Programación por etapas RLL^{PLUS}

En este capítulo...

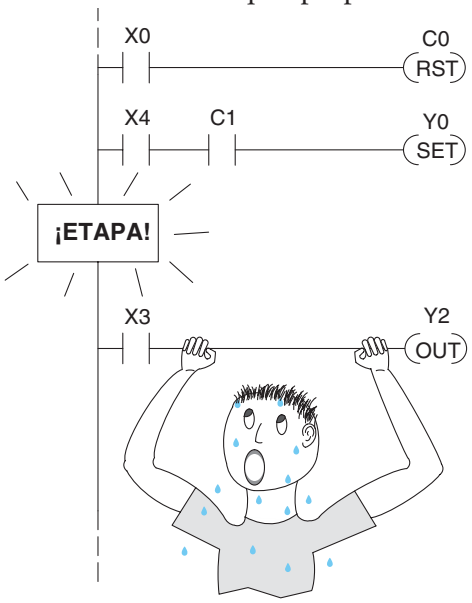
Introducción a la programación de etapas 7–2
Venciendo el temor de programar por etapas 7–3
Diseñando diagramas de transición de estados 7–3
Usando la instrucción de salto de etapas para transiciones de estados 7–7
Ejemplo de programa de etapas: Controlador de lámpara con flip flop 7–8
Ejemplo de programa de etapas: Abridor de un portón de garaje 7–10
Consideraciones de diseño del programa de etapas 7–15
Conceptos de procesamiento paralelo 7–19
Instrucciones de RLLPLUS 7–21
Preguntas y respuestas acerca de la programación de etapas 7–27

Introducción a la programación de etapas

La programación por etapas (etapa programming) le permite tener una forma de organizar y programar aplicaciones complejas con relativa comodidad, cuando es comparado a programar soluciones puramente con lógica ladder (RLL). La programación por etapas no reemplaza ni anula el uso del tradicional programa ladderbooleano. También se le llama RLL^{Plus}. Usted no tendrá que descartar ninguna experiencia ni entrenamiento que ya tenga. La programación por etapas le permite simplemente dividir y organizar un programa en grupos de instrucciones ladder llamado etapas. Esto le permite un desarrollo de un programa más rápido y más intuitivo del que proporciona un tradicional programa ladder.

Venciendo el temor de programar por etapas

Muchos programadores de PLC en la industria se sienten confortables usando RLL para cada programa de PLC que ellos escriben... pero a menudo permanece escéptico o aún temeroso del aprendizaje de técnicas nuevas tales como la programación por etapas. Aunque RLL es excelente para resolver relaciones de lógica booleana, tienen también desventajas:



- Los programas grandes pueden llegar a ser casi inmanejables, a causa de una falta de estructura.
- En RLL los enclavamientos deben ser hechos a partir de relevadores auto enclavados
- Cuándo un proceso se detiene es difícil de encontrar el renglón donde el error ocurrió.

Es fácil de ver que estas ineficacias consumen un tiempo adicional, y el tiempo es dinero. ¡La programación por etapas vence estos obstáculos! **¡Creemos que gastar un momento para estudiar el concepto de etapas es una de las mejores inversiones que un programador de PLC puede hacer para programar con velocidad y con eficiencia!**

Por lo tanto lo alentamos a estudiar programación por etapas y agregarlo a su "bloque de herramientas" de técnicas de programar. Este capítulo está diseñado para aprender a programar con etapas. Para mejores resultados:

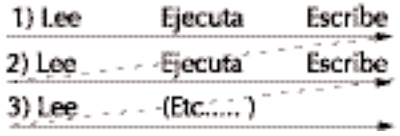
- Comience en el principio y no se salte sobre ninguna sección.
- Estudie cada concepto de programación de etapas trabajando con cada ejemplo. Los ejemplos se construyen progresivamente uno al otro.
- Lea las Preguntas y Respuestas al final del capítulo para una revisión rápida.

Diseñando diagramas de transición de estados

Introducción a estados de procesos

Los técnicos familiarizados con la ejecución de programas ladder saben que la CPU debe barrer el programa ladder repetidas veces. Sus tres pasos básicos son:

1. Lea las entradas
2. Ejecute el programa de escalera
3. Escriba las salidas



El beneficio es que un cambio en las entradas puede afectar las salidas en apenas unos pocos milisegundos.

La mayoría de los procesos de fabricación se componen de una serie de actividades o condiciones que duran varios segundos, minutos o aún horas. Podríamos llamar éstos "estados de proceso", que pueden estar activos o inactivos en algún tiempo determinado. Un desafío para programas de RLL es que cierto evento de entrada puede durar solamente un breve instante. Típicamente creamos relevadores autoenclavadods en RLL para mantener el evento de entrada para mantener un estado de proceso por una duración requerida.

Podemos organizar y poder dividir la lógica en secciones llamadas "etapas" que representan estados del proceso. Pero antes de describir las etapas con detalles, le diremos el secreto para la comprensión de la programación por etapas: diagramas de transición de estado.

Necesidad de diagramas de estado

A veces necesitamos olvidarnos de la naturaleza de PLCs en el sentido del ciclo continuo, y enfocar nuestro pensamiento hacia estados del proceso que necesitamos identificar. El análisis claro, pensado y conciso de una aplicación nos da la mejor oportunidad para escribir programas eficientes sin errores. ¡Los diagramas del estado son apenas una herramienta de ayuda para dibujar un retrato de nuestro proceso! ¡Usted descubrirá que si podemos obtener del retrato correcto, nuestro programa estará correcto también!

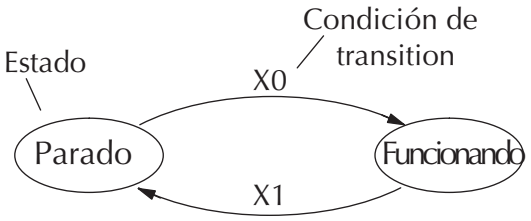
Proceso de 2 estados

Consideremos el proceso sencillo mostrado a la derecha, que controla un motor industrial. Usaremos un botón momentáneo verde para prender el motor, y un rojo para apagarlo. El operario de la máquina apretará el botón apropiado por aproximadamente un segundo. Los dos estados de nuestro proceso es **Funcionando** y **Parado**.

El próximo paso deberá dibujar un diagrama de transición de estados, como mostrado a la derecha. Muestra los dos estados **Funcionando** y **Parado**, con dos líneas de transición intermedias. Cuándo el evento de la entrada X0 es verdadero, pasa de **Parado** a **Funcionando**. Cuándo la entrada X1 es verdadera, se pasa de **Funcionando** a **Parado**.

Si ha seguido la explicación, está muy cerca de agarrar el concepto y el poder de resolver el problema de creación de diagramas de transición de estado. La salida del controlador es X0, que es verdadera cada vez que estamos en el estado **Funcionando**.

En un sentido booleano, el estado X0= Funcionando. Luego, aplicaremos el diagrama de estados primero como RLL y luego como un programa por etapas. Esto lo ayudará a ver la relación entre los dos métodos en la resolución de problemas

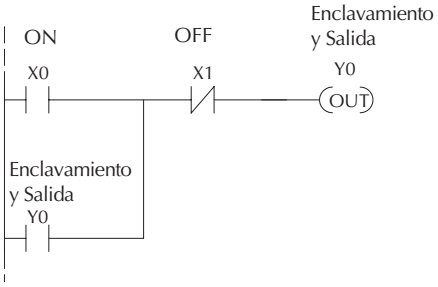


Ecuación de salida: Y0 = On

El diagrama de transición de estados es un retrato de la solución que necesitamos crear. La belleza de esto es: expresa el problema independientemente del idioma de programación que podríamos usar para ejecutarlo. ***¡En otras palabras, dibujando el diagrama nosotros hemos resuelto ya el problema del control!***

Primero, traduciremos el diagrama del estado a RLL tradicional. Entonces mostraremos cuán fácil deberá traducir el diagrama en una solución de programación por etapas.

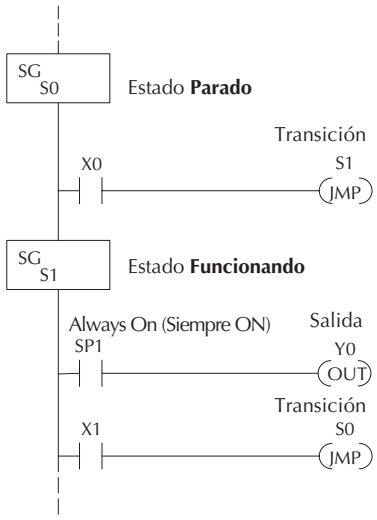
Equivalente RLL



La solución de RLL se muestra a la derecha. Se compone de un relevador de control y salida al misamo tiempo que se auto enclava, Y0. Cuándo el botón (X0) es apretado, la bobina de salida Y0 prende y el contacto Y0 en la segunda fila se cierra con lo cual mantiene energizada la bobina. De modo que X0 puede abrir y Y0 permanece activado después que el contacto X0 abre. El contactor del motor también se cierra ya quye está conectado a Y0, de modo que el motor ahora está **Funcionando**.

Cuándo se activa el botón X1, se abre el contacto normalmente cerrado X1 que libera la bobina y la salida Y0 se apaga cuando la bobina Y0 se desactiva. El motor ahora está **Parado**.

Equivalente con etapas



La solución del programa con etapa se muestra a la derecha. Los dos bloques S0 y S1de etapas corresponden a los dos estados **Funcionando** y **Parado**. Los renglones debajo de cada bloque de etapas pertenecen a cada etapa respectiva. ¡Esto significa que el PLC sólo tiene que barrer esos renglones cuando la etapa correspondiente es activa!

Por ahora, asumamos que comenzamos en el estado **Parado**, de modo que la etapa S0 está activa. Cuándo se aprieta el botón X0, ocurre una transición de etapas. Se ejecuta la instrucción JMP S1, que apaga simplemente el bit de la etapa S0 y prende el bit de la etapa S1. ¡Así que en el próximo barrido del PLC, la CPU no ejecutará la etapa S0, sino que ejecutará la etapa S1!

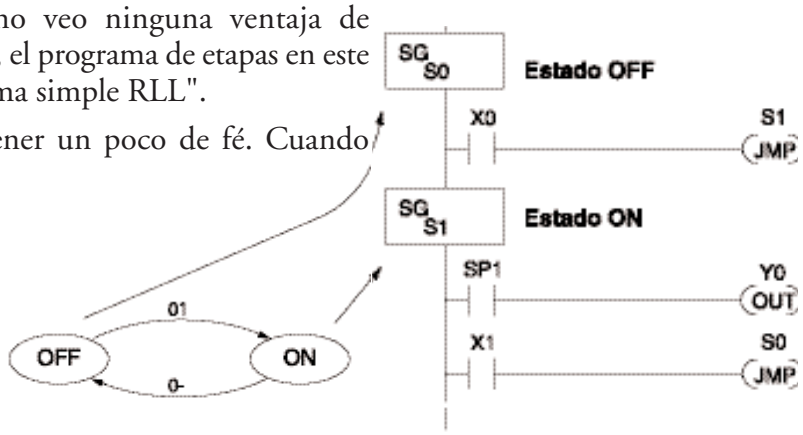
En el estado **Funcionando** (etapa S1), queremos que el motor siempre esté **Funcionando**. El contacto SP1 especial del relevador se define como siempre activado, de modo que Y0 activa el contactor que prende el motor.

Cuándo se aprieta el botón X1, ocurre una transición al estado **Parado**.

Se ejecuta la instrucción JMP S0, que apaga simplemente el bit de la etapa S1 y prende el bit de la etapa S0. En el próximo barrido del PLC , la CPU no ejecutará la etapa S1, de modo que la salida Y0 se apagará. El estado **Parado** (S0) estará listo para el próximo ciclo.

