains the IDs for the programmer. The low byte contains part of the library number.

4th word: The fourth word contains the rest of the library number.

5th word: The fifth word (low and high bytes) contains the length of the block, including the block header. The length is indicated in words.

8.3.2 Block Address List in Data Block DB 0

Data block DB 0 is located in the system RAM of the CPU (beginning at address E E200H). It contains a list with the start addresses of all blocks in the user memory of the CPU. The system program generates (COLD RESTART) or checks (WARM RESTART) this list after power up; it updates it automatically when you use a programmer to enter or change blocks.

Address list start addresses

DB 0 has a separate, reserved address list of 256 words in each type of block. Blocks that are not loaded or have been deleted have the start address ’0’.
The start addresses of each block address list are specified (see Section 8.2.1).

Block start addresses

The block start addresses in the address lists always point to the first word **after** the block header:

- with data blocks, to data word DW 0
- with logic blocks, to the first STEP 5 statement (in FBs to the ’JU’ operation before the name and the parameter list).

Since each block is located at a paragraph address (16 word boundary), each address list entry in DB 0 is restricted to one word with bits number 4 to 19 of the address.

Location of block addresses in DB 0

n = E E400H (start address of the PB address list)

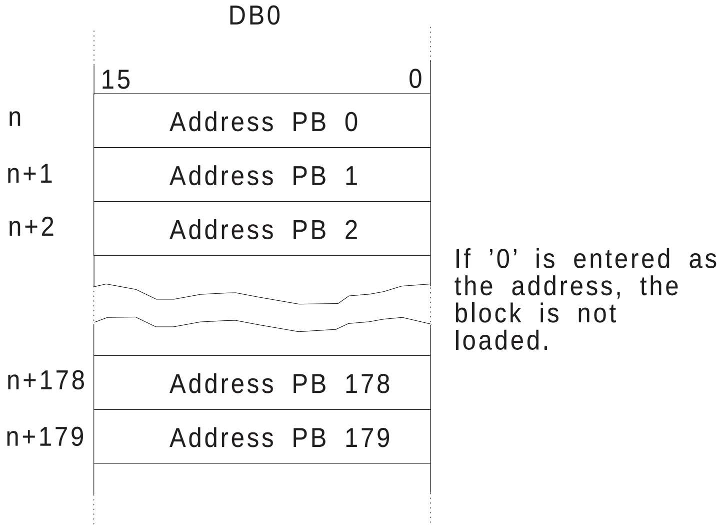


Fig. 8-6 Block addresses in DB 0

Example of how to obtain a block address

> The block start addresses of the **program blocks** are located in DB 0 and begin at address E E400H. The start address of PB 22 can therefore be read out by accessing memory at address E E416H (= start address of the PB + 16H).

8.3.3 RI/RJ Area

The RI area is an area that is 256 words long in the internal system RAM of the CPU. RI occupies addresses E F400H to E F4FFH.

The RJ area is an area that is 256 words long in the internal system RAM of the CPU. RJ occupies addresses E F600H to E F6FFH.

You can use the entire RI area (RI 0 to RI 255) and the entire RJ area (RJ 0 to RJ 255) for your own purposes.

The RI/RJ area is only filled with zeros following OVERALL RESET.

8.3.4 RS/RT Area

The RS and RT areas contain information for the system programmer and system internal data.

The **RS area** is an area that is 256 words long in the internal system RAM of the CPU. RS occupies addresses E F000H to E F0FFH.

> ⚠
>
> **Caution**
> **You should only write to system data words RS 60 to RS 63:**
>
> **All other system data should only be read:**
>
> Writing to the system data area can affect the functional capability of your programmable controller and connected programmers: serious disturbances can occur which may put both people and machines in danger.

The **RT area** is an area that is 256 words long in the internal system RAM of the CPU. RT occupies addresses E F200H to E F2FFH.

You can use the whole RT area (RT 0 to RT 255) for your own purposes if

1. you do not use standard FBs

and

2. you do not use PG functions via SINEC H1 and the parallel S5 bus.

Only an overall reset can clear the RS/RT areas.

Using the online function SYSTEM PARAMETERS, you can obtain the information contained in some of the system data (about the internal structure of the CPU, the software release, the CPU identifier etc.)

Assignment of the RS area

Table 8-2 Assignment of the RS area

RS	Name	Address
0	Input byte IB 0 image table (external process interrupts)	E F000H
1	External process interrupts in the processing queue (IB 0)	E F001H
2 to 4	System program	
5	Current cycle time	E F005H
6	System program	
7	STOP mode ID (ISTACK)	E F007H
8	Start/restart IDs (ISTACK)	E F008H
9 to15	System program	
16	Error area: output bytes 0 to 15	E F010H
17 to 23	Error area: output bytes 16 to 127	E F011H to E F017H
24 to 31	Error area: input bytes 0 to 127	E F018H to E F01FH
32 to 47	Error area: interprocessor communication flag bytes 0 to 255	E F020H to E F02FH
48 to 49	System program	
50	PAFE byte for "backplane bus functions"	E F032H
51 to 59	System program	
60 to 63	Available to user	E F03CH to E F03FH
64 to 67	System program	
68 to 71	Error address with QVZ and PARE errors	E F044H to E F047H
72 to 74	System program	

RS	Name	Address
Table 8-2 continued:		
75	System message, function number	E F04BH
76	System message, parameter1	E F04CH
77	System message, parameter 2	E F04DH
78	System message, parameter 3	E F04EH
79 to 95	System program	
96	Current time of day (seconds)	E F060H
97	Current time of day (hours)	E F061H
98	Current time of day (days)	E F062H
99	Current time of day (year/month)	E F063H
100 to 119	System program	
120	Software protection/password	E F078H
121 to 135	System program	
136 to 137	Locations for self-test function	E F088H to E F089H
138	System program	
139	Cycle time used after retriggering	E F08BH
140 to 252	System program	
253	free for distributed periphery	E F0FDH
254 to 255	System program	

As a supplement to the listing above, the following pages provide the bit assignments of a few system data registers that you can evaluate via STEP 5 operations or with your programmer (see Section 5.4 for information on the abbreviations).

8.3.5 Bit Assignment of the System Data Words

System data RS 0

Input byte IB 0 Image table (external process interrupts)

Address E F000H

Table 8-3 Bits in RS 0 (image of IB 0)

High byte	
Bit no.	Assignment
15	Occupied by system program
14	
13	
12	
11	
10	
9	
8	
Low byte	
7	Status of I 0.7
6	Status of I 0.6
5	Status of I 0.5
4	Status of I 0.4
3	Status of I 0.3
2	Status of I 0.2
1	Status of I 0.1
0	Status of I 0.0

System data RS 1

Condition code of external process interrupts currently in processing queue

Address: E F001H

Table 8-4 Bits of RS 1 (current process interrupts)

High byte	
Bit no.	Assignment
15	All the bits have the value ’0’
14	
13	
12	
11	
10	
9	
8	
Low byte	
7	Bit = ’1’: edge I 0.7
6	Bit = ’1’: edge I 0.6
5	Bit = ’1’: edge I 0.5
4	Bit = ’1’: edge I 0.4
3	Bit = ’1’: edge I 0.3
2	Bit = ’1’: edge I 0.2
1	Bit = ’1’: edge I 0.1
0	Bit = ’1’: edge I 0.0

System data RS 5

Current cycle time

Address: E F005H

Table 8-5 Bits of RS 5 (cycle time)

High byte and low byte	
Bit no.	Assignment
15	The entered binary value * 10 msec. equals the cycle time of the cycle processed last
14	
13	
12	
11	
10	
9	
8	
7	
6	
5	
4	
3	
2	
1	
0	

Example

Bit no.	15	14	13	12	11	10	9	8	7	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0

The time of the last cycle is as follows:

$(2^4 + 2^3) * 10 \text{ ms} = (16 + 8) * 10 \text{ ms} = 240 \text{ ms}$

System data RS 7

Programmable controller STOP mode IDs (ISTACK)

Address: E F007H

Table 8-6 Bits of RS 7 (PLC stop IDs)

High byte	
Bit no.	Assignment
15	Reserved
14	
13	
12	Faulty ISTACK level
11	Illegal start-up type (UANL)
10	Interruption in stop loop
9	Illegal call of system block (SYSFHL)
8	Error in start-up block (AFEL)
Low byte	
7	Interruption by system (USYS –warm restart possible)
6	Interruption by programming error (UPROG – cold restart)
5	"End of program test" (BEARBE)
4	Stop switch (STOPS)
3	End of operation stop (STS)
2	End of cycle stop (STP)
1	Multiprocessing stop (HALT)
0	PG stop (PGSTP)

System data RS 8

Start and restart IDs (ISTACK)

Address: E F008H

Table 8-7 Bits of RS 8 (start and start-up IDs)

High byte	
Bit no.	Assignment
15	Default: MANUAL COLD RESTART/ WARM RESTART (MSEG)
14	Default: AUTOMATIC COLD RESTART (ANEG)
13	Default: AUTOMATIC WARM RESTART (AWEG)
12	OVERALL RESET required (URLER)
11	WARM RESTART permitted (WIEZU)
10	COLD RESTART permitted (NEUZU)
9	OVERALL RESET executed (URLDF)
8	WARM RESTART executed (WIEDF)
Low byte	
7	COLD RESTART executed (NEUDF)
6	Automatic start after NAU
5	Manual start
4	COLD RESTART WITH MEMORY
3	PG overall reset
2	PG system start
1	PG warm restart
0	PG cold restart

System data words RS 16 to RS 47

Error areas

RS xx	Address(es)	Error area
RS 16	E F010	Output bytes 0 to 15
RS 17 to RS 23	E F011 to E F017	Output bytes 16 to 127
RS 24 to RS 31	E F018 to E F01F	Input bytes 0 to 127
RS 32 to RS 47	E F020 to E F02F	Interprocessor communication flag bytes 0 to 255

RS 16

Address: E F010H

Table 8-8 Bits of RS 16 (error area output bytes 0 to 15)

High byte	
Bit no.	Assignment
15	Output byte 0
14	Output byte 1
13	Output byte 2
12	Output byte 3
11	Output byte 4
10	Output byte 5
9	Output byte 6
8	Output byte 7
Low byte	
7	Output byte 8
6	Output byte 9
5	Output byte 10
4	Output byte 11
3	Output byte 12
2	Output byte 13
1	Output byte 14
0	Output byte 15

If errors appear during update of the process image input/output tables or interprocessor communication flags, the corresponding bits are set to ’1’. – The system data words RS 17 to 47 are structured analogous to RS 16.

Example of RS 16

```
The content of system data register RS 16  is

   "8020" hexadecimal or "1000 0000 0010 0000"
   binary.


The process image for output bytes 0 and 10 has
not been updated correctly.
```

System data RS 50

PAFE byte for "backplane bus functions"

System data RS 50 contains the parameter assignment error byte for the "backplane bus functions" of a CPU.

	15 8	7 0	
RS 50	PAFE	-	E F032H

System data words RS 68 to RS 71

Error addresses of QVZ and PARE errors

If a QVZ or PARE error occurs, the address at which the error was detected is entered here.

	15 0	
RS 68	QVZ error addr. high	E F044H
RS 69	QVZ error addr. low	E F045H
RS 70	PARE error addr. high	E F046H
RS 71	PARE error addr. low	E F047H

System data words RS 75 to RS 78

System message

The entries in system data words RS 75 to RS 78 refer to the error that occurred last. The message consists of four system data words with the following structure:

	15	0	
RS 75	Error number	Parameter type	E F04BH
RS 76	Parameter 1		E F04CH
RS 77	Parameter 2		E F04DH
RS 78	Parameter 3		E F04EH

RS 75

The **high byte** contains the **error number** which classifies the error. The error number assigns the error to one of the following three areas:

Error groups

• 01H to 2FH:	user error
• 30H to 3FH:	error in DX 0 or DB 1,
• 40H	system error

Parameter type

The **low byte** contains the **parameter type** that describes the structure of the succeeding parameter block in RS 76 to RS 78. The parameter types from 00H to 10H exist. The structure of the parameter field is described later.

The two following tables list the error groups "general errors" and "errors in DX 0 or DB 0".

General errors

Table 8-9 RS 75: general errors

Error number	Parameter type	Meaning
01H	01H	Block called is not loaded
02H	01H	Addressing error
03H	01H	Cycle time error
04H	01H	Substitution error
05H	02H	Timeout distributed peripherals
06H	03H	Timeout user memory
07H	01H	Load/transfer error with data blocks and extended data blocks
08H	01H	Bracket counter overflow
09H	04H	Data block to be opened does not exist
0AH	05H	Error with internal time interrupts/ other interrupts
0BH	03H	Timeout page frame area
0CH	03H	Timeout global communication area
0DH	07H	Timeout in process image updating
0EH	03H	Timeout in IPC flag synchronization error
0FH	03H	Timeout "P"/"O" peripherals
10H	03H	Timeout since interface module (IM 3/IM 4) missing
11H	03H	Parity error in user memory
12H	01H	Timeout process image transfer
13H	01H	Timeout input byte IB 0
14H	01H	BSTACK overflow
15H	01H	STS operation
16H	01H	RUN/STOP switch set to STOP position
17H	01H	Halt signal from the coordinator
18H	01H	Not used
19H	01H	Load/transfer error with L BY/T BY (process image update)
1AH	01H	Load/transfer error during addressing via the BR register
1BH	01H	I/Os not ready
1CH	08H	Timeout/parity error during initialization

Error number	Parameter type	Meaning
Table 8-9 continued:		
1DH	08H	AUTOMATIC WARM RESTART not possible; COLD RESTART required
1EH	00H	Illegal start-up type
1FH	01H	Load/transfer error during block transfer operation (incorrect memory area boundaries with TNW, TXB, TXW)
20H	09H	Illegal length for G DB/GX DX
21H	09H	DB/DX already exists for G DB/GX DX
22H	09H	Memory space insufficient for G DB/GX DX
23H	05H	Masked system interrupt is coming through
24H	00H	Block function (compress, transfer, input) required in STOP, COLD RESTART
25H	00H	Battery failure; no start-up possible
26H	00H	Change from single to multiprocessor operation prevents a WARM RESTART
27H	01H	STOP caused by STP operation
28H	01H	Continuous ready signal (I/O module defect))
29H	08H	Error in DB 0 struct. after OVERALL RESET

DX 0/DB1 errors

Table 8-10 RS 75: Errors in DX 0 or DB 1

Error number	Parameter type	Meaning
Errors in DX 0		
30H	00H	No DX 0 in multiprocessing
32H	00H	Proc. int. and sys. int. selected simultaneously
33H	00H	Interrupts and mode incompatible
34H	06H	Invalid DX 0 header
35H	07H	Error in DX 0 block ID
36H	07H	Error in DX 0 parameter
Errors in DB 1		
38H	06H	No DB 1 in multiprocessing
39H	06H	Invalid DB 1 header
3AH	07H	DB 1 ID set more than once
3BH	07H	DB 1 byte offset without ID

Error number	Parameter type	Meaning
Table 8-10 continued		
3CH	07H	Peripheral entered in DB 1 is not plugged in
3DH	07H	Offset too big (parameter error)
3EH	07H	Too many offsets

Error codes of the self test functions

Table 8-11 RS 75: error codes of the self test functions

Error number	Parameter type	Meaning
61H	0BH	Checksum error in system program code
62H	0AH	Checksum error in code of the STEP 5 logic blocks
63H	0BH	Address decoder error
64H	0CH	Error testing the user memory organized in words
65H	0CH	Error testing the user memory organized in bytes
66H	00H	Error testing cycle time monitoring
67H	00H	Error testing the BASP signal
68H	00H	Error testing the hardware clock

Structure of the parameter field (RS 76 to RS 78)

Table 8-12 RS 76 to RS 78: Parameter types

Parameter type	Structure of the parameter field		
00H	No parameter; parameter 1, 2, 3 = 0		
01H	Parameter 1:	block type/block number (IDs from block header)	
	Parameter 2:	operation that caused an interruption	
02H	Parameter 1:	interface module number (IM 302) (distributed periphery)	
	Parameter 2:	number of the defective interface	
	Parameter 3:	incorrect byte offset	
03H	Parameter 1:	not used	
	Parameter 2:	error address high	
	Parameter 3:	error address low	
04H	Parameter 1:	block type/block number (IDs from block header)	
	Parameter 2:	operation that caused the interruption	
	Parameter 3:	number of the DB/DX to be opened	
05H	Parameter 1:	The following bits are set depending on the error:	
		Bit no. 0	= 1: timed interrupt/period 1
		Bit no. 1	= 1: timed interrupt/period 2
		: :	: :
		Bit no. 7	= 1: timed interrupt/period 8
		Bit no. 8	= 1: timed interrupt/period 9
		Bit no. 9	= 1: clock masked (ignored) for too long = 0: queue overflow
		Bit no. 10	reserved
		Bit no. 11	reserved
		Bit no. 12	= 1: interrupt G
		Bit no. 13	= 1: interrupt F
		Bit no. 14	= 1: interrupt E
		Bit no. 15	= 1: interrupt X
06H	Parameter 1	data word expected in DX 0 or DB 1 header	
	Parameter 2	actual data word in DX 0 or DB 1 header	
07H	Parameter 1	block ID or code word in DX 0 or DB 1	
	Parameter 2	incorrect parameter 1 in DX 0 or DB 1 (FFFFH: parameter irrelevant)	
	Parameter 3	incorrect parameter 2 in DX 0 or DB 1 (FFFFH: parameter irrelevant)	

Parameter type	Structure of the parameter field		
Table 8-12 continued:			
08H	Parameter 1	The following bits are set depending on the error:	
		Bit no. 0	= 1: QVZ in initialization
		Bit no. 1	= 1: PARE in initialization
		Bit no. 2	= 1: content of the memory card too large
		Bit no. 3	= 1: operating system error
		Bit no. 4	= 1: incorrect block ID
		Bit no. 5	= 1: incorrect block delimiter
		Bit no. 6	reserved
		Bit no. 7	reserved
		Bit no.8	= 1: DB 0 changed since last COLD RESTART
		Bits no. 9 to 15	reserved
	Parameter 2	error address high, (when bit 2, 4 or 5 of parameter 1 = ’1’	
	Parameter 3	error address low, (when bit 2, 4 or 5 of parameter 1 = ’1’	
09H	Parameter 1	Opcode "GX DX" or "G DB" (indicates the block type)	
	Parameter 2	block number	
	Parameter 3	data block length	
10H	internal system error number		
0AH	Parameter 1	block type/block number (IDs from block header)	
	Parameter 2	expected checksum	
	Parameter 3	actual checksum	
0BH	Parameter 1	FFFFH	
	Parameter 2	error address high with address code error, actual checksum high when testing the system program code	
	Parameter 3	error address low with address code error, actual checksum low when testing the system program code	
0CH	Parameter 1	check pattern when testing the user memory	
	Parameter 2	error address high	
	Parameter 3	error address low	

Example of a system message

RS 75	21H	09H	E F04BH
RS 76	7804H		E F04CH
RS 77	0064H		E F04DH
RS 78	0078H		E F04EH

```
RS 75, error number = 21H:The error occurred in the STEP 5 user program when
                          generating a DB/DX.
RS 75, parameter type=09H:The parameter block in RS 76 to RS 78 contains
                          3 parameters.
RS 76, parameter 1 = 7804H: Opcode = "GX DX", this means block type "DX".
RS 77, parameter 2 = 0064H: Block number = 100 (dec.)
RS 78, parameter 3 = 0078H: Data block length = 120 data words
```

Information contained in the message:

In the STEP 5 user program, data block DX 100 should be generated with a length of 120 data words. However, this already exists.

System data words RS 96 to RS 99

Real-time clock

The current date and time of day are kept in system data areas RS 96 to RS 99 and can if necessary be read out from these locations.

	15	0	
RS 96	Seconds	1/10, 1/100 seconds	E F060H
RS 97	Hours	Minutes	E F061H
RS 98	Day	Day of the week	E F062H
RS 99	Year	Month	E F063H

The clock is updated by a 10 msec. pulse.

RS 96

Seconds and 1/100 seconds (address: E F060H):

Table 8-13 Structure of RS 96 (real-time clock: seconds, 1/100 seconds)

High byte	
Bit no.	Assignment
15	Seconds, tens, permitted: 00H to 05H
14	
13	
12	
11	Seconds, units, permitted: 00H to 09H
10	
9	
8	
Low byte	
7	1/10 second, permitted: 00H to 09H
6	
5	
4	
3	1/100 second, permitted: 00H to 09H
2	
1	
0	

RS 97

Hours and minutes (address: E F061H):

Table 8-14 Structure of RS 97 (real-time clock: hours, minutes)

High byte	
Bit no.	Assignment
15	0 = 12 hour format, 1 = 24 hour format
14	0 = AM, 1= PM
13	Hours, tens, permitted: 00/01H with 12 hour format, 00/02H with 24 hour format
12	
11	Hours, units, permitted: 00H to 09H
10	
9	
8	
Low byte	
7	Minutes, tens, permitted: 00H to 05H
6	
5	
4	
3	Minutes, units, permitted: 00H to 09H
2	
1	
0	

RS 98

Current date and day of the week (address: E F062H):

Table 8-15 Structure of RS 98 (real-time clock: date, day of the week)

High byte	
Bit no.	Assignment
15	Date, tens, permitted: 00H to 03H
14	
13	
12	
11	Date, units, permitted: 00H to 09H
10	
9	
8	
Low byte	
7	Day of the week, permitted: 00H to 06H for Mon. to Sun.
6	
5	
4	
3	0
2	
1	
0	

RS 99

Current year and month (address: E F063H):

Table 8-16 Structure of RS 99 (real-time clock: year, month)

High byte	
Bit no.	Assignment
15	Year, tens, permitted: 00H to 09H
14	
13	
12	
11	Year, units, permitted: 00H to 09H
10	
9	
8	
Low byte	
7	Month, tens, permitted: 00/01H
6	
5	
4	
3	Month, units, permitted: 00 to 09H
2	
1	
0	

System data RS 120

Software protection

System data RS 120 controls the system function "software protection". With this function, you can prevent blocks being read, overwritten or deleted by the PG (e.g. by unauthorized personnel) by specifying a password.

Password

The "software protection" function is linked to a password. The system program is informed of this via RS 120.

Specifying the password/activating protection

By specifying a password in RS 120, the password protection is automatically activated.
You can specify a new password after deleting the old one.

Deleting the password/deactivating protection

If you delete the password, password protection is automatically deactivated.
When the password is deleted, the system program must be informed via RS 120.

Maximum of five attempts to delete

If you attempt to delete the password and specify the wrong password, the attempt is rejected by the system program and a count started.
After a maximum of five unsuccessful attempts, the system program will no longer accept password entries. The password can then only be deleted after a COLD RESTART.
If the password is successfully deleted, the error counter is reset.

How to specify or delete the password

The password is specified/deleted (and the software protection activated/deactivated) by writing a bit pattern in RS 120 (see Assignment when writing) in one of the following ways:

- by the STEP 5 program or
- with a PG job "output address".

Note
When you first receive your CPU and following an overall reset, the password is deleted and the software protection switched off.

When is the software protection activated/deactivated?

A password can be specified at any time. Once it has been specified, software protection is, however, only active at certain times, as shown below:

- in the SOFT STOP mode:
  once after calling OB 38 [1],
  cyclically before calling OB 39 [1],
- in the START-UP mode:
  once after calling the start-up OBs (OB 20, OB 21 OB 22),
- in the RUN mode:
  cyclically before calling OB 1.

Assignment of the system data when writing

To call the software protection function, write system data RS 120 with a bit pattern for the function as shown in the following table.

Address: E F078H

Table 8-17 Assignment of RS 120 (software protection) when writing

High byte	
Bit no.	Assignment
15	Action bit: 1 = execute function
14	Function bit: 1 = set password, 0 = delete PW
13	Bit nos. 8 to 13 of a 14-bit password
12	
11	
10	
9	
8	
Low byte	
7	Bit nos. 0 to 7 of a 14-bit password
6	
5	
4	
3	
2	
1	
0	

[1] Processing a request does not depend on OB 38 or OB 39 being loaded. This means that software protection can be activated in the STOP mode.

Reading system data RS 120

By reading out system data RS 120, you can find out whether a "job" was executed by writing the system data. The system program enters a result code here.

Assignment of the system data when reading

After calling the software protection function, you can evaluate the result code to find out whether the job was successful.

Address: E F078H

Table 8-18 Assignment of RS 120 (software protection) when reading

High byte	
Bit no.	Assignment
15	0
14	Error bit: 0 = no error, 1 = error
13	0
12	0
11	0
10	binary delete error counter
9	
8	
Low byte	
7	0
6	0
5	0
4	1 = no password active
3	1 = deleting not possible, wrong password
2	1 = software protection (password) already activated
1	1 = illegal password
0	1 = error counter overflow

Valid result codes

Value	Explanations
0000H	No error
4x01H	The maximum number of delete attempts has been exceeded. The counter can only be reset with a cold restart.
4x02H	Illegal password (0000H or 3FFFH)
4x04H	You have attempted to specify a new password while the password protection was active (x = number of attempts to delete)
4x08H	You attempted to delete the existing password (deactivate protection) with an incorrect password. The error counter for incorrect attempts was incremented. The counter reading x is transferred in the result code (binary number in bits no. 8 to 10).
4010H	You attempted to delete a non-existent password.

The best time to activate software protection

The most effective protection is achieved when you activate the software protection in OB 38/OB 39 (SOFT STOP mode). Protection is then active immediately following an overall reset even with the memory card inserted.

Reactions to violations of the software protection

Once the software protection is active, the system program reacts to violations of the protection by PG jobs. The following table lists the reactions to various PG jobs.

PG function	Output on PG
Delete block	Message "Block type and number illegal"
Read block	Output of a dummy block: FB/FX: FB number NAME :DUMMY :BE DB/DX: DW0 6500 OB/PB/SB: :BE
Overwrite block (block does not yet exist)	The block is entered
Overwrite block (block already exists)	Message "Block exists"; after confirming with the enter key the message "Block type and number wrong" is displayed.

Examples of writing and reading RS 120

Activating the software protection in the start-up blocks:

(If you activate the protection in the program, it is best to activate it in a start-up OB (OB 20, OB 21, OB 22, OB 38).)

```
        :
        :L   KH C0AF    KH = bit pattern "specify password"
        :               (password = 00AFH)
        :T   RS 120
        :
```

Evaluate result in RS 120:

Using the following sequence of STEP 5 operations in **OB 1** or **OB 39**, you can react to an error occurring when specifying the password by evaluating the result.
Note that the result can only be evaluated after certain actions of the system program (see page 8 - 35).

```
        :
        :L   RS 120
        :L   KB 0
        :><F
        :JC  FB yyy     call function block for error processing
NAME    :    PW-ERROR
        :
```

Delete and modify the password on the PG using the OUTPUT ADDRESS function:

Initial status: The CPU is in the RUN or STOP mode.

Go through the following procedure on the PG:

1. Output the address E F078H.

2. Delete the old password by overwriting the content with 80AFH in hexadecimal ("00AFH" = old password).

3. Wait at least as long as the cycle time of OB 39 or OB 1.

4. Output the address E F078H again.

5. Enter the new password "1234H" by overwriting the content with the hexadecimal number D234H.

System data words RS 136 to RS 137

For self-test function

The system data words RS 136 to RS 137 are used for the self-test.

RS 136

Number of time slices (address: E F088H)

RS 137

Control bits (address: E F089H)

Using the control bits, the individual self-test functions can be included or excluded (refer to Section 5.7).
Bit = ’1’: self-test function is included
Bit = ’0’: self-test function is excluded

Table 8-19 Bits of RS 137 (control bits for self-test functions)

High byte	
Bit no.	Assignment
15	Memory test
14	Not used
13	Test cycle time monitoring
12	Not used
11	Test BASP signal
10	Clock test
9	Not used
8	Not used
Low byte	
7	Test address lines
6	Not used
5	Code test of the STEP 5 logic blocks in the user memory
4	Not used
3	Not used
2	Code test of the system program
1	Not used
0	Not used

System data RS 139

Cycle time used when retriggering

Address E F08AH

This system data word contains the time used for the cycle since the last system checkpoint (at the beginning of OB 1) to the next retriggering with OB 222 (if OB 222 is called more than once within the cycle, the time to the last retriggering).

The time value is the content of RS 139 * 10 ms.

System data RS 253

List of interface modules plugged in

Address: E F0FDH

Table 8-20 Bits of RS 253 (list of interface modules plugged in)

High byte	
Bit no.	Assignment
15	reserved
14	
13	
12	
11	IM number 11
10	IM number 10
9	IM number 9
8	IM number 8
Low byte	
7	reserved (IM number 8)
6	reserved (IM number 7)
5	reserved (IM number 6)
4	reserved (IM number 5)
3	reserved (IM number 4)
2	reserved (IM number 3)
1	reserved (IM number 2)
0	reserved (IM number 1)

8.3.6 Addressable System Data Area

The system program uses the memory area from E 8200H to E DEF0H as an addressable system data area.

PLC identification field

At the start of this area there is an information field of 12 words in which an identifier of the PLC is entered.

This field has the following structure:

Word			
0	’S’	’5’	E 8200H
1	’1’	’5’	E 8201H
2	’5’	’U’	E 8202H
3	’C’	’P’	E 8203H
4	’U’	’9’	E 8204H
5	’4’	’8’	E 8205H
6	’V’	’x’	E 8206H
7	’.’	’y’	E 8207H
8	0		
9	0		
10	0		
11	0		E 820BH

For ’x’ and ’y’ the current version number is entered.

System parameters

System parameters in the memory area beginning with the address E 8210H:

Word			
0	Start add. interface module input		E 8210H
1	Start add. interface module output		E 8211H
2	Start add. process image input table		E 8212H
3	Start add. process image output table		E 8213H
4	Start add. flags		E 8214H
5	Start add. timers		E 8215H
6	Start add. counters		E 8216H
7	Start add. system data		E 8217H
8	Status ID	PLC software version	E 8218H
9	User memory end address		E 8219H
10	System program memory		E 821AH
11	Length of DB list		E 821BH
12	Length of SB list		E 821CH
13	Length of PB list		E 821DH
14	Length of FB list		E 821EH
15	Length of OB list		E 821FH
16	Length of FX list		E 8220H
17	Length of DX list		E 8221H
18	Length of DB address list (DB 0)		E 8222H
19	Slot ID (see below)	CPU ID 2 (see below)	E 8223H
20	Block header length		E 8224H
21	CPU ID 1 (see below)	Programmer interface software version	E 8225H

You can also use the SYSTEM PARAMETER PG online function to find the information contained in a few system data registers (e.g., concerning the internal structure of the CPU, the software version, the CPU ID).

