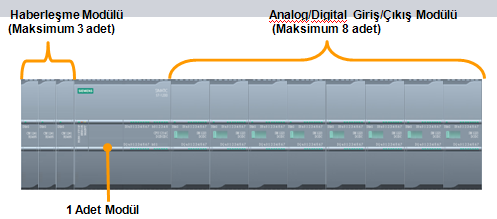
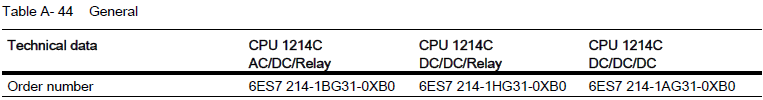
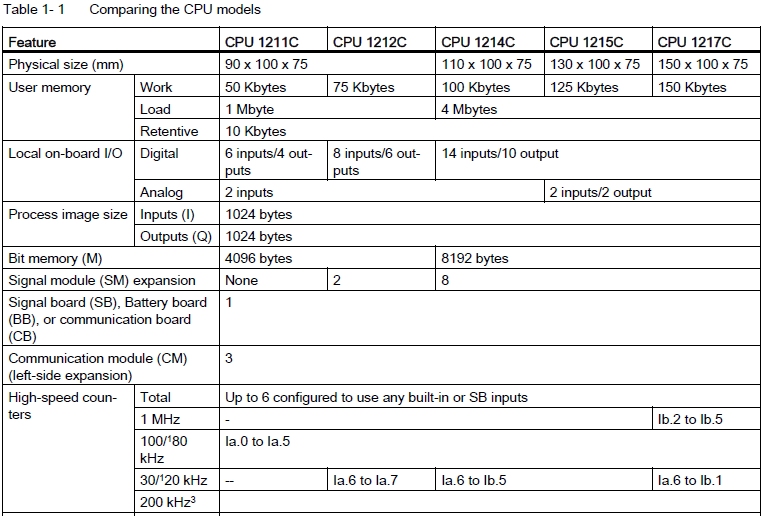
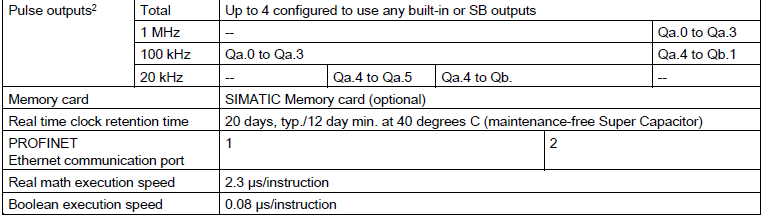
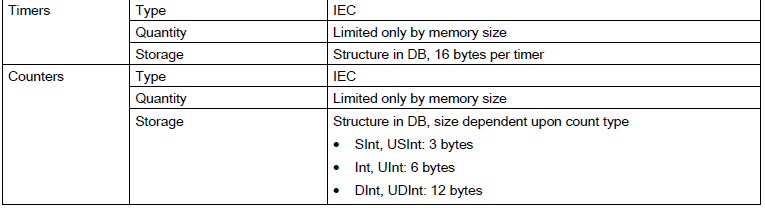
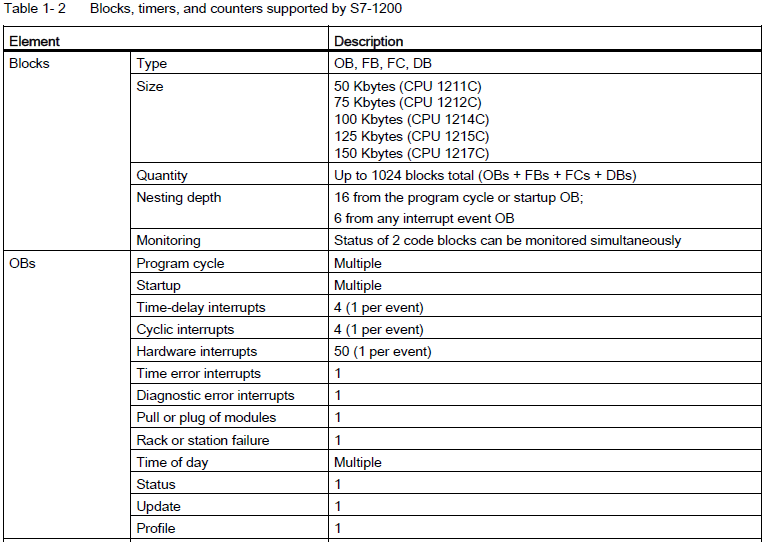
**S7-1200**

****



****

****

****

**CPU Hafıza Alanları**

Tablo 1.1’de gösterildiği gibi CPU üzerinde aşağıda kısaca açıklanan 3 ayrı hafıza alanı vardır.

**Load:** Kullanıcı programının, veri ve konfigürasyon ayarlarının saklandığı kalıcı (non-volatile) türde hafızadır. CPU üzerinde veya hafıza kartında tanımlanabilir.

