SIEMENS

# SIMATIC S5

# S5-155H Programmable Controller (CPU 948R)

**Programming Guide**
**Volume 2/2**

This manual is part of the documentation package with the order number:
**6ES5 998-4SR11**


How to Use the Manual, Contents

Introduction 1

User Program 2

Program Execution 3

Operating Statuses and Program Execution Levels 4

Interrupt and Error Diagnosis 5

Integrated Special Functions 6

Extended Data Block DX 0 7

Memory Assignment and Memory Organization 8

Memory Access Using Absolute Addresses 9

PG Interfaces and Functions 10

Appendix 11

Further Reading 12

List of Abbreviations
List of Keywords 13

*The Pocket Guide CPU 948R Order No. 6ES5 997-3UR21 is included in this manual.*



**03/2000**
**C79000-G8576-C136**
**Edition 05**

## 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.

**Copyright © Siemens AG 1994 All rights reserved**

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

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.

© Siemens AG 1994
Technical data subject to change.



Excellence in Automation & Drives: Siemens

# How to Use this Manual

## Scope

This programming guide describes the following models of the CPU 948R and its system software:

Models of the CPU 948R

- CPU 948R-1 with 640 Kbytes of user memory,
  Order no. 6ES5 948-**3UR11,** version 01 and higher

- CPU 948R-2 with 1 664 Kbytes of user memory,
  Order no. 6ES5 948-**3UR21**, version 01 and higher

## Overview of the Chapters

*Chapter 1*

This explains the typical mode of operation of the CPU and illustrates how a CPU program is structured.
The chapter also contains suggestions about how to tackle programming and which characteristics of the CPU 948R are important for programming.
If you have already worked with the CPU 946R/947R and want to know the differences between these CPUs and the CPU 948R you will find this information in this chapter.

*Chapter 2*

This explains the components of a STEP 5 user program and how the program can be structured.

*Chapter 3*

This is intended for readers who do not yet have much experience of using the STEP 5 programming language. It therefore deals with the basics of STEP 5 programming and explains the STEP 5 operations in detail (with examples).

Experienced readers who may find that the information about specific operations in the pocket guide is inadequate, can use Section 3.5 as a reference section.

*Chapter 4*

This provides an overview of the modes and program execution levels of the CPU 948R. It provides you with detailed information about various start-up modes and the associated organization blocks in which you can program your routines for differrent start-up situations.

The chapter also explains the differences between the program execution levels "cyclic processing", "time-controlled processing" and "interrupt-driven processing" and which blocks are available for your user program.

*Chapter 5*

This informs you about errors to be avoided when planning and writing your STEP 5 programs.
The chapter tells you about the help you can obtain from the system program for diagnosing errors and which reactions can be expected and informs you about the blocks in which you can program reactions to certain errors.
The chapter also explains the CPU 948R self-test.

| | |
|---|---|
| *Chapter 6* | This covers the special functions integrated in the system program. It tells you how to use the special functions and how to call and assign parameters to the special function OBs. The chapter also explains how to recognize and deal with errors in the processing of a special function. |
| *Chapter 7* | This describes the use of data block DX 0 and its structure. The chapter informs you of the significance of the various DX 0 parameters. Based on examples, you will learn how to create data block DX 0 or how to assign the parameters in a screen form. |
| *Chapter 8* | This is a reference section for experienced system users. It provides information about the memory organization of the CPU 948R and certain system data words which contain information that can be called up by the user. |
| *Chapter 9* | This is also for experienced system users. The chapter explains how to address data in certain memory areas using absolute addresses. |
| *Chapter 10* | This explains how to connect your CPU to a PG and the functions provided by the PG software for testing your STEP 5 program |
| *Chapter 11* | This contains the Appendix with technical data (typical execution times for operations, system times, memory capacity) of the CPU 946R/947R and the CPU 948R. |
| *Chapter 12* | This lists documentation for further reading |
| *Chapter 13* | This is intended to help you find topics quickly and contains a list of abbreviations and a list of keywords as well as lists of all the numbered tables and figures. |

## Conventions Used in the Text

To provide you with an overview of the contents of the pages, the manual uses the following conventions in addition to a 2nd and 3rd order of titles:

*Entries in the margin*

Entries in the margin are keywords printed in italics on the left-hand edge of a page. They provide information about the contents of one or more paragraphs on the page.

***Fourth order entries***

Fourth order entries are not numbered but appear in the margin in bold face and identify a longer section of text.

**The following conventions are also used.**

*Notes*

**Note**
Important information is indicated in this format.

*Instructions*

Instructions (often a sequence of operations to be performed) are represented in tables, e.g.

| Step | Action | Result |
|---|---|---|
| 1 | Switch the mode selector 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, switch the mode selector from STOP to RUN and back to STOP. | An OVERALL RESET is requested. The STOP LED flashes quickly. |

*Reference tables*

Specific information you may require at any time is contained in numbered tables as shown in the following example and can be found in the list of tables (refer to Chapter 13).

Table 3-2 Binary logic operations

| **Operation** | **Operand** | **Function** |
|---|---|---|
| A | | AND logic operation with scan for signal state "1" |
| O | | OR logic operation with scan for signal state "1" |
| | I 0.0 to 127.7<br>..... | of an input in the PII<br>..... |

*Examples*

Examples, some of which cover several pages, are highlighted by a gray frame. When the examples cover more than one page this is clearly indicated.

```
Example 1:   Calling and assigning parameters to a function block in the
             methods of representation STL and LAD/CSF in a program block


Method of representation STL

......
```

# Contents


**1 Introduction . . . . . 1-3**

1.1 Typical Mode of Operation of a CPU . . . . . 1-4

1.2 The Programs in a CPU . . . . . 1-6

1.3 Which Operands are Available to the User Program? . . . . . 1-10

1.4 How Much Memory is Available for the User Program? . . . . . 1-13

1.5 How to Tackle Programming . . . . . 1-14

1.6 What You Need to Know to Program an H System . . . . . 1-17

1.7 Programming Tools . . . . . 1-18

1.8 What is New with the CPU 948R? . . . . . 1-19

**2 User Program . . . . . 2-3**

2.1 STEP 5 Programming Language . . . . . 2-4

2.1.1 The LAD, CSF, STL Methods of Representation . . . . . 2-4
2.1.2 Structured Programming . . . . . 2-5
2.1.3 STEP 5 Operations . . . . . 2-6
2.1.4 Number Representation . . . . . 2-8
2.1.5 STEP 5 Blocks and Storing Them in Memory . . . . . 2-12

2.2 Program, Organization and Sequence Blocks . . . . . 2-16

2.2.1 Organization Blocks as User Interfaces . . . . . 2-18
2.2.2 Organization Blocks for Special Functions . . . . . 2-22

2.3 Function Blocks . . . . . 2-23

2.3.1 Structure of Function Blocks . . . . . 2-24
2.3.2 Programming Function Blocks . . . . . 2-26
2.3.3 Calling Function Blocks and Assigning Parameters to Them . . . . . 2-28
2.3.4 Special Function Blocks . . . . . 2-33

2.4 Data Blocks . . . . . 2-35

2.4.1 Creating Data Blocks . . . . . 2-37
2.4.2 Opening Data Blocks . . . . . 2-38
2.4.3 Special Data Blocks . . . . . 2-41

**3 Program Execution. . . . . 3-3**

3.1 Principle of Program Execution. . . . . 3-4

3.2 Program Organization . . . . . 3-5

3.3 Storing Program and Data Blocks . . . . . 3-10

3.4 Processing the User Program. . . . . 3-11
3.4.1 Definition of Terms used in Program Execution. . . . . 3-12

3.5 STEP 5 Operations with Examples . . . . . 3-15
3.5.1 Basic Operations. . . . . 3-19
3.5.2 Programming Examples in the STL, LAD and CSF Methods of Representation . . . . . 3-35
3.5.3 Supplementary Operations . . . . . 3-51
3.5.4 Executive Operations . . . . . 3-61

**4 Operating Statuses and Program Execution Levels . . . . . 4-3**

4.1 Introduction and Overview . . . . . 4-4

4.2 Program Execution Levels. . . . . 4-7

4.3 STOP Mode . . . . . 4-12
4.3.1 SOFT STOP . . . . . 4-12
4.3.2 HARD STOP . . . . . 4-16
4.3.3 OVERALL RESET . . . . . 4-17

4.4 START-UP Mode . . . . . 4-19
4.4.1 MANUAL and AUTOMATIC COLD RESTART . . . . . 4-21
4.4.2 MANUAL and AUTOMATIC COLD RESTART WITH MEMORY . . . . . 4-22
4.4.3 Comparison between COLD RESTART and COLD RESTART WITH MEMORY . . . . . 4-25
4.4.4 User Interfaces for Start-Up. . . . . 4-26
4.4.5 Interruptions during START-UP . . . . . 4-27
4.4.6 Examples of Programs for START-UP . . . . . 4-29

4.5 RUN Mode . . . . . 4-31
4.5.1 Cyclic Program Execution . . . . . 4-33
4.5.2 Specifying Time and Interrupt-Driven Program Execution . . . . . 4-36
4.5.3 Time-Controlled Program Execution. . . . . 4-37
4.5.4 Interrupt-Driven Program Execution . . . . . 4-45

**5 Interrupt and Error Diagnostics . . . . . 5-3**

5.1 Frequent Errors in the User Program . . . . . 5-4

5.2 Error Information . . . . . 5-5

5.3 Procedure for Error Analysis . . . . . 5-9

5.4 Control Bits and Interrupt Stack . . . . . 5-10
5.4.1 Control Bits . . . . . 5-11
5.4.2 ISTACK Content . . . . . 5-15
5.4.3 Example of Error Diagnosis using the ISTACK . . . . . 5-20

5.5 Error Data Block (Error DB ) . . . . . 5-21
5.5.1 What is Entered in the Error DB? . . . . . 5-21
5.5.2 Structure of the Error DB . . . . . 5-22
5.5.3 Calling a User Interface . . . . . 5-26

5.6 H Flag Word . . . . . 5-27

5.7 Error Handling Using Organization Blocks . . . . . 5-29

5.8 Causes of Error and Reactions of the CPU . . . . . 5-32
5.8.1 OB 19: Calling a Logic Block That Is Not Loaded (KB) . . . . . 5-33
5.8.2 OB 19: Opening a Data Block That Is Not Loaded (KDB) . . . . . 5-33
5.8.3 OB 23/24, OB 28/29: Timeout Error (QVZ) . . . . . 5-34
5.8.4 OB 25: Addressing Error (ADF) . . . . . 5-35
5.8.5 OB 26: Cycle Time Exceeded Error (ZYK) . . . . . 5-36
5.8.6 OB 27: (Substitution Error SUF) . . . . . 5-37
5.8.7 OB 30: Parity Error and Timeout Error in the User Memory (PARE) . . . . . 5-37
5.8.8 OB 32: Load and Transfer Error (TRAF) . . . . . 5-38
5.8.9 OB 33: Collision of Timed Interrupts Error (WEFES/WEFEH) . . . . . 5-39
5.8.10 OB 34: Error with G DB/GX DX (FEDBX) . . . . . 5-41
5.8.11 OB 37: Entry in the Error DB . . . . . 5-41

5.9 Self-Test . . . . . 5-43
5.9.1 Overview . . . . . 5-43
5.9.2 Organization of the Self-Test . . . . . 5-44

**6 Integrated Special Functions ..... 6-3**

6.1 Introduction ..... 6-4
6.1.1 Interfaces ..... 6-5
6.1.2 Error Handling ..... 6-6
6.1.3 Notes on the Special Functions OB 124, OB 125 and OB 254/255 ..... 6-7

6.2 OB 121: Set/Read System Time ..... 6-8

6.3 OB 122: "Disable Interrupts" On/Off ..... 6-12

6.4 OB 124: Delete STEP 5 Blocks ..... 6-14

6.5 OB 125: Generate STEP 5 Blocks ..... 6-17

6.6 OB 129: Evaluate Battery State ..... 6-20

6.7 OB 131: Delete ACCUs 1, 2, 3 and 4 ..... 6-21

6.8 OB 132/133: Roll-Up ACCU/Roll-Down ACCU ..... 6-22

6.9 OB 141: "Disable Single Cyclic Timed Interrupts" On/Off ..... 6-24

6.10 OB 142: "Delay All Interrupts" On/Off ..... 6-27

6.11 OB 143: "Delay Single Cyclic Timed Interrupts" On/Off ..... 6-30

6.12 OB 150: Set/Read System Time ..... 6-33

6.13 OB 151: Set/Read Time for Real Time Interrupt ..... 6-38

6.14 OB 153: Set/Read Time for Delayed Interrupt ..... 6-45

6.15 OB 180: Random Data Block Access ..... 6-48

6.16 OB 181: Test Data Blocks (DB/DX) ..... 6-52

6.17 OB 182: Copy Data Area ..... 6-54

6.18 OB 222: Restart Cycle Monitoring Time ..... 6-57

6.19 OB 254/255: Copy/Duplicate Data Blocks ..... 6-58

**7 Extended Data Block DX 0 ..... 7-3**

7.1 Application ..... 7-4

7.2 Parameter Assignment using the PG Screen Form ..... 7-5

7.3 Meaning of Some Parameters for Running the Program ..... 7-7
7.3.1 Updating the Timers ..... 7-7
7.3.2 Setting the Cycle Monitoring Times ..... 7-7
7.3.3 Level Priorities ..... 7-7

**8 Memory Assignment and Memory Organization . . . . . 8-3**

8.1 Structure of the Memory Area . . . . . 8-4

8.2 Memory Assignment in the CPU 948R . . . . . 8-5
8.2.1 Memory Assignment for the System RAM . . . . . 8-6
8.2.2 Memory Assignment for the I/O . . . . . 8-8

8.3 User Memory Organization in the CPU 948R . . . . . 8-10
8.3.1 Block Headers in User Memory . . . . . 8-12
8.3.2 Block Address List in Data Block DB 0 . . . . . 8-13
8.3.3 RI/RJ Area . . . . . 8-14
8.3.4 RS/RT Area . . . . . 8-15
8.3.5 Bit Assignment of the System Data Words . . . . . 8-18
8.3.6 Addressable System Data Area . . . . . 8-38

**9 Memory Access Using Absolute Addresses . . . . . 9-3**

9.1 Introduction . . . . . 9-4

9.2 Memory Access via Address in ACCU 1 . . . . . 9-8
9.2.1 LIR/TIR: Loading to or Transferring from a 16-Bit Memory Area Indirectly . . . . . 9-9
9.2.2 Examples of Access to DW > 255 . . . . . 9-15
9.2.3 LDI/TDI: Loading to or Transferring from a 32-Bit Memory Area Indirectly . . . . . 9-17

9.3 Transferring Memory Blocks . . . . . 9-19

9.4 Operations with the Base Address Register (BR Register) . . . . . 9-22
9.4.1 Operations for Transfer between Registers . . . . . 9-23
9.4.2 Accessing the Local Memory . . . . . 9-24
9.4.3 Accessing the Global Memory . . . . . 9-25
9.4.4 Accessing the Page Memory . . . . . 9-29

**10 PG Interfaces and Functions . . . . . 10-3**

10.1 Overview . . . . . 10-4

10.2 Notes on Connecting the PG and the PLC Modes . . . . . 10-5
10.2.1 Connecting the PG . . . . . 10-5
10.2.2 PLC Modes . . . . . 10-6

10.3 PG Functions . . . . . 10-8
10.3.1 Info . . . . . 10-9
10.3.2 Installation . . . . . 10-10
10.3.3 Program Test . . . . . 10-11

10.4 Serial Link PG - PLC via the Serial Interface . . . . . 10-18

10.5 PG Functions via the S5 Bus . . . . . 10-19
10.5.1 Application . . . . . 10-19
10.5.2 How the PG Functions via the S5 Bus Work . . . . . 10-20
10.5.3 General Notes on Installation. . . . . 10-21
10.5.4 Installation with One-Sided Operation . . . . . 10-22
10.5.5 Installation for Switched Operation. . . . . 10-26
10.5.6 Displaying Problems . . . . . 10-31

**11 Appendix . . . . . 11-3**

**12 Further Reading . . . . . 12-3**

**13 List . . . . . 13-1**

List of Abbreviations . . . . . 13-3

List of Keywords . . . . . 13-5

# Introduction 1

## Contents of Chapter 1


1.1 Typical Mode of Operation of a CPU . . . . . 1 - 4

1.2 The Programs in a CPU . . . . . 1 - 6

System Program . . . . . 1 - 6
User Program . . . . . 1 - 8

1.3 Which Operands are Available to the User Program? . . . . . 1 - 10

1.4 How Much Memory is Available for the User Program? . . . . . 1 - 13

1.5 How to Tackle Programming . . . . . 1 - 14

1.6 What You Need to Know to Program an H System . . . . . 1 - 17

1.7 Programming Tools . . . . . 1 - 18

1.8 What is New with the CPU 948R? . . . . . 1 - 19

# Introduction 1

*Aims of the manual*

This manual is intended to provide specialized information about programming the CPU 948R for users who already have basic knowledge of programming PLCs and want to use the CPU 948R in the S5-155H 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.

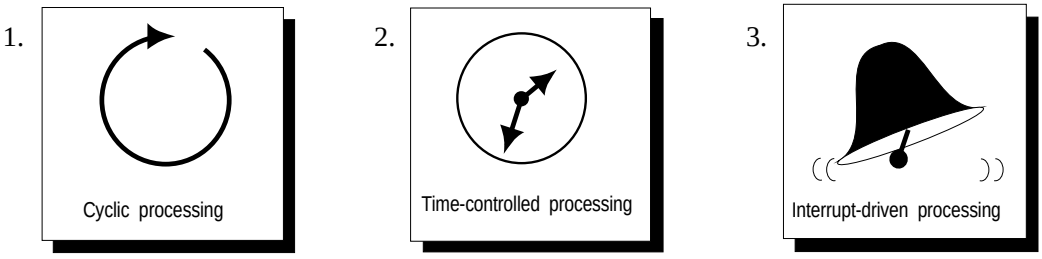
*Contents of Chapter 1*

Chapter 1 explains 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 948R which are important for programming.
If you have already worked with the CPU 946R/947R and would like to know the differences between these modules and the CPU 948R, refer to Section 1.8.

## 1.1 Typical Mode of Operation of a CPU

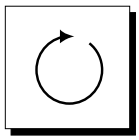
*Mode of operation of a CPU*

The following modes of operation are possible in a CPU:



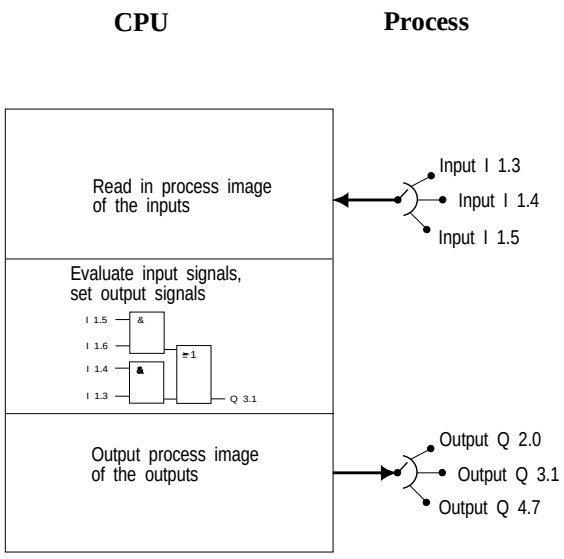

*Cyclic processing*

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.



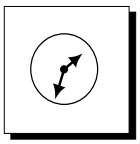
Cyclic processing can be divided into three main phases, as follows:

| 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. |



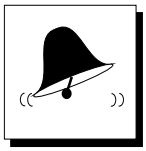
*Time-controlled processing*

In addition to cyclic processing, **time-controlled** processing is also available for processes requiring control signals at constant intervals, e.g. closed-loop control functions.



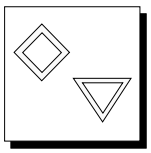
*Interrupt-driven processing*

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-driven processing depends, among other things, on the particular task. For this reason, the priority of time and interrupt-driven processing on the CPU 948R can be selected.

## 1.2 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. 1-1).

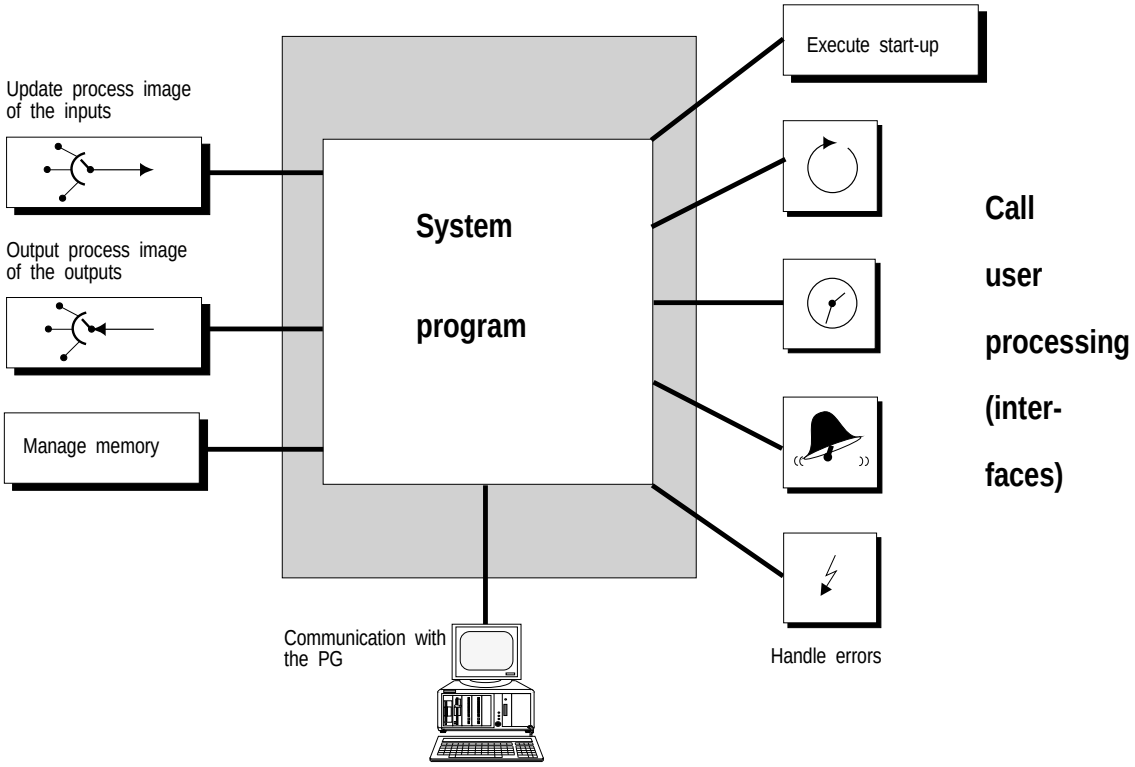



Fig. 1-1 Tasks of the system program

*Tasks*

The tasks include the following:

- restart (cold restart and cold restart with memory,
- 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**.

### *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 |
|---|---|
| Restart (cold restart and cold restart with memory) | To provide the conditions under which the other processing functions can start from a defined status following a cold restart of the controller (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:<br>- faster than the average cycle,<br>- at a time interval greater than the average cycle time,<br>- 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*

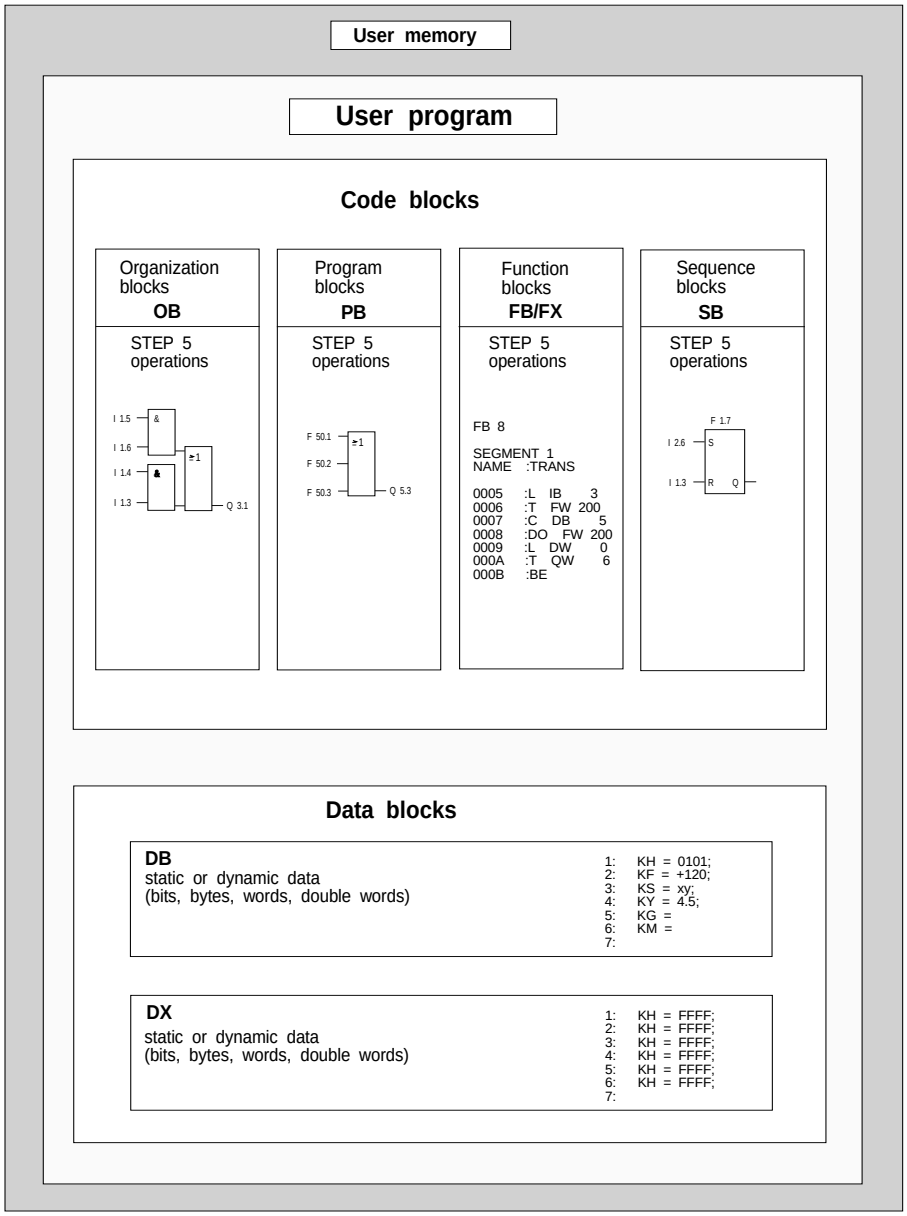



Fig. 1-2 Structure of a STEP 5 user program

*Storing the user program*

After programming the user program, you must load it in the user memory of the CPU 948R (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.

## 1.3 Which Operands are Available to the User Program?

The CPU 948R provides the following operand areas for programming:

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

*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:<br>- single bits,<br>- bytes,<br>- words,<br>- 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.<br><br>The following data types are possible:<br>- bytes,<br>- 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.<br><br>The following data types are possible:<br>- bytes,<br>- 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.<br>The flag area should be used ideally for working data required often.<br><br>The following data types can be accessed:<br>- single bits,<br>- bytes,<br>- words,<br>- double words.<br><br>Single flag bytes can be used as **interprocessor communication flags** (IPC flags) to exchange data between the CPU and CPs/IPs.<br>IPC flags are updated by the system program at the end of the cycle via a buffer in the CP/IP. | 2048 bits |

*S flags (extended flag area)*

| Characteristics | Size |
|---|---|
| The CPU 948R 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.<br><br>S flags **cannot**, however, be used as **actual operands** with function block calls nor as **IPC flags**. The bit test operations of the CPU 948R can also not be used with the S flags.<br><br>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).<br>The following data types can be accessed:<br>- single bits,<br>- bytes,<br>- words,<br>- 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).

## 1.4 How Much Memory is Available for the User Program?

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

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

- Version 1: with 640 Kbytes,
- Version 2: with 1,664 Kbytes.

## 1.5 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 | In contrast to the CPU 948, configure the structure of the I/Os and your transfer data (data block lists) with COM 155H (see Volume 1/1). |
| 4 | Start up the blocks one after the other (you may have to program a different OB 1 for each individual step, to call the logic blocks):<br>1a: load the block(s)<br>1b: test the block(s)<br><br>(For more detailed information please refer to your STEP 5 manual and Chapter 10). |
| 5 | 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.

## 1.6 What You Need to Know to Program an H System

***Range of operations***

Programming the CPU 948R is essentially the same as programming a CPU 948. With the exception of the operations for the multiprocessor mode, you can use all the STEP 5 operations.

***PG operation***

Operating the programmer (PG) is exactly the same as with the S5-155U. The PG is only connected to one of the two central controllers (CC); the data you input are automatically transferred to the second CC when the S5-155H is operating in the redundant mode.

***Program execution***

While the user program is running on the S5-155H, alongside the cyclic processing of the process images on the CPUs of the subunits other system functions are executed for synchronization, for exchanging and comparing data and, in some situations, operating modes.

***Event-synchronized user program execution***

The same user program is run in both devices (master and reserve) of the S5-155H. The master and reserve operate **event synchronized**. The system program of the CPU 948R makes sure that both devices work with identical data (for more information refer to Part I of Volume 1 of this manual).

***Process image of the inputs***

At the start of each cycle, the master and reserve read the process images of the one-sided and redundant inputs assigned to them. The process image of the switched inputs is only read in by the master. The two devices then exchange the complete process image of the inputs (for more information refer to Part I of Volume 1 of this manual).

## 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 948R in your automation software, 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 948R 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 What is New with the CPU 948R?

Compared with the CPU 946R/947R, the CPU 948R has new characteristics and functions and significant improvements in performance.

### *Faster processing*

The CPU 948R is between three and five times faster than the CPU 946R/947R. The exact execution times can be found in your Pocket Guide /1/.

### *Larger user memory*

| CPU 948R<br>only RAM (integrated) | CPU 946R/947R<br>RAM or EPROM cartridges |
|---|---|
| 640 Kbytes or 1664 Kbytes in internal RAM integrated in the CPU 948R, for logic and data blocks. | Max. 896 Kbytes, of which 128 Kbytes fixed as RAM plus additional 6 RAM or EPROM cartridges, each with 128 Kbytes in 355 memory modules. |

### *Memory Card*

| CPU 948R | CPU 946R/947R |
|---|---|
| SIMATIC Memory Card (Flash EPROM), 256 Kbytes to 2 Mbytes;<br>the memory card is programmed on the PG and its contents loaded in the user memory (internal RAM) following an OVERALL RESET. | SIMATIC memory cartridges RAM or EPROM, max. 128 Kbytes;<br>data blocks can be copied to the DB RAM from EPROM (OB 254/255). |

***Expansion of time-controlled program processing***

Apart from the familiar cyclic timed interrupts at fixed intervals, the following new interrupts are also processed:

- delayed interrupt (OB 6),
- real time interrupt (OB 9).

Both interrupts are only possible when "process interrupts via IB 0 = **off**" is set in DX 0.

***Expansion of PG communication***

- PG functions via SINEC H1 and parallel backplane bus.

***New special functions***

- Delete ACCU 1 to ACCU 4 (OB 131)
  Roll-up ACCU (OB 132)
  Roll-down ACCU (OB 133)
- Enable/disable "block single timed interrupts" (OB 141)
  Enable/disable "delay all interrupts" (OB 142)
  Enable/disable "delay single timed interrupts" (OB 143)
- Read/set system time, compatible with CPU 928B (OB 150),
- Assign parameters for real time interrupt (OB 9), (OB 151)
- Assign parameters for delayed interrupt (OB 6), (OB 153)
- Restart cycle monitoring time (OB 222)

***Larger BSTACK***

The BSTACK of the CPU 948R can contain 20 entries more (max. 59, 50 of which are for the user) than the BSTACK of the CPU 946R/947R.

# User Program 2

## Contents of Chapter 2


2.1 STEP 5 Programming Language . . . . . 2 - 4

2.1.1 The LAD, CSF, STL Methods of Representation . . . . . 2 - 4
2.1.2 Structured Programming . . . . . 2 - 5
2.1.3 STEP 5 Operations . . . . . 2 - 6
2.1.4 Number Representation . . . . . 2 - 8
2.1.5 STEP 5 Blocks and Storing Them in Memory . . . . . 2 - 12

2.2 Program, Organization and Sequence Blocks . . . . . 2 - 16

2.2.1 Organization Blocks as User Interfaces . . . . . 2 - 18
2.2.2 Organization Blocks for Special Functions . . . . . 2 - 22

2.3 Function Blocks . . . . . 2 - 23

2.3.1 Structure of Function Blocks . . . . . 2 - 24
2.3.2 Programming Function Blocks . . . . . 2 - 26
2.3.3 Calling Function Blocks and Assigning Parameters to Them . . . . . 2 - 28
2.3.4 Special Function Blocks . . . . . 2 - 33

2.4 Data Blocks . . . . . 2 - 35

2.4.1 Creating Data Blocks . . . . . 2 - 37
2.4.2 Opening Data Blocks . . . . . 2 - 38
2.4.3 Special Data Blocks . . . . . 2 - 41

# User Program

# 2

The following chapter explains the components that make up a STEP 5 user program for the CPU 948R 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.

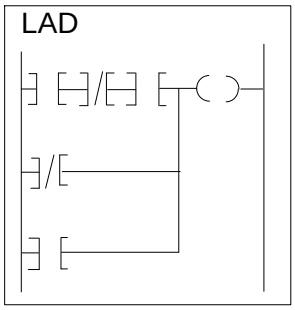
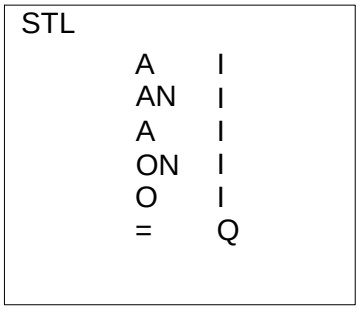
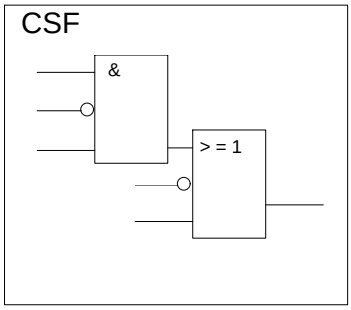
| Ladder diagram | Statement list | Control system flowchart |
|---|---|---|
| Programming with graphic symbols like a circuit diagram | Programming with mnemonic abbreviations of function designations | Programming with graphic symbols |
| complies with DIN 19239 | complies with DIN 19239 | complies with IEC 117-15 DIN 40700 DIN 40719 DIN 19239 |
|  |  |  |



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

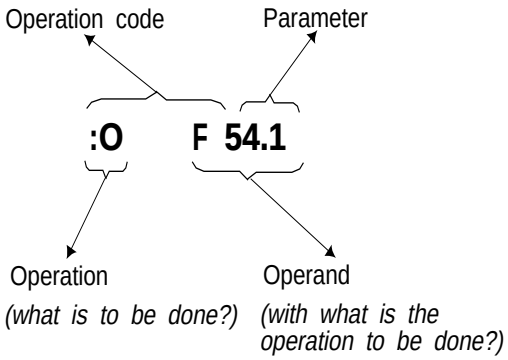
*What is a block?*

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*

-32768 to +32767 (16 bit)

-2147483648 to +2147483647 (32 bit)
(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 948R, 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:

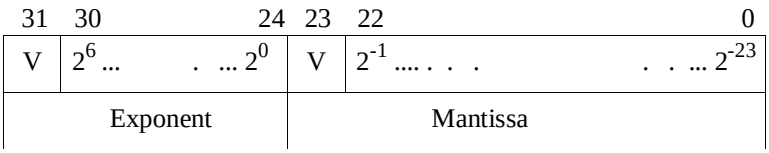
  $\mathbf{2^{-24} = 0.000000059604}$ (corresponds to 7 decimal places)

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:**




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

*Permissible numerical range*

$\pm 0.1469368 \times 10^{-38}$ to $\pm 0.1701412 \times 10^{39}$

*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**

PG display after you enter the line:

**:L KG + 1234567 + 02**

Mantissa with sign — Exponent (base 10) with sign

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

b) in a data block:

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:

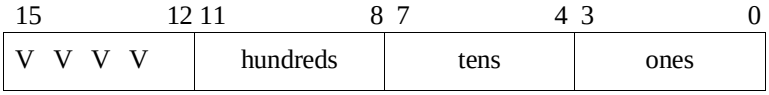
**6: KG =- 5000000 - 02**

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:




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 119 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 [1)] |
| DX | 3 to 255 | |

Data blocks DB 0, DB 1, DX 0 and DX 1 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.

1) Of these, two numbers from the 'DB' area must be left free for the error DB and RAM DB.

### Block storage

The PG 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.

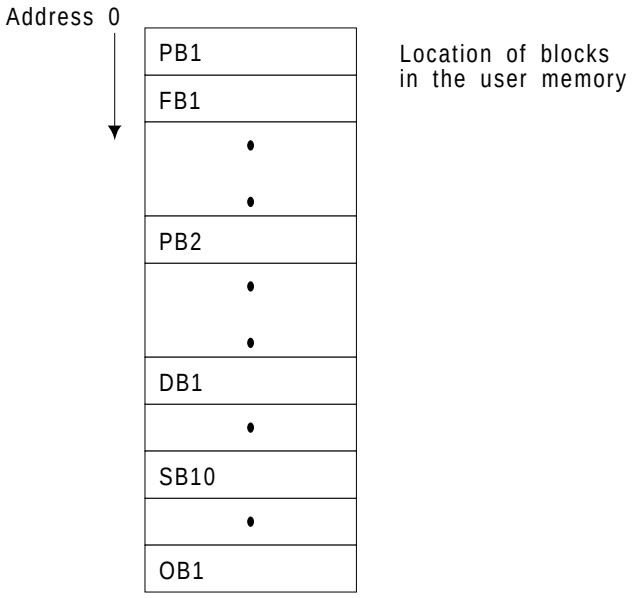



Fig. 2-2 Example of block storage in the user memory

### 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.<br><br>The following numbers are available for the type of block listed:<br>- program blocks 0 to 255<br>- sequence blocks 0 to 255<br>- organization blocks 1 to 39 |
| 2 | Enter your program in the STEP 5 programming language.<br><br>When programming PBs, OBs and SBs, you can only use the STEP 5 **basic** operations!<br><br>A STEP 5 block should always be a self-contained program section.<br>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.

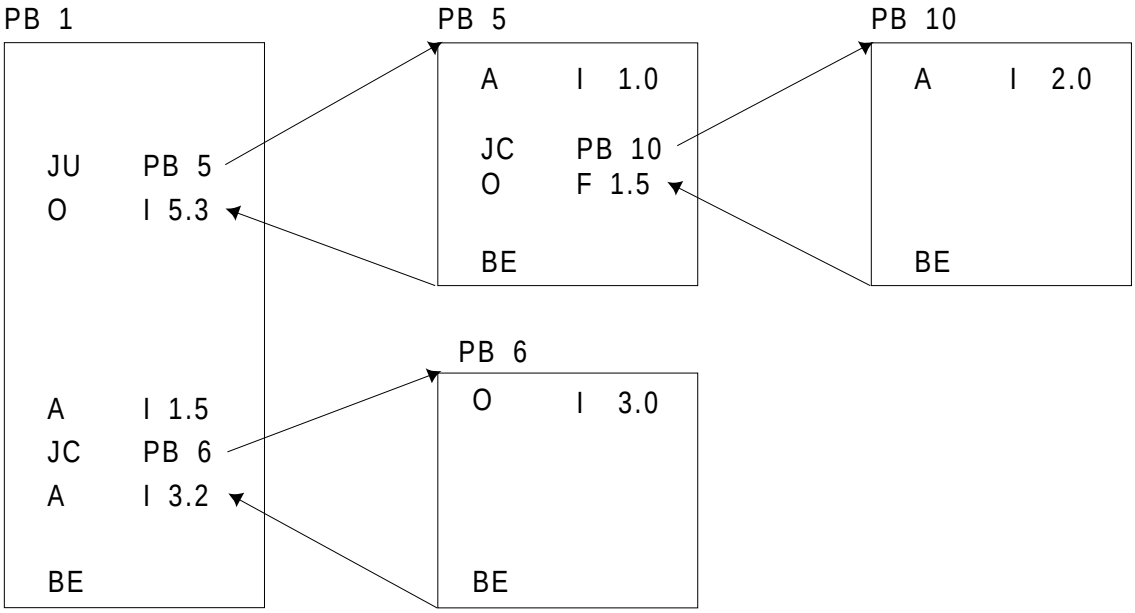



Fig. 2-3 Block calls that enable processing of a program block

*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 948R 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>OB 2<br>OB 3<br>OB 4<br>OB 5<br>OB 6<br>OB 7<br>OB 8<br>OB 9</td><td colspan="3">With DX-0 setting "Process interrupt servicing via input byte IB 0 =on":<br>(interruptability at block boundaries)<br><br>Call with signal state change in input byte IB 0 in bit:<br>I 0.0<br>I 0.1<br>I 0.2<br>I 0.3<br>I 0.4<br>I 0.5<br>I 0.6<br>I 0.7</td></tr>
<tr><td>OB 6<br>OB 9</td><td colspan="3">With DX-0 setting "Process interrupt servicing via input byte IB 0 = off":<br>(interruptability at block boundaries)<br><br>Delayed interrupt<br>Real time interrupt</td></tr>
<tr><td rowspan="2">OB 10<br>OB 11<br>OB 12<br>OB 13<br>OB 14<br>OB 15<br>OB 16<br>OB 17<br>OB 18</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; (interruptability at block boundaries)<br>Calls as follows:</td></tr>
<tr><td>Default setting</td><td>150U clock distributor</td><td>$2^n$ clock distributor</td></tr>
<tr><td></td><td>0.1 s<br>0.2 s<br>0.5 s<br>1.0 s<br>2.0 s<br>5.0 s<br>10.0 s<br>20.0 s<br>50.0 s</td><td>T * 1<br>T * 2<br>T * 5<br>T * 10<br>T * 20<br>T * 50<br>T * 100<br>T * 200<br>T * 500</td><td>T * 1<br>T * 2<br>T * 4<br>T * 8<br>T * 16<br>T * 32<br>T * 64<br>T * 128<br>T * 256</td></tr>
</table>

Table 2-2 Overview of the organization blocks of the CPU 948R 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 COLD RESTART WITH MEMORY |
| OB 22 | Call on request for AUTOMATIC COLD RESTART WITH MEMORY |

Table 2-3 Organization blocks of the CPU 948R 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 948R 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) |

| Organization blocks for reaction to device or program errors [1] | |
|---|---|
| **Block** | **Function and call criterion** |
| OB 26 | Cycle time exceeded (ZYK) |
| Table 2-4 continued: | |
| OB 27 | Substitution error (SUF) |
| OB 28 | Timeout input byte IB 0<br>(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 37 | Entry in the error DB |

[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, OB 24 (timeout), OB 29 or OB 33 do not exist, there is no reaction!