Word 19 and word 21

Structure of words 19 and 21:

Word 19

Bit no.	High byte
15	0
14	0
13	0
12	0
11	Slot ID CPU 4
10	Slot ID CPU 3
9	Slot ID CPU 2
8	Slot ID CPU 1
	Low byte
7	CPU type: 0010 = CPU 948 (only valid in conjunction with the CPU ID)
6	
5	
4	
3	CPU-ID 2: 1000 = S5-155U
2	
1	
0	

Word 21

Bit no.	High byte
15	reserved
14	
13	
12	
11	
10	
9	
8	
Bit no.	**Low byte**
7	Release of the PG interface software in the form "xyH" Example: 13H corresponds to release "V1.3"
6	
5	
4	
3	
2	
1	
0	

Memory Access Using Absolute Addresses

9

Contents of Chapter 9

9 Memory Access Using Absolute Addresses

This chapter explains how to use STEP 5 operations and special STEP 5 registers to address data in certain memory areas using absolute addresses.

9.1 Introduction

The STEP 5 programming language contains operations with which you can access the entire memory area. These operations belong to the "system operations".

The operations described in this chapter work with 20-bit absolute addresses. Consequently, they are dependent on the memory size and type, the peripherals, CPs, and IPs of your programmable controller.

⚠

Warning
If the operations described in this chapter are not used properly, STEP 5 blocks and system data can be overwritten. Therefore, only experienced programmers should use operations that work with absolute addresses.

Local memory

Local memory is the memory area that is available in each CPU. It includes the following: user submodule, RI/RJ area, RS/RT area, counters, timers, flags, process images.

Global memory

Global memory exists only once for all CPUs. You address it via the S5 bus.

Memory organization

Memory areas are organized **in bytes** or **in words** as follows:

- Bytes: Each address addresses a byte.
- Words: Each address addresses a 16-bit word (= 2 bytes).

Organization of the **local** memory is fixed (see Chapter 8)

Organization of the **global** memory depends on the type of modules that are plugged into the programmable controller:

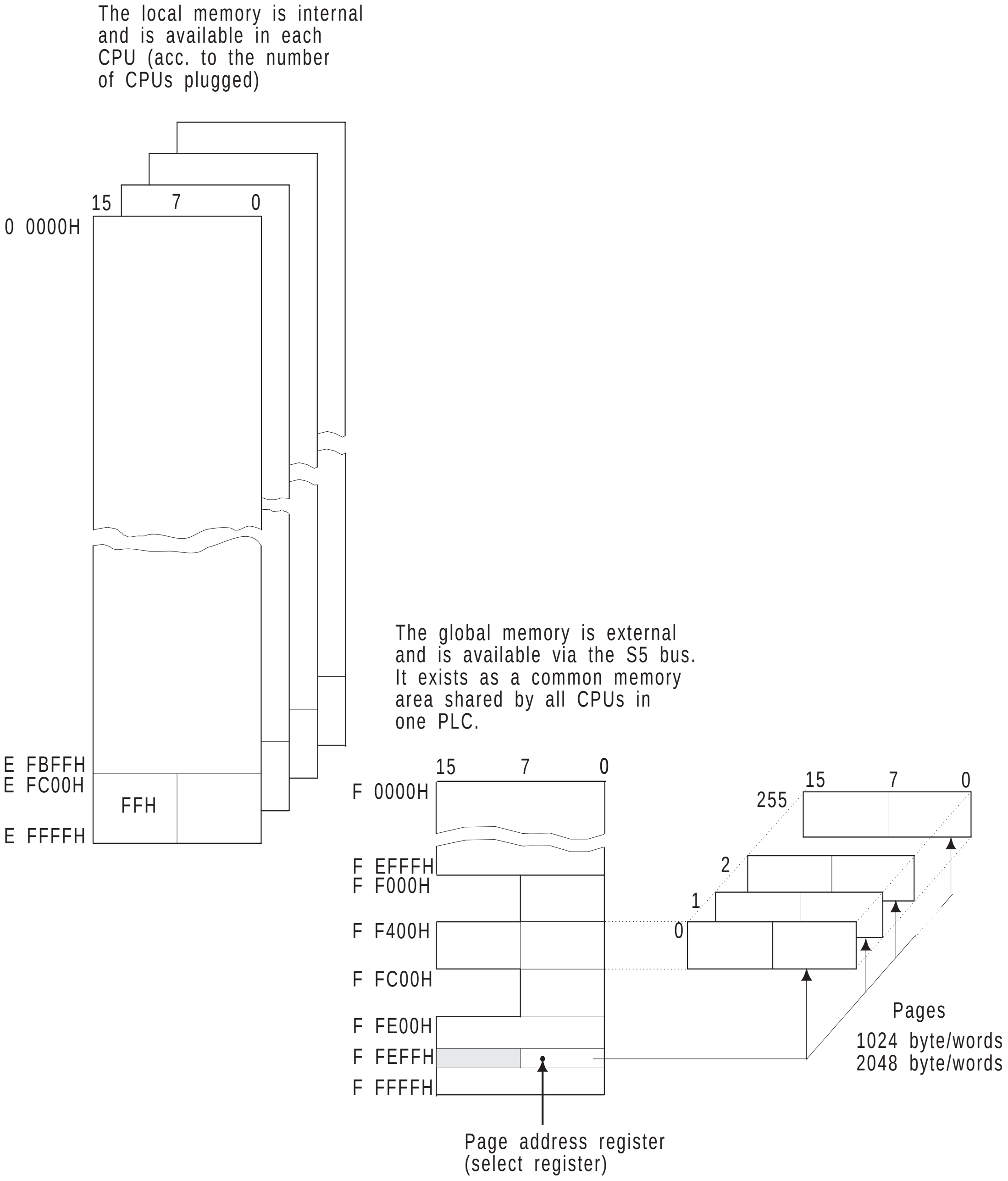

The local memory is internal and is available in each CPU (acc. to the number of CPUs plugged)
15
7
0
0 0000H
E FBFFH
E FC00H
FFH
E FFFFH
The global memory is external and is available via the S5 bus. It exists as a common memory area shared by all CPUs in one PLC.
15
7
0
F 0000H
F EFFFH
F F000H
F F400H
F FC00H
F FE00H
F FEFFH
F FFFFH
Page address register (select register)
255
15
7
0
2
1
0
Pages
1024 byte/words
2048 byte/words

Fig. 9-1 Global and local memory

Memory access

Using absolute addresses, you can access the following local or global memory areas with the operations indicated (refer to Fig. 9-2).

Access to the local and global area

You can access both the local and global areas:

- Local area (addresses 0 0000H to E FFFFH) and global area (addresses F 0000H to F FFFFH) with:

  LIR, TIR, LDI, TDI, TNW, TXB, TXW,

- Section of the local area organized in words (addresses 0 0000H to E FBFFH) or in bytes (addresses E FC000 to E FFFF) with:

  LRW, TRW, LRD, TRD.

Access only to the global area

You can access the following parts of the global area:

- Section of the global area organized in bytes (addresses F 0000H to F FFFFH) with:

  LY GB, LY GW, LY GD, TY GB, TY GW, TY GD, TSG,

- Section of the global area organized in words (addresses F 0000H to F FFFFH) with:

  LW GW, LW GD, TW GW, TW GD, TSG .

Access to the page area

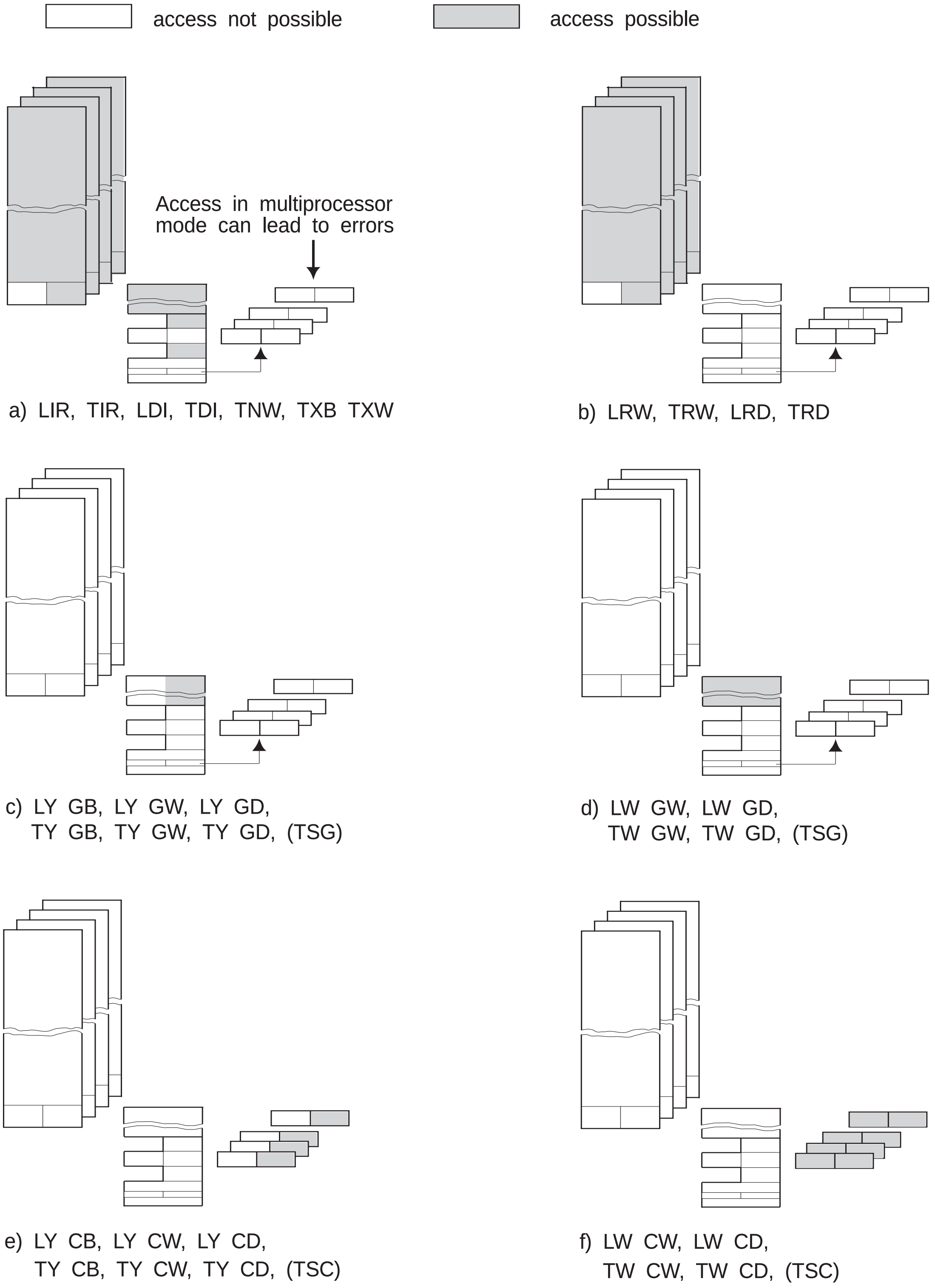
You can access the following parts of the page area:

- Section of the global area organized in bytes (addresses F F400H to F FBFFH, = dual-port RAM area) with:

  LY CB, LY CW, LY CD, TY CB, TY CW, TY CD, TSC,

- Section of the global area organized in words (addresses F F400H to F FBFFH, = dual-port RAM area) with:

  LW CW, LW CD, TW CW, TW CD, TSC



a) LIR, TIR, LDI, TDI, TNW, TXB TXW

b) LRW, TRW, LRD, TRD

c) LY GB, LY GW, LY GD,
TY GB, TY GW, TY GD, (TSG)

d) LW GW, LW GD,
TW GW, TW GD, (TSG)

e) LY CB, LY CW, LY CD,
TY CB, TY CW, TY CD, (TSC)

f) LW CW, LW CD,
TW CW, TW CD, (TSC)

Fig. 9-2 Access to local or global areas using absolute addresses

9.2 Memory Access via Address in ACCU 1

Application

The operations listed in this section are suitable primarily for access to data blocks and other operand areas. You should, however, **not** access blocks containing STEP 5 programs (OBs, FBs, PBs and SBs) with these operations.

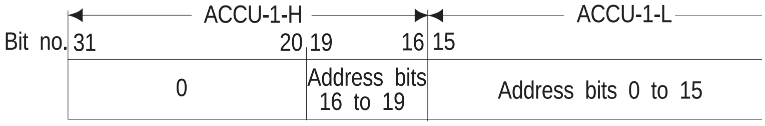
To access these areas, you can use various 16 or 32-bit wide registers. These registers include the accumulators ACCU 1 to ACCU 4 and other special registers used as resources by the CPU.

Operations

Table 9-1 Operations for indirect memory access using registers

Operation	Operand	Function
LIR	Register no. 0 to 15	Load the **16-bit register** with the content of a memory word addressed by ACCU 1 (20-bit address).
TIR	Register no. 0 to 15	Load the content of the **16-bit register** in the memory word addressed by ACCU 1 (20-bit address).
LDI	Register name	Load the **32-bit register** with the contents of the memory words ’n’ and ’n+1’ addressed by ACCU 1 (20-bit address).
TDI	Register name	Load the contents of the **32-bit register** in the memory words ’n’ and ’n+1’ addressed by ACCU 1 (20-bit address).

The absolute address of the memory word or the first of the two memory words is located in ACCU 1 in the following representation:



The following pages explain **which registers** you can use with the operations.

Examples explain **how** to use the operations.

9.2.1 LIR/TIR: Loading to or Transferring from a 16-Bit Memory Area Indirectly

The following table shows which register numbers you can use with the CPU 948 for the LIR and TIR operations and how these are assigned.

Table 9-2 16-bit register for LIR/TIR

Register no.	Register assignment (each 16 bits wide)
0	ACCU-1-H (left word of ACCU1, bits 16 to 31)[1]
1	ACCU-1-L (right word of ACCU1, bits 0 to 15)[1]
2	ACCU-2-H
3	ACCU-2-L
5	Block stack pointer (offset)
6	DBA (data block start address register)
8	DBL (data block length register)
9	ACCU-3-H
10	ACCU-3-L
11	ACCU-4-H
12	ACCU-4-L

[1] Loading the contents of an addressed memory register into register ’0’or ’1 overwrites the address stored in ACCU 1.

Registers 4, 7, 13, 14 and 15 do not exist on the CPU 948. LIR/TIR operations with these register numbers must not be used.

LIR/TIR: with 8-bit memory areas

If you use the LIR and TIR operations to access memory areas that are only 8 bits wide, remember that

- the LIR operation overwrites the high byte of the registers with **non-defined values** (except for flags, PIQ, PII; with these areas, FFH is written in the high byte)

  and

- the TIR operation transfers only the low byte of the register. The high byte of the register is lost.

Figs. 9-3 and 9-4 illustrate the difference when accessing word and byte-oriented memory areas using LIR/TIR.

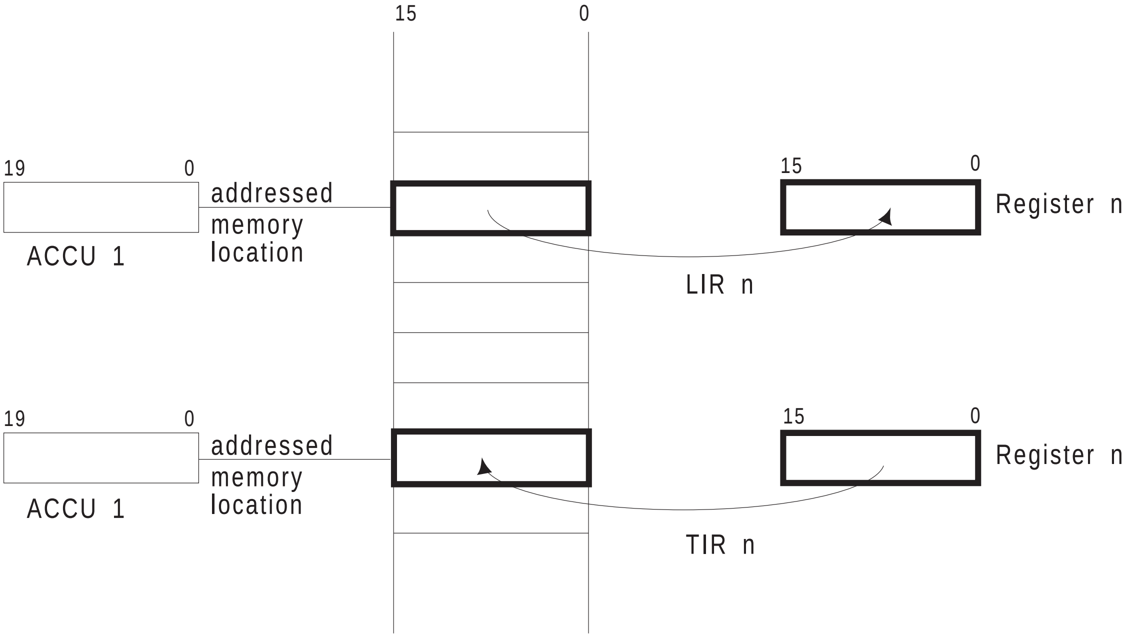


Fig. 9-3 LIR/TIR with 16-bit memory areas (word-oriented)

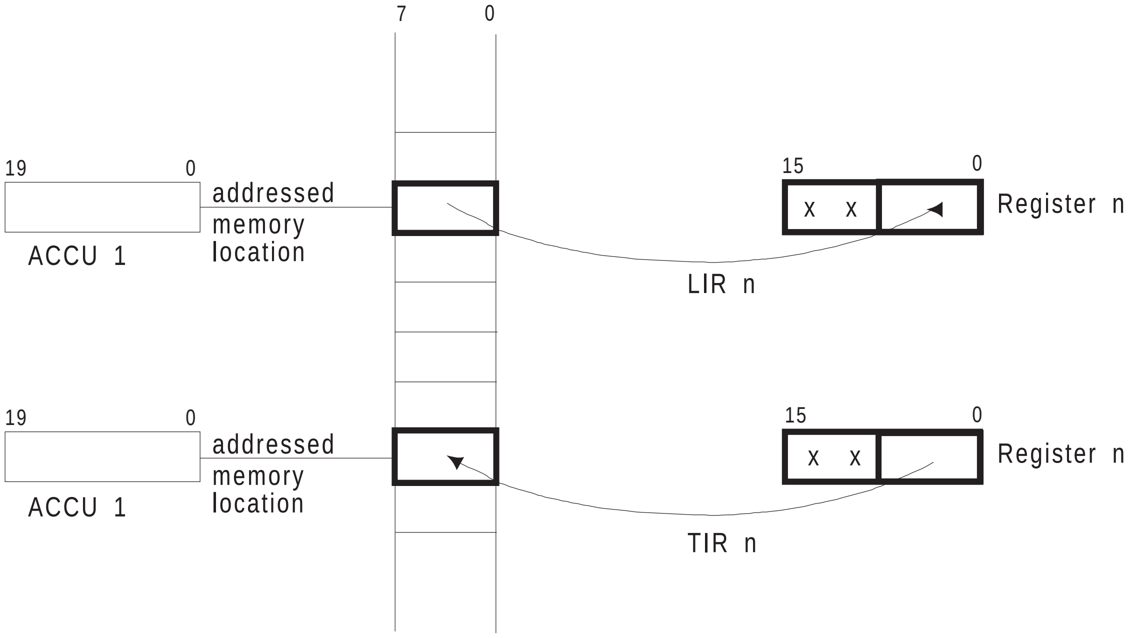


Fig. 9-4 LIR/TIR with 8-bit memory areas (byte-oriented)

Registers 0 to 3 and 9 to 12: ACCUs 1, 2, 3 and 4

During program processing, the accumulators are used as buffers for the CPU. The TIR operation transfers the contents of the accumulators into absolutely addressed memory registers. The LIR operation loads the contents of absolutely addressed memory registers into the accumulators. The absolute address of the memory locations is in ACCU 1, bit number 0 to 19.

Examples

```
The contents of the memory location with address E F800 are loaded in flag
word FW 100.

   :L   DH 000E F800 Load address E F800 of the memory location in ACCU 1
   :LIR 1            Load the contents of the memory location addressed by
   :                 ACCU 1 in register 1 (= ACCU-1-L)
   :T   FW 100       Store the contents of address E F800 in flag word FW 100
   :BE
```

```
The content of the flag word 200 is transferred to the memory location with
address E F800.

   :L   FW 200       Load flag word FW 200 in ACCU 1
   :L   DH 000E F800 Load address E F800 to which the data will be trans-
   :                 ferred in ACCU 1 (flag word FW 200 to ACCU 2)
   :TIR 3            Transfer the contents of register 3 = ACCU-2-L in to
   :                 the memory location addressed by ACCU 1
   :BE
```

Register 6: DBA (Data Block Start Address)

When you open a data block or an extended data block using the C DB or CX DX operations, the address of DW 0 in the opened data block is loaded into register 6. The block address list in DB 0 contains this address.
The DBA register is set to ’0’ before each OB 1 call.

The DBA register **remains the same** if one of the following occurs:

- a jump operation (JU/JC) causes program processing to continue in a different block,

  or

- the CPU activates a different program processing level.

The DBA register **changes** if one of the following occurs:

- Another data block is opened,

  or

- the program returns to a higher order block after a new data block is opened in the called block (refer to Section 2.4.3).

Example

Effect of the **"CX DX 17"** operation on the DBA register:

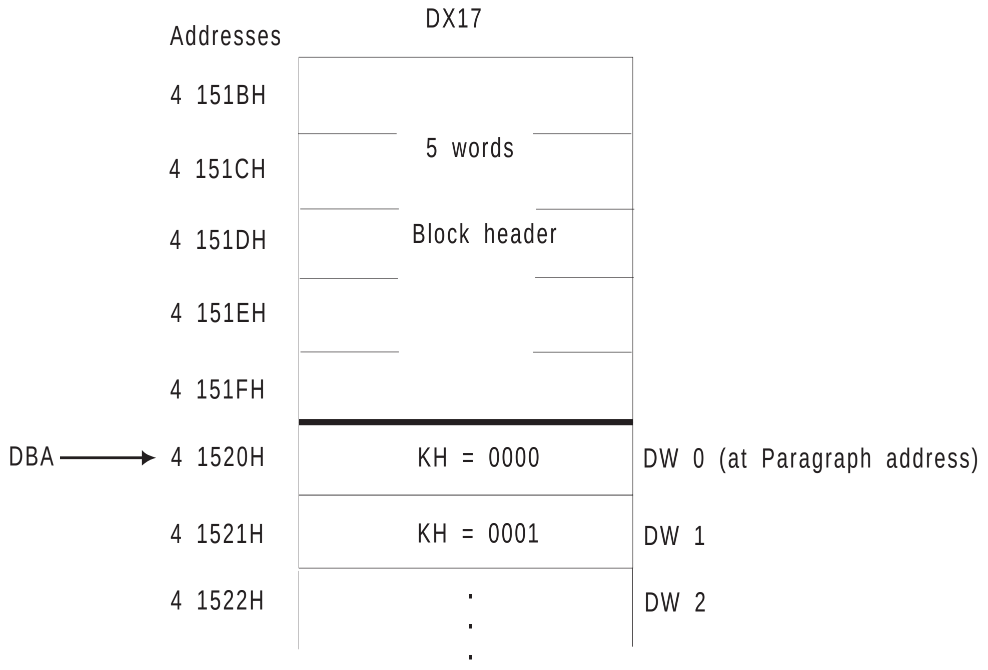


Fig. 9-5 Using the DBA register

When DX 17 is called, the address of the memory word in which DW 0 is stored is entered in the DBA register. In this example, the DBA is **4152H.**

Note: In the ISTACK, the address entered in the DBA register appears under the heading ’DB-ADD’.

Register 8:
DBL = Data Block Length

In addition to the DBA register, a DBL register is loaded every time a data block is called. It contains the length (in words) of the data block called, **without** the block header. The DBL register is set to ’0’ before each OB 1 call.

DBL register is **retained**, when

- program execution is continued in a different block following a jump statement (JU/JC)

  or

- a different program execution level is nested in.

It **changes** when

- a different data block is opened

  or

- program execution returns to a higher order block after a new data block is opened in the called block (refer to Section 2.4.2).

> **Note**
> You can change the DBA and DBL registers using LIR operations to address data block addresses higher than 255. The DBA register contains **paragraph addresses**. Make sure that a change in the DBA register does not automatically cause a change in the DBL register and vice-versa. **This would mean that transfer error monitoring is no longer guaranteed.**
> On the CPU 948, changes in the DBA/DBL registers are **undone**, as soon as the **current block is completed** (BSTACK entries). Manipulations on the DBA/DBL registers are therefore only effective in the block in which they were made.

Example

Effect of the **"CX DX 17"** operation on the DBL:

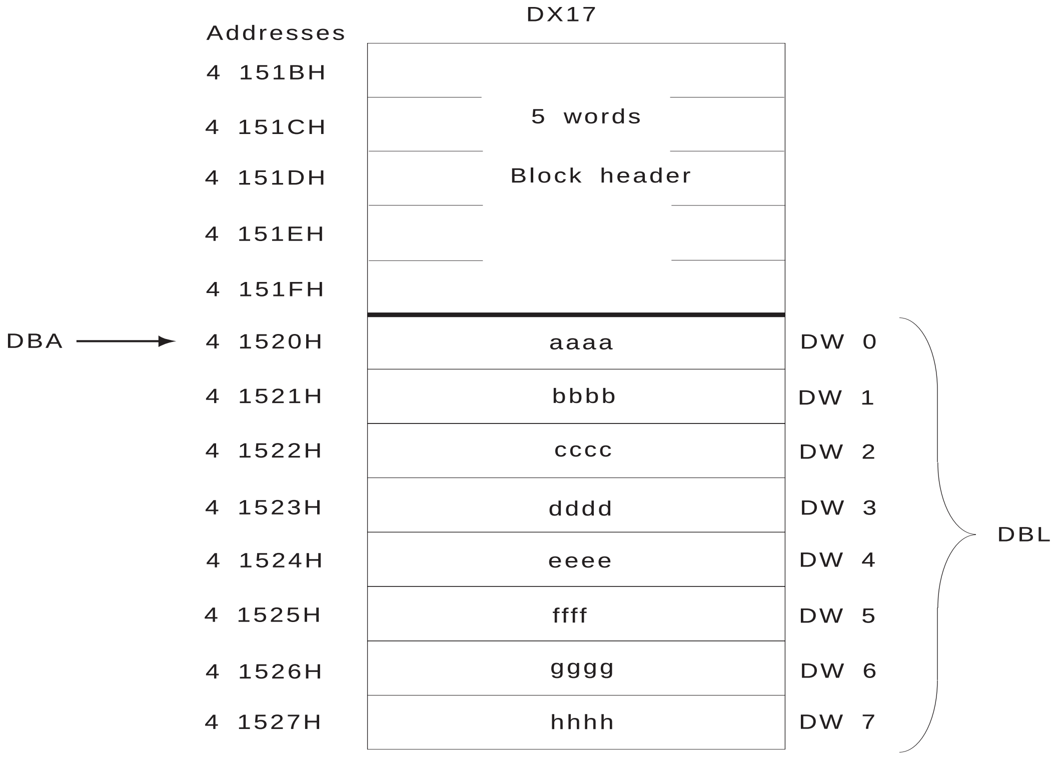


Fig. 9-6 Using the DBL register

When DX 17 is called, the number of existing data words is entered in the DBL register. In this example the DBL is **8** (DW 0 to DW 7)

Note: In the ISTACK, the number entered in the DBL register appears under the heading "DBL-REG".

9.2.2 Examples of Access to DW > 255

Example 1:

The content of data word DW 300 in DB 100 is read and transferred to flag word FW 100 (by changing the STEP 5 operation shown in bold face, it can also be used to read other data blocks (DB or DX)).

```
FB  5

SEGMENT 1          0000         Reading out DW 300 from DB 100
NAME :LIR DW

0005      :L   DH   000E EC00     Start addr. of the DB list
0008      :L   KF   +100           plus the DB number
000A      :+D                      = Address list entry of DB 100
000B      :                        (Bits 4 to 19)
000C      :LIR      1             Start addr. DB 100 to ACCU 1
000D      :SLD      4             Convert addr. to physical addr.
000E      :L   KF   +300          DW 300 is read out
0010      :+D                     Addr. = start addr. of DB + DW addr.
0011      :LIR      1             Store content of DW 300 to ACCU 1
0012      :T   FW 100              in FW 100
0013      :BE
```

Example 2:

All the data words of a data block will have a constant written to them.

The program shown below writes the constant KH = A5A5 to all data words of DB 100. After changing the STEP 5 operations shown in bold face, it can also be used to write values to other data blocks (DB or DX). Non-existent data blocks are detected and cause a jump to the NIVO marker.

The program uses three accumulators. Within the loop, the accumulator contents do not change.
ACCU 1 initially contains the address of the first data word and is incremented by one each time the loop is run through.
ACCU 2 contains the address of the last data word + 1. The loop is terminated as soon as the content of ACCU 1 is the same as the content of ACCU 2.

To write the data words, the operation TIR 10 is used which stores the content of ACCU-3-L (the constant) at the address contained in ACCU 1.

<u>ACCU contents within the loop:</u>

ACCU 1: address of the **current** data word to be written to
ACCU 2: address of the **last** data word to be written to **+ 1**
ACCU 3: constant

Continued on the next page

Example 2 continued:

Flag assignment:

```
FW 10:    Bits 4 to 19 of the start address of the DB/DX (points to DW 0)
FW 12:    Length of the DB/DX (number of data words)
FD 14:    Address of the last data word in the DB/DX + 1 (physical address)
```

```
FB  6

SEGMENT 1           0000          Writing a DB with a constant
NAME :FILL DB
0005        :                     ! Required flags: FY 10 to FY 17      !
0006        :L   DH   000E EC00 Start addr. of the DB list
0009        :L   KF   +100       plus the DB number
000B        :+D                    = Address list entry of DB 100
000C        :                      (Bits 4 to 19)
000D        :LIR       1          Start addr. DB 100 to ACCU 1
000E        :T    FW   10         Buffer start addr.
000F        :L    KB    0          (Paragraph address)
0010        :!=F                  If start addr. = 0, then
0011        :JC   =NIVO            DB does not exist
0012        :
0013        :L    FW   10         Start addr. of DB (1st DW)
0014        :SLD        4         Convert addr. to physical address
0015        :L    KB    1         Find out DB length via
0016        :-D                    5th word in block header
0017        :LIR        1         Length including block header to ACCU 1
0018        :ADD  BN   -5         Number of DWs = total length - 5 words (header)
0019        :
001A        :T    FW   12         Buffer length
001B        :
001C        :L    FW   12         Number of data words +
001D        :L    FW   10          start address (DW 0 converted to
001E        :SLD        4          physical address)
001F        :+D                    produces
0020        :T    FD   14          address of the last DW + 1
0021        :
0022        :L   KH   A5A5      Constant, written to all data words
0024        :
0025        :L    FD   14         Address of the last DW + 1
0026        :ENT                  Shift constant to ACCU-3-L
0027        :                      (= register 10)
0028        :L    FW   10         Convert address of 1st data word (DW 0)
0029        :SLD        4          to physical address
002A        :
002B SCHL   :                     Loop:
002C        :                      ACCU 1: address of DW to be written to
002D        :                      ACCU 2: address of last DW + 1
002E        :                      ACCU 3: constant
002F        :TIR       10         Store the value of ACCU-3-L in the DW with
0030        :                      the address in ACCU 1
0031        :
0032        :ADD  DH    0000 0001 Increment address by 1
0035        :
0036        :><D                  Scan whether last DW reached
0037        :JC   =SCHL            (if not, return to the loop)
0038        :
0039 WEIT   :                     Continue the program ...
003A        :                      after all DWs have been written to ...
003B        :
003C        :BEU
003D        :
003E NIVO   :                     If DB 100 does not exist
003F        :BE
```

9.2.3 LDI/TDI: Loading to or Transferring from a 32-Bit Memory Area Indirectly

The following table shows which register names you can use on the CPU 948 for the LDI and TDI operations and how these are assigned.