Hagamos comparaciones entre RLL y PLL[plus]

Ud. puede estar pensando "no veo ninguna ventaja de programar en etapas... de hecho, el programa de etapas en este caso es más largo que el programa simple RLL".

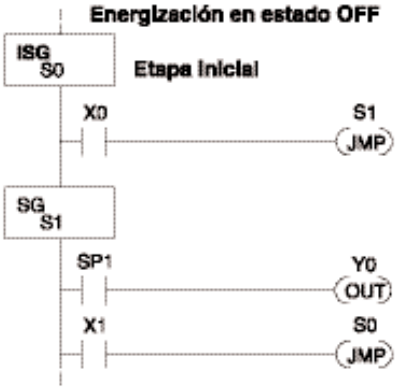


Bien, ahora es el tiempo de tener un poco de fé. Cuando crecen en complejidad los problemas de control, la programación por etapas gana rápidamente en la sencillez, el tamaño del programa, etc.

Por ejemplo, considere el diagrama adyacente. Note cuán fácil es establecer una correlación de estados del diagrama de la transición de estados al programa de etapas a la derecha. ¡Ahora, desafiamos a cualquiera a identificar los mismos estados en el programa de RLL en la página previa!

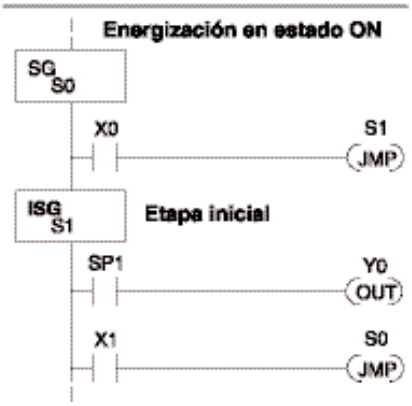
Etapas iniciales

Durante la energización del PLC y en la transición del modo Program para RUN, el PLC siempre comienza con todas etapas (**SG**) normales desactivadas. Las etapas del programa mostrado hasta ahora no tienen realmente una manera de iniciar el programa (porque no se examinan renglones a menos que la etapa esté activa).

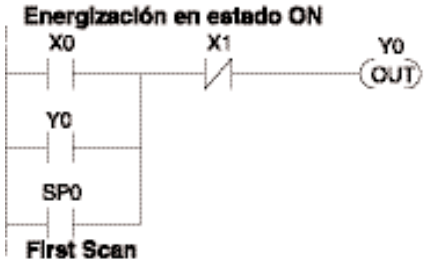


Asumamos que siempre comenzaremos en el Estado **Parado**, que es como funciona el programa en ladder. La etapa inicial **ISG** se define para ser activa en la energización.

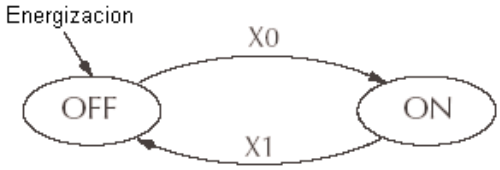
En el programa modificado a la derecha, hemos cambiado la etapa S0 al tipo **ISG**. Esto asegura que el PLC examina el contacto X0 después de la energización, porque la etapa S0 es activa. ¡**Después de la energización, una etapa inicial (ISG) trabaja como cualquier otra etapa**!



Podemos cambiar ambos programas para que el motor esté ON o funcionando en la energización. En el programa RLL a la izquierda, debemos agregar el relevador SP0 que se cierra en el primer barrido de la CPU, que enclava Y0. En el ejemplo de etapas a la derecha, hacemos simplemente la etapa S1 una etapa inicial (ISG) en vez de S0.



Podemos marcar el estado deseado de la energización como mostrado en la figura adyacente, que nos ayuda a recordar de usar la etapa inicial ISG apropiada cuándo se crea un programa de etapas. Es permisible tener tantas etapas iniciales ISG como requiera el proceso.



Qué hacen los bits de etapas

Recuerde que una etapa es apenas una sección del programa que es activa o inactiva en un momento dado. Todos los bits de etapas (S0 a 1777) residen en la memoria imagen del PLC como bits de estado individuales. Cada bit de etapa es un booleano 0 o 1 en cualquier momento.

La ejecución del programa siempre lee los renglones de arriba hacia abajo, y de la izquierda a la derecha. El dibujo debajo muestra el efecto del estado del bit de etapa. Los renglones debajo de la instrucción de etapa continuan a ser ejecutados hasta que la próxima instrucción de etapa o el fin del programa pertenezca a la estapa 0. Su operación equivalente se muestra a la derecha. Cuándo la etapa S0 es verdadera, los dos renglones conducen corriente.

- Si el bit de la etapa S0 = 0, los renglones *no son barridos* (ejecutados).
- Si el bit de la etapa S0 = 1, los renglones *son barridos* (ejecutados).

Como se ve el programa por etapas

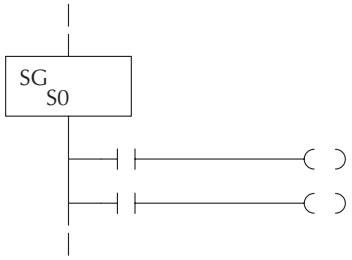
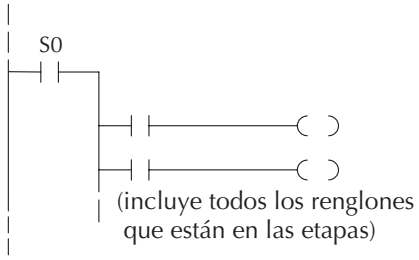


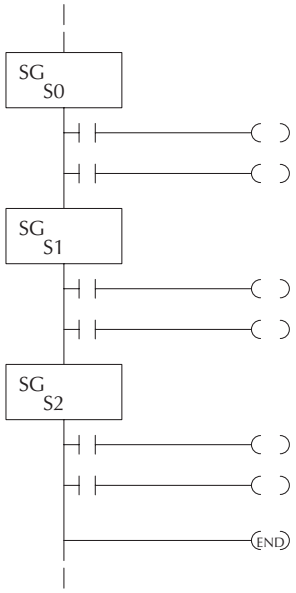
Diagrama ladder equivalente



Características de la instrucción de etapas

Los bloques de etapas en el riel de poder izquierdo dividen los renglones del programa en etapas. Algunas reglas para las etapas son:

- **Ejecución** – Sólo se ejecuta lógica en etapas activas en cualquier barrido.
- **Transiciones** – Las instrucciones de transición de etapas surten efecto en la próxima ocurrencia de las etapas implicadas.
- **Numeración octal** – Las etapas se numeran en octal, como puntos de entradas y salidas, etc. de modo que "S8" no es válido.
- **Cantidad de etapas posibles** – El DL06 ofrece hasta 1024 etapas (S0 a 1777 en octal).
- **No hay etapas duplicadas**–Cada número de etapa es único y puede ser usado solamente una vez.
- **Cualquier orden** – Usted puede saltarse los números y ordenar los números de etapas en cualquier orden.
- **Ultima etapa** – La última etapa en el programa incluye todos los renglones de su bloque de etapa hasta la bobina END.



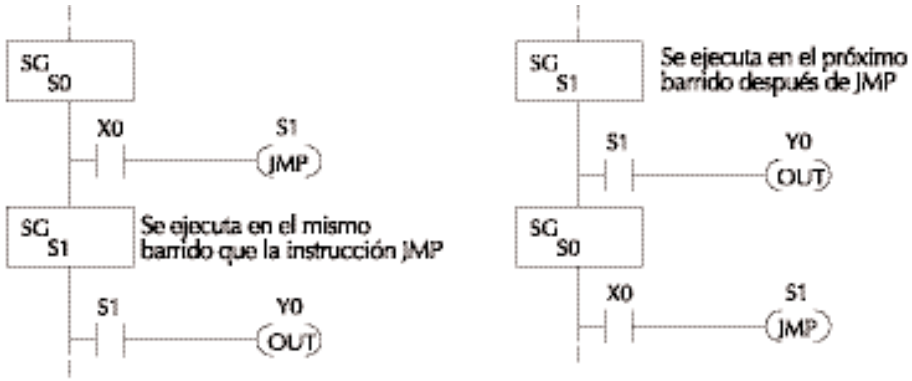
Usando la instrucción de salto de etapas para transiciones de estados

Las instrucciones de salto JMP, SET y RESET

La instrucción de salto de etapas JMP que hemos usado desactiva la etapa en que ocurre la instrucción, activando la etapa definida en la instrucción JMP. Vea la transición de estados mostrada abajo. Cuándo el contacto X0 se energiza, ocurre una transición de estados de S0 a S1. Los dos ejemplos de etapa mostrados debajo son equivalentes. La instrucción de salto de etapas JMP es igual a un reset de etapa de la etapa actual, más una instrucción SET para la etapa a que queremos hacer la transición.

Lea con mucho cuidado, por favor - La instrucción de salto JMP es muy mal entendida. El "salto" no ocurre inmediatamente como un GOTO o una instrucción de control de programa GOSUB cuando ejecutada. Así es cómo trabaja:

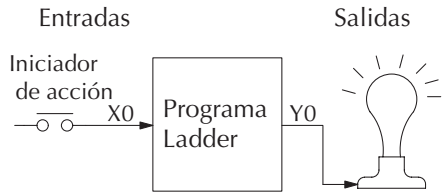
- La instrucción de salto coloca en OFF el bit de la etapa en que ocurre. ¡Todos los renglones en la etapa terminan la ejecución durante el barrido corriente, *aunque haya otros renglones en la etapa debajo de la instrucción de salto!*
- El estado OFF será vigente en el siguiente barrido, de modo que la etapa que ejecutó la instrucción de salto previamente será inactiva y descartada.
- El bit de etapa llamado en la instrucción de salto JMP se colocará ON inmediatamente, de modo que la etapa se ejecutará en su próxima ocurrencia. En el programa mostrado abajo a la izquierda, la etapa S1 se ejecuta durante el mismo barrido que el JMP S1 ocurre en S0. En el ejemplo a la derecha, la etapa S1 ejecuta en el barrido siguiente después que el JMP S1 se ejecuta, porque la etapa S1 se localiza arriba de la etapa S0.



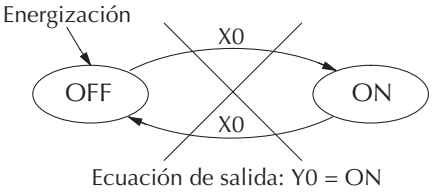
***Nota**: Asumimos que comenzamos con la etapa S0 activa y la etapa S1 inactiva en ambos ejemplos.*

Ejemplo de programa de etapas: Controlador de lámpara con flip flop

Proceso de 4 estados



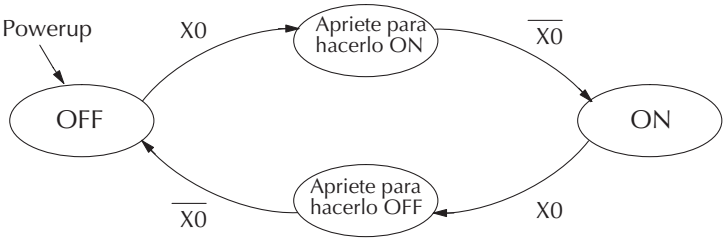
En el proceso mostrado a la derecha, usamos un botón momentáneo normal para controlar una lámpara. El programa enclavará la entrada del interruptor, de modo que empujaremos y liberaremos el botón para prender la luz y luego lo empujaremos y soltaremos para apagarlo (llamada a veces función flip flop). Claro, podríamos comprar un interruptor mecánico con la acción de mantener el estado ya en él. ¡Sin embargo, este ejemplo es educativo! Dibujemos ahora el diagrama de transición de estados.



