



Universidad de Valladolid

UNIVERSITY OF VALLADOLID
INDUSTRIAL ENGINEERING SCHOOL

FINAL PROJECT DEGREE
INGENIERO TÉCNICO INDUSTRIAL, ELECETRICIDAD

AUTOMATION PROCESS OF AN ELEVATOR SYSTEM

SAMUEL NOVAL SÁNCHEZ

2012



AUTOMATION PROCESS OF AN ELEVATOR SYSTEM
FINAL PROJECT DEGREE





*A mis padres Alfredo y M^a Dolores,
mi hermano Alfredo
y mis tías, Flor y Nani
por su ayuda y apoyo incondicional,
sin ellos nada de esto habría sido posible.*







RESUMEN

Un ascensor es un sistema de transporte vertical diseñado para mover personas o bienes entre diferentes niveles. Puede usarse para moverse arriba o abajo en un edificio o en una construcción subterránea. Está compuesto por partes mecánicas, eléctricas y electrónicas que trabajan conjuntamente para obtener un sistema seguro de movilidad.

La cabina debe acudir al piso solicitado por el usuario, ya sea interna o externamente.

El principal objetivo del presente proyecto es diseñar tanto la instalación eléctrica como el control automático para un ascensor instalado en un edificio de viviendas de diez pisos, teniendo en cuenta todos los aspectos relacionados con la seguridad.

El control automático del ascensor es llevado a cabo mediante un PLC (Controlador Lógico Programable), el cual es un dispositivo electrónico diseñado para programar y controlar procesos secuenciales en tiempo real.

Con el objetivo de demostrar la funcionalidad del algoritmo diseñado, se ha realizado una simulación con una maqueta de un ascensor de cuatro niveles, utilizando un controlador de "Klockner Moeller" llamado PS4-201-MM1, el cual se programa mediante el software "Sucosoft-S40".

-PALABRAS CLAVE.-

<i>Control</i>	<i>Dahlander</i>	<i>Ascensor</i>	<i>Grafcet</i>
<i>PLC</i>	<i>PS4-201-MM1</i>	<i>Seguridad</i>	<i>Sucosoft</i>



-INSTALACIÓN ELÉCTRICA.-

La instalación eléctrica comprende la instalación y dimensionamiento de la caja de protecciones del ascensor, instalación del panel de control, instalación de los controles manuales del cuarto de máquinas y de cabina, sensores, iluminación, tomas de corriente, conexionado del motor, etc.

La caja de protecciones está compuesta por un interruptor general de corte magneto-térmico con mando manual con enclavamiento, para evitar la conexión accidental durante los trabajos de mantenimiento o reparación del ascensor. Para la protección contra contactos indirectos la instalación cuenta con interruptores diferenciales de 30mA de corriente mínima de activación, mientras que para la protección contra sobrecargas y cortocircuitos los dispositivos utilizados son interruptores magneto-térmicos e interruptores fusibles. Para la protección del motor ante sobrecargas se instala un relé de sobrecarga de disparo retardado, para evitar el disparo en el arranque. En la caja de protecciones también se instalan los contactores (relés) que actúan sobre las diferentes bobinas del motor.

El panel de control alberga en su interior la fuente de alimentación, que transforma la corriente de 230V C.A. a 24V C.C. para hacerla apta para el funcionamiento del controlador y los sensores, y el propio controlador.

Los paneles de mando manual se instalarán en el techo de la cabina y en el cuarto de máquinas y están compuestos por un botón de parada, uno de subida y otro de bajada, para posibilitar al operario el movimiento manual de la cabina del ascensor durante los trabajos de mantenimiento o reparación del ascensor.

Para la iluminación de la cabina se ha instalado un panel de LEDs, con encendido y apagado controlado por el PLC, lo que conlleva un notable ahorro de energía en comparación con otros dispositivos de iluminación. Para la iluminación del cuarto de máquinas, del hueco y del foso han sido instalados tubos fluorescentes de 8W de potencia. La instalación y dimensionamiento de la iluminación de emergencia y antipánico también se contempla en el presente proyecto.

Las tomas de corriente instaladas en el cuarto de máquinas, el hueco y el foso son tomas Schucko de 16A.

El motor encargado del movimiento de la cabina es un motor trifásico sin reductora de 3kW de potencia con conexión Dahlander de dos velocidades con



inversión de giro y freno eléctrico. Se incluyen todos los esquemas de conexión de potencia y de mando del motor.

-SEGURIDAD.-

Todos los aspectos relacionados con la seguridad de los usuarios del ascensor han sido tenidos en cuenta. Para ello, la instalación cuenta con diversos sensores, como son los sensores de tensión del cable tractor, sensor de fallo de suministro eléctrico, sensor de sobrepeso, sensores de puertas, límites de carrera, limitadores de velocidad, amortiguadores de frenado de emergencia, etc.

La instalación cuenta con una batería conectada en paralelo con el circuito del ascensor para suministrar suficiente energía eléctrica como para propiciar que la cabina llegue a la planta baja del edificio (según la vigente normativa europea sobre ascensores) en caso de un fallo de suministro.

-SISTEMA DE POSICIONAMIENTO.-

El sistema de posicionamiento es el conjunto de sensores instalados durante todo el recorrido de la cabina encargados de indicarle al control automático la posición de la cabina en todo momento.

El sistema de posicionamiento elegido para el presente proyecto consta de imanes colocados en las propias guías metálicas del ascensor a lo largo de toda su trayectoria, y tres sensores magnéticos instalados sobre la cabina. Dos de los sensores son los encargados de detectar los cambios de velocidad del ascensor cuando la cabina llega al nivel solicitado por el usuario. Para ello han de instalarse dos imanes por nivel, uno para el cambio de velocidad en subida y otro para el cambio de velocidad en bajada. El tercero de los sensores es utilizado para detectar cuándo el ascensor ha llegado al nivel solicitado por el usuario, y debe por tanto detenerse.

En resumidas cuentas, cuando un usuario solicita el ascensor en un determinado nivel, la cabina arranca en velocidad rápida hasta unos instantes antes de llegar al nivel solicitado, donde cambia a velocidad lenta. La cabina viaja en velocidad lenta hasta que alcanza el nivel y se detiene.





Las ventajas principales de este sistema de posicionamiento son principalmente la facilidad de instalación y modificación de los imanes y sensores, así como su reducido mantenimiento.

-CONTROL DE PUERTAS.-

El control de puertas se realiza mediante control automático a través de un motor de apertura de puertas situado en el techo de la cabina. Las puertas instaladas son de tipo telescópico con apertura de derecha a izquierda. La apertura y cierre de las puertas externas se realiza al mismo tiempo que las internas, gracias a una elongación de la parte móvil del mecanismo, permitiendo la maniobra únicamente cuando la cabina se encuentra perfectamente alineada con la puerta de acceso a cada planta, lo que conlleva varias ventajas, como son la simplificación de la instalación, al tener tan sólo un control de puertas, y la mayor seguridad al permitir la apertura sólo cuando la cabina se encuentra en un determinado nivel, evitando así la posible precipitación de personas al hueco del ascensor.

El mecanismo consta de un motor de apertura y cierre y dos finales de carrera, uno para cierre y otro para apertura. Para la seguridad se han instalado también un sensor óptico para evitar el cierre de las puertas cuando una persona u objeto se encuentran en la trayectoria de cierre de las puertas, y un botón de apertura de puertas incluido en la botonera interna de la propia cabina del ascensor, para abrir las puertas en caso de emergencia.

-SIMULACIÓN.-

Con el objetivo de demostrar el funcionamiento del algoritmo creado, se ha realizado una simulación para un ascensor de cuatro niveles utilizando un PLC del fabricante *Klockner Moeller*, en concreto el modelo PS4-201-MM1. Este controlador consta de ocho entradas digitales y ocho salidas digitales, funciona con una tensión de 24V C.C. y permite la conexión de módulos de extensión.

El software utilizado para programar dicho controlador es *Sucosoft-S4*, el cual sólo permite la programación en modo *Lista de Instrucciones* (IL). La función más característica de este software es la conocida como *Función de Control Secuencial* (SK), la cual está basada en los esquemas GRAFCET, haciendo la programación basándose en





dichos esquemas mucho más sencilla. El programa creado para el control de la maqueta de simulación se incluye en los anexos del presente proyecto.

Una de las principales limitaciones para esta simulación es el limitado número de entradas y salidas del controlador. Para el sistema de posicionamiento se utilizan sensores ópticos, uno por cada nivel, conectados en las entradas IO.0 a IO.3. Existe un botón por cada piso conectados en las entradas IO.4 a IO.7, quedando así ocupadas todas las entradas disponibles del controlador. Para las salidas tan sólo se conectan Q0.1 y Q0.2, para actuar sobre las bobinas del motor, una para subida y otra para bajada.

Una vez demostrado el algoritmo para la simulación, se puede admitir que el algoritmo es válido para un ascensor con n pisos o niveles. En el caso concreto que en este proyecto se describe, se ha decidido la creación de un algoritmo para un ascensor en un edificio con diez niveles.

-CONTROL AUTOMÁTICO PARA DIEZ NIVELES.-

Debido a las limitaciones tanto de software y de hardware como a la poca disposición de medios para la creación de una maqueta para diez niveles, se decidió la realización únicamente de los esquemas GRAFCET para este caso, siguiendo el mismo algoritmo utilizado para la simulación con cuatro niveles, incluyendo todos los elementos adicionales necesarios para el control y seguridad del ascensor.

Así, para el presente caso, se necesitan un número mayor de entradas y salidas, por lo que es necesaria la instalación de módulos de expansión. Los módulos de expansión aptos para este controlador (expansión concentrada o local) son los módulos LE4-501-BS1, los cuales constan de ocho entradas digitales y seis salidas digitales más dos conexiones digitales que pueden ser utilizadas como entradas o como salidas según sea necesario. Funciona con corriente continua 24V, y el número máximo de módulos que pueden conectarse al controlador PS4 es 7. En total se necesitan 63 entradas, entre sensores, elementos de seguridad, botoneras externas e internas, etc. por lo que se necesitan conectar un total de 7 módulos. El número total de salidas es 13, cuatro para el motor que cuenta con velocidad lenta y rápida, en subida y en bajada, una para la activación del freno, dos para el control de puertas, y el resto para control de luces y alarmas de seguridad.





El graficet principal cuenta con varias macroetapas para hacer más fácil su comprensión y las transiciones correspondientes a seguridades se han representado con línea roja para distinguirlas del control principal.

La idea del algoritmo creado es de fácil comprensión. En la etapa inicial se inicializan los marcadores e indicadores. Después, dependiendo del sensor de nivel activado, se guarda un valor de tipo byte en una variable llamada "ALEVEL" (nivel actual), desde el valor 0 para el piso 0 hasta el valor 9 para el piso 9. Tras este paso, el sistema permanece en espera hasta que uno de los botones (interno o externo, dando prioridad a los internos) es presionado. Dependiendo del botón pulsado, se almacena un valor de tipo byte en una nueva variable llamada "LEVEL" (nivel deseado), desde el valor 0 para el piso 0 hasta el valor 9 para el piso 9. A continuación se comparan ambas variables. Si la variable LEVEL es menor que ALEVEL, se activa el motor en sentido bajada y velocidad rápida, hasta unos instantes antes de alcanzar el nivel deseado, donde cambia a velocidad lenta. Si la variable LEVEL es mayor que ALEVEL, se activa el motor en sentido subida y velocidad rápida, hasta unos instantes antes de alcanzar el nivel deseado, donde cambia a velocidad lenta. El ascensor se detiene al alcanzar el nivel deseado y el sistema vuelve a la etapa inicial. Si el valor de ambas variables coincide, se activa el control de puertas y el sistema vuelve a la etapa inicial. Las transiciones de seguridad se explican más detalladamente en la presente memoria.

-CONCLUSIÓN.-

Todos los cálculos necesarios para el dimensionamiento de los contrapesos, potencia del motor, sección de cables y características de las protecciones se incluyen en el presente proyecto.

También se ha incluido un régimen de mantenimiento tanto preventivo como correctivo.

Los principales objetivos cumplidos son la realización de la instalación eléctrica, creación de un algoritmo para el control de un ascensor de diez niveles, verificación del algoritmo mediante una simulación para cuatro niveles y por tanto, la posibilidad de admitir la validez del algoritmo diseñado para el caso de n niveles.

Como principales mejoras cabe destacar la posibilidad de diseño de un algoritmo con memoria, para poder atender peticiones consecutivamente, mientras el ascensor se encuentra ocupado, sin necesidad de espera por parte de los usuarios a la finalización del movimiento del ascensor. Otra posible mejora es la instalación de un





variador de frecuencia para regular la velocidad de ascenso y descenso de la cabina, haciendo así el viaje más confortable para los usuarios, pero con la consiguiente complejidad que lleva su programación. También cabe la posibilidad de la instalación de un bus de datos para reducir el cableado, o un microcontrolador que se encargue de todo lo relacionado con la seguridad, ocupando así sólo una entrada del controlador para controlar la seguridad.





ABSTRACT

An elevator is a vertical transport system designed to move persons or goods between different levels. It can be used to go up and down in a building or an underground construction. It consists of mechanical, electrical and electronic parts that operate together to get a safe way of mobility.

The car must go to any floor every time it is requested by a user, either internally or externally.

The main target of the present project is to design all the electrical installation as well as the automatic control for an elevator installed in a building with ten levels, taking into account all aspects related to security.

The automatic control of the elevator is conducted by a PLC (Programmable Logic Controller), which is an electronic device designed to program and control real time sequential processes.

In order to demonstrate the functionality of the algorithm, a simulation has been made by using a Klockner Moeller's PLC called PS4-201-MM1, which is programmed with the software "Sucosoft-S40", for an elevator with four levels.

KEYWORDS

Control

Dahlander

Elevator

Grafcet

PLC

PS4-200-MM1

Security

Sucosoft





INDEX

1.	PROJECT DESCRIPTION	17
1.1.	BRIEF HISTORY OF LIFTS	17
1.2.	OBJECTIVES OF THE PROJECT	18
1.3.	MARKET ORIENTATION	19
1.3.1.	One speed elevators.....	20
1.3.2.	Two speed elevators	20
1.3.3.	Variable speed elevators.....	20
1.4.	DESCRIPTION OF THE INSTALLATION	22
1.4.1.	Main structure.....	22
1.4.2.	Traction group.....	25
1.4.3.	Protection box.....	26
1.4.4.	Control Panel.....	26
1.4.5.	External and internal doors.....	27
1.4.6.	Security.....	28
1.5.	DESCRIPTION OF THE COMPONENTS.....	30
1.5.1.	Electrical protection devices	30
1.5.2.	Conductors	34
1.5.3.	Lights	34
1.5.4.	Plugs	37
1.5.5.	Protective earth.....	37
1.5.6.	Buttons	37
1.5.7.	Contactors	39
1.5.8.	Power supply	40
1.5.9.	PLC.....	41
1.5.10.	Door control	42
1.5.11.	Positioning system	42
1.5.12.	Emergency battery	47
1.5.13.	Security.....	48
1.6.	MOTOR CONNECTION	51
1.7.	MANUAL CONTROL	53
1.8.	PLC CONNECTION	55





2.	AUTOMATIC CONTROL DESIGN.....	57
2.1.	SIMULATION.....	58
2.1.1.	Controller PS4-201-MM1 and SUCOSOFT.....	58
2.1.2.	Four levels elevator application.....	67
2.1.3.	Grafcet for simulation.....	72
2.2.	GRAF CET FOR TEN LEVELS´ ELEVATOR.....	75
2.2.1.	Inputs and outputs identification.....	75
2.2.2.	Main grafcet.....	79
2.2.3.	M1 – Level control.....	84
2.2.4.	M2 – Button control.....	84
2.2.5.	M3 - Door control.....	85
3.	CALCULATIONS.....	88
3.1.	COUNTERWEIGHT.....	88
3.2.	MOTOR POWER.....	89
3.3.	CONDUCTORS SECTION AND ELECTRICAL DEVICES DIMENSIONING.....	90
4.	MAINTENANCE AND SECURITY.....	97
4.1.	MAINTENANCE.....	97
4.1.1.	Preventive maintenance.....	97
4.1.2.	Corrective maintenance.....	98
4.2.	SECURITY.....	98
5.	PLOTS AND GRAFCETS.....	100
5.1.	PLOTS.....	100
5.2.	GRAFCETS.....	100
6.	CONCLUSION.....	102
6.1.	IMPROVEMENTS.....	102
6.2.	ACHIEVEMENTS.....	104
7.	ANNEXES.....	106
7.1.	ANNEX I: PROGRAM FOR SIMULATION WRITTEN IN INSTRUCTION LIST (IL) WITH SUCOSOFT.....	106
7.2.	ANNEX II: KLOCKNER MOELLER PS4-201-MM1 INSTALLATION.....	107
8.	BIBLIOGRAPHY.....	108







PROJECT DESCRIPTION



1. PROJECT DESCRIPTION

1.1. BRIEF HISTORY OF LIFTS

The first reference of the use of lifts is the work of the roman architect Vitruvio, who said that Arquimedes built the first elevator in 236 B.C. In medieval times, the lifts worked with the help of animals or manually to transport persons or materials between different levels. In its easiest way, it was built with a wooden box hanging from a wire.

But lifts as they are known nowadays are dated from the XIX century.

In 1823, in London, Burton and Horner built what they called “ascending room”, which could elevate 20 persons at a height of 37 meters.

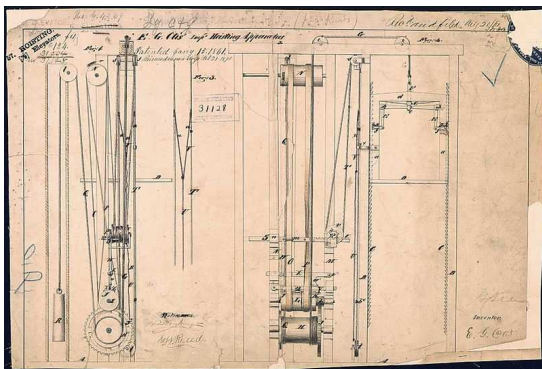


Figure 1. Elisha Otis's Elevator Patent Drawing, 01/15/1861

In 1853, Elisha Graves Otis invented the first emergency brake for lifts, which was actuated in case that the cable got broken, or the lift reached a high speed. The security device designed by Otis is very similar to one of the devices that are used nowadays.

In 1857, March 23, Otis installed his first elevator in New York through his company “OTIS”.

In 1880, the German inventor Werner von Siemens introduced the electrical motor in elevators.

In 1957, August 30, the first automatic system for doors was installed, what finished with the process of opening and closing doors manually.

In 1996, the company KONE built the first viable lift without machine room.

In 2003, ThyssenKrupp Elevators presented the “Twin” elevator: two cars in the same hole.

The World Trade Center, in New York (EE.UU), with its two towers of 110 levels, has 244 elevators with a capacity up to 4536 Kg and speed up to 488m/min.



1.2. OBJECTIVES OF THE PROJECT

The main goal of the project is to design an automatic control for an elevator system with 10 levels for a maximum of 6 persons (450Kg).

In order to demonstrate the functionality of the algorithm designed, a simulation has been made with the PLC PS4-201-MM1 for an elevator with 4 levels, principally due to the limitation of inputs and outputs of the PLC.

For the simulation, the goal is to design the GRAFCET and the program for this PLC using "SucoSoft", whereas that for the case with 10 levels the goal is to design the GRAFCET for the automatic control, taking into account everything related to security devices.

A second objective is to design the electrical installation for the elevator, which involves the connection between the power supply, the motor and the PLC, the connection of the different devices to the inputs and outputs of the PLC, and the installation of light and power plugs. The design of electrical protection for motor and persons is also included.



1.3. MARKET ORIENTATION

The current lifts can be divided in two big categories: lifts with machine room, and lifts without it. The last category has great advantages as far as space concerns, letting the lift, for instance, have access in the roof at the last floor of a building.

A second division can be made according to the type of energy used to move the car. Thus, the car can be moved by electricity or by a piston moved by a liquid. The first group are called electromechanical elevators and consist in an electrical motor that moves the car and its counterweight, whilst the second group are called hydraulic elevators and consist, as it was said at the beginning, in a piston moved by a liquid, generally oil, to make a platform go up and down.

According to the speed of the car, three types of lifts can be found: one speed elevators, two speeds elevators or variable speed elevators.

The classification can be easily observed in the following diagram:

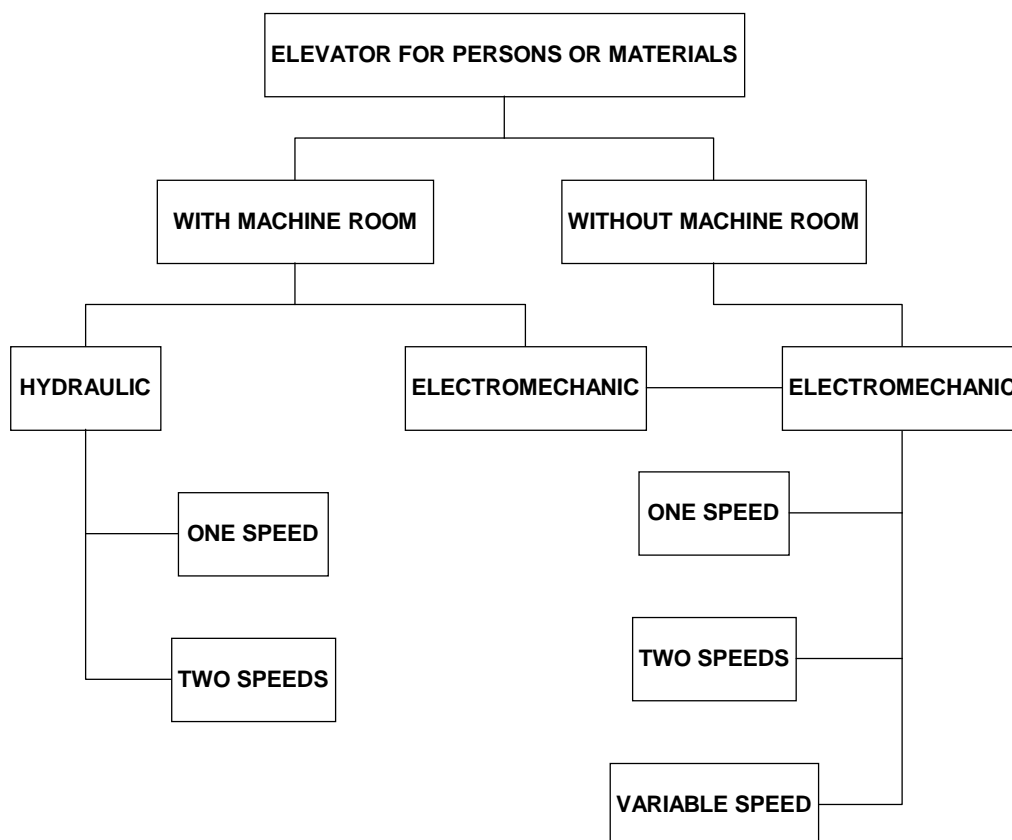


Figure 2. Elevators classification

1.3.1. One speed elevators

Elevators with one speed are the most numerous type of lifts installed nowadays, since they were the only type of elevators that were produced and installed until 1990's.

Its operation is very simple. The car goes to a given floor at a constant speed, and it stops as soon as it arrives.

This type of elevators is not produced or installed nowadays for persons, principally due to the abrupt start and stop. Another reason is the low speed of the car in this configuration.

1.3.2. Two speed elevators

Two speed elevators are the most current marketed elevators.

In this configuration, the car moves at a constant speed, but a few seconds before the target floor is reached, the car changes to a lower speed until it stops, resulting in a most comfortable trip between levels, as well as a chance to increase the speed of the car.

1.3.3. Variable speed elevators

The variable speed elevators are mainly used in high quality installations, due to a great number of users per hour.



Figure 3. Siemens frequency variators for phase motors

In this type of lifts, the speed is varied by modifying either the frequency or the voltage of the electrical power supply for the motor, with the aim of causing soft and progressive accelerations and decelerations. Thus, the users practically cannot detect abrupt movements and a higher speed of the car is allowed.



The current project is based on the configuration with machine room and two speeds. The main reason to choose this configuration is the market. While the one speed elevator is not used anymore and the variable speed elevator is more complex, expensive and for specific applications, the two speeds elevator can be used in most of the residential buildings or reforms, which are the most extended markets nowadays.

The two speeds motor also represents an energy saving with respect to the one speed elevator by reducing the friction at the stop and the energy peaks at the start.



1.4. DESCRIPTION OF THE INSTALLATION

1.4.1. Main structure

In an elevator with machine room, the main structure is composed of five different parts:

- Hole: is the space of the building provided to the movement of the car, it contains also the counterweight, the guides, limit switches, sensors, doors and traction cables.
- Machine room: this room is at the top of the installation and inside it are located the motor, the tractor group, the protections and all elements of automatic and manual controllers
- Car: is the platform where the users get into to go up and down through different levels. It must be a closed structure, provided with automatic doors to allow the users get into the car.
- Counterweight: in the case of an electromechanical elevator, as a general rule, a counterweight is installed in the opposite side of the traction cable in order to compensate the charge of the car and the persons inside, thus, the necessary power of the motor is considerably reduced
- Pit: is located at the bottom of the installation. There can be found the security dampers and other security devices

All this parts are shown in the next figure:

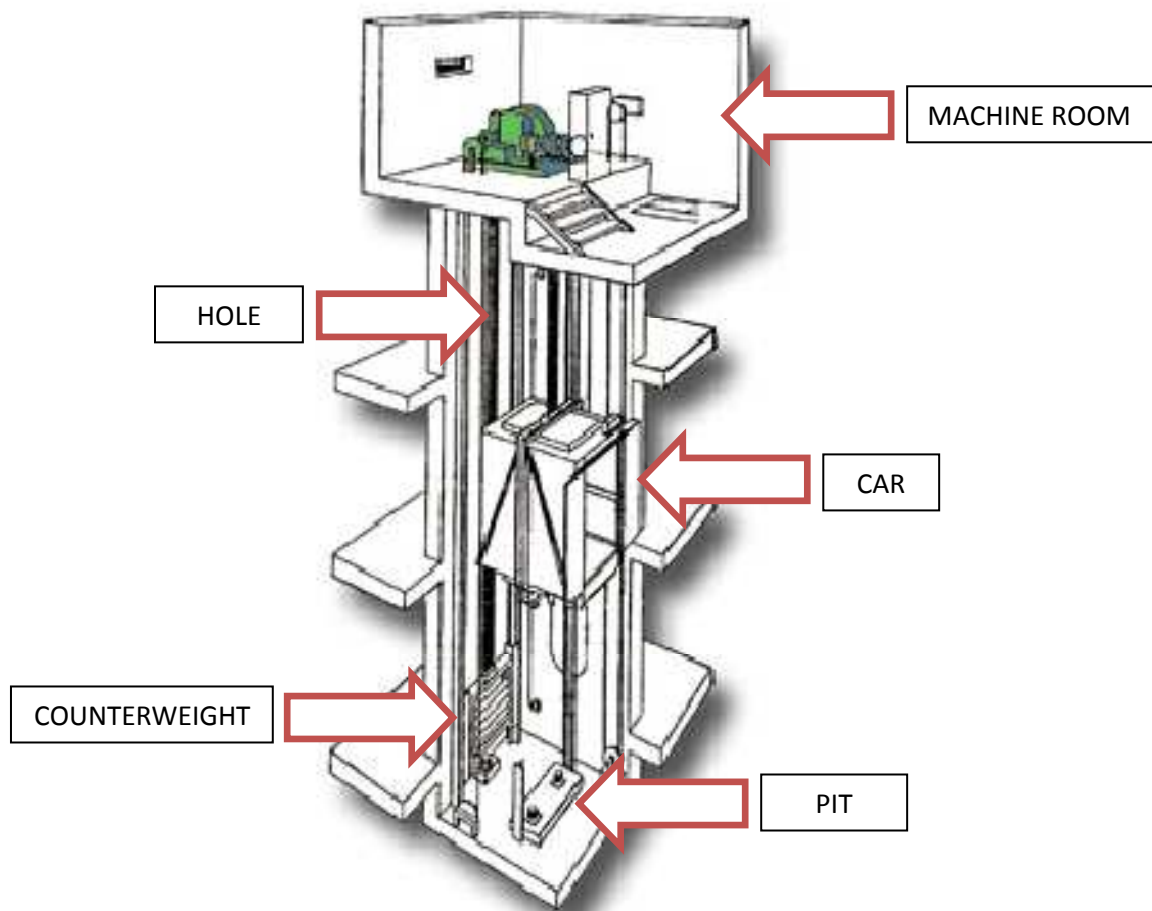


Figure 4. Main structure

In the next figure, a more detailed diagram of the components is shown. All necessary components for electrical installation and automation of the elevator are described in the next section.

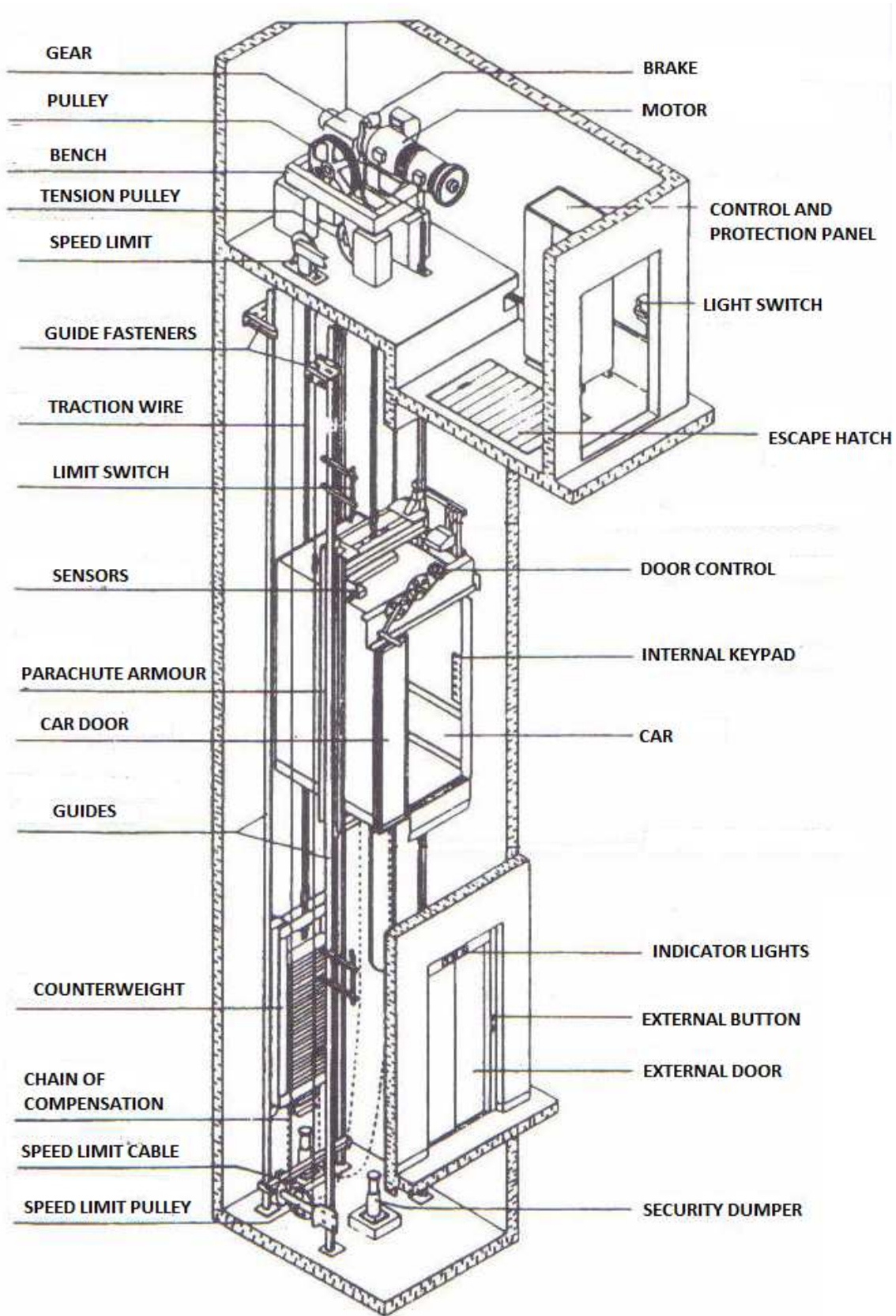


Figure 5. Components

1.4.2. Traction group



Figure 6. Tractor group

The traction group is located in the machine room and it is composed of the following parts:

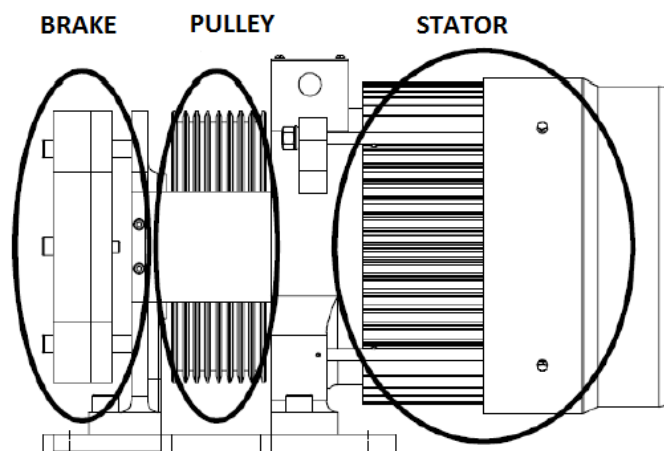


Figure 7. Motor distribution

The pulley is the part integral with the motor rotor, and from it the traction cables hang to make the car go up and down depending on the direction of rotation of the rotor.

The brake is actuated through the controller or when the power supply is off.

The motor that has been chosen for the current project is an electrical asynchronous pole-changing gearless motor with two speeds (2-4 poles). The power of the motor is **3 kW**, the main characteristics of the motor for both speeds are shown in the next table, and the justification of the power chosen can be found at the “calculations” section.

POWER CV kW	SPEED (rpm)	I _n 400V (A)	C _n (Nm)	cos φ	η (%)
6-4 4-3	2850-1420	9-8	22,4-19,6	0,82-0,85	78-79

It must be taken into account that the motor data above are only indicative and can change depending on the manufacturer.

1.4.3. Protection box

Inside the protection box are located all the electrical protection devices like the main miniature circuit breaker (MCB) to protect all the installation, the residual current circuit breakers (RCCB's) to protect against indirect contacts and the MCB's to protect against overloads and short-circuits, which will be explained later.

For this purpose, the protection box contains:

- 1 main MCB 25A/4p which must have a mechanical interlock to avoid the breaker close accidentally during repairs
- 1 RCCB 25A/4p/30mA to protect the motor against indirect contacts and 4 RCCB's 25A/2p/30mA to protect the different circuits of the installation
- 1 MCB 10A/4p, 1 MCB 16A/2p and 3 MCB's 10A/2p to protect the different five circuits of the installation

The distribution of each device for the different circuits can be found in plot number 5.

1.4.4. Control Panel

The control panel is the "brain" of the installation. Inside this panel are located the power source, the PLC, the relays and the contactors to manage the motor and the rest of the installation. The panel must be a closed receptacle and it must be accessible only by authorized and qualified persons.

The protection box and the control panel must be in different receptacles and separate each other a minimum distance of 50 cm.

1.4.5. External and internal doors

The external doors are located in the hole of the elevator, one per level, and the internal doors are located in the car.

The configuration that has been selected is telescopic doors with right-left opening and an automatic control to open and close the doors. The automatic control is located above the car door, and the external doors are opened or closed through the control of the internal door due to the elongation of the moving part of the door actuator. The external doors are equipped with a mechanism that allows the opening only if the car is aligned with the level of the external door and stopped.

This configuration has several advantages, since the external door cannot be opened when the car is not in the level, so it can be more secure to avoid persons fall down in the elevator's hole.



Figure 8. Telescopic external doors

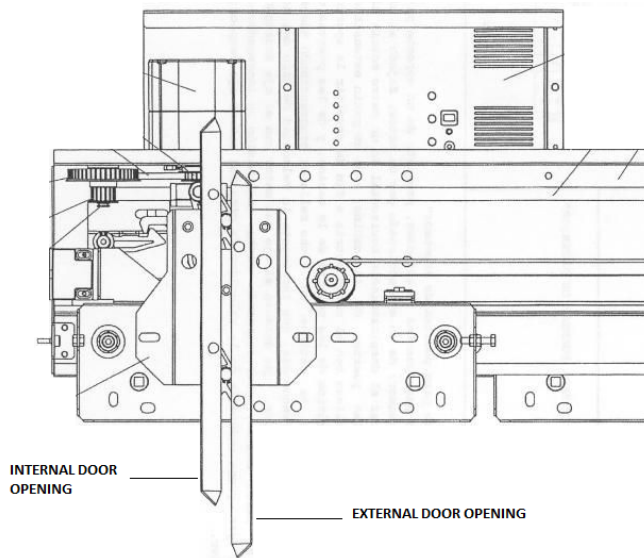


Figure 9. Door actuator

1.4.6. Security

The elevator has two kinds of securities. On one hand it has the mechanical securities, which are explained in this section, and on the other hand it has all the electrical and electronic security devices, controlled by the PLC, which will be explained later.

As mechanical securities, the elevator has two important components:

SPEED LIMITER:

The speed limiter consists of two pulleys, one mounted in the machine room and the other aligned vertically to the first in the pit. Through them passes a steel cable, whose ends are fixed, one of them to the chassis of the car, and the other to a lever system. Thus, the wire joins the car in its travel, by rotating the pulleys according to the car speed.

The upper pulley speed limiter, if it exceeds a set speed, it actuates and produces a sudden stoppage of the cable that actuates the lever system of the car.

This system frees wedges or rolls that are in a box along the guides. When this happens, the guidelines are “bitten” by wedges and the stop of the car is produced.

For the current installation, the car rated speed is 1 m/s.

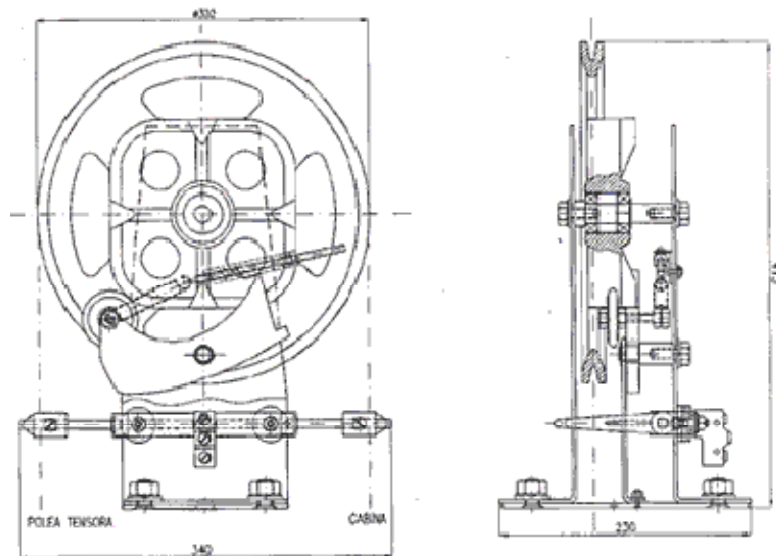


Figure 10. Speed limiter

SHOCK ABSORBER:

Elevators must be provided with shock absorbers to stop the car or the counterweight in case of speed limiter failure. These shock absorbers must be installed in the pit.

According to the rule EN-81-1, there are three types of shock absorbers:

- a) Energy storage shock absorbers: they can be used only for lifts with a maximum rated speed of 1 m/s.
- b) Energy storage shock absorbers with back movement absorption: for lifts with a rated speed above 1.6 m/s.
- c) Energy dissipation shock absorbers: for lifts with any speed.

For the current installation, the rated speed of the car is 1 m/s, so the types a) and c) can be chosen for this purpose. But the type c) uses oil inside to dissipate the energy, what means a higher maintenance as well as a danger for the environment, so the best option for this case is the energy storage shock absorber.



Figure 11. Shock absorber for elevators

1.5. DESCRIPTION OF THE COMPONENTS

In this chapter, every component of the installation is detailed in order to understand the function of each one in the elevator.

1.5.1. Electrical protection devices

POWER CONTROL SWITCH:

The power control switch is the main breaker of the installation. It is installed at the beginning of the installation, in the protection box. It is basically a MCB, with some particularities, and has several functions:

- It works as a breaker in case that an overload or short-circuit is not detected for the rest of the MCB's in the installation, maybe because a device failure or for such a big short-circuit that it cannot be broken for the other devices. It also must have protection against residual current, but with time delayed tripping, in order to have selectivity in case of a RCCB failure. The circuit break must be in all poles.
- But the main function of this device is as a general switch for the installation. It must have a manual actuator to cut the power for all the installation manually and it must also have a mechanical locking to prevent an accidental connection of the power, while the installation is being repaired or tested.



Figure 12. Main circuit breaker

PROTECTION AGAINST OVERLOAD AND SHORT-CIRCUIT:

For the protection against overload and short-circuits, the device used is a miniature circuit breaker (MCB). It breaks the circuit in case that a current a little bit higher than the rated current flows through the wire for a long time (overload), or a much more higher current than the rated for that circuit flows through the wire for a short time (short-circuit).

The category of protection is AC3 (curve C). This means that they are made to work with inductive charges and are suitable for home use.



Figure 14. MCB

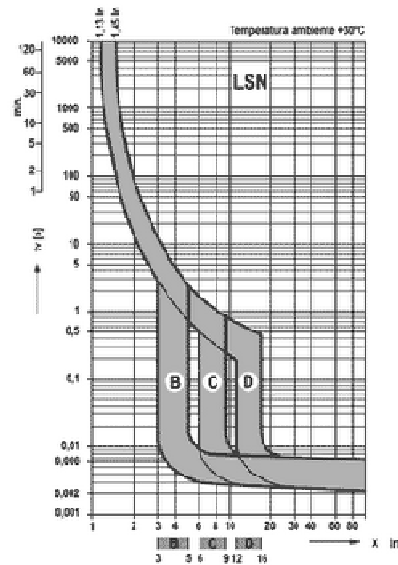


Figure 13. MCB- Curves

In order to protect the wires against overloads and short-circuit, fuses are installed in each phase of the circuits. Its essential component is a metal wire or strip that melts when too much current flows, which interrupts the circuit in which it is connected. Thus, it is installed one fuse for each of the three phases of the motor circuit, and one fuse for each of the remaining circuits.

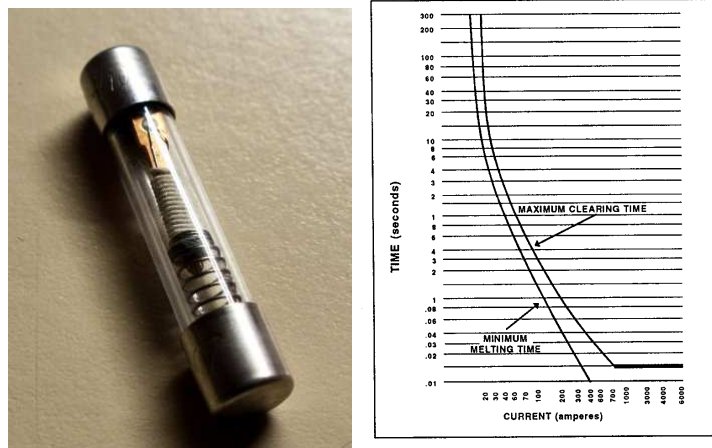


Figure 15. Fuse and fuse curve

PROTECTION AGAINST INDIRECT CONTACTS:

To protect the installation against indirect contacts, the device installed is Residential-Current Circuit Breaker (RCCB). It is an electrical wiring device that disconnects a circuit whenever it detects that the electric current is not balanced between the energized conductor and the return neutral conductor. Such an imbalance may indicate current leakage through the body of a person who is grounded and accidentally touching the energized part of the circuit. A lethal shock can result from these conditions. RCCBs are designed to disconnect quickly enough to prevent injury caused by such shocks. They are not intended to provide protection against overcurrent (overload) or short-circuit conditions.

They must be provided with a tester button to check their operation.



Figure 16. RCCB

OVERLOAD RELAY:

Overload relays are the most commonly used devices used to protect motors against weak and prolonged overloads. This protection ensures:

- Optimize durability of the engine, preventing it from operating in abnormal heating conditions
- Continuous operation of machines or installations to avoid unplanned shutdowns
- Restarting after a shot more quickly and the best possible safety conditions for persons and installation.
-



Figure 17. Overload relay

All devices detailed in this section are installed inside the protection box, in DIN rail, properly grounded for protection and the characteristics of each one as well as their distribution on each circuit can be found in the PLOTS section in “PLOT NUM: 5 - PROTECTIONS ”. The dimensioning of this installation is justified in the section CALCULATIONS.

1.5.2. Conductors

All the conductors used in the installation must be made by copper and must also be able to withstand a voltage isolation of 0,6 / 1000 V.

The minimum section for the conductors of each circuit is indicated in PLOT NUM: 5 and it is justified in CALCULATIONS.

Each type of conductor must have a different color:

- **Grounding conductors:** green and yellow
- **Phase conductors:** black, grey and brown
- **Neutral conductor:** blue
- **Control conductors:** red

1.5.3. Lights

Several kinds of lights are installed on the elevator; they will be detailed depending on the kind of light. Thereby, we will find four different types of lights:

FLUORESCENT TUBE OR INCANDESCENT LAMP:

Incandescent lamps with a power of 60W/230V or fluorescent tubes with a power of 11W/230V will be installed as follows:

- One in the machine room
- One above the door in each level
- One in the pit
- One under the car

The distribution of each light can be found in PLOT NUM: 6.

The switch for these lights will be installed in the machine room. The aim of these lamps is to light all parts of the elevator in case of reparation or maintenance.

CAR LIGHT:

For the car light, a LED panel has been chosen, due principally to energy savings reasons. As it can be seen in Figure 18, energy savings is substantial compared to other types of light.

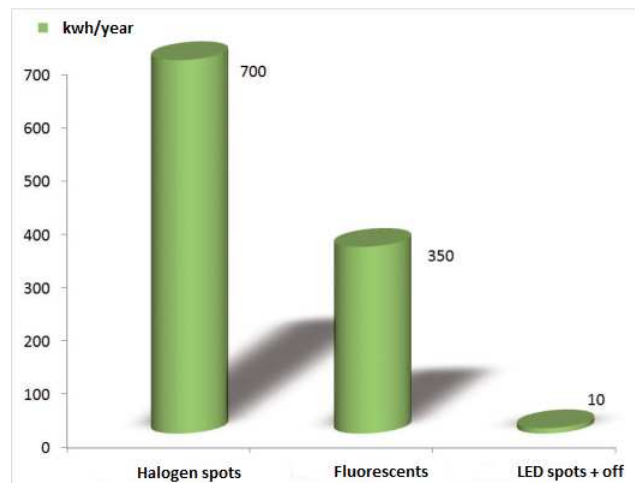


Figure 18. Light comparison

The car light is controlled by the PLC and it is turned off whenever the elevator is not used for a time, meaning a higher energy saving.



Figure 19. LED spots ceiling for elevators

The type of LED chosen is spot 230V / 7W / AC.



INDICATOR LIGHTS:

The elevator has many lights to indicate different states in order to give some information to the users. All this lights are LED 24V/DC and are controlled by the PLC. The color and connection to PLC's outputs of each light are detailed in PLOT NUM: 3. Thus, the users will find the next indicator lights:

- **External Button:** Every time that one of the buttons (external or internal) is pushed, all the external buttons on every floor are lighted to indicate to other users that the lift is being used in that moment. Once the lift is free again, the lights are turned off.
- **Up and down indicator:** a LED light informs the user if the lift is moving up or down.
- **Weight alarm:** if the maximum weight is exceeded, a light and an alarm are turned on to indicate the problem to the user.
- **Security fail:** another alarm is turned on if any security fail occurs.

EMERGENGENCY LIGHTS:

The emergency lights are turned on after a power failure. They are provided with a battery that is charged while the power supply is correct, and actuates when a power failure exists. They must have a little light to indicate if the battery works correctly.

They will be installed like follows:

- One in the machine room
- One above the door in each level
- One in the pit
- One inside the car

The characteristics of emergency lights are 230V/8W.



1.5.4. Plugs

Three plugs are installed, one in the machine room, one in the pit and one on the car's roof, as it is shown in PLOT NUM: 6.

The plugs type is called "SCHUKO". It is the colloquial name for a system of AC power plugs and sockets that is defined as "CEE 7/4". A Schuko plug features two round pins of 4.8 mm diameter (19 mm long, centers 19 mm apart) for the live and neutral contacts, plus two flat contact areas on the top and bottom side of the plug for protective earth (ground), as it is shown in Figure 20. The maximum carried by this plugs is 16A/230V.



Figure 20. Schuko plug

1.5.5. Protective earth

The protective earth is a protection system against direct contacts. It limits the contact tension in a value (50V in this installation) that cannot cause important injuries to a person who touches a metallic part that is accidentally in tension, due for example to an isolation failure.

For this purpose, all electrical elements and metallic parts of the installation are connected to the protective earth (ground) of the building where the elevator is installed.

1.5.6. Buttons

Three different types of buttons can be found in the elevator:

PUSH BUTTONS MANUAL CONTROL:

For manual control, NO (normally opened) buttons (230V A.C.) are used to control the motor while repairs or maintenance, as will be explained later.



Figure 21. Button manual control

EXTERNAL BUTTONS

One external button NO (normally opened) per level is installed to request the lift at each level. This buttons are provided with a LED light to indicate the users when the lift is being used, as it was explained in previous sections. The external buttons work with 24V/D.C., as they are PLC inputs.



Figure 22.
External button

INTERNAL KEYPAD

The internal keypad is installed inside the car and allows the users to choose the destination floor. They also are NO contacts 24V/D.C., PLC inputs.

Besides the number for level indicator, they must have Braille writing, according to the rule EN81-70, in order to let blind persons use the elevator.



Figure 23. Internal keypad



Figure 24. Braille writing on buttons

The internal keypad must also have an alarm button and a door opening button, as is explained later.

1.5.7. Contactors

A contactor is an electrically controlled switch used for switching a power circuit, similar to a relay except with higher current ratings. A contactor is controlled by a circuit which has a much lower power level than the switched circuit. The control circuit is based on NO and NC switches.

The contactors chosen for this project are provided with several switches NO and NC, but more modules with switches can be added if needed.

Another important feature of these contactors is the possibility to work with either A.C. or D.C. In fact, they have a special input to connect both controls: the manual control, which has a tension of 230V A.C., and the automatic control, which is directly connected with a PLC output at 24V D.C. as it is shown in Figure 25.

The contactors are installed inside the control panel on DIN rail.

Depending on the contactor activated, the rotor can rotate in one direction or another with two different speeds, to make the car goes up or down, fast or slow, as it is explained later.

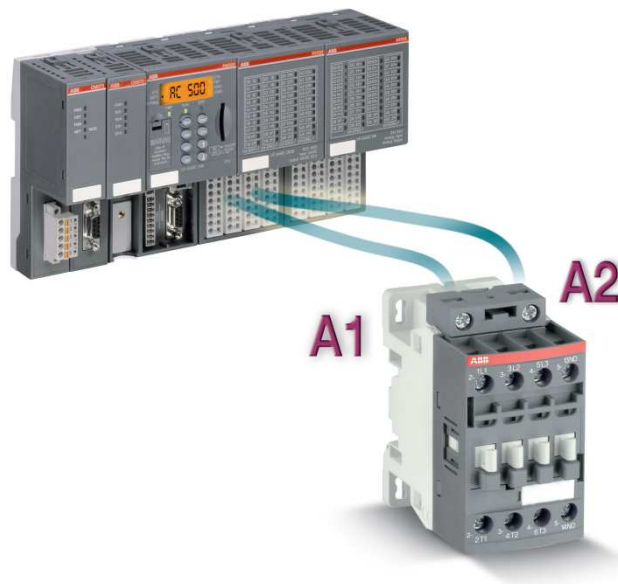


Figure 25. Contactor-PLC connection

1.5.8. Power supply

A power supply is installed before the PLC inputs in order to supply the PLC, including all the switches and sensors.

Its function is to convert the input current (230V A.C.) into 24V direct current suitable for the control circuit.

It consists basically in an electrical transformer and an inverter, and is the first device installed in the control panel, on DIN rail. A typical circuit of a power supply is shown in Figure 26.

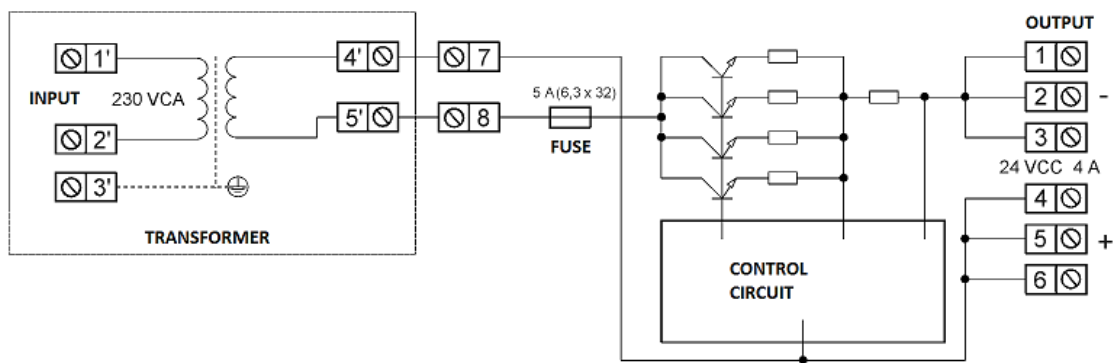


Figure 26. Power supply circuit



Figure 27. Power supply DIN rail

1.5.9. PLC

A programmable logic controller (PLC) or programmable controller is a digital computer used for automation of electromechanical processes, such as control of machinery on factory assembly lines, amusement rides, or light fixtures. PLCs are used in many industries and machines. Unlike general-purpose computers, the PLC is designed for multiple inputs and output arrangements, extended temperature ranges, immunity to electrical noise, and resistance to vibration and impact. Programs to control machine operation are typically stored in battery-backed-up or non-volatile memory. A PLC is an example of a hard real time system since output results must be produced in response to input conditions within a limited time, otherwise unintended operation will result.

For the case of ten levels, the PLC model has not been selected, inasmuch as the program for the control has not been designed, whereas that for the simulation, the PLC chosen has been the Klockner Moeller's PLC, model PS4-201-MM1, which is programmed with SUCOSOFT.

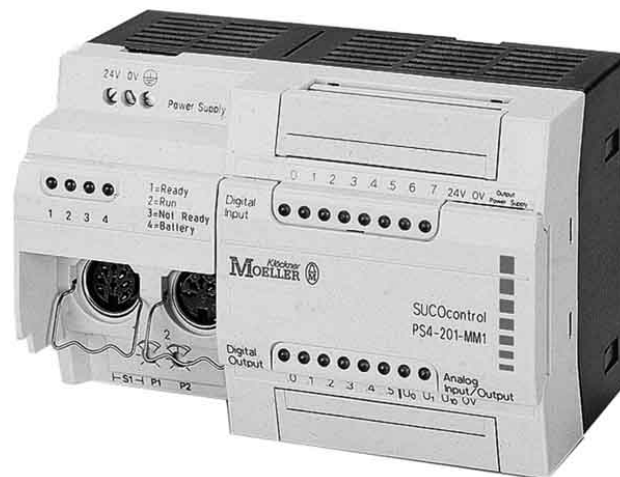


Figure 28. PS4-201-MM1

1.5.10. Door control



Figure 29. Door control

As it was said before, the door operator is the mechanism through which the elevator and floor doors are driven to both the opening and closing movements. This mechanism is located at the top of the car and consists of an electric one phase motor, independent of the principal motor, that moves several gears and belts to open and close both external and car doors.

This mechanism has two limit switches, one on each limit of the door travel, in order to detect when doors are either opened or closed.

The door operator is controlled by the PLC, as it will be explained later.

1.5.11. Positioning system

It is called positioning system the mechanism that the lift uses to detect in what position inside the hole the lift is in every moment.

Until a few years ago, most of elevators used mechanic switches to detect the car position. A mechanic “finger” in the car activates the switches installed on each level.

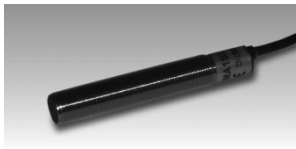
There is a new system, not too extended nowadays, for medium-high quality elevators, consisting of opposing photocells. During the car travel, photocells go through a holey metallic sheet that cuts or lets the infrared beam pass alternatively. Depending on the codification, the control calculates the car position.

A new trend in positioning systems for elevators is to use magnetic sensors to detect the position of the car. Magnets are installed on fixed parts in the hole, and the sensors are activated or deactivated whenever they pass in front of the magnets. Thereby, the automatic control calculates the position of the car.

The advantages of this system are numerous:

- Magnetic sensors are simple and reliable
- The magnets do not need electrical supply
- They offer free potential outputs, so it is not necessary a special circuit to fit the signal
- During installation or maintenance, the magnets are easily removable, as they do not need mechanical fasteners.

In the current project, this system is integrated both for controlling speed and positioning.



Magnetic sensors located in the car “read” the position depending on the magnets founded during the movement.

Figure 30. Magnetic sensor

There will be a magnet on every level that, with the car at the same level that the floor, will match exactly with sensors position, as it is shown in Figure 31. The magnet must have a length of 150 mm, and this length must be exactly the same that the separation between both level sensors. Two level sensors are installed as a security device, to detect the car level more precisely.

The distance between magnets and sensors depend on the manufacturer and must be set properly during the installation.

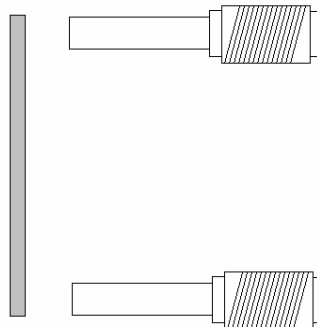


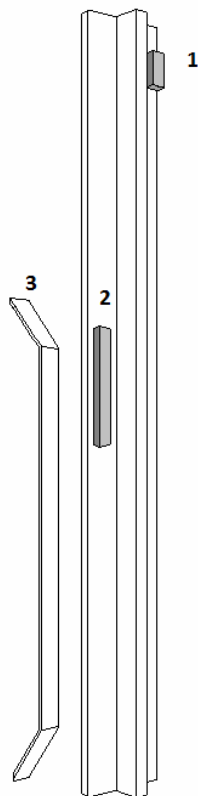
Figure 31. Magnetic sensors installation

Another magnet of about 50 mm long will be installed at approximately 700 mm over the level magnet, and another one, at the same distance, but under the level magnet. The function of these magnets, with their respective sensors, is to set the low speed in both movements, up and down.

The distance between the stop sensors (level sensors) and the up and down sensors must be set according to the motor speed, the brake, and the car length, trying to get a low speed during a balanced time between comfort and speed.

In next figures, it is shown the installation of the magnets depending on the level. The activators for limit switches are security devices that will be explained later.

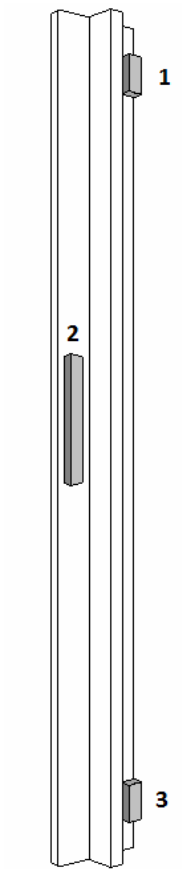
FIRST FLOOR:



1. Down pulse magnet
2. First floor stop magnet
3. Limit switch activator (bottom)

Figure 32. First floor magnets installation

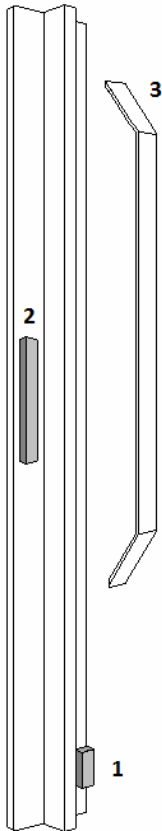
INTERMEDIATE FLOORS:



1. Down pulse magnet
2. Level stop intermediate floors
3. Up pulse magnet

Figure 33. Magnet installation intermediate floors

LAST FLOOR:



1. Up pulse magnet
2. Stop last level magnet
3. Limit switch activator (top)

Figure 34. Magnet installation last floor

1.5.12. Emergency battery

An emergency battery is an electrical device that supplies the installation in case of a power supply failure. It keeps on charging while the power supply is correct, and provides the installation with electrical current when the power supply is very low or even does not exist.

This battery will be installed only in case that the building does not have a generator group. In this case, the battery will be connected in parallel with the installation circuits, to provide energy to both the motor and the controller. It must work in such a long time that allows the security control to actuate during a power failure (down correction).

This device must be provided with a NC contact that informs the controller when the battery is supplying the installation.

The emergency battery must have a 400V input and two outputs 230/400V, with a minimal power of 10 kW / 12kVA.

A detailed document with all the characteristics of the battery can be found in section "ANNEXES", although the characteristics may change depending on the manufacturer chosen.



Figure 35. Emergency battery

1.5.13. Security

One of the most important points of the current project is everything related to security terms, due to the fact that the elevator is intended to be used by persons. To this end, all European rules in security terms related to elevators have been taken into account. For that, several devices will be installed to prevent possible injuries to the users.

Logic adopted for the implementation of the security system has been negative logic, i.e. all security devices are NC (normally closed) switches, in order to avoid a false security failure in case of malfunction of some sensors or power supply failure.

The following devices will be installed for security:

EMERGENCY STOP BUTTONS:



Figure 36. Emergency stop button

Four emergency stop buttons will be installed. Three of them will be 230V A.C. switches, installed in the manual control panel, one in machine room, one in car's roof and one in the pit, to stop the car in case of emergency during repairs or maintenance.

The fourth button will be installed in the machine room, next to the control panel, to stop the automatic control. It must be pressed before every maintenance or repair operations, or in case of emergency. This device must work at 24V D.C. insomuch as it is a PLC input.

SPEED LIMIT:



Figure 37. Speed limit

The speed limit is a NC contact 24V D.C. associate to the speed limit pulley installed on the elevator. Its function is to inform the automatic control in case the speed limit is activated.

WIRE SENSOR:



Figure 38. Wire sensor

The wire sensor is a device installed along all the traction cable of the car. Its function is to inform the controller in case of a low strain in the cable. The strain can be set in different values and the device's voltage is 24V D.C.

WEIGHT METER:

In order to avoid an overload in the motor, the weight of the charge inside the car is measured by an electronic device before it moves. If the charge is higher than a rated value, the device informs the controller about the failure. For the current elevator, the maximum weight is 450 kg (6 persons).

This device is installed under the car's floor, and it works at a voltage of 24V D.C.

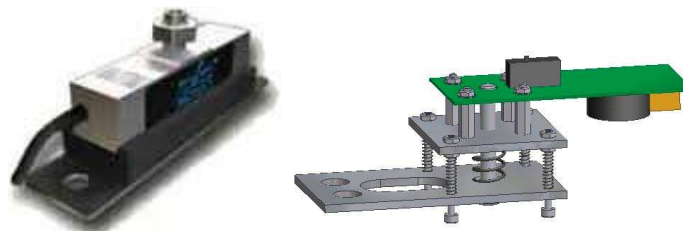


Figure 39. Weight meter

ENERGY SENSOR:

As it was said in the emergency battery section, the battery is provided with a contact to inform the controller when the installation is being supplied by the battery. This device also works at 24V D.C.

LIMIT SWITCHES:

Two limit switches will be installed as security devices: one at the top of the car's way and another one at the bottom. The function of this limit switches is to stop the motor in case the elevator reaches the limit of the way or a malfunction of the stop sensors.

There are two possibilities to stop the motor. The first one, and the most common is connecting the switches directly to the motor circuit, so when one of the switches is activated, the power supply of the motor is cut. The second one is to stop the motor through the controller. Although the first one is the best option, the second one has been chosen for the current project, principally due to the interest of implement an automatic control that involves all the security devices.

When the car is placed at floor level in extreme floors, the exact place of the switches must be calibrated in case the car reaches a distance of 5 or 6 cm of its way.



Figure 40. Limit switches

1.6. MOTOR CONNECTION

The goal of this section is to make a detailed explanation of the connection between the motor and the rest of the electrical devices.

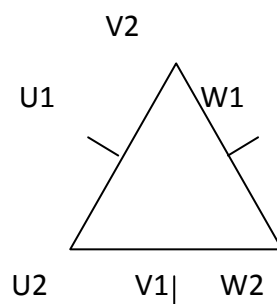
The diagram of the connection is presented in “PLOT NUM.: 1 – POWERMOTOR”.

The motor chosen for the elevator is a three phase “Dahlander” motor. It works in the same way that other three phase squirrel cage motors, except that in its windings it has intermediate connections to change the number of active poles, depending on the way it is connected. By changing the number of active poles, the motor speed will change. Logically, having two ways of connection, two different speeds are obtained, one slow and one fast. On its terminal box, there are 9 connections, corresponding to intermediate terminals, instead of 6 like in the rest of motors.

In this kind of motors is essential the use of two thermal protections, one per each speed, due to the difference of power for each one.

These motors have the particularity that their windings can be linked in three different ways: constant torque, variable torque, constant power. For the current installation the connection is for constant power, as it is needed the same power to move the car in both speeds.

The connection of motor terminals for a constant power in both speeds is as follows:



SLOW SPEED: U1, V1, W1 → to power supply; U2, V2, W2 → joined

FAST SPEED: U2, V2, W2 → to power supply; U1, V1, W1 → opened



The electrical characteristics of the necessary devices of control and protection for this configuration will be as follows:

- Contactor **K1** for slow speed down (SSD) connection. It will be for a rated current equal or higher than the motor current in triangle connection, and AC3 category.
- Contactor **K2** for slow speed up (SSU) connection. It will be for a rated current equal or higher than the motor current in triangle connection, and AC3 category. To invert the rotor direction, two phases are inverted.
- Contactor **K3** for fast speed down (FSD) connection. It will be for a rated current equal or higher than the motor current in double star connection, and AC3 category.
- Contactor **K4** for fast speed up (FSU) connection. It will be for a rated current equal or higher than the motor current in double star connection, and AC3 category. To invert the rotor direction, two phases are inverted.
- Contactor **K5** for star connection in fast speed. For fast speed, the motor start is made by a triangle-star connection to reduce the current intensity during the start. It will be for a rated current equal or higher than the motor current in double star connection, and AC3 category.
- Fuses **F1** and **F2** to protect the circuit against short-circuits. They will be aM type (special fuse type for electrical motors) and equal current than the motor current for each speed.
- Thermal relays **Q1** and **Q2** to protect the motor against overloads. Their current must be the same than the motor current for each speed.

The contactors are activated or deactivated by both manual and automatic control, as it is detailed in next sections, making the elevator go up and down in fast or slow speed.

NOTE: It is mandatory for the installer to check out the fast, slow, up and down movements of the motor before connecting it to the elevator, in order to be sure of each connection.



1.7. MANUAL CONTROL

Two manual controls are installed, one in the machine room and the other one on car's roof, to manually control the elevator during repairs or maintenance.

The manual control diagram is presented in "PLOT NUM.: 2 – MANUAL CONTROL", and it is explained in this section.

Basically, it is composed of several NC and NO contacts and buttons to activate or deactivate the contactors that manage the motor.

Between the two contactors of each investor K1-K2 and K3-K4, interlocks have been installed: one with auxiliary contacts of the contactors themselves (K1, K2, K3 and K4, 21-22) and the other one with button's auxiliary contacts (S1, S2, S3 and S4, 21-22). These contacts could be replaced by mechanical interlocks between each pair of contactors: K1-K2 and K3-K4, avoiding in this case the use of triple contacts switches for shifting S3 and S4. There are also interlocking between contactors used for low speed K3 and K4, and the remaining K3, K4 and K5, used for high speed, from the auxiliary contacts of the contactors (K1, K2, K3 and K4, 31-32) and (K5, 21-22).

In order to make it easy to understand, here it is only explained one of the controls (machine room). To make the other control (car's roof), other buttons are simply added in parallel with this circuit. Thus, we can find (S1, S2, S3 and S4, 13-14) for machine room control and (S5, S8, S9 and S10, 13-14) for car's roof control.

One fuse F5 protects the circuit against short-circuits. In case a motor overload, the circuit is opened by NC contacts associate to thermal relays Q1 and Q2.

Three emergency stop buttons are installed to stop the elevator when required. One of them is installed in the room machine, other in car's roof and the third one in the pit.

The following describes the circuit operation in each of the four movements:

a) Start and stop slow speed down (SSD):

- Start by pushing S1 (13-14 and 23-24).
- Contactor K1 closing and motor start in low speed and down direction, triangle connection.
- Interlocked by (K1, 13-14).
- Stop by pushing S0, S6 or S7 (21-22).

b) Start and stop slow speed up (SSU):

- Start by pushing S2
- Contactor K2 closing and motor starts in low speed and up direction, triangle connection.
- Interlocked by (K2, 13-14).
- Stop by pushing S0, S6 or S7 (21-22).

c) Start and stop fast speed down (FSD):

- Start by pushing S3 (13-14 and 23-24).
- Contactor K5 closing. It forms the star connection of the motor by short-circuiting U1, V1 and W1.
- Contactor K3 closing by (K5, 23-24), connecting the motor in fast speed mode, down direction and double star connection.
- Interlocked by (K3 and K5, 13-14).
- Stop by pushing S0, S6 or S7 (21-22).

d) Start and stop fast speed up (FSU):

- Start by pushing S4 (13-14 and 23-24).
- Contactor K5 closing. It forms the star connection of the motor by short-circuiting U1, V1 and W1.
- Contactor K4 closing by (K5, 23-24), connecting the motor in fast speed mode, down direction and double star connection.
- Interlocked by (K4 and K5, 13-14).
- Stop by pushing S0, S6 or S7 (21-22).

NOTE: It is mandatory for the installer to check out the fast, slow, up and down movements of the motor before connecting it to the elevator, in order to be sure of each connection.

1.8. PLC CONNECTION

The goal of this section is to explain how the inputs and outputs connection is made, as in the diagram presented “PLOT NUM.: 3 – PLC CONNECTION”, it is made to summarize all the connection, as it is explained now.

The PLC and all buttons, sensors, contactors, etc. connected to its inputs and outputs work with 24V D.C.

A generic connection has been made, representing the PLC like a box with inputs and outputs, but must be taken into account the fact that once the PLC is chosen, inputs and outputs must be selected according to the PLC selected, adding as many modules as need.

Thus, in the diagram, to make the comprehension easier, for external buttons is represented only one input, although ten inputs are needed (from P0 to P9), one per level. The same representation is made for the internal keypad (from A0 to A9) and the sensors (L0-L9, U0-U9 and D0-D9), while for the other inputs and outputs it is used only one slot, like it is represented on the diagram.

To actuate the motor in fast speed it must be taken into account that, as it was explained for the manual control, the connection must be double star, so contactor K5 must be actuated every time that K3 or K4 are actuated. For that, a parallel connection must be made between contactors K3 and K4 with K5, although K5 is not represented in the diagram, as far as it is not a controller output.

Several indicators are also installed. When the lift is been used, an external button light is turned on to indicate the other users that the lift is busy and it won't be ready until the light is off. In each floor a light indicator is installed to inform whether the lift is moving up or down. Inside the car two light and sound alarms are installed to inform the users if the maximum weight has been reached or a security failure has occurred.

Button lights and up and down lights can be installed in series connection for all levels. All input and output devices will be explained in next sections.

Once all the electrical components, connections and main installations have been explained, in next section is detailed the automatic control design, the main point of this project.



AUTOMATIC CONTROL DESIGN



2. AUTOMATIC CONTROL DESIGN

First of all, to understand the whole process, the simulation is described at the beginning. The algorithm designed is the same for the simulation and the case for ten levels, so once the simulation has been understood, the automation process for the elevator with ten levels is based on the simulation model, but adding all related to security.

The process is represented by "GRAFCETS". A GRAFCET or SFC in English (Sequential Function Chart) is a graphical programming language used for PLCs. It is one of the five languages defined by IEC 61131-3 standard. The SFC standard is defined in IEC 848, "Preparation of function charts for control systems", and was based on Petri nets.

It can be used to program processes that can be split into steps.

Main components of SFC are:

- Steps with associated actions
- Transitions with associated logic conditions
- Directed links between steps and transitions

Steps in an SFC diagram can be active or inactive. Actions are only executed for active steps. A step can be active for one of two motives:

- (1) It is an initial step as specified by the programmer
- (2) It was activated during a scan cycle and not deactivated since

Steps are activated when all steps above it are active and the connecting transition is superable (i.e. its associated condition is true). When a transition is passed, all steps above are deactivated at once and after all steps below are activated at once.

Actions associated with steps can be of several types, the most relevant ones being Continuous (N), Set (S) and Reset (R). Apart from the obvious meaning of Set and Reset, an N action ensures that its target variable is set to 1 as long as the step is active. An SFC rule states that if two steps have an N action on the same target, the variable must never be reset to 0. It is also possible to insert LD (Ladder Diagram) actions inside an SFC program (and this is the standard way, for instance, to work on integer variables).

SFC is an inherently parallel language in that multiple control flows (POUs in the standard's parlance) can be active at once.

2.1. SIMULATION

In order to demonstrate the algorithm designed for the ten levels elevator, a simulation has been made for an elevator with four levels. In this section all the simulation process is explained, including the controller, the software used to program it and all necessary devices for the elevator simulation.

2.1.1. Controller PS4-201-MM1 and SUCOSOFT

SUCO control PS4-201-MM1 is a compact controller, product of Klockner Moeller.

The controller PS4-201-MM1 (Figure 41) has eight numerical inputs and eight numerical outputs. This number can be increased if an extension module is linked to the main module, through the connectors SUCOnet K or SUCOnetK1. The added module can be linked at a maximum distance of 600m.

Programs to use the controller are inserted with the help of software installed in a computer, with standard programming language (IL), through the programmable interface unit.

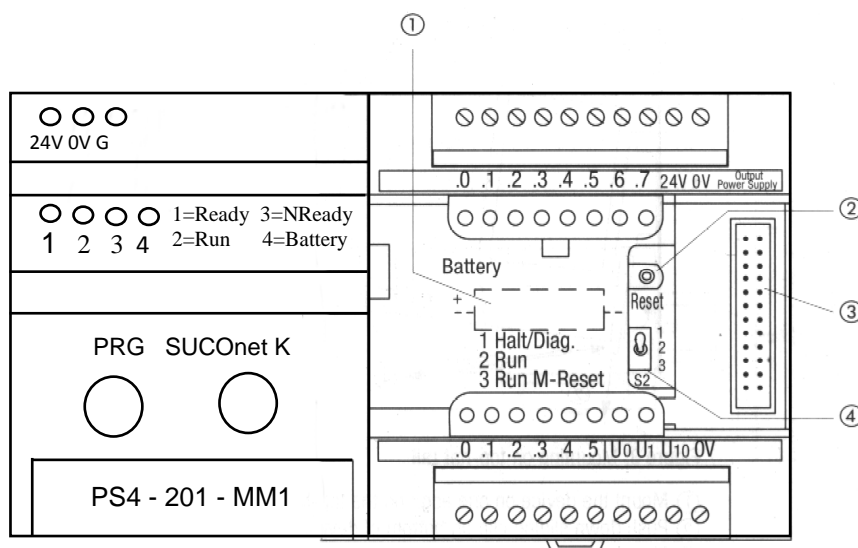


Figure 41. Controller PS4-201-MM1

1. Battery
2. Reset button
3. Extension connector
4. Work mode switch selector

Analogical inputs and outputs have a variation between 0 and 10 volts. The analogical inputs resolution is 10 bits (1.024 increases), but for analogical outputs is 12 bits (4.096 increases).

Numerical inputs and outputs are galvanic isolated of the central processing unit (CPU) and have separate electrical supplies. Each input and output has an state indication LED.

2.1.1.1. Operation mode

SYSTEM PARAMETERS SET:

The using program contains information about PS4 system configuration in the source file *.q42. These data are converted by the compiler and transferred to the controller.

To set system parameters, keys F1 and F3, from the computer's keyboard, are pressed, in order, starting from main menu:

F1 → Programming

F3 → System parameters edit

After that, name and directory of file are selected. Name and directory are required to activate next screen:

F2 → System parameters

Square brackets contain default set values. After introducing all parameters values, we return by pressing F1. Now set values can be saved. The options that can be set are presented as follows.

Program check in RUN: the compiler makes several verifications that are saved in compiled program in the defined directory. If YES=1 is selected, the PS4 operating system verifies the user program, during the operation of this algorithm. The controller



is stopped if some differences between verifications are detected. An error is introduced into corresponding verification diagnostic word. Default value is NO=0.