Table 9-3 32-bit register for LDI/TDI

Register name	Register assignment (each 32 bits wide)
A1	ACCU 1 (ACCU1, bits 0 to 31) [1)]
A2	ACCU 2 (ACCU1, bits 0 to 31)
SA	STEP address counter (bits 0 to 19)
BA	BA register (block start address, bits 0 to 19)
BR	BR register (base address register, bits 0 to 19)

[1)] Loading the contents of an addressed memory register into the A1 register overwrites the address stored in ACCU 1.

Byte addresses

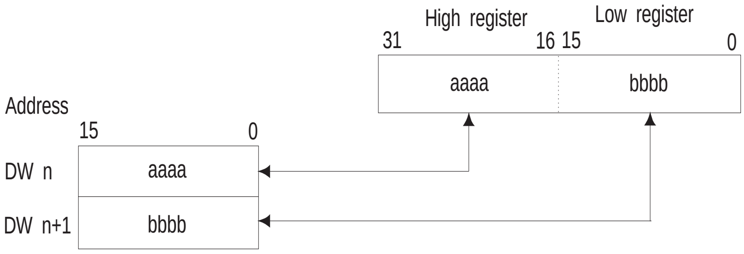
If you reference byte addresses with the LDI or TDI operations, note the following:

- the LDI operation overwrites the high byte of the register with **non-defined values** (except for flags, PIQ, PII; with these areas, FFH is written in the high byte)

  and

- the TDI operation transfers only the low bytes of the register (the high bytes are lost – refer to the example on the following page

Data storage with LDI/TDI



SA Register:
SAC = STEP Address Counter

On completion of the operation, the 20-bit absolute address **of the operation** to be processed next is entered in the SA register.

BA Register:
Block Start Address

During program processing of the STEP 5 user program, a 20-bit absolute address is entered in the BA register. This address is in the higher order block (corresponds to the return address). It is the address of the operation to be processed next.

BR Register:
Available Base Address Register

The base address register (20 bits) allows you to calculate addresses and to execute indirect load and transfer operations without using the ACCUs for the address. It can be used freely during STEP 5 program processing.

Example of TDI in the byte area

```
   :L   DH 1234 5678    Load data
   :L   DH 000E FC00    Load address of flag
   :                    byte FY 0
   :TDI A2              Store content of ACCU 2


E FC00 = 34  (The values ’12H’ and ’56H’ from
E FC01 = 78  ACCU 2 are lost)
```

9.3 Transferring Memory Blocks

Application

With the operations explained in this section, you can re-store data areas with a length of up to 255 words located in certain address areas.

Operations

Table 9-4 Operations for field transfer

Operation	Operand	Function
TNW	0 to 255	Field transfer 0 to 255 words [1)] (in the 16-bit memory area)
TXB	--	Field transfer from 8-bit to the 16-bit memory
TXW	--	Field transfer from the 16-bit to the 8-bit memory

[1)] Can also be used for transfer from the byte to the byte area.

Parameters

Field length

For TNW: Operand = number of words (0 to 255)

For TXB/TXW ACCU 3 = number of words (0 to 127)

End address of the source area

ACCU 2 = end address of the source area (20 bits)

End address of the destination area

ACCU 1 = end address of the destination area (20 bits)

The **entire** source and destination areas must be located in one of the memory areas listed in Table 9-5 **and cannot overlap**.

Permitted memory areas

Table 9-5 Memory areas permitted for TNW, TXB and TXW

Addresses	Memory area
0 0000H to C FFFFH	User memory: 16-bit area (dependent on memory configuration)

Addresses	Memory area
Table 9-5 continued:	
	System RAM:
E 8000H to E 9FFFH	System data, 16 bits
E B000H to E FBFFH	System data (RI/RS, timers, counters etc.), 16 bits
E A000H to E AFFFH	S flags, 8 bits, low byte in 16-bit word (High byte not defined)
E FC00H to E FFFFH	Flags, process image, 8 bits (High byte = FFH)
F 0000H to F FFFFH	I/Os, 8/16 bits

(Refer also to Chapter 8)

Sequence

The field transfer is made in descending order, i.e. it begins with the highest address of the source area (= end address) and ends with the lowest.

TNW, TXB and TXW operations

The operations TNW, TXB and TXW are long-running STEP 5 operations which can only be interrupted by POWER DOWN and QVZ.

Special features

Interruptions by POWER DOWN

If one of the operations is interrupted by a power failure (NAU) followed by a warm restart, the operation does not resume at the point at which it was interrupted but from the **beginning** again.

Interruptions by QVZ

If a timeout (QVZ) occurs during the transfer, the operation is interrupted and the appropriate error OB called.
The error address is the address at which an error occurred (refer to Section 5.6.3).

ADF during execution

If an addressing error (ADF) occurs once or more than once during the transfer, all the part fields are first transferred and then OB 25 is called before the next operation is executed.

Example

TXB and TXW between 8 and 16-bit memory areas:

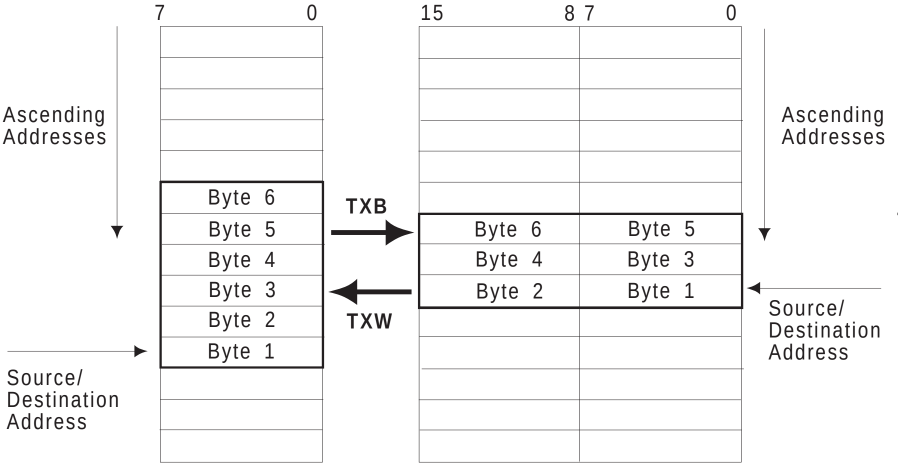


Fig. 9-7 Transferring memory fields

Transferring bytes 1 to 6 from an 8-bit to a 16-bit area:

```
   :L   <field length in words> e.g.      :L   KH 0003
   :L   <source address>                  :L   DH EFC10
   :ENT                                   :ENT
   :L   <destination address>             :L   DH EF208
   :TXB                                   :TXB
```

Transferring bytes 1 to 6 from a 16-bit to an 8-bit area:

```
   :L   <field length in words> e.g.      :L   KH 0003
   :L   <source address>                  :L   DH EF008
   :ENT                                   :ENT
   :L   <destination address>             :L   DH EFC10
   :TXW                                   :TXW
```

9.4 Operations with the Base Address Register (BR Register)

Application

The base address register (20 bits) allows you to calculate addresses and use indirect register load and transfer operations without using the ACCUs.
The memory location whose absolute address is calculated as the sum of the BR register content plus a constant is accessed.

Absolute address = BR register content + constant

Operations

Table 9-6 Load and arithmetic operations with the BR register

Operation	Operand	Function
MBR	Constant (0H to F FFFFH)	Load the BR register with a 20-bit constant
ABR	Constant (-32 768 to +32 767)	Add a 16-bit constant to the contents of the BR register

Changing the BR register

- The BR register is **retained**, when the program is continued in a **different block of the same program execution level** due to a jump statement (’JU FB’/’JC FB’).
- The BR register is **retained** after **nesting in** a different program execution level.
- If a different program execution level is **called** by the system program, the BR register is set to **’0’**.

9.4.1 Operations for Transfer between Registers

Application

You can use the operations described in this section for the fast exchange of values between the registers ACCU 1 (32 bits), step address counter (SAC - 20 bits) and BR register (20 bits).

Operations

Table 9-7 Register-register operations

Operation	Operand	Function
MAS	--	Transfer the contents of ACCU 1 to the STEP address counter (20 bits)
MAB	--	Transfer the contents of ACCU 1 to the base address register (20 bits)
MSA	--	Transfer the contents of the STEP address counter (20 bits) to ACCU 1 [1)]
MSB	--	Transfer the contents of the STEP address counter (20 bits) to the base address register (20 bits)
MBA	--	Transfer the contents of the base address register (20 bits) to ACCU 1 [1)]
MBS	---	Transfer the contents of the base address register (20 bits) to the STEP address counter (20 bits)

[1)] Bits 2^{20} to 2^{31} are set to ’0’.

The following figure illustrates how the registers are changed by the operations.

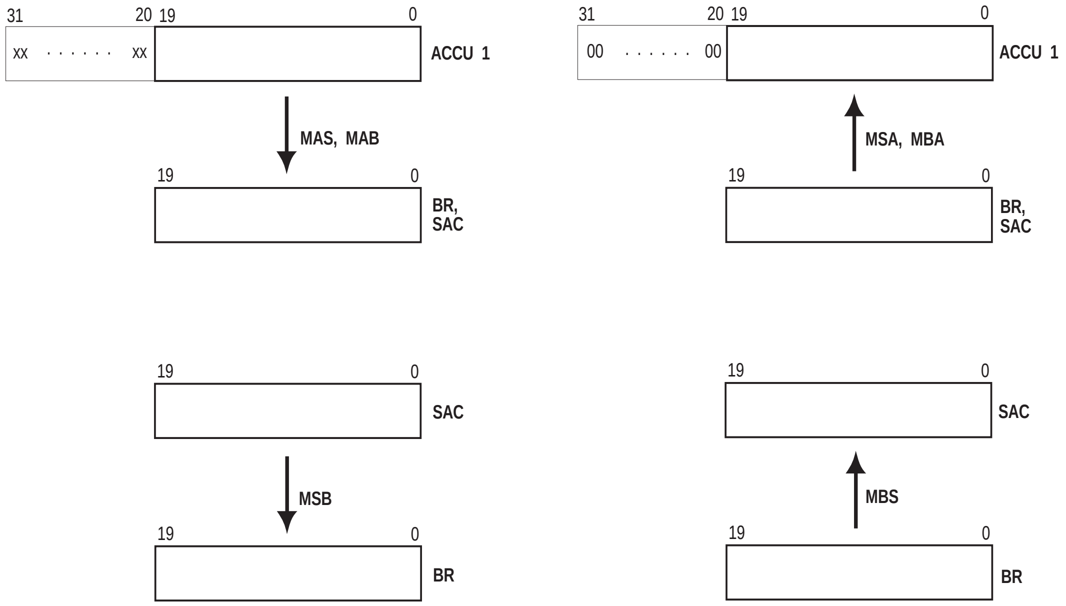


Fig. 9-8 Transfer operations from one register to another

9.4.2 Accessing the Local Memory

Application

The following operations allow access to the local memory organized in words using an absolute memory address. The absolute address is the sum of the contents of the BR register and the 16-bit constant contained in the operation (-32768 to +32767).

Operations

Table 9-8 Operations for accessing the local memory

Operation	Operand	Description
LRW	Constant (-32768 to +32767)	add the specified constant to content of the BR register and load the word addressed in this way in ACCU-1-L [1)]
LRD	Constant (-32768 to +32767)	add the specified constant to content of the BR register and load the double word addressed in this way in ACCU 1 [1)]
TRW	Constant (-32768 to +32767)	add the specified constant to content of the BR register and transfer the content of ACCU-1-L to the word addressed in this way

Operation	Operand	Description
Table 9-8 continued:		
TRD	Constant (-32768 to +32767)	add the specified constant to content of the BR register and transfer the content of ACCU 1 to the double word addressed in this way

[1] ACCU 2_{new} = ACCU 1_{old}

Error reaction

If the calculated address of the memory location is not between 0 0000H and E FFFFH, the CPU detects a load/transfer error (TRAF) and calls **OB 32**. If OB 32 is not loaded, the CPU changes to the stop mode with the error code TRAF (ISTACK).

9.4.3 Accessing the Global Memory

Application

This section describes the operations you can use with absolute memory addresses to access the global memory organized in bytes or words. The absolute address is the sum of the contents of the BR register and the constants contained in the operation (-32768 to 32767).

Testing and setting an "occupied" register in the global area

You can control access of individual CPUs to commonly used memory areas by using an "occupied" register. An "occupied" register is assigned to each commonly used memory area. Each participating CPU must test this register before accessing the memory area. The "occupied" register contains either the value ’0’ or the slot ID of the CPU that is presently using the memory area. When the CPU is finished using the memory area, it **writes ’0’ to the "occupied" register to re-enable the memory area**. (Note the explanations for the operations "set semaphore/SED" and "enable semaphore/SEE" in Section 3.5.5.)

The TSG operation enables testing and setting of "occupied" registers.

Operation	Operand	Description
TSG	-32768 to +32767	Add the specified constant to the content of the BR register and test and set the location addressed in this way.

Sequence

The location used is the low byte of the word addressed by the BR register plus the constant. If the content of the low byte is ’0’, the TSG operation enters the slot ID in the location.

Testing (reading) and possible occupation of the location (writing) form a program unit that cannot be interrupted.

Result

You can evaluate the result of the test using condition codes CC 0 and CC 1:

CC 1	CC 0	Description
0	0	The "occupied" register contains ’0’. The CPU enters its own slot ID.
1	0	The slot ID of the CPU is already entered in the "occupied" register.
0	1	The "occupied" register contains a different slot ID.

Note
All CPUs that require synchronized access to a **common global memory area** must use the TSG operation.

Error reaction

The absolute address must be between F 0000H and F FFFFH. If the absolute addresses are not in the range shown, the CPU detects a transfer error (TRAF) and calls **OB 32**. If OB 32 is not loaded, the CPU changes to the STOP mode with the error code TRAF (ISTACK).

Load and transfer operations on the global memory organized in bytes

Table 9-9 Operations for access to the global memory organized in bytes

Operation	Operand	Description
LY GB	-32768 to +32767	add the specified constant to content of the BR register and load the byte addressed in this way in ACCU-1-LL [1) 3)]
LY GW	-32768 to +32767	add the specified constant to content of the BR register and load the word addressed in this way in ACCU-1-L [2) 3)]
LY GD	-32768 to +32767	add the specified constant to content of the BR register and load the double word addressed in this way in ACCU 1[3)]
TY GB	-32768 to +32767	add the specified constant to content of the BR register and transfer the content of ACCU-1-LL to the byte addressed in this way
TY GW	-32768 to +32767	add the specified constant to content of the BR register and transfer the content of ACCU-1-L to the word addressed in this way
TY GD	-32768 to +32767	add the specified constant to content of the BR register and transfer the content of ACCU 1 to the double word addressed in this way

[1)] ACCU-1-LH and ACCU-1-H are set to ’0’.

[2)] ACCU-1-H is set to ’0’.

[3)] ACCU 2_{new} : = ACCU 1_{old}

Error reaction

The range of absolute addresses for the load and transfer operations for the global memory organized in bytes

- between F 0000H and F FFFFH (LY GB, TY GB),
- between F 0000H and F FFFEH (LY GW, TY GW)

  or
- between F 0000H and F FFFCH (LY GD, TY GD).

If the absolute addresses are not in the range shown, the CPU detects a load/transfer error (TRAF) and calls **OB 32**. If OB 32 is not loaded, the CPU changes to the STOP mode with the error code TRAF (ISTACK).

Load and transfer operations for the global memory organized in words

Table 9-10 Operations for access to the global memory organized in words

Operation	Operand	Description
LW GW	-32768 to +32767	add the specified constant to content of the BR register and load the word addressed in this way in ACCU-1-L [1] [2]
LW GD	-32768 to +32767	add the specified constant to content of the BR register and load the double word addressed in this way in ACCU 1 [2]
TW GW	-32768 to +32767	add the specified constant to content of the BR register and transfer the content of ACCU-1-L to the word addressed in this way
TW GD	-32768 to +32767	add the specified constant to content of the BR register transfer the content of ACCU 1 to the double word addressed in this way

[1] ACCU-1-H is set to ’0’.

[2] ACCU 2 $_{new}$: = ACCU 1$_{old}$

Error reaction

The range of absolute addresses must be located

- between F 0000H and F FFFFH (LW GW, TW GW)

  or

- between F 0000H and F FFFEH (LW GD, TW GD).

If the absolute addresses are not in the range shown, the CPU detects a load/transfer error (TRAF) and calls **OB 32**. ACCU 1 contains the error ID 1A01H. If OB 32 is not loaded, the CPU changes to the STOP mode with the error code TRAF (ISTACK).

9.4.4 Accessing the Dual-Port RAM Memory

Application

With the following operations, you can access pages organized as **bytes or words** using an absolute memory address. The absolute address is the sum of the BR register content and the constant contained in the operation (-32768 to 32767).

Sequence of page access

Between the addresses F F400H to F FBFFH, the global memory area has a window for accessing one of a maximum of 256 memory areas called dual-port RAM pages. One dual-port RAM page can occupy a maximum of 2K addresses and can be organized in either bytes or words. Before access to the dual-port RAM area, one of the 256 pages must be selected by entering its number in the **select register** (page address register). The procedure of writing to the select register and then accessing the dual-port RAM area cannot be interrupted.

Before you can access the dual-port RAM area (load/transfer), you must select one of the 256 dual-port RAM pages. Do this by putting the number of the dual-port RAM page that you want to open into ACCU-1-L. Use the ACR operation to enter this number into the CPU-internal **dual-port RAM register.** All dual-port RAM operations that follow ACR write the contents of the dual-port RAM register into the select register of the corresponding modules on the S5 bus before dual-port RAM access.

Changing the page register

- The page register is **retained** when another block is called.
- If the page register is changed in a block, its value is **retained** when the program returns to the calling block at the end of the block.
- The page register is **retained** after nesting in another program execution level.

Opening a dual-port RAM page

Operation	Parameter	Description
ACR		Open the dual-port RAM page whose number is located in ACCU-1-L , permissible values: 0 to 255

The dual-port RAM page number must be between 0 and 255. If it is not, the CPU detects a substitution error (SUF) and calls **OB 27**. If OB 27 is not loaded, the CPU changes to the STOP mode.

Testing and setting an "occupied" register in the dual-port RAM area

You can control access of individual CPUs to commonly used memory areas by using an "occupied" register. An "occupied" register is assigned to each commonly used memory area. Each participating CPU must test this register before accessing the memory area. The "occupied" register contains either the value ’0’ or the slot ID of the CPU that is presently using the memory area. When the CPU is finished using the memory area, it **writes ’0’ to the "occupied" register to re-enable the memory area**. (Note the explanations of the operations "set semaphore/SED" and "enable semaphore/SEE" in Section 3.5.5.)

The TSC operation handles the testing and setting of a location on the open page.

Operation	Operand	Description
TSC	-32 768 to +32 767	Add the specified constant to the content of the BR register and test and set the location addressed in this way on the open page.

Sequence

The low byte of the word addressed by the sum of the BR register + constant is used as the "occupied" register. If the low byte contains ’0’, the TSC operation enters the slot ID of the CPU in the "occupied" register.

Testing (reading) and possible occupation (writing) form a program unit that cannot be interrupted.

Result

You can evaluate the result of the TSC operation using condition codes CC 0 and CC 1:

CC 1	CC 0	Description
0	0	The "occupied" register contains ’0’. The CPU enters its own slot ID.
1	0	The slot ID of the CPU is already entered in the "occupied" register.
0	1	The "occupied" register contains a different slot ID.

Note
All CPUs that require **synchronized** access to a **common global memory area** (dual-port RAM area) must use the TSC operation.

Error reaction

The location must be on the corresponding module and on the common page between F F400H and F FBFFH. If this is not the case, the CPU recognizes a transfer error (TRAF) and calls **OB 32**. If OB 32 is not loaded, the CPU changes to the stop mode with the error code TRAF (ISTACK).

Load and transfer operations for the dual-port RAM memory organized in bytes

Table 9-11 Operations for access to pages organized in bytes

Operation	Operand	Description
LY CB	-32768 to +32767	add the specified constant to content of the BR register and load the byte addressed in this way in the open page into ACCU-1-LL 1) 3)
LY CW	-32768 to +32767	add the specified constant to content of the BR register and load the word addressed in this way in the open page into ACCU-1-L 2) 3)
LY CD	-32768 to +32767	add the specified constant to content of the BR register and load the double word addressed in this way in the open page into ACCU 1 3)

Operation	Operand	Description
Table 9-11 continued:		
TY CB	-32768 to +32767	add the specified constant to content of the BR register and transfer the content of ACCU-1-LL to the byte addressed in this way in the open page
TY CW	-32768 to +32767	add the specified constant to content of the BR register and transfer the content of ACCU-1-L to the word addressed in this way in the open page
TY CD	-32768 to +32767	add the specified constant to content of the BR register and transfer the content of ACCU 1 to the double word addressed in this way in the open page.

[1] ACCU-1-LH and ACCU-1-H are set to ’0’.

[2] ACCU-1-H is set to ’0’.

[3] ACCU 2 $_{new}$: = ACCU 1$_{old}$

Error reaction

The range of absolute addresses must be

- between F F400H and F FBFFH (LY CB, TY CB),
- between F F400H and F FBFEH (LY CW, TY CW)

  or

- between F F400H and F FBFCH (LY CD, TY CD).

If the absolute addresses are not in the range shown, the CPU detects a load/transfer error (TRAF) and calls **OB 32**. If OB 32 is not loaded, the CPU changes to the STOP mode with the error bit TRAF (ISTACK).

Load and transfer operations for the dual-port RAM memory organized in words

Table 9-12 Operations for access to pages organized as words

Operation	Operand	Description
LW CW	-32768 to +32767	add the specified constant to content of the BR register and load the word addressed in this way in the open page into ACCU-1-L [1)]
LW CD	-32768 to +32767	add the specified constant to content of the BR register and load the double word in the page addessed in this way in the open page into ACCU 1 [2)]
TW CW	-32768 to +32767	add the specified constant to content of the BR register and transfer the contents of ACCU-1-L to the word in the open page addressed in this way.
TW CD	-32768 to +32767	add the specified constant to content of the BR register and transfer the contents of ACCU 1 to the double word in the open page addressed in this way.

[1)] ACCU-1-H is set to ’0’.

[2)] ACCU 2 $_{new}$: = ACCU 1$_{old}$

Error reaction

The range of absolute addresses must be

- between F F400H and F FBFFH (LW CW, TW CW)

  or

- between F F400H and F FBFEH (LW CD, TW CD).

If the absolute addresses are not in the range shown, the system program detects a load/transfer error (TRAF) and calls **OB 32**. If OB 32 is not loaded, the CPU changes to the STOP mode with the error bit TRAF (ISTACK).

Multiprocessor Mode and Communication in the S5-155U

10

Contents of Chapter 10

Multiprocessor Mode and Communication in the S5-155U

10

At the beginning of this chapter, you will see when you can use the multiprocessor mode and which data exchange is possible in this mode. The chapter provides you with information about programming for multiprocessor operation (Section 10.1).
The second part of the chapter provides you with detailed instructions and examples of exchanging larger amounts of data in the multiprocessor mode (multiprocessor communication Sections 10.2 to 10.9).

10.1 Multiprocessor Mode

Definitions of terms

The S5-155U is set up for multiprocessor operation as soon as you plug in a coordinator module, regardless of how many CPUs or CP/IPs are plugged in. The CPUs must be plugged in without any gaps between them.

10.1.1 When to use the Multiprocessor Mode

- If your user program is too large for one CPU and there is not enough memory, distribute your program on several CPUs.
- When a particular part of your system has to be processed especially fast, separate the appropriate program part from the total program and run it on its own fast CPU.
- When your system consists of several parts that you can separate easily and control independently, let CPU 1 process system part 1, CPU 2 process system part 2, etc.

For more information on multiprocessing, read the information in your system manual. This will help you to decide which CPUs are best suited for your problem.

10.1.2 What Communications Mechanisms are Available?

- **"Interprocessor communication flags"** are available for cyclic exchange of binary data between CPUs (CPU 948, CPU 946/947, CPU 928B, CPU 928 and CPU 922) or between CPUs and communications processors (CPs).
- For the exchange of large amounts of data (e.g., entire data blocks) between the CPU 948, CPU 946/947, CPU 928B, CPU 928 and CPU 922 you are supported by the **"special functions for multiprocessing"** OB 200 to OB 205 (for more information refer to Section 10.2).
- **"Handling blocks"** are available for communication with intelligent input/output modules (IPs) and with CPs (These handling blocks must be ordered separately).

10.1.3 Exchanging Data via IPC Flags

Interprocessor communication (IPC) flags are available for cyclic exchange of binary data. They are used mainly for transmitting information **byte by byte**.

Data is transferred as follows:

CPU(s)	↔	CPU(s)
CPU(s)	↔	Communications processor(s)

The system program transfers IPC flags once per cycle. For data transfer between CPUs, the IPC flags are buffered physically on the coordinator.

IPC flags are bytes that are transferred. You define them in DB 1 for each CPU as IPC input or output flags. If, for example, you have defined flag byte 50 on the CPU 1 as an IPC **output** flag byte, its signal state is transferred cyclically via the coordinator to the CPU on which the flag byte F 50 is defined as an IPC **input** flag byte.

Note
There is **no** error message when the IPC flag byte exists physically but is only written by one CPU and never read out and vice-versa.

Memory area

With the CPU 948 the memory area for the IPC flags in the coordinator and the CPs covers the addresses **F F200H to F F2FFH**. On a CPU/communications processor there are 256 available IPC flag bytes.

Jumper settings

To avoid double assignments you must group the 256 available IPC flag bytes on the COR or CP modules. Fields of 32 bytes can be enabled or disabled (your system manual contains information about setting the jumpers).

Example

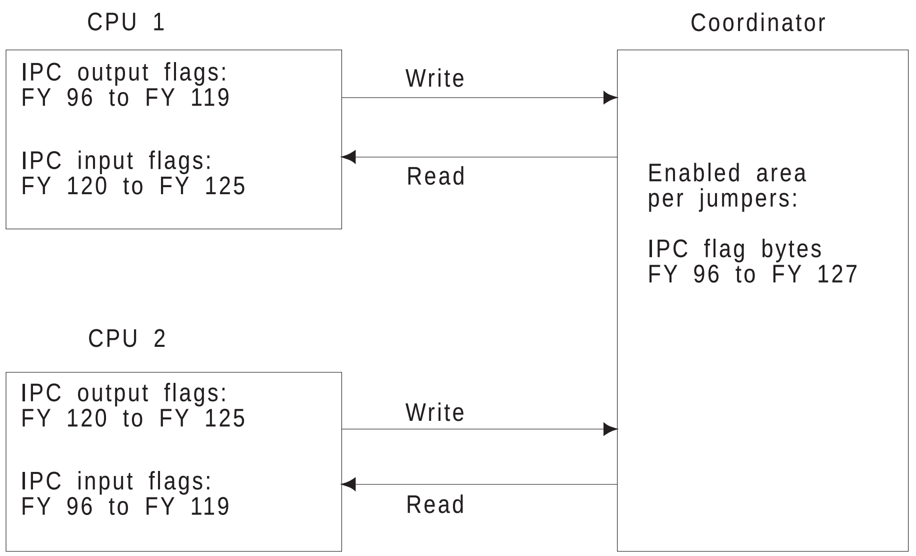


Fig. 10-1 Transferring IPC flags in the multiprocessor mode

Note
The only flag bytes that you can specify as IPC flags are the ones enabled on the coordinator or on the CP(s). **S flags cannot be used as IPC flags!**

A flag byte that is defined on one or more CPUs as an IPC **input** flag byte must be defined as an IPC **output** flag byte on one other CPU or CP. An **IPC output flag byte** is only allowed on **one CPU**, but this may be used as an IPC input flag in all other CPUs in the rack.

If you have flag bytes that you have not defined as IPC flags in a CPU, you can use them as normal flags!

In DB 1, indicate only the number of IPC flag bytes that you actually need: the smaller the number of IPC flag bytes, the shorter the transfer time!

Data exchange between CPUs and communication processors

If you want to exchange data between one CPU and one CP, you must enable the necessary number of IPC flags on the CP. You have 256 bytes available that you can divide into groups of 32 bytes.

If you want to transfer data from one CPU to several CPs, the areas you enable in the CPs and the coordinator must **not overlap**, otherwise the same address is assigned twice.

If you want to use IPC flags **simultaneously** on the coordinator and in one or more CPs, you must also prevent double addressing as follows: Divide the IPC flags among the coordinator and the CPs in groups of 32 bytes. Remove jumpers on the coordinator to mask the IPC flag bytes that you want to use in the CP (refer to the system manual).

You can define a specific flag byte as an IPC output flag in **one** CPU only. However, you can define a specific flag byte as in IPC input flag in several CPUs.

Example

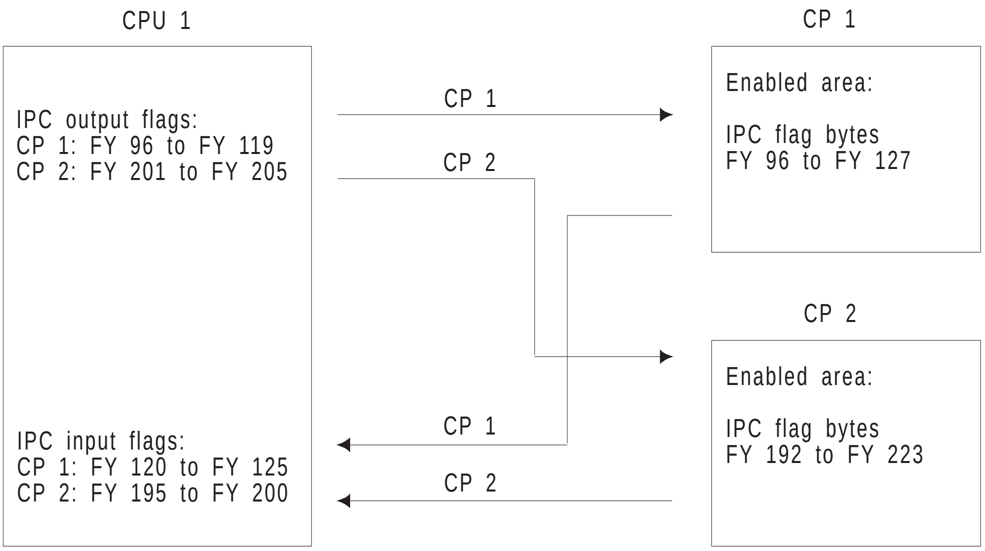


Fig. 10-2 Example of IPC flag areas on the CPs

Transmitting IPC flags in multiprocessor operation

At the end of each program cycle, along with the updating of the process image, the CPU transmits the IPC flags specified in DB 1 when the coordinator signals the CPU that it can access the S5 bus.