**Work:** Kullanıcı programının yürütülmesi esnasında çeşitli verilerin saklandığı kalıcı olmayan (volatile) hafızadır. Enerji kesildiğinde bu alanda tutulan veriler kaybolur.

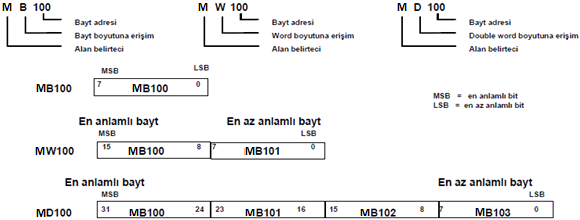
**Retentive:** Enerji kesilmesi durumunda “work” hafızanın önceden belirlenmiş olan sınırlı bir kısmının saklandığı kalıcı hafızadır. Enerji geldiğinde “Retentive” hafızadaki veriler ilgili “work” hafızaya geri transfer edilir.

**Hafıza Alanlarına Erişim**

PLC’ler bilgiyi müstakil bir adrese sahip değişik veri alanlarında saklar. Erişilmek istenilen hafıza adresi

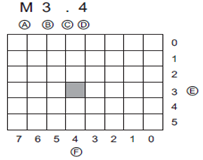
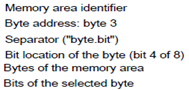
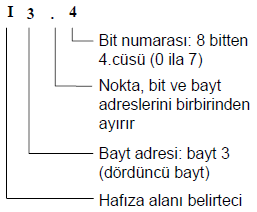
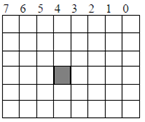
kesin olarak ifade edilebilir. Bu şekilde program içerisinde hafıza alanının herhangi bir adresinde tutulan veriye doğrudan erişilebilir. **(Mutlak Adresleme: I0.0, Q1.5, MB10, MW84 vb.)**

Mutlak adreslemede, (i)hafıza alanı belirteci (I, Q, M), (ii)veri alanı büyüklüğü (B,W,D) ve (iii) veri başlangıç adresi belirtilir.

****

Bit adreslere erişimde ise, (i) hafıza alanı belirteci (I, Q, M), (ii) bayt adresi ve (iii) bit adresi belirtilir.

Örnek: I0.7, Q0.1, M3.4

**STEP-7 sembolik programlamayı da desteklemektedir.** Yani kullanıcı hafıza alanlarına, giriş/çıkış birimlerine başlangıçta ilgili birimin kullanımını anlamlaştıracak özel isim (etiket, tag) atayıp program içerisinde bu özel isimle hafıza alanlarına, giriş/çıkış birimlerine erişebilir. (***Kapı\_1, Tank\_ust, Pt100\_0, vb***)

**Giriş İmge Kütüğü (Process Image Input, PII): I**

CPU her çevrimde tüm aktif fiziksel girişleri okur ve bu değerleri giriş imge kütüğü olarak tanımlanan hafıza alanına yazar. Giriş kütüğüne bit, byte, word veya double word olarak erişilebilir.

Bit: I*[byte adresi].[bit adresi]* I0.1

Byte, Word veya Double Word: I*[boyut][başlangıç byte adresi]* IB4, IW2, ID12

Ix.y:P ifadesi ile fiziksel girişlere doğrudan/ani erişilebilir. Ix.y:P ile erişilen girişlerin PII’deki değeri değişmez.

**Çıkış İmge Kütüğü (Process Image Output, PIQ): Q**

Kullanıcı programının yürütülmesi esnasında çıkış değerleri ilk önce çıkış imge kütüğü olarak tanımlanan hafıza alanına yazılır. Her çevrimin sonunda çıkış imge kütüğünde bulunan değerler fiziksel çıkışlara yazılır. Çıkış imge kütüğüne bit, byte, word veya double word olarak erişebilirsiniz:

Bit: Q*[byte addresi].[bit adresi]* Q1.1

Byte, Word veya Double Word: Q*[boyut][başlangıç bayt adresi]* QB5 QW10, QD40

Qx.y:P ifadesi ile fiziksel çıkışlara doğrudan/ani erişilebilir. Qx.y:P ile erişilen çıkışların hem fiziksel çıkış değeri hemde PIQ’deki değeri değişir.

**Bit Hafıza Alanı: M**

Bit hafıza alanını (M hafıza) herhangi bir işlemin ara değerini tutmak veya yardımcı röle olarak kullanabilir. M hafıza alanına bit, byte, word veya double word olarak erişilebilir.

Bit: M*[byte adresi].[bit adresi]* M26.7

Byte, Word veya Double Word: V*[boyut][başlangıç bayt adresi]* MB20, MW30, MD40

**Hafıza Alanları**

**Genel Hafıza (Global Memory):** Tüm kod blokları (OB, FC, FB) tarafından erişilebilen hafıza alanlarıdır. Giriş (I), Çıkış (Q) ve bit hafıza (M) global hafıza türündedir. Mutlak ve sembolik adresleme ile erişilebilir.