Start after NOT READY: defines the controller behavior after NOT READY. Default set is HALT.

The state of the controller can be set with the help of a switch which has three positions:

0 – Halt (Stop)

1 – Cold start

2 – Warm start

Maximum cycle time in ms: default set is 60ms. The maximum time value can be 255ms. This setting does not control the user program cycle time, it only defines the maximum limit to check defects. Shorter cycle time is set only if the programmer knows the real processing time. In this case, a longer processing time indicates an error. The selection of the maximum time for cycle depends on the type and length of the user program. If cycle time is reached, ETC bit is set in diagnostic word DSW and the controller changes to HALT.

Active marker range: default set is from MB0 to MB4096. This parameter sets memory length for the markers used in the user program. If the user program uses markers that were not defined in default settings, the compiler will emit the corresponding error message.

Retentive marker range: in case of low tension, the previous state of markers remains in the previous defined state. The previous state is also kept in case of system reset. This marker length forms a part of the selected active marker length, and does not overlap with the retained length of cold start.

PROGRAM EDIT FOR PS4-201-MM1 CONTROLLER:

The user program edition can be considered a specific operation for the programmable controller, containing a complete description of all control sequences.

To create a user program, the following function keys are selected, starting from main menu:

F1 → Programming

F2 → Programming IL



-Introduce source and reference files names and select directory

F2 → Edit program file

For that file *.q42, next screen is obtained, in which the program can be edited (Figure 41):

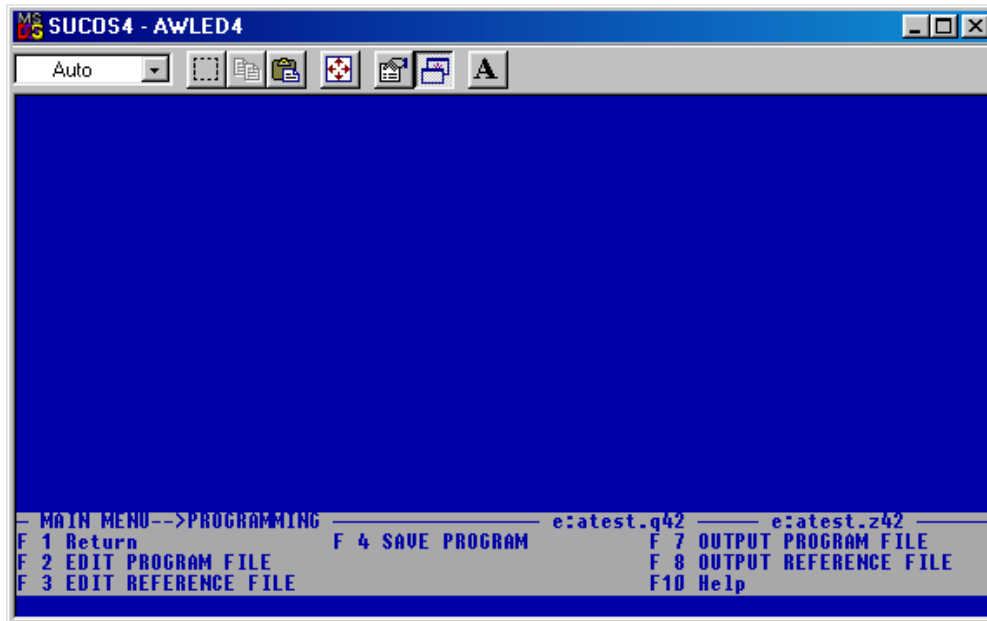


Figure 41. Edit program screen

Including the configuration file is necessary to compile the program, as the configuration file contains information about the physical structure of the connection between the controller and local extension modules or other stations. The compiler can check if addresses and specifications are correct.

The syntax for this instruction is:

```
# include "config._file_name.k42"
```

This instruction must be always the first one in the program.

PROGRAM COMPILATION:

The program writing must be compiled to obtain an executable program.

To compile the program, we must go through the following steps:



- Select F5-Compiler in Programming menu
- Specify source and reference file required through successive windows displayed.
- By pressing YES or NO, it is specified if configuration files are included or not. If the answer is YES, a standard drive must be chosen (A, B, C...). If the answer is NO, the directory for each "include" instruction is selected. NO answer is favorable just in case "include" files are saved in different drives.

After the corresponding drive has been selected by F1, the compilation starts. If the compilation is executed without any error, the executable program can be transferred to the controller. If not, the errors are listed with the name of the block and line in the program, and must be corrected. An executable program will be created only if all the errors have been corrected.

Once the program has been compiled, it only can be decompiled in original program code. For this reason, next files must be saved:

- .q42 – Source file;
- .z42 – Reference file;
- .k42 – Configuration file;

PROGRAM TRANSFER TO CONTROLLER PS4:

Once the compiler has converted the program into machine code, this can be transferred from the computer to the controller.

Before transferring by pressing the function keys F1, F2 and F6 (transfer Drive→PLC), starting from main menu, we need to follow next steps:

- Connect the computer to the controller, by using the programming cable ZB 4-303-KB1;
- Supply the controller with 24V DC, thus indicating the state Ready or Not Ready;
- The operating mode selector S2 is set in the controller to position 1-HALT.

The end of the transfer is indicated by a transfer verification message. Once the program transfer is confirmed, it can be executed, from the controller or from the computer, by Sucosoft.

To launch the program from the controller, the switch must be set in position Run M-SET or Run, and then, Reset button must be pushed.



2.1.1.2. *Logic and transfer instructions to program controller PS4-201-MM1*

The controller program is written by a sequence of instructions that describes the desired operation. It is composed by pre-processor directives and program blocks. The instruction EP ends the program. In usual mode, the pre-processor directive used is:

#include "name.k42", where name is the configuration file number.

The configuration file is made after program edition by selecting F1, F4 (Device configuration), selecting drive (C, D, E, F) and saving directory. With F2 configuration can be made and with F4 saved is made.

Each block is numbered by a five digits number, starting from 00000, and can be commented by adding simple quotation marks ("), which is not interpreted by the compiler.

To create a new block, the key F2-Open Block is pressed, from the edit menu. The instructions inside a block are numbered with a three digits number, starting from 001.

An instruction is composed by one operator and one operand. The operations described by the operator (logic, arithmetic and transfer operations) are used as the second working memory operand called accumulator (A). The result of the logic, arithmetic or transfer operations is saved on this memory A.

Now are described only those instructions that have been used to write the program for the simulation. The whole program can be found in section "ANNEXES".

TRANSFER INSTRUCTIONS:

- L operand: Load instruction → it loads in work memory (A) the corresponding operand, which can be:
 - Name of the loaded input (L IO.0. to load input IO.0. in A)
 - Name of a marker (L MO.0. to load marker MO.0. in A)
 - Name of a constant (L KB 1 loads a type byte constant in A)

- LN operand: loads the negative value of the operand in A

- = operand: transfers the value of the working memory to the corresponding operand address. The operand can be an output (Q0.0.) or a marker (M0.0.)

LOGICAL INSTRUCTIONS:

- A operand: effectuates AND logic between the accumulator and the operand, and the result is saved again in the accumulator. Operand can be input, marker or output.
- AN operand: effectuates AND logic between accumulator and the negative value of the operand and the result is saved again in the accumulator. Operand can be input, marker or output.
- O operand: effectuates OR logic between the accumulator and the operand, and the result is saved again in the accumulator. Operand can be input, marker or output.
- ON operand: effectuates OR logic between accumulator and the negative value of the operand and the result is saved again in the accumulator. Operand can be input, marker or output.
- EM: this instruction marks the end of a program module
- EP: represents the logical and physical end of the program. This instruction is located after the last step of the main program, producing a jump to the operating system.

JUMP INSTRUCTIONS:

- JP label: this instruction makes a no-conditioned jump inside the program to the address indicated in the label
- JC label: makes a conditioned jump, only if the content of the accumulator is 1. In other case, next instruction is executed.
- JCN label: makes a conditioned jump, only if the content of the accumulator is 1. In other case, next instruction is executed.

CONDITIONAL RAMIFICATIONS:

- BLT label: makes a jump to the address indicated in the label if the relation between compared terms by CP is lower (<).
- BGT label: makes a jump to the address indicated in the label if the relation between compared terms by CP is bigger (>).
- BLE label: makes a jump to the address indicated in the label if the relation between compared terms by CP is lower or equal (<=).
- BGE label: makes a jump to the address indicated in the label if the relation between compared terms by CP is bigger or equal (>=).
- BE label: makes a jump to the address indicated in the label if the relation between compared terms by CP is equal (=).

COMPARISON INSTRUCTIONS:

- CP operand: compares the specified operand with the content of the work registry and sets the conditional operators BE, BNE, BLT, BGT, BLE and BGE. These conditional operators are always used together with CP. Conditional bits are changed immediately after comparison. Operand can be constant, marker, input or output the type byte or word.

SEQUENTIAL CONTROL BLOCK:

This function uses GRAFCET representation.

Sequential control makes possible that different tasks are executed in order, as is represented in the GRAFCET. All actions provided in sequential control function are executed step by step. The sequence of one step ensures the activation of next step only if the previous step has been deactivated. This allows the user to program complex sequences in simple and clear steps. All the active steps are indicated too, thus simplifying the error diagnostic.

The pros of sequential block (SK) are:

- Clear structure in complex sequences
- Coupling steps between stages do not need to be programmed
- Simple characteristic of set and reset steps
- It is possible to modify the functional block SK without problems
- Easy diagnostic of errors through the program, that indicates active steps
- Fast processing of sequence steps.

The sequential control programming allows a graphical and structured representing of stages.

The first stage defines the initial position, and contents start and initialization conditions of the process. One important characteristic is that no more than one step is active at the same time. The next step is not activated until the next transition is executed. Only after the transition is executed, the program continues processing next step.

With type OR ramifications, only one step can be executed of several steps programmed in parallel (only one feather line). With type AND ramifications, several parallel steps are executed at the same time (double feather line). AND ramification can be synchronized, which ensures that the last transition of the AND sequence is not processed, it's only to wait for the rest of stages in parallel finish at the same time.

Input SINO specifies what the next stage to be processed is. Before the sequential step is called for the first time, this input must be initialized with the number of the next step to be processed. The initialization is accomplished by subprogram INIT, which remains active as long as RESET=1. The operand INB 0.0 (indicator for first cycle after Reset or after pushing reset button), can be used as RESET input for sequential control function. Reset input is set to 1 in the first cycle, after program starts. The program INIT assign to input SINO the first step number.

This initialization ensures that the functional block SK knows the number of the first step to be processed. The value of the step SINO is changed depending on what step is going to be executed. A logical sequence of the program can be easily written.

To activate the functional block, SET input must be 1. This activates simultaneously defined stage input.

The sequential control syntax can be founded in the printed program created for the simulation, in section "ANNEXES".

2.1.2. Four levels elevator application

In this section all the application for the simulation of the elevator is described. The simulation consists basically on an elevator with four levels. The device has four sensors to detect the position of the car, and four buttons to choose the level of the car.

BLOCK DIAGRAM FOR DEVICES CONNECTION:

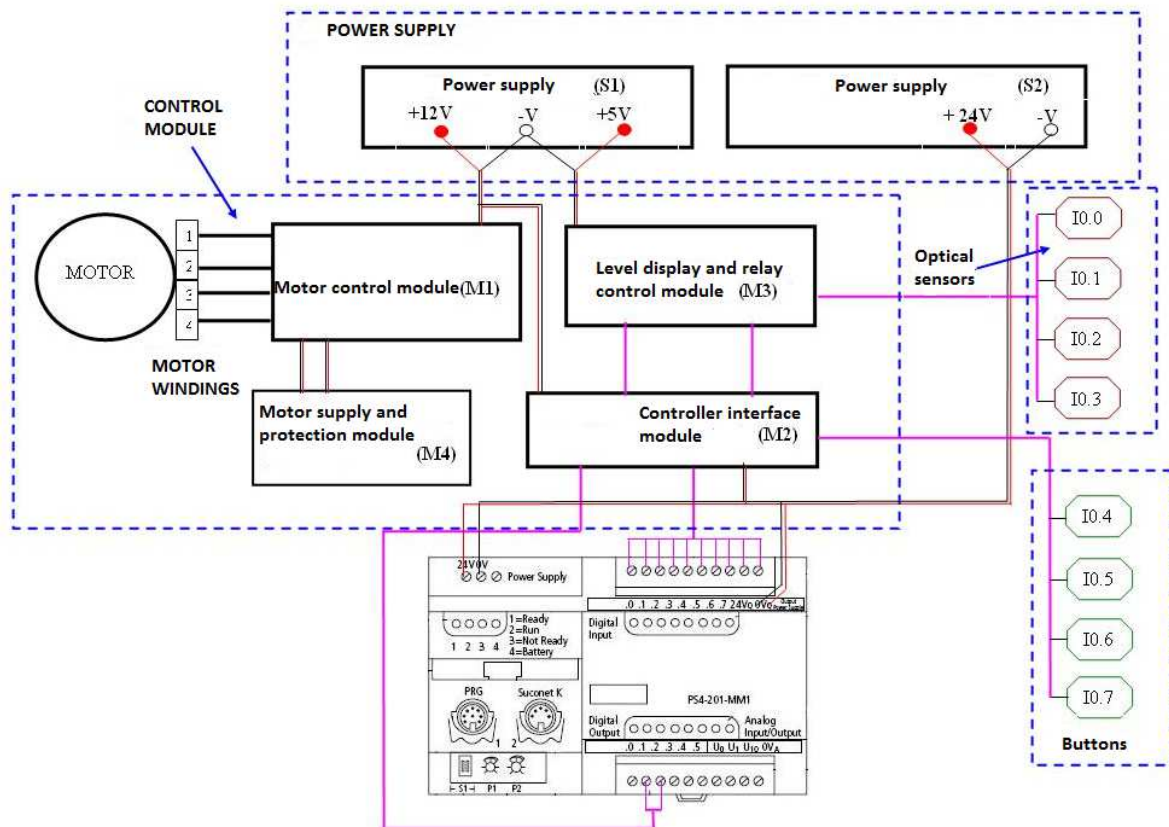


Figure 42. Simulation connection

The parts of the block diagram are:

- Driver motor (M1)
- Protection module and PS4 controller interface (M2)
- Level display and relay control module (M3)
- Motor supply and protection module (M4)
- PS4-201-MM1 controller
- Controller power source (S2)
- Power source for driver motor module, controller interface, level display and motor protection (S1)

DESCRIPTION OF THE APPLICATION:

A lift must move between the ground floor and the rest of the levels of a building.

The positioning system is based on four limit switches, one per level. When one of these switches is activated, the level is marked in a little screen and a logic signal is generated (1 logic). The sensors activate inputs I0.0, I0.1, I0.2 and I0.3 of the controller. This operation is made by the intermediate assembly of modules M3 and M2.

M3 contains two electronic assemblies that are commanded for the signal created by the four sensors. The first assembly is to show the level where the lift's car is, and has as a main device an integral circuit latch/decoder/divider BCD-7 segments MMC 4511. The second assembly has as main element 4 NPN low power transistors and an integrated circuit CDB 400, which has four logic NAND doors. Their role is to adapt the signal from the sensors to the module M2.

M2 has the function of transmission the logic signal from the CDB 400 to the controller. This signal is applied on the basis of 4 low power transistors. This transistors control 4 low battery relays (24V).

The transistors make the function of a buffer between logic outputs of the CDB 400, which can have a maxim level of 5V (1 logic), and the supply tension for relays windings (12V). For a visual review of the floors functionality, are installed in parallel with the relays windings 4 LEDs to indicate what the position of the lift is in the system, i.e. the actual level. Relays make an electrical isolation between the controller tension (24V) and the supply tension of the driver module (5V), thus protecting the controller against short-circuits or vortex currents. The modules ground is separated of the controller ground.

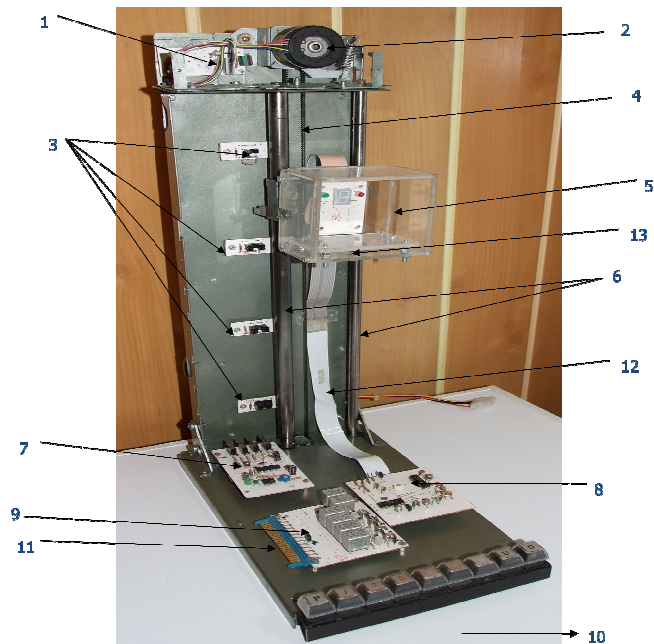
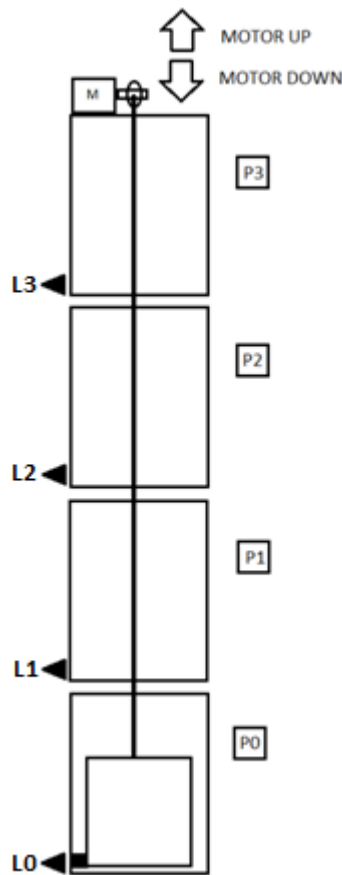


Figure 43. Simulation model

1. Supply and protection module (M1)
2. Motor – stepper motor 24V
3. Sensors
4. Driving belt
5. Lift car
6. Metal guides
7. Driver motor module (M1)
8. Level display and control relays
9. Controller interface module (M2)
10. Buttons
11. Coupling controller contacts
12. Mobile connection band
13. Pressure sensor

INPUT AND OUTPUT IDENTIFICATION:

One of the most important steps for the simulation is to identify the inputs and outputs of the controller, in order to write the program properly. Figure 44 shows the connection between sensors, buttons, relays and the controller, as well as a brief description of each one.



NAME	DESCRIPTION	I/O & MARKERS
L0	Sensor level 0	I0.0
L1	Sensor level 1	I0.1
L2	Sensor level 2	I0.2
L3	Sensor level 3	I0.3
P0	Button level 0	I0.4
P1	Button level 1	I0.5
P2	Button level 2	I0.6
P3	Button level 3	I0.7
MOTOR UP	Motor drive up	Q0.1
MOTOR DOWN	Motor drive down	Q0.2
LEVEL	Selected level	MB1
ALEVEL	Actual level	MB2

Figure 44. I/O identification

In Figure 45, the connection between inputs outputs and controller is shown.

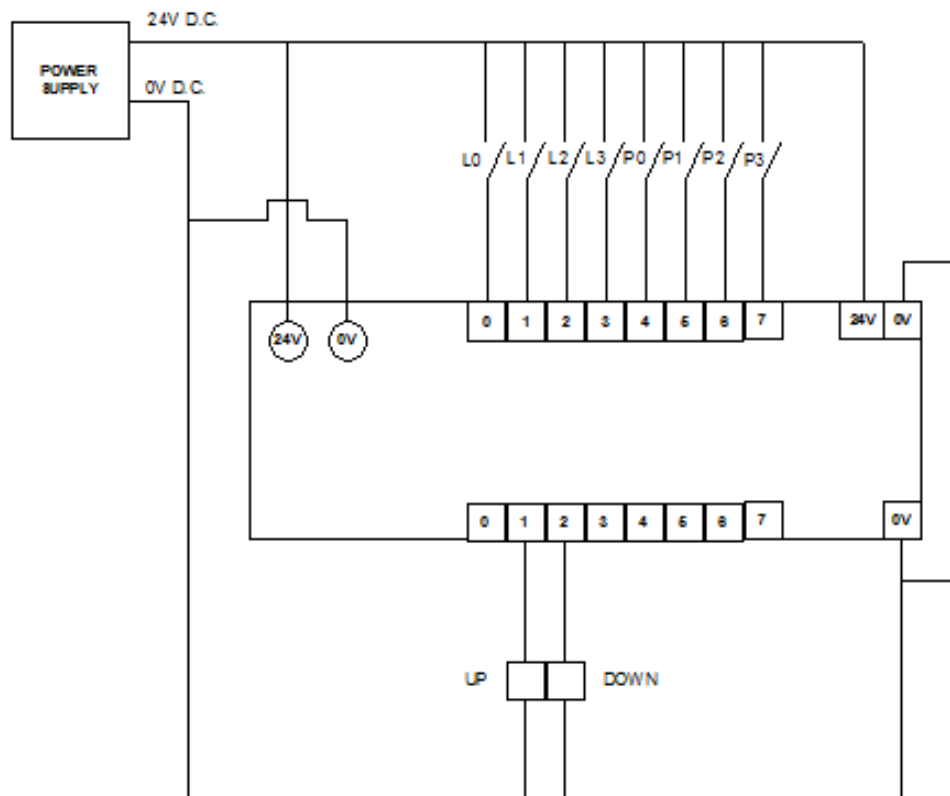


Figure 45. I/O connection

Now that all devices have been explained, in the next section the designed algorithm to solve the problem of the lift is detailed through the explanation of the GRAFCET.

2.1.3. Grafcet for simulation

The objective of this GRAFCET is to represent in an easy way the algorithm designed for the simulation. Once that is proved that the algorithm works for the simulation, it can be accepted that it works for the case with ten levels. The GRAFCET provides the programmer an easy comprehension of the problem and an easy way to write the program with the sequential block function SK from SUCOSOFT.

Two different GRAFCETS have been made for the simulation, in tow levels: level one represents the GRAFCET with the name of the variables, while level two represents the GRAFCET with the name of inputs and outputs to make a nearer diagram from the programming language.

Both GRAFCETS can be found in the section “GRAFCETS – NUMBER 5 AND 6”, and they are explained like follows.

In the first stage, all markers and outputs are set to 0. This is the initial stage so the elevator must be stopped (Figure 46).

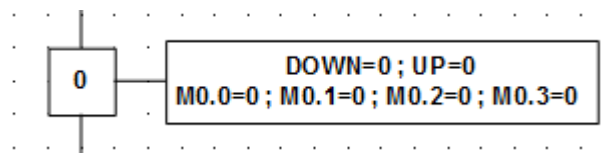


Figure 46. First stage

Next step is an OR ramification with four different stages, depending on the level where the car is, i.e. depending on the activated sensor, a different value is saved in the marker “ALEVEL-MB2”, starting from value 0 for level 0 to value 3 for level 3 (Figure 47).

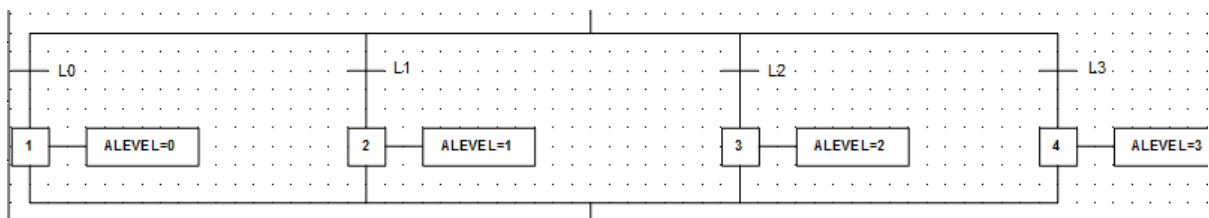


Figure 47. ALEVEL value

After that a stage is represented where the program does not make any action. It is because another stage is needed to pass from the ramification above to the next. Thus, after stage number 5, another OR ramification follows with the routine. In this ramification, with 4 stages too, a different value is saved in the marker "LEVEL-MB1", depending on the button pressed, starting from value 0 for button P0 to value 3 for button P3.

One problem that must be solved is that once the buttons have been pushed, they do not remain pushed, so we need to save somewhere in the program the fact that the button has been pushed. For this reason, some markers are used. Every time a button is pushed, one different marker is set to value 1, thus, markers starting from M0.0 for button P0 to M0.3 for button P3 are used in these stages, as it can be seen in (Figure 48).

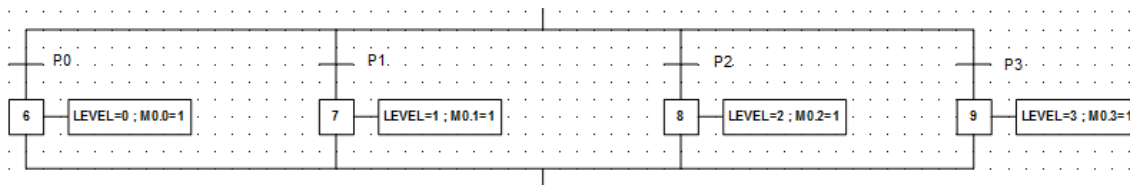


Figure 48. LEVEL value

Now that the markers LEVEL and ALEVEL have taken a value, we compare both with the CP function of SUCOSOFT. If the level required by the user is higher than the actual level of the car ($LEVEL > ALEVEL$), the motor is driven up, and obviously, if the level required by the user is lower than the actual level of the car ($LEVEL < ALEVEL$), the motor is driven down. If the level required by the user is the same level where the lift is ($LEVEL = ALEVEL$), it does not move, so the program goes to the initial stage and waits for next order (Figure 49).

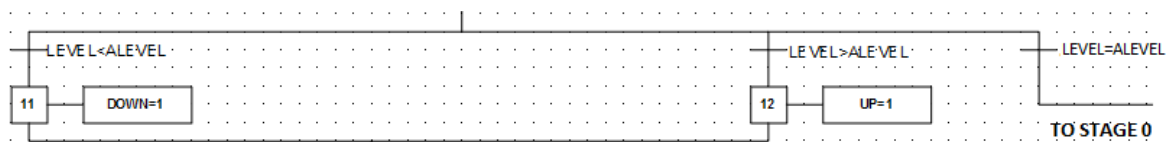


Figure 49. LEVEL & ALEVEL comparisson



As it is known, the elevator must stop when the car arrives to the level required by the user. To make this possible, the equation created is as follows:

$$L0*M0.0+L1*M0.1+L2*M0.2+L3*M0.3$$

Where “*” is AND logic operation and “+” is OR logic operation. That means that when the transition is fulfilled, i.e. when LX is activated and M0.X is 1 (being X a number between 0 and 3) the program goes to the initial stage, where all variables are set to 0 and the car is stopped, with another words, the elevator stops when the car arrives to the required floor and the process starts again.

All transitions with (=1) mean that the program changes to the next stage when the previous one has finished, without any condition.

As we can see, the algorithm has been explained in an easy way. Starting from this GRAFCET, the program is written in SUCOSOFT with the SK function block. Once the program has been created, it is compiled and transferred to the controller to test the simulation.

The program and reference file created in instruction list mode (IL) are printed in section “ANNEXES – PROGRAM AND REFERENCE FILE”.



2.2. GRAFCET FOR TEN LEVELS' ELEVATOR

Once that the simulation has been explained and tested, it can be accepted that the algorithm is valid for the case of an elevator with n levels, letting n be a natural number (1, 2, 3 ... n). For the current project, the GRAFCET designed is for a lift with ten levels (from level 0 to level 9). In this section all grafquets related to the automatic control design are explained. These grafquets can be found in section "GRAFCETS" of the project.

The programming part has been omitted for this case, due to the limit of inputs and outputs and modules of the PLC chosen, but with the design of the GRAFCET it is easy to write the program for that purpose and not only for the PLC used in this project, but for the rest of PLCs in the market, based on GRAFCET programming.

2.2.1. Inputs and outputs identification

As it was said before, the main GRAFCET (GRAFCET NUM.: 1) is based on the GRAFCET explained for the simulation, but adding all related to security, which makes it more complex.

But first of all the inputs and outputs must be identified, as it was said before, it is one of the most important steps of the automatic control in order to write the program correctly.

The connection between the inputs, outputs and the controller was explained in section "1.8.PLC CONNECTION" and the diagram can be found in section "PLOTS – PLOT NUM.: 3 – AUTOMATIC CONTROL".

In (Figures 50 and 51) it is represented a relation of inputs, outputs and a brief description of each one, as well as an elevator diagram to see the position of each component in the installation.

In total 63 inputs and 13 outputs are needed, so the installation of extension modules is necessary to make the control possible. For the controller PS4-201-MM1 the expansion modules required are LE4-501-BS1.

EXPANSION MODULE LE4-501-BS1:

The expansion module LE4 is suitable for digital inputs and outputs local expansion of Klockner Moeller's controller PS4, in case that more inputs and/or outputs are needed, as it happens in this case.



Figure 50. LE4-501-BS1

This module works with 24V D.C. It has 8 digital inputs and 6 digital outputs plus two digital connections that can be used indifferently as inputs or outputs. The maximum number of modules that can be connected to the PS4 controller is 7, just the number of modules that are necessary for the installation, as 63 inputs are needed ($7 \times 8 = 64$).

The connection between controller and modules is shown in (Figure 51).

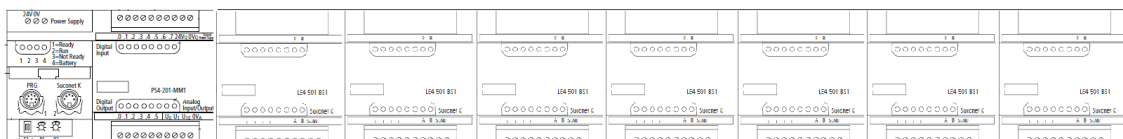


Figure 51. Modules connection

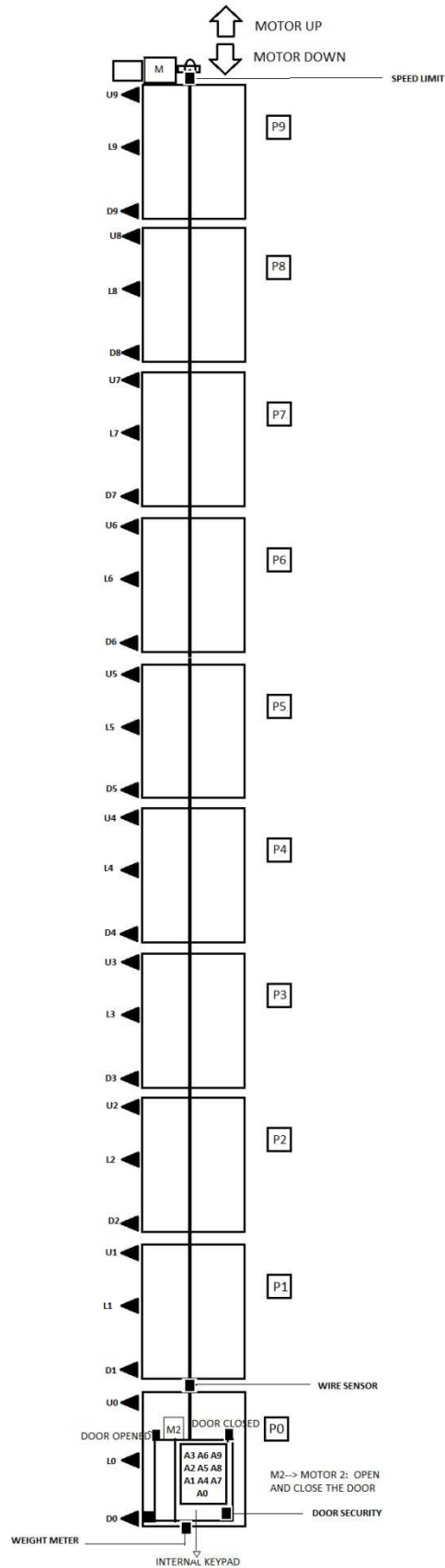


Figure 52. I/O diagram

NAME	DESCRIPTION	INPUTS
P0-P9	External buttons	I0.0-I0.7/I1.0-I1.1
A0-A9	Internal buttons	I1.2-I1.7/I2.0-I2.3
L0-L9	Level control	I2.4-I2.7/I3.0-I3.5
U0-U9	Up sensors	I3.6-I3.7/I4.0-I4.7
D0-D9	Down sensors	I5.0-I5.7/I6.0-I6.1
TOP	Top limit switch	I6.2
BOTTOM	Bottom limit switch	I6.3
STOP	Stop button	I6.4
WEIGHT	Weight meter contact	I6.5
WIRE	Wire sensor	I6.6
SPLIMIT	Speed limit sensor	I6.7
ENERGY	Energy failure sensor	I7.0
DOOR OPENED	Door opened sensor	I7.1
DOOR CLOSED	Door closed sensor	I7.2
OPEN BUT.	Open door button	I7.3
SECURITY	Security door sensor	I7.4

NAME	DESCRIPTION	OUTPUTS
K1	Activate contactor K1	Q0.1
K2	Activate contactor K2	Q0.2
K3	Activate contactor K3	Q0.3
K4	Activate contactor K4	Q0.4
BRAKE	Activate brake	Q0.5
OPEN DOOR	Activate door opening	Q1.0
CLOSE DOOR	Activate door closing	Q1.1
CAR LIGHT	Car light on	Q1.2
EXT. BUT. LIGHT	External buttons light on	Q1.3
UP LIGHT	Up light on	Q1.4
DOWN LIGHT	Down light on	Q1.5
ALARM	Alarm on	Q2.0
SEC. FAIL	Security failure indicator	Q2.1

Figure 53. I/O description

2.2.2. Main grafcet

Now that all inputs and outputs have been described, the MAIN GRAFCET is explained next. This grafcet is based on the grafcet designed for the simulation and it can be found in section “GRAFCETS – GRAFCET NUM.: 1 – MAIN GRAFCET”.

To make the explanation easier, the grafcet is explained step by step.

The first stage is where the program is initialized, so there all variables and outputs are reset (Figure 52):

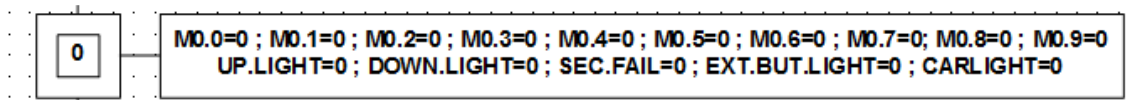


Figure 54. Stage 0

The next step has been clustered in a “macro-stage” (M1 – LEVEL CONTROL) to make the main grafcet easier to understand. In this macro-stage a value is saved on the variable ALEVEL, depending on the level of the car, as it will be explained later.

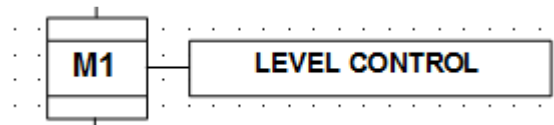


Figure 55. Level control

After that, there is an empty stage (STAGE 11 - NOP) where the program does not do anything, but this stage is necessary just to let the program wait there until a button is pushed. Then the program goes to the button control, which is included in another macro-stage (M2) that will be explained later. In this macro-stage basically a value is saved in variable LEVEL depending on the level required by the user.

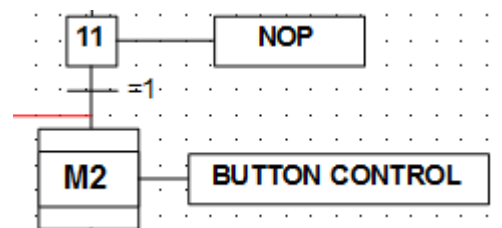


Figure 56. Button control

In the next stage, the program compares the value of the two variables LEVEL and ALEVEL.

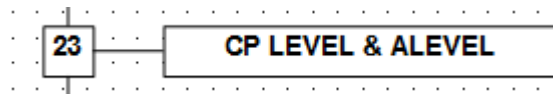


Figure 57. Compare LEVEL & ALEVEL

Next step is an OR condition. Depending on the value of the comparison, the program goes to a different way.

If $LEVEL < ALEVEL$, the elevator must go down and if $LEVEL > ALEVEL$ the elevator must go up. Up and down movements have several stages. After the comparison made before, the program waits 5 seconds for security. After this elapsed time, if there is not any security failure, the movement starts in fast speed. When the desired level is reached by up or down sensors, the controller changes to slow speed. To let the controller know what the desired level is, several markers are used. For example, if the desired level is level 1, it is saved $M0.1=1$, then the next equation is used to make the transition between slow and fast speeds (D for down and U for up sensors):

$$D0 * M0.0 + D1 * M0.1 + D2 * M0.2 + D3 * M0.3 + D4 * M0.4 + D5 * M0.5 + D6 * M0.6 + D7 * M0.7 + D8 * M0.8 + D9 * M0.9$$

If $LEVEL = ALEVEL$, the car does not move, the program goes to the door control (stage 30), i.e. the door is opened and then the program starts a new cycle it goes to the first stage and waits for next order. All this process is shown in Figure 56.

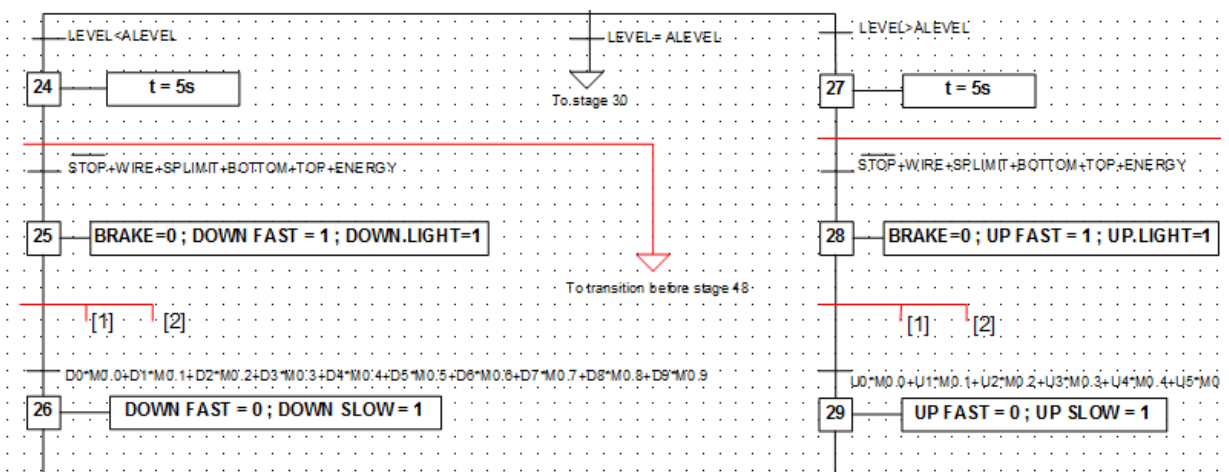


Figure 58. Up and down movement

The condition used to stop the elevator when it is in the desired level is the same than in the case before to change the speed of the motor:

$$(L0*M0.0)+(L1*M0.1)+(L2*M0.2)+(L3*M0.3)+(L4*M0.4)+(L5*M0.5)+(L6*M0.6)+(L7*M0.7)+(L8*M0.8)+(L9*M0.9)$$

Thus, when the car arrives to the required level, it is stopped. The program waits five seconds for security and after that the door control is activated, which is included in another macro-stage (M3). When the door is closed, the program waits five seconds again and after that it goes to the first stage to reset all variables and wait for a new order (Figure 57).

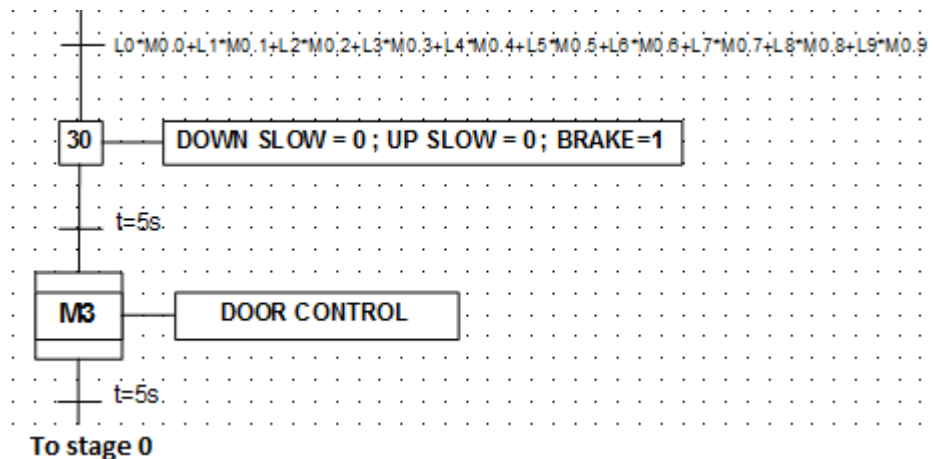


Figure 59. Car stop and door control

This is the explanation for the main grafcet when all the installation works right but, as it can be seen in MAIN GRAFCET, there are much more stages. All that stages are related to security, and they are explained next, step by step, to complete the explanation of the whole grafcet.

SECURITY:

All security transitions are represented with red line, to differentiate them clearly from main transitions and stages. There are three different securities. One of them is activated when a security fail has occurred, i.e. one or several security sensors are activated (WIRE, SPLIMIT, BOTTOM, TOP, ENERGY, STOP). In this case, the elevator is stopped and an acoustic and visual alarm is activated inside the car to inform the users. The door control can be activated inside the car by pushing OPEN BUTTON, to allow the users get out of the car in case of security failure. The program remains in this stage until the problem has been solved, i.e. the security sensors are not activated any more (Figure 58). After the security failure is solved, the program goes to the first

stage to start a new cycle. For all security sensors it is applied negative logic, i.e. all security sensors are N.C. contacts, to avoid a false security signal in case that the sensors do not work good. This security checking is done after a button is pushed, before the movement starts.

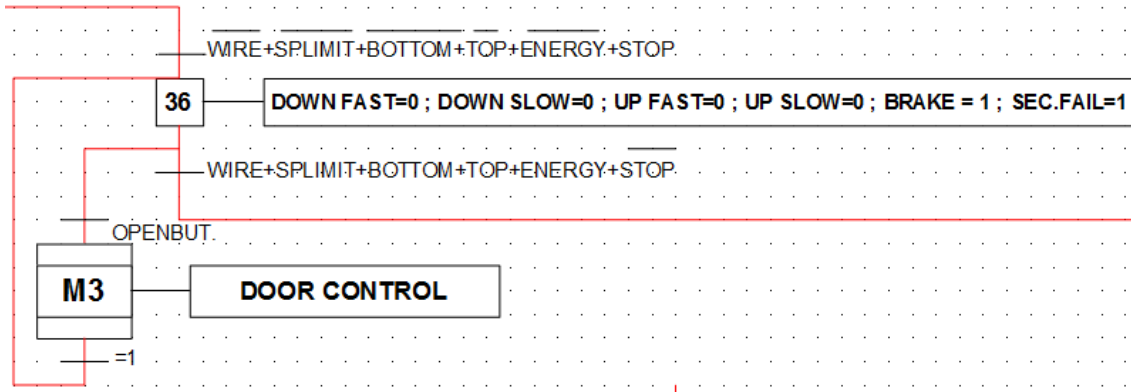


Figure 60. Security failure

Another security device is the weight meter. In case that the maximum rated weight is reached, to avoid an overload in the motor, the movement is not allowed, the door is opened to allow users to get out of the car and an acoustic and visual alarm is activated to report the event. The program remains in this stage until the weight is correct to start the movement. Then, the door is closed and the program goes back to the button control to wait for a new order (Figure 59).

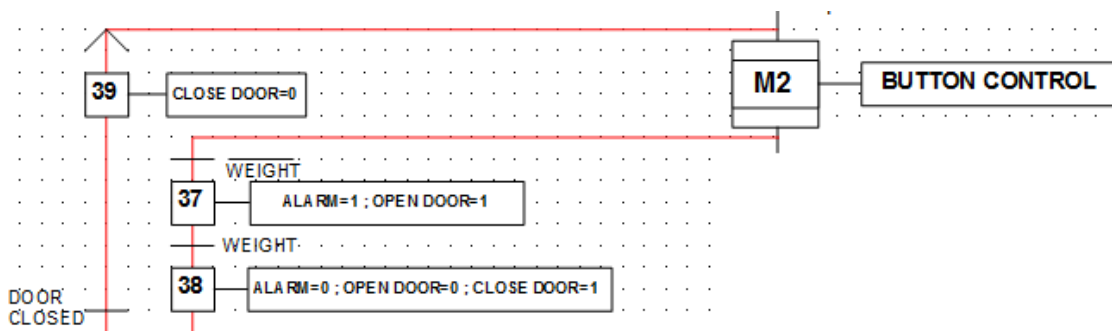


Figure 61. Weight control

During the movement, security checking is made in every transition, to stop the elevator in case of a security failure. If this occurs, the elevator stops in the next floor and the program goes to stage 36, which was explained before (Figure 60).

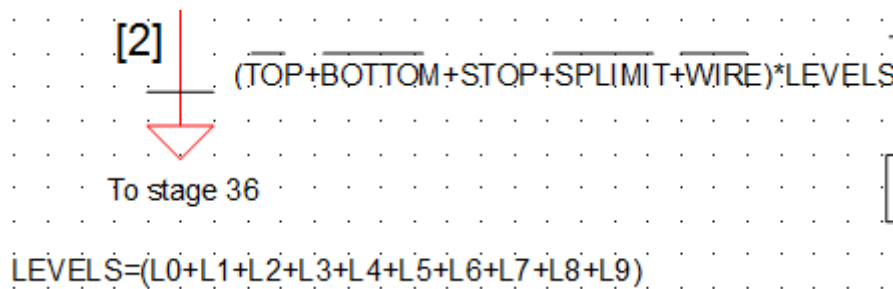


Figure 62. Security fail during movement

If during the movement an energy failure occurs, European standards require that the lift goes to the first floor to allow the users get out of the car there, for a possible evacuation of the building, as the energy failures are caused for several reasons. This is called “**DOWN CORRECTION**”, and in the main grafctet corresponds to stages 40 and 41 (Figure 61).

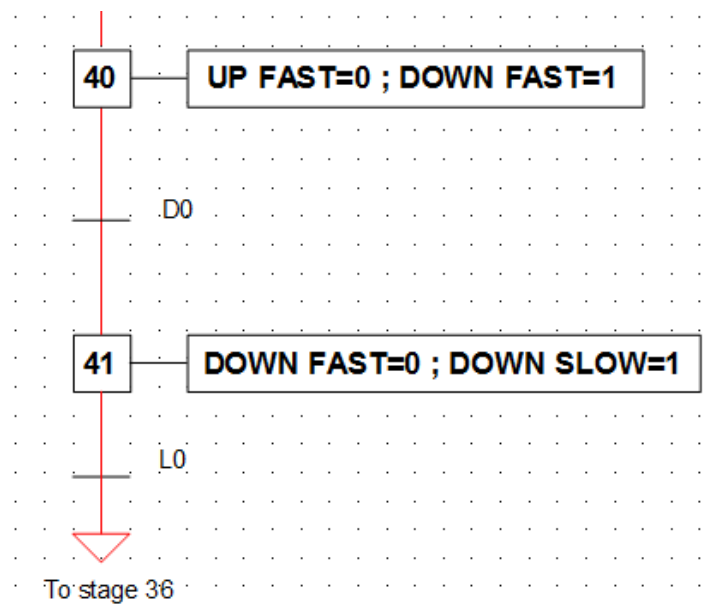


Figure 63. Down correction

Now that the main grafctet is explained, in next section we can see the different macro-stages that form all the automatic process.

2.2.3. M1 – Level control

The level control is easy to understand. Depending on the level where the car is, a value is saved in the variable ALEVEL, from value 0 for level 0 to value 9 for level 9 (Figure 62).

It can be found in section “GRAFSETS – GRAFCET NUM.: 2 – LEVEL CONTROL”.

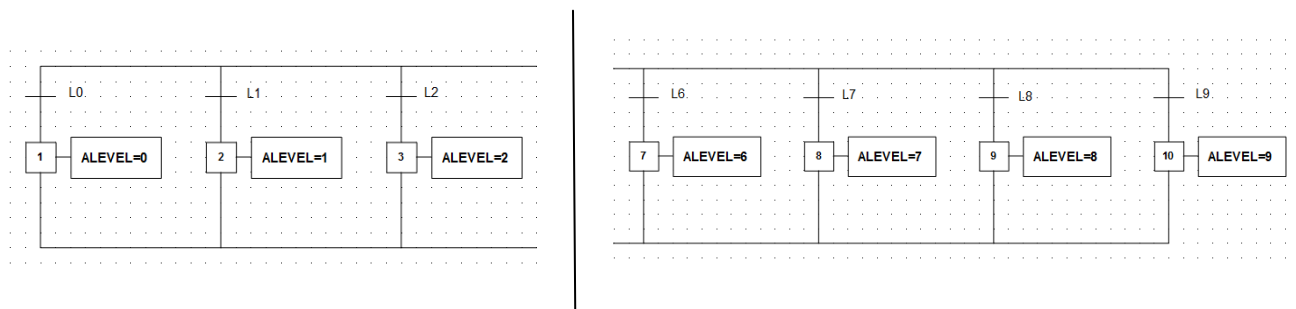


Figure 64. Level control

2.2.4. M2 – Button control

The button control is an OR disjunction like the level control, but in this case a different value is saved in variable LEVEL depending on the button pressed. When a button is pressed, it does not remain pressed, so the program must save in a marker the fact that a certain button has been pressed. This action is made by setting different markers, from M0.0 for buttons level 0 to M0.9 for buttons level 9. Other important characteristic of this control is that the internal keypad must take priority over the external buttons, as if a user is inside the car, the program must respond first to the user or users inside the car than the users outside. This is made by the next equation, for instance for level 0, but it is repeated for all levels.

$$EQ0=A0+P0*(A0+A1+A2+A3+A4+A5+A6+A7+A8+A9)$$

This equation means that the button P0 is only allowed if there is not any internal button pressed.

In (Figure 63) the grafcet for the first three levels is shown, and it is the same for the other levels.

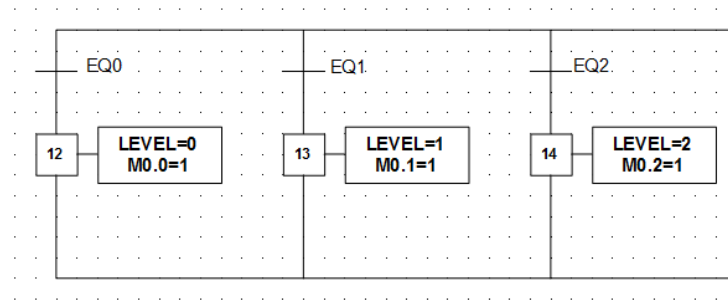


Figure 65. Button control

After a button has been pressed, the external buttons are lighted to indicate other users that the lift is being used, as well as the car light is turned on (Figure 64).

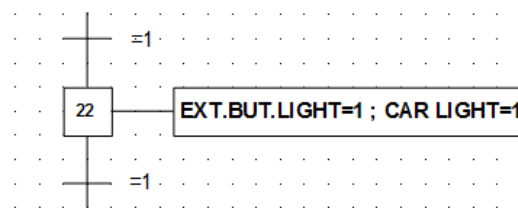


Figure 66. Car and external lights

After this stage the program continues in main grafcet.

This grafcet can be found in section “GRAFSETS – GRAFCET NUM.: 3 – BUTTON CONTROL”.

2.2.5. M3 - Door control

The last grafcet to be explained is the door control. When the door control is activated, the motor that controls the door device is activated in open direction. The door keeps opening until the limit switch DOOR OPENED is reached, then the motor stops opening the door. After that the program waits ten seconds until the motor starts closing the door, which keeps closing until the limit switch DOOR CLOSED is reached. If during the closing movement or before it starts, an object is placed in the

door's path the door opens again thanks to the SECURITY light sensor. The same action is made if the button OPENBUT placed inside the car is pressed (Figure 65).

This grafcet can be found in section "GRAFSETS – GRAFCET NUM.: 4 – LEVEL CONTROL".

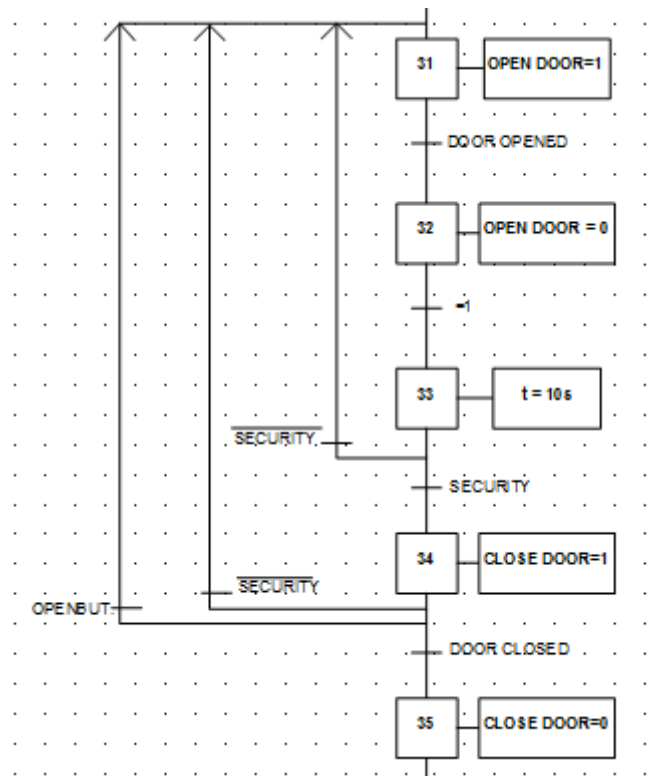


Figure 67. Door control

With the explanation of this grafcet all the automatic design process is completed, starting from the simulation to demonstrate that the algorithm designed is valid, and finishing with the grafcet design for the case of a lift with ten levels.



CALCULATIONS



3. CALCULATIONS

In this section all the calculations needed for the electrical installation are detailed. The most important point in this chapter is to calculate the necessary power for the motor to calculate after that all the devices and conductors sections for each circuit. Circuit distribution and conductors sections can be found in PLOT NUM.: 5.

3.1. COUNTERWEIGHT

The power of the motor is directly related with the counterweight installed. The point of this counterweight is to reduce the load of the motor, thus reducing the power and dimensions of the installation. In this case, the counterweight is calculated to balance the car weight and 50% of the load.

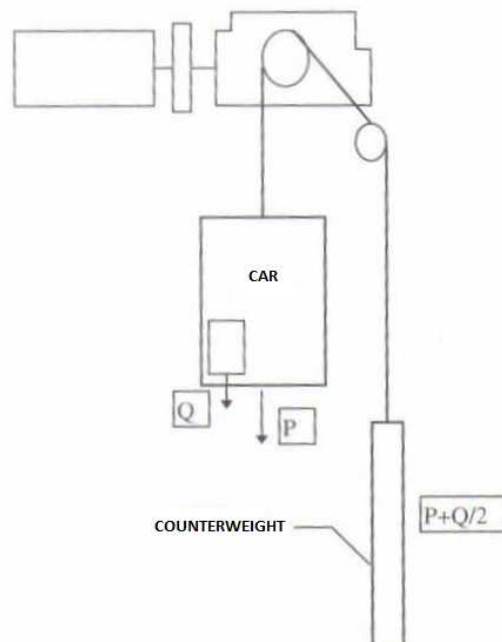


Figure 68. Counterweight calculation

This equation is valid when the total travel of the car is not higher than 35 metres, like in this case. If it is higher it is necessary to install a compensation cable. A general weight has been considered for the car, 650Kg and for the load it is used the maximum weight allowed for the European standard for an elevator with a capacity of six persons, 450Kg. Then, the counterweight is calculated as follows:

$$Z = P + \frac{Q}{2} = 650 + \frac{450}{2} = 875Kg.$$

3.2. MOTOR POWER

The motor power calculation is one of the most important goals of the electrical installation, as the rest of the installation is dimensioned depending on that power. With the counterweight installed this power is considerably reduced, thus using a smaller motor than if this counterweight is not installed, so the necessary torque is lower and so is the cost of the installation.

To calculate the motor power next formula is used:

$$P = \frac{1}{2} F \frac{v}{1000 \cdot \eta}$$

Where:

- P is the motor power, in kW.
- $\frac{1}{2}$ is because the car's weight and half of the payload are compensated by the counterweight.
- F is the maximum load allowed in the car in Kgf. For six persons the maximum weight allowed is 450 Kg. To change it into Kgf the force is multiplied by 9.8. So $F=450 \cdot 9,8=4410$ Kgf.
- v is the car medium speed during the travel. European standards establish that in lifts used by persons, like in this case, the medium speed of the car must be 1m/s.
- η is the mechanical performance. It is selected a minimum desired performance of 85%.

Thus, the necessary motor power is calculated as follows:

$$P = \frac{1}{2} F \frac{v}{1000 \cdot \eta} = \frac{1}{2} \cdot 4410 \cdot \frac{1}{1000 \cdot 0.85} = 2,6 \text{ kW}$$

The selected motor power is **3 kW**, in order to have an extra power in case of unforeseen loads.

3.3. CONDUCTORS SECTION AND ELECTRICAL DEVICES DIMENSIONING

To calculate the section of each circuit (PLOT NUM.: 5 - PROTECTIONS), next formulas are used.

For three phase circuits (MOTOR CIRCUIT) the formula used is:

$$S = \frac{\rho \cdot P \cdot L}{U \cdot \Delta U}$$

For the rest of the circuits, single phase circuits, the formula used is:

$$S = \frac{2 \cdot \rho \cdot P \cdot L}{U \cdot \Delta U}$$

Where:

- $S \rightarrow$ section in mm^2 .
- $\rho \rightarrow$ copper resistivity ($1/56 \Omega\text{m}$)
- $P \rightarrow$ Power transported through the circuit
- $L \rightarrow$ Circuit length
- $U \rightarrow$ Circuit tension
- $\Delta U \rightarrow$ maximum permissible tension fall (5%)

Thus, the section for each circuit is calculated as follows:

MOTOR CIRCUIT:

Data: $P=3000W$; $L=5m$; $U=400V$; $\Delta U=400 \cdot 0.05=20$

The amperage required by the motor is:

$$I = \frac{P}{\sqrt{3} \cdot V \cdot \cos\varphi} = \frac{3000}{\sqrt{3} \cdot 400 \cdot 0,85} = 5,094 \text{ A}$$

Standard section for this amperage is $1,5\text{mm}^2$ (*ITC-BT-19, table1.*)

It must be checked if the voltage drop allowed is not reached

$$\Delta U = \frac{P \cdot L}{e \cdot V \cdot S} = \frac{3000 \cdot 5}{56 \cdot 400 \cdot 1,5} = 0,45V < 20V$$

Then, the section $1,5\text{mm}^2$ is valid for the motor circuit. The maximum current for this section is 10A, so the mini circuit breaker installed is 4p/10A. The residential-current circuit breaker installed must be higher than the MCB. The next normalized RCCB higher than 10A is 25A, and the minimum current detection must be 30mA. Then, the RCCB installed for this circuit will be 4p/25A/30mA.

The maximum power in this circuit is:

$$P = V \cdot I \cdot \cos\varphi = 400 \cdot 10 \cdot 1 = 4000W$$

LIGHTS:

Data: $P=40W \cdot 12\text{bulbs} + 15W \text{ LED} \approx 500W$; $L=10 \text{ floors} \cdot 2,5\text{m/floor} + 5\text{extra}=30\text{m}$;
 $U=230V$; $\Delta U=230 \cdot 0,03=6,9$.

$$S = \frac{2 \cdot \rho \cdot P \cdot L}{U \cdot \Delta U} = \frac{2 \cdot \frac{1}{56} \cdot 500 \cdot 30}{230 \cdot 6,9} = 0,34 \text{ mm}^2$$

Standard section immediately above the calculated is $1,5\text{mm}^2$.

The maximum current for this section is 10A, so it must be checked if it is not reached:

$$I = \frac{P}{V \cdot \cos\varphi} = \frac{500}{230 \cdot 1} = 2,17 < 10A$$

And the maximum tension fall for this circuit is:

$$\Delta U = \frac{2 \cdot L \cdot P}{C \cdot V \cdot S} \cdot \frac{100}{400} = \frac{2 \cdot 30 \cdot 500}{56 \cdot 230 \cdot 1,5} \cdot \frac{100}{230} = 0,67\% < 5\%$$

Then, the section $1,5\text{mm}^2$ is valid for the lights circuit. The maximum current for this section is 10A, so the mini circuit breaker installed is 2p/10A. The residential-current circuit breaker installed must be higher than the MCB. The next normalized RCCB higher than 10A is 25A, and the minimum current detection must be 30mA. Then, the RCCB installed for this circuit will be 2p/25A/30mA.

The maximum power in this circuit is:

$$P = V \cdot I \cdot \cos\varphi = 230 \cdot 10 \cdot 1 = 2300W$$

CONTROL PANEL:

Data: $P=230V \cdot 5A=1150W$ (the maximum admissible current for the power supply of the control panel is 5A) ; $L=4m$ (connection between panels); $U=230V$; $\Delta U=230 \cdot 0.05=11,5$.

$$S = \frac{2 \cdot \rho \cdot P \cdot L}{U \cdot \Delta U} = \frac{2 \cdot \frac{1}{56} \cdot 1150 \cdot 4}{230 \cdot 11,5} = 0,06 \text{ mm}^2$$

Standard section immediately above the calculated is $1,5\text{mm}^2$.

The maximum current for this section is 10A, so it must be checked if it is not reached:

$$I = \frac{P}{V \cdot \cos\varphi} = \frac{1150}{230 \cdot 1} = 5 < 10A$$

And the maximum tension fall for this circuit is:

$$\Delta U = \frac{2 \cdot L \cdot P}{C \cdot V \cdot S} \cdot \frac{100}{400} = \frac{2 \cdot 4 \cdot 1150}{56 \cdot 230 \cdot 1,5} \cdot \frac{100}{230} = 0,2\% < 5\%$$

Then, the section $1,5\text{mm}^2$ is valid for the control panel circuit. The maximum current for this section is 10A, so the mini circuit breaker installed is 2p/10A. The residential-current circuit breaker installed must be higher than the MCB. The next normalized RCCB higher than 10A is 25A, and the minimum current detection must be 30mA. Then, the RCCB installed for this circuit will be 2p/25A/30mA.

The maximum power in this circuit is:

$$P = V \cdot I \cdot \cos\varphi = 230 \cdot 10 \cdot 1 = 2300W$$

POWER PLUGS:

Data: $P=230V \cdot 16A=3680W$ (The maximum admissible current for Shucko plugs is 16A); $L=10 \text{ floors} \cdot 2,5\text{m/floor}+5\text{extra}=30\text{m}$; $U=230V$; $\Delta U=230 \cdot 0.05=11,5$.

$$S = \frac{2 \cdot \rho \cdot P \cdot L}{U \cdot \Delta U} = \frac{2 \cdot \frac{1}{56} \cdot 3680 \cdot 30}{230 \cdot 11,5} = 1,48 \text{ mm}^2$$

Minimal standard section for 16A is $2,5\text{mm}^2$.

And the maximum tension fall for this circuit is:

$$\Delta U = \frac{2 \cdot L \cdot P}{C \cdot V \cdot S} \cdot \frac{100}{400} = \frac{2 \cdot 30 \cdot 3680}{56 \cdot 230 \cdot 1,5} \cdot \frac{100}{230} = 4,96\% < 5\%$$

Then, the section $2,5\text{mm}^2$ is valid for the plugs circuit. The maximum current for this section is 16A, so the mini circuit breaker installed is 2p/16A. The residential-current circuit breaker installed must be higher than the MCB. The next normalized RCCB higher than 16A is 25A, and the minimum current detection must be 30mA. Then, the RCCB installed for this circuit will be 2p/25A/30mA.

The maximum power in this circuit is:

$$P = V \cdot I \cdot \cos\varphi = 230 \cdot 16 \cdot 1 = 3680W$$

EMERGENCY CIRCUIT:

Data: $P=8W \cdot 13=104W$; $L=10$ floors $\cdot 2,5m/floor+5extra=30m$; $U=230V$;
 $\Delta U=230 \cdot 0.03m=6,9$.

$$S = \frac{2 \cdot \rho \cdot P \cdot L}{U \cdot \Delta U} = \frac{2 \cdot \frac{1}{56} \cdot 104 \cdot 30}{230 \cdot 6,9} = 0,07 \text{ mm}^2$$

Standard section immediately above the calculated is $1,5\text{mm}^2$.

The maximum current for this section is 10A, so it must be checked if it is not reached:

$$I = \frac{P}{V \cdot \cos\varphi} = \frac{104}{230 \cdot 1} = 0,45 < 10A$$

And the maximum tension fall for this circuit is:

$$\Delta U = \frac{2 \cdot L \cdot P}{C \cdot V \cdot S} \cdot \frac{100}{400} = \frac{2 \cdot 30 \cdot 104}{56 \cdot 230 \cdot 1,5} \cdot \frac{100}{230} = 0,14\% < 5\%$$

Then, the section $1,5\text{mm}^2$ is valid for the emergency circuit. The maximum current for this section is 10A, so the mini circuit breaker installed is 2p/10A. The residential-current circuit breaker installed must be higher than the MCB. The next normalized RCCB higher than 10A is 25A, and the minimum current detection must be 30mA. Then, the RCCB installed for this circuit will be 2p/25A/30mA.

The maximum power in this circuit is:

$$P = V \cdot I \cdot \cos\varphi = 230 \cdot 10 \cdot 1 = 2300W$$



MAINTENANCE AND SECURITY



4. MAINTENANCE AND SECURITY

4.1. MAINTENANCE

The maintenance work is carried out once a month at least to allow the elevator to work as efficiently and safely as possible. Furthermore, necessary repairs of the components are made if necessary. There are two types of maintenance: preventive and corrective.

4.1.1. Preventive maintenance

As the name suggests, this kind of maintenance is aimed at regular monthly review of several elements of the elevator with the consequent comfort and safety improvements. The review includes:

- Cleanliness: car, counterweight and motor.
- Lubrication: rails, boxes and balances, handles and pulleys, speed limiter
- Settings: rail, adjustments control and power lifts, level sensors
- Minor repairs: lights and buttons review
- Overhaul: includes the entire operation of the lift, including security methods such as brakes, door devices, car's speed.
- It will be checked the start and stop and the presence of abnormal sounds.
- Relay contacts and manual control checking
- Car roof and wall cleaning
- Safety devices checking

Once the preventive maintenance is finished, a report must be prepared containing the following information:

- Date, start time and end of each service, client name and technical and computer data.
- Confirmation of the completion of each of the activities requested in the Service Description
- General features of the materials and parts used
- Observations and suggest of the technician, and total time



4.1.2. Corrective maintenance

This service consists of major repairs and/or lower based on the emergency calls for damage or stops. Such faults are repaired on any day and time, so the elevator must have a maintenance service available 24 hours a day, 365 days a year.

4.2. SECURITY

The elevator must be a safety installation not only for the users, but also for the technicians responsible for repair. For this reason, the machine room must have the next components to prevent or resolve minor incidents:

- Dielectric helmet to protect against electrical contacts
- Polyester cotton work clothes
- Anti dust glasses
- Insulated gloves to protect against electrical contact voltage up to 5000V
- Belt for tools
- Emergency kit
- Fire extinguisher type ABC





PLOTS AND GRAFCETS





5. PLOTS AND GRAFCETS

5.1. PLOTS

1. MOTOR
2. MANUAL CONTROL
3. AUTOMATIC CONTROL
4. BRAKE AND LIGHT CONNECTION
5. PROTECTIONS
6. SENSORS AND MEASURES

5.2. GRAFCETS

1. MAIN CONTROL
2. LEVEL CONTROL
3. BUTTON CONTROL
4. DOOR CONTROL
5. SIMULATION LEVEL 1
6. SIMULATION LEVEL 2





CONCLUSION



6. CONCLUSION

As a final conclusion of the project, several possible improvements are described. To conclude the work, major achievements are highlighted.

6.1. IMPROVEMENTS

DATA BUS:

The main advantage of the data bus is that all the wiring and car installation (excluding securities) can be replaced by a industrial data bus that implements:

- Data input of external buttons and car to the PLC
- Data input of position sensors to the PLC

The main goal of the data bus is to simplify the installation. On the other hand it allows an economical way to make an extension of the installation.

The total price of the data bus installation, including the PC programming software is about 500€.

FREQUENCY VARIATOR:

The double-speed motor with two windings can be replaced by a frequency and voltage variation system. It is installed before the motor supply. To control its operation, the contactors must be replaced and the variation signals must be PLC inputs, as upward and downward, acceleration and deceleration. A start-up data should be set as values of acceleration and slowdown etc.

Such systems greatly increase the cost of the installation but gains in passenger comfort and considerable energy saving, and it reduces peak starting current and decreases friction and potential energy when stopping.



DECODER:

To indicate the users the level where they are, a BCD decoder can be installed to show in a screen inside the elevator the floor number. The installation can be made like in the simulation. A BCD decoder is connected to level sensors and depending on the floor, the screen shows the number of the level.

MICROCONTROLLER:

Instead of use several PLC inputs to control all security devices, a microcontroller can be installed to control them. Thus, all security signals are processed by this controller and only one input is used, to connect the microcontroller with the PLC. Then, if a security fail occurs, the microcontroller informs the PLC about it. A BCD screen can be installed in the control panel, connected to the controller to inform the technician about the type of error occurred.





6.2. ACHIEVEMENTS

The main achievements of this project are:

- ✓ Design the GRAFCET for an automatic control for an elevator with ten levels.
- ✓ The designed control operation has been demonstrated by programming a simulation with a model for a case of an elevator with four levels.
- ✓ Then, it can be accepted that the algorithm designed is valid for a general case of an elevator with n levels.
- ✓ Other important achievement is the fact that all related to security has been taken into account in the current project, thus making the lift be a safety system to be used by persons.
- ✓ But not only is the automatic control designed in the project. All the electrical installation and device dimensioning has been included, involving the motor connection and all electrical protection devices.





ANNEXES





7. ANNEXES

7.1. ANNEX I: PROGRAM FOR SIMULATION WRITTEN IN INSTRUCTION LIST (IL) WITH SUCOSOFT.





7.2. ANNEX II: KLOCKNER MOELLER PS4-201-MM1 INSTALLATION





8. BIBLIOGRAPHY

[1].- PROGRAMMING SUCOCONTROL PS4-201-MM1. Ref: 4/94 AWB 27-1186-GB, *Klockner Moeller*.

[2].- TRAINING GUIDE SUCOSOFT S40 PROGRAMMING SOFTWARE. Ref: 06/99 AWB 27-1307-GB, *Klockner Moeller*.