The coordinator allocates the bus enable signal to each CPU in sequence. When a CPU has access to the S5 bus, it can transmit only **one** byte. Because of this interleaved transmission, related (byte groups) IPC flag information can be separated and subsequently processed with old or incorrect values.

If you want to transfer information that takes up more than **one** byte, you can prevent corruption of data by setting a parameter in extended data block DX 0. This parameter uses semaphores to ensure that all IPC flags specified in DB 1 are transferred in groups (see Chapter 7). While one CPU is transmitting IPC flags, another CPU cannot interrupt it. Because the next CPU has to wait to transmit its data, cyclic program processing of this CPU is delayed accordingly. **Under certain circumstances, the setting you make in DX 0 can increase the cycle time considerably.**

Multiprocessor communication

For transferring data blocks or more exactly fields of data with a size of max. 64 byte (= 32 data words), the following special functions are integrated in the CPU:

• OB 200: INITIALIZE:	preassign
• OB 202: SEND:	send a data field
• OB 203: SEND TEST:	test sending capacity
• OB 204: RECEIVE:	receive a data field
• OB 205: RECEIVE TEST:	test receiving capacity

10.1.4 Exchanging Data via Handling Blocks

Handling blocks are capable of multiprocessing. A special parameter assignment for the multiprocessor mode is not necessary. For more information on handling blocks refer to the appropriate manual.

10.1.5 What needs to be Programmed for the Multiprocessor Mode?

- To allow the coordinator to coordinate access by the individual CPUs to the I/O area, **you must program data block DB 1**. Even if the CPU does not use I/Os or the IPC flags, a (empty) DB 1 must exist (for more information refer to Section 10.1.6)
- Data block DX 0 must also exist. Program this so that process interrupts via input byte IB 0 are disabled and the **system interrupts are activated**.

10.1.6 How to Create Data Block DB 1

For multiprocessing, you must program DB 1 for each CPU. This establishes the **inputs, outputs** (byte addresses 0 to 127) and **IPC input and output flags with which each CPU works.**

> **Note**
> The system program recognizes only the inputs and outputs defined in DB 1 when it updates the process image!

Inputting or changing DB 1

- Create/modify DB 1 on the PG using the DB 1 screen form

  or

- by editing DB 1 as a data block on the PG and then transferring it to the CPU.

> **Note**
> The CPU accepts the entered or changed DB 1 only after a cold restart!

Using the DB 1 screen form

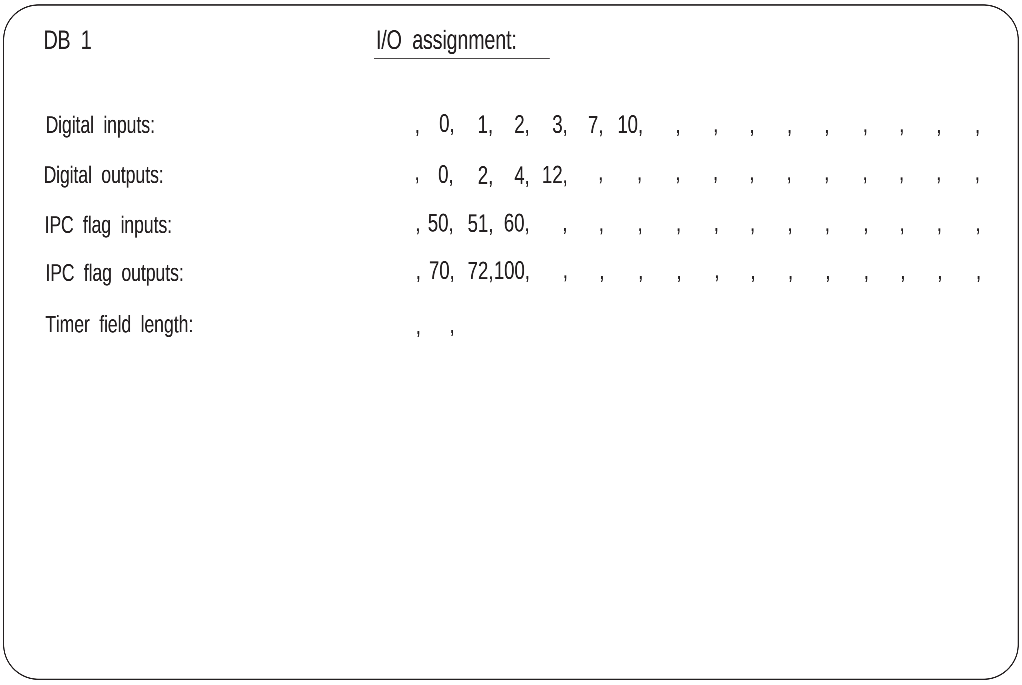
1. Select the editor for the DB 1 screen form on your PG (refer to Fig. 10-3).
2. Enter the required values for "digital inputs" etc. as decimal numbers.

3. Enter the values by pressing the enter key on the PG.
   The PG then generates DB 1.

4. Transfer DB 1 to the CPU.

> **Note**
> Entry of the timer field length is ignored! This parameter must be specified in DX 0 (see Chapter 7).

Example of the DB 1 screen form



```
DB 1                                 I/O assignment:

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 field length:                  ,  ,
```

Fig. 10-3 PG screen form for generating DB 1

Editing DB 1 as a data block

1. Write the DB 1 start ID in data words 0, 1 and 2:

   DW 0: KH = 4D41 (’M’ ’A’)
   DW 1: KH = 534B (’S’ ’K’)
   DW 2: KH = 3031 (’0’ ’1’)

2. Type in the individual operand areas (from data word 3 onwards). Before each operand area, you must specify an ID. The possible ID words are as follows:

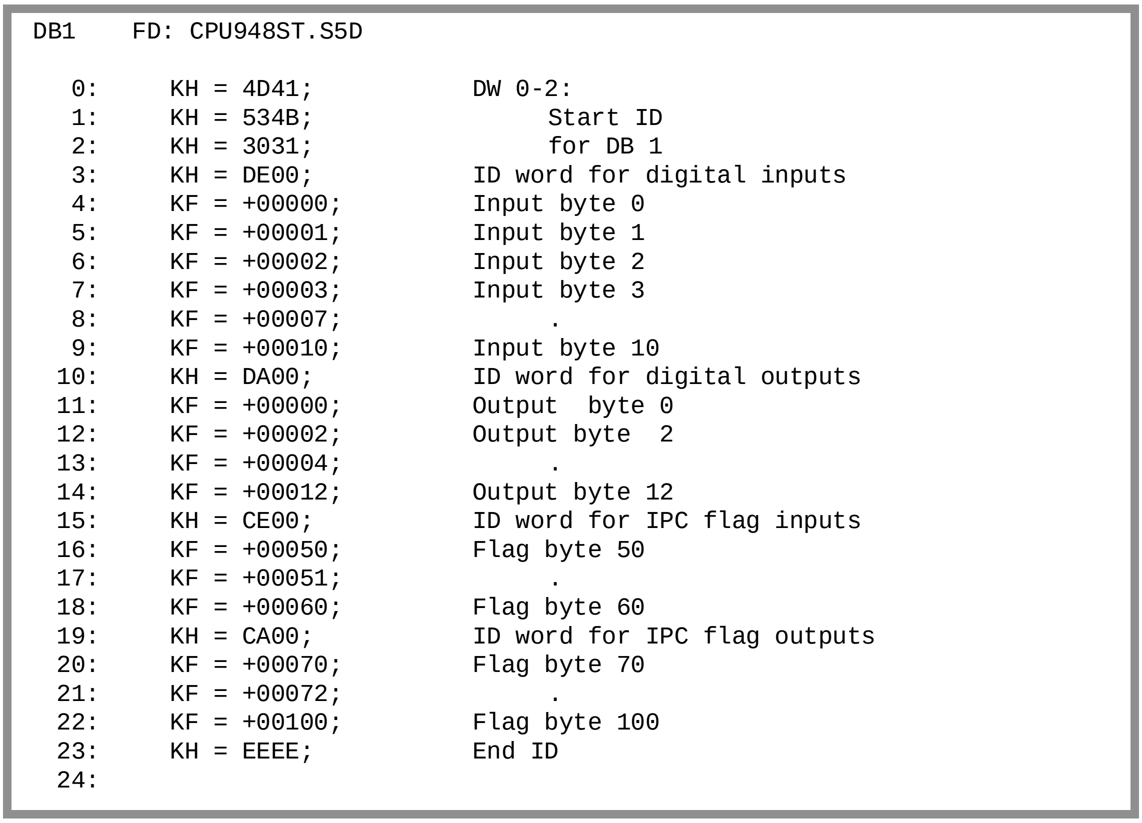
ID word for digital inputs	KH = DE00
ID word for digital outputs	KH = DA00
ID word for IPC input flags	KH = CE00
ID word for IPC output flags	KH = CA00

   After each ID word, use fixed-point format to list the numbers of the inputs and outputs used.

3. Complete the entries with the DB 1 end ID "KH = EEEE" and transfer DB 1 to the CPU.

Note
You can make the DB 1 entries in any order. Remember that the process image of the inputs and outputs is updated in the **reverse order** to which you store the addresses in DB 1 (i.e. the last entry is updated first).
Multiple entries of the same bytes (e.g., for test purposes) are possible. The system program makes multiple updates of the process images of bytes that are entered more than once.

Example of editing DB 1



```
DB1    FD: CPU948ST.S5D

   0:     KH = 4D41;           DW 0-2:
   1:     KH = 534B;                Start ID
   2:     KH = 3031;                for DB 1
   3:     KH = DE00;           ID 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;         Input byte 10
  10:     KH = DA00;           ID word for digital outputs
  11:     KF = +00000;         Output  byte 0
  12:     KF = +00002;         Output byte  2
  13:     KF = +00004;              .
  14:     KF = +00012;         Output byte 12
  15:     KH = CE00;           ID word for IPC flag inputs
  16:     KF = +00050;         Flag byte 50
  17:     KF = +00051;              .
  18:     KF = +00060;         Flag byte 60
  19:     KH = CA00;           ID word for IPC flag outputs
  20:     KF = +00070;         Flag byte 70
  21:     KF = +00072;              .
  22:     KF = +00100;         Flag byte 100
  23:     KH = EEEE;           End ID
  24:
```

Entering DB 1

The system program adopts DB 1 during a cold restart. The system program checks to see if the inputs and outputs or IPC flags indicated in DB 1 exist in their corresponding modules. If they are not present there, a DB 1 error causes the CPU to go into the STOP mode and the STOP LED flashes slowly. The CPU no longer processes your program.

After you program DB 1 and the CPU accepts it during a cold restart, the following rules apply:

- Only the inputs and outputs indicated in DB 1 can access peripheral modules via the process images (L.../T... ...IB, ...IW, ...ID, ...QB, ...QW, ...QD operations and logic operations with inputs and outputs). Access to process image addresses not entered in DB 1 cause addressing errors.

- You can **load** peripheral bytes directly by bypassing the process image using the L PB/L PY, L PW, L OY, L OW operations for all acknowledging inputs, regardless of entries in DB 1.

- You can **transfer** directly (T PB/T PY, T PW) to bytes 0 to 127 only for the outputs indicated in DB 1. This is because the process image is also written to during direct transfer. Writing to I/O addresses not entered in DB 1 causes an addressing error.

- **Transfer without a process image** :
  Direct transfer to byte addresses **>127** is possible **regardless of the entries in DB 1**.
  Direct transfer of byte addresses of the extended I/Os (T OY, T OW) is also possible regardless of the entries in DB 1.

10.1.7 Starting up in the Multiprocessor Mode

You can start the coordinator for multiprocessing in one of the following ways:

Initial status: The RUN/STOP switch of each CPU is in the RUN position. The RUN/STOP switch on the coordinator is in the STOP position.

Handling: Move the RUN/STOP switch on the coordinator from STOP to RUN. (Starting the S5-155U in multiprocessor operation simply by starting the coordinator is possible only if the coordinator itself caused the controller to change to the STOP mode).

or:

Initial status: The RUN/STOP switch of each CPU is in the RUN position as well as that of the coordinator.

Handling: Use the PG START function to start the CPU that caused the STOP in the required restart type.

Starting up individual CPUs

The restart type that each CPU now uses depends on what took place while the CPU was in the STOP mode. Some CPUs need MANUAL WARM RESTART, others, a COLD RESTART.

If the CPU settings were not changed in that time, execute a **manual warm restart**.

> **Note**
> Due to the various restart types, incorrect signal statuses can be transferred from one CPU to another via the IPC flags. This could happen if the controller was in cycle prior to entering the STOP mode. Prevent this by programming the start-up organization blocks OB 20, OB 21 and OB 22 appropriately.

You can call special function block OB 223 to check whether the start-up types of all the CPUs are the same (refer to Chapter 6).

Power failure/return of power

When power is shut off and then restored, the coordinator starts automatically. In this case, all CPUs execute an **AUTOMATIC WARM RESTART** or an **AUTOMATIC COLD RESTART**, depending on the setting in DX 0 (see Chapter 7).

The start-up of the individual CPUs in multiprocessor operation is synchronized. Each CPU waits until all others have ended their initialization phase. Then they begin their cycle simultaneously. However, you can use a setting in DX 0 to cancel start-up synchronization.

10.1.8 Test Mode

Proceed as follows:

1. Make sure that the "test mode" function is enabled on the coordinator.
2. Switch the mode selector on the COR from STOP to TEST.
   The BASP LED goes off.
3. Go through a COLD RESTART or WARM RESTART on the CPUs you want to change to the RUN mode.

Special features of test operation

In test operation, you can run CPUs **individually** or in any combination. CPUs in the STOP mode cannot disable the entire PLC.

Start-up of individual CPUs is **not** synchronized in test operation. The CPUs begin their cyclic operation at different times according to the length of the start-up organization blocks OB 20, OB 21 or OB 22.

If an error occurs in one CPU during test operation, **only that** CPU goes into the STOP mode. The error does not affect the other CPUs.

Warning
Since no CPU can output the BASP signal in case of an error in the test mode, the test mode must be switched inactive after successful installation to avoid a critical or even dangerous situation arising in the system.

10.2 Multiprocessor Communication

Definition

Multiprocessor **communication** means the exchange of larger amounts of data (data blocks) between CPUs operating in the multiprocessor mode. The COR C is necessary for multiprocessor communication.

10.2.1 Introduction

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:

• OB 200: INITIALIZE:	preassign
• OB 202: SEND:	send a field of data
• OB 203: SEND TEST:	test sending capacity
• OB 204: RECEIVE:	receive a data field
• OB 205: RECEIVE TEST:	test receiving capacity

The special function OBs, OB 200 and OB 202 to OB 205 are simply called "communication OBs" in the following sections.

Required knowledge

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 Further Reading.

Basic sequence

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" copy operation.

Data fields

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 field of data to be transferred is an exception and is less than 32 words long.

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.

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

10.2.2 How the Transmitter and Receiver are Identified

Each field of data exchanged between the CPUs is marked with a number to indicate the source and destination CPU.
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:

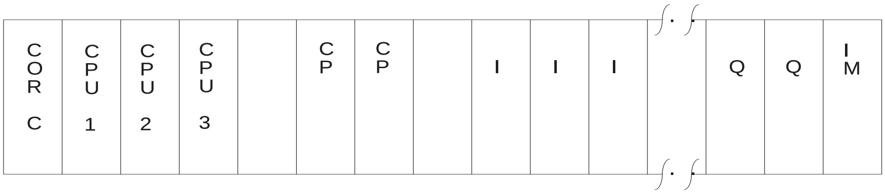


Fig. 10-4 Sender/receiver identification

10.2.3 Why Data is Buffered

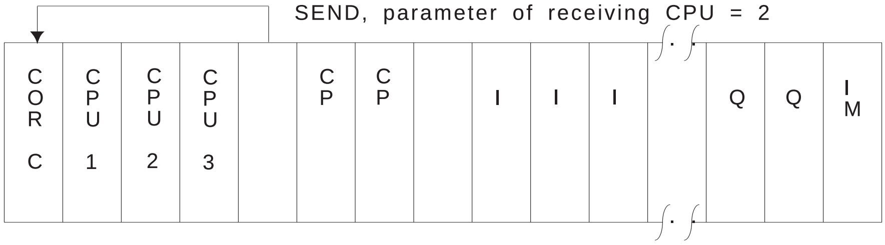
Generally, the multiprocessor mode is used to distribute tasks on several CPUs. Since the tasks are not identical and the performance of the CPUs involved can be different, the program execution of the individual CPs in the multiprocessor mode is always **asynchronous**. This means that the data sent by a CPU cannot always be received immediately by another CPU.

For this reason, the data to be transferred is buffered on the coordinator 923 C. The number of the sender and receiver are always included along with the data.

Example

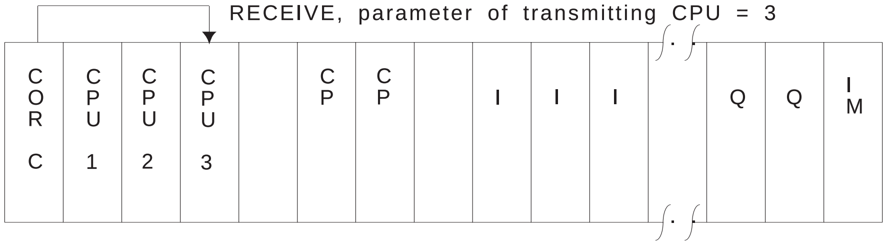
Data transfer from CPU 3 to CPU 2:

1st step:



CPU 3 buffers its data on the coordinator.

2nd step:



When CPU 2 is ready to receive, it copies the data from the coordinator buffer to the destination DB.

10.2.4 How the Buffer is Processed and Managed

Principle

The buffer is based on the FIFO principle (first in **-** first out, queue principle). The data is received in the order in which it is sent. This applies to each individual link (identified by the transmitting and receiving CPU) and is independent of other links.

Data protection

The buffer is battery-backed; this means that the "automatic warm restart following a power down" is possible without any restrictions. A loss of power during a data transfer does not cause any loss of data in the programmable controller.

Management

The coordinator 923 C has a memory capacity of 48 data fields each with a fixed length of 32 words. The INITIALIZE function assigns these fields to individual CPU links.
Each **memory field** can receive exactly one **field of data**. The length of the data can be from 1 data word to 32 data words. A **data field** is entered in a **memory field** by a SEND function and read out again by a RECEIVE function.
The number of memory fields assigned to a link is directly related to the parameters for the transmitting capacity (SEND, SEND TEST function) and receiving capacity (RECEIVE, RECEIVE TEST function).

The **transmitting capacity** indicates how many of the memory fields reserved for a link are free at any particular time.

The **receiving capacity** indicates how many of the memory fields reserved for a link are occupied at any particular time.

The sum of the transmitting and receiving capacity is always equal to the number of memory fields reserved for a link.

Example

Occupation of the buffer by a link

The link between CPU 3 and CPU 2 is initialized. The link is assigned seven memory fields in the buffer of the coordinator. Following this, the data transfer shown below would be possible.

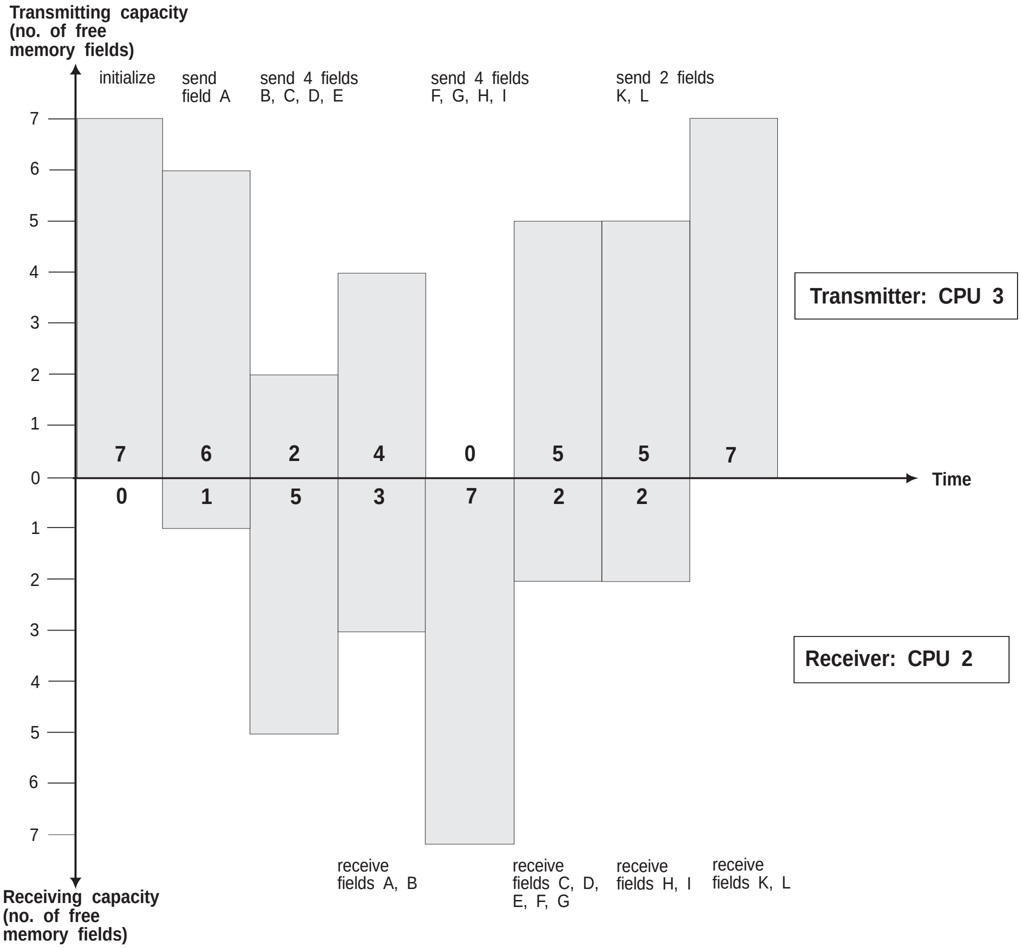


Fig. 10-5 Example of the occupation of the COR buffer

Sending/receiving n data fields means that the corresponding functions are called n times one after the other.

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 ("Parallel processing in a multiprocessor programmable controller"). In the example, fields H and I are received while fields K and L are sent.

The example illustrates the queue organization of the buffer: the fields of data sent first (A,B,C...) are received first (A,B,C...).

Summary

Buffering data on the coordinator COR 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).

Note
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

  or

- there are not enough memory fields assigned to the link.
  The capacity of the buffer is insufficient to compensate temporary imbalances in the frequency with which the CPUs transmit and receive data.

10.2.5 System Start-Up

If you require multiprocessor communication, then all 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 Section 10.1.7), 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) **only one** CPU must set up the buffer (in the COR 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.

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 sections).

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.

10.2.6 Calling Communication OBs

Proceed as follows:

1. Call the INITIALIZE function only in the cold restart organization block OB 20 on one CPU.
2. Call the SEND, SEND TEST, RECEIVE, RECEIVE TEST functions either **only** within the cyclic program or **only** within the time-driven program.

Double call

Depending on the assignment of parameters in DX 0 ("interrupts at operation boundaries"), 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 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 10.2.8).

Parallel processing

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 link (e.g. from CPU 2 to CPU 3) it is possible to execute the SEND function (CPU 2) and the RECEIVE function (CPU 3) simultaneously. While CPU 2 is sending data fields to the coordinator, CPU 3 can already receive (fetch) buffered data fileds from the coordinator.

Areas occupied

The communication OBs do not require a working area (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.

Results bits

The results bits (CC 1/CC 0, RLO etc.) are influenced by the communication OBs. For more detailed information refer to Section 10.2.8.

Changes in the ACCUs

- 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 communication OBs.
- CPU 946/947, CPU 948: The contents of all registers and ACCU 1, 2 and 3 remain the same, only the contents of ACCU 4 are affected.

10.2.7 How to Assign Parameters to Communication OBs

The communication OBs have the following types of parameter:

- input parameters,
- output parameters

  and
- call parameters.

Input and output parameters are located in a maximum 10 byte long **data field in the F flag area**. The data field is divided into an area for **input parameters** and an area for **output parameters**.

Input parameters

The input parameters specify how a function is handled. All or part of the parameters are read out by communication OBs and evaluated, no write access takes place.

Output parameters

The output parameters contain all the information that the calling program needs about the result of a job, e.g. error bits.
Some or all of the output parameters are written to by the communication OBs, this area is not read.

> **Note**
> 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 (Section 10.4ff).

Call parameters

For all communication OBs the number of the first flag byte in the data field (= pointer to data field) in ACCU-1-L is transferred as the call parameter. Permitted values are 0 to 246.

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).

10.2.8 How to Evaluate the Output Parameters

Among other things, the output parameters indicate whether or not a function could be executed and if not they indicate the reason for the termination of the function.

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 1 and CC 0 indicate whether a function has been executed correctly and completely.

Table 10-1 Condition codes of the communication OBs

Condition codes			**Evaluation**	**Meaning**
RLO	**CC 1**	**CC 0**		
0	0	0	JC=	Function executed completely and correctly
1	0	0	JC=	Function aborted, pointer to data field illegal (>246) Function aborted owing to an initialization conflict
1	0	1	JC= and JM=	Function aborted owing to an error (error number 1 to 9)
1	1	0	JC= and JP=	Function aborted owing to a warning (warning number 1 or 2)

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.

Condition code byte

Bit no.	7	6	5	4	3	2	1	0
	W	E	I	0	Number			

W = 1: Warning

E = 1: Error

I = 1: Initialization conflict

Number: - of a warning
- of an error
- of an initialization conflict

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 number 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 number 6 = 1) or an initialization conflict (bit number 5 = 1), all other output parameters remain unchanged.

Evaluation of the code byte

The identifiers ’W’, ’E’ and ’I’ indicate the significance of the numbers.
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:

Table 10-2 Code byte for the communication OBs/number groups

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

Errors are detected and indicated in the ascending order of the error numbers. This means that several errors may have occurred although (currently) only **one** is indicated. The other errors are then indicated by further calls.