[2] OB 31 only exists in the CPU 948R 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 * 100 ms".

| Operating system organization blocks of the CPU 948R | |
|---|---|
| **Block** | **Function** |
| OB 100 to<br>OB 119 | Internal blocks 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 948R 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 129 | Evaluate 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 real time interrupt time |
| OB 153 | Set/read time for delayed interrupt |
| OB 180 | Random data block access |
| OB 181 | Test data blocks (DB/DX) |
| OB 182 | Copy data area |
| OB 222 | Restart cycle monitoring time |
| 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,<br>- supplementary operations<br>- system operations |
| 2. | **Method of representation** | |
| | programming and call<br>in STL, LAD, CSF | programming only in AWL |
| 3. | **Name** | |
| | name environment not possible<br>(only number) | in addition to the number<br>a name with max. 8 chars. can be assigned |
| 4. | **Operands** | |
| | none | formal operands (block parameters).<br>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.

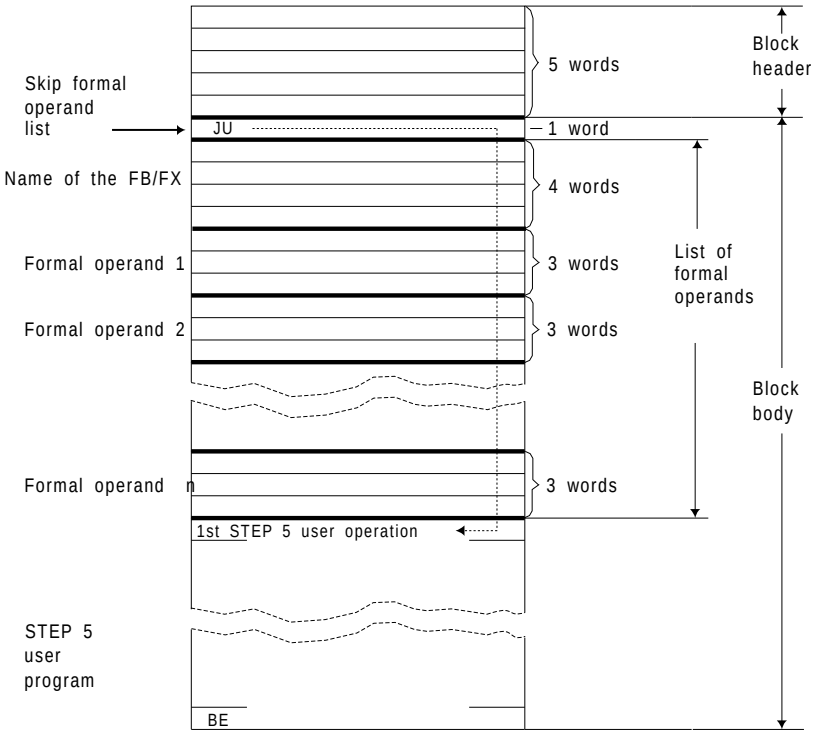



Fig. 2-4 Structure of a function block (FB/FX)

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.<br>Make sure you keep to the required order:<br>1. First program the FB/FX with the formal operands and keep it on the PG (offline) or in the CPU memory (online)<br>2. 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.<br><br>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.<br><br>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.<br><br>Enter the following information for each formal operand:<br><br>- the name of the block parameter (maximum 4 characters),<br>- the type of block parameter and the data type of the block parameter (if applicable)<br><br>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 2-6 Permitted formal operands for function blocks

| Parameter type | Data type |
|---|---|
| I = input parameter<br>Q = output parameter | BI/BY/W/D |
| D = data | KM/KH/KY/KS/KF/<br>KT/KC/KG |
| B = block operation<br>T = timer<br>C = counter | none<br>(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.<br><br>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.<br><br>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<br>**"DOU FXn"** for FX extended function blocks:<br>the referenced function block is processed regardless of the previous result of logic operation (RLO). | "JC FBn" for FB function blocks or<br>**"DOC FXn"** for FX extended function blocks:<br>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 always 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 2-7 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 |
| | D | for an operand with double word address | OW | n | word from extended periphery |
| | | | 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 2-7 continued: | | |
| **D** (Cont.) | KF for a fixed point number -32768 to +32767<br><br>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<br><br>FB n Function block (permitted only without parameters) called unconditionally (JU . .n)<br><br>OB n Organization block called unconditionally (JU . .n)<br><br>PB n Program blocks - called unconditionally (JU . .n)<br><br>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 | C 0 to 255 Counter |

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

**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.

*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

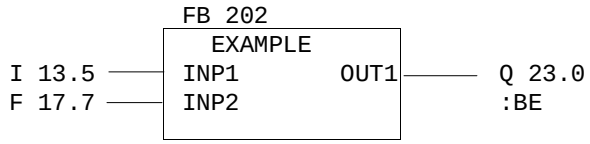
**Function block FB 202 is called and has parameters assigned to it in program block PB 25:**

STL method of representation

```
PB 25
SEGMENT 1

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

CSF/LAD method of representation




Formal operands

Actual operands

**The following operations are executed after the jump to FB 202**

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

**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**

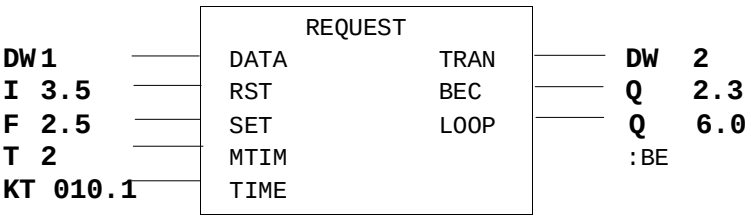
```
PB 25
SEGMENT 1
     :
     :  C  DB 5
     :
     :  JU   FB 201
NAME :  REQUEST
DATA :       DW   1
RST  :       I    3.5
SET  :       F    2.5
MTIM :       T    2
TIME :       KT   010.1
TRAN :       DW   2
BEC  :       Q    2.3
LOOP :       Q    6.0
     :  BE
```

Formal operands

Actual operands

**CSF/LAD method of representation**

```
PB 25
SEGMENT 1
                FB 201
```



### 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 S5-155H 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.

Occupied flag words: FW 238 to FW 254.

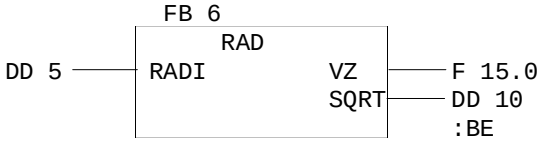
*Continued on next page*

*"Floating point root extractor" continued:*

STL 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
```

LAD method of representation




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.<br><br>(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 2-8 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.

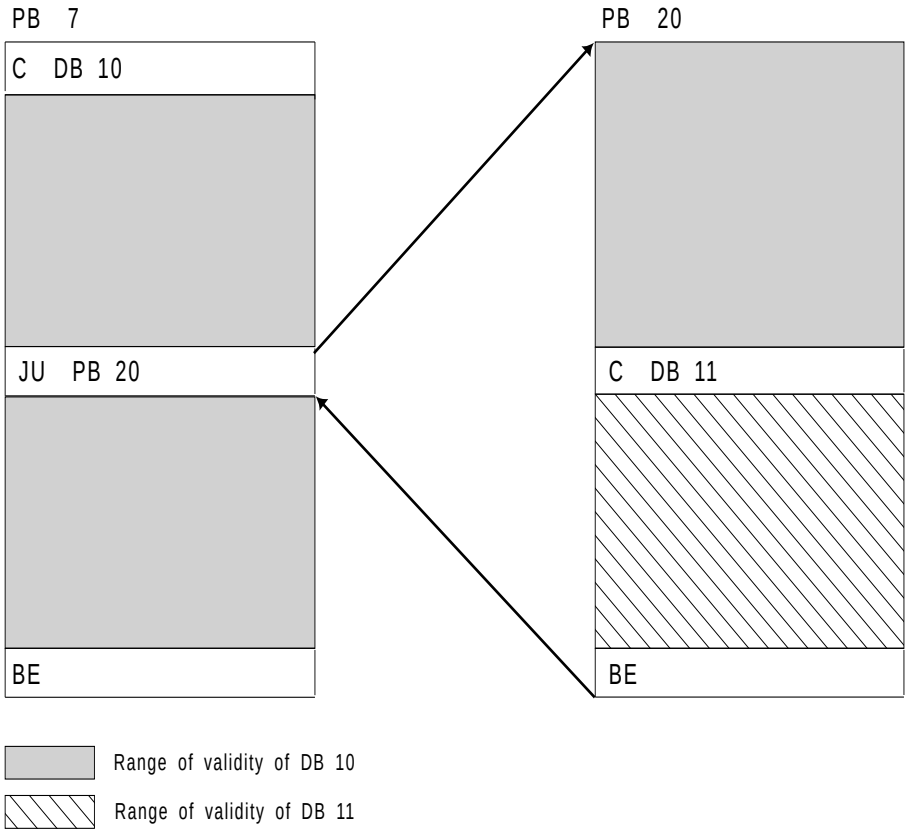



Fig. 2-5 Range of validity of an opened data block

## 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**

  Parameter assignment data block for COM 155H (see Part I of Volume 1)

*DX 2*

- **Data block DX 2**

  Reserved.

# Program Execution 3

## Contents of Chapter 3


3.1 Principle of Program Execution . . . . . 3 - 4

3.2 Program Organization . . . . . 3 - 5

3.3 Storing Program and Data Blocks . . . . . 3 - 10

3.4 Processing the User Program . . . . . 3 - 11

3.4.1 Definition of Terms used in Program Execution . . . . . 3 - 12

3.5 STEP 5 Operations with Examples . . . . . 3 - 15

3.5.1 Basic Operations . . . . . 3 - 19
- Binary logic operations . . . . . 3 - 19
- Set/reset operations . . . . . 3 - 20
- Load and transfer operations . . . . . 3 - 21
- Timer and counter operations . . . . . 3 - 26
- Arithmetic operations . . . . . 3 - 31
- Comparison operations . . . . . 3 - 32
- Block operations . . . . . 3 - 32
- NOP/display/stop operations . . . . . 3 - 34

3.5.2 Programming Examples in the STL, LAD and CSF Methods of Representation . . . . . 3 - 35

3.5.3 Supplementary Operations . . . . . 3 - 51
- Binary logic operations . . . . . 3 - 52
- Digital logic operations . . . . . 3 - 52
- Bit test operations . . . . . 3 - 53
- Set/reset operations . . . . . 3 - 54
- Timer and counter operations . . . . . 3 - 55
- Load and transfer operations . . . . . 3 - 57
- Arithmetic operations . . . . . 3 - 59

3.5.4 Executive Operations . . . . . 3 - 61
Jump operations. . . . . 3 - 61
Shift operations . . . . . 3 - 63
Conversion operations. . . . . 3 - 65
Decrement/increment . . . . . 3 - 68
Processing operations . . . . . 3 - 68
I/O operations . . . . . 3 - 74
Other operations . . . . . 3 - 74

# 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 948R.

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).

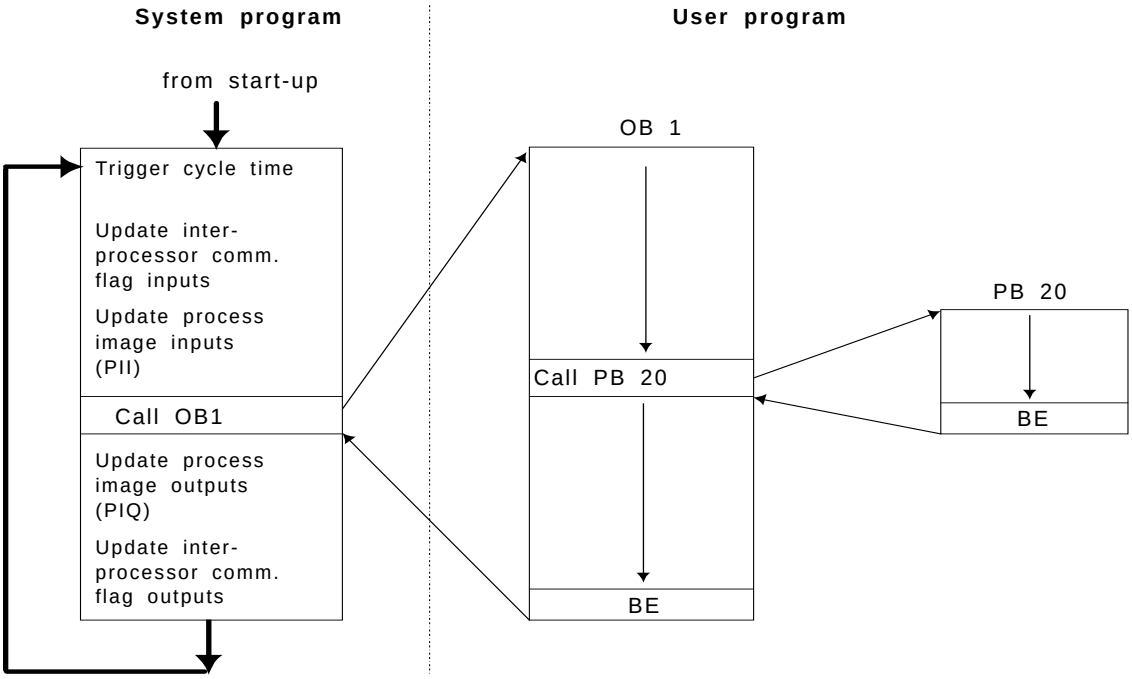



Fig. 3-1 Principle of cyclic program execution

## 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.

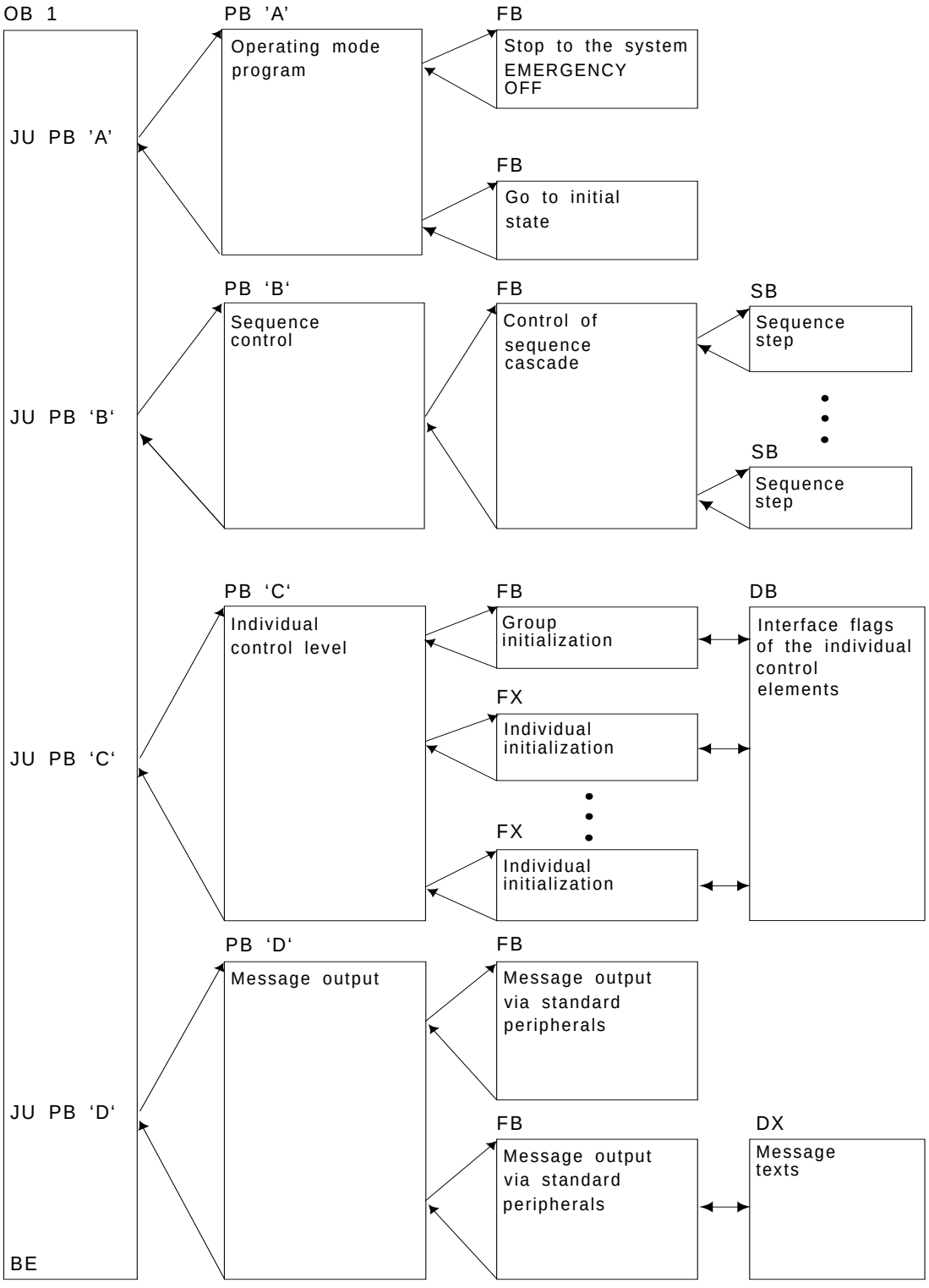



Fig. 3-2 Example of the organization of the user program according to the program structure

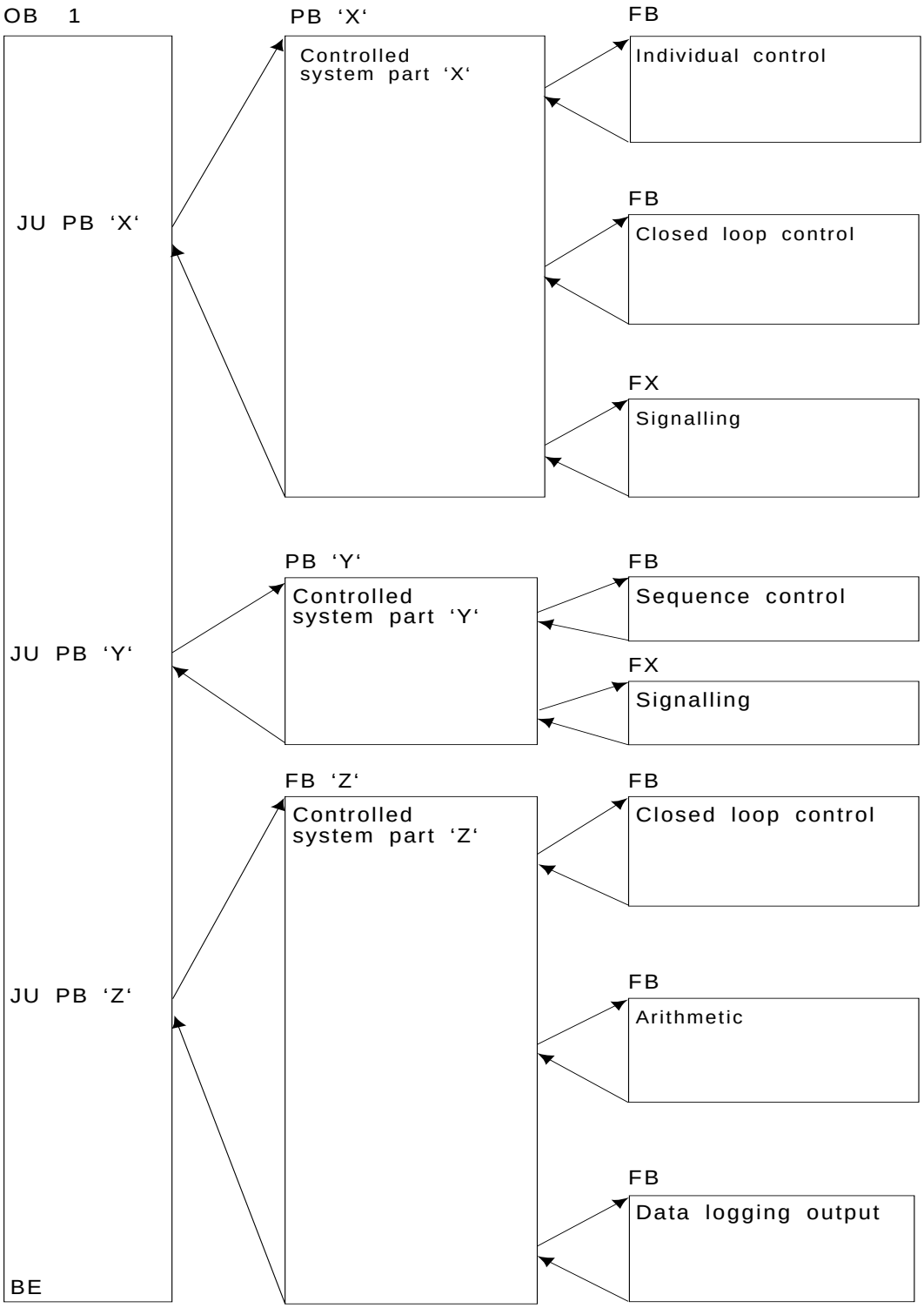
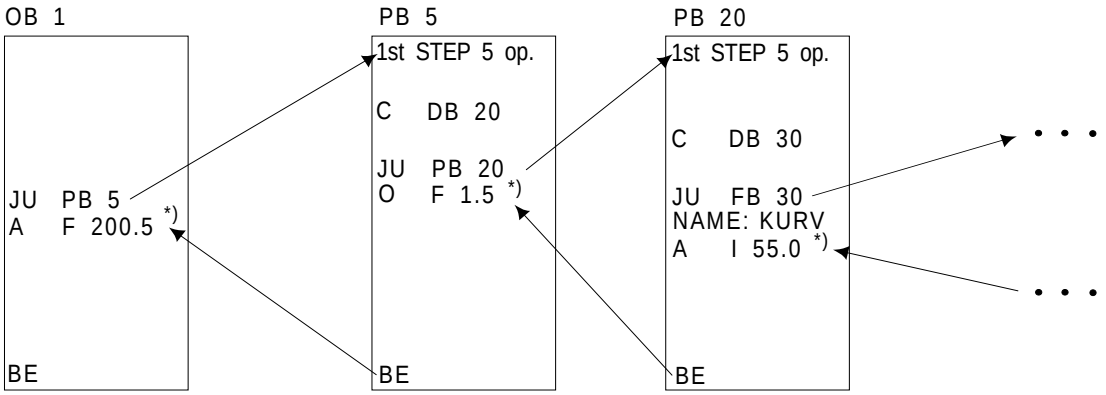



Fig. 3-3 Example of the organization of the user program according to the structure of the controlled system

### *Nesting blocks*

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




*) Operation to which the program returns

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 50 blocks within one another. If more than 50 blocks are called, the CPU signals an error and goes to the stop mode.

*Example of nesting depth*

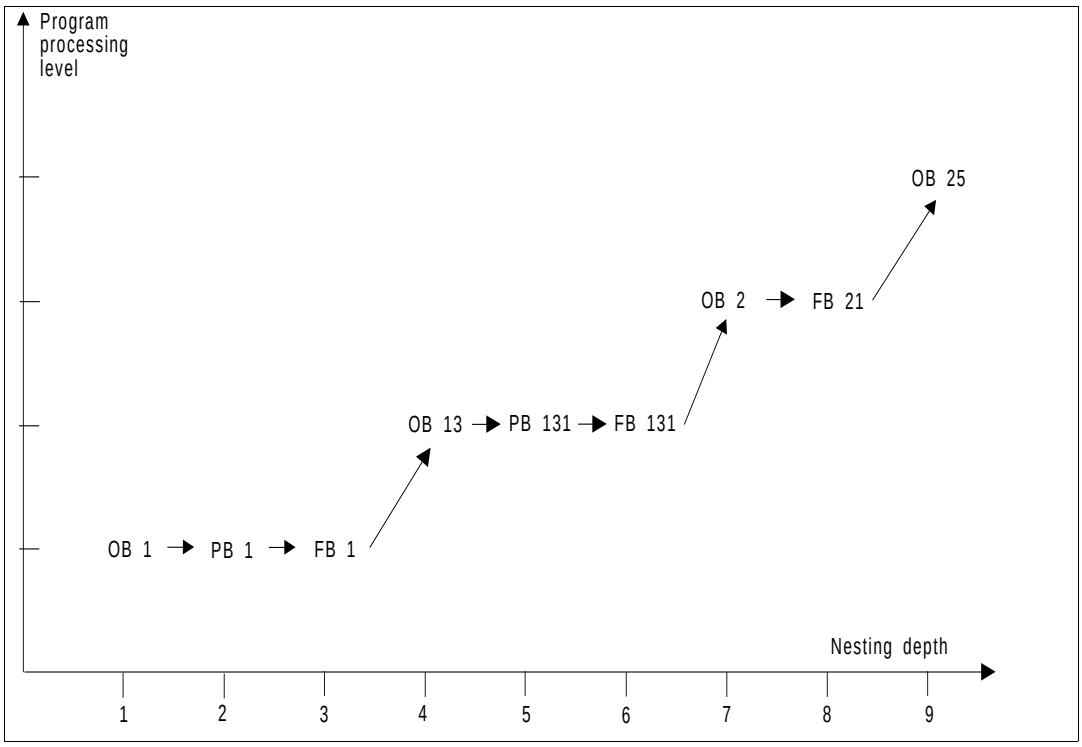



Fig. 3-5 Example of block nesting depth

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 948R, the user program runs solely in the internal RAM. The user program including data blocks must, therefore, be loaded in the CPU 948R 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 948R. 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. real time 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!

*Interprocessor communication (IPC) flags*

IPC flags exchange data between the CPU and some communications 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 communications processors.

You define the input and output IPC flags when you create data block DB 1 (STEP 5 screen on the PG).

*Interrupt events*

Cyclic program execution can be interrupted by the following:

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

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

- a program error
- operator intervention (using the PG function or STOP selector),
- a stop operation

## 3.5 STEP 5 Operations with Examples

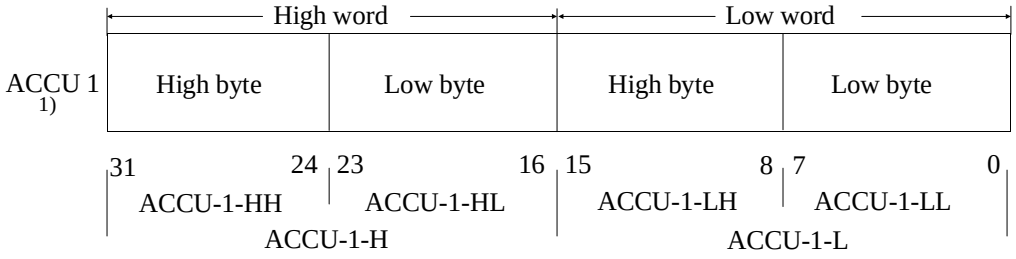
A STEP 5 operation consists of the operation and an operand. The operation specifies **what** the CPU is to do (operation). The operand specifies **with what** an operation is to be executed.

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 948R 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 ERAB = 1 is then set. While the remaining logic operations in the sequence are being performed, 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 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 $\overline{ERAB}$*

```
: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

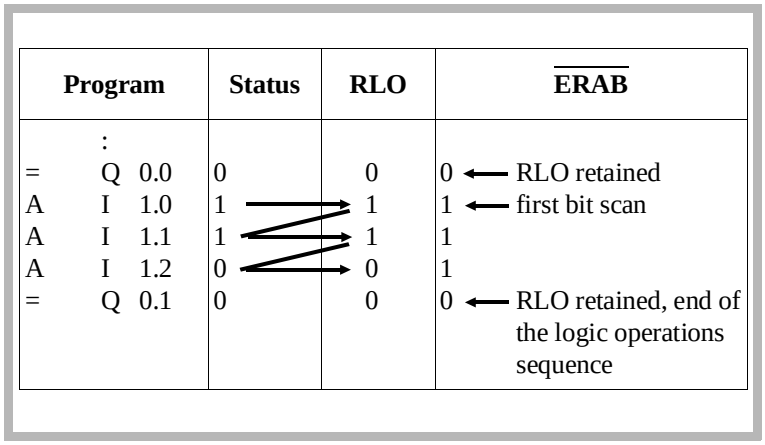
| Operation | Operand | Function |
|---|---|---|
| A | | AND logic operation after scanning for signal state "1" |
| O | | OR logic operation after scanning for signal state "1" |
| | I 0.0 to 127.7 | of an input in the PII |
| | Q 0.0 to 127.7 | of an output in the PIQ |
| | F 0.0 to 255.7 | of a flag 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<br>A(<br>O(<br>) | – | ANDing of expressions in parentheses<br>ORing of expressions in parentheses<br>Close parenthesis (to complete the bracketed expression)<br><br>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 ($\overline{\text{ERAB}} = 0$). Following this, the RLO can be evaluated but cannot be further combined.

*Example*




### *Set/reset operations*

Table 3-3 Set/reset operations

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

### *Load and transfer operations*

Table 3-4 Load and transfer operations/part 1

| Operation | Operand | | Function |
|---|---|---|---|
| L<br>T | | | Load<br>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 |
| | FY | 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] $\pm 0{,}1469368 \times 10^{-38}$ to $\pm 0{,}1701412 \times 10^{39}$

*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
```

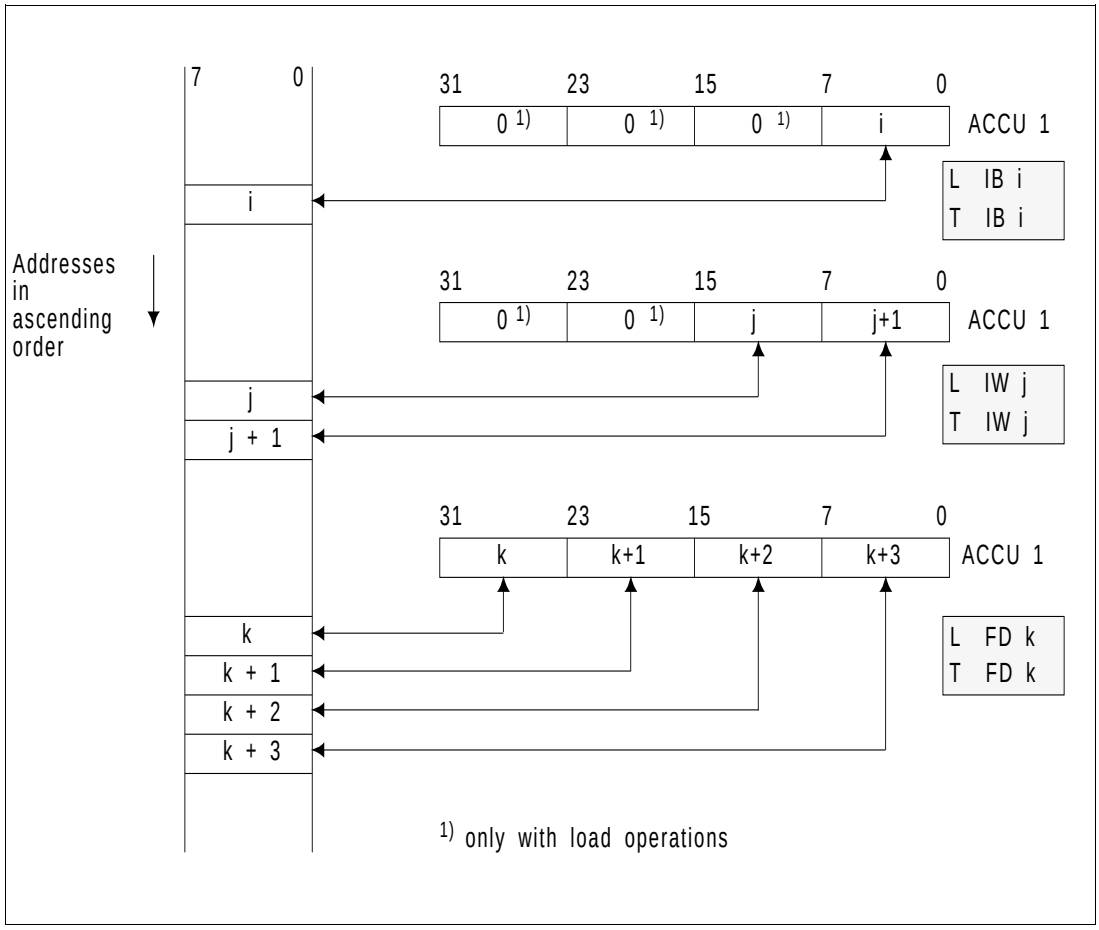



Fig. 3-6 Load and transfer operations in a byte-oriented memory area

**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
```

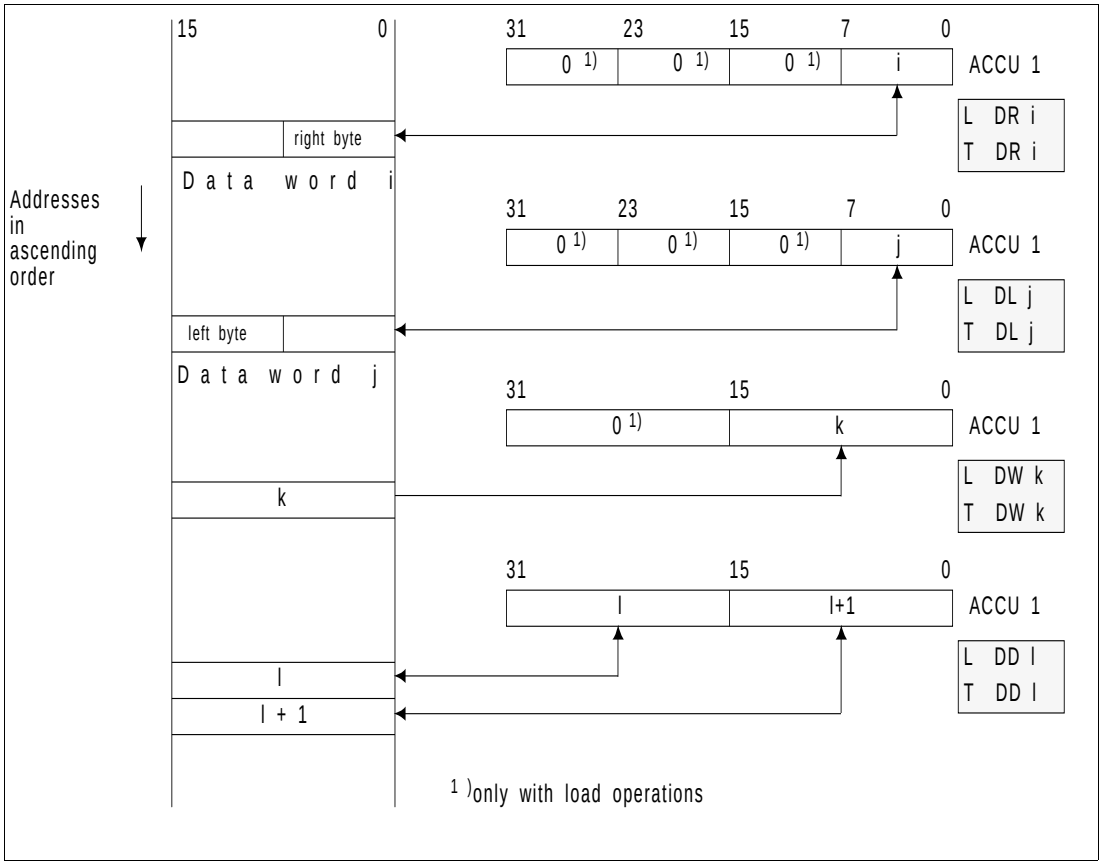



Fig. 3-7 Load and transfer operations in a word-oriented memory area

**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/Os:

- 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

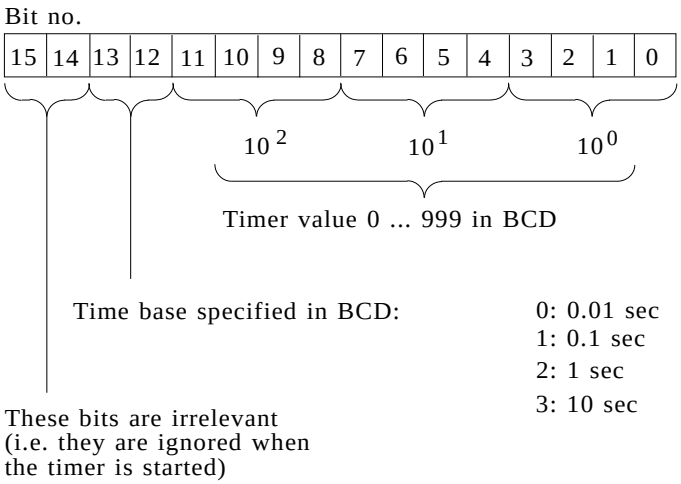
| Operation | Operand | RLO 1) | Function |
|---|---|---|---|
| SP | T 0 to 255 | ↑ | Start a timer as a pulse |
| SE | T 0 to 255 | ↑ | Start a timer as extended pulse |
| SD | T 0 to 255 | ↑ | Start a timer as ON delay |
| SS | T 0 to 255 | ↑ | Start a timer as stored ON delay |
| SF | T 0 to 255 | ↓ | 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*

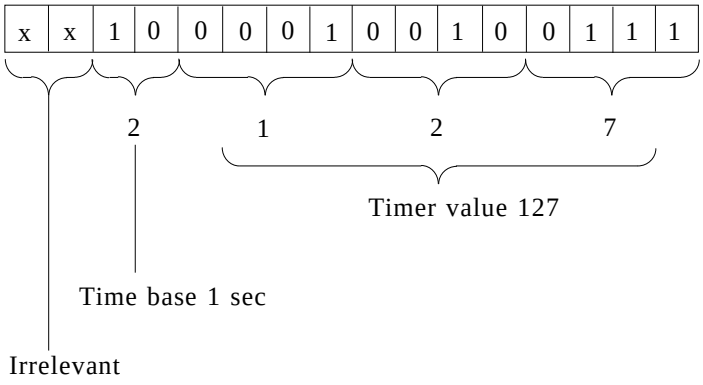
With the operation L KT, you can load a **timer value** directly into ACCU 1 or indirectly from a flag or data word. The value must have the following structure (with L KT, you specify the time base after the period in the operand as shown below):




*Example*

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

Bit assignment:
```



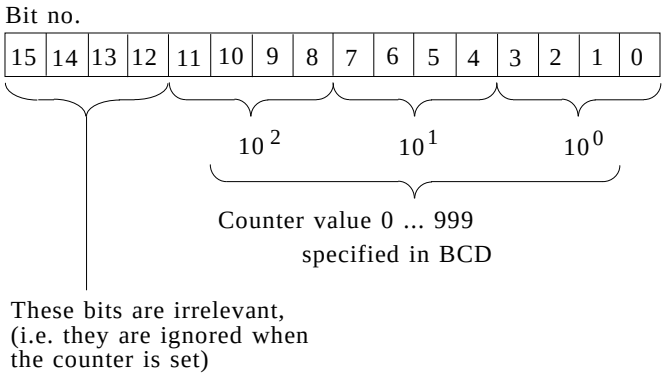

**Note**
The start of each timer is liable to an inaccuracy of 1 time base! When using timers, you should therefore select the smallest possible time base (time base < timer value):

**Example:**

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

*Counter value*

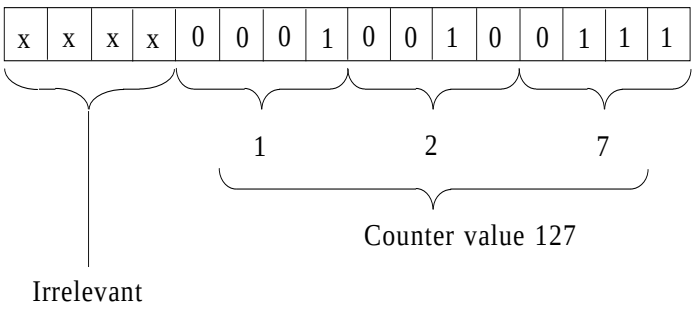
With the operation L KC, you can load a **counter value** directly in ACCU 1 or indirectly from a flag or a data word. The value must have the following structure:




*Example*

You want to specify a counter value of 127:

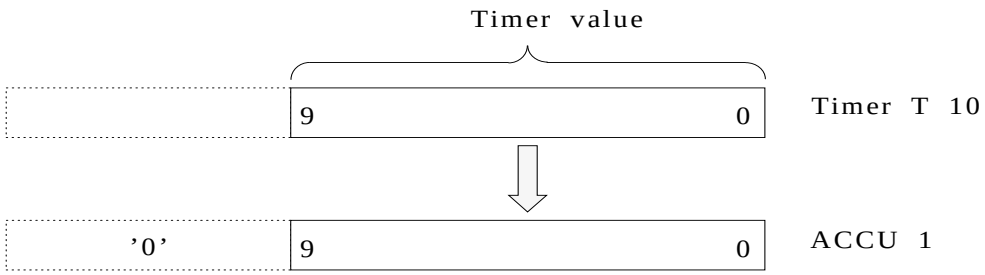
Bit assignment:




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*

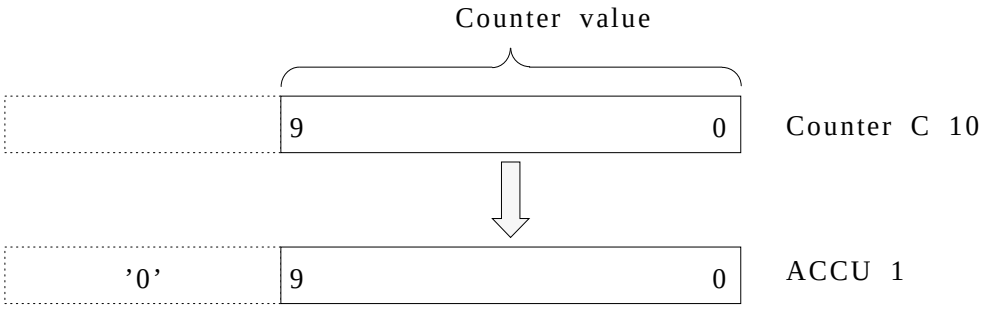
Loading **timer values** directly:




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

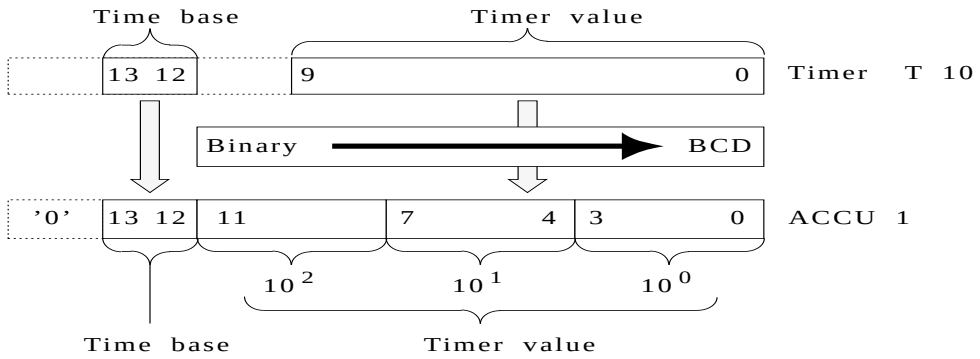
The time base is not loaded.