[3].- LABORATOR DE INFORMATICA INDUSTRIALĂ. ÎNDRUMAR II, *Prof. Dr. Ing. Culea G.*

[4].- ELECTRICAL MACHINES, *Jesús Fraile Mora, Ed. Mc Graw Hill, 5th edition, 2003.*

[5].- REGLAMENTO ELECTROTÉCNICO PARA BAJA TENSIÓN, *Real Decreto 842/2002, Ed. Paraninfo.*

WEB PAGES:

[<http://www.ascensoreszener.com/>] – General information

[<http://www.thyssenkruppelevadores.com/>] – General information

[<http://www.schindler.es/esp>] – General information

[http://www.otis.com/otis/1,1352,CLI15_RES1,00.html] – General information

[http://www.otis.com/products/detail/0,1355,CLI1_PRD16916_PRT30_PST708_RES1,00.html#flash] – General information

[<http://www.ascensoreslezama.com/variadores-de-frecuencia.aspx>] – Frequency variators

[<http://www.idemver.com.ar/ascensores%20productos.html>] – Lift components

[http://www.nersolar.com/compra_online/es/30-paneles-led] – Led panels

[<http://www.wikipedia.org/>] – Lifts' history and devices definitions





Figure 45. I/O connection..... 71

Figure 46. First stage 72

Figure 47. ALEVEL value 72

Figure 48. LEVEL value..... 73

Figure 49. LEVEL & ALEVEL comparisson 73

Figure 50. LE4-501-BS1..... 76

Figure 51. Modules connection..... 76

Figure 52. I/O diagram 77

Figure 53. I/O description..... 78

Figure 54. Stage 0..... 79

Figure 55. Level control..... 79

Figure 56. Button control 79

Figure 57. Compare LEVEL & ALEVEL 80

Figure 58. Up and down movement..... 80

Figure 59. Car stop and door control 81

Figure 60. Security failure 82

Figure 61. Weight control 82

Figure 62. Security fail during movement..... 83

Figure 63. Down correction..... 83

Figure 64. Level control..... 84

Figure 65. Button control 85

Figure 66. Car and external lights..... 85

Figure 67. Door control 86

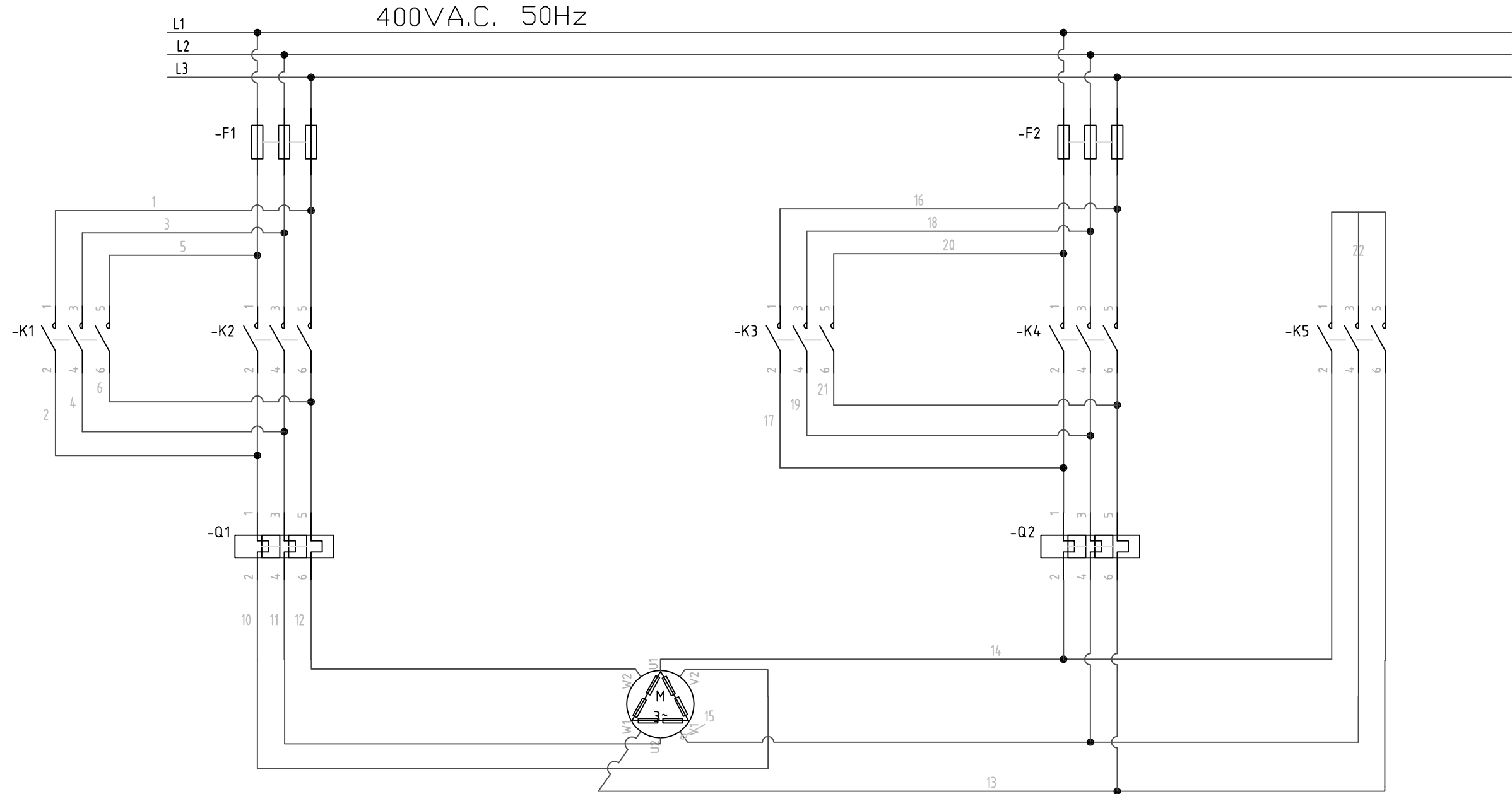
Figure 68. Counterweight calculation 88





1 2 3 4 5 6 7 8

A A



B B

C C

D D

E E

F F

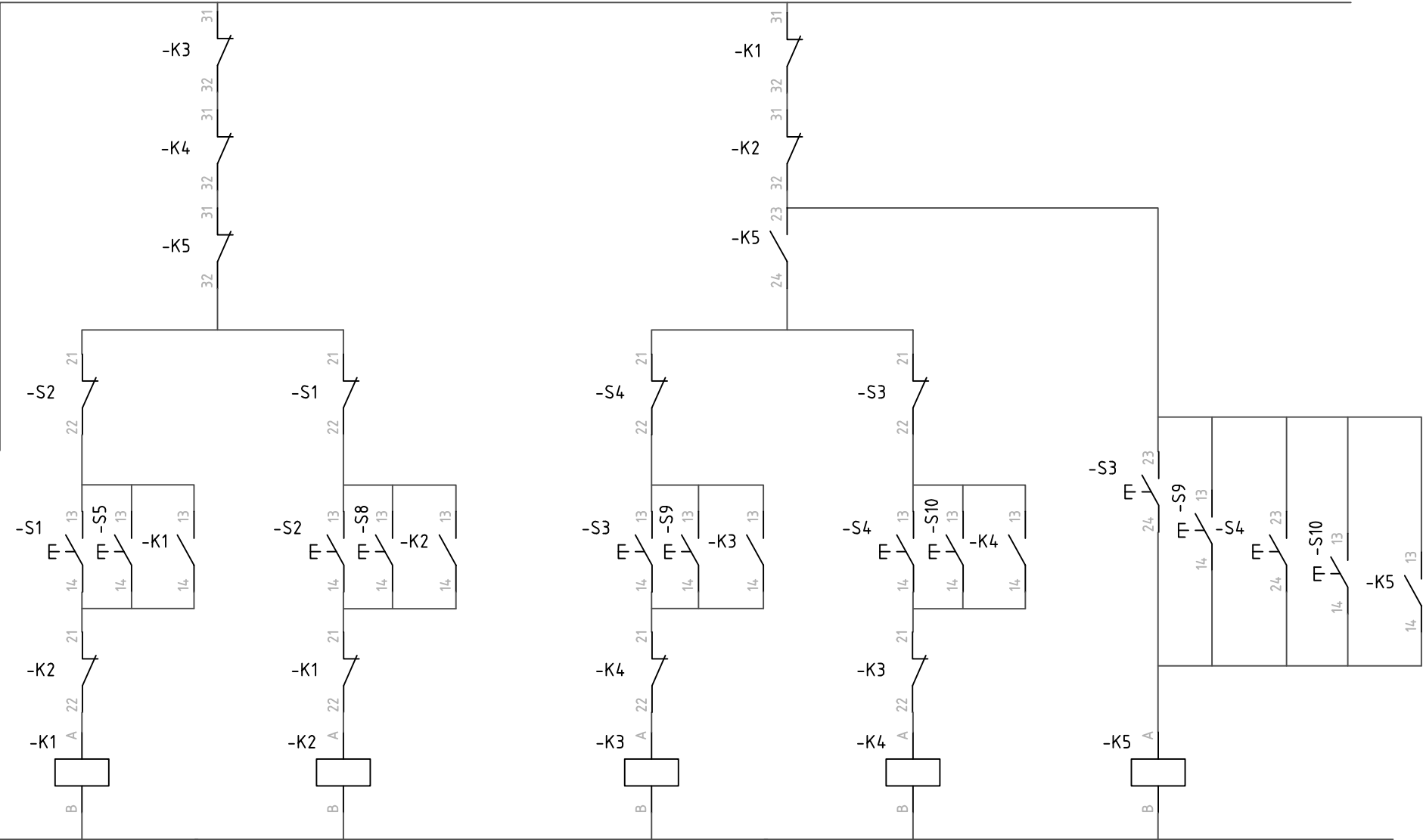
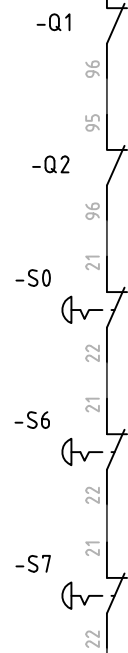
1 2 3 4 5 6 7 8

AUTOMATION PROCESS ELEVATOR SYSTEM						SCALE 1:1
				DATE	NAME	ELECTRICAL DIAGRAMS MOTOR POWER CIRCUIT
				Edit 16.03.2012	S. Noval	
				Chec.		
						PLOT NUM. : 1
						SHEETS 6
No:	Revision	Date	By			FILENAME POWERMOTOR

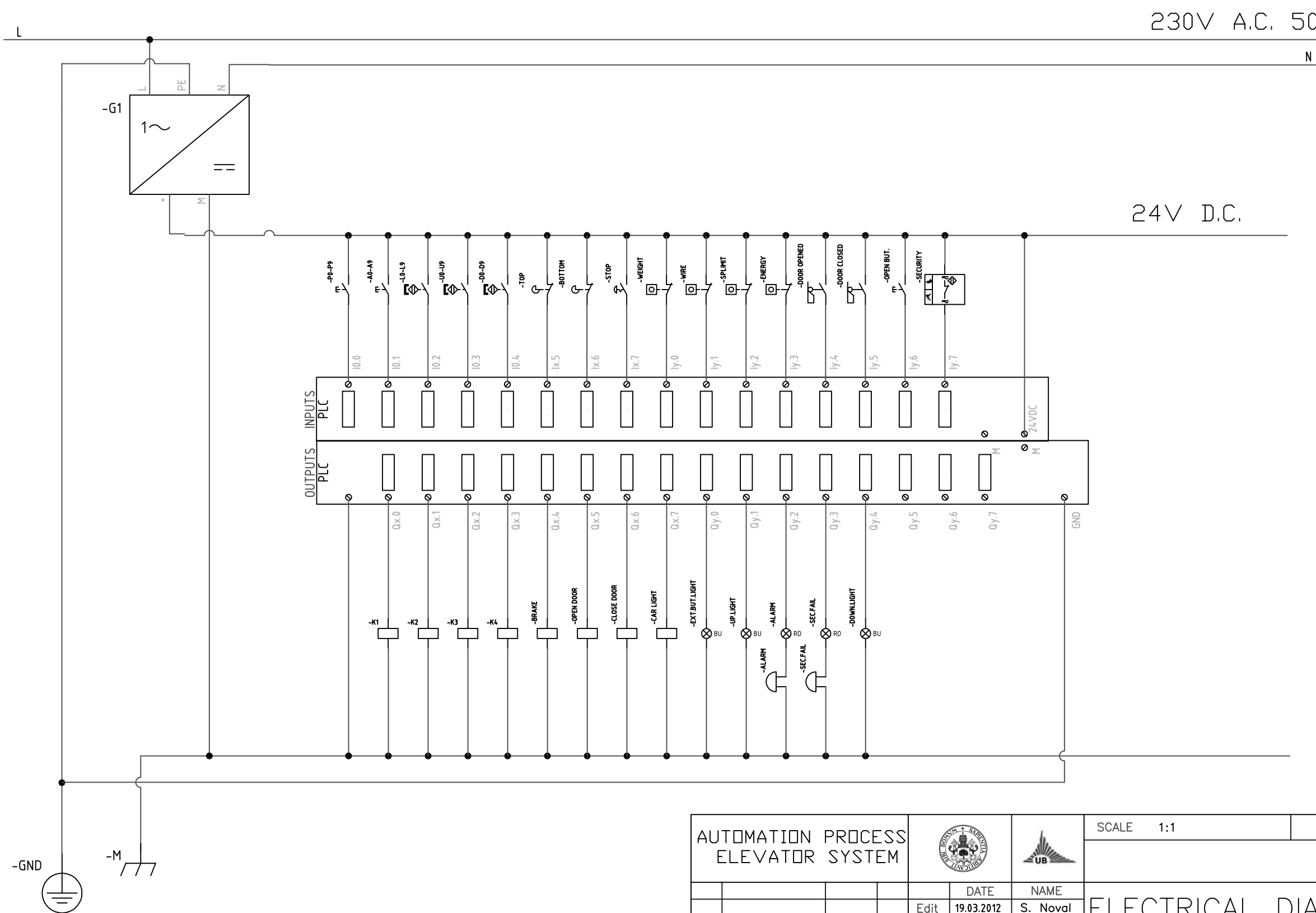
L1

-F5

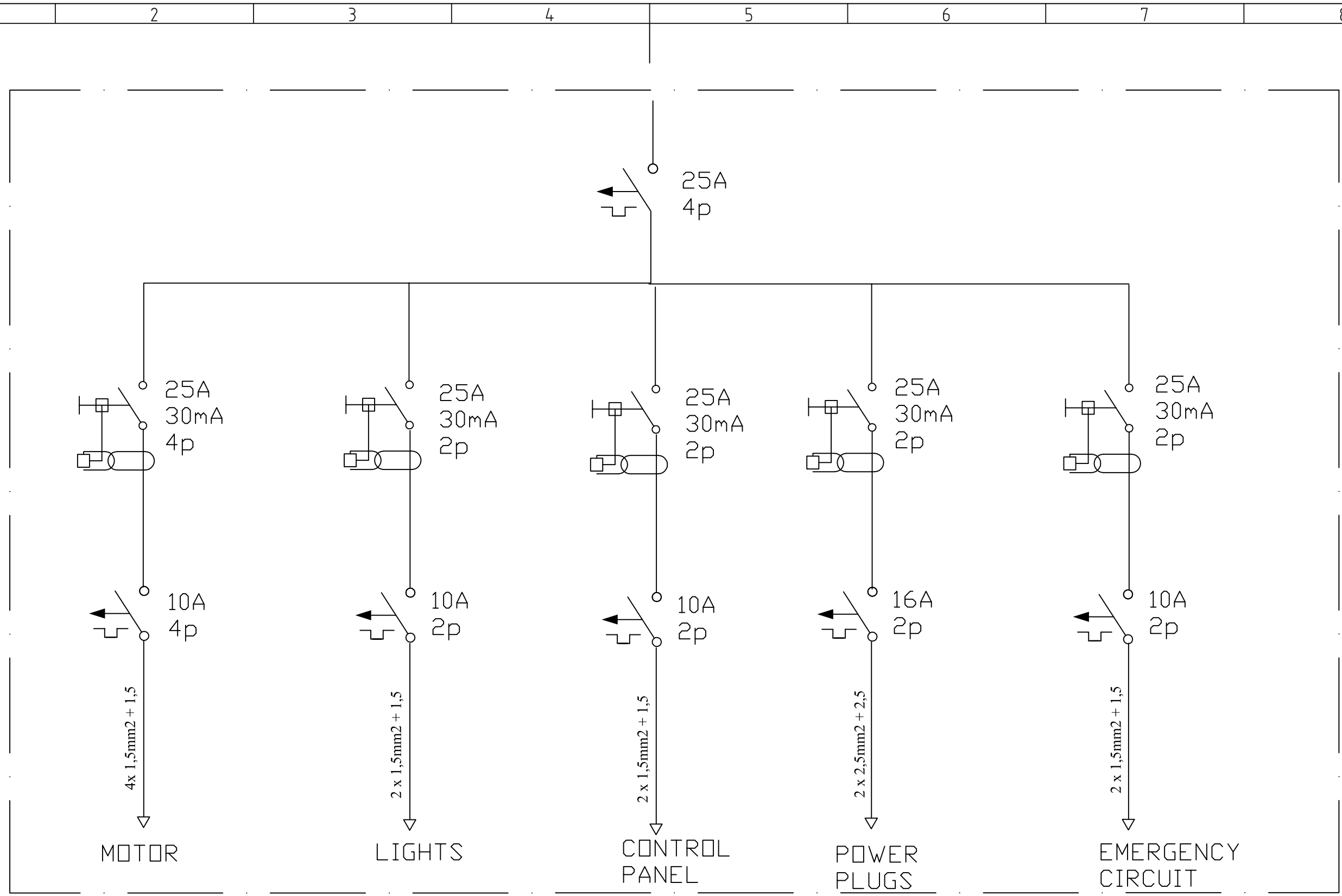
230V A.C. 50Hz.





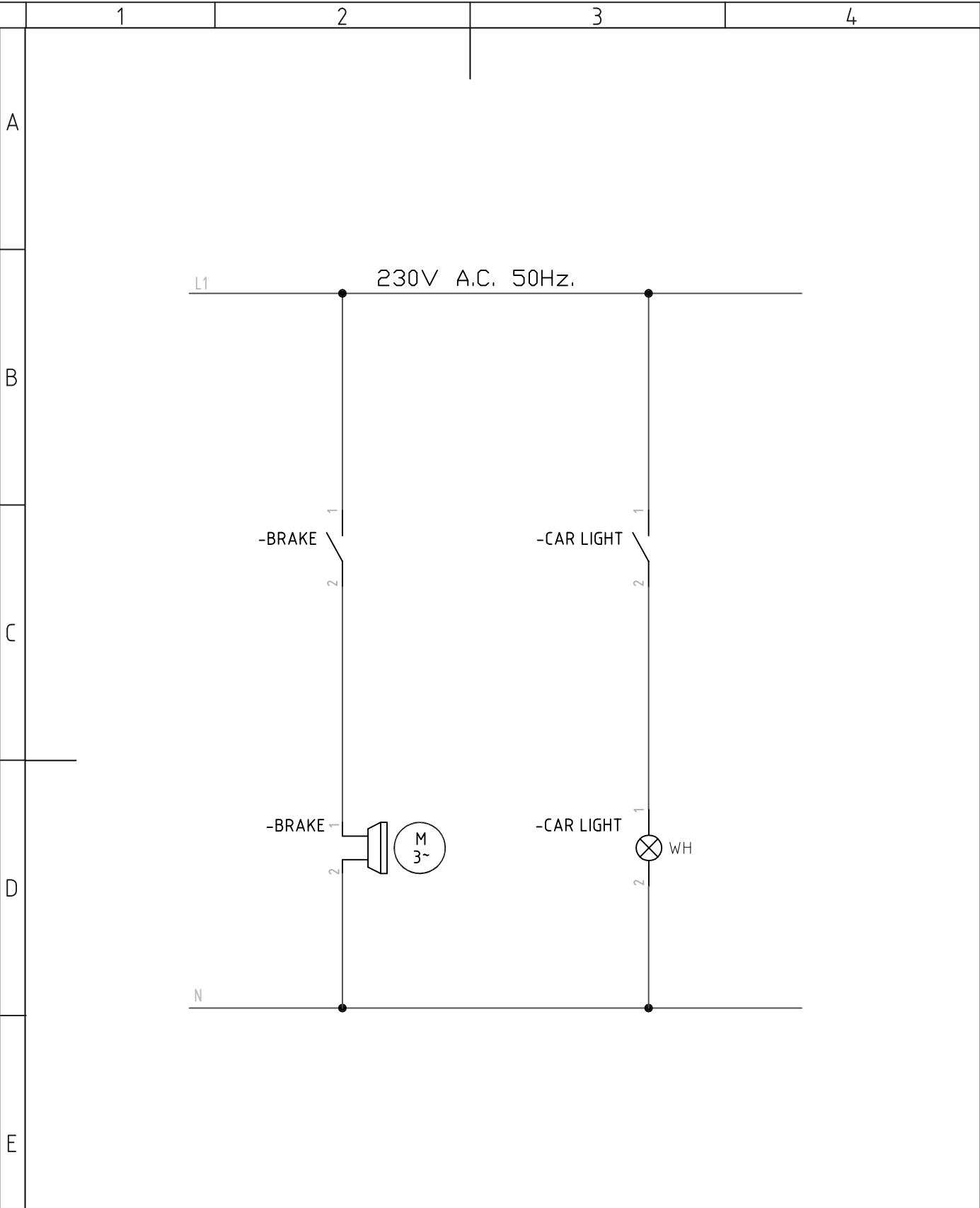
AUTOMATION PROCESS ELEVATOR SYSTEM						SCALE 1:1
				DATE	NAME	ELECTRICAL DIAGRAMS MANUAL CONTROL
				Edit 18.03.2012	S. Noval	
				Chec.		
						PLOT NUM.: 2
						SHEETS 6
No:	Revision	Date	By			FILENAME MANUALCONTROL



AUTOMATION PROCESS ELEVATOR SYSTEM						SCALE 1:1
				DATE	NAME	ELECTRICAL DIAGRAMS AUTOMATIC CONTROL
				Edit 19.03.2012	S. Noval	
				Chec.		
						PLOT NUM.: 3
No:	Revision	Date	By			FILENAME AUTOMCONTROL
						SHEETS 6



AUTOMATION PROCESS ELEVATOR SYSTEM						SCALE 1:1
				DATE	NAME	ELECTRICAL DIAGRAMS PROTECTIONS
				Edit 25.03.2012	S. Noval	
				Chec.		
						PLOT NUM.: 5
						SHEETS 6
No:	Revision	Date	By	FILENAME PROTECTIONS		



AUTOMATION PROCESS
ELEVATOR SYSTEM



SCALE 1:1

	DATE	NAME
Edit	25.03.2012	S. Noval
Chec.		

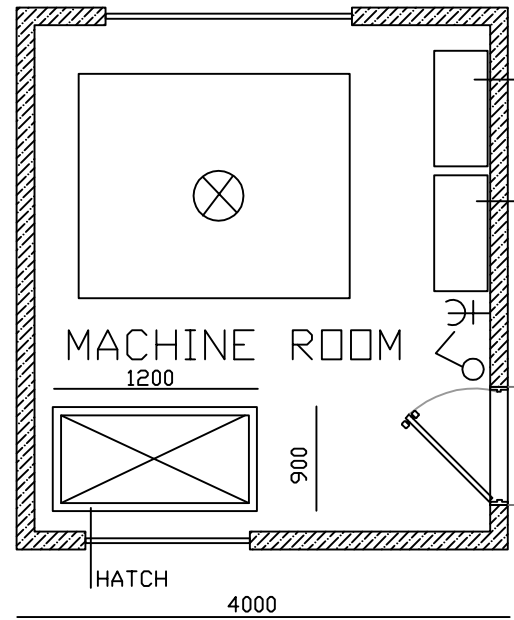
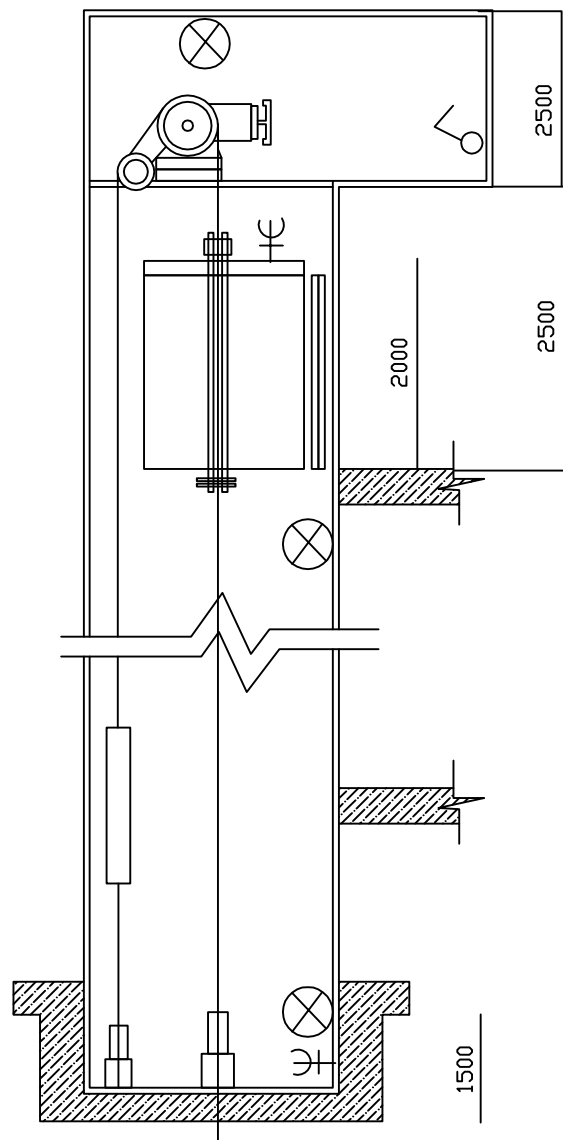
ELECTRICAL DIAGRAMS
BRAKE AND CAR LIGHT

PLOT NUM.: 4

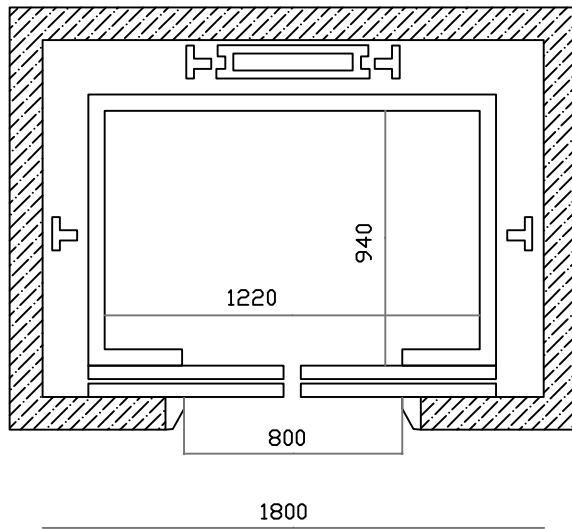
SHEETS
6

No:	Revision	Date	By

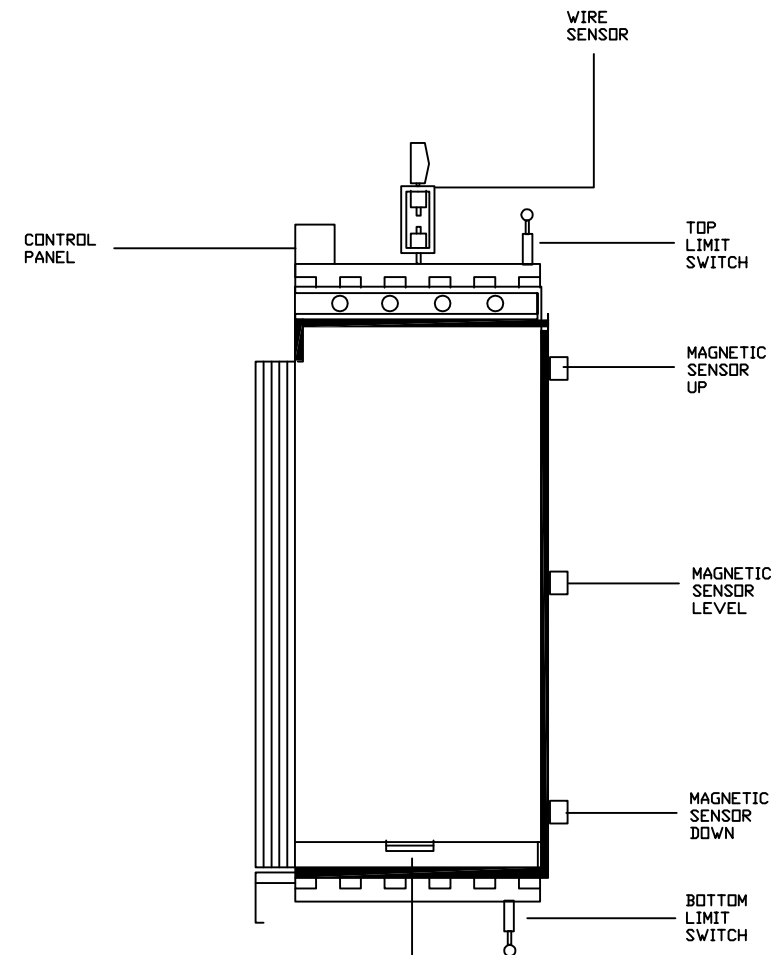
FILENAME BRAKE-LIGHT



CONTROL PANEL
PROTECTION PANEL



CAR DIMENSIONS



CAR SENSORS

AUTOMATION PROCESS ELEVATOR SYSTEM						SCALE 1:1
				DATE	NAME	ELECTRICAL DIAGRAMS SENSOR AND LIGHT POSITION
				Edit 25.03.2012	S. Noval	
				Chec.		
No:	Revision	Date	By			PLOT NUM.: 6
						SHEETS 6
						FILENAME SENSLIGHTPOS

KLOCKNER MOELLER PS4-201

Application Note

This document was written to provide assistance in configuring a G3 operator interface terminal to allow communications with a Klockner Moeller PS4-201 PLC. Please read this document carefully before attempting to configure communications with these devices.

KNOWLEDGE OF UNIT OPERATION IS ASSUMED

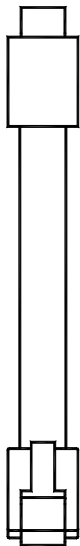
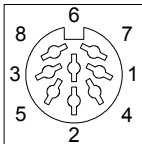
In all cases, both familiarity with the control functions and knowledge of system operation are assumed.

COMMUNICATIONS

Communications with the Klockner Moeller PS4-201 is via an RS-232 point-to-point link. The default serial communications format is RS-232, baud rate of 9600, 8 data bits, No parity, and 2 stop bits. RS-485 multi-drop cannot be used due to a baud rate incompatibility. The Moeller K-Suconet RS-485 port communicates at either 187.5K baud or 375K baud, whereas the current line of Red Lion HMI's cannot communicate at either of those baud rates.

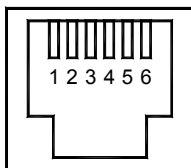
The connection details are described in the table below.

RS232



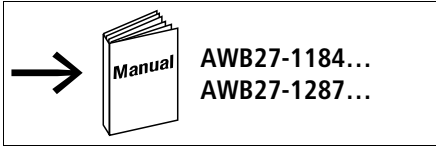
Connections			
FROM RLC UNIT	Name	CONNECTER PINOUT	
		RJ12	8P DIN
1	CTS	1, 6	-
2	Rx	2	5
3	COMM	3	3
4	COMM	-	-
5	Tx	5	2
6	RTS	1, 6	-

The above table denotes the pin names of the RS232 port. When connecting, the pin name at the RS232 port is connected to the opposite of that pin name at the destination device.



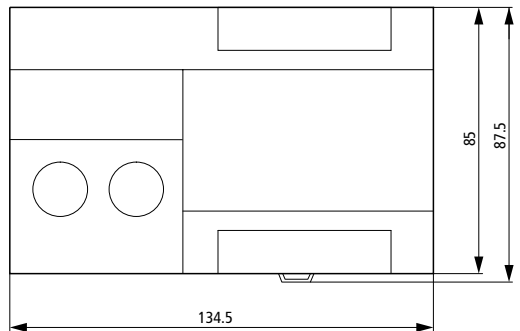
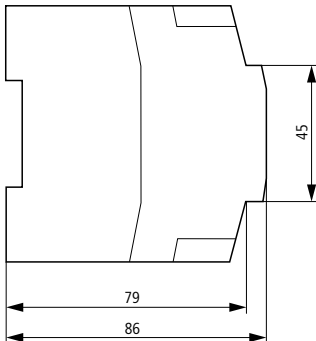
RS232 PORT
(FROM RLC UNIT)

PS4-201-MM1



Das Gerät ist für den industriellen Einsatz geeignet (→ EN 55011/22 Klasse A).
The device is suitable for use in industrial environments (→ EN 55011/22 Class A).
L'appareil a été conçu pour l'emploi en milieu industriel (→ EN 55011/22 classe A).
L'apparecchio è adatto per uso in ambienti industriali (→ EN 55011/22 Classe A).
El aparato es adecuado para uso en ambiente industrial (→ EN 55011/22 clase A).

Abmessungen – Dimensions – Dimensioni – Dimensiones [mm]

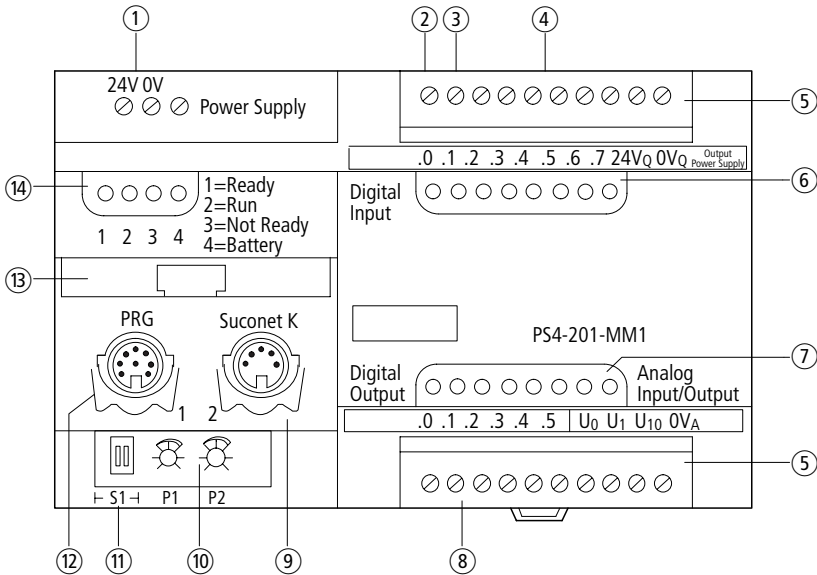


UL/CSA

Power Supply	# of channels and Input Rating	# of channels and Output Rating
24 V DC	8 digital	6 digital
0.8 A	24 V DC	24 V DC, 0.5 A
	2 analog	24 V DC, 0.5 A p. d. ¹⁾
		1 analog

1) p. d. Pilot Duty/Maximum operating temperature 55 °C/
Tightening torque 0.6 Nm/AWG 12-28

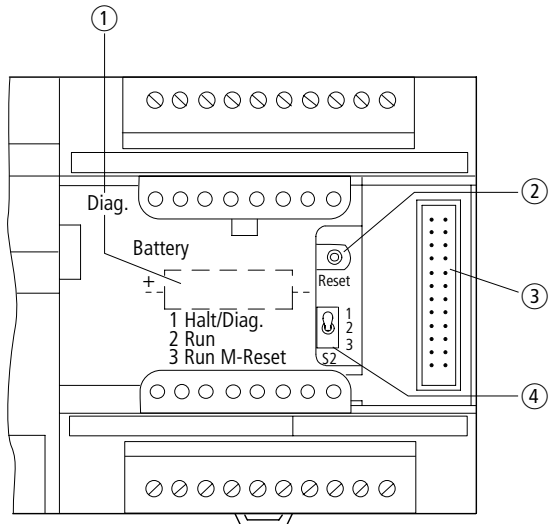
Frontansicht – Front view – Face avant – Vista frontale – Vista de frente



- | | | |
|---|--|---|
| <ul style="list-style-type: none"> ① 24-V-DC-Stromversorgung ② Eingang: „Schneller Zähler“, (3 kHz) ③ Alarmeingang ④ 8 Digital-Eingänge 24 V DC und 24-V-DC-Versorgung für die Ausgänge ⑤ Steckbare Schraubklemme | <ul style="list-style-type: none"> ⑥ Statusanzeige der Eingänge ⑦ Statusanzeige der Ausgänge ⑧ 6 Digital-Ausgänge 24 V DC/0,5 A; kurzschlussfest und überlastsicher 2 Analog-Eingänge (0 bis 10 V) 1 Analog-Ausgang (0 bis 10 V) ⑨ Suconet-K-Schnittstelle/seriell transparent (RS 485) | <ul style="list-style-type: none"> ⑩ Sollwertgeber P1, P2 ⑪ Schalter S1 für Busabschlusswiderstände ⑫ Programmiergeräte-Schnittstelle (PRG)/seriell transparent (RS 232) ⑬ Speichermodul ⑭ Statusanzeige der Steuerung |
|---|--|---|

**Offene Gehäuseklappe
Housing cover open
Couvercle ouverte
Calotta della custodia aperta
Tapa abierta**

- ① Batterie
- ② Reset-Taste
- ③ Stiftheiste für Lokale Erweiterung
- ④ Betriebsarten-Vorwahlschalter S2



- ① 24 V DC power supply
- ② Input high-speed counter (3 kHz)
- ③ Alarm input
- ④ 8 24 V DC digital inputs and 24 V DC supply for the outputs
- ⑤ Plug-in screw terminal
- ⑥ LED status display for the inputs
- ⑦ LED status display for the outputs
- ⑧ 6 24 V DC/0.5 A digital outputs; short-circuit proof and overload protected 2 analog inputs (0 to 10 V) 1 analog output (0 to 10 V)
- ⑨ Suconet K interface/serial transparent (RS 485)
- ⑩ Setpoint potentiometer P1, P2
- ⑪ S1 Switch for bus terminating resistors
- ⑫ Programming device interface (PRG)/ serial transparent (RS 232)
- ⑬ Memory module
- ⑭ LED status display of the PLC

-
- ① Battery
 - ② Reset button
 - ③ Terminal for local expansion
 - ④ Mode selector switch S2

- ① 24 V DC alimentazione
- ② Ingresso contatore veloce 3 kHz
- ③ ingresso interrupt
- ④ 8 ingressi digitali 24 V DC e 24 V DC alimentazione per uscite
- ⑤ Morsetto a vite sfilabile
- ⑥ Visualizzazione di stato a LED degli ingressi
- ⑦ Visualizzazione di stato a LED degli uscite
- ⑧ 6 uscite digitali 24 V DC/0,5 A; protetto da cortocircuito e sovraccarico 2 ingressi analogici (0 a 10 V) 1 uscita analogica (0 a 10 V)
- ⑨ Interfaccia Suconet K/in serie trasparente (RS 485)
- ⑩ Potenziometro P1, P2
- ⑪ Interruttore S1 resistenze di terminazione bus
- ⑫ Interfaccia di programmazione (PRG)/ in serie trasparente (RS 232)
- ⑬ Modulo di memoria
- ⑭ Visualizzazione a LED di PLC

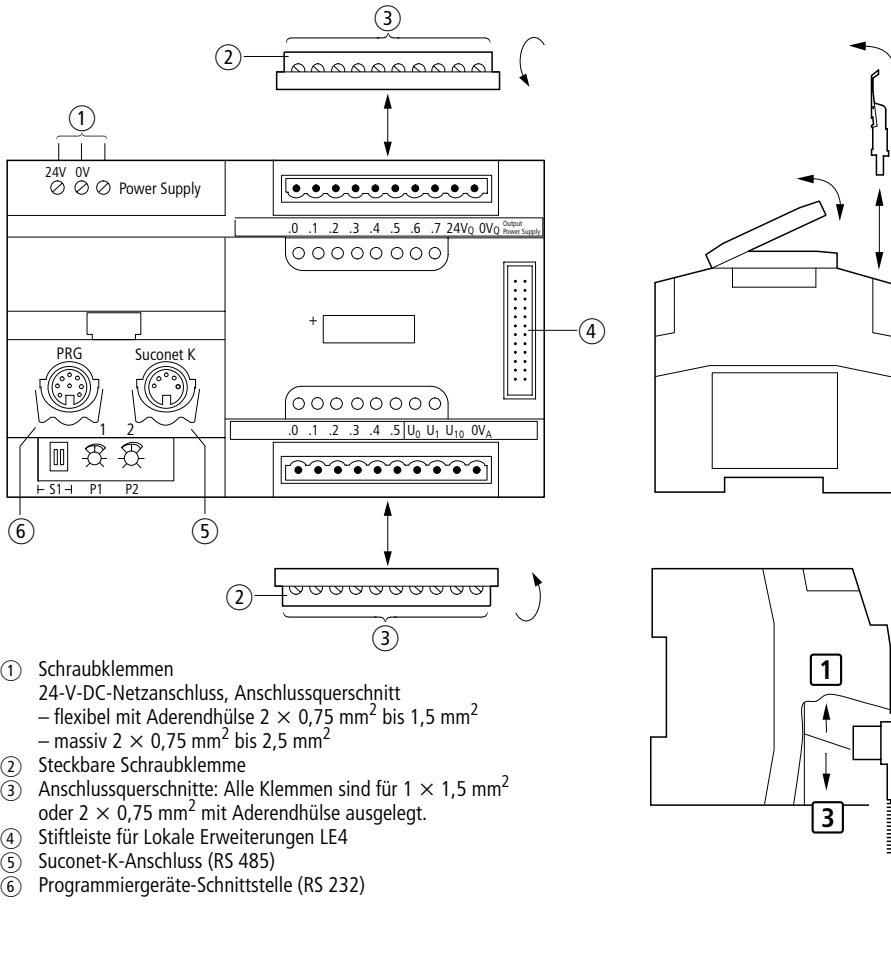
-
- ① Batteria
 - ② Tasto di reset
 - ③ Connettore per espansioni locali
 - ④ Selettore modo di funzionamento S2

- ① Alimentation secteur 24 V CC Terre de protection
- ② Entrée compteur rapide (3 kHz)
- ③ Entrée d'alarme
- ④ 8 entrées digitales 24 V CC et 24 V CC alimentation pur sorties
- ⑤ Bornier à vis enfichable
- ⑥ Afficheur d'état DEL entrées
- ⑦ Afficheur d'état DEL sorties
- ⑧ 6 sorties digitales 24 V CC/0,5 A; protection courts-circuits et surcharges 2 entrées analogiques (0 à 10 V) 1 sortie analogique (0 à 10 V)
- ⑨ Liaison Suconet K/séquentiel transparent (RS 485)
- ⑩ Module d'entrées de consignes
- ⑪ S1 interrupteur pour résistance de terminaison de bus
- ⑫ Liaison pour appareils de programmation (PRG)/ séquentiel transparent (RS 232)
- ⑬ Module de mémoire
- ⑭ Afficheur d'état DEL de l'automate

-
- ① Pile
 - ② Bouton RAZ
 - ③ Connecteur pour extensions locales
 - ④ Sélecteur modes de fonctionnement S2

- ① 24 V DC alimentación
- ② Entrada de contador rapido 3 kHz
- ③ Entrada de alarma
- ④ 8 entradas digitales 24 V DC y 24 V DC alimentación para las salidas
- ⑤ Terminal roscado enchufable
- ⑥ LED visualización entradas
- ⑦ LED visualización salidas
- ⑧ 6 salidas digitales 24 V DC/0,5 A; a prueba de cortocircuitos y seguridad contra sobrecargas 2 entradas analógicas (0 a 10 V) 1 salida analógica (0 a 10 V)
- ⑨ Interface Suconet K/en serie transparente (RS 485)
- ⑩ Encoder
- ⑪ Interruptor S1 para bus resistencias terminales
- ⑫ Interface aparatos de programación (PRG)/ en serie transparente (RS 232)
- ⑬ Módulo de memoria
- ⑭ LED visualización del PLC

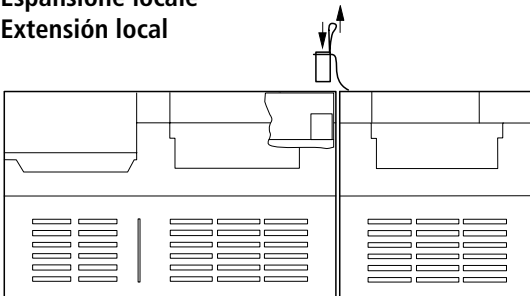
-
- ① Pila
 - ② Pulsador Reset
 - ③ Regleta de bornes para extensiones locales
 - ④ Selector de modo de servicio S2



- ① Schraubklemmen
24-V-DC-Netzanschluss, Anschlussquerschnitt
– flexibel mit Aderendhülse $2 \times 0,75 \text{ mm}^2$ bis $1,5 \text{ mm}^2$
– massiv $2 \times 0,75 \text{ mm}^2$ bis $2,5 \text{ mm}^2$
- ② Steckbare Schraubklemme
- ③ Anschlussquerschnitte: Alle Klemmen sind für $1 \times 1,5 \text{ mm}^2$
oder $2 \times 0,75 \text{ mm}^2$ mit Aderendhülse ausgelegt.
- ④ Stiftleiste für Lokale Erweiterungen LE4
- ⑤ Suconet-K-Anschluss (RS 485)
- ⑥ Programmiergeräte-Schnittstelle (RS 232)

06/01 AWA27-1589

Lokale Erweiterung
Local expansion
Extension locale
Espansione locale
Extensión local



Achtung!

Buchsenstecker nur im spannungslosen Zustand stecken oder ziehen.

Care!

Always switch off the power supply when fitting or removing the socket connector.

Attention !

Le connecteur ne doit être branché ou débranché qu' hors tension.

Attenzione!

Inserire o togliere il connettore solo a tensione disinserita.

¡Atención!

Enchufar o desenchufar al conector hembra sólo sin tensión.

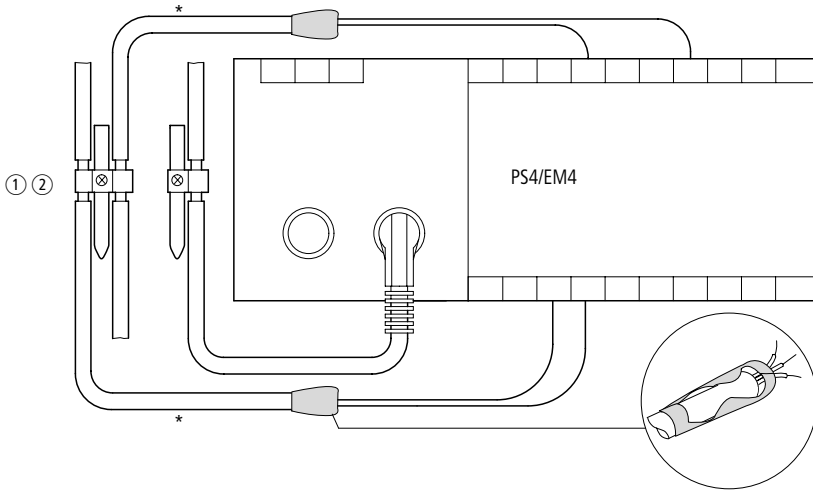
- ① Screw terminal
24 V DC power supply
Connection cross-section:
– flexible with ferrule: $2 \times 0.75 \text{ mm}^2$ to 1.5 mm^2
– without ferrule $2 \times 0.75 \text{ mm}^2$ to 2.5 mm^2
- ② Plug-in screw terminal
- ③ Connection cross-sections: All terminals are designed for $1 \times 1.5 \text{ mm}^2$ or $2 \times 0.75 \text{ mm}^2$ with ferrule
- ④ Terminal for LE4 local expander units
- ⑤ Suconet K connection (RS 485)
- ⑥ Programming device interface (RS 232)

- ① Morsetti a vite
Alimentazione 24 V DC
Sezione del cavo
– flessibile, con guaina $2 \times 0,75 \text{ mm}^2$ a $1,5 \text{ mm}^2$
– rigido $2 \times 0,75 \text{ mm}^2$ a $2,5 \text{ mm}^2$
- ② Morsetto a vite sfilabile
- ③ Sezione del cavo: tutti i morsetti sono utilizzabili per $1 \times 1,5 \text{ mm}^2$ oppure $2 \times 0,75 \text{ mm}^2$ con guaina
- ④ Connettore per espansioni locali LE4
- ⑤ Collegamento Suconet K (RS 485)
- ⑥ Interfaccia di programmazione (RS 232)

- ① Bornier à vis
Alimentation secteur 24 V CC
Section de raccordement :
– avec embout : $2 \times 0,75 \text{ mm}^2$ à $1,5 \text{ mm}^2$
– sans embout : $2 \times 0,75 \text{ mm}^2$ à $2,5 \text{ mm}^2$
- ② Bornier à vis enfichable
- ③ Section de raccordement : Toutes les bornes sont conçues pour une section $1 \times 1,5 \text{ mm}^2$ ou $2 \times 0,75 \text{ mm}^2$ et douille d'embout.
- ④ Connecteur pour extensions locales LE4
- ⑤ Raccordement Suconet K (RS 485)
- ⑥ Liaison pour appareils de programmation (RS 232)

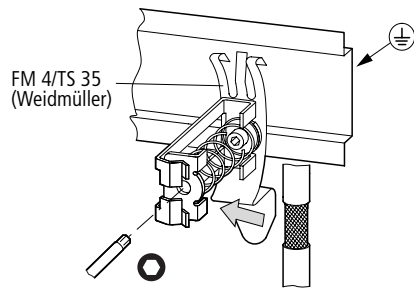
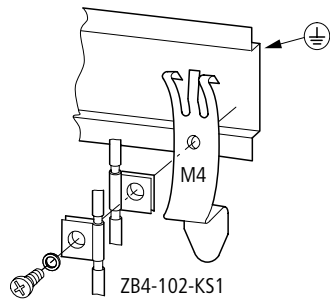
- ① Terminales roscados
Alimentación 24 V DC, Secciones de conexión:
– flexible con casquillo $2 \times 0,75 \text{ mm}^2$ a $1,5 \text{ mm}^2$
– macizo $2 \times 0,75 \text{ mm}^2$ a $2,5 \text{ mm}^2$
- ② Terminal roscado
- ③ Secciones de conexión: todos los terminales están dimensionados para $1 \times 1,5 \text{ mm}^2$ o bien $2 \times 0,75 \text{ mm}^2$ con casquillo
- ④ Regleta de bornes para extensiones locales LE4
- ⑤ Conexión Suconet K (RS 485)
- ⑥ Interface aparatos de programación (RS 232)

Schirmerdung – Earthing the screen – Mise à la terre du blindage – Collegamento alla terra dello schermo – Conexión a tierra de pantalla

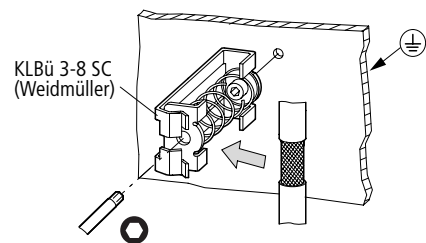
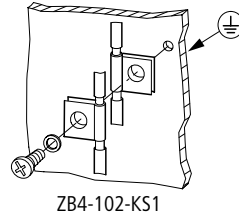


* Signalleitungen (abhängig vom Modul) – Signal cable (depending on module) – Ligne de signaux (en fonction du module) – Conduttore del segnale (dipendente dal modulo) – Línea de señalización (en función del módulo)

① für Hutschiene – for top-hat rail – pour profilé-support – per guida – para guía simétrica



② für Montageplatte – for mounting plate – pour plaque de montage – per piastra di montaggio – para placa de montaje



06/01 AWA27-1589

Alternative Schirmerdung – Alternative screen earth – Mise à la terre du blindage au choix – Collegamento alternativo dello schermo a terra – Puesta a tierra alternativa de la pantalla

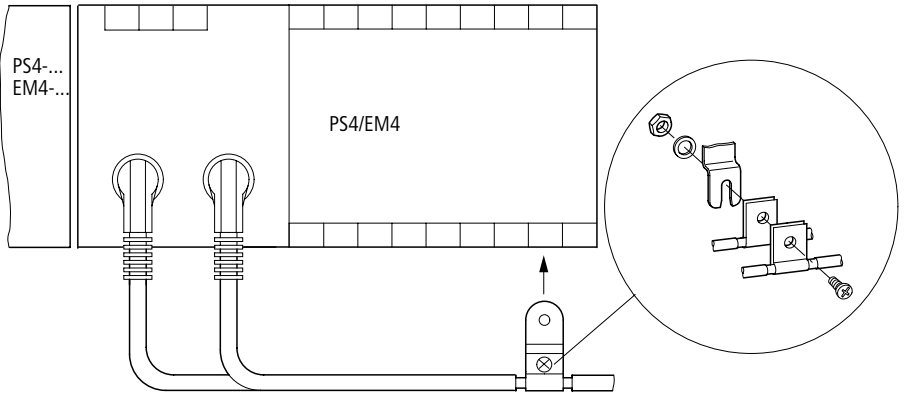
Falls die Schirmerdung auf Seite 6/10 aus Platzgründen nicht möglich ist.

In the event that the screen earthing arrangement on Page 6/10 is not possible due to lack of space.

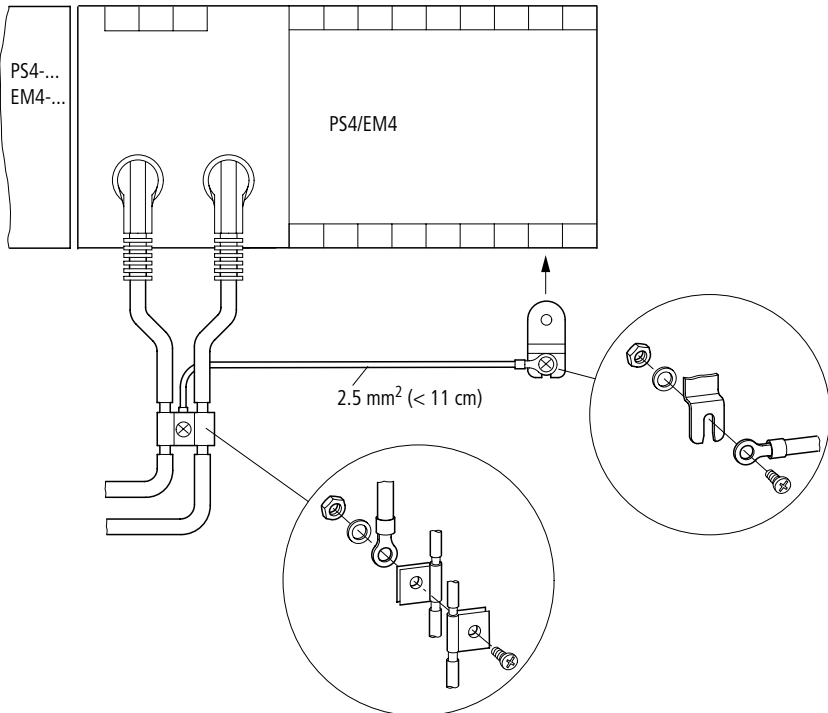
Au cas où la mise à la terre du blindage en page 6/10 est trop encombrante.

Se il collegamento a terra dello schermo di pag. 6/10 richiede troppo spazio.

En caso de que la puesta a tierra de la pantalla en página 6/10 no sea posible por razones de espacio.



06/01 AWA27-1589



Busabschlusswiderstände – Bus terminating resistors – Résistances de terminaison de bus – Resistenci di terminazione bus – Resistencias terminales de bus

Schalterstellung im Auslieferungszustand

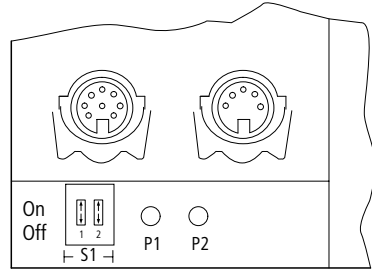
Factory setting

Position à la livraison

Impostazione di fabbrica

Posición de entrega

S1: 1 ON
2 ON



Batterie – Batery – Pile – Batteria – Pila



Achtung!

Nur im eingeschalteten Zustand stecken oder ziehen.

Attention!

Only fit or remove if switched on.

Attention !

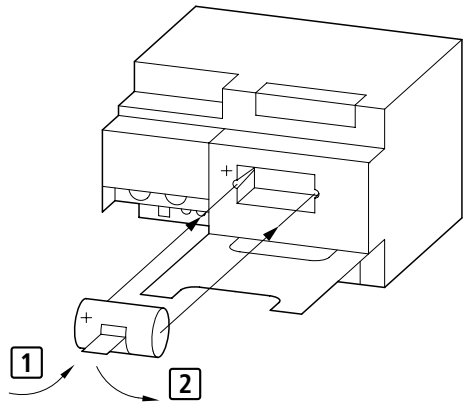
Ni enficher ni retirer que lorsque l'appareil est sous tension.

Attenzione!

Inserire/togliere solo a tensione inserita.

¡Atención!

Meter o sacar sólo con tensión.



06/01 AWA27-1589

Speichermodul – Memmory Module – Module de mémoire – Modulo di memoria – Módulo de memoria



Achtung!

Nur im spannungslosen Zustand stecken oder ziehen.

Attention!

Always switch off the power supply when fitting or removing.

Attention !

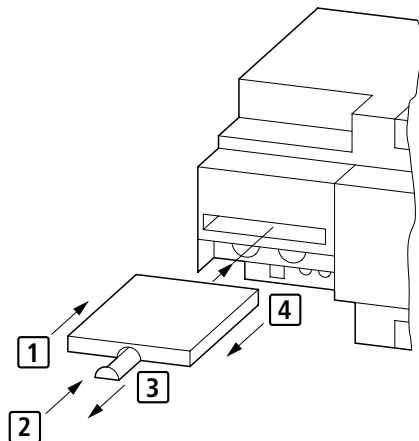
Brancher/débrancher uniquement hors tension.

Attenzione!

Inserire o togliere solo a tensione disinserita.

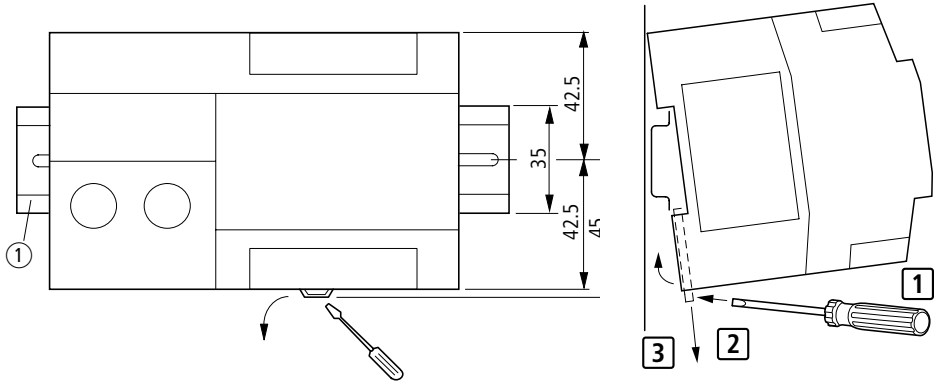
¡Atención!

Enchufar o desenchufar sólo sin tensión.



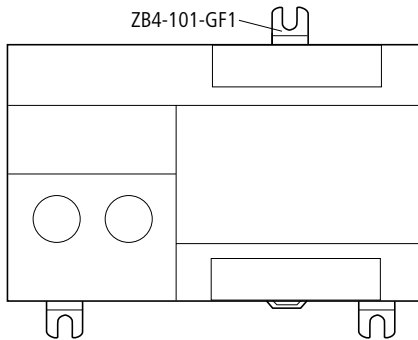
Montage – Fitting – Montaggio – Montaje

auf Montageplatte mit 35-mm-Hutschiene ① (senkrecht ohne LE oder waagrecht)
on mounting plate with 35 mm top-hat rail ① (vertical without LE or horizontal)
sur plaque de montage avec profilé-support 35 mm ① (vertical sans LE ou horizontal)
su piastra di montaggio con guida DIN 35 mm ① (verticale senza LE o orizzontale)
sobre placa de montaje con guía simétrica de 35 mm ① (vertical sin LE ó horizontal)



06/01 AWA27-1589

auf Montageplatte (senkrecht ohne LE oder waagrecht)
on mounting plate (vertical without LE or horizontal)
sur plaque de montage (vertical sans LE ou horizontal)
su piastra di montaggio (verticale senza LE o orizzontale)
sobre placa de montaje (vertical sin LE ó horizontal)



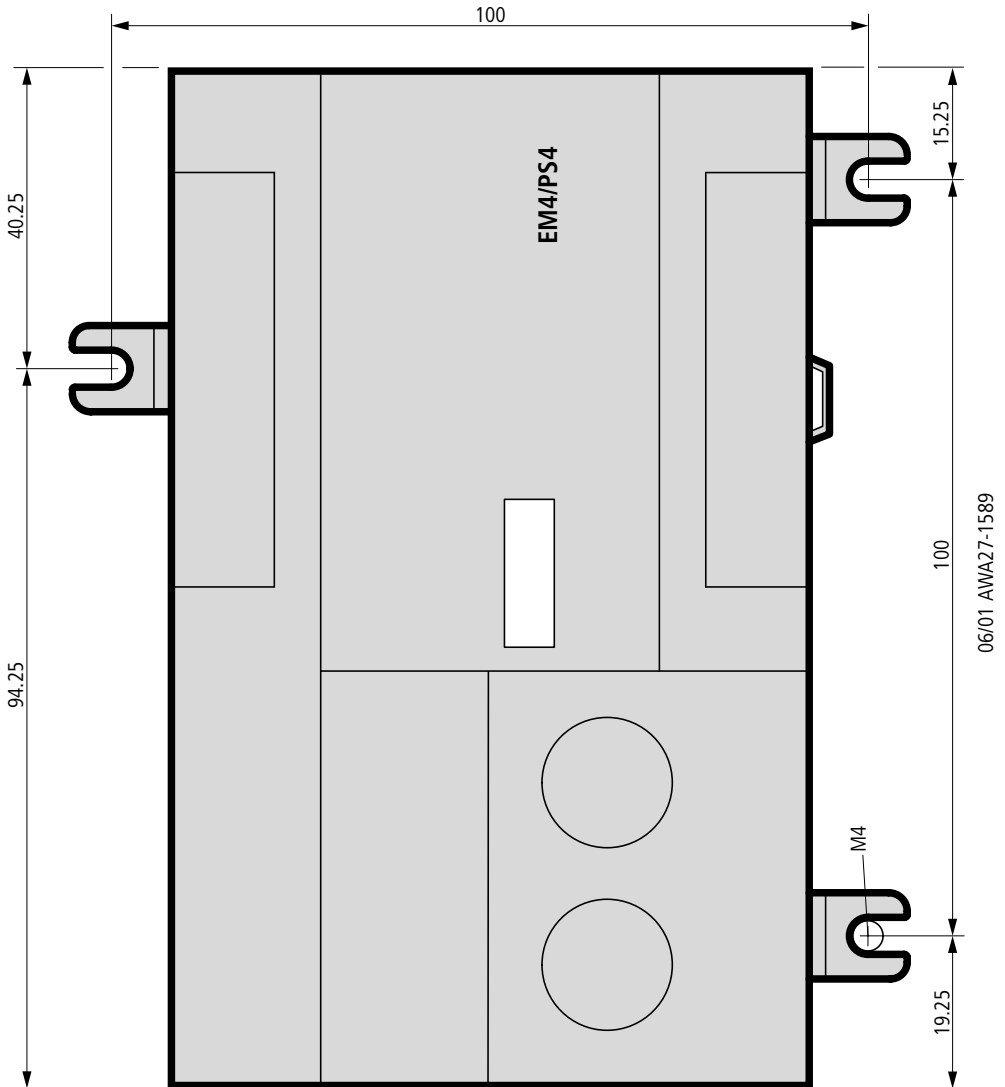
Bohrschablone M 1 : 1

Template for holes, scale 1 : 1

Gabarit de perçage, échelle 1 : 1

Dima di foratura, scala 1 : 1

Plantilla para taladros, escala 1 : 1



Programming



SUCOcontrol
PS 4-201-MM 1



SUCOS
Automation

IBM is a registered trademark of International Business Machines Corporation.

All other brand and product names are trademarks or registered trademarks of the owner concerned.

1st edition 4/94

© Klockner-Moeller, Bonn

Authors: Olaf Duda, Jiirgen Herrmann, Ralf Stang

Editor: Barbara Petrick

Translators: Karin Weber, Terence Osborn

All rights reserved, including those of the translation. No part of this manual may be reproduced in any form (printed, photocopy, microfilm or any other process) or processed, duplicated or distributed by means of electronic systems without the written permission of Klockner-Moeller, Bonn.

Subject to alteration without notice.

Printed on bleached cellulose.

100 % free from chlorine and acid.

Programming of the SUCOcontrol PS4-201-MM1

Contents

About this Manual	III
1 Programming: Procedure	1-1
2 Programming: Elements and Rules	2-1
3 Structuring Programs	3-1
4 Commissioning	4-1
5 IL Instructions	5-1
6 Function Blocks	6-1
7 SK Sequential Control Function Block	7-1
8 Indirect Addressing	8-1
9 Programming Examples	9-1
Appendix	A-1

Programming of the SUCOcontrol PS4-201-MM1 About this Manual

The documentation for the PS4-201-MM1 is divided into three sections:

- Hardware, operation, documentation
- Programming
- Hardware and engineering

The manual AWB27-1185-GB explains in which way you are supported by the SUCOsoft S30-S 4-200: with the creation of the user program in IL (Instruction List), with the device configuration and the commissioning of the controller. This manual also contains the installation instruction, the documentation of user programs, a chapter which describes the "Operation of SUCOsoft S30-S 4-200" with emphasis *on* the "IL editor" and the "Device configurator".

This manual, AWB27-1186-GB "Programming", contains information required for programming the PS 4-201-MM 1. The procedure for creating, structuring and commissioning the program is described first of all. Also included are overviews of all IL instructions and function blocks. The manual also includes practical examples of programming.

The manual AWB27-1184-GB, "Hardware and Engineering", explains how the PLC is to be mounted and designed. It describes the elements of the PS 4-201-MM1 and their functions. The chapter "Addressing" describes the general syntax rules for addressing the stations in a SUCOnet K/K1 network. This is also described in the chapter "Networking with SUCOnet K/K1" and is illustrated with examples.

Programming of the SUCOcontrol PS4-201-MM1

About this manual

The following table gives an overview of the topics described in the documentation and where they can be found. The topics are listed in the order they are normally required.

Steps	Described in
1. Installing SUCOsoft S 30-S 4-200	AWB 27-1185-GB, Chapter 1
2. Operation of SUCOsoft S 30-S 4-200	AWB 27-1185-GB, Chapter 2
3. Writing programs	AWB 27-1186-GB, Chapter 1
3.1 Setting system parameters	AWB 27-1186-GB, Chapter 1
3.2 Configuring stations	AWB 27-1186-GB, Chapter 2, Device configurator
3.3 Introduction to the IL editor	AWB 27-1185-GB, Chapter 2, IL editor
3.4 Introduction to program elements and programming rules	AWB 27-1186-GB, Chapter 2
3.4.1 Structuring programs	AWB 27-1186-GB, Chapter 3
3.5 Incorporating the configuration file in the program	AWB 27-1186-GB, Chapter 1
3.6 Entering program code	AWB 27-1186-GB, Chapter 1
4. Compiling programs	AWB 27-1186-GB, Chapter 1
5. Transferring programs to the PLC	AWB 27-1186-GB, Chapter 1
6. Commissioning the PLC	AWB 27-1186-GB, Chapter 4, AWB 27-1184-GB, Chapter 7
7. Error/diagnostics description	AWB 27-1186-GB, Chapter 4, AWB 27-1184-GB, Chapter 7
8. Program documentation	AWB 27-1185-GB, Chapter 3

1 Programming: Procedure

Contents

General	1-3
Setting system parameters	1-5
- Program test in RUN	1-6
- Start after NOT READY	1-7
- Maximum cycle time in ms	1-7
- Active marker range	1-7
- Retentive marker range (also after cold start)	1-7
- Retentive marker range	1-8
- Forcing marker range in RUN	1-8
- Password	1-8
- Save versions of function blocks	1-9
- Create a utilisation table	1-9
- Version number for user program	1-9
Creating a device configuration	1-11
Writing a program	1-13
- Incorporating the configuration file	1-14
Compiling a program	1-15
- Backup copies	1-16
Transferring a program to the PLC	1-17

r

Programming: Procedure

General

This chapter provides you with information which you need for the generation of a program. Besides the input of the program code, you will find several preparatory and final tasks which are explained in the order they will be required.

The following description presumes two requirements:

- SUCOsoft S30-S 4-200 is installed:
see AWB 27-1185-GB, Chapter 1.
- A knowledge on the general operation and the user interface of the SUCOsoft S 30-S 4-200:
see AWB 27-1185-GB, Chapter 2.

/

Programming: Procedure

Setting system parameters

The user program contains information on the system configuration of the PS 4 200 series in the header of the .q42 source file. This data is converted by the compiler and thus transferred to the controller.

Settings of the following functions can be made or modified via the system parameters:

- program memory test
- start behaviour after NOT READY
- maximum cycle time
- active marker range
- retentive marker range (also after cold start)
- retentive marker range
- password
- version number of user program

Starting from the main menu, press the following keys in order to set the system parameters:

- [F1] PROGRAMMING
- [F3] SYSTEM PARAMETER EDITOR

You are then asked to state the name of your program file and the corresponding drive, since system parameters are parts of the user program. Enter or select the required name and drive to activate the following mask:

- [F2] SYSTEM PARAMETERS

Programming: Procedure

Setting system parameters

```

          S Y S T E M      P A R A M E T E R S :

Program check in RUN                <Ves=1,No=0>:                [ 0 ]
Start after NOT READY              CHalt=0.Cold=1,Warm=2):        [ 0 ]
Maximum cycle time in ns           <1...255>:                    [ 60 ]

Aktive marker range                up to MB <0...32767>:          4096 ]
Retentive marker range             up to MB <0...32767>:          ]
<also after cold start>           up to MB <0...32767>:          ]
Retentive marker range             up to MB <0...32767>:          ]
Force marker range in RUN from MB C0...32767):                  ]
Password                           up to MB <0...32767>          ]
Save versions of function blocks   <Ves=1,No=0>                 [ 0 ]
Create utilisation table           <Yes=1,No=0>                   [ 0 ]
Uersion number for user program    [ 0 ]

          Mark non-retentive ranges with -
- MAIN MENU—PROGRAMMING—>SYSTEM PARAMETER EDITOR
F 1 Return

                                          F10 Help
```

Figure 1-1: System parameters menu

The square brackets contain the default setting values. After you have entered all system parameters according to your requirements, exit the menu via [F1] Return. You can now save the set values.

Program test in RUN

The compiler builds a checksum which is saved in the compiled program at a defined location. If you select yes = 1, the operating system of the PS 4 200 series checks the user program during the run time with this checksum algorithm. The PLC is stopped if deviations are detected between the checksums. The error is entered in the diagnostics status word where it is indicated. The check is repeated cyclically. The default setting is No = 0.

Programming: Procedure

Setting system parameters

Start after NOT READY

This defines how the controller should behave after NOT READY. The default setting is Halt.

0 = Halt

1 = Cold start

2 = Warm start

Detailed description in
AWB 27-1184-GB, Chapter 4

Maximum cycle time in ms

The default setting is 60 ms. The cycle time can be max. 255 ms. This setting does not control the cycle time of a user program but only defines an upper limit for the malfunction check. Only set a shorter cycle time if you know the real processing time of the program. In this case, a longer processing time indicates an error.

The maximum cycle time to be selected depends on the type and size of the user program concerned. If the set cycle time is exceeded, the ETC bit is set in the diagnostics status word (DSW) and the controller switches to Halt.

Active marker range

The default setting is MB 0 to MB 4096. Set the marker range to suit the requirements of the markers used in the user program since all markers set require memory. If you use markers in the user program which have not been defined in the default setting, the compiler will output a corresponding error message.

Retentive marker range (also after cold start)

Set the marker range for data which is to be kept retentively also with a cold start. This marker range forms a part of the-selected active marker range and may not overlap with the retentive marker range (see next paragraph).

Due to the dynamic memory management in the SUCOsoft S 30-S 4-200 it is necessary to save this marker range if the device configuration is modified (adding or removing input/output elements). Write the retentive marker range on a flash EEPROM memory module via the SDAT function block before transferring the modified user program. After the modified program has been transferred to the controller, the saved marker

Programming: Procedure

Setting system parameters

Retentive marker range (also after cold start)

range must be reloaded from the flash EEPROM memory to the PLC memory via the RDAT function block. This is only necessary if the retentive marker range is used.

Retentive marker range

In the event of a voltage failure the retentive markers keep their previously defined states. They are also kept with a restart of the operating system. This marker range forms a part of the selected active marker range and may not overlap with the cold start-retentive range (see previous paragraph).

Forcing marker range in RUN

The defined markers can be dynamically forced in the IL status display while the controller is in RUN. This marker range forms a part of the selected active marker range.

Dynamic forcing enables a desired program process to be forced or particular actions to be initiated by defining special data values. It is also possible to organize 18 markers in one block in order to observe their states. You can find detailed information in Chapter 4, IL Instruction List.

Password

The entry of a password prevents unauthorised access.

The default setting is "No password". The password can have up to eight characters. It is connected with the user program and is incorporated during the compiler run. A password that is already saved can be overwritten by a new one.

A password is scanned when data or the PLC status is to be modified. With the following functions the password is scanned if the controller contains a program which is password protected:

- Start
- Stop
- Diagnostic counter reset
- Diagnostic status word reset
- Retentive marker reset

Programming: Procedure

Setting system parameters

- Force setting
- Online programming
- Compare PS \longleftrightarrow Drive
- Transfer PS \longleftrightarrow Drive
- Transfer Drive \longleftrightarrow PS
- Set date in PS

The following functions can be executed without stating a password since the data involved is only read and not modified:

- Status indication
- Display range
- Device status
- I/Q indication

If no password or the wrong one is entered, the function is not executed and the error message "Incorrect password! Function cannot be executed" is output.

If you cannot remember your password, you can find it in your backup copy in the source file (name).q42 in the System parameter entry menu. See also section Compile programs/backup copies.

Whoever is able to access the source file with the system parameters, can execute all password protected functions.

Save versions of function blocks

The version of the used function blocks is saved with Yes = 1. This considerably facilitates a possible troubleshooting since different versions of function blocks can exist for the same function. The default setting is "0 = Not save".

Create a utilisation table

This allows you to save in the utilisation table the corresponding physical addresses for the used logic addresses (marker, etc.) in the PLC memory. See also AWB 27-1185-GB, Chapter 3.

Version number for user program

The default setting is 0. Use this field to identify specific program versions.

Programming: Procedure

Creating a device configuration

In order to create the device configuration, proceed as follows starting from the main menu:

[F1] Programming
[F4] Device configuration
Enter a name for the configuration file or select a file from the existing files
[F2] Configure

```
PS4-201-MM1    116-DX1    116-DX1    116-DXi

PS3-DC

EM4-201-DX2

EM4-201-DX1

-[EM4-201-DX1

RMQ-16I

- MAIN MENU -->PROGEMING ->DEUCE CONFIGURATION                c:e4000am.k42
F 1 Return          F 4 Replace module
F 2 Add station     F 5 Zoon/Normal
F 3 Add nodule      F & Parameter editor    F 8 Delete
                    F10 Help
```

Figure 1-2: Device configuration menu

This menu is used for creating the device configuration.

Use "Add station" to expand the configuration vertically. Use "Add module" to expand the configuration horizontally. After pressing one of these a selection box appears containing the stations/modules to be selected. Press [F3] SAVE PROGRAMS to save the file (name).k42.

The (name).k42 configuration file must **always** be incorporated at the beginning of the user program, also if the PS4-201-MM1 is used on its own.

You can find detailed information on the device configuration in AWB 27-1185-GB, Chapter 2.

Programming: Procedure

Writing a program

The user program can also be considered as a job specification for the programmable controller which contains a complete description of all control sequences.

Proceed as follows to create the user program, starting from the main menu:

- [F1] Programming
- [F2] Programming IL
- Entry of the source and reference file via selection boxes
- [F2] Edit program file

```
00000 CONFIG      ""
001              ttinclu.de "e4000an.k42"
002
00001 INPUT       "reading input s
001              L IB1.2.0.0
002              = MB32
003
00002 STRT        "Start of DEMO-Program
001              L K 1
002              S M 10.0          STRT_SK0
003              S n 18.0         Master-Mode
004
005              L K 1
006              AN M 11.1
007              = M 11.2
008              L K 1
009              - II 11.1
010              L M 11.2
011              JCN PROGRAM
012              L KHB 1
- MRIN MENU-->PRO
F 1 Return          F 4 Add line          F 7 Delete blocks insert
F 2 Open block     F 5 Find / Replace      8 Delete current line
F 3 Select block   F 6 Copy blocks         F 9 Delete range
```

Figure 1-3: Program editor menu

Programming: Procedure

Writing a program

The following overview shows the topics which you should know before entering the program code:

A knowledge of programming	Described in
IL editor	AWB27-1185-GB, Chapter 2
Programming: elements and rules	AWB27-1186-GB, Chapter 2
Structuring programs	AWB27-1186-GB, Chapter 3
IL instructions	AWB27-1186-GB, Chapter 5
Function blocks	AWB27-1186-GB, Chapter 6
The step sequencer function block (for advanced users)	AWB 27-1186-GB, Chapter 7
Indirect addressing (for advanced users)	AWB27-1186-GB, Chapter 8

Incorporating the configuration file

Incorporating the configuration file is necessary for the compiler run. Since this configuration contains information on the type of expansion modules, slave controllers, etc. and on which locations they are used, the compiler can thus check whether the addressing and other specifications are correct. The correct syntax for this instruction is the following:

```
•include "configuration file.k42"
```

This instruction must **always** be the first one in the program. This also applies if the PS 4-201-MM 1 is used on its own, otherwise an error message is output when compiling the user program.

Programming: Procedure

Compiling a program

The IL instructions must be compiled in order to obtain an executable program.

Note!

Please read the section Backup copies on the next page before you start the compiler run.

Select from the Programming menu [F5] Compiler in order to compile the program. Specify the source and reference file required via the two selection boxes displayed in succession. You are then asked whether Include files are always read in by the same drive. If you answer with Yes, you will be asked for the standard drive (A, B, C,...).

If you enter No, you are asked to state an appropriate drive for each Include instruction found during the compiler run. Answering with No would thus only be useful if the corresponding Include files are saved on different drives.

Select the drive via F1 in the selection boxes. The compiler run then starts. If the run is executed without errors, the executable program can be transferred into the controller, otherwise the errors which are listed according to block and line numbers must be rectified. An executable program is only generated if all errors have been rectified.

Programming: Procedure

Compiling a program

Backup copies

After the user program has been compiled, it cannot be discompiled into the original program code. We therefore advise you to make backups and/or create documentation of the following files:

- .q42 source file
- .z42 reference file
- .k42 configuration file

This measure is also recommended if you cannot remember your password anymore. You can find it in your backup copy of the source file (name).q42 in the System parameter entry menu.

Programming: Procedure

Transferring a program to the PLC

Select [F2] Test/Commissioning in the Programming menu and [F6] Transfer drive → PS in order to transfer the program to the controller. After you have selected the file (name.p42) to be transferred via the selection box, the file is transferred to the controller.

If there is already a user program in the PLC, it is overwritten by the new program. If the previous program is protected with a password, this password must be entered before the program can be overwritten.

After the transfer has been carried out successfully, a message is output which informs you on how many bytes (size of the file (name).p42) have been transferred and on the size of the remaining memory in the PLC.

You can find further information on other transfer operations, on Test/Commissioning functions and the online modification in Chapter 4.

2 Programming: Elements and Rules

Contents

Elements of an instruction	2-3
- Addressing the operands	2-5
- Digital inputs	2-7
- Analogue inputs	2-7
- Counter input	2-7
- Outputs	2-8
- Digital output	2-8
- Analogue output	2-9
- Markers	2-9
- Constants	2-11
- Real-time clock	2-11
- Function block parameters	2-13
- System specific operands	2-13
- Peripheral operands	2-14
- Symbolic operands	2-15
- Negation of operands	2-18
- Operations	2-18
Function blocks	2-21
- Overview	2-23
- Organisation and location of the function blocks	2-24
- Number of function blocks	2-26
- Call-up of the function blocks	2-26
- Behaviour of the function block inputs	2-27
- Incorporation into the user program	2-28
- Retentive function blocks	2-31
Registers	2-35
- Working register	2-37
- Auxiliary register	2-37
- Status register	2-37
- Stack register	2-39
Handling intermediate results	2-41

Programming: Elements and Rules

IL syntax rules	2-45
- Instruction line	2-45
- Sequence	2-45
- Block	2-48
- Main program	2-49
Pre-processor instructions	2-53
- Incorporating the configuration file	2-54
- Inserting files	2-54
- Combining files	2-55
- Control of documentation	2-55

Programming: Elements and Rules

Elements of an Instruction

The user program is a set of instructions for the programmable controller which describes the entire control process. An instruction is the smallest self-contained unit of a program. An instruction can be written in one line and contains a job for the controller which cannot be divided into further units. The AND sequence and the Addition function are typical instructions.

An instruction consists of an operator and an operand in accordance with DIN 19239 (IEC 65 A).

The operator specifies the function to be executed. It instructs the processor how to process the operand in question.

The operand consists of operand identifiers and parameters, and may be extended if necessary. The operand identifier specifies the type of operand involved whilst the parameters specify exactly which parameter section of the operand is to be selected. For this purpose, the location of the operand is related to the network (PS 4 200 series, EM 4, LE 4) and the word/byte address and the bit number are stated.