**Geçici Hafıza (Temp Memory): Sadece sembolik adresleme ile erişilebilir.** Whenever a code block is called, the operating system of the CPU allocates the temporary, or local, memory (L) to be used during the execution of the block. Access to the data in temp memory is restricted to the OB, FC, or FB that created or declared the temp memory location. When the execution of the code block finishes, the CPU reallocates the local memory for the execution of other code blocks. Temp memory locations remain local and are not shared by different code blocks, even when the code block calls another code block. For example: When an OB calls an FC, the FC cannot access the temp memory of the OB that called it.

The allocation of temp memory for a code block might reuse the same temp memory locations previously used by a different OB, FC or FB. The CPU does not initialize the temp memory at the time of allocation and therefore the temp memory might contain any value.

The CPU provides temp (local) memory for each of the three OB priority groups:

● 16 Kbytes for startup and program cycle, including associated FBs and FCs

● 4 Kbytes for standard interrupt events including FBs and FCs

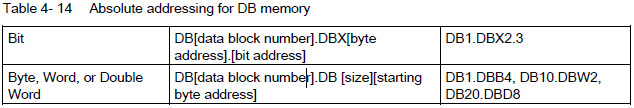
● 4 Kbytes for error interrupt events including FBs and FCs

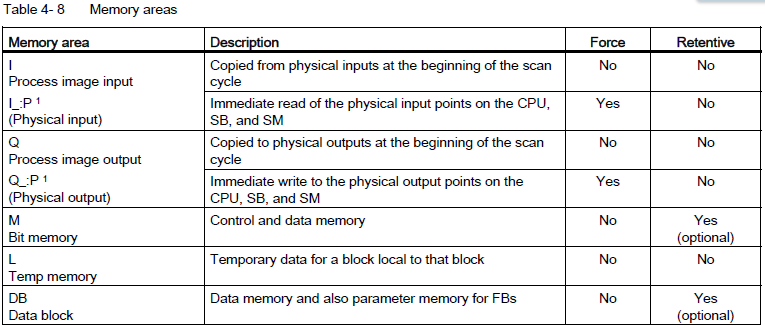
**Veri Bloğu (Data Block, DB):** You can include DBs in your user program to store data for the code blocks.

The data stored persists when the execution of the associated code block comes to an end.

A "global" DB stores data that can be used by all code blocks, while an instance DB stores data for a specific FB and is structured by the parameters for the FB.

Use the DB memory for storing various types of data, including intermediate status of an operation or other control information parameters for FBs, and data structures required for many instructions such as timers and counters. You can access data block memory in bits, bytes, words, or double words. Both read and write access is permitted for read/write data blocks. Only read access is permitted for read-only data blocks.



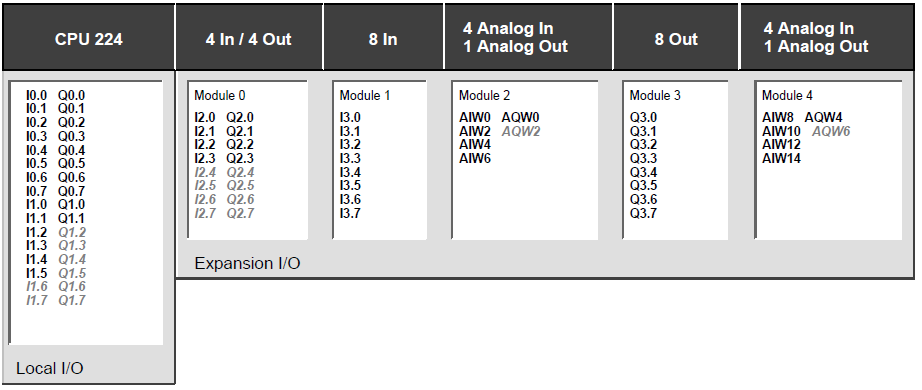


**I/O Adresleme (CPU ve I/O modülleri)**

When you add a CPU and I/O modules to your configuration screen, I and Q addresses are automatically assigned. You can change the default addressing by selecting the address field in the configuration screen and typing new numbers.

-Digital inputs and outputs are assigned in groups of 8 points (1 byte), whether the module uses all the points or not.

-Analog inputs and outputs are assigned in groups of 2 points (4 bytes).



**Organization blocks (OBs), Functions (FCs)** and **function blocks (FBs)**

The CPU supports the following types of code blocks that allow you to create an efficient structure for your user program: ***Organization blocks (OBs), Functions (FCs)*** *and* ***function blocks (FBs)***

The size of the user program, data, and configuration is limited by the available load memory and work memory in the CPU. There is no specific limit to the number of each individual OB, FC, FB and DB block. However, the total number of blocks is limited to 1024.