Example

```
The SEND function indicates an error and is not
executed. If you then make program and/or
parameter modifications and the SEND function
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 the parameters.

Initialization conflict numbers (evaluation of the condition code byte as a byte):

Table 10-3 Condition code byte: Initialization conflict numbers

Cond. code byte	**Significance**
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 incorrect, since it is a data block with a special significance. The following errors are possible: - if block ID = 1 DB 0, DB 1 - if block ID = 2 : DX 0
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.

Cond. code byte	Significance
Table 10-3 continued:	
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):

Table 10-4 Condition code byte: Error numbers

Cond. code byte	Significance
65	The parameter "receiving CPU" (SEND, SEND TEST) 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.
66	The parameter "transmitting CPU" (RECEIVE, RECEIVE TEST) 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 (SEND, RECEIVE, SEND TEST, RECEIVE TEST). The following errors are possible: - Secondary error, since the INITIALIZE function could not be called or was terminated by an initialization conflict. - Double call: the call for this function (SEND, SEND TEST, RECEIVE or RECEIVE TEST) is illegal, since one of these functions INITIALIZE, SEND, SEND TEST, RECEIVE or RECEIVE TEST has already been called in this CPU in a lower processing level (i.e. cyclic program execution). - 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.

Cond. code byte	Significance
Table 10-4 continued:	
68	The management data (queue management) of the selected links 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 - if the block ID = 2 : DX 0
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 data field 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):

Table 10-5 Condition code bytes: Warning numbers

Cond. code byte	**Significance**
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.

10.3 Runtimes of the Communication OBs

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, etc.).

Table 10-6 Runtimes of the communication OBs

Special function OB					
Block name	**CPU 922**	**CPU 928**	**CPU 928B**	**CPU 946/ 947**	**CPU 948**
OB 200/ initialize	230 ms	130 ms	130 ms	128 ms	90 ms
OB 202/ send	806 µs (294 µs basic time + 16 µs/word); 118 µs if a warning occurs	666 µs (250 µs basic time + 13 µs/word); 115 µs if a warning occurs	696 µs (280 µs basic time + 13 µs/word); 145 µs if a warning occurs	762 µs (426 µs basic time + 21 µs/ double word); 243 µs if a warning occurs	542 µs (220 µs basic time + 19 µs/ double word); 110 µs if a warning occurs
OB 203/ send test	72 µs	50 µs	80 µs	207µs	115 µs
OB 204/ receive	825 µs (281 µs basic time + 17 µs/word); 115 µs if a warning occurs	660 µs (244 µs basic time + 13 µs/word); 98 µs if a warning occurs	690 µs (274 µs basic time + 13 µs/word); 128 µs if a warning occurs	772 µs (421 µs basic time + 22 µs/ double word); 243 µs if a warning occurs	506 µs (218 µs basic time + 18 µs/ double word); 132 µs if a warning occurs
OB 205/ receive test	70 µs	48 µs	78 µs	223 µs	120 µs

The runtimes listed in table 10-6 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.

Transfer time

An important factor of a link (e.g. from CPU 1 to CPU 2) 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 COR 923C coordinator)

  and

- 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").

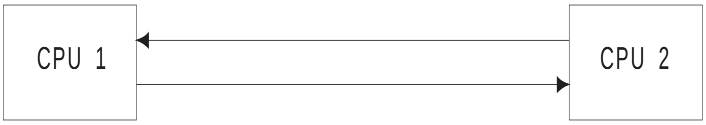
10.4 INITIALIZE Function (OB 200)

10.4.1 Function

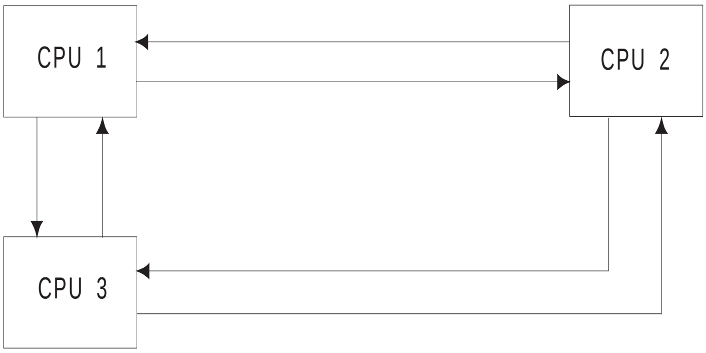
To transfer data from one CPU to another CPU, the data must be temporarily buffered. The INITIALIZE function sets up a buffer on the COR 923C coordinator.
The memory is initialized in fields with a fixed length of 32 words.

Each memory field accepts one data field with a length between 1 data word and 32 data words. A data field is entered in a memory field by a SEND function and read out by a RECEIVE function.

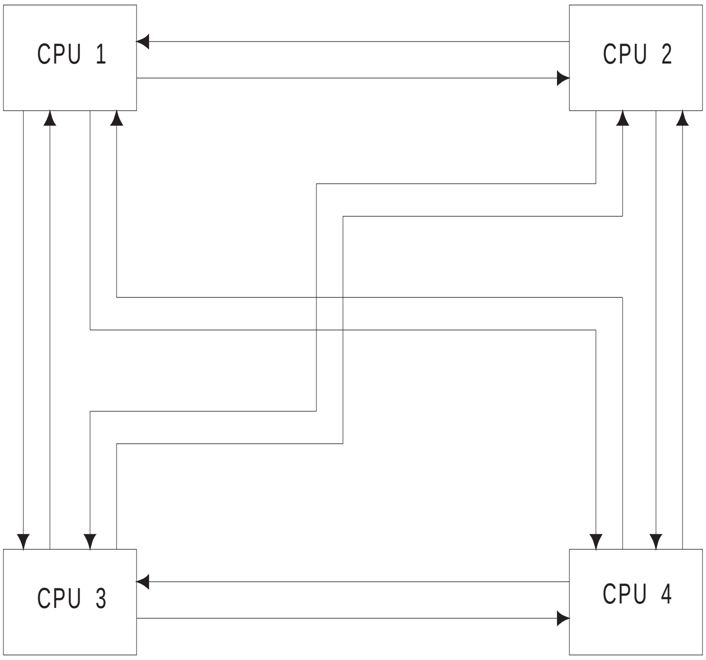
If you are using two CPUs, there are two links (transfer directions, "channels"):



If you are using three CPUs, there are six links:



If you are using four CPUs, there are twelve links:



The INITIALIZE function specifies how the total of **48** available memory fields are assigned to the maximum twelve links.
This means that each possible link, specified by the parameters "transmitting CPU" and "receiving CPU" has a certain memory capacity available.

Note
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.

If the INITIALIZE function is called several times, one after the other, the last assignment made is valid. While a CPU is processing the INITIALIZE function, no other multiprocessor communication functions including the INITIALIZE function can be called on other CPUs.

10.4.2 Call Parameters

Structure of the (parameter) 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: FY x + 5:	Start address of the assignment list	input parameter
FY x + 6:	Condition code byte	output parameter
FY x + 7:	Total capacity	output parameter

ACCU-1-L

When OB 200 is called, you transfer the flag byte number at which the parameter data field begins to ACCU-1-L:

ACCU-1-LH:	0
ACCU-1-LL:	0 to 246

10.4.3 Input Parameters

Mode (automatic/manual)

Mode = 1:	automatic
Mode = 2:	manual
Mode = 0 or 3 to 255:	illegal, causes an initialization conflict

Number of CPUs

This parameter is only relevant when you have selected the "automatic" mode. With the "automatic" setting, the memory fields are divided **evenly** according to the number of CPUs.

Number of CPUs	Number of links	Memory fields per link
2	2	24
3	6	8
4	12	4
0; 1; 5 to 255	Illegal, causes an initialization conflict	

Block ID, block number, address assignment list

The parameters are only relevant if you select the "manual" mode. You must then create an assignment list in a data block in which the 48 available memory fields (or less) are assigned to the maximum 12 links. This function is particularly useful when some CPUs transfer more data than others.
The CPUs not involved in the multiprocessor communication do not need and should not have memory fields assigned to them.
The parameters

- block ID,
- block number

  and

- start address of the assignment list

specify where the assignment list is stored.

Block ID

ID = 1:	DB data block
ID = 2:	DX data block
ID = 0 or 3 to 255 :	illegal, causes an initialization conflict

Block number

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 the data block in which the assignment list is stored.
As the address of the assignment list, specify the data word number at which the assignment list begins in flag bytes FY x+4 (high byte) and FY x+5 (low byte).

Assignment list

With the assignment list, you specify how many of the existing 48 memory fields are to be assigned to the links.

The list is **not changed** by the system program. It has the following structure.

Table 10-7 Assignment list for OB 200 (initialize)

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

Instead of the lower case letters a to m (in bold face) numbers between 0 and 48 must be inserted depending on the number of assigned memory fields. **The sum of these numbers must not exceed 48**.

Note
You must keep to the structure shown in table 10-7 even if you have less than four CPUs.

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**.

The assignment list is stored in data block DB 40 from DW 0 onwards and has the following parameters:

```
DB40    FD:  CPU948ST.S5D

   0:     KS = S1;       Transmitter:    CPU 1
   1:     KY = 2,2;      Receiver:       CPU 2/2 fields
   2:     KY = 3,0;      Receiver:       CPU 3/no field
   3:     KY = 4,0;      Receiver:       CPU 4 (does not exist)/no field
   4:     KS = S2;       Transmitter:    CPU 2
   5:     KY = 1,22;     Receiver:       CPU 1/22 fields
   6:     KY = 3,22;     Receiver:       CPU 3/22 fields
   7:     KY = 4,0;      Receiver:       CPU 4 (does not exist)/no field
   8:     KS = S3;       Transmitter:    CPU 3
   9:     KY = 1,0;      Receiver:       CPU 1/no field
  10:     KY = 2,2;      Receiver:       CPU 2/2 fields
  11:     KY = 4,0;      Receiver:       CPU 4 (does not exist)/no field
  12:     KS = S4;       Transmitter:    CPU 4 (does not exist)
  13:     KY = 1,0;      Receiver:       CPU 1/no field
  14:     KY = 2,0;      Receiver:       CPU 2/no field
  15:     KY = 3,0;      Receiver:       CPU 3/no field
  16:
```

10.4.4 Output Parameters

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.

All the numbers listed in Table 10-3 can occur in the condition code byte.

Errors

The "error" number group cannot occur with the INITIALIZE function.

Warning

The "warning" number group cannot occur with the INITIALIZE function.

Total capacity

This parameter specifies how many of the 48 available memory fields are assigned to links.
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.

10.5 SEND Function (OB 202)

10.5.1 Function

The SEND function transfers a data field to the buffer of the COR 923C coordinator. It also indicates how many data fields can still be sent or buffered.

10.5.2 Call Parameters

Structure of the (parameter) 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

ACCU-1-L

When OB 202 is called, transfer the flag byte at which the parameter data field begins to ACCU-1-L:

ACCU-1-LH:	0
ACCU-1-LL:	0 to 246

10.5.3 Input Parameters

Receiving CPU

CPU number of the receiver (destination); the permitted value is between 1 and 4 but must be different from the CPU’s own 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 and the field number 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:

- **DB is longer than source area**:
  If the data block is sufficiently long, you obtain a 32-word long area per field as shown in the table above.

- **DB is too short**:
  If the end of the data block is within the selected field, in the last field an area with a length between 1 and 32 words will be transferred.

- **Field is outside the DB:**
  If the first data word address of a field 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 no.: First data word: Last data word: Length:

   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
```

10.5.4 Output Parameters

Condition code byte

This byte informs you whether the SEND function was executed correctly and completely.

Initialization conflict

Has no significance with the SEND function.

Errors

When the SEND function is called, the following error numbers (evaluation of the condition code byte) can occur:

Condition code byte	Significance
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: - Secondary error, since the INITIALIZE function could not be called or was terminated by an initialization conflict. - 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 processing). - 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 links 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 - If the block ID = 2 : DX 0
71	The parameter "block number" 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.

The following warning numbers (evaluation of the condition code byte) can occur:

Condition code byte	**Significance**
129	The SEND function cannot transfer data, since the transmitting capacity was already zero when the function was called.

Transmitting capacity

The "transmitting capacity" indicates how many data fileds can still be sent and buffered.

10.6 SEND TEST Function (OB 203)

10.6.1 Function

The SEND TEST function determines the number of free memory fields in the buffer of the COR 923C coordinator.
Depending on this number m, the SEND function can be called m times to transfer m data fields.

10.6.2 Call Parameters

Structure of the (parameter) 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

ACCU-1-L

When OB 203 is called, transfer the flag byte number at which the parameter data field begins to ACCU-1-L:

ACCU-1-LH:	0
ACCU-1-LL:	0 to 246

10.6.3 Input Parameters

Receiving CPU

The CPU's own number and the number of the receiving CPU identify the link for which the transmitting capacity is determined.

10.6.4 Output Parameters

Condition code byte

This byte indicates whether the SEND TEST function was executed correctly and completely.

Initialization conflict

Has no significance for the SEND TEST function.

Errors

When calling the SEND TEST function, the following error numbers (evaluation of the condition code byte) can occur:

Condition code byte	Significance
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: - Secondary error, since the INITIALIZE function could not be called or was terminated by an initialization conflict. - 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 processing). - 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 links 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.

Transmitting capacity

The "transmitting capacity" parameter indicates how many data fields can be sent and buffered.

10.7 RECEIVE Function (OB 204)

10.7.1 Function

The RECEIVE function takes a data field from the buffer of the COR 923C coordinator. It also indicates how many data fields are still buffered and can still be received.
The RECEIVE function should be called in a loop until all the buffered data fields have been received.

10.7.2 Call Parameters

Structure of the (parameter) 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	

ACCU-1-L

When calling OB 204, transfer the flag byte number at which the parameter data field begins to ACCU-1-L:

ACCU-1-LH:	0
ACCU-1-LL:	0 to 246

10.7.3 Input Parameters

Transmitting CPU

The receive block receives data supplied by the transmitting CPU. Specify the number of the transmitting CPU. The permitted value is between 1 and 4, but must be different from the CPU’s own number.

10.7.4 Output Parameters

Condition code byte This byte informs you whether the RECEIVE function was executed correctly and completely.

Initialization conflict Has no significance with the RECEIVE function.

Errors When calling the RECEIVE function the following error numbers (evaluation of the condition code byte) can occur:

Condition code byte	Significance
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: - Secondary error, since the INITIALIZE function could not be called or was terminated by an initialization conflict. - 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 processing). - 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 links are incorrect; set up the buffer in the coordinator 923C again using the INITIALIZE function.
69	The block identifiers supplied by the transmitter are illegal. The following errors are possible: - The block ID is less than 1, - The block ID is greater than 2.

Condition code byte	Significance
Error numbers continued:	
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 - If the block ID = 2 : DX 0
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 data field supplied by the transmitter.

Warning

The function could not be executed; the function call must be repeated, e.g. in the next cycle.

The following warning number (evaluation of the condition code byte) can occur:

Condition code byte	Significance
130	The RECEIVE function cannot receive data, since the receiving capacity was already zero when the function was called.

Receiving capacity

The "receiving capacity" parameter indicates how many data fields are still buffered and can still be received.

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

Block number of the DB/DX in which the received data are stored (and from which they are taken by the SEND function in the transmitting CPU).

Remember that the receive data blocks must be in a random access memory, using read-only memories (EPROM) might possibly serve a practical purpose for transmit data blocks only.

Address of the first received data word

Data word number within the DB/DX in which the first transferred/received data word was stored.

Address of the last received data word

Data word number within the DB/DX in which the last transferred/received data word was stored.

> **Note**
> 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.

10.8 RECEIVE TEST Function (OB 205)

10.8.1 Function

The RECEIVE TEST function determines the number of occupied memory fields in the buffer of the COR 923C coordinator. Depending on this number m, the RECEIVE function can be called m times to receive m data fields.

10.8.2 Call Parameters

Structure of the (parameter) 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:	receiving capacity	output parameter

ACCU-1-L

When calling OB 204, transfer the flag byte number at which the parameter data field begins to ACCU-1-L:

ACCU-1-LH:	0
ACCU-1-LL:	0 to 246

10.8.3 Input Parameters

Transmitting CPU

The CPU’s own number and the number of the transmitting CPU identify the link for which the receiving capacity is determined.

10.8.4 Output Parameters

Condition code byte

This byte indicates whether the RECEIVE TEST function was executed correctly and completely.

Initialization conflict

Has no significance with the RECEIVE TEST function.

Errors

When calling the RECEIVE TEST function, the following error numbers (evaluation of the condition code byte) can occur:

Condition code byte	**Significance**
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: - Secondary error, since the INITIALIZE function could not be called or was terminated by an initialization conflict. - 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 processing). - 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 links 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.

Receiving capacity

The "receiving capacity" parameter indicates how many data fields can be received and buffered.

10.9 Applications

Based on examples, this section explains how to program multiprocessor communication.

Note
If you use the function blocks listed below and service interrupts on your CPU (e.g. with OB 2) remember to save the "scratchpad flags" at the start of interrupt servicing and to write them back when the interrupt is completed.
This also applies to the setting "interrupts at block boundaries", since the call of the special function organization blocks represents a block boundary.

10.9.1 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 data field
FB 203	SEND-TST	Test sending capacity
FB 204	RECEIVE	Receive a data field
FB 205	RECV-TST	Test receiving capacity

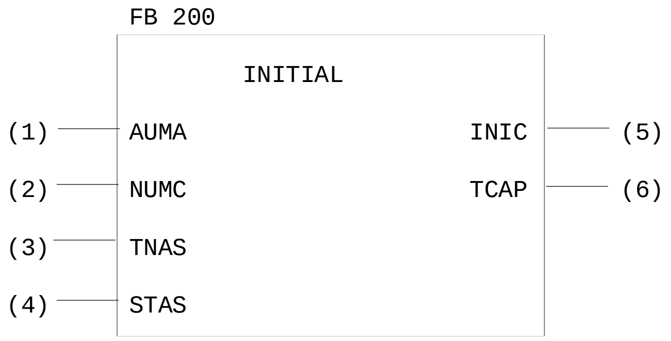
The flag area from FY 246 to maximum FY 255 is used by the function blocks as a parameter field for the special function organization blocks.

The exact significance of the input and output parameters is explained in the description of the special function organization blocks.

Note
The following examples of applications involve finished applications that you can program by copying them.

Programming function blocks

FB 200: initializing the links



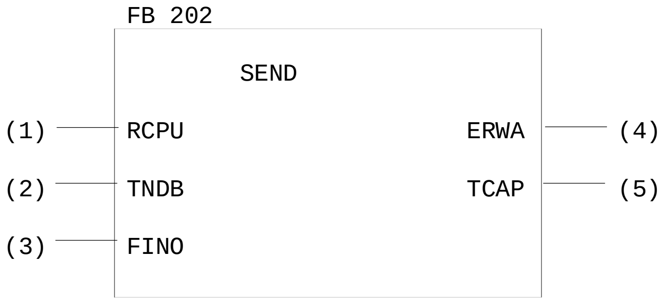
Parameter name	Significance	Parameter type	Data type	Parameter field
AUMA	**Au**tomatic/**ma**nual	I	BY	FY 246
NUMC	**Num**ber of **C**PUs	I	BY	FY 247
TNAS	**T**ype (H byte) and **n**umber (L byte) of the data block containing the **as**signment list	I	W	FW 248
STAS	**St**art address of the **as**signment list	I	W	FW 250
INIC	**Ini**tialization **c**onflict	Q	BY	FY 252
TCAP	**T**otal **cap**acity	Q	BY	FY 253

Continued on the next page

FB 200 continued

```
FB 200                                                  LEN=45
SEGMENT 1        0000
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    FY 248
001D     :L    =STAS             Start address of the assignment list
001E     :T    FW 250
001F     :
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
```

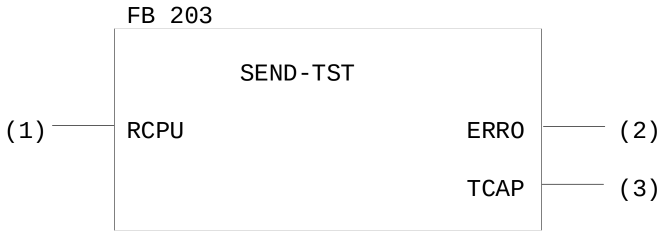
FB 202: Sending a data field



Parameter name	Significance	Parameter type	Data type	Parameter field
RCPU	**R**eceiving **CPU**	I	BY	FY 246
TNDB	**T**ype (H byte) and **n**umber (L byte) of the source **d**ata **b**lock	I	W	FW 247
FINO	**Field no**mber	I	BY	FY 249
ERWA	**Er**ror/**wa**rning	Q	BY	FY 250
TCAP	**T**ransmitting **cap**acity	Q	BY	FY 251

```
FB 202                                                    LEN=40

SEGMENT 1     0000
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  :FINO   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    =FINO            Field number
0019     :T    FY 249
001A     :
001B     :L    KB 246           SF OB:
001C     :JU   OB 202           "Send a data field"
001D     :
001E     :L    FY 250           Error/warning
001F     :T    =ERWA
0020     :L    FY 251           Transmitting capacity
0021     :T    =TCAP
0022     :BE
```

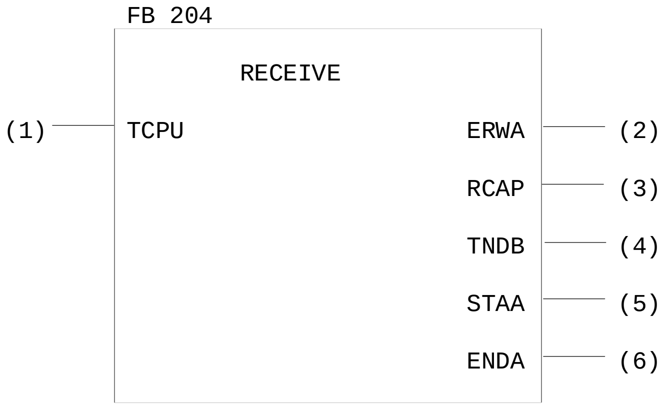
FB 203: Testing the transmitting capacity



Parameter name	Significance	Parameter type	Data type	Parameter field
RCPU	**R**eceiving **CPU**	I	BY	FY 246
ERRO	**Erro**r	Q	BY	FY 248
TCAP	**T**ransmitting **cap**acity	Q	BY	FY 249

```
FB 203                                                      LEN=30

SEGMENT 1     0000
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: Q     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
```

FB 204: Receiving a data field



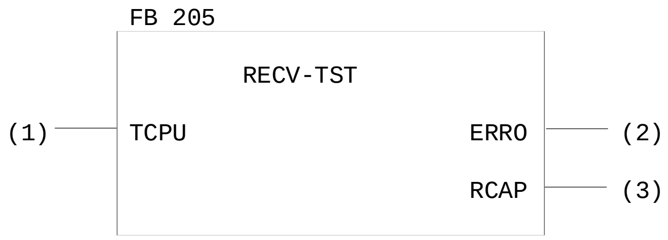
Parameter name	Significance	Parameter type	Data type	Parameter field
TCPU	**T**ransmitting **CPU**	I	BY	FY 246
ERWA	**Er**ror/**wa**rning	Q	BY	FY 248
RCAP	**R**eceiving **cap**acity	Q	BY	FY 249
TNDB	**T**ype (H byte) and **n**umber (L byte) of the destination **d**ata **b**lock	Q	W	FW 250
STAA	Address of the first received data word (**sta**rt **a**ddress)	Q	W	FW 252
ENDA	Address of the last received data word (**end a**ddress)	Q	W	FW 254

Continued on the next page

FB 204 continued:

```
FB 204                                                     LEN=45

SEGMENT 1      0000
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 data field"
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
```

FB 205: Testing the receiving capacity



Parameter name	Significance	Parameter type	Data type	Parameter field
TCPU	**T**ransmitting **CPU**	I	BY	FY 246
ERRO	**Erro**r	Q	BY	FY 248
RCAP	**R**eceiving **cap**acity	Q	BY	FY 249

```
FB 205 continued:

FB 205                                                     LEN=30

SEGMENT 1     0000
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
```

10.9.2 Transferring Data Blocks

In this example, the function block TRAN DAT (FB 110) transfers a selectable number of data fields from a data block in one CPU to the data block of the same type and same number in a different CPU. The FB number (FB 110) has been selected at random and you can use other numbers.

Programming FB 110 is described first followed by the application of FB 110.

Programming FB 110

```
FB 110: Transferring a data block

Task

The data area to be transferred is stipulated by the input parameter FIRB
(= number of the first data field to be transferred) and NUMB (= number of
data fields to be transferred). A data field normally consists of 32 data
words. Depending on the data block length, the last data field 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 data fields (according
to the NUMB parameter).

                                       Continued on the next page
```

FB 110 continued:

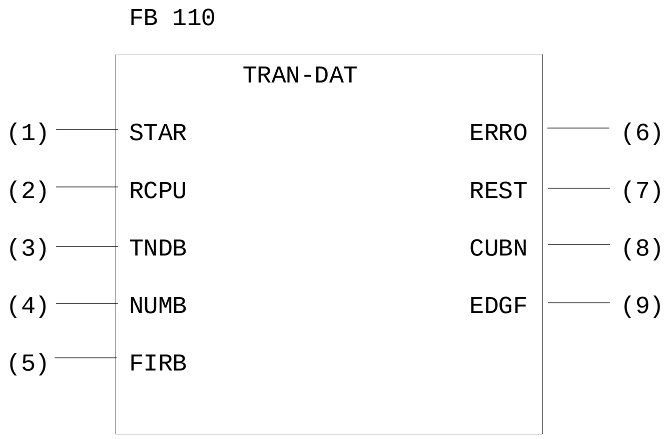
If, however, the REST output parameter has a value greater than zero, this means that the function block must be called again, for example in the next cycle. This means that you or the user program can only change the set parameters (i.e. the values of all parameters) when the REST parameter indicates 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 possibly is illustrated in the application example.

If the SEND function (OB 202) is not correctly executed with 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.

<u>Implementation</u>



Continued on the next page

FB 110 continued:

Parameter name	Significance	Parameter type	Data type
STAR	**Star**t the transfer of the data block on a positive-going edge	I	BI
RCPU	**R**eceiving **CPU**	I	BY
TNDB	**T**ype (H byte) and **n**umber (L byte) of the data block to be transferred.	I	W
NUMB	Number of data fields to be transferred.	I	BY
FIRB	Number of the first data field to be transferred.	I	BY
ERRO	**Erro**r	Q	BY
REST	Number of data fields still to be transferred.	Q	BY
CUBN [1]	**Cu**rrent **field n**umber	Q	BY
EDGF [1]	**Edg**e **f**lag	Q	BI

[1] Internal scratchpad flag, not intended for evaluation

```
FB 110                                                    LEN=89

SEGMENT 1     0000
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      :
```

Continued on the next page

FB 110 continued:

```
0025      :L    =REST              First send any remaining
0026      :L    KB 0               data fields
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 postive 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 to
0039      :!=F                     send data fields
003A      :JC   =GOOD
003B TRAN :L    =CUBN
003C      :T    FY 249
003D      :L    KB 246             SF OB:
003E      :JU   OB 202             "Send a data field"
003F      :L    FY 250
0040      :JM   =ERRO              Abort if error
0041      :JP   =GOOD              Abort if trans-cap. = 0
0042      :L    =CUBN              Increment
0043      :I    1                  field number
0044      :T    =CUBN
0045      :L    =REST              Decrement number of
0046      :D    1                  remaining data fields
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      :BE
004F      :
0050 ERRO :T    =ERRO              Program end if error:
0051      :L    KB 0
0052      :T    =REST              RLO = 1, ERRO contains error number
0053      :BE
```

Application of FB 110

Application of FB 110 on the S5-155U

Task

You want CPU 1 to transfer data blocks DB 3 (data fields 2 to 5) and DB 4 (data fields 1 to 3) to CPU 2 during the cyclic user program. The RECEIVE function (OB 204) is also called in the cyclic user program.

Implementation

Function	CPU 1	CPU 2
	called in:	called in:
Initialization (OB 200)	OB 20	–
Send organization (FB 1)	OB 1	–
Receive organization (FB 2)	–	OB 1
	exists:	exists:
Send DB	DB 3; DB 4	–
Receive DB	–	DB 3; DB 4

The user program in function block FB 1 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.

```
FB 1                                                      LEN=yy

SEGMENT 1      0000
NAME:S-ORG
0000      :L     KB 2             To CPU 2 ..
0001      :T     FY 0
0002      :L     KY 1,3           .. from data block DB 3
0003      :T     FW 1
0004      :L     KB 4             .. four data fields
0005      :T     FY 3
0006      :L     KB 2             .. send from 2nd data field
0007      :T     FY 4
0008      :
```

Continued on the next page

Application example continued:

```
0009      :JU    FB 110
000A NAME :TRAN-DAT
000B STAR :      I 2.0
000C RCPU :      FY 0
000D TNDB :      FW 1
000E NUMB :      FY 3
000F FIRB :      FY 4
0010 ERRO :      FY 5
0011 REST :      FY 6
0012 CUBN :      FY 7
0013 EDGF :      F 8.0
0014      :
0015      :
0016      :JC    =HALT             Abort after error
0017      :
0018      :L     KB 2              To CPU 2 ..
0019      :T     FY 10
001A      :L     KY 1,4            .. from data block DB 4
001B      :T     FW 11
001C      :L     KB 3              .. three data fields
001D      :T     FY 13
001E      :L     KB 1              .. send from 2nd data field
001F      :T     FY 14
0020      :
0021      :JU    FB 110
0023 NAME :TRAN-DAT
0024 STAR :      I 2.1
0025 RCPU :      FY 10
0026 TNDB :      FW 11
0027 NUMB :      FY 13
0028 FIRB :      FY 14
0029 ERRO :      FY 5
002A REST :      FY16
002B CUBN :      FY17
002C EDGF :      F 8.1
002D      :
002E      :
002F      :JC    =HALT             Abort after error
0030      :BEU
0031      :
0032 HALT :
0033      :                        The error handling takes place here
0034      :                        (e.g. stop, message output
0035      :                        on the printer, ...)
0036      :

00xx      :BE
```

Continued on the next page

Application example continued:

In CPU 2, the RECEIVE function (OB 204) called by FB 2 enters each transmitted data field into the appropriate data block. It may take several cycles before a data block has been completely received.

```
FB 2                                                  LEN=yy

SEGMENT 1     0000
NAME:RECV-DAT
0000      :L    KB 1            Receive data from CPU 1
0001      :T    FY 246
0002      :
0003 SCHL :L    KB 246          SF OB:
0004      :JU   OB 204          "Receive"
0005      :JM   =ERRO           Abort if error
0006      :L    FY 249          The RECEIVE function is
0007      :L    KB 0            called until there are no
0008      :><F                  further of data fields in
0009      :JC   =LOOP           the buffer, i.e. the
000A      :                     receiving capacity = 0.
000B      :BEU
000C ERRO :
000D      :                     The error handling takes place here
000E      :                     (e.g. stop, message output
000F      :                     on printer, ...)

00xx      :BE
```

10.9.3 Extending the IPC Flag Area

The problem

In the S5-135U/155U programmable controllers, 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.

The solution

Consecutive data words of a DB or DX data block are defined from DW 0 onwards as "IPC data words". Each link is assigned its own data block and is totally **in**dependent of the other links.

At the beginning of the cycle block, 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 links regardless of the other links:

- The transmitting CPU is only active when the receiving CPU has read out all the "old" data from the COR 923C buffer.
- The receiving CPU is only active when the transmitting CPU has written all the "new" data in the COR 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.**

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 link list (see the 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 data fields. Remember the following points:

- A complete data field consists of 32 data words.
- If the last data field 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 fields of data specified in the link 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 link list

Table 10-8 Link list for extending the IPC flag area

	SUB-LIST 1			SUB-LIST 2		
Link		DB type	DB number			No. of data fields
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	1 [1)]	10 [1)]	DW 22	**3**	2 [1)]
... 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}

[1)] Refer to the example on the following page

The link 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 link.

- **Number of data fields**

  The number of data fields specifies the size (= the number of data words) of the data word area to be transferred. (If links do not exist or you do not require them, enter 0 for the number of data fields, and for the DB type and DB number.)

- **DB type**

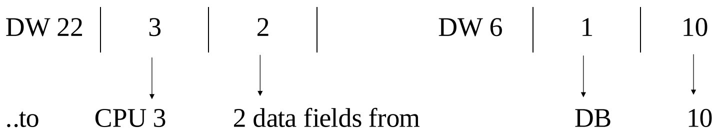
  Type of data block containing the data word area to be transferred.

- **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 data fields in data block DB 10 from CPU 2 (S2) to CPU 3, make the following entries:

CPU 2 (**S 2**) sends ..



DW 22 | 3 | 2 | DW 6 | 1 | 10

..to CPU 3 2 data fields from DB 10

Sub-list 2 is identical to the assignment ("manual" mode) required for the INITIALIZE function (OB 200). Within the data block, sub-list 2 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.

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 link as data fields to be transmitted on this link.

To send and receive data word areas, each CPU uses two function blocks:

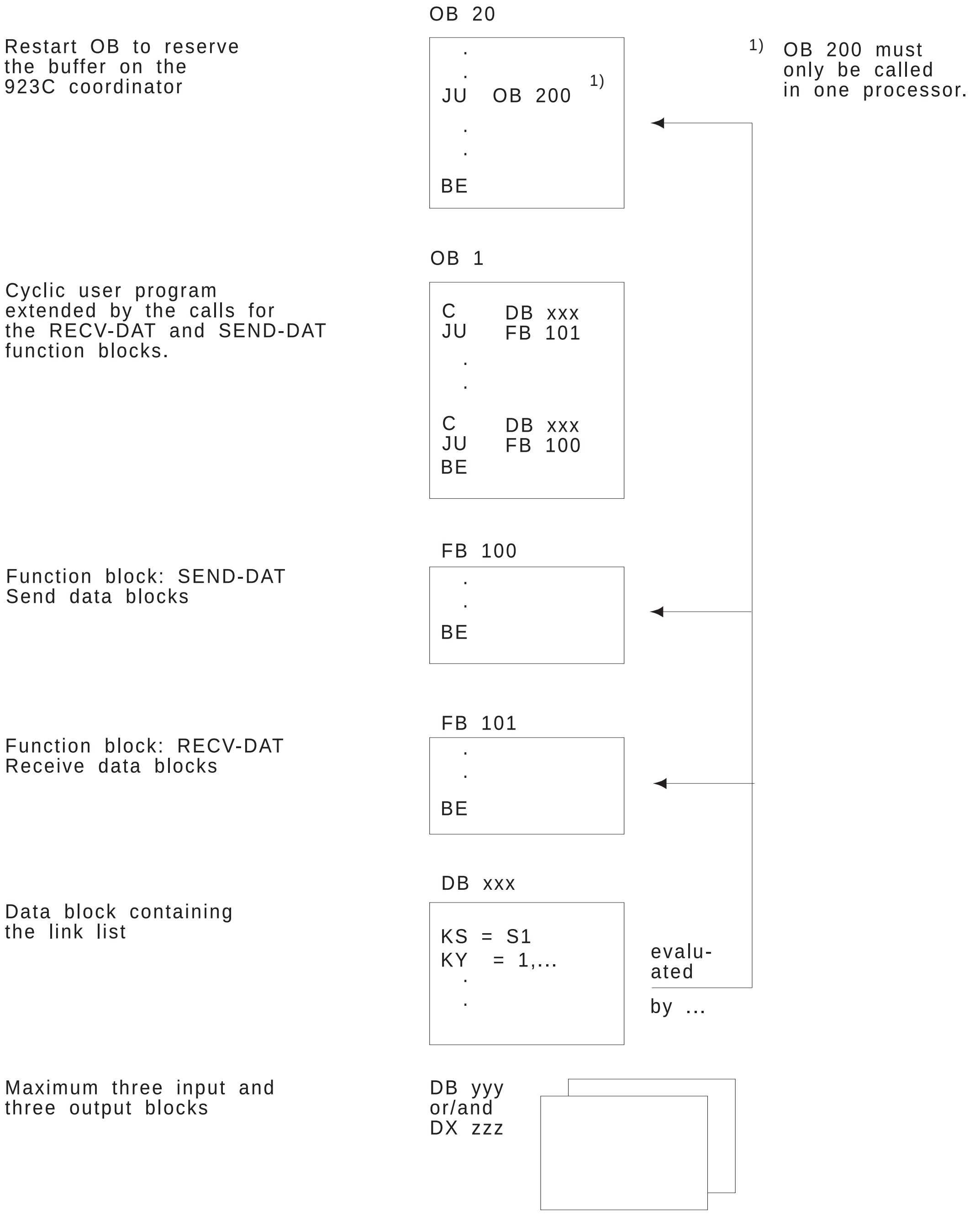
FB no.	Name	Function
FB 100	SEND-DAT	Send data word areas to the other CPUs
FB 101	RECV-DAT	Receive data word areas from the other CPUs

These FB numbers have been selected at random and you can use others.