Loading **counter values** directly:




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

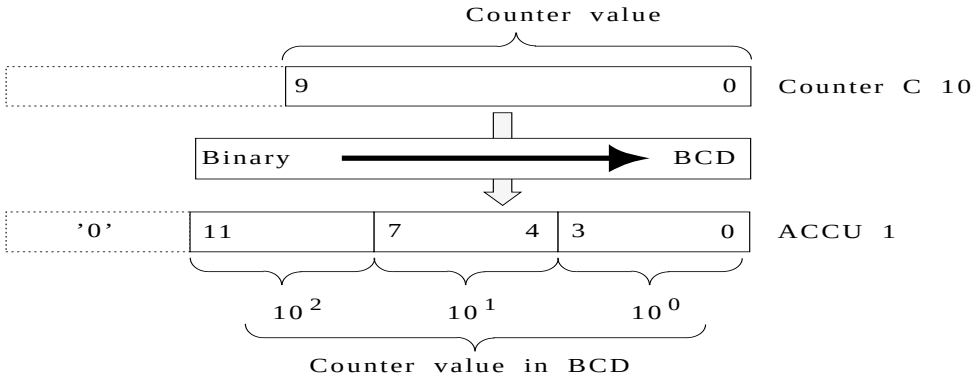
Loading **timer values** in BCD code:




"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

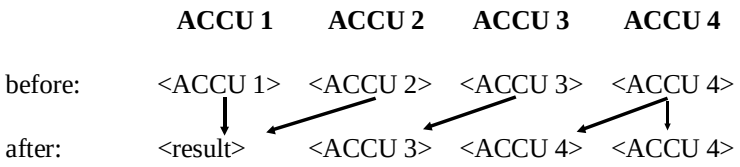
If you load values in BCD, status bits 14 and 15 of the timer or 12 to 15 of the counter are not loaded. They have the value 0 in ACCU 1. The value in the ACCU can now be processed further.

### *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):




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

### Block operations

Table 3-9 Block operations

| Operation | Operand | Function |
|---|---|---|
| JU<br>JC | <br><br><br>OB 1 to 39 [1)]<br>OB 121 to 255<br>PB 0 to 255<br>FB 0 to 255<br>SB 0 to 255 | Jump unconditionally<br>Jump conditionally (only when RLO = 1)<br><br>to an organization block<br>to a system program special function<br>to a program block<br>to an FB function block<br>to a sequence block |
| DOU<br>DOC | <br><br><br>FX 0 to 255 | Jump unconditionally<br>Jump conditionally (only when RLO = 1)<br><br>to an FX function block |
| BE<br>BEC<br>BEU | – | Block end<br>Block end, conditional (only when RLO = 1)<br>Block end, unconditional |
| C<br>CX | DB 2 to 255<br>DX 3 to 255 | Call a DB data block<br>Call a DX data block |
| G<br>GX | DB 2 to 255<br>DX 3 to 255 | Generate data block DB<br>Generate data block DX<br>(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 ($2 \leq x \leq 255$) 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 user memory and is otherwise the same as G DBx.

**Note**
The operations G DB/GX DX may only be executed during startup of the **master CPU**!
If you use these operations, note the information about the special functions OB 124, OB 125 and OB 254/255 at the start of Chapter 6

### *NOP/display/stop operations*

Table 3-10 NOP/display/stop operations

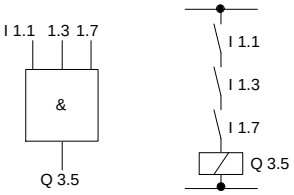
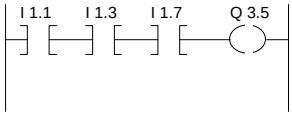
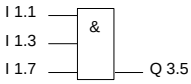
| Operation | Operand | Function |
|---|---|---|
| NOP 0<br>NOP 1 | – | No operation<br>No operation |
| BLD | 0 to 255<br><br><br>130<br>131/131/133<br>255 | Display generation operation for the PG:<br>the CPU handles the operation like a no operation<br><br>Create blank line with carriage return<br>Switch over between STL, CSF, LAD |
| STP | – | 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 |
|  | A I 1.1<br>A I 1.3<br>A I 1.7<br>= Q3.5 |  |  |







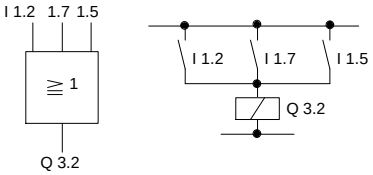
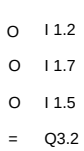
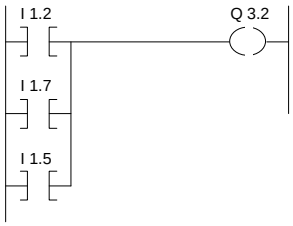
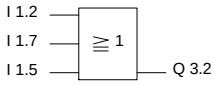
Output Q 3.5 is "1" when all inputs are "1" simultaneously

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 |
|  |  |  |  |

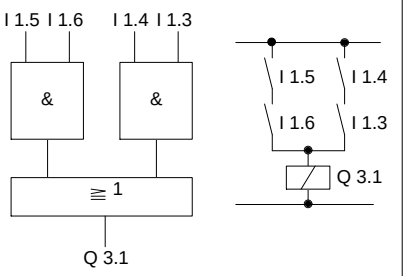
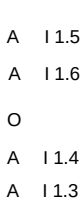
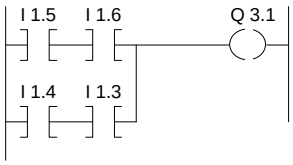
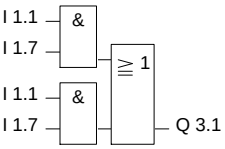


Output Q 3.2 is "1" when at least one of the inputs is "1"

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 |
|  |  |  |  |



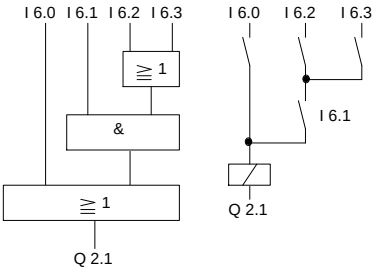
Q 3.1 is "1" when at least one AND condition is satisfied

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

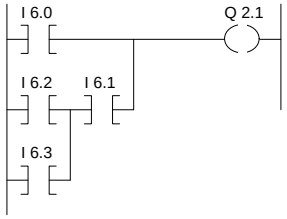
*Logic operations*
*(continued)*

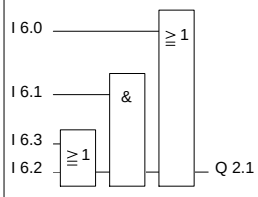
**OR-before-AND operation** /1st example

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




```
A   I 6.0
O
A   I 6.1
A (
O   I 6.2
O   I 6.3
)
=   Q2.1
```



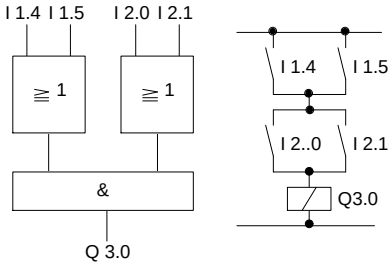




Output Q 2.1 is "1" when input I 6.0 or input I 6.1 and one of the inputs I 6.2 or I 6.3 has signal state "1"

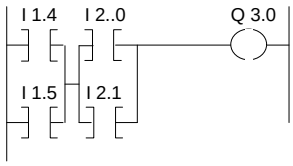
Output Q 2.1 is "0" when input I 6.0 has signal state "0" and the AND condition is not satisfied

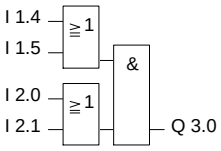
**OR-before-AND operation** /2nd example

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




```
A (
O   I 1.4
O   I 1.5
)
A (
O   I 2.0
O   I 2.1
)
=   Q3.0
```



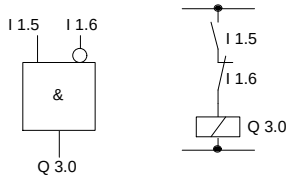
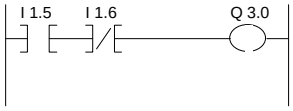
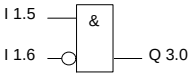




Output Q 3.0 is "1" when both OR conditions are satisifed

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

*Logic operations*
*(continued)*

**Scan for signal state "0"**

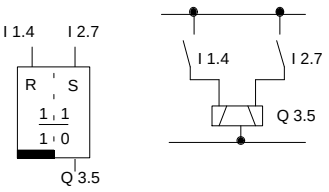
| Logical/circuit diagram | STEP 5 representation | | |
|---|---|---|---|
| | Statement list | Ladder diagram | Control system flowchart |
|  | A I 1.5<br>AN I 1.6<br>= Q3.0 |  |  |

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)

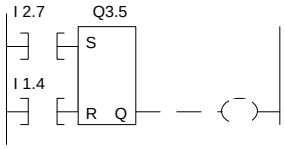
### *Set/reset operations*

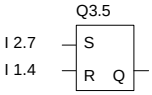
**RS flip-flop for a latching signal output**

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




```
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).

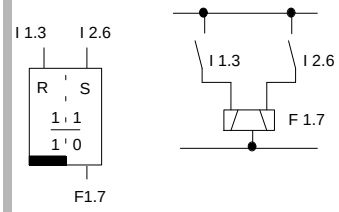
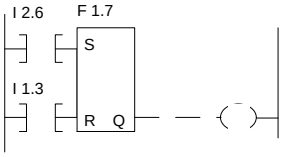
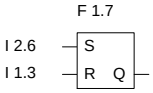
Signal state "1" at input I 1.4 resets the flip-flop (signal state "0" at output Q 3.5).

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 A I 1.4) remains in effect for the rest of the program (reset priority).

*Set/reset operations*
*(continued)*

**RS flip-flop with flags**

| Logical/circuit diagram | STEP 5 representation | | |
|---|---|---|---|
| | Statement list | Ladder diagram | Control system flowchart |
|  | A I 2.6<br>S F 1.7<br>A I 1.3<br>R F 1.7 |  |  |



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.

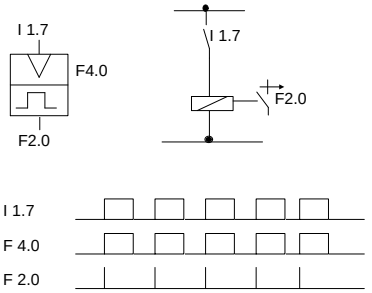
If the signal state at input I 1.3 changes to "0", the signal state of the flag is retained.

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 A I 1.3) remains in effect for the rest of the program (reset priority).

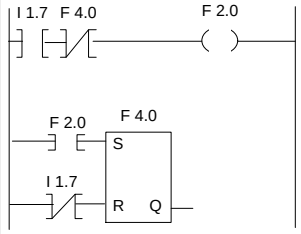
*Set/reset operations*
*(continued)*

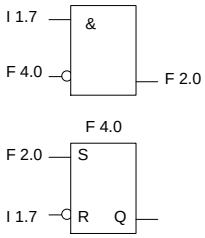
**Simulation of a momentary contact relay (one shot)**

| 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 (A I 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).

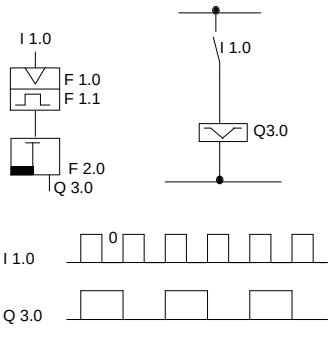
In the next processing cycle, the AND condition A I 1.7 and AN F 4.0 is not satisfied, since flag F 4.0 has already been set.

Flag F 2.0 is reset.

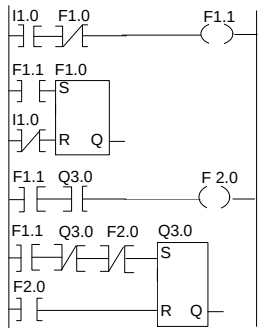
Flag F 2.0 therefore only remains "1" for one program run.

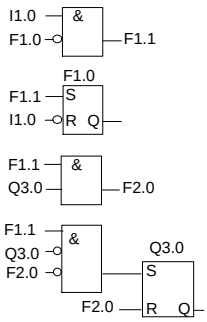
**Binary scaler (binary divider)**

| 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
```



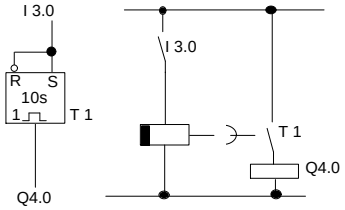




The binary scaler (output Q 3.2) changes its state each time input I 1.0 changes its signal state from 0 to 1 (leading edge). Therefore, only half the input frequency appears at the output of the memory cell.

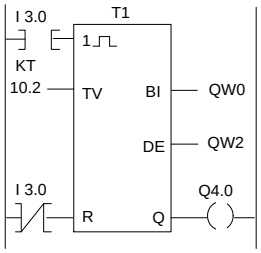
### *Timer operations*

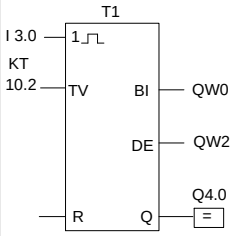
**Pulse timer**

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




```
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
```




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.

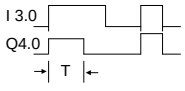
If the RLO is "0", the timer is reset (cleared).

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

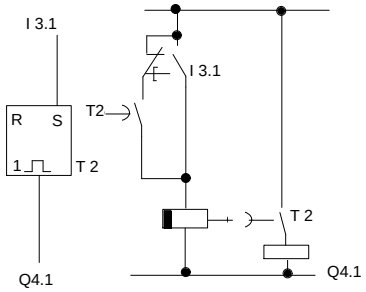
BI and DE are digital outputs of the timer. The time at output BI is in binary code. The time at DE is in BCD code with time base.



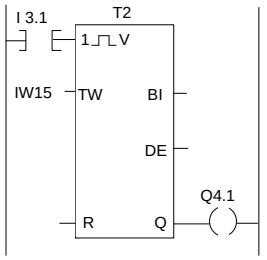
*Timer operations (continued)*

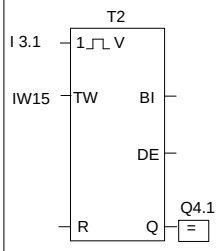
**Extended pulse timer**

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



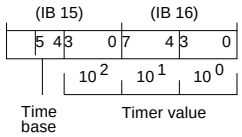

```
A     I   3.1
L     IW 15
SE    T   2
A     T   2
=     Q   4.1
```



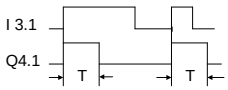

The timer is started during the first scan if the RLO is "1".

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.

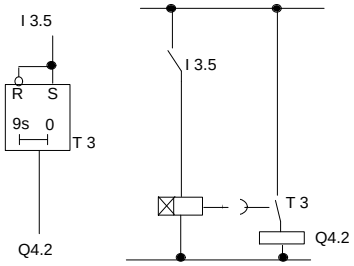
IW 15:
Set the timer with the value of the operand I, Q, F or D in BCD code (in this example, input word 15).



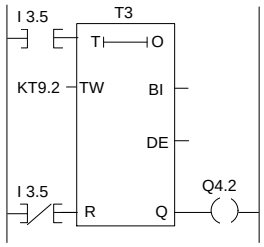
*Timer operations (continued)*

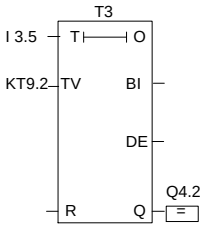
**ON-delay timer**

| 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.

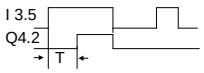
When the RLO is "0", the timer is reset (cleared).

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:

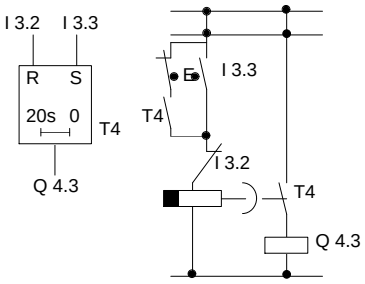
0 = 0.1sec  2 = 10 sec
1 = 0.1 sec  3 = 10 sec



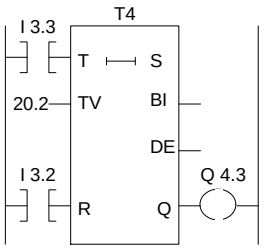
*Timer operations (continued)*

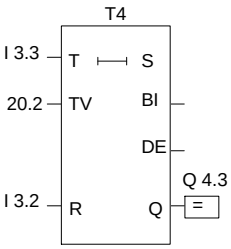
## Stored ON-delay timer

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

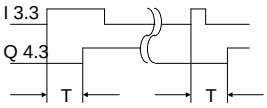



```
A   I   3.3
L   KT  20.2
SS  T   4
A   I   3.2
R   T   4
A   T   4
=   Q   4.3
```



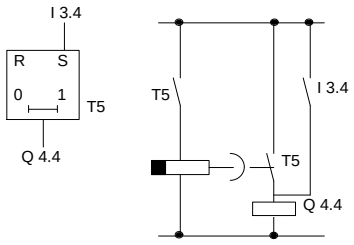




The timer is started during the first scan if the RLO is "1".
An RLO of "0" does not affect the timer.
The scan AT or OT produces the signal "1" when the timer has elapsed. The signal state does not change to "0" until the R T operation resets the timer.

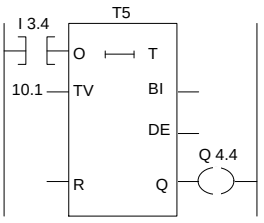


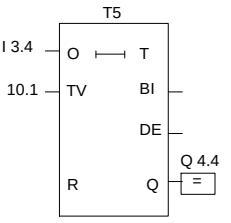

## OFF-delay timer

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

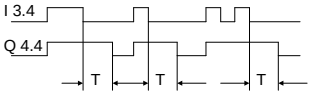



```
A   I   3.4
L   KT  10.1
SF  T   5
A   T   5
=   Q   4.3
```




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).

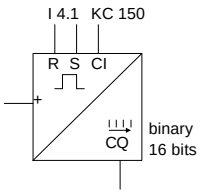



The scan AT or OT produces signal state "1" if the timer is running or the RLO at the input is "1".

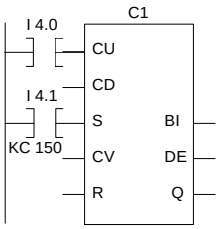
### *Counter operations*

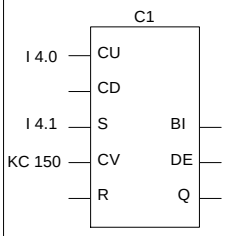
**Set counter**

| 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
```



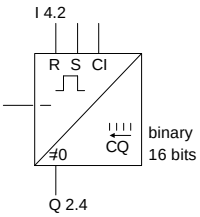




When the result of logic operation changes at the start input (I 4.1) from "0" to "1", the counter is loaded with the specified value (150).

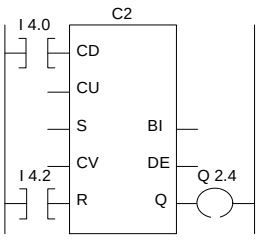
The flag necessary for edge evaluation of the set input is incorporated in the counter word.
BI and DE are digital outputs of the counter cell. The value at BI is in binary code and the value at DE is in BCD.

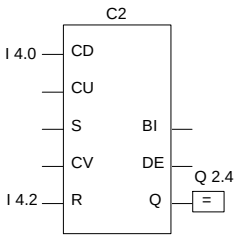
**Reset counter**

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




```
A    I    4.0
CD   C    2
A    I    4.2
R    C    2
A    C    2
=    Q    2.4
```



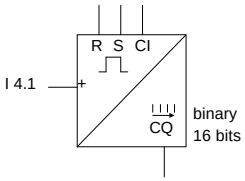

An RLO of "1" (I 4.2) resets the counter to zero.

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

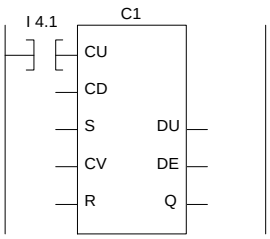
*Counter operations*
*(continued)*

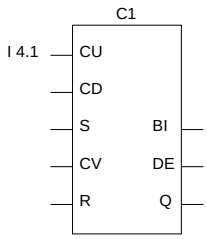
**Count up**

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




```
A   I   4.1
CU  C   1
```



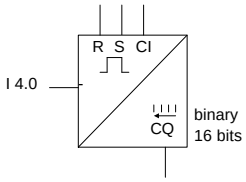

The value of the addressed counter is incremented by "1" to a maximum value of 999. The function CU is only executed on a positive edge (from "0" to "1") of the logic operation programmed before CU. The flags necessary for edge evaluation of the counter inputs are incorporated in the counter word.

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.

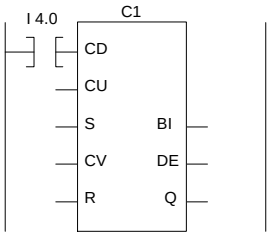
*Counter operations*
*(continued)*

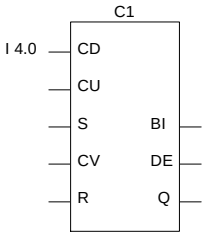
**Count down**

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




```
A   I   4.0
CD  C   1
```



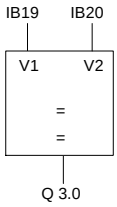
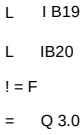
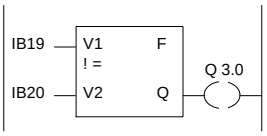
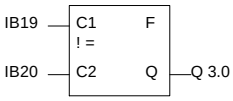




The value of the addressed counter is decremented by 1 to a maximum counter value of 0. The function is only executed on a positive edge (from "0" to "1") of the logic operation programmed before the CD. The flags necessary for edge evaluation of the counter inputs are incorporated in the counter word.

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 |
|  |  |  |  |





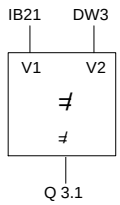
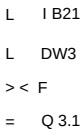
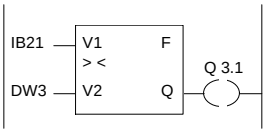
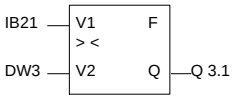




The first operand is compared with the second operand by the comparison operation. The RLO of the comparison is binary.
RLO = "1": comparison is satisfied if ACCU-1-L = ACCU-2-L
RLO = "0": comparison is not satisfied, when ACCU-1-L is not equal to ACCU-2-L.
The condition codes CC1 and CC0 are set as 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 |
|  |  |  |  |









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 Using Absolute Addresses".



**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 |
| | | 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 |
| | 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

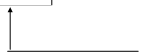
| Operation | Operand | Function |
|---|---|---|
| S = |  | Set a formal operand (binary) |
| RB = |  | Reset a formal operand (binary) |
| RD= |  | Reset a formal operand (digital) for timers and counters |
| = = |  | Assign the value of the RLO to a formal operand |
| |  | Insert formal operand |
|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

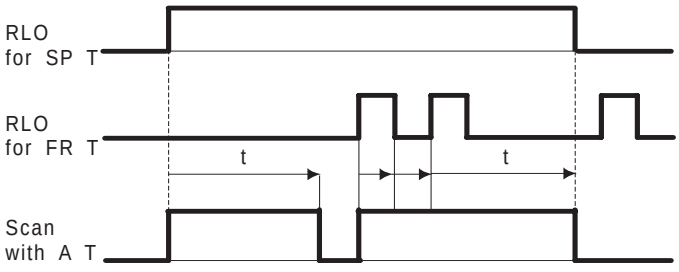
Table 3-15 Set and reset operations

| 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:<br>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:<br>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*

a)

| Function block call | Program in the function block | Program executed |
|---|---|---|
| :JU FB 203<br>NAME :EXAMPLE1<br>ANNA : I 10.3<br>BERT : T 17<br>JOHN : Q 18.4 | :A =ANNA<br>:L KT 010.2<br>:SSU =BERT<br>:A =BERT<br>:= =JOHN | :A I 10.3<br>:L KT 010.2<br>:SS T 17<br>:A T 17<br>:= Q 18.4 |

b)

| Function block call | Program in the function block | Program executed |
|---|---|---|
| :JU FB 204<br>NAME :EXAMPLE2<br>MAXI : I 10.5<br>IRMA : I 10.6<br>EVA : I 10.7<br>DORA : C 15<br>EMMA : F 58.3 | :A =MAXI<br>:SSU =DORA<br>:A =IRMA<br>:SFD =DORA<br>:A =EVA<br>:L KC 100<br>:SEC =DORA<br>:AN =DORA<br>:= =EMMA | :A I 10.5<br>:CU C 15<br>:A I 10.6<br>:CD C 15<br>:A I 10.7<br>:L KC 100<br>:S C 15<br>:AN C 15<br>:= F 58.3 |

c)

| Function block call | Program in the function block | Program executed |
|---|---|---|
| :JU FB 205<br>NAME :EXAMPLE3<br>BILL : I 10.4<br>JACK : T 18<br>EGON : IW 20<br>YOGI : F 100.7 | :A =BILL<br>:L =EGON<br>:SEC =JACK<br>:A =JACK<br>:= =YOGI | :A I 10.4<br>:L IW 20<br>:SE T 18<br>:A T 18<br>:= 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:<br>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:<br>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:<br>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:<br>The bit pattern of a formal operand is loaded into the ACCU (parameter type: D; data type: KG). |
| T = |  | Transfer to a formal operand:<br>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:<br><ACCU 4> := <ACCU 3><br><ACCU 3> := <ACCU 2><br><ACCU 2> := <ACCU 2><br><ACCU 1> := <ACCU 1><br>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<br><br>( addr =symbolic address with maximum 4 characters) | Jump unconditionally:<br>The jump is executed regardless of conditions |
| JC = | | | Jump conditionally:<br>the conditional jump is executed only if the RLO is 1.<br>If the RLO is 0, the statement is not executed and the RLO is set to 1. |
| JZ = | | | Jump if result is ’0’ :<br>the jump is executed only if CC 1 is 0 and CC 0 is 0.<br>The RLO is not changed. |

| Operation | | Operand | Function |
|---|---|---|---|
| Table 3-21 continued: | | | |
| | JN =<br><br><br><br><br>JP =<br><br><br><br><br>JM =<br><br><br><br><br>JO =<br><br><br><br><br>JOS = | addr<br><br>(addr = symbolic address with maximum 4 characters) | Jump if result is not 0 :<br>the jump is executed only if CC1 is not equal to CC0.<br>The RLO is not changed.<br><br>Jump if result > ’0’ :<br>the jump is only executed if CC 1 = 1 and CC 0 = O. The RLO is not changed.<br><br>Jump if result < ’0’:<br>the jump is only executed if CC 1 = 0 and CC 0 = 1.<br>The RLO is not changed.<br><br>Jump on overflow:<br>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.<br>An overflow occurs when an arithmetic operation exceeds the permissible range for a given numerical representation.<br><br>Jump when the OS (stored overflow) condition code is set:<br>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.<br>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.<br>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=<br>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
```

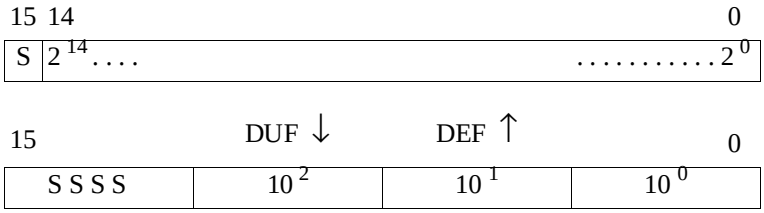
### *Conversion operations*

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.

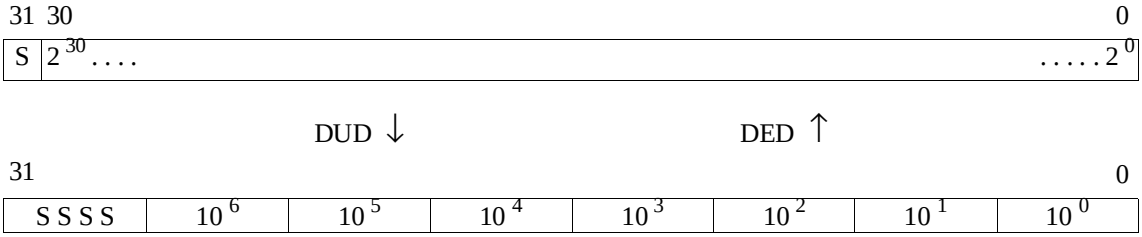
*DUF* The value in ACCU-1-L (bits 0 to 15) is interpreted as a 16-bit fixed point number. After the conversion, ACCU-1-L contains a BCD number.




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.

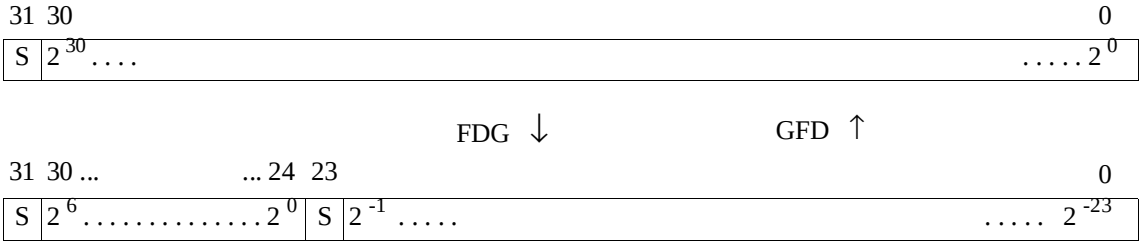
*DUD* The value in ACCU 1 (bits 0 to 31) is interpreted as a 32-bit fixed point number. After the conversion, ACCU 1 contains a BCD number.




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.




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 **GFD** | | 32-bit fixed point number |
|---|---|---|
| +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.

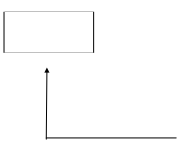
*Example*

```
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 14 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 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 |

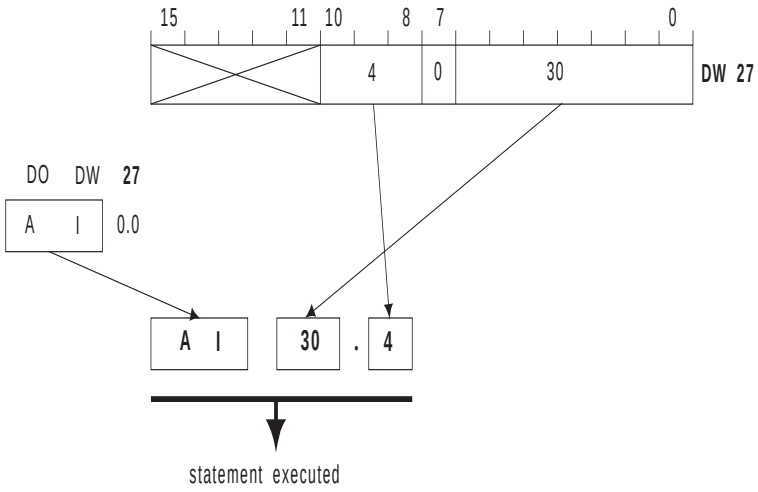
*Parameter word for S flags*

| 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**

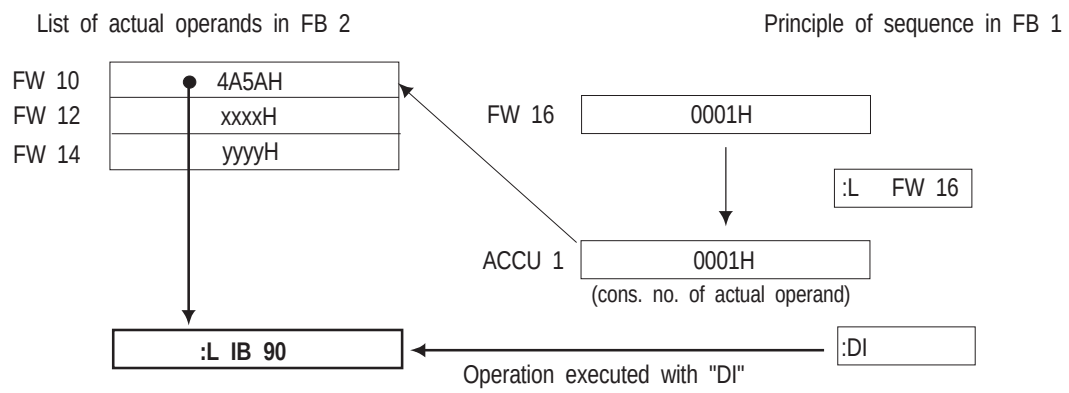
```
NAME  :TEST