**Organization blocks (OBs);** They serve as the interface between the operating system and the user program. Some OBs have predefined behavior and start events, but you can also create OBs with custom start events.

Execution of the user program begins with one or more optional start-up organization blocks (OBs) which are executed once upon entering RUN mode, followed by one or more program cycle OBs which are executed cyclically. An OB can also be associated with an interrupt event, which can be either a standard event or an error event, and executes whenever the corresponding standard or error event occurs

OBs control the execution of the user program.

Each OB must have a unique OB number.

The default OB numbers are reserved below 200. Other OBs must be numbered 200 or greater.

OBs cannot call each other or be called from an FC or FB.

Specific events (such as a diagnostic interrupt or a time interval) in the CPU trigger the execution of an organization block.

The CPU handles OBs according to their respective priority classes, with higher priority OBs executed before lower priority OBs.

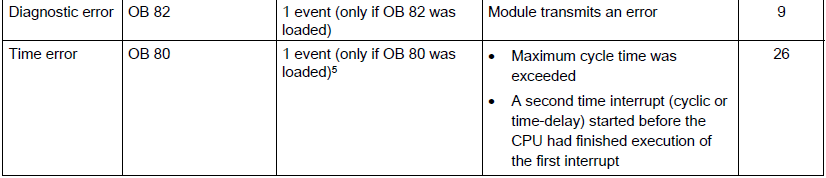
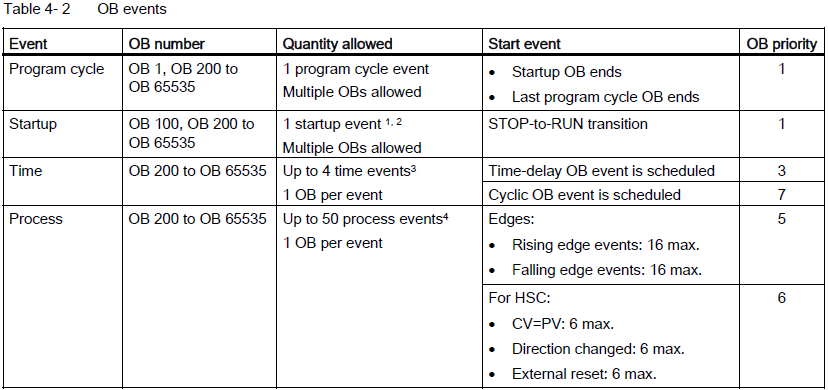
The lowest priority class is 1 (for the main program cycle), and the highest priority class is 26 (for the time-error interrupts).

**Startup OBs** execute one time when the operating mode of the CPU changes from STOP to RUN.

Multiple startup OBs are allowed. **OB 100** is the default. Others must be OB 200 or greater. After completion startup, the main "Program cycle" OB will begin executing.

**The program cycle OB** contains your main program. This is where you place the instructions that control

your program and where you call additional user blocks. Program cycle OBs execute cyclically while the CPU is in RUN mode. Multiple program cycle OBs are allowed. You can include more than one program cycle OB in your user program. They are executed in numerical order. OB 1 is the default. Other program cycle OBs must be identified as OB 200 or greater. During RUN mode, the program cycle OBs execute at the lowest priority level and can be interrupted by all other types of program processing.



**Cyclic interrupt OBs** execute at a specified interval.

A cyclic interrupt OB will interrupt cyclic program execution at user defined intervals, such as every 2 seconds. You can configure up to a total of 4 for both the time-delay and cyclic events at any given time,

with one OB allowed for each configured time-delay and cyclic event. The OB must be OB 200 or greater.

**Hardware interrupt OBs** execute when the relevant hardware event occurs, including rising and falling edges on built-in digital inputs and HSC events. A hardware interrupt OB will interrupt normal cyclic program execution in reaction to a signal from a hardware event. You define the events in the properties of the hardware configuration. One OB is allowed for each configured hardware event. The OB must be OB 200 or greater.

**A time error interrupt OB** executes when either the maximum cycle time is exceeded or a time error event occurs. The OB for processing the time error interrupt is OB 80. If triggered, it executes, interrupting normal cyclic program execution or any other event OB. The events that trigger the time error interrupt and the reaction of the CPU to those events are described below:

– Exceeding the maximum cycle time: You configure the maximum cycle time in the properties of the CPU. If OB 80 does not exist, the reaction of the CPU for exceeding the maximum time is to change to STOP.

– Time errors: If OB 80 does not exist, the reaction of the CPU is to stay in RUN. Time errors occur if the time of day event is missed or repeated, the queue overflows, or an event OB (time delay event, time of day event, or cyclic interrupt) starts before the CPU finishes the execution of the first.

The occurrence of either of these events generates a diagnostic buffer entry describing the event. The diagnostic buffer entry is generated regardless of the existence of OB 80.

