TEKNICA WEB

PROGRAMACIÓN INDUSTRIAL SIEMENS

CURSO AVANZADO - TEORÍA

MÓDULO 9:

PROGRAMACIÓN

INDICE DE CONTENIDOS

1 MÓDULO 9: PROGRAMACIÓN 2

1.1 PROGRAMACIÓN. 2

1.1.1 INTRODUCCIÓN. 2

1.1.2 DESCRIPCIÓN FUNCIONAMIENTO CPU. 2

1.1.2.1 DESCRIPCIÓN FUNCIONAMIENTO CPU. ORGANIZACIÓN GENERAL 2

1.1.2.2 DESCRIPCIÓN FUNCIONAMIENTO CPU. ESTADOS OPERATIVOS DE CPU 3

1.1.2.3 DESCRIPCIÓN FUNCIONAMIENTO CPU. BLOQUES DE ORGANIZACIÓN (OB) 4

1.1.2.4 DESCRIPCIÓN FUNCIONAMIENTO CPU. GESTIÓN DE LA MEMORIA 6

1.1.2.5 DESCRIPCIÓN FUNCIONAMIENTO CPU. MEMORIA REMANENTE 7

1.1.2.6 DESCRIPCIÓN FUNCIONAMIENTO CPU. ALMACENAMIENTO DE DATOS. 8

1.1.2.7 DESCRIPCIÓN FUNCIONAMIENTO CPU. TIPOS DE DATOS. 9

1.1.3 PRINCIPIOS BÁSICOS DE PROGRAMACIÓN. 10

1.1.4 PRINCIPIOS BÁSICOS DE PROGRAMACIÓN. PASOS INICIALES. 10

1.1.5 PRINCIPIOS BÁSICOS DE PROGRAMACIÓN. ESTRUCTURACIÓN DE PROGRAMA. 12

1.1.6 PRINCIPIOS BÁSICOS DE PROGRAMACIÓN. SELECCIÓN DE LENGUAJE DE PROGRAMACIÓN. 14

1.1.7 CONCLUSIONES. 17

1.1.8 EXAMEN. 18

1.1.9 AUTOCOMPROBACIÓN. 20

1.1.10 PROPUESTAS. 20

1.1.10.1 PROPUESTA 1. 20

1.1.11 LISTADO DE IMÁGENES. 20

1 MÓDULO 9: PROGRAMACIÓN

1.1 PROGRAMACIÓN.

1.1.1 INTRODUCCIÓN.

En este tema vamos a comenzar con la programación del autómata. Se trata de un tema muy teórico. A través del programa de usuario, podremos realizar diversas tareas de automatización, siempre referidas a las entradas y salidas que tengamos en nuestro equipo. Las posibilidades en principio son muy extensas, ya que estos equipos se pueden emplear para casi cualquier aplicación que se pueda imaginar: automatización de procesos, maquinaria, domótica, etc.

Iniciaremos el tema estudiando las características básicas de operación del sistema, puesto que la forma de trabajar de la CPU influye de manera muy importante en el desarrollo del programa, y por tanto en la forma de desarrollar nuestro programa para su correcto desempeño.

También veremos distintas recomendaciones relacionadas con la estructuración de los programas, lo cual nos servirá de ayuda no sólo a la hora de crear nuestro programa, sino también para su mantenimiento y modificación posterior, siguiendo unas pautas lógicas que nos faciliten la lectura y seguimiento del mismo, con orden y claridad en la definición de variables, entradas/salidas, procesos y rutinas, etc.

Existen distintos “lenguajes de programación”, que son las distintas maneras que tenemos de escribir las operaciones que queremos realizar en el equipo físico. Veremos las distintas posibilidades para que el alumno pueda elegir la que considere mejor en cada caso o proyecto.

1.1.2 DESCRIPCIÓN FUNCIONAMIENTO CPU.

El primer punto a tener en cuenta a la hora de comenzar la programación será entender la manera en la que el sistema PLC funciona. En gran medida, nuestro programa deberá adaptarse a la forma de funcionar del sistema, de aquí la importancia de conocer en detalle su organización y estructura.

1.1.2.1 DESCRIPCIÓN FUNCIONAMIENTO CPU. ORGANIZACIÓN GENERAL

La CPU soporta los siguientes tipos de bloques lógicos que permiten estructurar eficientemente el programa de usuario:

- Los bloques de organizaciónón (OBs) definen la estructura del programa. Algunos OBs tienen reacciones y eventos de arranque predefinidos. No obstante, también es posible crear OBs con eventos de arranque personalizados.
- Las funciones (FCs) y los bloques de función (FBs) contienen el código de programa correspondiente a tareas específicas o combinaciones de parámetros. Cada FC o FB provee parámetros de entrada y salida para compartir datos con el bloque invocante. Un FB utiliza también un bloque de datos asociado (denominado DB instancia) para conservar el estado de valores durante la ejecución que pueden utilizar otros bloques del programa.
- Los bloques de datos (DBs) almacenan datos que pueden ser utilizados por los bloques del programa.

La ejecución del programa de usuario comienza con uno o varios bloques de organización (OBs) de arranque que se ejecutan una vez al cambiar a estado operativo RUN, seguidos de uno o varios OBs de ciclo que se ejecutan cíclicamente. También es posible asociar un OB a un evento de alarma que puede ser un evento estándar o de error y que se ejecuta cada vez que ocurre el evento en cuestión.

Una función (FC) o un bloque de función (FB) es un bloque de código del programa que puede llamarse desde un OB, o bien desde otra FC u otro FB.

1.1.2.2 DESCRIPCIÓN FUNCIONAMIENTO CPU. ESTADOS OPERATIVOS DE CPU

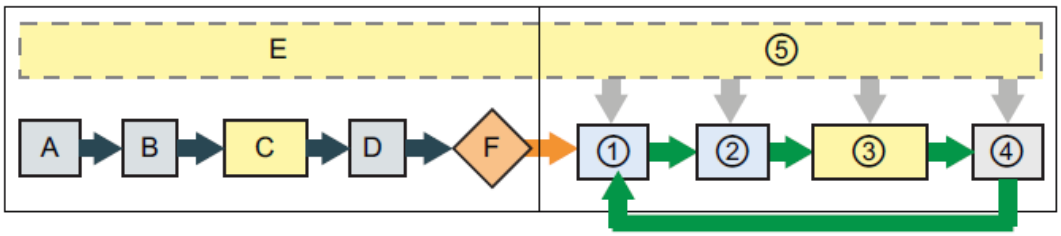
La CPU tiene tres estados operativos, a saber: STOP, ARRANQUE y RUN. Los LEDs de estado en el frente de la CPU indican el estado operativo actual.