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
```



### 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 |
| | | | Enable addressing error |
| | RAE | | Disable command output: PIQ is no longer influenced by the operations S Q, R Q, = Q, T PY and T PW. |
| | BAS | | Enable command output |
| | BAF | | |

"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 running of 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INTA | 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 |
| INTA | S5 bus/system interrupt A |
| 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, real time 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 |

# Operating Statuses and Program Execution Levels

# 4

## Contents of Chapter 4


4.1 Introduction and Overview . . . . . 4 - 4

4.2 Program Execution Levels . . . . . 4 - 7

4.3 STOP Mode . . . . . 4 - 12

4.3.1 SOFT STOP . . . . . 4 - 12
4.3.2 HARD STOP . . . . . 4 - 16
4.3.3 OVERALL RESET . . . . . 4 - 17

4.4 START-UP Mode . . . . . 4 - 19

4.4.1 MANUAL and AUTOMATIC COLD RESTART . . . . . 4 - 21
4.4.2 MANUAL and AUTOMATIC COLD RESTART WITH MEMORY . . . . . 4 - 22
4.4.3 Comparison between COLD RESTART and COLD RESTART WITH MEMORY . . . 4 - 25
4.4.4 User Interfaces for Start-Up . . . . . 4 - 26
4.4.5 Interruptions during START-UP . . . . . 4 - 27
4.4.6 Examples of Programs for START-UP . . . . . 4 - 29

4.5 RUN Mode . . . . . 4 - 31

4.5.1 Cyclic Program Execution . . . . . 4 - 33
4.5.2 Specifying Time and Interrupt-Driven Program Execution . . . . . 4 - 36
4.5.3 Time-Controlled Program Execution . . . . . 4 - 37
Delayed interrupt . . . . . 4 - 38
Real time interrupt . . . . . 4 - 39
Cyclic timed interrupts . . . . . 4 - 41
4.5.4 Interrupt-Driven Program Execution . . . . . 4 - 45
Disabling interrupt-driven processing . . . . . 4 - 47
Reaction time . . . . . 4 - 48
Program execution levels and flags . . . . . 4 - 48

# 4 Operating Statuses and Program Execution Levels

This chapter provides an overview of the operating statuses and program execution levels of the CPU 948R. 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 Introduction and Overview

With the CPU 948R, there are four operating statuses, as follows:

- **START-UP** mode
- **RUN** mode
- **SOFT STOP** mode
- **HARD STOP** mode

In the START-UP and RUN modes, certain events can occur to which the system program must react. In many situations, an organization block (one from OB 1 to OB 37) is called as the user interface for the specific event.

The modes are displayed by LEDs on the front panel of the CPU. Some of the modes must be activated using the operating elements on the front panel of the CPU. The position of the LEDs and operating elements can be seen in Fig. 4-1.

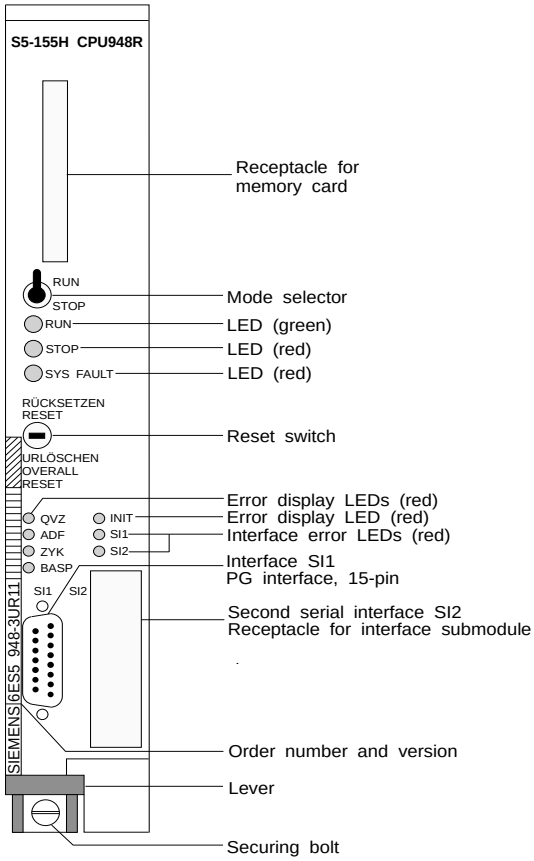



Fig. 4-1 Front panel of the CPU 948R with display and operating elements

### *Display of the modes by (LEDs)*

Various LEDs on the front panel of the CPU indicate its current mode. The following table shows the relationship between the STOP and RUN LEDs and SYSFAULT LED and the corresponding mode. These displays are supplemented by further LEDs (BASP, ADF, QVZ, ZYK, INIT, SI1, SI2).

Table 4-1 Meaning of the LEDs "RUN" , "STOP" and "SYSFAULT"

| LED | | | Meaning |
|---|---|---|---|
| **"RUN"** | **"STOP"** | **"SYS-FAULT"** | |
| **RUN or START-UP mode**: | | | |
| on | off | off | The CPU is in the RUN mode and is master (cyclic operation). |
| flashes | off | off | The CPU is in the RUN mode and is standby (cyclic operation). |
| on | flashes | off | The CPU is master, the parallel link has broken down. |
| on | on | on | Appears briefly after switching on. |
| on | on | off | The CPU is running through the start-up self-test. |
| off | off | off | The CPU is in the START-UP mode or the PG function "program test" is active. |
| **SOFT STOP mode**: | | | |
| off | on | off | Display when the power is switched on and the mode selector is set to STOP and no error occurred in the initialization.<br>A START-UP is possible. |
| off | flashes quickly | off | Display when OVERALL RESET was requested by the operator or by the CPU itself.<br>A START-UP is only possible if you go through an OVERALL RESET or remedy the error and then do the overall reset. |
| off | flashes slowly | off | This display occurs in the following situations:<br>- An error has occurred in the RUN mode, but no appropriate error reaction has been programmed (exceptions: see Chapter 5).<br>If you change the switch from RUN to STOP, the LED is once again lit permanently as long as the error does not occur again.<br>- Operator error, e.g. selecting an invalid type of start-up, DB 1/DX 0 error etc.<br>- When a stop statement is programmed (STP and STS).<br>- With the PG function "end program test".<br>- When a programming error or device fault occurs, (as an aid to locating the problem, the LEDs ADF, QVZ or ZYK light up with some of these errors). |

| LED | | | Meaning |
|---|---|---|---|
| "RUN" | "STOP" | "SYS-FAULT" | |
| on | flashes slowly | off | Defective parallel link. |
| Table 4.1 continued: | | | |
| **HARD STOP mode**: | | | |
| off | off | on | This display occurs in the following situations:<br>- With a timeout (QVZ) or parity error (PARE) in the system RAM.<br>- With ISTACK overflow.<br>- When the STW STEP 5 operation is programmed. |
| **"Coupling standby" mode**: | | | |
| flashes | flashes | off | The CPU is the standby and the mode is "coupling standby". |

### *Signalling and error LEDs*

*"BASP" LED* — Command output is disabled, the digital outputs are switched to the safe state (to ’0’).

*"QVZ" LED* — Timeout of an I/O module (for more detailed information refer to Section 5.6).

*"ADF" LED* — Addressing error; the user program has accessed an address in the process image for which there was no module inserted in the I/Os at the last cold restart or which was not entered in DB1.

*"ZYK" LED* — Cycle time exceeded.

*"INIT" LED* — During initialization following POWER UP and with system errors, this LED is lit **constantly**.

*"SI1" LED* — on: no communication possible; internal error
off: interface SI1 is initialized and operational

*"SI2" LED* — This LED is always off; otherwise there is a CPU error

## 4.2 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.

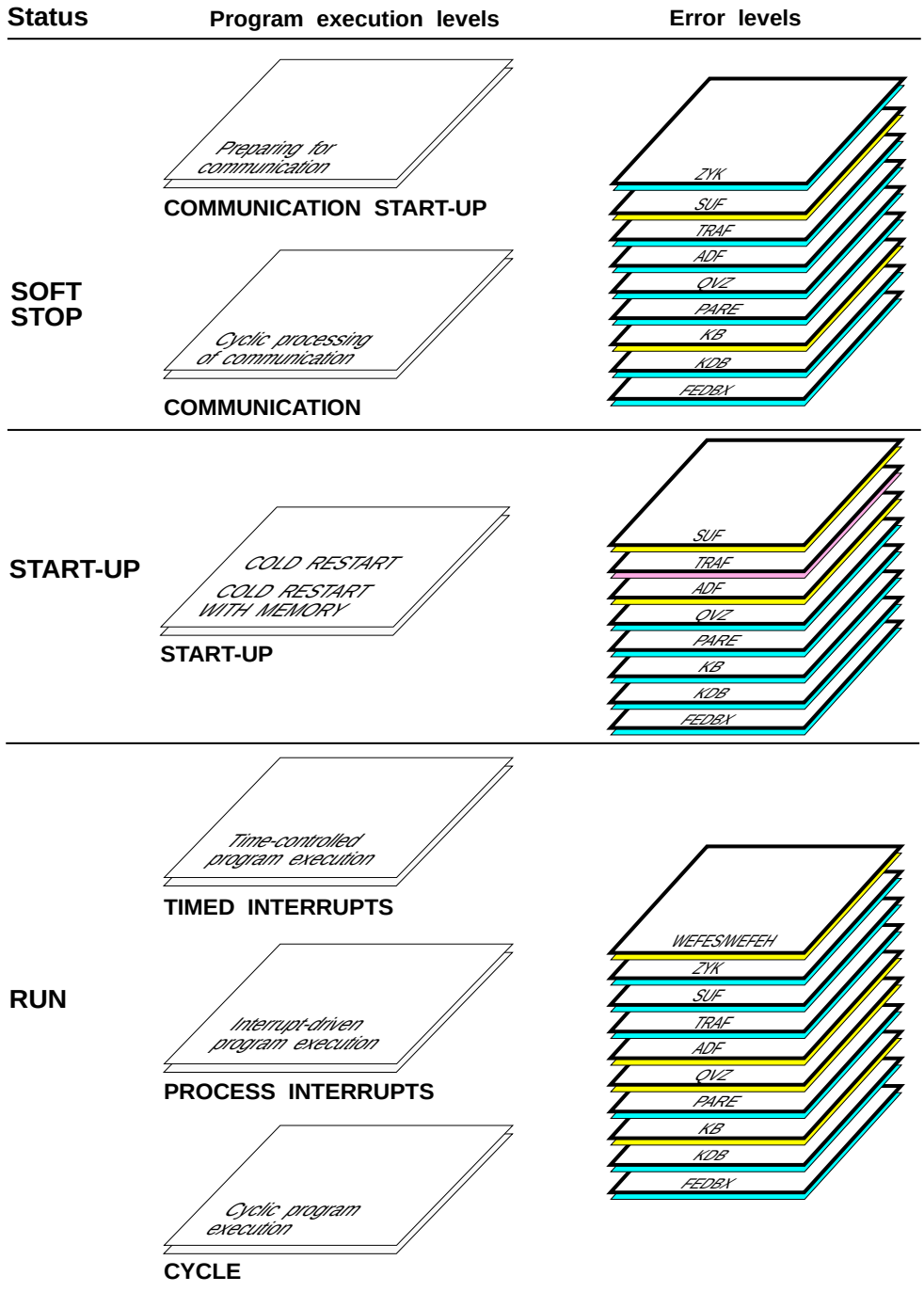



Fig. 4-2 Program execution levels

Table 4-2 Program execution levels

| Level | Meaning | Priority |
|---|---|---|
| **Error levels** | | |
| WEFES/WEFEH<br>ZYK<br>SUF<br>TRAF<br>ADF<br>QVZ<br>PARE<br>KB<br>KDB<br>FEDBX | Collision of timed interrupts<br>Cycle error<br>Substitution error<br>Transfer/load error<br>Addressing error<br>Timeout<br>Parity error<br>Called code block does not exist<br>Called data block does not exist<br>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<br>COMMUNICATION | Preparation (start-up) for communication<br>Cyclic processing of communication | – |
| **Program processing levels in START-UP** | | |
| COLD RESTART<br><br>COLD RESTART WITH MEMORY | Defined start of the user program<br><br>Defined start of the user program, certain data remain as they were before the START-UP | – |
| **Program execution levels in RUN** | | |
| TIMED INTERRUPTS<br>PROCESS INTERRUPTS<br>CYCLE | Time-controlled program execution<br>Interrupt-driven program execution<br>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 boundaries

**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 real time 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<br>cyclic timed interrupt, shortest period<br>...<br>...<br>cyclic timed interrupt, longest period<br>real time 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":**

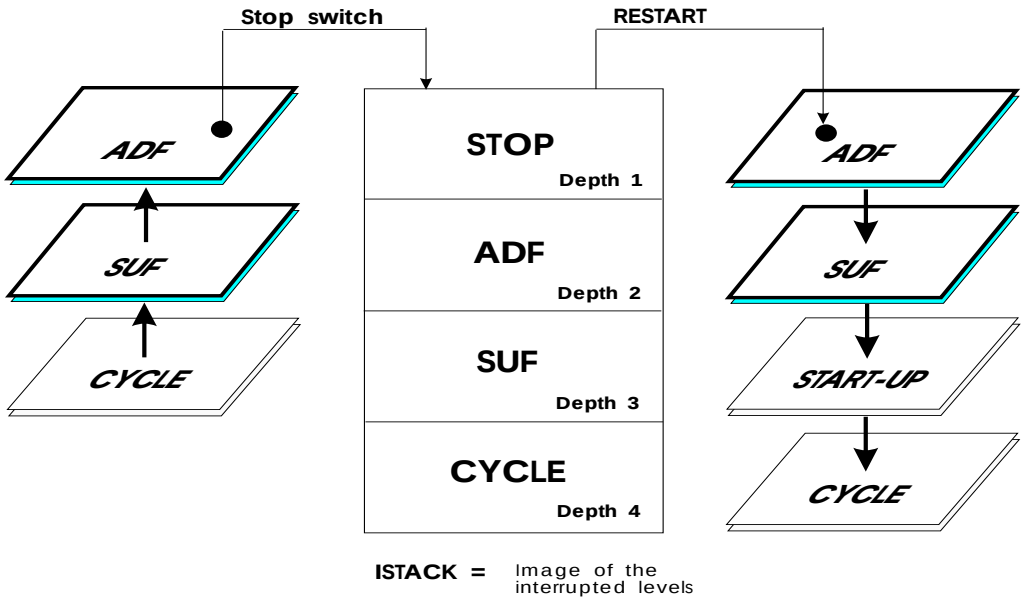



Fig. 4-3 Principle of changing level and the ISTACK

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.

## 4.3 STOP Mode

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

### 4.3.1 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 RESTART/AUTOMATIC COLD RESTART WITH MEMORY) 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).

*Timer processing in OB 38/39*

While OB 38/39 is running, timers and counters are not processed. If necessary, timers must then be processed with OB 121 or OB 150.

*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 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: with the PG function "force outputs", BASP is not active).

*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.

.

*OB 38/OB 39 call*

Figures 4-4 and 4-5 illustrate the principle of the OB 38 and OB 39 calls.

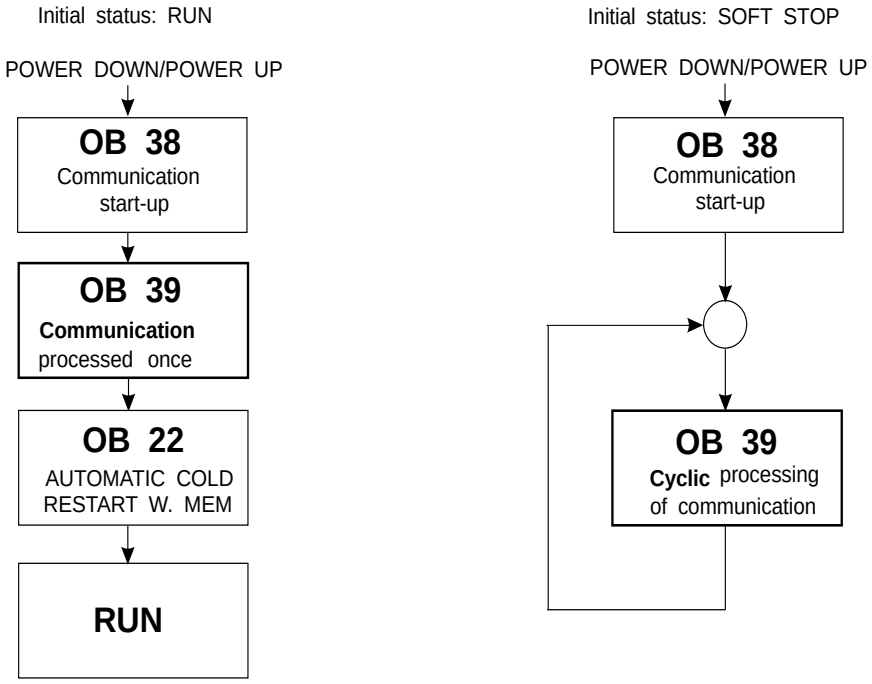



Fig. 4-4 Program execution after POWER UP

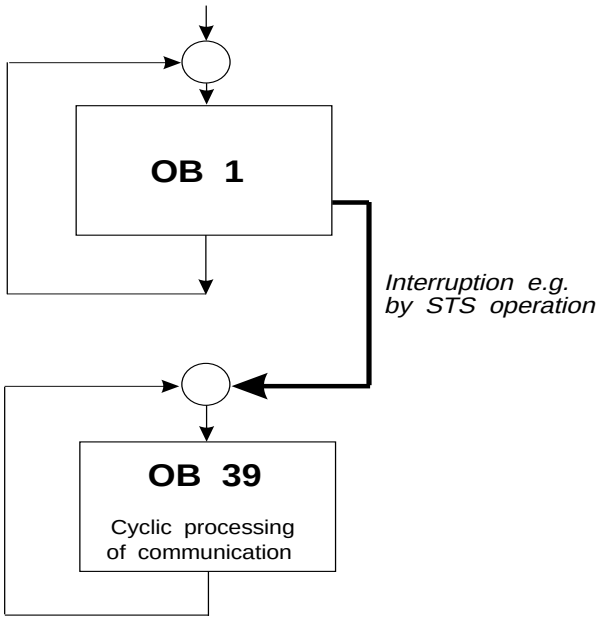



Fig. 4-5 Program execution after a cycle interruption

### *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 with the 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:

- by changing the mode selector from RUN to STOP,
- by the PG function PLC STOP,
- following an OVERALL RESET.

*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.

### 4.3.2 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 |

### 4.3.3 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 mode 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

### *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 all blocks". | An OVERALL RESET is performed. The STOP LED is lit continuously. |

**Note**
If you do an OVERALL RESET on the CPU 948R from the PG, only the **connected** CPU is reset. In this case, you must change the CPU to the STOP mode before the overall reset. The other CPU remains in the RUN mode.
If the connected CPU is **not** in the STOP mode, the error message "Memory error in PLC" appears on the PG screen.

During the OVERALL RESET, all the LEDs are off apart from the INIT and BASP LEDs. During the overall reset self-test, the RUN and STO LEDs are lit continuously.

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.

## 4.4 START-UP Mode

**Note**
The information about the START-UP applies to both CPUs. The standby CPU, however, takes all the status data from the master CPU before entering the CYCLE.

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 948R has the following start-up modes:

- COLD RESTART ( manual or automatic)
- COLD RESTART WITH MEMORY (manual or automatic)

(You can select the type of start-up with the 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.

*COLD RESTART WITH MEMORY*

The same as COLD RESTART, but flags, timers, counters and process images are not deleted.

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

- **OB 21** for a MANUAL COLD RESTART WITH MEMORY,
- **OB 22** for an AUTOMATIC COLD RESTART WITH MEMORY.

*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!

*BASP*

The BASP signal (disable output command) is active. This disables all the digital outputs:

*Interrupts*

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

*Start-up self-test*

With all types of start-up, the system program runs through a complete self-test. If an error is detected, the CPU of the subunit affected changes to the SOFT STOP and outputs an error message. During the start-up self-test, both the RUN and STOP LEDs are lit continuously.

*Subunit assignment*

During a COLD RESTART, the system program checks whether I/Os are plugged in which are specifically assigned to subunit A or B (one-sided I/Os). If this is the case, the assignment subunit A - subunit B is derived from this. If there are no such I/Os, subunit A is the subunit in which the IM 324R is plugged in. If the IM 324R module is defective, this becomes subunit B.

If no one-sided I/Os are plugged in, the subunit assignment is only re-determined after the next OVERALL RESET.

*Master-standby assignment*

If the self-test is completed successfully, the subunit signals that it is operational via the parallel link IM 324R. The system program then checks whether the partner PLC is already operational. If it is, the subunit becomes standby. If the partner PLC is not yet operational, the subunit becomes master.
When there is a cold restart on the master, the passivation is cancelled. The error data block is completely deleted.

*Start-up phase of the master CPU*

So that serious software and hardware configuration errors can be detected and indicated before the standby is coupled, the COM 155H configuration must be compared with the actual hardware configuration. If, for example, the I/Os were configured as two-channel and is plugged in as switching, the master CPU has access to its I/Os configured as two-channel. It must, however, establish whether or not it would have access as standby CPU. If the master CPU recognizes that the I/Os configured as two-channel are plugged in as switching, it signals the error and switches the I/Os passive

During START-UP, the system program of the master CPU sets the IM 314R to the "Test" mode and then immediately to "bus access". The "test and bus access" mode is retained until the complete start-up OB (OB 20, OB 21 or OB 22) has been processed. The system program then switches off the test mode on the IM 314R.

*START-UP without standby*

If the standby is not operational due to an error in the START-UP, the master PLC starts without a standby. This mode is the same as the solo mode.

*Generating the process image list during START-UP*

During START-UP, the process image list for digital inputs and outputs is created according to the data configured with COM 155H, regardless of which modules are actually plugged in.

### 4.4.1 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"
  inconsistent data are detected in the user memory after a COLD RESTART WITH MEMORY

*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

- "AUTOMATIC COLD RESTART after POWER UP" is set in DX 0 **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 (e.g. after an addressing error), then **AUTOMATIC** COLD RESTART is **not permitted**. The STOP mode can then only be exited 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.

### 4.4.2 MANUAL and AUTOMATIC COLD RESTART WITH MEMORY

*When is a COLD RESTART WITH MEMORY possible and permitted?*

A MANUAL COLD RESTART WITH MEMORY is **only possible** after stoppages caused by the following:

- the mode selector was changed from the RUN position to the STOP position,
- 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 COLD RESTART WITH MEMORY** is possible. The system program will only permit a COLD RESTART (without memory).

A MANUAL or AUTOMATIC COLD RESTART WITH MEMORY is **only permitted** when

- the user program was not modified during the stop mode

  and

- a COLD RESTART (without memory) is not necessary for other reasons (refer to Section 4.4.1).

*MANUAL COLD RESTART WITH MEMORY*

You **trigger** a MANUAL COLD RESTART WITH MEMORY 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 COLD RESTART WITH MEMORY*

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

With POWER UP, when

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

- the mode selector on the CPU is 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 (without memory) 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 COLD RESTART WITH MEMORY.

When the power returns in a central controller and there is a power failure on an expansion unit, (PEU signal), the CPU remains in the STOP mode. It remains in this mode until the PEU signal is switched inactive (however only for a maximum of 12 seconds) and then performs an AUTOMATIC COLD RESTART WITH MEMORY or an AUTOMATIC COLD RESTART. The subunit that starts up as standby goes through a COLD RESTART.

**Note**
With a COLD RESTART WITH MEMORY note the following special situations:

- STOP during error processing:
  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 COLD RESTART WITH MEMORY 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 **COLD RESTART WITH MEMORY** is executed.
  - If the error OB sets the CPU to the stop mode, then only a **COLD RESTART** (without memory) is possible.

- Both subunits change to STOP due to POWER OFF, HALT, stop switch or PG-STP:

  Reaction of the PLCs

  If POWER UP occurs simultaneously for both subunits (difference <50 ms), the system program makes sure that the previous master (with up-to-date process data) becomes master again. If, for any reason, the previous master does not restart, the previous standby becomes master and enters a message in the error DB.

*Aborting a COLD RESTART WITH MEMORY*

You can only abort a COLD RESTART WITH MEMORY 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 COLD RESTART WITH MEMORY is then possible.

### 4.4.3 Comparison between COLD RESTART and COLD RESTART WITH MEMORY

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

Table 4-3 Differences between COLD RESTART and COLD RESTART WITH MEMORY

| System program function | COLD RESTART | COLD RESTART WITH MEMORY |
|---|---|---|
| **Evaluation of**: | | |
| - DB 1 | yes | no |
| - DX 0 | yes | no |
| - DX 1 (create address lists for updating process images) | yes | no |
| **Create block address list in DB 0** | yes | no |
| **Delete**: | | |
| - ISTACK/BSTACK | yes | yes |
| - Process image of inputs | yes | no |
| - Process image of outputs | yes | no |
| - Digital outputs (2 * 128 bytes) | yes | no |
| - Analog outputs (2 * 128 bytes) | yes | no |
| - IPC flags | yes | no |
| - F and S flags | yes | no |
| - Timers and counters | yes | no |
| - Error DB | yes | no |
| **Cancel passive** | yes | no |
| **Start-up self-test** | yes | yes |
| **Display of restart type on PG in ISTACK control bits** | NEUDF | WIEDF |

### 4.4.4 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 WITH MEMORY, the system program calls OB 21 once if it is loaded. Here, you can store a STEP 5 program which executes preliminary steps for **resuming** the cyclic program.

*MANUAL COLD RESTART WITH MEMORY*

After processing OB 21, the system program executes a **COLD RESTART WITH MEMORY**, (the CPU continues program execution with the **first STEP 5 statement in OB 1**). The signal states of the flags, IPC flags and the block address list (DB 0) **are retained**.

***OB 22***

When the CPU executes an AUTOMATIC COLD RESTART WITH MEMORY, the system program calls OB 22 once if it is loaded. Here, you can store a STEP 5 program which executes preliminary steps (generally following a power failure) for **resuming** cyclic program execution.

*AUTOMATIC COLD RESTART WITH MEMORY*

After processing OB 22, the system program executes a **COLD RESTART WITH MEMORY**, (the CPU continues program execution with the **first STEP 5 statement in OB 1**). The signal states of the flags, IPC flags and the block address list (DB 0) **are retained**.

### 4.4.5 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-STP

  or

- Program errors or device faults (refer to Section 5.7).

*Basic rules for an interrupted START-UP*

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

| |
|---|
| 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.<br>Example: 1. POWER DOWN (NAU) in the cycle<br>2. Switch set to STOP<br>3. POWER UP<br>4. Switch set to RUN<br>Reaction: the CPU executes a MANUAL COLD RESTART WITH MEMORY |
| An interrupted COLD RESTART cannot be continued with a COLD RESTART WITH MEMORY, but must be repeated. |
| Following an interrupted COLD RESTART WITH MEMORY, both a COLD RESTART and a new COLD RESTART WITH MEMORY are possible. |

*Response of the CPU on the return of power after power failure*

If the start-up is interrupted by a power failure, 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).<br><br>No "**remaining start-up**" is processed, (OB 20 or OB 21 is not resumed). |
| | MANUAL COLD RESTART WITH MEMORY | OB 21 | |
| | AUTOMATIC COLD RESTART | OB 20 | |
| AUTOMATIC COLD RESTART WITH MEMORY | MANUAL COLD RESTART | OB 20 | STOP |
| | MANUAL COLD RESTART WITH MEMORY | OB 21 | An AUTOMATIC COLD RESTART WITH MEMORY is executed (OB 22 is called and processed from the beginning);<br>no "**remaining start-up**" is processed (OB 21 or OB 22 is not resumed). |
| | AUTOMATIC COLD RESTART WITH MEMORY | OB 22 | |

### 4.4.6 Examples of Programs for START-UP

This section contains simple examples of programs for H system applications, The sequences of operations can be programmed in the start-up OBs.

*Direct access to I/Os*

Switched I/Os must **only** be accessed directly by **the master** and must therefore be programmed in the start-up OBs OB 20, OB 21 and OB 22. Evaluating the H flag word (status byte) in the STEP 5 program shows whether the subunit is currently master or standby. This decides whether or not I/Os are accessed.

1. **Access to switched I/Os:**

```
          :
          :AN   F x.2
          :JC   =M001
          :T    PY 30       Access to switched I/O
  M001    :
          :
```

2. **Access to *one-sided I/Os* (subunit A):**

```
          :
          :AN   F x.4
          :JC   =M001
          :T    PY 34       Access to subunit A I/O
  M001    :
          :
```

3. **Access to *one-sided I/Os* (subunit B):**

```
          :
          :A    F x.4
          :JC   =M001
          :T    PY 36       Access to subunit B I/O
  M001    :
          :
```

4. **Access to *redundant I/Os*:**

```
  No scan necessary here!
```

*Generating/deleting/ copying/duplicating blocks*

Blocks can be generated or deleted using the operations G DB/GX DX or with the special functions OB 124, OB 125, OB 254 or OB 255 **only during START-UP and only by the master**. The example below shows how a sequence of block operations is skipped if the CPU is not the master. If the application requires such a sequence, it must be programmed in the start-up organization blocks.

```
Generate, delete blocks etc. only when the CPU is master:

          :
          :AN   F x.2      Not master?
          :JC   =RESE
          :
          :               Include operations
          :               to generate, delete
          :               and copy/duplicate
          :               blocks here
  RESE    :
          :
```

## 4.5 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.

When executing the user program on the S5-155H, apart from the normal cyclic processing of the process images in the CPUs of the subunits, other system functions are also executed for synchronization, exchange and comparison of data and, in some situations. also operating modes.

*Event-synchronized user program execution*

The same user programs are run in the two subunits (master and standby) of the S5-155H. The master and standby operate **event-synchronized**. The 155H system program ensures that both subunits operate with the same data.

*Timers, counters, process image*

All the timers and counters started in the program continue to run.

*Process image of the inputs (PII)*

At the start of every cycle, the master and standby read the process images of the one-sided and redundant inputs assigned to them. The process image of the switched inputs is only read by the master. Both subunits then exchange the complete process image of the inputs.

*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*

The execution of the user program is interrupt-driven (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 real time interrupt).

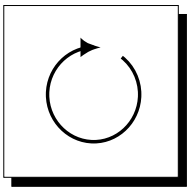
The execution levels differ from each other as follows:

- 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 948R can be programmed simultaneously. The levels are called by the system program according to the preset priority (refer to Section 4.2).

### 4.5.1 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 following figure illustrates the principle of cyclic program execution particularly the interaction between the subunits of the S5-155H.

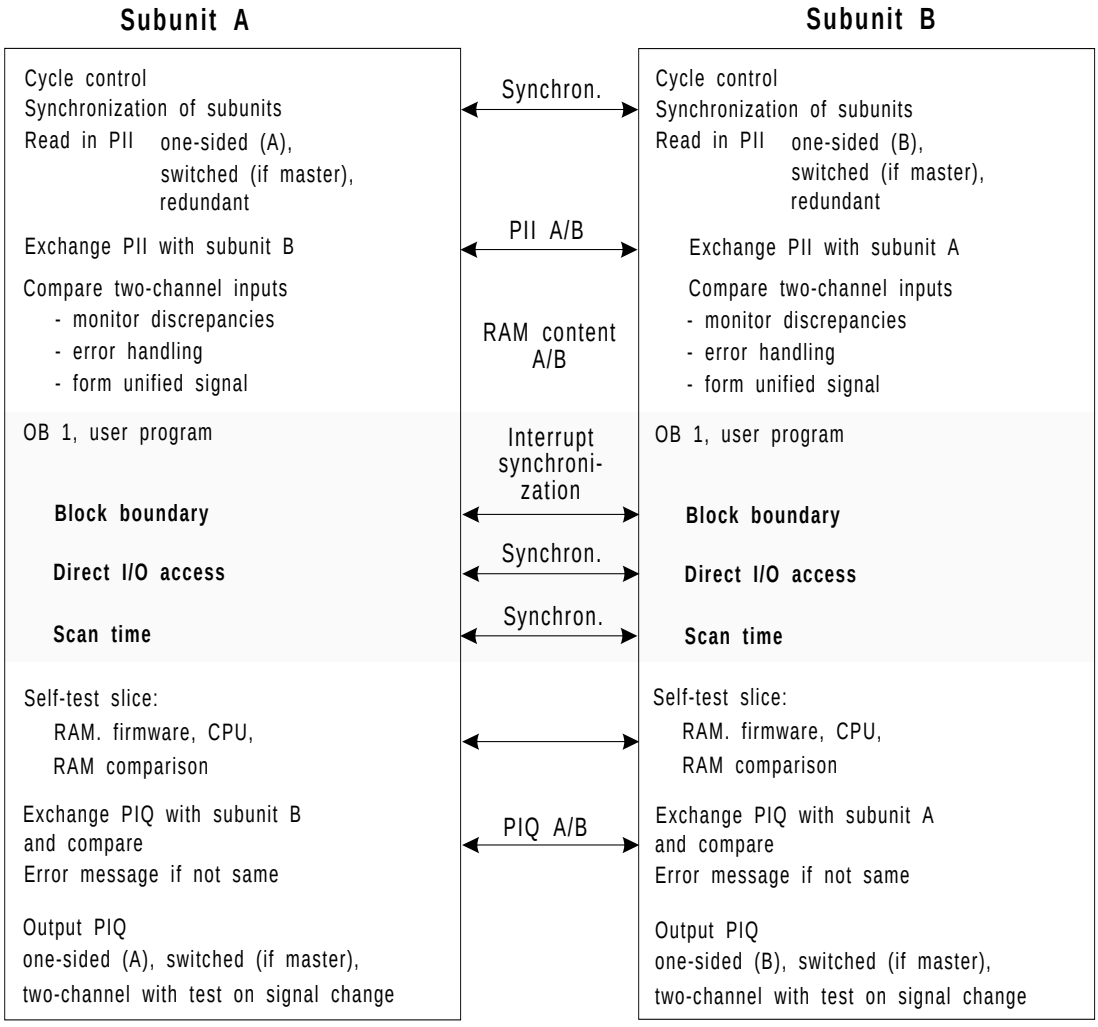



Fig. 4-6 Cyclic program execution with the S5-155H

*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 at **block boundaries** by the following:

- time-controlled program execution
- process interrupt-driven program execution via input byte IB 0

Cyclic program execution can be interrupted or **aborted** as follows:

- when a device fault or program error occurs (at operation boundaries,
- by operator intervention:
  - stop switch, (at operation boundaries),
  - PG function (at checkpoints - refer to Chapter 10),
- by the stop command STS (at operation boundaries),
- by a power failure on the central controller (at operation boundaries).

*Special situation: operation without connected EU*

If the IM 314R module is not plugged in (no switched EU) or if it is faulty and the IM 324R of subunit A is also faulty, both subunits change to the RUN mode and become master.
The green RUN LEDs are lit continuously. The red STOP LEDs flash slowly.

Since this mode is not permitted, if you are operating an S5-155H without an IM 314R, you must set one of the subunits to STOP. If a bus-switchable EU with IM 314R is configured and exists, the system program 155H detects this illegal mode and changes the standby PLC to STOP.

*Example of a program*

```
No bus-switchable expansion unit (EU)is configured or exists.

-  A one-sided DQ (Q y.z) must be connected to one-sided DI (I y.z).

-  Function block FB nnn is called in OB 1.



FB nnn
SEGMENT 1         0000
NAME:CONNECT

0005       :A    F x.2                      am master and operating alone
0006       :                                (evaluation of H flag word)
0007       :=    Q x.y                      one-sided DQ in subunit A
0008       :A    F x.2
0009       :A    I x.y                      one-sided DI in subunit B
000A       :JC   =STS
000B       :BEU
000C  STS  :STS
000D       :BE

The LEDs on the front panel provide information about the mode of the
subunit:

• Subunit is "master", the parallel link is intact
   - RUN LED:   on
   - STOP LED:  off

• Subunit is "master", the parallel link is defective
   - RUN LED:   on
   - STOP LED:  flashes slowly

• Subunit is "standby" in the redundant mode
   - RUN LED:   flashes slowly
   - STOP LED:  off
```

### 4.5.2 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:

- the delayed interrupt (processed by OB 6) **cannot** be used,
- the real time interrupt (processing by OB 9) **cannot** be used.

*"System interrupts" mode*

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

- delayed interrupts are processed by OB 6,
- real time interrupts are processed by OB 9.

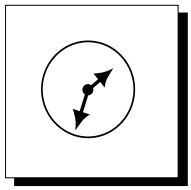
*Interrupting time and interrupt-driven program execution*

Time and interrupt-driven program execution can be interrupted or **aborted** as follows:

- when a device fault or program error occurs (at operation boundaries,
- by operator intervention:
  - stop switch, (at operation boundaries),
  - PG function (at checkpoints - refer to Chapter 10),
- by the stop command STS (at operation boundaries),
- by a power failure on the central controller or in the expansion unit (at operation boundaries).

### 4.5.3 Time-Controlled Program Execution

This type of program execution includes the delayed interrupt, the real time 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.

*Real time interrupt*

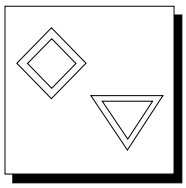
Triggered at a selected interval or once at an absolute point in time. Organization block OB 9 is called with this interrupt.

*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 | ↑ |
| 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 | ascending |
| cyclic timed int., period 5 | OB 14 | priority |
| 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 | |
| real time interrupt | OB 9 | |

**Note**
Real time 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 948R, small time intervals with a resolution of 5 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 6.13). 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.

In the CPU 948R, OB 6 can only be called at block boundaries. This means that **the time of the longest running block** may be added to the selected delay time.

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 occurring 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.

### *Real time interrupt*

The CPU 948R 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, real time 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 real time 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 real time interrupt, "process interrupts via IB 0 = off" must be set in data block DX 0.

#### *User interface OB 9*

For a real time 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*

As 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 real time 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 real time interrupt is only processed in the RUN mode. real time interrupts occurring in the STOP mode, when there is a **power failure** or during START-UP are **discarded**.
- Once a real time 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 real time interrupt, i.e. call OB 151 with new time values, a previously set real time interrupt is cancelled. A real time interrupt currently being processed is continued. This means that only **one** real time interrupt is valid at any one time.
- If a real time interrupt occurs without the previous one being completely processed, the new real time interrupt is discarded. **Real time 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 real time interrupts.

### *Cyclic timed interrupts*

On the CPU 948R, 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 4-4 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 or COLD RESTART WITH MEMORY. 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 948R there are two different types of collisions of timed interrupts:

| Type of error/cause | ISTACK ID | Reaction of the CPU |
|---|---|---|
| Timed interrupt queue overflow:<br>- there are more than three timed interrupts pending for one of the three shortest time bases (OB 10 to 12),<br>- 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**. |
| Due to the interrupts being 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 4-5 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. |

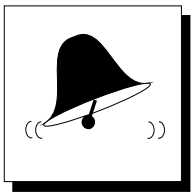
After processing OB 33, the program is resumed at the interrupted timed interrupt OB.

**Note**
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.

### 4.5.4 Interrupt-Driven Program Execution

The following interrupt-driven program execution is possible with the CPU 948R:



- **PROCESS INTERRUPTS**
  via input byte IB 0 (max. 8 interrupts),

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 real time 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.

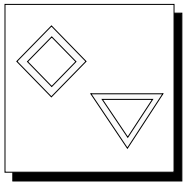
Table 4-6 User interfaces for process interrupts

| 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 |
| I 0.3 | OB 5 | priority |
| 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.

### *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:

- 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.

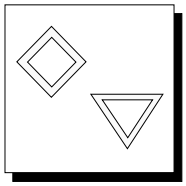
  Between these two operations, no process interrupt-driven program execution is permitted. The program section between these two operations **cannot** be interrupted.

- '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.

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. 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 and error OBs 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


5.1 Frequent Errors in the User Program . . . . . 5 - 4

5.2 Error Information . . . . . 5 - 5

5.3 Procedure for Error Analysis . . . . . 5 - 9

5.4 Control Bits and Interrupt Stack . . . . . 5 - 10

5.4.1 Control Bits . . . . . 5 - 11
5.4.2 ISTACK Content . . . . . 5 - 15
5.4.3 Example of Error Diagnosis using the ISTACK . . . . . 5 - 20

5.5 Error Data Block (Error DB) . . . . . 5 - 21

5.5.1 What is Entered in the Error DB? . . . . . 5 - 21
5.5.2 Structure of the Error DB . . . . . 5 - 22
5.5.3 Calling a User Interface . . . . . 5 - 26

5.6 H Flag Word . . . . . 5 - 27

5.7 Error Handling Using Organization Blocks . . . . . 5 - 29

5.8 Causes of Error and Reactions of the CPU . . . . . 5 - 32

5.8.1 OB 19: Calling a Logic Block That Is Not Loaded (KB) . . . . . 5 - 33
5.8.2 OB 19: Opening a Data Block That Is Not Loaded (KDB) . . . . . 5 - 33
5.8.3 OB 23/24, OB 28/29: Timeout Error (QVZ) . . . . . 5 - 34
5.8.4 OB 25: Addressing Error (ADF) . . . . . 5 - 35
5.8.5 OB 26: Cycle Time Exceeded Error (ZYK) . . . . . 5 - 36
5.8.6 OB 27: (Substitution Error SUF) . . . . . 5 - 37
5.8.7 OB 30: Parity Error and Timeout Error in the User Memory (PARE) . . . . . 5 - 37
5.8.8 OB 32: Load and Transfer Error (TRAF) . . . . . 5 - 38
5.8.9 OB 33: Collision of Timed Interrupts Error (WEFES/WEFEH) . . . . . 5 - 39
5.8.10 OB 34: Error with G DB/GX DX (FEDBX) . . . . . 5 - 41
5.8.11 OB 37: Entry in the Error DB . . . . . 5 - 41

5.9 Self-Test . . . . . 5 - 43
5.9.1 Overview . . . . . 5 - 43
5.9.2 Organization of the Self-Test . . . . . 5 - 44

# 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 948R.

## 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
- Error DB
- H flag word

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 3.1.1 "Block nesting") 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 containing the block that **called** the incorrect block.

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 of this block in the program memory (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.
```

*Error DB*

The system program enters all errors detected during the self-test and during program execution in the error DB along with an error class, the error number and the current date. The error DB also contains a static error image of all I/O addresses.

During system configuration with COM 155H, you specify a DB number for this data block (DX data block numbers are not allowed).

Using the COM 155H function "Diagnosis", you can output the error messages in the error DB in plain language, with the "Output DB" PG function you can read out the errors on the PG or evaluate them using your STEP 5 program.

*H flag word*

The H flag word provides you with important information about the status of your PLC in its "status "byte". Using its control byte, you can make certain settings with your STEP 5 program.

You specify the number of the H flag word with COM 155H.

## 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 | Output the content of the error DB as plain language using the DIAGNOSIS function of COM 155 H. |
| 5 | Correct the errors in the ascending order so that secondary errors occurring later do not make error location more difficult. |

**Note**
Correcting the error in step 5 of the table above is important with the S5-155H for the following reason:
Some errors would cause a CPU STOP in an S5-155U, but only cause an error message in the S5-155H because of the increased availability. Such errors are, however, serious and must nevertheless be corrected.

## 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 (BGRUN) | 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:<br>a) Selector switch on the coordinator (COR) is in the STOP position<br>or<br>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<br>(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** |
| NEUDF | 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 (BGRUN) | 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 | $\overline{\text{ERAB}}$ |
| | STATUS X | RLO X | | | | |
| CAUSE OF INTERR.: | KB | KDB | TRAF | SUF | STUEB | STUEU |
| | NAU | QVZ | ADF | PARE | ZYK X | STOP |
| | 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:<br>Contains machine code (first word of the instruction processed last in an interrupted program processing level. |
| SAC (new) | STEP address counter (new):<br>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 | Start addresses of the actual operands of a function block in the calling block. |
| BLK-STP | Block stack (BSTACK) pointer:<br>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 |
| DB-NO. | Number of the data block currently opened |

| Information about the error | |
|---|---|
| **ISTACK ID** | **Meaning** |
| Table 5-5 continued | |
| OB-NO. (depending on type OB, PB ...) | Block type and number of the last **calling** block |
| REL-SAC | Relative STEP address counter:<br>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):<br>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:<br>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:<br>"KEx abc" where<br>x = 1 to 7 levels,<br>a = ’OR’ (refer to bit codes),<br>b =’ RLO’ (result of logic operation, refer to bit codes),<br>c = 1: ’A(’,<br>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):<br>Processed STEP 5 operation cannot be substituted |
| STUEB | Block stack overflow:<br>Nesting depth too great; required action:<br>COLD RESTART |
| STUEU | Interrupt stack overflow:<br>Nesting depth too great; required action:<br>POWER DOWN, POWER UP,<br>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) |
| WEFEH | Collision of timed interrupts caused by the hardware clock (OB 33):<br>timed interrupt clock was masked (ignored)<br>for too long |

| CAUSE OF INTERR. | |
|---|---|
| **ISTACK ID** | **Meaning (called error OB)** |
| Table 5-6 continued: | |
| PEU | I/Os not ready = power failure in expansion unit (the CPU does not go into STOP):<br>After a statically pending PEU signal is removed, error message no. 17 "I/Os ready again" is entered in the error OB. |
| HALT | Not relevant for the CPU 948R |

### 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.
- The program processing level interrupt is then activated and **OB 3** called.
- A timed interrupt means that the INTERRUPT level is exited and the TIMED INTERRUPT level activated and **OB 13** processed.
- Incorrect addressing leads to the activation 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.