**Diagnostic error interrupt OBs** execute when a diagnostic error is detected and reported.

A diagnostic OB interrupts the normal cyclic program execution if a diagnostics-capable module recognizes an error (if the diagnostic error interrupt has been enabled for the module).

OB 82 is the only OB number supported for the diagnostic error event. You can include an STP instruction (put CPU in STOP mode) inside your OB 82 if you desire your CPU to enter STOP mode upon receiving this type of error. If there is no diagnostic OB in the program, the CPU ignores the error (stays in RUN).

**The CPU processing is controlled by events.**

Some events happen on a regular basis like the program cycle or cyclic events.

Other events happen only a single time, like the startup event and time delay events.

Some events happen when there is a change triggered by the hardware, such as an edge event on an input point or a high speed counter event.

There are also events like the diagnostic error and time error event which only happen when there is an error. The event priorities and queues are used to determine the processing order for the event interrupt OBs.

**The program cycle even**t happens once during each program cycle (or scan). During the program cycle, the CPU writes the outputs, reads the inputs and executes program cycle OBs. The program cycle event is required and is always enabled. You may have no program cycle OBs, or you may have multiple OBs selected for the program cycle event. After the program cycle event is triggered, the lowest numbered program cycle OB (usually OB 1) is executed. The other program cycle OBs are executed sequentially (in numerical order) within the program cycle.

**The cyclic interrupt** events allow you to configure the execution of an interrupt OB at a configured scan time. The initial scan time is configured when the OB is created and selected to be a cyclic interrupt OB. A cyclic event will interrupt the program cycle and execute the cyclic interrupt OB (the cyclic event is at a higher priority class than the program cycle event).

**The time delay interrupt** events allow you to configure the execution of an interrupt OB after a specified delay time has expired. The delay time is specified with the SRT\_DINT instruction. The time delay events will interrupt the program cycle to execute the time delay interrupt OB. Only one time delay interrupt OB can be attached to a time delay event. The CPU supports four time delay events.

**The hardware interrupt** events are triggered by a change in the hardware, such as a rising or falling edge on an input point, or a HSC (High Speed Counter) event. There can be one interrupt OB selected for each hardware interrupt event. The hardware events are enabled in Device configuration. The OBs are specified for the event in the Device configuration or with an ATTACH instruction in the user program. The CPU supports several hardware interrupt events. The exact events are based on the CPU model and the number of input points.

**The time and diagnostic error interrupt** events are triggered when the CPU detects an error.

These events are at a higher priority class that the other interrupt events and can interrupt the execution of the time delay, cyclic and hardware interrupt events. One interrupt OB can be specified for each of the time error and diagnostic error interrupt events.

**Understanding event execution priorities and queuing**

The number of pending (queued) events from a single source is limited, using a different queue for each event type. Upon reaching the limit of pending events for a given event type, the next event is lost. Refer to the following section on "Understanding time error events" for more information regarding queue overflows.

Each CPU event has an associated priority. You cannot change the priority of an OB. In general, events are serviced in order of priority (highest priority first). Events of the same priority are serviced on a "first-come, first-served" basis.

1 The startup event and the program cycle event will never occur at the same time because the startup event will run to completion before the program cycle event will be started (controlled by the operating system).

2 Only the diagnostic error event (OB 82) interrupts the startup event. All other events are queued to be processed after the startup event has finished.

3 The CPU provides a total of 4 time events that are shared by the time-delay OBs and the cyclic OBs. The number of time-delay and cyclic OBs in your user program cannot exceed 4.

4 You can have more than 50 process events if you use the DETACH and ATTACH instructions.

5 You can configure the CPU to stay in RUN if the maximum scan cycle time was exceeded or you can use the RE\_TRIGR instruction to reset the cycle time. However, the CPU goes to STOP mode the second time that the maximum scan cycle time was exceeded in one scan cycle.

After the execution of an OB with a priority of 2 to 25 has started, processing of that OB cannot be interrupted by the occurrence of another event, except for by OB 80 (time-error event, which has a priority of 26). All other events are queued for later processing, allowing the current OB to finish.

**Interrupt latency**

The interrupt event latency (the time from notification of the CPU that an event has occurred until the CPU begins execution of the first instruction in the OB that services the event) is approximately 175 μsec, provided that a program cycle OB is the only event service routine active at the time of the interrupt event.