- En estado operativo STOP, la CPU no ejecuta el programa. Entonces es posible cargar un proyecto en la CPU.
- En estado operativo ARRANQUE, los OBs de arranque (si existen) se ejecutan una vez. Los eventos de alarma no se procesan durante la fase de arranque del estado operativo RUN.
- El ciclo se ejecuta repetidamente en estado operativo RUN. Los eventos de alarma pueden ocurrir y procesarse en cualquier fase del ciclo del programa. En estado operativo RUN no es posible cargar proyectos en la CPU.

La CPU soporta el arranque en caliente para pasar al estado operativo RUN. El arranque en caliente no incluye la inicialización de la memoria. Los datos de sistema no remanentes y los datos de usuario se inicializan en un arranque en caliente. Se conservan los datos de usuario remanentes.

El borrado total borra toda la memoria de trabajo, así como las áreas de memoria remanentes y no remanentes. Además, copia la memoria de carga en la memoria de trabajo. El borrado total no borra el búfer de diagnóstico ni tampoco los valores almacenados permanentemente de la dirección IP.

En estado operativo RUN, la CPU ejecuta las tareas que muestra la figura siguiente.



ARRANQUE		RUN	
A	Borra el área de memoria I	①	Escribe la memoria Q en las salidas físicas
B	Inicializa las salidas con el último valor o el valor sustitutivo	②	Copia el estado de las entradas físicas en la memoria I
C	Ejecuta los OBs de arranque	③	Ejecuta los OBs de ciclo
D	Copia el estado de las entradas físicas en la memoria I	④	Realiza autodiagnóstico
E	Almacena los eventos de alarma en la cola de espera que deben procesarse en estado operativo RUN	⑤	Procesa alarmas y comunicaciones en cualquier parte del ciclo
F	Habilita la escritura de la memoria Q en las salidas físicas		

Ilustración 1.1.2-1 Tareas a ejecutar por CPU durante Arranque y en estado RUN

1.1.2.3 DESCRIPCIÓN FUNCIONAMIENTO CPU. BLOQUES DE ORGANIZACIÓN (OB)

Los OBs controlan la ejecución del programa de usuario. Todo OB debe tener un número de OB unívoco. Algunos números menores que 200 están reservados para OBs predeterminados. La numeración de los demás OBs debe comenzar a partir de 200.

Determinados eventos de la CPU disparan la ejecución de un bloque de organización. Un OB no puede llamar a otro. Tampoco es posible llamar un OB desde una FC o un FB. Sólo un evento de arranque, p. ej. una alarma de diagnóstico o un intervalo, puede iniciar la ejecución de un OB. La CPU procesa los OBs según su clase de prioridad. Los OBs de mayor prioridad se ejecutan antes que los de menor prioridad. La clase de prioridad más baja es 1 (para el ciclo del programa principal) y la más alta es 27 (para las alarmas de error de tiempo).

Los OBs controlan los siguientes procesos:

- Los OBs de ciclo se ejecutan cíclicamente cuando la CPU se encuentra en estado operativo RUN. El bloque principal del programa es un OB de ciclo. Éste contiene las instrucciones que controlan el programa y permite llamar otros bloques de usuario. Es posible utilizar varios OBs de ciclo. Éstos se ejecutan en orden numérico. El OB 1 es el bloque predeterminado. Los demás OBs de ciclo deben identificarse como OB 200 o superior.
- Los OBs de arranque se ejecutan una vez cuando el estado operativo de la CPU cambia de STOP a RUN, al arrancar a estado operativo RUN y en una transición ordenada de STOP a RUN. Una vez finalizado, se comienza a ejecutar el OB de ciclo. Es posible utilizar varios OBs de arranque. El OB 100 es el bloque predeterminado. El número de los demás OBs debe ser 200 o superior.

- Los OBs de alarma de retardo se ejecutan al cabo de un intervalo posterior a un evento configurado en la instrucción de alarma de arranque (SRT_DINT). El tiempo de retardo se especifica en el parámetro de entrada de la instrucción avanzada SRT_DINT. Los OBs de alarma de retardo interrumpen la ejecución cíclica del programa una vez transcurrido un tiempo de retardo especificado. Es posible configurar como máximo 4 eventos de retardo en cualquier momento. Por cada evento de retardo configurado se permite un OB. El número del OB de alarma de retardo debe ser 200 o superior.
- Los OBs de alarma cíclica se ejecutan en intervalos periódicos. Los OBs de alarma cíclica interrumpen la ejecución cíclica del programa en intervalos definidos, por ejemplo cada 2 segundos. Es posible configurar como máximo 4 eventos de alarma cíclica. Por cada evento de alarma cíclica configurado se permite un OB. El número del OB debe ser 200 o superior.
- Los OBs de alarma de proceso se ejecutan cuando ocurre el evento de hardware correspondiente, incluyendo flancos ascendentes y descendentes en las entradas digitales integradas y eventos de contadores rápidos (HSC). Los OBs de alarma de proceso interrumpen la ejecución cíclica del programa como reacción a una señal de un evento de hardware. Los eventos se definen en las propiedades de la configuración hardware. Por cada evento de hardware configurado se permite un OB. El número del OB debe ser 200 o superior.
- Los OBs de error de tiempo se ejecutan cuando se detecta un error de tiempo. Los OBs de error de tiempo interrumpen la ejecución cíclica del programa cuando se rebasa el tiempo de ciclo máximo. El tiempo de ciclo máximo se define en las propiedades del PLC. El OB 80 es el único número de OB soportado para el evento de error de tiempo. Es posible configurar la acción que debe realizarse si no existe el OB 80: ignorar el error o cambiar a STOP.
- Los OBs de alarma de diagnóstico se ejecutan cuando se detecta y notifica un error de diagnóstico. Los OBs de alarma de diagnóstico interrumpen la ejecución cíclica del programa cuando el módulo apto para diagnóstico detecta un error (si se ha habilitado la alarma de diagnóstico para ese módulo). El OB 82 es el único número de OB soportado para el evento de error de diagnóstico. Si el programa no contiene ningún OB de diagnóstico, la CPU se puede configurar para que ignore el error o cambie a STOP.