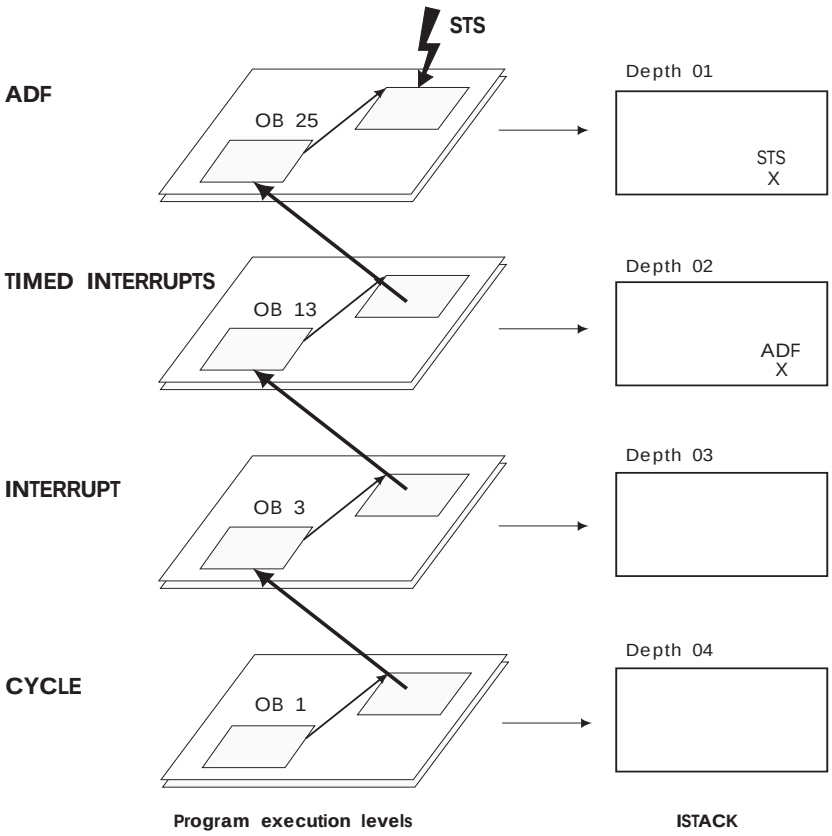



Fig. 5-3 Example of evaluating the ISTACK

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 Data Block (Error DB )

During the system configuration with COM 155H, you specify the number of an error DB. You can use numbers from 3 to 255. During START-UP, the system creates a data block **DB** (not DX) with a standard length of 2K words for error entries.

### 5.5.1 What is Entered in the Error DB?

The following entries are particularly important if errors occur:

*Error image*

This is a static error image organized so that every repairable unit (CPU, parallel link, I/O modules, CP, IP) is assigned its individual bit. These bits are arranged according to continuous addresses or interface numbers. A further distinction is also made between units belonging to subunit A and/or subunit B or the switched I/Os.

All the bits of the error image are preset to ’0’. If the system program recognizes an error during the self-test (see Section 5.9) or while the program is running, it sets the corresponding bit to ’1’.

*Status word*

In addition to the entry in the error image, each error is assigned to a particular error group (e.g. "parallel link error", "I/O error"). These groups are entered in the **status word**: one group error bit is reserved for every group.

Each group error bit is set to ’1’ as long as there is one error entered for the particular group.

*Error fields*

The system program creates an **error field** consisting of 7 words for every error detected. Normally, every detected error is entered in the data block **once**.

Each error field contains among other things a **time stamp**. This consists of the current time in one of two selectable formats. When configuring, you can, however, also have the content of a selected "**H flag double word**" entered as the time stamp instead of the current time (see COM 155H).

**Note**
The content of the error DB is **deleted completely**

- when the master goes through a COLD RESTART,
- when you cancel the passive setting by setting the appropriate bit in the H flag word.

## 5.5.2 Structure of the Error DB

This section describes the structure of the error DB and the various entry fields.

*Structure of the error DB*

Fig. 5-4 shows the global structure of the error DB.

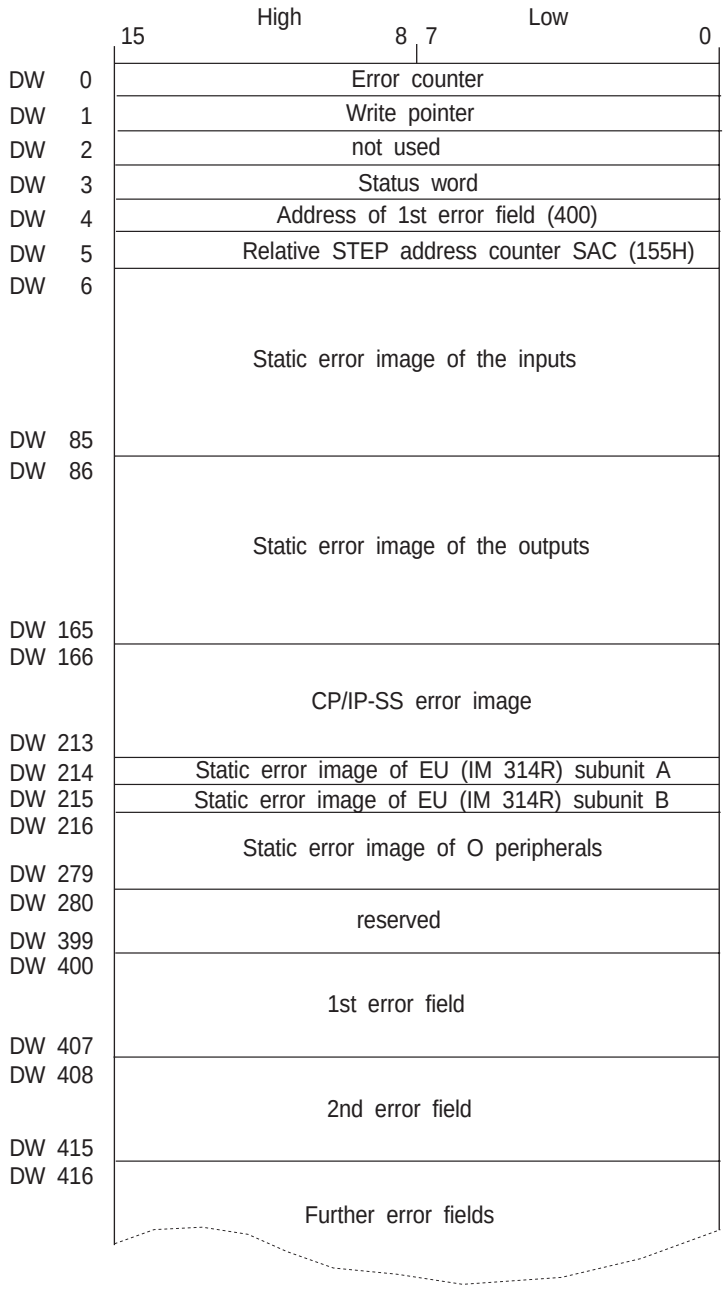



Fig. 5-4 Structure of the error DB

### *Meaning and structure of the entries in the error DB*

The following section describes the various entries in the error DB.

*Error counter (DW 0)*

The errors are counted here. Once the 32767 errors is reached, the counter stops. The counter is reset by a COLD RESTART or when the passive setting is cancelled.

*Write pointer (DW 1)*

The write pointer always points to the start (the first data word number) of the current error field. The last error to occur is entered in the current error field. As long as there is no error in the error DB, the write pointer has the value ’0’.

*DW 2*

Data word DW 2 is not used by the system program and is available for user entries.

*Status word (DW 3)*

H system errors can be divided into certain error groups. Each bit in the status word is assigned to such an error group. It is changed from ’0’ to ’1’ when the first error belonging to the group is entered in the error image and remains set to ’1’ as long as at least one error belonging to the group is present.

Table 5- 7 explains the meaning of the individual bits.

Table 5-7 Group error bits in the status word of the error DB

| Bit no. | Abbrev. | Meaning |
|---|---|---|
| 0 | – | not used |
| 1 | CPU | Group error bit "CPU error" |
| 2 | – | not used |
| 3 | ANW | Group error bit "error in user memory" (memory card) |
| 4 | PK | Group error bit "parallel link error" (IM 304 – IM 324R)" |
| 5 | PEB | Group error bit "I/O bus error" (IM 304 – IM 314R) |
| 6 | PER | Group error bit "I/O error" |
| 7 | CP | Group error bit "CP/IP error" (page area) |
| 8 | PRJ | Group error bit "configuration error" |
| 9 | HAN | Group error bit "handling error" |
| 10 | SYS | Group error bit "system error" |
| 11 | HW | Group error bit "hardware fault" (e.g. defect CPU) |
| 12 | GER | Group error bit "device error" (e.g. POWER DOWN on EU) |
| 13 | MLD | message |
| 14 | – | not used |
| 15 | UMK | circulate ID |

The circulate ID is set when all the error fields of the data block have been filled and the entries start again at the first error field. These earliest errors are then overwritten and can no longer be output.

*Address of the first error field (DW 4)*

This data word contains the start address of the first error field in the error DB: Data word number "400".

*Relative SAC (DW 5 – only for maintenance specialists)*

The data word contains the relative address (relative to the block start address) of the next operation to be executed in the last system program OB processed (e.g. I/O test OB). The last error entered was detected at this point.

*Static error image of the inputs (DW 6 to DW 85)*

Bit no. 0 in DW 6 corresponds to DI 0, bit no. 1 to DI 1 etc.

| DW no. | Meaning |
|---|---|
| 6 to 13 | Static error image DI 0 to 127, subunit A |
| 14 to 21 | Static error image AI 128 to 255, subunit A |
| 22 to 29 | Static error image DI 0 to 127, subunit B |
| 30 to 37 | Static error image AI 128 to 255, subunit B |
| 38 to 45 | Static error image DI 0 to 127, switched |
| 46 to 53 | Static error image AI 128 to 255, switched |
| 54 to 69 | Error image of the O peripherals I 0 to 255, switched |
| 70 to 85 | Error image of the IPC flags I 0 to 255 |

*Static error image of the outputs (DW 86 to DW 165)*

Bit no. 0 in DW 86 corresponds to DQ 0, bit no. 1 to DQ 1 etc.

| DW no. | Meaning |
|---|---|
| 86 to 93 | Static error image DQ 0 to 127, subunit A |
| 94 to 101 | Static error image AQ 128 to 255, subunit A |
| 102 to 109 | Static error image DQ 0 to 127, subunit B |
| 110 to 117 | Static error image AQ 128 to 255, subunit B |
| 118 to 125 | Static error image DQ 0 to 127, switched |
| 126 to 133 | Static error image AQ 128 to 255, switched |
| 134 to 149 | Error image of the O peripherals Q 0 to 255, switched |
| 150 to 165 | Error image of the IPC flags Q 0 to 255 |

*CP/IP-SS error image (DW 166 to DW 213)*

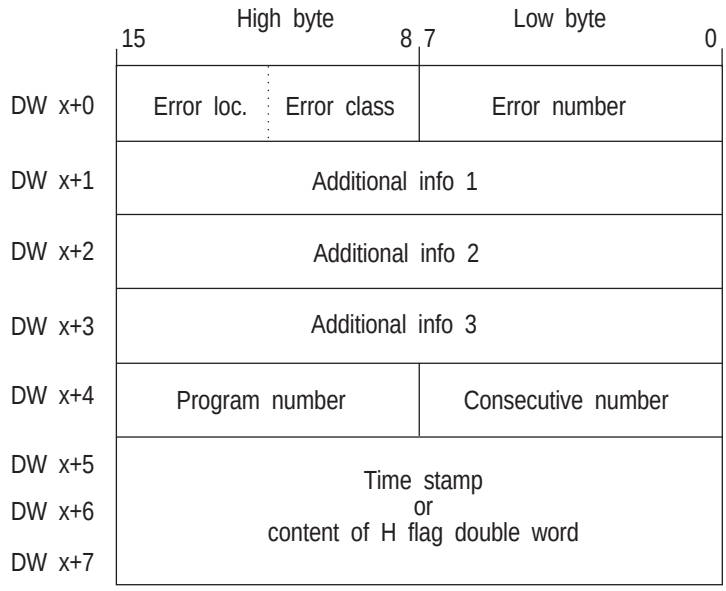
| DW no. | Meaning |
|---|---|
| 166 to 181 | CP/IP error image 0 to 255, subunit A |
| 182 to 197 | CP/IP error image 0 to 255, subunit B |
| 198 to 213 | CP/IP error image 0 to 255, switched |

*Error image of the O peripherals (DW 216 to DW 279)*

| DW no. | Meaning |
|---|---|
| 216 to 231 | Static error image DI/AI 0 to 255 Q, subunit A |
| 232 to 247 | Static error image DI/AI 0 to 255 Q, subunit B |
| 248 to 263 | Static error image DQ/AQ 0 to 255 Q, subunit A |
| 264 to 279 | Static error image DQ/AQ 0 to 255 Q, subunit B |

*Error field*

Fig. 5-5 shows the structure of an error field.




x = start address of the error block

Fig. 5-5 Structure of an error field

**Error class and error location**

| Bit no. | 15 | | | 12 | 11 | | | 8 |
|---|---|---|---|---|---|---|---|---|
| | B | A | RES | MA | – | PAS | WST | HST |

- Location:

  MA Error occurred in the master

  RES Error occurred in the standby

  A Error occurred in subunit A

  B Error occurred in subunit B

- Error class:

  The reaction to the error is entered here (default standard reaction).

  HST HARD STOP (e.g. with CPU error)

  WST SOFT STOP (e.g. with signature error in user program etc.)

  PAS Passivation (e.g. with QVZ, discrepancy error etc.)

  If, for example, a test error is entered in the error field, bit no. 1 is set too ’1’, since the standard reaction is a change to the SOFT STOP.

**Error number:**

The error number is a consecutive number from 1 to 255; each error number has a plain language text assigned to it (see "Structure of the user message field" in this section).

The meaning of the error numbers can be found in a table in the appendix.

**Additional information / user message field / time stamp:**

For detailed information, refer to Part II of Volume 1 of this manual.

### 5.5.3 Calling a User Interface

After every entry in the error DB, the system program calls **OB 37** as the user interface, if it is loaded. Here, you can write a STEP 5 program to evaluate the error DB.

## 5.6 H Flag Word

The H flag word contains important information about the status of your PLC (e.g. "PLC in error search mode"). You can select the number of the H flag word from 0 to 254 and specify it during configuration with COM 155H.

*Structure*

The H flag word consists of a **status byte** (high byte) and a **control byte** (low byte). The information in the status byte can be read bit-by-bit by the user program, the control information in the control byte can be set bit-by-bit.

*Status byte*

The bits in the status byte have the following meaning:

Table 5-8 Status byte in the H flag word

| Bit no. | Abbrev. | Meaning |
|---|---|---|
| 0 | SOL | Bit = ’1’: PLC is in the mode "solo mode"<br>Bit = ’0’: other mode |
| 1 | RED | Bit = ’1’: PLC is in the mode "redundant mode"<br>Bit = ’0’: other mode |
| 2 | MAS | Bit = ’1’: PLC is in the mode "master"<br>Bit = ’0’: other mode |
| 3 | ANK | Bit = ’1’: PLC is in the mode "coupling standby"<br>Bit = ’0’: other mode |
| 4 | ZGA | Bit = ’1’: central controller is subunit A<br>Bit = ’0’: central controller is subunit B<br>(in the redundant mode this bit is always ’0’) |
| 5 | AMA | Bit = ’1’: subunit A is master<br>Bit = ’0’: subunit B is master |
| 6 | ADL | Bit = ’1’: "standby being updated"<br>Bit = ’0’: other mode |
| 7 | AGF | Bit = ’1’: PLC error occurred in switched I/Os<br>Bit = ’0’: no error |

*Control byte*

The bits of the control byte have the following meaning:

Table 5-9 Control byte in the H flag word

| Bit no. | Abbrev. | Meaning |
|---|---|---|
| 0 | OAT | Bit = ’1’: restart master without test<br>Bit = ’0’: restart master with test<br>OAT is reset by the system program at a COLD RESTART and after OVERALL RESET |
| 1 | DPO | Bit = ’1’: cancel passive setting **without** deleting the error fields<br>Bit = ’0’: no action (default)<br>The bit is deleted by the system program after cancelling the passivation. |
| 2 | ARE | Bit = ’1’: updating standby disabled<br>Bit = ’0’: updating standby enabled (default) |
| 3 | – | reserved |
| 4 | DPA | Bit = ’1’: cancel passivation<br>Bit = ’0’: do not cancel passivation default) |
| 5 | AUM | Bit = ’1’: request switchover<br>Bit = ’0’: no switchover request (default)<br>AUM is reset by the system program after a switchover. |
| 6 | RST | Bit = ’1’: set standby to STOP<br>Bit = ’0’: do not set standby to STOP (default)<br>RST is reset by the system program after it is executed |
| 7 | – | reserved |

For more information about the meaning of the H flag word, refer to Part II of Volume 1 of this manual.

## 5.7 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 5-10 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 | 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 |
| Load and transfer error (TRAF) | OB 32 | STOP |

| Cause of error | Organization block called | Reaction of CPU if OB is not programmed |
|---|---|---|
| Table 5-9 continued: | | |
| Collision of timed interrupts:<br>a) queue overflow (control bit WEFES)<br>b) timed interrupt clock was masked (ignored) for too long (WEFEH) | OB 33 | <br>STOP<br>none |
| Error during STEP 5 operation "G DB/GX DX"<br>(control bit FEDBX) | OB 34 | STOP |
| Entry in error DB (refer to Section 5.5) | OB 37 | 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.

## 5.8 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 STOP CAUSE

**Note**
Points to note with the S5-155H with T PY/T PW and addressing errors:

If there is a QVZ and ADF (when the addressed I/Os are not configured or do not exist) when one of the operations above is being executed, only QVZ is signalled in the solo mode and both QVZ and ADF in the redundant mode.

The following sections explain possible causes of error in greater detail.

### 5.8.1 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.

### 5.8.2 OB 19: Opening a Data Block That Is Not Loaded (KDB)

If you open 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.

### 5.8.3 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 (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. | 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:<br>- F 0000H to F EFFFH<br>- 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 |

### 5.8.4 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).

### 5.8.5 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).

The cycle time required is extended by the time taken for a PG function.

> **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 **standby** always changes to the stop mode. The cycle monitoring time of the master CPU is started. The H system continues operation in the solo mode.

If a further cycle time exceeded error 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 master CPU changes to the stop mode.
If OB 26 is not loaded, the **master CPU** also changes to the STOP mode.
When you program OB 26, it should always include a reaction to errors and **not be left empty** (only BE operation) so that the error detection and localization capabilities of your S5-155H are not reduced.

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.

### 5.8.6 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.

### 5.8.7 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 occurs, the address which caused the error can be read out of the system data area (refer to Chapter 8):

| RS | Contents | Address |
|---|---|---|
| 70 | error address high | E F046H |
| 71 | error address low | E F047H |

### 5.8.8 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.

If a load/transfer error particular to the **H system** occurs (= field transfer to redundant I/Os or byte read access to redundant analog inputs), and OB 32 is not loaded, then **only the standby CPU changes to the STOP mode**, the master continues in the solo mode.

### 5.8.9 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 948R:

*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 WEFES 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.

*STP in OB 33*

If there is a collision of timed interrupts in the redundant mode and if there is a 'STP' operation programmed in the loaded OB 33, then **only the standby CPU changes to the STOP mode**, the master continues in the solo mode.
If a further collision occurs and if a 'STP' operation is programmed in OB 33, the master CPU also changes to the STOP mode.

### 5.8.10 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.
- The operation G DB/GX DX is not permitted at this time since the standby linking is currently active.
- The operation G DB/GX DX is not permitted at this time because an online PG function is running.

**Reaction:**

The system program calls **OB 34** is this is loaded. If OB 34 is not loaded, the CPU changes to the STOP mode (see also "Notes on the Special Functions OB 124, OB 125 and OB 254/255" on page 6-7).

### 5.8.11 OB 37: Entry in the Error DB

Since the system program automatically calls OB 37 after every entry in the error DB, it makes sense to evaluate the content of the error DB with a STEP 5 program (using the error counter, write and read pointers and status word etc.), so that the required reaction to an error can be triggered,
An example of such a STEP 5 program can be found on the following page.

***Example***

```
The following program shows an example of a reaction to a specific error:

FB 37 examines the last entered error field for the error number 'yy'. If
this error number is found, the reaction to this specific error can be
triggered.

OB 37                                                          LEN=yy

SEGMENT 1
0000      :JU  FB 37
0001
0002 NAME :ERR-EVAL
0003      :BE

FB 37                                                          LEN=yy

SEGMENT 1       0000
NAME:ERR-EVAL

0005      :MBA                          Error address from BR register
0006      :LRW +1                       Pointer to current error field
0008      :+D
0009      :MAB
000A      :LRW +0                       Load error
000C      :L   KB 255                     number
000D      :AW
000E      :L   KB yy                    Error
000F      :!=F                            number 'YY' ?
0010      :JC  =M001
0011      :JU  =BE
0012 M001 :                             Here, you can program the
0013      :                             reaction to error
0014      :                             number 'yy'.
0015      :
00xx BE   :BE
```

## 5.9 Self-Test

### 5.9.1 Overview

The CPU 948R contains integrated self-test routines in the system program. These routines detect and localize hardware faults and indicate which modules are faulty and must be replaced.

*What is tested?*

The self-test routines check the following system components:

| Which component? | What for (when)? |
|---|---|
| CPU | STEP 5 operations, timers, interrupt mask, cycle time monitoring etc. |
| EPROM, RAM | e.g. comparison of RAMs on the CPUs of both subunits; checksum test of OBs, SBs, PBs and FB and FX data blocks |
| I/O bus with IM 314R | Short circuits and breaks on the I/O bus cable 721 to the IM 314R are detected |
| Page addressing | In the cyclic mode, the page addressing is tested once per complete test run.<br>The test detects the following incorrect functions:<br>- A CP reacts to (acknowledges) not only its own interface number but also, the remaining 255 interface numbers.<br>- An unused interface number acknowledges. All unused interfaces are checked. This test is also performed once per START-UP. |
| Parallel link IM 304/IM 324R | The dual-port RAM of the parallel link is checked by both subunit A and subunit B.<br>This test reveals short circuits and breaks. |
| Locating facilities (LF) | The LF are tested once every 10 hours<br>The LF of the redundant DIs and DQs for constant 0 errors and (only in START-UP) for constant 1 errors. |

### 5.9.2 Organization of the Self-Test

The whole self-test of the S5-155H consists of approximately 3000 test slices.

*Configuring the test slices for the cycle*

You can configure the number of test slices processed once per cycle (see COM 155H, operating system parameter assignment). The number of test slices per cycle can be set to a value between 1 and 19. This corresponds to a test execution time of 2 to 38 ms. A longer test execution time reduces the error detection time.

*Handling pending interrupts*

If more than one test slice is configured the system program checks every 2 ms to find out whether or not a timed or process interrupt is waiting for servicing. If there is an interrupt, the corresponding OB is processed before the next test slice. The self-test execution time is then extended by the time required to process the interrupt.

***Self-test in the various operating modes***

Different self-test routines are called in the in the CPU modes listed below:

*Self-test in START-UP*

During the START-UP of a CPU, the complete self-test is executed. If the self-test routine detects an error here, the CPU changes to the STOP mode. An error message is entered in the error DB.

During the self-test in START-UP, the RUN and STOP LEDs on the front panel are lit constantly.

The self-test during START-UP is not dependent on the type of start-up being run through.

*Self-test in the RUN mode*

After OB 1 has been processed, i.e. once per cycle, **part of the test routines are run through at short intervals** (2 ms test slices). The test routine runs transparently in the background for the remaining software until a hardware failure is detected.

*Self-test in error search mode*

If an error cannot be assigned to a particular subunit, the standby PLC changes to the error search mode. in this mode, the **whole self-test** is run through taking between 10 and 30 seconds.

# Integrated Special Functions 6

## Contents of Chapter 6


6.1 Introduction . . . . . 6-4

6.1.1 Interfaces . . . . . 6-5
6.1.2 Error Handling . . . . . 6-6
6.1.3 Notes on the Special Functions OB 124, OB 125 and OB 254/255 . . . . . 6-7

6.2 OB 121: Set/Read System Time. . . . . 6-8

6.3 OB 122: "Disable Interrupts" On/Off . . . . . 6-12

6.4 OB 124: Delete STEP 5 Blocks . . . . . 6-14

6.5 OB 125: Generate STEP 5 Blocks . . . . . 6-17

6.6 OB 129: Evaluate Battery State . . . . . 6-20

6.7 OB 131: Delete ACCUs 1, 2, 3 and 4 . . . . . 6-21

6.8 OB 132/133: Roll-Up ACCU/Roll-Down ACCU . . . . . 6-22

6.9 OB 141: "Disable Single Cyclic Timed Interrupts" On/Off . . . . . 6-24

6.10 OB 142: "Delay All Interrupts" On/Off. . . . . 6-27

6.11 OB 143: "Delay Single Cyclic Timed Interrupts" On/Off . . . . . 6-30

6.12 OB 150: Set/Read System Time. . . . . 6-33

6.13 OB 151: Set/Read Time for Real Time Interrupt . . . . . 6-38

6.14 OB 153: Set/Read Time for Delayed Interrupt . . . . . 6-45

6.15 OB 180: Random Data Block Access . . . . . 6-48

6.16 OB 181: Test Data Blocks (DB/DX) . . . . . 6-52

6.17 OB 182: Copy Data Area . . . . . 6-54

6.18 OB 222: Restart Cycle Monitoring Time . . . . . 6-57

6.19 OB 254/255: Copy/Duplicate Data Blocks . . . . . 6-58

# 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 948R 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 120 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 948R

| 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<br>OB 125 | Delete STEP 5 blocks<br>Generate STEP 5 blocks | 6.4/6 - 14<br>6.5/6 - 17 |
| OB 129 | Evaluate Battery State | 6.6/6 - 20 |
| OB 131 | Delete ACCU 1, 2, 3 and 4 | 6.7/6 - 25 |
| OB 132<br>OB 133 | Roll-up ACCU<br>Roll-down ACCU | 6.8/6- 26<br>6.8/6- 26 |
| OB 141<br>OB 142<br>OB 143 | "Disable single cyclic timed interrupts" on/off<br>"Delay all interrupts" on/off<br>"Delay single cyclic timed interrupts" on/off | 6.9/6 - 28<br>6.10/6 - 31<br>6.11/6 - 34 |
| OB 150<br>OB 151<br>OB 153 | Set/read system time<br>Set/read time for real time interrupt<br>Set/read time for delayed interrupt | 6.12/6 - 37<br>6.13/6 - 42<br>6.14/6 - 49 |
| OB 180 | Random data block access | 6.15/6 - 48 |
| OB 181 | Test data blocks (DB/DX) | 6.16/6 - 52 |
| OB 182 | Copy data area | 6.17/6 - 54 |
| OB 222 | Restart cycle monitoring time | 6.18/6 - 57 |
| OB 254, 255 | Copy/duplicate DB and DX data blocks | 6.19/6 - 58 |

### 6.1.1 Interfaces

The following are available as interfaces to the special functions:

*Block call*

- Conditional/unconditional block call JC .. / JU ..

*Parameters*

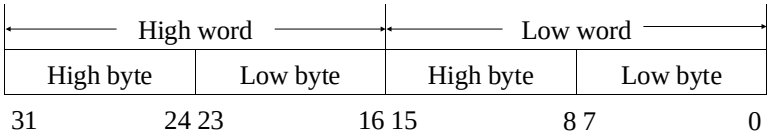
- Parameters for defaults via ACCU 1 and possibly ACCU 2 and/or memory locations

  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 |



### 6.1.2 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").

### 6.1.3 Notes on the Special Functions OB 124, OB 125 and OB 254/255

- If, following the special function "delete block" (OB 124), you want to run the special function "generate block" (OB 125) or a comparable special function (OB 254/255) or the operation G DB/GX DX, a waiting time of 1 sec. must be allowed first. If this waiting time is not kept to, the standby CPU may stop and lead to an error entry "EDB error" in the error DB.

- Before you generate a logic block in your program, you must disable the updating of the standby with data by setting the H flag bit ARE .

- A block generated with OB 125 is considered as invalid by the self-test and not checked. Once you have written the required data to newly generated logic blocks (OB, PB, SB, FB or FX), you must call OB 125 again in the S5-155H to regenerate the checksum (enter the block length ’0’ when you call OB 125).

  After the checksum has been generated, the updating of the standby can be enabled again (reset H flag bit ARE).

**Example: Sequence of operations for generating the checksum for the already generated PB 24.**

```
:
:
:L   KB 0       Length = 0 (ID for checksum generation)
:L   KY 1.24    PB 24
:JU  OB 125
```

## 6.2 OB 121: Set/Read System Time

***Function***

With OB 121 you can set or read the system time (date and time). This function is compatible with the CPU 946R/947R.

***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).

Possible time values:

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 ...10th seconds illegal |
| 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 10).

### 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.

> **Note**
> If this function is used during START-UP, it must only be executed by the master.
> Please note the information in Section 6.1.3.

***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<br>JZ |
| 1 | 1 | 0 | Processing of special function aborted with "warning" | JC<br>JP<br>JN |
| 1 | 0 | 1 | Processing of special function aborted with "error" | JC<br>JM<br>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<br>47H<br>4DH<br>4FH | Error:<br>Block type not permitted<br>Block does not exist<br>Online function COMPRESS MEMORY active<br>Function not possible at present, coupling standby active |
| 8DH<br><br>8EH | Warning:<br>Conflict with an online function (except for "compress memory")<br>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.

> **Note**
> If this function is used during START-UP, it must only be executed by the master.
> Please note the information in Section 6.1.3.

***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.

If you have generated a logic block with OB 125 and have filled it with data, you can **generate a checksum** by calling OB 125 again with **"number of words" = 0** in ACCU-2-L and the appropriate information in ACCU 1.

### 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<br>JZ |
| 1 | 1 | 0 | Processing of special function aborted with "warning" | JC<br>JP<br>JN |
| 1 | 0 | 1 | Processing of special function aborted with "error" | JC<br>JM<br>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 |
| 4FH | Function not possible at present, since coupling standby 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 129: Evaluate Battery State

***Function***

With OB 129 you can use a STEP 5 program to check whether the backup battery is still functioning or has failed (OB 129 scans the BAU signal to do this). Independent of the result, you can then set a fault indicator (lamp), for example.

***Forming the BAU signal***

The power supply contains 2 backup batteries, a lithium cell (MB for main battery) and a rechargeable battery (RB for reserve battery) as a backup. The BAU signal is formed by ANDing the two battery monitoring signals.

***Jumper settings in the power supply***

The PLC's power supply contains two jumpers via which the battery monitoring can be influenced. The jumper MB-NB determines the time at which the BAU signal is formed: if the jumper MB-NB is not plugged, BAU is formed once following power on. Otherwise the batteries are monitored cyclically during operation. The monitoring signal for the rechargeable battery can be switched off via the jumper MA-NA.

The various jumper settings result in the range of battery monitoring possibilities using OB 129 listed in the following table:

| Jumper MB-NB | Jumper MA-NA | Time BAU signal formed | BAU signal formed from the monitoring signal from the... |
|---|---|---|---|
| open | open | following power on | lithium cell |
| open | closed | following power on | lithium cell and rechargeable battery |
| closed | open | cyclically | lithium cell |
| closed | closed | cyclically | lithium cell and rechargeable battery |

***Parameters***

None

***Result***

RLO = '0' Battery state is in order
RLO = '1' Battery has failed

ACCU 1 = '1':Battery in subunit A failed
ACCU 1 = '2':Battery in subunit B failed
ACCU 1 = '3':Both batteries failed

## 6.7 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 bits) are deleted (’0’).

***Possible errors***

None

***Example***

```
With the following sequence of operations you can
test whether the backup battery is still in order
and switch on a warning lamp if a fault occurs:

     :
     :JU  OB 131
     :JU  =BATL     RLO = 1 -> battery failed
     :
     :
     :BEU
BATL :R Q22.0
     :R Q22.1
     :L QB22
     :O W
     :T QB22    Switch on warning lamp at output
     :          byte 22 bit 0 and 1
     :          or
     :          Bit 0 = battery in A failed
              Bit 1 = battery in B failed
```

## 6.8 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 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

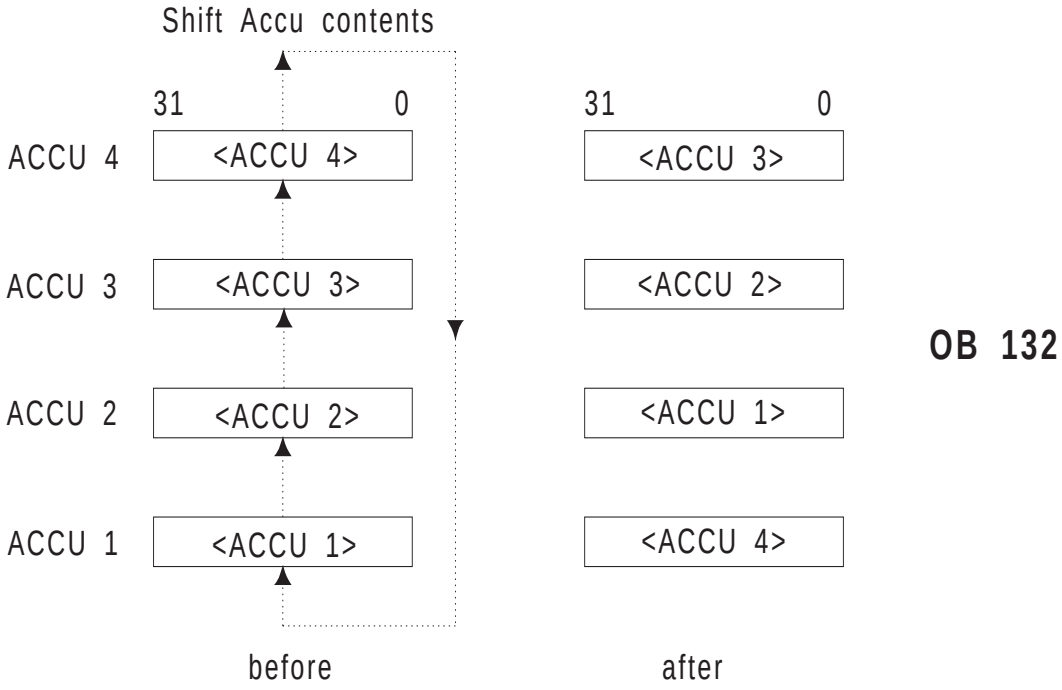



Fig. 6-1 Effect of the "roll-up" function

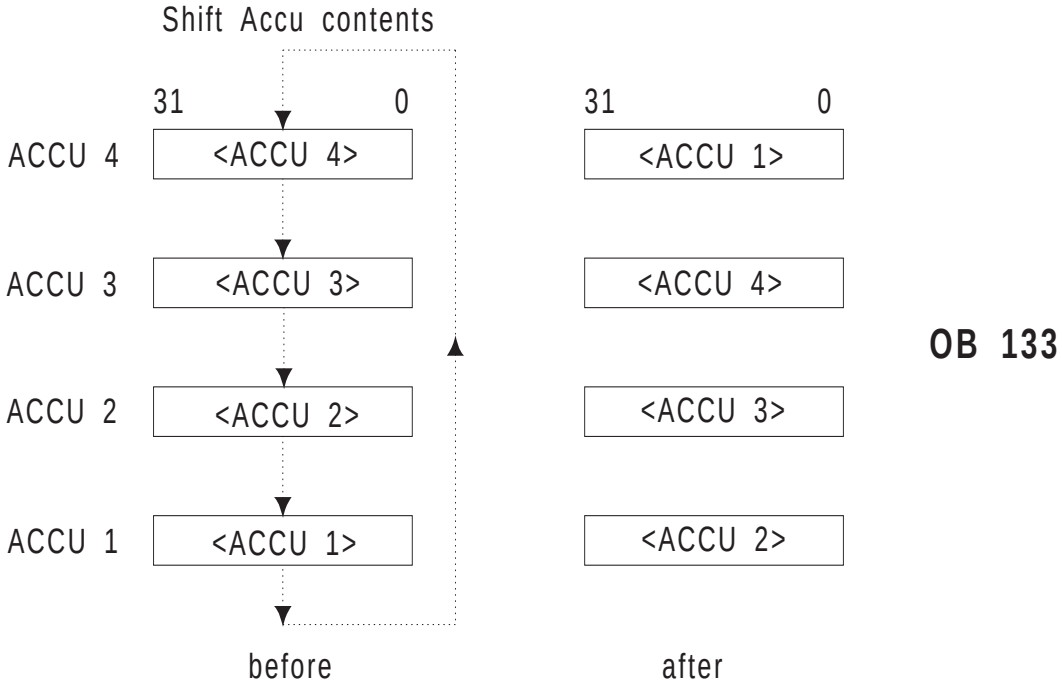



Fig. 6-2 Effect of the "roll-down" function

## 6.9 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 boundary .

### *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’<br>4 = ’1’<br>5 = ’1’<br>6 = ’1’<br>7 = ’1’<br>8 = ’1’ | Cyclic timed interrupts with fixed interval:<br>period 1 (OB 10)<br>period 2 (OB 11)<br>period 3 (OB 12)<br>period 4 (OB 13)<br>period 5 (OB 14)<br>period 6 (OB 15) |

| Bit no. | Interrupt |
|---|---|
| (...)<br>9 = ’1’<br>10 = ’1’<br>11 = ’1’ | <br>period 7 (OB 16)<br>period 8 (OB 17)<br>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 6-8 Error IDs of OB 141 in ACCU-1-L

| ID | Meaning |
|---|---|
| 8D01H | illegal function no. in ACCU-2-L [1] |
| 8D02H | 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.

## 6.10 OB 142: "Delay All Interrupts" On/Off

A STEP 5 program can be interrupted at block boundaries by programs with a higher priority. The process interrupts and all timed interrupts belong to these higher priority program exection 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 boundaries.

### *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:



The bits in the control word have the following meaning:

| Bit no. | Type of interrupt |
|---|---|
| 0 = ’1’ | Cyclic timed interrupts, fixed period |
| 1 = ’1’ | Real time 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 6-9 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.

## 6.11 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 boundary.

### *Parameters*

**1. Control word**

OB 143 enters the timed interrupts to be delayed in a system-internal control word:




The bits in the control word have the following meaning:

| Bit no. | Interrupt |
|---|---|
| 0 to 2 | Reserved, these bits must be ’0’ |
| 3 = ’1’<br>4 = ’1’<br>5 = ’1’<br>6 = ’1’<br>7 = ’1’<br>8 = ’1’<br>9 = ’1’ | Cyclic timed interrupts with fixed period<br>period 1 (OB 10)<br>period 2 (OB 11)<br>period 3 (OB 12)<br>period 4 (OB 13)<br>period 5 (OB 14)<br>period 6 (OB 15)<br>period 7 (OB 16) |

| Bit no. | Interrupt |
|---|---|
| (...)<br>10 = ’1’<br>11 = ’1’ | <br>period 8 (OB 17)<br>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 6-10 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.