**Functions (FCs)** and **function blocks (FBs)** contain the program code that corresponds to specific tasks or combinations of parameters. A function (FC) or a function block (FB) can be called from an OB or from another FC or FB, down to the following nesting depths:

● 16 from the program cycle or startup OB

● 4 from time delay interrupt, cyclic interrupt, time of day interrupt, hardware interrupt, time error interrupt, or diagnostic error interrupt OB

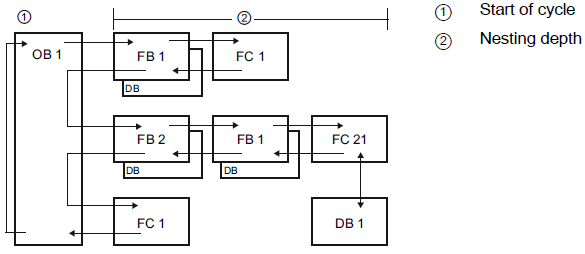
By creating generic code blocks that can be reused within the user program, you can simplify the design and implementation of the user program. Using generic code blocks has a number of benefits:

● You can create reusable blocks of code for standard tasks, such as for controlling a pump or a motor. You can also store these generic code blocks in a library that can be used by different applications or solutions.

● When you structure the user program into modular components that relate to functional tasks, the design of your program can be easier to understand and to manage. The modular components not only help to standardize the program design, but can also help to make updating or modifying the program code quicker and easier.

● Creating modular components simplifies the debugging of your program. By structuring the complete program as a set of modular program segments, you can test the functionality of each code block as it is developed.

● Creating modular components that relate to specific technological functions can help to simplify and reduce the time involved with commissioning the completed application.



**Function (FC)**

A function (FC) is a code block (is a subroutine) that is executed when called from another code block (OB, FB, or FC) and typically performs a specific operation on a set of input values.

For example, use FCs to perform standard and reusable operations (such as for mathematical calculations) or technological functions (such as for individual controls using bit logic operations).

An FC can also be called several times at different points in a program. This reuse simplifies the programming of frequently recurring tasks.

An FC does not have an associated instance data block (DB). The FC uses the local data stack for the temporary data used to calculate the operation.

The temporary data is not saved. To store data permanently, assign the output value to a global memory location, such as M memory or to a global DB.

**Function block (FB)**

A function block (FB) is a code block (is a subroutine) that is executed when called from another code

block (OB, FB, or FC) and uses an instance data block for its parameters and static data.

FBs have variable memory that is located in a data block (DB), or "instance" DB.

The instance DB provides a block of memory that is associated with that instance (or call) of the FB and stores data after the FB finishes.

You can associate different instance DBs with different calls of the FB. The instance DBs allow you to use one generic FB to control multiple devices. For example, one FB can control several pumps or valves, with different

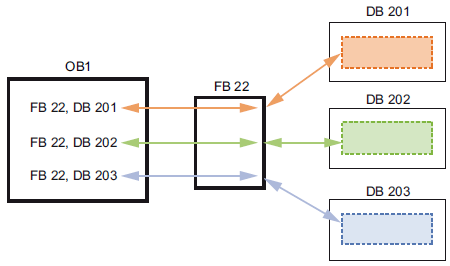
instance DBs containing the specific operational parameters for each pump or valve.

You structure your program by having one code block make a call to an FB and an instance DB. The CPU then executes the program code in that FB, and stores the block parameters and the static local data in the instance DB. When the execution of the FB finishes, the CPU returns to the code block that called the FB. The instance DB retains the values for that instance of the FB. These values are available to subsequent calls to the

function block either in the same scan cycle or other scan cycles.

The following figure shows an OB that calls one FB three times, using a different data block for each call. This structure allows one generic FB to control several similar devices, such as motors, by assigning a different instance data block for each call for the different devices. Each instance DB stores the data (such as speed, ramp-up time, and total operating time) for an individual device.

In this example, FB 22 controls three separate devices, with DB 201 storing the operational data for the first device, DB 202 storing the operational data for the second device, and DB 203 storing the operational data for the third device.



**Data block (DB)**

You create data blocks (DB) in your user program to store data for the code blocks.

All of the program blocks in the user program can access the data in a global DB, but an instance DB

stores data for a specific function block (FB).

The data stored in a DB is not deleted when the execution of the associated code block comes to an end.

There are two types of DBs:

● A global DB stores data for the code blocks in your program. Any OB, FB, or FC can access the data in a global DB.

● An instance DB stores the data for a specific FB. The structure of the data in an instance DB reflects the parameters (Input, Output, and InOut) and the static data for the FB. (The Temp memory for the FB is not stored in the instance DB.)

**Operating modes of the CPU**

The CPU has three modes of operation: STOP mode, STARTUP mode, and RUN mode. Status LEDs on the front of the CPU indicate the current mode of operation.

●In STOP mode, the CPU is not executing the program, the CPU handles any communication requests (as appropriate) and performs self-diagnostics. The CPU does not execute the user program, and the automatic updates of the process image do not occur. You can download your project only when the CPU is in STOP mode.