The SUCOsoft S 30 programming language recognizes the following operand types:

- Inputs - I
- Outputs - Q
- Markers - M
- Constants - K
- System-specific operands
- Symbolic operands

Programming: Elements and Rules

Elements of an Instruction

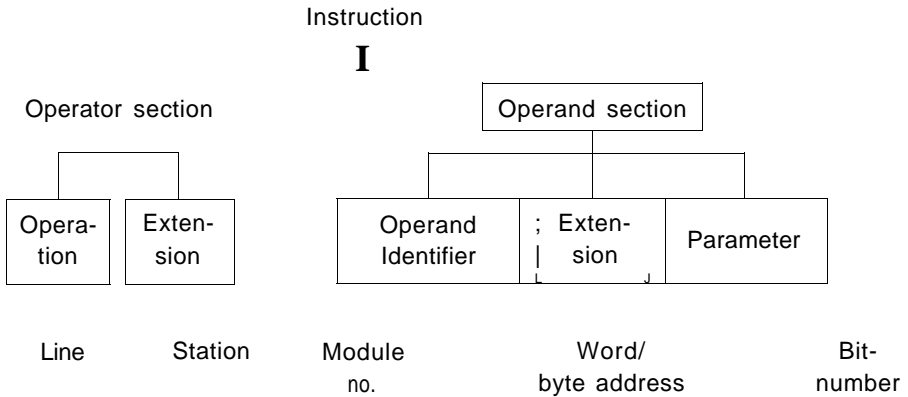


Figure 2-1: Structure of an instruction

The following table shows all operands which can be used with the instructions. Ensure that the data type (bit, byte, word) stated in each instruction is the same as the data type of the operands.

Table 2-1: Operand overview

Designation	Bit	Byte	Word
Inputs ¹⁾	I	IB, IAB, ICB	IW, IAW, ICW
Outputs	Q	QB, QAB	QW, QAW
Markers	M	MB	MW
Constants ¹⁾	K	KB, KHB	KW, KHW
Real-time clock ¹⁾	-	CKxx	-
Peripheral access	IP ¹⁾ , QP	IPB ¹⁾ , QPB	-
Status/diagnosis	IS ¹⁾	ISB ¹⁾	ISW ¹⁾
Communications data	-	RDB, SDB	-
Information	INB x.y ¹⁾	-	-

¹⁾These operands cannot be used for the following operations:

- Allocation (=)
- Reset (R)
- Set (S)

Programming: Elements and Rules

Elements of an Instruction

Addressing the operands

The following example shows the required logical syntax of the PS 4 200 series for the unique addressing of the operands:

I x.x.x.x.x

- Bit number (0 ...7)
- Byte number (0 ... y)
- Module (0 ... 6)
- Station (0 ... 8)
- Line number (0 ... 3)

y depends on the type of station/module concerned

The correct syntax for the seventh digital input bit in module 1 (LE) which belongs to the slave 1 (EM) and which is assigned to line 1 is the following:

11.1.1.0.7

If the inputs (I 0.0 -1 0.7) or the outputs (Q 0.0 - Q 0.5) are addressed in the PS 4-201-MM1 basic unit, the first three digits are not necessary. If they are entered by the user, they are removed automatically when the line is completed. The inputs/outputs of all expansion modules **must** be addressed via the five-digit address syntax.

The same applies to the addressing of the markers. If the markers are addressed in the basic unit, the addressing is identical with that of the other basic unit operands. Furthermore, parallel bus markers (LE bus markers) can be used in the horizontal (i.e. local) level. These are:

M O.O.Lx.y. - M 0.0.6.x.y. (bit, byte, word)

The access to these markers requires more cycle time than the access to the markers of the basic unit, since the parallel bus must be opened for each access.

All other stations on the line do **not** have markers.

Programming: Elements and Rules

Elements of an Instruction

Addressing the operands

SUCOsoft S 30-S 4-200 also offers the possibility of indirect addressing. You will find a detailed description in Chapter 8, Indirect addressing.

Inputs

The **Inputs** constitute the interface between the external environment of the programmable controller and the programmable controller itself. External signals reach the PLC via the inputs and are processed further.

Bit inputs are specified by the appropriate byte number and the bit number within the byte concerned. These two numbers are separated by a full stop.

Byte inputs are identified by the appropriate byte number and the letter **B**.

Word inputs do not require the bit number and the full stop. They are always even numbers. The letter **W** is required as an extension.

Programming: Elements and Rules

Elements of an Instruction

Digital inputs

IW 0.0.1.0

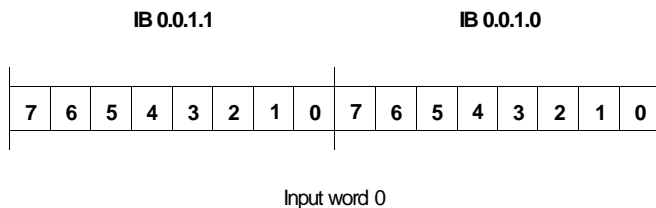


Figure 2-2: Input addresses

If the PS 4 200 series operates on its own, 8 digital bit inputs are available. The number of digital inputs can be increased by 6 x 16 bits by using LE4 modules.

The digital input words are **always** addressed via the image register.

Analogue inputs

The 2 standard analogue inputs of the PS 4 200 series are **not** optocoupled. Only analogue values from 0 to 10 V d.c. can be scanned and handled **with** a 10-bit resolution. The program addresses the inputs either as absolute or symbolic operands.

This addressing applies also for the two setpoint potentiometers of the PS4-201-MM1 which can be considered as two more analogue inputs.

Read the analogue inputs:

IAW 0	Setpoint potentiometer
IAW 2	Setpoint potentiometer
IAW 4	Terminal
IAW 6	Terminal

Counter input

The PS 4 200 series provides a high-speed counter as a standard feature which is accessed via the alarm function blocks.

Programming: Elements and Rules

Elements of an Instruction

Analogue output

The PS 4200 series provides one analogue output (0-10 V d.c.) as a standard feature with a 12-bit resolution (0...4095). The analogue output is either addressed as an absolute or symbolic operand. The analogue output cannot be read.

Example:

The value 4000 is to be output on QAW0.

L KW 4000

= QAW0

Markers

M markers are used to store intermediate results produced during the data processing operations of the PLC.

The number of the used markers (Bit, Byte, Word) is only limited by the memory range provided, i.e. the system parameters entered by the user.

Bit markers are defined with the byte number and the corresponding bit number, separated by a full stop.

Byte markers contain the byte number and also the letter **B**.

Word markers do not have a bit number or a full stop. They are always even. The letter **W** must be added.

MW0

MB1

MB0

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

M_____Marker word 0

Figure 2-4: Marker addresses

Programming: Elements and Rules

Elements of an Instruction

Markers

Markers can always be read back and their retentive behaviour in the event of a power failure can be set as required. Two marker ranges which retain their data in the event of a power failure can be selected for programming. One marker range is freely available. The other one is reserved for the cold start retentive data. The markers located outside of this range are always reset to "0" when the PLC is powered up after a cold start. These retentive ranges can be set in the System parameters menu of SUCOsoft S30 and only apply to the user program concerned.

Parallel bus markers

The user can access the parallel bus of the PS 4 200 series only in the horizontal string 0. Each of the stations LE1 to LE6 (0.0.1 - 0.0.6) is connected to the basic unit via the parallel bus and has a parallel bus page of 256 bytes each. The operand syntax is for example:

M 0.0.1.x.y with x = byte address and
y = bit address

The general operand syntax is the following:

<Marker operand>xData type>0.0.module.byte.bit

with data type = bit, byte, word

The access to the parallel bus markers always requires more cycle time than the access to the markers of the basic unit, since the parallel bus is normally closed and must be opened for each access.

Programming: Elements and Rules

Elements of an Instruction

Constants

Fixed starting and reference values can be entered by means of the **K constants**. Depending on the data type selected, the constant values are available in the following ranges:

Bit: K 0 and K 1

Byte: KB-128...KB 0...KB 127

Word: KW-32768...KW 0...KW 32767

In addition to the above representation, the programming language also accepts constants written in the "KH" hexadecimal form. Hexadecimal constants are possible within the following ranges:

Byte: KHB 0...KHB FF

Word: KHW 0...KHW FFFF

The plus/minus sign is not stated separately for hexadecimal constants. It is already contained in the most significant digit of the hexadecimal value.

The constants KHW FFFF thus corresponds to the decimal constant KW - 1 .

The constant values are fixed during programming and cannot be altered while the program is running.

Real-time clock

The CPU of the PS 4 200 series is provided with a real-time clock, thus enabling date and time dependent programming. The clock can be set via the SUCOsoft S30-S 4-200 software (Test and Commissioning menu) if the PC and PLC are connected.

The current time and date values are entered (seconds resolution) by the system routine into a specially reserved data range which is organised in byte format. This data can be accessed in byte format by the SUCOsoft S30-S 4-200 software using the Load instruction. The following syntax is permissible:

Programming: Elements and Rules

Elements of an Instruction

L	CKSS	Seconds	0-59
L	CKMM	Minutes	0-59
L	CKHH	Hours	0-23
L	CKYR	Year	0-99
L	CKMT	Month	1-12
L	CKDY	Day	1-31
L	CKWD	Weekday	0-6
			Sunday-Saturday

These values are made available by the PS 4 200 series in decimal form so that direct comparisons can be made in the IL program with constants specified by the user.

These operations enable cyclical alarms to be preset as required.

The real-time clock operates with an accuracy of ± 10 ppm, which corresponds to a maximum deviation of + 10 seconds in 11.5 days.

Example

A one-minute alarm is to be output every day at 1600 hrs.

ALARM	LK0 = M x.y LCKHH CPKB16 BNE CONTINUE LCKMM CPKBO BNE CONTINUE LK1 = M x.y	If it is not 1600 hrs, branch to CONTINUE. The current minute is compared with 0. This sequence is run for one minute at 1600 hrs.
CONTINUE		

Programming: Elements and Rules

Elements of an Instruction

Function block parameters

In addition to the settings carried out directly after the module is called up, the **function block parameters** can be set at any point in the user program. In this case, the function block name and number as well as the parameter concerned are used together to form an operand, e.g.

L C 25 Q Output Q of counter module 25
= CP 6 11 Input 11 of the comparator 6

With this kind of function block setting the function block inputs can be seen from the point of view of the user program as outputs, i.e. they can be read back, whilst function block outputs are treated as inputs. Function block outputs can only be read via the user program and cannot be written.

System specific operands

The PS 4 200 series has system-specific operands for monitoring internal faults via the program or for exchanging data between the operating system of the programmable controller and the user program. These operands are described in the following sections.

Peripheral operands IP, QP

Peripheral operands can be used to access the inputs/ outputs in the basic unit of the PS 4 200 series irrespective of the cycle time and the image register. The use of these operands in control sequences is analogue to all other operands. Alarm processing, for example, is a typical application for these operands.

The peripheral operands can be accessed in bit and byte format. When the QPBO is accessed, remember that the outputs QP0.6 and 0.7 cannot be accessed since they do not exist in the basic unit, they are set to zero when the entire byte is accessed.

Programming: Elements and Rules

Elements of an Instruction

Peripheral operands IP, QP

Example:
L IP 0.0
A IP 0.5
= QP0.3

The following applies for the inputs, since the access is only possible in the basic unit:

IP 0.0 ...0.7
IPB0

for the outputs:

QP 0.0 ...0.5
QPB0

Note!

The output image changes when using peripheral operands.

Status/diagnostics inputs IS

These inputs contain the information on the status or allow a diagnosis of the connected stations. All local expansion and external modules etc. are connected stations. All operations are possible with this data, taking into account the permitted data types.

You can find a detailed description of the diagnostics status word (status/diagnostics inputs IS) of the basic unit PS 4-201-MM1 in Chapter 4, Diagnostics status word. Local expansion and external modules are explained in the relevant hardware description.

Programming: Elements and Rules

Elements of an Instruction

Communications data RD/SD

This data () is exchanged among **active** SUCOnet stations or between the PS 4 200 series basic unit and the **active** stations via the SUCOnet K field bus. (**SD = SEND DATA, RD = RECEIVE DATA**). All operations are possible with this data, taking into account the permitted data types. You can find further information on this data and a detailed description of the communications data in AWB27-1184-GB, Chapter 6.

Information data INB

The information data contains information on the status of the controller. They can only be accessed in bit format and enable a reaction depending on the status of the controller.

You can find detailed information on the meaning of the individual bits of the information byte INB in Chapter 4, Commissioning.

Symbolic operands

All the operands (I, Q, M etc.) listed in this chapter can be addressed in the program via their respective operand identifier and parameter or via symbolic operands.

A symbolic operand consists of up to eight characters which the user can select as required.

Symbolic operands are listed in the reference file and are assigned with the appropriate operand code. Additional information such as behaviour (M/B), terminal designation and operand comments can also be listed in the reference list.

Programming: Elements and Rules

Elements of an Instruction

Symbolic operands

Example of a reference list:

Symbol	Operand	MB	Terminal	Operand comment
S0	I 0.0	M	1x0	Limit switch, deflector in "feed position"
S1	I 0.1	M	1x1	Limit switch, deflector in "deflect position"
S3	I 0.2	M	1x3	Select data input, terminal/selector switch
LS4	I 0.3	M	1x4	Light barrier "Incoming packets"
LS5	I 0.4	M	1x5	Light barrier "Outgoing packets"
S6	I 0.5	M	1x6	Reset packet counter
S7	I 0.6	M	1x7	Acknowledge fault, klaxon off
S8	IB 1.1.2.0	set	1x40...47	Select "Max. no. parcels in build-up zone"
Y0	Q0.0		2x0	Bring deflector into feed position (H2)
Y1	Q0.1		2x1	Bring deflector in deflector position (H3)
H2	Q0.2		2x2	Y0 -> end pos. 2 Hz flash; end pos. cont. light
H3	a 0.3		2x3	Y1 -> end pos. 2 Hz flash; end pos. cont. light
H7	0.0.5		2x7	Klaxon 5 sec. interval "Build-up zone full"
H8	QW 1.1.4.0		2x40...56	Digital indication ... Parcels in build-up zone

A symbolic operand is always preceded in the program by a ' character (single quotation mark) so that it can be identified clearly.

When symbolic operands are used in the user program, the operand comments stored in the reference file are automatically transferred.

Programming: Elements and Rules

Elements of an Instruction

Example: Example of program structure with symbolic operands

```
00001  FLASHING  " Flash generator movement indication
          TGEN0      Flash generator, 2 Hz
          [] S: L 'Y0  Bring deflector into feed position (H2)
              0 'Y1  Bring deflector into deflect position (H3)
          [W] I: KW 500 Constant 500 (ms)
          [] P:
```

00002 SIGNAL *

In the following signals, the indicator addressed will flash at 2 Hz as long as the positioning control is moving, but has not yet reached the end position. When the appropriate end position is reached, the signal changes to continuous light.

```
00003  SIGNAL1  " Indication: Travel to feed position

          LN M 10.0      Auxiliary marker, build-up indication
          A TGEN 0 P      Flash generator, 2 Hz
          0 'S0           Limit switch, deflector, in "feed position"
          = 'H2           Y0 -> end pos. 2 Hz flash; end pos. cont. light
```

```
00004  SIGNAL2  " Indication: Travel to deflect position

          LN M 10.0      Auxiliary marker, build-up indication
          A TGEN 0 P      Flash generator, 2 Hz
          0 'S1           Limit switch, deflector, in "deflect position"
          = 'H3           Y0 —> end pos. 2 Hz flash; end pos. cont. light
```

The possible relationships between symbols and operands:

Operation	Symbol	Operand	Programming
Inputs	S0	I 0.0	L'S0
Markers	HM2	M2.0	L 'HM2; = 'HM2
Outputs	Y0	Q0.0	= 'Y0; S 'Y0

Programming: Elements and Rules

Elements of an Instruction

Negation of operands

Apart from the function block parameters, the operands mentioned in this chapter can be **negated** as required. The character N is placed for this purpose in front of the operand to be negated. This causes the operand value to be negated when it goes to the instruction. The following instruction is an example of this:

L N Q0.3,

This instruction means that the inverted value of the 4th bit is loaded into the working register without changing the actual output Q0.3.

Operations

The operators available can perform Boolean and arithmetic instructions, comparator, shift, rotate and transfer operations. A number of operators are also available for organising programs.

All the operators are listed in the table below together with their respective data types. You can find detailed information on the meaning of the individual operators in Chapter 5, IL instructions.

Table 2-2: Available operators with their respective data types

Operations	Bit		ByteAVord	
		negated		negated
Boolean:				
AND sequence	A	AN	A	AN
OR sequence	O	ON	O	ON
EXCLUSIVE OR	XO	XON	XO	XON
NEGATION	NOT		NOT	
Arithmetic				
Addition			ADD	
Subtraction			SUB	
Multiplication			MUL	
Division			DIV	
Comparison				
Comparison			CP	

Programming: Elements and Rules

Elements of an Instruction

Operations	Bit		Byte/Word	
		negation		negation
Word modification Shift*— with carry Shift-* with carry Rotate <- Rotate ->			SHL SHbC SHR SHRC ROTL ROTR	
Transfer Load operand in working register or stack, set sequence result operand, Set operand Reset operand Load auxiliary register	L = S R	LN =N	L = GOR	LN =N
Program organisation Jumps to block label: absolute jump conditional jump Return from a subprogram to the calling program from the main program to the operating system absolute conditional Conditional branches to block labels dependent: on bit on carry on plus/minus sign on zero on overflow Compare on greater than on equal or greater than on equal on less than on equal or less than Others Zero operation End of module End of program	JP JC RET RETC NOP EM EP	JCN RETCN	JP RET BB BC BP BZ BV BGT BGE BE BLT BLE NOP EM EP	BNB BNC BM BNZ BNV BNE

Programming: Elements and Rules

Elements of an Instruction

Operations	Bit		Byte/Word	
		negated		negated
Program organisation:				
Program module call-up:				
unconditional	CM		CM	
conditional	CMC	CMCN		
Conditional branches to program module dependent:				
on bit in working register			CMB	CMNB
on carry			CMCY	CMNC
on plus sign +			CMP	
on minus sign -			CMM	
on zero			CMZ	CMNZ
on overflow			CMV	CMNV
on greater than >			CMGT	
on equal =			CME	CMNE
on less than <			CMLT	
on greater equal >=			CMGE	
on less equal <=			CMLE	

Programming: Elements and Rules

Function Blocks

Function blocks

Function blocks are part of the system program of the CPU and are supplied with data from the user program which also activates them. The function blocks enable complex functions to be executed which accept data from the user program according to specified rules and then return the results to the user program after completing certain specified functions. The operating system of the programmable controller performs the actual evaluation of the data itself.

These modules thus eliminate the need for lengthy instruction sequences, and increase the amount of user memory available. They also save the programmer any time consuming program testing required for the function concerned. An example of this is the comparator, which compares the contents of the two input words and sends the result to its function block outputs for further processing.

— W 11 GT — = 1, if Word 1 > Word 2
CP10
EQ — = 1, if Word 1 = Word 2

— W 12 LT — = 1, if Word 1 < Word 2
|

Figure 2-5: Example of a function block

The SUCOsoft S30 programming software supports the integration of the function blocks in the user program. It knows the function and designation of all function block inputs and outputs, enabling the function blocks to be called up and assigned parameters from within the user program itself. This data enables the programming system to display on screen the basic elements of the function blocks to be programmed. The user can then include connections from the function block into the program.

Programming: Elements and Rules

Function Blocks

Function blocks

The used function blocks are compiled together with the user program and transferred to the controller.

Individual instructions or function blocks are used as required in the application in hand.

The function block provides the user with a ready-made and well-tested solution which takes all programming aspects into consideration, monitors all possible errors, and provides program driven user-friendly handling.

The central unit requires more time to run a function block than to run an individual instruction.

Programming: Elements and Rules

Function Blocks

Overview

The function blocks mentioned in Table 2-3 are described in detail in chapter 6.

Table 2-3: Available function blocks

Group	F block	Designation	Data type	Retentive
Arithmetic (Fixed-point)	Comparison	CP	Word	-
Counters	Up/down counter	C	Word	yes
Timers	On-delayed	TR	-	yes
	Off-delayed	TF	-	yes
	Pulse transmitter	TP	-	yes
	Generator	TGEN	-	-
Real-time clock	Time/date comparator	CK	-	-
	Set real-time clock	SCK	-	-
Registers (cascadable)	Shift register	SR	Bit	yes
		SRB	Byte	yes
		SRW	Word	yes
	Stack register (Last in First out)	LIFOB	Byte	yes
		LIFOW	Word	yes
	Stack register (First in First out)	FIFOB	Byte	yes
		FIFOW	Word	yes
	Sequential control module	SK	—	yes
Alarm function blocks	High-speed counter	CALARM	-	-
	Edge counter	FALARM	-	-
	Timer	TALARM	-	-
Code converters	binary -> BCD	BID	Word	-
	BCD -> binary	DEB	Word	-
Block transfer	Indirect copy	ICPY	Byte	yes
Block compare	Comparator	ICP	Byte	yes
Working with data in cold start retentive range	Save	SDAT	-	-
	Reload	RDAT	-	-

Programming: Elements and Rules

Function Blocks

Organisation and location of the function blocks

The function blocks must be supplied with input data before being called up. After the call-up, the function block processes the input data and generates the result as output data.

The function block input data is entered via the user program and transferred to an input data field. The user program then takes the output data from the output data field of the function block.

For every function block used in the program the compiler stores these data fields in the data memory (see manual on central unit). The function block inputs and outputs are thus available in the user program in the same way as any other operands.

In the following example, it is therefore possible to use the result output "Q" of the binary/BCD converter 12 in conjunction with the function block.

It can also be used as an operand at any point in the program

BID 12 Q

and then incorporated into an instruction.

Example:

LBID12Q

= MW2

The contents of the data fields are retained until they are overwritten when the same function block is processed once more.

Programming: Elements and Rules

Function Blocks

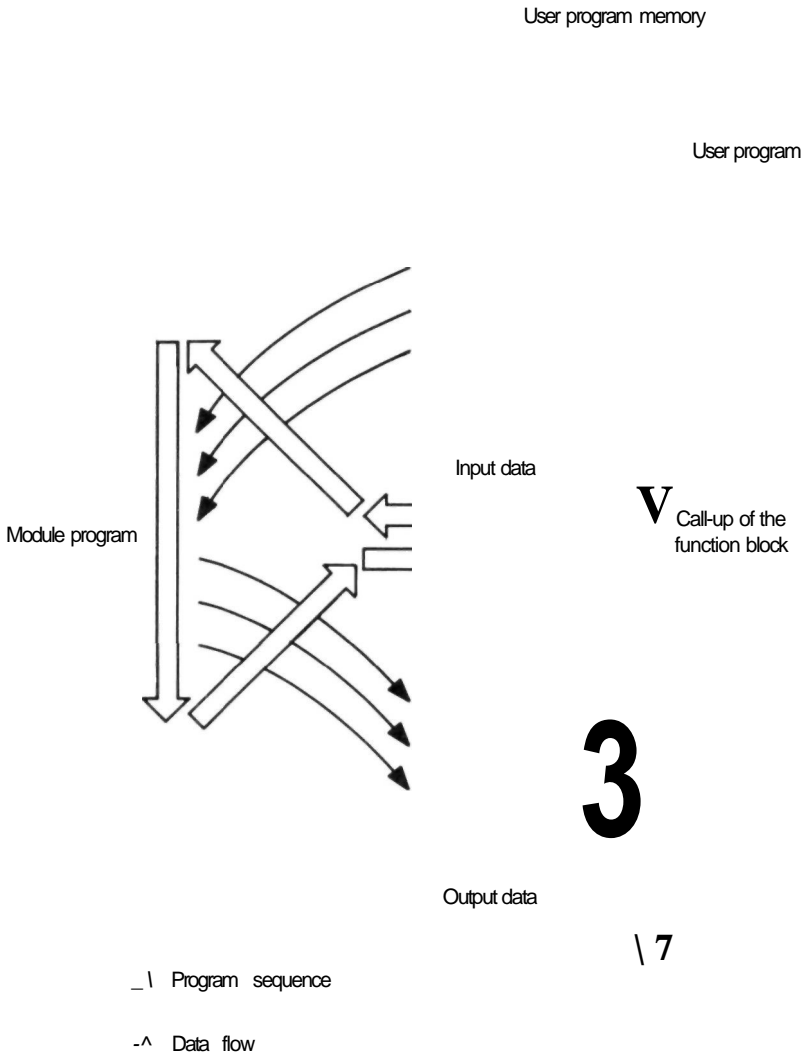


Figure 2-6: Program sequence and data flow during function block processing

Programming: Elements and Rules

Function Blocks

Organisation and location of the function blocks

Figure 2-6 shows the interaction between the user program and the function block. The function block must be called so that it can calculate the results. It is not enough to allocate data to the function block as operands via an instruction and to fetch the results in the same way (see also "Incorporation into the user program").

It should be noted that it is not necessary to reserve generally accessible marker ranges since function blocks have their own data ranges.

Number of function blocks

The number of the function blocks to be used is not restricted. A restriction is only given by the capacity of the user memory. Theoretically, the upper limit of function blocks is 65 535.

Call-up of the function blocks

The function blocks are called up using the appropriate reference code, the function block number and any supplementary settings are required (see Table 2-3 page 9). All additional settings are preceded by a hyphen "-".

The following additional settings are also possible.

-R	Retentive	The function block is incorporated in the battery back-up.
-MS	Millisecond	In timers, the basic time selected, millisecond (in 10 ms clock pulse).
-S	Second	In timers, the basic time selected second.
1...128	Register length	In registers, the selected register length (step length).

Programming: Elements and Rules

Function Blocks

Example:

C 13-R	Call-up of retentive up/down counter 13
SR 14-121 -R	Call-up of retentive bit shift register 14 and setting of register length to 121 steps.
TF 14-S	Call-up of off-delayed timer 14 and selection of the basic time, seconds.

Function blocks can be made retentive (zero-voltage proof) by the -R suffix.

The register lengths of register function blocks can be selected.

The timers can be set to the clock rates 10 ms and 1 s.

The inputs of the function block are either static or dynamic.

With static inputs, the function allocated to the input concerned is implemented when the input recognizes a logic High. All inputs have this feature except for the clock and set inputs.

The clock and set inputs behave dynamically. They must recognize a change from logic 0 to logic 1 (rising edge) before they can carry out their function. In order to be able to form these signals independently, the function block must be able to recognize a Low signal following a High signal. Particular attention should be paid to this when program branching is involved.

It is not necessary to use all function block inputs. Unoccupied inputs are passive; they behave as if the input recognizes a permanent Low signal.

Behaviour of the function block inputs

Programming: Elements and Rules

Function Blocks

Incorporation into the user program

The information given in the call-up instruction specifies the type of function block concerned as well as other features. This causes SUCOsoft S 30-S4-200 to display the function block with its inputs and outputs.

The function block inputs and outputs are shown with appropriate designations, including one of the following data type designations.

	Bit	1 bit
B	Byte	8 bit
W	Word	16 bit

Below is an example of a counter which has been called up.

```
0 M 1053
= Q0.7
C14R
[ ]U:
[ ]D:
[ ]S:
[ ]R:
PA I:
[ ]Z:
[M]Q:
```

In this example, the user has entered the instruction lines up to and including the function block call-up "C 14-R". The lines following this then appear automatically.

After the function block is shown on screen, the programmer must enter the necessary parameters for the inputs and outputs. The relevant operands or operand sequences are thus entered, which either supply the required data or receive it.

Programming: Elements and Rules

Function Blocks

```
0   M 105.3
=   Q0.3
C   1 4-R
[ ] U:      10.10      Up pulse
[ ] D:      I 0.2      Down pulse
[ ] S:      I 0.3      Set input
[ ] R:      LI 0.4     Reset input
      0 N M 24.6     Reset condition
[W] I:      KW256     Counter setting value
[ ] Z:      Q0.5     Zero indication
[W] O:      MW20     Counter status

L   MW22     Counter status
ADD MW21     Intermediate sum
```

The example above shows the function block after it has been incorporated into the user program.

When the program is run, the contents of the operands entered on the input side are copied to the appropriate data memory. The program then jumps to the appropriate function block program in the system section of the program memory. This program processes the input data and writes the result to the output data range of the function block. The user program can then fetch the results from here and process them.

In the example shown, the inputs and outputs are added directly next to the function block. As explained in the section "Organisation and location of the function block", the function block inputs and outputs can be incorporated into instructions in the same way as other operands.

Programming: Elements and Rules

Function Blocks

Function blocks, incorporation into the user program

The previous example could be modified whilst maintaining the same function. See below.

LKW256
= C 141

LKW557
= C 14 I

-- o <-

C14-R

t]U:	10.1	Up pulse
[]D:	I 0.2	Down pulse
[]S:	I 0.3	Set input
t]R:	LI 0.4	Reset input
	0 N M 24.6	Reset condition
DM I:		
[]Z:	Q0.3	Zero indication
[W]Q:	MW20	Counter status
L	MW20	Counter status
ADD	MW 22	Intermediate sum

The program now contains a counter with an external data supply. The I input is not assigned a parameter in the function block itself but rather in the program lines before it.

This procedure enables a central function block to be supplied with different data from a number of program sections located before the function block itself.

Programming: Elements and Rules

Function Blocks

Retentive function blocks

A memory module (RAM) which also has a flash EEPROM memory can be installed in the PS 4 200 in addition to the user program memory (RAM).

The compiled user program is stored in the **user program memory**. Areas are reserved in the data memory for any function blocks contained in the user program. The current data of the function blocks concerned (e.g. counter status values, register contents etc.) is stored in these data ranges.

The **data memory** is managed dynamically. This means that contents of the assigned function blocks are not stored in permanently specified data ranges, but rather in the order in which they were programmed (see Figure 2-7). The data range is dynamically managed by the compiler.

When the PS 4 200 series is switched on, the data fields of the non-retentive function blocks are cleared, whilst the contents of the retentive function blocks are kept.

Note!

As function blocks are added later with program modifications the data ranges of subsequent function blocks are shifted further (see Figure 2-8).

This can cause retentive data to be incorrectly assigned to function blocks and so it is advisable to use the following procedure when modifying programs:

- 1) Add new function blocks to the end of the program if the retentive values must be kept after the modification.
- 2) Function blocks can be inserted if the retentive data is no longer required after the modification to the program. The retentive data must then be deleted.

Programming: Elements and Rules

Function Blocks

Retentive function blocks

This can be done by switching off the PS 4 200 and then switching it back on again with the toggle switch in position 3 or via the programming device in the "Status menu" (Test and commissioning sub-menu) by pressing the "Delete retentive range" key F7.

	User program memory IL program	Data memory	System memory Function blocks
Counter CO	CO	CO data	
Shift register	SRB1-10	SRB1-10 data	
FIFO register (retentive)	FIF020-15-R	FIF020-15-R data	
Shift register	SRB1-20	SRB1-20 data	

Figure 2-7: Function blocks, types of memories and function block data storage

Programming: Elements and Rules

Function Blocks

	User program memory IS-Program	Data memory	System memory Function blocks
Counter CO	CO	CO data	
Shift register	SRB1-10	SRB1-10 data	
LIFO register	LIF05-30	UF05-30 with data from FIFO20-15-R	
FIFO register (retentive)	FIF020-15-R	FIF020-15-R data	
Shift register	SRB1-20	SRB1-20 data	

Figure 2-8: Function blocks, shifting of data ranges after inserting new function block data

Programming: Elements and Rules

Registers

All operands, whether negated or not, can be handled in a number of ways with the instructions described in Table 2-2. For this purpose the SUCOcontrol PS 4 200 series provide the user with several freely available registers, via which all sequences must run and in which values can be stored temporarily. These are the working register, the auxiliary register, the status register and the stack registers.

Programming: Elements and Rules

Registers

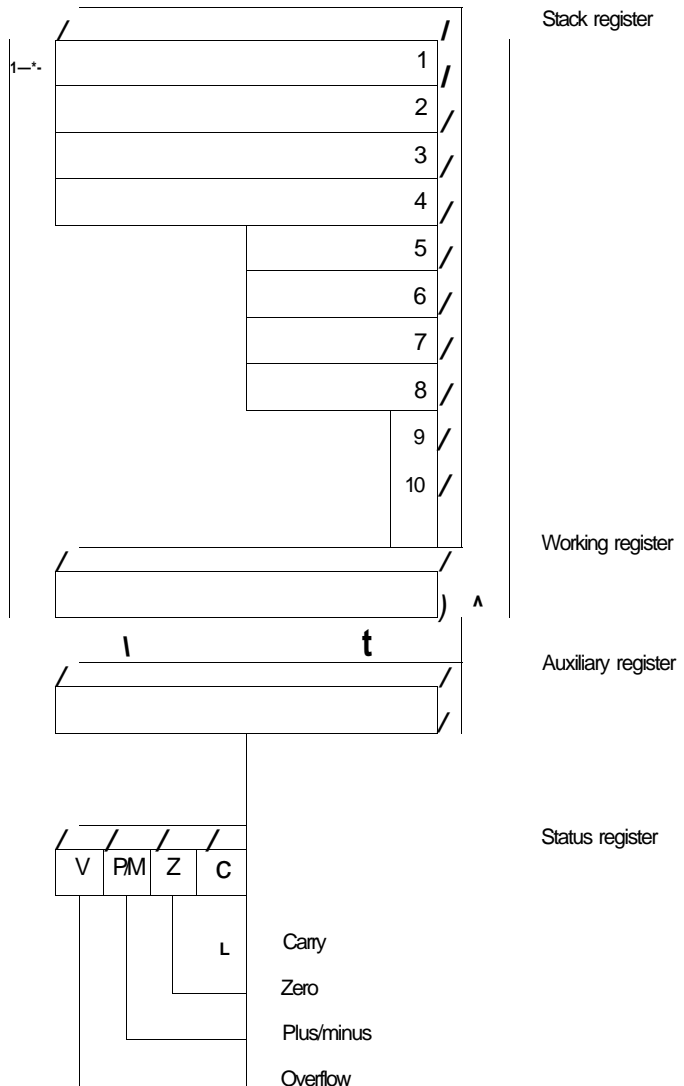


Figure 2-9: Register overview

Programming: Elements and Rules

Registers

Apart from the status register, all others have a variable width, i.e. the registers have the following widths depending on the data type of the operands in the relevant program sequence.

1 bit	for bit operations
1 byte	8-bit with byte operations with values between -128 and 127 as signed integers
1 word	16-bit with word operations with values from -32768 to 32767 as signed integers

Working register

The working register is the most frequently used register, all sequencing and operations being carried out here. It is used as a working memory for analysing the status of the operands and transferring values in any direction required.

Auxiliary register

The auxiliary register is required for some arithmetic operations. It is used to store the overflow (after multiplication) or the remainder (after division). These values can be loaded into the working register using the "GOR" instruction. Working register and auxiliary register together can form a pair. This combination is used in multiplication and division.

Programming: Elements and Rules

Registers

Status register

The status register is four bits long and contains data relating to the contents of the working register (zero or plus/minus bit) and to the result of the previous arithmetic or logic operation (carry and overflow bit).

The individual status bits have the following meanings:

Carry bit: (C)	Carry flag contains the carry over for arithmetic operations (virtually an extension of the working register by one bit).
Zero bit: (Z)	Zero flag describes the contents of the working register. It is <ul style="list-style-type: none">- high if the working register equals zero- low if the working register does not equal zero
Plus/minus: (P/M)	Sign flag indicates the plus/minus symbol of the number in the working register. Often the same as the most significant bit of the working register. It is <ul style="list-style-type: none">- high if the number is negative- low if the number is not negative
Overflow bit: (V)	The overflow flag indicates whether an overflow has taken place during an arithmetic operation. The overflow normally consists of several bits and thus cannot be absorbed by the carry bit. This bit indicates the validity ($V = 0$) or nonvalidity ($V = 1$) of the result of the arithmetic sequence.

When conditional jumps are programmed, the status bits are scanned individually or in combinations (with the branches BGT, BGE, BE, BLE, BLT). The condition of the status register after an operation is described in instructions Chapter 5, IL instructions.

Programming: Elements and Rules

Register

Stack register

Instructions are processed in the programmable controller sequentially, and the sequence result located in the working register is always processed further. This means that stack registers are required to form intermediate results when logical and arithmetical parenthesized expressions are involved.

Ten stack registers are available for bit operations, eight for byte operations and four for word operations. The stack register used is a LIFO register.

Programming: Elements and Rules

Handling Intermediate Results

If the current sequence result has not been stored or does not initiate conditional operations (conditional jumps or set or reset operations) and comes before a Load instruction, it is stored in one of the stack registers **before** the subsequent Load instruction is carried out.

The Load operation is then carried out afterwards. The value in the stack register is thus stored temporarily for further processing. It can be sequenced later with an arithmetic or logic operation with the contents of the working register:

```
LI 0.1          10.1 ^ Working register
AI 0.2          "Working register A I 0.2 —» working register
LI 0.3          "Working register -» Stack register 1
               " 10.3 -» Working register
AI 0.4          "Working register A I 0.4 —* working register
0              "Stack register 1 V working register ->
               "Working register
= M 3.4        "Working register —* M 3.4
```

The order in which values are entered is important for stack processing:

Stack register (operation) working register

The result of this kind of operation can be obtained from the working register.

This rule for the order of the operands can easily be demonstrated by comparing the following program sections.

```
LI 0.1
LI 0.2          LI 0.1
0              01 0.2
= Q 0.3        -*____• 01 0.2
               =Q 0.3
LI 0.1          LI 0.1
LN I 0.2        •*____• 0 N 10.2
0              =Q 0.3
= Q 0.3
```

Programming: Elements and Rules

Handling Intermediate Results

The order is very important when division and logic operations with a negation are involved.

```
LI W 0.0.0
LI W 0.0.02
DIV      "Means IW 0.0.0.0 :IW 0.0.0.2
```

```
LI 0.1
LI 0.2
A N      "Means 10.1 A I 0.2
```

In the same way, the negation is used in the working register in the following example:

```
L 10.1
A 10.2
L 10.3
O 10.4
XO N
```

The operation consists of an Exclusive-OR sequence consisting of the stack register (containing the result from I 0.1 A I 0.2) with the negated working register (contains the result from 10.3 V I 0.4).

Up to four values can be stored in the stack register when word operations are involved. With byte operations eight values can be stored and with bit operations ten. These values are stored in the order of their entry into the stack register. They are then processed in the reverse order, the last value stored being sequenced first of all. The second is then the next to be sequenced and this is continued until the entire stack is empty.

Programming: Elements and Rules

Handling Intermediate Results

```
LI 0.0      " I 0.0 --> Working register
LI 0.1      " I 0.0 --» Stack register 1;
            " I 0.1 --> Working register
LI 0.2      " I 0.1 --» Stack register 2;
            " I 0.2 --» Working register
LI 0.3      " I 0.2 --> Stack register 3;
            " I 0.3 --»Working register
O           "Stack register 3 V Working register
            —» Working register
XO          "Stack register 2 © Working register
            -» Working register
A           "Stack register 1 A Working register
            —» Working register
= Q 0.4     "Working register -> Q 0.4
```

From the above program section the following mathematical equation is obtained:

$$Q\ 0.4 = I\ 0.0\ A\ (I\ 0.1\ ©\ [I\ 0.2\ V\ I\ 0.3])$$

As can be seen from the above, the stack operations can be used as an effective bracketing technique which eliminates the need for auxiliary markers.

Programming: Elements and Rules

IL Syntax Rules

Instruction line

The instruction line consists of an instruction followed by a comment. The comment section contains an operand comment which is used to describe the operand concerned and to which it is permanently assigned. Thus the same text always appears in all instruction lines containing the same operand. This text is managed by the IL editor and is stored in the reference file (file with the suffix ".z42"). Operand-related comments are not kept in the program file (file with the suffix ".q42").

The instruction line is written directly on entry and is checked directly by the syntax check function contained in the IL editor. Any errors that occur are described in plain text. A high degree of protection against programming errors is therefore already ensured during the initial editing stage. Optional comments can also be entered in addition to the operand-related comments. These comments must be prefixed with the "**comment character**" in order to distinguish them from instructions.

The entire range of the character set available can be used for comments.

```
" This is an example of command lines with  
" Operand-related and optional  
" Comments
```

```
L I 0.0      Motor 1 on  
ANI0.2     Limit switch 12
```

Sequence

A sequence consists of several instruction lines which have to fulfill certain conditions. The first instruction line of a sequence must contain a Load instruction. The data type of the operand in this Load instruction determines the data type of the entire sequence. The data type cannot be changed within the sequence.

IL Syntax Rules

Sequence

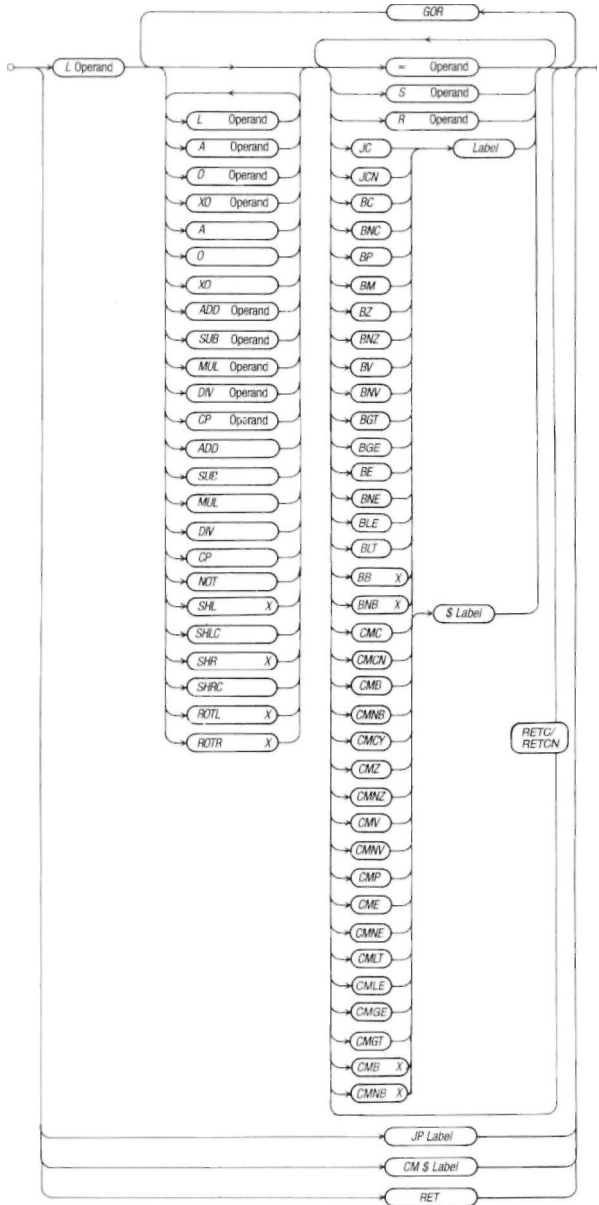


Figure 2-10: Sequence overview

Programming: Elements and Rules

IL Syntax Rules

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

- Allocation of the sequence result to an operand
- Set/reset dependent on the sequence result
- Conditional jumps dependent on the sequence result

"Example of a sequence in bit format.

L I 0.0	Motor 1 on
AN 102	Limit switch 12
A M 24	Run indication motor 1
= Q 34	Motor 1 start

The **GOR** instruction is an exception to this rule. Since it is used following a multiplication or division in order to enter an overflow of the result (after multiplication) or the remainder (after division) into the working register, an allocation followed by GOR is not valid as a sequence termination. In fact, the first part of the result is stored during the allocation operation and if the data types are identical, the processing of the second part result is started.

Programming: Elements and Rules

IL Syntax Rules

Sequence

"Example of a sequence in word format

```
L    IW 0.3.2.0
DN   KW2
=    MW4
"    Check whether IW 0.3.2.0 is even
"    or odd
GOR
OP   KWO
BE   EVEN
BNE  ODD
```

The handling of the stack registers is checked within a sequence. In this case, a sequence is not completed error-free until all the intermediate values stored in the stack registers have been processed again, i.e. the stack must be emptied by the end of the sequence. Values cannot be transferred to the next sequence from the stack registers or the working register.

```
L    IW 0.3.4.0
ADD  MW 4
L    MW 44
1 "IW 0.3.4.0" + "MW 4" in
1 stack register
1 "MW 44" in working register

SUB  QW 0.3.5.0
DN
1 Stack register:
1 ("MW 44" - "QW 0.3.5.0")

      MW 100
```

Block

The combination of several interrelated sequences in one block provides the program with a modular structure and also helps to make the program more legible and understandable.

o—*(Block number)-

>^ Sequence j-

Label

Function block call-up

Figure 2-11: Block structure

Programming: Elements and Rules

IL Syntax Rules

The block begins with a block number, block label and comment. The block numbers are automatically assigned by the IL editor. These numbers are issued in consecutive order and begin with 0. The block label following the number must be no more than eight characters long, and the first character must be a letter. This block label is absolutely necessary if the user program contains jumps or branches.

```
00001 PROGR1 "Program preselect
              LM 100.0
              JC PROGR2
              LI 0.0
              JCN PROGR2
              = M 101.1
              LI 0.2
              = M 101.2
00002 PROGR2 "Relay output of
              "program preselect
              LM 101.1
              = Q0.3
              LM 101.2
              = 0.0.4
```

Block labels are used as jump targets with conditional and unconditional branches. The block header can be used to give a brief description of the block concerned. It is introduced by the " character and must be no longer than one screen line. The block comment is particularly important for documentation tasks. The lines below the header can be used for comment text to provide a detailed functional description of the block.

Note!

It is advisable to write program additions via the # Include instruction in one block. Program instructions in blocks which contain include instructions are not accessible for online modification.

IL Syntax Rules

Main program, structure

Each program (user program with the suffix .q42) for the PS 4 200 series consists of one or several blocks followed by the control instruction EP (End of program).

$O \text{---} \wedge \text{---} \gg \{ \text{Block} \} \text{---} \wedge \text{---} \underline{K} \text{---} \overset{EP}{y}$

Figure 2-12: Main program structure

This instruction is used to control the sequence of a program. At the end of the program it carries out the jump to the operating system of the programmable controller where such operations as the communication with the programming unit are performed and the input and output image registers are updated.

Information about the system configuration of the programmable controller is also contained in the main program. This information is managed in the IL editor and stored in compressed form in front of block 0 in the source program.

This information consists of the following system parameters:

- Date of the last modification to the source program
- Operation with/without image register
- Program memory test active/inactive
The memory test is carried out during program cycle
- Maximum cycle time in ms
- A fault indication is output if this is exceeded
- Two retentive marker ranges
Markers which are declared as retentive do not lose their states in the event of a power failure. Non-retentive markers are cleared if this occurs.

Programming: Elements and Rules

IL Syntax Rules

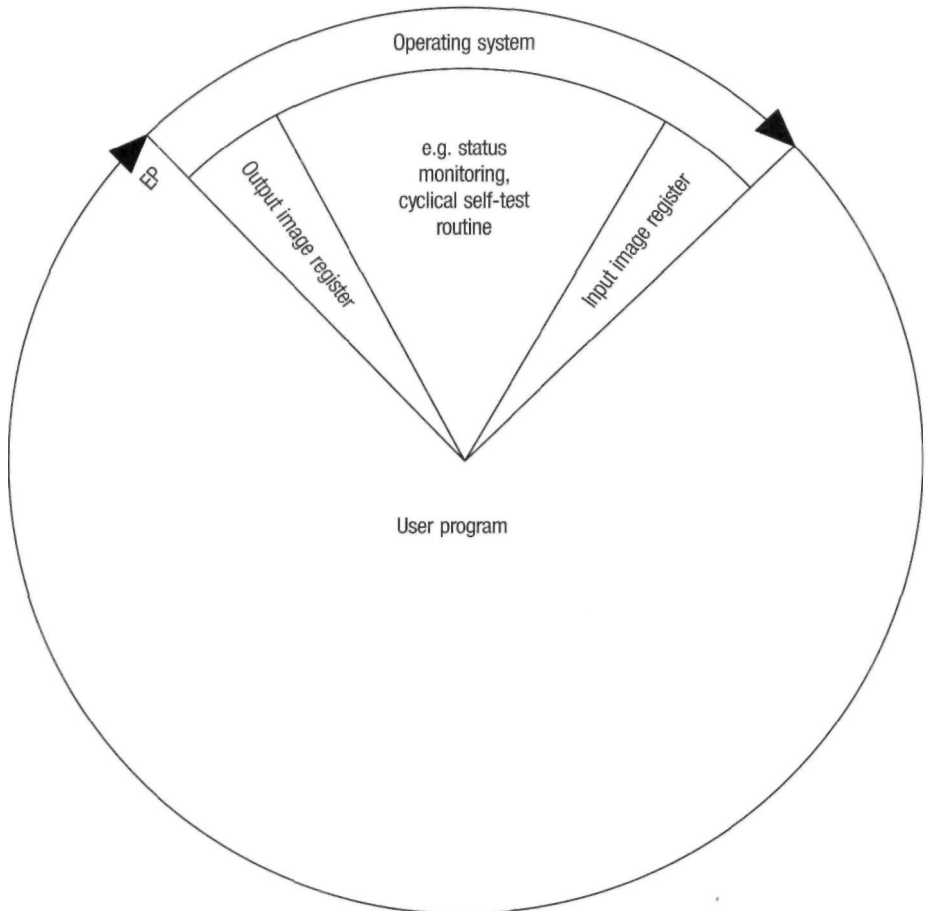


Figure 2-13: Program cycle

Programming: Elements and Rules

IL Syntax Rules

Main program,
structure

The source program has to be translated into machine code in order for it to run in the PS 4 200 series. A program file (file with the suffix .p42) is stored in machine code if this translation takes place error-free. This executable user program can then be transferred to the programmable controller concerned.

Detailed information on the creation of system parameters is provided in chapter 1, Compiling and transferring.

Programming: Elements and Rules

Pre-Processor Instructions

SUCOsoft S 30-S4 200 supports pre-processor instructions which are not translated into machine code but which are control instructions for the compiler and for documentation programs. Pre-processor instructions are preceded by the # character in order to distinguish them from normal IL instructions. These # characters must be put at the beginning of a line and can be assigned an optional comment text. The following pre-processor instructions are available:

Table 2-4: Available pre-processor instructions

Pre-processor Instructions	Effect
# include "Configuration file"	Include the device configuration file
# include (Program file)	Include other files
# include "Reference file"	Combine files
# title "title, 70 lines long"	
# page	
• list	• Control of documentation
# nolist	

Programming: Elements and Rules

Pre-Processor Instructions

Incorporating the configuration file

The incorporation of the configuration file is required for the compiler run. Since this configuration contains information on the type of expansion modules, slave controllers, etc. and at which locations they are used, the compiler can thus check whether the addressing and other specifications are correct.

The correct syntax for this instruction is the following:

```
#include"configuration file.k42"
```

This instruction must always be the first one in the program. This also applies if the PS 4 200 series is used on its own, otherwise an error message is output when the user program is compiled.

Inserting files

A pre-processor instruction with the syntax

```
# include (program file)
```

is replaced by the compiler with the content of the program file stated between the brackets. The files combined in this way in the memory are translated by the compiler into the machine code of the PS 4 200 series. Any errors are indicated by the compiler which also indicates the file name, the block and line number of the error concerned. The name of the main program is used for the file name of the translated instructions.

Include instructions can also be nested inside a file which is also included in the main program.

The # Include instruction (program file) can be used to create function blocks once and use them several times. You can find more details in chapter 3, Structuring programs.

Programming: Elements and Rules

Pre-Processor Instructions

Combining files

In addition to the Include instruction for program files, a similar instruction is also available for inserting or combining reference files. In order to distinguish the text file to be included from a program file, its name is written in between " characters in the following way:

```
# include "Reference file"
```

The complete name with the suffix z.42 for a reference file must be stated.

You can find more details in chapter 3, Structuring programs.

Control of documentation

The following instructions can be used for the printout of documentation:

```
# title " "      Generation of new sub-titles.  
# page          Generation of a new page break.  
# nolist        The program section following this  
                instruction up to the # list instruction is not  
                shown in the documentation.  
# list          See # nolist.
```

More information on these control instructions is provided in the manual AWB 27-1185-GB, chapter 3, Documentation of user programs.

3 Structuring Programs

Contents

General	3-3
The Include instruction	3-5
- Inserting source files	3-5
- Inserting reference files	3-7
- Example	3-8
- Nested program structure	3-10 1
Structuring with program modules	3-11 1
- Definition of a program module	3-11
- Procedure	3-11 "
- Program module calls from an Include file	3-13
Use of program modules	3-15
- Designation of program modules	3-16
- Instruction set of the program module	3-16
- Handling the program modules	3-16
- Calling the module	3-17
- Execution time	3-19
- Rules for the program module call	3-19
- Rules for module programming	3-19
- Program modules as independent files	3-21
- Data transfer with multiple program module calls	3-22
- Example: main program and program modules in one file	3-24
- Example: main program and program modules in different files	3-25
- Example: multiple program module calls	3-26
Control functions	3-29
- Editor	3-29
- Compiler	3-30
- CPU	3-31
Test functions	3-35
- Online programming	3-35
- Status display	3-38

Structuring Programs

General

The structuring of programs in blocks is a recommended way of maintaining a clear overview of the user program even when long user programs are involved.

In addition to ensuring a greater transparency of the program, this technique provides other advantages:

- The effects of any changes are limited to specific sections in the program
- Programming and testing can be carried out in sections
- Error handling is simplified by restricting the search to a specific section of the *program*
- Program libraries can be set up, i.e. reduction of software costs

Individual program sections which handle as self-contained a function as possible within the program sequence can be created and then stored on diskette (hard disk). In this case, these programs should be written without an EP instruction at the end of the program.

With programs that are not too lengthy it is advisable to create a common reference file for all program blocks. If very lengthy files are involved, it may be that the size of the user memory in the personal computer is insufficient. In this case, the reference file must also be subdivided into two or several sub-files.

Structuring Programs

The Include Instruction

Inserting source files

With traditional programming, a linear structure is used by which the individual program sections are sequenced one after the other. The compiler translates the program sections in the existing order and stores them accordingly in the code file.

In order to obtain a clear structuring of the program, programming with the Include instruction enables any number of program sections to be stored in a separate file. When the program sections are translated, the compiler generates a sequential code in the normal way. This means that the compiler inserts the appropriate file each time it comes to an Include instruction. There is therefore no difference between the code files produced by these two programming methods (see Figure 3-1).

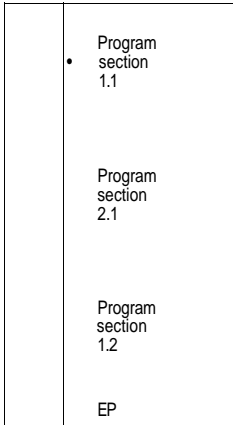
Structuring Programs

The Include Instruction

Inserting source files

Source file

PROGLQxx



Code file

PROGIPxx

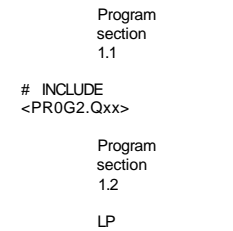
1.1
2.1
1.2
EP

Compiler

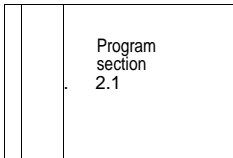
Linear programming structure

Source file

PROG1.Qxx



PROG2.Qxx



Compiler

Code file

PROGIPxx

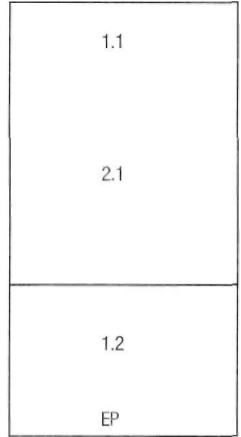


Figure 3-1: Programming with an inserted source file

Structuring Programs

The Include Instruction

Inserting reference files

In addition to this Include instruction for program files, there is a different instruction which has a similar effect when inserting or collating reference files. The file name is enclosed with quotation marks (") in order to distinguish it from program files, e.g.:

```
# include "Reference file name".
```

The text file name must meet the same requirements that apply to the naming of program files:

- Full name must be given, i.e. with suffix.
- The suffix must be a proper reference file suffix, i.e. z42 for a reference file of the PS 4 200 series. The reference file is sorted and checked for errors by the compiler, e.g. multiple definition of elements. The following example will explain the effect of the Include instruction.

Structuring Programs

The Include Instruction

Example

Program MAIN.QXX

```
00000 START " Start of main program
           LI 0.0
           = M0.0
           # include <module1.qxx>
           L INPUT1
           = QB 0.0.0.0
           = MB0
           # include <module2.qxx>
00003 END " End of main program
           EP
```

Reference file MAIN.ZXX

```
INPUT1      IB 0.0.0.0      INPUT byte 0.0.0.0
```

Program MODULE1.QXX

```
00000 START1 " START of 'module1.qxx'
           # include <module1.zxx>
           L AM0
           = 'OUTPUT0
00001 END1 " End of 'module1.qxx'
```

Reference file MODULE1.ZXX

```
AM0          M 0.0          Auxiliary marker MOO
OUTPUT0      Q 0.0          Output bit Bit 0.0.0
```

Program MODULE2.QXX

```
00000 START2 " Start of 'module2.qxx'
           # include "module2.zxx"
           L'AM1
           OI0.4
           = 'OUTPUT1
00001 END2 " End of 'module2.qxx'
```

Reference file MODULE2.ZXX

```
AM1          M1.0          Auxiliary marker M1.0
OUTPUT1      Q 0.0          Output bit Q 0.0
```

Program after pre-processor run

```
00000 START " Start of main program
           LI 0.0
           = M0.0
00000 START " Start of 'module1.qxx'
           L'AM0
           = 'OUTPUT0
00002 END1 " End of 'module1.qxx'
           L 'INPUT1
           = QB 0.0.0.0
           = MB0
00000 START2 " Start of 'module2.qxx'
           L AM1
           OI0.4
           = 'OUTPUT1
00001 END2 " End of 'module2.qxx'
00003 END " End of main program
           EP
```

Reference file after pre-processor run

```
INPUT1      IB 0.0.0.0      INPUT byte 0.0.0.0
AM0          MO.0           Auxiliary marker MOO
OUTPUT0      QO.0           Output bit QOO
AM1          M1.0           Auxiliary marker M1.0
OUTPUT1      QO.0           Output bit Q 0.0
```

Structuring Programs

The Include Instruction

The "module1.qxx" program file is inserted at the appropriate point in the "main.qxx" main program. The instruction

```
# include "module1.zxx"
```

contained in "module1.qxx" ensures that the "module1.zxx" file is inserted at the end of the "main.zxx" reference file. The program file "module2.qxx" is then inserted in the main program with the following instruction:

```
# include <module2.qxx>
```

After all these operations have been completed, the instructions are compiled into the machine code of the controller. All the reference file data for the main program and for the other INCLUDE files and modules involved is then available for compiling. The machine code thus produced is stored on the drive stated, under the name "main.pxx".

Structuring Programs

The Include Instruction

Nested program structure

Include instructions can be used to produce nested, linear or mixed program structures.

Up to 32 program files can be nested with SUCOsoft S 30, and up to 1024 Include instructions can be contained in each main program. Each program file "Name.qxx" may only be included **once** in the main program using the Include instruction.

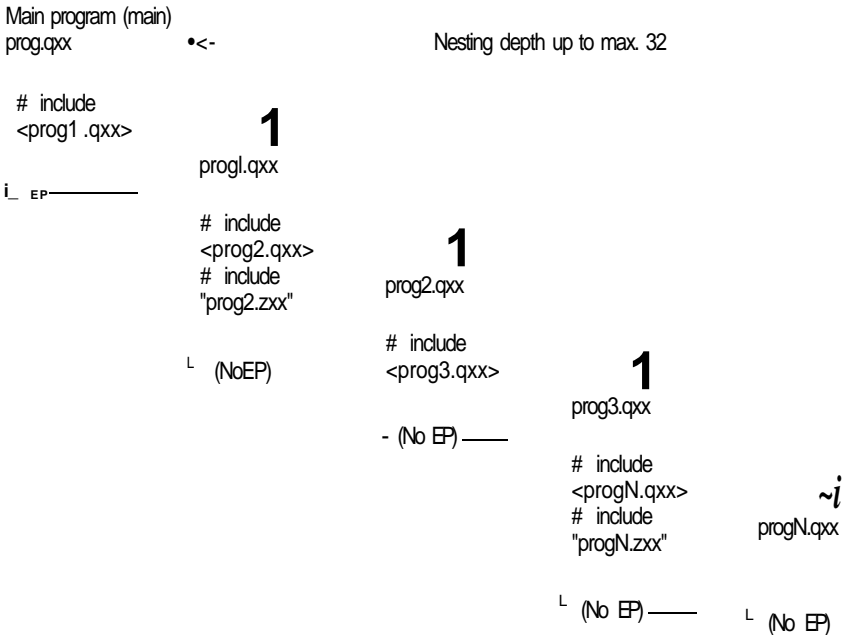


Figure 3-2: Nested program structure

Structuring Programs

Structuring with Program Modules

Program modules form a programming tool with which programs can be structured, thus enabling the programmer to economically store frequently used program sections. It is also possible to store these modules under different names (i.e. in different files).

Definition of a program module

A program module is part of a user program consisting of any number of blocks and which is programmed as a related software module after the main program, i.e. after the "EP" instruction.

A program module can be called from any point in the main program or from other program modules, thus enabling the program modules to be nested. After the program module has been executed, a jump takes place to the program from which the module was called. The program continues with the instruction following the call.

Both before and after a module is called up it is supplied with and cleared of data via globally accessible markers (global data).

Procedure

The user program consists of a main program and any number of PM program modules - see Figure 3-3. The main program is terminated with the EP (end of program) instruction. A program module can be given any name and this is used for calling the module from the main program or another program module by means of the CM.. (Call module) instruction.

The CM., instruction is the point where the processing of the program module is "inserted", i.e. a jump is made to the first instruction in the program module. The EM instruction (end of module) marks the end of the module. The program then continues with the instruction following the CM., call instruction, in the program from which the module was called.

Up to 16 program modules *can* be nested with the PS 4 200 series.

Structuring Programs

Structuring with Program Modules

Procedure

In the following diagrams, the program modules have been given the name PMx. The modules can, of course, be given other names.

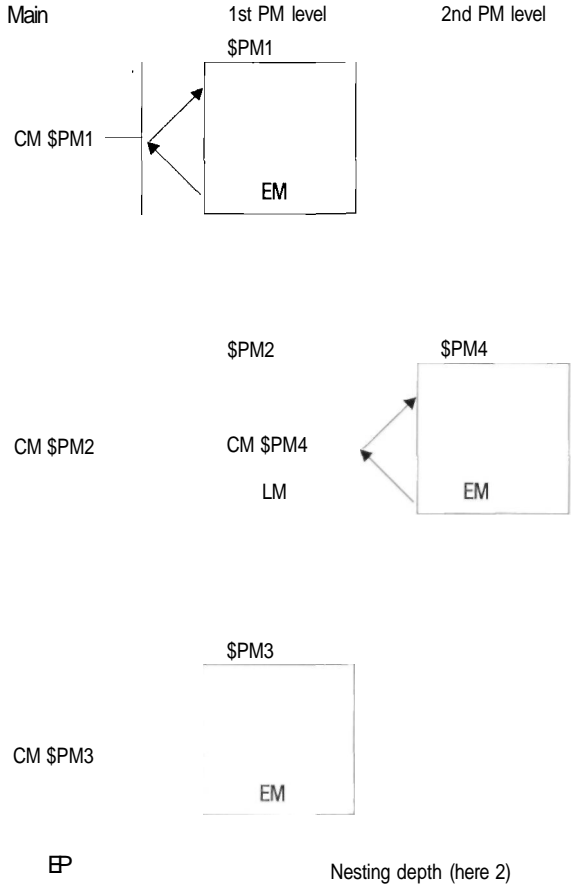


Figure 3-3: Program module calls initiated from the main program and from a program module

Structuring Programs

Structuring with Program Modules

Program module call from an Include file

Program modules can also be called from an Include file (see Include instruction). The example shown in Figure 3-4 shows the source include file "Inc.qxx" which is called up in the main program. This file calls up the PM1 program module, which in turn contains the call instruction for the PM4 module.

After the processing of the PM4 module has been completed, a return is made with the EM instruction back to the PM1 module. The EM instruction in the PM1 module then causes a return to the Include file "Inc.qxx". The main program then continues with other module calls.

The PM4 module is called twice in this example, firstly from PM1 and later from PM2. This kind of multiple calling can be implemented as often as required as long as the cycle time restrictions of the CPU are observed.

Structuring Programs

Structuring with Program Modules

Program module
call from an Include
file

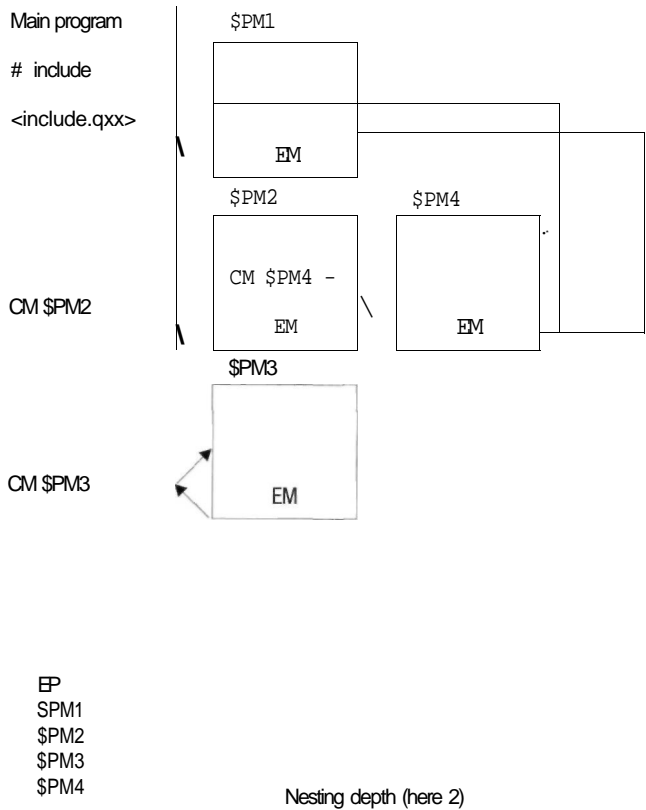


Figure 3-4: Program module calls from an "Include" file of the main program.

The example above shows the logical sequence resulting from the program module structure. The following sections look at the operation of program modules and their physical location in the program memory.

Structuring Programs

Use of Program Modules

In the source program, the program modules are stored in any order after the main program (see Figure 3-5). Each module is terminated with the EM instruction.

The following examples are shown with symbolic operands in IL.

Block	Instruction	
M1	L 'COM-ON A 'VOLT-ON = 'PUMP1 CM \$PM2 L 'END-POS A 'ENABLE = 'MOTOR3	Main program start block 0
M2	L .. CM \$PM1 EP	Main program block 1 End of main program
\$PM1	L 'LEV-MIN O 'PUMP-ERR = 'IND7 EM	PM1 start block 2 End of PM1
\$PM2	L 'BI-MET XO 'INSUFFP = 'IND19	Start of block 3
M1	EM	PM2 block 4 PM2 END

Figure 3-5: Location of main program and program modules in IL

Structuring Programs

Use of Program Modules

Designation of program modules

In order to identify a program module, the first block name of each module is marked with a \$ sign. This means that the name of a program module always starts with the \$ sign followed by up to seven optional characters. The individual blocks are numbered consecutively from 0 up to the number of the last block of the last program module to be programmed.

It is also possible to open additional blocks in a program module. The block names used inside a program module may be the same as the block names in the main program and those of other program modules as the compiler is able to distinguish them.

Instruction set of the program module

All possible jump and branch instructions are only possible within the program modules themselves - jump and branch instructions into and out of a module are not permissible. All other instructions, however, which are permissible in the programmable controller concerned, including the function blocks, can be used in the program modules.

Handling the program modules

The use of program modules requires both the program module calls and the program modules themselves. A program module must be provided in an executable program for every different module call. It is not necessary, however, for every program module in the program to be called. In this case the compiler generates an alarm which informs the user that a program module has not been called. The order in which the modules are entered and the call instruction is optional.

Structuring Programs

Use of Program Modules

Calling the module

The program containing the call instruction is interrupted and the first instruction of the program module is executed. As with JP and JC/BC instructions, the call can be conditional, unconditional or at the end of the program. Calls can be executed in bit, byte or word sequences.

The following different types of calls are available.

Unconditional call:

CM\$(name) *Absolute module call*

Conditional calls (logical calls)

CMOS(name) Call when RA=1
Conditional call of the module when working register is logic 1 in bit sequences

CMCN\$(name) Call when RA=0
Conditional module call when working register with logic 0 in bit sequences

CMBxS(name) Call on Bit x
Conditional module call when bit (x=0-15) on logic 1 in byte/word sequences

CMNB x\$(name) Call on Not Bit X
Conditional module call when bit (x=0-15) is logic 0 in byte/word sequences

Conditional calls: (arithmetic calls)

CMCY\$(name) Call on Carry
Conditional module call when carry bit is set in byte/word sequences

CMNC\$(name) Call on Not Carry
Conditional module call when carry bit is not set in byte/word sequences

CMZ\$(name) Call on Zero
Conditional module call when zero bit is set in byte/word sequences

Structuring Programs

Use of Program Modules

Calling the module	CMNZ \$(name>	Call on Not Zero Conditional module call when zero bit is not set In byte/word sequences
	CMES(name)	Call on Equal Conditional module call when compared values are equal In byte/word sequences
	CMNES(name)	Call on Not Equal Conditional module call when compared values are not equal In byte/word sequences
	CMV\$(name)	Call on Overflow Conditional module call when overflow bit is set In byte/word sequences
	CMNV\$(name>	Call on Not Overflow Conditional module call when overflow bit is not set In byte/word sequences
	CMP\$(name)	Call on Plus Conditional module call when sign bit positive In byte/word sequences
	CMM \$(name)	Call on Minus Conditional module call when sign bit negative In byte/word sequences
	CMGT\$(name)	Call on Greater Than Conditional module call when comparison finds value greater than reference value In byte/word sequences
	CMLT\$(name>	Call on Less Than Conditional module call when comparison finds value less than reference value In byte/word sequences
	CMGES(name)	Call on Greater Than or Equal Conditional module call when comparison finds value greater than or equal to reference value In byte/word sequences
	CMLE \$(name)	Call on Less Than or Equal Conditional module call when comparison finds value less than or equal to reference value In byte/word sequences

Structuring Programs

Use of Program Modules

Execution time

With the PS 4 200 the execution time and the memory required for a program module call and for the return after the EM instruction are 38 LIS/18 byte. These values apply irrespective of the type of module call involved.

The program module calls have the same sequence position as the Jump and Branch instructions. The status registers are not altered. After the return from the program module, the status registers have the same data as before the module call, even if they were modified by the PM operation.

Rules for the program module call

Below is a summary of the rules for calling a program module:

1. The unconditional call must be made outside of a sequence.
2. The name must be no more than 7 characters long. The characters can be alphanumeric characters and the special characters "-", "_", and "/". The special character "\$" must be placed before the name in order to distinguish it from the block label.
3. There are no restrictions on the possible number of calls within one program for a particular program module.

Rules for module programming

After the EP instruction a block is opened with a name beginning with the \$ program module indicator. The module program is then written in IL. The program ends with an "EM" instruction. Include instructions are also permissible within a program module.

Block	Instructions	
\$PM1	L I 0.0 O M32.1 = QO.O EM	PM1 begin block x PM1 end

Program module declaration in IL

Structuring Programs

Use of Program Modules

Rules for module programming

Below is a summary of the rules for a module program:

1. Naming convention: Similar to block labels, the name must begin with "\$".
2. The writing of a program module with a name is only possible once within the entire program.
3. Jumps and branches into the program module from outside are not permissible.
4. Jumps out of a program module into the main program or into other program modules are only permissible with the RET/RETCN instructions.
5. There are no restrictions on branch operations within an individual program module.
6. Program modules can contain other program module calls. The maximum nesting depth is 16. The operating system is responsible for monitoring the nesting depth.
7. Program modules can also be contained in Include files (see paragraph "Program modules as independent files").
8. Module programs can contain Include instructions.
9. Block names in module programs are only valid within the program section where they are located.
10. Block names have a "local status", i.e. the blocks can have the same name in the program module as in the main program or in another program module.
11. A program module call should never be contained within the program module of the same name (recursive calling not permissible).
12. The names of jump labels must not contain the "\$" character.

Structuring Programs

Use of Program Modules

Program modules as independent files

The description of the use of program modules has so far assumed that the program module is kept in the same file as the main program. It is possible, however, to store the program module in a separate file and to incorporate it in the main program using the Include function.

Main program

Block Instructions

MO	L 'COM-ON A 'VOLT-ON = 'PUMP1 CMC \$PM2	Main program start block 0
MI	L .. CM \$PM1 EP	Main program block! Main program end
	#include (dpml.qxx)	Incorporate the PM1 declaration
	• include (dpm2.qxx)	Incorporate the PM2 declaration

Program module dpml.qxx

Block Instructions

```
$PM1 L 'BHMETAL           Start of program module PM1
      A 'HIGH PRESS
      = 'ALARM4

      'KLAXON2
      EM
```

Module declaration in Include file

Structuring Programs

Use of Program Modules

Program modules
as independent files

Programm module dpb2.qxx

Block	Instructions	
\$PM2	L AUTO A 'MIN PRESS = 'ALARM 1 = 'IND7 EM	Start PM2 module

Module declaration in Include file (continued)

The above example shows the call of two program modules \$PM1 and \$PM2. The module programs are written separately to the main program and are kept in the files dpm1.q42 and dpm2.q42. The first block of these files carries the name of the program module, in this case \$PM1 and \$PM2; the program module ends with the EM instruction.

The Include instruction incorporates these files behind the EP of the main program so that the compiled .pxx file has the same structure as shown in Figure 3-5.

It is also possible to store both program modules in one file and incorporate them into the main program with one Include instruction.

Data transfer with multiple program module calls

In addition to program structuring, program module programming has the advantage of enabling modules to be called several times within the main program. After an algorithm has been programmed once, therefore, it can be run several times with a different data content, thus saving valuable memory.

In this case, marker ranges are used for the data transfer, these being reserved by the programmer.

Structuring Programs

Use of Program Modules

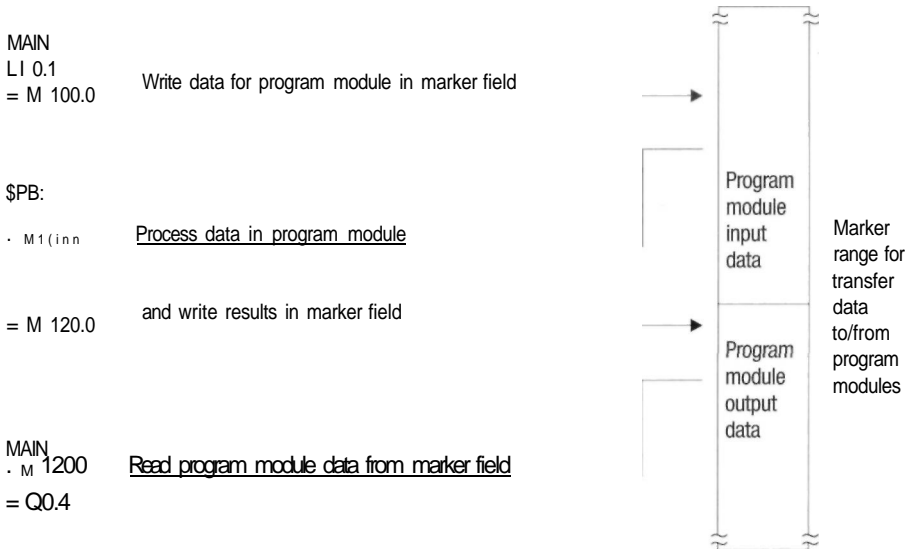


Figure 3-6: Data transfer between the main program and program modules via the marker range

The input data required for the program module is written in the main program to a marker field. The program module is then called and this accesses the input data, performs the algorithm involved and writes the results to an output data field. The main program then allocates the data to the outputs or markers. The input fields are thus filled with values before the program module is called once more. See also "Fault indication module" in the paragraph: Multiple program module calls.

Structuring Programs

Use of Program Modules

The following examples show the principles of program module programming and some of their possible applications.