The function blocks SEND-DAT and RECV-DAT read the link 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.

Note
This example (IPC flag extension using function blocks SEND-DAT and RECV-DAT) can only run correctly when the special function organization blocks for multiprocessor communication OB 202 to OB 205 are not called in any of the CPUs.

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.



Restart OB to reserve the buffer on the 923C coordinator

OB 20

```
  .
  .
JU   OB 200 1)
  .
  .
BE
```

Cyclic user program extended by the calls for the RECV-DAT and SEND-DAT function blocks.

OB 1

```
C      DB xxx
JU     FB 101
  .
  .
C      DB xxx
JU     FB 100
BE
```

Function block: SEND-DAT Send data blocks

FB 100

```
  .
  .
BE
```

Function block: RECV-DAT Receive data blocks

FB 101

```
  .
  .
BE
```

Data block containing the link list

DB xxx

```
KS = S1
KY  = 1,...
  .
  .
```

Maximum three input and three output blocks

DB yyy or/and DX zzz

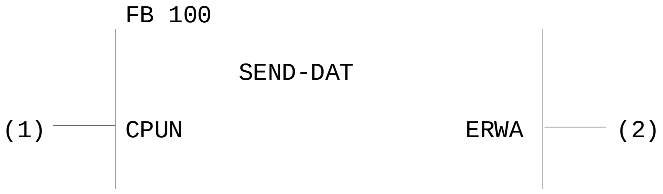
1) OB 200 must only be called in one processor.

Fig. 10-6 Overview of the blocks required in each CPU

Programming function blocks

FB 100: Sending data word areas

Before you call FB 100, the data block containing the link 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 link 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 no. 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	**N**umber of the **CPU** on which FB 100 is called. The numbers 1 to 4 are permitted.	D	KF
ERWA	**Er**ror/**wa**rning (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
```

Continued on the next page

FB 100 continued:

```
0013      :
0014      :SLW   2                      CPUN = CPUN * 4
0015      :T     FY 245                 Base address
0016      :
0017      :L     KB 1
0018      :T     FY 244                 Link 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      :B     FW 242
0029      :L     DL 0                   No. of the receiving CPU
002A      :T     FY 246
002B      :L     KB 246                 SF OB:
002C      :JU    OB 203                 "Test sending capacity"
002D      :L     FY 248                 Abort if error
002E      :JC    =OBER
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      :B     FY 240
0039      :L     DW 0                   Type and number of
003A      :T     FW 247                 the source DB
003B      :
003C TRAN :L     KB 246                 SF OB:
003D      :JU    OB 202                 Send a data field
003E      :L     FY 250                 Abort if error/warning
003F      :JC    =OBER
0040      :
0041      :L     FY 249                 Field no. = field no. + 1
0042      :I     1
0043      :T     FY 249                 All data fields transferred?
0044      :L     FY 239
0045      :<F
0046      :JC    =TRAN
0047      :
```

Continued on the next page

FB 100 continued:

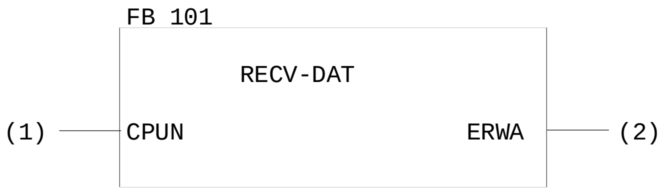
```
0048 EMPT :L     FY 244                        Increment
0049      :I     1                             link counter
004A      :T     FY 244
004B      :L     KB 4                          All  links
004C      :<F                                  processed ?
004D      :JM    =LOOP
004E      :L     KB 0                          Regular program end:
004F      :T     =ERWA                         RLO = 0, ERWA = 0
0050      :BEU
0051      :
0052 ERWA :L     KB 16                         Program end if error:
0053 OBER :T     =ERWA                         RLO = 1, ERWA contains
0054      :BE                                  error/warning number
```

FB 101: Receive data word areas

Before you call FB 101, the data block containing the link 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 link 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 no. 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	**N**umber of the **CPU**, on which FB 101 is called. The numbers 1 to 4 are permitted.	D	KF
ERWA	**Er**ror/**wa**rning (see RECEIVE function / OB 204)	Q	BY

Continued on the next page

FB 101 continued:

```
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                       Link 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
001B      :I     1                          the next entry for the
001C      :T     FW 244                     receiving CPU with the
001D      :DO    FW 244                     number ’CPUN’ 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                               pointer to sub-list 2.
002E      :SRW   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      :
```

Continued on the next page

FB 101 continued:

```
0037      :L     FY 249                       Receiving capacity = number
0038      :L     FY 243                       of reserved
0039      :><F                                memory fields ?
003A      :JC    =  EMPT
003B      :
003C RECV :L     KB 246                       SF OB:
003D      :JU    OB 204                       "Receive a data field"
003E      :L     FY 248
003F      :JM    =OBER                        Abort if error/warning
0040      :L     FY 249                       if receiving capacity = 0
0041      :L     KB 0                         process next
0042      :><F                                link
0043      :JC    =RECV
0044      :
0045 EMPT :L     FY 242                       Increment
0046      :I     1                            link counter
0047      :T     FY 242
0048      :L     KB 4                         All links
0049      :<F                                 processed ?
004A      :JM =  SRCH
004B      :L     KB 0                         Regular program end:
004C      :T     =ERWA                        RLO = 0, ERWA = 0
004D      :BEU
004E      :
004F ERWA :L     KB 16                        Program end if error:
0050 OBER :T     =ERWA                        RLO = 1, ERWA contains
0051      :BE                                 error/warning number
```

Application example

Application of FB 100/101 on the S5-155U

<u>Task</u>

You want to exchange data between three CPUs:

- **From CPU 1 to CPU 2:** data block DB 3, DW 0 to DW 127 (= 4 data fields)
- **From CPU 1 to CPU 3:** data block DX 4, DW 0 to DW 63 (= 2 data fields)
- **From CPU 2 to CPU 1 and CPU 3:** data block DB 5, DW 0 to DW 95 (= 3 data fields)

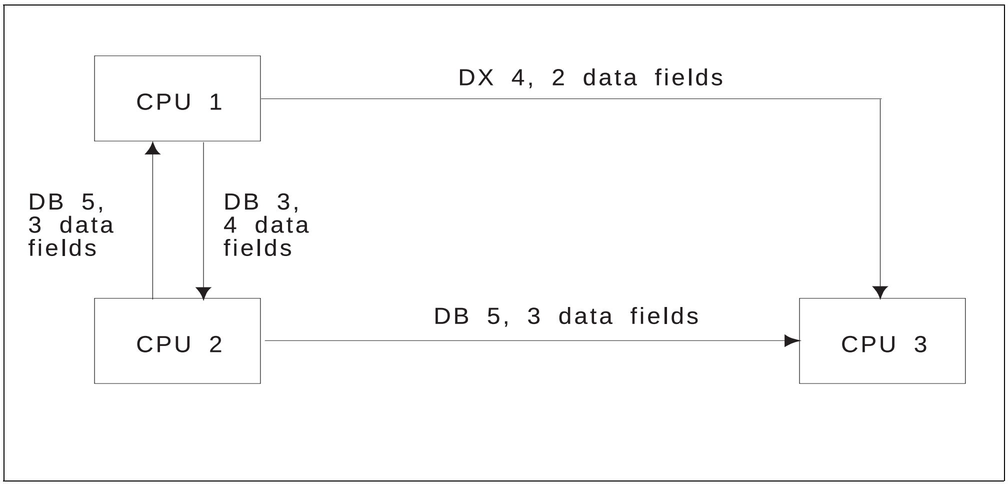


Fig. 10-7 Data exchange between 3 CPUs

Function block FB 1 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 link list is in data block DB 100.

Continued on the next page

Application example continued:

Implementation

1. Loading blocks

The following blocks must be loaded in the indivitual CPUs:

Function	CPU 1	CPU 2	CPU 3
Restart OB	OB 20	—	—
User program	FB 1	FB 1	FB 1
FB: SEND-DAT	FB 100	FB 100	FB 100
FB: RECV-DAT	FB 101	FB 101	FB 101
Link 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	—

2. Creating the link list

The link list is created and entered in data block DB 100:

```
DB100                                                LEN=37
                                                          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;
```

Continued on the next page

Application example continued:

```
– – Sub-list 1 – –

16:      KS = ’S1’;                      Send from CPU 1 to ..
17:      KY = 002,004;                   .. CPU 2 (four data fields)
18:      KY = 003,002;                   .. CPU 3 (two data fields)
19:      KY = 004,000;
20:      KS = S2’;                       Send from CPU 2 to ..
21:      KY = 001,003;                   .. CPU 1 (three data fields)
22:      KY = 003,003;                   .. CPU 3 (three data fields)
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;
```

Data words DW 16 to DW 31 contain the assignment list required for the manual INITIALIZATION function (OB 200).

3. Program OB 200 call in the start-up block OB 20 for CPU 1

OB 200 is called by the OB 20 shown below in CPU 1 during the restart.

```
OB 20                                                  LEN=yyABS

SEGMENT 1
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     :
0010     :                               The error handling routine
0011     :                               is inserted here if an
0012     :                               initialization clonflict
0013     :                               occurs (e.g. stop, output
0014     :                               message on printer, or ...)

00xx     :BE
```

Continued on the next page

Application example continued:

4. Program calls for the function blocks in FB 1 of the CPUs:

The user program on each CPU is extended by the RECV-DAT and SEND-DAT call. Function block FB 1 shown below is for CPU 1. For the other CPUs, the input parameter CPUN (CPU number) must be modified.

```
FB 1                                                       LAN=yy

SEGMENT 1     0000
NAME:EM-SE
0000
0000      :C    DB100                   Link list DB 100
0001      :JU   FB101                   Receive the input
0002      :                             data blocks
0003 NAME :RECV-DAT
0004 CPUN :     KF+1
0005 ERWA :     FY0
0006      :JC   =ERWA                   Abort if error/warning
0007      :
0008      :
0009      :                             Here, the cyclic user program that
000A      :                             reads data from the input data
000B      :                             blocks and enters data in the
000C      :                             output data blocks is inserted.
000D      :
000E      :
000F      :
0010      :C    DB 100                  Link list DB 100
0011      :JU   FB100                   Send the output
0012      :                             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 :                             Run an error handling routine
0019      :                             following an error/warning (here,
001A      :                             the error handling routine is
001B      :                             inserted, e.g. stop, output error
001C      :                             message on printer or screen, or ..)

00xx      :BE
```

PG Interfaces and Functions 11

Contents of Chapter 11

PG Interfaces and Functions 11

This chapter explains how to connect your PG to the CPU 948 and the functions provided by the PG software with which you can test your STEP 5 program.
If you only use the standard PG interface (1st serial PG interface) you do not need to read Sections 11.4 and 11.5. These sections tell you about further interfaces with which you can connect a PG to your CPU. These sections also contain points to note if you use PG functions on both interfaces.

11.1 Overview

You can load and test your user program using the online functions of the STEP 5 software.

To use these functions, the CPU must be connected to the PG. The following interfaces are available for this link:

- link via the serial standard interface "PG - PLC",
- link via the 2nd serial interface of the CPU 948,
- link via the S5 bus with SINEC H1.

The PG functions can operate simultaneously on the two serial interfaces, however, via the SINEC H1 link, the PG functions can only be used **alternately** with those on the serial interfaces

PG functions provide the following support for installing and testing your STEP 5 program:

Table 11-1 Functions for installation and testing

Function	**Section**
Info	
Size of the internal RAM and free user memory	"Memory configuration"
List of loaded blocks	"Output DIR"
Display contents of memory words/bytes and I/O bytes	"Output address"
Memory management	
Delete the whole memory	"Overall reset"
Create more memory space	"Compress memory
Manage blocks	"Transfer/delete blocks"
Program test	
Start/stop CPU	"Start/stop"
Test the operation sequence in a block	"Status block"
Test single program steps	"Program test"
Display signal state of process variables	"Status variables"
Output signals in the stop mode	"Force"
Display/change process variables	"Force variables"

11.2 PG Functions

> **Note**
> The terms used in this section for the PG functions may in some cases differ from the terms in your PG software. Please refer to your STEP 5 manual.

Calling and using functions

How to call and use the individual PG functions is described in the manual for your PG.

Checkpoint

The PG functions are performed at defined **checkpoints** in the CPU. In the CPU 948 there are four different checkpoints. Each of these checkpoints has certain test functions assigned to it.

Checkpoint "Stop"

You can access PG functions that are permissible only in the STOP mode at checkpoint "stop" (e.g. "start", "overall reset", "compress memory" in the stop mode).
The "stop" checkpoint is located immediately **before** OB 39 is called (the block for cyclic processing in the soft STOP mode).

Checkpoint "Cycle"

PG functions that you want to execute during cyclic program processing are called at checkpoint "cycle" (e.g. "compress memory" in the RUN mode, "stop", "status"). The "cycle" checkpoint is located immediately **before** the updating of the process image of the inputs (PII). At this point, the system program has not updated the PII yet.

Checkpoint "Test"

PG functions that you want to execute as soon as the next breakpoint is reached are called at checkpoint "test" (in the program test), see Section 11.2.3).

Checkpoint "General Functions"

This checkpoint exists both in the STOP and the RUN mode. Online PG functions that can be executed in all operating modes of the programmable controller are called at checkpoint "general functions". These functions include "transfer block", "delete block", "status variables". In the STOP mode, this checkpoint is located immediately **before** OB 39 is called (the block for cyclic processing in the soft STOP mode). During cyclic program processing, the checkpoint is located **after** updating the process image of the inputs and before calling OB 1.

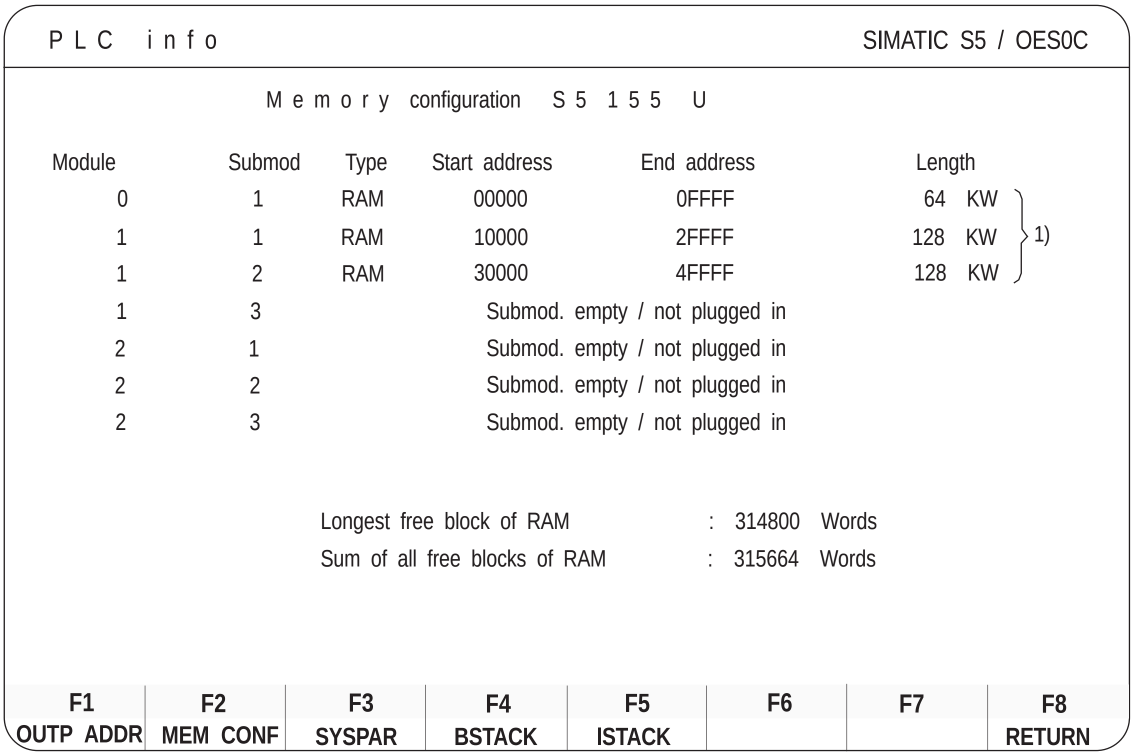
11.2.1 Info

Memory configuration

The CPU 948 is available with two memory versions and you can check the memory capacity using the PG function "memory configuration". With this function, the following information about the CPU user memory is transferred to the PG (from PG software version V6.0 upwards with "Delta diskette" for the CPU 948):

- memory capacity (640 Kbytes/1664 Kbytes)
- longest free block of user memory
- sum of all free blocks of user memory.

The PG software V6.3 with "Delta diskette" for the CPU 948 displays the memory configuration exactly.
With older PG software versions (e.g. V3.0 or MT1.0) the memory configuration is as for the CPU 946/947 (refer to Fig. 11-1). The total memory configuration of the CPU 948 must then be calculated from the total of the submodule values.



```
 PLC info                                                  SIMATIC S5 / OES0C

                       Memory configuration  S5 155 U

 Module          Submod   Type   Start address      End address          Length
      0             1     RAM        00000            0FFFF                64  KW  \
      1             1     RAM        10000            2FFFF               128  KW   > 1)
      1             2     RAM        30000            4FFFF               128  KW  /
      1             3                     Submod. empty / not plugged in
      2             1                     Submod. empty / not plugged in
      2             2                     Submod. empty / not plugged in
      2             3                     Submod. empty / not plugged in


                           Longest free block of RAM      :  314800  Words
                           Sum of all free blocks of RAM  :  315664  Words