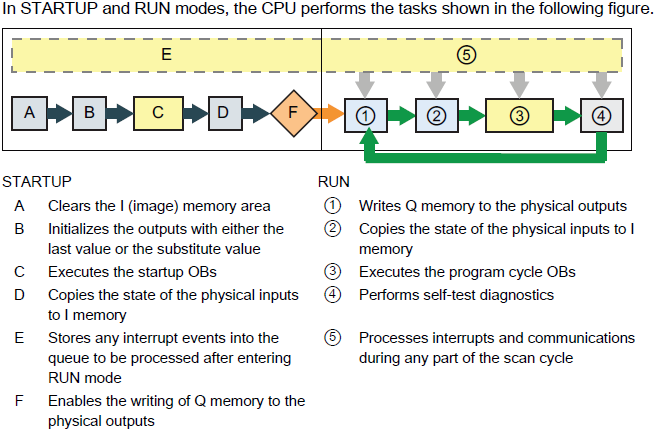
● In STARTUP mode, the startup OBs (if present) are executed once. Interrupt events are not processed during the startup mode. Whenever the operating mode changes from STOP to RUN, the CPU clears the process image inputs, initializes the process image outputs and processes the startup OBs. Any read accesses to the process-image inputs by instructions in the startup OBs read zero rather than the current physical input value. Therefore, to read the current state of a physical input during the startup mode, you must perform an immediate read. The startup OBs and any associated FCs and FBs are executed next. If more than one startup OB exists, each is executed in order according to the OB number, with the lowest OB number executing first.

● In RUN mode, the program cycle OBs are executed repeatedly. Interrupt events can occur and be processed at any point within the RUN mode. Some parts of a project can be downloaded in RUN mode (detayı var..)

*The CPU supports a warm restart for entering the RUN mode. Warm restart does not include a memory reset. All non-retentive system and user data are initialized at warm restart. Retentive user data is retained.*

*A memory reset clears all work memory, clears retentive and non-retentive memory areas, and copies load memory to work memory. A memory reset does not clear the diagnostics buffer or the permanently saved values of the IP address.*

**PLC Çevrimi :**



**Configuring the outputs on a RUN-to-STOP transition**

You can configure the behavior of the digital and analog outputs when the CPU is in STOP mode.

For any output of a CPU, SB or SM, you can set the outputs to either freeze the value or use a substitute value:

● Substituting a specified output value (default): You enter a substitute value for each output (channel) of that CPU, SB, or SM device.

The default substitute value for digital output channels is OFF, and the default substitute value for analog output channels is 0.

● Freezing the outputs to remain in last state: The outputs retain their current value at the time of the transition from RUN to STOP.

After power up, the outputs are set to the default substitute value.

You configure the behavior of the outputs in Device Configuration. Select the individual devices and use the "Properties" tab to configure the outputs for each device.

When the CPU changes from RUN to STOP, the CPU retains the process image and writes the appropriate values for both the digital and analog outputs, based upon the configuration.

**PLC Çevrim Süresi:** **(Min: 1ms, Max:6000ms Default:150ms)**

The cycle time is the time that the CPU operating system requires to execute the cyclic phase of the RUN mode. The CPU provides two methods of monitoring the cycle time:

● Maximum scan cycle time

● Fixed minimum scan cycle time

If the maximum scan cycle timer expires before the scan cycle has been completed, an error is generated and is handled one of two ways: If the user program does not include an OB 80, then the CPU generates an error and goes to STOP. (You can change the configuration of the CPU to ignore this time error and stay in RUN. The default configuration is for the CPU to go to STOP.) If the user program includes an OB 80, then the CPU executes OB 80

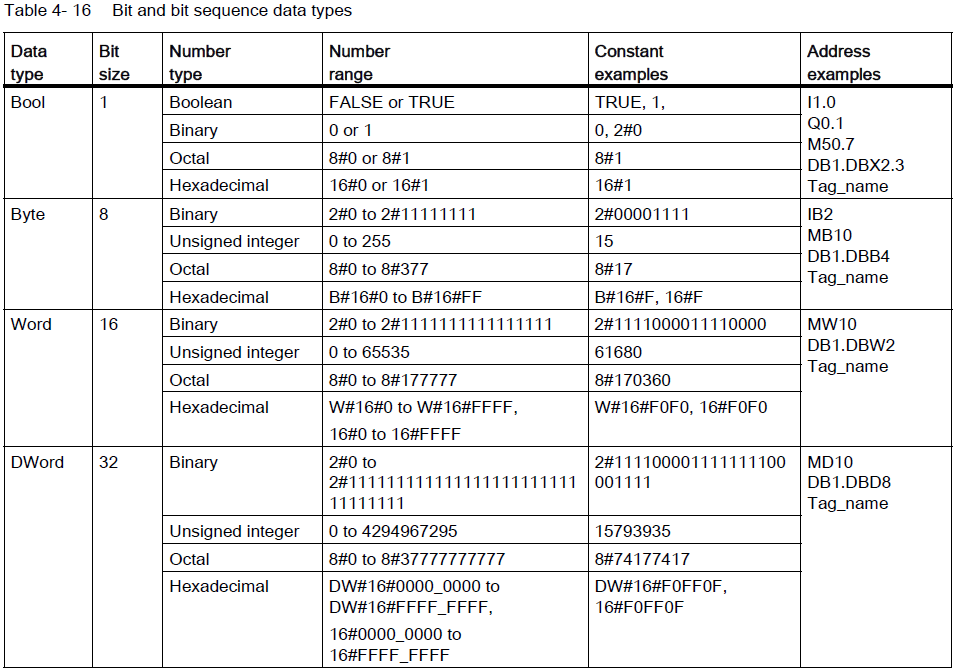
The RE\_TRIGR instruction (**RE\_TRIGR(): Re-trigger scan time watchdog) is used to extend the maximum time allowed before the scan cycle watchdog timer generates an error.**) allows you to reset the timer that measures the cycle time. However, this instruction only functions if executed in a program cycle OB; the RE\_TRIGR instruction is ignored if executed in OB 80. If the maximum scan cycle time is exceeded twice within the same program cycle with no RE\_TRIGR instruction execution between the two, then the CPU transitions to STOP immediately. The use of repeated executions of the RE\_TRIGR instruction can create an endless loop or a very long scan.