## 6.12 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 948R 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<br>1/100 seconds<br>Minutes<br>Hours<br><br>Weekday<br>Day of month [1]<br><br>Month<br>Year | 0 to 59<br>0 to 99 (when setting = 0)<br>0 to 59<br>0 to 23 or 1 to 12, depending on the "format"<br>0 to 6 for Mon to Sun<br>1 to 31<br>1 to 12<br>0 to 99 | BCD format |
| Format | Format for the hour field with the following meaning:<br>Bit 15 = 0: 12 hour format ("am" or "pm" selected in bit 14)<br>Bit 15 = 1: 24 hour format (bit 14 = 0)<br>Bit 14 = 0: "am"<br>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 6-11 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 following 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)
```

## 6.13 OB 151: Set/Read Time for Real Time Interrupt

***Function***

By calling OB 151 you can do the following:

- cause the CPU 948R to activate the real time 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 real time 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** real time 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:<br>0: cancel job or no job active<br>1: every minute<br>2: every hour<br>3: every day<br>4: every week<br>5: every month<br>6: every year<br>7: once | BCD format |
| Seconds<br>1/100 seconds<br>Minutes<br>Hours<br><br>Weekday<br>Day of month [1]<br><br>Month<br>Year | 0 to59<br>0 to 99 (when setting = 0)<br>0 to 59<br>0 to 23 or 1 to 12, depending on the format<br>0 to 6 for Mon to Sun<br>1 to 31<br>1 to 12<br>0 to 99 | BCD format |
| Format [2] | Format for the hour field with the following meaning:<br>Bit 15 = 0: 12 hour clock<br>Bit 15 = 1: 24 hour clock (bit 14 = 0)<br>Bit 14 = 0: "am"<br>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, 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 6-12 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 OB 121/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 real time interrupt (timed job) is to be triggered, the individual time parameters must be specified in certain combinations. Depending on the selected time for the real time 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 6-13 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 following 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
```

## 6.14 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 via OB 153.

*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: 0005H 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, 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 6-14 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.
        :
```

## 6.15 OB 180: Random Data Block Access

***Application***

You can use OB 180 when working with data blocks that are longer than 261 words (incl. 5 words header). You use OB 180 to shift an 'access window' of 256 data words in a 16-word grid through a data block (to paragraph addresses). By calling OB 180 more than once, you can shift the access window further.

In contrast to the CPU 928B, you cannot position the access window continuously, but only in **multiples of 16** in the CPU 948R.

***Function***

With OB 180 the start address of the current data block is shifted by a given value towards the end of the block. This takes into consideration the fact that the DB length still available is decreasing (DBA and DBL registers - see Sections 8.3 and 9.2.1 - are loaded according to the shift).

> **Note**
> Before calling OB 180, a **sufficiently long** data block (DB or DX) must be opened.

***Parameters***

**ACCU-1-L**

Shift digit V: number of data words by which the data block start address is to be shifted

Permitted values: $0 \leq V < DBL$, **V = n * 16 (16, 32, 48...)**

***Result***

When OB 180 has been called **successfully**

- The relative address of DW 0 has been shifted according to the value in ACCU-1-L (DBA and DBL registers have been adjusted accordingly),
- The RLO is deleted (RLO = 0),
- All other bit and word condition codes are cleared,
- The content of ACCU-1-L = 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. The other bit and word condition codes are cleared.
**The values for DBA and DBL remain unchanged.**

Table 6-15 Error IDs of OB 180

| ID | Meaning |
|---|---|
| B401H | No data block is open |
| B410H | Shift digit V is not a multiple of 16 |
| B411H | Shift digit V is too large; the block end was exceeded by the new position of the window. |
| B412H | Shift digit V is negative |

### Resetting access window to start value

Opening the data block again with the operations C DB or CX DX restores the original state.

### Behavior with nested blocks

If the access window is shifted in a code block by calling OB 180 and a further code block is then called, the position of the access window in the called code block is **retained until OB 180 is called again** (DBA/DBL values do not change).

If, however, the access window is shifted in a called code block by calling OB 180, it is returned again **to the position it was in when the nested code block was called** with the jump out of the called block (block end operation).

***Example***

```
The data block start address (DBA = 4152H in DB 17 (length =
256 DW) is to be shifted towards the block end by 32 data words.



     :C   DB 17     Open DB 17
     :L   KB 32     Shift/offset as constant
     :JU  OB 180    Call OB 180: shift access window

After OB 180 is called, the data word stored e.g. at the address 4 1543H is
no longer addressed with DW 35, but with DW 3 etc. (see Fig. 6-3).

Owing to the fact that the DBL register was changed at the same time, the
error monitoring is guaranteed: the operation T DW 223 is permitted but T
DW 224/L DW 224 are illegal.

By calling OB 180 several more times, the DBA can be further increased
(meaning the DBL is further decreased): the operation C DB 17 restores the
original state (DBA = 4152H, length = 256 DW).

If DB 17 had a length of e.g. 258 data words, you could no longer access
DW 256 and DW 257 using STEP 5 operations. By shifting the DBA register by
16, the data words 256 and 257 can be addressed as DW 240 and DW 241.

Continued on following page
```

*Example continued:*

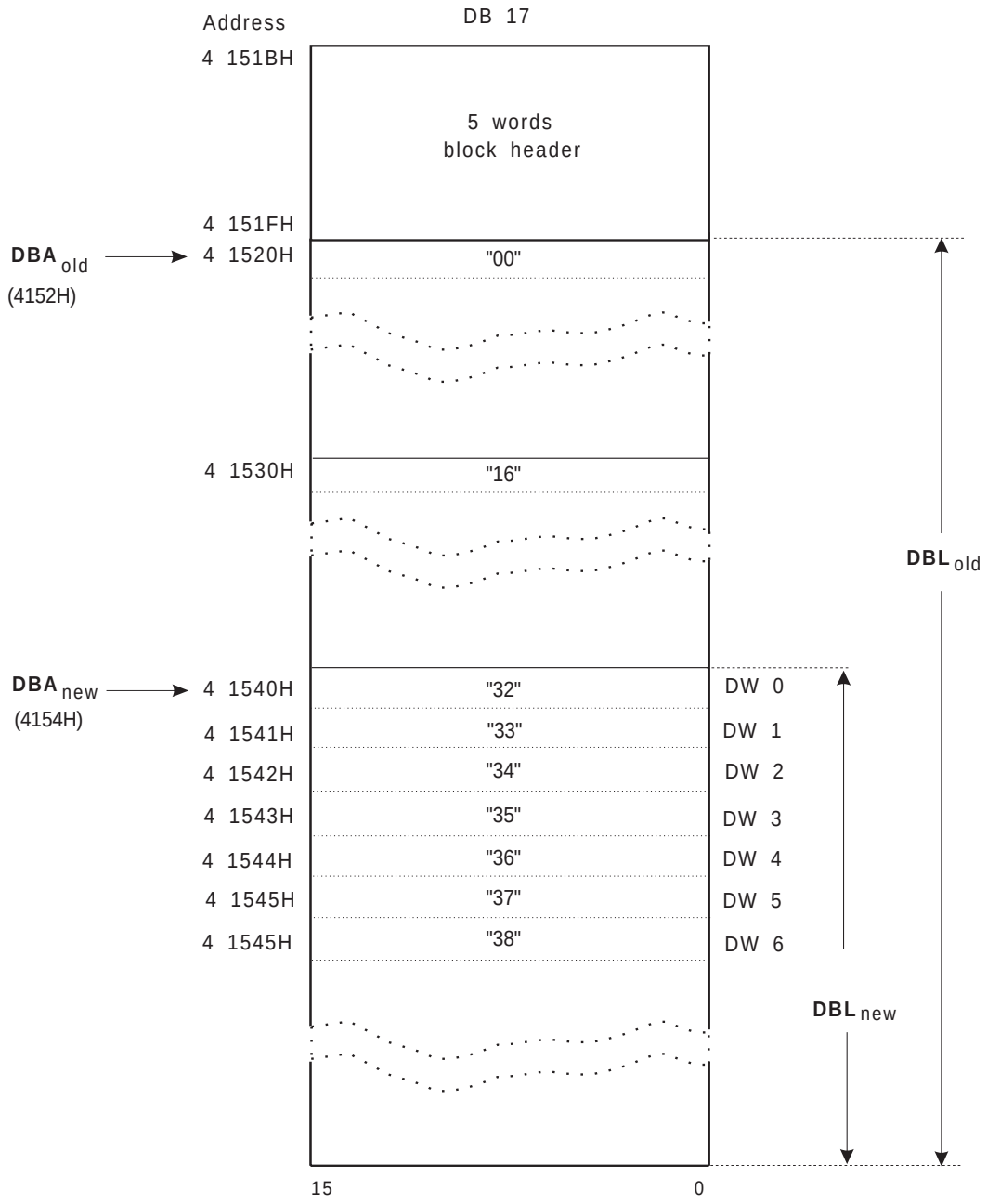



Fig. 6-3 Shifting the DB start address

## 6.16 OB 181: Test Data Blocks (DB/DX)

Using the special function organization block OB 181 you can check

- whether a specific DB or DX data block is available
- at which address the first data word of the data block is stored
- how many data words this data block contains.

***Application***

The function "testing DB/DX" is useful before the operations TNB/TNW, G DB/GX DX and before calling the special function organization blocks OB 182, OB 254 and OB 255.

For example, before transferring a field of data words, you can call OB 181 to ensure that the destination data block is valid and long enough to accommodate all the data words to be copied.

***Function***

OB 181 checks whether a given data block is available and returns the characteristic parameters of the data block as a result.

***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 error and the block tested is **present** on the CPU, the system program passes the following values as results:

**ACCU-1-L**: address of first data word (DW 0),
20-bit address

**ACCU-2-L**: length of data block in words (excluding block header)
Example: ACCU-2-L contains the value ’7’:
The data block consists of the data words DW 0 to DW 6

RLO: = 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 sets the other condition codes shown in the table below. It also writes an error ID to ACCU-1-L.

Table 6-16 Error IDs of OB 181 and their scan

| RLO | CC1 | CC0 | ACCU 1 | ACCU 2 | Meaning | Scan |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | Address | Length | No error | JC |
| 1 | 0 | 1 | B501H | Irrelevant | Block not available | JC<br>JM<br>JN |
| 1 | 1 | 0 | B502H | Irrelevant | Wrong block number | JC<br>JP<br>JN |
| 1 | 1 | 0 | B503H | Irrelevant | Wrong block ID | JC<br>JP<br>JN |

## 6.17 OB 182: Copy Data Area

***Function***

OB 182 copies a data field of variable length from one data block to another. DB or DX blocks can both be used as source block and destination block. The field start in both source and destination block can be freely selected. OB 182 can copy a maximum of 4091 data words.

> **Note**
> Source and destination may be identical. The data areas of the source and destination may overlap. The **original data** of the source area are still copied unchanged to the **destination area** even if they overlap. The **overlap area in the source** is overwritten after copying. You can use these characteristics to move a data area within a block.

***Parameters***

**1. Data field with copy parameters**

Before calling OB 182, pass the required parameters to a data field for the copy procedure. 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 describes the source and destination data block, the field start address in both blocks and the number of data words to be transferred. It consists of five words:

| Bit no. | 15 8 | 7 0 |
|---|---|---|
| word 1 | Source DB type | Source DB number |
| word 2 | Number of first data word in source DB to be transferred | |
| word 3 | Destination DB type | Destination DB number |
| word 4 | Number of first transferred data word in destination DB | |
| word 5 | Number of data words | |

The parameters have the following meaning and permitted range of values:

| Parameter | Permitted range of values |
|---|---|
| Data block type (source and destination) | 1 = DB<br>2 = DX |
| Data block number (source and destination) | 3 to 255 |
| Number of first data word (source and destination) | 0 to 4090 |
| Number of data words | 1 to 4091 |

*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 182 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 |
| Data field word 5 | Flag byte x+8 | Flag byte x+9 |

**2. ACCUs**

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,
permitted 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 correct processing of OB 182, 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, an error ID is written to ACCU 1 (see table) and the system program sets the RLO to '1'.

| ID | Meaning |
|---|---|
| B601H | Data block not loaded |
| B60FH | Multiple call for the block |
| B611H | Illegal data field description |
| B612H | Address area type illegal |
| B613H | Data block number illegal |
| B621H | "Number of 1st data field word" illegal |
| B622H | "Source data block type" illegal |
| B623H | "Source data block number" illegal |
| B624H | "Number of first data word in source DB to be transferred" illegal |
| B625H | Length of source data block in block header < 5 words |
| B626H | "Destination data block type" illegal |
| B627H | "Destination data block number" illegal |
| B628H | "Number of first transferred data word in destination DB" illegal |
| B629H | 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 |

## 6.18 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

## 6.19 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.
- If one of the functions is used in the START-UP, it must only be executed by the master.
- Note the information in Section 6.1.3.

***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)<br>DX (OB 254) | 3 to 255<br>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)<br>DX (OB 254) | 3 to 255<br>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 6-17 Result codes for OB 254/255

| RLO | CC 1 | CC 0 | Meaning | Scan |
|---|---|---|---|---|
| 0 | 0 | 0 | Special function correctly processed | JB<br>JZ |
| 1 | 1 | 0 | Special function aborted with "warning" | JB<br>JP<br>JN |
| 1 | 0 | 1 | Special function aborted with "error" | JB<br>JM<br>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 6-18 Result IDs for OB 254/255 in ACCU-1-L

| 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 |
| 4FH | Function not possible at present, since coupling standby active |
| | 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


7.1 Application . . . . . 7 - 4

7.2 Parameter Assignment Using the PG Screen Form. . . . . 7 - 5

7.3 Meaning of Some Parameters for Running the Program . . . . . 7 - 7

7.3.1 Updating the Timers . . . . . 7 - 7
7.3.2 Setting the Cycle Monitoring Times . . . . . 7 - 7
7.3.3 Level Priorities . . . . . 7 - 7

# Extended Data Block DX 0

# 7

The following chapter explains how to use the data block DX 0. You will find information about assigning parameters using a screen form and the meaning of the various DX 0 parameters.

## 7.1 Application

You can adapt certain system program functions to meet your own requirements by selecting alternative settings in the DX 0 screen form compared with the standard defaults .

The defaults of the system program are automatically set during each COLD RESTART and DX 0 is then evaluated. If you have not assigned parameters for 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 the settings in DX 0 if the PG software STEP 5/ST, Version 6.6 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**.

## 7.2 Parameter Assignment using the PG Screen Form

A screen form is available in the PG system software for assigning parameters in DX 0 for the CPU 948R. The PG software automatically generates data block DX 0 according to the default parameters (in bold face) and parameters you have specified.

How to select and complete PG screen forms is explained in your STEP 5 manual.

*Layout of the screen form*

The PG screen form for completing DX 0 contains the parameter groups (Fig. 7-1):

- Restart after power up
- Number of timer cells
- Cycle time monitoring
- Clock pulse timed interrupts
- Clock processing
- Interrupt priorities

```
DX - param. ass  (S5 155H/  CPU 948R)                          DX  0

Restart after power up:                          1    ( 1 = cold restart with memory
                                                        2 = cold restart
                                                        3 = manual restart )
Number of timer cells :                          256  ( 0 . . . 256 )
Cycle time monitoring  ( X 10 ms ) :             20   ( 1 . . . 255 )
Clock pulse timed ints ( X 10 ms ) :             10   ( 1 . . . 255 )
Clock processing :                               1    ( 1 = factor 1, 2, 5, 10
                                                        2 = factor 1, 2, 4, 8 )

Interrupt priorities :
  Time int. servicing    :                       YES  Priority :  1
  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
  IBO Interrupt          :                       YES  Priority :  2