**Example:
main program and
program module in
one file**

```
00002    START    "The main program starts here.  
           L 'PUMP-ST  
           ='COM-ST  
           CM $PM1    unconditional call  
           L M 20.3  
           = Q0.3  
  
00003    CONTROL  " Control program  
           LI 0.1  
           OM 12.5  
           CMC $PM2   conditional call  
           = 0.0.1  
  
00004    END      " End of main program  
           EP
```

```
00005    $PM1     " The functions of PM1 are  
           ST-CON1  " programmed from  
                   " this point.  
00006    " Program module 1 starts here.  
           " Start conditions  
           LMB 24  
           CP IB 0.0.0.0  
           BE END  
           = QB 0.0.0.0  
00007    END      " End of program module 1  
           EM
```

```
           The functions of PM2 are  
           programmed from  
           this point.  
  
00008    $PM2     Program module 2 starts here.  
00009    CON-COND Control function  
           LI 0.3  
           A M 99.0  
           = TGEN 0 S  
           TGENO  
           S:  
           I: KW250  
           P: Q0.1  
00010    END      " End of program module 2  
           EM
```

Structuring Programs

Use of Program Modules

Example:
main program and
program module in
different files

00002	START	" The main program starts here. L 'KEY23 = 'DRV7 CM \$PM1 unconditional call L M 20.3 = 0.0.4
00003	BLOCK1	LI 0.5 OM 12.5 CMC SPM2 conditional call = Q0.3
00004	END	" End of main program EP " The functions of " program module 1 are " programmed " from here. • include (prgmod1.q42) " The functions of " program module 2 are " programmed " from here. # include (prgmod2.q42)

Structuring Programs

Use of Program Modules

Example: multiple program module calls

"Printout of file c:fault.q42 of 24.3.94"

```
00002      FAULT      "Fault indication program
001
002              "Implemented by multiple calling
003              "of a program module
004
005
006              "Fltn
007
008
009              "INDn      _ru"L_n_n_r
010
011
012              "ACK              .TL
013
014              "A fault (signal 1 of an "FLTn input) is
015              "indicated by the fast flashing of an INDn
016              "indicator light. After the acknowledgement via
017              "the ACK key, this flashing changes to a steady
018              "light. If the fault is no longer present, the
019              "steady light can be acknowledged via the ACK key.
020
021              "A centralized alarm is output on CENI
022              "indicator as soon as the fault is present.
023
00003      ACK              "Edge evaluation of the ACK key
001
002              L ACK              LAcknowledge key
003              AN ACK-EDG        IVLAcknowledge key edge marker
004              =ACK-PUL          IVLAcknowledge key pulse marker
005
006              L ACK              LAcknowledge key
007              =ACK-EDG          IVLAcknowledge key edge marker
008
00004      FAULT1         "First fault
001              "Input data for FIM (fault indication module)
002
003              L 'FLT1            LFault 1
004              = 'FLT            M_Fault n
005
006              LM 100.0          Fault 1 edge marker
007              = 'FLT-EDG        IVLEdge marker fault n
008
009              LM 120.0          Flash marker for fault 1
010              =FLT-FL           IVLFlashing fault n
011
```

Structuring Programs

Use of Program Modules

```
00005      FIMCAL1      "Call of the fault program module
001
002      CM $FIM
00006      IND1          "Output data of FIM for fault 1
001
002      L 'FLT-EDG      M_Edge marker fault n
003      = M 100.0      Fault 1 edge marker
004
005      L 'FLT-FL       M_Flashing fault n
006      = M 120.0      Flash marker for fault 1
007
008      L'IND           MJndication output n
009      = 'IND1        CLOutput light indicator fault 1
010
00007      FAULT2      "Second fault
001      "Input data for FIM (fault indication module)
002
003      L 'FLT2         LFault 2
004      =FLT           M_Fault n
005
006      LM 100.1        Edge marker fault 2
007      = 'FLT-EDG     M_Edge marker fault n
008
009      LM 120.1        Flash marker for fault 2
010      =FLT-FL        M_Flashing fault n
011
00008      FIMCAL2      "Call of fault indication module
001
002      CM $FIM
00009      IND2          "Output data of FIM fault 2
001
002      L 'FLT-EDG      M_Edge marker fault n
003      =M 100.1      Edge marker fault 2
004
005      L 'FLT-FL       M_Flashing fault n
006      = M 120.1      Flash marker for fault 2
007
008      L'IND           MJndication output n
009      = 'IND 2        O_Output fault 2 indicator light
010
00010      CEN          "Output central indication
001
002      L 'CENI         M_Central indication
003      = 'CENQ        CLCentral indication
00011      END          "End of main program
001      EP
002      •include (pm1.q42)
```


Structuring Programs

Use of Program Modules

"Printout of file pm1.q42 of 24.3.94"

```
00002      SFIM
001
002          "Edge evaluation of fault indication
003
004          L'FLT          MJault n
005          AN 'FLT-EDG    MJEdge marker fault n
006          S 'FLT-FL      M_Jlashing fault n
007          S 'FLT-EDG     MJEdge marker fault n
008
009          LN 'FLT        M_Fault n
010          A ACK-PUL      IVLPulse marker Acknowledge key
011          R 'FLT-EDG     M_Edge marker fault n
012
00003      FLACK          "Acknowledge flash
001
002          L 'ACK-PUL     M_Pulse marker Acknowledge key
003          R 'FLT-FL      M_Flashing fault n
004          R 'CENI        M-Central indication
005
00004      TIMER          "Set timer
001
002          TGEN13
003          S: K1
004          I: KW 62
005          P:
006
00005      IND            "Output indication
001
002          L 'FLT_FL      M_Flashing fault n
003          ATGEN13P
004          LN 'FLT-FL     M_ Flashing fault n
005          A 'FLT-EDG     M_Edge marker fault n
006          0
007          = 'IND        MJndication output n
008
00006      CENI          "Central indication
001
002          L 'CENI        M_Central indication
003          O'IND         MJndication output n
004          = 'CENI       M_Central indication
005          EM
```

Structuring Programs

Control Functions

This paragraph gives a description of the control functions which are carried out by the CPU of the programmable controller when the program is entered, during compiling and during the run time.

Editor

The line syntax of the editor checks the following cases:

The special character "\$"'

- must only be placed at the first space in the block name
- must only be present once in the block name
- must not be present in jump labels
- must be present within the program module call instruction.

The line syntax recognizes:

- whether EP or EM was programmed.
- whether a block name or a program module is involved.

Appropriate error messages are generated if the line syntax detects a violation of the syntax rules. The editor knows whether the main program or the program module is currently being processed. It outputs messages that indicate to the user any incorrect use of EP and EM instructions.

Structuring Programs

Control Functions

Compiler

The compiler performs the same monitoring functions as the editor. It also monitors the following events:

- It generates an error message if a program module is called which was not declared at any point:

"Program module: xyz does not exist"

- Program modules can on the other hand be declared but need not be called.

Warning:

"Program module: xyz has not been called"

- Program modules must always be written outside the main program or outside other program modules, i.e. if there is a module program within the main program and/or within another program module, the compiler detects this error:

"No correct program module declaration"

- A so-called recursive call instruction, i.e. the call of the program module currently being processed is not permitted by the compiler.

The recursive call instruction is explained in the following example:

\$PM2		Declaration of program module 2
	L'FLT	
	A'VER	
	= 'POINT6	
	CM \$PM2	Call of PM2 module
	EM	End of module

Summary of error messages:

- Program modules cannot be declared in the main program or within other program modules
- Main program/program module is not terminated correctly (EP or EM missing)
- No correct program module declaration
- Program module: XYZ is present more than once
- Incorrect program module call
- Program module: XYZ has not been declared

Structuring Programs

Control Functions

- Jumps to program modules are not permissible
- Program module name has already been issued
- Recursive program module calls are not permitted
- Main program must be terminated with EP.
- Program modules must be terminated with EM.
- Block name/program module name not correct (impermissible special characters in the name).

Warnings after compiling program:

- Program module: XYZ declared but not called.

CPU (of the PLC)

The individual program modules are stored after the EP command and before the module directory in the program file (.pxx) and are also transferred accordingly to the user program memory (see Figure 3-7).

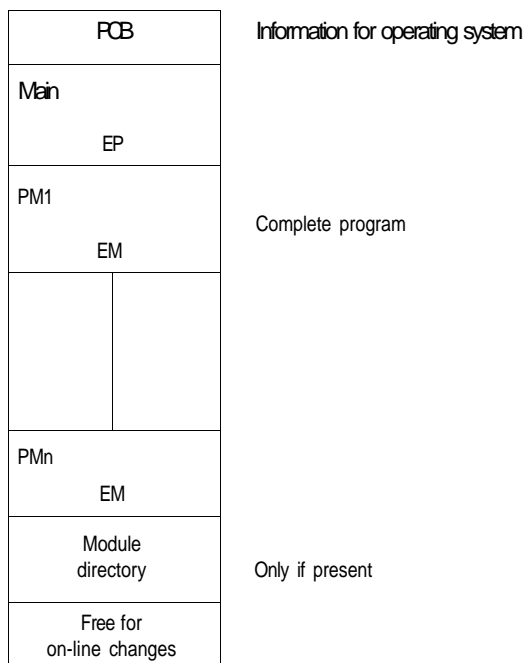


Figure 3-7: Location of the program modules in the user memory

Structuring Programs

Control Functions

The CPU saves the current data before a program module is entered. Saved data is read back on return to the main program.

Operations on entering a program module:

- Saving of bit, byte, word working registers
- Saving of status registers
- Monitoring of nesting depth

Operation on leaving a program module:

- Writing back of status registers
- Writing back of bit, byte, word working registers

After leaving the program module, the working register and all status registers contain the same data as before the program module was called up. This is also the case when the status registers are used in the program module and when these registers therefore contain temporary data. It is not possible to transfer the data to program modules via the registers.

As shown in the following example, the result of a comparison for the conditional call instruction can be used by several program modules.

L IB 0.0.0.0	"Load input byte 0.0.0.0
CP KB 20	"Compare constant 20
CMESPM1	"Call PM1 if compared value equal
CMB 6 SPM2	"Call PM2 if bit 6 of IB 0 = 1
CMGT\$PM3	"Call PM3 if IB 0 > 20
= QB 0.0.0.0	"Store content of IB 0 on output byte 0.

Structuring Programs

Control Functions

Recursive call instructions:

Indirect recursive call instruction of program modules cause cycle time error messages. These types of instruction must be avoided.

Example of an indirect recursive call instruction:

```
$PM2                                Program module 2
    L'BER01
    A'ENAB
    = TLT-EN
    CM $PM3                          PM3 module call
    EM

$PM3                                Program module 3
    L'CANC
    A 'CANC-EDG
    O 'CANC-D
    = 'IND7
    CM $PM2                          PM2 module call
    EM
```

Monitoring of the nesting depth:

If the nesting depth is exceeded (max. 16), a cycle time error message (DCT) is initiated and halts the CPU.

J

I

Structuring Programs

Test Functions

Online programming

Online programming in IL is also possible in program modules in the same way as for the main program.

Note!

The complete scope of online programming functions and restrictions are described in detail in Chapter 4.

Familiarise yourself with these details before using Online programming with program modules and Include files.

All possible modifications using the offline compiler are available with the following exceptions:

- New program module declarations are not permissible. Program module calls can be supplemented or deleted as required (same as for function modules).
- The termination of a module program (EM) cannot be deleted.
- The online programming does not check whether a declared program module is called up.

The online menu provides two search functions in order to make it more easy to find a program module.

- Program module selection.

The F8 key "Program module selection" provides a summary of all program modules present. Program modules that are kept in another file (# include) are also handled.

Structuring Programs

Test Functions

Online
programming

Example: The screen shows the following program section:

```
00012 PUMP3      "Control Pump 3
                  L 'STRT-EN
                  = 'STRT-IND
                  CM SPM1 unconditional call
                  L M 20.3
                  = 0.0.3
00013 PUMP3S     "Pump 3 fault indication program
                  L I 0.3
                  OM 12.5
                  CMC $PM2 conditional call
                  = 0.0.4
00014 END        "End of the main program
                  EP
00015 $PM1       "General pump interlocks
                  L 'INT-COND
                  A 1NT-7
```

F8 is pressed to display at the top right of the screen a summary of all program modules which are programmed:

```
$Main
$PM1
$PM2
$Fit
```

Select the program module required with the cursor and the [Enter] key to display the first screen of the module program. The online editor can be selected by pressing F2.

Select program file

Key F7 "Select program file" is used to provide a summary of all programmed Include files. This comprises all Include files which have been incorporated by the main program or by program modules, in addition to the main program itself.

Structuring Programs

Test Functions

Example: The screen shows the following program section:

```
00012      Pump3      "Control pump 3
                    L 'STRT-EN

                    = 'PUMP3
                    CM $PM1 unconditional call
                    L M 20.3
                    = Q0.3

00013      Pump3F     "Pump 3 fault indication program
                    LI 0.3
                    OM 12.5
                    CMC $PM2 conditional call
                    = Q0.4
                    • include (Flt.q42)

00014      END        "End of main program
                    EP
                    • include (pm1.q42)
                    • include <pm2.q42)
```

A summary of all files that have been combined with the Include instruction is shown in a window at the top right after function key F7 is pressed:

```
main.q42
flt.q42
pm1.q42
pm2.q42
```

Select file required with the cursor and the [Enter] key required to display the first screen of the include file. The online editor can be selected by pressing F2 in order for any necessary changes to be made within this file. The F1 "Return" key is used to leave the file and save the alterations made.

The source file is overwritten with the modified program with F4 "Save program file".

If the PS 4 200 series has a memory module which, e.g. contains a flash EEPROM memory, you can select in the menu

Structuring Programs

Test Functions

Online
programming

whether the program should also be overwritten in this flash EEPROM memory. If you answer with "No", the program modification is lost when switching off the PLC. When switching on the PLC again, the old program is loaded from the flash EEPROM memory.

Status display

The operand states of program modules are also displayed in IL without any restrictions or changes in the form of display involved. The status display can be directly called from the "Test and Commissioning" menu.

If the status display is carried out in IL during commissioning, the status display can also be obtained from the "Online Menu" via F9 and then F4. This path enables simple changing between "Change" and "Observe" operations.

4 Commissioning

Contents

General notes	4-3
Test/Commissioning main menu	4-5
PS 4 200 status	4-7
- Diagnostic status word (DSW)	4-8
- Information byte (INB)	4-14
Device, I/Q Status	4-17
Status display IL	4-23
- Dynamic forcing	4-26
Online program modification	4-29
- General	4-29
- Handling	4-31
- Function key F2 EDIT PROGRAM FILE	4-32
- Function key F7 Select program file	4-35
- Function key F8 Program module selection	4-37
Date/time	4-39

Commissioning

General Notes

The greater the complexity of installations with programmable controllers, the higher the costs are for program testing, commissioning and servicing.

The selection of a suitable PLC system therefore requires particular importance to be attached not only to the functionality of the system itself but also to the testing facilities available for use in program creation, commissioning and service. Particularly in the event of errors, it is important to identify and rectify the cause of the fault as quickly as possible, in order to minimize the costs resulting from loss of production.

The SUCOcontrol PS 4 200 series programmable controllers fulfill this requirement particularly well through a range of functions for testing and commissioning.

This range involves the software tools and the LEDs which indicate the status.

This manual only deals in detail with the test functions in connection with the programmer (personal computer). It should be noted that in fault diagnosis, approximately 90 % of all faults occur in the process peripherals, i.e. in the sensors and actuators.

The test functions described below can be illustrated and explained clearly with the aid of a personal computer and a PS 4 200 series.

e

Commissioning Test/Commissioning Main Menu

The figure below shows the Test and Commissioning menu with the particular test functions that it offers. They are divided into the following sub-menus which are each divided up into further sub-menus. Integral help texts give information on the operation of the individual test functions.

S U C O s o f t - S 3 0

```
TEST / COMMISSIONING          S4-200
F 1  MAIN MENU
F 2  PLC STATUS
F 3  DEVICE I/Q STATUS
F 4  STATUS DISPLAY IL
F 5  ONLINE PROGRAMMING IL
F 6  TRANSFER DRIVE  -> PLC
F 7  TRANSFER      PLC  -> DRIVE
F 8  COMPARISON   PLC <-> DRIVE
F 9  DATE • TIME
F10  H E L P
```

Figure 4-1: Test/Commissioning main menu

Commissioning

PS 4 200 Status

Figure 4-2 on the next page shows the PS 4 200 Status menu. The left side of the display shows the central processing unit. The position of the mode selector switch, the status of the PS 4 200 series and indications are shown. These status indications mean the following:

PS 4 200 series	Explanation
RUN	User program is being processed.
READY	The memory tests performed whenever the power supply is switched on have been completed successfully. The user program can be started.
NOT READY	The program or memory test executed was not successfully completed. The PLC cannot change to the RUN status.
CHANGE H -	The voltage monitor on the backup batteries fitted has indicated that a battery change is required (indication of necessary battery change).
CYCLE TIME	Cycle time exceeded (max. permissible time can be preselected in range from 1 to 255 ms).

Commissioning

PS 4 200 Status

In the centre of the menu, 16 possible events are identified and shown in a diagnostic status word (DSW). The PS 4 200 series detects 14 events. The cycle time is shown on the right of the menu, when the PLC is in the RUN mode. The system version of the CPU and the memory capacity are also shown. This information is of particular use for servicing.

The PS 4 200 series PLC can be started and stopped in this menu. The retentive markers can also be reset, e.g. in order to restart an installation.

PS4 200 STATUS: Last change registered at 14:36:54h	DIAGN. COUNTER Last reset at 00:00:00h	DIAGNOSTICS WORD Last reset at 00:00:00h
<div style="display: flex; flex-direction: column; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="font-size: 2em; margin: 0;">i</p> <p>RUN READY NOT READY CHANGE !! CYCLE TIME</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="font-size: 2em; margin: 0;">i</p> <p>TOGGLE SWITCH: 1 HLT 2 - RUN 3 - RUN <M-RESET></p> </div> <p>lhursdcij,7.7.1994 14:36:E8</p> </div>	<pre> -----ECT I -----i EDC 1 -----EWD 1 00001 EPM 00001 EDR I ===== ERT 1 -----ENR -----DAC BB001 DBM 1 00001 DMC I =====E DLK 1 -----DLS -----DDK I -----DDS 1 </pre>	<pre> Cyc le t ine : 001 ns External memory: 128 KB-FLASH Program: hans Date: 7Jul94 Uersion: ul00 Operating system: U1.00 L SUCOsoft uersion: U1.0 </pre>
<pre> Status - 1 RETURN 2 PLC - Start 3 PLC - Stop </pre>	<pre> F 4 Reset Diagn- counter F 5 Reset Diagn. word </pre>	<pre> F 7 Retentive markers reset F10 HELP </pre>

Figure 4-2: Status menu

Diagnostic status word DSW

The diagnostics status word of the PS 4 200 series consists of 16 bits which are defined as either D bits, which only have an indication function, or E bits, which stop the PLC in addition to the indication function.

Commissioning

PS 4 200 Status

Table 4-1: Diagnostics status word DSW

The first letter stands for category E or D	Meaning
Bit 15 ECT	Cycle time exceeded
Bit 14 EDC	DC failure in the basic unit
Bit 13 EWD	CPU failure
Bit 12 EPM	Error in the program memory
Bit 11 EDR	Retentive data in operating system
Bit 10 ERT	RUN TIME error
Bit 9 ENR	New start only with retentive marker reset
Bit 8	not assigned
Bit 7 DAC	Input voltage drop
Bit 6 DBM	Battery monitoring
Bit 5 DMC	Backup not present
Bit 4 DLK	Local configuration error
Bit 3 DLS	Input/output error
Bit 2 DDK	Remote configuration error
Bit 1 DDS	SBI or network station error
Bit 0	not assigned

The diagnostics bit 0 to 7 are scanned via the user program with the L IS O.X (X = 0-7) instruction. The *error* messages of the bits 8 to 15 cause a stop of the controller so that these bits must not be read.

After the error has been rectified the diagnostics bits of category D can be reset via the Reset button on the front panel of the PS 4 200 series or via the programming device (PC).

Each diagnostics bit has been assigned a counter in order to provide information on the frequency of the

Commissioning

PS 4 200 Status

Diagnostic status
word DSW

errors. These counters can be read and, if required, be reset by the programming device.

If a category E error occurs, the controller switches to the HALT status and can be started again after the error has been rectified. A cold start is always carried out with the switch in the M-RESET position. The restart behaviour with the switch in RUN is preset via the entry in the System parameters submenu of the Programming main menu.

Depending on this configuration SUCOsoft initiates a warm/cold start as required. The standard setting HALT means that it is necessary to cold start the controller via M-RESET. All retentive data is lost.

Description of errors and reactions

ECT

Cycle time exceeded

The maximum cycle time (standard setting: 60 ms/max. 255 ms) set in the System parameters menu has been exceeded during programming execution

Reaction of the running controller:

Indication and stop

EDC

DC failure in the basic unit

Short-circuit or overload in the basic unit.

Reaction of the running controller:

Indication and stop

EWD

Failure of the CPU

CPU hardware watchdog indicates the failure.

Reaction of the running controller:

Indication and stop

Commissioning

PS 4 200 Status

EPM	<p>Error in the program memory</p> <p>An error has been detected during the checksum test or the plausibility check of the user program. The program must be loaded again.</p> <p>Reaction of the running controller: Indication and stop</p>
EDR	<p>Retentive data destroyed in the operating system</p> <p>Important operating system control data is destroyed or does not exist after a RAM change or in a new controller.</p> <p>Reaction of the controller: Does not start; also overall reset of all memory ranges and a re-initialisation.</p>
ERT	<p>Run time error</p> <p>The controller detects an error during operation.</p> <p>Reaction of the running controller: Indication and stop</p>
ENR	<p>Restart only with retentive marker reset</p> <p>This message only occurs if the controller has been configured to HALT (0) in the "Start after NOT READY" option of the System parameters menu and if the user has tried to carry out a warm start after a category E error.</p>
DAC	<p>Input voltage drop</p> <p>The supply voltage of the basic unit has dropped temporarily.</p> <p>Reaction of the running controller: Indication Restart with RESET or M-RESET.</p>

Commissioning

PS 4 200 Status

DBM	<p>Battery monitoring</p> <p>The battery voltage is below the tolerance threshold. The battery module must be changed if the PLC is operated with a RAM module or if retentive data ranges or the real-time clock were used.</p> <p>Reaction of the running controller: Indication Restart with RESET or M-RESET</p>
DMC	<p>No backup</p> <p>The backup which is automatically configured in the memory module is faulty.</p> <p>Reaction of the running controller: Indication Restart with RUN or M-RESET.</p>
DLK	<p>Error in the local configuration</p> <p>The configuration of the PS 4 200 series is not correct or a local expansion (LE module) connected to the basic unit is faulty.</p> <p>Reaction of the running controller: Indication Restart with RUN or M-RESET</p>
DLS	<p>Input/output error</p> <p>Short-circuit or overload of the digital outputs or the value range of analogue outputs of the basic unit or its local expansion modules (LE module) is exceeded.</p> <p>Reaction of the running error: Indication Restart with RUN or M-RESET.</p>

Inbetriebnahme

PS 4 200 Status

DDK

Remote configuration error

The configuration of one or several network stations is faulty, i.e. the entered type designation does not agree with the existing device.

Reaction of the running controller:

Indication

Restart with RUN or M-RESET.

DDS

SBI error or network station error

An error of a network station has been detected via the internal serial interface of the basic unit. The exact location of the error is possible via the diagnostics bytes of the individual network stations.

Reaction of the running controller:

Indication

Restart with RUN or M-RESET.

Note!

The diagnostics bits described above only apply to the PS 4 200 series. Other devices (LE, EM, PS 3) have other status information which is described in the manuals concerned.

Commissioning

PS 4 200 Status

INB information byte

The information byte informs the user on the status of the controller, the images of the network stations, start behaviour of the controller etc. The operating system of the PS 4 200 series generates this INB information byte. Its information bits can be evaluated in the user program but not be written.

The following bits are assigned in the information byte:

Table 4-2: INB information byte

Bit	Meaning
INB0.0	1 st cycle after Reset or Reset button
INB0.1	1 st cycle after Reset button
INB0.3	Information bit for the remaining cycle
INB0.4	Information for the restart: 0 = warm start 1 = cold start
INB0.5	Information on new data in the image of the 1st network line

- INB 0.0 The INB 0.0 bit is High during the first cycle after the PLC is started up with the toggle switch in positions RUN and RUN M-RESET. INB 0.0 can be used in the user program for application related initialization routines, e.g.
- L INB0.0
 JC START
- INB 0.1 INB 0.1 is High during the first cycle if the programmable controller is started via the Reset button on the CPU. INB 0.1 can be used for initialization routines, e.g.
- L INB 0.1
 JC START

Commissioning

PS 4 200 Status

INB0.3 If a user program has been stopped in the middle of the cycle, the INB0.3 bit remains set after the restart until the remaining cycle and the updating of the output image are completed. If required, a special handling of the data ranges can be initialized by scanning the INB0.3 bit when the user program starts again.

Example:

```
L    INB0.3
JC   DATRESET
```

INB0.4 This bit indicates how the controller was last started. With INB0.4 = 0 a warm start was last carried out. With INB0.4 = 1 a cold start was last carried out and all data ranges are initialized. This information is only valid during the first cycle after the restart. The bit is then automatically reset.

INB0.5 This bit is set for exactly one cycle if a network station has sent new data to the basic unit. This INB0.5 bit can be used in simple sequence to determine when new data has been received since the program cycle and the communications process are not synchronized. This data can then be processed directly as required.

Example:

```
L    INB0.5
JC   NEWDATA
```


Commissioning

Device, I/Q Status

This function particularly supports commissioning when checking cables on-site.

A dynamic display of the digital and analogue input states is provided for this purpose without the user program being run.

The cables to the control devices are checked by actuating the control devices concerned, and the relevant signal change is displayed on screen. The analogue input variables for analogue inputs are converted internally and displayed as decimal values, thus allowing analogue transducers to be balanced easily.

When selecting F3 DEVICE, I/Q STATUS in the TEST/ COMMISSIONING menu the set device configuration is read out of the connected PS 4 200 series device. This configuration which has been transferred to the controller together with the user program is displayed graphically on the screen.

```
PS4-201-MM1      116-XD1      116-DX1

EM4-201-DX1

PS3-AC

PS3-DC

EM4-101-DD1/106

- MAIN MENU-->TEST/COMMISSIONING-->DEVICES,I/Q STATUS -----
F 1 Return          F 4 Device status on/off
F 2 PLC - Start     F 5 I/Q display on/off
F 3 PLC - Stop      F 6 Forcing on/off      F10 Help
```

Figure 4-3: Display of the device configuration and the I/Q status

Commissioning

Device, I/Q Status

The current status of the connected devices is indicated to the user by clicking the F4 Device status on function key. The message "OK" or "DIAG" is indicated in the above right corner of the graphical display.

```

      I116-XD1    116-DX1

EM4-201-DX1    HJJJEEI
PS3-AC        -ami
PS3-DC        -aEEi
r4-----M3S/
EM4-101-DD1/106

- MAIN MENU ->TEST/COMMISSIONING->DEVICES,I/Q STATUS aiMHUJ ntHm-rarrew
F 1 Return          F 4 Device status on/off
F 2 PLC - Start     F 5 I/Q display on/off
F 3 PLC - Stop      F 6 Forcing on/off      FIB Help
```

Figure 4-4: Device status

You can select the desired device via the cursor keys and the errors can be displayed by pressing Return.

```

      I116-XD1    116-DX1

EM4-201-DX1    oaaii
PS3-AC        -MM
PS 3-DC        GffiEh
^ EM4-101-DD1/106

-Diagnostics display-----
Bit 2: DDK- Remote configuration error
Bit 4: DLK- Error in local configuration
Bit 6: DBM- Battery monitor

- MAIN MENU ->TEST/COMMISSIONING->DEVICES,I/Q STATUS •UJ'MJJ BEB^BBEi
F 1 Return          F 4 Device status on/off
F 2 PLC - Start     F 5 I/Q display on/off
F 3 PLC - Stop      F 6 Forcing on/off      F10 Help
```

Figure 4-5: Diagnostics display for individual devices

Commissioning

Device, I/Q Status

You can obtain the current input and output states of the device selected with the cursor keys via the F5 I/Q display menu point. Remember that in the HALT status of the controller all input states are refreshed, but the outputs reset. If the PS 4 200 series is configured as a slave, the communications input and output data is indicated as well.

```
PS4-201-riM1      L16-XD1      116-DX1

-Communication input-----
B 0: X00 x00 x00 x00 X00 X00
  iomiunication output-----
EJ 0: x00 x00 x00 x00 x00 x00 x00

-Digital input-----
B0: 00000000 0x00
  -Analogue input-----1
  I0: 00000      0X0000
  U1: 00000      0X0000
  U2: 00000      0X0000
  U3: 00000      0X0000
  -Analogue output-----1
U0: 00000      0x0000

- (IAIN MENU-->TEST/COMMISSIONING-->DEVICES,I/Q STATUS
F 1 Return                F 4 Device status on/off
F 2 PLC - Start           F 5 I/Q display on/off
F 3 PLC - Stop            F 6 Forcing on/off          F10 Help
```

Figure 4-6: Input/output display for individual devices (here: PS4-201-MM1 as slave)

The status of the digital and analogue outputs can be forced via F6 Forcing on/off when the controller is in Halt. This allows the wiring and function of the connected actuators and signal encoders to be checked.

Danger!

The forcing of outputs may lead to unexpected reactions of the controlled machine/plant. Before forcing, ensure that no persons and objects are in the endangered area.

Commissioning

Device, I/Q Status

116-XDI 116-DXi

```
┌ Digital output ─┐
│ B: 11001100 0xcc │
│                   │
│ analogue output-  │
│ W0: 00030 0x001eI │
│                   │
│ Leave box         │
│ Set values        │
│ Reset values      │
```

```
- ttfIN MENU >TnST/COMMISSIONING-->DEVICES,I/Q STATUS *a.HIAlit->-.TTOIB>-
F 1 Return                F 4 Device status on/off
F 2 PLC - Start           F 5 I/Q display on/off
F 3 PLC - Stop            F 6 Forcing on/off      F10 Help
```

Figure 4-7: Forcing of outputs

Remember that the PS 4 200 series can have a different resolution when PS 3-..., PS 306-..., or EM 4-101-AA 1 devices are used as slaves:

Table 4-3: Resolution of the analogue values

	PS 4 200 series	PS 3-...	PS 306-...	EM4-101-AA1
IA	10 bits	8 bits	10 bits	8/12 bits ¹⁾
QA	12 bits	8 bits	12 bits	8/12 bits ¹⁾

¹⁾ Can be selected on the EM 4 device

The cables to the actuators are checked in the usual way by activating the PLC output and checking the reaction of the actuator concerned.

Outputs within the entire device configuration can be forced to 1 or 0 using the forcing function in order to check the cabling to the actuators.

Commissioning

Device, I/Q Status

Danger!

Before checking the cables, ensure that there are no persons and objects in the endangered area!

After the cursor has been positioned (via the cursor control keys) on a module which has digital or analogue outputs, the entry box for forcing can be opened via F6, Forcing ON/OFF (Figure 4-7). In this box you can enter values for the outputs of the selected module.

Forcing applies to digital and analogue outputs. For increased safety, the set states must then be activated with F7.

If there is no reaction when the inputs or outputs are checked, the reference file indicates the signal path from the sensor or actuator to the terminal on the PLC.

Commissioning

Status Display IL

The states of the operands (Figure 4-8) and the function blocks are displayed dynamically in order to test the program. The following information appears on screen depending on the data type of the operand concerned:

- For bit operands, 1 or 0 depending on status
- For multiple-bit operands (byte and word), their content, either as a decimal figure with +/- sign, or a positive value or in binary representation. Display is also possible in hexadecimal form.

The operator can also freeze the Status monitoring function via any key (except the function and cursor keys) in order, for example, to allow sufficient time to evaluate the situation at the exact moment a particular event occurs.

001	LC/TIMER	"Closed Loop Control Timer	
002	TR63 -MS	Closed Loop Control Timer	
003	»•[[] S:	LK -LC/TR-S	Set M-Control-Timer
004	at 1 R:		
005	Bit 3 STOP:		
006	•feL^Ulu1 I:	KH 3000	
007	Elf 3 EQ:		
008	• H l t u] Q=	'LC/TR-Q	Mil-actual value of control timer
009			
010	MOB I*	'LC/TR-Q	MU—actual value of control timer
011	CP KW 2000		
012	"		Timer not yet elapsed
013	BLI LC/TIM-1		
014			
015	"Timer has elapsed	<2000 ms>	retrigger
016			
017	L K 1		
018	MM 'LC/TR-S		Set M-Control-Timer
019	"		Controller operation
-	MAIN MENU->	STATUS DISPLAY	— Decimai uata transfer accine
F 1	Re		
F 2	Select block	F 5 Status monitoring off	F 8 +/- display
F 3	Find string	F 6 DISPLAY RANGE	F 9 Binary display

Figure 4-8: Status display IL

Commissioning

Status Display IL

The function key F6 allows entry into the DISPLAY RANGE menu.

Press function key F2 and enter the marker (M) range required. The following type of menu will then appear on screen showing the markers previously selected, see Figure 4-9.

MU0	H	0	0	0	0	0	0	0	0
MU18		0	0	0	0	0	0	0	0
MW36	0	0	0	0	0	0	0	0	0
MU54									

- DISPLAY RANGE —
F 1 Return
F 2 Display range
F 3 LIFO/FIFO

—————Decimal ————
F 4 Double range
F 6 VARIABLE WINDOW

F 7 Decimal/hex display
F 8 +/- display
F 9 Binary display

Figure 4-9: Display range

Commissioning

Status Display IL

Press the function key F4 and enter the required marker (M) areas. The following type of menu will then appear on screen showing the operand areas previously selected. See Figure 4-11.

This display is particularly useful when checking the function of the ICP and ICPY function blocks for indirect addressing.

MB0	0	0	0	0	0	0	0	0	0	0	0	0
MB12	0	0	0	0	0	0	0	0				

MB50	0	0	0	0	0	0	0	0	0	a	0	0
MB62	0	0	M	0	0	0	0	0	0	0	0	0
MB74	0	0	0	0	0	0	0	0	0	0	0	0
MB86	0	0	0	0	0	0	0	0	0	0	0	0
MB98	0	0										

- DISPLAY RANGE	Decimal	Data	transfer error
F 1 Return	F 4 Double range	F V Decimal/hex display	
F 2 Display range	F 5 Select window	F 8 * /- display	
F 3 LIFO/FIFO	F 6 VARIABLE WINDOW	F 9 Binary display	

Figure 4-10: Double range

Commissioning

Status Display IL

It is also possible in the DISPLAY RANGE menu via F3 to check the status of the First-in-First-out and Last-in-First-out registers (LIFO and FIFO), see Figure 4-11.

Dynamic forcing

You can force byte and word markers dynamically during the RUN operation of the controller via the F6 Display range function key in the VARIABLE WINDOW menu of the IL status display (Figure 4-9). Only operands can be force set dynamically which have been released for this by the System parameters menu (see chapter 1).

0	20	40	60	80	100	120
1903	1222	1000				
2906	2232	2000				
803	12	1903				
2	3	2906				
8092	103	803				
8092	1105	2				
3	1000	8092				
113	2000	8092				
1100	1903	3				
1111	2906	113				
1222	803					LIFOU0-50
1222	2					
1222	8092					CF: 1
2232	8092					CE: 1
12	3					R: 0
3	113					I: 0
103	1100					F: 0
1105	1111					E: 1
1000	1222					Q: 0
2000	1222					

Interrupt with any function key

^^7^*^*7^TTTT7T^^TTr?AAAAAAAAAAAA^M

Figure 4-12: UF0/FIF0 status display

Commissioning

Status Display IL

You can add an entry in the IL via the F2 Add line function key. The operand type, the operand number, the input type and the input value are assigned to the entry in succession. The operand type (MB or MW) and the input type can be modified with the PgUp/PgDn keys. The format of the input value display agrees with the selected input type automatically.

Operand type	Operand number	Current value	Input type	Input value
MB	0	2	Decimal	30
MB	4	00	Hexadecimal	fftt
MU	10	0000	Binary	0010001010111000
ML)	2H	0	Decimal	*/- B

- VARIABLE UINDOU
F 1 Return
F 2 Add line

—**HBHHBBBfflimB**
F 6 Force operand

M-1«4J.I.»i4».U4UfJ-
F 8 Delete current line
F10 Help

Figure 4-12: Dynamic forcing

Danger!

The forcing of markers may lead to unexpected reactions of the controlled machine/plant. Before forcing ensure that persons and objects are not in the endangered area.

The input value selected with the cursor is transferred to the corresponding operand by pressing the F6 Force operand function key, and then processed in the user program, i.e. the value only needs to be forced once. The value of the marker after the dynamic forcing is completed thus only depends on the program logic: The

Commissioning

Status Display IL

markers are **not** reset to the previous value or to zero when you exit the menu.

The display "Current value" in the variable list enables a check of the current data value. The current value is displayed in the format selected under "Input type"; only data entered as binary values is displayed in hexadecimal format.

Dynamic forcing can be used to initiate a desired program sequence or certain actions by setting certain data values. The VARIABLE WINDOW menu can also be used to display up to 18 markers in the list as required in order to monitor their states.

Commissioning

Online Program Modification

General

This function (Figure 4-13) is particularly useful when commissioning. It allows program modifications with the PS 4 200 series in "Run" mode. The following modifications can be carried out:

- Open new blocks
- Insert instructions and allocations
- Delete instructions and allocations
- Alter jumps and jump targets
- Alter function block input parameters

Press F5 in the Test/Commissioning menu and enter the file name and the drive specification. The program will be displayed as in the example below.

```
mm INIT "Initialisation device-configuration
001:II PROG ttinclude "doku.k42"
001 L I 0.0
002 A I 0.1
003 - Q 0.1
004 L KB 20
005 = MBI
"IIIIW 11 "Include 1
001 ttinclude <incl1.q42>
sIsIsM 12 "Include 2
001 ttinclude <incl2.q42>
SBBia Mfinin F "
001 CM $PB1
002 CM $PB2
003 CM SSTATIC
jlsIsSH END "
001 EP
smsRia SPBI "
001 L I 0.0
- MAIN MENU ->ONLINE MODIFICATION $MAIN - d:doku.q42 - d:doku.z42 -
F 1 Return F 4 Save program file JM m r m fcüüü>T:v:nm
F 2 EDIT PROGRAM F 8 Select program module
F 9 STATUS DISPLAY
```

Figure 4-13: Online modification

In the Online menu (figure 4-13), select EDIT PROGRAM FILE F2 and make any modifications in blocks to the program that are necessary.

Commissioning

Online Program Modification

General

Press the Activate key F6 (Figure 4-14) which transfers the modifications to the PS 4 200 series. In order to cancel modifications before activation, use the Undo modification key F7.

```
TMjT=l INIT "Initialisation device-configuration
001 ttinclude "doku.k42"
*ililsln PROG "Mini Program
001 L I 0.0
002 A I 0.1
003 = Q 0.1
004 L KB 20
005 = MB1
'slsflfW ii "Include 1
001 ttinclude <incl1.q42>
'JBJSM Ia "Include 2
001 ttinclude <incl2.q42>
ililslE! MfinillF ""
001 CM $PB1
002 CM $PB2
003 CM SSIATIC
$TITSH FN1) ""
001 EP
ilJHsKH SPR1 ""
001 L I 0.0
=====
- MB IN MENU -->C $MAIN - Insert -
F 1 Return F 4 Add line F 7 Undo modification
F 2 Open block F B Find string F 8 Delete current line
F 3 Select block F 6 fctuate F 9 STATUS DISPLAY
Available PC memory:204.000 Byte Available PLC memory:23.556 Byte Status:RUN
```

Figure 4-14: Online modification

Online modifications involve intentional modifications to program instructions of a machine or plant which is in operation.

Danger!

Before starting with the online modifications, ensure that there are no persons and objects in the endangered area since Online modifications may cause unexpected reactions of the controlled machine/system!

Commissioning

Online Program Modification

Handling

SUCOsoft S 30 programming software and later versions feature the Online modification function integrated in the Test and Commissioning menu. **This** menu is called up from the main menu via F2.

Once in the Test and Commissioning menu, press the ONLINE PROGRAMMING IL function key. The PS 4 200 series must be connected with the programming device in order to use this function.

Online modifications are possible with the programmable controller in either RUN or HALT mode.

The last valid program and reference files are required for the next operation, including all Include files that may be used.

Enter the relevant drive specification for your program files following the prompts on screen.

Note!

It is advisable to store these program files on the hard disk in order to ensure faster processing as well on account of the large data volumes that are possibly involved.

After entering the drive specification, select one of the following functions from the menu

F1	RETURN	Leave Online menu
F2	EDIT PROGRAM FILE	Online programming and modification function

Commissioning

Online Program Modification

Handling

F4 Save Save the program file under the program file current name

If the PLC has a memory module with Flash EEPROM memory, you will be asked if the program file is also to be updated in the Flash EEPROM memory (backup). If you answer with "No", the program modification is lost when switching off the PLC. When switching it on again, the previous program is loaded out of the Flash EEPROM memory.

F7 Select source file Select from a list all PROGRAM and Include files in the main program.

F8 Select program module Select from a list all program modules which are linked to the main program.

F9 STATUS DISPLAY Activate the IL status display.

F10 HELP Supplementary help texts

Function key F2 EDIT PROGRAM FILE

This menu enables the following operations to be carried out:

- Modifying user programs block by block
- Modifying instructions and allocations → operands and existing symbolic names
- Deleting instructions and allocations
- Inserting instructions and allocations -» operands and existing symbolic names
- Inserting comment texts
- Altering jumps and jump targets
- Deleting jump instructions
- Inserting jumps
- Modifying function block parameters
- Opening new blocks by assigning new block labels

Commissioning

Online Program Modification

Restrictions **of the ONLINE** modification:

The following functions cannot be carried out via the ONLINE EDITOR:

- Assigning new symbolic names
- Simultaneous modification in several blocks
- Online modifications in blocks mixed with program instructions and # Include <... .q42>
- Re-programming of function blocks

Note!

Online modifications may **not** be carried out with subprograms which are called up by alarm function blocks (CALARM, FALARM, TALARM). In certain circumstances this may lead to program failure, e.g. if the event signal required to activate the alarm function block occurs when the online modification is activated.

The following cannot be deleted:

- Function blocks

End of program module instruction (EM)

- End of program instruction (EP)

The following cannot be modified or deleted:

- Existing block labels
- Operand comments
- Pre-processor instructions

The function key F6 of the ONLINE MODIFICATION menu (Figure 4-14) activates the modified program. This compiles the modified block and transfers it to the PLC.

The program files <... .p42> and <.r42> and the source file <... .q42> are modified at the same time.

Commissioning

Online Program Modification

Function key F2
PROGRAM ENTRY

The program file is stored in the form of an auxiliary program file with the extensions000.001 etc., and modified, depending on the number of online modifications involved.

When the Online modification menu is left, these auxiliary files are automatically transferred to the current program file and then deleted.

Caution!

Do not therefore switch off your PC before the Online editor has been left via RETURN. Otherwise your program will no longer correspond with the program file on the disk.

The amount of user program memory available in the PLC is reduced with every online modification made. Keep the blocks in your user program as short as possible. The memory capacity as well as the remaining memory is indicated with every online modification made.

If your program memory capacity is used up on account of too many online modifications, you can transfer your optimized (i.e. shortened) program after the new compiler run with the CPU in Halt mode (offline).

Danger!

If active outputs set by the previous program cycle are edited online so that they then no longer have an allocation within the program, they will remain High until the next POWER OFF in the PS 4 200 series.

Commissioning

Online Program Modification

The outputs are also reset when switching the controller to the HALT status.

In some applications it is necessary to return edited, but not yet activated program sections to their original state. Function key F7 (Figure 4-14) should be used for this.

These applications include:

- Discarding (abandoning) the last modification(s) made
- Switching to Status display
- Calling up an Include program file

Function key F7 Select program file

If the main program contains Include program files, these too may be edited with the ONLINE EDITOR. In order to call up the appropriate Include file, press function key F7. All the Include files that are called up by the main program are then listed.

Mark the desired file name with the cursor keys in the list right above in order to call up the Include file concerned.

Note!

Put # Include instructions in separate blocks since the program instructions of blocks containing Include instructions cannot be edited via the ONLINE EDITOR if they are written before or behind an Include instruction.

Commissioning

Online Program Modification

Function key F7

Select program file

Example:

00005	"Block 5 LI 0.0 AM 10.0 = Q 0.0 # INCLUDE <PROGRAM.q42> LQ 0.0 AI 0.1 = M 10.0		ONLINE modification not possible
00006	"Block 6 LI 0.0 AM 10.0 = Q 0.0	ONLINE modification possible	
00007	"Block 7 # INCLUDE <PROGRAM.q42>		ONLINE modification not possible
00008	"Block 8 LQ 0.0 AI 0.1 = M 10.0	ONLINE modification possible	

Commissioning

Online Program Modification

Function key F8 Program module

Online modifications can also be made if the main program contains program modules.

Call up the required program module by pressing function key F8 (Figure 4-15). All the program modules called up by the main program are then listed.

Mark the desired file name with the cursor keys in the list shown at the top right in order to call up the program module concerned. See Figure 4-15.

```
-----
<del>S</del>S<del>I</del>S<del>I</del> INII      "Initialisation device-configuration"
001          tt include "doku.k42"
<del>M</del>T<del>H</del>U  PROG      "Mini Program"
001          L I 0.0
002          A I 0.1
003          - Q 0.1
004          L KB 20
005          = MBI
<del>S</del>I<del>S</del>I<del>B</del>H 11      "Include 1"
001          ttinclude <incl1.q42>
<del>S</del>I<del>U</del>K<del>M</del> 12      "Include 2"
001          ttinclude <incl2.q42>
<del>M</del>T<del>H</del>U  MODULE  "
001          CM $PB1
002          CM $PB2
003          CM JSTATIC
JBBBB END
001          W
<del>M</del>T<del>H</del>U  $PB1
001          L I 0.0
- MAIN MENU-->ONLINE MODIFICATION  SMAIN -      d:doku.q42 --      d:doku.z42 --
F 1 Return          F 4 Save program file          F 7 Select source file
F 2 EDIT PROGRAM FILE
-----
F 9 STATUS DISPLAY
```

fpB1
SPB2
SSTATIC

Figure 4-15: Program module directory

Commissioning

Date/Time

Figure 4-16 shows the Date/Time menu which you find in the TEST/COMMISSIONING menu via F9 the date and the time can be specified for

- the PC
- the real-time clock of the PS 4 200 series
- summer/winter time changes

	Date <DD.MM.YYYY>	Time <HH:MM:SS>
PC	Fr.,08.07.1994	10:03:34
PS	Fr.,08.07.1994	10:03:34

- Date/Time

F 1 RETURN

F 2 PC date

F 3 PC time

F 4 PC date/time > PLC

F 5 PLC date

F 6 PLC time

F 7 PLC date/time -> PC

F 8 Uinter/sunner time

F10 HELP

Figure 4-16: Date/time

5 IL Instructions

Contents

General		5-3
- Abbreviations		5-4
- Conditional bit		5-4
=	Allocation	5-6
A	AND	5-8
ADD	Addition	5-10
B...	Conditional branches	5-12
CM...	Possible program module call-ups	5-14
CP	Compare	5-16
DIV	Division	5-18
EM	End of module	5-21
EP	End of program	5-22
GOR	Load auxiliary register	5-23
JC, JCN	Conditional jumps	5-24
JP	Unconditional jump	5-25
L	Load	5-26
MUL	Multiplication	5-28
NOP	No operation	5-30
NOT	Negation	5-31
0	OR	5-32
R	Reset	5-34
RET	Return	5-36
RETC, RETCN	Conditional returns	5-37
ROTL	Rotate left	5-38
ROTR	Rotate right	5-40
S	Set	5-42
SHL	Shift left	5-44
SHLC	Shift left with carry	5-46
SHR	Shift right	5-48
SHRC	Shift right with carry	5-50
SUB	Subtraction	5-52
XO	Exclusive OR	5-56

IL Instructions

General

This manual gives a comprehensive description of the instructions for the PS 4 200 series as well as an overview of the modified conditional bits for each instruction. They are listed in alphabetical order.

The following table shows all operands which can be used with the instructions. Please ensure that the data type (bit, byte, word) stated in each instruction is the same as the data type of the operands.

Table 5-1: Operand overview

Designation	BIT	BYTE	WORD
Inputs ¹⁾	I	IB, IAB, ICB	IW, IAW, ICW
Outputs	Q	QB, QAB	QW, QAW
Markers	M	MB	MW
Constants ¹⁾	K	KB, KHB	KW, KHW
Real-time clock ¹⁾	-	CKxx	-
Peripheral access	IP ¹⁾ , QP	IPB ¹⁾ , QPB	-
Status/diagnosis	IS ¹⁾	ISB ¹⁾	ISW ¹⁾
Communication data	-	RDB, SDB	-
Information	INB x.y ¹⁾	-	-

¹⁾ These operands cannot be used for the following operations:

- Allocation (=)
- Return (R)
- Set (S)

Special features of each instruction are explained in examples if required.

Pressing (RETURN) after the valid IL syntax has been entered automatically causes the instruction concerned to be written in the correct format thus ensuring a standard display format. This is particularly useful, for example, when using the FIND/REPLACE function in the IL program editor.

IL Instructions

General

Note!

When entries are made in hexadecimal form, the syntax check cannot distinguish the following points:

e.g. KHBB1
 KHBC1
 KHBE1

Entries like the one above must be made **with** a space as follows:

 KHBuBI
 KHBuCI
 KHBuE1

The description in the header shows in which sequences (bit, byte, word) the instructions may be used.

Bit	Byte	Word
-----	------	------

Abbreviations

The following abbreviations are used:

RA	Bit working register (1 bit wide)
RAb	Byte working register (8 bit wide)
RAw	Word working register (16 bit wide)
RHw	Word auxiliary register (16 bit wide)
RS	Bit stack register (1 bit wide)
RSb	Byte stack register (8 bit wide)
RSw	Word stack register (16 bit wide)

Conditional bit

C	Carry bit
Z	Zero bit
P/M	Plus/minus bit
V	Overflow bit

IL Instructions

Allocation

Bit Byte Word

Description

The contents of the working register are allocated to the operand indicated. The original value of the operand is overwritten. In an allocation to a negated operand, the negated contents of the working register are allocated to the operand.

When an allocation is made to a peripheral output, the corresponding output is written in the image register only in the program section "Operating system activities".

All operands of Table 5-1 can be used with an allocation, except the operands marked¹). Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

The working register and the auxiliary register are not altered by the allocation.

Updated conditional bit

Carry bit (C)		Not altered
Zero bit (Z)	1	Not altered
	0	Otherwise
Plus/minus bit (P/M)	1	Not altered
	0	Otherwise
Overflow bit (V)		Not altered

IL Instructions

Allocation

Bit Byte Word

Example

IL	Status	Remarks
LI 0.0 A 10.1	1 0	The states of inputs 0.0 and 0.1 are ANDed and the result is stored in the working register.
= M0.1	0	The content of the working register is transferred to marker 0.1.

IL Instructions

AND

Bit Byte Word

Description

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

With AND sequences of word operands, the corresponding bit of every operand involved is sequenced.

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

All operands of Table 5-1 can be used with an AND sequence. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

The auxiliary register is not altered by the AND function.

IL Instructions

AND

Bit Byte Word

Updated conditional bit

Carry bit (C)	1	Not altered
Zero bit (Z)	1	Set if working register equals zero
	0	Otherwise
Plus/minus bit (P/M)	1	Set if the result is negative, i.e. the most significant bit is set;
	0	Otherwise
Overflow bit (V)		Not altered

Example

IL	Status	Remarks
LIB 0.0.0.0	10001010	Load input byte 0.0.0.0 in the working register RAb.
AMBO	01001011	The bits of marker byte 0 are then ANDed with the bits in the working register. The result is set in the working register.
= QB 0.0.0.0	00001010	The contents of the working register are then allocated to output byte 0.0.0.0.

Byte Word

Description

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

An operand is added in the same way to the last value stored in the stack register.

Note!

The values involved in the addition are interpreted as integers (whole numbers) with plus/minus symbols.

All operands of Table 5-1 can be used with the Addition instruction. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

The auxiliary register is not influenced by the addition.

Byte Word

**Updated
conditional bit**

Carry bit (C)	1	Set if a carry-over has taken place, i.e. if the sum is higher than 8 bits with byte operations and 16 bits with word operations.
	0	Otherwise
Zero bit (Z)	1	Set if the result equals zero
	0	Otherwise
Plus/minus bit (P/M)	1	Set if the most significant bit is set
	0	Otherwise
Overflow (V)	1	Set if an arithmetical overflow has occurred, i.e. if the result exceeds the limits of the signed number range (-128 to +127 for byte or -32768 to +32767 with word operations)
	0	Otherwise

Example

IL	Status	Remarks
LMB 3	00110110	Load marker byte 3 in the working register RAB.
ADD MB 3	00011010	Add marker byte 3 to the working register. The result will be in the working register.
BV ERROR = MB4	01010000	When the overflow bit is set branch to the ERROR routine. The working register is allocated to marker byte 4.
JP CONTINUE		The calculated value is valid. The permissible number range was not exceeded.
ERROR		The result of the addition is not valid. The limits of the number range were broken. An error routine can be entered here if required.
CONTINUE		Rest of program

Byte Word

Description

Conditional Branches		Explanation
BB	BNB	Bit of working register
BC	BNC	Carry bit
BZ	BNZ	Zero bit
BP	BM	Sign (+,-) bit
BV	BNV	Overflow bit
BE	BNE	Equal
BLT		Less than
BGT		Greater than
BLE		Less equal
BGE		Greater equal

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

Conditional branches are permissible only in byte and word sequences and are of practical value only if they follow arithmetical operations.

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

IL Instructions

Conditional Branches

B.

Byte Word

Example of BE (Branch when Equal)

	IL	Status	Remarks
	LKB25	00011001	Load constant 25 to the working register.
	CP IB 0.0.0.0	00101011	Compare with input byte 0.0.0.0 and set the status register.
	BE EQUAL		If equal, branch to block label "EQUAL".
	JP CONTINUE		This instruction is only executed if values not equal. The program section with the label "EQUAL" is then jumped.
EQUAL			This program section is only executed when the compared values are equal.
CONTINUE			

Example of BB (Branch depending on state of individual bits in working register)

	IL	Status	Remarks
	LMB110	10100011	Load marker byte 110 into the working register.
	BNB 7 VPOS	x = 1	Examine bit 7 of the working register. If bit is Low, branch to VPOS.
	JPEND		If bit 8 of the working register is High then jump to END. The program section labelled VPOS is jumped.
VPOS			This program section is only executed when marker bit $110.7 = 0$, i.e. when MB 110 is positive.
END	EP		End program

IL Instructions

Possible Program Module Call-ups

Bit Byte Word

Syntax of the PM call-up	Call-up depending on	Data type
Unconditional call-up ¹⁾ CM \$ (Name)		Bit, Byte, Word
Conditional call-ups (Working register) CMC(N) \$ (Name) CM(N)B (x) \$ (Name)	RA working register Bit x of the working register RAborRAw	Bit Byte, Word
Conditional call-ups (Status register) CMCY \$ (Name)		Byte, Word
CMCNC \$ (Name) CM(N)Z \$ (Name) CM(N)V \$ (Name) CMP \$ (Name)	Carry bit Zero bit Overflow bit	Byte, Word Byte, Word Byte, Word Byte, Word
CMM \$ (Name)	Sign bit (Plus/minus)	Byte, Word
Conditional call-ups (after comparison) CM(N)E \$ (Name) CMGT \$ (Name) CMLT \$ (Name) CMGE \$ (Name) CMLE \$ (Name)	Equal Greater than Less than Greater equal Less equal	Byte, Word Byte, Word Byte, Word Byte, Word Byte, Word

¹⁾The CM \$ command is to be considered as a sequence which consists of an instruction. The unconditional call-up may thus be programmed only after the sequence has been completed.

Bit Byte Word

Description

Program modules can be called up as absolute or conditional call instruction depending on the state of the working register. They can also be called up as conditional instructions dependent *on* the state of the status register or on a specific comparison (<; =; >; <; >). These call-ups can also be used for negative conditions as shown in the table on the previous page. The name of the program module must always be given after the instruction, and should be prefixed with the \$ character, which indicates that the program concerned is a program module.

If the condition is not fulfilled, the program continues with the next instruction.

The working, auxiliary and status registers are assigned the same data content after the module has been processed. When the program module is being processed, these registers can be used freely without any restrictions.

Byte Word

Description

Compare via working register:

The indicated operand is compared with the content of the working register and the relevant conditional bits are set for evaluation by means of the arithmetical branch instructions. Comparison is carried out internally in the form of a subtraction: the operand (subtrahend) is subtracted from the content of the working register (minuend). The values involved are not altered.

When making a comparison with the last value stored in a stack register, the content of the working register (subtrahend) is subtracted from the content of the stack register (minuend).

Note!

The values involved in the comparison are interpreted as integers with plus/minus symbols.

All operands of Table 5-1 can be used with the Comparison instruction. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

The auxiliary register is not affected by the comparison.

Byte Word

**Updated
conditional bits**

The conditional bits are altered immediately via the comparison. They cannot, however, be used for evaluation.

Only the following are possible when used in conjunction with CP:

- BE (equal)
- BIT (less than)
- BGT (greater than)
- BLE (less equal)
- BGE (greater equal)

Example

	IL	Status	Remarks
	LMB2 CPMB3	10100011 01101110	Load marker byte 2 and compare with MB 3 in working register by subtraction.
	BLT LESS		If MB 2 is < MB 3, branch to LESS.
	JP CONTINUE		This program section is processed if MB 2 is > 3.
LESS			This program section is processed if MB 2 is < MB 3.
CONTINUE			

Byte Word

Description

Division via working register:

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

Division via the stack register is as follows. The content of the stack register (dividend) is divided by the content of the working register (divisor). The result is written into the working register, the remainder into the auxiliary register.

Note!

The values involved in the division are interpreted as unsigned integers (i.e. without plus/minus symbols).

Division can produce one of the following two types of results. Depending on the dividend and divisor:

1. If the quotient is within the range of 0 to 65 535 inclusive, i.e., constitutes a valid number, the quotient and the remainder are stored as valid results in the relevant registers. The zero bit is set depending on the quotient; the overflow bit is deleted.
2. If the divisor equals zero, the values in the working and auxiliary registers are invalid. This can, in this case, be indicated by the overflow bit which is set.

All operands of Table 5-1 can be used with the Division instruction. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

Reminder:	Dividend	Quotient
	Divisor	

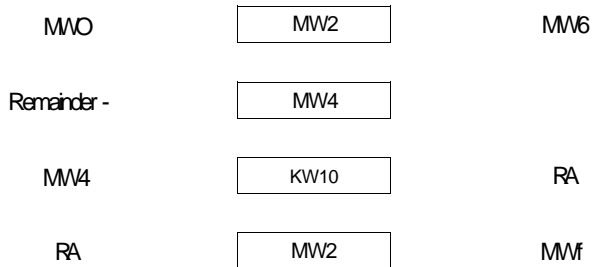
Byte Word

Updated conditional bit

Carry bit (C)		Not altered
Zero bit (Z)	1	Set if the result equals zero
	0	Otherwise
Plus/minus bit (P/M)	1	If the result is negative, i.e., if the most significant bit is set
	0	If the result is positive
Overflow bit M	1	Set if the divisor equals zero
	0	Otherwise

Example

To divide the contents of MW 0 by MW2 and store the result in MW6. The decimal part of the answer is stored in MW8.



Byte Word

Example

	IL	Remarks
	LMW0 DIV MW 2 = MW6	Division of MW 0 by MW 2 Store result in MW 6
	GOR	The auxiliary register is loaded into the working register.
	BV DIVZERO	If the divisor is equal to zero then the division is invalid. Branch to the label "DIVZERO".
	= MW4	The remaining integer is stored in MW4.
	LMW4 MULKW10 DIV MW 2	To calculate the value of the first decimal place of the remainder first multiply by 10 and then divide by divisor.
	= MW8	The result is the first decimal place. It is stored in MW8.
DIVZERO	JPEP LKWO = MW6 = MW8	End of calculation Divisor is zero; the result is invalid and is deleted.
EP	EP	

End of Module

Bit Byte Word

The end of module command EM marks the end of a program module. It must always be written as the last instruction at the end of each program module.

Example

\$BP1

LI 00

A I 0.1

O1 02

=Q0.3

EM

EP

IL Instructions

End of Program

Bit Byte Word

Description

The EP instruction is the logical and physical end of the program. This instruction must be placed at the last step in the main program and causes a jump to the operating system.

Registers and data are not altered.

Example

00001

1 L I 0.0

2 A I 0.1

3 0 1 0.2

4 = Q 0.3

5 EP

Byte Word

Description

The content of the auxiliary register is loaded into the working register.

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

The content of the auxiliary register is not altered.

The status registers are not affected.

Example

Division of MWO by MW2 with rounding. If the first decimal place is greater than 5, it is rounded up. The result is stored in MW6.

	IL	Remarks
	LMWO DIV MW 2 = MW6	Division of MW 0 by MW 2 Store result in MW 6
	GOR	Load auxiliary register in working register.
	= MW8	Store remaining integer in MW8.
	LMW8 MUL KW 10 DIV MW 2	Calculation of the first decimal place (Rest x 10): MW 2
	CPKW5 BLTEP	If the first decimal place is less than 5, round off. The result in MW 6 is therefore correct. Continue with EP
ROUNDUP	LMW6 ADDKW1 = MW6	The first decimal place is greater than 4; it must be rounded up.
EP	EP	

See also example for MUL and DIV.

Bit

Description

The current sequence result is compared with 1 or 0. If they agree, the program is continued at the location which is indicated as the jump target. If the condition is not fulfilled, no jump occurs. The jump target must always be the beginning of a block (a block label).

Conditional jumps are permissible only in bit sequences.

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

Example

Pulse generator

IL	Remarks
LKO = Q1.0	Reset output 1.0
TPO [] S: LN Q 1.0 [] R: [W] I: KW 5000 [] P: [W] Q: LTPOP	Generate a 5s cycle pulse
JCN CONTINUE	If the timer T5 has not yet timed out, the program jumps on the label "CONTINUE".
LK1 = 0.1.0 •	This program section is run for one cycle if the timer T5 times out.
CONTINUE	

IL Instructions

Unconditional Jump

JP

Bit Byte Word

Description

The program will continue wherever the jump was targetted to. The target must be the beginning of a block.

This command is a sequence consisting of one instruction. Unconditional jumps should therefore only be used at the end of a sequence.

The auxiliary and status registers are unchanged by this jump instruction.

Example

IL		Remarks
	LMBO CPMB1 BGT TARGET 1 BLT TARGET 2 JP CONTINUE	By comparing the values of the markers the program is either branched to TARGET 1 if greater than or TARGET 2 if less than. If the marker values are equal then both TARGETS must be jumped.
TARGET 1	JP CONTINUE	The greater than comparison is valid.
TARGET 2		The less than comparison is valid.
CONTINUE		Further program

IL Instructions

Load

Bit Byte Word

Description

The value of the indicated operand is loaded into the working register. The original content of the register is overwritten.

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

The operand is not altered.

All operands of Table 5-1 can be used with the Load instruction. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

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

Example

IL	Status	Remarks
LI 0.0 O 10.1	1 0	Input 0.0 is ORed with input 0.1 and the result is stored in the working register.
LI 0.2	1	The result stored in the working register is shifted to the bottom of the stack register and the state of input 0.2 is stored in the working register.
A		The value stored in the bottom of the stack register is ANDed with the value in the working register.
= Q0.0	1	The value in the working register is transferred to output 0.0.

Byte Word

Description

Multiplication via working register:

The indicated operand is multiplied by the content of the working register and the product is then stored. The original content of the working register is overwritten. The operand is not altered.

Multiplication via stack register:

The operand is multiplied by the last value stored in a stack register in the same manner.

Note!

The values involved in the multiplication are interpreted as unsigned integers (i.e. without plus/minus signs).

The product of the multiplication of two 16 bit numbers is a 32 bit number. The lower word value of the product (16 bits) is stored in the working register, while the higher word value is stored in the auxiliary register. This overflow, as it is known, can be processed with the instruction "GOR".

All operands of Table 5-1 can be used with the Multiplication instruction. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

Byte Word

**Updated
conditional bit**

Are undefined and must thus be scanned after this command.

Example

IL	Status	Remarks
L MB 200	00010101	Load the value of marker byte MB 200 to the working register RAb.
MUL MB 201	00000101	Multiply the contents of the working register with the value of marker byte 201. Store the result in the working register.
= QB 0.0.0.0	01101001	Display the result on the output byte QB 0.0.0.0
GOR		The auxiliary register is loaded into the working register.
= QB 0.0.1.0	00000000	The higher section of the results is now in the working register and is allocated to the output byte Q 0.0.1.0
CONTINUE		

NOP

IL Instructions

No Operation

Bit Byte Word

Description

The NOP instruction does not influence registers or data. It can be written at any point in the program regardless of the data type in the sequence.

Example

IL	Status	Remarks
LI 0.0	1	The inputs are ANDed.
A 10.1 NOP NOP	1	The two NOP instructions are reminders that two more conditions are to be added subsequently to the sequence.
= 0 0.3	1	The result is allocated to output Q 0.3.

Bit Byte Word

Description

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

The operation may be carried out in the bit, byte and word sequences.

The auxiliary register is not affected by the negation.

Updated conditional bit

Carry bit (Q)		Not altered
Zero bit (Z)	1	Set if the result equals zero
	0	Otherwise
Plus/minus bit (P/M)	1	Set if the result is negative, i.e. the most significant bit is set
	0	Otherwise
Overflow bit (V)		Not altered

Example

IL	Status	PF*	Remarks
LI 0.0	0	0	The result of the functions is stored in the working register.
A I 0.1	1	0	
A I 0.2	1	0	
O I 0.3	0	0	
NOT = Q0.5	1	1	The result stored in the working register is inverted (negated). The state of the working register is transferred to the output Q 0.5

*PF sequence result (Power flow)

IL Instructions

OR

Bit Byte Word

Description

OR sequencing of the indicated operand with the content of the working register. The result is stored in the working register. The original content of the working register is overwritten. The operand is not altered.

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

The operand is paralleled (OR function) with the last value stored in the stack register in the same manner. If a negation is entered, it influences the content of the working register, i.e., the last value stored in the stack register is paralleled with the negated content of the working register.

All operands of Table 5-1 can be used with the OR sequence. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

The auxiliary register is not affected by the OR sequence.

IL Instructions

OR

Bit Byte Word

Updated conditional bit

Carry bit (C)		Not altered
Zero bit (Z)	1	If the sequence result equals zero
	0	Otherwise
Plus/minus bit (P/M)	1	If there is a negative result with byte or word operations, i.e. the most significant bit is set
	0	Otherwise
Overflow bit (V)		Not altered

Example

IL	Status	PF	Remarks
LM3.0	1		The markers 3.0 to 3.3 are ORed together and the result allocated to marker 6.0.
OM3.1	0		
OM3.2	1		
OM3.3	0		
= M6.0	1		

IL Instructions

Reset

Bit

Description

The indicated bit of the operand is deleted if the content of the working register equals " 1 ". If this reset condition is not fulfilled, the operand is not altered. The operation is permissible only in bit sequences.

All operands of Table 5-1 can be used with the Reset instruction, except the operands marked¹⁾. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

The working and auxiliary registers are not altered.

Updated conditional bit

Carry bit (C)		Not altered
Zero bit (Z)	1	If the working register equals zero
	0	Otherwise
Plus/minus bit (P/M)		Not altered
Overflow bit (V)		Not altered

Example

IL	Status	PF*	R
LI 0.0 AN I 0.1 SQ 0.4	1 0	1 1 1	If the input 0.0 is High and input 0.1 is Low output 0.4 will be set High.
LI 0.1 RQ 0.4	1 0	1 1	The self-holding function is stopped as soon as the input 0.1 is set.

*PF sequences result (Power Flow)

RET

IL Instructions

Return

Bit Byte Word

Description

This instruction causes a return from the current program level to the next higher level, e.g. from the sub-program to the main program or from the main program to the operating system.

The RET command is a sequence which consists of an instruction. The return may thus only be programmed after a sequence is completed.

Example

IL	Comment
	Sub-program
LI 0.3 = M3.5	Instruction
RET	Return to the main program

Bit

Description

The current sequence result is compared with 1 or 0. If they agree, a return to the next higher level is carried out.

IL	Comment
LI 0.0 AM 3.4	AND sequence of I 0.0 and M 3.4
RETCN	Return to the next higher level if AND sequence is not fulfilled.

ROTL

IL Instructions

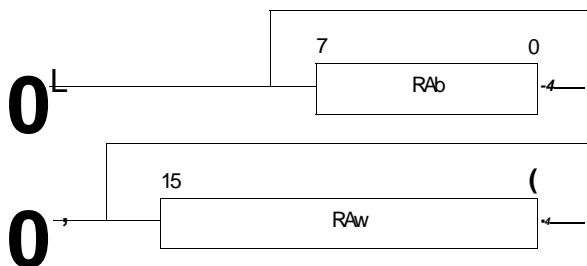
Rotate to the Left

Byte Word

Description

The content of the working register is shifted to the left. In the process, the most significant bit moves to the least significant location of the working register and simultaneously into the carry bit of the status register.

With byte sequences 8 rotation steps are possible and with word sequences 16 rotation steps.



The operation is permissible in byte and word sequences.

The content of the auxiliary register is not altered.

Syntax

Datatype	Instruction	Action
Byte	ROTL ROTLn	rotate RAb left n = 1...8
Word	ROTL ROTLn	rotate RAw left n-1...16

IL Instructions

ROTL

Rotate to the left

Byte Word

Updated conditional bit

Carry bit (C)	1	Set if the last rotated bit was set
	0	Otherwise
Zero bit (Z)	1	Set if the working register equals zero
	0	Otherwise
Plus/minus bit (P/M)	1	Set if the result of rotation is negative, i.e., the most significant bit is set
	0	Otherwise
Overflow bit		Not altered

Example

IL	Status	Remarks
LMB4	01011011	The bits in marker byte 4 are shifted to the left and the most significant bit will be shifted to the least significant bit.
ROTL	10110110	
= MB4		

ROTR

IL Instructions

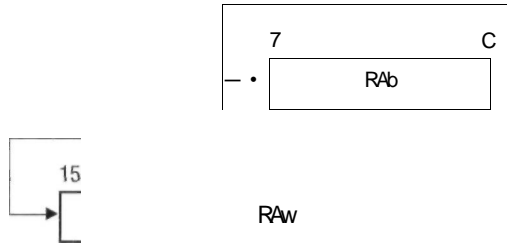
Rotate to the Right

Byte Word

Description

The content of the working register is shifted to the right. In the process the least significant bit moves to the most significant location of the working register and at the same time into the carry bit of the status register.

With byte sequences 8 rotation steps are possible and with word sequences 16 rotation steps.



The operation is permissible in byte and word sequences.

The content of the auxiliary register is not altered.

Syntax

Data type	Instruction	Action
Byte	ROTR ROTRn	rotate RAb right n = 1...8
Word	ROTR ROTRn	rotate RAw right n = 1...16

IL Instructions

Rotate to the Right

Byte Word

Updated conditional bit

Carry bit (Q)	1	Set if the last rotated (most significant) bit was set
	0	Otherwise
Zero bit (Z)	1	Set if the working register equals zero
	0	Otherwise
Plus/minus bit (P/M)	1	Set if the result of rotation is negative, i.e., the most significant bit is set
	0	Otherwise
Overflow bit (V)	1	Set if the plus/minus symbol has altered after rotation
	0	Otherwise

Example

IL	Status	Remarks
LMBO	01101110	The bits of marker byte 0 are rotated to the right twice. The state of bit 8 is transferred to bit 1 and the state of bit 7 is transferred to bit 0.
ROTR2	10011011	
= MB0	10011011	

Bit

Description

The indicated bit of the operand is set if the content of the working register equals " 1 ". If this setting condition is not fulfilled, the operand is not altered.

All operands of Table 5-1 can be used with the Set instruction, except the operands marked¹⁾. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

The working and the auxiliary registers are not altered.

IL Instructions

Set

Bit

Updated conditional bit

Carry bit (C)		Not altered
Zero bit (Z)	1	Set if the working register equals zero
	0	Otherwise
Plus/minus bit (P/M)		Not altered
Overflow (V)		Not altered

Example

IL	Status	PF*	Remarks
L I 0.5	1	1	When input 0.5 is on the marker 0.6 is set High.
S M 0.6	1	1	This marker will maintain this condition regardless of the status of input 0.5.

*PF sequence result (Power flow)

SHL

IL Instructions

Shift to the Left

Byte Word

Description

The content of the working register is shifted to the left. In the process the most significant bit moves to the carry bit of the status register. A "0" is drawn into the least significant location.



The operation is permissible only in byte and word sequences. The highest shift step number is 8 in byte operation, 16 in word operation.

The content of the auxiliary register is not altered.

Syntax

Data type	Instruction	Action
Byte	SHL SHLn	shift RAb left n = 1...8
Word	SHL SHLn	shift RAw left n = 1...16

Byte Word

**Updated
conditional bit**

Carry bit (C)	1	Set if the last shifted (most significant) bit was set
	0	Otherwise
Zero bit (Z)	1	Set if the working register equals zero after the shift operation
	0	Otherwise
Plus/minus bit (P/M)	1	Set if the result of the shift operation is negative, i.e., the most significant bit is set
	0	Otherwise
Overflow bit (V)		Not altered

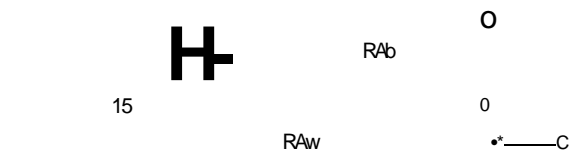
Example

IL	Status	Remarks
L MB 151	00000010	The bits in marker byte 151 are shifted left by one place. This is equivalent to multiplying by 2.
SHL	00000100	
= MB 151	00000100	

Byte Word

Description

The value in the working register is shifted to the left. The carry bit of the status register is shifted to the least significant bit of the working register and the most significant bit of the working register is shifted to the carry bit.



The operation is possible in byte or word sequences.

The content of the auxiliary register is not altered.

Updated conditional bit

Carry bit (C)	1	Set when the most significant bit is set after the shift operation
	0	Otherwise
Zero bit (Z)	1	Set when the working register equals zero after the shift operation
	0	Otherwise
Plus/minus bit (P/M)	1	Set when the working register is negative, i.e., the most significant bit is set
	0	Otherwise
Overflow bit (V)		Not altered

Byte Word

Example

IL	Status	Remarks
LMB12 SHL = MB12	01001101 10011010 10011010	The bits of the marker byte 12 are shifted to the left. In this sequence the lowest significant bit becomes a zero. The most significant bit moves into the carry bit.
LMB13 SHLC = MB13	10110110 01101100 01101100	The bits of the marker byte 13 are shifted to the left. In this sequence the lowest significant bit takes the value of the carry bit. The most significant bit moves to the carry bit.

SHR

IL Instructions

Shift to the Right

Byte Word

Description

The content of the working register is shifted to the right. In the process, the least significant bit moves into the carry bit of the status register. A "0" is drawn into the most significant location.



The operation is permissible only in byte and word sequences. The highest shift step number is 8 in byte operation and 16 in word operation.

The content of the auxiliary register is not altered.

Syntax

Data type	Instruction	Action
Byte	SHR SHRn	shift RAb right n = 1..8
Word	SHR SHRn	shift RAw right n = 1...16

Byte Word

**Updated
conditional bit**

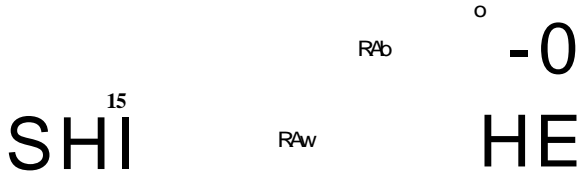
Carry bit (C)	1	If the last shifted (most significant) bit was set
	0	Otherwise
Zero bit (Z)	1	Set if the working register equals zero after the shift operation
	0	Otherwise
Plus/minus bit (PM)	1	Set if the result of the shift operation is negative, i.e., the most significant bit is set
	0	Otherwise
Overflow bit (V)		Not altered

Example

IL	Status	Remarks
L IB 0.0.0.1 SHR 4 = MB20	00101001 00000010 00000010	The value of input byte 0.0.0.1 is to be divided by 16 and stored in maker byte 20. Any remainder is to be ignored.

Byte Word**Description**

The value in the working register is shifted to the right. In this case the status of the carry bit in the status register moves to the most significant bit of the working register and the least significant bit moves to the carry bit.



The operation is permissible in byte and word sequences.

The content of the auxiliary register is not altered.

Updated conditional bit

Carry bit (C)	1	Set when bit 0 (the least significant bit) is set
	0	Otherwise
Zero bit (Z)	1	Set when contents of working register is zero
	0	Otherwise
Plus/minus bit (PM)	1	Set when content of working register is negative, the most significant bit is set
	0	Otherwise
Overflow bit (V)		Not altered

Byte Word

Example

Cascade 4 marker bytes to construct a 32 bit shift register

IL	Status	Remarks
LMB9 SHR = MB9	10101001 C = 1 01010100	The contents of MB 9 is shifted to the right and the state of the least significant bit is stored in the carry bit.
LMB8 SHRC = MB8	11101000 C = 0 11110100	The state of the carry bit will be transferred to the most significant bit of MB 8 and the contents will be shifted to the right.
LMB10 SHRC = MB10	11100101 C = 1 01110010	The value of the least significant bit is shifted to the carry register. The value of the carry bit is shifted to the most significant location and the marker byte 10 is shifted to the right.
LMB9 SHRC = MB9	00001110 C = 0 10000111	The state of the least significant bit is stored in the carry bit. The state of the carry bit is transferred to the most significant bit and the marker byte 9 is shifted to the right. The state of the least significant bit is shifted to the carry bit.