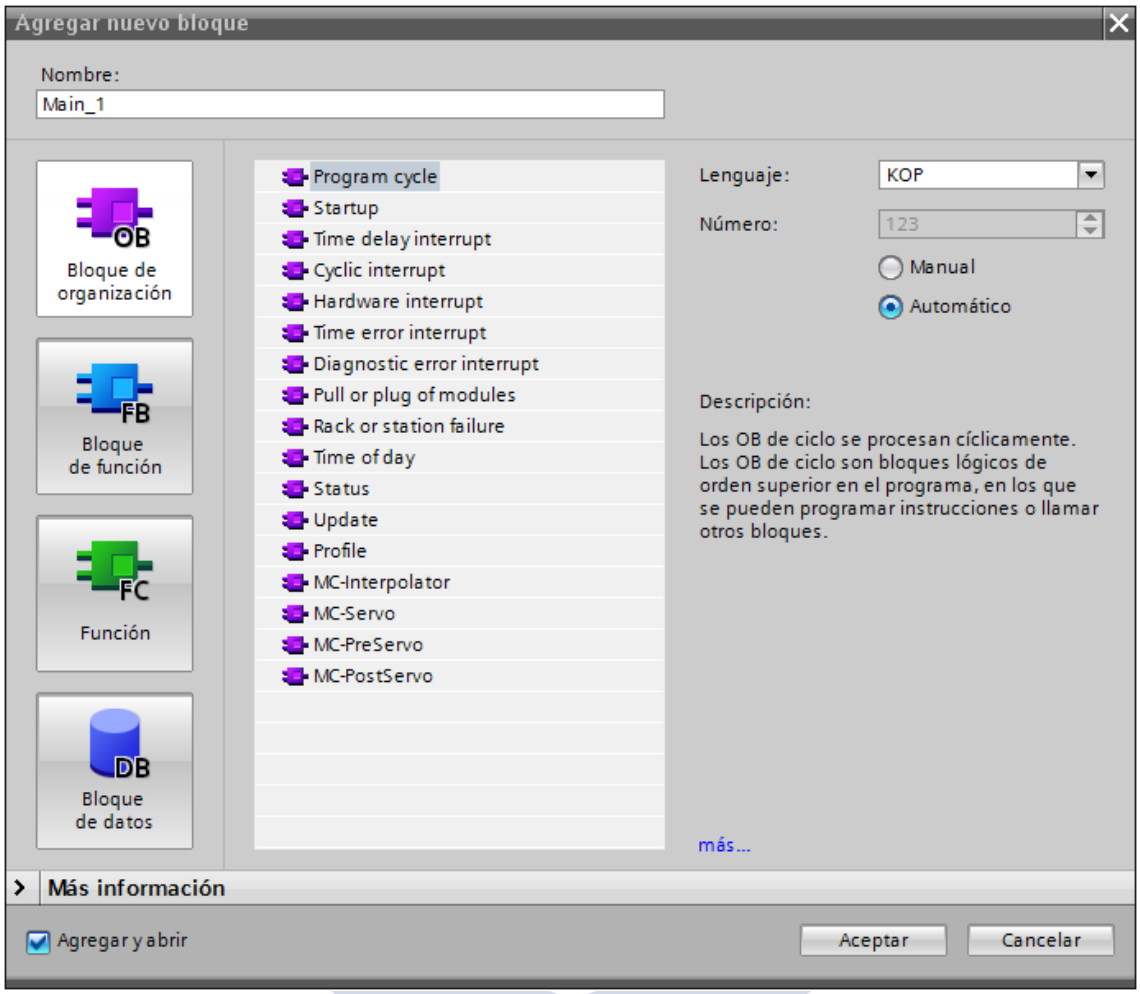


Ilustración 1.1.2-2 Listado con distintos tipos de OB para crear en un proyecto

1.1.2.4 DESCRIPCIÓN FUNCIONAMIENTO CPU. GESTIÓN DE LA MEMORIA

La CPU provee las áreas de memoria siguientes para almacenar el programa de usuario, los datos y la configuración:

Memoria de Carga:

La memoria de carga permite almacenar de forma no volátil el programa de usuario, los datos y la configuración. Cuando un proyecto se carga en la CPU, se almacena primero en el área de memoria de carga. Esta área se encuentra bien sea en una Memory Card (si está disponible) o en la CPU. Esta área de memoria no volátil se conserva incluso tras una pérdida de potencia. La Memory Card ofrece mayor espacio de almacenamiento que el integrado en la CPU.

Memoria de Trabajo:

La memoria de trabajo ofrece almacenamiento volátil para algunos elementos del proyecto mientras se ejecuta el programa de usuario. La CPU copia algunos elementos del proyecto desde la memoria de carga en la memoria de trabajo. Esta área volátil se pierde si se desconecta la alimentación. La CPU la restablece al retornar la alimentación.

Memoria Remanente:

La memoria remanente permite almacenar de forma no volátil un número limitado de valores de la memoria de trabajo. El área de memoria remanente se utiliza para almacenar los valores de algunas posiciones de memoria durante una pérdida de potencia. Si ocurre un corte de alimentación, la CPU dispone de suficiente tiempo de retención para respaldar los valores de un número limitado de posiciones de memoria definidas. Estos valores remanentes se restablecen al retornar la alimentación. Para ver el uso de memoria del proyecto actual, haga clic con el botón derecho del ratón en la CPU (o uno de sus bloques) y elija el comando "Carga de la memoria" del menú contextual. Para ver el uso de memoria de la CPU actual, haga doble clic en "Online y diagnóstico", expanda "Diagnóstico" y seleccione "Memoria".

1.1.2.5 DESCRIPCIÓN FUNCIONAMIENTO CPU. MEMORIA REMANENTE

Para impedir la pérdida de datos tras un corte de alimentación, es posible definir que ciertos datos sean remanentes. Los siguientes datos pueden configurarse para que sean remanentes:

Área de marcas (M):

El ancho preciso de la memoria para el área de marcas puede definirse en la tabla de variables PLC o el plano de ocupación. El área de marcas remanente comienza siempre en MB0, abarcando consecutivamente un determinado número de bytes. Para definir este valor, haga clic en el botón "Remanencia" de la barra de herramientas de la tabla de variables PLC o del plano de ocupación. Introduzca el número de bytes M que deben ser remanentes a partir de MB0.

Variables de un bloque de función (FB):

Si un FB se ha creado estando activada la casilla "Sólo con direccionamiento simbólico", la interfaz del bloque de este FB incluirá una columna "Remanencia". En esta columna es posible seleccionar "Remanente" o "No remanente" individualmente para cada una de las variables. Un DB instancia que haya sido creado al insertar este FB en el editor de programas muestra asimismo la columna "Remanencia", aunque no permite editarla. El estado remanente no se puede modificar desde la interfaz del bloque del DB de instancia para un FB que haya sido configurado "Sólo con direccionamiento simbólico".

Si un FB ha sido creado estando desactivada la casilla "Sólo con direccionamiento simbólico", la interfaz del bloque de este FB no incluirá la columna "Remanencia". Un DB instancia que haya sido creado al insertar este FB en el editor de programas muestra y permite editar la columna "Remanencia". En este caso, si se activa la opción "Remanente" para alguna de las variables, se seleccionarán todas las variables. Por analogía, si se desactiva la opción "Remanente" para alguna de las variables, se deseleccionarán todas las variables. Si un FB se ha configurado sin el atributo "Sólo con direccionamiento simbólico", el estado remanente se puede cambiar desde la interfaz del bloque del DB instancia, pero todas las variables se ajustan conjuntamente al mismo estado remanente.