F 1      F 2      F 3 Select   F 4      F 5      F 6      F 7 Enter   F 8 Cancel
```

Fig. 7-1 PG screen form for assigning parameters to DX 0

The screen forms in older PG software versions can be used to assign parameters.

The following settings are obligatory for DX 0 parameter assignments (S5-155U, CPU 948) for the S5-155H:

- Interruptability: at block boundaries

- For the setting
  Restart after power up: 1 (warm restart)
  the option
  Warm restart procedure: 2 (cold restart with memory)
  must be selected

For DX 0 parameter assignment with older PG software versions

- Mode 150U

## 7.3 Meaning of Some Parameters for Running the Program

### 7.3.1 Updating the Timers

The system program processes the timers as shown below:

- 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 | .... |

### 7.3.2 Setting the Cycle Monitoring Times

You can set the cycle monitoring time with the OB 31 or DX 0 parameter.
If the cycle monitoring time is set with both OB 31 and DX 0, the system program uses the setting from OB 31 because it evaluates DX 0 earlier. For this reason, you should not use both possibilities at the same time. We recommend setting the cycle monitoring time in DX 0.
Since PG functions extend the cycle time, you should add a further 30% to the cycle time setting.

### 7.3.3 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.

# Memory Assignment and Memory Organization

# 8

## Contents of Chapter 8


8.1 Structure of the Memory Area . . . . . 8 - 4

8.2 Memory Assignment in the CPU 948R . . . . . 8 - 5

8.2.1 Memory Assignment for the System RAM . . . . . 8 - 6
8.2.2 Memory Assignment for the I/O . . . . . 8 - 8

8.3 User Memory Organization in the CPU 948R . . . . . 8 - 10

8.3.1 Block Headers in User Memory. . . . . 8 - 12
8.3.2 Block Address List in Data Block DB 0 . . . . . 8 - 13
8.3.3 RI/RJ Area. . . . . 8 - 14
8.3.4 RS/RT Area . . . . . 8 - 15
8.3.5 Bit Assignment of the System Data Words . . . . . 8 - 18
8.3.6 Addressable System Data Area . . . . . 8 - 38

# 8 Memory Assignment and Memory Organization

You can use this chapter as a reference section to check on the organization of the CPU 948R 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 948R 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 (I/O) 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 | |
| | IM3/IM4 area | 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.

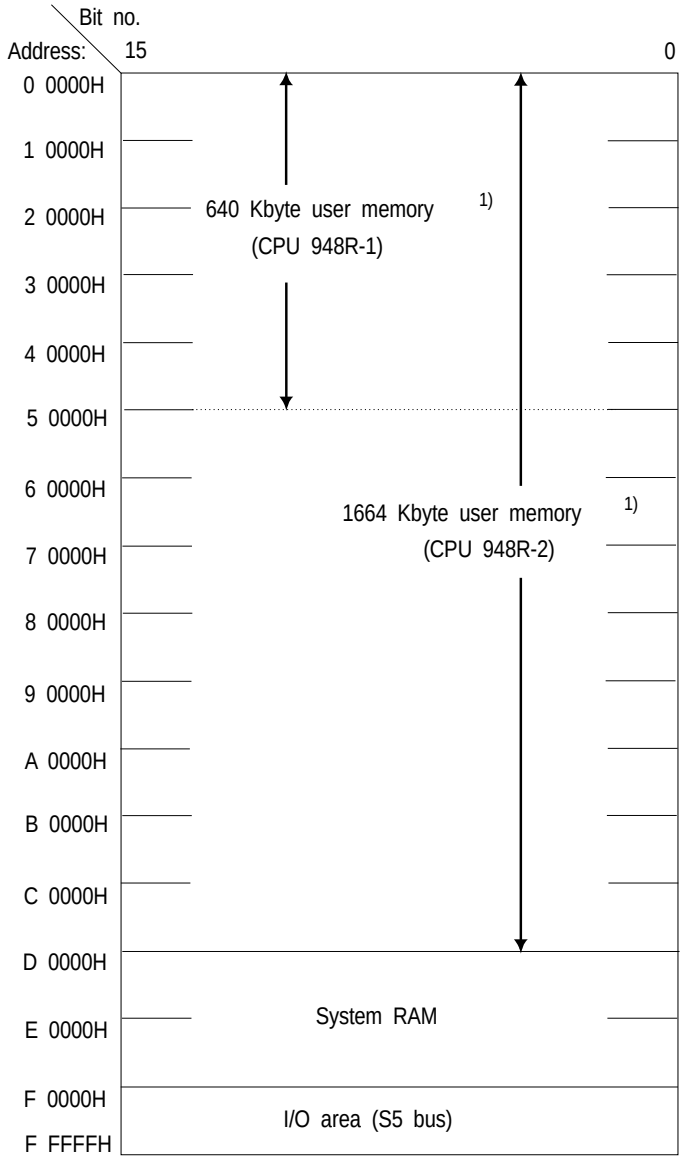
Direct access to the areas I, Q and F result in ’FFH’ in the high byte and the data in the low byte. For direct access to S flags, the **high byte is undefined**!

## 8.2 Memory Assignment in the CPU 948R

With the CPU 948R, there are two possible user memories (RAM) available:

- the CPU 948R-1 with 640 Kbytes of user memory
- the CPU 948R-2 with 1664 Kbytes of user memory.

Fig. 8-1 illustrates the distribution of the address area of the CPU 948R and the location of the user memory versions.




1) The last 20 words of the user memory cannot be used.

Fig. 8-1 Memory assignment in CPU 948R/overview

### 8.2.1 Memory Assignment for the System RAM

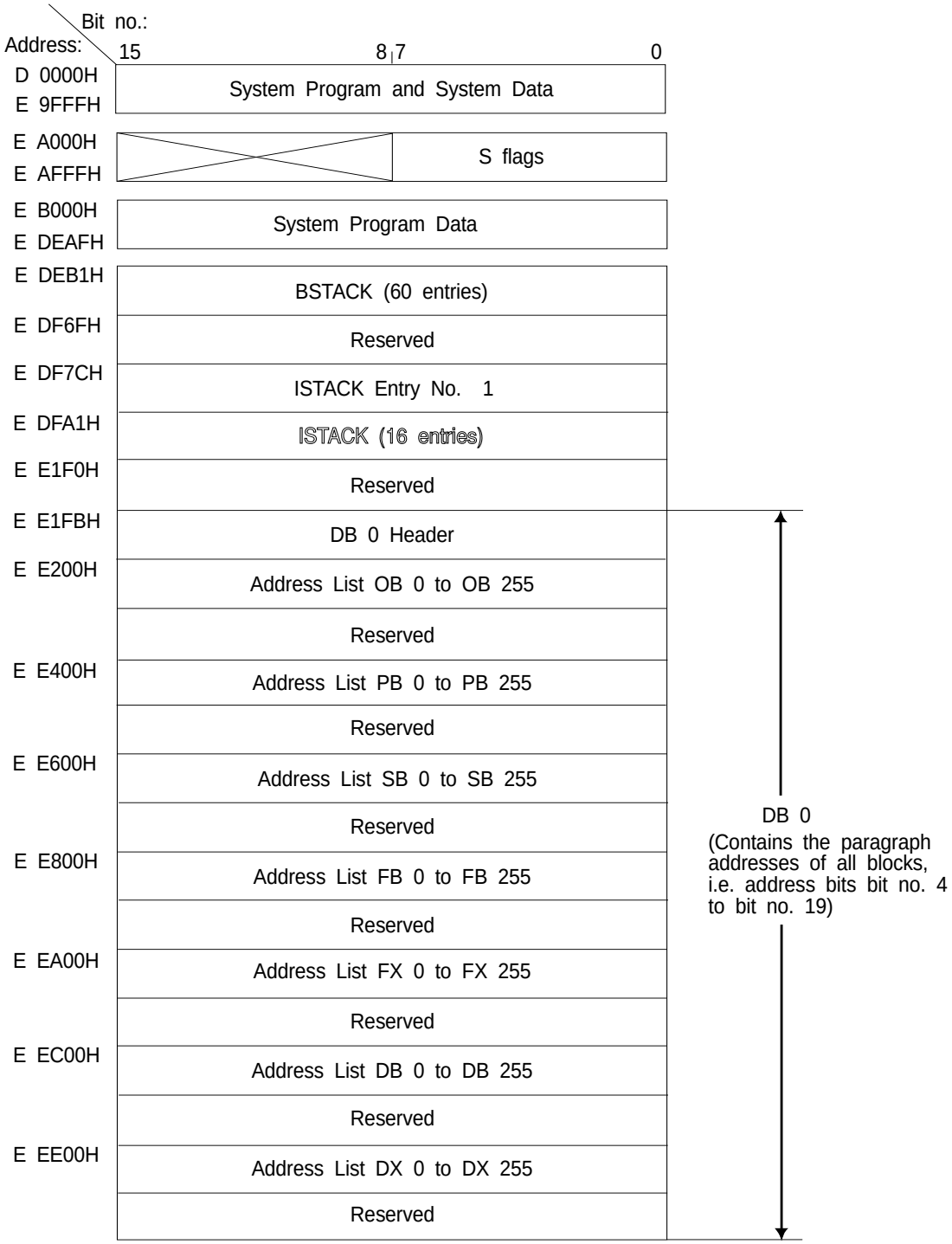



Fig. 8-2 Memory assignment for the system RAM/Part 1

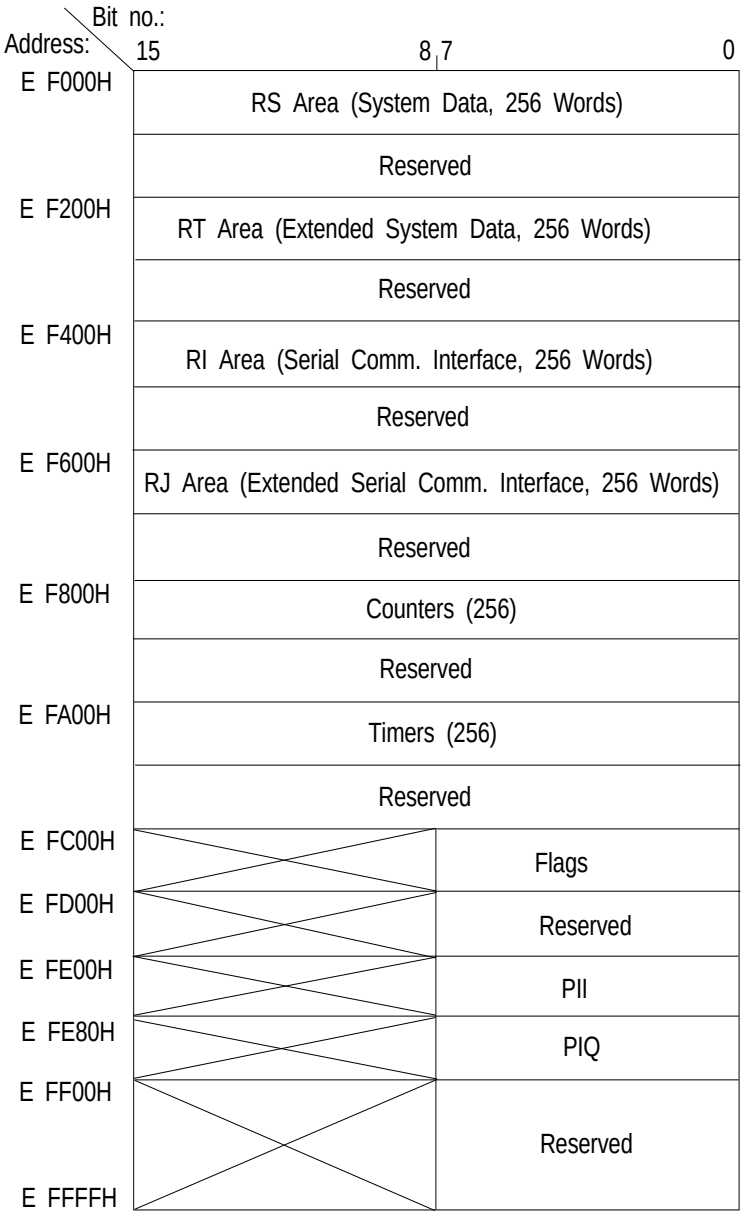



Fig. 8-3 Memory assignment of the system RAM/Part 2

### 8.2.2 Memory Assignment for the I/O

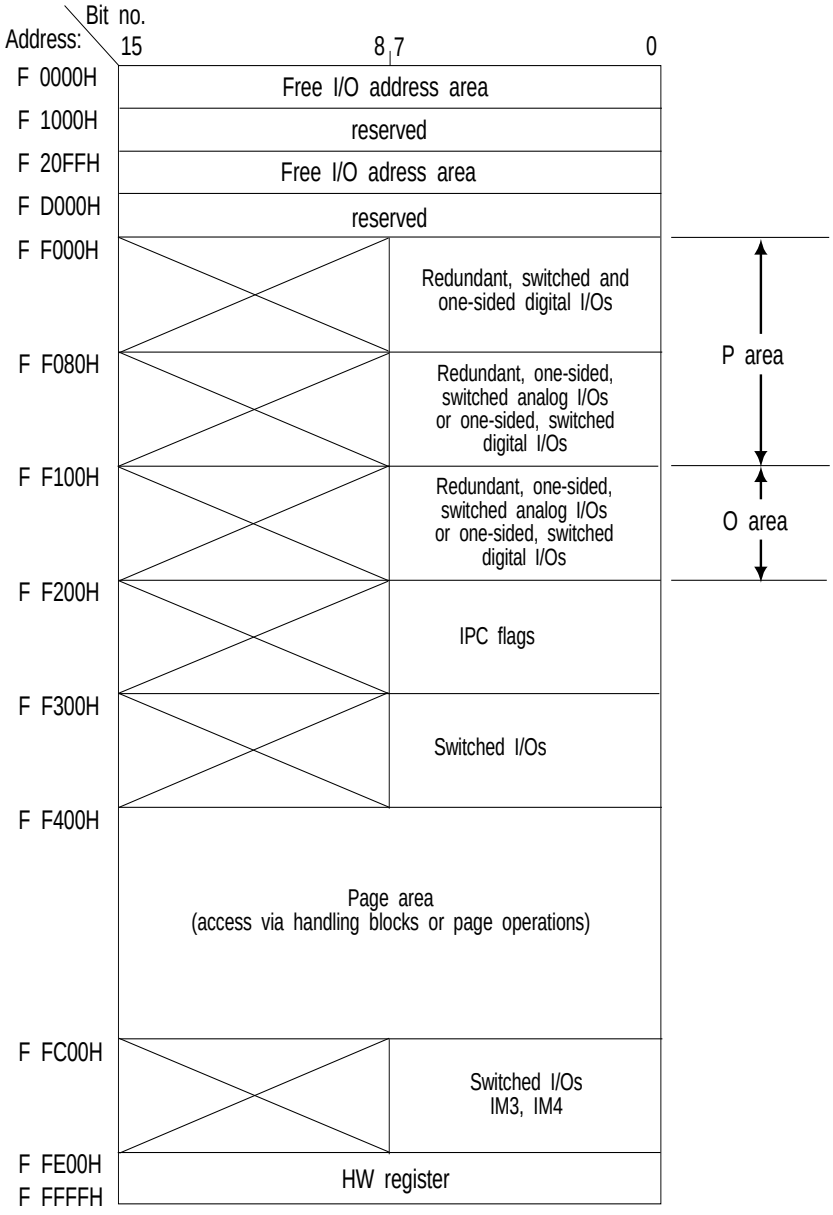



Fig. 8-4 Address areas for the I/O (8 bits) on the S5 bus

### *Address areas for the I/O and programming them*

| Area (absolute address) | Referenced with | Parameter |
|---|---|---|
| E FE00 – E FE7F: PII (Process image input) | L IB / T IB<br>L IW / T IW<br>L ID / T ID<br>A I/ AN I / O I / ON I<br>S I / R I / = I | 0 to 127<br>0 to 126<br>0 to 124<br>0.0 to 127.7 |
| E FE80 – E FEFF: PIQ (Process image output) | L QB / T QB<br>L QW / T QW<br>L QD / T QD<br>A Q / AN Q / O Q / ON Q<br>S Q / R Q / = Q | 0 to 127<br>0 to 126<br>0 to 124<br>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<br>L PW / T PW | 0 to 127<br>0 to 126 |
| F F080 – F F0FF: Digital or analog peripherals Inputs/outputs | L PY / T PY<br>L PW / T PW | 128 to 255<br>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<br>L OW / T OW | 0 to 255<br>0 to 254 |
| **"O" peripherals** | The inputs and outputs are addressed directly in bytes or words. | |

Using STEP 5 operations, you can access the I/O 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 IM3/IM4 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 948R

Depending on the version of the CPU 948R 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 948R, 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*

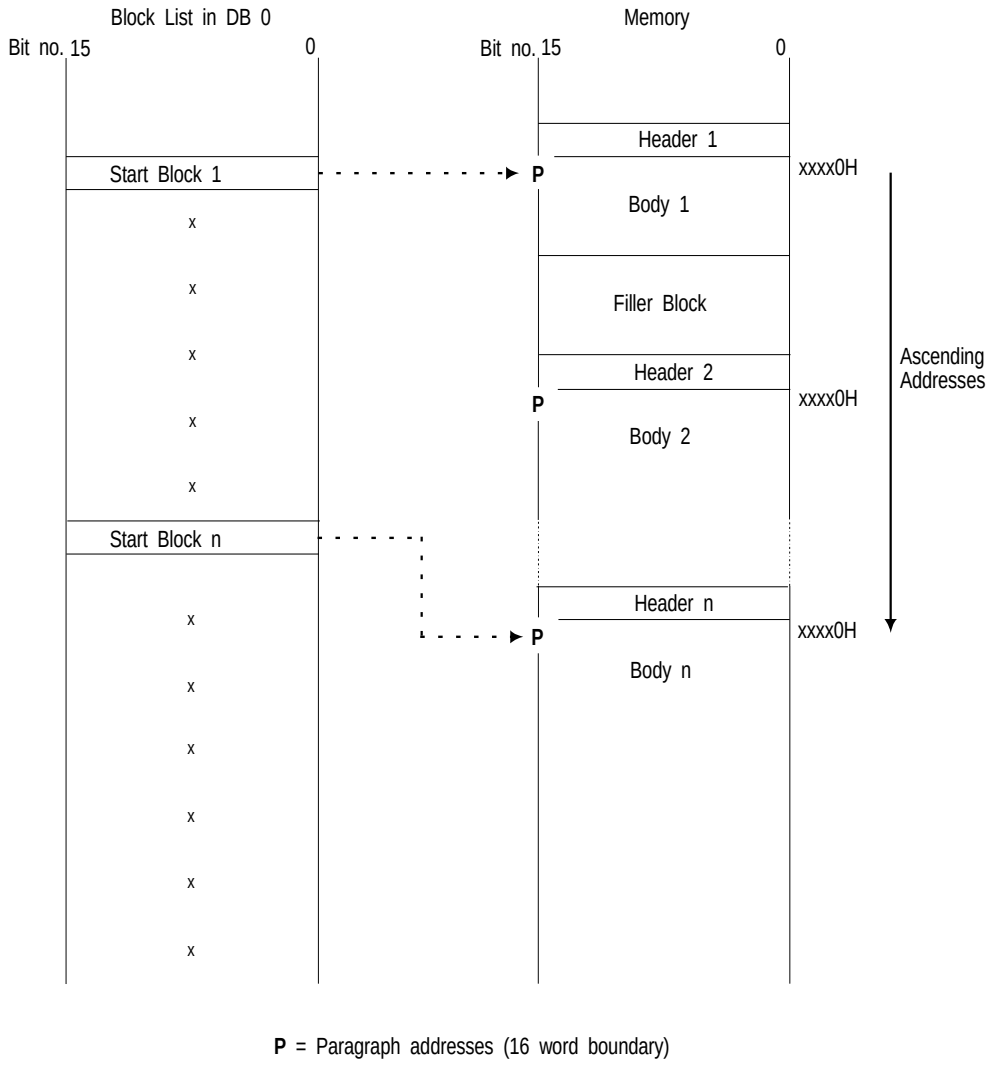



Fig. 8-5 Example: Location of blocks in memory

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.

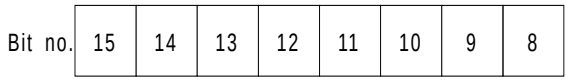
| **Difference** | **Calculation for length of filler block (including header)** |
|---|---|
| 0 to 5 | Add 10 to the difference |
| 6 | No filler block is inserted |
| 7 to 10 | Add 10 to the difference |
| 11 to 15 | Subtract 6 from the difference |

### 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)

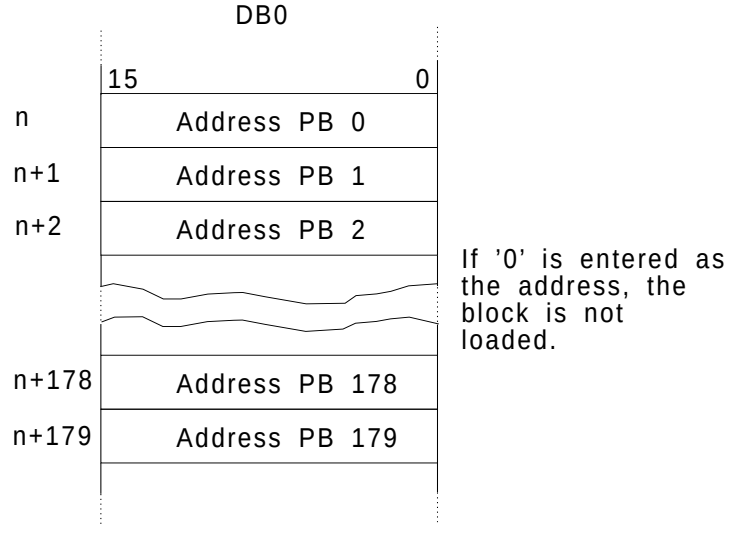



Fig. 8-6 Block addresses in DB 0

*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**
**Only system data words RS 60 to RS 63 and bit RS 9.0 can be written.**

**The system data words reserved for the system program** (on a dark shaded background in Table 8.2) **must not be read:**

**System data words RS 50, RS 68 to 71 and RS 75 to 78** (on a light shaded background in Table 8.2) ***must only be read via the PG,* all other system data** (on a white background in Table 8.2) **can also be read by the user program**.

Illegal access 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 **cannot** use the **whole RT area** (RT 0 to RT 255) for your own purposes.

Only an overall reset can clear the RS/RT areas.

The information contained in some system data (about the internal structure of the CPU, the software release, the CPU ID etc.) is also available using the PG function SYSTEM PARAMETERS.

## Assignment of the RS area

Table 8-2 Assignment of the RS area

| RS | Name | Address |
|---|---|---|
| 0 | Input byte IB 0 image table (process interrupts) | E F000H |
| 1 | Process interrupts currently pending (IB 0) | E F001H |
| 2 to 4 | System program | |
| 5 | Current cycle time of the S5-155H | E F005H |
| 6 | System program | |
| 7 | STOP IDs (ISTACK) | E F007H |
| 8 | Start/restart IDs (ISTACK) | E F008H |
| 9 | QVZ test (bit 9.0) | E F009H |
| 10 to 15 | 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 59 | System program | |
| 60 to 63 | Free for user | E F03CH to E F03FH |
| 64 to 67 | System program | |
| 68 to 71 | Error address for QVZ and PARE errors (must only be read via PG) | E F044H to E F047H |
| 72 to 74 | System program | |
| 75 to 81 | System message (must only be read via PG) | E F04BH to E F04EH |

| RS | Name | Address |
|---|---|---|
| Table 8-2 continued | | |
| 82 to 83 | Interrupt code reset word (UALW) | E F052H to EF053H |
| 84 to 250 | System program | |
| 251 | Current cycle time of S5-155H | E F0FBH |
| 252 to 255 | System program | |

Access by the STEP 5 user program to the system data words reserved for the system program causes RAM comparison errors or S5-155H synchronization errors. All these words can be read using the PG.

As a supplement to the table above, the following pages provide the bit assignments of some of the system data that you can evaluate using 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 of the S5-155H**
**Address: E F005H**

Table 8-5 Bits of RS 5 (current cycle time of the S5-155H)

| **High byte and low byte** | |
|---|---|
| Bit no. | Assignment |
| 15 | The entered binary value * 10 ms 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:

$(2^4 + 2^3) * 10\ ms = (16 + 8) * 10\ ms = 240\ ms$

***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 | reserved |
| 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 RS 9***

**QVZ test**
**Address: E F009H**

Table 8-8 Bits of RS 9 (QVZ test)

| **High byte** | |
|---|---|
| Bit no. | Assignment |
| 15 | Occupied by system program |
| 14 | |
| 13 | |
| 12 | |
| 11 | |
| 10 | |
| 9 | |
| 8 | |
| **Low byte** | |
| 7 | Occupied by system program |
| 6 | |
| 5 | |
| 4 | |
| 3 | |
| 2 | |
| 1 | |
| 0 | QVZ test |

### *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 8-9 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 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 … 8 | 7 … 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 8-10 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 | Opened data block 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 |

| Error number | Parameter type | Meaning |
|---|---|---|
| Table 8-10 continued: | | |
| 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 |
| 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/DB 1 errors*

Table 8-11 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 |
| 3CH | 07H | Peripheral entered in DB 1 is not plugged in |
| 3DH | 07H | Offset too big (parameter error) |
| 3EH | 07H | Too many offsets |

*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: | block type/block number (IDs from block header) | |
| 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<br>= 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 | |

| Parameter type | Structure of the parameter field | | |
|---|---|---|---|
| Table 8-12 continued | | | |
| 07H | Parameter 1 | block ID or code word in DX 0 or DB 1<br><br>"DI08": input switched<br>"DI01": input in subunit A<br>"DQ08": output switched<br>"DQ01": output in subunit A<br>"CE00": IPC input flag<br>"CA00": IPC output flag | |
| | Parameter 2 | incorrect parameter 1 in DX 0 or DB 1<br>(FFFFH: parameter irrelevant) | |
| | Parameter 3 | incorrect parameter 2 in DX 0 or DB 1<br>(FFFFH: parameter irrelevant) | |
| 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 | |

| Parameter type | Structure of the parameter field | |
|---|---|---|
| Table 8-12 continued | | |
| 0BH | Parameter 1 | FFFFH |
| | Parameter 2 | error address high with address code error,<br>actual checksum high when testing the system program code |
| | Parameter 3 | error address low with address code error,<br>actual checksum low when testing the system program code |
| 0CH | Parameter 1 | pattern for 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 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.

*Special Features for the CPU 948R*

In redundant mode (both CPUs in RUN), both CPUs are protected when software protection is activated in BS120 by a user program or programming device. If one or both of the CPUs is in STOP, the software protection is only active on the CPU which the BS120 has set it (by user program or programming device operation), the other CPU remains unchanged. Both CPUs have to be separately protected in this case.
The master status (protected / unprotected, password, delete error counter) is accepted by the reserve when updating. In detail this means:

| Master | Reserve before update | Reserve after update |
|---|---|---|
| Protected | Unprotected | Protected |
| Unprotected | Protected | Unprotected |
| Protected with password 1, delete error counter 1 | Protected with password 2, delete error countr 2 | Protected with password 1, delete error counter 1 |

*Assignment of the system data when writing*

To call the system software protection function, write system data RS 120 with a bit pattern for the function as shown in the following table.
**Address: E F078H**

| **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**

| **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:<br>FB/FX:<br>FB number<br>NAME :DUMMY<br>:BE<br><br>DB/DX: DW0 6500<br><br>OB/PB/SB:<br>:BE |
| Overwrite block<br>(block does not yet exist) | The block is entered |
| Overwrite block<br>(block already exists) | Message "Block exists"; after confirming with the enter key<br>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 RS 251***

**Current cycle time of the S5-155H**
**Address: E F0FBH**

Table 8-13 Bits of BS 251 (current cycle time of the S5-155H)

| **High byte and low byte** | |
|---|---|
| Bit no | Assignment |
| 15 | The entered binary value * 10 ms = the cycle time of the cycle processed last |
| 14 | |
| 13 | |
| 12 | |
| 11 | |
| 10 | |
| 9 | |
| 8 | |
| 7 | |
| 6 | |
| 5 | |
| 4 | |
| 3 | |
| 2 | |
| 1 | |
| 0 | |

## 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 | ’R’ | ’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 | 0 |
| 10 | 0 |
| 9 | 0 |
| 8 | Slot ID CPU 1 |
| | **Low byte** |
| 7 | CPU type:<br>0011 = S5-155H/CPU 948R<br>(only valid in conjunction with the CPU ID) |
| 6 | |
| 5 | |
| 4 | |
| 3 | CPU-ID 2:<br>1000 = S5-155U |
| 2 | |
| 1 | |
| 0 | |

*Word 21*

| Bit no. | High byte |
|---|---|
| 15 | reserved |
| 14 | |
| 13 | |
| 12 | |
| 11 | |
| 10 | |
| 9 | Bit 8 = 0, bit 9 = 1 for "150U" |
| 8 | |
| **Bit no.** | **Low byte** |
| 7 | Release of the PG interface software<br>in the form "xyH"<br><br>Example:<br>13H corresponds to release "V1.3" |
| 6 | |
| 5 | |
| 4 | |
| 3 | |
| 2 | |
| 1 | |
| 0 | |

# Memory Access Using Absolute Addresses

# 9

## Contents of Chapter 9


| | | |
|---|---|---|
| 9.1 | Introduction | 9 - 4 |
| 9.2 | Memory Access via Addresses in ACCU 1 | 9 - 8 |
| 9.2.1 | LIR/TIR: Loading to or Transferring from a 16-Bit Memory Area Indirectly | 9 - 9 |
| 9.2.2 | Examples of Access to DW > 255 | 9 - 15 |
| 9.2.3 | LDI/TDI: Loading to or Transferring from a 32-Bit Memory Area Indirectly | 9 - 17 |
| 9.3 | Transferring Memory Blocks | 9 - 19 |
| 9.4 | Operations with the Base Address Register (BR Register) | 9 - 22 |
| 9.4.1 | Operations for Transfer between Registers | 9 - 23 |
| 9.4.2 | Accessing the Local Memory | 9 - 24 |
| 9.4.3 | Accessing the Global Memory | 9 - 25 |
| 9.4.4 | Accessing the Page Memory | 9 - 29 |

# 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:

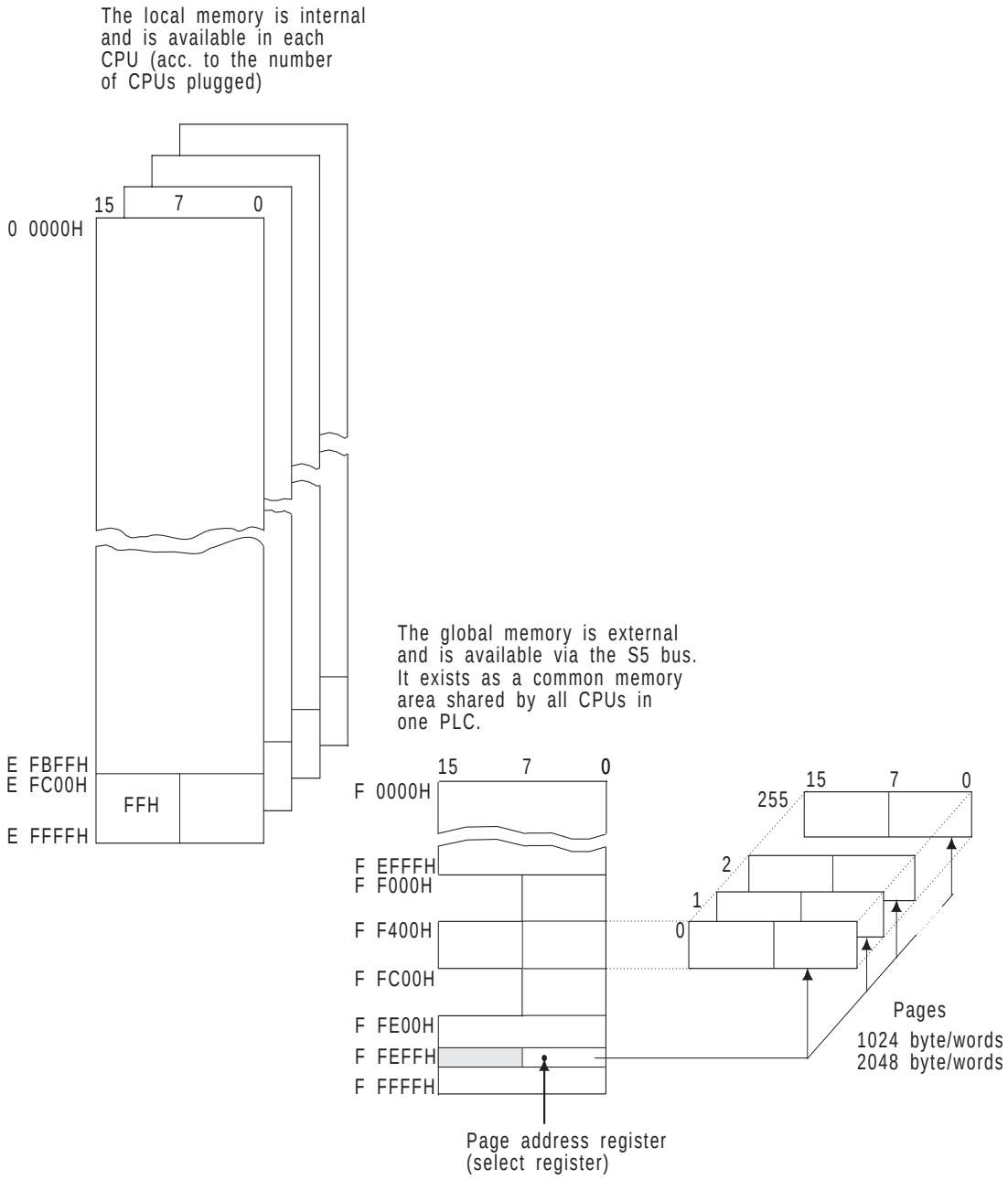



Fig. 9-1 Global and local memory

### *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

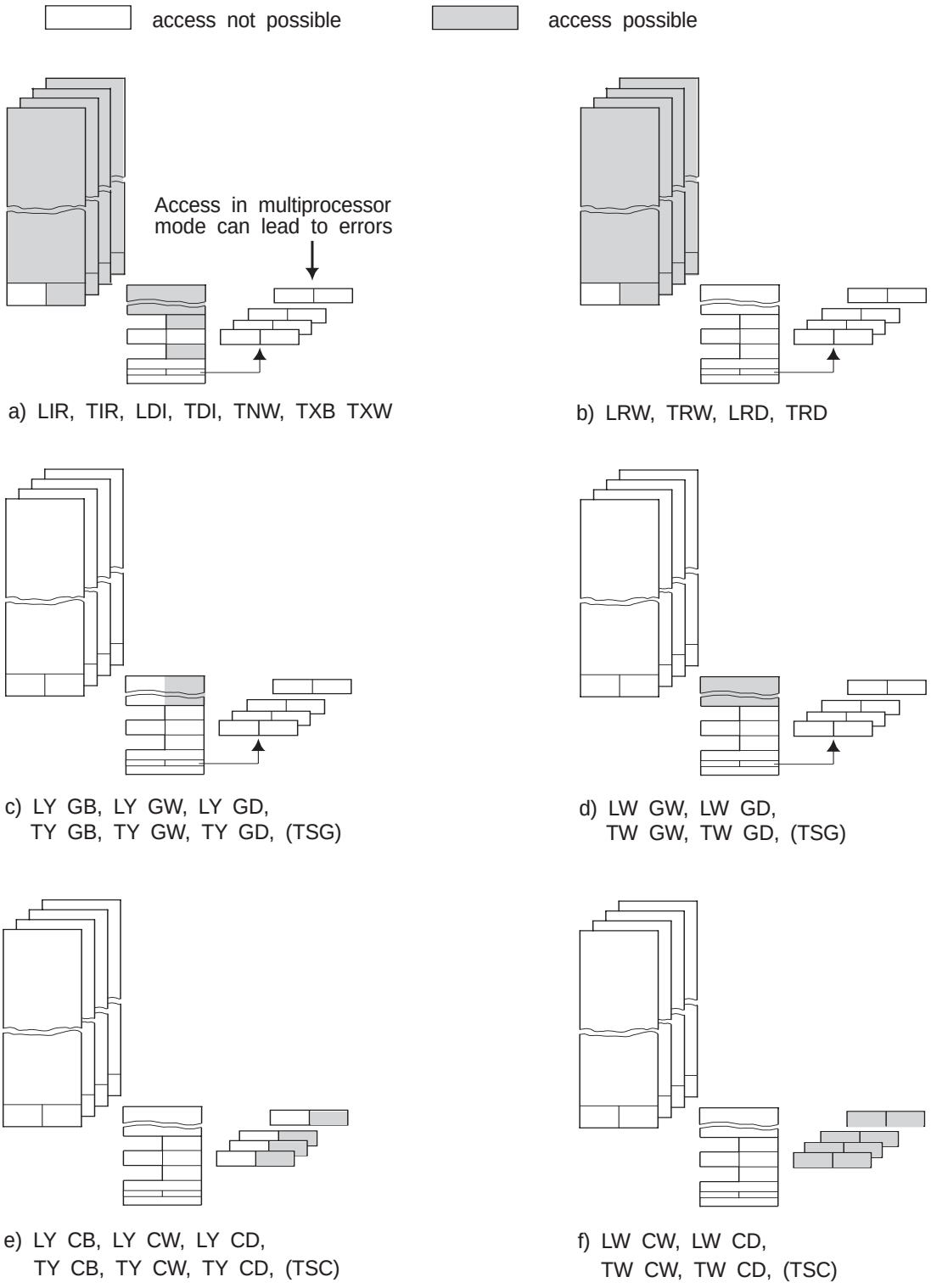



Fig. 9-2 Access to local or global areas using absolute addresses

## 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.

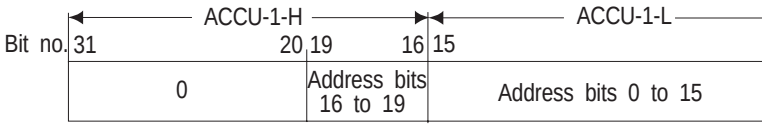
To access these areas, you can use various 16 or 32-bit wide registers. These registers include the accumulators ACCU 1 to ACCU 4 and other special registers used as resources by the CPU.

*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 948R 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 948R. 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 register with **non-defined values** (except flags, the PII and PIQ; in these areas, FFH is written to 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.

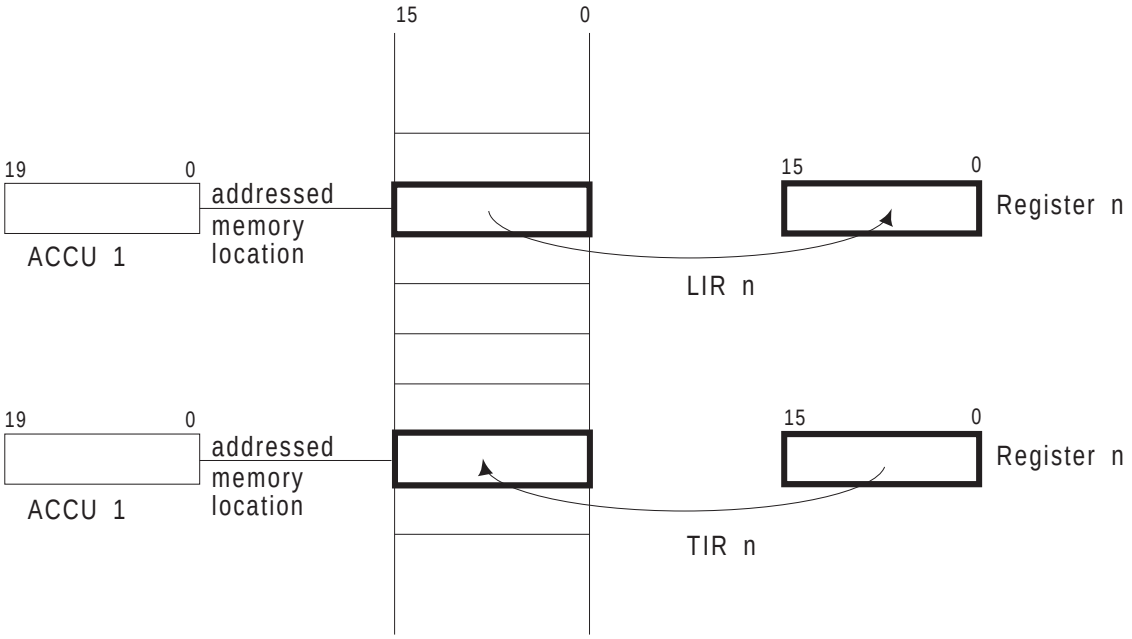



Fig. 9-3 LIR/TIR with 16-bit memory areas (word-oriented)

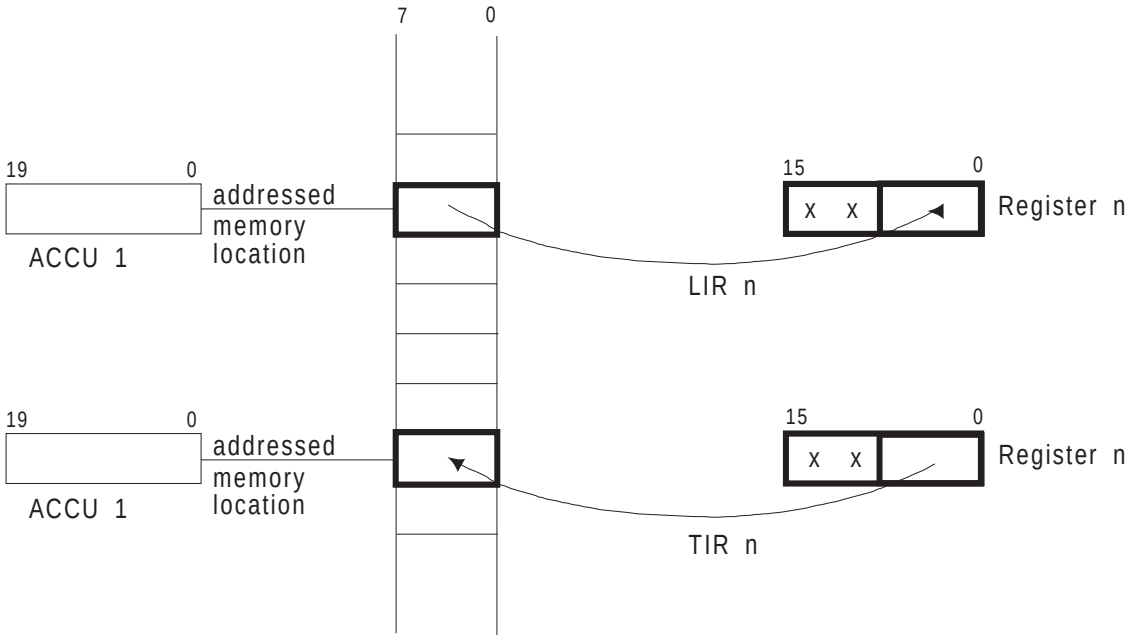



Fig. 9-4 LIR/TIR with 8-bit memory areas (byte-oriented)

*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
   :                 transferred 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:

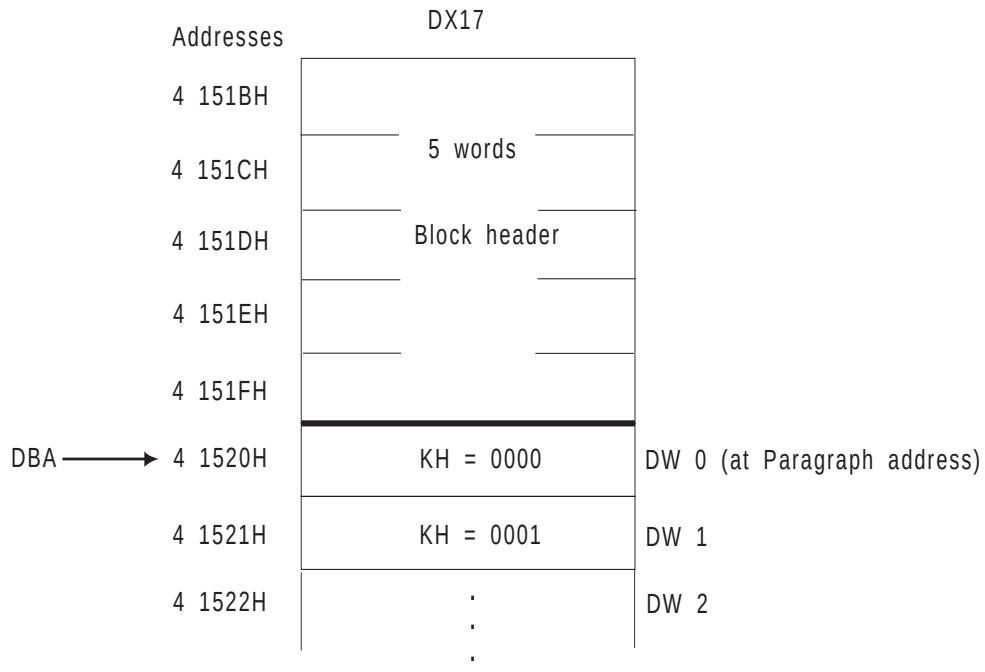



Fig. 9-5 Using the DBA register

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 948R, 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:

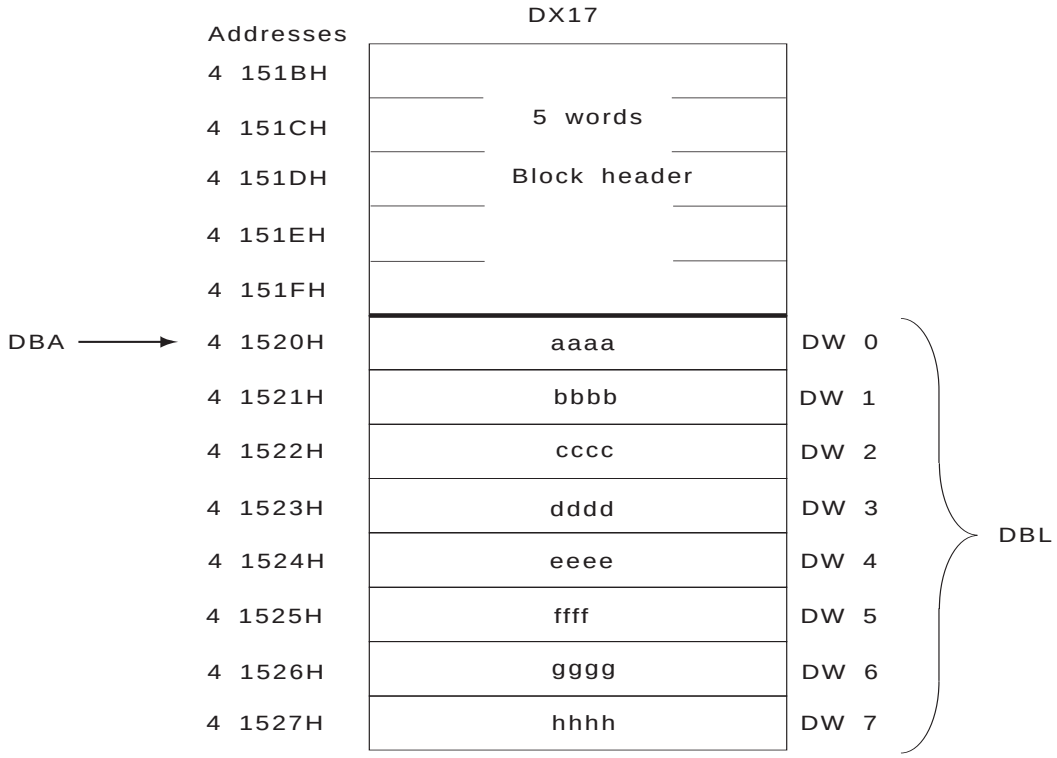



Fig. 9-6 Using the DBL register

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.

ACCU contents within the loop:

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 948R 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 (start address of actual operands, 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*

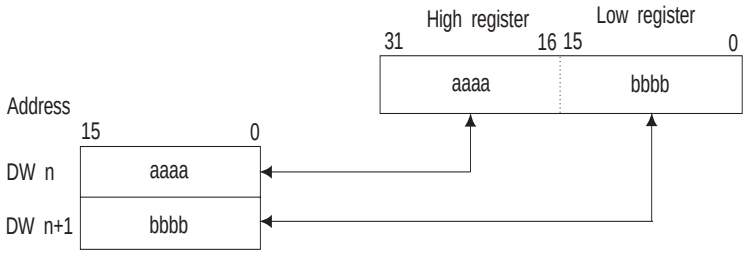
If you reference byte addresses with the LDI or TDI operations, note the following:

- With LDI, **non-defined values** are written to the high bytes of the referenced register (except flags, the PII and PIQ; in these areas, FFH is written to the high byte) ,

  and

- with TDI, only the low bytes of the referenced register are transferred (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: Start Address of the Actual Operands*

During program processing of a function block int the STEP 5 user program, a 20-bit absolute address of the actual operands in the higher order block is entered in the BA register.

*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:<br>16-bit area (dependent on memory configuration) |

| Addresses | Memory area |
|---|---|
| Table 9-5 continued: | |
| E 8000H to E 9FFFH | System RAM:<br>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:**

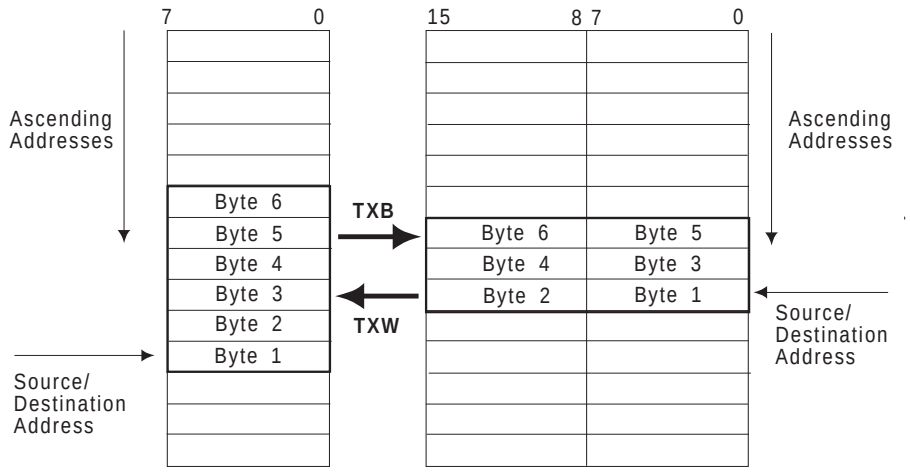



Fig. 9-7 Transferring memory fields

**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.

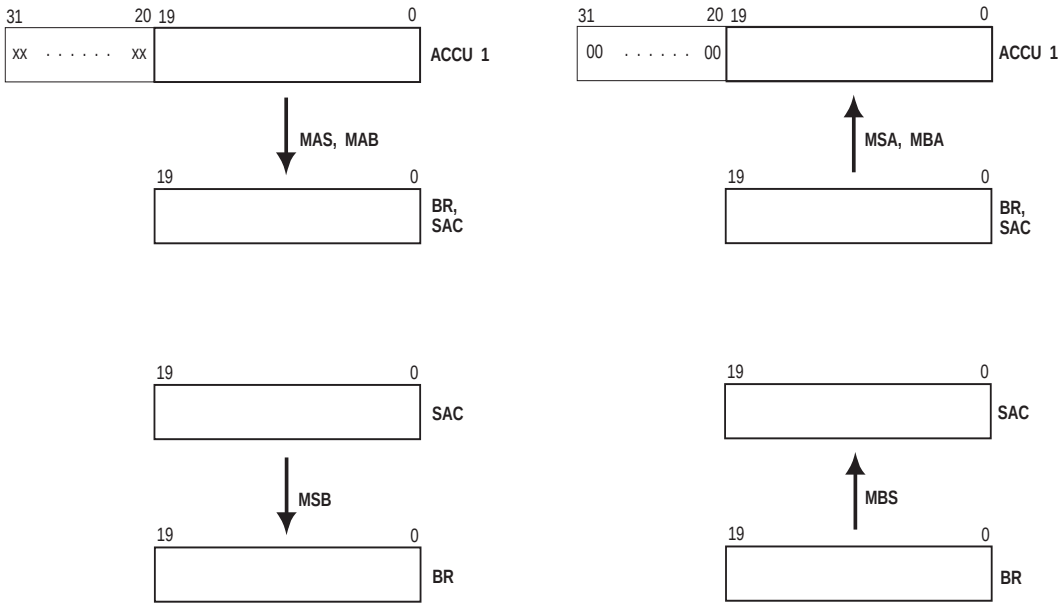



Fig. 9-8 Transfer operations from one register to another

### 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 [1)] of the BR register and load the word addressed in this way in ACCU-1-L |
| LRD | Constant (-32768 to +32767) | add the specified constant to content [1)] of the BR register and load the double word addressed in this way in ACCU 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 Page 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 (pages). One page can occupy a maximum of 2K addresses and can be organized in either bytes or words. Before access to the page 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 page area cannot be interrupted.

Before you can access the page area (load/transfer), you must select one of the 256 dual-port RAM pages. Do this by putting the number of the page that you want to open into ACCU-1-L. Use the ACR operation to enter this number into the CPU-internal **page register.** All page operations that follow ACR write the contents of the page register into the select register of the corresponding modules on the S5 bus before page 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 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 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 page area

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. |

*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 page 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 page memoryorganized 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).

# PG Interfaces and Functions 10

## Contents of Chapter 10


10.1 Overview . . . . . 10 - 4

10.2 Notes on Connecting the PG and the PLC Modes . . . . . 10 - 5

10.2.1 Connecting the PG . . . . . 10 - 5
10.2.2 PLC Modes . . . . . 10 - 6

10.3 PG Functions . . . . . 10 - 8

10.3.1 Info . . . . . 10 - 9
Memory configuration . . . . . 10 - 9
Output DIR . . . . . 10 - 10
Output address . . . . . 10 - 10
10.3.2 Installation . . . . . 10 - 10
Overall reset . . . . . 10 - 10
Compress memory . . . . . 10 - 10
Transfer block . . . . . 10 - 11
Delete block . . . . . 10 - 11
10.3.3 Program Test . . . . . 10 - 11
Start/stop . . . . . 10 - 11
Status block . . . . . 10 - 12
Program test . . . . . 10 - 13
Status variables . . . . . 10 - 16
Force . . . . . 10 - 16
Force variables . . . . . 10 - 17

10.4 Serial Link PG - PLC via the Serial Interface . . . . . 10 - 18

10.5 PG Functions via the S5 Bus . . . . . 10 - 19

10.5.1 Application . . . . . 10 - 19
10.5.2 How the PG Functions via the S5 Bus Work . . . . . 10 - 20
10.5.3 General Notes on Installation . . . . . 10 - 21
10.5.4 Installation with One-Sided Operation . . . . . 10 - 22
The CP 143 is only used for PG functions . . . . . 10 - 23
The CP 143 is used for PG functions and communication SINEC H1 . . . . . 10 - 24
10.5.5 Installation for Switched Operation . . . . . 10 - 26
Configurations . . . . . 10 - 26
Installation . . . . . 10 - 29
Special features . . . . . 10 - 30
Specifying and editing bus paths . . . . . 10 - 30
10.5.6 Displaying Problems . . . . . 10 - 31

# PG Interfaces and Functions 10

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 Section 10.5. This section tells how to connect a PG to your CPU via the S5 bus with SINEC H1. This section also contains points to note if you use PG functions on both interfaces.

## 10.1 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 S5 bus with SINEC H1.

The PG functions via the S5 bus with SINEC H1 can only be used **alternately** with those on the serial interface.

PG functions provide the following support for installing and testing your STEP 5 program:

Table 10-1 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" |

**Note**

The PG functions extend the cycle time required

## 10.2 Notes on Connecting the PG and the PLC Modes

### 10.2.1 Connecting the PG

- If a PG function is interrupted by the cable between the PG and PLC being disconnected or by an error in the processing of a function (timeout etc.) it can take up to 30 sec. before access to the PLC is possible again. Requests for functions during this time are rejected with the message -"AS function disabled: function active".

- In the redundant mode, with longer cycle times (greater than 500 ms), some functions called with the LAD, CSF, STL package cause the monitoring time set on the PG to be exceeded (PLC timeout). **Remedy**: switch on timed interrupts in the DX 0 screen form.

### 10.2.2 PLC Modes

Remember that the S5-155H has four modes:

1. Solo mode (only the master in RUN),
2. Coupling standby,
3. Redundant mode,
4. Error search mode (error search is only performed by the standby; during this time, the master is in the solo mode). In terms of the PG functions, this mode has the same effect as the coupling standby mode.

#### *Working at the PG during the solo and coupling standby modes*

*Solo mode*

**PG online with standby PLC:**



**Caution**
If programs are modified on a standby PLC in the STOP mode, the changes are lost as soon as the standby PLC is connected to the master!

*Coupling standby*

During the coupling standby phase, all write access "PG – PLC" both to the master and to the standby is rejected with the error message "Wrong mode at PLC". The status "coupling standby" is not directly recognizable on the PG.

### *Working at the PG during the redundant mode*

*General information about the PG functions*

While the redundant mode is active, both PLCs are in the RUN mode. PG functions are possible with both PLCs. The master and standby are ranked equally.

- All **read functions** in the redundant mode are the same as with the CPU 948. Only the data of the connected PLC are output. This means that with the functions **"Status" of formal operands on F page** (F 0000 to F FFFF) and **"Output address" on F page**, no valid values are indicated at the standby PLC which cannot access the switched I/Os.

- The effects of all **write functions** in the redundant mode are extended compared with those of the CPU 948. The data entered are transferred to the partner PLC at the cycle checkpoint. All write functions therefore take longer than with the U system.

*Working with subunit A and subunit B simultaneously*

If a programmer is connected to both subunits, remember the following points :

- Read functions on one side are generally possible at the same time as **write functions** on the partner.

- "Output address" (read) at the same time as, for example "Input block" (write) is, however, pointless and is therefore not permitted.

- Write functions are permitted on both sides. If the write functions are simultaneous, only the **first to be recognized** is executed. The second is rejected with the error message "AS function disabled: function active".

## 10.3 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 948R 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 yet updated the PII.

**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 10.2.3).

**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 (block for cyclic program execution in RUN).

## 10.3.1 Info

***Memory configuration***

The CPU 948R 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 STEP 5/ST V6.3 upwards or STEP 5/MT V6.0 both 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 mentioned above 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 946R/947R (refer to Fig. 10-1). The total memory configuration of the CPU 948 must then be calculated from the total of the submodule values.

P L C  i n f o  SIMATIC S5 / OES0C

M e m o r y  configuration  S 5  1 5 5  U

| Module | Submod | Type | Start address | End address | Length | |
|---|---|---|---|---|---|---|
| 0 | 1 | RAM | 00000 | 0FFFF | 64 KW | 1) |
| 1 | 1 | RAM | 10000 | 2FFFF | 128 KW | |
| 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 948R-1: 320 Kw = 640 Kbytes

Fig. 10-1 PG display of the memory configuration with older PG software

***Output DIR***

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'.

### 10.3.2 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).

> **Note**
> The "delete all blocks" function is not permitted in the RUN mode. If you attempt to execute the function, it will be rejected with the message "Wrong mode at PLC" or "Memory error on PLC".

***Compress memory***

The function is permitted in the solo and redundant modes.

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.

*Restrictions when transferring from FD*

If you want to transfer blocks from FD to PLC, remember the following restrictions:

1: "TRANSFER SOURCE : FD BLOCK [DX1] TO DEST: PC"

This function is not permitted in RUN. if you attempt this, the system message "Block type/no. illegal" is displayed.

2. "TRANSFER SOURCE : FD BLOCK [RAM-DB] TO DEST: PC"

**The RAM-DB must not be transferred in RUN !** If you attempt this, the system message "DMA interface not ready" or "Timeout" is displayed.

***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.

### 10.3.3 Program Test

***Start/stop***

These functions on the PG correspond to manual operation.

The "Start" function is only effective on the subunit to which your PG is connected.

With the "Stop" PG function in the RUN mode, you stop the subunit to which the PG is connected.

The COM H functions "PLC START" and "PLC STOP" affect either subunit A or B as required or both subunits.

You exit the SOFT STOP status with a COLD RESTART or COLD RESTART WITH MEMORY. The CPU exits the stop mode.

### *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.

If the actual operands involve direct access to the I/Os then

- with one-sided I/Os, the actual values are only displayed for the subunit in which the I/Os are plugged in,
- with switched I/Os only the actual values for the master are displayed.

The "status block" function (with correction) may extend the time for the master-standby switchover.

### *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**).

### *Program test*

The function is not permitted in the redundant mode.

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.

> **Note**
> You must not select a substitution operation as the breakpoint with DO FW/DO DW otherwise the function is aborted and the CPU changes to the cyclic mode.

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.

#### *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).

*Interruptions*

- Program processing → STOP mode:

  If an interruption occurs during program processing (e.g., STOP switch, 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 COLD RESTART WITH MEMORY, 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 next 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 at your PG

- Output ISTACK
- Output BSTACK
- Load block
- Read block
- Delete block
- Output block list
- Force variables
- Force

### *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).
>
> With the CPU 948R, the "force" function is possible with switched I/Os providing the system configuration (DX 1) is loaded and both PLCs are in the STOP mode.

*Sequence of the function*