Byte Word

Description

Subtraction via working register:

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

Subtraction via stack register:

The content of the working register (subtrahend) is subtracted from the content of the stack register (minuend). The result is written into the working register.

Note!

The values involved in the subtraction are interpreted as integers with plus/minus symbols.

All operands of Table 5-1 can be used with the Subtraction instruction. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

The auxiliary register is not influenced by the subtraction.

Remember: Minuend - Subtrahend = Difference

Byte Word

**Updated
conditional bit**

Carry bit (C)	1	Set if a so-called "borrowed bit" was required, i.e., the minuend was smaller than the subtrahend
	0	Deleted if there is carry-over from the most significant bit (this is ignored with the subtraction)
Zero bit (Z)	1	Set if the result equals zero
	0	Otherwise
Plus/minus bit (P/M)	1	Set if the result is negative, i.e., if the most significant bit is set
	0	Otherwise
Overflow bit (V)	1	Set if an arithmetical overflow has taken place, i.e., if the result exceeds the permissible range (-128 to +127 for byte operations, -32768 to +32767 for word operations)
	0	Otherwise

SUB

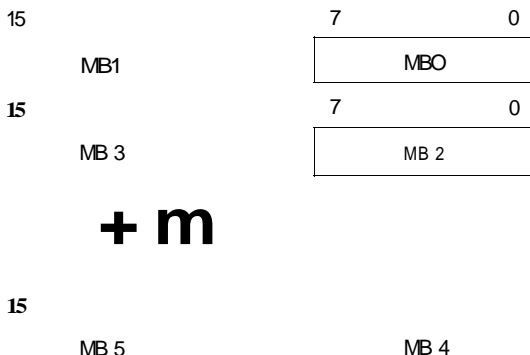
IL Instructions

Subtraction

Byte Word

Example

of Cascading



	IL	Status	Remarks
SUBTRA	L MB 0 SUB MB 2 BV ERROR = MB 4 BC CARRY JP CONTINUE 1	00101011 00010110 00010101 C = 0	Subtract the lower order bytes and store the result in MB 4. If this calculation results in an overflow, branch to the CARRY routine.
CARRY	L MB 3 ADD KB 1 BV ERROR = MB 3		If a carry-over (borrow bit) is present, a 1 is added to the subtrahend of the higher order byte.
CONTINUE 1	L MB 1 SUB MB 3 = MB 5 BV ERROR JP CONTINUE 2	01011101 00110111 00100110	Subtract the higher order bytes and store the result in MB 5.
ERROR			ERROR routine if an overflow occurs in an arithmetic operation.
CONTINUE 2			

IL Instructions

Exclusive OR

Bit Byte Word

Description

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

In exclusive OR sequencing of byte or word operands the corresponding bits of each operand involved are sequenced.

The operand is paralleled in an exclusive OR sequence with the last value stored in the stack register in the same manner. If a negation is inserted here, it will affect the content of the working register, i.e., the last value stored in the stack register is paralleled (XOred) with the negated content of the working register.

All operands of Table 5-1 can be used with the Exclusive-OR sequence. Please ensure that the data type mentioned above (Bit, Byte, Word) is the same as the data type of the operands.

The auxiliary register is not altered by the exclusive OR sequencing.

Truth Table:

W1	W2	W1 e W2
0	0	0
0	1	1
1	0	1
1	1	0

Bit Byte Word

Updated conditional bit

Carry bit (C)		Not altered
Zero bit (Z)	1	Set if the sequencing result equals zero
	0	Otherwise
Plus/minus bit (PM)	1	Set if the result is negative with byte and word operations, i.e., the most significant bit is set
	0	Otherwise
Overflow bit (V)	1	Not altered

Example

IL	Status	Remarks
LI 0.1	1	The output 0.2 will only be set when input 0.1 or 0.2 are High but not when both are.
XO I 0.2	0	
= 0 0.2	1	

6 Function Blocks

Contents

General			6-3
- Engineering and commissioning notes			6-3
- Key to symbols			6-4
- Designations			6-4
BID	Code converter: binary to BCD	Word	6-5
C	Up/down counter	Word	6-6
CALARM	Counter alarm function block		6-9
CK	Time/Date comparator		6-12
CP	Comparator	Word	6-16
DEB	Code converter: BCD to binary	Word	6-17
FALARM	Edge alarm function block		6-19
FIFOB	First In-First Out register	Byte	6-22
FIFOW	First In-First Out register	Word	6-24
ICP	Block comparison	Byte	6-26
ICPY	Block transfer	Byte	6-30
LI FOB	Last In-First Out (Stack register)	Byte	6-34
LIFOW	Last In-First Out (Stack register)	Word	6-36
RDAT	Reload data		6-38
SCK	Set real-time clock		6-40
SDAT	Save data		6-42
SK	Sequential control		6-44
SR	Shiftregister	Bit	6-46
SRB	Shift register	Byte	6-50
SRW	Shift register	Word	6-52
TALARM	Timer alarm function block		6-54
TF	Off-delayed timer		6-58
TGEN	Generator function block		6-60
TP	Pulse transmitter		6-61
TR	On-delayed timer		6-62

Function Blocks

General

This manual gives individual descriptions of the function blocks, the pages being arranged alphabetically in accordance with the code references of the function blocks. The header lines contain the most important function block data and the syntax, followed by the designation of inputs and outputs and, where appropriate, truth tables. This first part of the function block description is intended to provide a brief overview. In the second part the function of the function block is explained with the aid of texts and diagrams.

Engineering and commissioning notes

New retentive function blocks should always be added to the end of the user program during commissioning on account of the dynamic memory management feature of the controller.

The number of the function blocks to be used is not restricted. A restriction is only given by the capacity of the user memory. Theoretically, the upper limit of function blocks is 65 535.

The organisation of the function blocks, their incorporation in the user program and the behaviour of function block inputs is described in Chapter 2.

Function Blocks

General

Key to symbols

	Optional value
	Rising edge; the function block has to recognize a change from 0 to 1.
	Separation sign in front of register length, retentive behaviour or time base
	Retentive; the function block called up in this way becomes zero-voltage proof (retains its content)
(Block No.)	The appropriate value or term from the heading is entered here.
(Register length)	The pointed brackets are not written.
(Time base)	
(Input/Output)	
(No. of inputs)	

Designations

The following symbol designations are used to identify the data of the input or output to which the symbol is assigned.

IL	Typ
[]	Bit
[B]	Byte
[W]	Word
[&]	Address
[\$]	Subprograms
[*]	Time and date parameters (only for CK function block)

Function Blocks

BID

Code Converter:
Binary to Decimal

Word

Syntax

Call-up:
BID < Block No. >

As operand:
BID < Block No. >u< Input/output >

Number ranges: -32768...0...+32767

Result: 5 decades

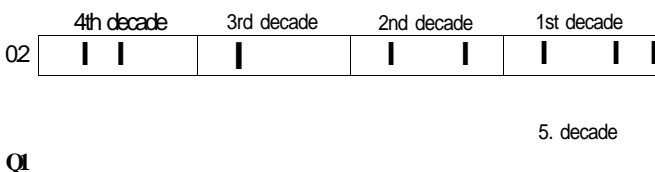
Performance time: 460...560 (is)

Representation

IL
 BID 10
 [W] I: binary input
 [I] QS: Sign output
 [W] Q1: Decade 5
 [W] Q2: Decade 1-4

Description

A 16-bit binary coded number is converted into a 5-decade BCD number. The sign is displayed at the QS output (0 4 +, 1 A -).



'hex	dez	QS	Q1	Q2
0000	0	0	0 0 0 0	0 0 0 0
0001	1	0	0 0 0 0	0 0 0 1
7FFF	32767	0	0 0 0 3	2 7 6 7
8000	-32768	1	0 0 0 3	2 7 6 8
8001	-32767	1	0 0 0 3	2 7 6 7
FFFF	-1	1	0 0 0 0	0 0 0 1

Function Blocks

Up/down Counter

Word

Syntax

Call-up:

C < Block No. > - R

- R only, if retentive operation is required.

As operand:

C < Block No. > < Input/Output >

Number range: 0...65535 (unsigned)

Performance time: 170 µs

Representation

IL

C17-R

11: Pulse forward (Up)
D: Pulse reverse (Down)
S: Set
R: Reset

[W] 1: Preset value input

Z: Count zero

[W] 0: Count

Truth table

Input Type Function	U Bit	D Bit	S Bit	R Bit	I Wort
Count up	1	x	x	0	x
Count down	x	1	x	0	x
Set	x	x	1	0	Value
Delete	x	x	x	1	x

Description

When a rising edge is at the S-input, the value at the I-input is transferred to the counter. The counter is incremented when there is a rising edge at the U-input and is decremented when there is a rising edge at the D-input.

The counter status evaluated depends on the user program. If negative values also occur, use the signed number range -32 768 to 32 767. If only positive values are required, the range is from 0 to 65 535. The number circle is run through cyclically.

Function Blocks

Up/Down Counter

Word

This means that the value 0 appears after the value 65 535 and -32 768 appears after 32 767. This progression must be taken into account in the user program.

When the R-input is High, the counter is reset into its initial position and the contents deleted. The Z-output of the counter is High, when the content of the counter equals 0. The Q-output always shows the counter actual value.

Note!

The pulse must at least be as long as one cycle, so that the counter can recognize every counting pulse. The counter must be able to recognize a subsequent Low of the signal in order to ensure automatic edge formation. The maximum counting frequency is therefore

$$F = \frac{1}{\text{cycle}}$$

Function Blocks

Up/Down Counter

Word

Example

The counter 11 is to count one step further each time I0.5 closes. I0.6 resets the counter. The actual count is indicated via the marker word MW12. Reverse counting and setting is not used.

The program:

C11

10.5 Forward pulse (Up)

10.6 Reset

MW12 Actual value

Cycle 1. 2. 3. 4. 5. 6. 7. 8. 9.10.11.12.13.14.15.16.17.18.19.20.21.22.23.24.25.

I 0.5 U .

I 0.6 R .

MW12 D ' .

r
i Xo

SyntaxCall-**up**:
CALARM 0Execution time: 20 μ s without subprogram (UP)
240 μ s + SP time with subprogram**Representation**

IL

CALARM 0

[]	EN	Enable/disable interrupt (0 = disable, 1 = enable)
[W]	VT	Predivider counter
[W]	SOLL	Setpoint value counter
[B]	ERR	Error output
[W]	CNT	Alarm counter (number of module call-ups)
[\$]	AC	Address of the subprogram which is to be called up

Description

This function block is assigned the I 0.0 hardware input in the basic unit (PS 4-201-MM1).

The EN input controls the start of the counter. The counter is started if the input is 1. When switching from 1 to 0, the counter is stopped and reset.

The VT input (values: 1 - 65535) indicates how many signals occur on the hardware input until the counter is incremented by 1. The SOLL input (1 - 65535) indicates after how many counted signals the alarm is to be enabled or the function block is to be called up.

The ERR output contains the code of the errors:

- 0 = no error
- 1 = setpoint value is 0
- 2 = predivider is 0

The CNT output (0 - 65535) indicates how often the setpoint value SOLL has been reached.

The AC input allows an event-driven program to be implemented. For this it is necessary to define the address (\$ name) of the subprogram that is to be executed when reaching the event. If no address is stated, only the CNT counter is incremented.

Description

The following points must be observed for the event-driven program:

- After the event has occurred, the user program is interrupted, the register status is saved and the subprogram stated under the AC address is executed. The alarm function block cannot be interrupted by other alarm function blocks (all alarm function blocks have the same priority).
- The max. execution time of alarm function blocks is restricted by the user program to 5 ms (approx. 1K IL instructions) since the alarm function blocks cannot be interrupted even by the operating system in the event of a voltage drop. If the execution time is exceeded, an EDC error may occur when switching off the power supply.
- The execution time of the alarm function block is added to the execution time of the cyclical user program and also monitored by the cycle time monitoring function.
- Since the event-driven program processing enables access to the entire image register, access to data that is used by the event-controlled and cyclical user program must be disabled. Bit accesses may not occur on the same byte in the cyclical user program and in the Alarm function block.
- Since an Alarm function block requires, due to its fast reactions, a high-speed peripheral access (direct output), the QB, QPB peripheral operands available in the basic unit should be used.
- An alarm function block can be used several times (multiple instantiation) although this should normally be avoided since each function block group has the same event source (hardware input I 0.0.0.0.0) and only the last function block instance in the program is valid.
- By multiple instantiation is meant the reservation of several data ranges for each parameter set of a particular function block type.

Example

In the following example the signals of a rotary encoder are counted. The time between the signals is shorter than the cycle time of the PLC. The encoder outputs 1000 signals per rotation degree. The divider ratio on the function block is to be set to 100 so that the rotation position is 1/10 degree on the CNT output.

Printout of the c:cala.q42 file Dated: 6. 4. 94

```
00000      BLOCK0  "Incorporate configuration file
001
002          #include "config.k42"
003
00001      BLOCK1  "Start of program
001
00002      BLOCK2  "Call up CALARM0 function block in order
001          "the signals via I 0.0 counter input
002
003          CALARM0
004          [ ] EN: I 0.5          Set function block
005          [w] VT:KW100
006          [w] SOLL KW 1
007          [b] ERR: MB25
008          [w] CNT: MW10
009          [$] AC:
010
00003      END      "End of program
001
002      HP
```


CK

Function Blocks Time/Date Comparator Clock

Syntax

Call-up:

CK < Block No. >

As operand:

CK < Block No. > u < Input/Output >

Number ranges	TIME:	1) Hours (0...23) 2) Point 3) Minutes (0...59)
	DAY:	(0...6) 0 = Sunday (SU) 1 = Monday (MO) 2 = Tuesday (TU) 3 = Wednesday (WE) 4 = Thursday (TH) 5 = Friday (FR) 6 = Saturday (SA)
	DATE:	1) Month (1...12) 2) Point 3) Day (1...31)
	VDATE:	Day. Month (1...31 .1...12)
	VTIM:	h . min (0...23 . 0...59)

Performance time: 200 |is

Representation

IL
CK10

[]	S:
[X]	TIME
[X]	DAY:
[X]	DATE
[W]	VDATE
[W]	VTIM
[]	GT:
[]	EQ:
[]	LT:
[]	ERR:

Truth table

	LT	EQ	GT	ERR
tREF < tACT	1	0	0	0
tREF = tACT	0	1	0	0
tREF > tACT	0	0	1	0
tREF invalid	0	0	0	1

Description

The time/date comparator function block scans the internal real-time clock which is battery-backed for when the controller is switched off. Time and date can be set or corrected by means of the SUCOsoft "Time/date" menu.

When in operation, the function block compares the preset values, such as time (hours, minutes), day (week-day), date (day, month) with the running real-time clock.

The values can be preset in two ways:

1. Via function block inputs TIME, DAY, DATE.

The data is entered via PC during programming. These constant values cannot be changed during processing. They are only valid when the inputs VDAT and VTIM have *no* defined operands.

The ERR output is Low.

Possible parameter settings.

TIME: 20.35 DAY: - DATE: -

TIME: 20.35 DAY: 1, DATE: -

TIME: 20.35 DAY: - DATE: 3.28

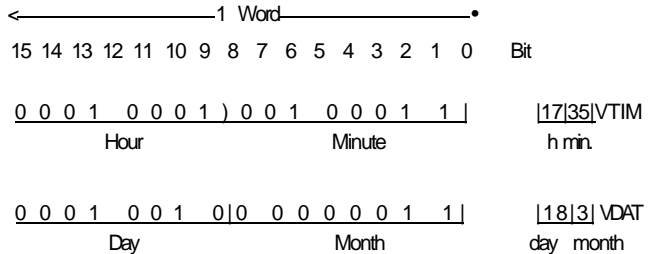
Function Blocks
Time/Date Comparator
Clock

Description

2. Via function block inputs VDAT, VTiM.

When VTIM or VTIM and VDAT have been set, the inputs TIME, DAY, DATE are no longer active. The settings in word format is carried out, as usual, by means of a function block call-up or in the user program.

e.g.		or
LKB35		LI 0.0
= MB10		= CK10S
LKB17		
= MB 11		LKB35
LKB3		= MB10
= MB12		LKB17
LKB18		= MB 11
= MB13		LKB3
		= MB12
CK10		LKB18
[]	S: 1 0.0	= MB13
[X]	TIME:	LMW10
[X]	DAY:	= CK10VTIM
[X]	DATE:	LMW12
[W]	VDAT: MW 12	= CK10VDAT
[W]	VTIM: MW 10	
[1	GT:	LCK 10 EQ
[]	EQ: Q 0.0	= 0.0.1
[]	LT:	LCK 10 ERR
[1	ERR: Q 0.1	= Q0.1
etc.		
		CK10
		etc.

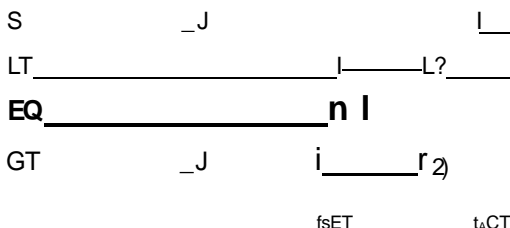


All possible values are acceptable for the marker words.
If invalid values are given, e.g. 25 hours, the ERR output is High.

Function

The function block is activated when the S input is High.
A Low signal sets all the outputs Low.

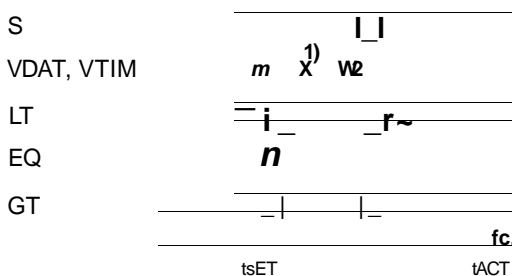
Example 1:



- EQ remains High for one minute when the set time has been reached.
- Change from "greater than" to "less than" depends on time preselect
of TIME or VTIM at 24 hours,
of DAY on Sunday 24 hours,
of DAT or VDAT at the end of the year at 24 hours.

Example 2

(Data change with ttle program running):



- Change of value: old = W1, new = W2
The new value is accepted in the next processing cycle of the function block in the user program.

Word

Syntax

Call-up:

CP < Block No. >

As operand:

CP < Block No. >u< Input/output >

Number ranges: -32768...0...+32767 decimal

Performance time: 105 us

Representation

IL

CP10

[W]

Value 1

[W] 12

Value 2

[] GT:

11 < 12

[] EQ:

11 = 12

[] LT:

11 > 12**Truth table**

	LT	EQ	GT
11 < 12	1	0	0
11 = 12	0	1	0
11 > 12	0	0	1

Description

The function block compares the values at the word inputs 11 and 12, then sets the outputs according to the truth table.

Code Converter:
 Decimal to Binary
 ~ ~ ~ ~ T ~ ~ ~ I Word

Syntax

Call-up:
DEB < Block No. >

As operand:
DEB < Block No. > u < Input/output >

Number ranges: 4 decades BCD
 Output range: -9999...0...+9999

Performance times: 145 μ s

Representation

IL
 DEB 47
 [] 11: Sign input
 [W] I: BCD input
 [W] Q: Binary output

Description

A 4-decade, BCD coded number is converted into a 16-bit, binary coded number. If the BCD value is to be converted into a positive binary value, the sign "IS" must be Low. For negative numbers the input must be High.

Example

A 4-decade BCD number received from a preselector switch is to be converted into a binary number for further processing. The input values are +11, +9999, -1311, -9999.

IL	I	Q	Q
0	0011	11	B
0	9999	9999	270F
1	13 11	-1311	FAE1
1	9999	-9999	D8F1

Syntax

Call-up:

FALARM 0

Execution time: 20 LIS without subprogram (SP)
240 us + SP time with subprogram

Representation

IL

FALARM 0

EN : Enable, disable interrupt (0 = disable, 1 = enable)
ACT : Rising edge (0 = positive, 1 = negative edge)
SOLL : Setpoint value counter
ERR : Error output
CNT : Alarm counter (number of function block call-ups)
At; : Address of the subprogram which is to be called up

Description

This function block is assigned the I 0.1 hardware input in the basic unit (PS 4-201-MM1).

The EN input controls the start of the function block. The counter is started if the input is 1. When switching from 1 to 0, the counter is stopped and reset.

The ACT input indicates at which edge a signal is to be counted. The SOLL input (1 - 65535) indicates after how many counted signals the alarm is to be enabled or the function block is to be called up.

The ERR output contains the code of the errors:

0 = no error

1 = setpoint value is 0

The CNT output (0 - 65535) indicates how often the setpoint value SOLL has been reached.

The AC input allows an event-driven program to be implemented. For this it is necessary to define the address (\$ name) of the subprogram that is to be executed when reaching the event. If no address is stated, only the CNT counter is incremented.

Description

The following points must be observed for the event-driven program:

- After the event has occurred, the user program is interrupted, the register status is saved and the subprogram stated under the AC address is executed. The alarm function block cannot be interrupted by other alarm function blocks (all alarm function blocks have the same priority).
- The max. execution time of alarm function blocks is 5 ms (approx. 1K IL instructions) since the alarm function blocks cannot be interrupted even by the operating system in the event of a voltage drop. If the execution time is exceeded, an EDC error may occur when switching off the power supply.
- The execution time of the alarm function block is added to the execution time of the cyclical user program and also monitored by the cycle time monitoring function.
- Since the event-driven program processing enables access to the entire image register, access to data that is used by the event-controlled and cyclical user program must be disabled. Bit accesses may not occur on the same byte in the cyclical user program and in the Alarm function block.
- Since an Alarm function block requires, due to its fast reactions, a high-speed peripheral access (direct output), the QB, QPB peripheral operands available in the basic unit should be used.
- An alarm function block can be used several times (multiple instantiation) although this should normally be avoided since each function block group has the same event source (hardware input I 0.0.0.0) and only the last function block instance in the program is valid.
- By multiple instantiation is meant the reservation of several data ranges for each parameter set of a particular function block type.

Example

Printout of the c:fala.q42 file Dated 6. 4. 94

```
00000      BLOCK0      "Incorporate configuration file
001
002                          #include "conflg.k42"
003
00001      BLOCK1      "Start of program
001
00002      BLOCK2      "In the FALARMO function block the water level of
001      "a tank is controlled via the 10.1 alarm input and
002      "calls up the UP0 subprogram when reaching a
003      "level mark (positive edge on the alarm input).
004
005
006                          FALARMO
007      [ ] EN: I 0.2 Enable alarm function block
008      [ ] ACT: K 0
009      [w] SOLL: KW 1
010      [b] ERR:MB22
011      [w] CNT: MW124
012      [$] AC:$UP0
013
014      "UP0 subprogram is called up when the
015      "edge is received on the alarm input.
016
00003      END          "End main program
001
002                          EP
003
00004      SUP0          "Subprogram 0
001
002                          L...
003

0..          EM
```

FIFOB

Function Blocks

First In - First Out

Byte

Syntax

Call-up:

FIFOB < Block No. > - < Register length > - R

- R only if retentive operation is required.

As operand:

FIFOB < Block No. >u< Input/output >

Register length: Optional 1...128

Performance time: 265 us

Representation

IL

FIFOB 57-60-R

CF:	Fill pulse
CF:	Read out pulse
R:	Reset
I:	Data input
F:	Register full
E:	Register empty
Q:	Data output

Truth table

Inputs/outputs Type Function	CF Bit	CE Bit	R Bit	I Byte	Q Byte
Fill	<u>r</u>	0	0	value	x
Read out	0	<u>r</u>	0	x	value
Reset	x	x	1	x	0

Description

With a rising edge at the "CF" input, any value at the "I" input is entered at the beginning of the FIFO.

A rising edge at the "CE" input copies the first value into the output Q and all the values in the register are shifted forward one step. The outputs "E" and "F" use a High signal to indicate if the FIFO memory is either empty or full.

The FIFO memory is reset into the initial state and deleted when the R-input is High.

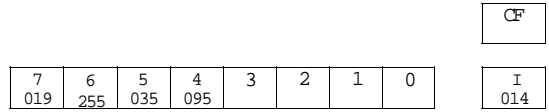
Function Blocks

First In - First Out

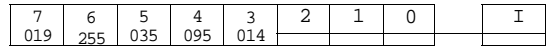
FIFOB

Byte

Examples

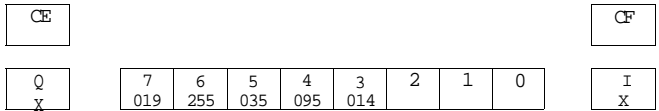


CE

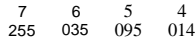


First in - first out register

An input byte is taken from I to the topmost free location with the rising edge at "CF"



5



First in - first out register

The lowest byte is transferred from the first in - first out to memory output Q with the rising edge at "CE".

FIFOW

Function Blocks

First In - First Out

Word

Syntax

Call-up:

FIFOW < Block No. > - < Register length > - R

- R only if retentive operation is required.

As operand:

FIFOW < Block No. > u < Input/output >

Register length: Optional 1 ...128

Performance time: 265 |.ts

Representation

IL

FIFOW 57-60-R

[CF:	Fill pulse
[CE:	Read out pulse
[R:	Reset
[W	I:	Data input
[F:	Register full
[E:	Register empty
[W	Q:	Data output

Truth table

Input/output Type Function	CF Bit	CE Bit	R Bit	I Word	Q Word
Fill	$_r$	0	0	value	x
Read out	0	$_r$	0	x	value
Reset	x	x	1	x	0

Description

With a rising edge at the "CF" input, any value at the "I" input is entered at the beginning of the FIFO.

A rising edge at the "CE" input copies the first value into the output Q and all the values in the register are shifted forward *one* step. The outputs "E" and "F" use a High signal to indicate if the FIFO memory is either empty or full.

The FIFO memory is reset into the initial state and deleted when the R-input is High.

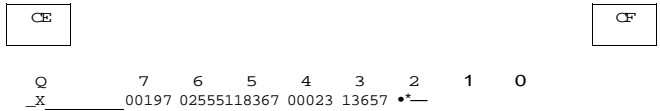
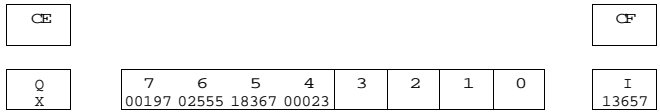
Function Blocks

First In - First Out

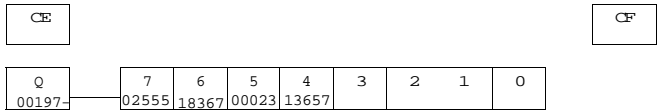
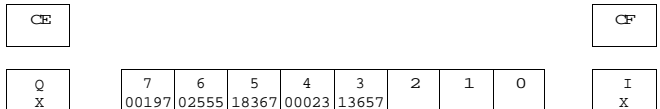
FIFOW

Word

Examples



First in - first out register
An input word is taken from I to the topmost free location when the rising edge is at "CE".



First in - first out register
The lowest word is transferred from the first in - first out to the memory output Q when the rising edge is at "CE".

Byte**Syntax**

Call-up:

ICP < Block No. > - R

- R if retentive operation is required.

As operand:

ICP < Block No. >u< Input/output >

Number of

elements: 1...255

Performance time: The performance time of this function block heavily depends on the type of source and destination operands involved.

Standard values:

MOD0: $(244 + 7 \times n)$ usMOD 1: $(264 + 20 \times n)$ us

n = number of elements

Representation

```

IL
CP 5
[ ] MOD:
[&] SADR:
[&] DADR:
[B] NO:
[ ] GT:
[ ] EQ:
[ ] LT:
[B] Q:
[B] ERR:

```

Note!

A detailed description of the function block "ICP" is given in Chapter 8, Indirect Addressing.

Byte

Description

Inputs:

- MOD Operating mode
 = 1 Compare data fields
 = 2 Search for data value
- SADR Source address
 Start address of the source data block from
 which the comparison is to be made
- DADR Destination address
 Destination address from which the comparison
 is to be made
- NO Number of elements
 1-255 to be compared (depending on data type
 Of SADR/DADR)

Outputs:

Note:

Comparisons are not signed

- GT Greater than
 = 1 Data value in SADR > data value in DADR
- EQ Equal
 = 1 Data values are identical
- LT Less than
 = 1 Data value in SADR < data value in DADR
- Q Output
 indicates the relative offset address of the
 unequal value (comparison) or of the found data
 value (data value search).
 The offset is determined from the beginning of
 the block (DADR) and is dependent on the data
 type in DADR. The calculation of the offset is
 restricted to the following limits: $0 < Q < NO$.
- ERR = 0 Data limits are permissible
 = 1 NO is 0
 = 2 SADR has not been defined
 = 3 DADR has not been defined
 = 4 SADR is the same as DADR

Byte

Description

The function block has a data search or block compare mode. The coding on the MOD input determines whether a comparison or a data value search is to be carried out.

Search mode

Search mode is used to search for a particular value in a data block. The compared value is located at address SADR (source address). The start of the data block to be examined is specified by DADR (data address). The address is prefixed by the address operator "&".

If the value defined under SADR is found within the NO elements starting from the DADR address, the location is indicated via output Q and the output EQ is set (=1).

The following applies when a character is found:

$Q = 0 \dots NO - 1$; $EQ = 1$; $LT = GT = 0$;

If the data value is not found in the block, the output Q is equal to NO. The EQ output is set to 0 and the outputs LT and GT are set according to the last comparison.

The following applies when a character is not found:

$Q = NO$; $EQ = 0$; LT, GT according to the last comparison

In the following example the value 7D in MB 23 is searched for in the marker field from MB 27 to MB 32.

Function block

Marker field:

ICPO				
[]	MOD: KO	MB 23	7D	
[&]	SADR:&MB 23	MB 24	00	
M	DADR:&MB 27	MB 13	00	
	NO: KB6	MB 26	00	Search
	GT:	MB 27	3D	
	EQ:	MB 28	7D	
	IT:	MB 29	4D	
[til	0:	MB 30	7D	
[B]	ERR:	MB 31	7D	Found: therefore
Result		MB 32	70	search routine
Q = 3		MB 33	00	aborted
LT = 0		MB 34	00	
EQ = 1				
LT = 0				

The data value 7D was found at the address DADR + Q (here: MB27 + 3) and the search was terminated.

Byte

Compare mode

The block compare mode is used to compare two data blocks with each other. The start of both blocks is defined by SADR and DADR. The size of the block is specified by the number of elements NO. If both data blocks are found to be equal, the Q output equals NO and the EQ output is set to 1.

The following applies when the compared data blocks are equal:

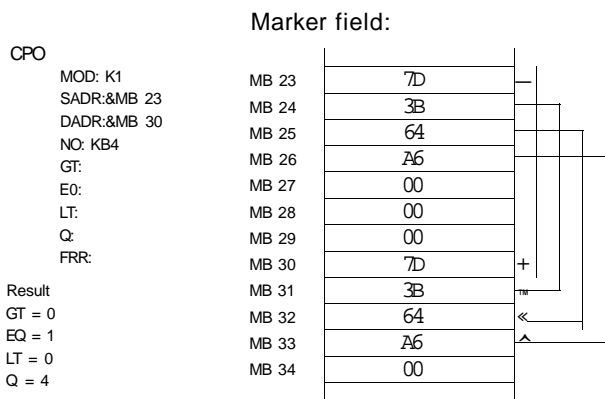
$Q = NO; EQ = 1; LT = GT = 0;$

If the compared data blocks are not equal, output Q indicates the location of the unequal data. The EQ output is set to 0 and the LT and GT output are set according to the result of the comparison (either 1 or 0).

The following applies when the compared data blocks are not equal:

$Q = 0 \dots NO-1; EQ = 0; LT, GT$ depending on the result of the last comparison.

In the following example the marker field from MB 23 to MB 26 is compared with the marker field from MB 30 to MB 33.



The two data blocks are identical. This is indicated by $EQ = 1$ and $Q = NO$ (run completed).

ICPY

Function Blocks

Block Transfer

Indirect Copy

Byte

Syntax

Call up:

ICPY < Block No. > - R

- R, if retentive operation is required.

As operand:

ICPY < Block No. >u< Input/output >

Number of

elements: 1...255

Performance time: The execution time of this function block heavily depends on the type of source and destination operands involved.

Standard values:

MOD0: $(355 + 25 \times n)$ *µs*

MOD 1: $(355 + 10 \times n)$ *µs*

n = Number of elements

Representation

IL

ICPY 63

[] MOD:

[&] SADR:

[&] DADR:

[B] NO:

[B] ERR:

Note!

A detailed description of the function block "ICPY" block transfer is given in Chapter 8, Indirect Addressing.

Block Transfer

Indirect Copy

Byte

Description

Inputs:

MOD Initialize/Copy mode

= 1 Copy data fields

= 2 Initialize data fields

SADR Source address of source data block from which the transfer is to begin

DADR Destination address

Destination address to which the source data is to be transferred or from where initializing is to begin

NO Number of elements to be transferred 1-255 (depending on data type SADR/DADR)

Outputs:

ERR 0 Data limits are permissible

1 NOisO

2 SADR has not been defined

3 DADR has not been defined

4 SADR is the same as DADR

The ICPY function block supports the transfer of data blocks within the system. A transfer is always made from a "source" to a "destination". Markers M, communications data RD/SD and the address inputs of other function blocks are permitted as operands for the address operator "&".

The function block can be used in the copy mode and the initialize mode which are selected by setting a 1 or a 0 at the MOD input (1 or 0). The differentiation between address and data is important with this function block. With typical operations such as L M 2.2, it is always the data that is stored, in this case in the marker cell, which is accessed. In the case of the block transfer, the source address SADR from which the copying is to be made and the destination address DADR must be specified. The address operator "&" must be used here. This signifies that the operand behind it is an address and not a data value.

ICPY

Function Blocks

Block Transfer

Indirect Copy

Byte

Copy mode

The number of data cells specified by the NO value are copied from the source address specified by SADR to the destination address specified by DADR.

In the following example the data from the marker fields MB 23 to MB 26 is copied to marker field MB 30 to MB 33.

IL	Marker field:
ICPY 0	
I] MOD: K1	MB 23 ?0
[&] SADR:&MB 23	MB 24 3B
[&] DADR:&MB 30	MB 25 64
[B] NO: KB4	MB 26 A6
[B] ERR:	MB 27 00
	MB 28 00
	MB 29 00
	MB 30 7D
	MB 31 3B
	MB 32 64
	MB 33 A6
	MB 34 00

Example of the copy mode of the ICPY function block

Byte

Initialize mode

This involves a transfer of the data (byte or word) stored under address SADR in a number of data cells specified by NO, beginning with the DADR destination address.

In the following example the marker field from MB 27 to MB 32 is initialized with the data value 7Dh which is stored in MB 23.

IL:		Marker field:	
ICPYO			
[]	MOD: KO	MB 23	<i>iD</i>
[&]	SADR:&MB 23	MB 24	00
[&]	DADR:&MB 27	MB 13	00
[B]	NO: KB6	MB 26	00
[B]	ERR:	MB 27	<i>/D</i>
		MB 28	70
		MB 29	70
		MB 30	<i>iV</i>
		MB 31	<i>iV</i>
		MB 32	70
		MB 33	00
		MB 34	00

Example of initialize mode of the ICPY function block

LIFOB

Function Blocks

Last In - First Out
Stack Register
Byte

Syntax

Call-up:

LIFOB < Block No. > - < Register length > - R

- R only if retentive operation is required.

As operand:

LIFOB < Block No. > u < Input/output >

Register length: Optional 1...128

Performance time: 255 us

Representation

IL
LIFOB 8-40-R

CF: Fill pulse
CE: Read out pulse
R: Reset
I: Data input
F: Register full
E: Register empty
O: Data output

Truth table

Inputs/outputs Type Function	CF Bit	CE Bit	R Bit	I Byte	Q Byte
Fill	<u>r</u>	0	0	Value	x
Read out	0	<u>r</u>	0	x	Value
Reset	x	x	1	x	0

Description

With rising edge at the "CE" input any value at the "I" input is entered in the stack. A rising edge at the "CE" input copies the value at the top of the stack into the output "Q". When High, the "E" and "F" outputs indicate respectively whether the stack is empty or full. The stack memory is reset into the initial state, and deleted when the "R" input is High.

Function Blocks

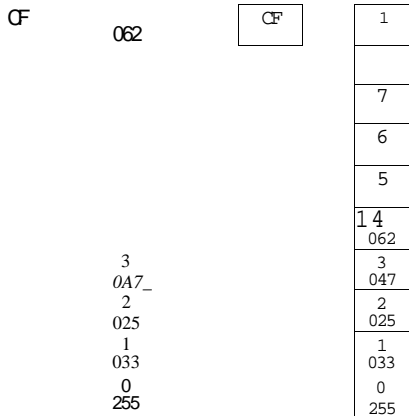
LIFOB

Last In - First Out

Stack Register

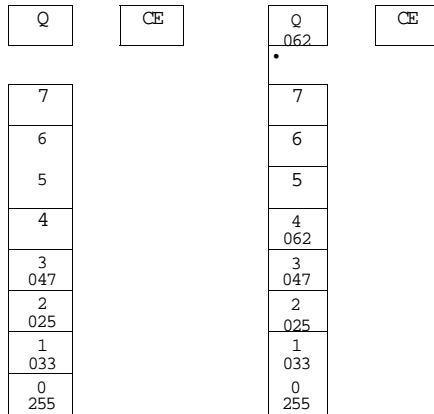
Byte

Examples



Last in - first out (stack register)

The input byte is entered on the stack on "1" when the rising edge is at "CF".



Last in - first out (stack register)

The topmost byte is transferred from the stack to the output "Q" when the rising edge is at "CE".

LIFO

Function Blocks

Last In - First Out
Stack Register

Word

Syntax

Call-up:

LIFO < **Block No.** > - < **Register length** > - **R**

- R only if retentive operation is required.

As operand:

LIFO < **Block No.** > < **Input/output** >

Register length: Optional 1 ...128

Performance time: 265 us

Representation

IL

LIFO 8-40-F

	CF:	Fill pulse
	CE:	Read out pulse
	R:	Reset
[W]	I:	Data input
	F:	Register full
	E:	Register empty
[W]	Q:	Data output

Truth table

Inputs/outputs Type Function	CF Bit	CE Bit	R Bit	I Word	Q Word
Fill	$_r$	0	0	Value	x
Read out	0	$_r$	0	x	Value
Reset	x	x	1	x	0

Description

With a rising edge at the "CF" input any value at the "I" input is entered on the stack. A rising edge at the "CE" input copies the value at the top of the stack into the output "Q".

When High, the "E" and "F" outputs indicate respectively when the stack is empty or full. The stack memory is reset into the initial state, and deleted when the "R" input is High.

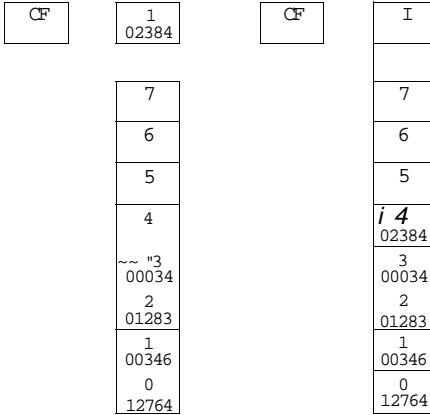
Function Blocks

Last In - First Out

Stack Register

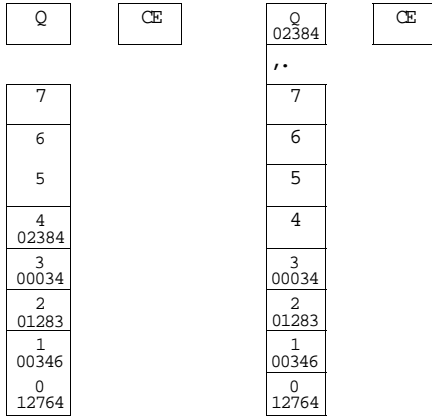
		Word	
--	--	------	--

Examples



Last in - first out (stack register)

The input word is entered on the stack on "I" when the rising is at "CF"



Last in - first out (stack register)

The topmost word is transferred from the stack to the output "Q" when the rising edge is at "CE".

Syntax

Call-up:

RDAT < Block No. >

Execution time: (330 + 25 x n) (iS
n = number of elements)

Representation

IL

RDAT1

[]	S:	Set input
[&]	DADR:	Destination address (address operator)
[W]	SGNO:	Segment number (0-511)
[B]	LEN:	Segment length (1-128)
[B]	ERR:	Error output

Description

Any data which is already stored in the SDAT function block can be written via the RDAT function block out of the memory module to the corresponding data range. This data can be saved if a memory module is fitted on the PS 4 200 series which reserves 64 Kbytes for the cold-start retentive range. This memory is logically divided into 512 segments of 128 bytes each.

Another important application for the RDAT and SDAT function blocks is the required saving of the retentive marker range (with cold start) before modifying the device configuration. You will find further information in Chapter 1, System Parameters.

The RDAT function block is designed for programming with indirect addressing. See Chapter 8, Indirect Addressing.

Error messages

The ERR output contains the code of possible errors:

- no memory module present
- access not possible due to online connection
- access not possible due to simultaneous use of SDAT
- SGNO is greater than 511
- LEN is greater than 128
- DADR parameters incorrectly set

Example

```
RDAT 12
|  | S:      I 0.3
| & | DADR:   &MB23
| fW | SGNO:   KW5
| I B | LEN:   KB 127
| I B | ERR:   MB 12
```

If I 0.0.0.0.3 changes to 1, the data stored in the memory module on segment number 5 is rewritten to the marker range starting from the address MB 23. Errors are saved in marker byte 12.

SCK

Function Blocks

Set Real-Time Clock
Set Clock

Syntax

Call-up:

SCK < **Block No.** >

Execution time: 790 us

Representation

IL

SCKO

[] S: Set

[&] SADR: Data address (indirect)

[B] ERR: Error byte

Description

If the input S changes to 1, the clock is set again with the values stated under SADR. The clock setting information is stated indirectly via the SADR address operand.

The length of the block must not be stated since a standard 7 bytes (year, month, day, week day, hour, minute, second) are transferred. This means that the clock will only function if all clock parameters are defined.

The SCK function block is designed for programming with indirect addressing. See Chapter 8, Indirect Addressing.

Error messages

The ERR output contains the code of possible errors:

1 = SADR parameters incorrectly set

2 = Incorrect year stated (0-99)

3 = Incorrect month stated (1-12)

4 = Incorrect day stated (1-31)

5 = Incorrect week day stated (0-6, 0 = Sunday)

6 = Incorrect hour stated (0-23)

7 = Incorrect minute stated (0-59)

8 = Incorrect second stated (0-59)

Example

```

00000      BLOCK1      "Define new parameters for the SCKO function
001          "block to set the real-time clock.
002
003
004          LKB94          Year (19)94
005          = MB10
006
007          LKB3
008          = MB11          Month: March
009
010          L KB 27
011          = MB12          Day: 27.
012
013          LKB0
014          = MB13          Weekday: (Sunday)
015
016          LKB3
017          = MB14          Hour: 3
018
019          LKB0
020          = MB15          Minute: 0
021
022          LKB0
023          = MB16          Second: 0
024
00001      BLOCK2      "Set real-time clock with SCK 0 function block
001
002
003
004          SCKO
005          [ ] S:M0.0          Set SCKO
006          [&] SADR:&MB10
007          [b] ERR:MB20
008
00002      END          "End of program
001
002          EP

```

This example shows how a marker range is defined with the required data for setting the real-time clock and how the clock is set with the SCKO function block.

SDAT

Function Blocks

Save Data

Syntax

Call-up:

SDAT < Block No. >

Execution time: $(330 + 25 \times n) \text{ } \mu\text{s}$
n = number of elements

Representation

IL

SDAT1

[]	S:	Set input
[&]	SADR:	Source address (address operator)
[W]	SGNO:	Segment number (0-511)
[B]	LEN:	Segment length (1-128)
[B]	ERR:	Error output

Description

Any data can be saved in the memory module of the PS 4 200 series if this memory module has 64 Kbytes for the cold-start retentive range. This memory is logically divided into 512 segments of 128 bytes each. The data can be reloaded to the data range concerned via the RDAT function block.

Another important application for the RDAT and SDAT function blocks is the required saving of the retentive marker range (with cold start) before modifying the device configuration. You will find further information in Chapter 1, System Parameters.

The SDAT function block is designed for programming with indirect addressing. See Chapter 8, Indirect Addressing.

Error messages

The ERR output contains the code of possible errors:

- 1 no memory module present
- 2 access not possible due to online connection
- 3 access not possible due to simultaneous use of RDAT
- 4 SGNO is greater than 511
- 5 LEN is greater than 128
- 6 SADR parameters incorrectly set

Example

SDAT 12		
	S:	I 0.3
[&	SADR:	&MB23
fW	SGNO:	KW5
f B	LEN:	KB 127
[B	ERR:	MB 12

If I 0.3 changes to 1, the data stored in the memory module on segment number 5 is rewritten to the marker range starting from the MB 23 address. Errors are saved in marker byte 12.

Syntax

Call-up:

SK < Block No. > - < Number of steps > - R

- R only if retentive operation is required.

As operand:

SK < Block No. > < Input/output >

Step number 1...99

Nesting depth: 8

Executing time:	with SET = RESET = 0	approx. 130 u.s
	with RESET = 1	approx. 240 ns
	with SET = 1	approx. 250 [is
	with invalid SINO	approx. 150 u.s

ion

IL

SK 3-14

S:	Set
R:	Reset
[B SINO:	Step input number
[B ERR:	Error output
[B SQNO:	Step number display
TG:	Step change indication
[S INIT:	Step call-up after reset
[\$ AC1:	Step 1 (action program 1)
[\$ AC2:	Step 2 (action program 2)
[S AC3:	Step 3 (action program 3)

[S] AC14: Step 14 (action program 14)

Note:

Refer to Chapter 7 for a detailed description of the sequential control function block.

Description

The sequential control function block enables the user program to be structured simply and clearly.

Each SK function block can control up to 99 steps. Each step can itself activate other sequences, allowing a nesting depth of 8. The individual steps are formed by means of subprograms which contain the actions to be executed. The logical structure of the step sequence control can therefore be incorporated directly in the user program via the function block.

The inputs and outputs of the function block have the following meaning:

S Set

Activate the sequential control function block

R Reset

This resets the function block and the initialisation program is activated via the INIT input.

SINO Step Input Number

The number of the current step is assigned to this input.

ERR Error

Display of faulty conditions.

Error number:	Cause or error:	Error behaviour:
Binary: 00000001 or decimal: 1	The SINO input shows the value 0 (S=1)	The function block is passive. No step is processed.
Binary: 00000010 or decimal: 2	Step number exceeds the maximum possible step number	Error output is set. The function block remains in its current status.
Binary: 00000100 or decimal: 4	No subprogram on selected AC output	Error output is set. The selected step is transferred. Nothing is processed since there is no action program present.

SQNO Step Output Number

The SQNO output indicates the number of the step currently being processed.

TG Toggle outputs

The TG output indicates the transition to a new step. In normal operation this output is High, only in the first cycle is it Low after a change.

INIT Initialisation

Name of the initialisation subprogram which runs on activation of a reset.

Example: "\$INIT"

AC Action

Name of the current step subprogram.

Example: "\$STEP1"

Bit

Syntax

Call-up:

SR < Block No. > - < Register length > - R

- R only if retentive operation is required.

As operand:

SR < Block No. > u < Input/output >

Register length: Optional 1...128

Performance time: typ.: 148 us + (n-1) x 20 (is
n = register length)

Representation

IL

SR 54-13-R

[]	U:	Pulse input forward
[]	D:	Pulse input reverse
[]	R:	Reset
[]	IU:	Data input forward
[]	ID:	Data input reverse
[]	Q0:	Output 0

[]	Q12:	Q (n-1) Last output
		t— Register length

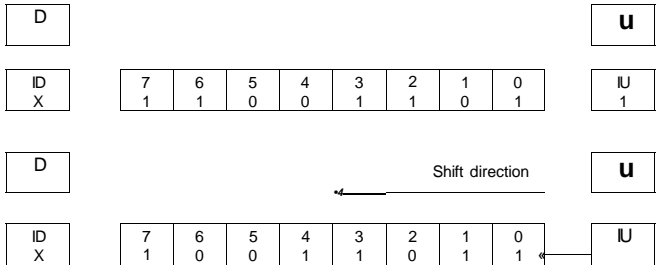
Truth table

Inputs Type Function	U Bit	D Bit	R Bit	IU Bit	ID Bit
Shift, forward	<u>r</u>	0	0	Value	x
Shift, reverse	0	<u>r</u>	0	x	Value
Delete	x	x	1	x	x

Bit

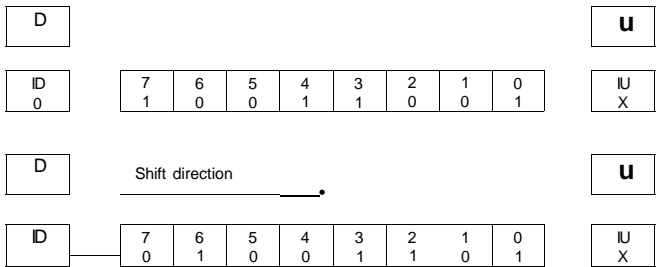
Description

When the rising edge is at the "U" input, the value of the "IU" input is entered in the first register field after all other register fields have been shifted in the positive direction by one step.



Bit shift register eight steps long; one forward pulse and acceptance of an input bit from IU.

A rising edge at the D input causes the entry of the "ID" value into the last register field after the contents of all other registers fields have been shifted in the negative direction by one step.



Bit shift register eight steps long; one reverse pulse and acceptance of a bit from ID into the last register field.

Bit

Description

The contents of all register fields are externally displayed via the Q outputs. When the "R" input is High, the shift register is reset to the initial state, and all the register fields are cleared. If there is a rising edge at U and D simultaneously, a forward shift is performed first before a reverse shift.

The register length is restricted to 128 register fields. Several shift registers can be linked together if more than 128 shift steps are required.

Example

SR 14-8		
U:	10.1	Pulse forward
D:	I 0.2	Pulse reverse
R:	I 0.3	Reset
IU:	I 0.4	Data input forward
ID:	SR15Q0	Data input reverse
00:	= M 100.0	Shift register output
01:	= M 100.1	
02:	= M 100.2	
03:	= M 100.3	
04:	= M 100.4	
0b:	= M 100.5	
06:	= M 100.6	
07:	= M 100.7	Shift register output
SR 15-8		
U:	10.1	Pulse forward
D:	I 0.2	Pulse reverse
R:	I 0.3	Reset
IU:	SR 14 Q7	Data input
ID:		
00:	= M 101.0	Shift register output
01:	= M 101.1	
02:	= M 101.2	
03:	= M 101.3	
04:	= M 101.4	
05:	= M 101.5	
06:	= M 101.6	
0/:	= M 101.7	Shift register output

Bit		
-----	--	--

The coupling of two bit-shift registers of 8 steps each to form one register of double bit length (16 steps).

(In order to simplify the diagram, two eight-bit registers have been connected together in the example. In practice shift register 14 would have been pre-selected from the outset as a 16-step register.)

Byte

Syntax

Call-up:

SRB < Block No. > - < Register length > - R

- R only if retentive operation is required.

As operand:

SRB < Block No. >_u < Input/output >

Register length: Optional 1...128

Performance time: typ.: $148 \text{ } \mu\text{s} + (n-1) \times 20 \text{ } \mu\text{s}$
n = register length

Representation

IL

SRB 54-13-R

[]	U:	Pulse input forward
[]	D:	Pulse input reverse
[]	R:	Reset
[]	IU:	Data input forward
[]	ID:	Data input reverse
[]	0.0:	Output 0

[] Q27: Q (n-1) Last output
t— Register length

Truth table

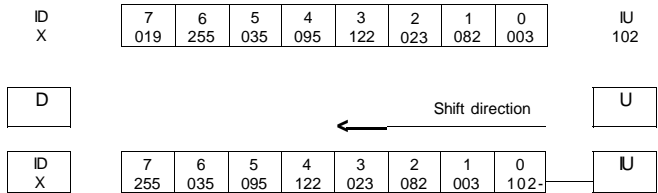
Inputs Type Function	U Bit	D Bit	R Bit	IU Byte	ID Byte
Shift, forward	<u>r</u>	0	0	Value	x
Shift, reverse	0	<u>r</u>	0	x	Value
Delete	x	x	1	x	x

Shift Register

Byte

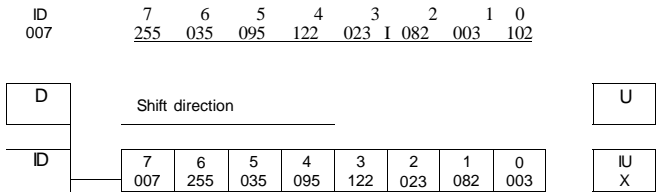
Description

When the rising edge is at the "U" input, the value of the "IU" input is entered in the first register field after all other register fields have been shifted in the positive direction by one step.



Byte shift register eight steps long; one forward pulse and acceptance of an input byte from IU.

A rising edge at the D input causes the entry of the "ID" value into the last register field after the contents of all other register fields have been shifted in the negative direction by one step.



Byte shift register eight steps long; one reverse pulse and acceptance of a byte from ID into the last register field.

The contents of all the register fields are externally displayed via the Q outputs. When the "R" input is High, the shift register is reset to the initial state, and all register fields are cleared. If there is a rising edge at U and D simultaneously the forward shift is carried out before the reverse shift.

The register length is limited to 128 register fields. If longer shift registers are required, several registers can be coupled (see SR bit shift register).

Word

Syntax

Call-up:

SRW < Block No. > - < Register length > - R

- R only if retentive operation is required.

As operand:

SRW < Block No. >u< Input/output >

Register length: Optional 1...128

Performance time: typ.: 158 LIS + (n-1) x 65 LIS
n = register length

Representation

IL

SRW115-R

[] U: Pulse input forward

[] D: Pulse input reverse

[] R: Reset

[W] IU: Data input forward

[W] ID: Data input reverse

[W] QQ: Output 0

[W] Q114: Q(n-1) Last output
— Register length

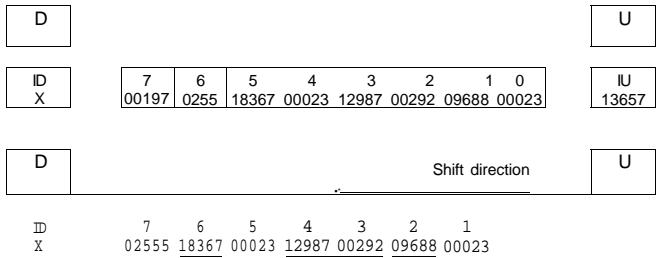
Truth table

Inputs Type Function	U Bit	D Bit	R Bit	IU Word	ID Word
Shift, forward	<u>r</u>	0	0	Value	x
Shift, reverse	0	<u>r</u>	0	x	Value
Delete	x	x	1	x	x

Word

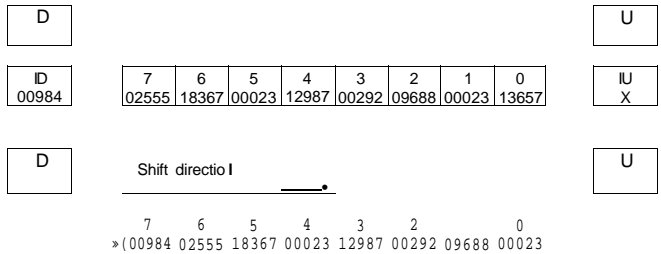
Description

When the rising edge is at the "U" input, the value of the "IU" input is entered in the first register field after all other register fields have been shifted in the positive direction by one step.



Word shift register eight steps long; one forward pulse and acceptance of an input word from IU.

A rising edge at the D input causes the entry of the "ID" value into the last register field after the contents of all other register fields have been shifted in the negative direction by one step.



Word shift register eight steps long; one reverse pulse and acceptance of a word from ID into the last register field.

The contents of all register fields are externally displayed via the Q outputs. When the "R" input is High, the shift register is reset into the initial state, and all the register fields are deleted. If there is a rising edge at U and D simultaneously the forward shift is carried out before the reverse shift.

The register length is limited to 128 register fields. Several shift registers can be coupled, if more shift steps are required.

TALARM

Function Blocks

Time Alarm Function Block
Timer Alarm

Syntax

Call-up:
TALARM 0

Execution time: 20 μ s without subprogram (SP)
240 jxs + SP time with subprogram

Representation

IL

[1	EN	Enable/disable interrupt (0 = disable, 1 = enable)
[B]	MOD	Mode 1 = timer, mode 2 = signal generator
[W]	VT	(Predivider) signal length
[W]	SOLL	Set number of signals
[B]	ERR	Error output
[W]	CNT	Alarm counter (number of module call-ups)
[\$]	AC	Address of subprogram which is to be called up

Description

The EN input controls the start of the function block. If this input is 1, the function block is started, the subprogram address, the VT and SOLL are accepted and temporarily stored. CNT is reset. The function block is stopped and reset when switching from 1 to 0.

The function block is a timer in the operating mode MOD = 1. The time is then defined in μ s by the VT input. SOLL defines the setpoint value of the timer. After the set number of signals has been completely processed, the CNT counter is increased and, if required, a subprogram is called up via \$AC.

The ERR output contains the code of the errors:

- 0 = no error
- 1 = SOLL = 0
- 2 = VT less than 512
- 3 = MOD may only be 1 or 2

In the operating mode MOD = 2 the function block is a signal generator which is connected with the Q 0.0 hardware output of the basic unit (PS 4-201-MM1). The VT input then defines the signal length/time base in μ s. SOLL defines the half of the set number of signals (= sum of rising and falling edge) on Q 0.0. QP 0.0 is output completely and CNT counts how often SOLL has been reached.

The ERR output contains the code of errors:

- 0 = no error
- 1 = SOLL = 0
- 2 = VT less than 512
- 3 = MOD may only be 1 or 2

The AC input allows an event-driven program to be implemented. For this it is necessary to define the address (\$ name) of the subprogram that is to be executed when reaching the event. If no address is stated, only the CNT counter is incremented.

The following points must be observed for the event-driven program:

- After the event has occurred, the user program is interrupted, the register status is saved and the subprogram stated under the AC address is executed. The alarm function block cannot be interrupted by other alarm function blocks (all alarm function blocks have the same priority).
- The max. execution time of alarm function blocks is restricted by the user program to 5 ms (approx. 1K IL instructions) since the alarm function blocks cannot be interrupted even by the operating system in the event of a voltage drop. If the execution time is exceeded, an EDC error may occur when switching off the power supply.
- The execution time of the alarm function block is added to the execution time of the cyclical user program and also monitored by the cycle time monitoring function.
- Since the event-driven program processing enables access to the entire image register, access to data that is used by the event-controlled and cyclical user program must be disabled. Bit accesses may not occur on the same byte in the cyclical user program and in the Alarm function block.

TALARM

Function Blocks

Time Alarm Function Block

Timer Alarm

Description

- Since an Alarm function block requires, due to its fast reaction, a high-speed peripheral access (direct output), the QB, QPB peripheral operands available in the basic unit should be used.
- An alarm function block can be used several times (multiple instantiation) although this should normally be avoided since each function block group has the same event source (hardware input I 0.0.0.0.0) and only the last function block instance in the program is valid.
- By multiple instantiation is meant the reservation of several data ranges for each parameter set of a particular function block type.

Example

The following example shows the call-up of the timer alarm function block as a signal generator (MOD = 2). The time/signal length is 1 ms (VT = 1000). The output on Q0.0 is shown below (a total of 50 signals are output):

 1 ms 1 ms
Q0.0 1 ms

Printout of the c:tala.q42 file Dated: 7. 4. 94

```
00000      BLOCK0      "Incorporate configuration file
001
002                      #include "config.k42"
003
00001      BLOCK1      "Call-up the timer alarm function block
001
002                      TALARM0
003                      [ ] EN: I 0.3                      Set function block
004                      [b] MOD: KB 2
005                      [w] VT:KW1000
006                      [w] SOU: KW 100
007                      [b] ERR:MB25
008                      [w] CNT:MW12
009                      [$] AC:
010
00002      END          "End of program
001
002                      EP
```

Syntax

Call-up:

TF < Block No. > - R

- R only if retentive operation is required.

As operand:

TF < Block No. > < Input/output >

Time ranges: 1...65 535 ms

* 10...65535 ms

Performance time: 300 \wedge s

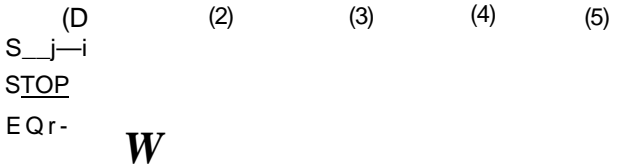
Basic time: - ms milliseconds
 - s for seconds

Representation

IL

TF 7-ms-R

- [] S: Start and set
- [] R: Reset
- [] STOP: Interruption for timer
- [W] 1: Set time value
- [] EQ: Control output
- [W] Q: Actual time value



Description

With the rising edge of "S" the delay factor at "I" is accepted and the "EQ" output is set High. If the "S" input is Low, the "EQ" output follows, delayed by the time "T" (1).

$T = \text{delay factor} \times \text{time base}$

The set time should always be greater than the cycle time to ensure that the delay is always detected. The output "Q" shows the time elapsed in units of the selected time base.

The time count can be interrupted via an "H" signal at the "STOP" input; i.e. the delay time "T" is extended by the time for which the "STOP" input is High (2), (3).

The EQ output only follows the S output if the STOP input is Low (4), (5), when the set input is High.

The timer is reset into the initial state if the "R" input is High.

Example

The timer 56 is to indicate the dropping-out of input 10.2 - delayed by the time constant $T = 50\,000$ - to the output Q 0.3 In order to enter values greater than 32 767, calculate the corresponding HEX or signed value. It should be possible to stop the measurement of the delay time via I 0.3 if there is an external event. The elapsed time is indicated via the marker word MW 10. The delay time in progress must be continued even after a power loss, or after switching off the system and subsequent restarting. The timer therefore has to be retentive.

The program:

Option 1

```
TF 56-S-I1
S: 10.2
R:
STOP: 10.3
I: KW-1
EQ: Q0.3
O: MW10
```

Option 2

```
TF 56-S-I1
S: 10.2
R:
STOP 10.3
I: KH-W C350
EQ: Q0.3
Q: MW10
```


Syntax

Call-up:

TGEN < Block No. >

As operand:

TGEN < Block No. >u< Input/output >

Cycle time: 2 x 1 ...2 x 65535 ms

Performance time: 50 μ S

Time base: ms, no specification required

Representation

IL

TGEN 63

[]	S:	Start and set
[W]	I:	Period T
[]	P:	Pulse output

Description

When the "S" input is High, the period in milliseconds, at the "I" input, is transferred. The "P" output generates pulses with a pulse/pause ratio of 1:1 for as long as the "S" input carries a "H" signal.

The time period should always be at least twice the cycle time so that the High and Low signals can be recognized clearly.

If the time period at the "I" input is changed, a rising edge must be generated at the "S" input in order that the new value can be accepted.

Syntax

Call-up:

TP<Block No.>-R

- R only if retentive operation is required.

As operand:

TP < Block No. >u< Input/output >

Time range: 1...65535 ms

Performance time: 300 μ s

Basic time: ms, no specification required

Representation**IL**

TP5-R

	S:	Start and set
	R:	Reset
[W]	I:	Pulse time
[]	P:	Pulse output T
[W]	O:	Actual time value

<T

Description

When the rising edge is at the "S" input, the "P" output is set High, and the pulse time at the "I" input (always hV milliseconds) is transmitted. Irrespective of the status of the "S" input the "P" output remains High for the duration of the pulse, and then goes back to Low.

The set time should always be greater than the cycle time so that the pulse can be recognized clearly.

The Q output indicates the running time in milliseconds.

The timer can be reset to the initial state with a High signal at the "R" input.

Syntax

Call-up:

TR < Block No. > - R

- R only with retentive operation.

As operand:

TR < Block No. > u < Input/output >

Time ranges: 1...65 535 ms

1...65 535 s

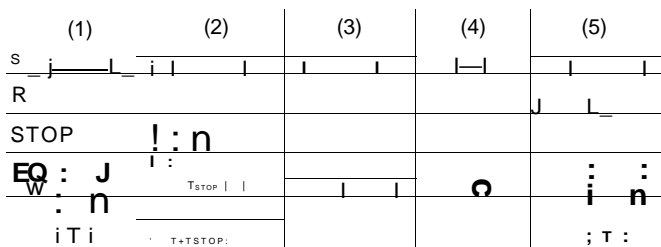
Performance time: 300 μ sBasic time: - ms for milliseconds
- s for seconds

Representation

IL

TR 7-s-R

[]	S:	Start und set
[]	R:	Reset
[]	STOP:	Interruption of timer
[W]	I:	Set time value
[]	EQ:	Control output
[W]	Q:	Actual time value



Description

The delay factor at "I" is accepted with the rising edge at the "S" input and the "EQ" output is set to the "H" level, delayed by the time "T"

$T = \text{delay factor} \times \text{time base}$

The set time should always be greater than the cycle time to ensure that the delay is always detected.

The output "Q" indicates the time elapsed in units of the selected time base.

The time count can be interrupted by setting the "STOP" input High; i.e. the delay time "T" is extended by the time for which the "STOP" input is High (2).

The STOP input should only be set in the time between the setting of the S input and that of the EQ output. If the STOP input is already High when the S input is reset (3) or set (4), the EQ output will respond as shown in the signal diagram.

The timer can be reset to the initial state with a "H" signal at the R input.

If the R input is set from 0 to 1 while the S input is High, the EQ output will be set with a delay (5).

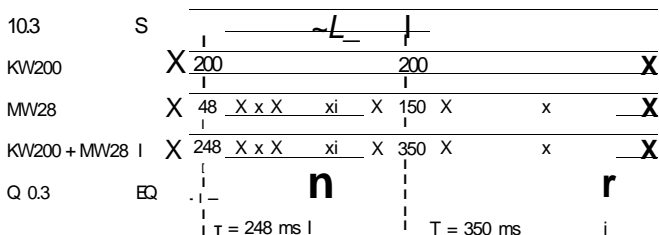
Example

After a time delay of T ms, the timer 12 is to transmit a signal received from I 0.3 to the Q 0.3. The time T is variable and is obtained by adding to the basic time of 200 ms (KW 200), a time determined by the process, which is contained in MW 28.

The program

```

TR 12-ms
S: 10.3      Input signal
R:
STOP:
I: LKW200   Constant basic time
   ADD MW 28 Variable time ratio
EQ: Q0.3    Output signal
Q:
    
```



7 SK Function Block

Sequential Control Function Block

Contents

Basic principles of sequential control programming	7-3
- Applications	7-3
- Graphical symbols	7-4
- Application example	7-7
- Elementary sequence control	7-10
Description of function block	7-13
- Syntax	7-13
- Representation	7-13
- Description	7-14
Program sequence with the sequential control function block	7-17
- Initialisation	7-17
- Processing	7-19
- Status indication	7-22
Program examples	7-25
- Linear sequence	7-25
- OR sequence	7-28
- AND sequence with synchronisation	7-31

SK Function Block

Basic Principles of Sequential Control

Applications

Process-dependent or time-dependent sequential control enables various tasks that are stored in different programs to be executed in a particular order. These tasks are executed step by step, according to the logical structure of the sequence control. The sequential control function block provides a user-friendly solution for implementing this structure in the user program.

The program of every step is a self-contained unit. This means that a self-maintaining function or an interlock do not need to be programmed as well. The SK (sequential control) function block handles the management of the step sequence.

A step sequence ensures that a step is only activated if the previous step has been deactivated. This allows complex sequences to be programmed simply and clearly.

The currently active steps are always indicated, thus simplifying error diagnostics.

The advantages of the sequential control function block are:

- clear structuring of complex sequences
- reduced work load; self-maintaining functions and step interlocks need not be programmed
- simple set and reset features of steps
- modifications to the sequential control function block are possible without any problems
- simple error diagnostics by program indication of the active step
- fast step sequence processing, unlike jump destination list

SK Function Block

Basic Principles of Sequential Control

Applications

The sequential control function block furthermore provides a large number of possibilities for nesting, thus allowing for the creation of highly flexible step sequences with a range of functions far beyond the normal range.

Graphical symbols

Sequential control programming allows sequences to be shown graphically or as a structure.

The symbols used here comply with DIN 40 719 Part 6 (corresponding to IEC 3B (sec) 49).

The start or initial step defines the basic or initial position, containing the start and reset conditions at the beginning of the process. The initial step is always shown in a double frame.

Steps are numbered consecutively, and each step is assigned one or several actions. Only when the step is active can the corresponding actions be carried out.

An essential feature of the step sequence is that only one step is active whilst all the others are ignored. The program works in the current step until the appropriate transition (step condition) has been fulfilled. Only then can the program continue processing in the next step.

SK Function Block

Basic Principles of Sequential Control

Representation
of the SK

Symbol

Name

»



Initial step

1



Transition or step condition



Step

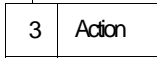


Alternative branch
(OR sequence)

2

3

Action



Step with action

Simultaneous branch
(AND sequence)

Synchronisation

SK Function Block

Basic Principles of Sequential Control

Graphical symbols

Cyclical processing	Active step
	Transition

Figure 7-1: Cyclical processing of the step sequence

With alternative branches (OR sequences), only one of the subsequent steps programmed in parallel can be executed. Alternative branches are indicated by a single horizontal line.

With simultaneous branches (AND sequences) several parallel branches can be processed at the same time. These branches are indicated by a double horizontal line. A simultaneous branch is synchronised.

The synchronisation ensures that the following transition is not processed until the last steps of all parallel branches are active.

SK Function Block

Basic Principles of Sequential Control

Application example

The following example of a paint filling plant is used to illustrate the procedures involved in sequential control programming.

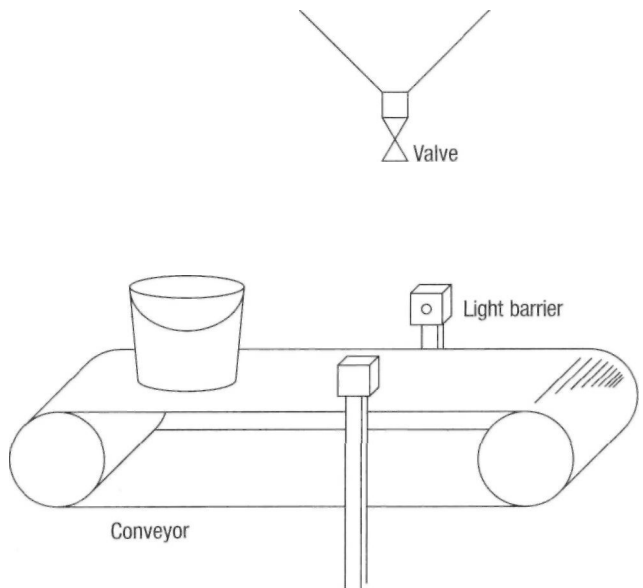


Figure 7-2: Paint filling plant

SK Function Block

Basic Principles of Sequential Control

Application example The conveyor belt is switched on by pressing a button. The paint can reaches the light barrier after a period of time. The conveyor belt then stops and the paint filling valve is opened. Once the required filling quantity has been reached, the valve is reclosed. After the "Flow = 0?" check, the paint can is transported further.

The individual working steps are programmed in the following order:

- Press start button
- Conveyor belt motor running
- Paint can reaches light barrier
- Switch off conveyor belt
- Open valve
- Measuring filling volume
- Close valve
- Check flow
- Switch on conveyor belt motor

SK Function Block

Basic Principles of Sequential Control

This example therefore produces the following step sequence:

- Start button on?
- 1 Conveyor belt motor on
- Light barrier reached 7
- 2 Conveyor belt off/Valve open
- Filling quantity reached?
- 3 Valve closed
- Flow = 0?
- 4 Conveyor belt on

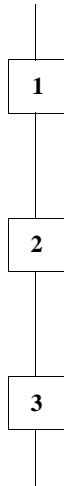
Figure 7-3: Step sequence for a filling plant

SK Function Block

Basic Principles of Sequential Control

Elementary sequence control

Linear sequence



The sequence passes from step S1 via transition T1 to step S2 and via transition T2 to step S3.

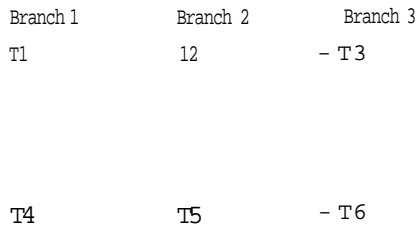
When T1 is enabled, i.e. when the transition condition for T1 is fulfilled, step S1 is deactivated whilst step S2 is activated. After step S2 has been processed, T2 is also enabled. Step S2 is then deactivated and step S3 activated.

The individual steps are always separated by transitions.

SK Function Block

Basic Principles of Sequential Control

OR step sequence
(Alternative branch)



After step S1 has been deactivated, either step S2, step S3 or step S4 is activated, depending on whether T1, T2 or T3 is enabled. If more than one transition condition is fulfilled, the transition that is located nearest to the left is enabled first.

Step S5 is activated if one of the preceding transitions T4, T5 or T6 is enabled.

SK Function Block

Basic Principles of Sequential Control

Elementary
sequence control

AND step sequence
(Simultaneous branch with synchronization)

li

Branch 1

Branch 2

— T2

- T3

When T1 is enabled, step S1 is deactivated. Step S2 and step S3 are activated at the same time.

When T2 is enabled, step S3 is deactivated and step S4 is activated. The branches are executed separately.

The convergence of a simultaneous branch is synchronized.

The validity of T3 is not checked until step S2, step S3 and step S4 have been executed. If T3 is enabled, the preceding steps are deactivated whilst step S5 is activated.

SK Function Block

Description of Function Block

Syntax

Call-up:

SK Function block no.> - (No. of steps) - R

(R only if retentive function is required)

As operand:

SK <Function block no.> <Input/Output>

Function block nc the number of the function block
 tie memory size of the PS 4 200 i

Number of steps 1...99

Nesting depth: 3

(Cascading)

Execution time:

With Set = Reset = 0 approx. 130 us

With Reset = 1 approx. 240 us

With Set = 1 approx. 250 us

With invalid SINO approx. 150 us

Representation

AML

SK 3 - 14

[] S:	S	Set
[] R:	R	Reset
[b] SINO:	SINO	Step number
[b] ERR:	ERR	Error output
[b] SQNO:	SQNO	Step number indication
[] TG:	TG	Step change indication
[\$] INm	INIT	Step call-up after reset
[\$] AC1:	AC1	Name of subprogram for Step 1
[\$] AC2:	AC2	Name of subprogram for Step 2
[S] AC3:	AC3	Name of subprogram for Step 3
AC14	AC14	Name of subprogram for Step 3

SK Function Block

Description of Function Block

Description

User programs can be structured simply and clearly through the use of sequential control function blocks.

Each SK function block can activate up to 99 steps. A step can itself also activate another step sequence. The maximum nesting depth possible is 8. The individual steps are created by subprograms which in turn contain actions to be executed. The necessary transition (step condition) must be programmed in between the end of one step and the beginning of another. Steps can be executed in succession, in parallel or in a particular order, thus allowing very complex sequences to be formed according to the requirements of the application at hand.

The inputs and outputs of the function block have the following meaning:

S Set

Set activates the sequential control function block

R Reset

Reset deactivates the sequential control function block and activates the initialisation program at the INIT input

SINO Step Input No.

The SINO input defines the number of the current step

ERR Error

Indication of error states

SQNO Step Output
 Number

The SQNO output indicates the number of the current step

SK Function Block

Description of Function Block

TG Toggle

The TG output indicates the change to another step. In normal operation this output has the signal 1. Only in the first cycle after a transition does the TG output go to 0.

INIT Initialisation

Name of initialisation subprogram run when Reset is active.

Example: "\$INIT"

AC Action

Name of the current step subprogram.

Example: "\$STEP1"

SK Function Block Program Sequence with the SK Function Block

The function of the sequential control function block and the linear sequencing of the steps always consist of the following elements:

1. Initialization of the SK function block
2. Processing of the SK function block
3. Status indication of the function block

Initialisation

The SINO input specifies which step is to be processed. Before the step sequence is called up for the first time, this input must be assigned with the number of the first step to be processed.

The initialisation is best carried out by the INIT subprogram which stays active as long as the Reset input = 1.

The operand INBO.O (flag for identifying the first cycle after a Reset or Pushbutton reset) can be used to start an initialisation automatically. The Reset input is set to 1 in the first cycle after the program start.

The INIT program assigns the number of the first step to the SINO input.

SK Function Block Program Sequence with the SK Function Block

Initialisation

Program example: Initialisation

```
SKO 2
S:   K 1
R:   INB 0.0
SINO:
ERR:
SQNO:
TG:
INIT: SINIT
AC1:  SSTEP1
AC2:  SSTEP2
EP
SINrr "Initialisation of the function block
      "Start step 1:
      LKB 1
      = SKO SINO
      EM
SSTEP1 "Step 1

      EM
SSTEP2 "Step 2

      KM
```

This ensures that the SK function block knows the number of the first step to be processed directly after the program is started. The step subprogram of the SINO input is then changed. The logical sequence of the PLC user program can therefore be programmed simply.