     F1          F2          F3         F4         F5        F6       F7       F8
 OUTP ADDR    MEM CONF     SYSPAR     BSTACK     ISTACK                      RETURN
```

[1] corresponds to the memory configuration of the CPU 948-1: 320 Kw = 640 Kbytes

Fig. 11-1 PG display of the memory configuration

Output DIR

If you want to display a list of all the programmed blocks on the PG with the CPU 948, OB 0 is displayed instead of the system program blocks.

The function is permitted in the operating modes RUN, SOFT STOP, HARD STOP and can also be called within the "program test" function.

Output address

With the "output address" function, you can display the contents of memory and I/O addresses in hexadecimal format. You can access all addresses (RAM, S5 bus, areas with no modules assigned). In the process image area no ADF is triggered, in the I/O area there is no QVZ.

In the areas addressed as bytes (flags, process image) the high byte is represented as ’FF’.

11.2.2 Installation

Overall reset

With the function "delete all blocks" you can carry out an overall reset of the CPU from the PG. The overall reset is carried out unconditionally (refer to Section 4.2).

Compress memory

This function shifts all valid blocks in the user memory to the beginning of the user memory. Unused areas that resulted from deleting or correcting blocks are eliminated. This function shifts complete blocks to the beginning of the memory area. Ideally, one large free area results from many small unused areas. You can load blocks into the resulting large space.

You can call this function in the RUN and soft STOP modes. In the RUN mode, DBs and DXs that are longer than 512 data words are **not** shifted. In the STOP mode, **all** blocks are shifted.

The blocks are shifted via a buffer so that no data is lost if there is a power failure. If this buffer is insufficient for intermediate storage of a block, compressing continues at the next unused memory area. Consequently, some unused areas can still remain after compressing.

See also Section 8.3.

Transfer block

With this function you can transfer new or existing logic and data blocks to the user memory of the CPU.

If a block already exists in the user memory of the CPU, it is declared invalid and the new block becomes valid.

Delete block

With this function you declare a logic or data block in the user memory as invalid.

The space taken up by such blocks is released and can be used again.

11.2.3 Program Test

Start/stop

When you use the START and STOP PG functions, operating the PG corresponds to manual operation.

You can put the programmable controller into the STOP mode by calling the STOP function while the controller is in the RUN mode.

PGSTP is marked in the control bit display. In multiprocessor operation, the HALT control bit is set for the other CPUs.

You exit the SOFT STOP status with a COLD RESTART or WARM RESTART. In the single processor mode, the CPU exits the stop mode. In multiprocessor operation, the restart type is registered initially (the NEUDF or WIEDF control bit is set). However, the CPU stays in the soft STOP mode until all CPUs are initialized for multiprocessing. With the next operation "system start" you can start the programmable controller. This corresponds to switching the coordinator switch to RUN.

You can call the START PG function in the multiprocessor mode to select the restart type you want for all the CPUs you are using. After that, you can start the programmable controller with the last CPU.

Status block

You can call the "status" PG function to test related operational sequences (STEP 5 operations) in one block at any location in the user program.
The current signal status of operands, the accumulator contents, and the RLO are output on the PG screen for every executed operation in the block (i.e., step mode). You can also use this function to test the parameter assignment of function blocks (i.e., field operation):
The signal status of the actual operands is displayed.

Calling the function and specifying a breakpoint

When you call the "status" function on a PG and enter the type and number of the block you want to test (possibly including the nesting sequence and search key), you enter a breakpoint.

When the "status" function is called during program processing in the RUN mode, program processing continues until it reaches the operation marked by the specified breakpoint in the correct nesting sequence. Then the system program executes each of the monitored operations up to the operation boundary, outputting the processing results to the PG.

Note

The results of operation processing are not output in each of the subsequent cycles.

Nesting and interruptions

A sequence of operations marked by a breakpoint is completed even if a different program execution level (e.g., an error OB or interrupt OB) is activated and processed. With this you can see whether data has been changed by nested program sections.

If an interruption in a nested program execution level puts the CPU into the STOP mode, data is output up to the operation that was executed before the program execution levels changed. The data of the remaining operations is padded with zeros (the SAC is also 0).

The "status" function is possible in the following modes. RUN, RESTART (OB 20, OB 21, OB 22) soft STOP (OB 39 **only**).

DO FW/DO DW operations

While the "status block" function is active, if the cursor is positioned exactly on the operation following DO FW or DO DW, the message "Statement not processed" appears on the PG.

Remedy:
Avoid positioning the cursor on the operation following DO FW or DO DW.

Older PG software versions

If you move the cursor quickly in the "status block" function with older PG software versions, each cursor movement means a wait of 3 to 5 seconds.

Remedy:
Cancel the status with the abort key, reposition the cursor and then continue the status function again. There are then no waiting times.

Program test

You can call the "program test" function to test individual program steps anywhere in your user program. When you do this, you stop program processing and allow the CPU to process one operation after the other. The PG outputs the current signal status of operands, the accumulator contents, and the RLO for each operation executed.

Calling the program test and specifying the first breakpoint

You can call the "program test" function in the RUN and soft STOP modes. To call the function, specify the **type** and **number** of the block you want to test. You may also want to include the nesting sequence. At the PG, mark the first operation you want to test. This is how you specify the first breakpoint.

When you specify the first breakpoint during **program processing**, the CPU continues processing the program until it reaches the operation marked by the specified breakpoint. The operation is executed up to the operation boundary. (The DO FW and DO DW operations are processed **including** the substituted operation.) The CPU checks to see if the current block nesting sequence matches the block nesting sequence that you specified. If the nesting sequences do not match, the CPU continues program processing.

If program processing does not reach the specified breakpoint (e.g., because the CPU goes into the STOP mode or there is a continuous loop in the user program), the PG displays the message "Statement not processed". However, the function and the specified stopping point remain active.

If the nesting depths match, the output command is disabled (the "BASP" LED is on) and the PG displays the data of the processed operation. The CPU waits for further instructions from the PG.

DO FW/DO DW operations

When the "program test" function is active, the cursor cannot be moved beyond the operation following DO FW/DO DW.

Remedy:
Cancel the function, skip the sequence of operations mentioned above and set a new breakpoint after the operation following
DO FW/DO DW

Calling test functions in SOFT STOP

You can also call the "program test" function and specify an initial breakpoint when the CPU is in the soft STOP mode. The CPU remains in the soft STOP mode, and you can execute either a COLD RESTART or a MANUAL WARM RESTART. The CPU processes the program up to the marked operation and it proceeds as outlined above.

Executing the function and specifying another breakpoint

Initial situation: the CPU has processed the 1st break point.

To continue the function, you have two possibilities:

1. Specify the next operation as the **following** breakpoint:

   Move the cursor down to the next operation to specify the following breakpoint.
   The CPU continues by processing this operation up to the operation boundary. Then the CPU outputs the data and waits for further instructions from the PG. However, if a nested program execution level interrupts operation processing at the following breakpoint, the CPU processes the nested program first. Then the CPU returns to the 2nd breakpoint that you specified.

> **Note**
> You **cannot** specify a following breakpoint when the CPU is in the STOP mode.

2. Specify a **new** breakpoint:

   At the PG, specify any other operation in the same block or in a different block. The CPU continues program processing until it reaches the new breakpoint. The CPU processes the operation up to the operation boundary, then it outputs the data.

Cancelling the breakpoint

You can cancel the breakpoint by pressing the <BREAK> key if the CPU has not reached this breakpoint. After that you can specify a new breakpoint or terminate the program test.

Aborting the function

You can abort the "program test" function during program processing and when the CPU is in the soft STOP mode by calling the "program test end" function. The CPU goes into the STOP mode (or stays in STOP). The STOP-LED flashes slowly. BEARBE is marked in the control bits display. Afterwards a COLD RESTART is required.
The function is also aborted if an interface error occurs during the "program test" function (i.e., the cable between the PG and the programmable controller is disconnected).

Nesting with "interruptability at operation boundaries"

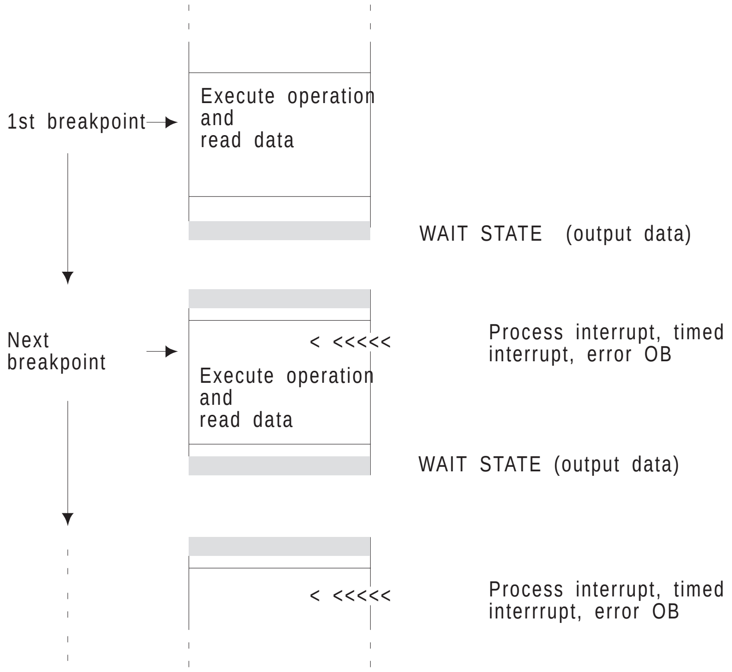
While the "program test" function is running, the other program execution levels can be activated, if the mode "interruptability at operation boundaries" is set.

When an operation has been processed at a breakpoint and a different program execution level is called at this point (e.g. an error OB or an interrupt OB) this is first processed completely before the program is continued at the next breakpoint.

Note

The system program reads data and outputs it at an operation boundary. At this point, all related program execution levels have not yet been processed.

The sequence of the "program test" function is illustrated in Fig. 11-2.



1st breakpoint
Execute operation and read data
WAIT STATE (output data)
Next breakpoint
< <<<<<
Process interrupt, timed interrupt, error OB
Execute operation and read data
WAIT STATE (output data)
< <<<<<
Process interrupt, timed interrrupt, error OB

Fig. 11-2 Sequence of "program test"

Note

If an operation has been processed at a breakpoint and activation of a different program execution level is requested, you can set a breakpoint at an operation in the different program execution level (e.g., you can look at a QVZ error OB directly after an operation that triggers a QVZ error).

Interruptions

- Program processing → STOP mode:
  If an interruption occurs during program processing (e.g., multi-processor stop, I/O not ready/STOP, error OB not programmed etc.) before the program reaches the specified breakpoint, the CPU goes into the STOP mode immediately. If you execute a COLD RESTART or a MANUAL WARM RESTART, the "program test" function is still in effect and the breakpoint is still set.

- Program processing at breakpoint → STOP mode:

  If stop conditions occur at the breakpoint or following breakpoint during program processing, the CPU goes directly into the soft STOP mode and outputs the data.
  If you do not specify a new breakpoint while the CPU is in the STOP mode, the "program test" function is still in effect after the restart.

While the "program test" function is in effect, you can execute the following other functions on your PG

- Output ISTACK
- Output BSTACK
- Load block
- Read block
- Delete block
- Output block list
- Force variables
- Force

In rare situations, the function may be terminated and the CPU is subsequently in the STOP mode.

Status variables

Using the "status variables" function, you can display the current signal states of certain operands (process variables).

When a checkpoint is reached, the PG displays the present signal status of the desired process variable. You can display the following process variables: inputs, outputs, flags, timers, counters and data words. No addressing error (ADF) is triggered in the process image area.

*Sequence in **RUN***

If the function is active in the RUN mode, the signal states of the operands are scanned and displayed when the checkpoint is reached. The system program reads inputs from the **process image**. As long as the function is not aborted, signal states are updated cyclically.

Note

If the program does not reach the system checkpoint, the system program does not output the signal states (e.g., in a continuous loop in the user program).

Sequence in
SOFT STOP

When the function is active in the STOP mode, the signal states of the operands are displayed when they exist at the system checkpoint. It is important to note that the inputs are scanned and output directly on the I/O module.

Force

You can call the "force" PG function to manually set the output bytes of the programmable controller to the signal states you want.

Note

The "force" function is only permitted in the **stop mode** (SOFT STOP mode or within the "program test" function).

Sequence of the function

When you call the "force" function in the STOP mode, the disable output command signal is suppressed (i.e., the BASP LED is off). **All** digital peripherals are cleared (i.e., the value 0 is written to each address). While the peripherals are being cleared, this function cannot be interrupted. If any timeout signals (QVZs) occur while the outputs are being cleared, they are ignored.

The peripheral outputs are forced byte by byte.

In multiprocessor operation, you can force **all** peripheral outputs (regardless of the peripheral assignment in DB 1).

Timeout errors that occur are detected when outputs are changed (PG message "I/O module does not exist").

Terminating the function

You can terminate the function by pressing the <BREAK> key on the PG. The disable output command signal is active again (i.e., the "BASP" LED is on). The function also ends if the CPU goes into the RUN mode between calling the function and actually forcing outputs.

Force variables

You can call the "force variables" function to look at the values of operands (process variables) in the process image table and change them. You can use this function in the RUN and soft STOP modes and within the "program test" function. You can display the following process variables: inputs, outputs, flags, timers, counters and data words.

Special features

Any change becomes effective at the next system checkpoint, i.e. regardless of the system checkpoint (start of cycle or end).

Note that the forced values can be overwritten again (e.g. by the user program or by the process image updating).

Note

The PG forces process variables in bytes.

If you are forcing **several operands**, the bytes are changed in memory one after the other distributed over several cycles.

11.3 Serial Link PG - PLC via 1st or 2nd Serial Interface

For the serial link PG - PLC there are the following possibilities:

- Direct link to the CPU via the standard cable.
- Link to the PG via the coordinator COR C. In this case the PG is connected via the cable to the coordinator. This means that the 1st serial interface is no longer available.
- Link to the PG via a PG multiplexer 757. The permitted cables can be found in the system manual 135U/155U /2/.
- Link to the PG via SINEC H1/L2/L1 and "swing cable"; the COR C or PG multiplexer can be connected in the link.

11.4 Parallel Operation of Two Serial PG Interfaces

You can use the second interface on the CPU 948 (SI 2) as a **PG interface** in exactly the same way as the first interface.

To be able to link your PG via this interface, you must also order the PG interface module in addition to your CPU 948 (the order number is listed in the system manual 135U/155U /2/).

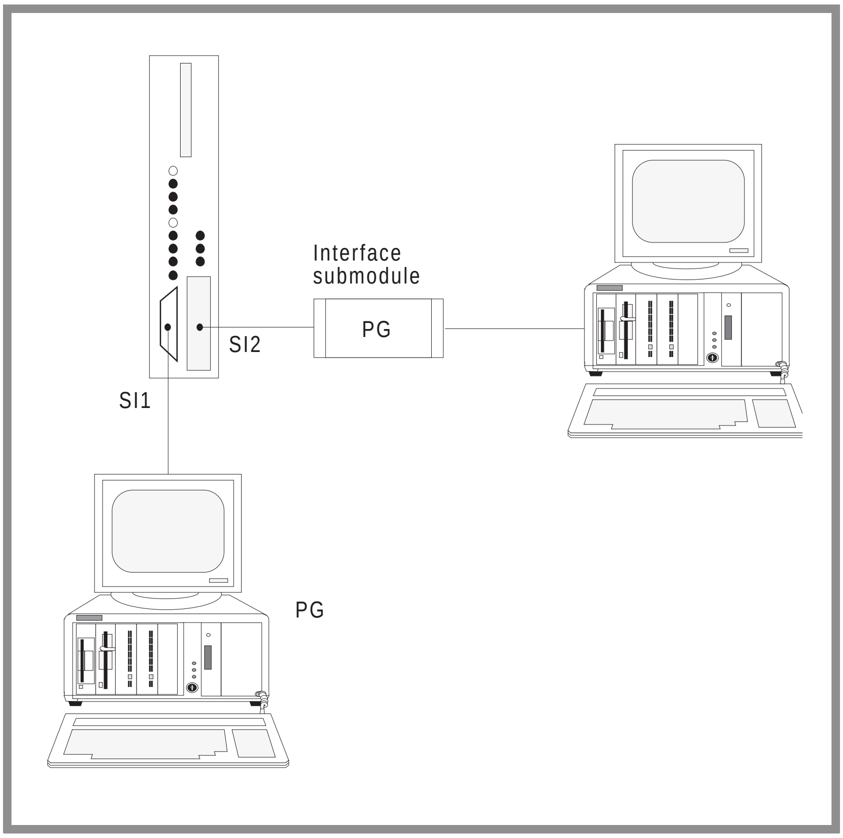


Fig. 11-3 Using the second interface as a PG interface

All the PG functions are available on both interfaces. The following sections contain only the information that you require if you work with PGs or OPs on both interfaces simultaneously.

Examples of configurations

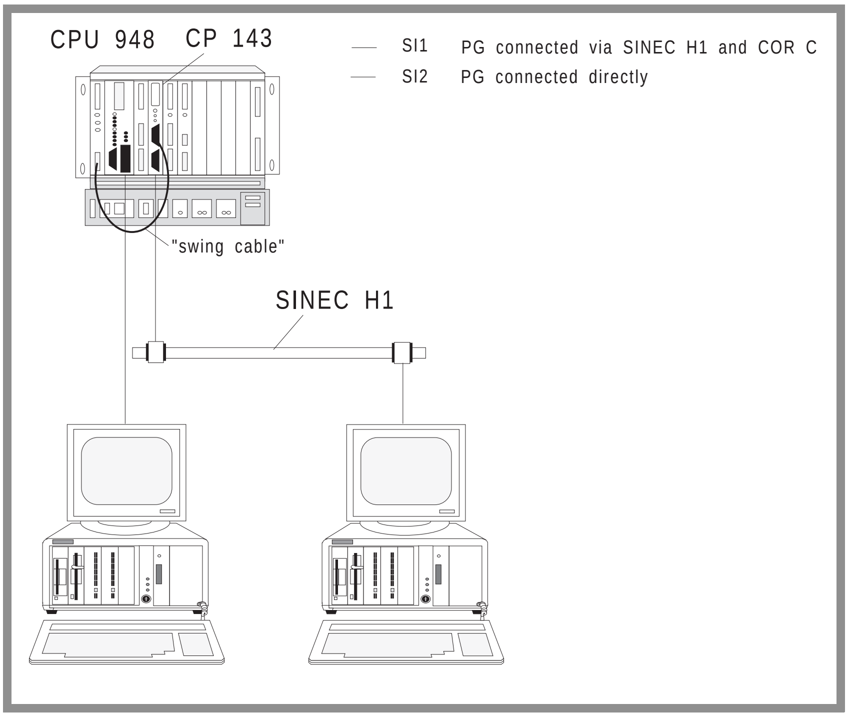

CPU 948
CP 143
SI1 PG connected via SINEC H1 and COR C
SI2 PG connected directly
"swing cable"
SINEC H1

Fig. 11-4 First example of a configuration

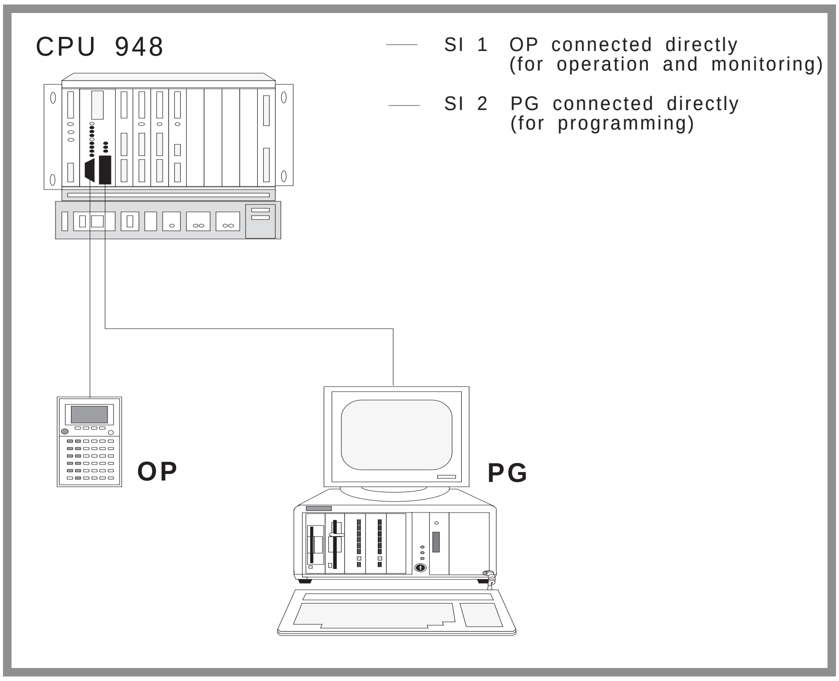

CPU 948
SI 1 OP connected directly (for operation and monitoring)
SI 2 PG connected directly (for programming)
OP
PG

Fig. 11-5 Second example of a configuration

11.4.1 Installation

To use the second interface of the CPU 948 as a PG interface, follow the steps outlined below:

Step	Action
1	Install the PG submodule in the CPU 948. (refer to the instructions in the Appendix)
2	Connect the PG to the serial interface SI2.

11.4.2 Operation

If you use the second interface as a PG interface then initially the full range of functions of the standard PG interface is available on each interface. This remains true, providing the individual functions do not influence each other, i.e., called sequentially one after the other.

To understand the exceptions to this, the PG functions can be divided into three groups:

Group	Name
Short-running functions	Functions that execute a job and then are terminated. (e.g. "transfer", "delete" etc.)
Long-running functions	Functions that process a fixed number of jobs: - "force", - "program test".
Cyclic functions	Functions that execute a job repeatedly until you terminate them: - "status block", - "status variables", - "force variables".

⚠ **Caution**

With long-running and cyclic functions you must coordinate the activation of these functions on both PGs.

The table below lists the pairs of functions that you **cannot work with simultaneously**.

Table 11-2 Functions which cannot run simultaneously on both PGs

Function active on the first PG:	**You *must not activate* this function on the second PG**
"Force"	Any function
"Program test"	Any function
A "status" function"	"Force"
A "status" function"	"Program test"
A "status" function"	"Overall reset"
"Status" on long running blocks or blocks which are not processed	Any function

If you attempt to start one of the illegal functions, the second PG displays an error message, e.g.: *"AS function disabled: function active"*.

The same error message or *"Overflow in data exchange with PG"* appears if the CPU 948 is currently processing functions of the other PG, which prevent your PG accessing the CPU within the monitoring time. Your input is then rejected. Repeat your input once the functions are completed on the other PG.

Note

Owing to the different performances and range of functions, time monitoring and the response to errors is not identical in all PGs and OPs.

If you activate the function "memory configuration" simultaneously on both PGs, the displays may be incorrect.

⚠

Caution

If you input, correct or delete blocks online on both PGs simultaneously, you must make sure that the blocks are not protected by the other PG before you access them.
"Status" of a block which is not processed or "status" in the STOP mode blocks the other interface for all functions.

11.4.3 Sequence in Certain Operating Situations

Parallel operation with short-running functions

If you work with PGs on both interfaces simultaneously, both PGs want to execute their functions independently of each other. As long as they stagger the jobs they send to the CPU, the jobs will be processed in the order in which they arrive.

The situation may, however, arise that the CPU 948 either receives two jobs simultaneously or receives a job from the second PG while a job from the first PG is still active.
Since simultaneous processing is not possible, the jobs are processed one after the other; the second job is, however, delayed by such a short time that it is hardly noticeable for the user.

When jobs are sent simultaneously, the sequence is as follows:

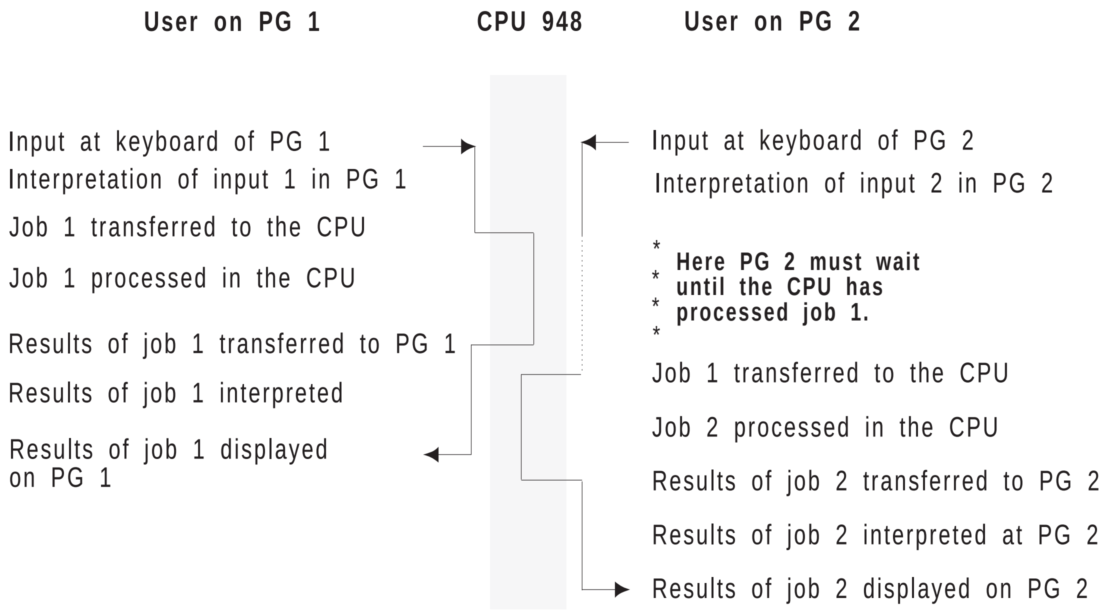

User on PG 1
CPU 948
User on PG 2
Input at keyboard of PG 1
Interpretation of input 1 in PG 1
Job 1 transferred to the CPU
Job 1 processed in the CPU
Results of job 1 transferred to PG 1
Results of job 1 interpreted
Results of job 1 displayed on PG 1
Input at keyboard of PG 2
Interpretation of input 2 in PG 2
*
* Here PG 2 must wait
* until the CPU has
* processed job 1.
*
Job 1 transferred to the CPU
Job 2 processed in the CPU
Results of job 2 transferred to PG 2
Results of job 2 interpreted at PG 2
Results of job 2 displayed on PG 2

Fig. 11-6 Handling simultaneous jobs

From this sequence, you can see that both PGs can operate independently from each other, but that the one nevertheless affects the other.
It is possible that both PGs process the same block simultaneously or that a block currently being processed by one PG is deleted by the other PG.
With this configuration, you must always take into account the way in which input at one PG affects the other PG.

Parallel operation with long-running functions

The long-running functions "force" and ""program test" cannot interrupt other functions and cannot be interrupted by other functions. They can therefore not be executed parallel to other functions, i.e. they are treated as a standard job "en bloc".

Parallel operation with cyclic functions

Cyclic functions can be executed both parallel to other cyclic and to short-running functions. The following example shows the standard sequence of the "status variables" function.

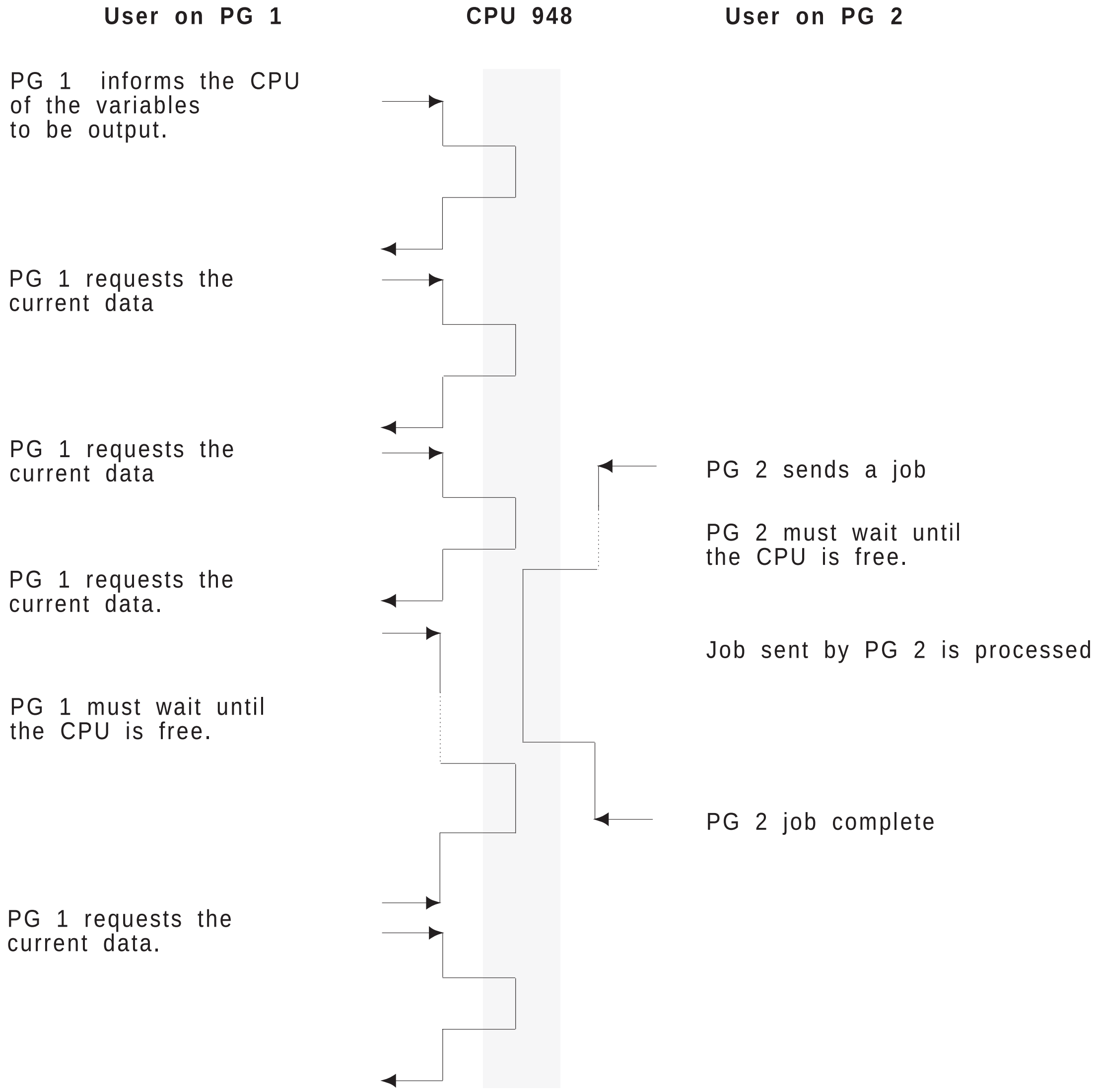

User on PG 1
CPU 948
User on PG 2
PG 1 informs the CPU of the variables to be output.
PG 1 requests the current data
PG 1 requests the current data
PG 2 sends a job
PG 2 must wait until the CPU is free.
PG 1 requests the current data.
Job sent by PG 2 is processed
PG 1 must wait until the CPU is free.
PG 2 job complete
PG 1 requests the current data.

Fig. 11-7 Typical sequence of a cyclic function and parallel short-running function

To allow a second PG to send a job to the CPU, the status function is interrupted between two requests and then continued on completion of the inserted job. Since the interrupting function requires CPU facilities, the whole CPU system facilities must be divided between the two functions, e.g. the updating of the data output by the "status variables" function takes somewhat longer.

With both PGs working simultaneously, the sequence shown in figure 11.8 results.

This also applies when cyclic functions are active on both PGs; the two PGs then access the PLC alternately.

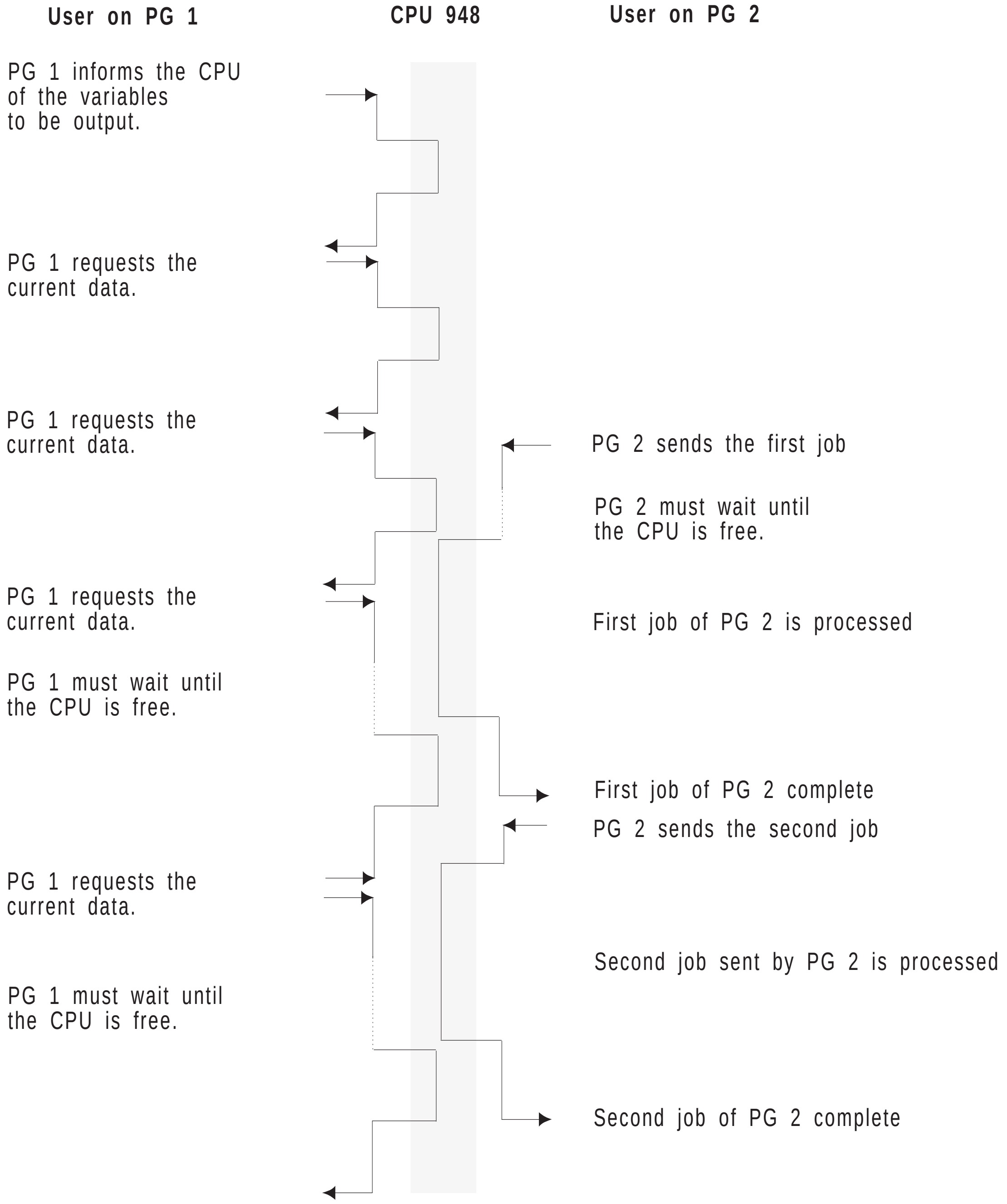
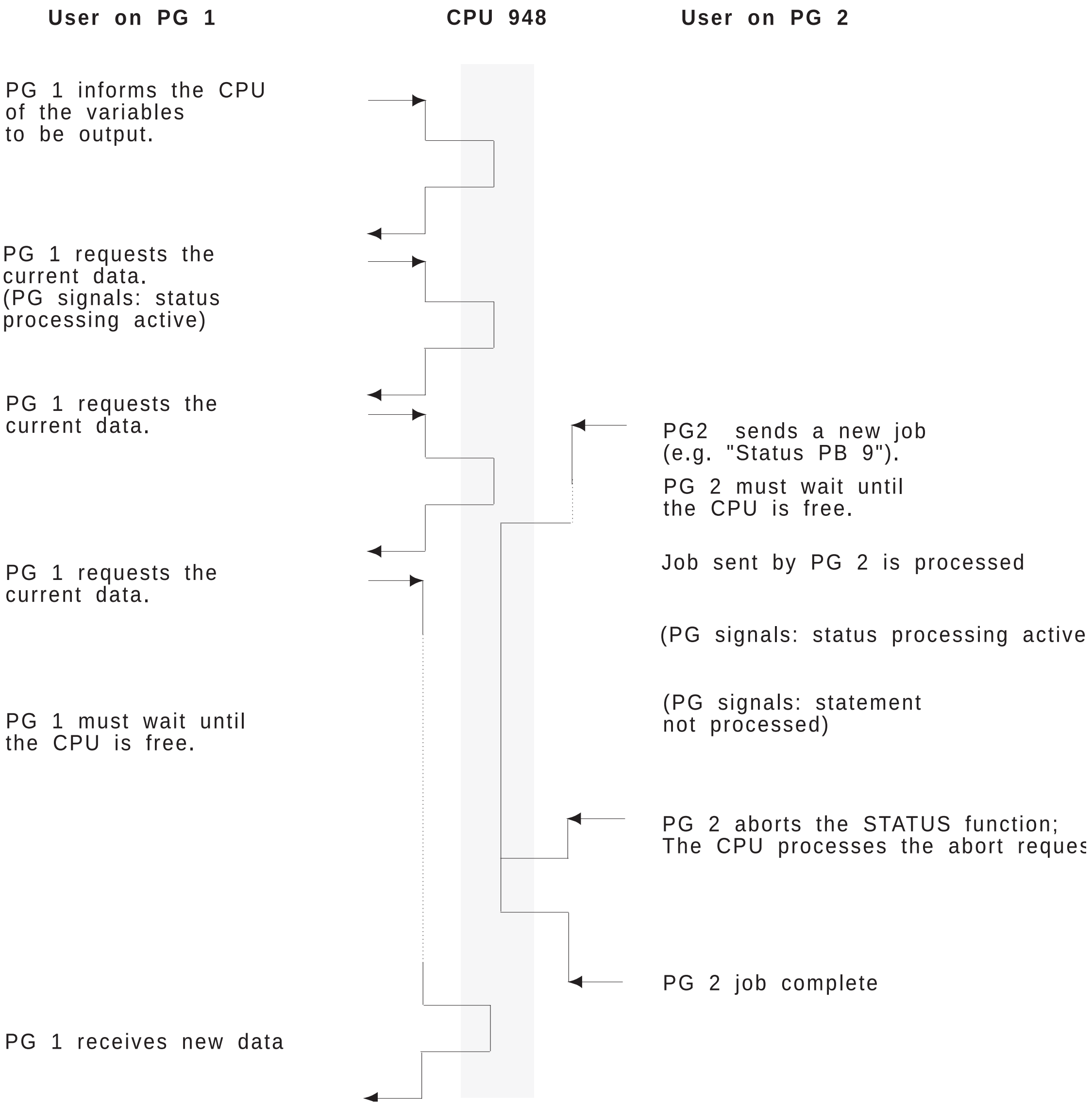

User on PG 1
CPU 948
User on PG 2
PG 1 informs the CPU of the variables to be output.
PG 1 requests the current data.
PG 1 requests the current data.
PG 2 sends the first job
PG 2 must wait until the CPU is free.
PG 1 requests the current data.
First job of PG 2 is processed
PG 1 must wait until the CPU is free.
First job of PG 2 complete
PG 2 sends the second job
PG 1 requests the current data.
Second job sent by PG 2 is processed
PG 1 must wait until the CPU is free.
Second job of PG 2 complete

Fig. 11-8 Sequence of two parallel cyclic functions

Special feature with cyclic functions on both PGs

If the interrupting function blocks the CPU 948 ("status" in a block that is not executed) the interrupted function is also blocked. It can only be resumed when the interrupting function is terminated.

When working simultaneously with two PGs, the following sequence results:



User on PG 1

CPU 948

User on PG 2

PG 1 informs the CPU of the variables to be output.

PG 1 requests the current data. (PG signals: status processing active)

PG 1 requests the current data.

PG2 sends a new job (e.g. "Status PB 9").

PG 2 must wait until the CPU is free.

Job sent by PG 2 is processed

PG 1 requests the current data.

(PG signals: status processing active

(PG signals: statement not processed)

PG 1 must wait until the CPU is free.

PG 2 aborts the STATUS function; The CPU processes the abort reques

PG 2 job complete

PG 1 receives new data

Fig. 11-9 Sequence when a function blocks the CPU 948

General notes

If "status variables", "force variables" (with the status display) or "status" is output on **one** interface and "compress memory", "delete block" or "transfer block" on the **other**, the status display can be corrupted.

11.5 PG Functions via the S5 Bus

11.5.1 Application

The PG functions via the S5 bus allow you to load and control S5-155U programmable controllers with the CPU 948 connected via SINEC H1 using the PG 7xx. With the PG functions via the S5 bus, the CPU 948 can be loaded up to eight times faster than via the PG interface. The actual speed depends on the length of the blocks to be transferred.

You can also use the PG functions via the S5 bus in the multiprocessor mode.