When you call the "force" function in the STOP mode, the disable output command signal is cancelled (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.

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.

## 10.4 Serial Link PG - PLC via the 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 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.

## 10.5 PG Functions via the S5 Bus

### 10.5.1 Application

The PG functions via the S5 bus with SINEC H1 S5-155H programmable controllers with the CPU 948R networked via SINEC H1 to be loaded and controlled from the PG 7xx. Using the PG functions via the S5 bus the CPU 948R can be loaded in the STOP mode up to eight times faster than via the PG interface. The speed depends on the length of the blocks to be transferred.

The PG functions via the S5 bus are part of the system program of the CPU 948R.



**Caution**
The PG functions via the S5 bus can **only** be used **alternately** with the PG functions via the serial interface.
On the CPU 948R, interface numbers ≥ 232 are reserved for the PG functions via the S5 bus. These numbers cannot be used freely for handling blocks.

*Assigning parameters to the CPU*

Before you can use the PG functions via the S5 bus in the RUN mode, you must first configure the system (CP interfaces, switched EUs ...) with COM 155H in data block DX 1 and transfer the data block to the PLC.

*Technical requirements*

The PG functions via the S5 bus with the CPU 948R require that the PG and PLC are networked via SINEC H1.

The following is necessary:

- a PG 7xx with a SINEC H1 connection and with STEP 5 software version 6.3 (ST) or 6.0 (MT) + delta diskette CPU 948 installed.
- in the PLC (central controller or expansion unit EU 185) there must be a communications processor CP 143 version 06 or higher (firmware version 3.0) with selected base interface number 232 or 236 (the base interface number is set on the hardware with jumpers and in the SYSID with COM 143).

**Note**
For the PG functions via the S5 bus, CP 143 switched operation must not be mixed with one-sided CP 143 operation.

### 10.5.2 How the PG Functions via the S5 Bus Work

*Using pages*

With the S5 bus functions, the pages of the CP are divided into two pages for user HDBs and two pages for PG functions.
The pages for user HDBs can be used as normal for SINEC H1 applications. Remember, however, the special points listed in Section 10.5.4 .
The pages for PG functions are used by the CP 143 and the CPU 948R for 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 on the CP 143 automatically, when you set **base interface number 232 or 236** on the CP (jumpers **and** SYSID!). You then occupy interface numbers 232 to 239. The interface numbers 240 to 247 are reserved for later expansions (e.g. CPs with 8 pages/interfaces).

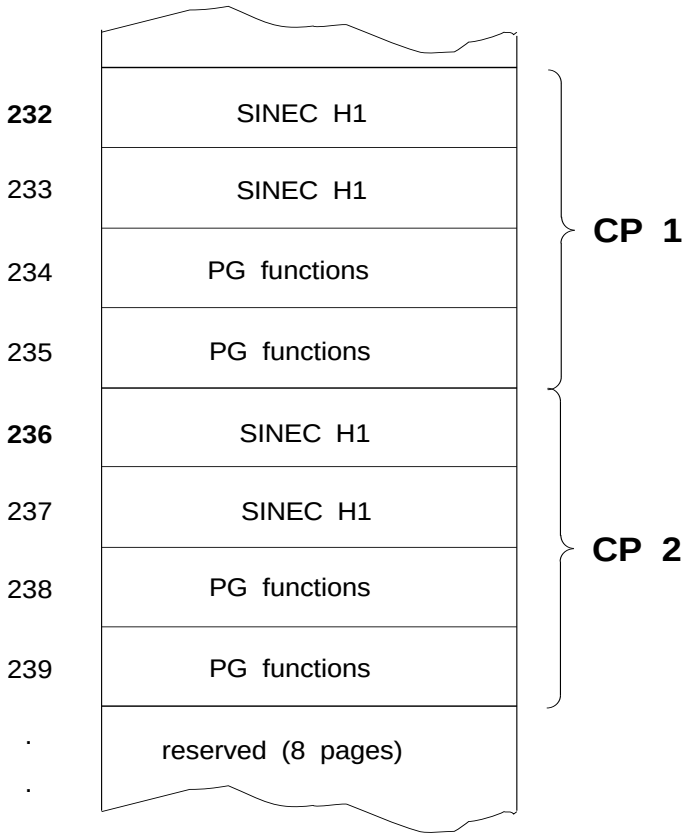



Fig. 10-2 Interface assignment of the PG functions via the S5 bus

*Assigning parameters to the CP 143*

Parameter assignment for the CP 143 is described in the CP 143 manual /6/.



**Caution**
The interface numbers 232ff and 236ff must not be used when the CP 143 is operating with other SIMATIC CPUs. When using the CPU 948R with other CPs interface numbers 232 to 247 must not be used.

### 10.5.3 General Notes on Installation

#### *Installing the link*

*Installing the CP 141x in the PG 7xx*

This step is only necessary when you add the CP 141x to your PG 7xx at a later point in time.

Installing the CP 141x module (including parameter assignment) is described in the Product Information for the CP 141x, which is supplied with the module.

*Installing the CP 143 in the S5-155H*

Installing the CP 143 is described in Part I of Volume 1 of this manual under "CP 143 with COM 143".

The slots permitted for installation of the CP 143 in the S5-155H can also be found in Part I of Volume 1 of this manual.

*Connecting the PG 7xx and CP 143*

Connect the CP 143 and the PG 7xx with th cable supplied with the PG 7xx.

#### *Points to note*

- At any one time, a subunit of the CPU 948R processes only one PG function. If you attempt to activate a second function from a second PG via the serial PG interface while a PG is running, the message "AS function disabled: function active" is displayed on the second PG..
- If you abort a PG function by changing the mode selector on the CPU from RUN to STOP or if an error occurs, this results in waiting times of longer than 15 sec. in the communication via SINEC H1.
- If you make a mistake on the PG (e.g. for example you switch off the PG while a function is active, it may be necessary to repeat the "Selection" function when the path is re-established.

### 10.5.4 Installation with One-Sided Operation

The Figures 10-3 and 10-4 illustrate two possible configurations with a one-sided CP 143

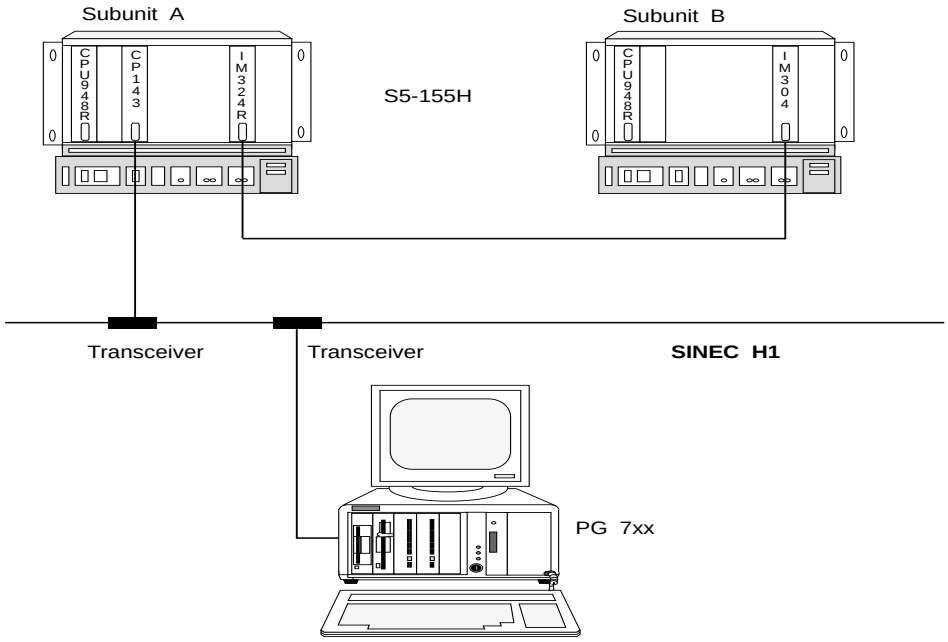



Fig. 10-3 One-sided operation, configuration 1: with **one** one-sided CP 143

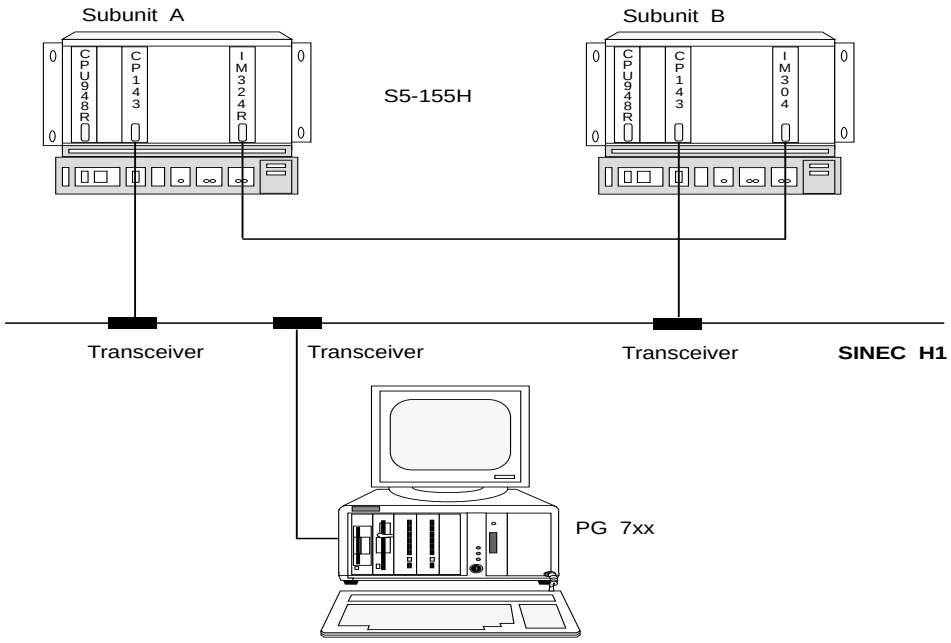



Fig. 10-4 One-sided operation, configuration 2: with **two** one-sided CP 143s

During installation, remember the two following alternatives:

***The CP 143 is only used for PG functions***

After switching the POWER ON; the PG functions via the S5 bus on the CPU 948R are always available without the CP 143 being previously synchronized using HDB SYNCHRON (FB 125).
**An "empty" CPU 948R can be started up via the S5 bus without an OVERALL RESET.**

The CPU 948R synchronizes the pages for PG functions via the S5 bus on the CP 143 automatically after POWER UP.

The following steps are necessary:

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td>1</td><td colspan="2">Set the interface number (SSNR) on the CP 143 (jumpers):<br>Select the SSNR according the existing hardware configuration as described below.<br>Remember the instructions in /6/.</td></tr>
<tr><td></td><td>Possible hardware configuration</td><td>Corresponding SSNR of the CP 143(s)</td></tr>
<tr><td></td><td>Configuration 1<br><br>Configuration 2</td><td>232 or 236<br><br>232 on 1st CP, 236 on 2nd CP</td></tr>
<tr><td>2</td><td colspan="2">Plug the CP 143 into the S5-155H (the power to the PLC must be off!).</td></tr>
<tr><td>3</td><td colspan="2">Connect the PG to the PG interface of the CP 143 and load COM 143.</td></tr>
<tr><td>4</td><td colspan="2">Set the same hardware interface number as in step 1 and the Ethernet address in the SYSID of the CP 143 with COM 143.</td></tr>
<tr><td>5</td><td colspan="2">Load the parameter data on the CP 143:<br>The parameter data of the CP 143 can be stored<br>- in an EPROM<br>or<br>- in the RAM of the CP 143.<br><br>You can transfer the parameter data via the serial interface of the PG 7xx. The individual steps for loading the parameter data of the CP 143 can be found in /6/.</td></tr>
<tr><td>6</td><td colspan="2">Switch the power for the PLC on again.</td></tr>
<tr><td>7</td><td colspan="2">Edit the path to the CPU 948R in the STEP 5 bus selection screen.</td></tr>
<tr><td>8</td><td colspan="2">Select the path via SINEC H1/CP 143 to the CPU 948R in the STEP 5 presets screen.</td></tr>
</table>

The PG functions via the S5 bus can be used after these steps have been taken. You can now load and run or test your user program.

### *The CP 143 is used for PG functions and communication via SINEC H1*

If you want to use the CP 143 for communication via SINEC H1 as well as for the PG functions via the S5 bus, you must make further settings in addition to those described above and must take into account further special features.

When installing, follow the procedure outlined below:

| Step | Action |
|---|---|
| 1 to 7 | Steps 1 to 7 are identical to those when the CP 143 is used only for PG functions. |
| 8 | **In the start-up OBs, OB 20 and OB 22, program the call for the HDB SYNCHRON (FB 125), so that during MANUAL/AUTOMATIC COLD RESTART and AUTOMATIC COLD RESTART WITH MEMORY, the CP 143 is synchronized for SINEC H1 communication.** |

### *Pages for communication via handling blocks*

Fig.10-5 illustrates how the PG functions via the S5 bus occupy the pages of the CP 143. The free pages for user HDBs can be used by the CPUs "subunit A" and "subunit B" for communication via SINEC H1.

### *Special features*

In the following situations, the link between the PG and CPU must be re-established:

- After a change from STOP to RUN
  (Owing to this reaction, you cannot use the PG function **"status block"** via the S5 bus for **start-up OBs**
- after depassivation
- after a change to the redundant mode.

.

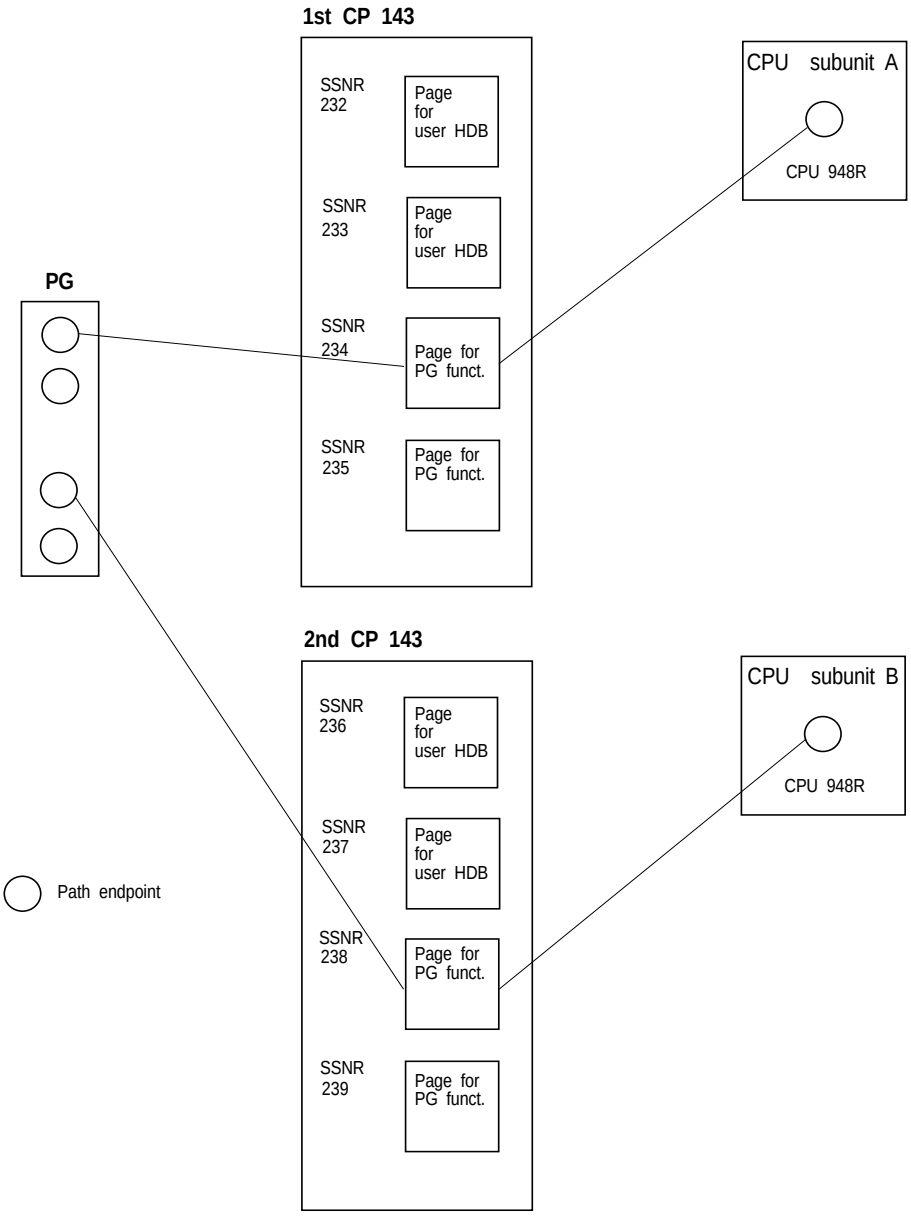



Fig. 10-5 Paths between PG and CPU 948R and assignment of the pages of the CP 143 in one-sided operation

**Note**
When you start up with COM 155H, the connected PLC automatically becomes master; the standby PLC then automatically accepts the user program during START-UP.

## 10.5.5 Installation for Switched Operation

### *Configurations*

Each switched EU has two IM 314R (bus converters). The bus converters connect the master PLC to the EU directly.

The EUs must be set to EU numbers ’14’ and/or ’15’ using the DIP switches on the IM 314R. You must also configure the EUs and CPs in data block DX 1 with COM 155H.

The following three figures illustrate possible configurations of the PG functions via the S5 bus with switched expansion units.

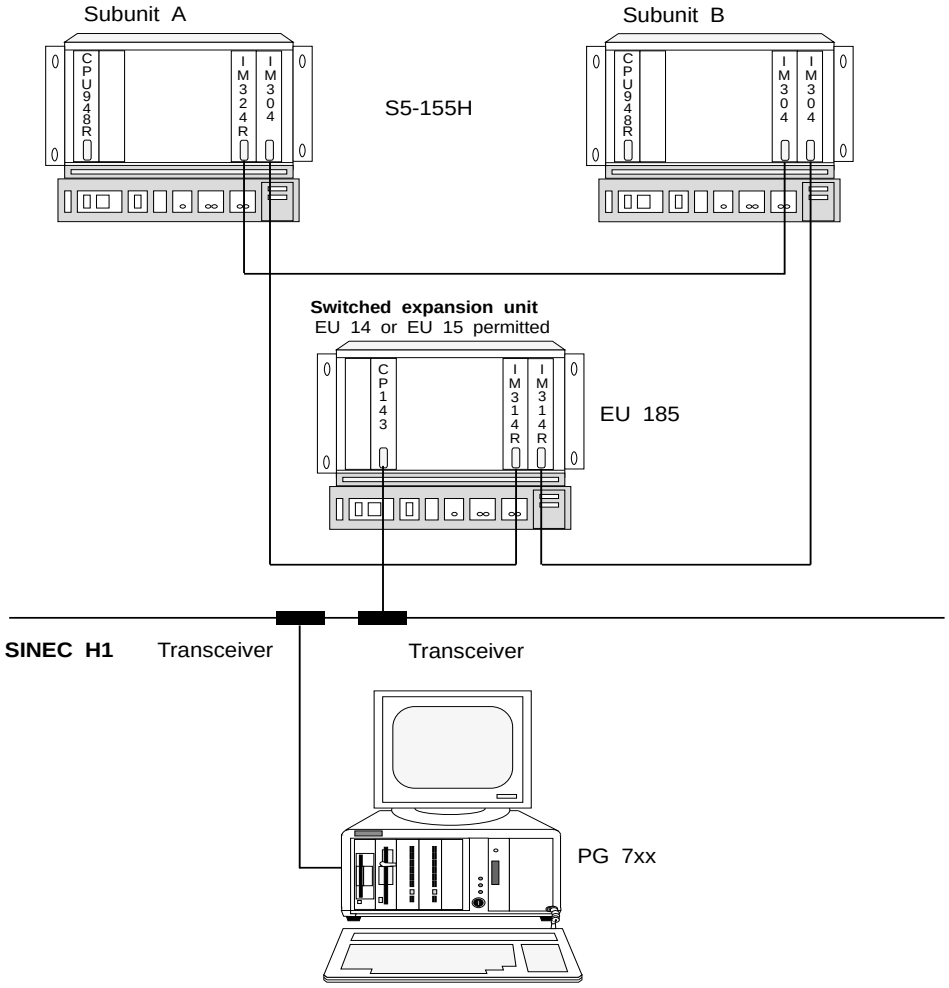



Fig. 10-6 Basic configuration with switched EU: operation with **one** CP 143

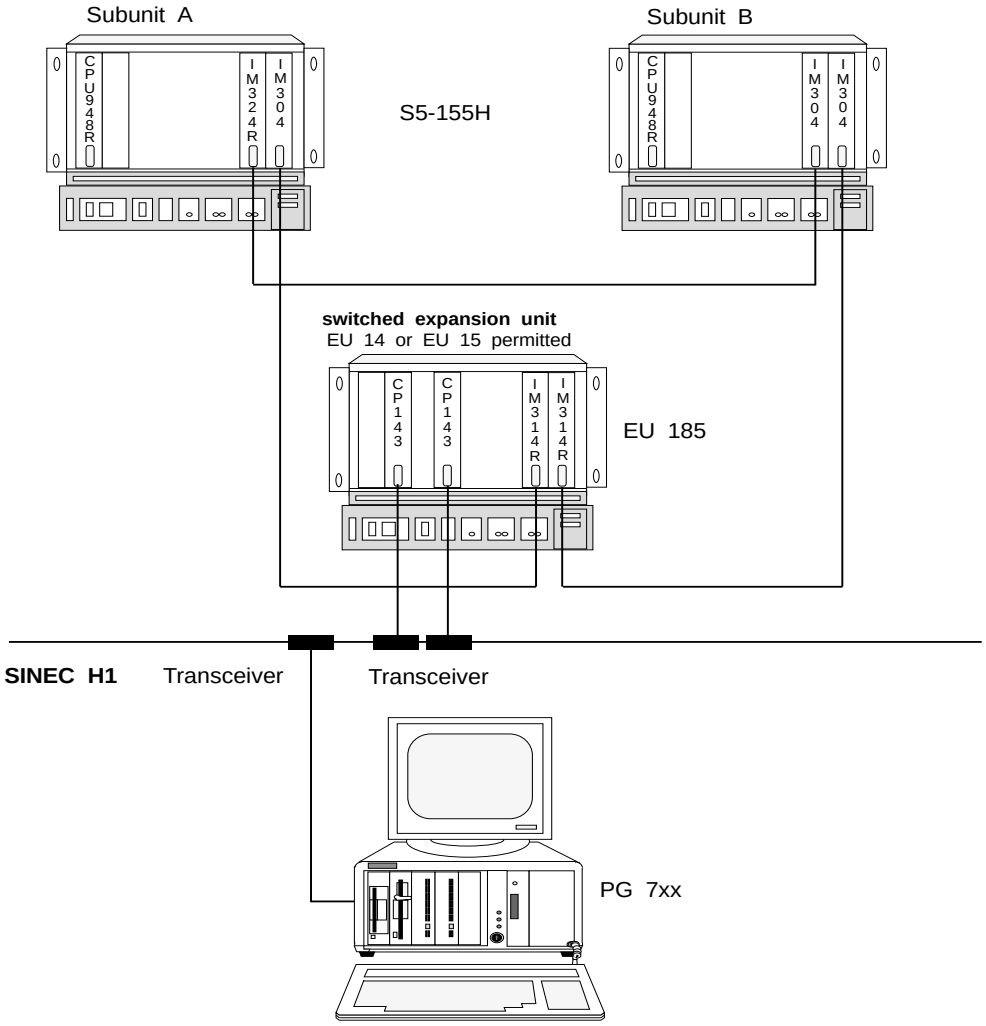



Fig. 10-7 Configuration 1 of the basic configuration: operation with **two** CP 143s (increased availability)

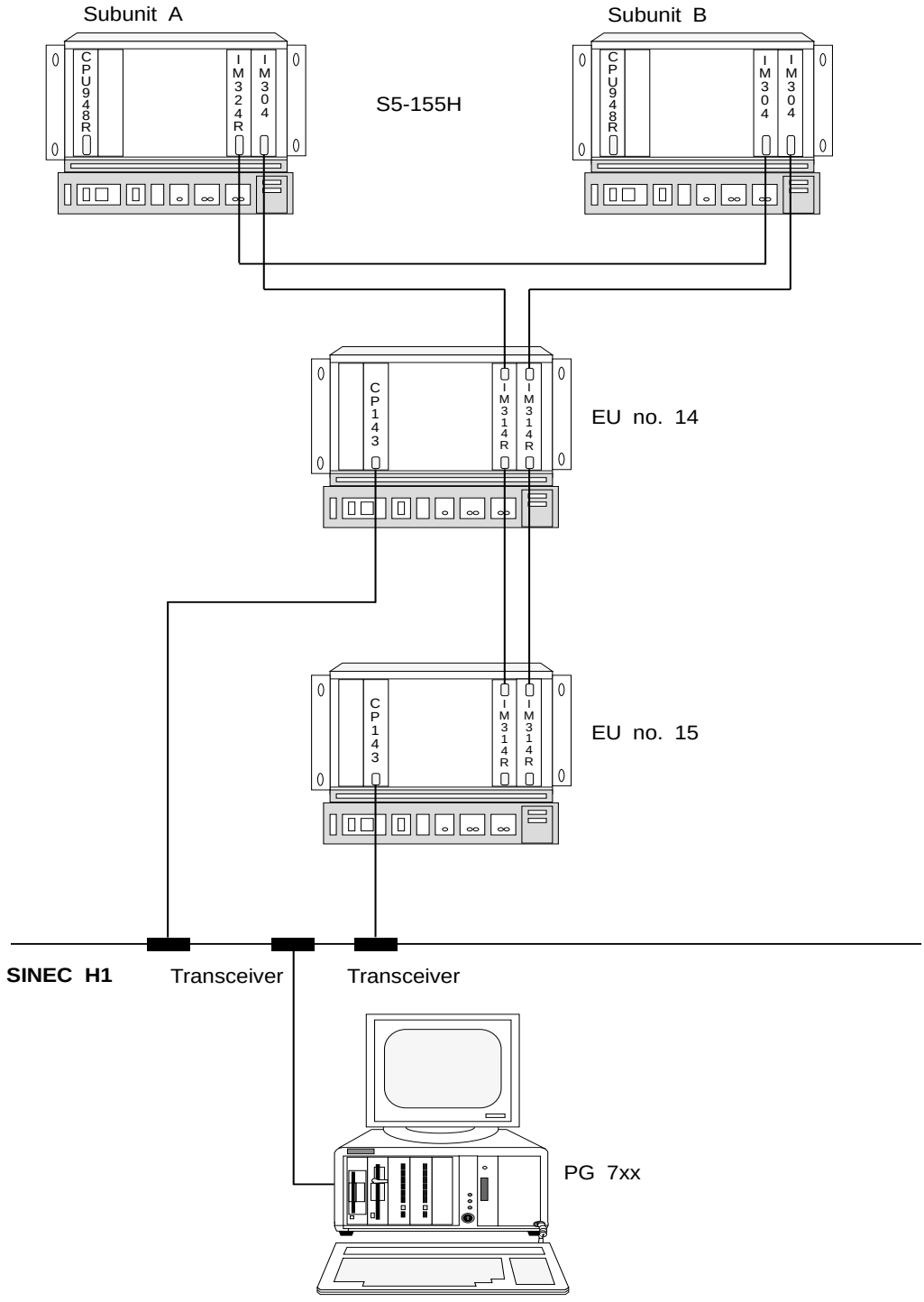



Fig. 10-8 Configuration 2 of the basic configuration: operation with **two** CP 143s (increased availability)

***Installation*** The following steps are necessary during installation:

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td>1</td><td colspan="2">Set the interface number (SSNR) on the CP 143 (jumpers):<br>Select the SSNR according the existing hardware configuration as described below.<br>Remember the instructions in /6/.</td></tr>
<tr><td></td><td>Possible hardware configuration</td><td>Corresponding SSNR of the CP 143(s)</td></tr>
<tr><td></td><td>Basic configuration<br><br>Configuration 1<br><br>Configuration 2</td><td>232 or 236<br><br>232, 236<br><br>232, 236</td></tr>
<tr><td></td><td colspan="2">The interface numbers 232 and 236 are reserved exclusively for PG functions via the S5 bus.<br>For all other CPs and IPs SSNR < 232 must be selected.<br>Each CP 143 has four pages that are used as follows for communication:<br>SSNR + 0: for user communication via HDB page<br>SSNR + 1: for user communication via HDB page<br>SSNR + 2: system page for PG functions via S5 bus<br>SSNR + 3: reserved</td></tr>
<tr><td>2</td><td colspan="2">Plug the CP 143 into the S5-155H (the power to the PLC must be off!).</td></tr>
<tr><td>3</td><td colspan="2">Connect the PG to the PG interface of the CP 143 and load COM 143 .</td></tr>
<tr><td>4</td><td colspan="2">Set the same hardware interface number as in step 1 and the Ethernet address in the SYSID of the CP 143 with COM 143.</td></tr>
<tr><td>5</td><td colspan="2">Load the parameter data on the CP 143:<br>The parameter data of the CP 143 can be stored<br>- in an EPROM<br>or<br>- in the RAM of the CP 143.<br><br>You can transfer the parameter data via the serial interface of the PG 7xx. The individual steps for loading the parameter data of the CP 143 can be found in /6/.</td></tr>
<tr><td>6</td><td colspan="2">Set the mode selector on the CPU to RUN.</td></tr>
<tr><td>7</td><td colspan="2">Switch the power for the PLC on again.</td></tr>
<tr><td>8</td><td colspan="2">Edit the path to the CPU 948R in the STEP 5 bus selection screen.</td></tr>
<tr><td>9</td><td colspan="2">Select the path via SINEC H1/CP 143 to the CPU 948R in the STEP 5 presets screen.</td></tr>
</table>

### Special features

The CPU 948R synchronizes the pages for PG functions via the S5 bus on the CP 143 automatically after POWER UP. The mode selector must be set to RUN.

For cyclic operation, the four interface numbers must be assigned with COM 155H.

In the following situations, the link between the PG and CPU must be re-established:

- After a change from STOP to RUN
  (Owing to this reaction, you **cannot** use the PG function **"status block"** via the S5 bus for **start-up OBs**.
- after a change from RUN to STOP,
- after a master-standby switchover,
- after depassivation
- after a change to the redundant mode.

> **Note**
> When you start up with COM 155H the **subunit which has access to the switched expansion unit at the start** is automatically started first (the subunit with access can be recognized by the green LED lit on the assigned IM 314R module).
> **The other PLC** has the loaded user program transferred to it automatically at START-UP. You recognize the master unit by the green LED lit on the assigned IM 314R module.

### Specifying and editing bus paths

Specify the paths you want to use in the following test to match your configuration.

Using the STEP 5 software package, edit the bus paths and save them under a suitable name in the path file.
When you load and test the user program, you simply have to enter the file name and path name in the presets screen.
The CPU 948R only accepts one established path for PG communication at any one time.

Paths can only be activated at the PG.

### 10.5.6 Displaying Problems

The CPU 948R (subunit A or subunit B) sets error bits in its RT area if errors occur during PG functions via the S5 bus.

These bits are set in a parameter assignment error byte (PAFE) for every possible connection and a condition codeword (ANZW) to indicate the current status of the Send and Receive blocks.
The bits are largely the same as those of the handling blocks.

*RT area*

An information field of 16 words is set up in the RT area of the CPU 948 only when the pages for PG functions exist and the connection to the CP 143 is established. The structure of this area is described below.

**Note**
The RT area is **reset** after an OVERALL RESET!

If you use the PG functions via the S5 bus, the RT area is occupied as described below and is then no longer available for other programs (e.g. standard FBs). Please remember this when configuring your system.

As long as there is no connection to the CP 143 (PAFE = 71), for example, because no page with SSNR 232ff or 236ff exists, **no additional information** is entered in the RT area.

*Structure of the information field*

| Address | | | |
|---|---|---|---|
| E F2E8H | Condition codeword for data direction from CPU to PG for SS 234 or 238 | | RT 232 |
| E F2E9H | Condition codeword for data direction from CPU to PG for SS 235 or 239 | | RT 233 |
| E F2EAH | reserved | | RT 234 |
| E F2EBH | reserved | | RT 235 |
| E F2ECH | Data direction from PG to CPU for SSNR 234 or 238 | | RT 236 |
| E F2EDH | Data direction from PG to CPU for SSNR 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 238 | RT 242 |
| E F2F3H | reserved | | RT 243 |
| E F2F4H | | | RT 244 |
| E F2F5H | | | RT 245 |
| E F2F6H | | | RT 246 |
| E F2F7H | | | RT 247 |

*Meaning of the PAFE values*

All errors occurring in the interaction with the CP 143 are indicated. The following errors can be indicated in PAFE:

| **PAFE value** | **Meaning** |
|---|---|
| 00H | no error |
| 71H | interface (page) does not exist |
| 81H | interface not ready |
| 91H | interface overloaded |
| A1H | interface being used by other CPU |
| B1H | job number of field size illegal (FB SYNCHRON) |
| C1H | interface not reacting or not in time |
| D1H | other interface errors, including error messages for the CP |

**Meaning of value 71H:**

The value 71H means that the page does not exist. The PG functions via the S5 bus cannot be started if this message occurs. In this case, check the interface assignment of the CP 143: interface numbers 232 ff or 236 ff must be set (jumpers and in SYSID!).

*ANZW*

ANZW contains the current status of the Send and Receive blocks. The bits have the following meaning:

| **High byte** | |
|---|---|
| Bit no. | Meaning |
| 15 | not used |
| 14 | |
| 13 | |
| 12 | |
| 11 | |
| 10 | |
| 9 | |
| 8 | |
| **Low byte** | |
| 7 | not used |
| 6 | data received |
| 5 | data transferred |
| 4 | 1: error |
| 3 | job terminated with error |
| 2 | job completed without error |
| 1 | 0: SEND enabled [1)]<br>1: SEND disabled |
| 0 | 0: RECEIVE disabled [1)]<br>1: RECEIVE enabled |

[1)] only for PG functions via the S5 bus

# Appendix 11

## Contents of Chapter 11

Technical Data of the CPU 9487R and CPU 946R/947R. . . . . 11 - 4

# 11

# Appendix

This chapter contains technical data of the CPU 948R and the CPU 946R/947R as an overview table.

## Technical Data of the CPU 948R and CPU 946R/947R

| Operation / Processing | CPU 948R | CPU 946R/947R |
|---|---|---|
| Typical execution times of bit operations | | |
| with<br>F, I, Q<br>D<br>Formal operand | <br>0.18 µs<br>0.70 µs<br>0.91 µs | <br>1.4 µs<br>2.4 µs<br>2.7 µs |
| Typical execution times of word operations: | | |
| - Load operations<br>L FY (byte)<br>L FW (word)<br>L FD (double word)<br><br>- Fixed point arithmetic<br>- Floating point arithmetic | <br>0.18 µs<br>0.50 µs<br>0.71 µs<br><br>0.55 ... 3.8 µs<br>3.3 ... 6.3 µs | <br>1.5 µs<br>1.8 µs<br>2.8 µs<br><br>1.0 ... 8.7 µs<br>12.8 ... 18 µs |
| Cyclic program execution | | |
| Basic load calling OB 1: | 10 ms | 30 ms |
| Additional time for process image update:<br><br>per DI byte, three-channel:<br>per DI byte, redundant:<br>per DI byte, switched:<br>per DI byte, one-sided:<br><br>per DQ byte, redundant:<br>per DQ byte, switched:<br>per DQ byte, one-sided: | <br><br>200 µs<br>50 µs<br>25 µs<br>40 µs<br><br>100 µs<br>4 µs<br>25 µs | <br><br>400 µs<br>136 µs<br>60 µs<br>86 µs<br><br>236 µs<br>5 µs<br>64 µs |
| Additional time for IPC flag transmission:<br><br>per input:<br><br>per output: | <br><br>25 µs<br><br>4 µs | <br><br>60 µs<br><br>5 µs |
| Additional time for timer processing depending on timer field length (TFL)<br>TFL =0<br><br>TFL #0<br>n = number of timers running (interval: 10 ms) | <br>every 10 ms<br>11.6 µs<br><br>11.6 µs<br>+ TFL * 0.32 µs | <br>every 10 ms<br>4.4 µs<br><br>4.4 µs<br>+ TFL * 1.3 µs |

| Operation / Processing | CPU 948R | CPU 946R/947R |
|---|---|---|
| Interrupt-driven program execution | | |
| Extension of the cycle time<br>by nesting an empty<br>OB 2 (without STEP 5 operations)<br>at a block boundary | 280 µs | 430 µs |
| Reaction time | 200 µs | 300 µs |
| Time-controlled program execution | | |
| Extension of the cycle time<br>by nesting an empty<br>OB 13 (without STEP 5 operations)<br>at a command boundary | 300 µs | 430 µs |
| Clock pulse for calling the time-controlled program<br>(timed interrupts OB 10 to OB 18) | variable<br>basic clock pulse from<br>1 to 255 ms.<br>specified relative<br>to 10 ms:<br>10, 20, 50, 100<br>200, 500 ms,<br>1, 2, 5 s<br>or<br>10, 20, 40, 80, 160,<br>320, 640 ms,<br>1.28, 2.56 s | variable<br>basic clock pulse from<br>1 to 255 ms.<br>specified relative<br>to 10 ms:<br>10, 20, 50, 100<br>200, 500 ms,<br>1, 2, 5 s<br>or<br>10, 20, 40, 80, 160,<br>320, 640 ms,<br>1.28, 2.56 s |
| Resolution for real time interrupt<br>(OB 9) | every minute,<br>hourly,<br>daily,<br>weekly,<br>monthly,<br>annually,<br>once | – |
| Resolution for<br>delayed interrupt (OB 6) | 1 ms | – |
| Cycle time monitoring | | |
| Selectable between<br><br>triggerable | 200 ms<br>1 ... 2550 ms<br><br>yes | 200 ms<br>1 ... 2550 ms<br><br>no |

| Operation / Processing | CPU 948R | CPU 946R/947R |
|---|---|---|
| Memory capacity | | |
| Size of the user memory module (in Kbytes) | 640 or 1664 | 128 + n * 128 (n = number of submodules in memory module 355 (max. 6 modules in 2 memory modules 355) |
| Timers, counters, flags | | |
| Number of timers and counters | 256 of each | 256 of each |
| Number of flags | 2048 flags + 32768 S flags | 2048 flags + 32768 S flags |

# Further Reading

# 12

## Further Reading

/1/ S5-155H
CPU 948R
Pocket Guide

Order no. 6ES5 997-3UR21

/2/ System Manual S5-135U/155U

Order no. 6ES5 998-0SH21

/3/ STEP 5 Manual

Order no. C79000-G8576-C140

/4/ GRAPH 5: Graphic programming of sequential controls under the S5-DOS SIMATIC S5 operating system

Order no. 6ES5 998-1SA01

/5/ Handling Blocks
Standard Function Blocks
CPU 948

Order no. C79000-G8563-C572

/6/ SINEC
Manual
CP 143 with COM 143

Order no. 6GK1970-1AB43-0AB0

/7/ Hans Berger:
Automating with SIMATIC S5-155U

SIEMENS AG ISBN 3-8009-1538-3

/8/ Programmable Controllers
Basic Concepts

SIEMENS AG
Order no. E80850-C293-X-A2

/9/ Catalog ST 59: Programmers
SIMATIC S5

/10/ Catalog ST 54.1: Programmable Controllers
S5-135U, S5-155U and S5-155H

/11/ Catalog ST 57: Standard Function Blocks
and drivers for
programmable controllers of the U series
SIMATIC S5

/12/ Manual SCL

Order no. C79000-G8576-C162

/13/ S5-C-Compiler for S5-155U

Order no. 6ES5 886-1MA21

# Indexes 13

## Contents of Chapter 13

List of Abbreviations . . . . . 13 - 3

List of Keywords . . . . . 13 - 5

# List of 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 |

# Index


**A**

Access to DW 255 9-15
ACCU writing conventions 6-5
Accumulators 6-21
Actual operands
  of function blocks 2-29
ADF (addressing error) 5-35
Arithmetic operations 3-59
Assignment list 2-24

**B**

Basic operations 2-4, 3-19
BASP LED 4-6
BASP signal 4-31
Binary logic operations 3-52
Binary numbers 2-8
Block
  block preheader 2-35
  body 2-13, 2-24, 2-36
  formal operands (block parameters) 2-27
  header 2-13, 2-36
  number 2-12, 2-36, 3-33
  preheader 2-14
Block address list 3-8, 8-13
Block calls 2-16, 3-8, 3-32
Block ID 2-36
Block operations 3-32
Blocks
  deleting 6-14
  generating 6-17
BR register 9-22
BSTACK (block stack) 1-20, 5-5
  evaluating 5-7
  output 5-6

**C**

CC 1 and CC 0
  *See* results codes
Checkpoints 10-8
COLD RESTART 4-25
COLD RESTART WITH MEMORY 4-25
Collision of timed interrupts 4-43
Comparison operations 3-32
Control bits 5-5, 5-10
Correcting blocks 2-15
Counter value 3-28
Counters C 1-12
CP 143 10-19
  assigning parameters 10-21
CPU type and ID 8-40
CSF (control system flowchart) 2-4
CYCLE 3-11, 4-33
  interrupt points 4-34
  program execution level 4-31
  user interface OB 1 4-34
Cycle monitoring time 6-57
Cycle time
  current cycle time of S5-155H 8-38
Cyclic communication 4-12
Cyclic processing 1-4
Cyclic program execution 1-15, 4-33
Cyclic timed interrupt 4-37
Cyclic timed interrupts 4-32, 4-41

**D**

Data block DB 0 2-41, 3-8, 8-13
Data block DB 1 2-41
Data block DB 2 2-41
Data block DX 0 2-41, 7-4
Data block DX 1 2-41
Data blocks
  general 1-12
Data blocks (DB/DX)
  copying from memory card 6-58
  copying/duplicating 6-58
  general 2-13, 2-35
  generating 3-33
  programming 2-37
  structure 2-35
  validity 2-38
Data word 1-12, 2-35, 2-39
Date 6-33
DBA (data block start address) 9-11
DBL (data block length) 9-13
Decimal numbers 2-8
Decrementing 3-68
Defaults, modifying 1-7
Delay interrupts 6-27, 6-30
Delayed interrupt 1-20, 4-32, 4-37 - 4-38, 6-45
Digital logic operations 3-52
Disable interrupts 6-12, 6-24
Dual-port RAM 9-29

**E**

ERAB
*See* results codes
Error analysis 5-9
Error DB 5-8
Error in self-test 5-41
Error information 5-5
Errors
avoiding 5-4
cause 5-32
handling 5-29

**F**

F flags 1-11
FEDBX (error in G DB/GX DX) 5-41
Fixed point numbers 2-9
Flags
multiple use 4-48
Floating point numbers 2-8
Formal operands 2-25, 3-54
Function blocks (FB/FX)
general 2-13, 2-23
programming 2-25
standard function blocks 2-23, 2-33
structure 2-24

**G**

Global memory
access 9-25
general 9-4
GRAPH 5 2-5

**H**

HARD STOP 4-4

**I**

I/Os
modules 1-10
O area 1-10
P area 1-10
Incrementing 3-68
Interface
to system program 1-7, 1-9, 2-18
Interprocessor communication flags
general 3-13
Interrupt-driven processing 1-5
Interrupt-driven program execution 4-36, 4-45
ISTACK (interrupt stack) 4-8
code bits 5-16
contents 5-15
error information 5-5
information in ISTACK 5-16
output 5-6, 5-10

**J**

Jump operations 3-61

**L**

LAD (ladder diagram) 2-4
LED display 4-5 - 4-6, 4-15 - 4-16
Library number 2-36
Load operations 3-21, 3-57
Local memory
access 9-24
general 9-4
Logic operations 3-52

**M**

Mantissa
*See* floating point number
Memory access 9-6
via the BR register 9-22
via address in ACCU 1 9-8
Memory card 1-19, 3-10
Memory organization 9-4
Mode of operation of CPU 1-4

**N**

NAU (power failure) 4-23, 4-27
Nesting depth 3-9
Nesting program execution levels 4-9
No operation 3-34

**O**

O area
*See* I/Os
Operand areas 1-10
Operating statuses 4-4
OR
*See* results codes

Organization block (OB)
general 2-12, 2-16
Organization blocks
for communication in SOFT STOP 2-20
Organization blocks (OB)
as user interfaces 2-18
for processing errors 2-20
special function organization blocks 6-4
OS (overflow latching)
*See* results codes
OV (overflow)
*See* results codes
OVERALL RESET 4-17

**P**

P area
*See* I/Os
Page area/DPR
occupied register 9-30
Pages
accessing 9-29
Parameters for DX 0 1-7
PARE (parity error) 5-37
PG functions 10-4
PG functions via the S5 bus 10-19
error bits 10-31
interface numbers 10-20
technical requirements 10-19
PG screen form
for DX 0 parameter assignment 7-5
PG software 1-18
PLC identification field 8-40
Priority 1-5, 4-8, 4-10, 4-37, 4-46, 7-7
Process image
outputs (PIQ) 1-4
inputs (PII) 1-4
general 1-10
inputs (PII) 1-10
outputs (PIQ) 1-10
updating 4-31
PROCESS INTERRUPT 4-31
Process interrupts
disable 3-74
enable 3-74, 4-47
Process interrupts via input byte IB 0
user interfaces 4-45
Program
storing 3-10
Program blocks (PB) 2-12, 2-16
Program execution levels
general 4-9, 6-27
Programming
general 1-14
Programming H systems 1-17
Programming language
C with S5-C compiler 1-18
GRAPH 5 1-18
SCL 1-18
STEP 5 1-18
Programming language SCL 1-18
Programming tools 1-18

**Q**

QVZ (timeout error) 5-34

**R**

Reaction time 4-48
Reactions with error OBs not loaded 5-30
Real time interrupt 1-20, 4-32, 4-37, 4-39, 6-38
Results codes
ERAB 3-16, 3-20
CC 1 and CC 0 3-18, 3-63
OR 3-17
OS 3-17
OV 3-17
RLO 2-7, 3-17, 3-20
STA 3-17, 3-20
RI/RJ area 8-14
RLO
*See* results codes
RS/RT area 8-15
assignment of RS area 8-16
RUN
general 4-4, 4-31
RUN LED 4-5

**S**

S flags 1-11
Self-test 5-43
Sequence blocks (SB) 2-12
Serial link PG - PLC 10-18
Set/reset operation 3-20

Shift operations 3-63
SOFT STOP 4-4
Software protection 8-32
Special functions
  calling 6-5
  errors in processing 6-6
  general 6-4
  interfaces to special functions 6-5
  new 1-20
STA (status)
  *See* results codes
Standard function blocks
  *See also* function blocks
START-UP 3-11
  general 3-11, 4-4, 4-19
  interruptions 4-27
  triggering 4-21, 4-23
STEP 5 operations 3-15
STL (statement list) 2-4
STOP
  mode 4-12
STOP LED 4-5
Stop operations 3-34
Structure of the memory area 8-4 - 8-5
Structured programming 2-5
Sub-level 4-10
SUF (substitution error) 5-37
Supplementary operations 2-4
System data 8-15
System data words
  bit assignment 8-18
System operations 2-4, 3-61, 9-4
System program 1-6
System program defaults 1-7
System RAM 8-6
System time 6-8, 6-33

**T**

Time-controlled processing 1-5
Time-controlled program execution 4-37
Timed interrupts 4-32
Timed job 4-39, 6-38
Timer and counter operations 3-26, 3-55
Timers T 1-12
TRAF (load and transfer error) 5-38
Transfer operations 3-21, 3-57
Transferring memory fields 9-19

**U**

User interfaces
  for cyclic program execution 4-34
  for delayed interrupt 4-38
  for process interrupts 4-45
  for real time interrupt 4-39
  for start-up 4-26
  for timed interrupts 4-42
User memory 1-13, 3-10
User program 1-6, 1-8
  processing 3-4, 3-11
  *See* program
  storing 1-9
  tasks 1-8

**W**

WEFES/WEFEH (collision of timed interrupts) 5-39

**Z**

ZYK (cycle time error) 5-36

Siemens AG
A&D AS E81

Östliche Rheinbrückenstr. 50
D-76181 Karlsruhe
Federal Republic of Germany

From:

Your Name: _ _ _

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:

_ _ _ _