Tras haber creado el FB no es posible modificar la opción "Sólo con direccionamiento simbólico". Esta opción sólo se puede seleccionar cuando se crea el FB. Para determinar si un FB existente se ha configurado "Sólo con direccionamiento simbólico", haga clic con el botón derecho del ratón en el FB en el árbol del proyecto, elija "Propiedades" y seleccione luego "Atributos".

Variables de un bloque de datos global:

El comportamiento de un DB global respecto a la asignación del estado remanente es similar al de un FB. En función del ajuste de direccionamiento simbólico, es posible definir el estado remanente de algunas o todas las variables de un bloque de datos global.

- Si el atributo "Sólo con direccionamiento simbólico" está activado para el DB, el estado remanente se podrá ajustar para cada una de las variables.
- Si el atributo "Sólo con direccionamiento simbólico" está desactivado para el DB, el ajuste de remanencia se aplicará a todas las variables del DB. Por tanto, todas o ninguna de las variables serán remanentes.

Un total de 2048 bytes de datos pueden ser remanentes. Para ver cuánto espacio está disponible, haga clic en el botón "Remanencia" de la barra de herramientas de la tabla de variables PLC o del plano de ocupación. Aunque aquí se especifica el rango remanente para la memoria M, la segunda fila indica la memoria restante disponible en total para M y DB conjuntamente.

1.1.2.6 DESCRIPCIÓN FUNCIONAMIENTO CPU. ALMACENAMIENTO DE DATOS.

La CPU ofrece varias opciones para almacenar datos durante la ejecución del programa de usuario:

- Memoria global: La CPU ofrece distintas áreas de memoria, incluyendo entradas (I), salidas (Q) y marcas (M). Todos los bloques lógicos pueden acceder sin restricción alguna a esta memoria.
- Bloque de datos (DB): Es posible incluir DBs en el programa de usuario para almacenar los datos de los bloques lógicos. Los datos almacenados se conservan cuando finaliza la ejecución del bloque lógico asociado. Un DB "global" almacena datos que pueden ser utilizados por todos los bloques lógicos, mientras que un DB instancia almacena datos para un bloque de función (FB) especifico y está estructurado según los parámetros del FB.
- Memoria temporal: Cada vez que se llama un bloque lógico, el sistema operativo de la CPU asigna la memoria temporal o local (L) que debe utilizarse durante la ejecución del bloque. Cuando finaliza la ejecución del bloque lógico, la CPU reasigna la memoria local para la ejecución de otros bloques lógicos

Toda posición de memoria diferente tiene una dirección unívoca. El programa de usuario utiliza estas direcciones para acceder a la información de la posición de memoria. La figura muestra cómo acceder a un bit (lo que también se conoce como direccionamiento "byte.bit"). En este ejemplo, el área de memoria y la dirección del byte (I = entrada y 3 = byte 3) van seguidas de un punto (".") que separa la dirección del bit (bit 4).

A los datos de la mayoría de las áreas de memoria (I, Q, M, DB y L) se puede acceder como bytes, palabras o palabras dobles utilizando el formato "dirección de byte". Para acceder a un byte, una palabra o una palabra doble de datos en la memoria, la dirección debe especificarse de forma similar a la dirección de un bit. Esto incluye un identificador de área, el tamaño de los datos y la dirección de byte inicial del valor de byte, palabra o palabra doble. Los designadores de tamaño son B (byte), W (palabra) y D (palabra doble), p. ej. IB0, MW20 o QD8. Las direcciones tales como I0.3 y Q1.7 acceden a la memoria imagen de proceso. Para acceder a la entrada o salida física es preciso añadir ":P" a la dirección (por ejemplo I0.3:P, Q1.7:P o "Stop:P").

1.1.2.7 DESCRIPCIÓN FUNCIONAMIENTO CPU. TIPOS DE DATOS.

Los tipos de datos se utilizan para determinar el tamaño de un elemento de datos y cómo deben interpretarse los datos. Todo parámetro de instrucción soporta como mínimo un tipo de datos. Algunos parámetros soportan varios tipos de datos. Sitúe el cursor sobre el campo de parámetro de una instrucción para ver qué tipos de datos soporta el parámetro en cuestión.

Un parámetro formal es el identificador en una instrucción que indica la ubicación de los datos que deben utilizarse (ejemplo: la entrada IN1 de una instrucción ADD). Un parámetro actual es la posición de memoria o constante que contiene los datos que debe utilizar la instrucción (ejemplo: %MD400 "Número_de_elementos"). El tipo de datos del parámetro actual definido por el usuario debe concordar con uno de los tipos de datos que soporta el parámetro formal especificado por la instrucción.

Al definir un parámetro actual es preciso indicar una variable (símbolo) o una dirección absoluta. Las variables asocian un nombre simbólico (nombre de variable) con un tipo de datos, área de memoria, offset y comentario. Se pueden crear bien sea en el editor de variables PLC, o bien en la interfaz del bloque (OB, FC, FB o DB). Si se introduce una dirección absoluta que no tenga una variable asociada, es preciso utilizar un tamaño apropiado que coincida con el tipo de datos soportado. Al realizar la entrada se creará una variable predeterminada.

También es posible introducir un valor de constante para numerosos parámetros de entrada.

La tabla siguiente muestra los tipos de datos simples soportados, incluyendo ejemplos de entrada de constantes. Todos los tipos de datos, excepto String, están disponibles en el editor de variables PLC y en la interfaz del bloque. String sólo está disponible en la interfaz del bloque. La tabla siguiente muestra los tipos de datos simples.