SK Function Block Program Sequence with the SK Function Block

Processing

To activate the function block, the Set input must be 1. This simultaneously activates the step defined at the SINO input. If both S and R inputs are 1 at the same time, only the initialisation subprogram is executed.

In the following examples, the S input is permanently set to 1. The variables T1 and T2 are freely definable transitions.

Several actions can be carried out in the subprogram. They remain active until the transition at the end of the subprogram has been fulfilled. Once the transition is fulfilled, the SINO input is assigned the number of the next step to be processed.

In the following cycle the new step is automatically activated and the old step deactivated.

SK Function Block Program Sequence with the SK Function Block

Processing

Program example: check transition

```
SKO- <
S:   K1
R:   INB 0.0
SINO:
ERR:
SQNO
TG:
INIT: $INIT
AC1:  SSTEP1
AC2:  SSTEP2

EP

SINIT "Initialisation of the function block

EM

$STEP1 "Step 1
        1st action
        2nd action

        "Check transition
        L 'T1 Transition 1
        JCN END
        "Start step 2
        LKB 2
        = SKO SINO

END

EM

SSTEP2 "Step 2
        1st action
        2nd action

        "Check transition
        L T2 Transition 2
        JCN END
        "Start step 3
        LKB 3
        = SKO SINO

END

CM
```


SK Function Block Program Sequence with the SK Function Block

Status indication

Various status signals are provided for monitoring the sequential control function block.

ERR output

This output shows malfunctions in the processing of the input data.

Table 7-1: Error signals at the ERR output

Error number:	Error cause:	Error behaviour:
binary: 00000001 or decimal: 1	SINO input = 0 (S=1)	SK not active. No step being processed.
binary: 00000010 or decimal: 2	Step number exceeds max. possible step no.	Error output set. Function block stays in current state.
binary: 00000100 or decimal: 4	No subprogram on selected AC output	Error output is set. The selected step is accepted. No instructions are executed since there is also no action program.

SQNO output

The SQNO output (step output number) indicates the number of the current step. If this output indicates the value 3, the 3rd step has been selected. With 5, the 5th step has been selected.

The value 0 indicates the initialisation subprogram.

SK Function Block Program Sequence with the SK Function Block

TG output

The toggle output indicates the change from one step to the other. The toggle output is 1 for as long as a step is still active. Only in the first cycle after the change to a new step is the TG output 0.

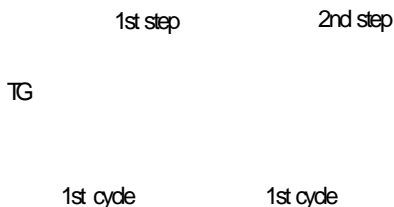


Figure 7-5: Step change indication on TG output

The TG output can be used to initialise individual steps. For example, it can be used to implement a time monitoring function. If an operation is too long because either the limit switch has not been reached or there is an electrical fault, a fault indication signal can be output.

SK Function Block Program Sequence with the SK Function Block

Status indication

The TG output pulse is used to start a timer.

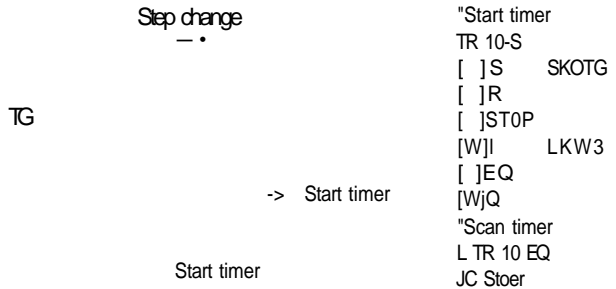


Figure 7-56: Program example using the step change indication

If three seconds expire after the timer was set, the timer monitoring function initiates a troubleshooting function in the program.

INIT/AC output

The I NIT and AC outputs are 1 if the appropriate subprograms are active. They cannot be scanned as operands but only be shown in the Status display menu.

SK Function Block

Program Examples

Be sure to use meaningful block labels in your user program to facilitate programming and the legibility of the program.

The following program examples show some typical applications using the sequential control function block. These examples use the bottling plant as the basic application.

Linear step sequence

Once the plant is switched on, the controller is in the Wait state. The individual steps are not carried out until the start button has been pressed. The program returns to step 2 "Conveyor belt motor On" from step 4 "Close valve".

SK Function Block

Program Examples

Linear step
sequence

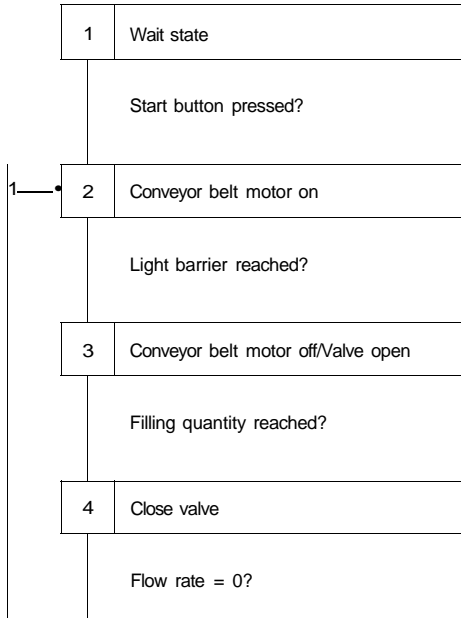


Figure 7-7: Linear step sequence

Program example: linear step sequence

```
"Linear step sequence
SKO-4
S:    K 1
R:    INB 0.0
SINO:
ERR:
SQNO:
TG:
INIT: $INIT
AC1   SSTART
AC2   $MOT_ON
AC3   $MOT_OFF
AC4   $VALV_OFF
EP
```

SK Function Block

Program Examples

```
$INIT      "Initialisation of the function block
           "Start first step
           L K B 1
           = SKO SINO
           EM

$START     "Start conditions
           "Start button pressed?
           L 'START                      Start button
           JCN END
           L K B 2
           = SKO SINO

END

           EM

$MOT_ON    "Conveyor belt motor
           L K 1
           = 'MOTOR                      Conveyor belt motor
           "Light barrier reached?
           L 'LIGHTBAR                   Light barrier
           JCN END
           L K B 3
           = SKO SINO

END

           EM

$MOT_OFF   "Conveyor belt motor off,
           valve off
           L K O                          1st action
           = 'MOTOR                      Conveyor belt motor
           L K 1                          2nd action
           = 'VALVE                      Filling valve
           "Filling volume reached?
           L 'FILLQUAN                   Filling quantity
           JCN END
           L K B 4
           = SKO SINO

END

           EM
```


SK Function Block

Program Examples

Linear step sequences

```
$VAW_OFF  "Valve off
           LKO
           = 'VALVE           Filling valve
           "Flow = 0?
           L 'FLOW           Flow
           JCN END
           LKB2
           = SKO SINO

END

FM
```

OR step sequence

The paint filling plant is programmed to fill either red or blue paint. A toggle switch selects either the red or blue paint containers.

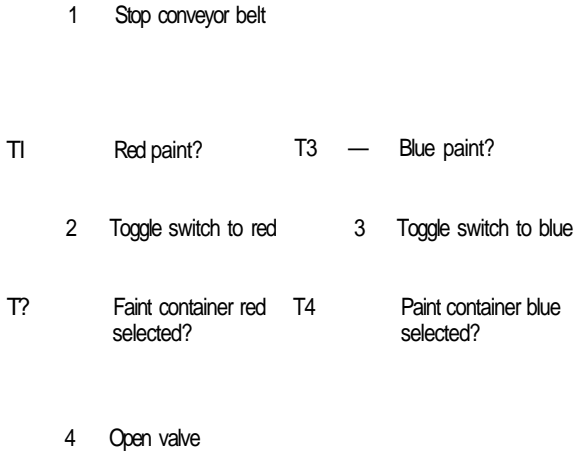


Figure 7-8: OR step sequence

SK Function Block

Program Examples

Only one SK function block is required for the OR step sequence since only one sequence can be run.

As soon as one of the two transitions (T1 or T3) is fulfilled, the sequence moves to the appropriate branch. This branch can then no longer be changed, even if the other transition was also fulfilled.

The transitions T1 and T3 are programmed at the end of the first step (\$MOT^OFF). A jump destination list is created for this purpose.

The SINO input is then loaded with the constants 2 or 4.

Program example: OR step sequence

```
SKO-x
[]S:      K1
[]R:      INB 0.0
[b]SNC);
.b]ERR
[b]SQNO:
[]TG:
[$]INIT | $INIT
[$]AC1  | $MOT_OFF
[$]AC2  | $U_RED
[$]AC3  | $U_BLUE
[$]AC4  | $VALV_ON
```

EP

```
$INIT      "Initialisation of the function block
LKB1
= SKO SINO           Step sequence 0
EM
```

SK Function Block

Program Examples

OR step sequence

```
$M0T_OFF "Motor off
          LKO
          = 'MOTOR      Conveyor belt motor
          L'RED        Red paint?
          JCN CONTINUE
          LKB2
          = SKO SINO   Step sequence 0
          JPEND
CONTINUE "
          L 'BLUE      Blue paint?
          JCN END
          LKB3
          = SKO SINO   Step sequence 0
END
          EM
$U_RED   "Toggle to red
          LKB1
          = 'TOGG      Paint toggle
          L 'CONT_RED  Red paint container
          JCN END
          LKB4
          = SKO SINO   Step sequence 0
END
          EM
$U_BLUE "Toggle to blue
          LKB2
          = 'TOGG      Colour toggle
          L 'CONT_BLUE Blue paint container
          JCN END
          LKB4
          = SKO SINO   Step sequence 0
END
          EM
$VALV_ON "Valve open
          LK1
          = VALV      Filling valve
          LM
```

SK Function Block

Program Examples

AND step sequence with synchronisation

A paint mixing system is to be implemented in which both paints are to be filled simultaneously through two different valves. The metering is variable according to the colour mix required.

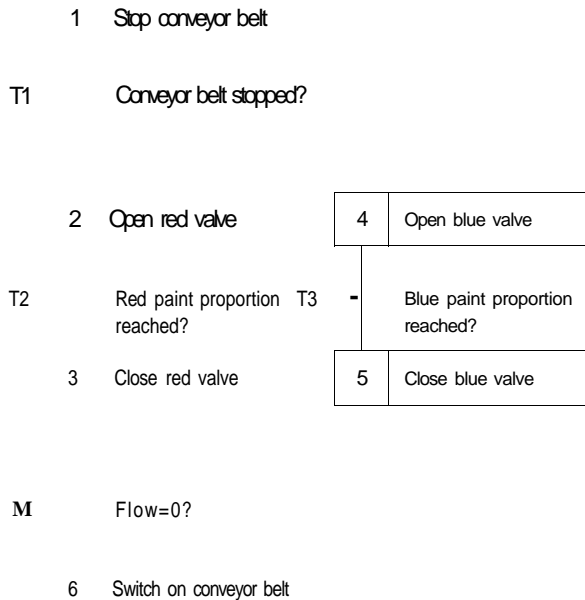


Figure 7-9: AND step sequence

The entire AND step sequence is divided into several parallel sequences. Each individual branch requires its own step sequence control function block which is called up by the main step sequence.

The convergence of the parallel branches is synchronised.

SK Function Block

Program Examples

AND step sequence
with synchronisation

A check is made whether the two parallel branches are
in their last step, before the transition to the next step of
the central step sequence is made.

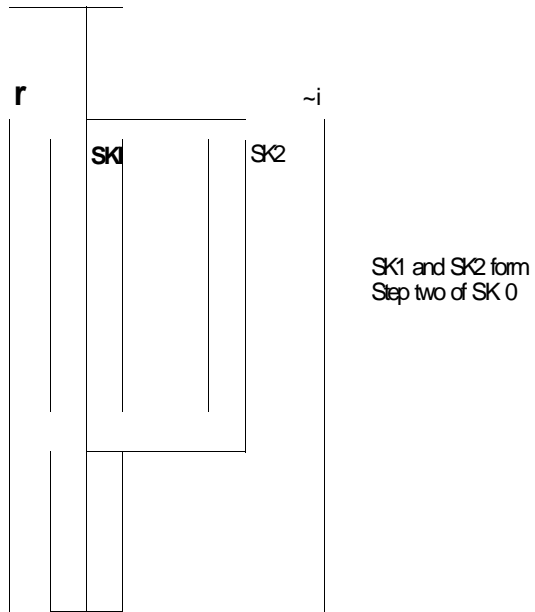


Figure 7-10: Example of a nested AND step sequence

SK Function Block

Program Examples

Program example: AND step sequence

```
SKO -x           Step sequence 0
[ ] S:   K 1
[ ] R:   INBO.O
[b] SINO:
[b] ERR:
[b] SQNO:
[ ] TG:
[$] INIT:  SINIT
[$] AC1:  $MOT_OFF
[$] AC2:  $MIX
[$] AC3:  $MOT_ON
```

LP

```
SINIT           "Initialisation of SK 0
                "Start step 1 of SK 0
                LKB 1
                = SKO SINO           Step sequence 0
                EM

$MOT_OFF        "Belt motor off
                LKO
                = 'MOTOR
                L 'SPEED             Belt speed
                BNZ END
                "Move to step 2 of step sequence 0
                LKB 2
                = SKO SINO           Step sequence 0

END

EM
```

SK Function Block

Program Examples

AND step sequence
with synchronisation

```
;MIX          "Mix colours
SK1 -2          Step sequence 1
MS:      K1
[ ]R:          N SKO TG          Step sequence 0
[b] SINO:
[b] ERR:
[b] SQNO
[ ]TG:
[ $] INIT:    SINIT1
[ $] AC1:     $RED_ON
[ $] AC2:     $RED_OFF

SK2-2          Step sequence 2
[ ]S:          K1
[ ]ft     N SKO TG          Step sequence 0
[b] SINO:
[b] ERR:
[b] SQNO
[ ]TG:
[ S] INIT:    SINIT2
[ $] AC1:     $BL_ON
[ $] AC2:     SBLOFF

"Synchronisation
"Step sequence 1 already finished?
L SK1 SQNO          Step sequence 1
CPKB2
BNE END
"Step sequence 2 already finished?
L SK2 SQNO          Step sequence 2
CPKB2
BNE END

"Volume flow=0?
L FLOW          Volume flow
JCN END

"Start step sequence 0 step 3
LKB3
=SK0 SINO          Sequence 0

END
```

SK Function Block

Program Examples

```
$M0T_ON    "Motor on
           L K 1
           = 'MOTOR                Conveyer belt motor

           EM

SINIT1     "Initialisation of function block 1
           "Start step 1 of step sequence 1
           LKB1
           = SK1 SINO                Step sequence 1
           EM

SREDJDN    "Open valve for red paint
           L K 1
           = 'V_RED                Valve for red paint
           L 'R_PROP                Red proportion
           JCN END
           "Start step sequence 1, step 2
           LKB2
           = SK1 SINO                Step sequence 1

END

           EM

$RED_OFF   "Close valve for red paint
           LKO
           = 'V_RED                Valve for red paint
           EM

SINIT2     "Initialisation of function block 2
           "Start step 1 of sequence 2
           LKB1
           = SK2 SINO                Step sequence 2
           EM

$BL ON     "Open valve for blue paint
           L K 1
           = 'V_BLUE                Valve for blue paint
           L 'R_PROP                Blue proportion
           JCN END
           "Start step sequence 2, step 2
           LKB2
           = SK2 SINO                Step sequence 2

END

           LM

$BL0FF     "Close valve for blue paint
           LKO
           = 'V_BLUE                Close valve for blue paint
           EM
```


8 Indirect Addressing

Contents

General	8-3
- Areas of application	8-3
- Basic principles	8-3
- Definition: indirect addressing	8-4
- Operands	8-6
Block transfer and block comparison	8-7
- Block transfer copy mode	8-7
- Block transfer initialize mode	8-8
- Block comparison compare mode	8-9
- Block comparison search mode	8-10
Working with ICPY and ICP	8-11
- Block transfer	8-13
- Block comparison	8-16
Working with the "&" address operator	8-21
Application examples	8-23
Test functions	8-27
- Status indication	8-27

l

Indirect Addressing

General

Areas of application

This manual describes the indirect addressing of operands and the functions Block transfer and Block comparison. These functions are realized by the function blocks ICPY (transfer) and ICP (comparison) which are described in detail on the following pages.

The block transfer and block comparison function blocks are used for indirect addressing and enable copy, compare and search functions to be carried out. The programming advantages over direct addressing provided by these function blocks are:

- Reduction in memory requirements in the user memory
- Reduction in write operations
- Greater program transparency

The indirect addressing principle can also be used for the function blocks RDAT, SDAT and SCK.

The number of the function blocks to be used is not limited. A limit is only given by the capacity of the user memory. Theoretically, the upper limit of function blocks is 65535.

Basic principles

Direct addressing is the most frequently used addressing method with programmable controllers. With this method, the address of the required data is specified directly so that, for example, the instruction L MW 234 causes the data that is to be loaded to be accessed directly from address MW 234. In comparison to indirect addressing, this method ensures greater data handling safety in all operations.

With indirect addressing, data contained in the defined address is interpreted as the address of the data required for the operation concerned rather than the actual data itself. Memory locations are therefore processed which are not actually defined until the program is running.

Indirect Addressing

General

Definition: indirect addressing

With direct addressing, the operand is given in addition to the operation concerned, this operand containing the address of the data required.

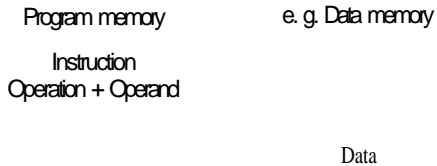


Figure 8-1: Direct addressing

If indirect addressing is used, the address of the data required is first stored in a cell in the memory (see Figure 8-2).

This cell can be set, raised or lowered while the program is running. Indirect Read or Write instructions that access this data cell take the data stored in it as the address of the data with which the operation is to be carried out.

Indirect Addressing

General

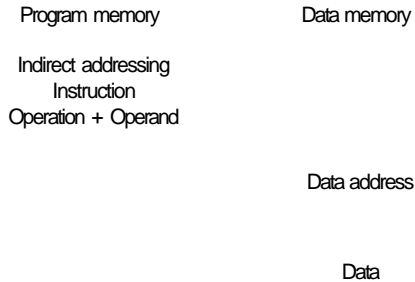


Figure 8-2: Indirect addressing

The contents of the operands in this case serve as a pointer to the addresses actually required in the data memory. This means that several different memory locations can be accessed with one single instruction. Only the pointer (operand containing the data address) needs to be changed with further instructions.

Indirect addressing enables changes to be made in the operand addresses written in the user program. This enables operations that have to be carried out repetitively with different operands to be implemented with less program memory required.

Indirect Addressing

General

Operands

The following operands can be used as address operands:

- M Markers
- SD Communication data
- RD Communication data
- Address inputs of other function blocks

Indirect Addressing

Block Transfer and Block Comparison

Two function blocks are used for the indirect addressing function. These are the ICPY (indirect copy) function block for block data transfers and the ICP function block (indirect compare) for comparing data blocks.

Block transfer

The block transfer function block features two modes for:

- Copying data fields (see Figure 8-3)
- Initializing data fields (see Figure 8-4)

Block transfer Copy mode

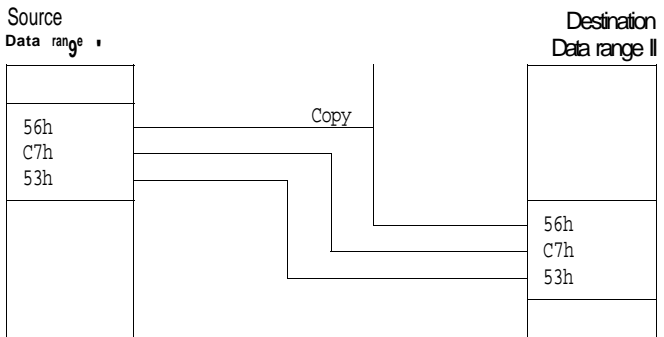


Figure 8-3: Copy function of the block transfer function block

In the copy mode, the function block makes a copy of a data field (in Figure 8-3 data field I) with a specified source address and transfers it to a destination address in the same data range or to a destination in a different data range (in Figure 8-3 data range II). The size of the field to be copied is optional. Between one and 255 data values can be copied. The data format must be byte.

Indirect Addressing

Block Transfer and Block Comparison

Block transfer Initialize mode

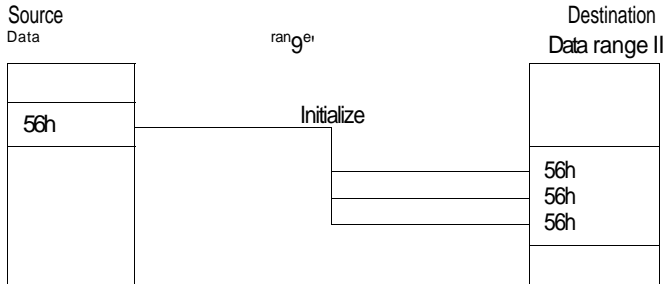


Figure 8-4: Initialize function of the block transfer function block

When the function block is in the initialize mode, the source is **one** data cell containing values that are copied to a data field. The destination data field can contain between one and 255 data values which must be in byte format.

This special type of copying function is termed initializing since one entire data field can be written with the same data value in one operation of the function block.

The zeroing of outputs or marker ranges, for example, can be carried out with the initialize function.

Block comparison

The block comparison function block features two modes for:

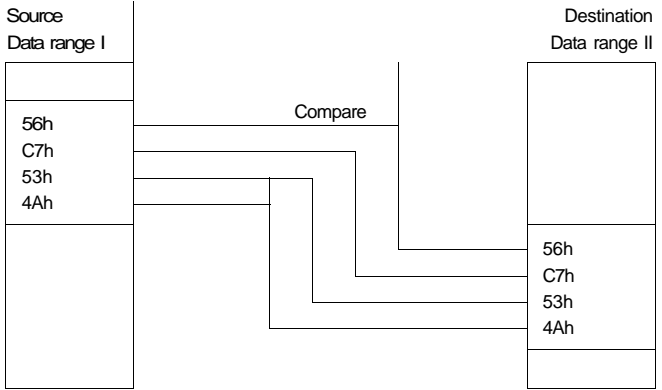
- Comparing data fields (see Figure 8-5)
- Searching for data values (see Figure 8-6)

Indirect Addressing

Block Transfer and Block Comparison

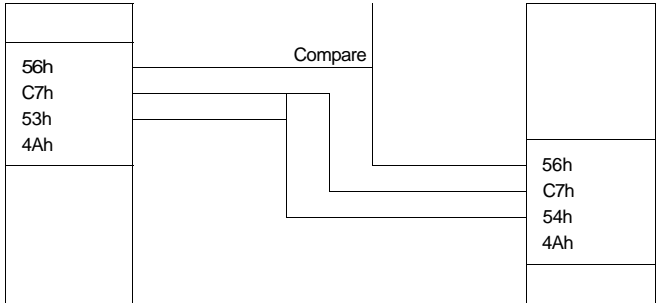
Block comparison Compare mode

1st case: Equal



Result:
Data fields are equal

2nd case: Not Equal



Result:
Data fields not equal:
Abort compare operation
- 3rd value not equal
- Source > Destination

Figure 8-5: Compare function of the block comparison function block

Indirect Addressing

Block Transfer and Block Comparison

Block comparison Compare mode

When used in compare mode the function block checks two data fields in order to determine differences. The terms source and destination are used also here in the same way as with the block transfer function block. The source and destination data field can contain between one and 255 data values which must be in byte format.

If a difference between the two fields is determined, the compare operation is aborted and the function block states whether the destination field is greater or smaller than the source field for the first unequal value.

If, for example, a data field needs to be monitored for any changes in its content, this function can be used to compare the current data with that of the previous cycle.

Block comparison Search mode

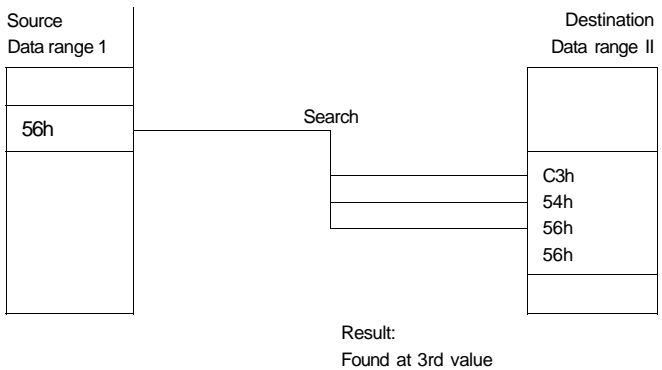


Figure 8-6: Data value search using the block comparison function block

The block comparison module also enables a specified data value to be searched for in a data field. The result is given as the offset address for where the value can be found. The destination data field can contain between one and 255 data values which must be in byte format.

If a particular article, for example, needs to be searched for in a stock management program, this function can be used as the core of the evaluation program.

Indirect Addressing

Working with ICP and ICPY

Only marker and communication data ranges can be accessed by indirect addressing. The access to address inputs of other function blocks is also permitted. The system-internal data is protected from unintentional accesses when the program is running.

This chapter describes how to operate the block transfer and block comparison function block in the SUCOsoft package.

The following brief definition of terms is given prior to the operating instructions for the function block so as to provide greater clarity and ease of comprehension.

The address operator "&" is used exclusively with these indirect addressing function blocks and is placed in front of the operand. It signifies that the operation is related to the operand address inside the system and not to the data of the operand as is otherwise the case.

Two data ranges have to be defined in order to use the indirect addressing function blocks. The source range (S) and the destination range (D) are marked, each of these ranges being defined by two variables (see Figure 8-7):

Indirect Addressing

Working with ICP and ICPY

1. The operand address SADR or DADR;

Data type: Address

in the example [&] SADR: MB 230

2. Number of elements NO:

Data type: Byte

in the example [B] NO: KB 4

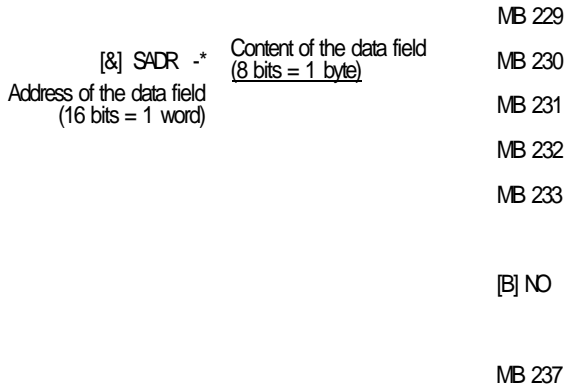


Figure 8-7: Definition of a data block

The data type of the operand given under SADR must be byte. The data type of NO is also byte.

Indirect Addressing

Working with ICP and ICPY

Block transfer syntax

Call: ICPY < Function block No. > < -R >

As operand:

ICPY < Function block No. > < Input/output >

-R, if remanent data is required for the operation

Number of elements: 1...255

Execution time:

Due to the hardware structure of the PS 4 200 series the execution time of this function block heavily depends on the type of source and destination operands involved.

Approx. values:

MOD 0: $(355 + 25 \times n)$ μ s

MOD 1: $(355 + 10 \times n)$ μ s

n = number of elements

Representation

```
CPY 63  
] MOD:  
&] SADR  
&] DADR  
B] NO  
[B] ERR
```

Description

Inputs:

MOD Copy/Initialize mode
 = 1 Copy data fields
 = 0 Initialize data fields

SADR Source address
 Start address of source data block from which the transfer is to begin

DADR Destination address
 Destination address to which the source data is to be transferred or from where initializing is to begin

NO Number of elements to be transferred 1-255

Outputs:

ERR = 0 Data limits are permissible
 = 1 NO is 0
 = 2 SADR parameters incorrectly set
 = 3 DADR parameters incorrectly set
 = 4 SADR is the same as DADR

Indirect Addressing

Working with ICP and ICPY

Description

The ICPY function block supports the transfer of data blocks within the system. A transfer is always made from a "source" to a "destination".

The following operands are valid:

Source	Destination	Data format
M	M	Byte
SD	SD	
RD	RD	
Address inputs	Address inputs	

It is therefore not possible to write input image registers using the block transfer function.

The function block can be used in the copy mode and the initialize mode which are selected by setting a 1 or a 0 at the MOD input.

The differentiation between address and data is important with this function block. With typical operations such as L M 2.2, it is always the data that is stored in this case in the marker cell which is accessed. In the case of the block transfer, the source address SADR from which the copying is to be made and the destination address DADR must be specified. The address operator "&" must be used here. This signifies that the operand behind it is an address and not a data value.

Copy mode

The number of data cells specified by the NO value are copied from the source address specified by SADR to the destination address specified by DADR.

In the following example (see Figure 8-8) the data from marker fields MB 23 to MB 26 is copied to marker field MB 30 to MB 33.

Indirect Addressing

Working with ICP and ICPY

Function block in IL:	Marker field:
ICPYO	Address:
[] MOD: K1	MB 23 7Dh
Source: [&] SADR:&MB 23	MB 24 3Bh
Destination: [&] DADR:&MB 30	MB 25 64h
NO: KB4	MB 26 A6h
[B] ERR:	MB 27 00h
	MB 28 00h
	MB 29 00h
	MB 30 7Dh
	MB 31 3Rh
	MB 32 64h
	MB 33 A6h
	MB 34 00h

Figure 8-8: Example of the copy mode of the ICPY function block

Initialize mode

This involves a transfer of the data stored under address SADR in a number of data cells specified by NO, beginning with the DADR destination address.

In the following example (see also Figure 8-9) the marker field from MB 27 to MB 32 is initialized with the data value 7Dh which is stored in MB 23.

Function block in IL:	Marker field:
ICPYO	
[] MOD: K0	MB 23 7Dh
[&] SADR:&MB 23	MB 24 00h
[&] DADR:&MB 27	MB 25 00h
[B] NO: KB6	MB 26 00h
[BI] ERR:	MB 27 7Dh
	MB 28 7Dh
	MB 29 7Dh
	MB 30 7Dh
	MB 31 7Dh
	MB 32 7Dh
	MB 33 00h
	MB 34 00h

Figure 8-9: Example of initialize mode of the ICPY function block

Indirect Addressing

Working with ICP and ICPY

Block comparison syntax

Call: ICP < Function block > < -R >

As operand:

ICP < Function block > < Input/Output >

-R, if remanent data is required for the operation

Number of

elements: 1...255

Execution time:

Due to the hardware structure of the PS 4 200 series, the execution time of this function block heavily depends on the type of source and destination operands involved.

Approx. values:

MOD 0: $(244 + 7 \times n)$ LIS

MOD 1: $(264 + 20 \times n)$ LIS

n = number of elements

Representation

IL

ICP 5

MOD:

SADR:

DADR:

NO:

GT:

EQ:

U:

Q:

ERR:

Description

Inputs:

MOD Block/Single character comparison

= 1 Compare data fields

= 0 Search for data value

SADR Source address

Start address of the source data block from which the comparison is to be made

DADR Destination address

Destination address from which the comparison is to be made

NO Number of elements 1 - 255 to be compared

Indirect Addressing

Working with ICP and ICPY

Outputs:

GT	Greater Than = 1 Data value in SADR > data value in DADR +/- signs are not included
EQ	Equal = 1 Data values are identical +/- signs are not included
LT	Less Than = 1 Data value in SADR < data value in DADR
Q	Offset output Indicates the relative offset address of the unequal value (comparison) or of the found data value (data value search). The offset from the beginning of the block is determined (DADR). The calculation of the offset is restricted to the following limits: $0 \leq Q \leq \text{NO}$
ERR:	= 0 Data limits are permissible = 1 NOisO = 2 SADR parameters incorrectly set = 3 DADR parameters incorrectly set = 4 SADR is the same as DADR

The coding on the MOD input determines whether a comparison or a data value search is to be carried out.

Compare mode

The block compare mode makes a comparison between NO elements starting with the source address given by SADR and the same number of elements starting from the destination address specified by DADR. If both data blocks are found to be equal, the Q = NO output and the EQ output are set to 1.

Indirect Addressing

Working with ICP and ICPY

Compare mode

The following applies when the compared data blocks are equal:

Q = NO; EQ = 1; LT = GT = 0.

If the compared data blocks are not equal, output Q indicates the location of the unequal data. The EQ output is set to 0 and the LT and GT output are set according to the result of the comparison (either 1 or 0). The following applies when the compared data blocks are not equal:

0 < Q < NO; EQ = 0; LT and GT depending on the result of the last comparison.

In the following example (see Figure 8-10) the marker field from MB 23 to MB 26 is compared with the marker field from MB 30 to MB 33.

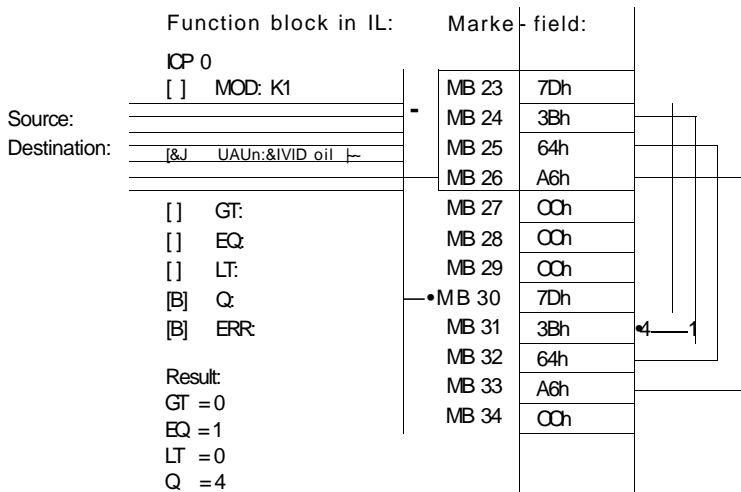


Figure 8-10: Example of the block compare mode of the ICP function block

The two data blocks are identical, this is indicated by EQ = 1 and Q = NO (run completed).

Indirect Addressing

Working with ICP and ICPY

Search mode

The data value whose address is SADR is searched for in the data field of NO elements starting with the destination address given by DADR.

If this value is found, the location is indicated via output Q and the output EQ is set (=1).

The following applies when a character is found:

$0 < Q < NO$; EQ = 1; LT = GT = 0.

If the data value is not found in the block, the output Q is equal to NO. The EQ output is set to 0 and the outputs LT and GT are set according to the last comparison.

The following applies when a character is not found:

Q = NO; EQ = 0; LT and GT according to the last comparison.

In the following example (see Figure 8-11), the value 7Dh in MB 23 is searched for in the marker field from MB 27 to MB 32.

Function block	Marker field:	
inIL:	MB 23	7Dh
[] MOD: KO	MB 24	00h
I&] SADR:&MB 23	MB 25	00h
[&] DADR:&MB 27	MB 26	00h
IB] NO: KB6	MB 27	3Dh
I I GT:	MB 28	8Dh
I I EQ:	MB 29	4Dh
M LT:	MB 30	7Dh
[B] Q:	MB 31	5Dh
[B] ERR:	MB 32	5Dh
Result:	MB 33	00h
Q = 3	MB 34	00h
LT = 0		
EQ = 1		
LT = 0		

Search

Found;
therefore
search
routine
aborted

Figure 8-11: Example of a data value search using the ICP function block

The data value 7Dh was found at the address DADR+Q (here: MB 27+3) and the search was terminated.

Indirect Addressing

Working with the "&" Address Operator

In order to identify an indirect address, the address operator "&" is placed in front of the operand concerned.

e.g. L&MB12

This also applies to symbolic programming

e. g. L& 'Input 1

See the introduction to this chapter for possible operands.

A sequence beginning with the "&" address operator must end with one or several allocation instructions to the address inputs of the function blocks used for indirect addressing.

e.g. L&MBO.O
 = ICPYO SADR
 = ICPY6 DADR

The address operands can be processed in word format in SUCOsoft like normal operands. Addresses can thus be used with addition, subtraction, multiplication, division, comparison operations etc. It must be ensured that only operations are used which can be processed with word operands (see Chapter 5, IL instructions).

When using the address operator "&", **only byte format** is permitted.

All other sequences can **only** be processed in **word format** since, however, the content of the data field 8 bit = 1 byte, but the address for each data field is 16 bit = 1 word.

Example:

```
1.   L       &MB0
      ADD    MW6
      SUB    MW234
           ICPY255 SADR
```

Indirect Addressing

Working with the "&" Address Operator

2. ICPY255
MOD:
SADR: L &MB26
DIV ICP254 DADR
ROTR
O &MB22
DADR:
NO:
ERR:

Only operands with the "&" address operator are permissible with the function block parameters SADR, DADR (for ICPY and ICP).

e. g. ICPY8
[&] SADR: &MB0
[&] DADR: &MB6

The address operator can only be used in byte operands.

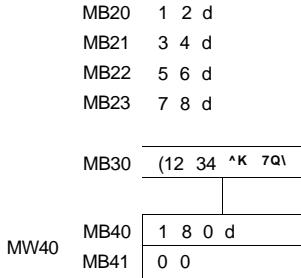
Indirect Addressing

Application Examples

Summation

The contents of the four marker bytes MB20 to MB23 are added up via the auxiliary marker byte MB30. The result is written in MW40 (e. g. for averaging).

Data memory (byte)



Summation program

```
0010 SSUM    "Start of SUM
             LKWO
             = MW0
             = MW40

0011 LOOP    "Start of SUM
             ICPY5
             [ ] MOD: K1
             [&] SADR:&MB20
             ADD MB0
             [&] DADR:&MB30
             [B] NO: KB1
             [B] ERR:
             LMW30
             ADD MW40
             = MW40

             LMB0
             ADD KB1
             = MB0

             LMB0
             CPKB4
             BLT LOOP

0012 END     "End PM SUM
             EM
```

The summation program was written in the SSUM function block. The call instruction for the function block is not shown here.

Indirect Addressing

Application Examples

Multiplication of delay factor to mixing time table

A marker field contains time values for several different mixing operations. Due to the difference in temperature of the various components added, all the mixing times must be modified by the same factor. This can be carried out using the ICPY function block in the following way:

			1 st run	2nd run	3rd run	4th run
Delay factor	MB99	02 h				
	MB100	72 h				
Basic mixing times	MB101	3Eh				
	MB102	6Fh				
	MB103	03 h				
	MB104					
	MB121					
Intermediate marker word	MB199		MJL	MJL	-> MJL	MJL
	MB200	E4h				
	MB201	7Ch				
Current mixing times	MB202	DEh				
	MB203	06 h				

Run 1 up to 122 is performed in one PLC cycle

Indirect Addressing

Application Examples

"Printout of file: c: mixtim.q42 dated 2. 3. 94"

```
00000      "Initialisation
001
002      L INBO.O Scan during 1 st cycle
003      JCN M2
004
005      L KB 0
006      = MB50
007
008      LKB 114
009      = MB100
010
011      L KB 62
012      =MB102
013
014      L KB 111
015      =MB104
016      LKB3
017      =MB106
018
00001 M2
001      "A delay factor < 50 which specifies the
002      "multiplier for the individual mixing
003      "times is stored in MB99.
004      "The current mixing times are
005      "generated from the product of
006      "the individual mixing time times the delay factor
007
008      L IBO.0.0.0
009      = MB99
010
00002 M3
001      "The current mixing time is now to be
002      "stored in marker field MB 200 to
003      "MB 203.
004      "The ICPY function block is used for this purpose.
005
006      "In each cycle, a mixing time is
007      "copied to auxiliary marker byte
008      "MB 199.0
009
010      ICPY0
```

Indirect Addressing

Application Examples

Continued

```
011          [ ] MOD: K 1
012          [&]SADR:&MB100
013          ADO MB50
014          [&] DADR: & MB199
015          [b] NO: KB 1
016          [b] ERR: QBO.0.1.0
017
00003 M4     "Calculate current mixing time
001
002          LMB199
003          MUL MB99
004          = MB199
005
00004 M5     "Write value in destination range
001          ICPY1
002          [ ] MOD: K1
003          [&]SADR:&MB199
004          [&] DADR: & MB200
005          ADD MB50
006          [b] NO: KB 1
007          [b] ERR: QBO.0.2.0
008
00005 M6     "Increase offset for next cycle
001
002          LMB50                                Loop counter
003          CP KB 20
004          BGT RESET
005
006          LMB50                                Loop counter
007          ADD KB 1
DOB         = MB50                                Loop counter
009
010          JPM3
011
00006 RESET "Reset Loop counter
001
002          LKBO
003          = MB50                                Loop counter
00007 END
001          EP
```

Indirect Addressing

Test Functions

Status indication

The status indication facility enables data ranges to be displayed in the form of a table. This is useful for checking the indirect transfer functions. It is also possible to show on screen two non-relating data ranges at the same time (double range).

This "range display" function can be called up in the test and commissioning of SUCOsoft using the F6 DISPLAY RANGE function key
F5 ONLINE PROGRAMMING
F9 STATUS DISPLAY
F6 DISPLAY RANGE.

Further entries:

F2 Display range

From: MB 0

To: MB 20

After making this entry, the contents of MB 0 to MB 20 are displayed on screen in byte format.

9 Program Examples

Contents

Foreword	9-3
General examples	
- Fail-safe programming	9-5
- Creating a configuration file	9-6
Examples with bit sequences	
- AND/OR sequence	9-9
- OR/AND sequence	9-10
- Binary divider	9-12
- Fleeting make contact, constant	9-13
- Fleeting make contact, variable	9-14
- Fleeting break contact, constant	9-15
- Fleeting break contact, variable	9-16
Examples with function blocks	
- SDAT: Save data in retentive range	9-17
- RDAT: Reload data from retentive range	9-21
- CK, SCK: Summer-/wintertime	9-22
- TR: Rolling shutter control	9-24
- TR: Two-point controller with hysteresis	9-26
- FALARM: Bottling plant	9-30
- CALARM: Encoder	9-33
- TALARM: Encoder	9-34
- TALARM: Encoder with delay	9-36
- TR: Pulse generator	9-38
- C: Down counter	9-41

Program Examples

Foreword

This manual gives advice on programming the SUCOcontrol PS 4 200 series in instruction list (IL) using program examples. A basic knowledge of control engineering is assumed.

The solutions of the problems shown in this manual illustrate the scope of programming possibilities of the SUCOsoft S 30-S 4-200.

These examples do not claim to be a complete representation of all programming features and do not exclude the possibility of other solutions.

Program Examples

Fail-safe Programming

The central processing unit of the PS 4 200 series recognizes whether an external transducer is on (voltage present). Of course it does not recognize whether this state comes from a make contact or a break contact, i. e. whether the transducer has been actuated or not.

To ensure fail-safety in the event of wire breakage, as in all control engineering, make contacts should be used for switching on and break contacts for switching off. For PLC programming, all external break contacts should be programmed as make contacts (see Figure 9-1).

LI 0.1
AI 0.2
= Q0.1

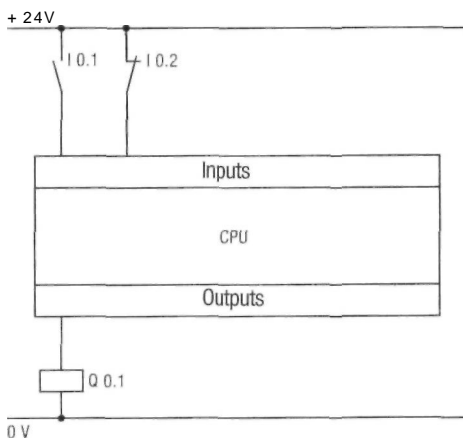


Figure 9-1: Fail-safe circuit of the PS 4 200 series

The output Q 0.1 is 1 if the input I 0.1 is activated and the input I 0.2 is **not activated**.

This means for the inputs of the PS 4 200 series that the output Q 0.1 is 1 if the input Q 0.1 **and** the input I 0.2 are 1.

Program Examples

Creating a Configuration File

The following program printout shows how the device configuration file created in SUCOsoft is incorporated in the user program.

Printout the file: c:examplea.q42 Date 24. 3. 94

```
00000      BLOCK0  "Incorporate configuration file
001
002                      #include"examplea.k42"
003
00001      BLOCK1  "Start of program
```

s

Program Examples

AND/OR sequence

The circuit diagram in Fig. 9-3 contains an "AND" and an "OR" sequence.

The output of the PS 4 200 series is activated if the switches are closed on the inputs I 0.0 "AND" I 0.4 "OR" if the switch is closed on the input I 0.2.

10.0 \ 10.2\

I 0.4\

Q0.0

Figure 9-3: Circuit diagram of AND/OR sequence

Printout of file: c:\exampleb.q42 Date 29. 3. 94

```
00000      BLOCK0  "Incorporate configuration file
001
002          =H=include"exampleb.k42"
003
00001      BLOCK1  "Start of program
001
002          LI 0.0          Input 0
003          A I 0.4          Input 4
004          O I 0.2          Input 2
005          = Q 0.0          Output 0
006
00002      BLOCK2  "End of program
001
002          tP
```

Program Examples

OR/AND sequence

The circuit diagram in Fig. 9-5 contains an "OR" and an "AND" sequence.

The output Q 1.2.1.0.1 of the LE 4-116-XD 1 is activated if the switch is closed on the input 11.2.0.0.0 "OR" on the input 11.2.0.0.3 of the EM 4-201-DX 2 "AND" if the switch on the input I 0.5 of the PS 4 200 series is closed.

This description shows that the EM 4-201-DX 2 and the LE 4-116-XD 1 are connected to the PS 4 200 series. In this case the connection is made via SUCOnet K. The following figure shows the device configuration.

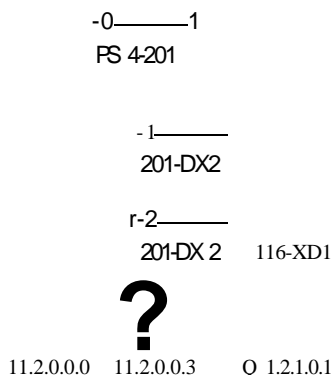


Figure 9-4: Device configuration of CR/AND sequence

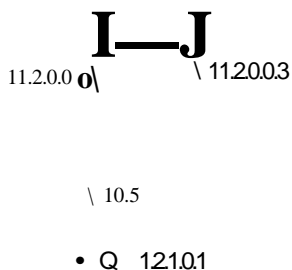


Figure 9-5: Circuit diagram of OR/AND sequence

Program Examples

OR/AND sequence

Printout of file: c:examplec.q42 Date 29. 3. 94

```
00000      BLOCK0  "Incorporate configuration file
001
002          #Include"examplec.k42"
003
00001      BLOCK1  "Start of program
001
002          L11.2.0.0.0      Input 0EM4
003          011.2.0.0.3     Input3EM4
004          AI 0.5          Input 5 PS4
005          = 0 1.2.1.0.1   Output 1LE4
006
00002      BLOCK2  "End of program
001
002          EP
```


Program Examples

Binary Divider

A binary divider is to be created with the ratio 2:1 according to the following diagram.

10.5 \bar{r} J L
Q0.3

Figure 9-6: Signal sequence of the binary divider

With the first H signal on the input I 0.5 the output Q 0.3 is set (= 1); with the second signal the output is reset (Q 0.3 = 0) and with the next signal the output is set, etc.

Printout of file: c:exampled.q42 Date: 12.4.94

```
00000 BLOCK0 "Incorporate configuration file
001
002 #Include"exampled.k42"
003
00001 BLOCK1 "Start of program
001
002 LI 0.5 scal input 10.5
003 AN M 0.1
004 JCN BLOCK2
005
006 LN M 0.2
007 = M 0.2
008
00002 BLOCK2 "
001
002 LI 0.5 Scan input 10.5
003 = M 0.1
004
005 LM 0.2
006 = 0.3
007
00003 BLOCK3 "End of program
001
002 FI-
```

Program Examples

Fleeting Make Contact, Constant

A fleeting make contact with a constant time is programmed. When changing from 0 to 1 on the input 10.1, the output Q 0.2 switches to 1 for one program cycle. The following diagram shows the sequence.

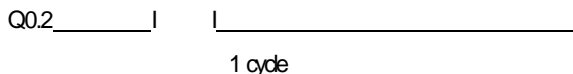


Figure 9-7: Signal sequence of fleeting make contact with constant time

Printout of file: c:\examplee.q42 Date: 12.4.94

```
00000      BLOCK0  "Incorporate configuration file
001
Oil?
003          #Include"examplee.k42"
00001      BLOCK1  "Start of program
001
002          L I 0.1          Scan input
003          AN M 0.0
004          = Q 0.2          Output Q 0
005
DOfi
007          L I 0.1          Scan input
008          = M 0.0
00002      BLOCK2  "End of program
001
002          IP
```

Program Examples

Fleeting Make Contact, Variable

A fleeting make contact with a variable time is programmed. When changing from 0 to 1 on the input 10.1, the output Q 0.2 switches to 1 for the duration of the programmed time (here: 5 seconds). The following diagram shows the sequence.

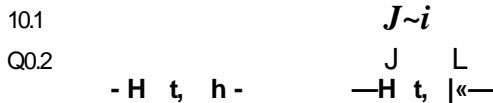


Figure 9-8: Signal sequence of fleeting make contact with variable time
t_i is set in the TRD timer

Printout of file: c:\examplef.q42 Date: 12.4.94

```
00000 BLOCK0 "Incorporate configuration file
001
002           #Include"examplef.k42"
003
00001 BLOCK1 "Start of program
001
002           L I 0.1           Scan input I 0.1
003           0 Q 0.2           Output Q 0.2
004           = M0.1
005
006           LM 0.1
007           AN M 0.2
008           = Q 0.2           Output Q 0.2
009
010           "The signal length is 5 seconds in this example
(III
012           TRO -S
013           [ ] S: M0.1
014           [ ] R:
015           [ ] STOP:
016           [w] I: KW5
iII/           [ ] EQ: M0.2
018           [w] Q:
019
00002 BLOCK2 "End of program
001
002           EP
```

Program Examples

Fleeting Break Contact, Constant

A fleeting break contact with a constant time is programmed. When changing from 1 to 0 on the input 10.1, the output Q 0.2 switches to 1 for one program cycle. The following diagram shows the sequence.

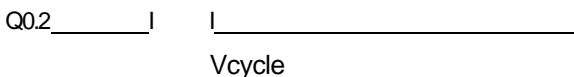


Figure 9-9: Signal sequence of fleeting break contact with constant time

Printout of file: c:\exampleg.q42 Date: 13.4.94

```
00000      BLOCK0  "Incorporate configuration file
001
002                      #Include"exampleg.k42"
003
00001      BLOCK1  "Start of program
001
002                      LI 0.1           Scan input
00:!                      =N M 0.1
004
005                      LM0.1
006                      AN M 0.0
007                      = 0.0.2         Output Q 0
008
009                      LM0.1
010                      = M0.0
011
00002      BLOCK2  "End of program
001
002                      LP
```

Program Examples

Fleeting Break Contact, Variable

A fleeting break contact with a variable time is programmed. When changing from 1 to 0 on the output 10.1, the output Q 0.2 switches to 1 for the programmed time (here: 8 seconds). The following diagram shows the sequence.

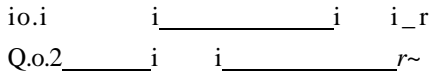


Figure 9-10: Signal sequence fleeting break contact with variable time
t, is set in the TRO timer

Printout of file: c:\exampleh.q42 Date: 13.4.94

```
00000 BLOCK0 "Incorporate configuration file
001
002 #Include"exampleh.k42"
003
00001 BLOCK1 "Start of program
001
002 LI 0.1 Scan input I 0.1
003 OM 0.1
004 AN M 0.4
005 = M0.1
006
007 LN I 0.1 Scan input I 0.1
008 AM 0.1
009 OQ 0.2 Output Q 0.2
010 AN M 0.4
011 = 0.0.2 Output Q 0.2
012
013 "The signal length is 8 seconds in this example:
014
015 TRO -S
016 [ ] S: Q0.2 Output Q 0.2
017 [ ] R:
018 [ ] STOP:
019 [w] I: KW8
020 [ ] EQ: M0.4
ii? I [w] Q:
022
00002 BLOCK2 "End of program
001
002 EP
```

Program Examples

SDAT: Save Data in Retentive Range

Bottling plant data needs to be saved retentively before switching off the plant since the RAM of the PS 4-201-MM1 is not battery backed. This data is contained in the marker bytes MB 100 to MB 119 and must be saved in the 64 byte flash EEPROM memory module (ZB 4-160-SM 1 or ZB 4-128-SF 1) before the plant is switched off. This is required so that the bottling plant starts with the same data after it has been switched on again.

The marker bytes MB 100 to MB 119 should be saved on the segment number 11 of the 64 Kbyte flash EEPROM memory for retentive marker ranges (cold start retentive range). This is carried out by activating the input I 0.0 of the PS 4 200 series.

511
510
509

13
12
11*)
10
9
8
7
6
5
4
3
2
1
0

Figure 9-11: Segment structure of the 64 Kbyte flash EEPROM memory for saving the retentive marker ranges, which keep their data also with a cold start

*) 20 bytes are saved under the segment number 11.

Program Examples

SDAT: Save Data in Retentive Range

Printout of file: c:\example1.q42 Date: 24.3.94

```
00000 BLOCK0 "Incorporate configuration file
001
fur,'          #Include"example1.k42"
003
00001 BLOCK1 "Start of program
001
002
00002 BLOCK2 "value assignment for retentive marker ranges
001 "to be saved with the SDATO function block
00? "in the memory module.
003 "This assignment is variable.
004
005 LI 0.0.1.0.1          Input 1 of the 1st LE4 module
(mil, AM 0.2
007 SM 100.0              Retentive marker for saving
008
009 LI 0.0.1.0.2          Input 2 of the 1st LE4 module
010 AM 0.6
(i11 OM 0.5
OK ' RM 100.0            Retentive marker for saving
013 SM 101.6            Retentive marker for saving
014
01',
010
01/'
018
019 LI 0.6                Input 6 of the PS 4 200 series
020 OM 0.5
0/1 AI 0.0.1.0.1          Input 1 of the 1st LE4 module
022 SM 119.7            Retentive marker for saving
023 RM 117.3            Retentive marker for saving
024

00003 BLOCK3 "Call of the SDATO function block for saving the
001 "retentive marker range in the flash EEPROM memory
002 "module. The marker ranges are saved on segment
on; "number 11 of the flash EEPROM memory. The
004 "segment length is 20 bytes.
005
006
007 SDATO
008 [ ] S: 10.0          Set input
HO'i [&] SADR: & MB 100 Source address
010 [w] SGNO: KW11
011 [b] LEN: KB 20
012 [b] ERR: MB 0      Error message on MBO
013

00004 BLOCK4 "End of program
001
002 EP
```

SDAT: Save Data in Retentive Range

Warning!

If program modifications are made in the "Device configuration" menu and the IL program is re-compiled, the retentive data, that keeps its information also after a cold start, can be modified after the program transfer. Before a modification of the device configuration, the data should thus be written to the flash EEPROM memory module with the SDAT function block and written out of the flash EEPROM memory with the RDAT function block after the modification. You can find further descriptions on this operation in Chapter 1, System parameters.

;

Program Examples

RDAT: Reload Data from Retentive Range

With the bottling plant mentioned in the previous example the data saved retentively must be re-written from the 64 Kbyte flash EEPROM memory module to the marker range MB 100 to MB 119 when the plant is switched on.

The data that is on the segment number 11 in the 64 Kbyte flash EEPROM memory is re-written to the marker bytes MB 100 to MB 119 by activating the digital input 10.1 of the PS 4 200. The data can thus be processed again in the program.

Printout of file: c:examplej.q42 Date: 24.3.94

```
00000 BLOCK0 "Incorporate configuration file
001
002           #Include"examplej.k42"
003
00001 BLOCK1 "Start of program
001
00002 BLOCK2 "Call the RDATO function block for the regeneration of the
001           "retentive data of the flash EEPROM memory in the
002           "marker range MB 100 to MB 119.
003           "The data is in the flash EEPROM memory on the
004           "segment number 11
005           "The segment is 20 bytes.
006           "The error messages of the RDATO function block are
007           "written in the marker byte MB 1 for further processing.
008
009
010           RDATO
011           [ ] S:      10.1      Set input
012           [&] • DADR: & MB 100 Destination address
013           [w] SGNO: KW11
014           [b] LEN:   KB 20
015           [ b ] ERR:  MB 1      Error message on MB1
016
00003 BLOCK3 "Further program processing
001
002           L MB 110           Data from flash EEPROM memory
003           ADD KB 20
004           = MB 130
005
006           "
007
00004 BLOCK4 "End of program
001
002           EP
```

Program Examples

CK, SCK: Summer/Winter Time

In this example the real-time clock of the PS 4 200 series is to be put forward by one hour in order to switch from winter to summer time on the 27.3.1994 at 2:00 h.

This operation is to be carried out automatically.

In order to solve this task, the function blocks CKO and SCKO are required in the user program.

When switching from summer time to winter time and vice versa, it is not necessary to use the programming device.

Printout of file: c:examplek.q42 Date: 24.3.94

```
00000 BLOCK0 "Incorporate configuration file
001
002           #Include"examplek.k42"
003
00001 BLOCK1 "Start of program
001
00002 BLOCK2 "Set parameters when the CKO function is to set the
001           "SCKO function block
002
003           LKB0
004           = MB 2           Minute 0
005
006           LKB2
007           = MB 3           Hour 2
008
009           LKB3
010           = MB 4           Month March
011
012           LKB27
013           = MB 5           Day 27
014
00003 BLOCK3 "The CKO function block has the task of putting forw
001           "the real-time clock of the PS 4 200 series on
002           "27.03.1994 at 2:00 h to 3:00 h via its EQ output an
003           "SCKO function block
004
005
```

Program Examples

CK, SCK: Summer/Winter Time

```
006          CK0
007          [ ] S:    K I
008          [x] TIME:
009          [x] DAY:
010          [x] DATE:
011          [w] VDAT:  MMW4
012          [w] VTIM:  MMW2
013          [ ] GT:
014          [ ] EQ:    MO.O    Set SCKO
015          OI',      [ ] LT:
016          nil,      [ ] ERR:
017
00004 BLOCK4 "New parameters for SCKO function block in orde
001          "put forward the real-time clock by one hour
002
003          mi;;
004          LKB94
005          = MB10                      Year 1994
006
007          LKB3
008          = MB 11                      Month March (3)
009
010          LKB27
011          = MB12                      Day 27
012
013          LKB0
014          = MB13                      Weekday (Sunday)
015
016          LKB3
017          = MB14                      Hour 3 (1 hour forward)
018
019          LKB0
020          = MB15                      Minute 0          * *
021
022          LKB0
023          = MB16                      Second 0
024
00005 BLOCK5 "Put forward real-time clock by one hour via
001          "SCKO function block
002
003          SCKO
004          [ ] S:    MO.O    Set SCKO
005          [&] SADR: &MB10   Year 1994
006          [b] ERR:  MB 20   Error message
007
00006 BLOCK6 "End of program
001
002          EP
```

Program Examples

TR: Rolling Shutter Control

This example describes a simple rolling shutter control. The rolling shutter at the entrance of a multi-storey car park should be opened and closed via an electric motor. This is implemented by a key switch outside and by pressing a button inside. The rolling shutter closes itself after a set time has elapsed. A warning lamp is lit which thus ensures that no car enters the garage while the shutter is closing.

A light barrier prevents the shutter from closing as long as a car is in the entrance area. The lighting inside the garage and the outside lighting in front of the shutter are kept on automatically for a fixed period after the shutter is closed.

Printout of file: c:example1.q42 Date: 25.3.1994

```
00000 BLOCK0 "Incorporate configuration file
001
002          #Include"example1.k42"
003
00001 BLOCK1 "Start of program
001
002
00002 BLOCK2 "Control the shutter motor (open)
001
002          L I 0.1          Button "Shutter open"
003          0 I 0.2          Key button "Shutter open"
004          0 I 0.5          Light barrier "Car in entrance"
005          0 Q 0.1          Load contactor "Shutter open"
006          AN I 0.3         Limit switch "Shutter open"
007          AN Q 0.2         Load conlactor "Shutter close"
008          = Q 0.1          Load contactor "Shutter open"
009
00003 BLOCK3 "Control the shutter motor (close)
001
002          L M 0.7          Contactor relay 1 "Close delay"
003          AN Q 0.1         Load contactor "Shutter open"
004          = Q 0.2          Load contactor "Shutter close"
005
00004 BLOCK4 "Control the warning lamp when the shutter is closed
001
002
003          L M 0.7          Contactor relay 1 "Close delay"
004          = Q 0.3          Load contactor "Shutter close"
005
```

Program Examples

TR: Rolling Shutter Control

```
00005 BLOCK5 "Control of garage lighting
001
002          LM0.6          Contactor relay 2 "Time of lighting"
003          = 0.0.4          Load contactor "lighting"
004
00006 BLOCK6 "Set the time for close delay via setpoint potentiometer (P1)
001
002
003          LI 0.3          Limit switch "Shutter open"
004          OM0.7          Contactor relay 1 "Close delay"
005          AN I 0.4        Limit switch "Shutter closed"
006          AN I 0.5        Light barrier "Car in entrance"
007          = M1.7          Contactor relay 3 "Close delay"
008
009          TR0-S          Close delay 0-1023 seconds
010          [ ] S:      M1.7          Contactor relay 3 "Close delay"
011          [ ] R:
012          [ ] STOP:
013          [w] I:      MW18          Time value 0-1023 seconds
014          [ ] EQ:      M0.7          Contactor relay 1 "Close delay"
015          [w] Q:
016
017          LIAW0          Time value 0-1023 seconds
018          = MW18          Time value 0-1023 seconds
019
00007 BLOCK7 "Set time for internal and external lighting
001          "via setpoint potentiometer (P2)
002
003          LI 0.3          Limit switch "Shutter open"
004          SM0.6          Contactor relay 2 "Lighting time"
005
006          TR1 -S          Lighting time 0-1023 seconds
007          [ ] S:      M0.6          Contactor relay 2 "Lighting time"
008          [ ] R:
009          [ ] STOP:
010          [w] I:      MW20          Time value 0-1023 seconds
011          (III) [ ] EQ:      M1.6          Contactor relay 4 "Lighting time"
012          [w] Q:
013
014          LIAW2          Time value 0-1023 seconds
015          = MW20          Time value 0-1023 seconds
016
017          LM1.6          Contactor relay 4 "Lighting time"
018          RM0.6          Contactor relay 2 "Lighting time"
019
00008 BLOCK8 "End of program
001
002          EP
```


Program Examples

TR: Two-point Controller with Hysteresis

Printout of file: c:examplm.q42 Date: 29.3.94

```
00000 BLOCK0 "Incorporate configuration file
001
002           #Include"examplm.k42"
003
00001 BLOCK1 "Start of program
001
002
00002 BLOCK2 "Initialize the controller parameters
001
no?           "Hysteresis Xu is 1 volt.
003
004           LKW102
00b           = MW100           Hysteresis Xu
006
00/           "Scan time T is 1 second.
008
009           LKW1000
010           = MW102           Scan time T
011
00003 BLOCK3 "Closed-loop controller enable
001           "I 0.0 = H — closed-loop controller active; L — closed-
           loop controller disabled
002
003           LI 0.0           Closed-loop controller enable
004           JC BLOCK4
005
006           LK0
007           = Q0.0           Manipulated variable "ON" or "OFF"
008
009           LI 0.0           Closed-loop controller enable
010           JCNBLOCK13
011
00004 BLOCK4 "Setting the scan time T
001
002           TR0-MS           Scan time 0.01 to 65.53 seconds
003           [ ] S:           NMO.O           Basic pulse
004           [ ] R:
005           [ ] STOP:
006           [w] I:           MW102           Scan time T
007           [ ] EQ:           MO.O           Basic pulse
008           [w] Q:
009
010           LN M 0.0           Basic pulse
011           JCBLOCK13
012
```


Program Examples

TR: Two-point Controller with Hysteresis

```
00005 BLOCK5 "Read in setpoint value (0-10 volt) via setpoint potentiometer
P|
r|
001 "(0-1023 increments)
002
003 LIAW0 Setpoint value x(t)
004 = MW104 Digital setpoint value w(t)
005
00006 BLOCK6 "Read in actual value (0-10 volt) via analogue input
001 "(0-1023 increments)
002
003 LIAW4 Actual value w(t)
004 = MW106 Digital actual value x(t)
005
00007 BLOCK7 "+/- sign calculation and calculation of the
001 "system deviation xd(t)
002.
003 LMW100 Hysteresis Xu
004 DIV KW 2
005 = MW108 Auxiliary marker byte 1
006
007 LMW106 Digital actual value x(t)
008 ADD MW108 Auxiliary marker byte 1
009 = MW110 Auxiliary marker 2
010
011 LMW104 Digital setpoint value w(t)
012 SUBMW110 Auxiliary marker byte 2
013 = MW112 System deviation at t = xd(t)
014
00008 BLOCK8 "Two-point controller calculation
001
002 LMW112 System deviation at t = xd(t)
003 BC BLOCK9
004
005 LK0
006 = M0.1 Auxiliary marker 1
007 JPBLOCK10
008
00009 BLOCK9 "
001
002 LK1
003 = M0.1 Auxiliary marker 1
```

Program Examples

TR: Two-point Controller with Hysteresis

```
00010 BLOCK10 "Compare hysteresis and system deviation
001
002          L KH-W FFFF
003          SUBMW112          System deviation at t = xd(t)
004          ADD KW 1
005          = MW114          Remainder of system deviation xd(t)
006
007          LMW100          Hysteresis Xu
008          CPMW114          Remainder of system deviation xd(t)
009          BLTBLOCK11
010
011          LK0
012          = M0.2          Auxiliary marker 2
013          JPBLOCK12
014
00011 BLOCK11 "
001
002          LK1
003          = M0.2          Auxiliary marker 2
004
00012 BLOCK12 "
001
002          LM0.2          Auxiliary marker 2
003          AM 0.1          Auxiliary marker 1
004          SM0.3          Auxiliary marker 3
005
006          LM0.2          Auxiliary marker 2
007          AN M 0.1          Auxiliary marker 1
008          RM0.3          Auxiliary marker 3
009
010          LM0.3          Auxiliary marker 3
011          =N Q 0.0          Manipulated variable "ON" or "OFF"
012
00013 BLOCK13 "End of program
001
002          FP
```

Program Examples

FALARM: Bottling Plant

A light barrier is to be used in a bottling plant for checking whether the bottles are filled as required. If more than three bottles are not filled correctly, the motor of the conveyor belt which transports the bottles must be stopped as soon as possible.

The conveyor belt can only be stopped if it is requested by the user. This is implemented if, for example, the input I 0.5 of the PS 4 200 series is connected with a switch which sends a High signal to the PLC.

Printout of file: c:examplen.q42 Date: 6.4.94

```
00000 BLOCK0 'Incorporate configuration file
001
no;'          #Include"examplen.k42"
003
00001 BLOCK1 "Start of program
001
00002 BLOCK2 "Call FALARM0 function block in order to obtain the
001 "message via the alarm input I 0.1 whether the bottles are
002 "filled correctly.
003 "The positive edge of the alarm input I 0.1 is evaluated.
004
005
006          FALARM0
007          [ ] EN:  I 0.1          Enable alarm function block
008          [ ] ACT:  K 0
009          [w] SOU:  KW4
010          [b] ERR:
011          [w] CNT:
012          [$] AC:   SUP0
013
014          "With the fourth edge on the alarm input I 0.1 the SPO
015          "subprogram is called.
016
00003 BLOCK3 "End of main program
001
002          EP
003
```

Program Examples

FAL7RM: Bottling Plant

```
00004 $UP0 "Subprogram in order to stop the conveyor belt if requested
001 "by the user.
002
003 L IP 0.5 Command stop conveyor belt
004 = QP 0.4 Conveyor belt stops
005
00005 BLOCKA "End of subprogram
001
002 EM
```


Program Examples

CALARM: Encoder

The signals of a positioning device are to be counted. The time between the signals is shorter than the cycle time of the PS 4 200 series. The division ratio on the CALARMO function block should be 50 so that the feed rate value is 1/10 mm on the ALARM counter (CNT output) of the function block.

Printout of file: c:exampleo.q42 Date: 6.4.94

```
00000 BLOCK0 "Incorporate configuration file
001
002           #Include"exampleo.k42 "
003
00001 BLOCK1 "Start of program
001
00002 BLOCK2 "Call CALARMO function block in order to count the signals
001           "via the I 0.0 input
002
003           CALARMO
004           [ ] EN: I 0.6           Set function block
005           [w] VT: KW50
006           [w] SOLL: KW1
007           [b] ERR:
008           [w] CNT: MW10           Feed rate value in 1/10 mm
009           [$] AC:
010
00003 BLOCK3 "End of program
001
002           FP
```

Program Examples

TALARM: Encoder

200 square wave signals are to be output by the PS 4 200 series for a positioning device. The signal length is 1 ms and the pulse/pause ratio is 1 : 1.

1 ms 1 ms
h — - h — •!

1

Figure 9-13: Signal sequence for positioning device

The output of the signals is started via the digital input 1 0.3 and the signals are output by the digital output Q0.0.

Printout of file: c:examplep.q42 Date: 11.4.94

```
00000 BLOCK0 "Incorporate configuration file
001
002           #Include"examplep.k42"
003
00001 BLOCK1 "Start of program
001
00002 BLOCK2 "Fleeting make contact for set condition and
001           "reset condition of the TALARM0 function block
002
003           L10.3           Start TALARM0
004           AN M 3.0
005           = M 3.1
006
007           L10.3           Start TALARM0
008           = M 3.0
009
010           LM3.1
011           S M 2.0           Set condition for TALARM0
012
013           LMO.0
014           R M 2.0           Set condition for TALARM0
015
```

Program Examples

TALARM: Encoder

```
00003 BLOCK3 "Call TALARMO in order to output the 200 signals
001          "(= 400 edges) via the digital output Q 0.0
002
003          TALARMO
004          [ ] EN:  M2.0      Set condition for TALARN
005          [b] MOD:  KB 2
006          [w] VT:   KW1000
007          [w] SOLL: KW400
008          [b] ERR:
009          [w] CNT:  MWO
010          [ $] AC:
011
00004 BLOCK4 "End of program
001
002          EP
```


Program Examples

TALARM: Encoder with Delay

With a positioning device the digital output Q 0.3 of the PS 4 200 series should be set 1.5 ms after the activation of the digital input I 0.3. The output Q 0.3 is reset by activating the digital input I 0.0. The inputs I 0.0 and I 0.3 must be interlocked.

I 0.3

Q0.3

1 0.0 _____ I 0.3

Figure 9-14: Signal sequence positioning with delay

Printout of file: c:\exampleq.q42 Date: 12.4.94

```
00000 BLOCK0 "Incorporate configuration file
001
002           #Include"exampleq.k42"
003
00001 BLOCK1 "Start of program
001
00002 BLOCK2 "Fleeting make contact and set condition of the
001           TALARM0 function block
11(1?
iiu'i           L IP 0.3           Start TALARM0
004           AN IP 0.0           Reset Q 0.3
1105           AN M 3.0
006           = M3.1
007
(1DH           L IP 0.3           Start TALARM0
HID           AN IP 0.0           Reset Q 0.3
010           = M3.0
1111
012           LM3.0
013           SM2.0           Set condition fc
014
```

Program Examples

TALARM: Encoder with Delay

```
00003 BLOCK3 "Reset condition for output Q 0.3 and
001          "TALARMO function block
002
003          L IP 0.0          Reset Q 0.3
004          AN IP 0.3        Start TALARMO
005          R QP 0.3         Output Q 0.3
006          R M 2.0          Set condition for TALARMO
nfi7

00004 BLOCK4 "Call TALARMO function block to implement a
001          "delay time of 1.5 ms
002
003          TALARMO
004          [ ] EN: M2.0      Set condition for TALARMO
005          [b] MOD: KB1
006          [w] VT: KW 1500
007          [w] SOLL: KW1
008          [b] ERR:
009          [w] CNT:
010          [$] AC: $UP0
011

00005 BLOCK5 "End of main program
001
002          EP
003

00006 $UP0 "Subprogram to set the output Q 0.3 after 1.5 ms
001
IKI:"          LK 1
tm           S QP 0.3          Output Q 0.3
004

00007 BLOCKA "End of subprogram
001
002          EM
```

Program Examples

TR: Pulse Generator

The application requires a pulse generator with different pulse/pause time with two function blocks (TRO and TR1). The pulse should be output via the output Q 0.4. The time for the H pulse is 4 seconds and for the pause time 6 seconds. The pulse generator starts automatically if the controller switches from the "Halt" status to the "Run" status. The following diagram shows the function sequence.

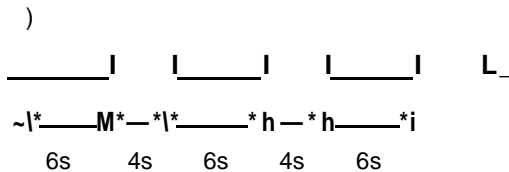


Figure 9-15: Signal sequence pulse generator

1) Programmstart

Printout of file: c:exampler.q42 Date: 13.4.94

```
00000 BLOCK0 "Incorporate configuration file
001
00?            #Include"exampler.k42"
003
00001 BLOCK1 "Start of program
001
002            "TRO function block generates the time for the
Hii;;            "L-level (6 seconds).
004
005            TR0-S
006            [ ] S:    NM0.0
007            [ ] R:
008            [ ] STOP:
009            [w] 1:    KW6
010            [ ] EQ:    Q0.4        Pulse output
011            [w] Q:
012
013            "TR 1 function block generates the time for the
014            "H-level (4 seconds)
015
```

Program Examples

TR: Pulse Generator

```
016          TR1 -S
017          [ ] S:   Q0.4      Pulse output
018          [ ] R:
019          [ ] STOP:
020          [w] I:   KW4
021          [ ] EQ:  MO.O
022          [w] Q:
023
00002 BLOCK2 "End of program
001
002          EP
```


Program Examples

C: Down Counter

The application requires a down counter with the CO function block. The CO function block outputs an output signal if it has reached the counter status "0" after a predefined number (setpoint value) of signals (30 pulses in this example) has been input. With more signals the function block counts down, starting with 65535.