Un primer enfoque típico sería usar X0 para ambas transiciones (como el ejemplo mostrado a la derecha). Sin embargo, *esto no es correcto* (siga leyendo por favor).

Note que este ejemplo difiere del ejemplo del motor porque ahora tenemos solamente un botón. Cuándo nosotros apretamos el botón se tienen dos condiciones de transición. Haremos una transición alrededor del diagrama de estado a una alta velocidad. ¡Si se usan etapas, esta solución prendería la luz cada barrido (obviamente indeseable)!

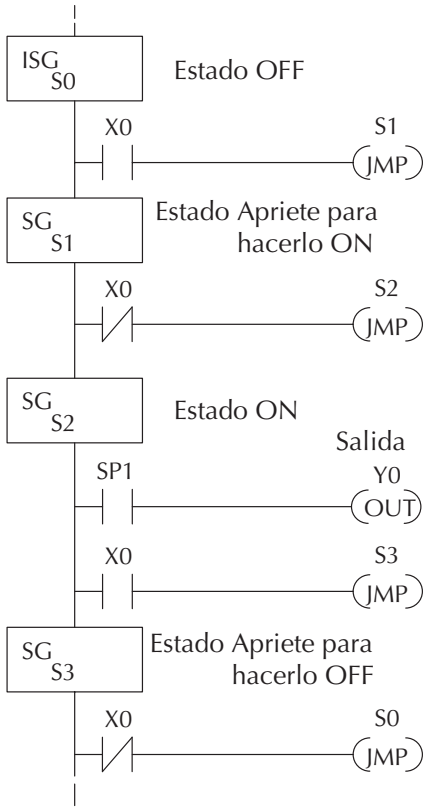
La solución deberá hacer que el apretar y soltar el botón sean eventos separados. Vea al nuevo diagrama de transición de estados de la figura de abajo. En la energización entramos el estado OFF.



Cuándo se aprieta el botón X0, entramos al estado de "apriete para hacerlo ON". Cuándo se suelta, entramos al estado ON. Note que X0 con la barra encima denota X0 negado.

Cuándo se está en el estado ON, otro ciclo de apretar y soltar nos lleva de forma similar al estado OFF. Ahora tenemos dos estados únicos (OFF y ON) usados cuando el botón se suelta, que es lo que se necesitaba para resolver el problema de control.

El programa equivalente de etapas se muestra a la derecha. El estado deseado de la energización es OFF, de modo que haremos S0 una etapa inicial ISG. En el estado ON agregamos el contacto del relevador especial SP1 que es siempre ON. ¡Note que aunque los programas van siendo más complejos, es todavía fácil poner en correlación el diagrama de transición de estado con el programa de etapas!



Cuatro pasos para escribir un programa por etapas

Por ahora, usted ha notado probablemente que seguimos los mismos pasos para resolver cada problema ejemplo. Los pasos le vendrán probablemente automáticamente a su memoria si usted trabaja en todos los ejemplos en este capítulo. Es útil tener una lista de verificación para indicarnos la resolución de problemas. Los pasos siguientes resumen el procedimiento del diseño del programa de etapas:

1. Escriba una descripción de la operación de la aplicación

Describa todas funciones del proceso en sus propias palabras. Comience listando lo que sucede primero, luego lo que viene en segundo lugar, en tercer lugar, etc. Si usted encuentra que hay demasiadas cosas que suceden inmediatamente, trate de dividir el problema en más de un proceso. Recuerde, usted puede tener todavía los procesos comunicándose uno con otro para coordinar su actividad completa.

2. Dibuje un diagrama de bloques

Las entradas representan toda la información que el proceso necesita para hacer decisiones y las salidas se conectan a todos los aparatos controlados por el proceso.

- Haga una lista de entradas y salidas del proceso.
- Asigne números de entradas y salidas (X y Y) a entradas y salidas físicas.

3. Dibuje el diagrama de transición de estados

. El diagrama de transición de estados describe la función central del diagrama de bloques, leyendo entradas y generando salidas.

- Identifique y denomine los estados del proceso.
- Identifique el o los eventos requeridos para cada transición entre estados.
- Asegúrese que el proceso tenga una manera de volver a encender o es cíclico.
- Escoja el estado de energización para su proceso.
- Escriba las ecuaciones de salidas.

4. Escriba el programa por etapas

Traduzca el diagrama de transición de estados en un programa de etapas.

- Haga cada estado una etapa. Recuerde de numerar las etapas en octal. Hay hasta 1024 etapas totales disponibles en el DL06, numeradas 0 a 1777 en octal
- Ponga la transición lógica dentro de la etapa que origina cada transición (la etapa que es abandonada por cada flecha).
- Use una etapa inicial ISG para cualquiera estado que deba ser activo en la energización.
- Coloque las salidas o las acciones en las etapas apropiadas.

Usted notará que los pasos 1 a 3 solamente nos preparan para escribir el programa de etapas en el paso 4. Sin embargo, el programa se escribe virtualmente a causa de la preparación anterior. ¡Luego Ud. será capaz de comenzar con una descripción escrita de una aplicación y crear un programa de etapas en una sesión fácil!

Ejemplo de programa de etapas: Abridor de un portón de garaje

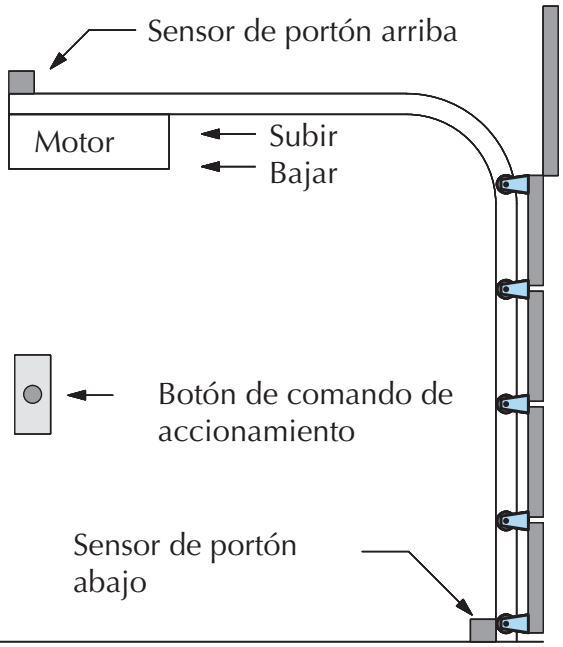
Ejemplo del control para abrir un portón de garaje.

En este programa ejemplo crearemos un controlador de apertura de portón de garaje. ¡La mayoría de los lectores están familiarizados con esta aplicación!

El primer paso que debemos tomar deberá ser describir cómo trabaja un abridor de portón. Comenzaremos describiendo la operación básica, y luego agregaremos más características. Los programas de etapas son muy fáciles de modificar.

El controlador del portón del garaje tiene un motor que levanta o baja el portón al apretar un botón. El dueño del garaje aprieta un botón momentáneo para levantar una vez el portón. Después que el portón está arriba, otro apriete del botón bajará el portón.

Para identificar las entradas y las salidas del sistema, es a veces útil hacer un croquis sus componentes principales, como mostrado en la figura a la derecha del lado del portón. El portón tiene un interruptor de límite arriba y uno abajo. Cada interruptor de límite se cierra sólo cuando el portón llega al fin del movimiento en la dirección correspondiente. Durante el movimiento no se acciona ningún interruptor límite.



El motor tiene dos entradas de comando: subir y bajar. Cuándo ninguna entrada es activada, el motor se para. El comando del portón es un botón sencillo, ya sea si es montado en la pared como mostrado, o es un control remoto de radio, o ambos, actuando como un control OR como dos contactos abiertos en paralelo.

Dibuje el diagrama de bloques

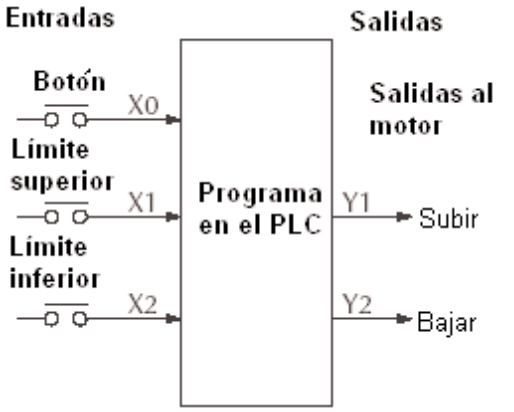
Se muestra a la derecha el diagrama de bloques del controlador.

La entrada X0 corresponde al control del portón (el botón).

La entrada X1 se energiza cuando el portón alcanza la posición superior.

La entrada X2 se energiza cuando el portón llega a la posición inferior.

Cuándo el portón esté posicionado entre arriba o abajo, ambos interruptores límite están abiertos. El controlador tiene dos salidas para manejar el motor. Y1 es el comando para levantar el portón (Subir), y Y2 el comando para bajar el portón (Bajar).



Dibuje el diagrama de estados

Ahora estamos listos para dibujar el diagrama de transición de estados. Como el ejemplo previo del controlador de una lámpara, esta aplicación tiene también solamente un interruptor para entrar el comando. Vea la figura de abajo.

- Cuando el portón está completamente abajo (Estado ABAJO), nada sucede hasta que X0 se energice. Al apretar y soltar el botón el portón pasa al estado de SUBIR, donde la salida Y1 se prende y causa que el motor levante el portón.
- Cuando el portón está completamente abajo (Estado ABAJO), nada sucede hasta que X0 se energice. Al apretar y soltar el botón el portón pasa al estado de SUBIR, donde la salida Y1 se prende y causa que el motor levante el portón.
- Nada sucede hasta que ocurra otro ciclo de apretar y soltar el botón X0. Eso hace que el portón pase al estado de BAJAR y se prende la salida Y2 para causa que el motor baje el portón. Se llega al estado ABAJO cuando el interruptor límite X2 se activa.

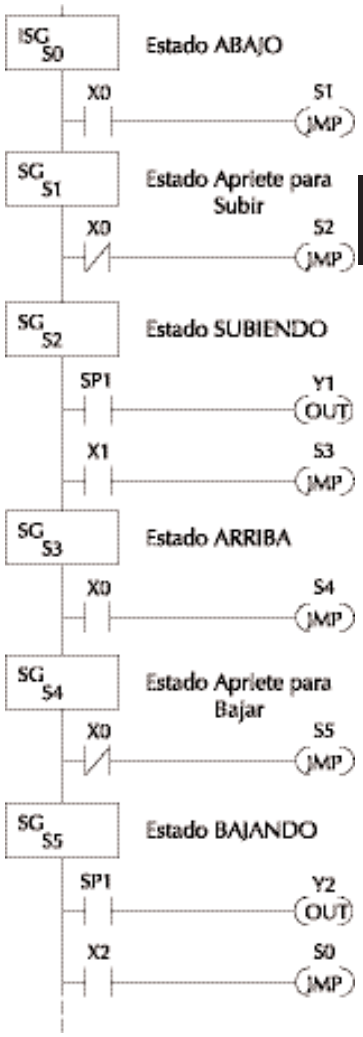
Ecuaciones de salidas: Y1 = SUBIENDO Y2 = BAJANDO

El programa equivalente de etapas se muestra a la derecha. Por ahora, asumiremos que el portón está abajo en la energización, así que el estado deseado en la energización es ABAJO. Hacemos S0 una etapa inicial ISG. La etapa S0 permanece activa hasta que se active el botón de control del portón. Luego hacemos la transición (JMP) a la etapa de transición "Apriete para subir", estado S1.

La acción de apretar y soltar el botón (etapa S1) nos conduce a la etapa de SUBIENDO, S2. Usamos el contacto SP1 siempre ON para energizar el motor con el comando de subir, Y1. Cuándo el portón alcanza la posición completamente levantada, el interruptor límite X1 se activa. Esto nos lleva a la etapa ARRIBA, S3, donde esperamos hasta que ocurra otro comando de control de portón.

Estando en la etapa S3, ARRIBA, apretando y soltando del botón nos llevará a la etapa S5, ABAJO, donde activamos Y2 para ordenar al motor bajar el portón. Esto continúa hasta que el portón alcance el interruptor límite inferior, X2 .

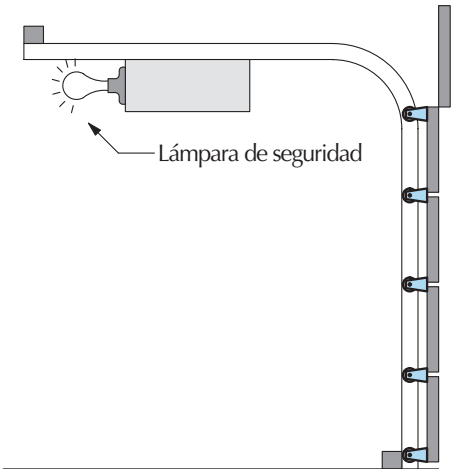
Cuándo X2 se cierra, saltamos de la etapa S5 ABAJO a la etapa S0, donde comenzamos.



NOTA: *La única cosa especial acerca de la etapa inicial ISG es que es automáticamente activa en la energización. Después, es igual que cualquier otra etapa.*

Agregue una lámpara para iluminar el garage por un corto tiempo

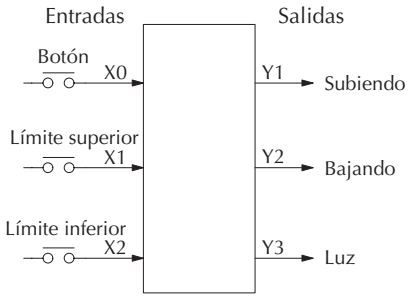
A continuación agregaremos una lámpara de seguridad al sistema del abridor del portón. Es mejor asegurarse que la principal función esté funcionando como hemos hecho y luego se agrega esta característica secundaria.



La luz de seguridad es normal en los abridores de portón de garaje comercialmente disponibles. Se muestra a la derecha junto con el motor. La lámpara se enciende con cualquier movimiento del portón y queda encendida por aproximadamente 3 minutos.

Esta parte del ejercicio le mostrará el uso de estados paralelos en nuestro diagrama del estado. En vez de usar la instrucción JMP, usaremos las instrucciones SET y RESET

Modifique los diagramas de bloque y de estado

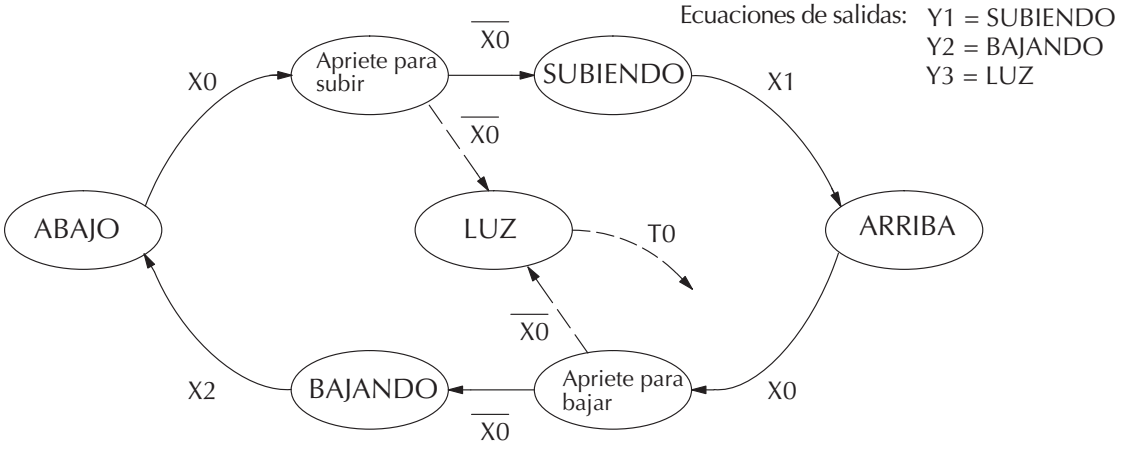


Para controlar la lámpara, le agregamos una salida a nuestro diagrama de bloques del controlador, mostrado a la derecha, Y3 es la salida de control de la lámpara. En el diagrama de abajo, agregamos un estado adicional llamado "LUZ".

Cuando el dueño del garaje aprieta el interruptor de control del portón, el estado de SUBIR o BAJAR es activo y el estado LUZ se hace activo al mismo tiempo. La línea al estado LUZ es segmentada, porque no es el sendero primario.

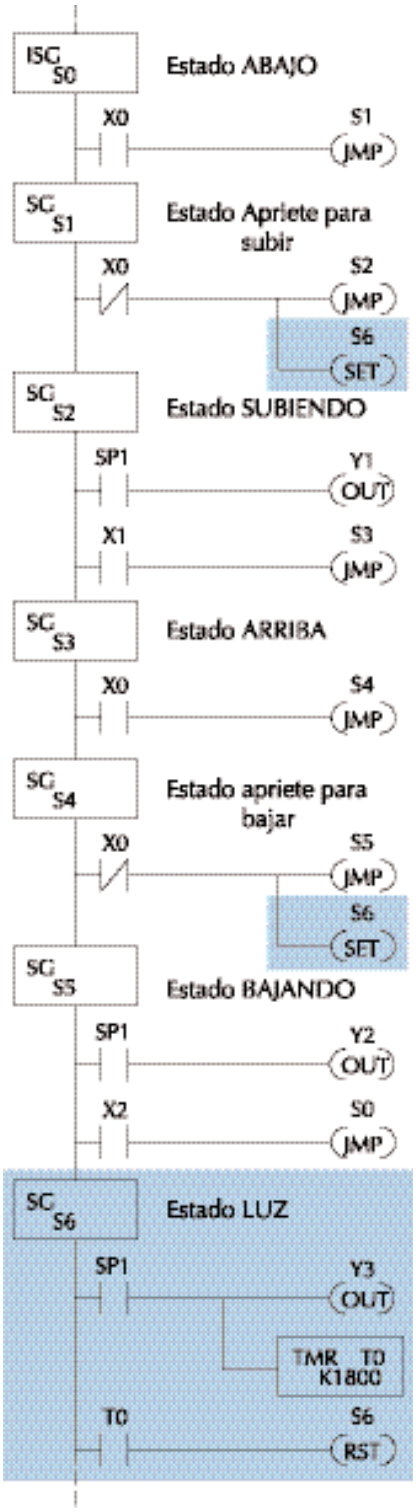
Podemos pensar en el estado LUZ como un proceso paralelo al estado de SUBIR y BAJAR. El paso al estado LUZ no es una transición (una etapa JMP), sino una instrucción SET.

En la lógica de la etapa LUZ, colocaremos un temporizador de 3 minutos. Cuándo el temporizador expira su tiempo, el bit de estado T0 del temporizador se activa y desactiva la etapa LUZ. ¡El paso para fuera de la etapa LUZ no va a ningún lugar, indica que la etapa LUZ se desactiva, y la lámpara se apaga!



Usando un temporizador dentro de una etapa

El programa modificado terminado se muestra a la derecha. Las áreas con sombra indican las partes agregadas del programa.



En la etapa S1 agregamos la instrucción SET etapa S6. Cuándo el contacto X0 abre, pasamos de S1 a dos estados activos nuevos: S2 y S6. En el estado S4, agregamos la mismas instrucción. De modo que cada vez que alguien aprieta el botón de control de' portón, la lámpara se enciende.

La mayoría de los programadores nuevos de etapa se preocupan donde colocar la etapa LUZ y cómo numerarla. ¡La buena noticia es que no importa!

- Escoja un número nuevo de etapa, y úselo para la etapa nueva y como referencia desde otras etapas.
- La colocación en el programa no es crítica, así que lo colocamos al fin del programa.

Usted talvez piense que cada etapa tiene que estar directamente debajo de la etapa anterior. Aunque es una buena práctica, no es necesario (eso es bueno, porque hay 2 localizaciones para la instrucción SET S6 lo que hace esto imposible). Los números de las etapas y cómo ellos son usados determinan como se hace la transición.

En la etapa S6, encendemos la lámpara energizando Y3. El contacto del relevador especial SP1 está siempre ON o activado.

El temporizador T0 cuenta el tiempo en un unidades de 0,1 segundo. Para lograr períodos de 3 minutos, calculamos:

K = 3 min. X 60 s/min

=> K = 1800 conteos

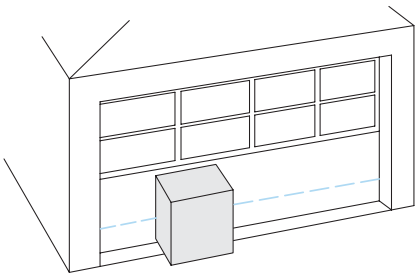
0,1 s/conteo

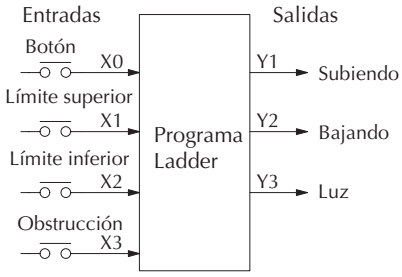
El temporizador está activado cuando la etapa S6 está activa. El bit T0 correspondiente del temporizador se activa cuando el temporizador cumple su tiempo. Así que en tres minutos transcurridos, T0=1 y la instrucción RESET S6 hace que la etapa sea inactiva.

Mientras la etapa S6 es activa y la lámpara está encendida, las transiciones de etapas en el transcurso primario continúan normalmente e independientemente de la etapa 6. Eso es, el portón puede subir o ir hacia abajo pero la lámpara estará encendida por exactamente 3 minutos.

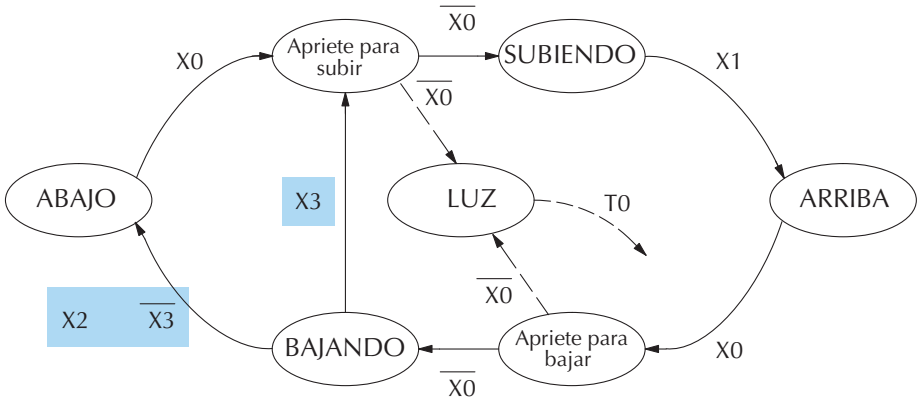
Agregue una parada de emergencia

Algunos abridores de portón de garaje detectan actualmente un objeto bajo el portón. Este evento para el descenso del portón. Generalmente es implementado con una fotocelda, y al pasar un objecto en frente al sensor y el portón bajando, este parará y comenzará a subir. Definiremos la característica de seguridad para trabajar de esta manera, agregando la entrada de la fotocelda al diagrama del bloque como mostrado a la derecha. X3 estará ON si un objeto está en el sendero del portón





Luego agregamos estos estados al diagrama de la transición de estados, mostrado en áreas sombreadas en la figura de abajo. Note el sendero nuevo de la transición a la cabeza del estado BAJAR. Si bajamos el portón y detectamos una obstrucción (X3), entonces saltamos al Estado de Apriete para subir. Hacemos esto en vez de saltar directamente al estado de SUBIR para dar a la salida BAJAR Y2 un barrido para apagarse, antes que la salida Y1 se energice.



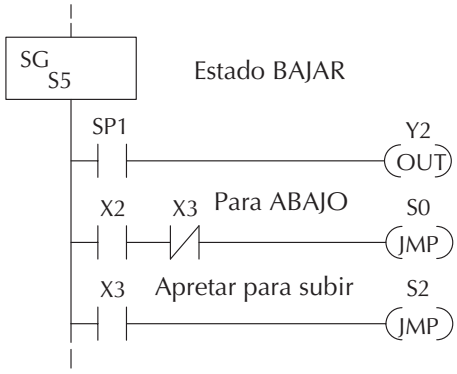
Transiciones exclusivas

Es teóricamente posible que el límite inferior (X2) y la obstrucción (X3) se podrían energizar en el mismo momento. En ese caso, "saltaríamos" simultáneamente a los estados **Apriete para subir** y **Apriete para bajar**, que no tiene sentido.

En vez de eso, le daremos prioridad a la obstrucción cambiando la condición de transición del estado ABAJO [X2 y (AND) no X3].

Esto asegura que el evento obstrucción tenga prioridad. Las modificaciones a la lógica que debemos hacer a la etapa MAS BAJA (S5) se muestra a la derecha. El primer renglón permanece igual. El segundo y tercer renglones implementan las transiciones que se necesitan.

Note el uso del contacto cerrado del relevador X3, que asegura que la etapa ejecute sólo una de las instrucciones JMP.

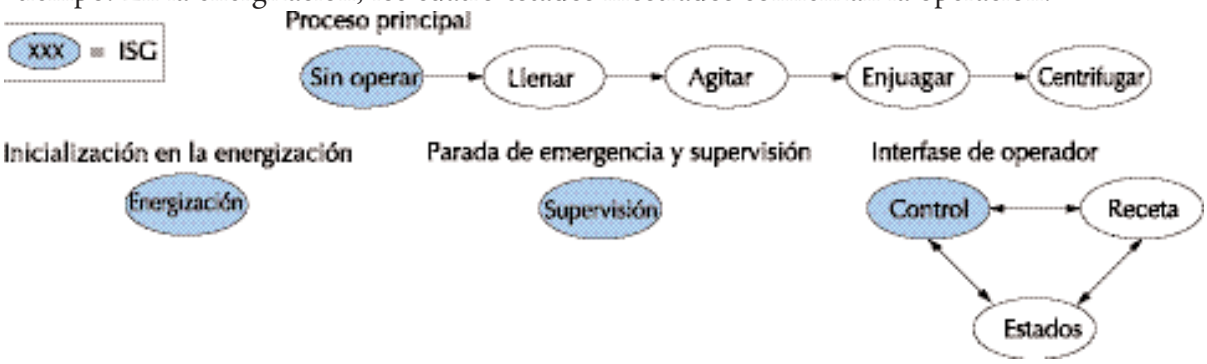


Consideraciones de diseño del programa de etapas

Organización del programa

Los ejemplos en este capítulo usaron hasta ahora un diagrama auto contenido del estado para representar el proceso principal. Sin embargo, se pueden tener múltiples procesos aplicados en etapas, todo en el mismo programa ladder. Los programadores nuevos de etapas tratan a veces de prender y apagar una etapa cada barrido, basado en la suposición falsa que sólo una etapa puede estar activa a la vez. Para renglones ladder que usted quiere ejecutar en cada barrido, colóquelos en una etapa que esté siempre activada.

La figura siguiente muestra una aplicación típica. Durante la operación de la actividad primaria de fabricación, los estados **Proceso principal**, **Inicialización en la energización**, **Parada de Emergencia** y **Supervisión de alarmas e interfase de operador** funcionan todos al mismo tiempo. En la energización, los cuatro estados mostrados comienzan la operación.



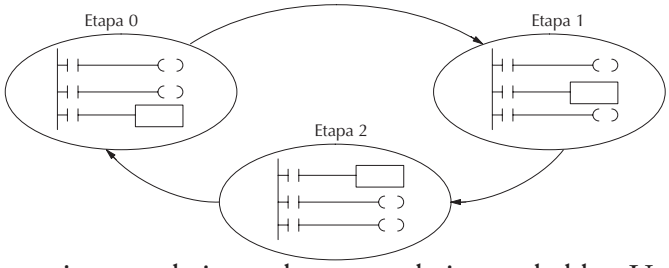
En una aplicación típica, las sucesiones separadas de etapas encima operan como sigue:

- **Inicialización en la energización** – Esta etapa contiene tareas hechas solamente una vez en la energización. El último renglón hace que se desactive la etapa, de modo que esta etapa es sólo activa por un barrido (o sólo tantos barridos como sean requeridos).
- **Proceso principal** Esta sucesión de etapas controla el corazón del proceso o la máquina. Un paso por la sucesión representa un ciclo de parte de la máquina, o de una serie en el proceso.
- **Parada de emergencia y supervisión de alarmas** –Esta etapa está siempre activa porque cuida de errores que podrían indicar una condición de alarma o requerir una parada de emergencia. Es común para esta etapa para reponer las etapas en el proceso principal o en otra parte, para inicializarlos después de una condición de error. .
- **Interfase de operador** –Esta es otra tarea que siempre debe estar activa y lista para responder a un operador. Permite que un operador se comunique para cambiar los modos, etc. independientemente del paso principal actual del proceso.

Aunque tengamos procesos separados, puede haber coordinación entre ellos. Por ejemplo, en una condición de error, la etapa de estados puede querer cambiar automáticamente la interfase de operador al modo de estados para mostrar la información de error como mostrado a la derecha. La etapa de supervisión podría configurar el bit para los estados y reponer (volver al valor original) el control de etapas y receta.

Cómo trabajan las instrucciones dentro de una etapa

Podemos pensar en estados o etapas simplemente dividiendo el programa ladder como está representado en la figura de abajo. Cada etapa contiene sólo los renglones necesarios para el estado correspondiente del proceso. La lógica para hacer una transición fuera de una etapa es hecha dentro de esa etapa. Es fácil escoger cuáles renglones estarán activos durante la energización del PLC usando un tipo "inicial" de etapa ISG.



La mayoría de las instrucciones trabajan tal como trabajan en ladder. Usted puede pensar en una etapa así como un programa miniatura ladder que está activo o inactivo.

Las bobinas de salidas – Como esperado, las bobinas en las etapas activas prenderán las salidas si la bobina tiene un flujo de corriente. Sin embargo, note lo siguiente:

- Las salidas trabajan como de costumbre, si es que cada salida (tal como "Y3") es usada en sólo una etapa. Las bobinas de salida se apagan automáticamente cuando la etapa es desactivada. Sin embargo, las instrucciones SET y RESET no dejan que la bobina cambie de estado para OFF.
- Una salida se puede referenciar en más de una etapa, pero sólo una de las etapas será activa cada vez.
- Si una bobina de salida es controlada por más de una etapa simultáneamente, la etapa activa más cercana al final del programa determina la posición final de la salida durante cada barrido. Por lo tanto, use la instrucción OROUT en vez de OUT cuando usted quiera que múltiples etapas tengan el control de una salida lógica OR.

Bobinas PD o One Shot – - tenga cuidado al usar una bobina PD (Diferencial) en una etapa. Recuerde que la entrada a la bobina debe hacer una transición de OFF para ON. Si la bobina está energizada ya en el primer barrido cuando la etapa se hace activa, la bobina PD no trabajará. Esto es porque no ocurrió la transición de OFF para ON.

Alternativa de bobina PD: Si hay una tarea que usted quiere de hacer sólo una vez (en 1 barrido), se puede colocar en una etapa que haga la transición a la próxima etapa en el mismo barrido.

Contador – Al usar un contador dentro de una etapa, la etapa debe estar activa por un barrido antes de que la entrada al contador haga una transición de OFF para ON. De otro modo, no hay una verdadera transición y el contador no contará.

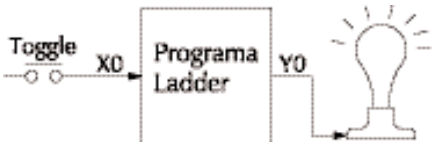
La instrucción contador normal tiene una restricción dentro de un programa de etapas: no puede ser repuesta desde otras etapas que usan la instrucción RST para el bit de contador. Sin embargo, el contador especial de etapa provee una solución (vea próximo párrafo).

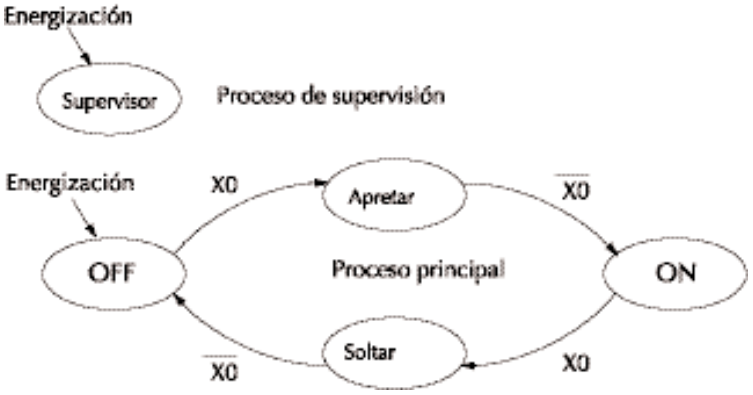
Contador de etapas – El Contador de etapa tiene el beneficio que el conteo puede estar puede reponerse globalmente de otras etapas usando la instrucción RST. Tiene una entrada de conteo, pero no una entrada reset. Esto es la única diferencia de un contador uniforme..

Tambor – Por favor considere que el secuenciador de tambor es su propio proceso y es un método diferente de programación que la programación de etapas. Si necesita usar un tambor con etapas, esté seguro de colocar la instrucción de tambor en una etapa ISG que esté siempre activa.

Usando una etapa como un proceso de supervisión

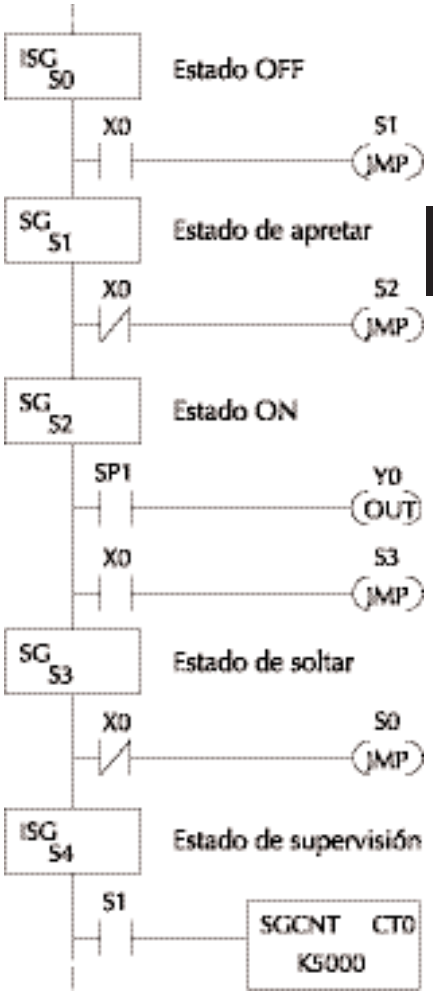
Usted puede recordar la lámpara en el ejemplo del controlador en este capítulo. Para propósitos de ilustración, suponga que queremos controlar la "productividad" del proceso de la lámpara, contando el número de ciclos que ocurren. Esta aplicación requerirá la adición de un contador sencillo, pero la decisión clave está en donde poner el contador.





Los estudiantes nuevos de programación de etapas típicamente tratarán de colocar el contador dentro de una de las etapas del proceso que se trata de controlar. El problema con este enfoque es que la etapa esta activa solamente parte del tiempo. Para que el contador cuente, la entrada de conteo hacer la transición de OFF para ON por lo menos un barrido después que su etapa se activa. Asegurar esto requiere una lógica extra que puede ser complicada.

En este caso, sólo necesitamos agregar otra etapa de supervisión como mostrado arriba, para "mirar" el proceso principal. El contador dentro de la etapa de supervisión usa el bit S1 de la etapa del proceso principal como su entrada de conteo. ¡*Los bits de etapas usados como un contacto nos deja controlar un proceso!*



Note que la etapa de supervisión y la etapa de estado OFF son las etapas iniciales. La etapa de supervisión permanece activa indefinidamente.

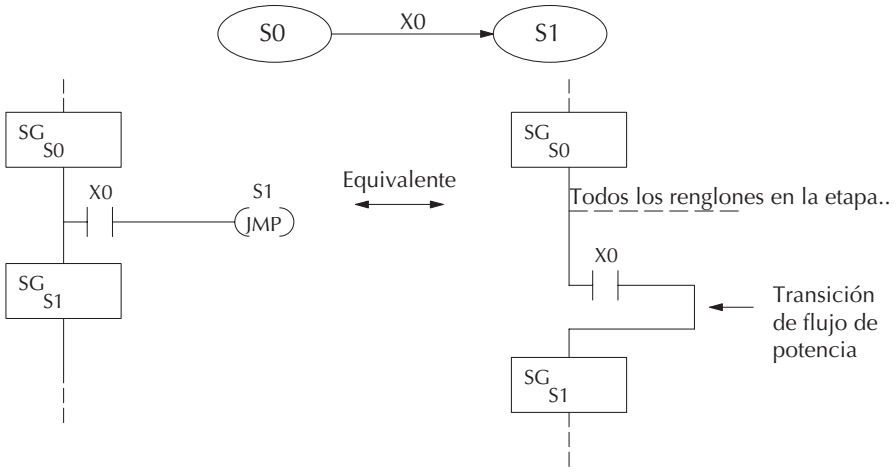
Contador de etapas

El contador en el ejemplo de encima es un contador de etapa especial. Note que no tiene una entrada RESET. El conteo es repuesto (reset) ejecutando la instrucción RESET, llamando al bit de estado del contador (CT0 en este caso). El contador de etapa tiene el beneficio que su conteo puede reponerse globalmente desde otras etapas. La instrucción contador normal no tiene esta capacidad de reset global. Puede usar también una instrucción contador normal dentro de una etapa... sin embargo, la entrada RESET del contador es la única manera de reponerlo.

La técnica de transición del flujo de potencia

Nuestra discusión de transiciones de estado ha mostrado cómo la instrucción de etapa JMP hace la etapa corriente inactiva y la próxima etapa (denominada en el JMP) activa. Como una manera alternativa de entrar esto en DirectSOFT32, usted puede usar el método del flujo de potencia para transiciones de etapa.

El requisito principal es que la etapa actual esté localizada directamente encima de la próxima (el salto a) etapa en el programa ladder. Este arreglo se muestra en el diagrama abajo, para las etapas S0 y S1, respectivamente.

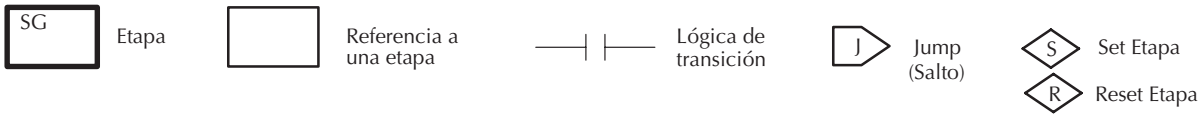


Recuerde que la instrucción JMP puede ocurrir dondequiera en la etapa actual y el resultado es el mismo. Sin embargo, las transiciones de flujo de potencia (mostrado arriba) deben ocurrir en el último renglón en una etapa. Todos los otros renglones en la etapa deben ser anteriores. El método de la transición de flujo de potencia es también factible en el programador portátil, simplemente siguiendo la condición de la transición con la instrucción de etapa para la próxima etapa.

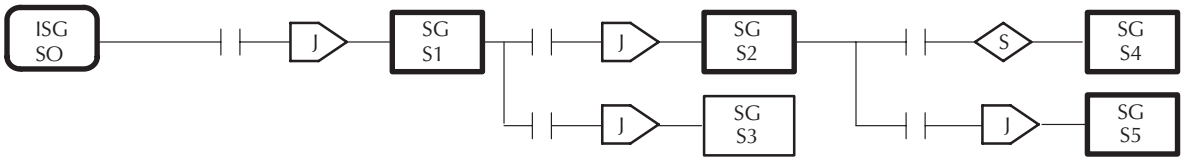
El método de la transición de flujo de potencia elimina una instrucción JMP, su única ventaja. Sin embargo, no es tan fácil de hacer los cambios del programa como usando la etapa JMP. Por lo tanto, recomendamos usar las transiciones JMP para la mayoría de los programadores.

La pantalla Stage View en *Direct*SOFT

La opción de tener una ventana con el diagrama de etapas (Etapa View) en *Direct*SOFT le permitirá ver el programa como un diagrama de flujo. La figura de abajo le muestra las convenciones de símbolos usada en los diagramas. Usted puede encontrar que el diagrama de etapas es útil como una herramienta para verificar que su programa de etapas ha reproducido fielmente la lógica del diagrama de transición de estado que usted



piensa ejecutar. El diagrama siguiente es un diagrama típico de etapas de un programa ladder que contiene etapas. Note la dirección de izquierda a derecha del diagrama de flujo.



Conceptos de procesamiento paralelo

Procesos paralelos

Previamente en este capítulo discutimos cómo un estado puede hacer una transición de un estado a otro, llamado una transición exclusiva. En otros casos, podemos necesitar bifurcar simultáneamente a dos o más procesos paralelos, como mostrado abajo. Es aceptable usar todas las instrucciones JMP como mostrado o podríamos usar una instrucción JMP y una instrucción SET de etapa (por lo menos una debe ser un JMP, para dejar S1). Recuerde que todas instrucciones en una etapa se ejecutan, aún cuando se hagan transiciones (el JMP no es un GOTO).

Note que si queremos que las etapas S2 y S4 se activen exactamente en el mismo barrido, ambos debe estar localizados abajo o encima de la etapa S1 en el programa ladder. ¡En general, bifurcar estados paralelos es fácil!

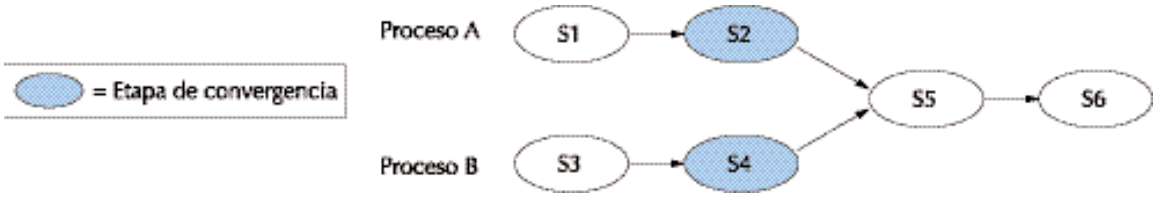
7

Procesos de convergencia

Ahora consideraremos el caso opuesto de bifurcar en paralelo, que es el proceso de convergencia. Esto significa simplemente que paramos de hacer múltiples etapas y seguimos haciendo uno a la vez. En la figura de abajo, el proceso A y el B convergen cuándo etapas S2 y S4 hacen una transición a S5 en algún momento. En este caso, S2 y S4 son las etapas de Convergencia.

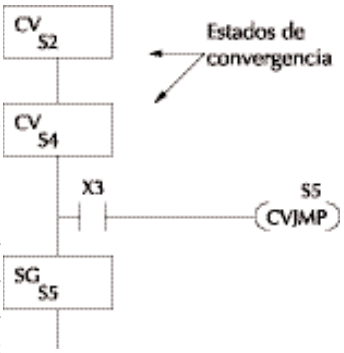
Etapas de convergencia (CV)

Aunque el principio de convergencia es suficientemente sencillo, trae una nueva complicación. Cuando el procesamiento paralelo se completa, los múltiples procesos casi nunca terminan al



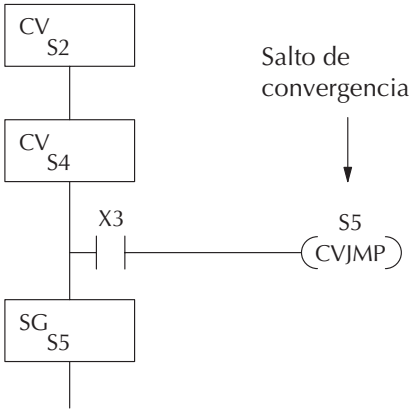
mismo tiempo. ¿En otras palabras, cómo podemos saber si una de las etapas S2 o S4 terminará por último? Esto es un punto importante, porque hay que decidir cómo a se hace la transición para la etapa S5.

La solución es coordinar la condición de la transición fuera de las etapas de convergencia Convergencia de etapas. Logramos esto con un tipo de etapa diseñado para este propósito: la etapa de Convergencia (tipo CV). En el ejemplo a la derecha, se requieren las etapas de convergencia S2 y S4 para agruparse juntas como mostrado. ¡NO **se permite ninguna lógica entre etapas CV!** La condición de transición (X3 en este caso) debe estar localizada **dentro de** la última etapa de convergencia. La condición de la transición sólo tiene flujo de corriente cuando todas las etapas de convergencia en el grupo están activas.



Salto de convergencia (CVJMP)

Recuerde que la última etapa de convergencia sólo tiene flujo de corriente cuando todas las etapas CV en el grupo están activas. Para complementar la etapa de convergencia, necesitamos una nueva instrucción de salto. La instrucción salto de convergencia (CVJMP) mostrado a la derecha hace la transición a la etapa S5 cuando la entrada X3 está activa (como se puede esperar), pero también desactiva automáticamente todas las etapas de convergencia en el grupo. Esto hace al CVJMP una instrucción muy poderosa. Note que esta instrucción puede sólo ser usada con etapas de convergencia.



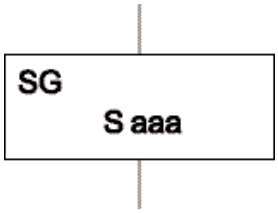
Reglas de uso de la etapa de convergencia

Lo siguiente hace un resumen de los requisitos en el uso de etapas de convergencia, inclusive algunas sugerencias para su aplicación efectiva:

- Una etapa de convergencia debe ser usada como la última etapa de un proceso que corre en paralelo a otro proceso o procesos. Una transición a la etapa de la convergencia significa que cierto proceso se ha acabado y representa un punto de espera hasta todos los otros procesos paralelos también terminen.
- El número máximo de etapas de convergencia que componen un grupo es 16. En otras palabras, hasta máximo de 16 etapas puede convergir en una etapa.
- Las etapas de convergencia del mismo grupo se deben colocar junto en el programa, conectado en el riel de poder sin cualquier otra lógica en el medio.
- Las etapas pueden ocurrir en cualquier orden dentro de un grupo de convergencia de arriba para abajo. No importa cuál etapa es la última a entrar el grupo, porque toda etapa de convergencia debe estar activa antes de la última etapa que tenga el flujo de poder.
- La última etapa de convergencia de un grupo puede tener lógica dentro de la etapa. Sin embargo, esta lógica no se ejecutará hasta que todas etapas de convergencia del grupo sean activadas.
- El salto de convergencia (CVJMP) es el método destinado a ser usado para hacer una transición del grupo de etapas de convergencia a la próxima etapa. El CVJMP repone todas etapas de convergencia del grupo, y activa la etapa denominada en el salto.
- La instrucción CVJMP debe sólo ser usada en una etapa de convergencia, y es inválida en etapas regulares o iniciales.
- Las etapas de convergencia o instrucciones de CVJMP no se pueden usar en rutinas de subprogramas o interrupción.

Instrucciones de RLL*PLUS*

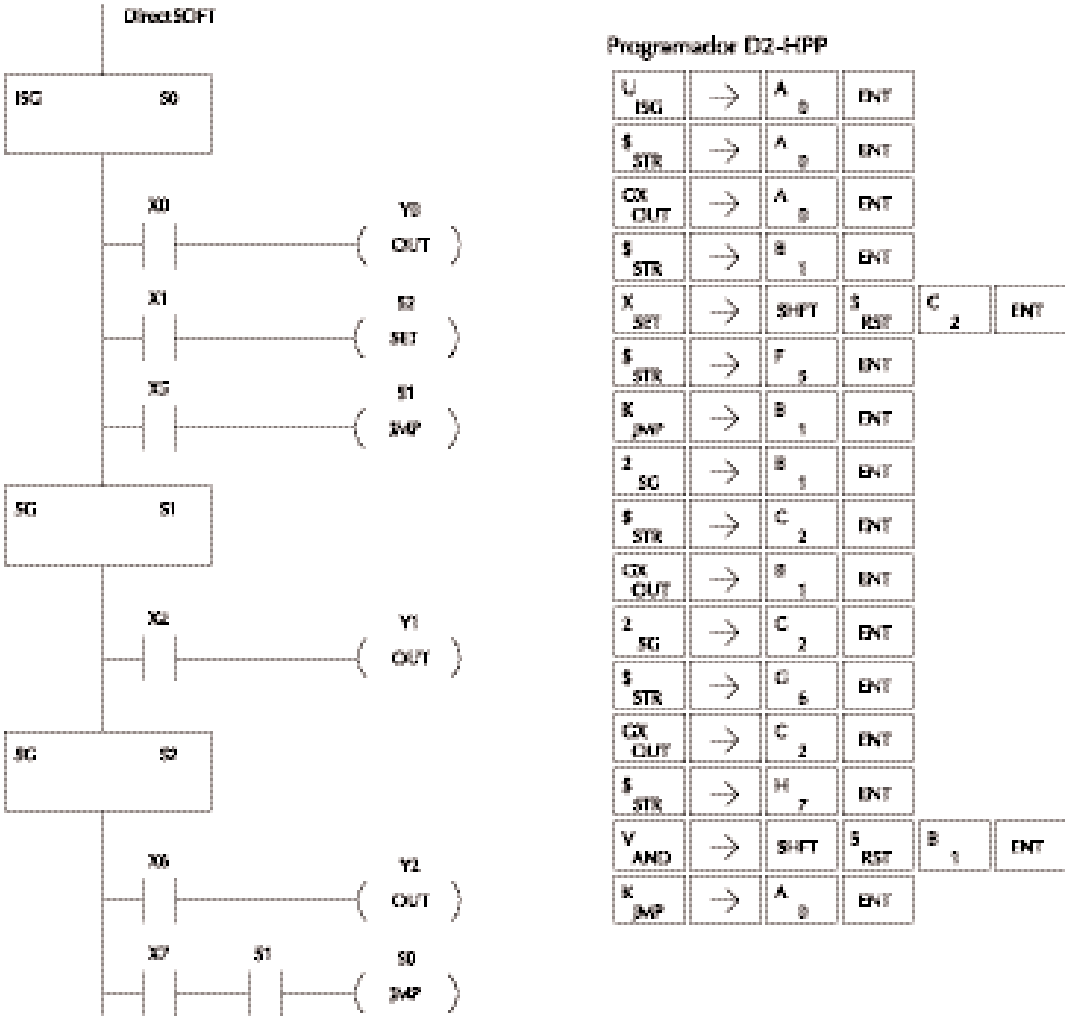
Etapa (o Stage) (SG)



Las instrucciones de etapa se usan para crear programas estructurados de RLL*PLUS*. Las etapas son los segmentos de programa que pueden ser activados por una lógica de transición, un salto o una etapa que se ejecuta de una etapa activa. Las etapas se desactivan un barrido después que se ejecuta la lógica de transición, la instrucción de etapa, de salto, o un RESET.

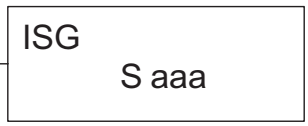
Tipo de datos del operando	Rango del DL06
	aaa
Etapa . . . . . S	0–1777

El ejemplo siguiente es un programa sencillo de RLL*PLUS*. Este programa utiliza instrucciones etapa inicial ISG, etapa SG y salto JMP para crear un programa estructurado.



Etapa inicial (ISG)

La instrucción Inicial de etapa se usa normalmente como el primer segmento de un programa RLL*PLUS*. Se permiten múltiples etapas Iniciales en un programa. Ellos serán activos cuando la CPU entra al modo RUN para tener un punto de partida en el programa.

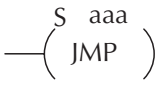


Tipo de datos del operando	Rango del DL06
	aaa
Etapa. . . . . S	0–1777

Las etapas iniciales son activadas también por la lógica de transición, un salto o una etapa fija ejecutados de una etapa activa.

El salto o JUMP (JMP)

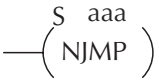
La instrucción de salto permite al programa hacer una transición de una etapa activa que contiene la instrucción del salto a otra etapa (especificada en la instrucción). El salto ocurre cuando la lógica de entrada es verdadera. La etapa activa que contiene el salto se desactivará 1 barrido mas tarde.



Tipo de datos del operando	Rango del DL06
	aaa
Etapa. . . . . S	0–1777

No Salto (NJMP)

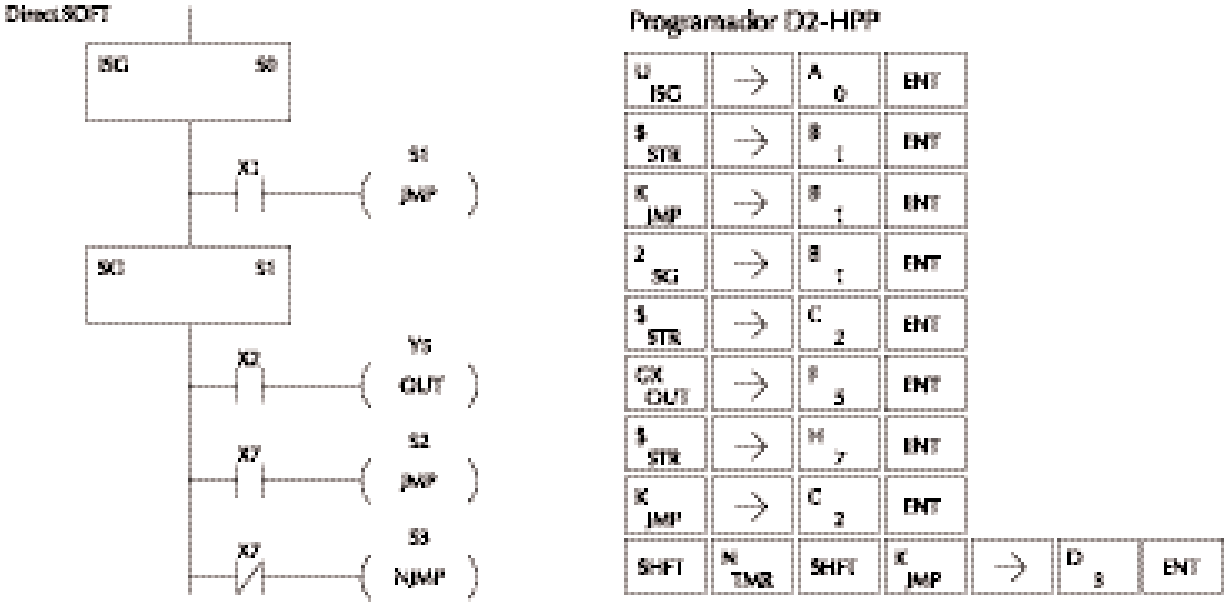
La instrucción de No Salto permite al programa hacer una transición de una etapa activa que contiene la instrucción de salto a otra que se especifica en la instrucción. El salto ocurrirá cuando la lógica de entrada está apagada. La etapa activa que contiene el No Salto se desactivará 1 barrido después que se ejecuta la instrucción de No Salto.



Tipo de datos del operando	DL06 Range
	aaa
Etapa. . . . . S	0–1777

En el ejemplo siguiente, la etapa ISG0 sólo estará activa cuando la ejecución del programa comienza.

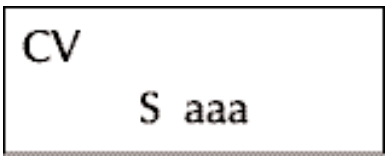
Cuándo X1 esté ON, la ejecución del programa saltará desde la etapa Inicial 0 a etapa 1.



7

Etapa de convergencia (CV) y el salto de convergencia (CVJMP)

La instrucción de etapa de convergencia se usa para agrupar ciertas etapas juntas definiéndolas como etapas de convergencia.



Cuándo todas las etapas de convergencia dentro de un grupo se hacen activas, será ejecutada la instrucción CVJMP (y cualquier lógica adicional en la etapa final de CV).

Todas las etapas CV anteriores debe estar activas antes que la lógica final de etapas CV se puedan ejecutar.

Todas las etapas de convergencia se desactivan un barrido después que se ejecuta la instrucción CVJMP.

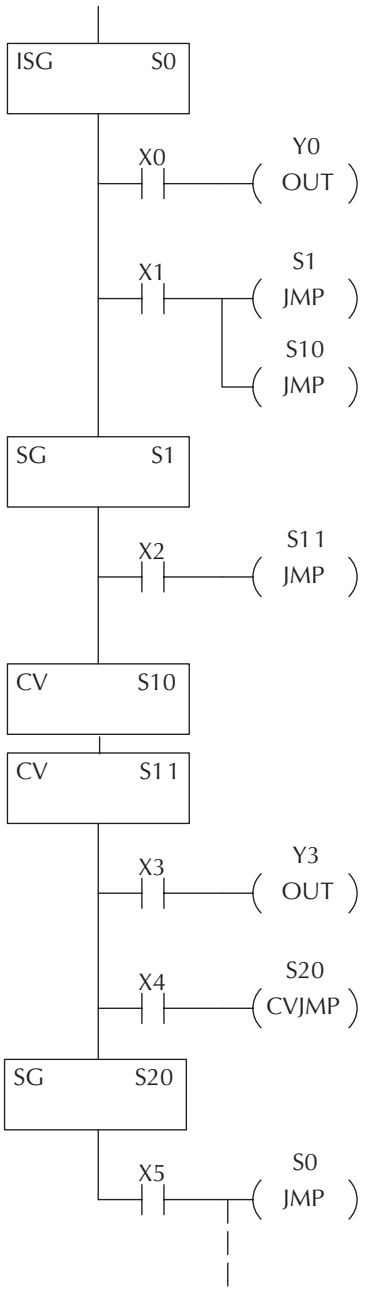
Sólo son permitidas instrucciones adicionales de lógica que siguen la última instrucción de etapa y antes de la instrucción CVJMP. Se permiten múltiples instrucciones CVJMP.

Las etapas de convergencia se deben programar en el cuerpo principal del programa de la aplicación. Esto significa que no pueden ser programadas en subrutinas o Subprogramas de interrupción.

Tipo de datos del operando	Rango del DL06
	aaa
Etapa. . . . . S	0–1777

En el ejemplo siguiente, cuando las etapas de convergencia S10 y S11 están activas al mismo tiempo, se ejecutará la instrucción CVJMP cuándo X4 esté ON. La instrucción CVJMP desactivará S10 y S11, y activará S20. Luego, si X5 está ON, la ejecución del programa saltará de vuelta a la etapa inicial, S0.

Direct SOFT



Programador D2-HPP

U ISG	→	A 0	ENT			
$ STR	→	A 0	ENT			
GX OUT	→	A 0	ENT			
$ STR	→	B 1	ENT			
K JMP	→	B 1	ENT			
K JMP	→	B 1	A 0	ENT		
2 SG	→	B 1	ENT			
$ STR	→	C 2	ENT			
K JMP	→	B 1	B 1	ENT		
SHFT	C 2	V AND	→	B 1	A 0	ENT
SHFT	C 2	V AND	→	B 1	B 1	ENT
$ STR	→	D 3	ENT			
GX OUT	→	D 3	ENT			
$ STR	→	E 4	ENT			
SHFT	C 2	V AND	SHFT	K JMP	→	C 2	A 0	ENT
2 SG	→	C 2	A 0	ENT		
$ STR	→	F 5	ENT			
K JMP	→	A 0	ENT			

Llamada de bloque (BCALL)

Las instrucciones BCALL se usan para activar un bloque de etapas. Las instrucciones Llamada de Bloque **BCALL**, Bloque **BKL** y Fin de Bloque **BEND** se deben usar juntas. La instrucción BCALL se usa para activar un bloque de etapa. Hay varias cosas que es necesario saber acerca de la instrucción BCALL.

- Usa números C - La instrucción BCALL aparece como una bobina de salida, pero no se refiere verdaderamente a un número de etapa como se podría pensar. En vez de eso, el bloque se identifica con un relevador de control (Caaa). Este relevador de control no se debe usar como una salida en ninguna otra parte del programa.
- Debe permanecer activa - La instrucción BCALL verdaderamente controla todas las etapas entre las instrucciones **BLK** y **BEND** aún después que las etapas dentro del bloque han comenzado a ejecutarse. La instrucción BCALL debe permanecer activa o todas las etapas en el bloque se apagarán automáticamente. Si la instrucción BCALL o la etapa que contiene la instrucción BCALL se desactiva, entonces las etapas en el bloque definido se apagarán automáticamente.
- Activa la primera etapa de bloque - Cuando se ejecuta la instrucción BCALL se activa automáticamente la primera etapa que sigue a las instrucciones BLK.

Tipo de datos del operando	Rango del DL06
	aaa
Relevador de control . . . . . S	0–1777

BLOCK (BLK)

La instrucción de bloque **BLK** es una etiqueta que marca el comienzo de un bloque de etapas que se pueden activar como un grupo. Una instrucción de etapa debe seguir inmediatamente la instrucción de Bloque BLK. No se permiten instrucciones iniciales de etapa en un bloque. El relevador de control (Caaa) especificado en la instrucción BLK no se debe usar como una salida en ningún otro lugar en el programa.

Tipo de datos del operando	Rango del DL06
	aaa
Relevador de control . . . . . S	0–1777

El fin de bloque (BEND)

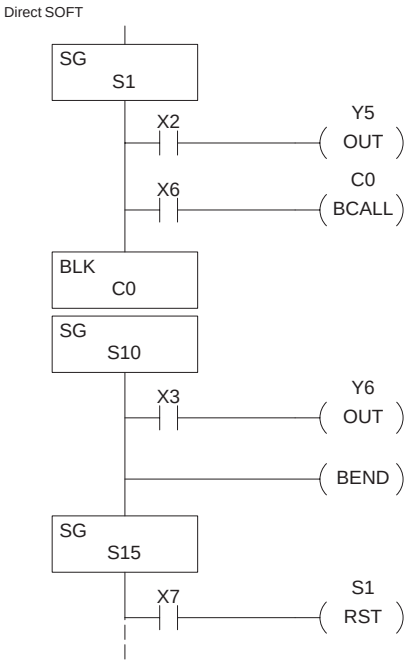
La instrucción Fin de bloque BEND es una etiqueta usada con la instrucción de bloque. Marca el fin de un bloque de etapas. No hay operando con esta instrucción. Sólo se permite una instrucción Fin de Bloque **BEND** por bloque.

En este ejemplo, la llamada de bloque se ejecuta cuando la etapa 1 es activa y X6 está ON.

La llamada de bloque activa automáticamente la etapa S10, que sigue inmediatamente la instrucción de bloque.

Esto permite que las etapas entre S10 y la instrucción Fin de Bloque operen como programado. Si se apaga la instrucción BCALL, o si la etapa que contiene la instrucción BCALL se apaga, entonces todas las etapas entre las instrucciones BLK y BEND son apagadas automáticamente.

Si examina S15, notará que X7 podría desactivar la etapa S1, que incapacitaría la instrucción BCALL, así desactivando todas las etapas dentro del bloque.



Programador D2-HPP

SG	→	S(SG)	1	ENT					
STR	→	X(IN)	2	ENT					
OUT	→	Y(OUT)	5	ENT					
STR	→	X(IN)	6	ENT					
SHFT	B	C	A	L	L	→	C(CR)	0	ENT
SHFT	B	L	K	→	C(CR)	0	ENT		
SG	→	S(SG)	1	0	ENT				
STR	→	X(IN)	3	ENT					
OUT	→	Y(OUT)	6	ENT					
SHFT	B	E	N	D	ENT				
SG	→	S(SG)	1	5	ENT				
STR	→	X(IN)	7	ENT					
RST	→	S(SG)	1	ENT					

Preguntas y respuestas acerca de la programación por etapas

Incluimos la siguientes preguntas comúnmente hechas en la programación de etapas como una ayuda a nuevos estudiantes. Todos los asuntos en las preguntas se cubren en más detalle en este capítulo.

¿ Qué hace la programación por etapas que no se pueda hacer con programas regulares?

Respuesta: Las etapas le permiten identificar todos los estados de su proceso antes que comience a programar. Este enfoque es más organizado, porque usted divide un programa en secciones. Como etapas, éstas secciones de programas son activas sólo cuando se necesitan por el proceso. La mayoría de los procesos se pueden organizar en una sucesión de etapas, conectado por transiciones basadas en eventos.

¿Qué son los bits de etapa?

Respuesta: Un bit de etapa es un solo bit en el registro de imagen de la CPU, representando la posición activa o inactiva de la etapa en el tiempo real. Por ejemplo, el bit de la etapa 0 es referenciado como "S0". Si S0 = 0, entonces los renglones en la etapa 0 son evitados (no ejecutado) en cada barrido de la CPU. Si S0 = 1, entonces los renglones en la etapa 0 son ejecutados en cada barrido de la CPU. Los bits de etapa, cuando se usan como contactos, permiten que una parte del programa controle otra parte detectando estados de etapa activa o inactiva.

¿ Cómo una etapa llega a ser activa?

Respuesta: Hay tres maneras:

- Si la etapa es una etapa inicial (ISG), es automáticamente activo en la energización.
- Otra etapa puede ejecutar una instrucción etapa JMP nombrabdo esta etapa, que la hace activa en la próxima ocurrencia en el programa.
- Un renglón de programa puede ejecutar una instrucción Fija del Bit de la etapa (tal como Fijo S0).

¿ Cómo una etapa llega a ser inactiva?

Respuesta: Hay tres maneras:

- las etapas (SG) son automáticamente inactivas en la energización.
- Una etapa puede ejecutar una instrucción de etapa JMP, colocando el bit de etapa a 0.
- Un renglón en el programa puede ejecutar la instrucción RESET Bit de etapa (tal como Reset S0).

¿ Como es la técnica de flujo de poder en transiciones de etapa?

Respuesta: El método del flujo de poder de conectar etapas adyacentes (directamente encima o abajo en el programa) es realmente lo mismo que la instrucción de salto de etapa JMP ejecutada en la etapa arriba, nombrando la etapa abajo. Las transiciones de flujo de poder son más difíciles de modificar en *DirectSOFT* y destacamos esto separadamente de las dos preguntas anteriores.

¿ Puedo tener una etapa que es activa solamente por un barrido?

Respuesta: Sí, pero esto no es un uso normal para una etapa. En vez de eso, haga un renglón activo en 1 barrido por inclusive una instrucción del Salto de etapa en el fin del renglón. Luego el codigo ejecutará en el ultimo barrido antes que la etapa salte a un nuevo.

¿ No es una etapa JMP igual que una instrucción regular GOTO usada en DirectSoft?

Respuesta: No, es muy diferente. Una instrucción GOTO manda la ejecución del programa inmediatamente a la localización del código denominada por el GOTO. Una etapa JMP repone simplemente el bit de etapa de la etapa actual, al preparar el camino bit de etapa denominó en la instrucción de JMP. Los bits de la etapa son 0 o 1, determinando el inactivo/la posición activa de las etapas correspondientes. Una etapa JMP tiene los resultados siguientes:

- Cuando el JMP se ejecuta, el resto del renglonesactual de la etapa se ejecuta, aunque ellos residan pasado (abajo) la instrucción de JMP. En el siguiente barrido, eso prepara no es ejecutado, porque es inactivo.
- La etapa denominada en la instrucción de la etapa JMP se ejecutará sobre su próxima ocurrencia. Si localizó el pasado (abajo) la etapa actual, se ejecutará en el mismo barrido. Si localizó antes (arriba) la etapa actual, se ejecutará en el siguiente barrido.

¿Cómo puedo saber cuando se usa la instrucción JMP en vez de la instrucción SET o RESET un bit de etapa?

Respuesta: Estas instrucciones se usan según la topología de diagrama de estado que usted ha derivado:

- Use una instrucción de etapa JMP para una transición de estado... moverse de un estado a otro.
- Use una instrucción SET bit de etapa cuando el estado actual crea una sucesión paralela nueva de estado o etapa, o cuando un estado de supervisión comienza una sucesión del estado bajo su comando.
- Use la instrucción RESET bit cuando el estado actual es el último estado en una sucesión y su tarea es completa o cuando un estado de supervisión finaliza una sucesión del estado bajo su comando.

¿Qué es una etapa inicial, y cuándo se puede usar?

Respuesta: Una etapa (ISG) inicial es automáticamente activa en la energización. Después, trabaja así como cualquier otra etapa. Usted puede tener el múltiplo las etapas iniciales, si requirió. Use una etapa inicial para escalera que siempre debe ser activa, o cuando un punto de partida

¿Puedo tener renglones de programa fuera de las etapas, de modo que estén siempre ON?

Respuesta: Es posible, pero no es buena práctica de diseño de software. Coloque el codigo que siempre debe estar activo en una etapa inicial y no resete esa etapa ni use una instrucción de etapa JMP dentro de el. Puede comenzar otras sucesiones de etapa en el tiempo apropiado colocando ON el correspondiente bit (s) de etapa.

¿ Puedo tener más que una etapa activa a la vez?

Respuesta: Sí, y esto es una ocurrencia normal en muchos programas. Sin embargo, es importante organizar su aplicación en procesos separados, cada un compuesto de etapas. Y un diseño bueno de proceso será en su mayor parte secuencial, con sólo una etapa ON a la vez. Sin embargo, todos los procesos en el programa pueden estar activos simultáneamente.