Tipo de datos	Tamaño (bits)	Rango	Ejemplos de entrada de constantes
Bool	1	0 a 1	TRUE, FALSE, 0, 1
Byte	8	16#00 a 16#FF	16#12, 16#AB
Word	16	16#0000 a 16#FFFF	16#ABCD, 16#0001
DWord	32	16#00000000 a 16#FFFFFFFF	16#02468ACE
Char	8	16#00 a 16#FF	'A', 't', '@'
Sint	8	128 a 127	123, -123
Int	16	32.768 a 32.767	123, -123
Dint	32	-2.147.483.648 a 2.147.483.647	123, -123
USInt	8	0 a 255	123
UInt	16	0 a 65.535	123
UDInt	32	0 a 4.294.967.295	123
Real	32	+/-1,18 x 10^{-38} a +/-3,40 x 10^{38}	123,456, -3,4, -1,2E+12, 3,4E-3
LReal	64	+/-2,23 x 10^{-308} a +/-1,79 x 10^{308}	12345.123456789 -1,2E+40
Time	32	T#-24d_20h_31m_23s_648ms a T#24d_20h_31m_23s_647ms Almacenado como: -2,147,483,648 ms a +2,147,483,647 ms	T#5m_30s 5#-2d T#1d_2h_15m_30x_45ms
String	Variable	0 a 254 caracteres en tamaño de byte	'ABC'

Ilustración 1.1.2-3 Tipos de datos que puede manejar una CPU

1.1.3 PRINCIPIOS BÁSICOS DE PROGRAMACIÓN.

En este apartado vamos a estudiar los principios básicos de programación. Partiremos de lo más básico, que es la definición correcta del proceso o máquina que vamos a desarrollar. Un diseño inicial adecuado y acertado nos va a suponer una reducción importante del tiempo de desarrollo del proyecto, especialmente en lo que a diseño de estructura general se refiere.

1.1.4 PRINCIPIOS BÁSICOS DE PROGRAMACIÓN. PASOS INICIALES.

A continuación se describen, de manera simplificada, los pasos fundamentales a la hora de iniciar un proyecto de automatización.

Dividir el proceso o máquina

Divida el proceso o máquina en secciones independientes. Estas secciones determinan los límites entre los controladores e influyen en las especificaciones funcionales y la asignación de recursos.

Crear las especificaciones funcionales

Describa el funcionamiento de cada una de las secciones del proceso o máquina, tales como las entradas y salidas, la descripción funcional de la operación, los estados que deben adoptarse antes de que puedan entrar en acción los actuadores (como p. ej. electroválvulas, motores o accionamientos), la descripción de la interfaz de operador y cualquier interfaz con otras secciones del proceso o máquina.

Diseñar los circuitos de seguridad

Determine los equipos que puedan requerir cableado fijo por motivos de seguridad. Recuerde que los dispositivos de control pueden fallar y provocar condiciones no seguras, causando a su vez un arranque inesperado o cambios de funcionamiento de la maquinaria. El funcionamiento inesperado o incorrecto de la maquinaria puede causar lesiones corporales o daños materiales considerables. Por tanto, prevea dispositivos de protección electromecánicos (que funcionen independientemente del PLC) para evitar las condiciones no seguras. Las siguientes tareas deben incluirse en el diseño de circuitos de seguridad:

- Definir el funcionamiento erróneo o inesperado de los actuadores que pudiera resultar peligroso.
- Definir las condiciones que garanticen un funcionamiento seguro y determinar cómo detectar estas condiciones, independientemente del PLC.
- Definir cómo el PLC y los módulos de ampliación deben influir en el proceso al conectarse y desconectarse la alimentación eléctrica, así como al detectarse errores. Utilice esta información sólo para proyectar el funcionamiento normal y el funcionamiento anormal esperado. Por motivos de seguridad, no conviene fiarse del supuesto más favorable.
- Prever dispositivos de parada de emergencia manual o dispositivos de protección electromecánicos que impidan el funcionamiento peligroso, independientemente del PLC.
- Proporcionar información de estado apropiada desde los circuitos independientes al PLC para que el programa y las interfaces de operador dispongan de la información necesaria.
- Definir otros requisitos adicionales de seguridad para el funcionamiento seguro del proceso.

Determinar las estaciones de operador

Según los requisitos de las especificaciones funcionales, cree los siguientes dibujos de las estaciones de operador:

- Dibujo general de la ubicación de todas las estaciones de operador con respecto al proceso o máquina
- Dibujo de la disposición mecánica de los dispositivos de la estación de operador, p. ej. display, interruptores y lámparas
- Esquemas eléctricos con las E/S asociadas del PLC y los módulos de señales

Crear los dibujos de configuración

Según los requisitos de las especificaciones funcionales, cree dibujos de configuración de los equipos de control:

- Dibujo general de la ubicación de todos los PLCs con respecto al proceso o máquina
- Dibujo de la disposición mecánica de todos los PLCs y módulos de E/S, incluyendo los armarios y otros equipos.

- Esquemas eléctricos de todos los PLCs y módulos de E/S, incluyendo los números de referencia de los dispositivos, las direcciones de comunicación y las direcciones de E/S.

Crear una lista de nombres simbólicos

Cree una lista de los nombres simbólicos correspondientes a las direcciones absolutas. Incluya no sólo las E/S físicas, sino también los demás elementos (p. ej. los nombres de variables) que se utilizarán en el programa.

1.1.5 PRINCIPIOS BÁSICOS DE PROGRAMACIÓN. ESTRUCTURACIÓN DE PROGRAMA.

Según los requisitos de la aplicación, es posible seleccionar una estructura lineal o modular para crear el programa de usuario:

- Un programa lineal ejecuta todas las instrucciones de la tarea de automatización de forma secuencial, es decir, una tras otra. Generalmente, el programa lineal deposita todas las instrucciones del programa en el OB encargado de la ejecución cíclica del programa (OB 1).
- Un programa modular llama bloques de función específicos que ejecutan determinadas tareas. Para crear una estructura modular, la tarea de automatización compleja se divide en tareas subordinadas más pequeñas, correspondientes a las funciones tecnológicas del proceso. Cada bloque lógico provee el segmento del programa para cada tarea subordinada. El programa se estructura llamando uno de los bloques lógicos desde otro bloque.

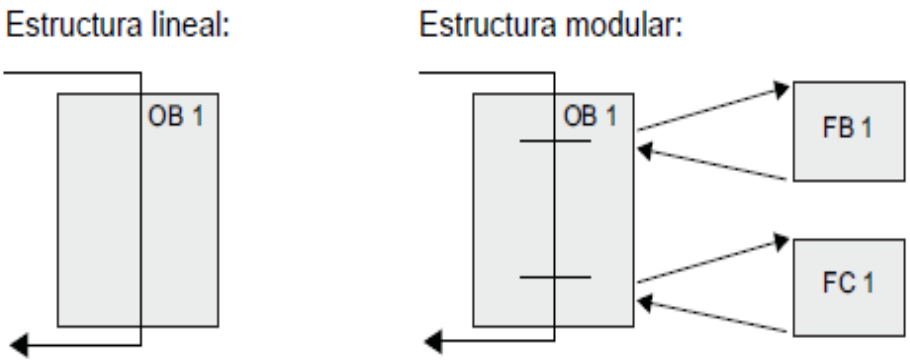


Ilustración 1.1.5-1 Ejemplos de estructura de programa lineal y estructura modular (más común)

Creando bloques lógicos genéricos que pueden reutilizarse en el programa de usuario, es posible simplificar el diseño y la implementación del programa de usuario. La utilización de bloques lógicos genéricos ofrece numerosas ventajas:

- Es posible crear bloques lógicos reutilizables para tareas estándar, tales como el control de una bomba o motor. También es posible almacenar estos bloques

lógicos genéricos en una librería, de manera que puedan ser utilizados por diferentes aplicaciones o soluciones.