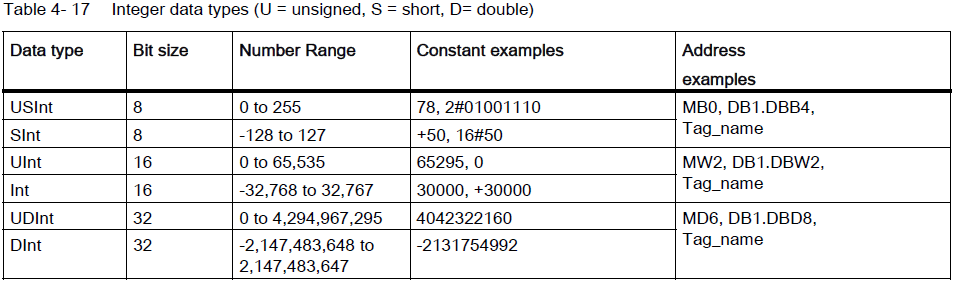
Typically, the scan cycle executes as fast as it can be executed and the next scan cycle begins as soon as the current one completes. Depending upon the user program and communication tasks, the time period for a scan cycle can vary from scan to scan. To eliminate this variation, the CPU supports an optional fixed minimum scan cycle time (also called fixed scan cycle). When this optional feature is enabled and a fixed minimum scan cycle time is provided in ms, the CPU will maintain the minimum cycle time within ±1 ms for the completion of each CPU scan.

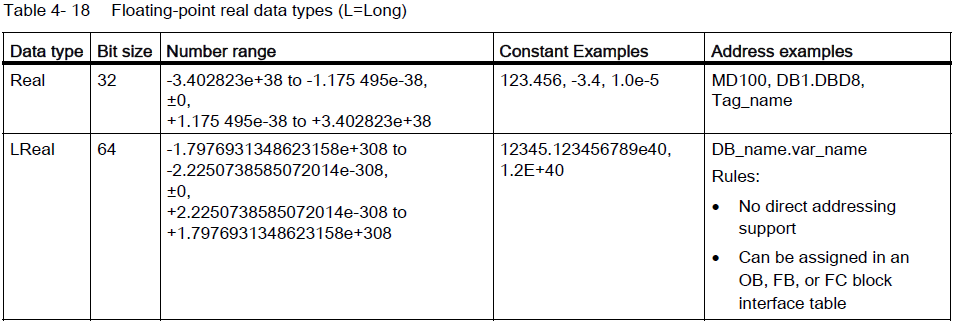
In the event that the CPU completes the normal scan cycle in less time than the specified minimum cycle time, the CPU spends the additional time of the scan cycle performing runtime diagnostics and/or processing communication requests. In this way the CPU always takes a fixed amount of time to complete a scan cycle.

In the event that the CPU does not complete the scan cycle in the specified minimum cycle time, the CPU completes the scan normally (including communication processing) and does not create any system reaction as a result of exceeding the minimum scan time. The following table defines the ranges and defaults for the cycle time monitoring functions.

**Veri Türleri (Hatırlatma)**







Real (or floating-point) numbers are represented as 32-bit single-precision numbers (Real), or 64-bit double-precision numbers (LReal) as described in the ANSI/IEEE 754-1985 standard. Single-precision floating-point numbers are accurate up to 6 significant digits and double-precision floating point numbers are accurate up to 15 significant digits. You can specify a maximum of 6 significant digits (Real) or 15 (LReal) when entering a floating-point constant to maintain precision.

Calculations that involve a long series of values including very large and very small numbers can produce inaccurate results. This can occur if the numbers differ by 10 to the power of x, where x > 6 (Real), or 15 (LReal). For example (Real): 100 000 000 + 1 = 100 000 000.

Hafıza Adres Alanları..