Pulse input: I 0.7
Set counter: I 0.0
Reset 10.1 (set counter to 0)
Counter signal: Q 0.5 (counter status = 0)

Printout of file: c:examples.q42 Date: 13. 4. 94

```
00000 BLOCK0 "Incorporate configuration file
001
002          #include"examples.k42"
003
00001 BLOCK1 "Start of program
001
002          CO
003          [ ] U:
004          [ ] D: 10.7      Pulse input
005          [ ] S: 10.0      Set counter
006          [ ] R: 10.1      Reset counter
007          [w] I: KW30
01)8          [ ] Z: Q0.5
(ID)          [w] Q:
010
00002          "End of program
001
(ID-          EP
```


Appendix

Contents

List of figures	A-3
List of tables	A-7
Index	A-9

Appendix

List of Figures

1-1	System parameters menu	1-6
1-2	Device configuration menu	1-11
1-3	Program editor menu	1-13
2-1	Structure of an instruction	2-4
2-2	Input addresses	2-7
2-3	Output addresses	2-8
2-4	Marker addresses	2-9
2-5	Example of a function block	2-21
2-6	Program sequence and data flow during function block processing	2-25
2-7	Function blocks, types of memories and function block data storage	2-32
2-8	Function blocks, shifting of data ranges after inserting new function block data	2-33
2-9	Register overview	2-36
2-10	Overview of sequences	2-46
2-11	Block structure	2-48
2-12	Structure of the main program	2-50
2-13	Program cycle	2-51
3-1	Programming with inserted source file	3-6
3-2	Nested program structure	3-10
3-3	Program module call-ups initiated from the main program and a program module	3-12
3-4	Program module call-ups from an "Include" file of the main program	3-14
3-5	Location of main program and program modules in IL	3-15
3-6	Data transfer main program - program modules via the marker range	3-23
3-7	Location of program modules in the user memory	3-31
4-1	Test/commissioning main menu	4-5
4-2	PS 4-200 status menu	4-8

Appendix

List of Figures

4-3	Display of the device configuration and I/Q status	4-17
4-4	Device status	4-18
4-5	Diagnostics display for individual devices	4-18
4-6	Input/output display for individual devices (here: PS 4-201-MM 1 as slave)	4-19
4-7	Forcing of outputs	4-20
4-8	IL status display	4-23
4-9	Display range	4-24
4-10	Double range	4-25
4-11	LIFO/FIFO content display	4-26
4-12	Dynamic forcing	4-27
4-13	Online modifications	4-29
4-14	Online modifications	4-30
4-15	Program module directory	4-37
4-16	Date/time	4-39
7-1	Cyclical processing of the step sequence	7-6
7-2	Paint filling plant	7-7
7-3	Step sequence for a filling plant	7-9
7-4	Processing the sequential control function block within the user program	7-21
7-5	Step change indication on TG output	7-23
7-6	Program example for step change indication	7-24
7-7	Linear step sequence	7-26
7-8	OR step sequence	7-28
7-9	AND step sequence	7-31
7-10	Example of nested AND step sequence	7-32
8-1	Direct addressing	8-4
8-2	Indirect addressing	8-5
8-3	Copy function of the block transfer function block	8-7
8-4	Initialize function of the block transfer function block	8-8

Appendix

List of Figures

8-5	Compare function of the block comparison function block	8-9
8-6	Data value search using the block comparison function block	8-10
8-7	Definition of a data block	8-12
8-8	Example of the copy mode of the ICPY function block	8-15
8-9	Example of the initialize mode of the ICPY function block	8-15
8-10	Example of the block compare mode of the ICP function block	8-18
8-11	Example of a data value search using the ICP function block	8-19
9-1	Fail-safe circuit of the PS 4 200 series	9-5
9-2	Device configuration menu	9-6
9-3	Circuit diagram of AND/OR sequence	9-9
9-4	Device configuration of OR/AND sequence	9-10
9-5	Circuit diagram of OR/AND sequence	9-10
9-6	Signal sequence of the binary divider	9-12
9-7	Signal sequence of fleeting make contact with constant time	9-13
9-8	Signal sequence of fleeting make contact with variable time	9-14
9-9	Signal sequence of fleeting break contact with constant time	9-15
9-10	Signal sequence of fleeting break contact with variable time	9-16
9-11	Segment structure of the 64 Kbyte flash EEPROM memory for saving the retentive marker ranges, which keep their data also with a cold start	9-17
9-12	Function sequence two-point controller with hysteresis	9-26
9-13	Signal sequence for positioning device	9-34
9-14	Signal sequence for positioning with delay	9-36
9-15	Signal sequence pulse generator	9-38

Appendix

List of Tables

Operand overview	2-4
Available operators with their appropriate data types	2-18
Available function blocks	2-23
Available pre-processor instructions	2-53
DSW diagnostics status word	4-9
INB information byte	4-14
Resolution of analogue values	4-20
Operand overview	5-3
Error signals at ERR output	7-22

Appendix

Index

A	
Abbreviations	5-4
Addition	5-10
Address operator &	8-21
Addressing the operands	2-5
Allocation	5-6
Analogue	
- inputs	2-7
- output	2-9
- transducers	4-17
AND	5-8
AND sequence	5-8
AND/OR sequence (example)	9-9
Areas of application	8-3
Auxiliary register	2-36
B	
Backup copies	1-16
Binary divider	9-12
Block	2-48
- comparison, ICP	6-26
- comparison	8-16
- structure	2-48
- transfer, ICPY	6-30
- transfer, initialize mode	8-15
- transfer, syntax	8-13
Block comparison	
- search mode	8-10
C	
C: down counter	9-41
Cabling, check	4-17
CAL4RM: encoder	9-33
Calling the module	3-17
Carry bit	2-38
CK, SCK: summer/winter time	9-22
Code converter:	
- Binary to Decimal, BID	6-5
- Decimal to Binary, DEB	6-17

Appendix

Index

Communications data	2-15
Comparator, CP	6-16
Compare mode	8-17
Comparison	5-16
Compiling a program	1-15
Compiling the user program	1-15
Conditional	
- bit	5-4
- branches	5-13
- jumps	5-24
- returns	5-37
Constants	2-11
Copying data fields	8-7
Copy mode	8-14
Counter	
- alarm function block, CAU\RM	6-9
- input	2-7
Create a utilisation table	1-9
Cycle time	1-7,4-8
- exceed the cycle time	1-7
- setting the cycle time	1-7
D	
Data block	
Date/time	4-39
- display	4-39
- specify	4-39
Designations	6-4
Destination range	8-11
Device configuration	1-7
- modify	1-7
Diagnostics status word	4-9
Digital inputs	2-7
Direct addressing	8-4
Division	5-18
Edge alarm function block, FALARM	6-19
- encoder with delay	9-38

Appendix

Index

End of module	5-21
End of program	5-22
Exclusive OR	5-56
F	
Fail-safe programming	9-5
FAL ^{ARM} : bottling plant	9-30
FIFO register, status	4-25
First In - First Out	
- FIFOB	6-22
- FIFOW	6-24
Fleeting break contact (examples)	
- constant	9-15
- variable	9-16
Fleeting make contact (examples)	
- constant	9-13
- variable	9-14
Force setting	
- dynamic	1-8
Forcing	4-21
Forward/reverse counter, C	6-6
Function block	
- additional settings	2-26
- behaviour of the inputs	2-27
- call up	2-26
- dataflow	2-25
- definition	2-21
- incorporation into the user program	2-28
- location	2-24
- organisation	2-24
- parameters	2-13
- program sequence	2-25
- retentive	2-31
- shifting of data ranges after inserting new module data	2-33
- status display	4-23
- types of memories and module data storage	2-32

Appendix

Index

G	
Generator function block	6-60
GOR instruction	2-47
I	
ICP/ICPY, check	4-25
INCLUDE instruction	3-5
Incorporating the configuration file	1-14
Indirect addressing	8-4
Indirect compare	8-7
Indirect copy	8-7
Information	
- byteINB	4-14
- data	2-15
Initialize mode	8-15
Initializing data fields	8-7
Inputs	2-6
Inserting reference files	3-7
Inserting source files	3-5
Instruction	
- definition	2-3
- line	2-45
- operand section	2-4
- operation section	2-4
- structure	2-4
Instruction set of the program module	3-16
Intermediate results	2-41
K	
Key to symbols	6-4
L	
Last In - First Out	
- LIFOB (Stack register)	6-34
- LIFOW (Stack register)	6-36
LIFO register, status	4-25
Load	5-26
Load auxiliary register	5-23

Appendix

Index

M	
Main program, structure	2-50
Marker addresses	2-9
Marker range	
- content	4-24
- force setting in RUN	1-8
- retentive, set	1-7
- set	
Markers	2-9
Mode selector switch	4-7
Modifications with the PS 4 200 series in "RUN"	4-29
Multiple program module call	3-13
Multiplication	5-28
N	
Negation	5-31
- of operands	2-18
Nested program structure	3-10
Nesting depth with INCLUDE	3-10
Nesting depth with program modules	3-11
No operation	5-30
Number of elements NO	8-12
Number of function blocks	6-3
O	
Observe marker states	1-8
Off-delayed Timer, TF	6-58
On-delayed Timer, TR	6-62
Online program modification	
- exit	4-33
- handling	4-31
- include files	4-35
- memory requirement	4-34
- program entry	4-32
- program modules	4-37
- restrictions	4-32
- special features	4-32

Appendix

Index

Operand	
- address	8-12
- overview	2-4
- status display	4-23
- types	2-3
Operations	2-18
Operators	2-18
OR	5-32
OR/AND sequence (example)	9-10
Order of program modules	3-15
Output addresses	2-8
Overflow bit	2-38
P	
Parallel bus markers	2-10
Password	
- accessible ranges	1-8
- enter	1-8
- protected ranges	1-8
Peripheral operands	2-13
Plus/minus	2-38
Possible program module call-ups	5-14
Pre-processor instructions	2-53
Program cycle	2-51
Program module	
- advantages	3-3
- call, rules	3-19
- character	3-16
- control function	3-29
- data transfer	3-23
- data transfer with multiple program module calls	3-22
- execution time	3-19
- independent files	3-21
- memory requirement	3-19
- multiple call	3-26

Appendix

Index

- online programming	3-35
- programming rules	3-20
- program sequence	3-31
- recursive call	3-33
- status display	3-38
- test functions	3-35
Program modules as independent files	3-27
Program processing in working register	2-41
Pulse transmitter, TP	6-61
R	
RDAT: Reload data from retentive range	9-21
Real-time clock	2-11
Register overview	2-36
Reload data, Restore Data, RDAT	6-38
Reset	5-34
Return RET	5-36
Retentive markers	4-8
Rotate to the left	5-38 5-39
Rotate to the right	5-40
S	
Save	
- data, SDAT	6-42
SDAT: Saving data in retentive range	9-17
Search mode	8-19
Select start behaviour	1-7
Sequence	2-25
- overview	2-46
Sequential control module, SK	6-44
Set	5-42
Setting system parameters	1-5
Setting real-time clock, SCK	6-40
Shift left with carry	5-46
Shift register	
- SR	6-46
- SRB	6-50
- SRW	6-52
Shift right with carry	5-50

Appendix

Index

Shift to the left	5-44
Shift to the right	5-48
SK sequential control function block	
- alternative branch	7-5,7-11
- AND sequence	7-5
- AND step sequence	7-12
- applications	7-3
- cyclical processing	7-21
- cyclical processing of the step sequence	7-6
- diagnostics	7-22
- initialisation	7-17
- initial step	7-5
- inputs	7-14
- nesting	7-32
- nesting depth: cascading	7-13
- OR sequence	7-5
- OR step sequence (alternative branch)	7-11
- outputs	7-14
- processing	7-3,7-6, 7-19
- representation	7-5, 7-13
- self-maintaining function	7-3
- simultaneous branch (AND sequence)	7-5
- simultaneous branch with synchronization	7-12
- status indication	7-22
- step	7-5
- step condition	7-5
- synchronisation	7-5, 7-31,7-32
- syntax	7-13
- transition	7-4, 7-5
Space	5-4
Stack operations	2-41
Stack register	2-36, 2-39
Status/diagnostics inputs	2-14
Status indication	8-27

Appendix

Index

Status register	2-36
Subtraction	5-52
Summation	8-23
Symbolic operands	2-15
System specific operands	2-13
T	
TALARM:	
encoder	9-34
Encoder with delay	9-36
Test functions	4-5
Time	
- alarm function block, TALARM	6-54
- date Comparator, CK	6-12
- generator, TGEN	6-60
TR	
- clock generator	9-38
- pulse generator	9-39
- rolling shutter control	9-24
- two-point controller with hysteresis	9-26
Transferring a program to the PLC	1-17
U	
Unconditional Jump	5-25
User program	
- activate checksum	1-6
- definition	2-3
- save version number	1-9
V	
Voltage failure, behaviour of the markers	1-8
W	
Working register	2-36
Z	
Zero bit	2-38

ELEV2

Documentation of programs in IL

Program file :elev2.q42

Reference file:elevator.z42

System parameters :

Program check in RUN : No

Start after NOT READY : Halt

Maximum cycle time in ms : 60 ms

Aktive marker range : MB4096

Retentive marker range from : -

to :

-

Retentive marker range from : -

(also after cold start) to : -

Force marker range from : -

to :

-

Version number for user program : 0000

ELEV2

```

UAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
3      3Date      3Name      3Project/Location/Inventory number
3Company/Drawing number      3Page      3
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
3Signed  305/04/20123S. Noval  3Automation process of an elevator system
3Univ. "Vasile Alecsandri" din Bacau 31 3
3Output  3      3      3      3
3Index  3      3      3      3
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAU
♀

```

Directory

Chapter/Page

```

00000 control elevator algorithm 2
00001 $INI Initializare
00002 $E0
00003 ET1
00004 ET2
00005 ET3
00006 ET4
00007 END
00008 $E1
00009 END
00010 $E2
00011 END
00012 $E3
00013 END
00014 $E4
00015 END
00016 $E5
00017 ET6
00018 ET7
00019 ET8
00020 ET9
00021 END
00022 $E6
00023 END
00024 $E7
00025 END
00026 $E8
00027 END
00028 $E9
00029 END
00030 $E10
00031 DOWN
00032 UP
00033 EQUAL
00034 END
00035 $E11
00036 END
00037 $E12
00038 END

```

ELEV2

```

UAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA;
3          3Date          3Name          3Project/Location/Inventory number
3Company/Drawing number          3Page          3
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
3Signed 305/04/20123S. Nova1          3Automation process of an elevator system
3Univ. "Vasile Alecsandri" din Bacau 31          3
3Output          3          3          3          3
3Index 3          3          3          3          3

```

```

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAU
♀3Line 3 Program line 3 Operand comment
3Symb./Oper.3M/B3 Terminal 3 Cross-reference file3
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
300000 "control elevator algorithm 2
3          3          3          3

```

```

3 1 #include "jose.k42"
3          3          3          3
3 2
3          3          3          3
3 3 SK0 -13          Functie de control secvential
3          3          3          3
3 4 S: L K 1          3
3          3          3          3
3 5 R: INB0.0          3
3          3          3          3
3 6 SINO:          3
3          3          3          3
3 7 ERR:          3
3          3          3          3
3 8 SQNO:          3
3          3          3          3
3 9 TG:          3
3          3          3          3
3 10 INIT: $INI          3
3          3          3          3
3 11 AC1: $E0          3
3          3          3          3
3 12 AC2: $E1          3
3          3          3          3
3 13 AC3: $E2          3
3          3          3          3
3 14 AC4: $E3          3
3          3          3          3
3 15 AC5: $E4          3
3          3          3          3
3 16 AC6: $E5          3
3          3          3          3
3 17 AC7: $E6          3
3          3          3          3
3 18 AC8: $E7          3
3          3          3          3
3 19 AC9: $E8          3
3          3          3          3
3 20 AC10: $E9          3
3          3          3          3
3 21 AC11: $E10          3
3          3          3          3
3 22 AC12: $E11          3
3          3          3          3
3 23 AC13: $E12          3
3          3          3          3

```

ELEV2

```

3      24      EP
3
3      00001 $INI      "Initializare
3
3      1      L KB 1
3
3      2      = SK0 SINO      Functie de control secvential
3
3      3      EM
3
3      00002 $E0      "
3
3      1      L K 0
3
3      2      = 'DOWN      Actiuni motor jos
3Q 0.2
3      3      = 'UP      Actiuni motor sus
3Q 0.1
3      4      = M 0.0
3
3      5      = M 0.1
3
3      6      = M 0.2
3
3      7      = M 0.3
3
3      8      L 'L0      sensor level 0
3I 0.0
3      9      JC ET1
3
3      10     L 'L1      sensor level 1
3I 0.1
3      11     JC ET2
3
3      12     L 'L2      sensor level 2
3I 0.2
3      13     JC ET3
3
3      14     L 'L3      sensor level 3
3I 0.3
3      15     JC ET4
3
3      16     JCN END
3
3      00003 ET1      "
3
3      1      L KB 2
3
3      2      = SK0 SINO      Functie de control secvential
3
3      3      JP END
3
3      00004 ET2      "
3
3      1      L KB 3
3
3      2      = SK0 SINO      Functie de control secvential
3
3      3      JP END
3
3      00005 ET3      "
3
3      1      L KB 4
3
3      2      = SK0 SINO      Functie de control secvential
3
3      3      JP END
3

```

ELEV2

```

3 00006 ET4      "
3 3 3 3
3 1      L KB 5
3 3 3 3
3 2      = SK0 SINO      Functie de control secvential
3 3 3 3
3 3      JP END
3 3 3 3
3 00007 END      "
3 3 3 3
3 1      EM
3 3 3 3
3 00008 $E1      "
3 3 3 3
3 1      L KB 0
3 3 3 3
3 2      = 'ALEVEL
3 MB2 3 3 3
3 3      L KB 6
3 3 3 3
3 4      = SK0 SINO      Functie de control secvential
3 3 3 3
3 5      JP END
3 3 3 3
3 00009 END      "
3 3 3 3
3 1      EM
3 3 3 3
3 00010 $E2      "
3 3 3 3
3 1      L KB 1
3 3 3 3
3 2      = 'ALEVEL
3 MB2 3 3 3
3 3      L KB 6
3 3 3 3
3 4      = SK0 SINO      Functie de control secvential
3 3 3 3
3 5      JP END
3 3 3 3
3 00011 END      "
3 3 3 3
3 1      EM
3 3 3 3
3 00012 $E3      "
3 3 3 3
3 1      L KB 2
3 3 3 3
3 2      = 'ALEVEL
3 MB2 3 3 3
3 3      L KB 6
3 3 3 3
3 4      = SK0 SINO      Functie de control secvential
3 3 3 3
3 5      JP END
3 3 3 3
3 00013 END      "
3 3 3 3
3 1      EM
3 3 3 3
3 00014 $E4      "
3 3 3 3
3 1      L KB 3
3 3 3 3
3 2      = 'ALEVEL
3 MB2 3 3 3
3 3      L KB 6
3 3 3 3

```

3	4	= SK0 SINO	ELEV2	
3			Functie de control	secvential
3	5	JP END		
3	00015	END	"	
3	1	EM		
3	00016	\$E5	"	
3	1	L 'P0	Buton level 0	
3	I 0.4			
3	2	JC ET6		
3	3	L 'P1	Buton level 1	
3	I 0.5			
3	4	JC ET7		
3	5	L 'P2	Buton level 2	
3	I 0.6			
3	6	JC ET8		
3	7	L 'P3	Buton level 3	
3	I 0.7			
3	8	JC ET9		
3	9	JCN END		
3	00017	ET6	"	
3	1	L KB 7		
3	2	= SK0 SINO	Functie de control	secvential
3	3	JP END		
3	00018	ET7	"	
3	1	L KB 8		
3	2	= SK0 SINO	Functie de control	secvential
3	3	JP END		
3	00019	ET8	"	
3	1	L KB 9		
3	2	= SK0 SINO	Functie de control	secvential
3	3	JP END		
3	00020	ET9	"	
3	1	L KB 10		
3	2	= SK0 SINO	Functie de control	secvential
3	3	JP END		
3	00021	END	"	
3	1	EM		
3	00022	\$E6	"	
3	1	L KB 0		

ELEV2

```

3      2      = 'LEVEL      3
3MB1      3      L K  1      3
3      4      = M 0.0      3
3      5      L KB 11      3
3      6      = SK0 SINO      3      Functie de control secvential
3      7      JP  END      3
300023 END      "      3
3      1      EM      3
300024 $E7      "      3
3      1      L KB 1      3
3MB1      2      = 'LEVEL      3
3      3      L K  1      3
3      4      = M 0.1      3
3      5      L KB 11      3
3      6      = SK0 SINO      3      Functie de control secvential
3      7      JP  END      3
300025 END      "      3
3      1      EM      3
300026 $E8      "      3
3      1      L KB 2      3
3MB1      2      = 'LEVEL      3
3      3      L K  1      3
3      4      = M 0.2      3
3      5      L KB 11      3
3      6      = SK0 SINO      3      Functie de control secvential
3      7      JP  END      3
300027 END      "      3
3      1      EM      3
300028 $E9      "      3
3      1      L KB 3      3
3MB1      2      = 'LEVEL      3
3      3      L K  1      3
3      4      = M 0.3      3
3      5      L KB 11      3

```

3	6	= SK0 SINO	ELEV2	
3			Funcție de control	secvential
3	7	JP END		
3				
3	00029	END	"	
3				
3	1	EM		
3				
3	00030	\$E10	"	
3				
3	1	L 'LEVEL		
3	MB1			
3	2	CP 'ALEVEL		
3	MB2			
3	3	BLT DOWN		
3				
3	4	BGT UP		
3				
3	5	BE EQUAL		
3				
3	6	JP END		
3				
3	00031	DOWN	"	
3				
3	1	L KB 12		
3				
3	2	= SK0 SINO	Funcție de control	secvential
3				
3	3	JP END		
3				
3	00032	UP	"	
3				
3	1	L KB 13		
3				
3	2	= SK0 SINO	Funcție de control	secvential
3				
3	3	JP END		
3				
3	00033	EQUAL	"	
3				
3	1	L KB 1		
3				
3	2	= SK0 SINO	Funcție de control	secvential
3				
3	3	JP END		
3				
3	00034	END	"	
3				
3	1	EM		
3				
3	00035	\$E11	"	
3				
3	1	L K 1		
3				
3	2	= 'DOWN	Actiuni motor jos	
3	Q 0.2			
3	3	L 'L0	sensor level 0	
3	I 0.0			
3	4	A M 0.0		
3				
3	5	= M 0.4		
3				
3	6	L 'L1	sensor level 1	
3	I 0.1			
3	7	A M 0.1		
3				
3	8	= M 0.5		
3				

3	9	L 'L2	ELEV2	
3	I 0.2	3 3	sensor level 2	3
3	10	A M 0.2		3
3	11	= M 0.6		3
3	12	L 'L3	sensor level 3	
3	I 0.3	3 3		3
3	13	A M 0.3		3
3	14	= M 0.7		3
3	15	L M 0.4		3
3	16	O M 0.5		3
3	17	O M 0.6		3
3	18	O M 0.7		3
3	19	JCN END		3
3	20	L KB 1		3
3	21	= SK0 SINO	Funcție de control secvential	3
3	00036 END	"		3
3	1	EM		3
3	00037 \$E12	"		3
3	1	L K 1		3
3	2	= 'UP	Actiuni motor sus	3
3	Q 0.1	3 3		3
3	3	L 'L0	sensor level 0	
3	I 0.0	3 3		3
3	4	A M 0.0		3
3	5	= M 0.4		3
3	6	L 'L1	sensor level 1	
3	I 0.1	3 3		3
3	7	A M 0.1		3
3	8	= M 0.5		3
3	9	L 'L2	sensor level 2	
3	I 0.2	3 3		3
3	10	A M 0.2		3
3	11	= M 0.6		3
3	12	L 'L3	sensor level 3	
3	I 0.3	3 3		3
3	13	A M 0.3		3
3	14	= M 0.7		3
3	15	L M 0.4		3
3	16	O M 0.5		3
3	17	O M 0.6		3
3	18	O M 0.7		3

```

3          3 3          3          3          3          3
3          3 3          3          3          3          3
3 19      3 3          3          3          3          3
3          3 3          3          3          3          3
3 20      3 3          3          3          3          3
3          3 3          3          3          3          3
3 21      3 3          3          3          3          3
3          3 3          3          3          3          3
300038 3 3          3          3          3          3
3 END    3 3          3          3          3          3
3 1      3 3          3          3          3          3
3          3 3          3          3          3          3

```

ELEV2

JCN END

L KB 1

= SK0 SINO

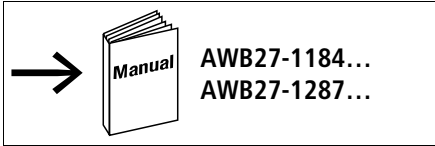
"

EM

Funcție de control secvențial

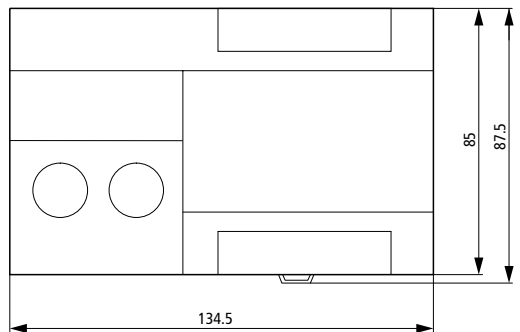
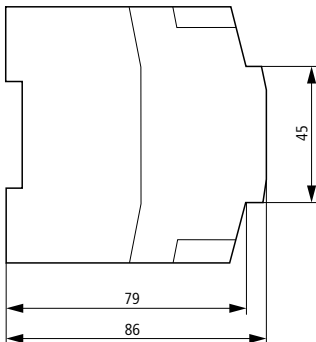
P R O G R A M E N D

PS4-201-MM1



→ Das Gerät ist für den industriellen Einsatz geeignet (→ EN 55011/22 Klasse A).
The device is suitable for use in industrial environments (→ EN 55011/22 Class A).
L'appareil a été conçu pour l'emploi en milieu industriel (→ EN 55011/22 classe A).
L'apparecchio è adatto per uso in ambienti industriali (→ EN 55011/22 Classe A).
El aparato es adecuado para uso en ambiente industrial (→ EN 55011/22 clase A).

Abmessungen – Dimensions – Dimensioni – Dimensiones [mm]

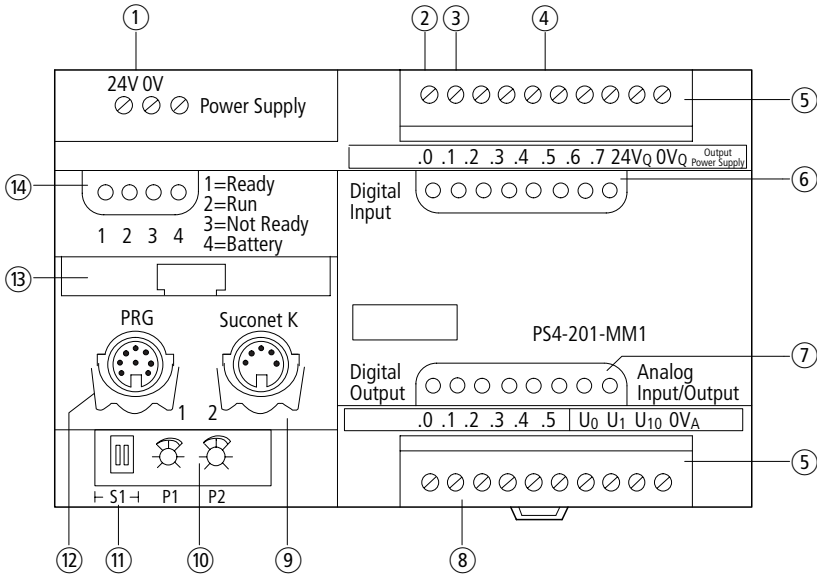


UL/CSA

Power Supply	# of channels and Input Rating	# of channels and Output Rating
24 V DC	8 digital	6 digital
0.8 A	24 V DC	24 V DC, 0.5 A
	2 analog	24 V DC, 0.5 A p. d. ¹⁾
		1 analog

1) p. d. Pilot Duty/Maximum operating temperature 55 °C/
Tightening torque 0.6 Nm/AWG 12-28

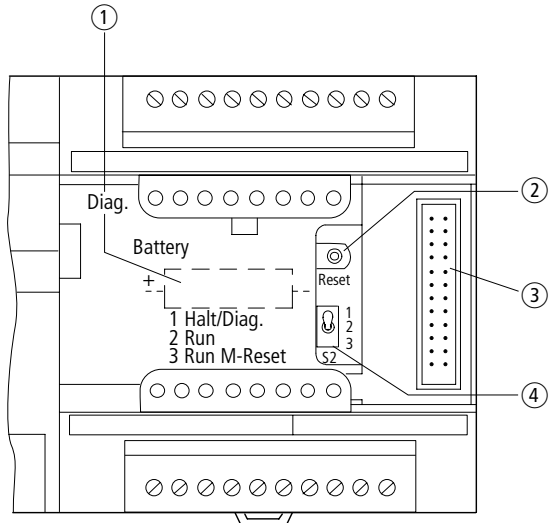
Frontansicht – Front view – Face avant – Vista frontale – Vista de frente



- | | | |
|---|---|--|
| ① 24-V-DC-Stromversorgung | ⑥ Statusanzeige der Eingänge | ⑩ Sollwertgeber P1, P2 |
| ② Eingang: „Schneller Zähler“, (3 kHz) | ⑦ Statusanzeige der Ausgänge | ⑪ Schalter S1 für Busabschlusswiderstände |
| ③ Alarmeingang | ⑧ 6 Digital-Ausgänge
24 V DC/0,5 A; kurzschlussfest und überlastsicher | ⑫ Programmiergeräte-Schnittstelle (PRG)/seriell transparent (RS 232) |
| ④ 8 Digital-Eingänge
24 V DC und 24-V-DC-Versorgung für die Ausgänge | ⑨ Suconet-K-Schnittstelle/seriell transparent (RS 485) | ⑬ Speichermodul |
| ⑤ Steckbare Schraubklemme | | ⑭ Statusanzeige der Steuerung |

**Offene Gehäuseklappe
Housing cover open
Couvercle ouverte
Calotta della custodia aperta
Tapa abierta**

- | |
|--------------------------------------|
| ① Batterie |
| ② Reset-Taste |
| ③ Stiftheiste für Lokale Erweiterung |
| ④ Betriebsarten-Vorwahlschalter S2 |



- ① 24 V DC power supply
- ② Input high-speed counter (3 kHz)
- ③ Alarm input
- ④ 8 24 V DC digital inputs and 24 V DC supply for the outputs
- ⑤ Plug-in screw terminal
- ⑥ LED status display for the inputs
- ⑦ LED status display for the outputs
- ⑧ 6 24 V DC/0.5 A digital outputs; short-circuit proof and overload protected 2 analog inputs (0 to 10 V) 1 analog output (0 to 10 V)
- ⑨ Suconet K interface/serial transparent (RS 485)
- ⑩ Setpoint potentiometer P1, P2
- ⑪ S1 Switch for bus terminating resistors
- ⑫ Programming device interface (PRG)/ serial transparent (RS 232)
- ⑬ Memory module
- ⑭ LED status display of the PLC

-
- ① Battery
 - ② Reset button
 - ③ Terminal for local expansion
 - ④ Mode selector switch S2

- ① 24 V DC alimentazione
- ② Ingresso contatore veloce 3 kHz
- ③ ingresso interrupt
- ④ 8 ingressi digitali 24 V DC e 24 V DC alimentazione per uscite
- ⑤ Morsetto a vite sfilabile
- ⑥ Visualizzazione di stato a LED degli ingressi
- ⑦ Visualizzazione di stato a LED degli uscite
- ⑧ 6 uscite digitali 24 V DC/0,5 A; protetto da cortocircuito e sovraccarico 2 ingressi analogici (0 a 10 V) 1 uscita analogica (0 a 10 V)
- ⑨ Interfaccia Suconet K/in serie trasparente (RS 485)
- ⑩ Potenziometro P1, P2
- ⑪ Interruttore S1 resistenze di terminazione bus
- ⑫ Interfaccia di programmazione (PRG)/ in serie trasparente (RS 232)
- ⑬ Modulo di memoria
- ⑭ Visualizzazione a LED di PLC

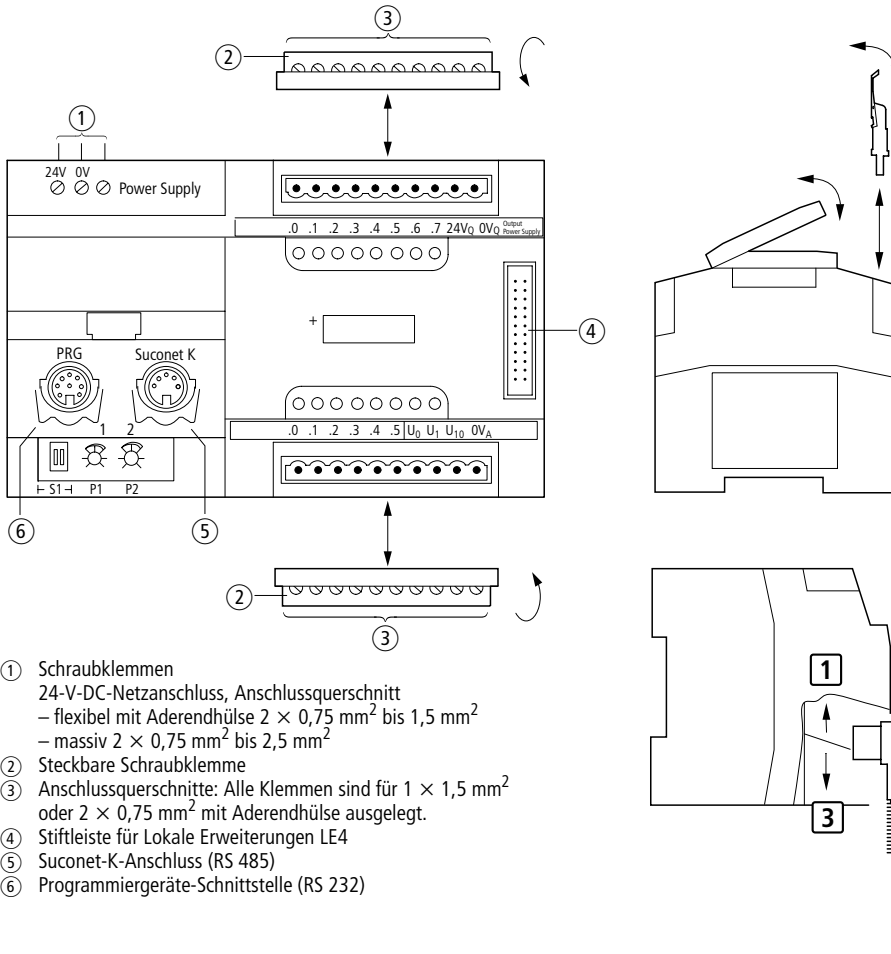
-
- ① Batteria
 - ② Tasto di reset
 - ③ Connettore per espansioni locali
 - ④ Selettore modo di funzionamento S2

- ① Alimentation secteur 24 V CC Terre de protection
- ② Entrée compteur rapide (3 kHz)
- ③ Entrée d'alarme
- ④ 8 entrées digitales 24 V CC et 24 V CC alimentation pur sorties
- ⑤ Bornier à vis enfichable
- ⑥ Afficheur d'état DEL entrées
- ⑦ Afficheur d'état DEL sorties
- ⑧ 6 sorties digitales 24 V CC/0,5 A; protection courts-circuits et surcharges 2 entrées analogiques (0 à 10 V) 1 sortie analogique (0 à 10 V)
- ⑨ Liaison Suconet K/séquentiel transparent (RS 485)
- ⑩ Module d'entrées de consignes
- ⑪ S1 interrupteur pour résistance de terminaison de bus
- ⑫ Liaison pour appareils de programmation (PRG)/ séquentiel transparent (RS 232)
- ⑬ Module de mémoire
- ⑭ Afficheur d'état DEL de l'automate

-
- ① Pile
 - ② Bouton RAZ
 - ③ Connecteur pour extensions locales
 - ④ Sélecteur modes de fonctionnement S2

- ① 24 V DC alimentación
- ② Entrada de contador rapido 3 kHz
- ③ Entrada de alarma
- ④ 8 entradas digitales 24 V DC y 24 V DC alimentación para las salidas
- ⑤ Terminal roscado enchufable
- ⑥ LED visualización entradas
- ⑦ LED visualización salidas
- ⑧ 6 salidas digitales 24 V DC/0,5 A; a prueba de cortocircuitos y seguridad contra sobrecargas 2 entradas analógicas (0 a 10 V) 1 salida analógica (0 a 10 V)
- ⑨ Interface Suconet K/en serie transparente (RS 485)
- ⑩ Encoder
- ⑪ Interruptor S1 para bus resistencias terminales
- ⑫ Interface aparatos de programación (PRG)/ en serie transparente (RS 232)
- ⑬ Módulo de memoria
- ⑭ LED visualización del PLC

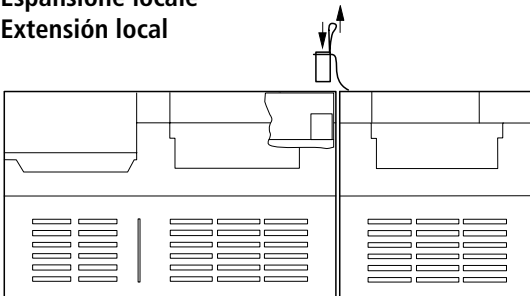
-
- ① Pila
 - ② Pulsador Reset
 - ③ Regleta de bornes para extensiones locales
 - ④ Selector de modo de servicio S2



- ① Schraubklemmen
24-V-DC-Netzanschluss, Anschlussquerschnitt
– flexibel mit Aderendhülse $2 \times 0,75 \text{ mm}^2$ bis $1,5 \text{ mm}^2$
– massiv $2 \times 0,75 \text{ mm}^2$ bis $2,5 \text{ mm}^2$
- ② Steckbare Schraubklemme
- ③ Anschlussquerschnitte: Alle Klemmen sind für $1 \times 1,5 \text{ mm}^2$
oder $2 \times 0,75 \text{ mm}^2$ mit Aderendhülse ausgelegt.
- ④ Stiftleiste für Lokale Erweiterungen LE4
- ⑤ Suconet-K-Anschluss (RS 485)
- ⑥ Programmiergeräte-Schnittstelle (RS 232)

06/01 AWA27-1589

Lokale Erweiterung
Local expansion
Extension locale
Espansione locale
Extensión local



Achtung!

Buchsenstecker nur im spannungslosen Zustand stecken oder ziehen.

Care!

Always switch off the power supply when fitting or removing the socket connector.

Attention !

Le connecteur ne doit être branché ou débranché qu' hors tension.

Attenzione!

Inserire o togliere il connettore solo a tensione disinserta.

¡Atención!

Enchufar o desenchufar al conector hembra sólo sin tensión.

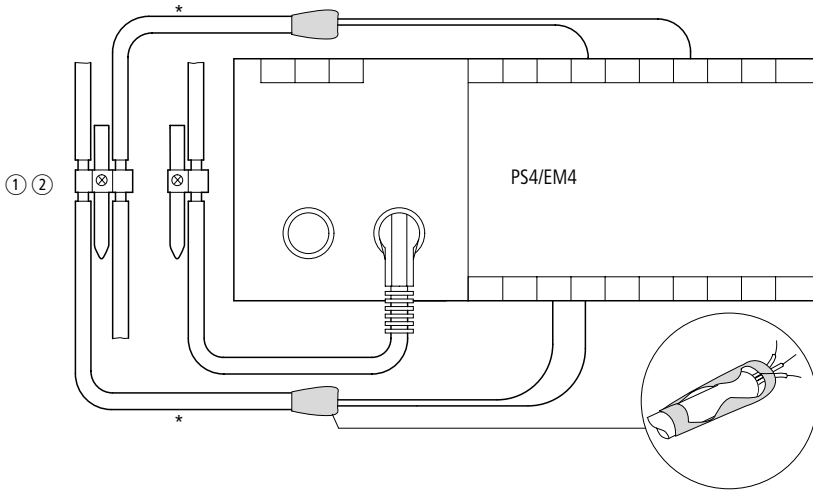
- ① Screw terminal
24 V DC power supply
Connection cross-section:
– flexible with ferrule: $2 \times 0.75 \text{ mm}^2$ to 1.5 mm^2
– without ferrule $2 \times 0.75 \text{ mm}^2$ to 2.5 mm^2
- ② Plug-in screw terminal
- ③ Connection cross-sections: All terminals are designed for $1 \times 1.5 \text{ mm}^2$ or $2 \times 0.75 \text{ mm}^2$ with ferrule
- ④ Terminal for LE4 local expander units
- ⑤ Suconet K connection (RS 485)
- ⑥ Programming device interface (RS 232)

- ① Morsetti a vite
Alimentazione 24 V DC
Sezione del cavo
– flessibile, con guaina $2 \times 0,75 \text{ mm}^2$ a $1,5 \text{ mm}^2$
– rigido $2 \times 0,75 \text{ mm}^2$ a $2,5 \text{ mm}^2$
- ② Morsetto a vite sfilabile
- ③ Sezione del cavo: tutti i morsetti sono utilizzabili per $1 \times 1,5 \text{ mm}^2$ oppure $2 \times 0,75 \text{ mm}^2$ con guaina
- ④ Connettore per espansioni locali LE4
- ⑤ Collegamento Suconet K (RS 485)
- ⑥ Interfaccia di programmazione (RS 232)

- ① Bornier à vis
Alimentation secteur 24 V CC
Section de raccordement :
– avec embout : $2 \times 0,75 \text{ mm}^2$ à $1,5 \text{ mm}^2$
– sans embout : $2 \times 0,75 \text{ mm}^2$ à $2,5 \text{ mm}^2$
- ② Bornier à vis enfichable
- ③ Section de raccordement : Toutes les bornes sont conçues pour une section $1 \times 1,5 \text{ mm}^2$ ou $2 \times 0,75 \text{ mm}^2$ et douille d'embout.
- ④ Connecteur pour extensions locales LE4
- ⑤ Raccordement Suconet K (RS 485)
- ⑥ Liaison pour appareils de programmation (RS 232)

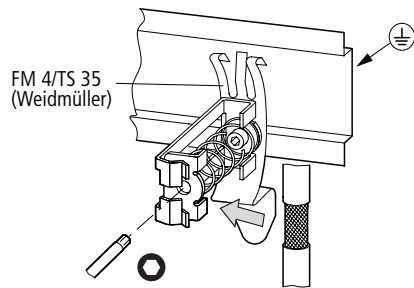
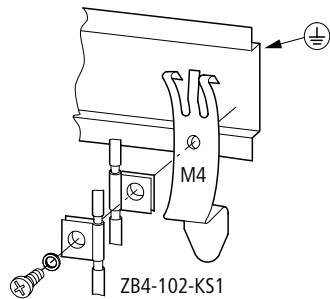
- ① Terminales roscados
Alimentación 24 V DC, Secciones de conexión:
– flexible con casquillo $2 \times 0,75 \text{ mm}^2$ a $1,5 \text{ mm}^2$
– macizo $2 \times 0,75 \text{ mm}^2$ a $2,5 \text{ mm}^2$
- ② Terminal roscado
- ③ Secciones de conexión: todos los terminales están dimensionados para $1 \times 1,5 \text{ mm}^2$ o bien $2 \times 0,75 \text{ mm}^2$ con casquillo
- ④ Regleta de bornes para extensiones locales LE4
- ⑤ Conexión Suconet K (RS 485)
- ⑥ Interface aparatos de programación (RS 232)

**Schirmerdung – Earthing the screen – Mise à la terre du blindage –
Collegamento alla terra dello schermo – Conexión a tierra de pantalla**

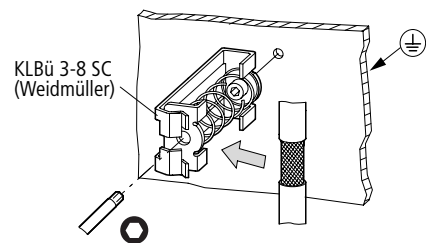
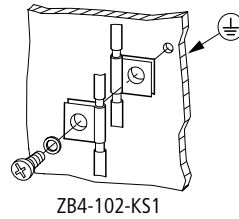


* Signalleitungen (abhängig vom Modul) – Signal cable (depending on module) – Ligne de signaux (en fonction du module) – Conduttore del segnale (dipendente dal modulo) – Línea de señalización (en función del módulo)

① für Hutschiene – for top-hat rail – pour profilé-support – per guida – para guía simétrica



② für Montageplatte – for mounting plate – pour plaque de montage – per piastra di montaggio – para placa de montaje



06/01 AWA27-1589

Alternative Schirmerdung – Alternative screen earth – Mise à la terre du blindage au choix – Collegamento alternativo dello schermo a terra – Puesta a tierra alternativa de la pantalla

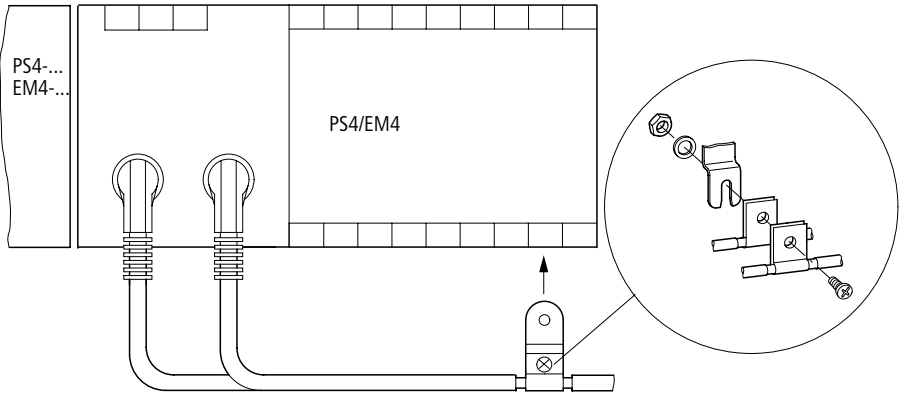
Falls die Schirmerdung auf Seite 6/10 aus Platzgründen nicht möglich ist.

In the event that the screen earthing arrangement on Page 6/10 is not possible due to lack of space.

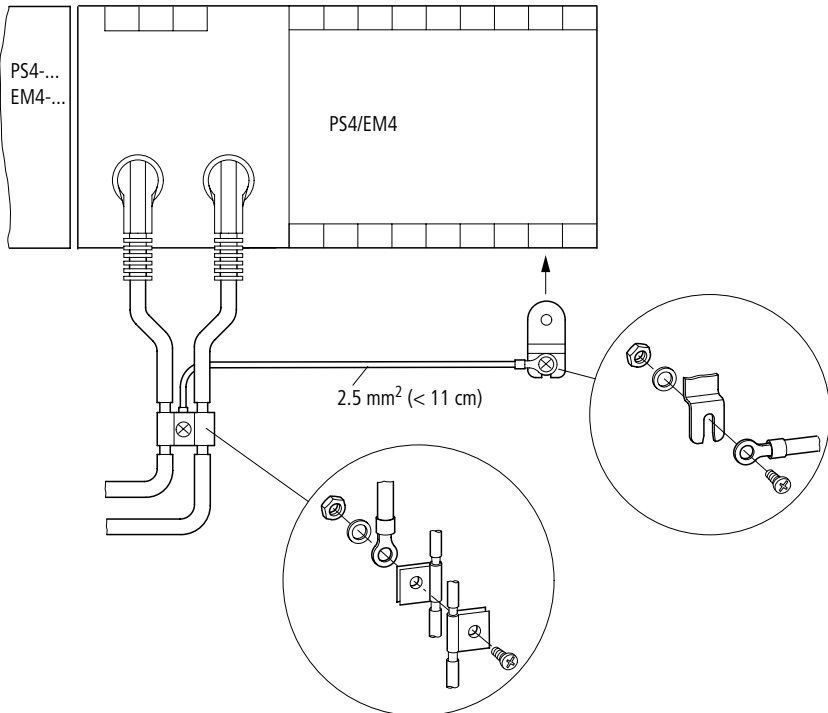
Au cas où la mise à la terre du blindage en page 6/10 est trop encombrante.

Se il collegamento a terra dello schermo di pag. 6/10 richiede troppo spazio.

En caso de que la puesta a tierra de la pantalla en página 6/10 no sea posible por razones de espacio.



06/01 AWA27-1589



Busabschlusswiderstände – Bus terminating resistors – Résistances de terminaison de bus – Resistenci di terminazione bus – Resistencias terminales de bus

Schalterstellung im Auslieferungszustand

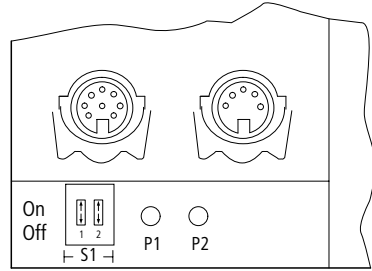
Factory setting

Position à la livraison

Impostazione di fabbrica

Posición de entrega

S1: 1 ON
2 ON



Batterie – Batery – Pile – Batteria – Pila



Achtung!

Nur im eingeschalteten Zustand stecken oder ziehen.

Attention!

Only fit or remove if switched on.

Attention !

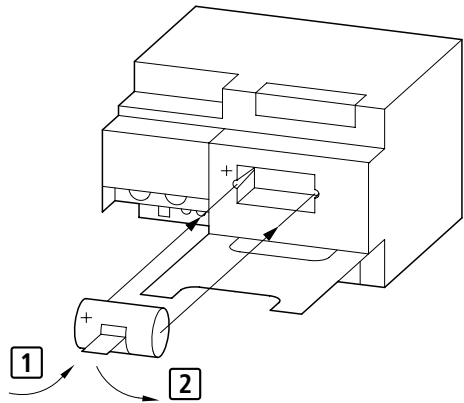
Ni enficher ni retirer que lorsque l'appareil est sous tension.

Attenzione!

Inserire/togliere solo a tensione inserita.

¡Atención!

Meter o sacar sólo con tensión.



06/01 AWA27-1589

Speichermodul – Memmory Module – Module de mémoire – Modulo di memoria – Módulo de memoria



Achtung!

Nur im spannungslosen Zustand stecken oder ziehen.

Attention!

Always switch off the power supply when fitting or removing.

Attention !

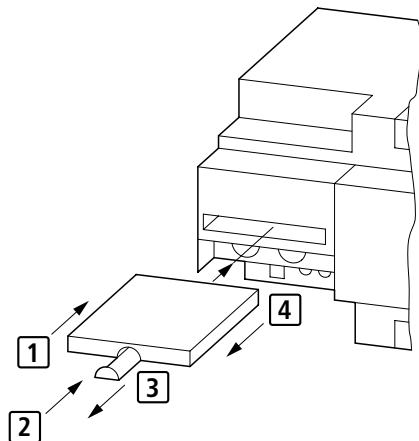
Brancher/débrancher uniquement hors tension.

Attenzione!

Inserire o togliere solo a tensione disinserita.

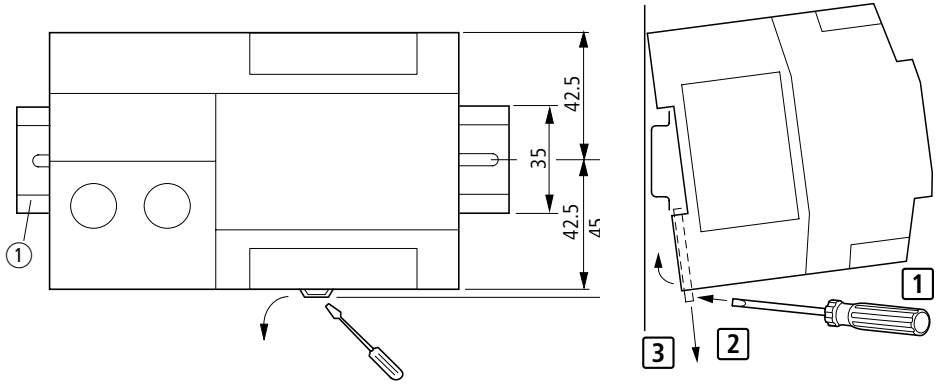
¡Atención!

Enchufar o desenchufar sólo sin tensión.



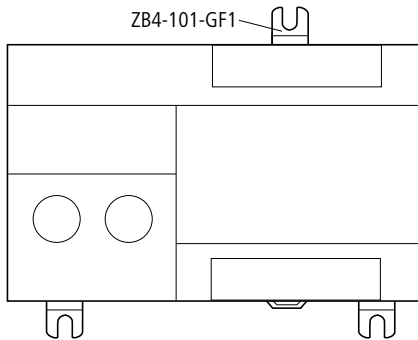
Montage – Fitting – Montaggio – Montaje

auf Montageplatte mit 35-mm-Hutschiene ① (senkrecht ohne LE oder waagrecht)
on mounting plate with 35 mm top-hat rail ① (vertical without LE or horizontal)
sur plaque de montage avec profilé-support 35 mm ① (vertical sans LE ou horizontal)
su piastra di montaggio con guida DIN 35 mm ① (verticale senza LE o orizzontale)
sobre placa de montaje con guía simétrica de 35 mm ① (vertical sin LE ó horizontal)



06/01 AWA27-1589

auf Montageplatte (senkrecht ohne LE oder waagrecht)
on mounting plate (vertical without LE or horizontal)
sur plaque de montage (vertical sans LE ou horizontal)
su piastra di montaggio (verticale senza LE o orizzontale)
sobre placa de montaje (vertical sin LE ó horizontal)



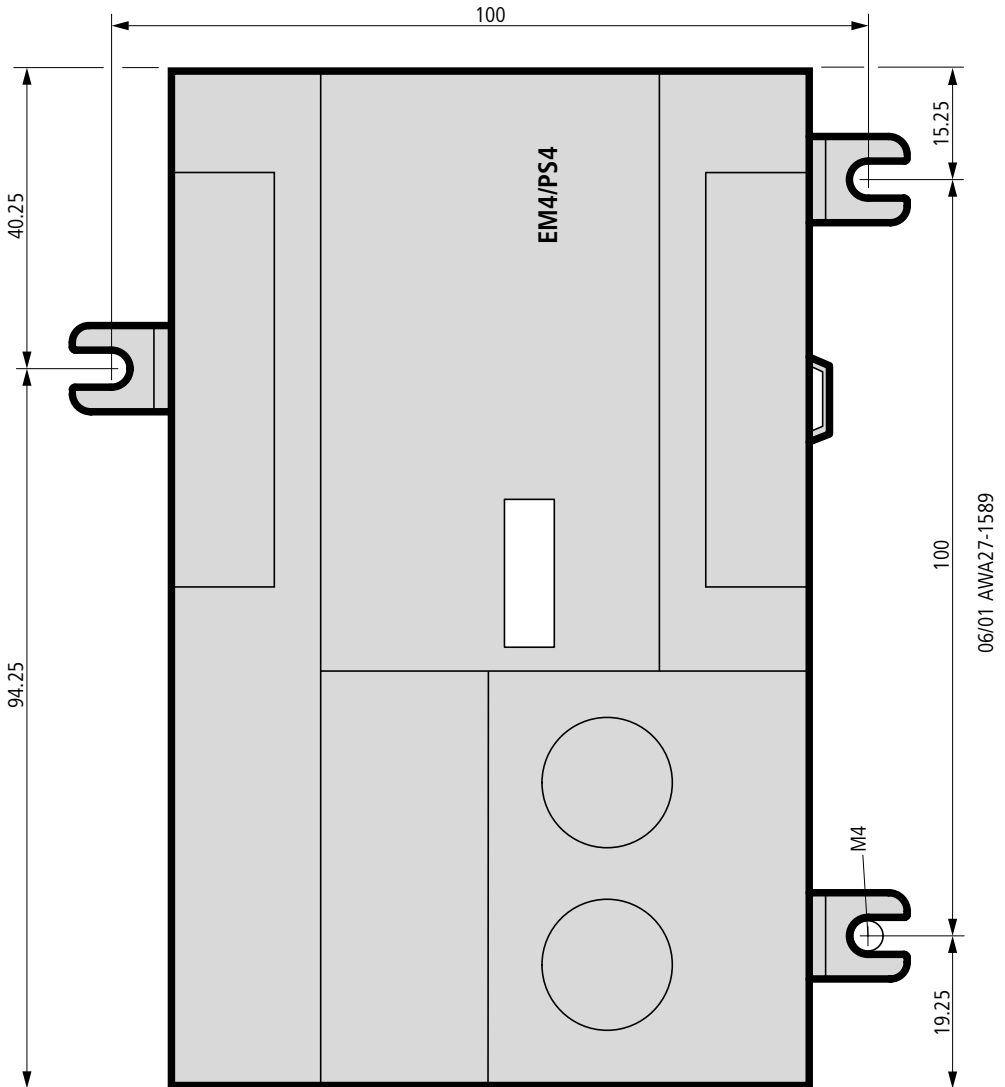
Bohrschablone M 1 : 1

Template for holes, scale 1 : 1

Gabarit de perçage, échelle 1 : 1

Dima di foratura, scala 1 : 1

Plantilla para taladros, escala 1 : 1

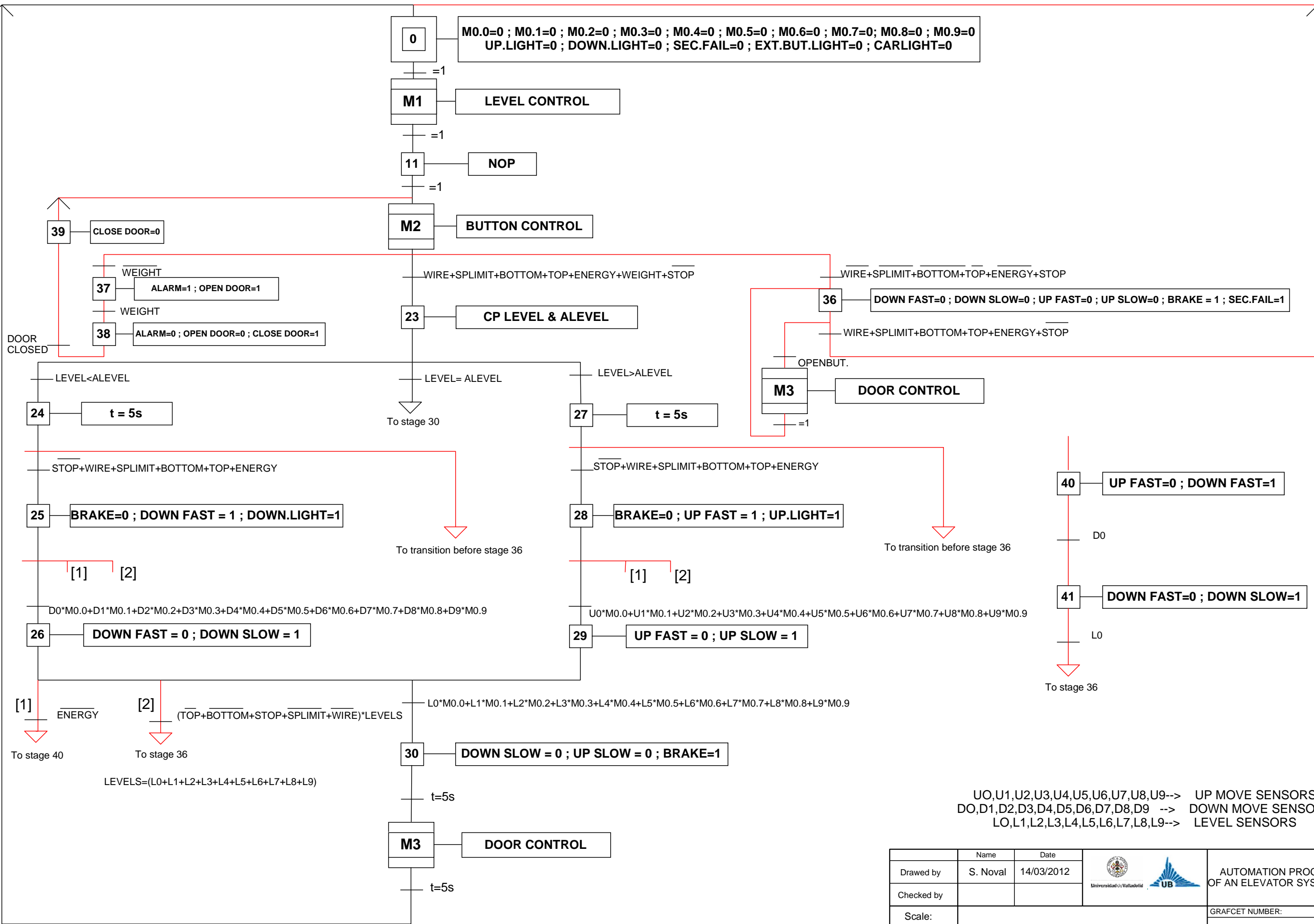


06/01 AWA27-1589

ELEVATOR

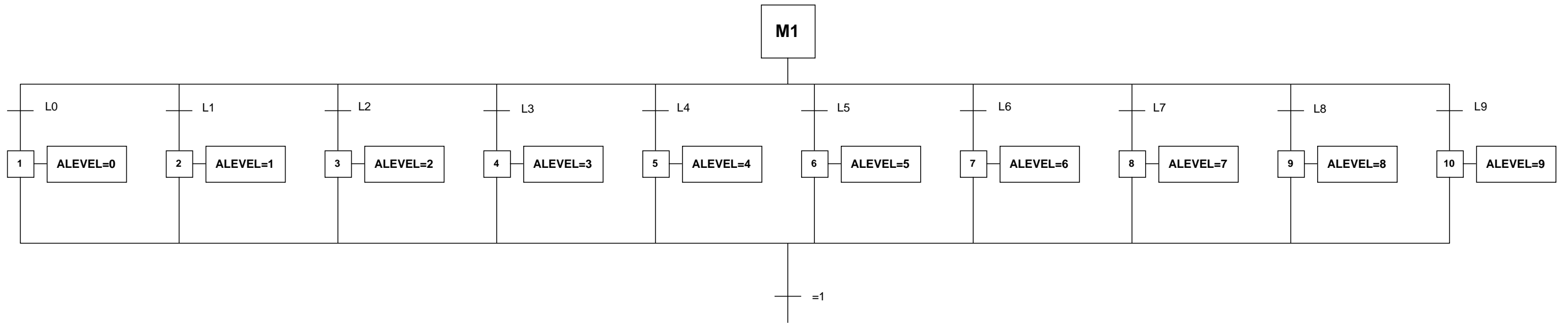
" Printout of file: c:elevator.z42 Date: 3. 4.12"

Symbol	Operand	M/B	Terminal	Operand	comment
L0	I 0.0			Sensor level 0	
L1	I 0.1			Sensor level 1	
L2	I 0.2			Sensor level 2	
L3	I 0.3			Sensor level 3	
P0	I 0.4			Buton level 0	
P1	I 0.5			Buton level 1	
P2	I 0.6			Buton level 2	
P3	I 0.7			Buton level 3	
DOWN	Q 0.1			Actiuni motor jos	
UP	Q 0.2			Actiuni motor sus	
LEVEL	MB1				
ALEVEL	MB2				
	SK0				Funcție de control secvential
	INB0.0				
	M 0.0				
	M 0.1				
	M 0.2				
	M 0.3				
	M 1.1				

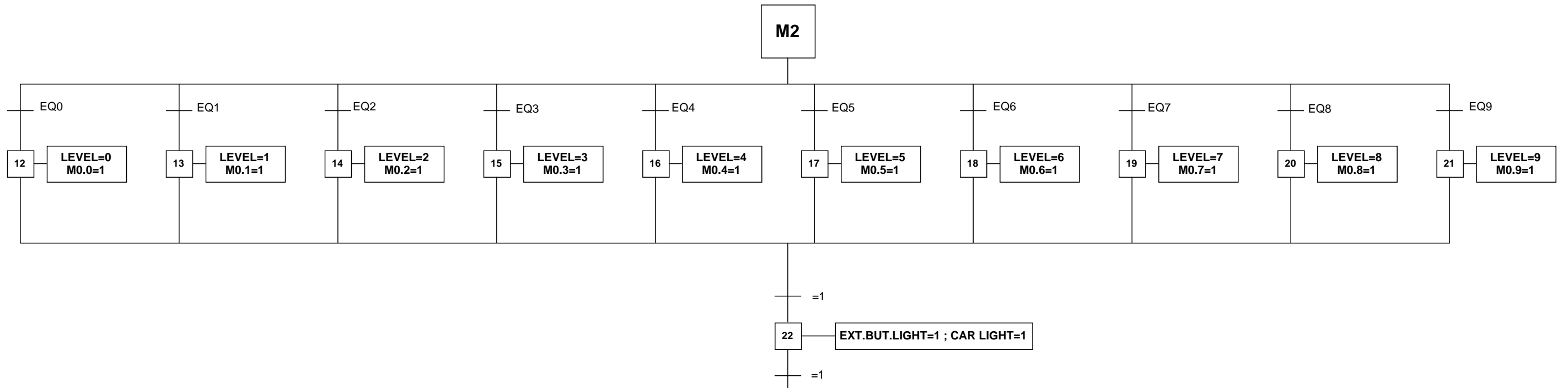


U0,U1,U2,U3,U4,U5,U6,U7,U8,U9--> UP MOVE SENSORS
 D0,D1,D2,D3,D4,D5,D6,D7,D8,D9 --> DOWN MOVE SENSORS
 L0,L1,L2,L3,L4,L5,L6,L7,L8,L9--> LEVEL SENSORS

	Name	Date		AUTOMATION PROCESS OF AN ELEVATOR SYSTEM	
Drawn by	S. Noval	14/03/2012			
Checked by					
Scale:	GRAFSET MAIN CONTROL			GRAFSET NUMBER:	1

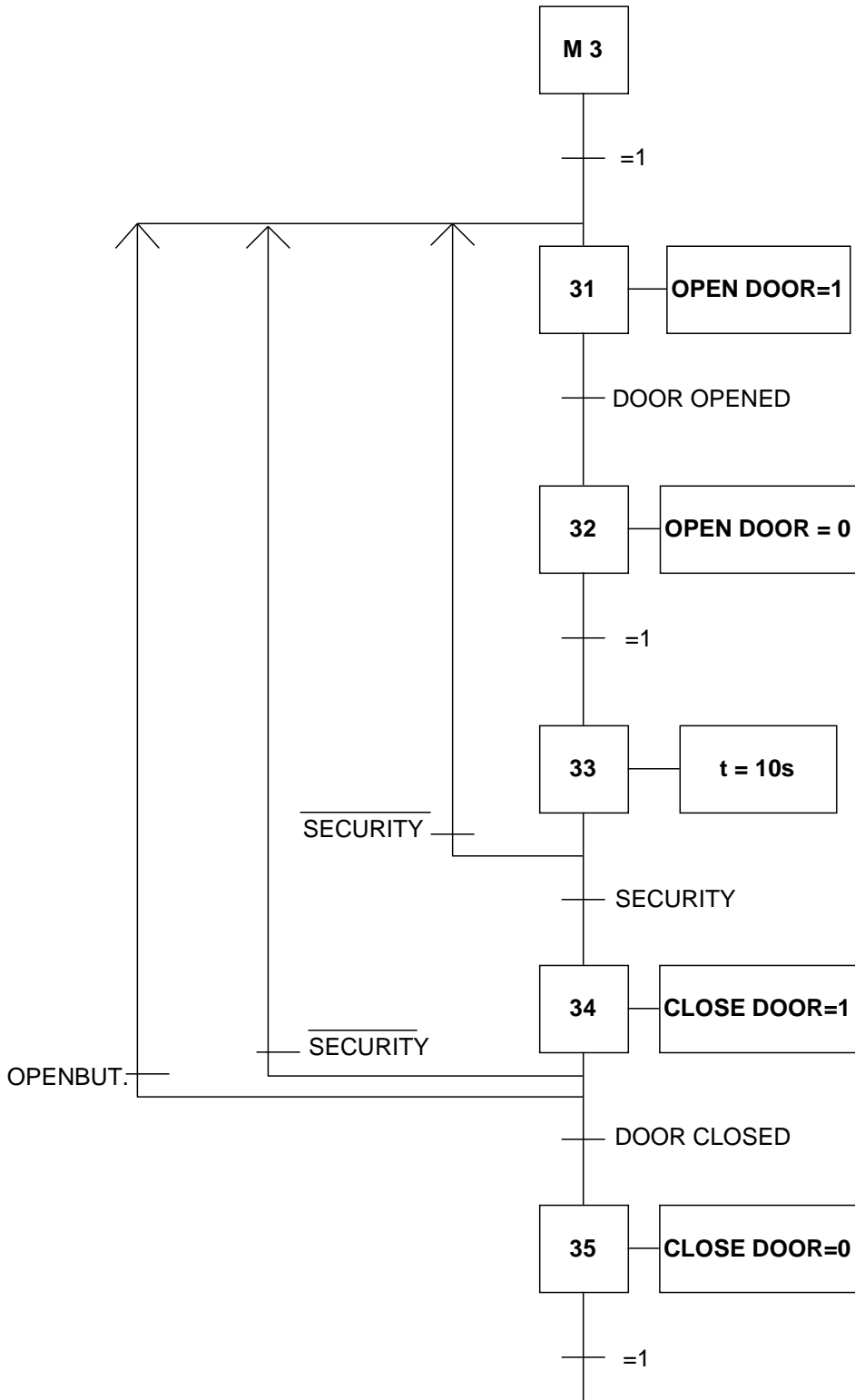


	Name	Date	 	AUTOMATION PROCESS OF AN ELEVATOR SYSTEM
Drawn by	S.Noval	14/03/2012		
Checked by				
Scale:	GRAFCET LEVEL CONTROL			GRAFCET NUMBER:
				2

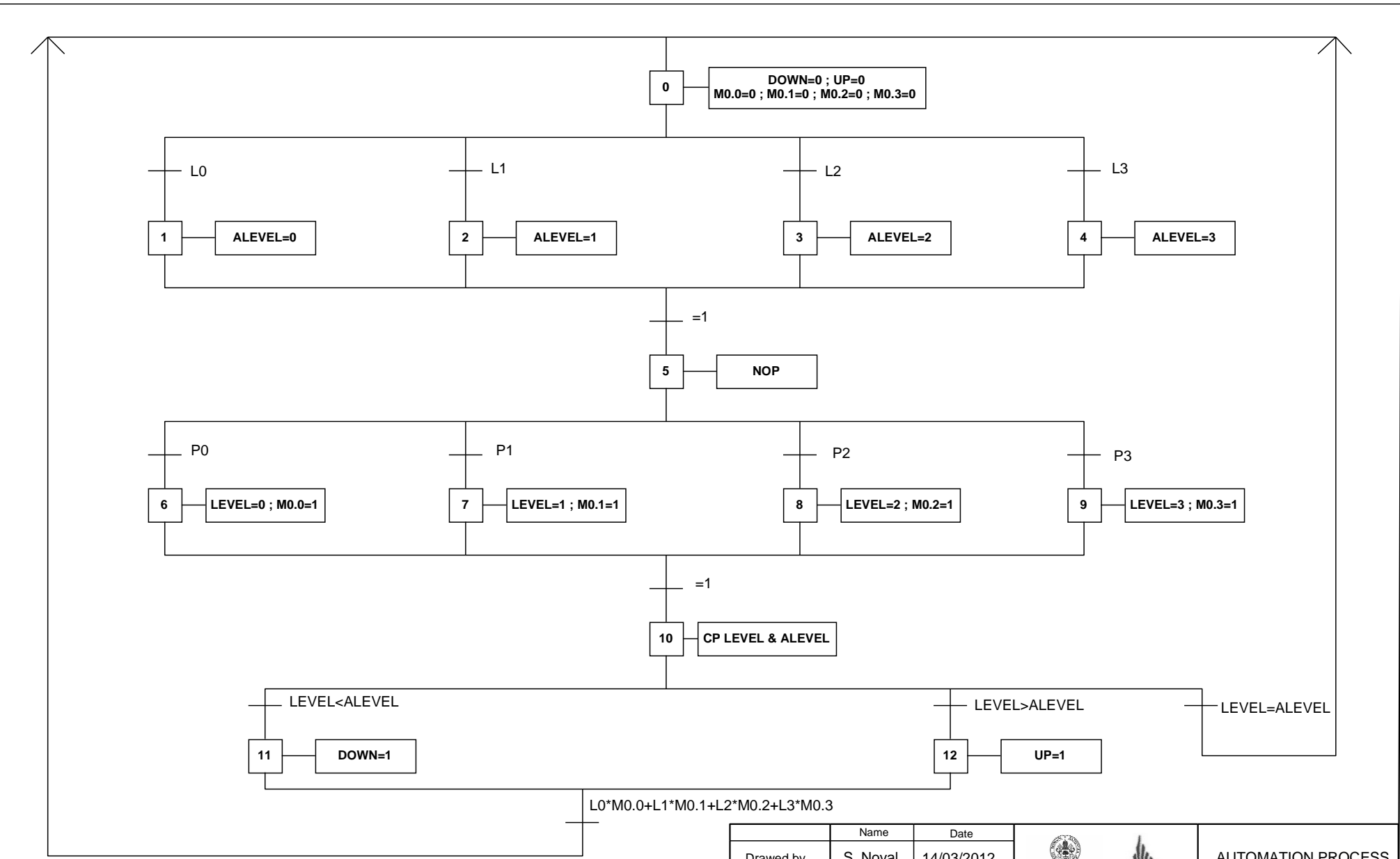


$EQ0=A0+P0(\overline{A0+A1+A2+A3+A4+A5+A6+A7+A8+A9})$
 $EQ1=A1+P1(\overline{A0+A1+A2+A3+A4+A5+A6+A7+A8+A9})$
 $EQ2=A2+P2(\overline{A0+A1+A2+A3+A4+A5+A6+A7+A8+A9})$
 $EQ3=A3+P3(\overline{A0+A1+A2+A3+A4+A5+A6+A7+A8+A9})$
 $EQ4=A4+P4(\overline{A0+A1+A2+A3+A4+A5+A6+A7+A8+A9})$
 $EQ5=A5+P5(\overline{A0+A1+A2+A3+A4+A5+A6+A7+A8+A9})$
 $EQ6=A6+P6(\overline{A0+A1+A2+A3+A4+A5+A6+A7+A8+A9})$
 $EQ7=A7+P7(\overline{A0+A1+A2+A3+A4+A5+A6+A7+A8+A9})$
 $EQ8=A8+P8(\overline{A0+A1+A2+A3+A4+A5+A6+A7+A8+A9})$
 $EQ9=A9+P9(\overline{A0+A1+A2+A3+A4+A5+A6+A7+A8+A9})$

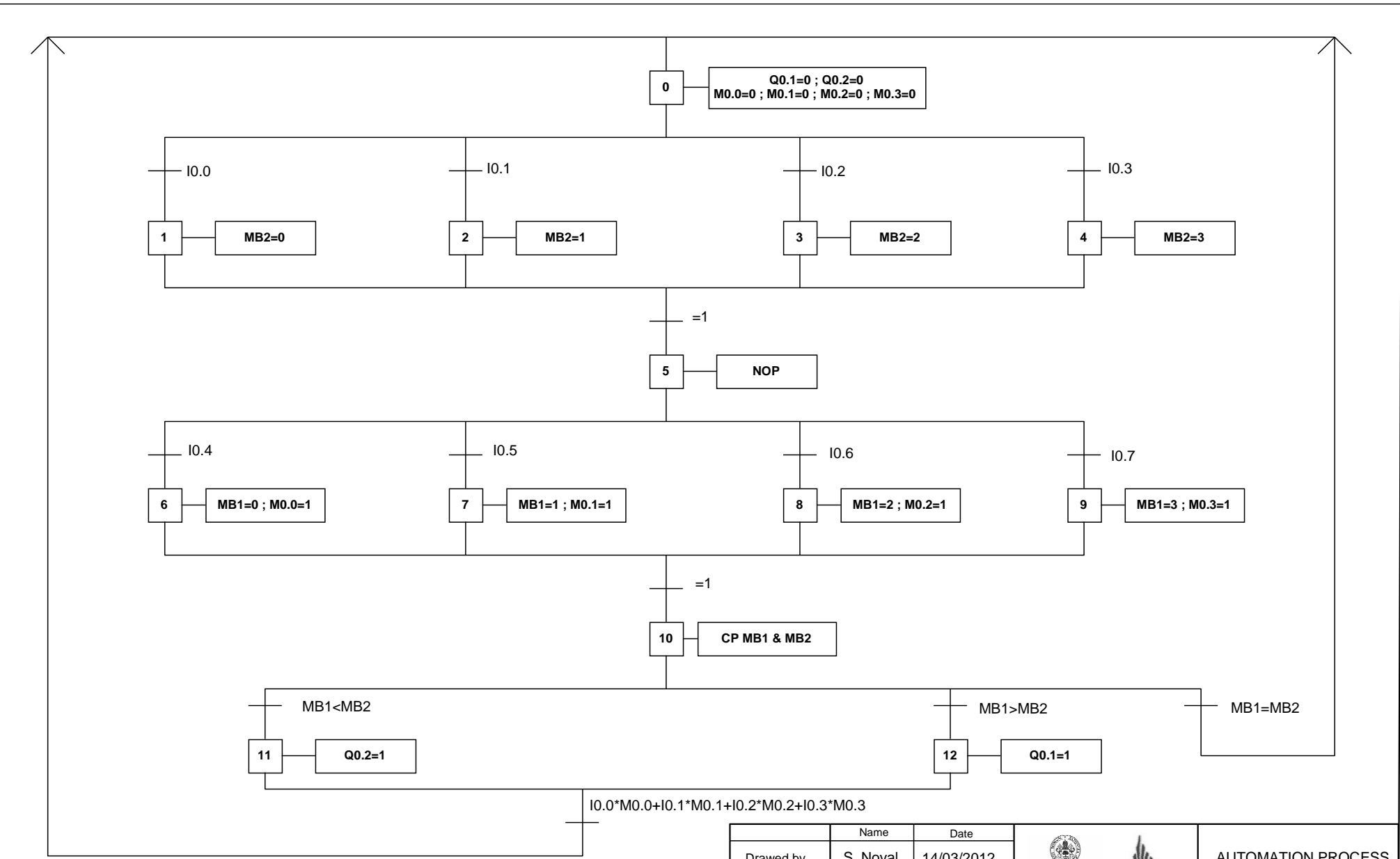
	Name	Date		AUTOMATION PROCESS OF AN ELEVATOR SYSTEM
Drawn by	S. Noval	14/03/2012		
Checked by				GRAFSET NUMBER:
Scale:	GRAFSET BUTTON CONTROL			3




	Name	Date	 	AUTOMATION PROCESS OF AN ELEVATOR SYSTEM
Drawn by	S. Noval	14/03/2012		
Checked by			GRAFSET NUMBER:	
Scale:	GRAFSET DOOR CONTROL			4



	Name	Date		AUTOMATION PROCESS OF AN ELEVATOR SYSTEM
Drawn by	S. Noval	14/03/2012		
Checked by				GRAF CET NUMBER:
Scale:	GRAF CET SIMULATION LEVEL 1			5



	Name	Date		AUTOMATION PROCESS OF AN ELEVATOR SYSTEM
Drawn by	S. Noval	14/03/2012		
Checked by			GRAF CET NUMBER:	
Scale:	GRAF CET SIMULATION LEVEL 2			6