- El programa de usuario puede dividirse en componentes modulares para las tareas funcionales, facilitando así su comprensión y gestión. Los componentes modulares ayudan no sólo a estandarizar el diseño del programa, sino que también pueden facilitar y agilizar la actualización o modificación del código del programa.
- La creación de componentes modular simplifica la depuración del programa. Dividiendo el programa completo en segmentos de programa modulares, es posible comprobar las funciones de cada bloque lógico a medida que se va desarrollando.
- La creación de componentes modulares para las distintas funciones tecnológicas permite simplificar y reducir el tiempo de puesta en marcha de la aplicación.

A Bloque que llama
B Bloque llamado (o que interrumpe)
① Ejecución del programa
② Operación que llama otro bloque
③ Ejecución del programa
④ Fin del bloque (regresa al bloque que llama)

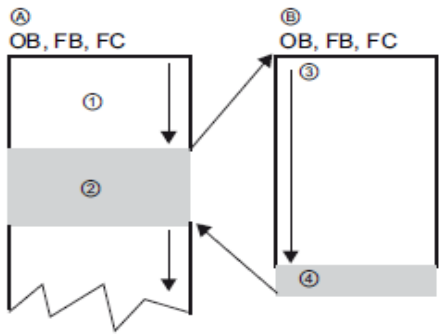


Ilustración 1.1.5-2 Estructura modular de programación. El bloque A se ejecuta y llama a la ejecución del bloque B.

Cuando un bloque lógico llama otro bloque lógico, la CPU ejecuta el código del programa en el bloque llamado. Una vez finalizada la ejecución del bloque llamado, la CPU reanuda la ejecución del bloque que ha efectuado la llamada.

El procesamiento continúa con la ejecución de la instrucción siguiente a la llamada de bloque. Las llamadas de bloque pueden anidarse para crear una estructura más modular.

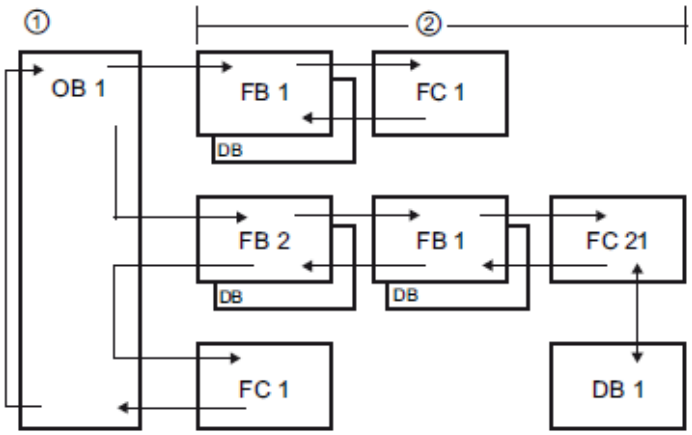


Ilustración 1.1.5-3 Estructura modular de programación, anidamiento (llamada de funciones dentro de otras funciones).

Una gran ventaja de los Bloques de función (FB) es que pueden ser llamados (instancia) desde distintos puntos del programa con distintos bloques de instancia (DB). Esto ocurre

por ejemplo, cuando en un proyecto tenemos por ejemplo un FB para controlar una bomba. Si en ese proyecto hay varias bombas con un funcionamiento similar, se llamará a un único FB (por ejemplo FB_Bomba) con una instancia (bloque de datos) propio para cada bomba concreta (con datos como por ejemplo, la potencia, altura que da, etc.).

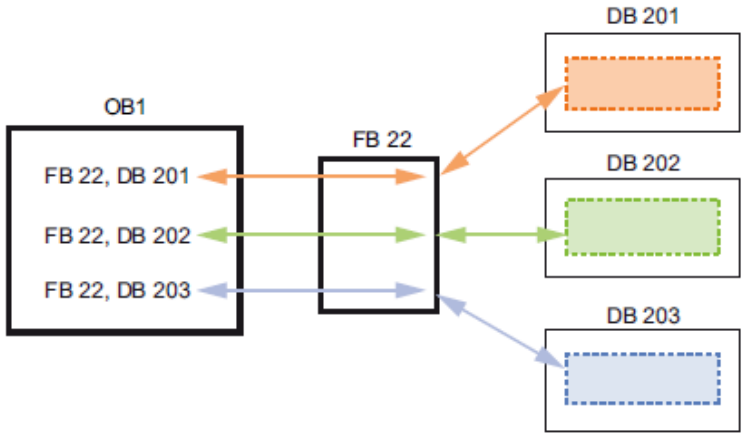


Ilustración 1.1.5-4 Ejemplo de FB llamado en distintos puntos de un programa con distintos DB de instancia

1.1.6 PRINCIPIOS BÁSICOS DE PROGRAMACIÓN. SELECCIÓN DE LENGUAJE DE PROGRAMACIÓN.

Tal como se ha comentado anteriormente, existen distintos tipos de lenguaje que se pueden emplear para programar nuestro código que se ejecutará en el PLC. Cada uno de ellos tiene unas características concretas, y conviene estudiarlas para poder saber en qué momento puede resultar ventajoso usar una u otra.

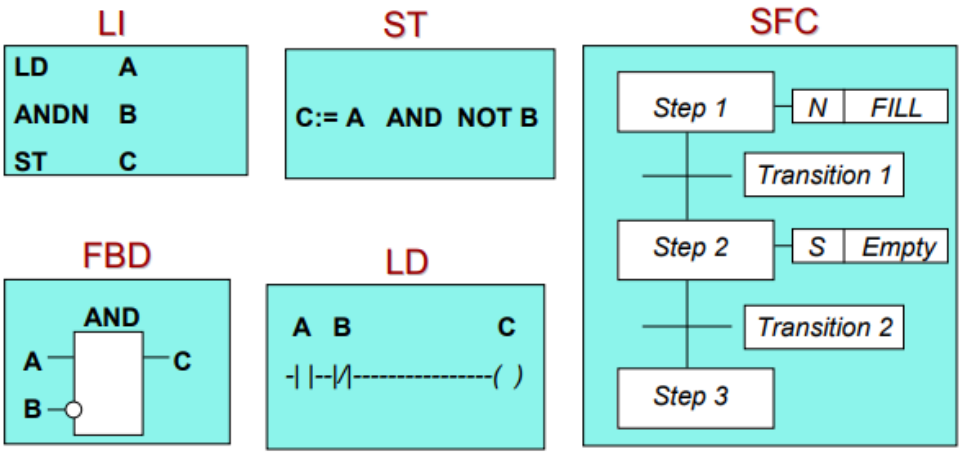


Ilustración 1.1.6-1 Listado de distintos lenguajes de programación disponibles según normativa.

Lenguaje KOP (lenguaje de contactos, de escalera, Ladder, LD)

KOP es un lenguaje de programación gráfico. Su representación es similar a los esquemas de circuitos. Los elementos de un esquema de circuitos, tales como los contactos normalmente cerrados y normalmente abiertos, así como las bobinas, se combinan para formar segmentos.

Para crear la lógica de operaciones complejas, es posible insertar ramas para los circuitos paralelos. Las ramas paralelas se abren hacia abajo o se conectan directamente a la barra de alimentación. Las ramas se terminan hacia arriba.

KOP ofrece instrucciones con cuadros para numerosas funciones, p. ej. matemáticas, temporizadores, contadores y transferencia. Este lenguaje está muy extendido, ya que suele ser comprensible para el personal de mantenimiento que no tiene conocimiento de automatización/programación, pero sí de circuitos eléctricos.

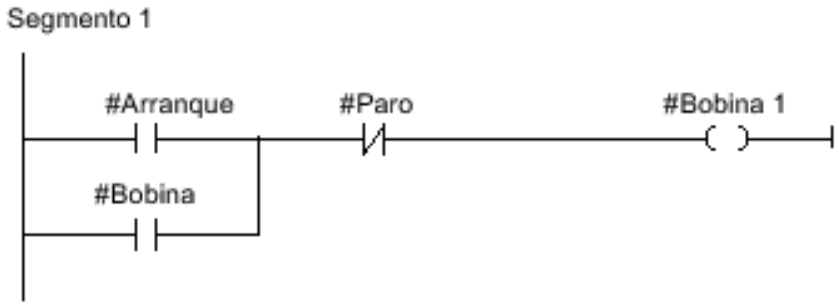


Ilustración 1.1.6-2 Ejemplo de Segmento programado en lenguaje Ladder

Lenguaje FUP (lenguaje de bloques de función, FBD)

Al igual que KOP, FUP es un lenguaje de programación gráfico. La representación de la lógica se basa en los símbolos lógicos gráficos del álgebra booleana. Las funciones matemáticas y otras operaciones complejas pueden representarse directamente en combinación con los cuadros lógicos. Para crear la lógica de operaciones complejas, inserte ramas paralelas entre los cuadros.

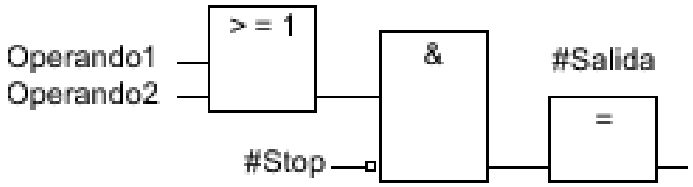


Ilustración 1.1.6-3 Ejemplo de bloques funcionales en lenguaje FUP

Lenguaje de programación SCL

SCL (Structured Control Language) es un lenguaje de programación de alto nivel que se orienta a PASCAL. Este lenguaje se basa en la norma DIN EN-61131-3 (internacional: IEC 1131-3).

Además de incluir elementos típicos del PLC, como entradas, salidas, temporizadores o marcas, SCL incluye elementos de lenguajes de programación de alto nivel:

- Expresiones
- Asignaciones de valor
- Operadores
- Control del programa

SCL ofrece instrucciones prácticas para el control del programa que permiten realizar, por ejemplo, ramas, bucles o saltos del programa.

Por estos motivos, SCL es especialmente adecuado para los siguientes campos de aplicación: Gestión de datos; Optimización de procesos; Gestión de recetas; Tareas matemáticas/estadísticas

```
 9  //Ejemplos SCL (C) REEA 2013
10  //Activar una salida (Set)
11 IF "Entrada1"=true THEN
12    "Salida1":=true;
13 END_IF;
14  //Desactivar una salida (Reset)
15 IF "Entrada2"=true THEN
16    "Salida1":=false;
17 END_IF;
```

Ilustración 1.1.6-4 Ejemplo de programación SCL

Lenguaje de programación AWL (sólo para S7-1500)

AWL es un lenguaje de programación basado en un texto con el que se podrán programar bloques de código.

El programa AWL se divide en segmentos. Cada segmento puede tener una o varias líneas. La numeración de las líneas comienza en cada segmento con 1 y continuará de manera ascendente con cada nueva línea. En las líneas del segmento se programarán las instrucciones individuales de AWL, pudiéndose indicar sólo una instrucción AWL por línea. Cada instrucción representa una prescripción de trabajo para la CPU. La CPU ejecuta las instrucciones de arriba abajo.

Lenguaje de programación GRAPH (sólo para S7-1500):

GRAPH es un lenguaje de programación gráfico para crear controles secuenciales. Permite programar controles secuenciales de manera clara y rápida utilizando cadenas secuenciales. El proceso se descompone en etapas individuales con un alcance funcional delimitable y se organiza en las cadenas secuenciales. En las etapas individuales se determinan las acciones que se deben ejecutar. El paso entre las etapas lo forman las transiciones. Estas contienen condiciones para pasar a la etapa siguiente.

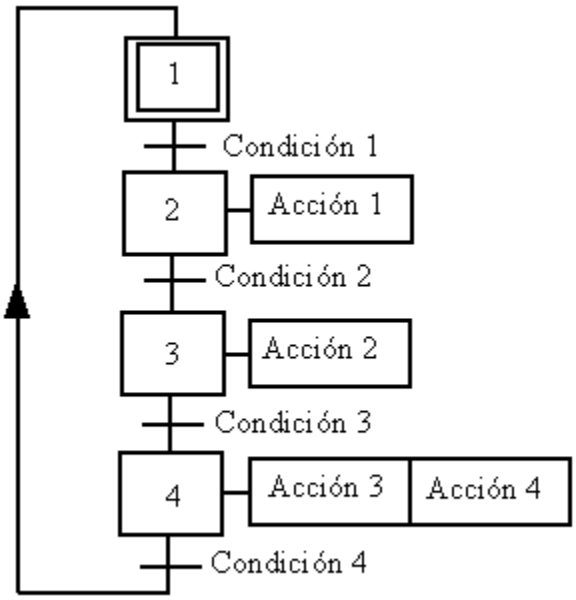


Ilustración 1.1.6-5 Ejemplo de bloque programado con lenguaje Graph (similar a programación Grafcet).

1.1.7 CONCLUSIONES.

En esta unidad hemos aprendido:

- Información relativa al funcionamiento del PLC, necesaria para poder estructurar nuestro proyecto de una manera adecuada a la realidad física que tenemos.
- A estructurar los distintos elementos principales (OB, FB, FC, DB) de un proyecto, de manera que podamos realizar una estructura modular simple y fácil de seguir.
- A saber los distintos estados en que se puede encontrar un PLC (RUN, STOP), así como las tareas que realiza en cada uno de ellos.
- A saber cómo se organiza la memoria y la memoria remanente, para saber en qué lugares se encuentra la información del proyecto.
- A conocer los distintos tipos de datos que se pueden manejar en un equipo.
- Los principales criterios a seguir a la hora de diseñar un proceso de automatización.
- Que existen distintos tipos de lenguaje de programación, con sus ventajas e inconvenientes, que nos permiten realizar las tareas de maneras distintas.

1.1.8 EXAMEN.

1. En un PLC existen únicamente 2 tipos de lenguajes de programación.
    a. Verdadero, lenguaje de contactos y lenguaje de bloques funcionales.
    b. Falso, existen más de 2.
    c. Falso, sólo hay 1, lenguaje de contactos, por su analogía con los esquemas eléctricos.
2. Un programa modular tiene la ventaja de...
    a. Estar realizado en partes pequeñas, que son más fácilmente manejables.
    b. Resultar más rápido en ejecución que uno lineal.
    c. No necesitan entradas ni salidas.
3. Existen modificadores a la hora de llamar a una zona de memoria, en función de si necesitamos acceder a un byte o a un Word.
    a. Si, verdadero.
    b. Falso, sólo podemos acceder al dato en el formato que se haya creado.
    c. Verdadero, siempre que esté conectado al PLC on-line.
4. Cada FB tiene asociado un único DB que se debe llamar igual que el FB.
    a. Cierto, la llamada es transitiva y bipolar.
    b. Cierto, es la única manera que tiene el programa para comprobar que el conjunto FB+DB está correctamente definido.
    c. Falso, puede tener varios DB de instancia con distintos nombres.
5. El lenguaje SCL, debido a sus características, está recomendado:
    a. Para programar Servidor Web.
    b. Para PLC sólo gama S7-1500.
    c. Para Gestión de datos; Optimización de procesos; Gestión de recetas; Tareas matemáticas/estadísticas.
6. Durante el ciclo de arranque, el PLC…
    a. Ejecuta un OB específico de arranque (depende de la definición y número de OBs que se hayan creado en proyecto).
    b. Ejecuta el ciclo principal, como siempre.
    c. Necesita que en modo on-line se active la orden de RUN
7. La diferencia entre FB y FC es...
    a. La zona de memoria que ocupan, que se designa por letras igual que en Windows.
    b. La existencia de un DB de instancia asociado a los FB, mientras que FC no los tiene.
    c. Ambos pueden tener DB de instancia, pero el del FC es virtual y local.
8. La memoria de entradas y salidas se denomina…
    a. I y Q respectivamente.
    b. IB y QB respectivamente.
    c. I y O respectivamente.
9. Para impedir la pérdida de datos tras un corte de alimentación, es posible definir que ciertos datos sean...
    a. resistentes.
    b. preponderantes.
    c. remanentes

10. Únicamente puede haber un OB cíclico, y de este llamar a otros OB, FB, FC o DB.
    a. Verdadero, pero un OB no puede llamar a otro OB, a menos que sea un OB de llamada remanente.
    b. Verdadero, es la estructura modular típica.
    c. Falso, puede haber varios OB cíclicos. Su ejecución depende del número que se asigna a cada uno de ellos. Para ejecutar uno, el anterior debe haber finalizado.

1.1.9 AUTOCOMPROBACIÓN.

1. Solución: b
2. Solución: a
3. Solución: a
4. Solución: c
5. Solución: c
6. Solución: a
7. Solución: b
8. Solución: a
9. Solución: c
10. Solución: c

1.1.10 PROPUESTAS.

Se trata de un tema muy teórico, por lo que es complejo realizar propuestas prácticas

1.1.10.1 PROPUESTA 1.

Se propone de todos modos:

1. Buscar información sobre los lenguajes de programación en internet.
2. En relación a los pasos iniciales a seguir antes de comenzar la programación, se propone realizar un listado de los mismos aplicados al caso práctico de realización de un proyecto de automatización de un semáforo de vehículos.

1.1.11 LISTADO DE IMÁGENES.

ILUSTRACIÓN 1.1.2-1 TAREAS A EJECUTAR POR CPU DURANTE ARRANQUE Y EN ESTADO RUN .4
ILUSTRACIÓN 1.1.2-2 LISTADO CON DISTINTOS TIPOS DE OB PARA CREAR EN UN PROYECTO....6
ILUSTRACIÓN 1.1.2-3 TIPOS DE DATOS QUE PUEDE MANEJAR UNA CPU 10
ILUSTRACIÓN 1.1.5-1 EJEMPLOS DE ESTRUCTURA DE PROGRAMA LINEAL Y ESTRUCTURA MODULAR (MÁS COMÚN).....12
ILUSTRACIÓN 1.1.5-2 ESTRUCTURA MODULAR DE PROGRAMACIÓN. EL BLOQUE A SE EJECUTA Y LLAMA A LA EJECUCIÓN DEL BLOQUE B. 13
ILUSTRACIÓN 1.1.5-3 ESTRUCTURA MODULAR DE PROGRAMACIÓN, ANIDAMIENTO (LLAMADA DE FUNCIONES DENTRO DE OTRAS FUNCIONES). 13
ILUSTRACIÓN 1.1.5-4 EJEMPLO DE FB LLAMADO EN DISTINTOS PUNTOS DE UN PROGRAMA CON DISTINTOS DB DE INSTANCIA 14
ILUSTRACIÓN 1.1.6-1 LISTADO DE DISTINTOS LENGUAJES DE PROGRAMACIÓN DISPONIBLES SEGÚN NORMATIVA.....14
ILUSTRACIÓN 1.1.6-2 EJEMPLO DE SEGMENTO PROGRAMADO EN LENGUAJE LADDER.....15
ILUSTRACIÓN 1.1.6-3 EJEMPLO DE BLOQUES FUNCIONALES EN LENGUAJE FUP 15
ILUSTRACIÓN 1.1.6-4 EJEMPLO DE PROGRAMACIÓN SCL 16
ILUSTRACIÓN 1.1.6-5 EJEMPLO DE BLOQUE PROGRAMADO CON LENGUAJE GRAPH (SIMILAR A PROGRAMACIÓN GRAFCET). 17