The PG functions via the S5 bus are a component of the system program of the CPU 948.

⚠

Caution

The PG functions via the S5 bus can only be used **alternately** with the PG functions via the first and second serial interface (i.e. one function only at a time).With some functions, (simultaneous/nested functions) data or blocks may be corrupted.

With the CPU 948 numbers ≥ 232 are reserved for the PG functions via the S5 bus. These numbers are not freely available for handling blocks.!

Fig 11-10 shows a typical configuration for the multiprocessor mode.

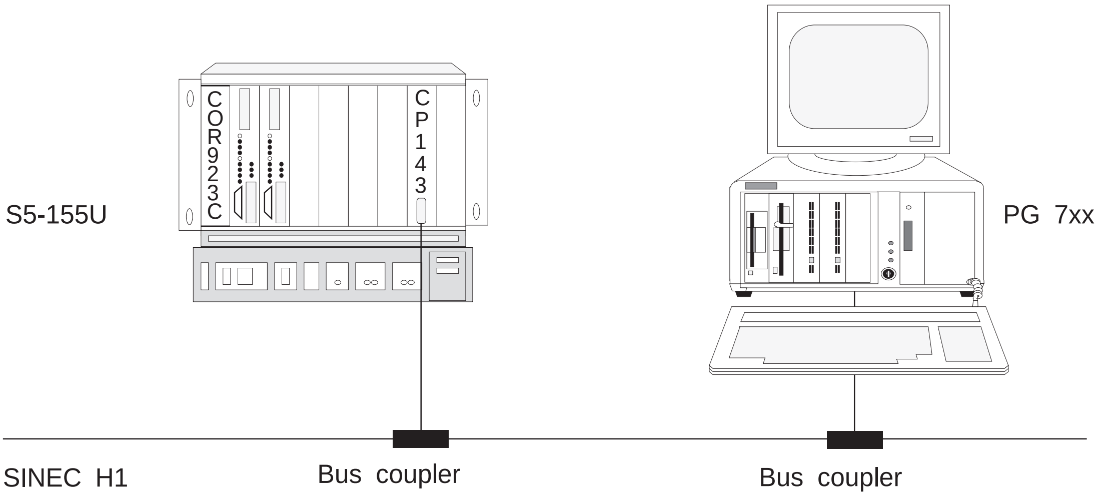


Fig. 11-10 **Multi**processor mode with a CP 143
(2 x CPU 948, 1 x CP 143)

No parameter assignment for the CPU

No parameter assignment is necessary on the **CPU 948** to use the PG functions via the S5 bus.

Technical requirements

The PG functions via the S5 bus with the CPU 948 can only be used when the PG and PLC are networked via SINEC H1.

You require the following:

- a PG 7xx with SINEC-H1 connection and with STEP 5 software version 6.3 (ST) or 6.0 (MT) installed with the delta diskette CPU 948
- in the PLC (central controller or expansion unit EU 185), a CP 143 communications processor from version 06 (firmware version 3.0) upwards with the base interface number 232 selected (the base interface number is set in the hardware using jumpers and in the SYSID using COM 143).

11.5.2 How the PG Functions Work via the S5 Bus

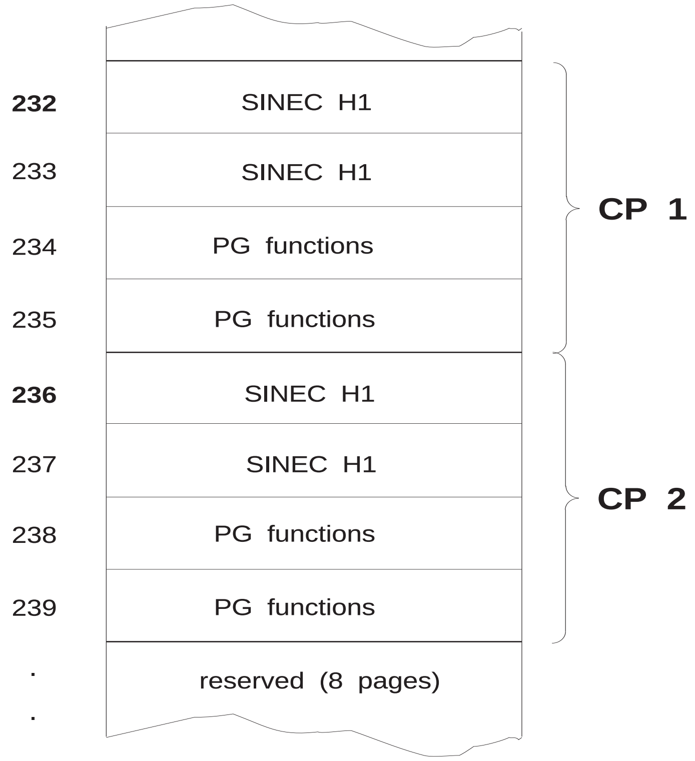
Using pages

For communication with the CPUs, the CP 143 has four pages (interfaces). If you do not use the PG functions via the S5 bus, all the pages are available for communication using handling blocks (HDBs).

When using the S5 bus functions, the pages of the CPU are divided into two pages for user HDBs and two pages for PG functions.
The pages for user HDBs can be used as previously for SINEC H1 applications. Remember, however, the special features listed in Section 11.5.3.
The pages for PG functions are used by the CP 143 and the CPU 948 for the PG functions via the S5 bus and are therefore no longer available for communication via handling blocks.

Interface numbers (SSNR)

The PG functions via the S5 bus are activated automatically in the CP 143 when you set the **base interface number** of the CP to **232 or 236** (jumpers **and** SYSID). You then occupy interface numbers 232 to 239. Interface numbers 240 to 247 are intended for later expansions (e.g. CPs with eight pages/interfaces).



232	SINEC H1	CP 1
233	SINEC H1	
234	PG functions	
235	PG functions	
236	SINEC H1	CP 2
237	SINEC H1	
238	PG functions	
239	PG functions	
. .	reserved (8 pages)	

Fig. 11-11 Interface assignment of the PG functions via the S5 bus

Parameters for the CP 143

Assigning parameters for the CP 143 is described in the CP 143 manual (Further Reading /6/).

⚠

Caution

The interface numbers 232ff and 236ff must not be assigned on the CP 143 when operating with other SIMATIC CPUs. When operating the CPU 948 with other CPs, the use of interface numbers 232 to 247 is restricted.

Multiprocessor mode

The PG functions via the S5 bus can also be used in the multiprocessor mode with the CPU 948.

With **one** CP 143, **two CPUs** (948) can use the online functions in the S5-155U. The CP 143 can also be used in the expansion unit (EU 185). In the multiprocessor mode, **CPU 1 uses the page with SSNR 234**, and **CPU 2 the page with SSNR 235**.

If a second CP 143 is inserted and has appropriate parameters assigned, this is reserved for the online functions via the S5 bus with CPU 3 and CPU 4 (with SSNR 238 and 239).

11.5.3 Installation and Getting Started

During installation, remember the following alternatives.

The CP 143 is used exclusively for PG Functions

If the CP 143 is used exclusively for PG functions via the S5 bus, no further parameters other than the SINEC H1 parameters must be set.

After POWER UP, the PG functions are always available on CPU 948 via the S5 bus without the CP 143 previously being synchronized with the HDB SYNCHRON (FB 125). The mode selector on the CPU must, however, be set to RUN.
An "empty" CPU 948 can be started up via the S5 bus without an OVERALL RESET.

After POWER UP, the CPU 948 automatically synchronizes the pages assigned to it on the CP 143 for PG functions via the S5 bus.

The following steps are necessary for starting up:

Step	Action	
1	**Set the interface number (SSNR) on the CP 143 (jumpers):** Select the SSNR according to the existing hardware configuration as shown below. Keep in mind the explanations in Further Reading /6/.	
	Possible hardware configuration	Corresponding SSNR on the CP 143
	1 x CPU 948, 1 x CP 143	232
	1 x CPU 948, 2 x CP 143	232 on 1st CP, 236 on 2nd CP
	2 x CPU 948, 1 x CP 143	232
	3 x CPU 948, 2 x CP 143	232 on 1st CP, 236 on 2nd CP
	4 x CPU 948, 2 x CP 143	232 on 1st CP, 236 on 2nd CP
2	**Insert the CP 143 in the S5-155U (power supply to the PLC must be off).**	
3	Connect the PG to the PG interface of the CP 143 and load the COM program.	
4	**Set the interface number selected in step 1 in the SYSID of the CP 143 using COM 143 and set the Ethernet address.**	

Step	Action
5	**Load the parameter data on the CP 143:** The parameter data of the CP 143 can either be stored - in an EPROM cartridge or - in the RAM of the CP 143. You can transfer the parameter data via the serial interface of the PG 7xx. The operations necessary for loading the parameter data of the CP 143 are described in /6/.
6	**Perform an overall reset on the CPU, switch the power supply to the PLC off and on again.**
7	**Edit the path to the CPU 948 in the bus selection screen form of STEP 5.**
8	**Select the path to the CPU 948 via SINEC H1/CP 143 in the presets screen form of STEP 5.**

Once these actions have been performed, the PG functions can be used via the S5 bus. You can now load your user program and run or test it.

Alternative operation via the serial PG interface

At any one time, the CPU 948 only processes one PG function. If you attempt to activate further PG functions on a second PG via the serial PG interface while a PG function is already being processed, a message, for example, "AS function disabled: function active" is displayed on this PG.

Notes

If a PG function is aborted by switching the mode selector on the CPU from RUN to STOP or by an error, wait times of greater than 15 seconds are activated for communication via SINEC H1.

If you make a mistake on the PG (e.g. switching off the PG with a PG function still active) it is possible that the "selection" function must be repeated when the path is re-established.

The CP 143 is used for PG Functions and Communication via SINEC H1

If you use the CP 143 for communication via SINEC H1 as well as for PG functions via the S5 bus, you must make further settings in addition to those described in Section 3.1 and must take certain special features into account.

During installation follow the procedure outlined below:

Step	Action
1 to 8	Steps 1 to 8 are identical to those described for the alternative "The CP 143 is used exclusively for PG functions"
9	**Program the HDB SYNCHRON (FB 125) call in the start-up OBs OB 20 and OB 22 so that the CP 143 is synchronized for SINEC H1 communication during MANUAL/AUTOMATIC COLD RESTART and AUTOMATIC WARM RESTART.** The HDB SYNCHRON should only be called when the interface is actually used, since the connection to the PG is subsequently terminated and must then be re-established manually on the PG.

Using pages for communication via handling blocks

The following diagram illustrates how the PG functions via the S5 bus use the pages of the CP 143. The free pages for user HDBs can be used by CPUs 1 to 4 for communication via SINEC H1.

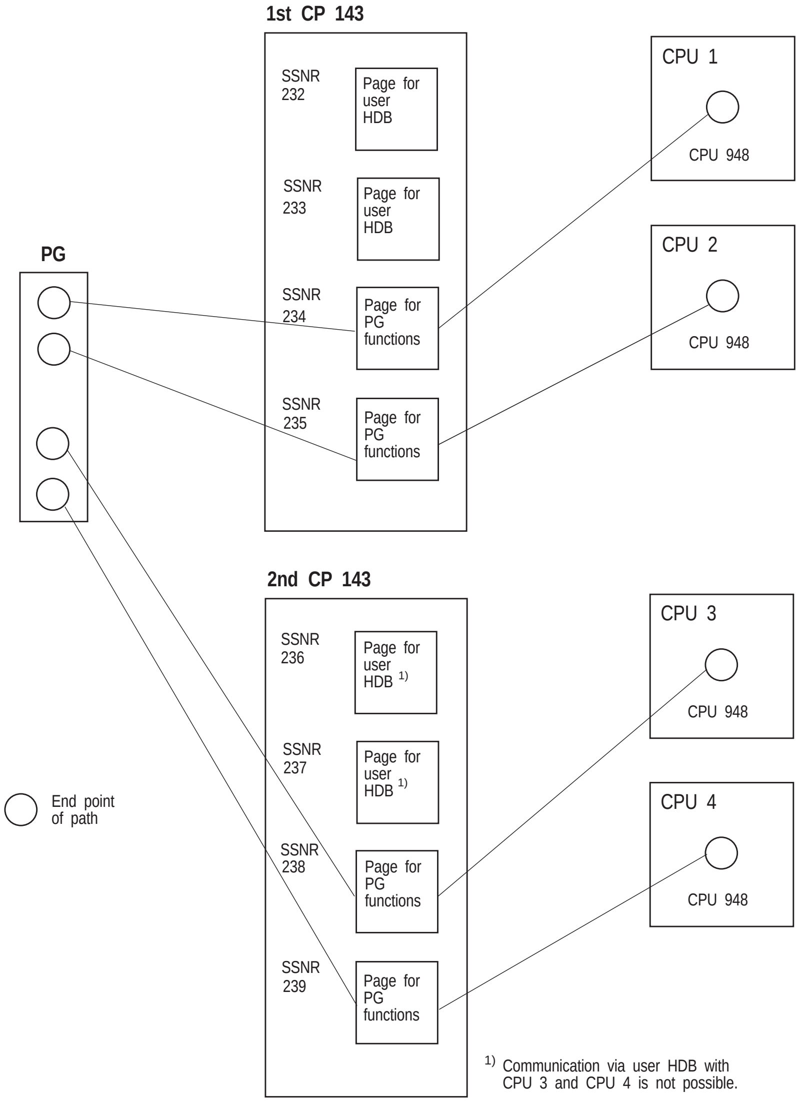


Fig. 11-12 Paths between the PG and CPU 948 and assignment of the CP 143 pages

Special features when communicating via pages for user HDBs

By synchronizing the CP 143 for communication (FB 125 called with SSNR 232, 233 or 236, 237) the existing connections are terminated by the CP 143
The paths must then be re-established. Waiting times then occur on the PG (even if you press the abort key). The path selection must be repeated on the PG.

Owing to this response, you cannot use the PG function "**status block**" via the S5 bus for the **start-up OBs** if you use pages for user HDBs for communication. You should therefore call FB 125 (HDB SYNCHRON) only in a COLD RESTART or in a restart following POWER UP (COLD RESTART or WARM RESTART).

11.5.4 Condition Codes Indicating Problems

Each of the maximum four CPUs (CPU 948), for which the PG functions via the S5 bus are activated, writes condition codes to its RS and RT areas if an error occurs in the PG functions via the S5 bus.

These condition codes consist of a parameter assignment error byte (PAFE) for each possible connection and a condition code word (ANZW) to indicate the current sequence of the transmit and receive blocks. These codes largely correspond to those of the handling blocks.

RS 50

PAFE codes set during the synchronization of the PG functions are stored in system data word RS 50 (address E F032H).

Evaluating PAFE in RS 50

The PAFE byte is always in the high byte of RS 50.

CPU no.	RS 50 high byte	RS 50 low byte
1	PAFE SSNR 234	-
2	PAFE SSNR 235	-
3	PAFE SSNR 238	-
4	PAFE SSNR 239	-

Significance of the PAFE codes

All errors are indicated which occur in the interaction with the CP 143. The following PAFE codes are then set:

PAFE value	Significance
00H	No error
71H	Interface (page) does not exist
81H	Interface not ready
91H	Interface overloaded
A1H	Interface being used by a different CPU
B1H	Job number or field size illegal (FB SYNCHRON)
C1H	Interface not responding or not in time
D1H	Other interface errors, also error code for the CP

Meaning of the code 71H:

The code 71H means that the page does not exist. If this error occurs, the PG functions cannot be used via the S5 bus. In this case, check the interface assignment of the CP 143. Interface numbers 232ff or 236ff must be set (jumpers **and** SYSID!).

RT area

If the pages for PG functions exist and the connection to the CP 143 is established, an information field consisting of 16 words is set up in the RT area of the CPU 948 with the structure shown below.

Note

As long as there is no connection to the CP 143 (PAFE = 71), e.g. because there is no page with SSNR 232ff or 236ff, **no additional information** is stored in the RT area.

Address			
E F2E8H	Data sent from CPU to PG for interface 234 or 238		RT 232
E F2E9H	Data sent from CPU to PG for interface 235 or 239		RT 233
E F2EAH	reserved		RT 234
E F2EBH	reserved		RT 235
E F2ECH	Data sent from PG to CPU for interface 234 or 238		RT 236
E F2EDH	Data sent from PG to CPU for interface 235 or 239		RT 237
E F2EEH	reserved		RT 238
E F2EFH	reserved		RT 239
E F2F0H	reserved	PAFE 234	RT 240
E F2F1H	reserved	reserved	RT 241
E F2F2H	reserved	PAFE 235	RT 242
E F2F3H	reserved	reserved	RT 243
E F2F4H	reserved	PAFE 238	RT 244
E F2F5H	reserved	reserved	RT 245
E F2F6H	reserved	PAFE 239	RT 246
E F2F7H	reserved	reserved	RT 247

Note

The RT area is **reset** during an OVERALL RESET.

If you use the PG functions via the S5 bus, the RT area is occupied as described above and is then no longer available for other programs (e.g. standard FBs). You should bear this in mind when planning your system.

ANZW

ANZW contains the current status of the send and receive blocks. The individual bits of an ANZW have the following significance:

High byte	
Bit no.	Assignment
15	Not used
14	
13	
12	
11	
10	
9	
8	
Low byte	
7	Not used
6	Data acceptance complete
5	Data transfer complete
4	1: error
3	Job complete with error
2	Job complete without error
1	0: SEND enabled [1] 1: SEND disabled
0	0: RECEIVE disabled [1] 1: RECEIVE enabled

[1] Specifically for PG functions via the S5 bus

Appendix 12

Contents of Chapter 12

Appendix 12

This chapter provides additional information about the CPU 948 such as jumper settings for system interrupts, notes on inserting and removing the PG submodule, comparisons of runtimes with CPU 946/947 and CPU 928B, and results IDs of some of the special function OBs .

Appendix 1: Jumper Settings for System Interrupts

For interrupt-controlled program execution with the CPU 948, there are four system interrupts available, as follows:

- INT A/B/C/D (dependent on the CPU slot, see System Manual /2/,
- INT E,
- INT F
  and
- INT G.

The interrupts you want to use must be enabled using jumpers. The jumpers are located on the basic board above the receptacle for the memory card. The exact position can be seen in the following diagram:

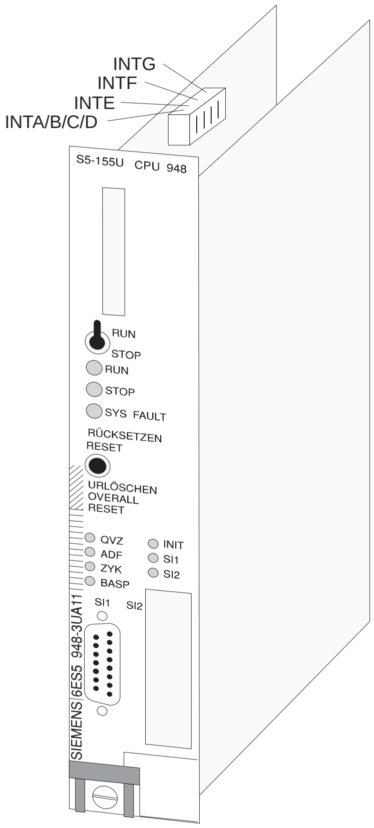


Fig. 12-1 Location of the jumper

Appendix 2: Inserting and Removing the PG Submodule

If you want to use a PG submodule, this must first be added to the CPU (before the CPU is installed in the central controller).

Caution
Switch off the power supply to the programmable controller before you remove the CPU.

Inserting the submodule

Note
The jumpers on the PG submodule are already correctly set when supplied. If, following installation you encounter difficulties, compare the jumper setting with the settings shown in the System Manual /2/.

Insert your PG submodule as follows:

Step	Action
1	Switch off the power supply to your PLC.
2	Remove the CPU from the central controller.
3	Undo the two screws securing the cover of the submodule receptacle on the CPU and remove the cover.
4	Insert the PG submodule through the front panel into the connector (components in the same direction as those of the CPU).
5	Secure the submodule with the two screws previously used for the cover.
6	Insert the CPU in the central controller.
7	Switch on the power supply to your PLC again.

Removing the submodule

You remove the PG submodule as follows:

Step	Action
1	Switch off the power supply to your PLC.
2	Remove the CPU from the central controller.
3	Undo the two screws securing the submodule and remove the submodule from the receptacle.
4	Insert a further submodule (as described above) or close the submodule receptacle with the cover. Use the same screws used to secure the submodule.
6	Insert the CPU in the central controller.
7	Switch the power supply to your PLC on again.

Note
Screwing the interface submodule to the CPU diverts disturbance pulses via the screen of the CPU. The CPU must only be operated with the submodule receptacle closed (cover or submodule).

Appendix 3: Technical Data of the CPU 948 and CPU 928B

Operation / Processing	**CPU 948**	**CPU 928B**
Typical command execution times for bit commands:		
with F, I, Q D Formal operand	0.18 µs 0.7 µs 0.91 µs	0.57 µs 3.4 µs 2.4 µs
Typical command execution times for word commands:		
- Load operations L FY (byte) L FW (word) L FD (double word)	0.18 µs 0.5 µs 0.71 µs	0.81 µs 0.9 µs 1.6 µs
- Fixed point arithmetic - Floating point arithmetic	0.55 ... 3.8 µs 3.3 ... 6.3 µs	0.9 ... 10.4 µs 9.1 ... 15.6 µs
Cyclic program execution (single processor mode)		
Basic overhead calling OB 1/FB 0:	65/– µs	104/106 µs
Extra time for process image updating depending on the number of I/O bytes (n) where 0 < n ≤128	n ≤ 64: 64 µs + n * 2.3 µs n > 64: 92 µs + n * 2.3 µs	I: 14 µs + n * 1.1 µs Q: 5 µs + n * 4.1 µs
Extra time for IPC flag transfer depending on the number of IPC flags (n) where 0 < n ≤ 256	n ≤ 64: 64 µs + n * 2.1 µs n > 64: 92 µs + n * 2.1 µs	I: 14 µs + n * 1.4 µs Q: 5 µs + n * 4.3 µs
Extra time for timer processing depending on the timer block length Timer block length (TBL) = 0	every 10 ms 11.6 µs	every 10 ms 10 µs
timer block length #0 n = number of timers running (steps: 10 ms)	11.6 µs + TBL* 0.32 µs	16 µs + TBL * 0.2 µs (no difference between running and stopped timers)
Interrupt driven program execution		
Cycle time extension from nesting an empty OB 2 (without STEP 5 operations) at a block boundary	262 µs	300 µs
Reaction time	175 µs	270 µs

Operation / Processing	CPU 948	CPU 928B
Time-driven program execution		
Cycle time extension from nesting an empty OB 13 (without STEP 5 operations) at a block boundary	287 µs	310 µs for the 1st timed int. OB 170 µs for each further timed int. OB due at the same time
Clock rate for time-driven program (timed interrupts OB 10 to OB 18)	Variable basic clock rate from 1 to 255 ms. Spec. in steps of 10 ms: 10, 20, 50, 100 200, 500 ms, 1, 2, 5 s or 10, 20, 40, 80, 160, 320, 640 ms, 1.28, 2.56 s	10, 20, 50, 100, 200, 500 ms, 1, 2, 5 sec
Resolution for clock-driven timed interrupts (OB 9)	every minute, hourly, daily, weekly, monthly, yearly, once	every minute, hourly, daily, weekly, monthly, yearly, once
Resolution for delayed interrupt (OB 6)	1 ms	1 ms
Cycle time monitoring		
Default selectable between triggerable	200 ms 1 to 2550 ms yes	150 ms 1 to 13000 ms yes
Memory sizes		
Size of the user memory module (in Kbytes)	640 or 1664	64
Size of memory for data blocks (DB-RAM, in Kbytes)	–	approx. 46.6
Timers, counters and flags		
Number of timers and counters	256 of each	256 of each
Number of flags	2048 flags + 32768 S flags	2048 flags + 8192 S flags

Appendix 4: Results IDs of some of the Special Function OBs in ACCU 1

Byte IDs in ACCU-1-LL

Some of the byte IDs are used by several special function OBs. Their significance therefore depends on the OB called.

SF-OB	ACCU-1-LL	Meaning
OB 124	01H	Function processed correctly
		Errors:
	45H	Block type not permitted
	47H	Block does not exist
	4DH	Online function COMPRESS MEMORY ACTIVE
		Warning
	8DH	Conflict with an online function (except "compress memory")
	8EH	10-ms waiting time not yet elapsed
OB 125	01H	Function processed correctly
		Errors:
	42H	Block already exists
	43H	Not enough memory
	44H	Block length not permitted
	45H	Block type not permitted
	4DH	Online function COMPRESS MEMORY active
		Warnings:
	8DH	Conflict with an online function (except "compress memory")
	8EH	10-ms waiting time not yet elapsed
OB 126	01H	Function processed correctly
		Errors:
	02H	Function no. illegal
	03H	Pointer in ACCU-1-L (flag no.) illegal
	04H	Block type/number illegal or block DB/DX does not exist
	05H	The 1st ID word is not in the specified data word of the data block (wrong DW no.) or the address list contains an incorrect ID word
	06H	Address list no. illegal
	07H	The function cannot be called at the current program execution level
OB 223	01H	Start-up modes same
	02H	Internal system error
	03H	Start-up modes not same
	04H	Single processor mode, no comparison of start-up modes possible

SF-OB	ACCU-1-LL	Meaning
OB 254/ 255	01H	Function processed correctly
	 41H 43H 48H 4AH 4BH 4CH 4DH 4EH	Errors: Block header on memory card invalid Not enough memory Source data block does not exist Block number or type illegal/source DB Block number or type illegal/destination DB Destination data block already exists in user memory Online function COMPRESS MEMORY active No memory card inserted
	 8DH 8EH	Warnings: Conflict with an online function (except COMPRESS MEMORY) 10-ms waiting time not yet elapsed

Word IDs in ACCU-1-L

Word results IDs are only used once (with one exception). The following table is therefore sorted according to ID values.

ID in ACCU-1-L	ID from SF OB:	Meaning
8D01H	OB 141	Illegal function no. in ACCU-2-L [1]
8D02H		One of the reserved bits in ACCU 1 is ’1’ [1]
8E01H	OB 142	Illegal function no. in ACCU-2-L [1]
8E02H		One of the reserved bits (no. 4 to 15) in ACCU 1 is ’1’ [1]
8EFFH		Wrong mode (e.g. when the delayed interrupt is to be disabled and DX 0 contains the parameter "process interrupts via IB0 = on"
8F01H	OB 143	Illegal function no. in ACCU-2-L [1]
8F02H		One of the reserved bits in ACCU 1 is ’1’ [1]
9601H	OB 150	Data block not loaded
960FH		Block called more than once
9611H		Illegal function no.
9612H		Address area type illegal
9613H		Data block no. illegal
9614H		"Number of first data field word" illegal
9615H		Data block length < 4 words
9621H		Year specified in data field illegal
9622H		Month specified in data field illegal
9623H		Day of month specified in data field illegal
9624H		Day of week specified in data field illegal
9625H		Hours specified in data field illegal
9626H		Minutes specified in data field illegal
9627H		Seconds specified in data field illegal
9628H		1/100 seconds in data field not 0
9629H		Hour format not as in OB 151

[1] the incorrect value is in ACCU-2-L

ID in ACCU-1-L	ID from SF OB:	Meaning
9701H 970FH 9710H 9711H 9712H 9713H 9714H 9715H 9721H 9722H 9723H 9724H 9725H 9726H 9727H 9728H 9729H 972AH	OB 151	Data block not loaded Block called more than once Wrong mode ("process interrupts via IB 0 = on") Illegal function no. Address area type illegal Data block no. illegal "Number of first data field word" illegal Data block length < 4 words Year specified in data field illegal Month specified in data field illegal Day of month specified in data field illegal Day of week specified in data field illegal Hours specified in data field illegal Minutes specified in data field illegal Seconds specified in data field illegal 1/100 seconds in data field not 0 Hour format not as in OB 121/OB 150 Job type illegal
990FH 9910H 9911H 9921H	OB 153	Block called more than once Wrong mode ("process interrupts via IB 0 = on") Illegal function no. Delay time illegal
B401H B410H B411H	OB 180	No data block is open The shift number S is not a multiple of 16 a) The shift number is too high; the block end is exceeded by the new window position. b) The shift number is negative.
B501H B502H B503H	OB 181	Block does not exist Wrong block number Wrong block ID

ID in ACCU-1-L	ID from SF OB:	Meaning
B601H	OB 182	Data block not loaded
B60FH		Block called more than once
B611H		Content of data field incorrect
B612H		Address area type illegal
B613H		Data block no. illegal
B621H		"Number of first data field word" illegal
B622H		"Source data block type" illegal
B623H		"Source data block no." illegal
B624H		"No. of first data word to be transferred in source DB" illegal
B625H		Length of source data block in block header < 5 words
B626H		"Destination data block type" illegal
B627H		"Destination data block no." illegal
B628H		"No. of first data word to be written in destination DB" illegal
B629H	OB 182 (cont.)	Length of destination data block in block header< 5 words
B62AH		"Number of data words to be transferred" illegal (= 0 or > 4091)
B62BH		Source data block too short
B62CH		Destination data block too short
F001H	OB 121	Illegal function no.
F00FH		Block called more than once
F101H		Year illegal
F102H		Month illegal
F103H		Day illegal
F104H		Day of week illegal
F105H		Hours illegal
F106H		Minutes illegal
F107H		Seconds illegal
F108H		1/100 to1/10 seconds illegal
F109H		Hour format not as in OB 151
F001H	OB 122	Illegal function no.

Indexes 13

Contents of Chapter 13

List of Abbreviations

Abbreviations

(An explanation of the ISTACK abbreviations can be found in Section 5.4)

ACCU-1 (2, 3, 4)-L	low word in accumulator 1 (2, 3, 4), 16 bit
ACCU-1 (2, 3, 4)-H	high word in accumulator 1 (2, 3, 4), 16 bit
ACCU-1 (2 ,3, 4)-LL	low byte of low word in accumulator 1 (2, 3, 4), 8 bit
ACCU-1 (2, 3, 4)-LH	high byte of low word in accumulator 1 (2, 3, 4), 8 bit
ADF	addressing error
ANZW	condition code word
BASP	disable command output (signal on S5 bus)
BCD	binary coded decimal
BR	base address register
BSTACK	block stack
CC 1, CC 0	condition code bits for digital operations
COR	coordinator module
CP	communications processor
CPU	central processing unit
CSF	control system flowchart
DB	data block
DBA	data block start address (in register 6)
DBL	data block length (in register 8)
DX	extended data block
EPROM	erasable programmable read only memory
ERAB	first scan (bit code)
EU	expansion unit
FB	function block
FX	extended function block
IM	interface module
INT	(system)interrupt
IP	intelligent peripheral module
ISTACK	interrupt stack

KB	call for a non-existent logic block
KDB	opening a non-existent DB/DX-data block
LAD	ladder diagram
LED	light-emitting diode
NAU	power failure
OB	organization block
OR	or (bit code)
OS	overflow latching (word code)
OV	overflow (word code)
PAFE	parameter assignment error byte
PARE	parity error
PB	program block
PEU	power failure on expansion unit
PG	programmer
PI	process image
PII	process image of the inputs
PIQ	process image of the outputs
PLC	programmable controller
QVZ	timeout
RAM	random-access memory
RLO	result of logic operation
SAC	step address counter
SB	sequence block
SPU	operating system processor
STA	status (bit code)
STL	statement list
STS	stop statement
SUF	substitution error
STUEB	BSTACK overflow
STUEU	ISTACK overflow
TRAF	transfer or load error
ZYK	cycle error

List of Key Words

Siemens AG
A&D AS E 81

Östliche Rheinbrückenstr. 50
D-76181 Karlsruhe
Federal Republic of Germany

From:
Your Name: _
Your Title: _
Company Name: _
Street: _
City, Zip Code_ _
Country: _
Phone: _

Please check any industry that applies to you:

❐ Automotive
❐ Chemical
❐ Electrical Machinery
❐ Food
❐ Instrument and Control
❐ Nonelectrical Machinery
❐ Petrochemical
❐ Pharmaceutical
❐ Plastic
❐ Pulp and Paper
❐ Textiles
❐ Transportation
❐ Other _ _ _ _ _ _ _ _ _ _ _

Remarks Form

Your comments and recommendations will help us to improve the quality and usefulness of our publications. Please take the first available opportunity to fill out this questionnaire and return it to Siemens.

Please give each of the following questions your own personal mark within the range from 1 (very good) to 5 (poor).

1. Do the contents meet your requirements? □
2. Is the information you need easy to find? □
3. Is the text easy to understand? □
4. Does the level of technical detail meet your requirements? □
5. Please rate the quality of the graphics/tables: □

Additional comments: