Manual de referencia del lenguaje

Ejemplos

Las aventuras de Docteur R.

www.irai.com

1. Elementos comunes

Este capítulo detalla los elementos comunes a todos los lenguajes utilizables en AUTOMGEN.

El logo  identifica las novedades utilizables en la versión 7 de AUTOMGEN.

1.1. Variables

Existen los siguientes tipos de variables:

⇨ tipo booleano: la variable puede adoptar el valor verdadero (1) o falso (0).

⇨ tipo numérico: la variable puede adoptar un valor numérico; existen diferentes subtipos: variables 16 bits, 32 bits y coma flotante.

⇨ tipo temporización: tipo estructurado; es una combinación del booleano y el numérico.

A partir de la versión 6 la sintaxis de los nombres de variables puede ser la de AUTOMGEN o la de la norma CEI 1131-3.

1.1.1. Variables booleanas

La tabla siguiente ofrece la lista exhaustiva de variables booleanas utilizables.

Tipo	Sintaxis AUTOMGEN	Sintaxis CEI 1131-3	Comentario
Entradas	I0 a I9999	%I0 a %I9999	Puede corresponder o no a entradas físicas (depende de la configuración de las E/S del destino).
Salidas	O0 a O9999	%Q0 a %Q9999	Puede corresponder o no a entradas físicas (depende de la configuración de las E/S del destino).
Bits Sistema	U0 a U99	%M0 a %M99	Ver el manual sobre el entorno para el detalle de los bis Sistema.
Bits Usuario	U100 a U9999	%M100 a %M9999	Bits internos de uso general.

Etapas Grafcet	X0 a X9999	%X0 a %X9999	Bits de etapas Grafcet.
Bits de palabras	M0#0 a M9999#15	%MW0:X0 a %MW9999:X15	Bits de palabras: el número del bit se expresa en decimal y está comprendido entre 0 (bit de peso débil) y 15 (bit de peso fuerte).

1.1.2. Variables numéricas

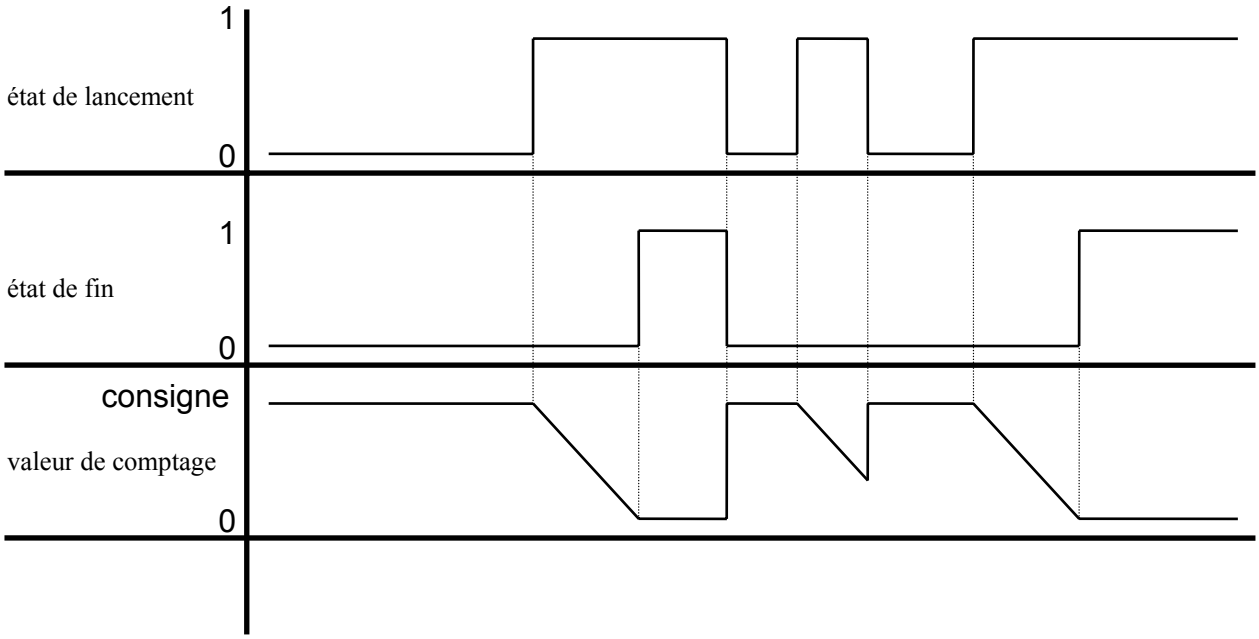
La tabla siguiente ofrece la lista exhaustiva de variables numéricas.

Tipo	Sintaxis AUTOMGEN	Sintaxis *CEI 1131-3*	Comentario
Contadores	C0 a C9999	%C0 a %C9999	Contador de 16 bits; puede inicializarse, incrementarse, decrementarse y testearse con los lenguajes booleanos sin utilizar el lenguaje literal.
Palabras Sistema	M0 a M199	%MW0 a %MW199	Ver el manual sobre el entorno para el detalle de las palabras Sistema.
Palabras Usuario	M200 a M9999	%MW200 a %MW9999	Palabra de 16 bits de uso general.
Largos	L100 a L4998	%MD100 a %MD4998	Valor entero sobre 32 bits.
Flotantes	F100 a F4998	%MF100 a %MF4998	Valor real sobre 32 bits (formato IEEE).

1.1.3. Temporizaciones

La temporización es un tipo compuesto que agrupa dos variables booleanas (estado de lanzamiento, estado de fin) y dos variables numéricas sobre 32 bits (la consigna y el contador).

El esquema siguiente muestra el cronograma de funcionamiento de una temporización:



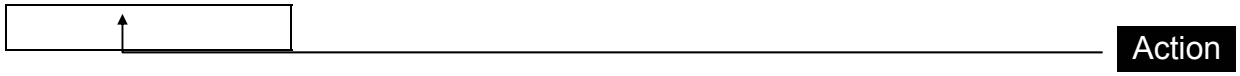
El valor de consigna de una temporización está comprendido entre 0 ms y 4294967295 ms (vale decir, poco más de 49 días)

La consigna de la temporización puede modificarse por programa; ver el capítulo 1.10.3. El lenguaje literal bajo nivel (instrucción STA).
El contador de la temporización puede leerse por programa; ver el capítulo 1.10.3. El lenguaje literal bajo nivel (instrucción LDA).

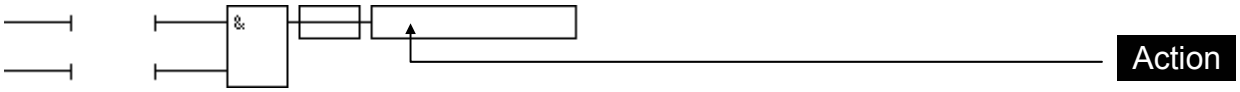
1.2. Acciones

Las acciones se utilizan en:

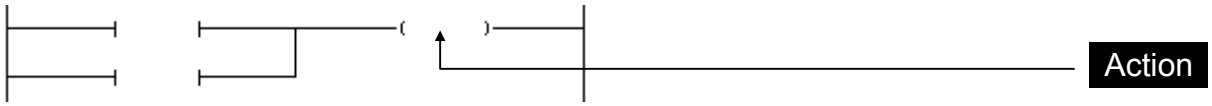
⇨ los rectángulos de acción del lenguaje Grafcet,



⇨ los rectángulos de acción del lenguaje logigrama,



⇨ las bobinas del lenguaje ladder.



1.2.1. Asignación de una variable booleana

La sintaxis de la acción « Asignación » es:

«variable booleana»

Funcionamiento:

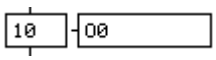
⇨ si el comando del rectángulo de acción o de la bobina está en estado verdadero, la variable se pone en 1 (estado verdadero),

⇨ si el comando del rectángulo de acción o de la bobina está en estado falso, la variable se pone en 0 (estado falso).

Tabla de verdad:

Comando	Estado de la variable (resultado)
0	0
1	1

Ejemplo:



Si la etapa 10 está activa, O0 adopta el valor 1; si no, O0 adopta el valor 0.

Es posible utilizar varias acciones « Asignación » para una misma variable dentro de un programa. En este caso, los diferentes comandos se combinan en « O » lógico.

Ejemplo:

Estado de X10	Estado de X50	Estado de O5
0	0	0
1	0	1
0	1	1
1	1	1

Asignación complementada de una variable booleana

La sintaxis de la acción « Asignación complementada » es:

«N variable booleana»

Funcionamiento:

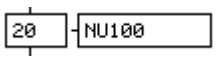
⇨ si el comando del rectángulo de acción o de la bobina está en estado verdadero, la variable se pone en 0 (estado falso),

⇨ si el comando del rectángulo de acción o de la bobina está en estado falso, la variable se pone en 1 (estado verdadero).

Tabla de verdad:

Comando	Estado de la variable (resultado)
0	1
1	0

Ejemplo:



Si la etapa 20 está activa, U100 adopta el valor 0; si no, U100 adopta el valor 1.

Es posible utilizar varias acciones « Asignación complementada» para una misma variable dentro de un programa. En este caso, los diferentes comandos se combinan en « O » lógico.

Ejemplo:

Estado de X100	Estado de X110	Estado de O20
0	0	1
1	0	0
0	1	0
1	1	0

1.2.2. Puesta en uno de una variable booleana

La sintaxis de la acción « Puesta en uno » es:

«S variable booleana»

Funcionamiento:

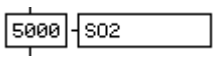
⇨ si el comando del rectángulo de acción o de la bobina está en estado verdadero, la variable se pone en 1 (estado verdadero),

⇨ si el comando del rectángulo de acción o de la bobina está en estado falso, el estado de la variable no se modifica.

Tabla de verdad:

Comando	Estado de la variable (resultado)
0	no cambia
1	1

Ejemplo:



Si la etapa 5000 está activa, O2 adopta el valor 1; si no, O2 conserva su estado.

1.2.3. Puesta en cero de una variable booleana

La sintaxis de la acción « Puesta en cero » es:

«R variable booleana»

Funcionamiento:

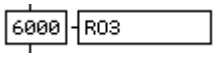
⇨ si el comando del rectángulo de acción o de la bobina está en estado verdadero, la variable se pone en 0 (estado falso),

⇨ si el comando del rectángulo de acción o de la bobina está en estado falso, el estado de la variable no se modifica.

Tabla de verdad:

Comando	Estado de la variable (resultado)
0	no cambia
1	0

Ejemplo:



Si la etapa 6000 está activa, O3 adopta el valor 0; si no, O3 conserva su estado.

1.2.4. Inversión de una variable booleana

La sintaxis de la acción « Inversión » es:

«I variable booleana»

Funcionamiento:

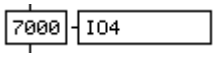
⇨ si el comando del rectángulo de acción o de la bobina está en estado verdadero, el estado de la variable se invierte a cada ciclo de ejecución,

⇨ si el comando del rectángulo de acción o de la bobina está en estado falso, el estado de la variable no se modifica.

Tabla de verdad:

Comando	Estado de la variable (resultado)
0	no cambia
1	invertido

Ejemplo:



Si la etapa 7000 está activa, el estado de O4 se invierte; si no, O4 conserva su estado.

1.2.5. Puesta en cero de un contador, una palabra o un largo

La sintaxis de la acción « Puesta en cero de un contador, una palabra o un largo» es:

«R contador o palabra»

Funcionamiento:

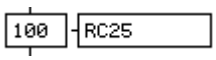
⇨ si el comando del rectángulo de acción o de la bobina está en estado verdadero, el contador, la palabra o el largo se pone en cero,

⇨ si el comando del rectángulo de acción o de la bobina está en estado falso, el valor del contador, de la palabra o del largo no se modifica.

Tabla de verdad:

Comando	Valor del contador, de la palabra o del largo (resultado)
0	No cambia
1	0

Ejemplo:



Si la etapa 100 está activa, el contador 25 se pone en cero; si no, C25 conserva su valor.

1.2.6. Incremento de un contador, una palabra o un largo

La sintaxis de la acción « Incremento de un contador » es:

«+ contador, palabra o largo»

Funcionamiento:

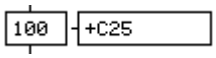
⇨ si el comando del rectángulo de acción o de la bobina está en estado verdadero, el contador, la palabra o el largo se incrementa a cada ciclo de ejecución,

⇨ si el comando del rectángulo de acción o de la bobina está en estado falso, el valor del contador no se modifica.

Tabla de verdad:

Comando	Valor del contador, de la palabra o del largo (resultado)
0	No cambia
1	valor actual +1

Ejemplo:



Si la etapa 100 está activa, el contador 25 se incrementa; si no, C25 conserva su valor.

1.2.7. Decremento de un contador, una palabra o un largo

La sintaxis de la acción « Decremento de un contador » es:

«- contador, palabra o largo»

Funcionamiento:

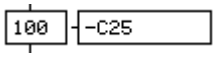
⇨ si el comando del rectángulo de acción o de la bobina está en estado verdadero, el contador, la palabra o el largo se decrementa a cada ciclo de ejecución,

⇨ si el comando del rectángulo de acción o de la bobina está en estado falso, el valor del contador no se modifica.

Tabla de verdad:

Comando	Valor del contador, de la palabra o del largo (resultado)
0	no cambia
1	valor actual –1

Ejemplo:



Si la etapa 100 está activa, el contador 25 se decrementa; si no, C25 conserva su valor.

1.2.8. Temporizaciones

Las temporizaciones se consideran como variables booleanas y pueden utilizarse con las acciones « Asignación », « Asignación complementada », « Puesta en uno », « Puesta en cero » e « Inversión ». La consigna de la temporización puede escribirse a continuación de la acción. La sintaxis es:

« temporización(duración) »

La duración se expresa en forma predeterminada en décimas de segundo. El carácter « S » ubicado al final de la duración indica que se expresa en segundos.

Ejemplos:

La etapa 10 lanza una temporización de 2 segundos que permanecerá activa mientras la etapa permanezca activa. La etapa 20 arma una temporización de 6 segundos que permanecerá activa aunque la etapa 20 esté desactivada.

Una misma temporización puede utilizarse en varios sitios con una misma consigna en momentos diferentes. En este caso, la consigna de la temporización debe indicarse una sola vez.

Observación: existen otras sintaxis para las temporizaciones. Ver el capítulo 1.3.3. Temporizaciones

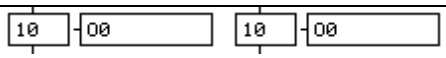
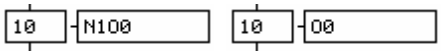
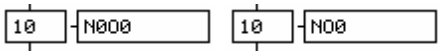
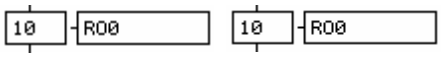
1.2.9. Interferencias entre las acciones

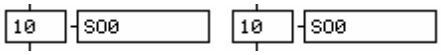
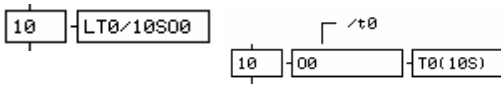
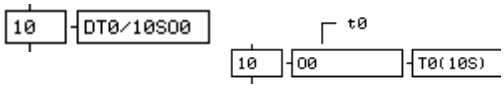
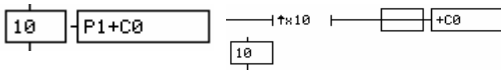
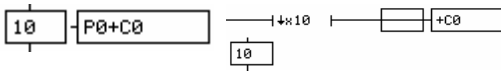
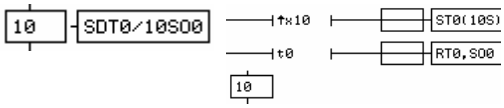
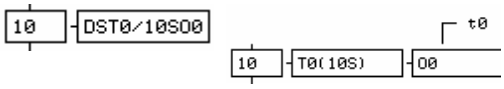
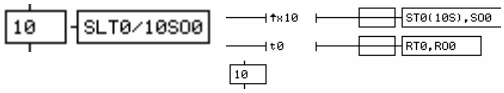
Ciertos tipos de acción no pueden utilizarse simultáneamente con una variable. La siguiente tabla resume las combinaciones posibles.

	Asignación	**Asignación complementada**	**Puesta en uno**	**Puesta en cero**	**Inversión**
Asignación	SÍ	NO	NO	NO	NO
Asignación complementada	NO	SÍ	NO	NO	NO
Puesta en uno	NO	NO	SÍ	SÍ	SÍ
Puesta en cero	NO	NO	SÍ	SÍ	SÍ
Inversión	NO	NO	SÍ	SÍ	SÍ

1.2.10. Acciones de la norma CEI 1131-3

La siguiente tabla ofrece la lista de acciones de la norma CEI 1131-3 utilizables en AUTOMGEN V>=6 con respecto a la sintaxis estándar de AUTOMGEN. V5.

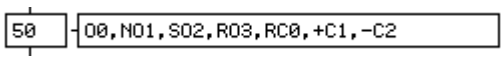
Nombre	*Sintaxis AUTOMGEN V>=6*	*Sintaxis AUTOMGEN V5*	*Ejemplo AUTOMGEN V>=6*	*Ejemplo de equivalente AUTOMGEN V5*
No memorizado	Ninguno	Ninguno		
No memorizado	N1	Ninguno		
No memorizado complementado	N0	N		
Puesta en cero	R	R		

Puesta en 1	S	S	
Limitado en el tiempo	LTn/duración	Inexistente	
Temporizado	DTn/ duración	Inexistente	
Impulsión sobre frente ascendente	P1	Inexistente	
Impulsión sobre frente descendente	P0	Inexistente	
Memorizado y temporizado	SDTn/ duración	Inexistente	
Temporizado y memorizado	DSTn/ duración	Inexistente	
Memorizado y limitado en el tiempo	SLTn/ duración	Inexistente	

1.2.11. Acciones múltiples

Dentro de un mismo rectángulo de acción o de una bobina, es posible escribir varias acciones separándolas con el carácter « , » (coma).

Ejemplo:



Es posible yuxtaponer varios rectángulos de acción (Grafcet y logigrama) o bobinas (ladder). Consultar los capítulos correspondientes a estos lenguajes para más detalles.

1.2.12. Código literal

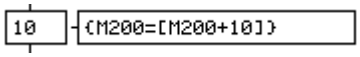
En un rectángulo de acción o una bobina es posible insertar código literal.

La sintaxis es:

« { código literal } »

Es posible escribir entre las llaves varias líneas de lenguaje literal. También aquí el separador es el carácter « , » (coma).

Ejemplo:

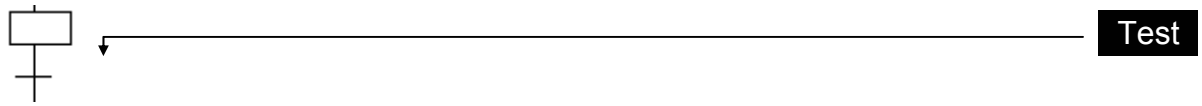


Consultar los capítulos « Lenguaje literal bajo nivel », « Lenguaje literal extendido » y « Lenguaje literal ST » para más detalles.

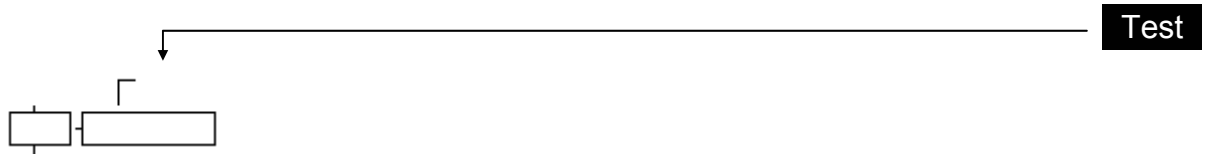
1.3. Tests

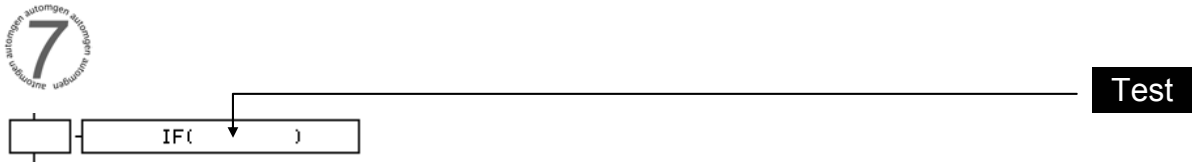
Los tests se utilizan en:

⇨ transiciones del lenguaje Grafcet,

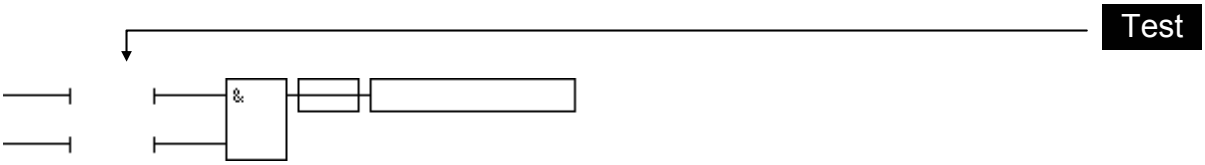


⇨ condiciones sobre acción del lenguaje Grafcet,

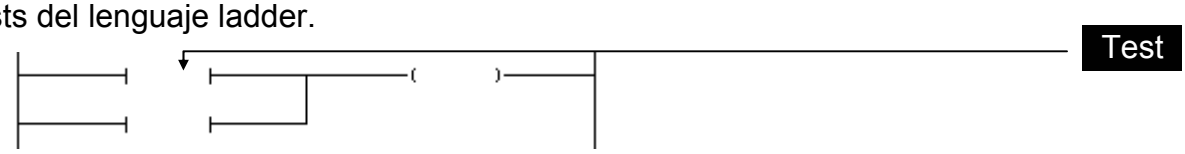




⇨ tests del lenguaje logigrama,



⇨ tests del lenguaje ladder.



1.3.1. Forma general

Un test es una ecuación booleana compuesta por una o por n variables separadas por los operadores « + » (o) o « . » (y).

Ejemplo de test:

```
i0 (test entrada 0)
i0+i2 (test entrada 0 « o » entrada 2)
i10.i11 (test entrada 10 « y » entrada 11)
```

1.3.2. Modificador de test

Si se especifica sólo el nombre de una variable, el test predeterminado es « si igual a uno» (si verdadero). Es posible utilizar modificadores para testear el estado complementado, el frente ascendente y el frente descendente:

⇨ el carácter « / » ubicado delante de una variable testea el estado complementado,

⇨ el carácter « u » o el carácter « ↑* » ubicado delante de una variable testea el frente ascendente,

⇨ el carácter « d » o el carácter « ↓** » ubicado delante de una variable testea el frente descendiente.

Los modificadores de tests pueden aplicarse a una variable o a una expresión entre paréntesis.

Ejemplos:

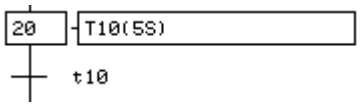
```
↑ i0
/i1
/(i2+i3)
↓(i2+(i4./i5))
```

* Para obtener este carácter durante la edición de un test presione la tecla [↑].

** Para obtener este carácter durante la edición de un test presione la tecla [↓].

1.3.3. Temporizaciones

Hay cuatro sintaxis disponibles para las temporizaciones.
En la primera, se activa la temporización en la acción y se menciona simplemente la variable temporización en un test para verificar el estado de fin:



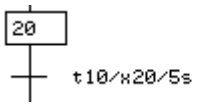
En las otras, todo se escribe en el test. La forma general es:
« temporización / variable de lanzamiento / duración »
o
« duración / variable de lanzamiento / temporización »
o

« duración / variable de lanzamiento »
En este caso, se atribuye una temporización automáticamente. La zona de atribución es la de los símbolos automáticos; ver el capítulo 1.4.2. Símbolos automáticos.

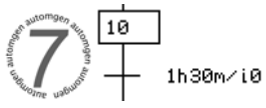
La duración se expresa en forma predeterminada en décimas de segundo.


La duración puede expresarse en días, horas, minutos, segundos y milisegundos con los operadores « d », « h », « m », « s » y « ms ». Por ejemplo: 1d30s = 1 día y 30 segundos.

Ejemplo con la segunda sintaxis:



Ejemplo con la sintaxis normalizada:



1.3.4. Prioridad de los operadores booleanos

El operador booleano « . » (Y) tiene una prioridad predeterminada mayor que el operador « + » (O). Es posible utilizar paréntesis para definir otra prioridad.

Ejemplos:

```
i0.(i1+i2)
((i0+i1).i2)+i5
```

1.3.5. Test siempre verdadero

La sintaxis del test siempre verdadero es:

« » (ninguno) o « =1 »

1.3.6. Test sobre variable numérica

Los tests sobre variable numérica deben utilizar la siguiente sintaxis:

« variable numérica » « tipo de test » « constante o variable numérica »

El tipo de test puede ser:

⇨ « = » igual,

⇨ « ! » o « <> » diferente,

⇨ « < » menor (sin signo),

⇨ « > » mayor (sin signo),

⇨ « << » menor (con signo),

⇨ « >> » mayor (con signo),

⇨ « <= » menor o igual (sin signo),

⇨ « >= » mayor o igual (sin signo),

⇨ « <<= » menor o igual (con signo),

⇨ « >>= » mayor o igual (con signo).

Un flotante puede compararse sólo con otro flotante o con una constante real.

Un largo puede compararse sólo con otro largo o con una constante larga.

Una palabra o un contador puede compararse sólo con una palabra, un contador o una constante 16 bits.

Las constantes reales deben estar seguidas del carácter « R ».

Las constantes largas (32 bits) deben estar seguidas del carácter « L ».

Les constantes enteras 16 o 32 bits están predeterminadas en decimal. Pueden escribirse en hexadecimal (sufijo « $ » o « 16# ») o en binario (sufijo « % » o « 2# »).

Los tests sobre variables numéricas se utilizan en las ecuaciones como los tests sobre variables booleanas. Pueden utilizarse con los modificadores de test siempre que estén encerrados por paréntesis.

Ejemplos:

```
m200=100
%mw1000=16#abcd
c10>20.c10<100
f200=f201
m200=m203
%md100=%md102
f200=3.14r
l200=$12345678L
m200<<-100
m200>>1000
%mw500<=12
/(m200=4)
↓(m200=100)
/(l200=100000+l200=-100000)
```

1.3.7. Transiciones en varias líneas

El texto de las transiciones puede ocupar varias líneas. El fin de una línea de transición debe ser indefectiblemente un operador « . » o « + ». Las combinaciones de teclas [CTRL] + [↓] y [CTRL] + [↑] permiten desplazar el cursor de una línea a otra.

1.4. Utilización de símbolos

Los símbolos permiten asociar un texto a una variable.
Los símbolos pueden utilizarse con todos los lenguajes.
Un símbolo debe asociarse a una y sólo una variable.

1.4.1. Sintaxis de los símbolos

Los símbolos están compuestos por:

- ⇨ un carácter « _ » opcional (subrayado, generalmente asociado a la tecla [8] en los teclados) que marca el principio del símbolo,
- ⇨ el nombre del símbolo,
- ⇨ un carácter « _ » opcional (subrayado) que marca el fin del símbolo.

Los caracteres « _ » que encierran los nombres de los símbolos son opcionales. Deben utilizarse si el símbolo empieza con una cifra o un operador (+,-, etc…).

1.4.2. Símbolos automáticos

A veces es incómodo tener que definir la atribución entre cada símbolo y una variable, especialmente si la atribución precisa de un número de variable tiene poca importancia. Los símbolos automáticos son una solución a este problema, ya que confían al compilador la tarea de generar automáticamente la atribución de un símbolo a un número de variable. El tipo de variable a utilizar está dado por el nombre del símbolo.

1.4.3. Sintaxis de los símbolos automáticos

La sintaxis de los símbolos automáticos es la siguiente:

```
_« nombre del símbolo » %« tipo de variable »_
```

El « tipo de variable » puede ser:

I , O o Q, U o M, T, C, M o MW, L o MD, F o MF.

Es posible reservar varias variables para un símbolo. Esto es útil para definir tablas. En este caso la sintaxis es:

```
_« nombre del símbolo » %« tipo de variable »« longitud »_
```

La « longitud » representa el número de variables a reservar.

1.4.4. ¿Cómo gestiona el compilador los símbolos automáticos?

Para compilar una aplicación, el compilador borra todos los símbolos automáticos que se encuentran en el archivo « .SYM » de la aplicación. Cada vez que encuentra un símbolo automático, crea una atribución única para ese símbolo en función del tipo de variable especificado en el nombre del símbolo. El símbolo generado se escribe en el archivo « .SYM ». Si un mismo símbolo automático aparece varias veces en una aplicación, hará referencia a la misma variable.

1.4.5. Zona de atribución de las variables

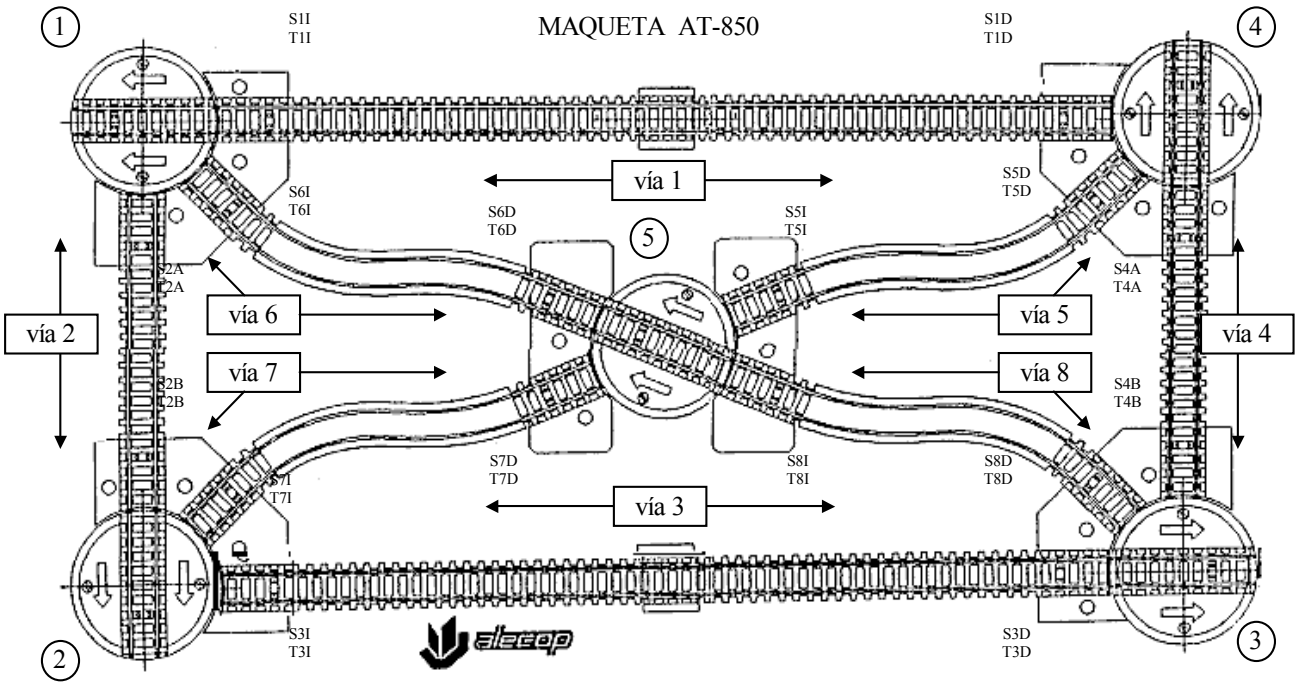
Cada tipo de variable tiene una zona de atribución predeterminada:

Tipo	Principio	Fin
I o %I	0	9999
O o %Q	0	9999
U o %M	100	9999
T o %T	0	9999
C o %C	0	9999
M o %MW	200	9999
L o %MD	100	4998
F o %MF	100	4998

La zona de atribución puede modificarse para cada tipo de variable utilizando la directiva de compilación #SR« tipo »=« principio », « fin »
« tipo » designa el tipo de variable; « principio » y « fin », los nuevos límites a utilizar.
Esta directiva modifica la atribución de las variables automáticas en cada sitio del folio donde está escrita hasta la siguiente directiva « #SR ».

1.5. A propósito de ejemplos

Para ilustrar mejor este manual, hemos desarrollado ejemplos que funcionan con una maqueta de tren cuyo esquema es el siguiente:



Para pilotear esta maqueta hemos utilizado mapas de E/S en PC. Los símbolos definidos por el constructor de la maqueta se han conservado.

Se ha creado el siguiente fichero de símbolos:

AV1	O0	alimentation voie 1
AV2	O1	alimentation voie 2
AV3	O2	alimentation voie 3
AV4	O3	alimentation voie 4
AV5	O4	alimentation voie 5
AV6	O5	alimentation voie 6
AV7	O6	alimentation voie 7
AV8	O7	alimentation voie 8
AP1	O8	alimentation plateforme 1
AP2	O9	alimentation plateforme 2
AP3	O10	alimentation plateforme 3
AP4	O11	alimentation plateforme 4
AP5	O12	alimentation plateforme 5
IP1	O13	rotation plateforme 1
IP2	O14	rotation plateforme 2
IP3	O15	rotation plateforme 3
IP4	O16	rotation plateforme 4
IP5	O17	rotation plateforme 5
ZP1	O18	initialisation plateforme 1
ZP2	O19	initialisation plateforme 2
ZP3	O20	initialisation plateforme 3
ZP4	O21	initialisation plateforme 4
ZP5	O22	initialisation plateforme 5
DV1	O23	direction voie 1
DV2	O24	direction voie 2
DV3	O25	direction voie 3
DV4	O26	direction voie 4
DV5	O27	direction voie 5
DV6	O28	direction voie 6
DV7	O29	direction voie 7
DV8	O30	direction voie 8
S1D	O31	feu droit voie 1
S1I	O32	feu gauche voie 1
S2A	O33	feu haut voie 2
S2B	O34	feu bas voie 2
S3D	O35	feu droit voie 3
S3I	O36	feu gauche voie 3
S4A	O37	feu haut voie 4
S4B	O38	feu bas voie 4
S5D	O39	feu droit voie 5
S5I	O40	feu gauche voie 5
S6D	O41	feu droit voie 6
S6I	O42	feu gauche voie 6
S7D	O43	feu droit voie 7
S7I	O44	feu gauche voie 7
S8D	O45	feu droit voie 8
S8I	O46	feu gauche voie 8
T1D	i0	train droit voie 1
T1I	i1	train gauche voie 1
T2A	i2	train haut voie 2
T2B	i3	train bas voie 2
T3D	i4	train droit voie 3
T3I	i5	train gauche voie 3
T4A	i6	train haut voie 4
T4B	i7	train bas voie 4
T5D	i8	train droit voie 5

T5I	i9	train gauche voie 5
T6D	i10	train droit voie 6
T6I	i11	train gauche voie 6
T7D	i12	train droit voie 7
T7I	i13	train gauche voie 7
T8D	i14	train droit voie 8
T8I	i15	train gauche voie 8
TP1	i16	train plateforme 1
TP2	i17	train plateforme 2
TP3	i18	train plateforme 3
TP4	i19	train plateforme 4
TP5	i20	train plateforme 5
P1P	i21	index plateforme 1
P2P	i22	index plateforme 2
P3P	i23	index plateforme 3
P4P	i24	index plateforme 4
P5P	i25	index plateforme 5
P1Z	i26	init plateforme 1
P2Z	i27	init plateforme 2
P3Z	i28	init plateforme 3
P4Z	i29	init plateforme 4
P5Z	i30	init plateforme 5
ERR	i31	court-circuit

1.6. Grafcet

AUTOMGEN soporta los elementos siguientes:

- ⇨ divergencias y convergencias en « Y » y en « O »,
- ⇨ etapas pozos y fuentes,
- ⇨ transiciones pozos y fuentes,
- ⇨ sincronización,
  - ⇨ forzados de Grafcets,
  - ⇨ memorización de Grafcets,
  - ⇨ fijación,
  - ⇨ macro-etapas.

1.6.1. Grafcet simple

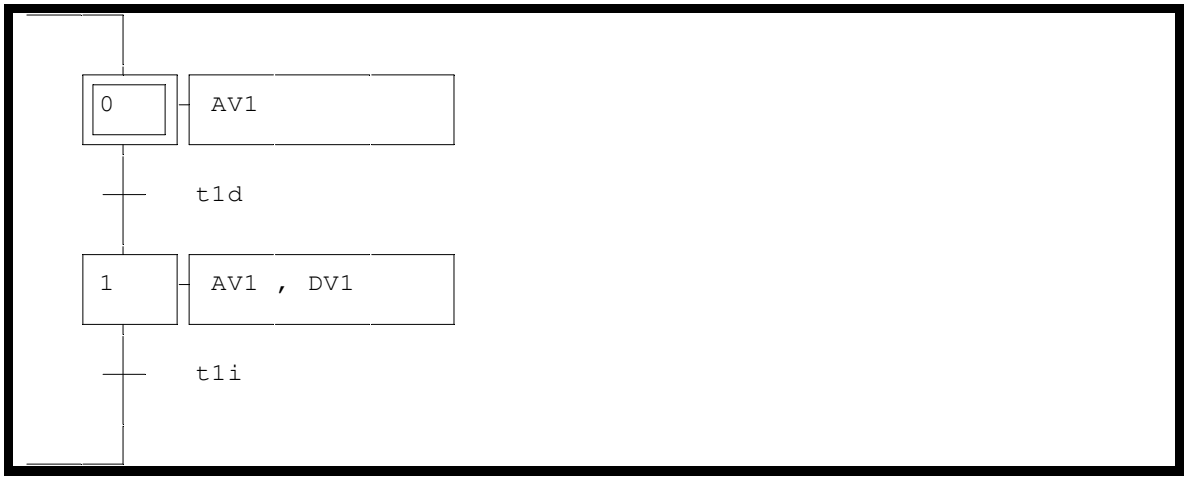
La escritura de Grafcet en línea se resume a la yuxtaposición de etapas y transiciones.

Ilustremos un Grafcet en línea con el ejemplo siguiente:

Condiciones:

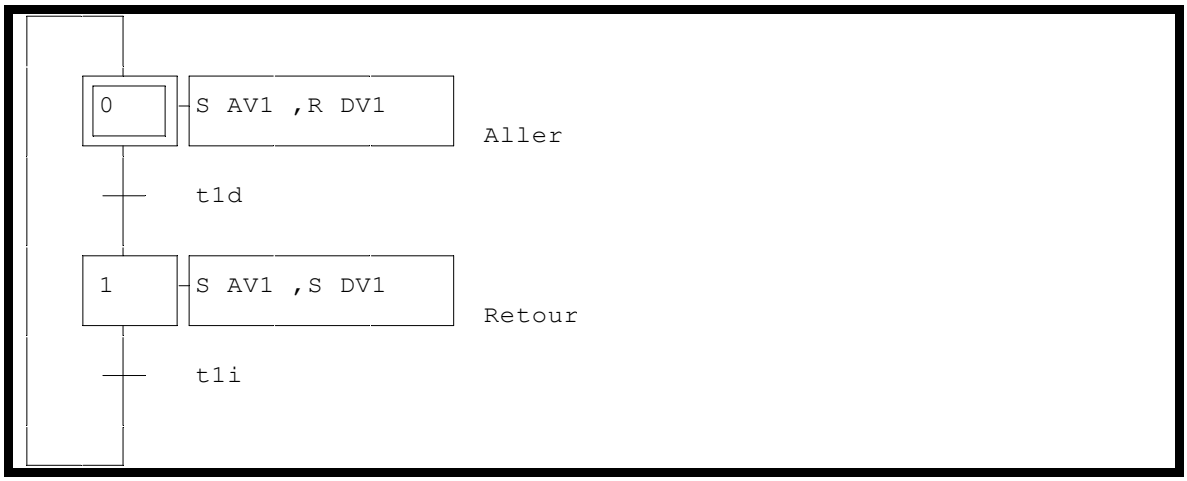
La locomotora debe partir por la vía 3 hacia la derecha, hasta el extremo de la vía. Luego regresa en sentido inverso hasta el otro extremo y vuelve a empezar.

Solución 1:



exemples\grafcet\simple1.agn

Solución 2:

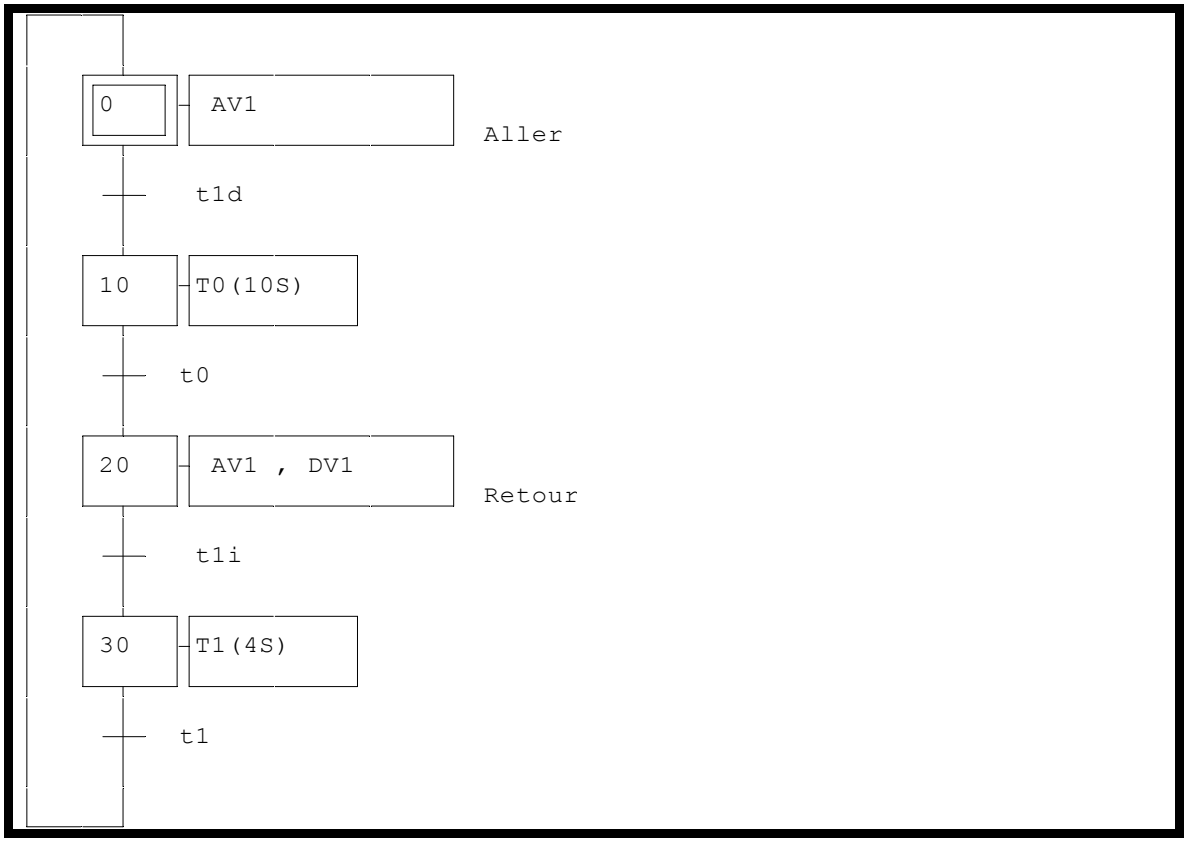


exemple\grafcet\simple2.agn

La diferencia entre estas dos soluciones reside en el uso de las acciones « Asignación » para el primer ejemplo y de las acciones « Puesta en uno » y « Puesta en cero » para el segundo.

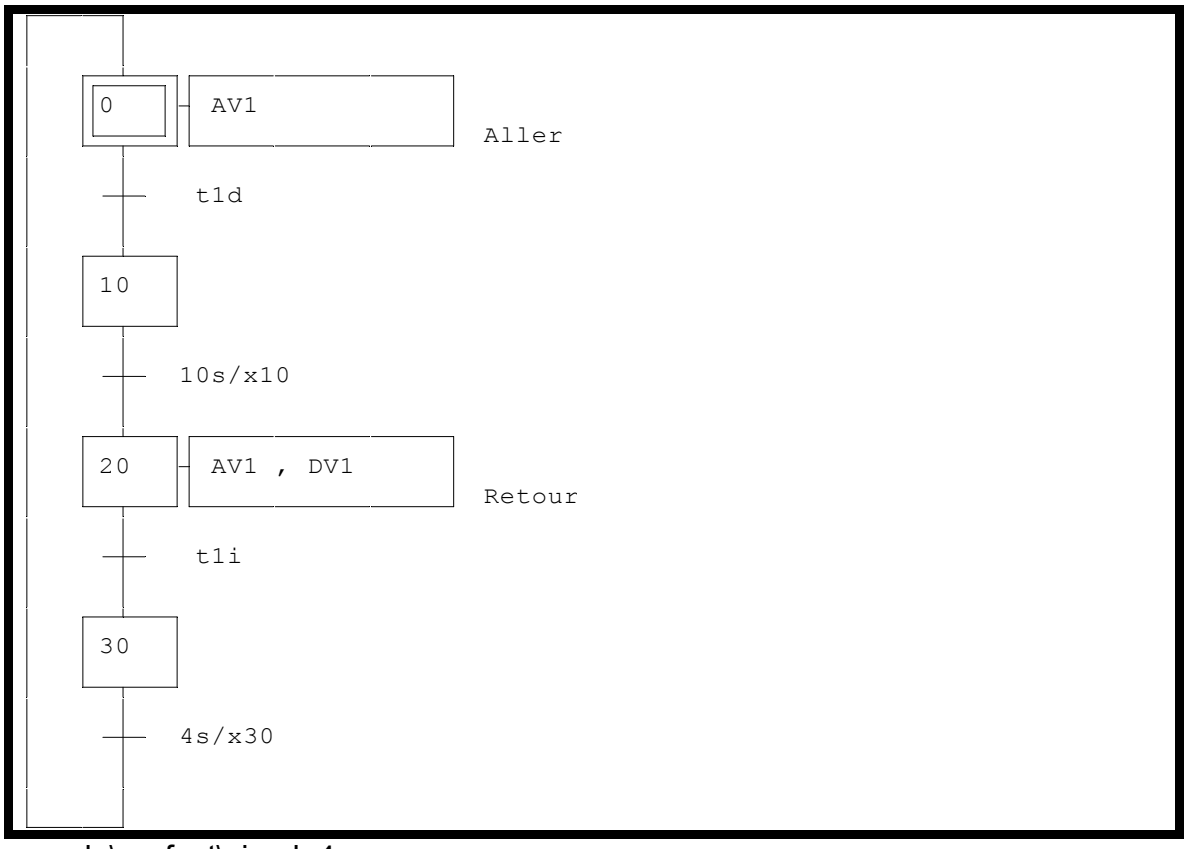
Modifiquemos las condiciones con una espera de 10 segundos cuando la locomotora llega a la derecha de la vía 1 y una espera de 4 segundos cuando la locomotora llega a la izquierda de la vía 1.

Solución 1:



exemple\grafcet\simple3.agn

Solución 2:

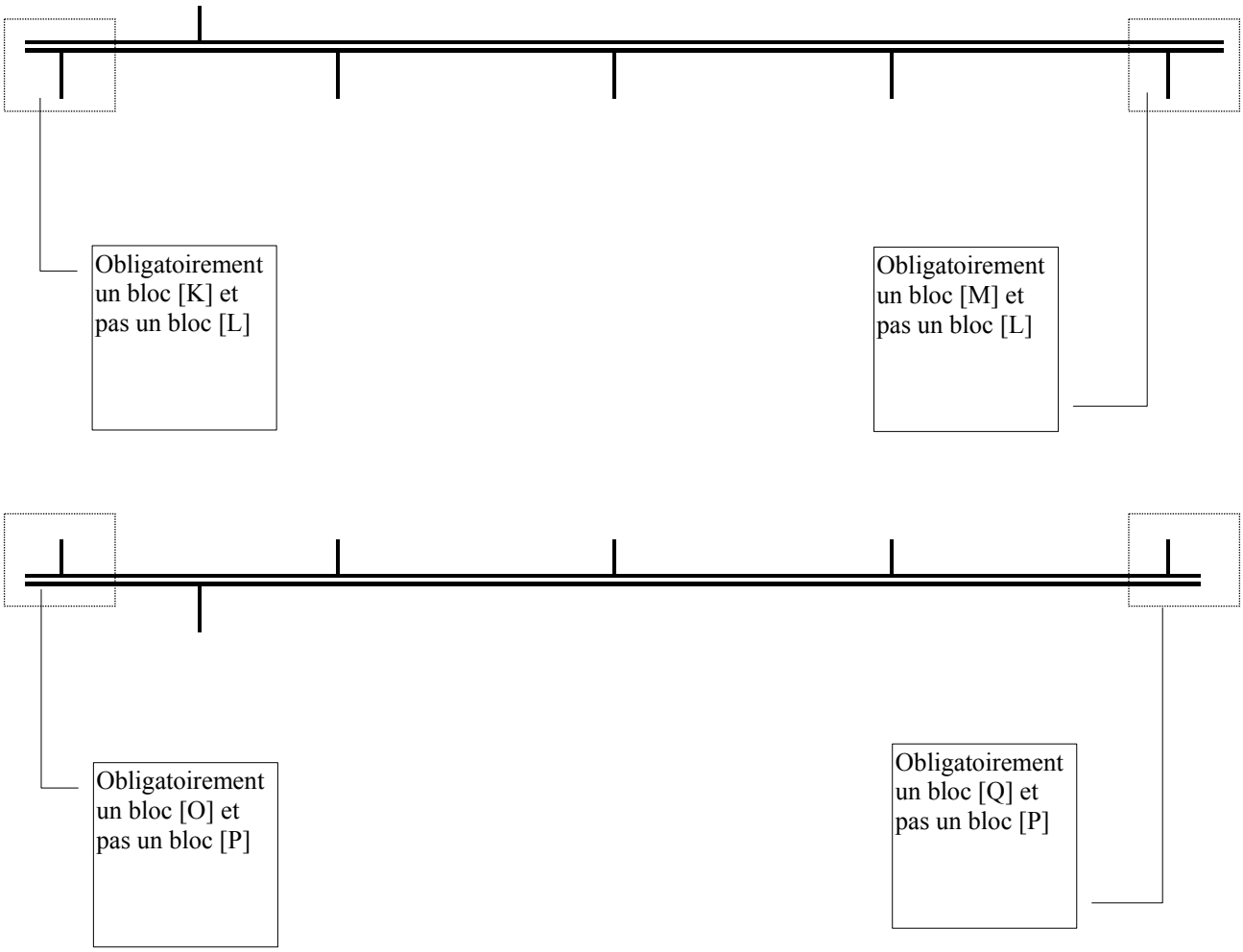


exemple\grafcet\simple4.agn

La diferencia entre los ejemplos 3 y 4 reside en la elección de la sintaxis utilizada para definir las temporizaciones. El resultado desde el punto de vista funcional es idéntico.

1.6.2. Divergencia y convergencia en « Y »

Las divergencias en« Y » pueden tener n ramas. Lo importante es respetar la utilización de los bloques de función:

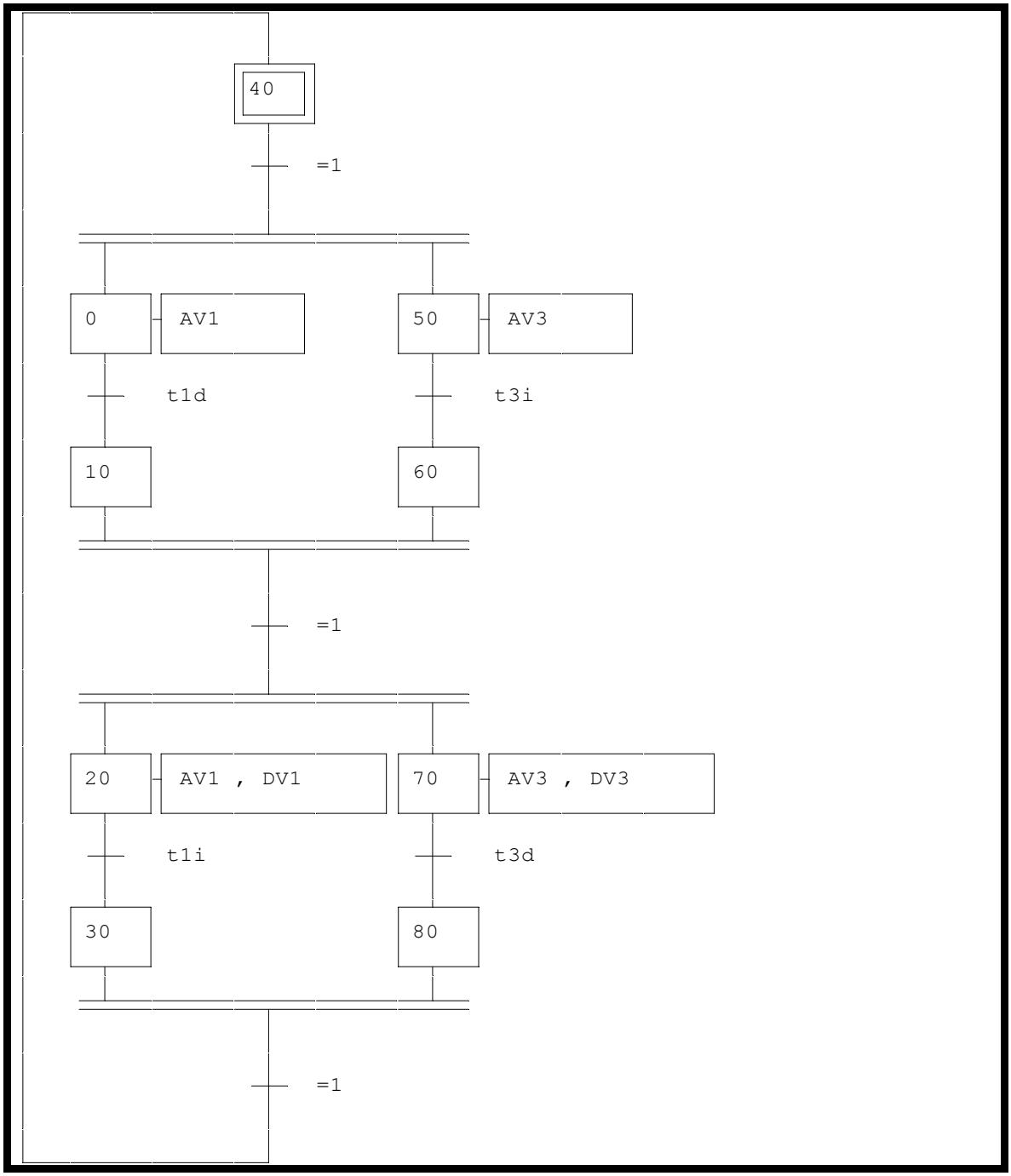


Ilustremos la utilización de las divergencias y convergencias en « Y ».

Condiciones:

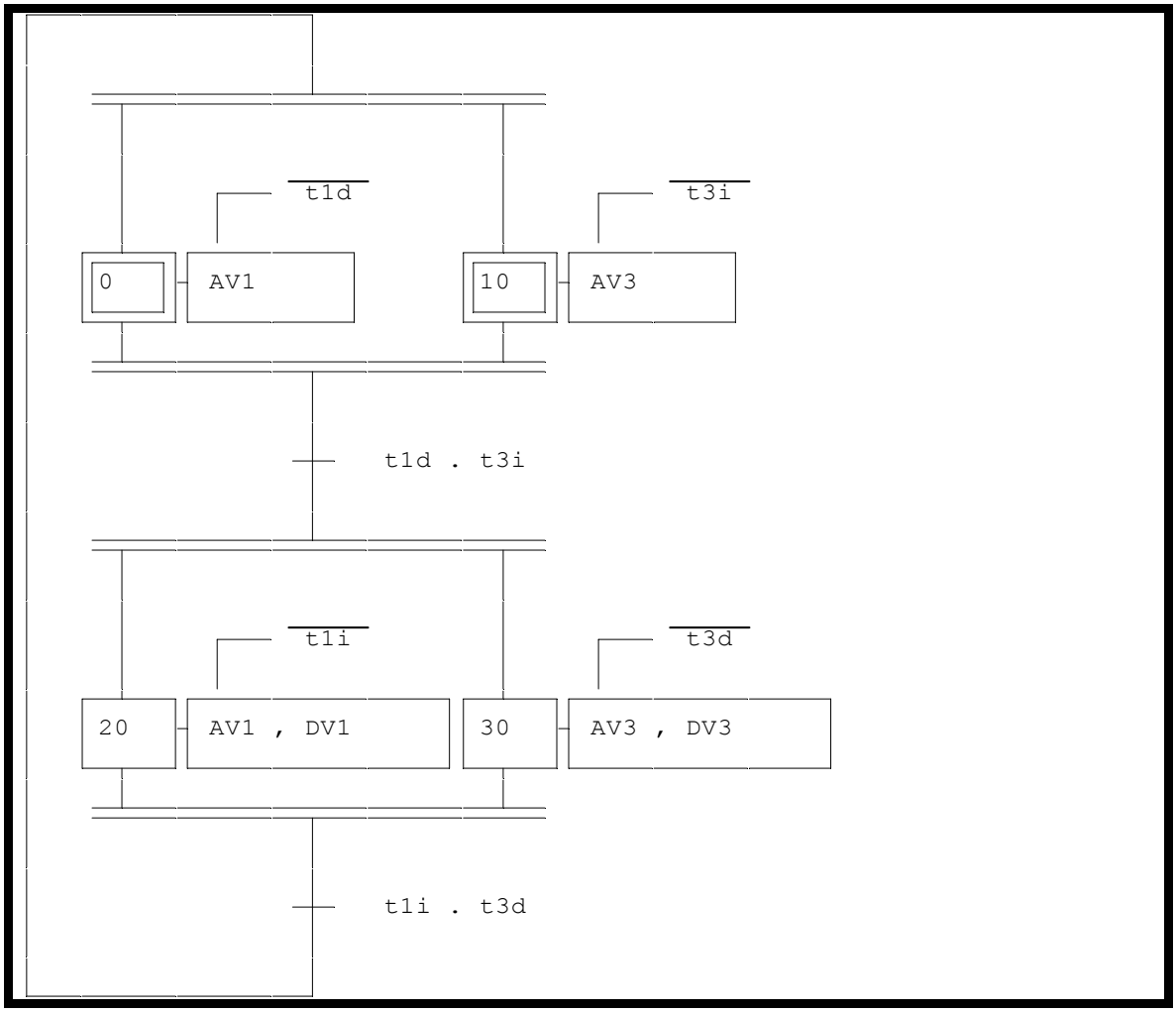
Vamos a utilizar dos locomotoras: la primera efectuará idas y vueltas por la vía 1, la segunda por la vía 3. Las dos locomotoras estarán sincronizadas (se esperarán en el extremo de la vía).

Solución 1:



exemple\grafcet\divergence et 1.agn

Solución 2:

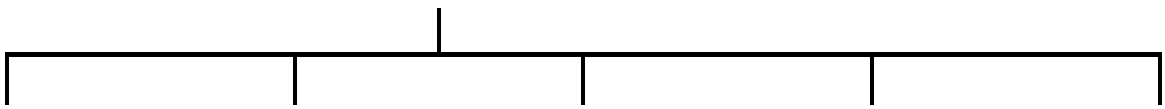


exemple\grafcet\divergence et 2.agn

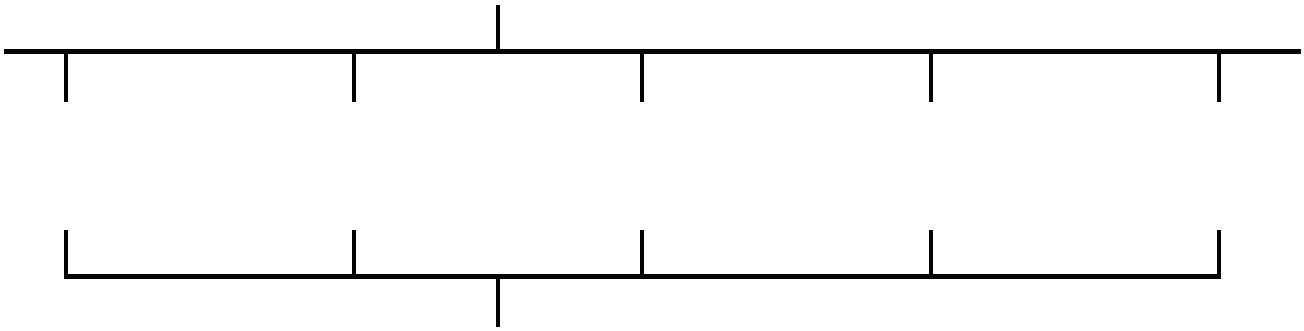
Estas dos soluciones son equivalentes desde el punto de vista funcional. La segunda es una versión más compacta que utiliza acciones condicionadas.

1.6.3. Divergencia y convergencia en « O »

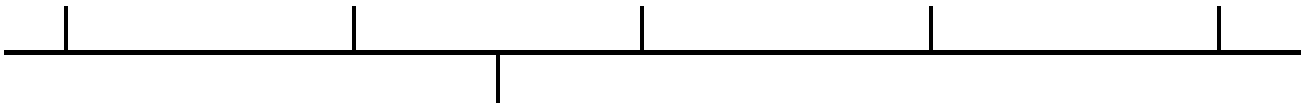
Las divergencias en « O » pueden tener n ramas. Lo importante es respetar la utilización de los bloques de función:



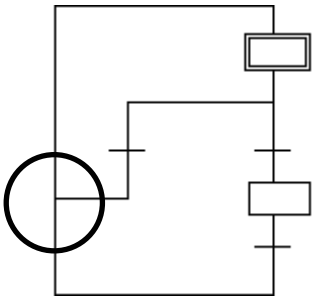
o



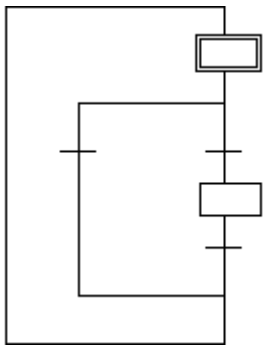
o



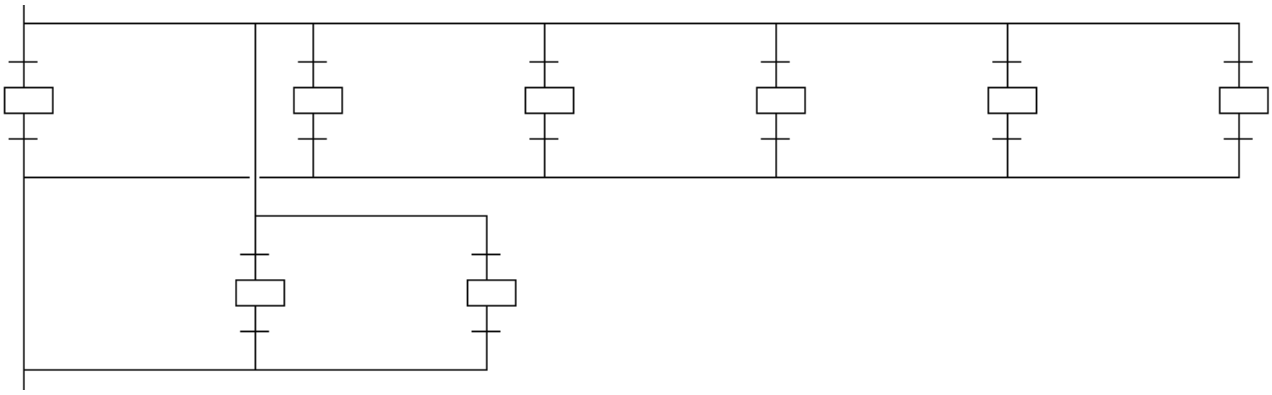
Las divergencias en « O » deben ramificarse obligatoriamente en vínculos descendentes. Por ejemplo:



incorrecto; el diseño correcto es:



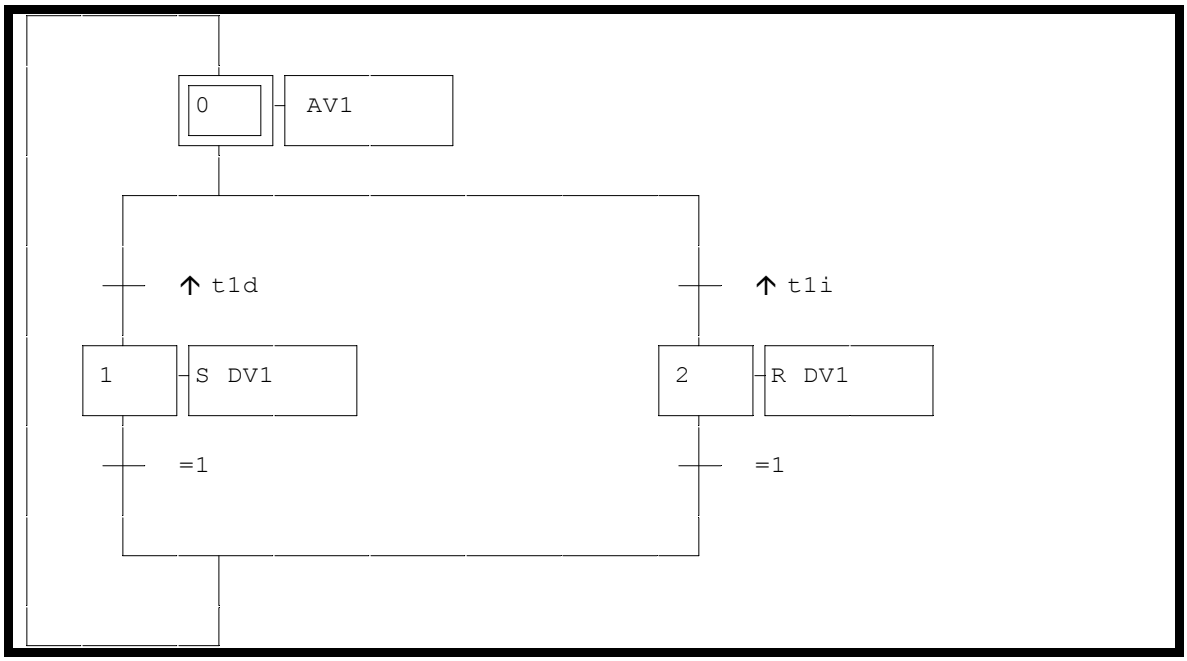
Si el ancho de la página no permite escribir un gran número de divergencias, se puede adoptar una estructura del tipo:



Veamos un ejemplo para ilustrar la utilización de las divergencias y convergencias en « O »:

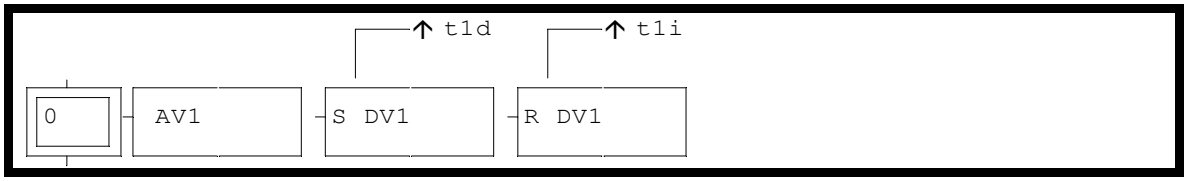
Condiciones:
Retomemos las condiciones del primer ejemplo del capítulo: ida y vuelva de una locomotora por la vía 1.

Solución:



exemple\grafcet\divergence ou.agn

Este Grafcet podría resumirse en una etapa utilizando acciones condicionadas, como en este ejemplo:



exemple\grafcet\action conditionnée.agn

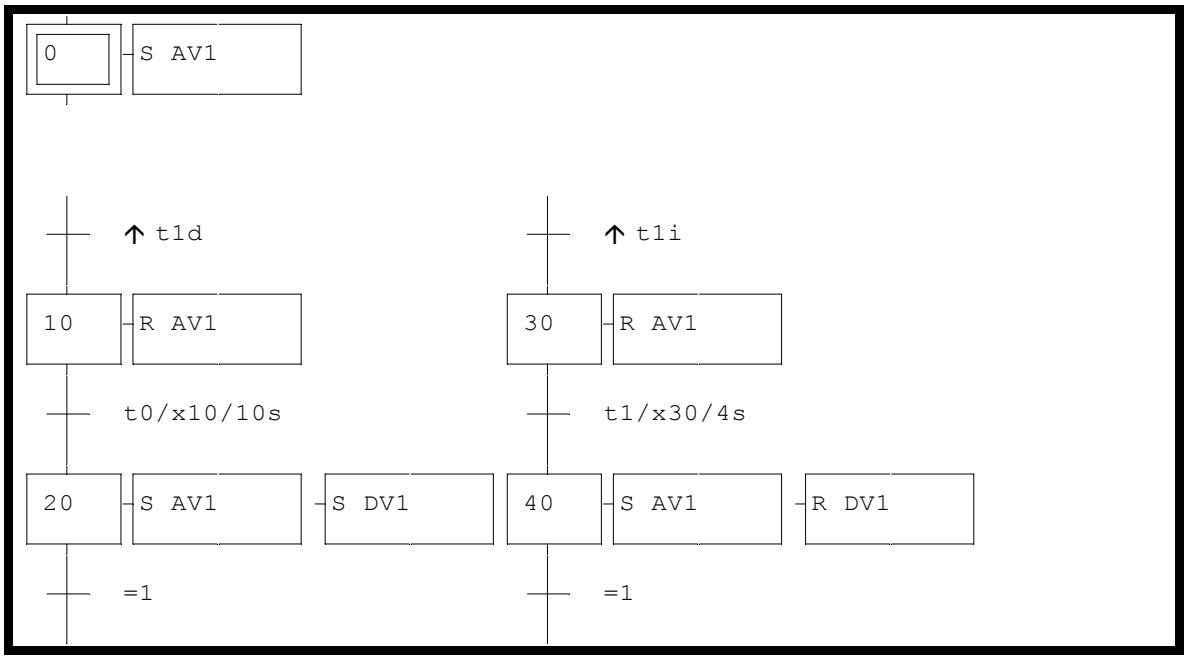
1.6.4. Etapas pozos y fuentes, transiciones pozos y fuentes

Ilustremos estos principios con ejemplos:

Condiciones:

Tratemos de nuevo el segundo ejemplo de este capítulo: ida y vuelta de una locomotora por la vía 1 con espera al final de la vía.

Solución:



exemple\grafcet\étapes puits et sources.agn

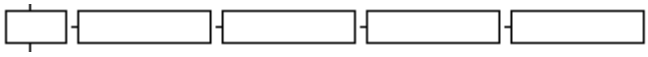
1.6.5. Acciones múltiples, acciones condicionadas

Ya hemos utilizado en este capítulo acciones múltiples y acciones condicionadas. Detallemos estos dos principios.

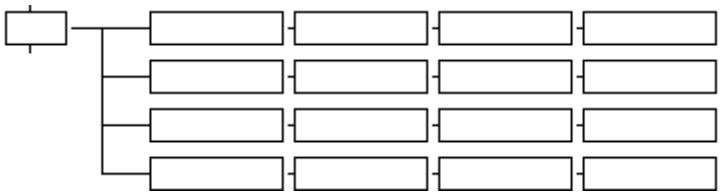
Como se indica en el capítulo sobre el compilador, es posible escribir varias acciones en un mismo rectángulo; en este caso, el carácter « , » (coma) sirve como delimitador.

Cuando una condición se añade a un rectángulo de acción, se condiciona el conjunto de las acciones contenidas en el rectángulo.

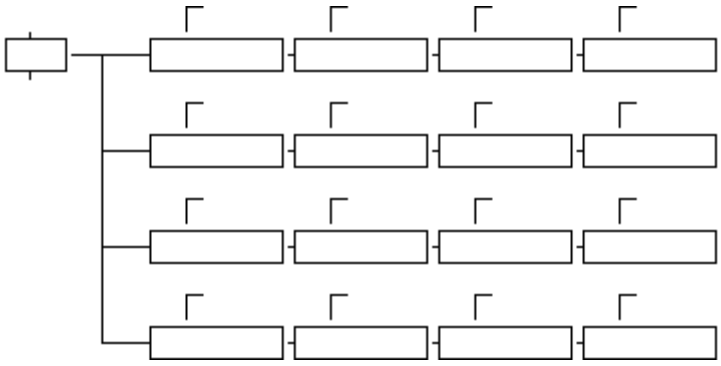
Es posible asociar varios rectángulos de acción a una etapa:



Otra posibilidad:



Cada rectángulo puede recibir una condición diferente:



Para diseñar una acción condicionada, ubique el cursor en el rectángulo de acción, haga clic con el botón derecho del ratón y elija « Acción condicional » en el menú. Para documentar la condición sobre acción, haga clic en el elemento  .

La sintaxis IF(condición) permite escribir una condición sobre acción en el rectángulo de acción.

1.6.6. Sincronización

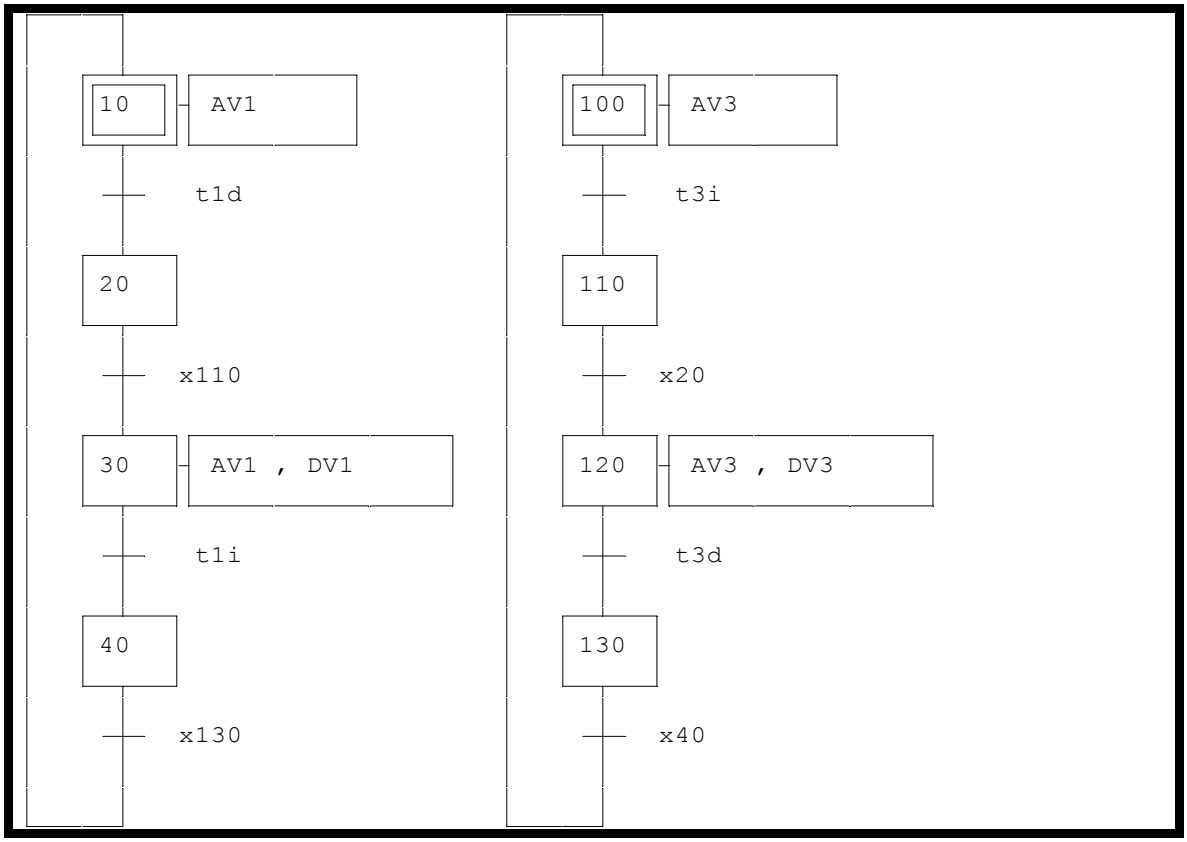
Retomemos un ejemplo ya tratado para ilustrar la sincronización de Grafcets.

Condiciones:

Ida y vuelta de dos locomotoras por las vías 1 y 3 con espera entre las locomotoras en el extremo de la vía.

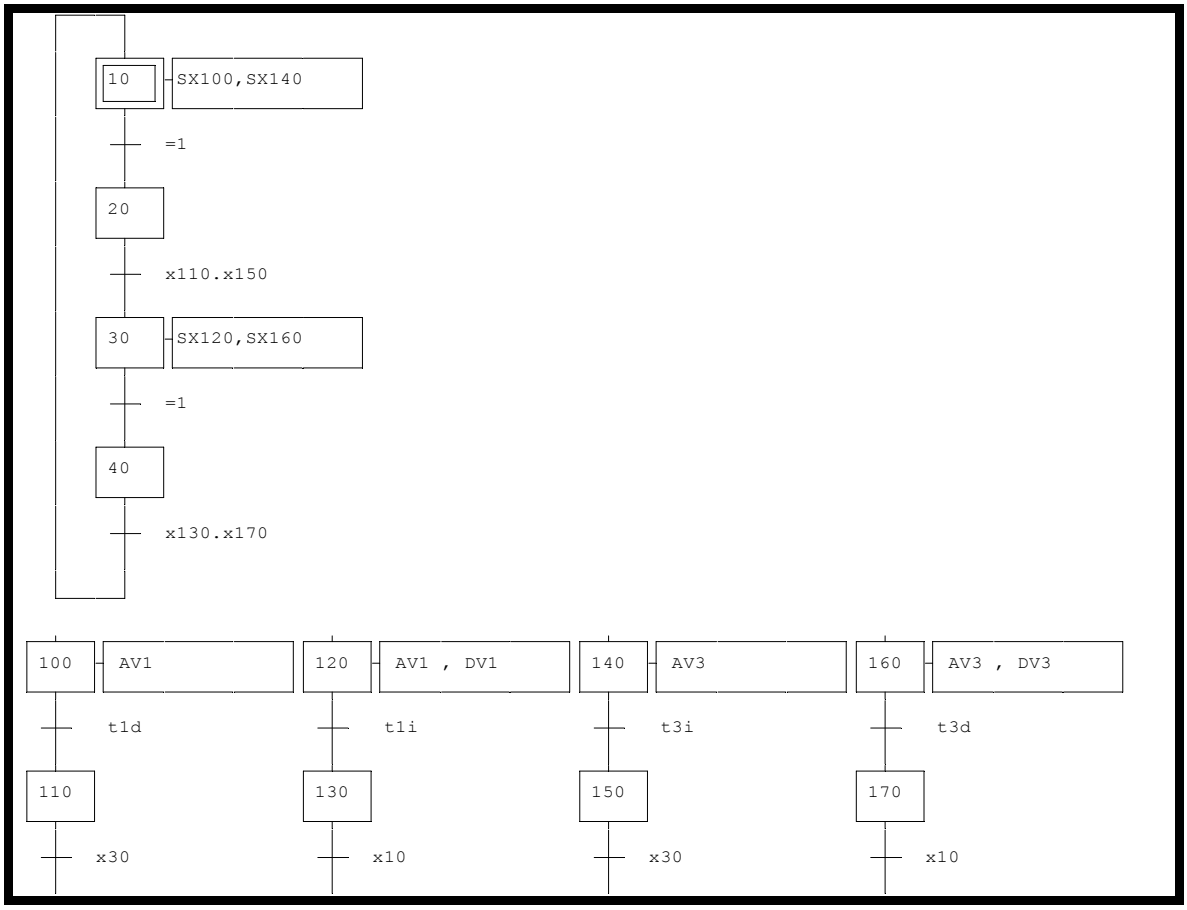
Este ejemplo se había tratado con una divergencia en « Y ».

Solución 1:



exemple\grafcet\synchro1.agn

Solución 2:



exemple\grafcet\synchro2.agn

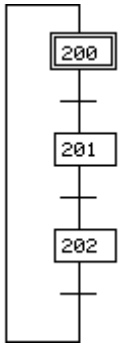
Esta segunda solución es un excelente ejemplo del arte de complicar las cosas más simples con fines pedagógicos.

1.6.7. Forzados de Grafcet

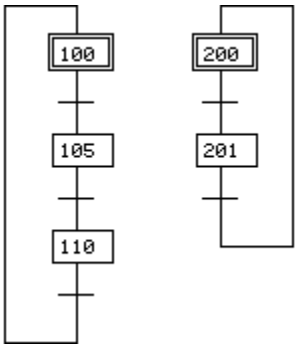
El compilador agrupa las etapas en función de los vínculos establecidos entre ellas. Para designar un Grafcet, es suficiente hacer referencia a una de las etapas que componen ese Grafcet.

Del mismo modo se puede designar el conjunto de Grafcets presentes en un folio mencionando el nombre del folio.

Por ejemplo:



Para designar este Grafcet hablaremos de Grafcet 200, Grafcet 201 o Grafcet 202.
Al ser un conjunto de etapas, el Grafcet se convierte en una variable estructurada compuesta por n etapas; cada etapa puede estar activa o inactiva.
Como hemos visto, AUTOMGEN divide las etapas en conjuntos independientes que pueden agruparse, lo cual permite considerarlos como un solo Grafcet. Para agrupar varios Grafcets se debe utilizar la directiva de compilación « `#G:g1,g2` » (comando a incluir en un comentario). Este comando agrupa los Grafcets g1 y g2. Recordemos que la designación de un Grafcet se efectúa invocando el número de una de sus etapas.
Veamos un ejemplo:
#G:105,200
esta directiva de compilación agrupa estos dos Grafcets:



Observación: es posible utilizar varias directivas « #G » para agrupar más de dos Grafcets.
Ahora vamos a detallar las órdenes de forzado utilizables. Se escribirán simplemente en rectángulos de acción como asignaciones clásicas. Soportarán tanto los operadores S(puesta en uno), R(puesta en cero), N(asignación complementada) e I(inversión) como las acciones condicionales.

1.6.7.1. Forzado de un Grafcet según una lista de etapas activas

Sintaxis:

« F<Grafcet>:{<lista de etapas activas>} »

o

 « F/<nombre de folio>:{<lista de etapas activas>} »

El o los Grafcets designados se forzarán al estado definido por la lista de etapas activas que se encuentra entre llaves. Si deben estar activas varias etapas, es necesario separarlas con el carácter « , » (coma). Si el o los Grafcets deben forzarse al estado vacío (ninguna etapa activa), no hay que precisar ninguna etapa entre las dos llaves.

 El número de etapas puede estar precedido de « X ». Así se puede asociar un símbolo al nombre de una etapa.

Ejemplos:

« F10:{0} »

fuerza todas las etapas del Grafcet 10 a 0 salvo la etapa 0, que se fuerza a 1.

« F0:{4,8,9,15} »

fuerza todas las etapas del Grafcet 0 a 0, salvo las etapas 4, 8, 9 y 15, que se fuerzan a 1.

« F/marcha normal:{} »

fuerza todos los Grafcets que se encuentran en el folio « marcha normal » al estado vacío.

1.6.7.2. Memorización del estado de un Grafcet

Estado actual de un Grafcet:

Sintaxis:

« G<Grafcet>:<N° de bit> »

o

 « G/<nombre de folio>:<N° de bit> »

Este comando memoriza el estado de uno o varios Grafcets en una serie de bits. Es necesario reservar un espacio para almacenar el estado del o de los Grafcets

designados (un bit por etapa). Estos bits de almacenamiento deben ser consecutivos. Utilice un comando #B para reservar un espacio lineal de bit.

 El número de la etapa que designa el Grafcet puede estar precedido de « X ». Así se puede asociar un símbolo al nombre de una etapa. El número del bit puede estar precedido de « U » o de « B ». Así se puede asociar un símbolo al primer bit de la zona de almacenamiento de estado.

Estado particular de un Grafcet:
Sintaxis:
« G<Grafcet>:<N° de bit> {lista de etapas activas} »
o
 « G/<nombre de folio>:<N° de bit> {lista de etapas activas} »
Este comando memoriza el estado definido por la lista de etapas activas aplicadas a los Grafcets especificados a partir del bit indicado. También aquí es necesario reservar un número suficiente de bits. Si debe memorizarse una situación vacía, no debe aparecer ninguna etapa entre las dos llaves.

 El número de las etapas puede estar precedido de « X ». Así se puede asociar un símbolo al nombre de una etapa. El número del bit puede estar precedido de « U » o de « B ». Así se puede asociar un símbolo al primer bit de la zona de almacenamiento de estado.
Ejemplos:
« G0:100 »
memoriza el estado actual del Grafcet 0 a partir de U100.
« G0:U200 »
memoriza el estado vacío del Grafcet 0 a partir de U200.
« G10:150{1,2} »
memoriza el estado del Grafcet 10, en el que sólo las etapas 1 y 2 están activas, a partir de U150.
« G/PRODUCCIÓN:_ GUARDAR ESTADO PRODUCCIÓN _ »
memoriza el estado de los Grafcets que se encuentran en el folio « PRODUCCIÓN » en la variable _GUARDAR ESTADO PRODUCCIÓN_.

1.6.7.3. Forzado de un Grafcet a partir de un estado memorizado

Sintaxis:

« F<Grafcet>:<N° de bit> »

o

 « F/<Nombre de folio>:<N° de bit> »

Fuerza el o los Grafcets con el estado memorizado a partir del bit precisado.

El número de la etapa que designa el Grafcet puede estar precedido de 'X'. Así se puede asociar un símbolo al nombre de una etapa. El número del bit puede estar precedido de « U » o de « B ». Así se puede asociar un símbolo al primer bit de la zona de almacenamiento de estado.

Ejemplo:

« G0:100 »

memoriza el estado actual del Grafcet 0

« F0:100 »

restaura ese estado

1.6.7.4. Fijación de un Grafcet

Sintaxis:

« F<Grafcet> »

o

 « F/<Nombre de folio> »

Fija uno o varios Grafcets: impide su evolución.

Ejemplo:

« F100 »

fija el Grafcet 100

« F/producción »

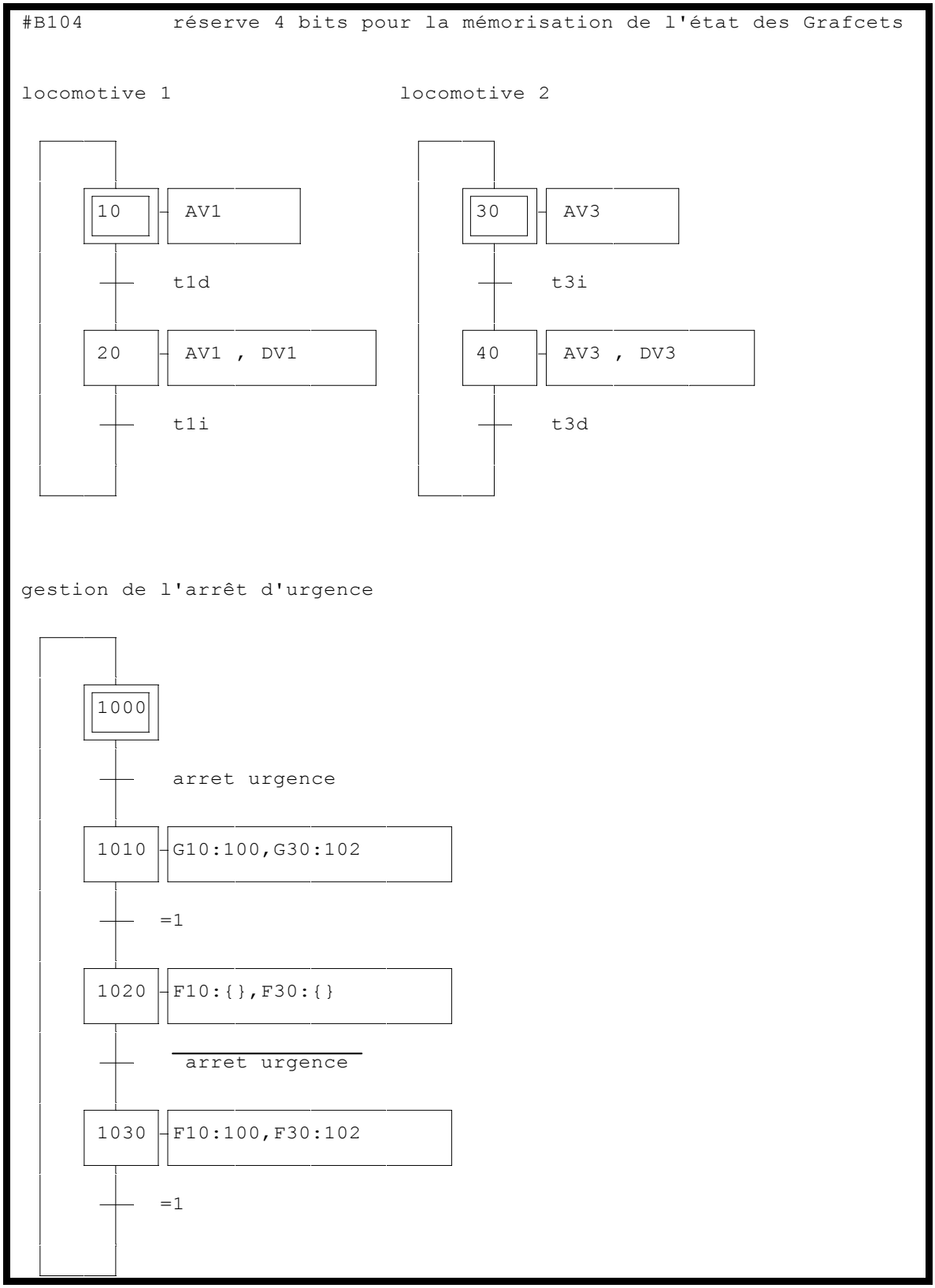
fija los Grafcets contenidos en el folio « producción »

Ilustremos los forzados con un ejemplo.

Condiciones:

Retomemos un ejemplo ya tratado: ida y vuelta de dos locomotoras por las vías 1 y 3 (esta vez sin espera entre las locomotoras). Añadamos una parada de urgencia. Cuando se detecta la parada de urgencia, todas las salidas se ponen en cero. Al desaparecer la parada de urgencia, el programa debe reanudar el recorrido desde donde se detuvo.

Solución 1:

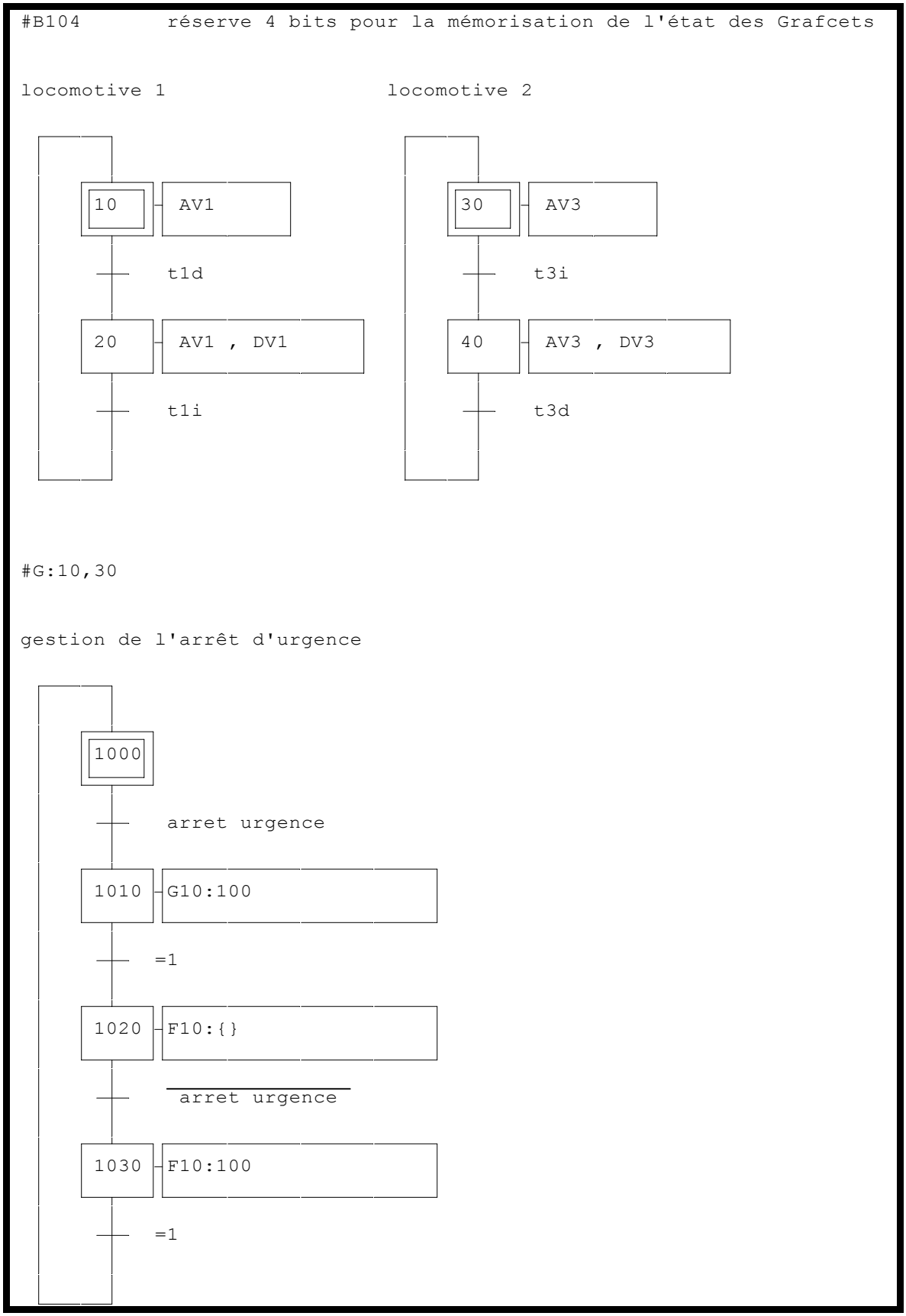


exemple\grafcet\forçage1.agn

La utilización de la directiva #B104 permite reservar cuatro bits consecutivos (U100 a U103) para memorizar el estado de los dos Grafcets.

« _parada urgencia_ » ha sido asociado a un bit (U1000). Por lo tanto su estado puede modificarse desde el entorno posicionándose encima y haciendo clic cuando la visualización dinámica está activada.

Solución 2:



exemple\grafcet\forçage2.agn

Esta segunda solución muestra la utilización de la directiva de compilación « #G » que permite agrupar los Grafcets para los comandos de forzado.

1.6.8. Macro-etapas

AUTOMGEN implementa las macro-etapas.

A propósito de este tema recordemos que:

Una macro-etapa ME es la única representación de un conjunto único de etapas y transiciones llamado « expansión de ME ».

Una macro-etapa obedece a las reglas siguientes:

⇨ la expansión de ME implica una etapa particular llamada **etapa de entrada** y una etapa particular llamada **etapa de salida.**

⇨ la etapa de entrada tiene la propiedad siguiente: si superamos una transición anterior a la macro-etapa, se activa la etapa de entrada de su expansión.

⇨ la etapa de salida tiene la propiedad siguiente: participa de la validación de las transiciones posteriores a la macro-etapa.

⇨ fuera de las transiciones anteriores y posteriores a ME, no existe ningún vínculo estructural entre, por una parte, una etapa o una transición de la expansión ME y, por otra, una etapa o una transición que no pertenece a ME.

La utilización de las macro-etapas en AUTOMGEN se define así:

⇨ la expansión de una macro-etapa es un Grafcet en un folio distinto,

⇨ la etapa de entrada de la expansión de una macro-etapa deberá llevar el número 0 o la identificación Exxx (xxx = un número cualquiera),

⇨ la etapa de salida de la expansión de una macro-etapa deberá llevar el número 9999 o la identificación Sxxx (xxx = un número cualquiera),

⇨ fuera de estas dos últimas obligaciones, la expansión de una macro-etapa puede ser un Grafcet cualquiera y como tal puede contener macro-etapas (la imbricación de macro-etapas es posible).

1.6.8.1. Cómo definir una macro-etapa

Debe utilizarse el símbolo  . Para colocar este símbolo, haga clic en un sitio vacío del folio y elija « Más …/Macro-etapa » en el menú contextual. Para abrir el menú contextual, haga clic con el botón derecho del ratón sobre el fondo del folio.

Para definir la expansión de la macro-etapa, cree un folio, diseñe la expansión y modifique las propiedades del folio (haciendo clic con el botón derecho del ratón en el nombre del folio en el explorador). Ajuste el tipo de folio en « Expansión de macro-etapas » y el número de la macro-etapa.

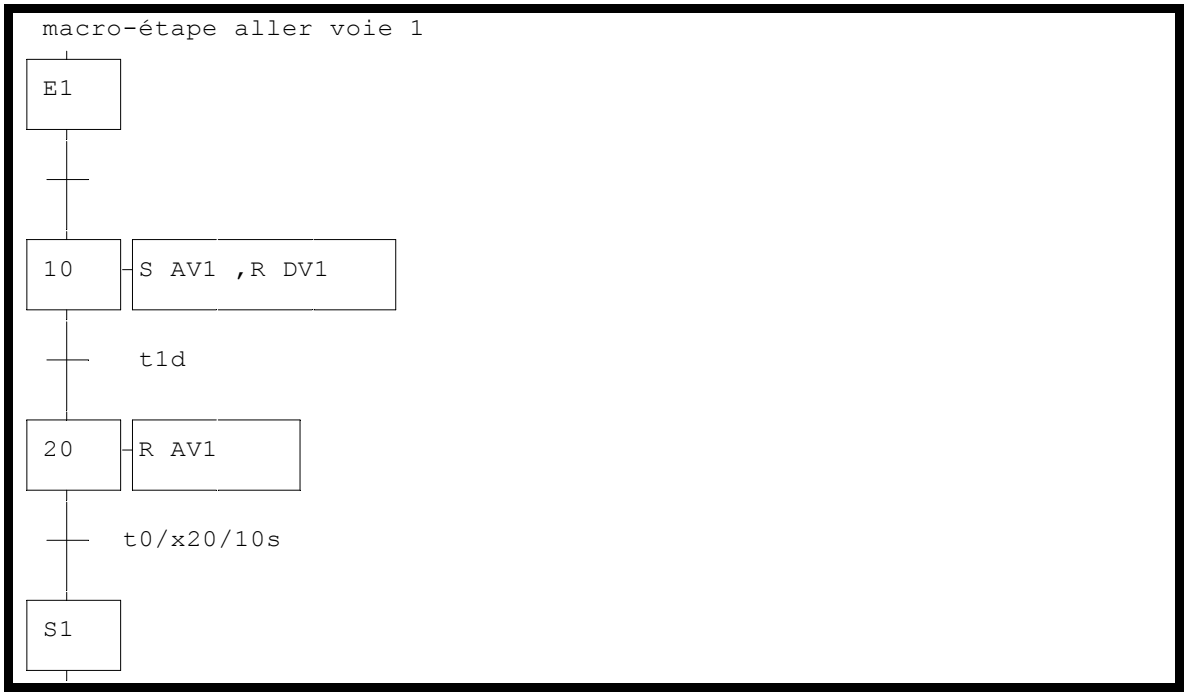
En modo ejecución, es posible visualizar una expansión de macro-etapa. Para ello, es necesario ubicar el cursor sobre la macro-etapa y hacer clic con el botón izquierdo del ratón.

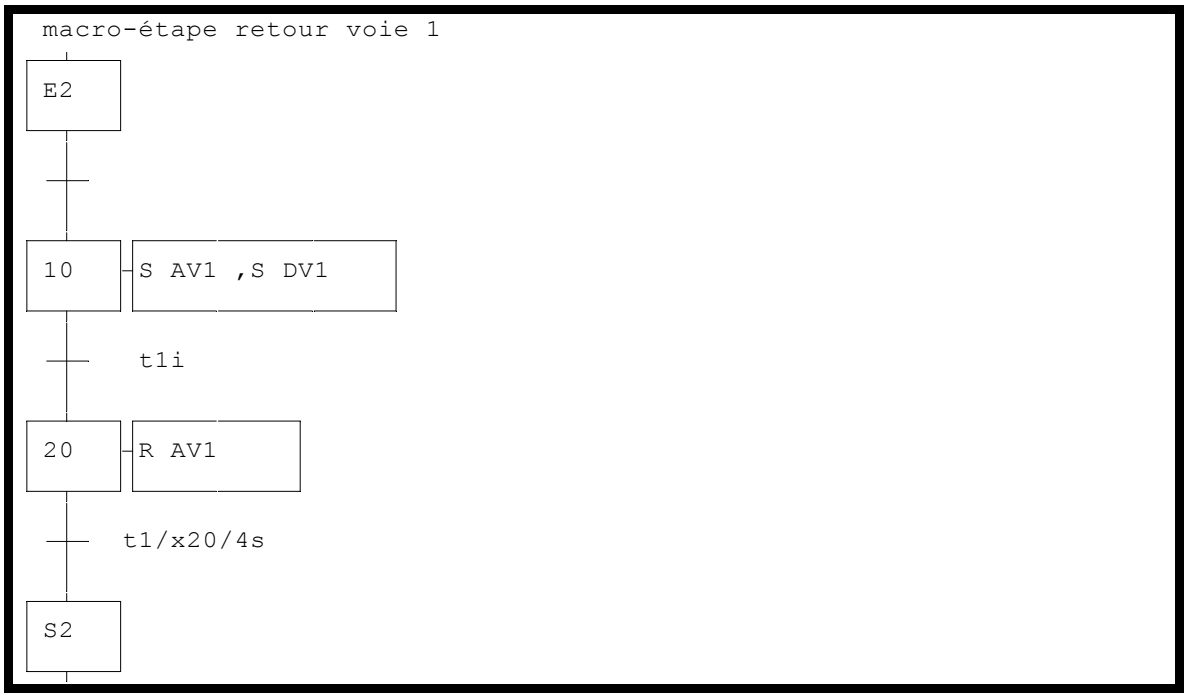
Observaciones:

- ⇨ las etapas y los bits Usuario utilizados en una expansión de macro-etapa son locales, es decir que no tienen ninguna relación con las etapas y los bits de otros Grafcets. Los otros tipos de variables no presentan esta característica: son comunes a todos los niveles.
- ⇨ si una zona de bits debe utilizarse de manera global, es necesario declararla con la directiva de compilación « #B ».
- ⇨ la asignación de variables no locales por diferentes niveles o diferentes expansiones no es gestionada por el sistema. En otros términos, es necesario utilizar las asignaciones « S » « R » o « I » para garantizar un funcionamiento coherente del sistema.

Ilustremos la utilización de las macro-etapas con un ejemplo ya tratado: ida y vuelta de una locomotora por la vía 1 con espera en el extremo de la vía. Descompondremos la ida y la vuelta en dos macro-etapas distintas.

Solución:


macro-étape aller voie 1
E1
10
S AV1 ,R DV1
t1d
20
R AV1
t0/x20/10s
S1


macro-étape retour voie 1
E2
10
S AV1 ,S DV1
t1i
20
R AV1
t1/x20/4s
S2

exemple\grafcet\macro-étape.agn

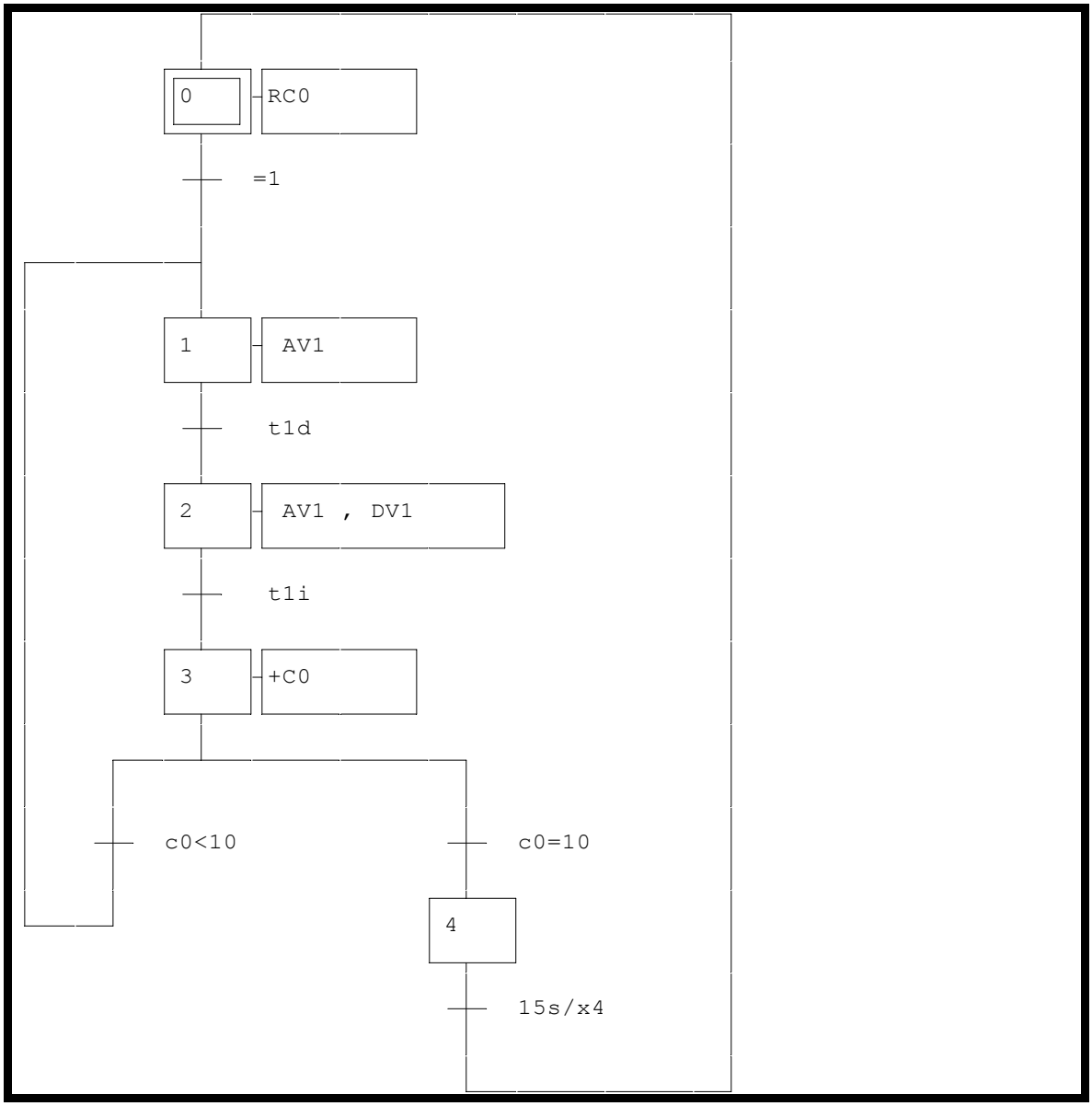
1.6.9. Contadores

Ilustremos la utilización de los contadores con un ejemplo.

Condiciones:

Una locomotora debe efectuar 10 idas y vueltas por la vía 1, inmovilizarse durante quince segundos y volver a arrancar.

Solución:

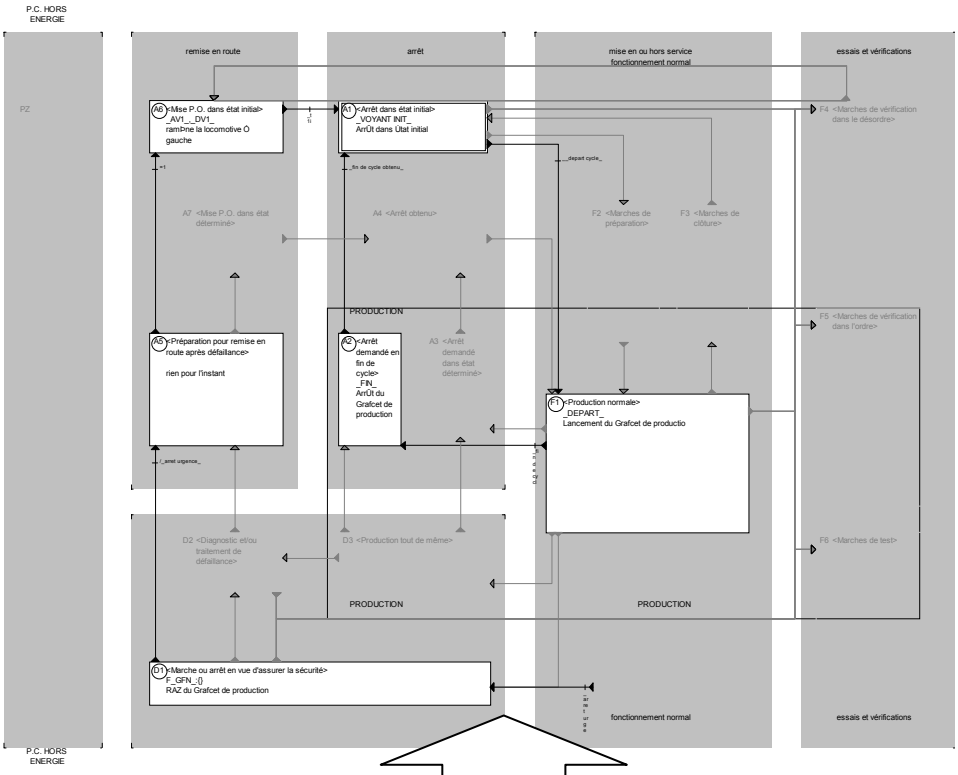


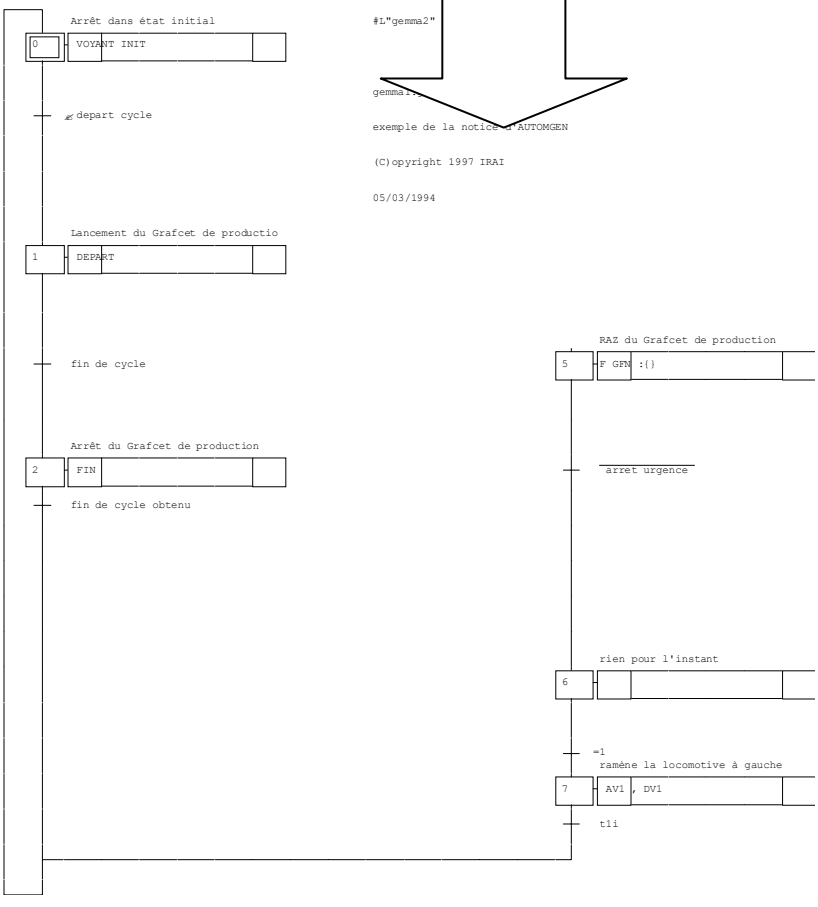
exemple\grafcet\compteur.agn

1.7. Gemma

AUTOMGEN implementa la descripción de Grafcet de gestión de los modos de marcha bajo forma de Gemma. El principio es un modo de edición transparente al modo Grafcet. Es posible pasar del modo de edición Grafcet al modo de edición Gemma. La traducción de un Gemma en Grafcet de gestión de modos de marcha es automático e inmediato.

El comando « Editar bajo la forma de un Gemma » del menú « Caja de herramientas » permite pasar de un modo al otro.






```
#L"gemma2"

gemma2.

exemple de la notice d'AUTOMGEN

(C)opyright 1997 IRAI

05/03/1994
```

1.7.1. Creación de un Gemma

Para crear un Gemma:

⇨ haga clic en el elemento « Folio » del explorador con el botón derecho del ratón y seleccione el comando « Añadir un nuevo folio »,

⇨ en la lista de tamaños elija « Gemma »,

⇨ haga clic en el pulsador « OK »,

⇨ haga clic con el botón derecho del ratón en el nombre del folio creado en el explorador,

⇨ elija « Propiedades » en el menú,

⇨ pinche la opción « Mostrar bajo la forma de un Gemma ».

La ventana contiene un Gemma en el que todos los rectángulos y vínculos están sombreados. Para validar un rectángulo o un vínculo hay que posicionarse encima y hacer clic con el botón derecho del ratón.

Para modificar el contenido de un rectángulo o la naturaleza de un vínculo hay que posicionarse encima y hacer clic con el botón izquierdo del ratón.

El contenido de los rectángulos del Gemma se ubicará en los rectángulos de acción del Grafcet. La naturaleza de los vínculos se ubicará en las transiciones del Grafcet.

A cada rectángulo del Gemma puede asociarse un comentario, que aparecerá cerca del rectángulo de acción correspondiente en el Grafcet.

1.7.2. Contenido de los rectángulos del Gemma

Los rectángulos del Gemma pueden recibir cualquier acción utilizable en el Grafcet. Como se trata de definir una estructura de gestión de los modos de parada y de marcha, es conveniente utilizar órdenes de forzado hacia Grafcets de nivel más bajo; ver el capítulo 1.6.7. Forzados de Grafcet.

1.7.3. Obtener un Grafcet correspondiente

Una vez más, la opción « Mostrar bajo la forma de un Gemma » en las propiedades del folio permite volver a una representación Grafcet. Es posible volver en cualquier momento a una representación Gemma mientras la estructura del Grafcet no se modifique. Las transiciones, el contenido de los rectángulos de acción y los comentarios pueden modificarse con una actualización automática del Gemma.

1.7.4. Anular los espacios vacíos en el Grafcet

Es posible que el Grafcet obtenido ocupe más espacio del necesario en la página. El comando « Reorganizar la página » del menú « Herramientas » permite suprimir todos los espacios no utilizados.

1.7.5. Imprimir el Gemma

Cuando la edición está en modo Gemma, el comando « Imprimir » permite imprimir el Gemma.

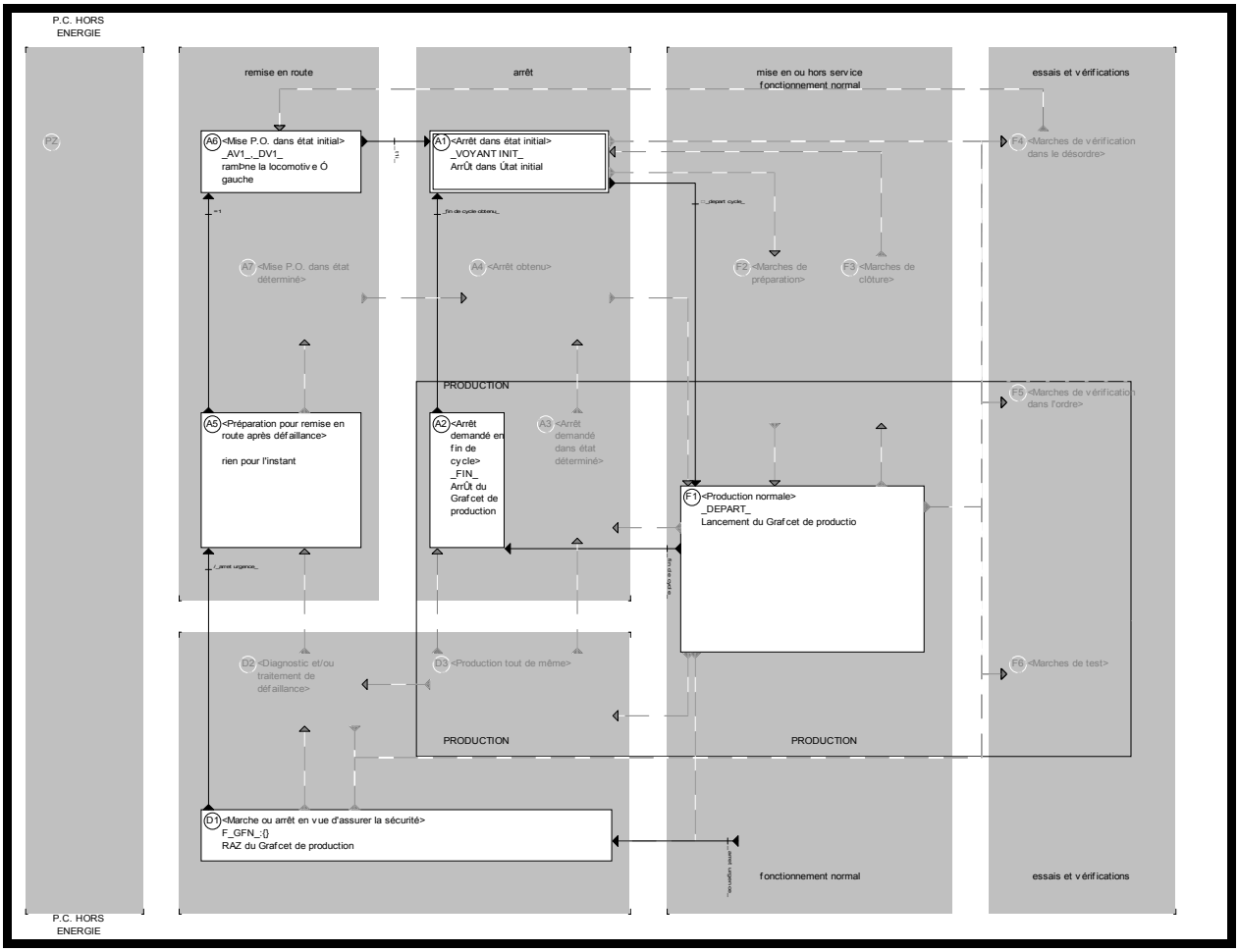
1.7.6. Exportar el Gemma

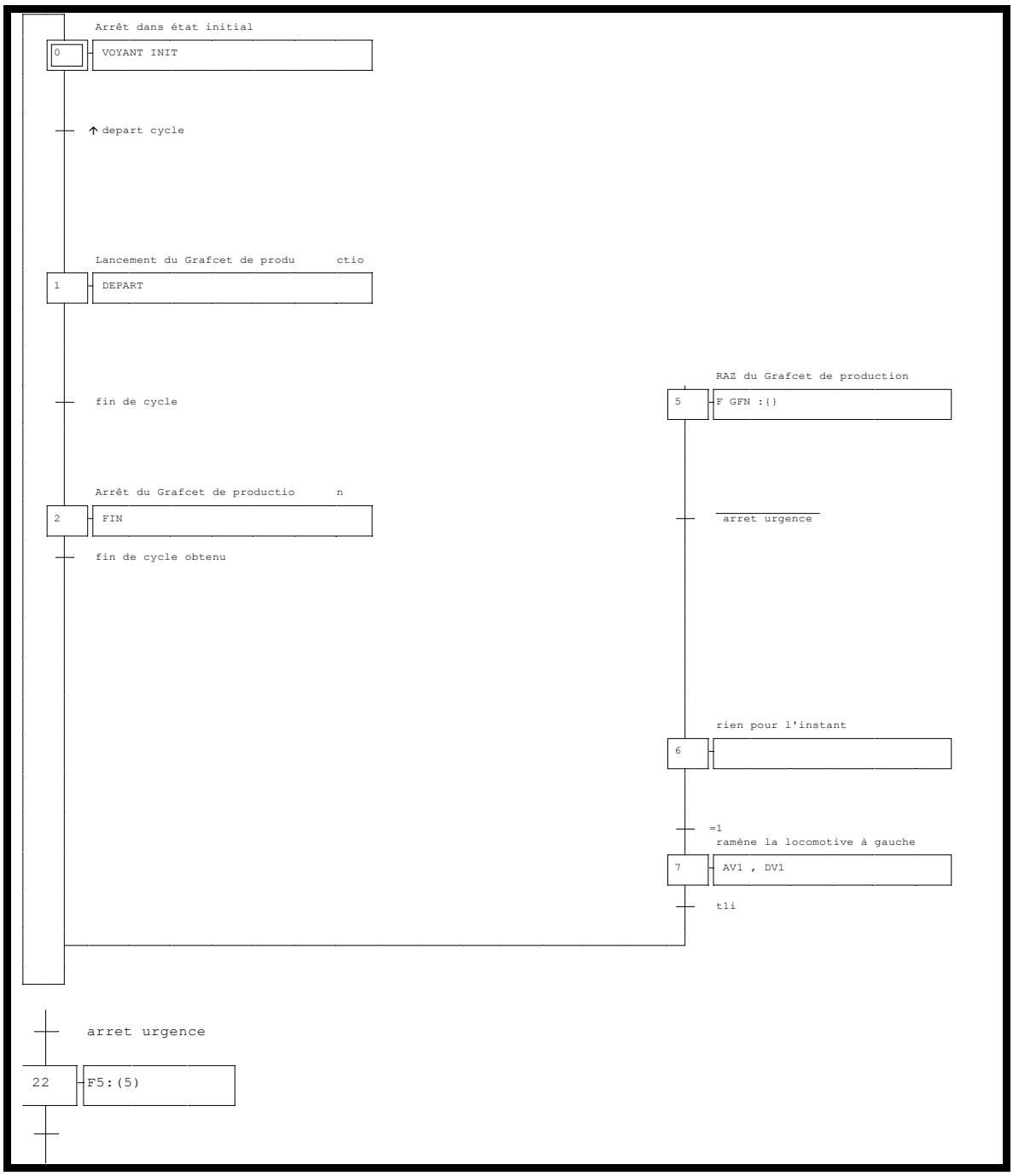
El comando « Copiar en formato EMF » del menú « Edición » permite exportar un Gemma en formato vectorial.

1.7.7. Ejemplo de Gemma

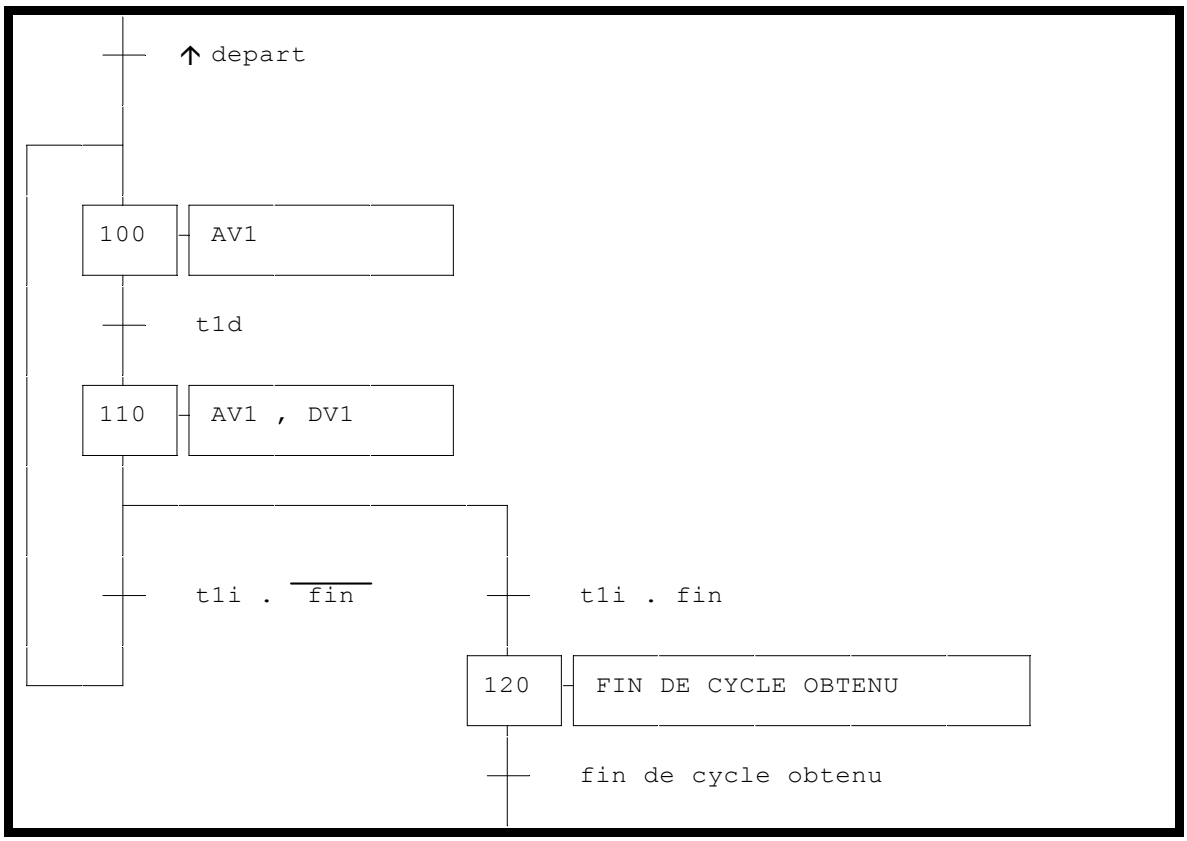
Ilustremos la utilización del Gemma.
Condiciones:
Imaginemos un escritorio compuesto por los pulsadores « inicio ciclo », « fin de ciclo » y « parada de urgencia » y el testigo « INIT ».
El propósito del programa principal será hacer efectuar idas y vueltas a una locomotora por la vía 1.

Solución:


P.C. HORS ENERGIE
remise en route
arrêt
mise en ou hors service fonctionnement normal
essais et vérifications
P2
A6 <Mise P.O. dans état initial> _AV1_..._DV1_ ramÞne la locomotive Ó gauche
A1 <Arrêt dans état initial> _VOYANT INIT_ ArrÛt dans Ûtat initial
F4 <Marches de vérification dans le désordre>
A7 <Mise P.O. dans état déterminé>
A4 <Arrêt obtenu>
F2 <Marches de préparation>
F3 <Marches de clôture>
PRODUCTION
F5 <Marches de vérification dans l'ordre>
A5 <Préparation pour remise en route après défaillance> rien pour l'instant
A2 <Arrêt demandé en fin de cycle> _FIN_ ArrÛt du Grafcet de production
A3 <Arrêt demandé dans état déterminé>
F1 <Production normale> _DEPART_ Lancement du Grafcet de productio
D2 <Diagnostic et/ou traitement de défaillance>
D3 <Production tout de même>
F6 <Marches de test>
PRODUCTION
PRODUCTION
D1 <Marche ou arrêt en vue d'assurer la sécurité> F_GFN_:{} RAZ du Grafcet de production
fonctionnement normal
essais et vérifications
P.C. HORS ENERGIE



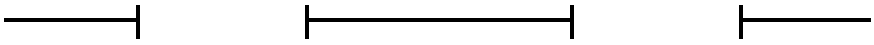
(editado bajo la forma de un Grafcet)



exemple\gemma\gemma.agn

1.8. Ladder

El lenguaje Ladder, también llamado « esquema de contacto », permite describir gráficamente ecuaciones booleanas. Para realizar una función lógica « Y », es necesario escribir contactos en serie. Para realizar una función « O », es necesario escribir contactos en paralelo.



Función « Y »

Función « O »

El contenido de los contactos debe respetar la sintaxis definida para los tests y detallada en el capítulo « Elementos comunes » de este manual.

El contenido de las bobinas debe respetar la sintaxis definida para las acciones y detallada en el capítulo « Elementos comunes » de este manual.

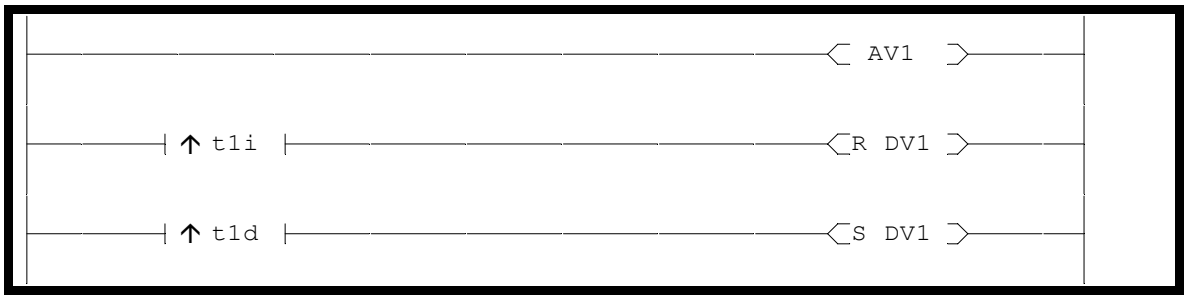
1.8.1. Ejemplo de Ladder

Empecemos con el ejemplo más simple.

Condiciones:

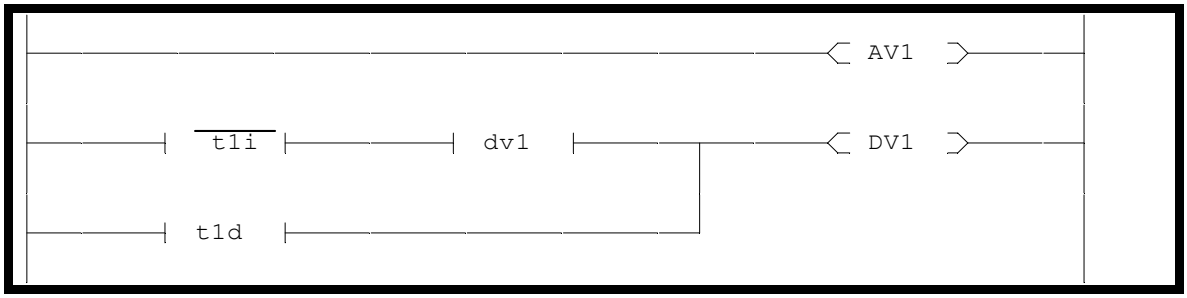
Ida y vuelta de una locomotora por la vía 1.

Solución 1:



exemple\ladder\ladder1.agn

Solución 2:



exemple\ladder\ladder2.agn

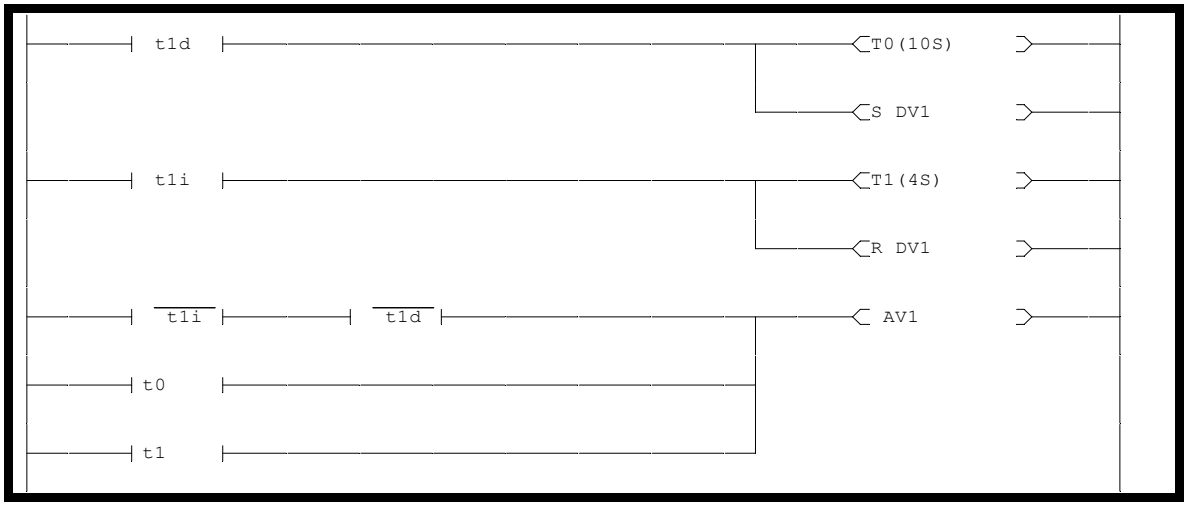
La segunda solución es idéntica desde el punto de vista funcional. Su interés es mostrar la utilización de una variable en auto mantenimiento.

Enriquezcamos nuestro ejemplo.

Condiciones:

La locomotora deberá detenerse 10 segundos a la derecha de la vía 1 y 4 segundos a la izquierda.

Solución:



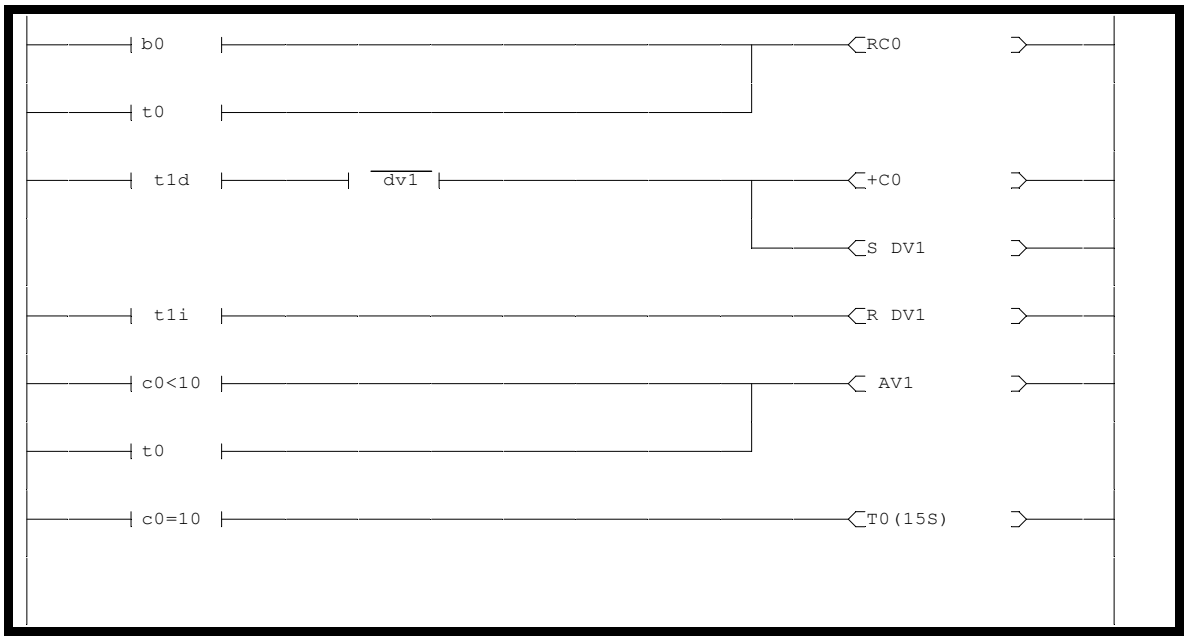
exemple\ladder\ladder3.agn

Un último ejemplo un poco más complejo.

Condiciones:

Siempre una locomotora que va y viene por la vía 1. Cada 10 idas y vueltas deberá marcar un tiempo de parada de 15 segundos.

Solución:



exemple\ladder\ladder4.agn

1.9. Logigrama

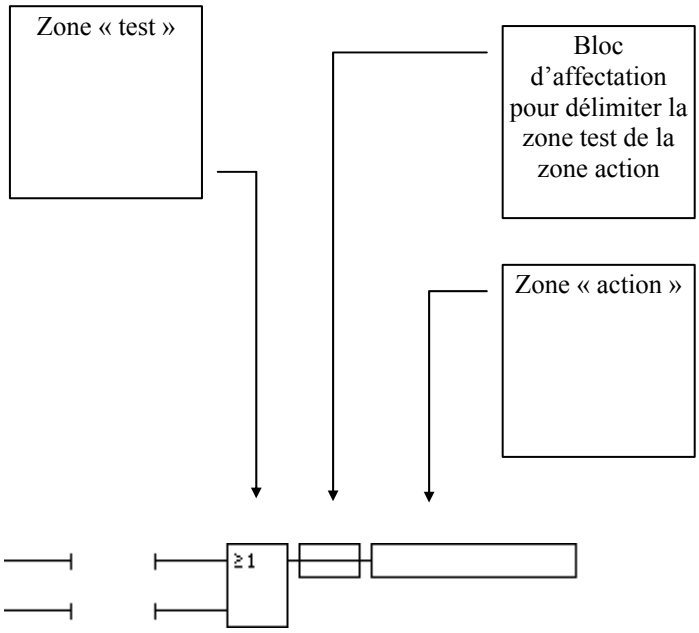
AUTOMGEN implementa el lenguaje logigrama de la siguiente manera:

⇨ utilización de un bloque especial llamado « bloque de asignación »; este bloque separa la zona de acción de la zona test, tiene esta forma y está asociado a la tecla [0] (cero),

⇨ utilización de las funciones « No », « Y » y « O »,

⇨ utilización de rectángulos de acción a la derecha del bloque de acción.

El lenguaje logigrama permite escribir gráficamente ecuaciones booleanas.
El contenido de los tests debe respetar la sintaxis definida en el capítulo « Elementos comunes » de este manual.
El contenido de los rectángulos de acción debe respetar la sintaxis definida para las acciones y detallada en el capítulo « Elementos comunes » de este manual.



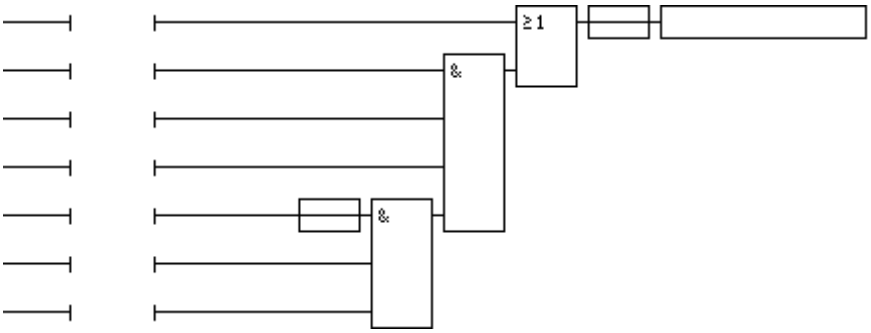
1.9.1. Diseño de los logigramas

1.9.1.1. Número de entradas de las funciones « Y » y « O »

Las funciones « Y » y « O » se componen respectivamente de un bloque (tecla [2]) o de un bloque (tecla [3]), eventualmente de bloques (tecla [4]) para añadir entradas a los bloques, y por último de un bloque (tecla [5]).
Las funciones « Y » y « O » implican pues un mínimo de dos entradas.

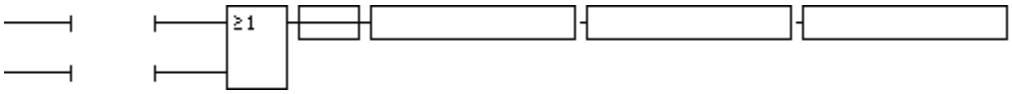
1.9.1.2. Encadenamiento de las funciones

Las funciones pueden encadenarse.

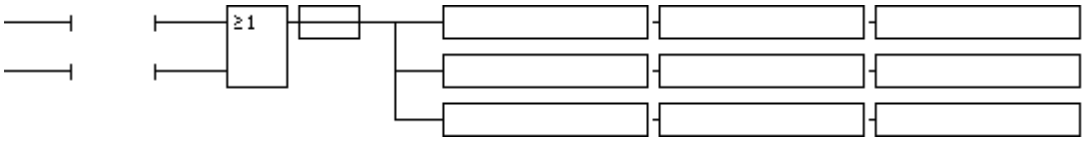


1.9.1.3. Acciones múltiples

Es posible asociar varios rectángulos de acción a un logigrama después del bloque de asignación.



o



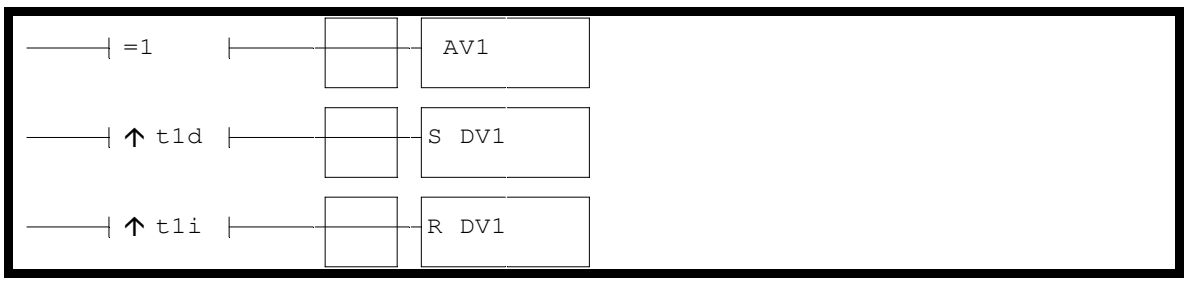
1.9.2. Ejemplo de logigrama

Empecemos con el ejemplo más simple.

Condiciones:

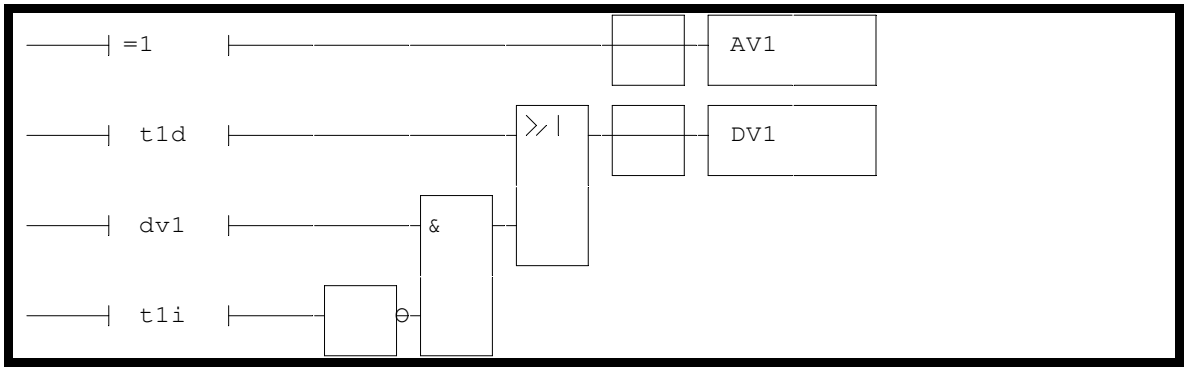
Ida y vuelta de una locomotora por la vía 1.

Solución 1:



exemple\logigramme\logigramme1.agn

Solución 2:



exemple\logigramme\logigramme2.agn

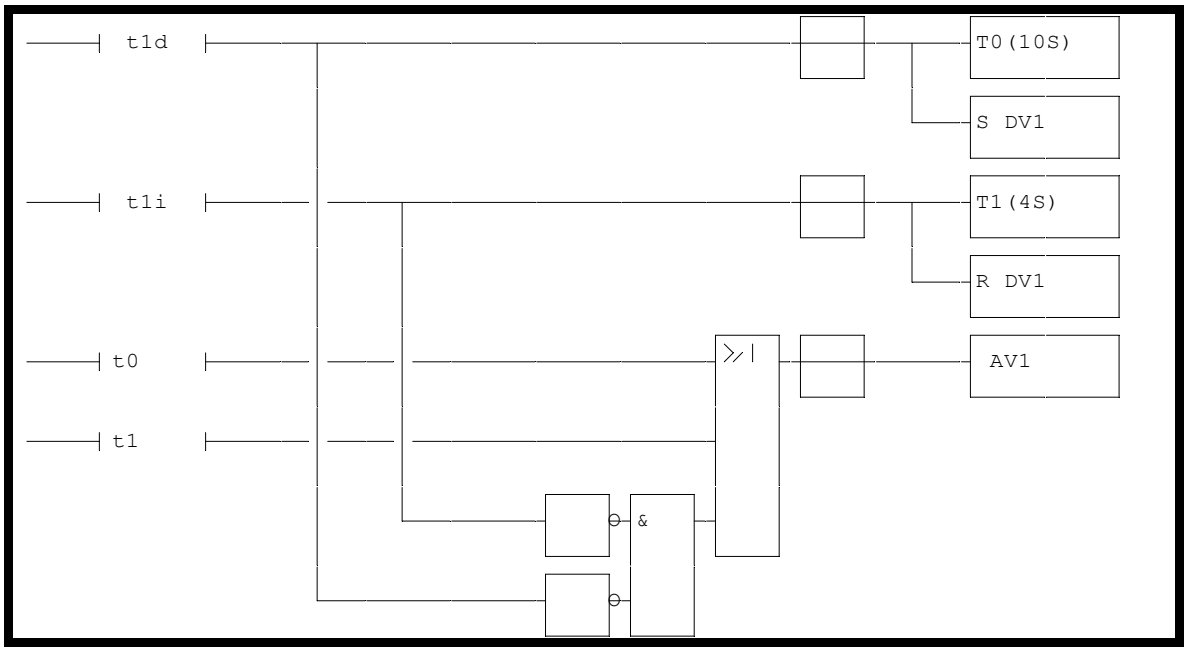
La segunda solución es idéntica desde el punto de vista funcional. Su interés es consiste en mostrar la utilización de una variable en auto mantenimiento.

Enriquezcamos nuestro ejemplo.

Condiciones:

La locomotora deberá detenerse 10 segundos a la derecha de la vía 1 y 4 segundos a la izquierda.

Solución:



exemple\logigramme\logigramme3.agn

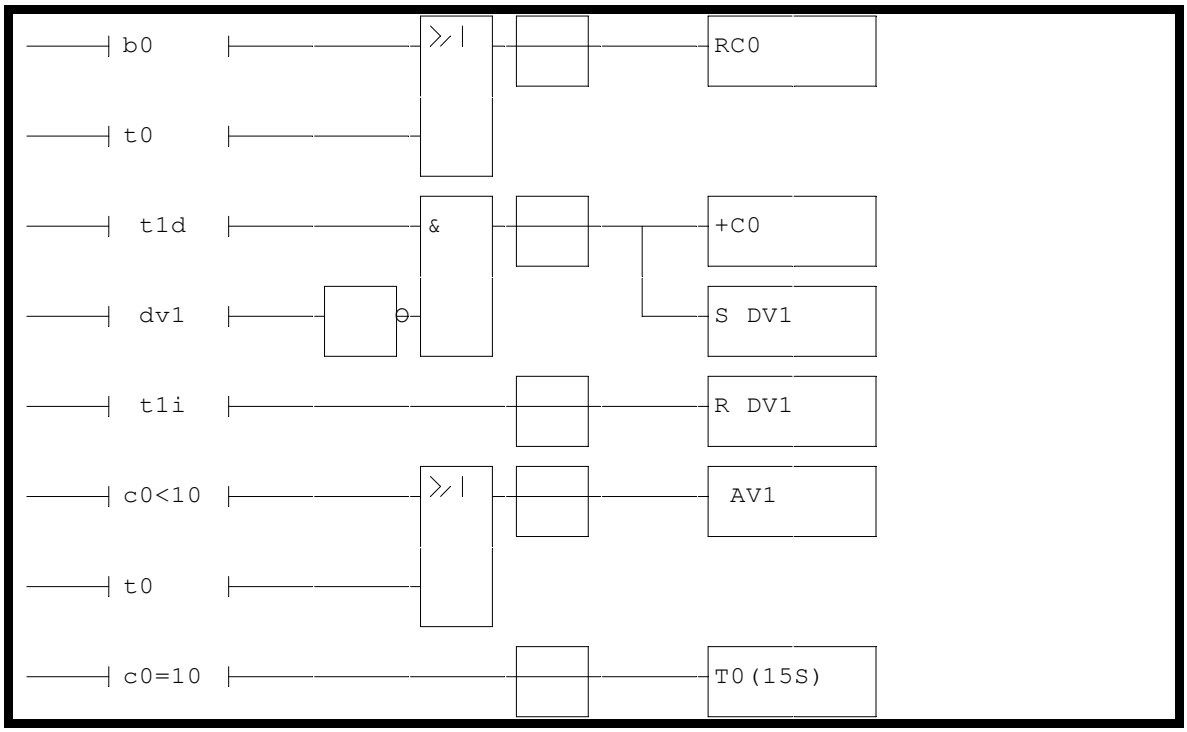
Cabe notar la repetición del bloque « Y » de la parte inferior del ejemplo en las entradas « _t1d_ » y « _t1i_ ». Esto evita tener que escribir por segunda vez estos dos tests.

Un último ejemplo un poco más complejo.

Condiciones:

Siempre una locomotora que va y viene por la vía 1. Cada 10 idas y vueltas deberá marcar un tiempo de parada de 15 segundos.

Solución:



exemple\logigramme\logigramme4.agn

1.10. Lenguajes literales

Este capítulo describe la utilización de las tres formas de lenguaje literal disponibles en AUTOMGEN:

⇨ lenguaje literal bajo nivel,

⇨ lenguaje literal extendido,

⇨ lenguaje literal ST de la norma CEI 1131-3.

1.10.1. Cómo utilizar el lenguaje literal

El lenguaje literal puede utilizarse bajo la forma siguiente:

⇨ archivo de código asociado a una acción (Grafcet, Ladder, logigrama),

⇨ caja de código asociada a una acción (Grafcet, logigrama),

⇨ código literal en un rectángulo de acción o una bobina (Grafcet, Ladder, logigrama),

⇨ caja de código utilizada bajo forma de organigrama (consultar el capítulo « Organigrama »),

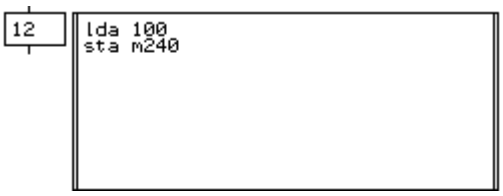
⇨ archivo de código que rige el funcionamiento de un bloque funcional (consultar el capítulo « Bloques funcionales »),

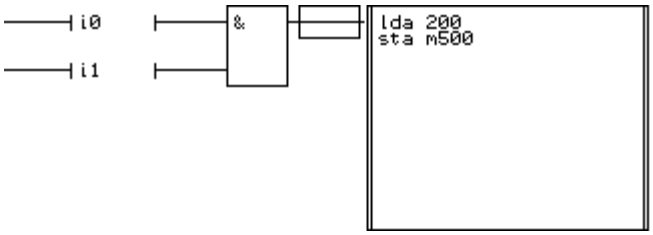
⇨ archivo de código que rige el funcionamiento de una macro-instrucción; ver el capítulo 1.10.4. Macro-instrucción.

1.10.1.1. Caja de código asociada a una etapa o un logigrama

Una caja de código asociada a una acción permite escribir algunas líneas de lenguaje literal en una página de la aplicación.

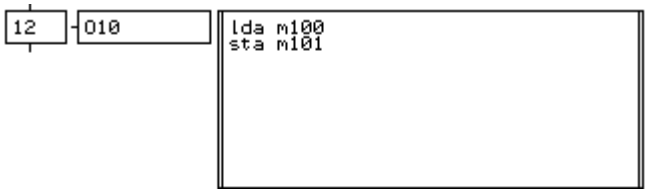
Ejemplos:





El código utilizado es escrutado mientras la acción es verdadera.
Es posible utilizar conjuntamente rectángulos de acción y cajas de código.

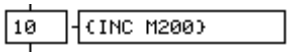
Ejemplo:

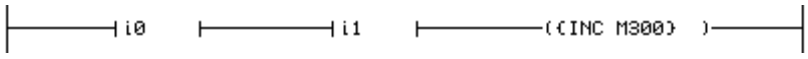


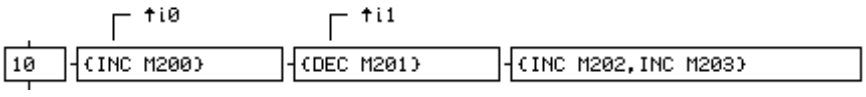
1.10.1.2. Código literal en un rectángulo de acción o una bobina

Los caracteres « { » y « } » permiten insertar instrucciones en lenguaje literal directamente en un rectángulo de acción (lenguajes Grafcet y logigrama) o una bobina (lenguaje ladder). El carácter « , » (coma) se utiliza como separador si entre « { » y « } » hay varias instrucciones.
Este tipo de inserción puede utilizarse con órdenes condicionadas.

Ejemplos:







1.10.2. Definición de una caja de código

Para diseñar una caja de código siga las etapas siguientes:

⇨ haga clic con el botón derecho del ratón en un sitio vacío del folio,

⇨ elija en el menú « Más … / Caja de código »,

⇨ haga clic sobre el borde de la caja de código para modificar su contenido.

Para salir de la caja después de modificarla utilice la tecla [Enter] o haga clic en el exterior.

1.10.3. El lenguaje literal bajo nivel

Este capítulo detalla la utilización del lenguaje literal bajo nivel. Este lenguaje es un código intermedio entre los lenguajes evolucionados Grafcet, logigrama, ladder, organigrama, bloques funcionales, literal extendido, literal ST y los lenguajes ejecutables. También se conoce con el nombre de código pivote. Los post-procesadores traducen el lenguaje literal bajo nivel en código ejecutable para PC, autómata o tarjeta con microprocesador.
El lenguaje literal también puede utilizarse en una aplicación para efectuar diversos tratamientos booleanos, numéricos o algorítmicos.
El lenguaje literal bajo nivel es un lenguaje de tipo ensamblador. Utiliza una noción de acumulador para los tratamientos numéricos.
El lenguaje literal extendido y el lenguaje literal ST descritos en los capítulos siguientes ofrecen una alternativa simplificada y de más alto nivel para escribir programas en lenguaje literal.
Sintaxis general de una línea de lenguaje literal bajo nivel:
«acción » [[[« Test »] « Test »]...]
Las acciones y los tests del lenguaje literal bajo nivel están representados por mnemónicos compuestos de tres caracteres alfabéticos. Cada instrucción va seguida de una expresión: variable, constante, etc...
Una línea se compone de una sola acción y eventualmente de un test. Si una línea contiene únicamente una acción, la instrucción se ejecutará siempre, por convención.

1.10.3.1. Variables

Las variables utilizables son las mismas que las definidas en el capítulo « Elementos comunes ».

1.10.3.2. Acumuladores

Ciertas instrucciones utilizan la noción de acumulador. Los acumuladores son registros internos del sistema que ejecuta el programa final y permiten acumular valores temporalmente.

Existen tres acumuladores: un acumulador entero 16 bits llamado AAA, un acumulador entero 32 bits llamado AAL y un acumulador flotante llamado AAF.

1.10.3.3. Banderas

Las banderas son variables booleanas posicionadas en función del resultado de las operaciones numéricas.

Las cuatro banderas que permiten testear el resultado de un cálculo son:

- ⇨ indicador de retención C: indica si una operación ha generado una retención (1) o no (0),
- ⇨ indicador de cero Z: indica si una operación ha generado un resultado nulo (1) o no (0),
- ⇨ indicador de signo S: indica si una operación ha generado un resultado negativo (1) o positivo (0),
- ⇨ indicador de exceso O: indica si una operación ha generado un exceso de capacidad (1).

1.10.3.4. Modos de direccionamiento

El lenguaje literal bajo nivel posee 5 modos de direccionamiento. Un modo de direccionamiento es una característica asociada a cada una de las instrucciones del lenguaje literal.

Los modos de direccionamiento utilizables son:

TIPO	SINTAXIS	EJEMPLO
Inmediato 16 bits	{constante}	100
Inmediato 32 bits	{constante}L	100000L
Inmediato flotante	{constante}R	3.14R
Absoluto	{variable} {identificación de variable}	O540
Acumulador 16 bits	AAA	AAA
Acumulador 32 bits	AAL	AAL

Acumulador flotante	AAF	AAF
Indirecto	{variable}{(identificación de palabra)}	O(220)
Label	:{nombre de label}:	:bucle:

Una instrucción posee dos características: el tipo de variable y el modo de direccionamiento. Una instrucción puede soportar o no ciertos modos de direccionamiento o ciertos tipos de variables. Por ejemplo, hay instrucciones que se aplican únicamente a palabras, y no a otros tipos de variables.

Observación: Las variables X y U no pueden asociarse a un direccionamiento indirecto debido a la no linealidad de sus asignaciones. Si es necesario acceder a una tabla de variables U, habrá que utilizar una directiva de compilación #B para declarar una tabla de bits lineales.

1.10.3.5. Tests

Los tests que pueden asociarse a las instrucciones se componen de un mnemónico, un tipo de test y una variable.
Los mnemónicos de tests permiten definir tests combinatorios de varias variables (y, o). Si un test está compuesto por una sola variable, de todas maneras debe asociarse un operador AND.
Existen solamente tres mnemónicos de test:
AND y
ORR o
EOR fin de o
He aquí algunos ejemplos de equivalencias entre ecuaciones booleanas y lenguaje literal bajo nivel:

```
o0=i1                          : and i1
o0=i1.i2                       : and i1 and i2
o0=i1+i2                       : orr i1 eor i2
o0=i1+i2+i3+i4                 : orr i1 orr i2 orr i3 eor i4
o0=(i1+i2).(i3+i4)             : orr i1 eor i2 orr i3 eor i4
o0=i1.(i2+i3+i4)               : and i1 orr i2 orr i3 eor i4
o0=(i1.i2)+(i3.i4)             ; imposible traducir directamente,
                               ; hay que utilizar
                               ; variables intermedias:
```

```
equ u100 and i1 and i2
equ u101 and i3 and i4
equ o0 orr u100 eor u101
```

Los modificadores de tests permiten testear cosas que no son simplemente el estado verdadero de una variable:

⇨ / no

⇨ # frente ascendente

⇨ * frente descendente

⇨ @ estado inmediato

Observaciones:

⇨ las variables booleanas se actualizan después de cada ciclo de ejecución. En otros términos, si durante un ciclo una variable binaria se posiciona en un estado, su nuevo estado será realmente reconocido en el ciclo siguiente. El modificador de test @ permite obtener el estado real de una variable booleana sin esperar el ciclo siguiente.

⇨ los modificadores de tests no son utilizables con los tests numéricos.

Ejemplos:

```
set o100
equ o0 and @o100          ; test verdadero desde el primer ciclo
equ o1 and o100           ; test verdadero en el segundo ciclo
```

Para los tests, hay sólo dos modos de direccionamiento disponibles: el modo absoluto y el modo indirecto.
Hay un test de contadores, palabras, largos y flotantes:

Sintaxis:

```
« {variable} {=, !, <, >, << , >>} {constante o variable} »
```

= significa igual,
! significa diferente,
< significa menor sin signo,
> significa mayor sin signo,

<< significa menor con signo,

>> significa mayor con signo,

Las constantes están predeterminadas en decimal. Los sufijos « $ » y « % » permiten escribirlas en hexadecimal o en binario. Las comillas permiten escribirlas en ASCII.

Las constantes 32 bits deben estar seguidas del carácter « L ».

Las constantes reales deben estar seguidas del carácter « R ».

Una palabra o un contador puede compararse con una palabra, un contador o una constante 16 bits.

Un largo puede compararse con un largo o una constante 32 bits.

Un flotante puede compararse con un flotante o una constante real.

Ejemplos:

```
and c0>100 and m225=10
orr m200=m201 eor m202=m203 and f100=f101 and f200<f203
orr m200<<-100 eor m200>>200
and f200=3.14r
and l200=$12345678L
and m200=%1111111100000000
```

1.10.3.6. Comentarios

Los comentarios deben empezar con el carácter « ; » (punto y coma); todos los caracteres siguientes se ignoran.

1.10.3.7. Base de numeración

Los valores (identificaciones de variables o constantes) pueden escribirse en decimal, hexadecimal, binario o ASCII.

La siguiente sintaxis debe aplicarse para las constantes 16 bits:

⇨ decimal: eventualmente el carácter « - » y luego 1 a 5 dígitos « 0123456789 »,

⇨ hexadecimal: prefijo « $ » o « 16# » seguido de 1 a 4 dígitos « 0123456789ABCDEF »,

⇨ binario: prefijo « % » o « 2# » seguido de 1 a 16 dígitos « 01 »,

⇨ ASCII: carácter « " » seguido de 1 o 2 caracteres seguidos de « " ».

La siguiente sintaxis debe aplicarse para las constantes 32 bits:

⇨ Decimal: eventualmente el carácter « - » y luego 1 a 10 dígitos « 0123456789 »,

⇨ Hexadecimal: prefijo « $ » o « 16# » seguido de 1 a 8 dígitos « 0123456789ABCDEF »,

⇨ Binario: prefijo « % » o « 2# » seguido de 1 a 32 dígitos « 01 »,

⇨ ASCII: carácter « " » seguido de 1 a 4 caracteres seguidos de « " ».

La siguiente sintaxis debe aplicarse para las constantes reales:
[-] i [[.d] Esx]
i es la parte entera
d una eventual parte decimal
s eventualmente el signo del exponente
x eventualmente el exponente

1.10.3.8. Predisposiciones

Una predisposición permite fijar el valor de una variable al inicio de la aplicación.
Las variables T o %T, M o %MW, L o %MD y F o %F pueden estar predispuestas.

La sintaxis es la siguiente:

```
« $(variable)=constante{,constante{,constante...}} »
```

Para las temporizaciones el valor debe estar escrito en decimal y comprendido entre 0 y 65535.
Para las palabras debe utilizarse la siguiente sintaxis:

⇨ Decimal: eventualmente el carácter « - » y luego 1 a 5 dígitos « 0123456789 »,

⇨ Hexadecimal: prefijo « $ » o « 16# » seguido de 1 a 4 dígitos « 0123456789ABCDEF »,

⇨ Binario: prefijo « % » o « 2# » seguido de 1 a 16 dígitos « 01 »,

⇨ ASCII: (dos caracteres por palabra) el carácter « " » seguido de n caracteres seguidos de « " »,

⇨ ASCII: (un carácter por palabra) el carácter « ’ » seguido de n caracteres seguidos de « ’ ».

Para los largos debe utilizarse la siguiente sintaxis:

⇨ Decimal: eventualmente el carácter « - » y luego 1 a 10 dígitos « 0123456789 »,

⇨ Hexadecimal: prefijo « $ » o « 16# » seguido de 1 a 8 dígitos « 0123456789ABCDEF »,

⇨ Binario: carácter « % » o « 2# » seguido de 1 a 32 dígitos « 01 »,

⇨ ASCII: (cuatro caracteres por largo) carácter « " » seguido de n caracteres seguidos de « " »,

⇨ ASCII: (un carácter por largo) carácter « ’ » seguido de n caracteres seguidos de « ’ »

Para los flotantes, el valor debe escribirse de la siguiente manera:
[-] i [[.d] Esx]
i es la parte entera
d una eventual parte decimal
s eventualmente el signo del exponente
x eventualmente el exponente

Ejemplos:

```
$t25=100
```

fija la consigna de la temporización 25 en 10 s

```
$MW200=100,200,300,400
```

coloca los valores 100, 200, 300, 400 en las palabras 200, 201, 202, 203

```
$m200="ABCDEF"
```

coloca la cadena de caracteres « ABCDEF » a partir de m200 (« AB » en m200, « CD » en m201, « EF » en m202)

```
$m200='ABCDEF'
```

coloca la cadena de caracteres « ABCDEF » a partir de m200; cada palabra recibe un carácter.

```
$f1000=3.14
```

coloca el valor 3,14 en f1000

```
$%mf100=5.1E-15
```

coloca el valor 5,1 * 10 exponente -15 en %mf100

```
$l200=16#12345678
```

coloca el valor 12345678 (hexa) en el largo l200

Es más fácil escribir texto en las predisposiciones.

Ejemplo:

```
$m200=" Parada compuerta N°10 "
```

Coloca el mensaje a partir de la palabra 200 con dos caracteres en cada palabra.

```
$m400=‘ Defecto motor ‘
```

Coloca el mensaje a partir de la palabra 400 con un carácter en el byte de peso débil de cada palabra; el byte de peso fuerte contiene 0.

La sintaxis « $...= » permite proseguir una tabla de predisposiciones a continuación de la anterior.

Por ejemplo:

```
#$m200=1,2,3,4,5
#$...=6,7,8,9
```

Coloca los valores de 1 a 9 en las palabras m200 a m208.

Las predisposiciones pueden escribirse de la misma manera que el lenguaje literal bajo nivel o en una directiva de compilación en un folio. En este caso, la predisposición empieza con el carácter « # ».

Ejemplo de predisposición escrita en una caja de código:

Ejemplo de predisposición escrita en una directiva de compilación:

```
#$m200=12,13
```

1.10.3.9. Direccionamiento indirecto

El direccionamiento indirecto permite efectuar una operación con una variable señalada por un índice.

Son las variables M (las palabras) las que sirven de índice.

Sintaxis:

« variable (índice) »

Ejemplo:

```
lda 10       ; carga 10 en el acumulador
sta m200     ; lo coloca en la palabra 200
set o(200)   ; puesta en uno de la salida señalada por la palabra 200 (o10)
```

1.10.3.10. Direccionamiento de una variable

El carácter « ? » permite especificar el direccionamiento de una variable.

Ejemplo:

```
lda ?o10            ; coloca el valor 10 en el acumulador
```

Esta sintaxis es interesante sobre todo si se utilizan símbolos.

Ejemplo:

```
lda ?_compuerta_    ; coloca en el acumulador el número de la variable
                    ; asociada al símbolo « _compuerta_ »
```

Esta sintaxis también puede utilizarse en las predisposiciones para crear tablas de direccionamientos de variables.

Ejemplo:

```
$m200=?_compuerta1_,?_compuerta2_,?_compuerta3_
```

1.10.3.11. Saltos y labels

Los saltos deben hacer referencia a un label. La sintaxis de un label es:

«:nombre del label: »

Ejemplo:

```
jmp:continuación:
...
:continuación:
```

1.10.3.12. Lista de funciones por tipo

1.10.3.12.1. Funciones booleanas

SET	puesta en uno
RES	puesta en cero
INV	inversión
EQU	equivalencia
NEQ	no-equivalencia

1.10.3.12.2. Funciones de carga y almacenamiento con enteros y flotantes

LDA	carga
STA	almacenamiento

1.10.3.12.3. Funciones aritméticas con enteros y flotantes

ADA	suma
SBA	resta
MLA	multiplicación
DVA	división
CPA	comparación

1.10.3.12.4. Funciones aritméticas con flotantes

ABS	valor absoluto
SQR	raíz cuadrada

1.10.3.12.5. Funciones de acceso a los puertos de entrada/salida en PC

AIN	lectura de un puerto
AOU	escritura de un puerto

1.10.3.12.6. Funciones de acceso a la memoria en PC

ATM	lectura de una dirección de memoria
MTA	escritura de una dirección de memoria

1.10.3.12.7. Funciones binarias con enteros

ANA	y bit a bit
ORA	o bit a bit
XRA	o exclusivo bit a bit
TSA	test bit a bit

SET	puesta en uno de todos los bits
RES	puesta en cero de todos los bits
RRA	desfase a derecha
RLA	desfase a izquierda

1.10.3.12.8. Otras funciones con enteros

INC	incremento
DEC	decremento

1.10.3.12.9. Funciones de conversión

ATB	entero a booleanos
BTA	booleanos a entero
FTI	flotante a entero
ITF	entero a flotante
LTI	entero 32 bits a entero 16 bits
ITL	entero 16 bits a entero 32 bits

1.10.3.12.10. Funciones de ramificación

JMP	salto
JSR	salto a un subprograma
RET	retorno de subprograma

1.10.3.12.11. Funciones de test

RFZ	flag de resultado nulo
RFS	flag de signo
RFO	flag de exceso
RFC	flag de retención

1.10.3.12.12. Funciones de accesos asíncronos a entradas salidas

RIN	lectura de entradas
WOU	escritura de salidas

1.10.3.12.13. Información contenida en la lista de funciones

Para cada instrucción se dan:

⇨ Nombre: el mnemónico.

⇨ Función: una descripción de la función realizada por la instrucción.

⇨ Variables: tipos de variables utilizables con la instrucción.

⇨ Direccionamiento: tipos de direccionamientos utilizables.

⇨ Ver también: las otras instrucciones que tengan relación con este mnemónico.

⇨ Ejemplo: un ejemplo de utilización.

Los post-procesadores que generan lenguajes constructores están sujetos a ciertas restricciones. Consultar las notas sobre estos post-procesadores para conocer en detalle dichas restricciones.

ABS

Nombre	:	ABS - abs accumulator
Función	:	calcula el valor absoluto del acumulador flotante
Variables	:	ninguna
Direccionamiento	:	acumulador
Ver también	:	SQR
Ejemplo	:	

```
lda f200
abs aaf
sta f201
; deja en f201 el valor absoluto de f200
```


ADA

Nombre	:	ADA - adds accumulator
Función	:	añade un valor al acumulador
Variables	:	M o %MW, L o %MD, F o %MF
Direccionamiento	:	absoluto, indirecto, inmediato
Ver también	:	SBA
Ejemplo	:	

```
ada 200
; añade 200 al acumulador 16 bits

ada f124
; añade el contenido de f124 al acumulador flotante

ada l200
; añade el contenido de l200 al acumulador 32 bits

ada 200L
; añade 200 al acumulador 32 bits

ada 3.14R
; añade 3.14 al acumulador flotante
```

AIN

Nombre	:	AIN - accumulator input
Función	:	lectura de un puerto de entrada (8 bits) y almacenamiento en la parte baja del acumulador 16 bits;
		 lectura de un puerto de entrada 16 bits y almacenamiento en el acumulador 16 bits (en este caso la dirección del puerto debe escribirse como una constante de 32 bits) utilizable sólo con el ejecutor PC
Variables	:	M o %MW
Direccionamiento:		indirecto, inmediato
Ver también	:	AOU
Ejemplo	:	

```
ain $3f8
; lectura del puerto $3f8 (8 bits)

ain $3f8l
; lectura del puerto $3f8 (16 bits)
```

ANA

Nombre	:	ANA - and accumulator
Función	:	efectúa un Y lógico entre el acumulador 16 bits y una palabra o una constante o el acumulador 32 bits y un largo o una constante
Variables	:	M o %MW, L o %MD
Direccionamiento:		absoluto, indirecto, inmediato
Ver también	:	ORA, XRA
Ejemplo	:	

```
ana %1111111100000000
; oculta los 8 bits de peso débil del
; acumulador 16 bits

ana $ffff0000L
; oculta los 16 bits de peso débil del acumulador 32 bits
```

AOU

Nombre : AOU - accumulator output

Función : transfiere la parte baja (8 bits) del contenido del acumulador 16 bits a un puerto de salida;

transfiere los 16 bits del acumulador 16 bits a un puerto de salida (en este caso la dirección del puerto debe escribirse como una constante 32 bits)
utilizable sólo con el ejecutor PC

Variables : M o %MW

Direccionamiento: indirecto, inmediato

Ver también : AIN

Ejemplo :

```
lda "A"
aou $3f8
; coloca el carácter « A » en el puerto de salida $3f8

lda $3f8
sta m200
lda "z"
aou m(200)
; coloca el carácter « z » en el puerto de salida $3f8

lda $1234
aou $300l
; coloca el valor 16 bits 1234 en el puerto de salida $300
```

ATB

Nombre	:	ATB - accumulator to bit
Función	:	transfiere los 16 bits del acumulador 16 bits a 16 variables booleanas sucesivas; el bit de peso débil corresponde a la primera variable booleana
Variables	:	I o %I, O o %Q, B o %M, T o %T, U*
Direccionamiento:		absoluto
Ver también	:	BTA
Ejemplo	:	

```
lda m200
atb o0
; recopia los 16 bits de m200 en las variables
; o0 a o15
```

* Atención: para poder utilizar los bits U con esta función es necesario realizar una tabla lineal de bits con la directiva de compilación #B.

ATM

Nombre	:	ATM - accumulator to memory
Función	:	transfiere el acumulador 16 bits a una dirección memoria; la palabra o la constante especificada define el offset de la dirección de memoria a alcanzar; la palabra m0 debe estar cargada con el valor del segmento de la dirección de memoria a alcanzar; utilizable sólo con el ejecutor PC
Variables	:	M o %MW
Direccionamiento:		indirecto, inmediato
Ver también	:	MTA
Ejemplo	:	

```
lda $b800
sta m0
lda 64258
atm $10
; coloca el valor 64258 en la dirección $b800:$0010
```

BTA

Nombre	:	BTA - bit to accumulator
Función	:	transfiere 16 variables booleanas sucesivas a los 16 bits del acumulador 16 bits; el bit de peso débil corresponde a la primera variable booleana
Variables	:	I o %I, O o %Q, B o %M, T o %T, U*
Direccionamiento:		absoluto
Ver también	:	ATB
Ejemplo	:	

```
bta i0
sta m200
; recopia las 16 entradas i0 a i15 en la palabra m200
```

* Atención: para poder utilizar los bits U con esta función es necesario realizar una tabla lineal de bits con la directiva de compilación #B.

CPA

Nombre	:	CPA - compares accumulator
Función	:	compara un valor al acumulador 16 bits o 32 bits o flotante; efectúa la misma operación que SBA pero sin modificar el contenido del acumulador
Variables	:	M o %MW, L o %MD, F o %MF
Direccionamiento:		absoluto, indirecto, inmediato
Ver también	:	SBA
Ejemplo	:	

```
lda m200
cpa 4
rfz o0
; pone o0 en 1 si m200 es igual a 4, si no o0
; se pone en 0

lda f200
cpa f201
rfz o1
; pone o1 en 1 si f200 es igual a f201, si no o1
; se pone en 0
```


DEC

Nombre	:	DEC – decrement
Función	:	decrementa una palabra, un contador, un largo, el acumulador 16 bits o 32 bits
Variables	:	M o %MW, C o %C, L o %MD
Direccionamiento:		absoluto, indirecto, acumulador
Ver también	:	INC
Ejemplo	:	

```
dec m200
; decrementa m200

dec aal
; decrementa el acumulador 32 bits

dec m200
dec m201 and m200=-1
; decrementa un valor 32 bits compuesto de
; m200 (peso débil)
; y m201 (peso fuerte)
```

DVA

Nombre	:	DVA - divides accumulator
Función	:	división del acumulador 16 bits por una palabra o una constante; división del acumulador flotante por un flotante o una constante; división del acumulador 32 bits por un largo o una constante; para el acumulador 16 bits el resto se coloca en la palabra m0; en caso de división por 0 el bit sistema 56 pasa a 1
Variables	:	M o %MW, L o %MD, F o %MF
Direccionamiento:		absoluto, indirecto, inmediato
Ver también	:	MLA
Ejemplo	:	

```
lda m200
dva 10
sta m201
; m201 es igual a m200 dividido por 10, m0 contiene el
; resto de la división

lda l200
dva $10000L
sta l201
```


EQU

Nombre	:	EQU - equal
Función	:	fuerza una variable a 1 si el test es verdadero, en caso contrario, la variable se fuerza a 0
Variables	:	I o %I, O o %Q, B o %M, T o %T, X o %X, U
Direccionamiento:		absoluto, indirecto (salvo con las variables X)
Ver también	:	NEQ, SET, RES, INV
Ejemplo	:	

```
equ o0 and i10
; fuerza la salida o0 al mismo estado que la entrada i10

lda 10
sta m200
equ o(200) and i0
; fuerza o10 al mismo estado que la entrada i0

$t0=100
equ t0 and i0
equ o0 and t0
; fuerza o0 al estado de i0 con un retraso en la activación
; de 10 segundos
```

FTI

Nombre	:	FTI - float to integer
Función	:	transfiere el acumulador flotante al acumulador 16 bits
Variables	:	ninguna
Direccionamiento:		acumulador
Ver también	:	ITF
Ejemplo	:	

```
lda f200
fti aaa
sta m1000
; deja la parte entera de f200 en m1000
```


INC

Nombre	:	INC - increment
Función	:	incrementa una palabra, un contador, un largo, el acumulador 16 bits o 32 bits
Variables	:	M o %MW, C o %C, L o %MD
Direccionamiento:		absoluto, indirecto, acumulador
Ver también	:	DEC
Ejemplo	:	

```
inc m200
; añade 1 a m200

inc m200
inc m201 and m201=0
; incrementa un valor sobre 32 bits, m200
; representa los
; pesos débiles, y m201 los pesos fuertes

inc l200
; incrementa el largo l200
```

INV

Nombre	:	INV - inverse
Función	:	invierte el estado de una variable booleana, o invierte todos los bits de una palabra, de un largo o del acumulador 16 bits o 32 bits
Variables	:	I o %I, O o %Q, B o %M, T o %T, X o %X, U, M o %MW, L o %MD
Direccionamiento:		absoluto, indirecto, acumulador
Ver también	:	EQU, NEQ, SET, RES
Ejemplo	:	

```
inv o0
; invierte el estado de la salida 0

inv aaa
; invierte todos los bits del acumulador 16 bits

inv m200 and i0
; invierte todos los bits de m200 si i0 está en estado 1
```


ITF

Nombre	:	ITF - integer to float
Función	:	transfiere el acumulador 16 bits al acumulador flotante
Variables	:	ninguna
Direccionamiento:		acumulador
Ver también	:	FTI
Ejemplo	:	

```
lda 1000
itf aaa
sta f200
; deja la constante 1000 en f200
```

ITL

Nombre	:	ITL - integer to long
Función	:	transfiere el acumulador 16 bits al acumulador 32 bits
Variables	:	ninguna
Direccionamiento:		acumulador
Ver también	:	LTI
Ejemplo	:	

```
lda 1000
itl aaa
sta f200
; deja la constante 1000 en l200
```

JMP

Nombre	:	JMP - jump
Función	:	salto a un label
Variables	:	label
Direccionamiento:		label
Ver también	:	JSR
Ejemplo	:	

```
jmp :fin de programa:
; ramificación incondicional en label :fin
; de programa:


jmp :continuación: and i0
set o0
set o1
:continuación:
; ramificación condicional en label :continuación:
; según el estado de i0
```

JSR

Nombre : JSR - jump sub routine
Función : efectúa una ramificación a un subprograma
Variables : label
Direccionamiento: label
Ver también : RET
Ejemplo :

```
        lda m200
        jsr :cuadrado:
        sta m201
        jmp :fin:

:cuadrado:
        sta m53
        mla m53
        sta m53
        ret m53

:fin:

; el subprograma « cuadrado » eleva al cuadrado
; el contenido del acumulador
```

LDA

Nombre	:	LDA - load accumulator
Función	:	carga en el acumulador 16 bits una constante, palabra o contador; carga en el acumulador 32 bits un largo o constante, carga en el acumulador flotante un flotante o una constante, carga un contador de temporización en el acumulador 16 bits
Variables	:	M o %MW, C o %C, L o %MD, F o %MF, T o %T
Direccionamiento:		absoluto, indirecto, inmediato
Ver también	:	STA
Ejemplo	:	

```
lda 200
; carga la constante 200 en el acumulador 16 bits

lda 0.01R
; carga la constante real 0.01 en el acumulador flotante

lda t10
; carga el contador de la temporización 10 en
; el acumulador
```

LTI

Nombre	:	LTI - long to integer
Función	:	transfiere el acumulador 32 bits al acumulador 16 bits
Variables	:	ninguna
Direccionamiento:		acumulador
Ver también	:	ITL
Ejemplo	:	

```
lda l200
lti aaa
sta m1000
; deja los 16 bits de peso débil de l200 en m1000
```

MLA

Nombre	:	MLA - multiples accumulator
Función	:	multiplicación del acumulador 16 bits por una palabra o constante ; multiplicación del acumulador 32 bits por un largo o constante, multiplicación del acumulador flotante por un flotante o constante; para el acumulador 16 bits, los 16 bits de peso fuerte del resultado de la multiplicación se transfieren a m0
Variables	:	M o %MW, L o %MD, F o %MF
Direccionamiento:		absoluto, indirecto, inmediato
Ver también	:	DVA
Ejemplo	:	

```
lda m200
mla 10
sta m201
; multiplica m200 por 10, m201 está cargado con los
; 16 bits de peso débil, y m0 con los 16 bits de
; peso fuerte
```

MTA

Nombre	:	MTA - memory to accumulator
Función	:	transfiere el contenido de una dirección de memoria al acumulador 16 bits; la palabra o constante especificada define el offset de la dirección de memoria a alcanzar, la palabra m0 debe estar cargada con el valor del segmento de la dirección de memoria a alcanzar; utilizable sólo con el ejecutor PC
Variables	:	M o %MW
Direccionamiento:		indirecto, inmediato
Ver también	:	ATM
Ejemplo	:	

```
lda $b800
sta m0
mta $10
; coloca el valor contenido en la dirección $b800:$0010
; en el acumulador 16 bits
```

NEQ

Nombre	:	NEQ - not equal
Función	:	fuerza una variable a 0 si el test es verdadero, en caso contrario, la variable se fuerza a 1
Variables	:	I o %I, O o %Q, B o %M, T o %T, X o %X, U
Direccionamiento:		absoluto, indirecto (salvo con las variables X)
Ver también	:	EQU, SET, RES, INV
Ejemplo	:	

```
neq o0 and i00
; fuerza la salida o0 al estado complementado de la entrada
; i10

lda 10
sta m200
neq o(200) and i0
; fuerza o10 al estado complementado de la entrada i0

$t0=100
neq t0 and i0
neq o0 and t0
; fuerza o0 al estado de i0 con un retraso en la
; desactivación de 10 segundos
```

ORA

Nombre	:	ORA - or accumulator
Función	:	efectúa un O lógico entre el acumulador 16 bits y una palabra o una constante, o entre el acumulador 32 bits y un largo o una constante
Variables	:	M o %M, L o %MD
Direccionamiento:		absoluto, indirecto, inmediato
Ver también	:	ANA, XRA
Ejemplo	:	

```
ora %1111111100000000
; fuerza los 8 bits de peso fuerte del
; acumulador 16 bits a 1

ora $ffff0000L
; fuerza los 16 bits de peso fuerte del acumulador 32 bits
; a 1
```

RES

Nombre	:	RES - reset
Función	:	fuerza una variable booleana, una palabra, un contador, un largo, el acumulador 16 bits o el acumulador 32 bits a 0
Variables	:	I o %I, O o %Q, B o %M, T o %T, X o %X, U, M o %MW, C o %C, L o %MD
Direccionamiento:		absoluto, indirecto (salvo con las variables X), acumulador
Ver también	:	NEQ, SET, EQU, INV
Ejemplo	:	

```
res o0
; fuerza la salida o0 a 0

lda 10
sta m200
res o(200) and i0
; fuerza o10 a 0 si la entrada i0 está en 1

res c0
; fuerza el contador 0 a 0
```

RET

Nombre	:	RET - return
Función	:	marca el retorno de un subprograma y coloca en el acumulador 16 bits una palabra o una constante; o coloca en el acumulador 32 bits un largo o una constante, o coloca en el acumulador flotante un flotante o una constante
Variables	:	M o %MW, L o %MD, F o %MF
Direccionamiento:		absoluto, indirecto, inmediato
Ver también	:	JSR
Ejemplo	:	

```
ret 0
; retorno de subprograma colocando 0 en
; el acumulador 16 bits

ret f200
; retorno de subprograma colocando el contenido
; de f200 en el acumulador flotante
```


RFC

Nombre	:	RFC - read flag: carry
Función	:	transfiere el contenido del indicador de retención a una variable booleana
Variables	:	I o %I, O o %Q, B o %M, T o %T, X o %X, U
Direccionamiento:		absoluto
Ver también	:	RFZ, RFS, RFO
Ejemplo	:	

```
rfc o0
; transfiere el indicador de retención a o0

lda m200
ada m300
sta m400
rfc b99
lda m201
ada m301
sta m401
inc m401 and b99
; efectúa una suma sobre 32 bits
; (m400,401)=(m200,201)+(m300,301)
; m200, m300 y m400 son los pesos débiles
; m201, m301 y m401 son los pesos fuertes
```

RFO

Nombre	:	RFO - read flag: overflow
Función	:	transfiere el contenido del indicador de exceso a una variable booleana
Variables	:	I o %I, O o %Q, B o %M, T o %T, X o %X, U
Direccionamiento:		absoluto
Ver también	:	RFZ, RFS, RFC
Ejemplo	:	

```
rfo o0
; transfiere el indicador de exceso a o0
```


RFS

Nombre	:	RFS - read flag: signe
Función	:	transfiere el contenido del indicador de signo a una variable booleana
Variables	:	I o %I, O o %Q, B o %M, T o %T, X o %X, U
Direccionamiento:		absoluto
Ver también	:	RFZ, RFC, RFO
Ejemplo	:	

```
rfs o0
; transfiere el indicador de signo a o0
```

RFZ

Nombre	:	RFZ - read flag: zero
Función	:	transfiere el contenido del indicador de resultado nulo a una variable booleana
Variables	:	I o %I, O o %Q, B o %M, T o %T, X o %X, U
Direccionamiento:		absoluto
Ver también	:	RFC, RFS, RFO
Ejemplo	:	

```
rfz o0
; transfiere el indicador de resultado nulo a o0

lda m200
cpa m201
rfz o0
; posiciona o0 en 1 si m200 es igual a m201
; o en 0 en caso contrario
```


RIN

Nombre	:	RIN - read input
Función	:	efectúa una lectura de las entradas físicas. Esta función se implementa únicamente en destinos Z y varía según el destino. Consultar la documentación relativa a cada ejecutor para más detalles.
Variables	:	ninguna
Direccionamiento:		inmediato
Ver también	:	WOU

RLA

Nombre : RLA - rotate left accumulator
Función : efectúa una rotación a la izquierda de los bits del acumulador 16 bits o 32 bits; el bit evacuado a la izquierda entra a la derecha; el argumento de esta función es una constante que precisa el número de desfases a efectuar; el tamaño del argumento (16 o 32 bits) determina cuál es el acumulador que debe sufrir la rotación

Variables : ninguna

Direccionamiento: inmediato

Ver también : RRA

Ejemplo :

```
ana $f000
; aísla el dígito de peso fuerte del acumulador 16 bits
rla 4
; y lo lleva a la derecha

rla 8L
; efectúa 8 rotaciones a la izquierda de los bits del
; acumulador 32 bits
```


RRA

Nombre	:	RRA - rotate right accumulator
Función	:	efectúa una rotación a la derecha de los bits del acumulador 16 bits o 32 bits; el bit evacuado a la derecha entra a la izquierda; el argumento de esta función es una constante que precisa el número de desfases a efectuar; el tamaño del argumento (16 o 32 bits) determina si es el acumulador 16 o 32 bits el que deberá sufrir la rotación
Variables	:	ninguna
Direccionamiento:		inmediato
Ver también	:	RLA
Ejemplo	:	

```
ana $f000
; aísla el dígito de peso fuerte del acumulador 16 bits
rra 12
; y lo lleva a la derecha.

rra 1L
; efectúa una rotación de los bits del acumulador 32 bits
; de una posición hacia la derecha
```

SBA

Nombre	:	SBA - substacts accumulator
Función	:	quita el contenido de una palabra o una constante al acumulador 16 bits; quita el contenido de un largo o una constante al acumulador 32 bits; quita el contenido de un flotante o una constante al acumulador flotante
Variables	:	M o %MW, L o %MD, F o %MF
Direccionamiento:		absoluto, indirecto, inmediato
Ver también	:	ADA
Ejemplo	:	

```
sba 200
; quita 200 al acumulador 16 bits

sba f(421)
; quita el contenido del flotante cuyo número está
; contenido en la palabra 421 al acumulador flotante
```

SET

Nombre	:	SET - set
Función	:	fuerza una variable booleana a 1; fuerza todos los bits de una palabra, contador, largo o del acumulador 16 bits o del acumulador 32 bits a 1
Variables	:	I o %I, O o %Q, B o %M, T o %T, X o %X, U, M o %MW, C o %C, L o %MD
Direccionamiento:		absoluto, indirecto (salvo con las variables X), acumulador
Ver también	:	NEQ, RES, EQU, INV
Ejemplo	:	

```
set o0
; fuerza la salida o0 a 1

lda 10
sta m200
set o(200) and i0
; fuerza o10 a 1 si la entrada i0 está en 1

set m200
; fuerza m200 al valor -1

set aal
; fuerza todos los bits del acumulador 32 bits a 1
```


SQR

Nombre	:	SQR - square root
Función	:	calcula la raíz cuadrada del acumulador flotante
Variables	:	ninguna
Direccionamiento:		acumulador
Ver también	:	ABS
Ejemplo	:	

```
lda 9
itf aaa
sqr aaf
fti aaa
sta m200
; deja el valor 3 en m200
```

STA

Nombre	:	STA - store accumulator
Función	:	almacena el acumulador 16 bits en un contador o una palabra; almacena el acumulador 32 bits en un largo, almacena el acumulador flotante en un flotante, almacena el acumulador 16 bits en una consigna de temporización
Variables	:	M o %MW, C o %C, L o %MD, F o %MF, T o %T
Direccionamiento:		absoluto, indirecto
Ver también	:	LDA
Ejemplo	:	

```
sta m200
; transfiere el contenido del acumulador 16 bits
; a la palabra 200

sta f200
; transfiere el contenido del acumulador flotante
; al flotante 200

sta l200
; transfiere el acumulador 32 bits al largo l200
```


TSA

Nombre	:	TSA - test accumulator
Función	:	efectúa un Y lógico entre el acumulador 16 bits y una palabra o una constante; efectúa un Y lógico entre el acumulador 32 bits y un largo o una constante, opera de manera similar a la instrucción ANA, pero sin modificar el contenido del acumulador
Variables	:	M o %MW, L o %MD
Direccionamiento:		absoluto, indirecto, inmediato
Ver también	:	ANA
Ejemplo	:	

```
tsa %10
rfz b99
jmp :continuación: and b99
; ramificación en label :continuación: si el bit 1
; del acumulador 16 bits está en 0
```


WOU

Nombre	:	WOU - write output
Función	:	efectúa una escritura de las salidas físicas. Esta función se implementa únicamente en destinos Z (y varía según el destino). Consultar la documentación relativa a cada ejecutor para más detalles.
Variables	:	ninguna
Direccionamiento:		inmediato
Ver también	:	RIN

XRA

Nombre	:	XRA - xor accumulator
Función	:	efectúa un O EXCLUSIVO entre el acumulador 16 bits y una palabra o una constante; efectúa un O EXCLUSIVO entre el acumulador 32 bits y un largo o una constante
Variables	:	M o %MW, L o %MD
Direccionamiento:		absoluto, indirecto, inmediato
Ver también	:	ORA, ANA,
Ejemplo	:	

```
xra %1111111100000000
; invierte los 8 bits de peso fuerte del acumulador 16 bits

xra 1L
; invierte el bit de peso débil del acumulador 32 bits
```

1.10.4. Macro-instrucción

Las macro-instrucciones son nuevas instrucciones del lenguaje literal detrás de las cuales se esconde un conjunto de instrucciones de base.
Sintaxis de llamado de una macro-instrucción:
« %<nombre de la macro-instrucción*> {parámetros ...} »
Sintaxis de declaración de una macro-instrucción:

```
#MACRO
<programa>
#ENDM
```

Esta declaración se encuentra en un archivo que lleva el nombre de la macro-instrucción y la extensión « .M ».

El archivo .M puede estar en el subrepertorio « lib » del repertorio de instalación de AUTOMGEN o en los recursos del proyecto.

Es posible pasar diez parámetros a la macro-instrucción. A ser llamados, estos parámetros se ubicarán sobre la misma línea que la macro-instrucción y estarán separados por un espacio.
En el programa de la macro-instrucción la sintaxis « {?n} » hace referencia al parámetro n.

Ejemplo:
Realicemos la macro-instrucción « cuadrado » que eleva el primer parámetro de la macro-instrucción al cuadrado y pone el resultado en el segundo parámetro.
Llamado de la macro-instrucción:

```
lda 3
sta m200
%cuadrado m200 m201
; m201 contendrá 9
```

* El nombre de la macro-instrucción puede ser un camino de acceso completo al archivo « .M », puede contener una designación de lector y de repertorio.

Archivo « CUADRADO.M »:

```
#MACRO
lda {?0}
mla {?0}
sta {?1}
#ENDM
```

1.10.5. Biblioteca

La noción de biblioteca permite definir recursos que se compilarán una sola vez en una aplicación, más allá de la cantidad de llamados a esos recursos.

Sintaxis de definición de una biblioteca:

```
#LIBRARY <nombre de la biblioteca>
<programa>
#ENDL
```

<nombre de la biblioteca> es el nombre de la función que será llamada por una instrucción
jsr :<nombre de la biblioteca>:
Al primer llamado encontrado por el compilador, el código de la biblioteca se compila. Para los siguientes, el llamado se dirige simplemente a la rutina existente.
Este mecanismo se adapta particularmente al empleo de bloques funcionales y macro-instrucciones para limitar la generación de código en caso de utilización múltiple de los mismos recursos programas.
Las palabras m120 a m129 se reservan a las bibliotecas y pueden utilizarse para el pasaje de los parámetros.

1.10.6. Macro-instrucciones predefinidas

En el subrepertorio « LIB » del repertorio donde se ha instalado AUTOMGEN hay macro-instrucciones de conversión.
También están presentes los equivalentes en bloques funcionales.

1.10.7. Descripción de las macro-instrucciones predefinidas

1.10.7.1. Conversiones

```
%ASCTOBIN <dos primeros dígitos> <dos últimos dígitos> <resultado en binario>
```

Efectúa una conversión ASCII hexadecimal (dos primeros parámetros) a binario (tercer parámetro). En salida el acumulador contiene $FFFF si los dos primeros parámetros no son números ASCII válidos, y si no 0. Todos los parámetros son palabras de 16 bits.

```
%BCDTOBIN <valor en BCD> <valor en binario>
```

Efectúa una conversión BCD a binario. En salida el acumulador contiene $FFFF si el primer parámetro no es un número bcd válido, y si no 0. Los dos parámetros son palabras de 16 bits.

```
%BINTOASC <valor en binario> <resultado parte alta> <resultado parte baja>
```

Efectúa una conversión binario (primer parámetro) a ASCII hexadecimal (segundo y tercer parámetros). Todos los parámetros son palabras de 16 bits.

```
%BINTOBCD <valor en binario> <valor en BCD>
```

Efectúa una conversión BCD (primer parámetro) a binario (segundo parámetro). En salida el acumulador contiene $FFFF si el número binario no puede convertirse a BCD, y si no 0.

```
%GRAYTOB <valor en código GRAY> <valor en binario>
```

Efectúa una conversión código Gray (primer parámetro) a binario (segundo parámetro).

1.10.7.2. Tratamiento con tabla de palabras

```
%COPY <primera palabra tabla fuente> <primera palabra tabla destino> <número de palabras>
```

Copia una tabla de palabras fuente en una tabla de palabras destino. La longitud se da en número de palabras.

```
%COMP <primera palabra tabla 1> <primera palabra tabla 2> <número de palabras> <resultado>
```

Compara dos tablas de palabras. El resultado es una variable binaria que adopta el valor 1 si todos los elementos de la tabla 1 son idénticos a la tabla 2.

```
%FILL <primera palabra tabla> <valor> <número de palabras>
```

Llena una tabla de palabras con un valor.

1.10.7.3. Tratamiento en cadena de caracteres

La codificación de las cadenas de caracteres es la siguiente: un carácter por palabra, una palabra que contiene el valor 0 marca el fin de la cadena. En las macro-instrucciones, las cadenas pasan a parámetros designando la primera palabra que las compone.

`%STRCPY <cadena fuente> <cadena destino>`

Copia una cadena en otra.

`%STRCAT <cadena fuente> <cadena destino>`

Añade la cadena fuente al final de la cadena destino.

`%STRCMP <cadena 1> <cadena 2> <resultado>`

Compara dos cadenas. El resultado es una variable booleana que pasa a 1 si las dos cadenas son idénticas.

`%STRLEN <cadena> <resultado>`

Coloca la longitud de la cadena en la palabra resultado.

`%STRUPR <cadena>`

Transforma todos los caracteres de la cadena en mayúsculas.

`%STRLWR <cadena>`

Transforma todos los caracteres de la cadena en minúsculas.

Ejemplo:

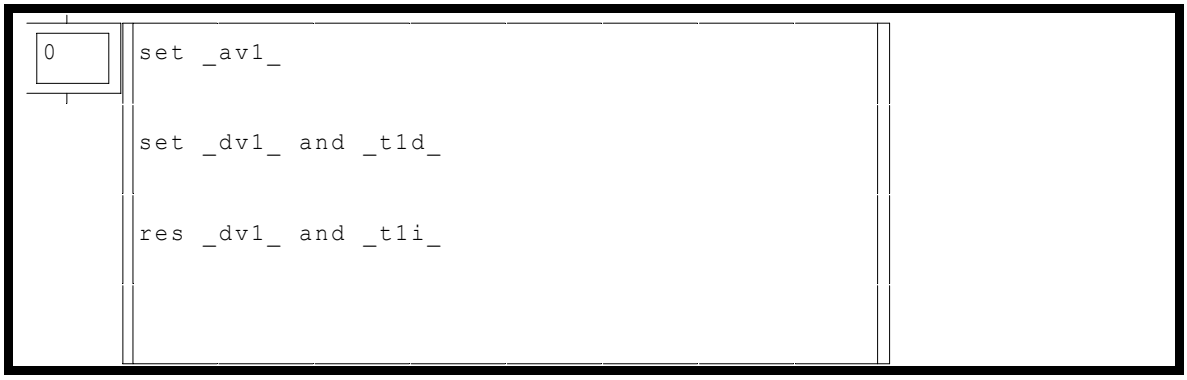
Conversión de m200 (binario) a m202, m203 en 4 dígitos (ASCII bcd)

```
%bintobcd m200 m201
%bintoasc m201 m202 m203
```

1.10.8. Ejemplo en lenguaje literal bajo nivel

Condiciones: empecemos con el ejemplo más simple: ida y vuelta de una locomotora por la vía 1.

Solución:



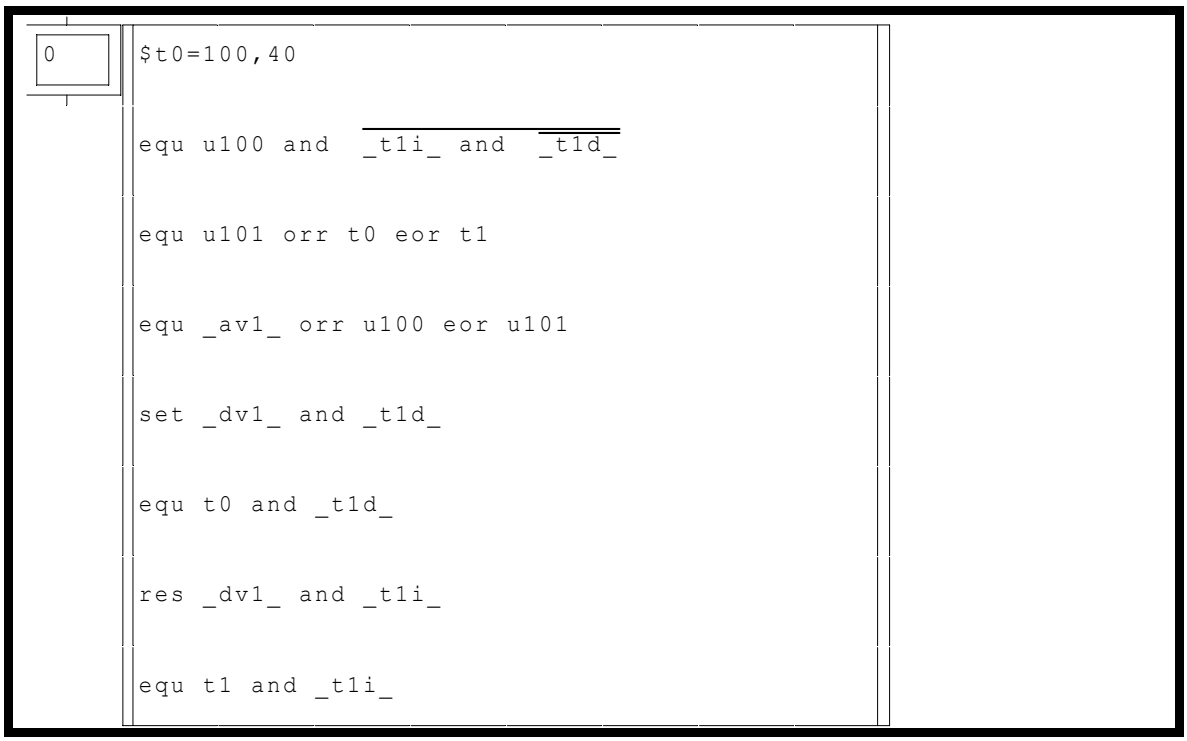
exemple\lit\littéral bas niveau1.agn

Un ejemplo un poco más evolucionado.

Condiciones:

La locomotora ahora deberá marcar una espera de 10 segundos en el extremo derecho de la vía y otra de 4 segundos en el izquierdo.

Solución:



exemple\lit\littéral bas niveau 2.agn

Otro ejemplo:

Condiciones:

Hacer parpadear todos los semáforos de la maqueta.

Solución:

0

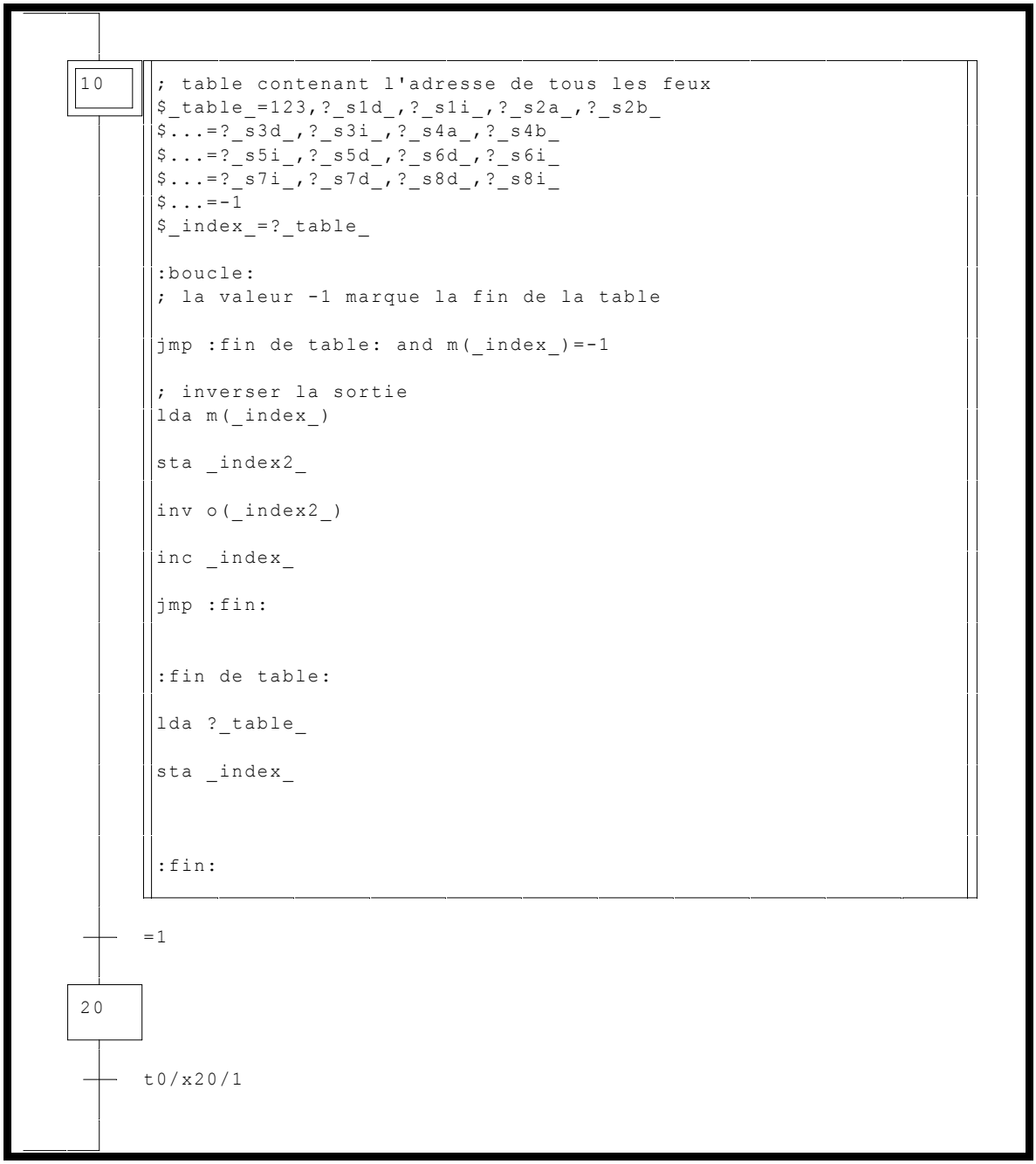
```
; table contenant l'adresse de tous les feux
$_table_=123,?_s1d_,?_s1i_,?_s2a_,?_s2b_
$...=?_s3d_,?_s3i_,?_s4a_,?_s4b_
$...=?_s5i_,?_s5d_,?_s6d_,?_s6i_
$...=?_s7i_,?_s7d_,?_s8d_,?_s8i_
$...=-1

; initialise l'index sur le debut de la table
lda ?_table_
sta _index_

:boucle:
; la valeur -1 marque la fin de la table
jmp :fin: and m(_index_)=-1

; inverser la sortie
lda m(_index_)

sta _index2_

inv o(_index2_)

inc _index_

jmp :boucle:









:fin:
```

exemple\lit\littéral bas niveau 3.agn

Este ejemplo muestra cómo emplear las predisposiciones, que aquí se utilizan para crear una tabla de direcciones de variables. La tabla contiene la dirección de todas las salidas que pilotean los semáforos de la maqueta.
A cada ciclo de ejecución, el estado de todos los semáforos se invierte.

Se presenta un problema y es que los semáforos parpadean demasiado rápido. Modifiquemos el ejemplo.
Condiciones:
Ahora hay que invertir el estado de los semáforos uno por uno a cada décima de segundo.

Solución:



exemple\lit\littéral bas niveau 4.agn

1.11. Lenguaje literal extendido

El lenguaje literal extendido es un « superconjunto » del lenguaje literal bajo nivel.

Permite escribir más simplemente y bajo una forma más concisa ecuaciones booleanas y numéricas.

También permite escribir estructuras de tipo IF ... THEN ... ELSE y WHILE ... ENDWHILE (bucle).

El lenguaje literal extendido está sujeto a las mismas reglas que el lenguaje literal bajo nivel, utiliza la misma sintaxis para las variables, los mnemónicos, los tipos de test (frentes, estado complementado, estado inmediato) y los modos de direccionamiento.
Es posible « mezclar » el lenguaje literal bajo nivel con el lenguaje literal extendido.
Cuando el compilador de lenguaje literal detecta una línea escrita en lenguaje literal extendido, la descompone en instrucciones de lenguaje literal bajo nivel y luego la compila.

1.11.1. Escritura de ecuaciones booleanas

Sintaxis general:

```
« variable bool.=(tipo de asignación) (variable bool. 2 operador 1 variable bool. 3... operador n -1 variable bool. n ) »
```

El tipo de asignación debe precisarse si no es « Asignación ».
Puede ser:

⇨ « (/) »: asignación complementada,

⇨ « (0) »: puesta en cero,

⇨ « (1) »: puesta en uno.

Los operadores pueden ser:

⇨ « . »: y,

⇨ « + »: o.

Las ecuaciones pueden contener varios niveles de paréntesis para precisar el orden de evaluación. Las ecuaciones se evalúan en forma predeterminada de izquierda a derecha.

Ejemplos y equivalencias con el lenguaje literal bajo nivel:

`o0=(i0)`	`equ o0 and i0`
`o0=(i0.i1)`	`equ o0 and i0 and i1`
`o0=(i0+i1)`	`equ o0 orr i0 eor i1`
`o0=(1)`	`set o0`
`o0=(0)`	`res o0`
`o0=(1)(i0)`	`set o0 and i0`
`o0=(0)(i0)`	`res o0 and i0`

```
o0=(1)(i0.i1)                          set o0 and i0 and i1
o0=(0)(i0+i1)                          res o0 orr o0 eor i1
o0=(/)(i0)                             neq o0 and i0
o0=(/)(i0.i1)                          neq o0 and i0 and i1
o0=(/i0)                               equ o0 and /i0
o0=(/i0./i1)                           equ o0 and /i0 and /i1
o0=(c0=10)                             equ o0 and c0=10
o0=(m200<100+m200>200)                 equ o0 orr m200<100 eor m200>200
```

1.11.2. Escritura de ecuaciones numéricas

Sintaxis general para los enteros:

```
« variable num.1=[variable num.2 operador 1 ... operador n-1 variable num.n] »
```

Las ecuaciones pueden contener varios niveles de corchetes para precisar el orden de evaluación. Las ecuaciones se evalúan en forma predeterminada de izquierda a derecha.

Los operadores para los enteros 16 y 32 bits pueden ser:

```
« + »: suma (equivalente a la instrucción ADA),
« - »: resta (equivalente a la instrucción SBA),
« * »: multiplicación (equivalente a la instrucción MLA),
« / »: división (equivalente a la instrucción DVA),
« < »: desfase a izquierda (equivalente a la instrucción RLA),
« > »: desfase a derecha (equivalente a la instrucción RRA),
« & »: « Y » binario (equivalente a la instrucción ANA),
« | »*: « O » binario (equivalente a la instrucción ORA),
« ^ »: « O exclusivo » binario (equivalente a la instrucción XRA).
```

Los operadores para los flotantes pueden ser:

⇨ « + »: suma (equivalente a la instrucción ADA),

⇨ « - »: resta (equivalente a la instrucción SBA),

⇨ « * »: multiplicación (equivalente a la instrucción MLA),

⇨ « / »: división (equivalente a la instrucción DVA).

No se puede precisar una constante en las ecuaciones con los flotantes. Si es necesario, hay que utilizar predisposiciones sobre los flotantes.

* Este carácter suele asociarse a la combinación de teclas [ALT] + [6] en los teclados.

Las ecuaciones con los flotantes pueden llamar a las funciones « SQR » y « ABS ».
Observación: según la complejidad de las ecuaciones, el compilador puede utilizar variables intermedias. Estas variables son las palabras m53 a m59 para los enteros 16 bits, los largos l53 a l59 para los enteros 32 bits y los flotantes f53 a f59.

Ejemplos y equivalencias con el lenguaje literal bajo nivel:

```
M200=[10]                                lda 10
                                         sta m200
M200=[m201]                              lda m201
                                         sta m200
M200=[m201+100]                          lda m201
                                         ada 100
                                         sta m200
M200=[m200+m201-m202]                    lda m200
                                         ada m201
                                         sba m202
                                         sta m200
M200=[m200&$ff00]                        lda m200
                                         ana $ff00
                                         sta m200
F200=[f201]                              lda f201
                                         sta f200
F200=[f201+f202]                         lda f201
                                         ada f202
                                         sta f200
F200=[sqr[f201]]                         lda f201
                                         sqr aaa
                                         sta f200
F200=[sqr[abs[f201*100R]]]               lda f201
                                         mla 100R
                                         abs aaa
                                         sqr aaa
                                         sta f200

L200=[l201+$12345678L]                   lda l201
                                         ada $12345678L
                                         sta l200
```

1.11.3. Estructura de tipo IF ... THEN ... ELSE ...

Sintaxis general:

```
IF(test)
      THEN
      acción si test verdadero
      ENDIF
      ELSE
      acción si test falso
      ENDIF
```

El test debe respetar la sintaxis descrita en el capítulo sobre ecuaciones booleanas.
Puede figurar sólo una acción si test verdadero, o una acción si test falso.
Es posible imbricar varias estructuras de este tipo.
Los bits Sistema u90 a u99 se utilizan como variables temporales para gestionar este tipo de estructura.

Ejemplos:

```
IF(i0)
      THEN
      inc m200            ; incrementar la palabra 200 si i0
      ENDIF

IF(i1+i2)
      THEN
      m200=[m200+10]      ; añadir 10 a la palabra 200 si i1 o i2
      ENDIF
      ELSE
      res m200            ; si no borrar m200
      ENDIF
```

1.11.4. Estructura de tipo WHILE ... ENDWHILE

Sintaxis general:

```
WHILE(test)
      acción a repetir mientras el test sea verdadero
ENDWHILE
```

El test debe respetar la sintaxis descrita en el capítulo sobre ecuaciones booleanas.
Es posible imbricar varias estructuras de este tipo.
Los bits Sistema u90 a u99 se utilizan como variables temporales para gestionar este tipo de estructura.

Ejemplos:

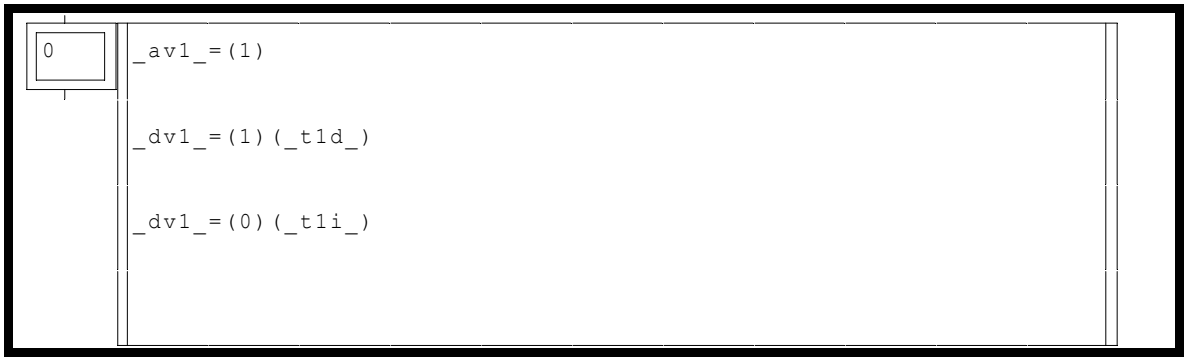
```
m200=[0]
WHILE(m200<10)
      set o(200)
      inc m200           ; incrementar la palabra 200
ENDWHILE
```

Este ejemplo pone en uno las salidas o0 a o9.

1.11.5. Ejemplo de programa en lenguaje literal extendido

Retomemos el primer ejemplo del capítulo anterior.

Solución:



exemple\lit\littéral étendu 1.agn

Enriquezcamos nuestro ejemplo con cálculos.
Condiciones:
Calcular la velocidad en milímetros por segundo y en metros por hora de la locomotora en el trayecto de izquierda a derecha.

Solución:

exemple\lit\littéral étendu 2.agn

La palabra 32 se utiliza para leer el tiempo Sistema. El valor luego se transfiere a flotantes para poder efectuar los cálculos sin perder precisión.

1.12. Lenguaje literal ST

El lenguaje literal ST es el lenguaje literal estructurado definido por la norma CEI1131-3. Este lenguaje permite escribir ecuaciones booleanas y numéricas y estructuras de programación.

1.12.1. Generalidades

El lenguaje literal ST se utiliza en los mismos sitios que el lenguaje literal bajo nivel y el lenguaje literal extendido.

Estas directivas permiten definir secciones en lenguaje literal ST:
« #BEGIN_ST » marca el principio de una sección en lenguaje ST.
« #END_ST » marca el fin de una sección en lenguaje ST.
Ejemplo:

```
m200=[50]          ; lenguaje literal extendido
#BEGIN_ST
m201:=4;           (* lenguaje ST *)
#END_ST
```

También es posible optar por el uso del lenguaje ST para todo un folio. La elección se efectúa en el cuadro de diálogo de propiedades de cada folio.

En un folio con lenguaje predeterminado ST es posible insertar lenguaje literal bajo nivel y extendido encerrando las líneas con las dos directivas « #END_ST » y « #BEGIN_ST ».

Para el lenguaje ST los comentarios deben empezar con « (* » y terminar con « *) ».

Las instrucciones del lenguaje ST terminan con el carácter « ; ». Es posible escribir varias instrucciones en una misma línea.

Ejemplo:

```
o0:=1; m200:=m200+1;
```

1.12.2. Ecuaciones booleanas

La sintaxis general es:

```
variable := ecuación booleana;
```

La ecuación booleana puede componerse de una constante, una variable o varias variables separadas por operadores.

Las constantes pueden ser: 0, 1, FALSE o TRUE.

Ejemplos:

```
o0:=1;
o1:=FALSE;
```

Los operadores que permiten separar varias variables son: + (o), . (y), OR o AND.

El « Y » tiene prioridad sobre el « O ».

Ejemplo:

```
o0:=i0+i1.i2+i3;
```

Será tratado como:

```
o0:=i0+(i1.i2)+i3;
```

Los paréntesis pueden utilizarse en las ecuaciones para especificar las prioridades.

Ejemplo:

```
o0:=(i0+i1).(i2+i3);
```

Pueden utilizarse tests numéricos.

Ejemplo:

```
o0:=m200>5.m200<100;
```

1.12.3. Ecuaciones numéricas

La sintaxis general es:

```
variable := ecuación numérica;
```

La ecuación numérica puede componerse de una constante, una variable o varias variables y constantes separadas por operadores.

Las constantes pueden ser valores expresados en decimal, hexadecimal (prefijo 16#) o binario (prefijo 2#).

Ejemplos:

```
m200:=1234;
m201:=16#aa55;
m202:=2#100000011101;
```

Los operadores que permiten separar varias variables o constantes se ordenan según la prioridad:
* (multiplicación), / (división), + (suma), - (resta), & o AND (Y binario), XOR (O exclusivo binario), OR (O binario).

Ejemplos:

```
m200:=1000*m201;
m200:=m202-m204*m203;            (* equivalente a m200:=m202-(m204*m203) *)
```

Los paréntesis pueden utilizarse en las ecuaciones para especificar las prioridades.

Ejemplo:

```
m200:=(m202-m204)*m203;
```

1.12.4. Estructuras de programación

1.12.4.1. Test SI ENTONCES SI NO

Sintaxis:

```
IF condición THEN acción ENDIF;
```

y

```
IF condición THEN acción ELSE acción ENDIF;
```

Ejemplo:

```
if i0
       then o0:=TRUE;
       else
       o0:=FALSE;
       if i1 then m200:=4; endif;
endif ;
```

1.12.4.2. Bucle MIENTRAS

Sintaxis:

```
WHILE condición DO acción ENDWHILE;
```

Ejemplo:

```
while m200<1000
      do
      m200:=m200+1;
endwhile;
```

1.12.4.3. Bucle HASTA QUE

Sintaxis:

```
REPEAT acción UNTIL condición; ENDREPEAT;
```

Ejemplo:

```
repeat
      m200:=m200+1;
until m200=500
endrepeat;
```

1.12.4.4. Bucle DESDE HASTA

Sintaxis:

```
FOR variable:=valor de inicio TO valor de fin DO acción ENDFOR;
```

o

```
FOR variable:=valor de inicio TO valor de fin BY no DO acción ENDFOR;
```

Ejemplo:

```
for m200:=0 to 100 by 2
      do
      m201:=m202*m201;
      endfor;
```

1.12.4.5. Salida de bucle

La palabra clave « EXIT » permite salir de un bucle.

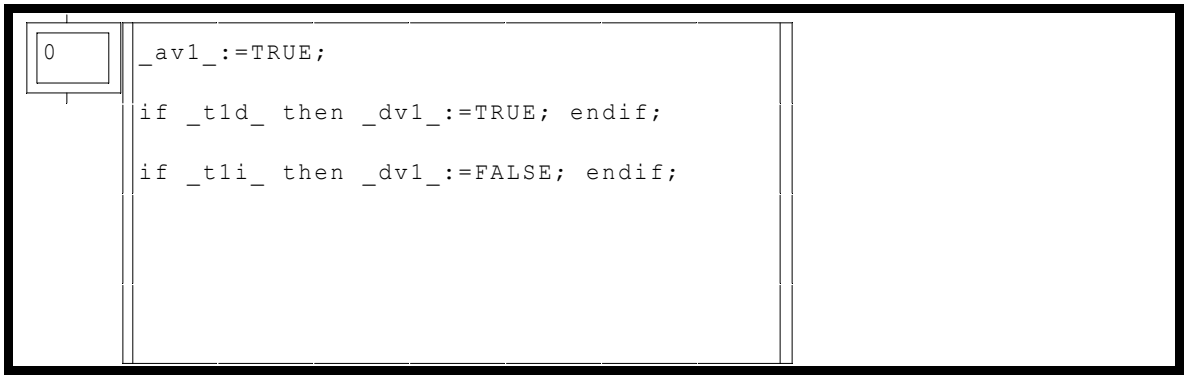
Ejemplo:

```
while i0
      m200:=m200+1;
      if m200>1000 then exit; endif;
endwhile;
```

1.12.5. Ejemplo de programa en lenguaje literal extendido

Retomemos el primer ejemplo del capítulo anterior.

Solución:



exemple\lit\littéral ST 1.agn

1.13. Organigrama

AUTOMGEN implementa una programación de tipo « organigrama ».
Para este tipo de programación se deben utilizar los lenguajes literales. Consultar los capítulos anteriores para saber cómo utilizarlos.
La ventaja de la programación bajo forma de organigrama es la representación gráfica de un tratamiento algorítmico.
Contrariamente al lenguaje Grafcet, la programación bajo forma de organigrama genera un código que se ejecutará una vez por ciclo de escrutación. Esto significa que no podemos quedarnos a la espera en un rectángulo de organigrama; es obligatorio que la ejecución salga del organigrama para poder seguir ejecutando el programa.
Es importante no olvidar este punto cuando se elige este lenguaje.

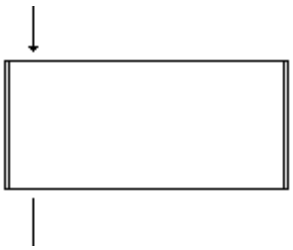
Es posible diseñar sólo rectángulos. El contenido de los rectángulos y los vínculos determinan si el rectángulo es una acción o un test.

1.13.1. Diseño de un organigrama

Los rectángulos se diseñan eligiendo el comando « Más … / Caja de código » del menú contextual (hacer clic con el botón derecho del ratón sobre el fondo del folio para abrir el menú contextual).

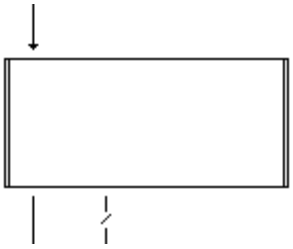
Es necesario colocar un bloque (tecla [<]) en la entrada de cada rectángulo; esta entrada debe estar ubicada en la parte superior del rectángulo.
Si el rectángulo representa una acción, habrá una sola salida materializada por un bloque (tecla [E]) abajo y a la izquierda del rectángulo.

Un rectángulo de acción:



Si el rectángulo representa un test, habrá obligatoriamente dos salidas. La primera, materializada por un bloque (tecla [E]) abajo y a la izquierda representa la salida si el test es verdadero; la segunda, materializada por un bloque (tecla [=]) inmediatamente a la derecha de la otra salida representa la salida si el test es falso.

Un rectángulo de test:



Las ramas de organigramas deben terminar con un rectángulo sin salida que eventualmente puede quedar vacío.

1.13.2. Contenido de los rectángulos

1.13.2.1. Contenido de los rectángulos de acción

Los rectángulos de acción pueden contener cualquier instrucción de lenguaje literal.

1.13.2.2. Contenido de los rectángulos de test

Los rectángulos de test deben contener un test que respete la sintaxis de la parte test de la estructura de tipo IF...THEN...ELSE... del lenguaje literal extendido.
Por ejemplo:

```
IF (i0)
```

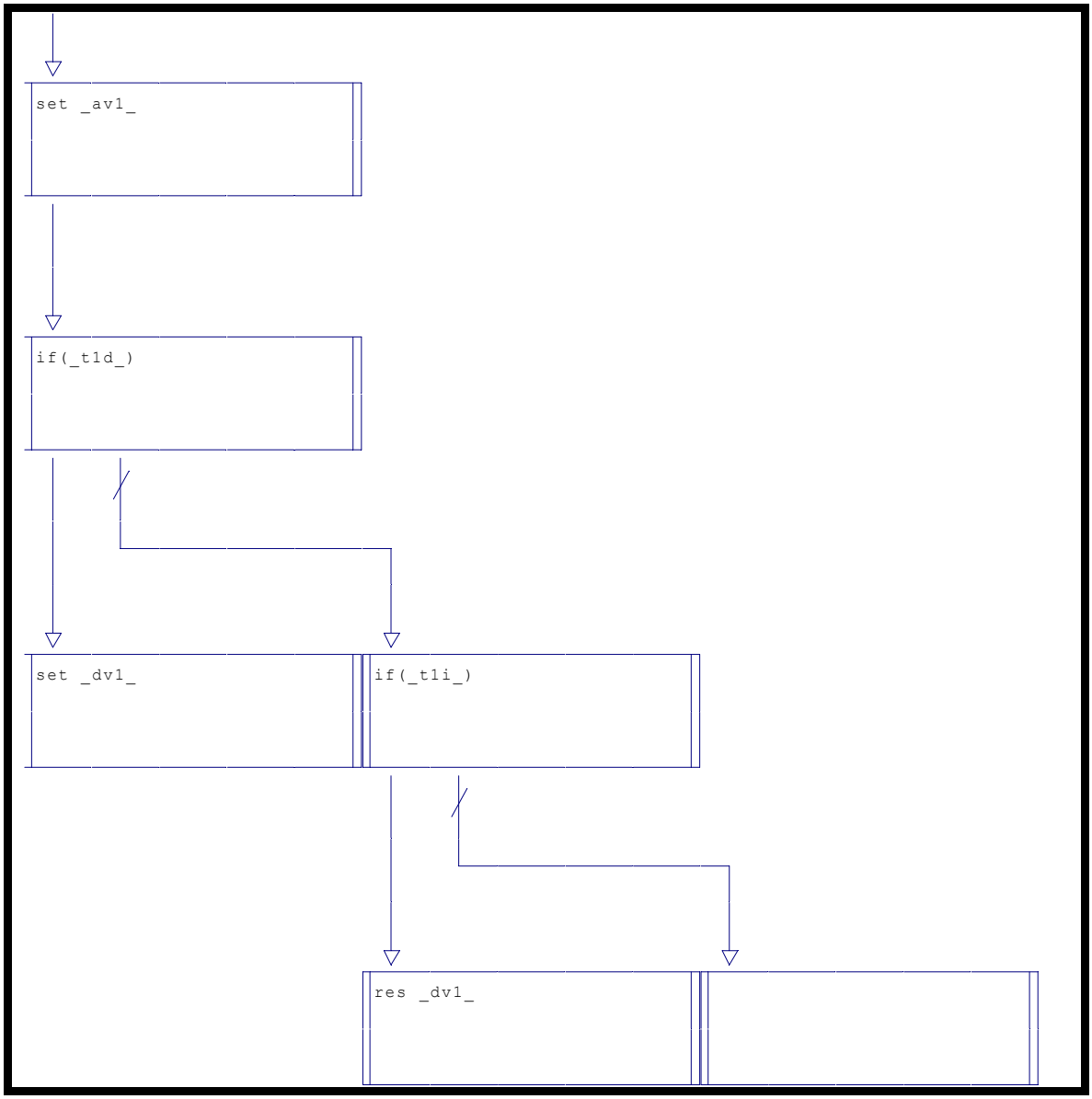
Es posible escribir antes de este test acciones en el rectángulo de test.
Esto permite, por ejemplo, efectuar ciertos cálculos antes del test.
Si por ejemplo queremos testear si la palabra 200 es igual a la palabra 201 más 4:

```
m202=[m201+4]
IF(m200=m202)
```

1.14. Ilustración

Nuestro ya clásico primer ejemplo consiste en hacer ir y venir a una locomotora por la vía 1 de la maqueta.

Solución:



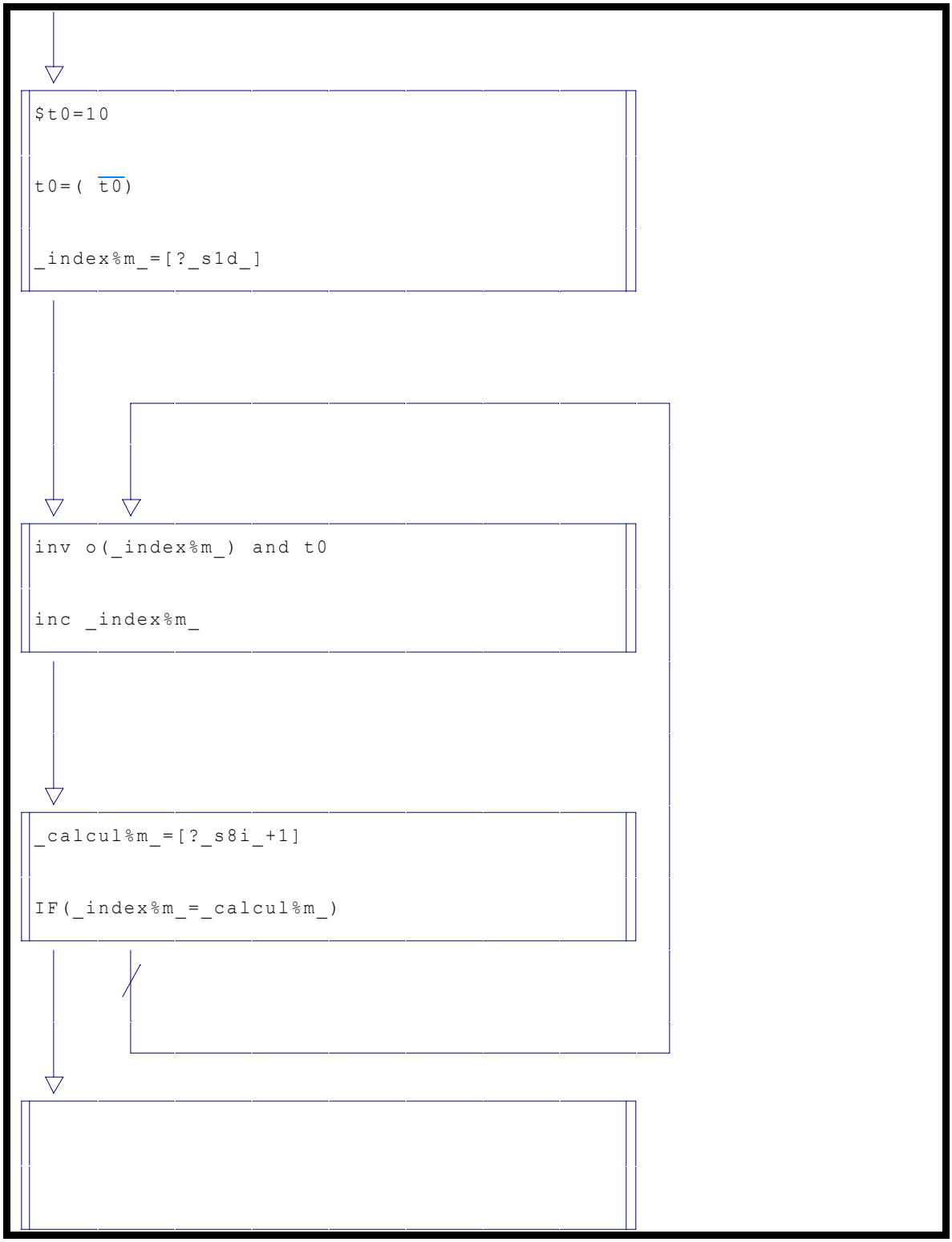
exemple\organigramme\organigramme 1.agn

Segundo ejemplo

Condiciones:

Hacer parpadear todos los semáforos de la maqueta. Los semáforos cambian de estado cada segundo.

Solución:



exemple\organigramme\organigramme 2.agn

Notar la utilización de símbolos automáticos en este ejemplo.

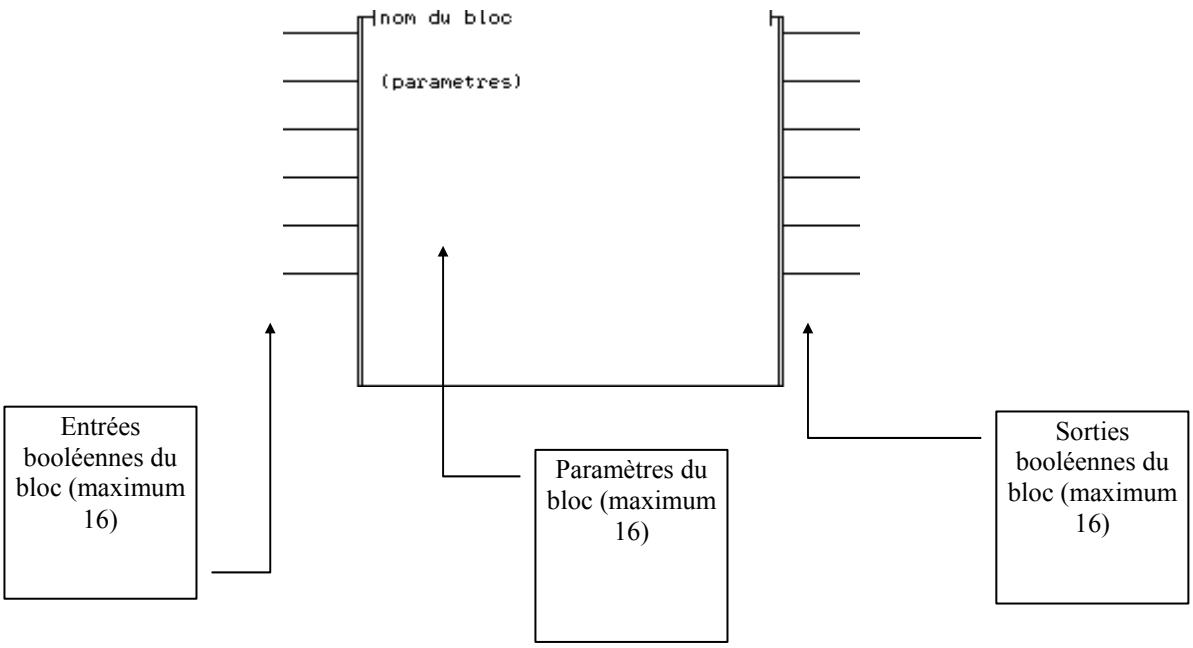
1.15. Bloques funcionales

AUTOMGEN implementa la noción de bloques funcionales.

Este método de programación modular permite asociar a un elemento gráfico un conjunto de instrucciones escritas en lenguaje literal.

Los bloques funcionales pueden ser definidos por el programador. No tienen un número limitado. Por eso es posible constituir conjuntos de bloques funcionales que permiten una concepción modular y estandarizada de las aplicaciones.

Los bloques funcionales se utilizan dentro de esquemas de tipo logigrama o ladder, poseen de una a n entradas booleanas y de una a n salidas booleanas. Si el bloque debe tratar variables no booleanas, éstas se mencionarán en el diseño del bloque funcional. El interior del bloque puede recibir parámetros: constantes o variables.



1.15.1. Creación de un bloque funcional

Un bloque funcional está compuesto por dos archivos distintos: un archivo de extensión « .ZON » que contiene el diseño del bloque funcional y un archivo de extensión « .LIB » que contiene una serie de instrucciones escritas en lenguaje literal que definen el funcionamiento del bloque funcional.

Los archivos « .ZON » y « .LIB » deben llevar el nombre del bloque funcional. Por ejemplo, si decidimos crear un bloque funcional « MEMORIA », deberemos crear los

archivos « MEMORIA.ZON » (para el diseño del bloque) y « MEMORIA.LIB » (para el funcionamiento del bloque).

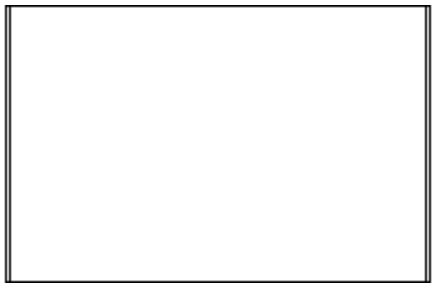
1.15.2. Diseño del bloque y creación del archivo « .ZON »

La envoltura de un bloque funcional está constituida por una caja de código a la que hay que añadir bloques dedicados a los bloques funcionales.
Para diseñar un bloque funcional se deben efectuar las siguientes operaciones:

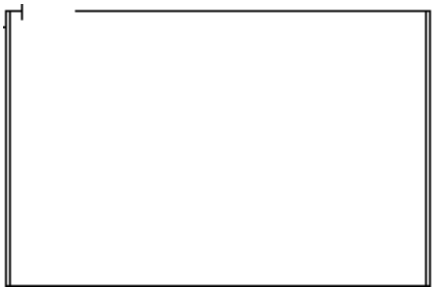
⇨ utilizar el asistente (recomendado)

O:

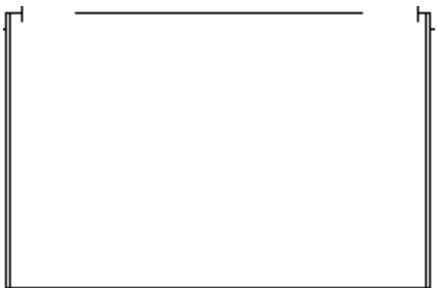
⇨ diseñar una caja de código (utilizar el comando « Más …/Caja de código » del menú contextual):



⇨ colocar un bloque (tecla [8]) en el ángulo superior izquierdo de la caja de código:

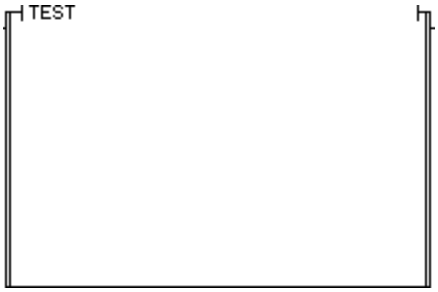


⇨ colocar un bloque  (tecla [9]) en el ángulo superior derecho de la caja de código:



⇨ borrar la línea que queda en la parte superior del bloque (la tecla [A] permite poner bloques blancos):

⇨ hacer clic con el botón izquierdo del ratón en el ángulo superior izquierdo del bloque funcional, ingresar el nombre del bloque funcional, que no debe superar los 8 caracteres (los archivos « .ZON » y « .LIB » deberán llevar ese nombre), presionar [ENTER];



⇨ si se necesitan entradas booleanas suplementarias, se debe utilizar un bloque (tecla [;]) o (tecla [:]); las entradas añadidas deben encontrarse inmediatamente debajo de la primera entrada, sin dejar ningún espacio libre;

⇨ si se necesitan salidas booleanas suplementarias, se debe añadir un bloque (tecla [>]) o (tecla [?]); las salidas añadidas deben encontrarse inmediatamente debajo de la primera salida, sin dejar ningún espacio libre;

⇨ el interior del bloque puede contener comentarios o parámetros; los parámetros se escriben entre llaves « {...} ». Todo lo que no se escriba entre llaves será ignorado por el compilador. Es interesante identificar el uso de las entradas y salidas booleanas dentro del bloque;

⇨ una vez terminado un bloque, se debe utilizar el comando « Seleccionar » del menú « Edición » para seleccionar el diseño del bloque funcional y guardarlo en un archivo « .ZON » con el comando « Copiar en » del menú « Edición ».

1.15.3. Creación del archivo « .LIB »

El archivo « .LIB » es un archivo de texto que contiene instrucciones en lenguaje literal (bajo nivel o extendido). Estas instrucciones definen el funcionamiento del bloque funcional.

Una sintaxis especial permite hacer referencia a las entradas booleanas del bloque, a las salidas booleanas del bloque y a los parámetros del bloque.

Para hacer referencia a una entrada booleana del bloque, se debe utilizar la sintaxis « {Ix} », donde x es el número de la entrada booleana expresado en hexadecimal (0 a f).

Para hacer referencia a una salida booleana del bloque, se debe utilizar la sintaxis « {Ox} », donde x es el número de la salida booleana expresado en hexadecimal (0 a f).

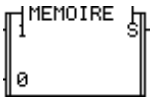
Para hacer referencia a un parámetro del bloque, se debe utilizar la sintaxis « {?x} », donde x es el número del parámetro en hexadecimal (0 a f).

El archivo .LIB puede estar ubicado en el subrepertorio « lib » del repertorio de instalación de AUTOMGEN o en los recursos del proyecto.

1.15.4. Ejemplo simple de bloque funcional

Creemos el bloque funcional « MEMORIA », que posee dos entradas booleanas (puesta en uno y puesta en cero) y una salida booleana (el estado de la memoria).

El diseño del bloque contenido en el archivo « MEMORIA.ZON » es:

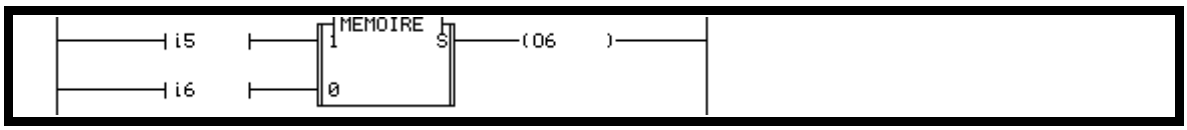


El funcionamiento del bloque contenido en el archivo « MEMORIA.LIB » es:

```
{O0}=(1)({I0})
{O0}=(0)({I1})
```

El bloque puede utilizarse de la siguiente manera:

o



Para utilizar un bloque funcional en una aplicación, se debe elegir el comando « Pegar a partir de » del menú « Edición » y elegir el archivo « .ZON » correspondiente al bloque funcional a utilizar.

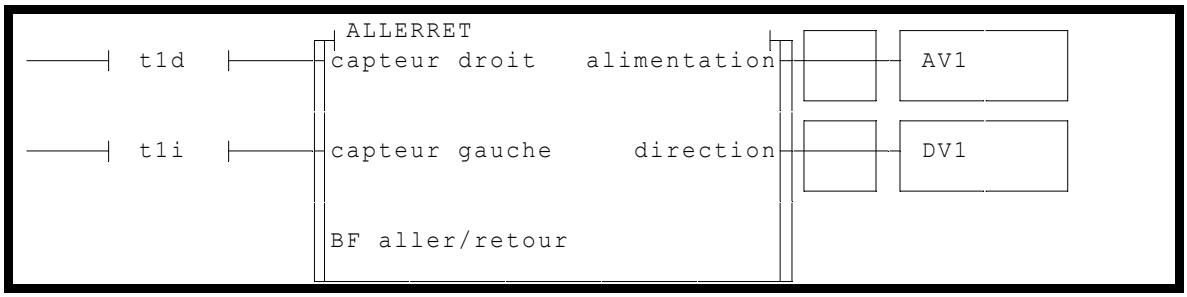
1.15.5. Ilustración

Retomemos un ejemplo ya clásico:

Condiciones:

Ida y vuelta de una locomotora por la vía 1 de la maqueta.

Solución:



exemple\bf\bloc-fonctionnel 1.agn

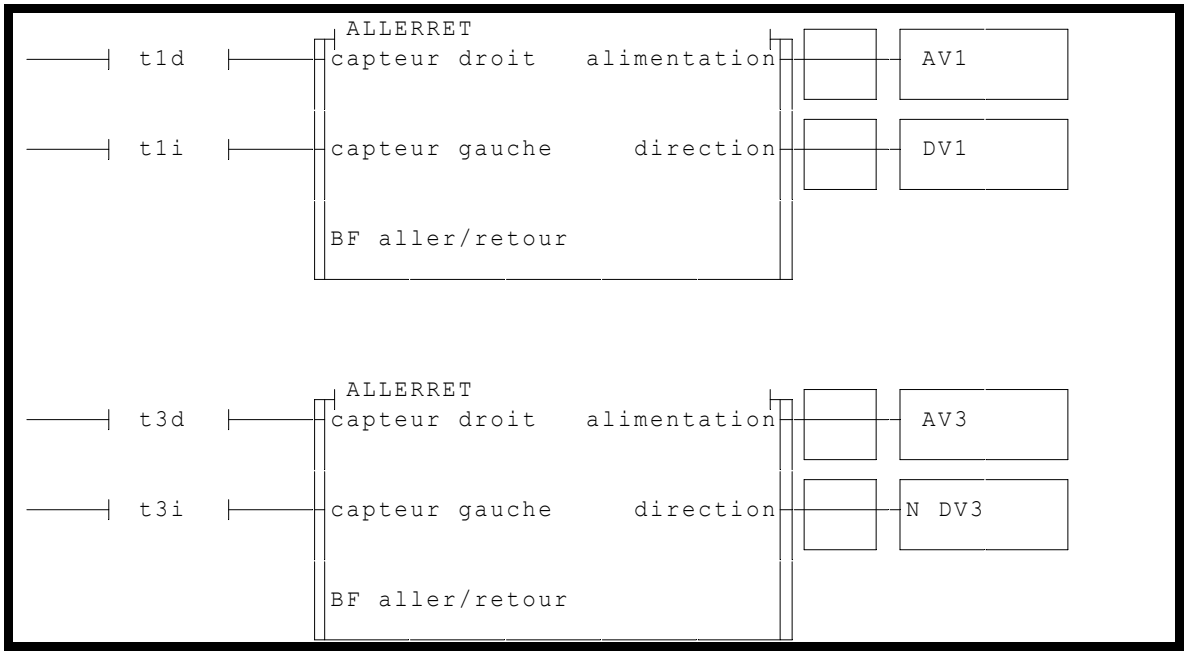
```
; bloque funcional ALLERRET
; ida vuelta de una locomotora por una vía
; las entradas booleanas son los fines de carrera
; las salidas booleanas son la alimentación de la vía (0) y la
dirección (1)

; alimentar siempre la vía
set {O0}

; pilotear la dirección en función de los fines de carrera

{O1}=(1)({I0})
{O1}=(0)({I1})
```

Para ilustrar el interés de la utilización de los bloques funcionales, completemos nuestro ejemplo.

Condiciones:

Ida y vuelta de dos locomotoras por las vías 1 y 3.

Solución:



exemple\bf\bloc-fonctionnel 2.agn

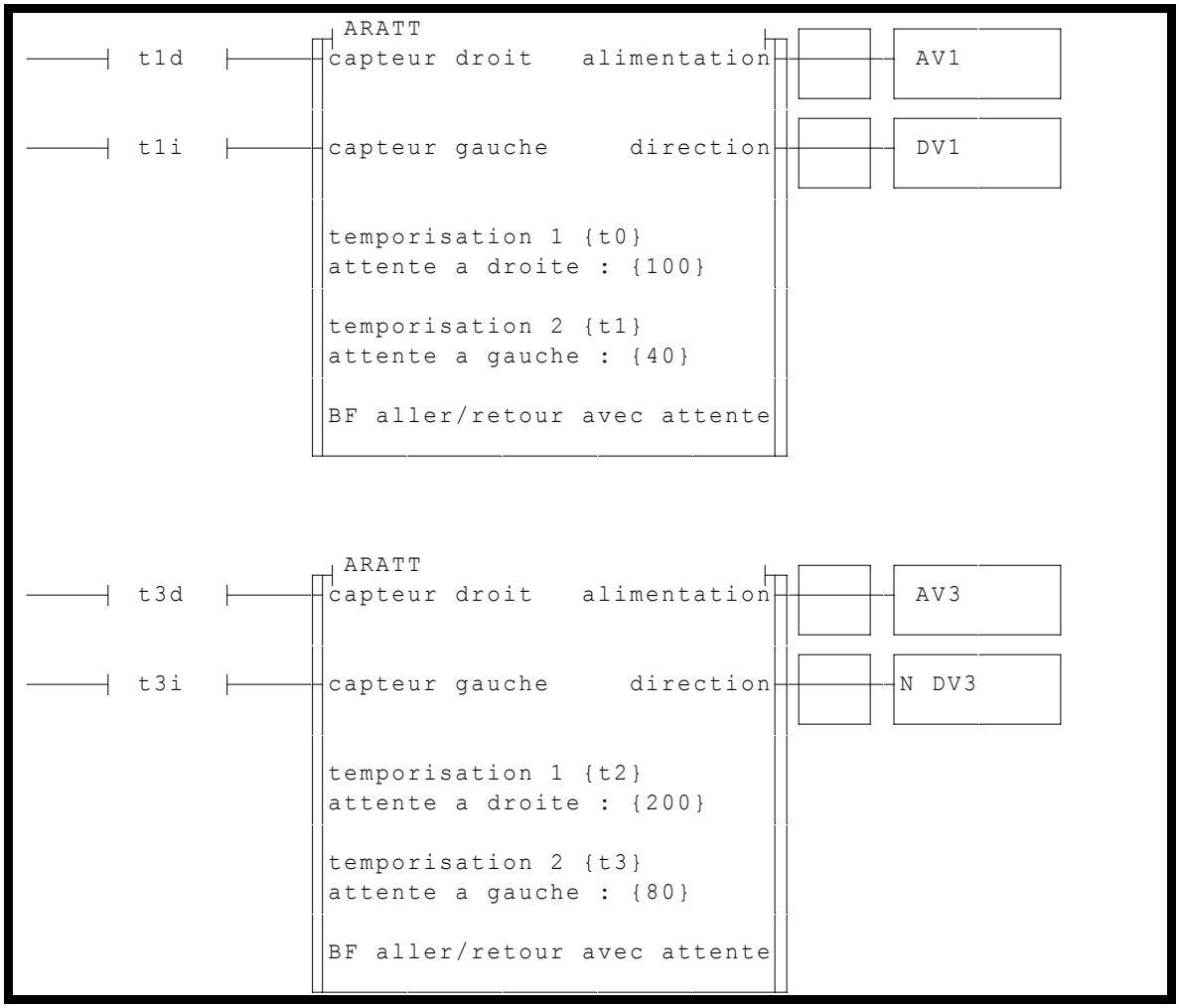
Este ejemplo muestra que con el mismo bloque funcional se pueden hacer funcionar de manera idéntica diferentes módulos de una parte operativa.

Completemos nuestro ejemplo para ilustrar la utilización de parámetros.

Condiciones:

Ahora las dos locomotoras deben marcar una espera al final de la vía. Para la locomotora 1: 10 segundos a la derecha y 4 segundos a la izquierda; para la locomotora 2: 20 segundos a la derecha y 8 segundos a la izquierda.

Solución:


ARATT
t1d
capteur droit
alimentation
AV1
t1i
capteur gauche
direction
DV1
temporisation 1 {t0}
attente a droite : {100}
temporisation 2 {t1}
attente a gauche : {40}
BF aller/retour avec attente
ARATT
t3d
capteur droit
alimentation
AV3
t3i
capteur gauche
direction
N DV3
temporisation 1 {t2}
attente a droite : {200}
temporisation 2 {t3}
attente a gauche : {80}
BF aller/retour avec attente

```
; bloque funcional ARATT
; ida vuelta de una locomotora por una vía con espera
; las entradas booleanas son los fines de carrera
; las salidas booleanas son la alimentación de la vía (0) y la
dirección (1)
; los parámetros son:
;               0: primera temporización
;               1: duración de la primera temporización
;               2: segunda temporización
;               3: duración de la segunda temporización


; predisposición de las dos temporizaciones
${?0}={?1}
${?2}={?3}

; alimentar la vía si no fines de carrera o si tempor. terminadas
set {O0}
res {O0} orr {I0} eor {I1}
set {O0} orr {?0} eor {?2}

; gestión de las temporizaciones
{?0}=({I0})
{?2}=({I1})

; pilotear la dirección en función de los fines de carrera
{O1}=(1)({I0})
{O1}=(0)({I1})
```

exemple\bf\bloc-fonctionnel 3.agn

1.15.6. Complemento de sintaxis

Una sintaxis complementaria permite efectuar un cálculo con los números de variables referenciadas en el archivo « .LIB ».

La sintaxis « ~+n » añadida a continuación de una referencia a una variable o un parámetro suma n.

La sintaxis « ~-n » añadida a continuación de una referencia a una variable o un parámetro resta n.

La sintaxis « ~*n » añadida a continuación de una referencia a una variable o un parámetro multiplica por n.

Es posible escribir varios de estos comandos seguidos; se evalúan de izquierda a derecha.

Este mecanismo es útil cuando un parámetro del bloque funcional debe permitir hacer referencia a una tabla de variables.

Ejemplos:

```
{?0}~+1
```

hace referencia al elemento siguiente al primer parámetro; por ejemplo, si el primer parámetro es m200, esta sintaxis hace referencia a m201.

```
M{?2}~*100~+200
```

hace referencia al tercer parámetro multiplicado por 100 más 200; por ejemplo, si el tercer parámetro es 1, esta sintaxis hace referencia a M 1*100 + 200, es decir M300.

1.16. Bloques funcionales evolucionados

Esta funcionalidad permite crear bloques funcionales muy potentes con más simplicidad que los bloques funcionales gestionados por archivos escritos en lenguaje literal. Este método de programación permite un enfoque de tipo análisis funcional.

Cualquier folio o conjunto de folios puede convertirse en un bloque funcional (a veces se habla de encapsular un programa).

El o los folios que describen el funcionamiento de un bloque funcional pueden acceder a las variables externas del bloque funcional: las entradas booleanas del bloque, las salidas booleanas y los parámetros.

El principio de utilización, particularmente de la utilización de las variables externas, es idéntico al de los viejos bloques funcionales.

1.16.1. Sintaxis

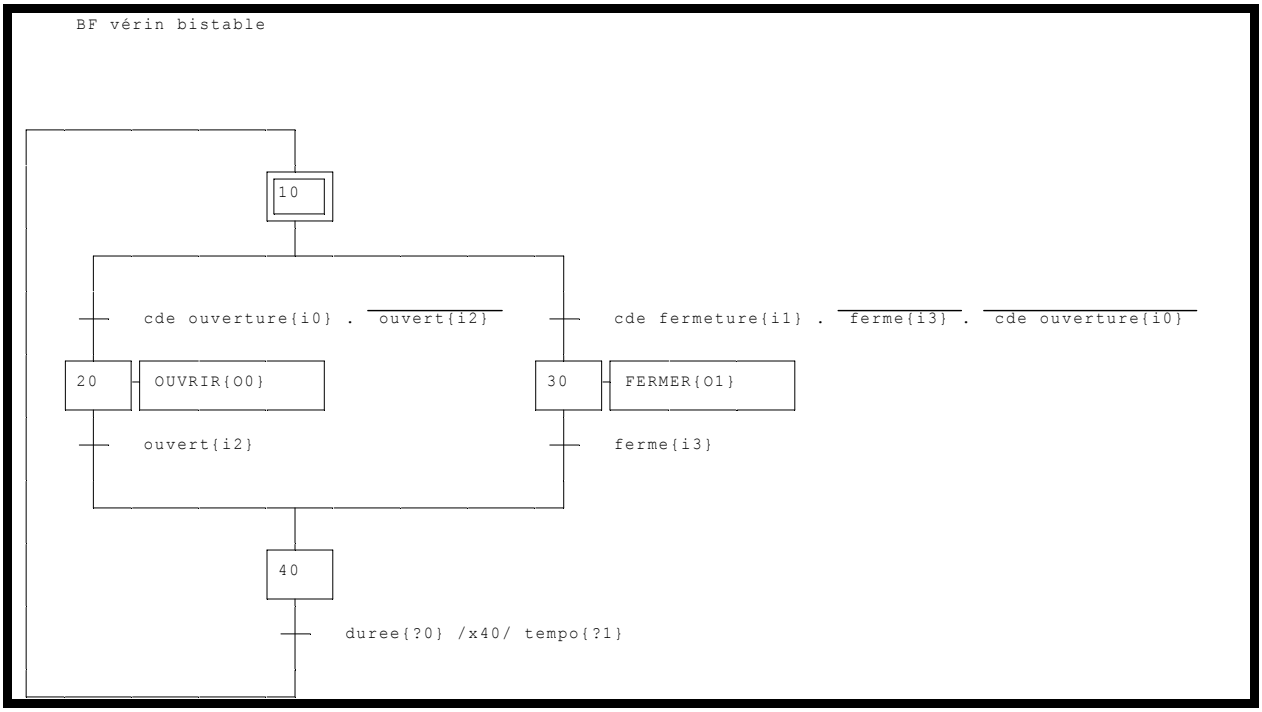
Para referenciar una variable externa de un bloque funcional se debe utilizar un mnemónico que incluya el texto siguiente: {In} para referenciar la entrada booleana n, {On} para referenciar la salida booleana n, {?n} para referenciar el parámetro n. El mnemónico debe empezar con una letra.

1.16.2. Diferenciar viejos y nuevos bloques funcionales

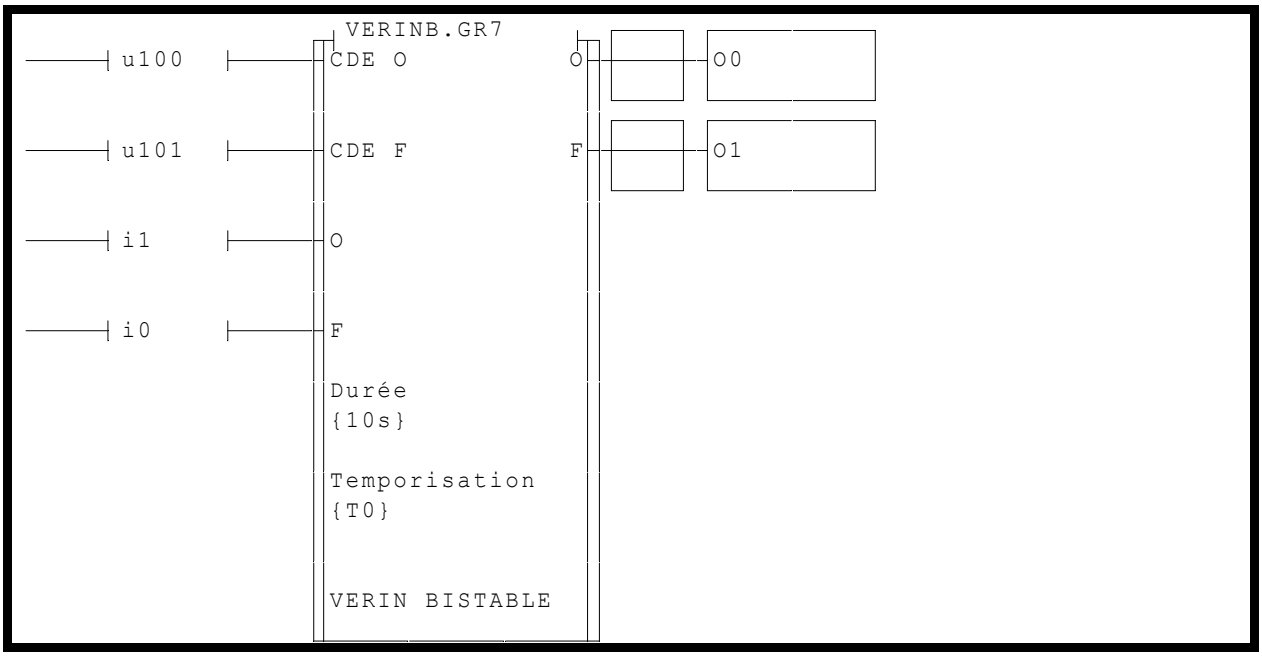
El nombre de archivo inscrito en el diseño bloque funcional indica si se trata de un viejo bloque funcional (gestionado por un archivo .LIB) o de un nuevo bloque funcional (gestionado por un folio .GR7). Para un viejo bloque funcional, el nombre no lleva extensión; para uno nuevo debe añadirse la extensión .GR7. El folio que contiene el código que gestiona el funcionamiento del bloque funcional debe estar integrado en la lista de folios del proyecto. En las propiedades del folio, debe elegirse el tipo « Bloque-funcional ».

1.16.3. Ejemplo

Contenido del folio VERINB:



Llamado del bloque funcional:



exemple\bf\bloc-fonctionnel 3.agn

1.17. Bloques funcionales predefinidos

En el subrepertorio « \LIB » del repertorio donde se ha instalado AUTOMGEN hay bloques funcionales de conversión.

También están los equivalentes en macro-instrucciones; ver el capítulo 1.10.3. El lenguaje literal bajo nivel.

Para insertar y parametrar un bloque funcional en una aplicación se debe seleccionar el comando « Insertar un bloque funcional » del menú « Herramientas ».

1.17.1. Bloques de conversión

ASCTOBIN: conversión ASCII a binario

BCDTOBIN: conversión BCD a binario

BINTOASC: conversión binario a ASCII

BINTOBCD: conversión binario a BCD

GRAYTOB: conversión código Gray a binario

16BINTOM: transferencia de 16 variables booleanas a una palabra

MTO16 BIN: transferencia de una palabra a 16 variables booleanas

1.17.2. Bloques de temporización

TEMPO: temporización a la subida

PULSOR: salida a intervalos

PULSE: impulsión temporizada

1.17.3. Bloques de manipulación de cadena de caracteres

STRCMP: comparación

STRCAT: concatenacíón

STRCPY: copia

STRLEN: cálculo de la longitud

STRUPR: puesta en minúsculas

STRLWR: puesta en mayúsculas

1.17.4. Bloques de manipulación de tabla de palabras

COMP: comparación

COPY: copia

FILL: llenado

1.18. Técnicas avanzadas

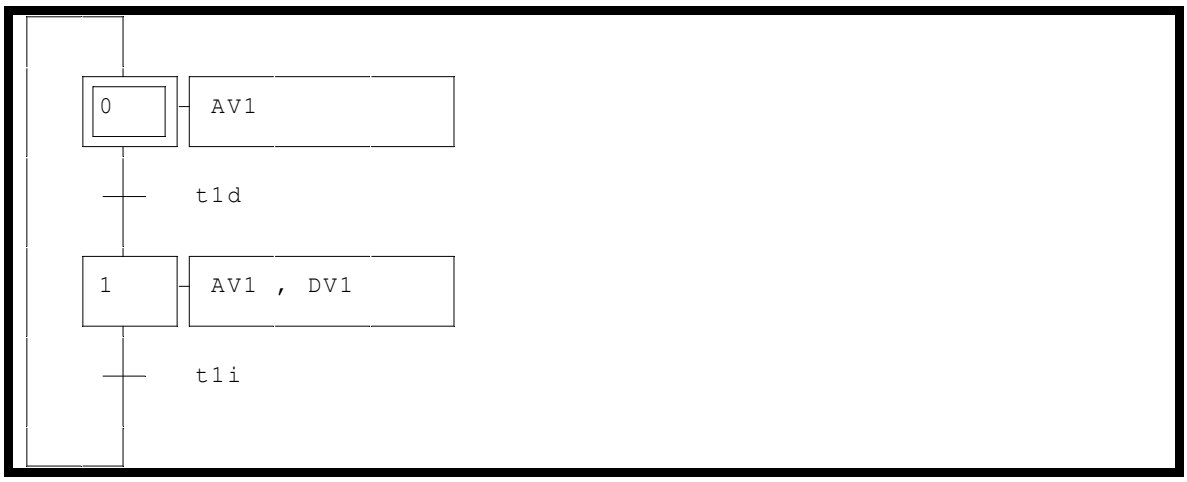
1.18.1. Código generado por el compilador

En este capítulo abordaremos la forma del código generado por la compilación de tal o cual tipo de programa.

El utilitario « CODELIST.EXE »* permite traducir « en claro » un archivo de código intermedio « .EQU » (también llamado lenguaje pivote).

Hagamos la experiencia siguiente: carguemos y compilemos el primer ejemplo de programa del capítulo « Grafcet »: « simple1.agn » del repertorio « exemple\grafcet »:

* Este utilitario debe ejecutarse a partir de la línea de comando DOS.



Hacer doble clic en el elemento « Archivos generados/Código pivote » en el explorador.

Se obtiene la siguiente lista de instrucciones:

```
; El código siguiente ha sido generado por la compilación de: 'Folio:
GRAF1'
:00000000: RES x0   AND i0
:00000002: SET x0   AND b0
:00000004: SET x0   AND x1     AND i1
:00000007: RES x1   AND i1
:00000009: SET x1   AND x0     AND i0
; El código siguiente ha sido generado por la compilación de:
'asignaciones (acciones Grafcet, logigramas y ladder)'
:0000000C: EQU o0   ORR @x0    EOR @x1
:0000000F: EQU o23  AND @x1
```

Representa la traducción de la aplicación « simple1.agn » en instrucciones del lenguaje literal bajo nivel.

Los comentarios indican el origen de las porciones de código, lo cual es útil si una aplicación se compone de varios folios.

Obtener esta lista de instrucciones puede ser útil para responder a las preguntas sobre el código generado por tal o cual forma de programa o la utilización de tal o cual lenguaje.

En ciertos casos « críticos » para los que es importante conocer informaciones como « ¿al cabo de cuántos ciclos esta acción es verdadera? » el modo paso a paso y el examen exhaustivo del código generado resultan indispensables.

1.18.2. Optimización del código generado

Son posibles varios niveles de optimización.

1.18.2.1. Optimización del código generado por el compilador

La opción de optimización del compilador permite reducir considerablemente el tamaño del código generado. Esta directiva exige al compilador que genere menos líneas de lenguaje literal bajo nivel; como consecuencia, el tiempo de compilación aumenta.

Según los post-procesadores utilizados, esta opción implica una ganancia sobre el tamaño del código o el tiempo de ejecución. Conviene efectuar pruebas para determinar si la directiva es interesante o no según la naturaleza del programa y el tipo de destino utilizado.

En general es interesante utilizarla con los post-procesadores para destinos Z.

1.18.2.2. Optimización del código generado por los post-procesadores

Cada post-procesador puede poseer opciones para optimizar el código generado. Para los post-procesadores que generan código constructor consulte la nota correspondiente.

1.18.2.3. Optimización del tiempo de ciclo: reducir el número de temporizaciones en destinos Z

Para los destinos Z, el número de temporizaciones declaradas influye directamente en el tiempo de ciclo. Declare el mínimo de temporizaciones en función de las necesidades de la aplicación.

1.18.2.4. Optimización del tiempo de ciclo: anular la escrutación de ciertas partes del programa

Sólo los destinos que aceptan instrucciones JSR y RET soportan esta técnica.

Es posible validar o « invalidar » la escrutación de ciertas partes del programa utilizando directivas de compilación especiales.

Dichas porciones de aplicación son definidas por los folios.

Si una aplicación se descompone en cuatro folios, cada uno de ellos podrá ser « validado » o « invalidado » independientemente.

Una directiva « #C(condición) » colocada en un folio condiciona la escrutación del folio hasta el folio que contiene una directiva « #R ».

Esta condición debe utilizar la sintaxis definida para los tests; ver el capítulo 1.3. Tests

Ejemplo:

Si un folio contiene las dos directivas:

```
#C(m200=4)
#R
```

Entonces todo lo que contiene se ejecutará sólo si la palabra 200 contiene 4.

2. Ejemplos

2.1. A propósito de ejemplos

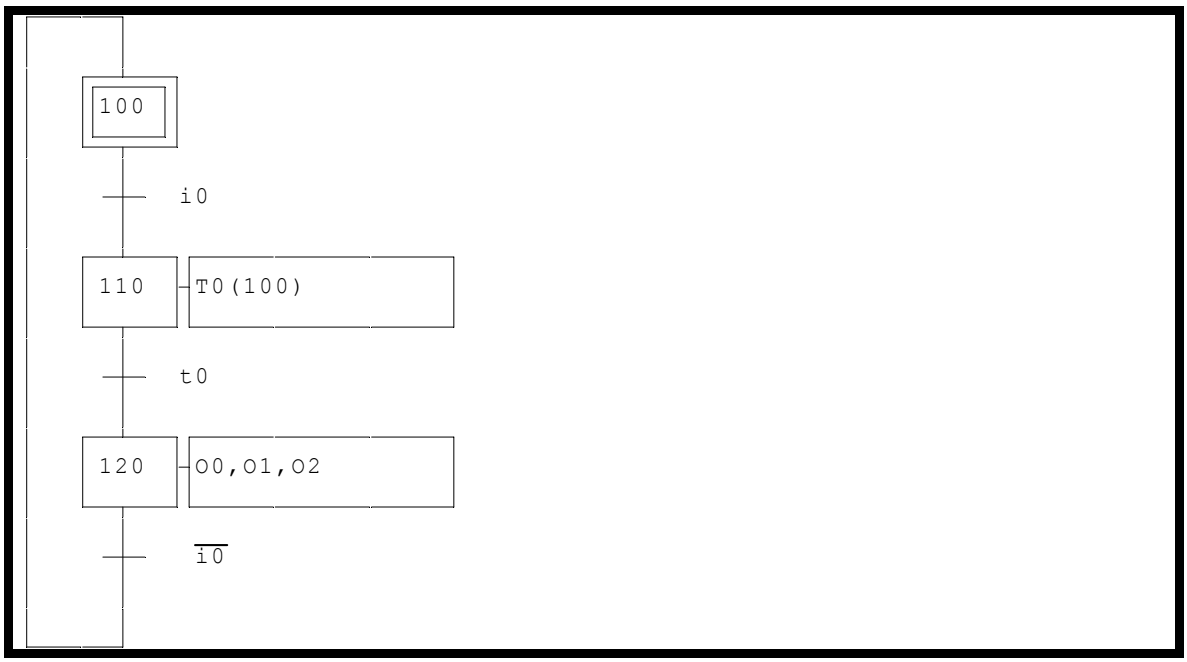
Esta parte agrupa una serie de ejemplos que ilustran diferentes posibilidades de programación con AUTOMGEN.

Todos los ejemplos se encuentran en el subrepertorio « ejemplo » del repertorio donde se ha instalado AUTOMGEN.

Esta parte también contiene ejemplos más completos y más complejos desarrollados para una maqueta de tren. La descripción de esta maqueta se encuentra al principio del manual de referencia del lenguaje.

2.1.1. Grafcet simple

Este primer ejemplo es un Grafcet simple en línea:

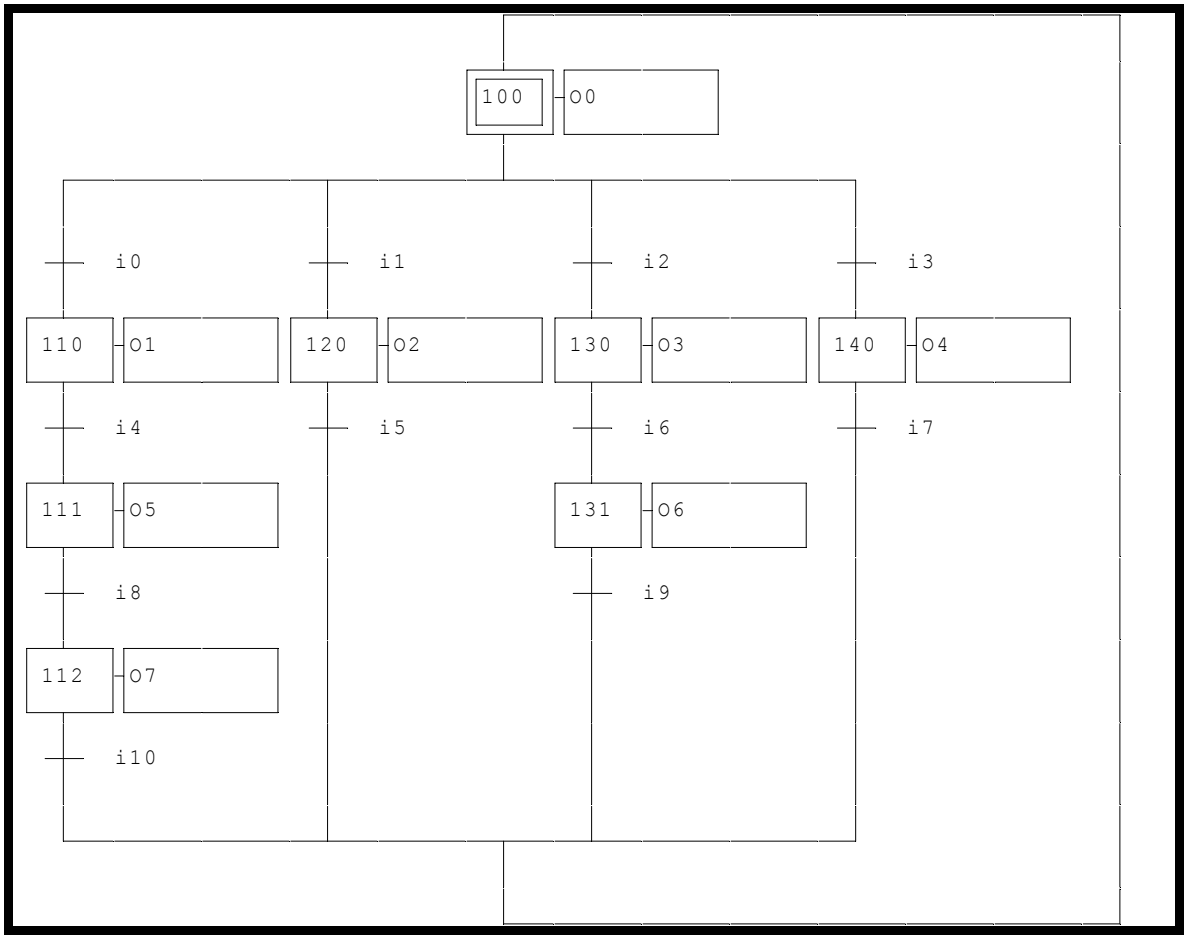


exemple\grafcet\sample1.agn

⇨ la transición entre la etapa 100 y la etapa 110 está constituida por el test en la entrada 0,

- ⇨ la etapa 110 activa la temporización 0 de una duración de 10 segundos; esta temporización se utiliza como transición entre la etapa 110 y 120,
- ⇨ la etapa 120 activa las salidas 0, 1 y 2,
- ⇨ el complemento de la entrada 0 sirve de transición entre la etapa 120 y 100.

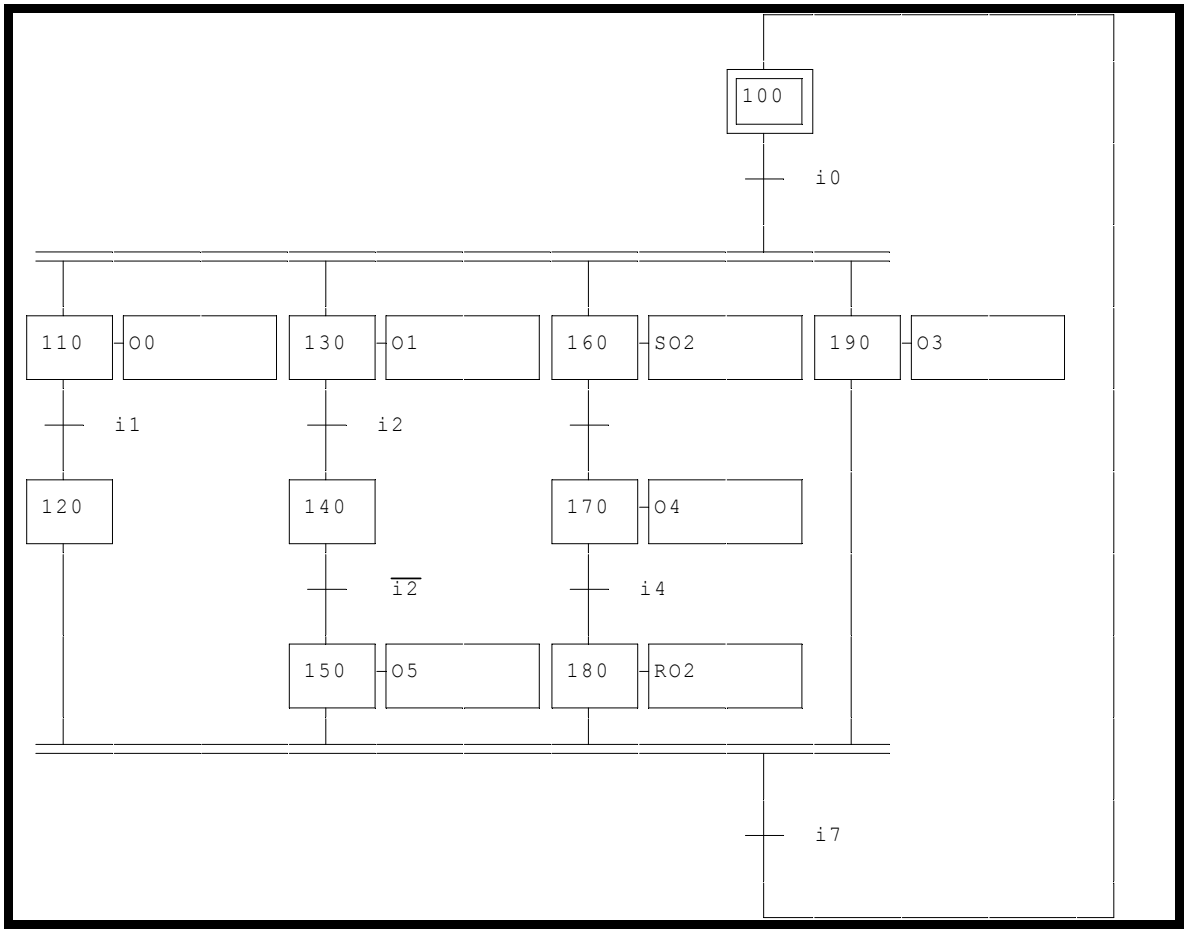
2.1.2. Grafcet con diagrama en O



exemple\grafcet\sample2.agn

Este ejemplo ilustra la utilización de las divergencias y convergencias en « O ». El número de ramas está limitado sólo por el tamaño del folio. Se trata, como establece la norma, de un « O » no exclusivo. Si por ejemplo las entradas 1 y 2 están activas, las etapas 120 y 130 se ponen en uno.

2.1.3. Grafcet con divergencia en Y

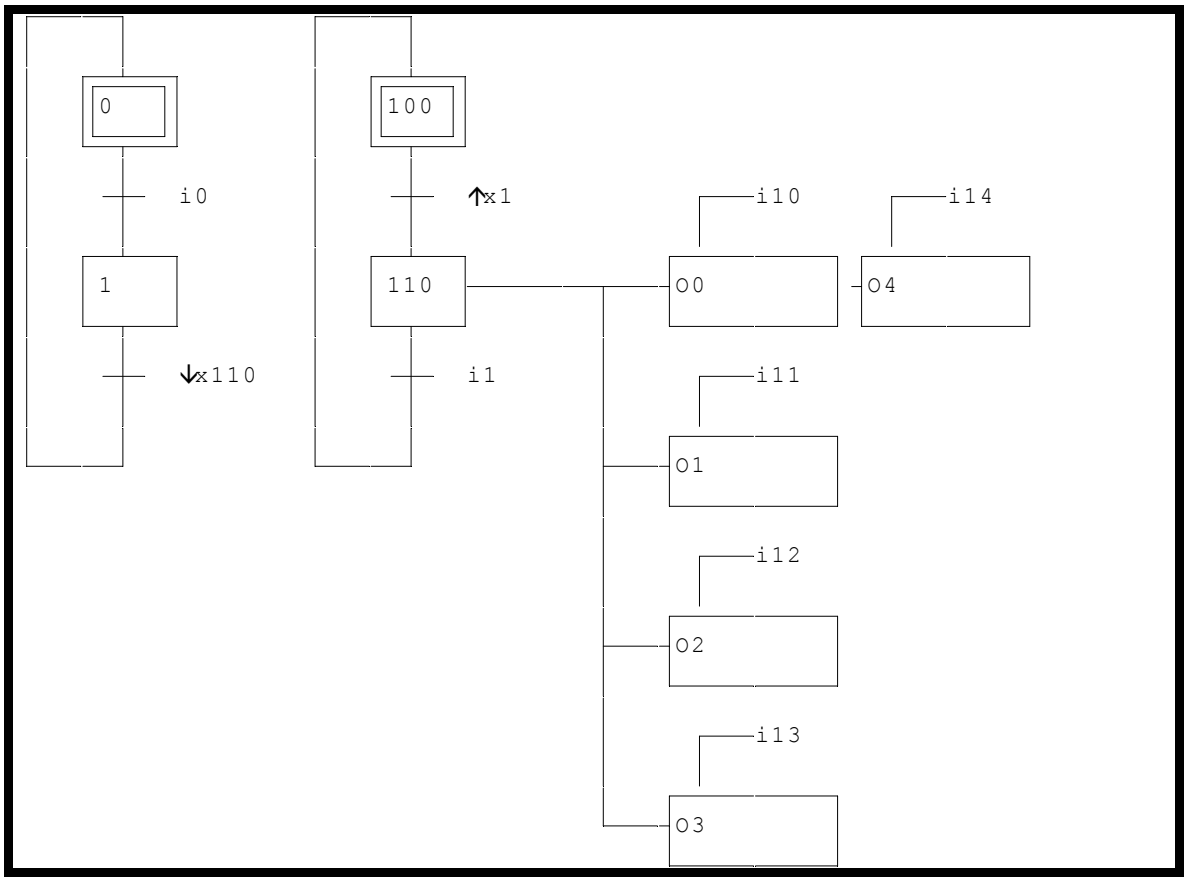


exemple\grafcet\sample3.agn

Este ejemplo ilustra la utilización de las divergencias y convergencias en « Y ». El número de ramas está limitado sólo por el tamaño del folio. Notemos también los puntos siguientes:

- ⇨ una etapa puede no implicar acciones (caso de las etapas 100, 120, y 140),
- ⇨ las órdenes « S » y « R » se han utilizado con la salida o2 (etapas 160 y 180),
- ⇨ la transición entre la etapa 160 y 170 queda en blanco, por lo que siempre es verdadera; también se podría haber utilizado la sintaxis « =1 ».

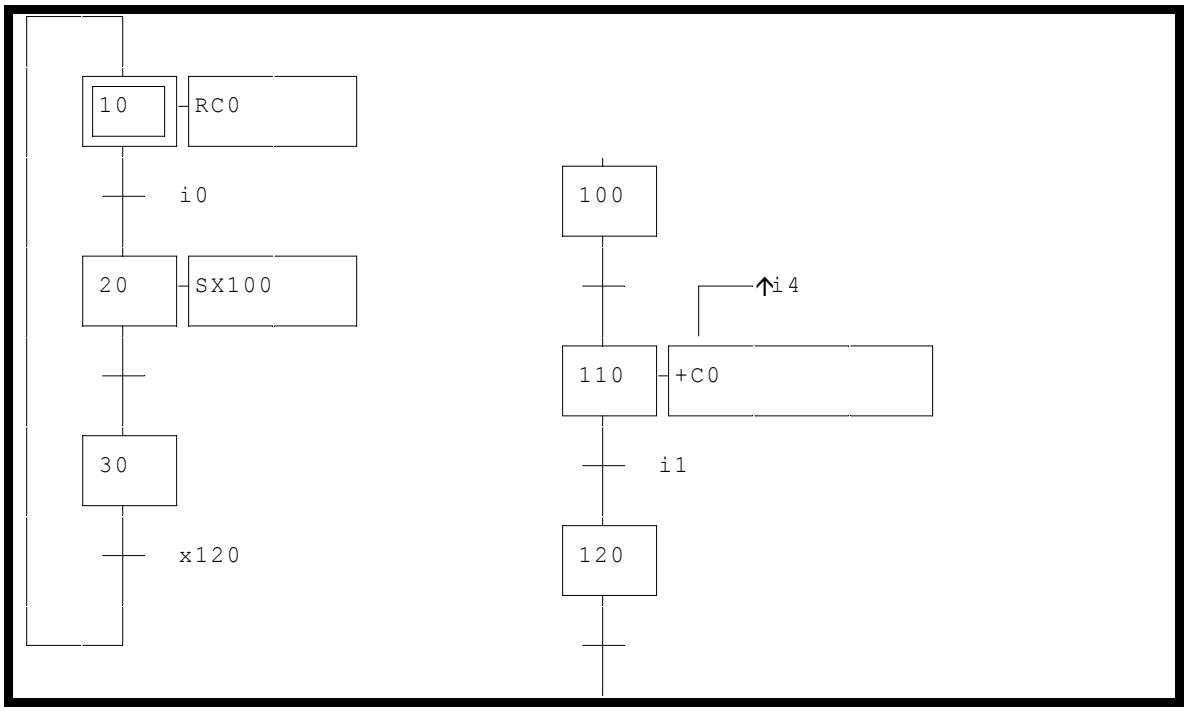
2.1.4. Grafcet y sincronización



exemple\grafcet\sample4.agn

Este ejemplo ilustra una de las posibilidades ofrecidas por AUTOMGEN para sincronizar varios Grafcets. La transición entre la etapa 100 y 110 « ↑x1 » significa « esperar un frente ascendente sobre la etapa 1 ». La transición « ↓x110 » significa « esperar un frente descendente sobre la etapa 110 ». La ejecución paso a paso de este programa muestra la evolución exacta de las variables y de su frente a cada ciclo. Esto permite comprender exactamente lo que pasa durante la ejecución. Notemos también la utilización de acciones múltiples asociadas a la etapa 110, que están condicionadas individualmente.

2.1.5. Forzado de etapas

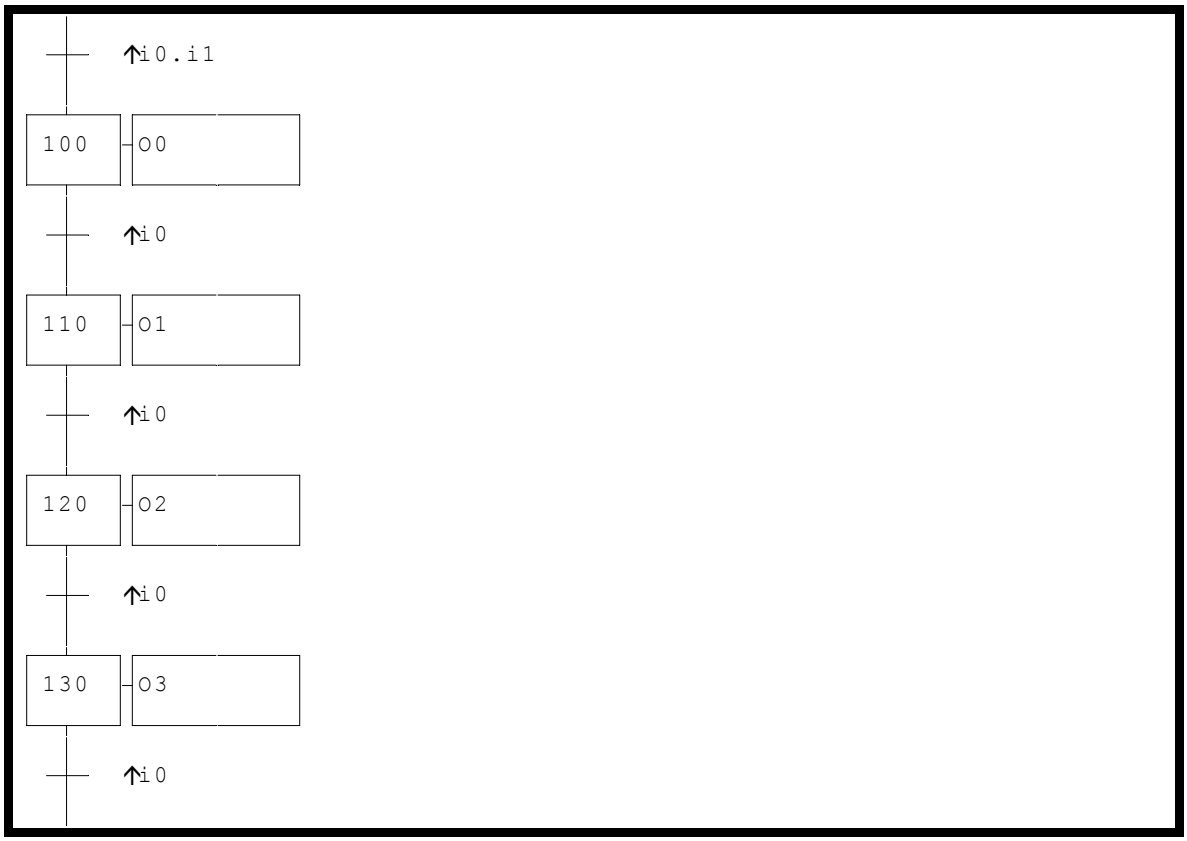


exemple\grafcet\sample5.agn

En este ejemplo, se ha utilizado una orden « S » (puesta en uno) para forzar una etapa. AUTOMGEN autoriza también el forzado de un Grafcet entero (ver ejemplos 8 y 9). El modo de ejecución paso a paso permite, también en este ejemplo, comprender de manera precisa la evolución del programa en el tiempo. Notemos también:

- la utilización de un Grafcet sin bucle (100, 110, 120),
- la utilización de la orden « RC0 » (puesta en cero del contador 0),
- la utilización de la orden « +C0 » (incrementar el contador 0), condicionada por el frente ascendente de la entrada 4; para ejecutar la incrementación del contador, la etapa 110 debe estar activa y un frente ascendente debe detectarse en la entrada 4.

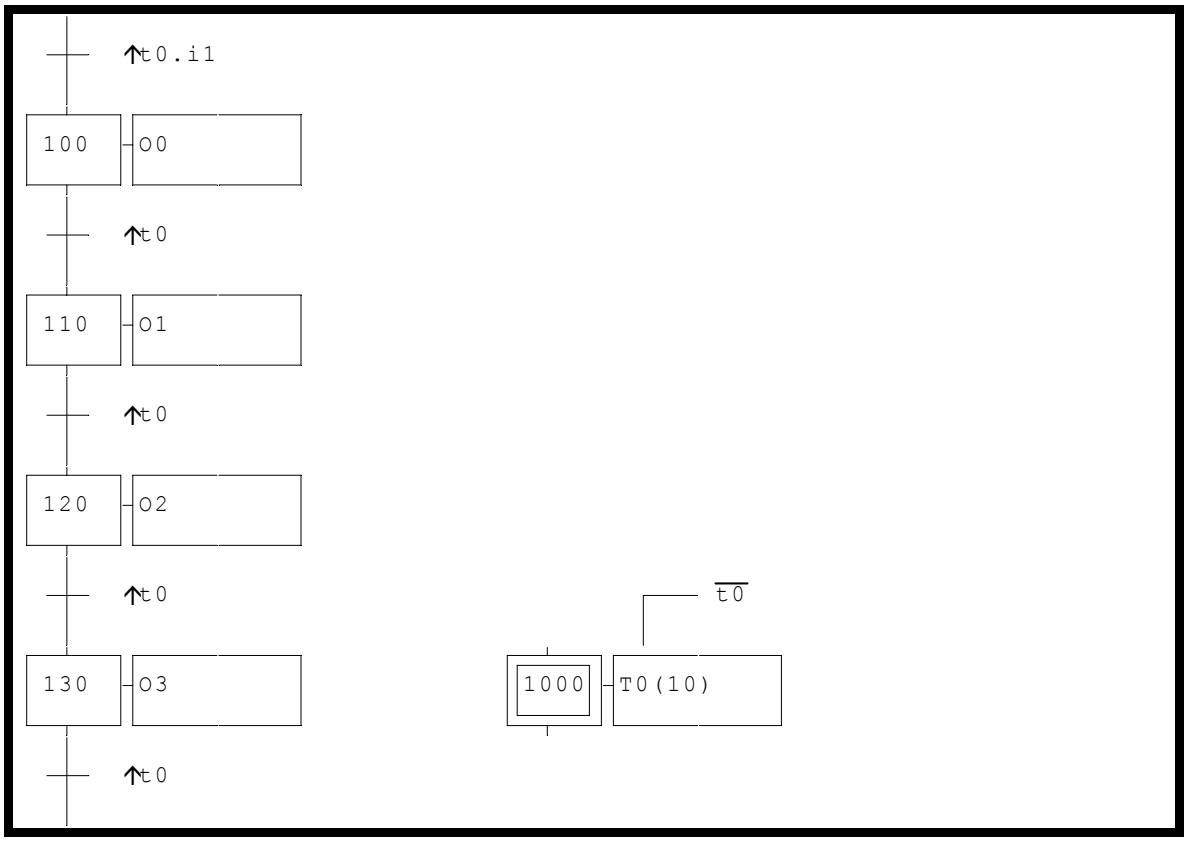
2.1.6. Etapas pozos y fuentes



exemple\grafcet\sample6.agn

Ya hemos visto formas similares en las que la primera etapa era activada por otro Grafcet. Aquí la etapa 100 es activada por la transición « ↑i0 . i1 » (frente ascendente de la entrada 0 y la entrada 1). Este ejemplo representa un registro de desfase. « i1 » es la información a memorizar en el registro y « i0 » es el reloj que hace progresar el desfase. El ejemplo 7 es una variante que utiliza una temporización como reloj.

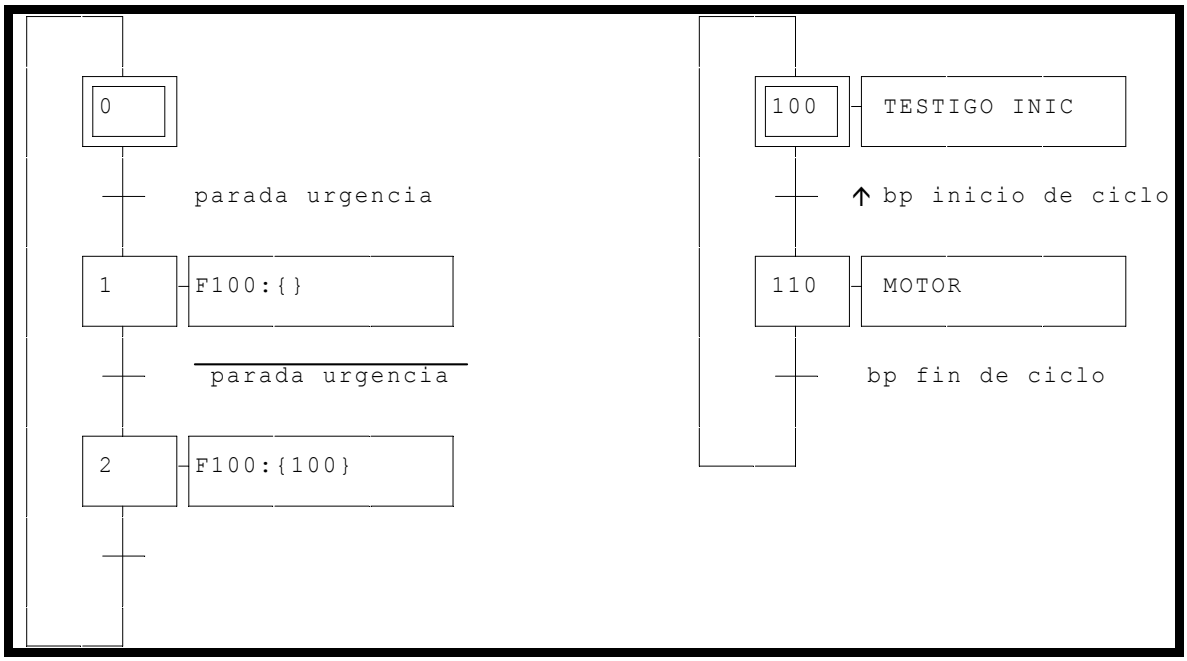
2.1.7. Etapas pozos y fuentes



exemple\grafcet\sample7.agn

Volvemos a ver la estructura de registro de desfase utilizada en el ejemplo 6. La información de desfase esta vez es generada por una temporización (t0). « ↑t0 » representa el frente ascendente de la temporización; esta información es verdadera durante un ciclo cuando la temporización ha terminado de descontar. La etapa 1000 gestiona el lanzamiento de la temporización. La acción de esta etapa puede resumirse así: « activar el descuento si éste no ha terminado; en caso contrario, restablecer la temporización ». El diagrama de funcionamiento de las temporizaciones de este manual ayuda a entender el funcionamiento de este programa.

2.1.8. Forzado de Grafcets

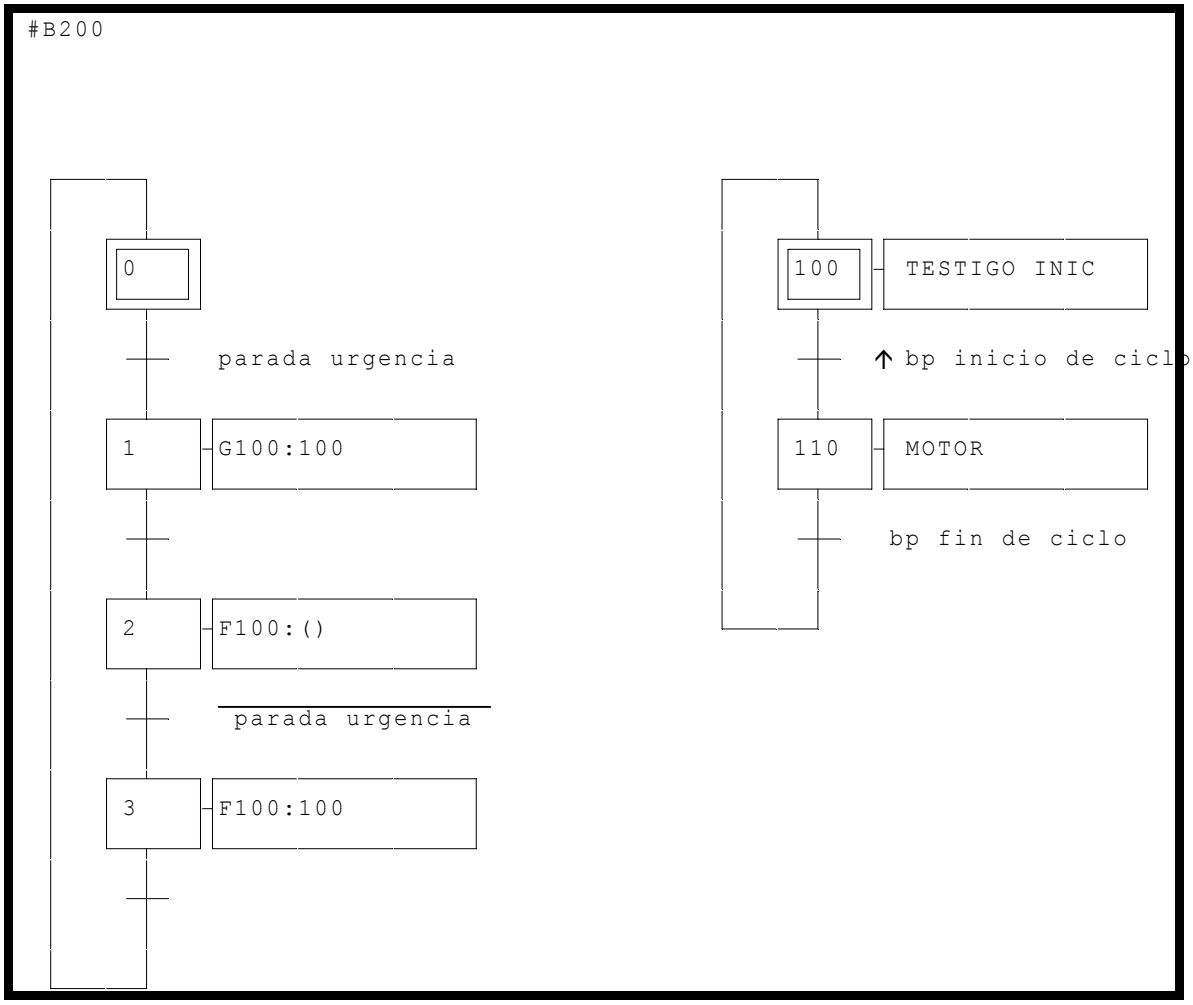


exemple\grafcet\sample8.agn

Este ejemplo ilustra la utilización de un comando de forzado de Grafcet. La orden « F100:{} » significa « forzar todas las etapas del Grafcet que tiene una etapa que lleva el número 100 a cero ». La orden « F100:{100} » es idéntica pero fuerza la etapa 100 a 1. Para este ejemplo hemos utilizado símbolos:

parada urgencia	i0
bp inicio de ciclo	i1
bp fin de ciclo	i2
TESTIGO INIC	o0
MOTOR	o1

2.1.9. Memorización de Grafcets



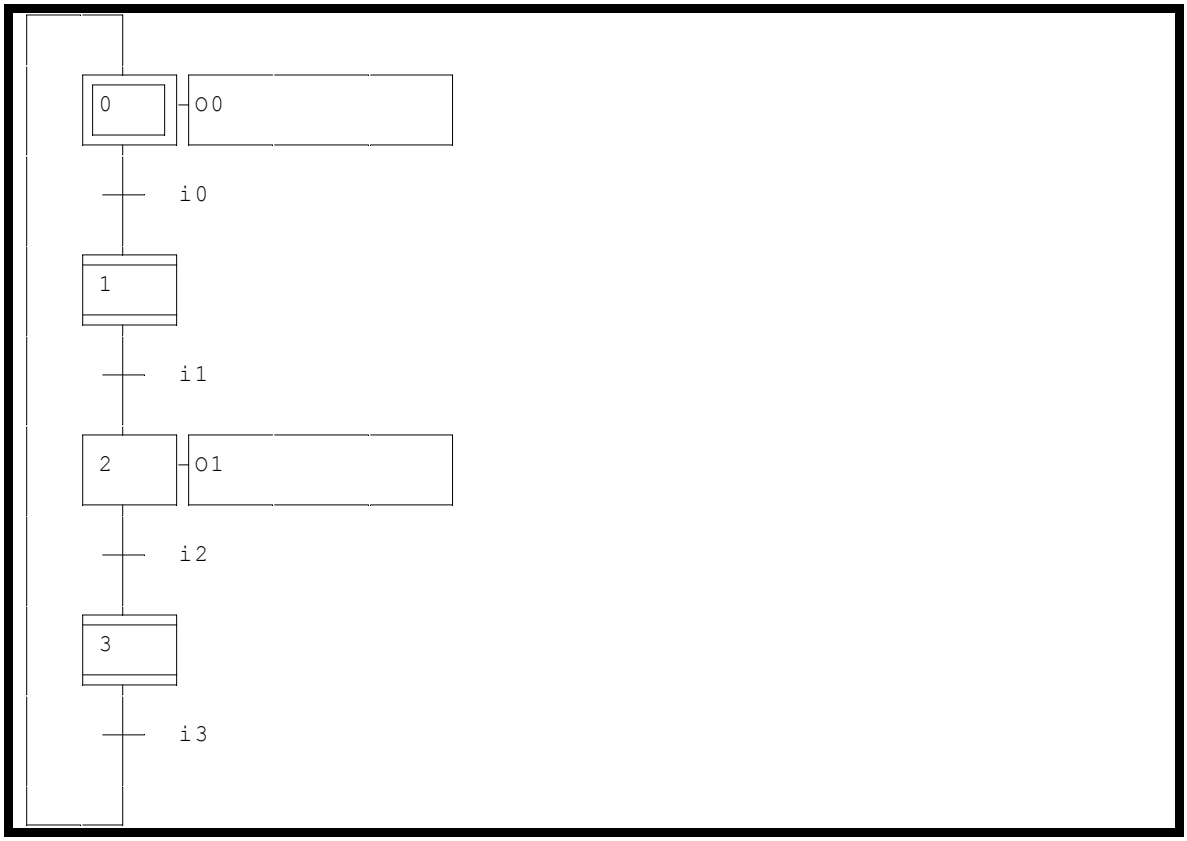
exemple\grafcet\sample9.agn

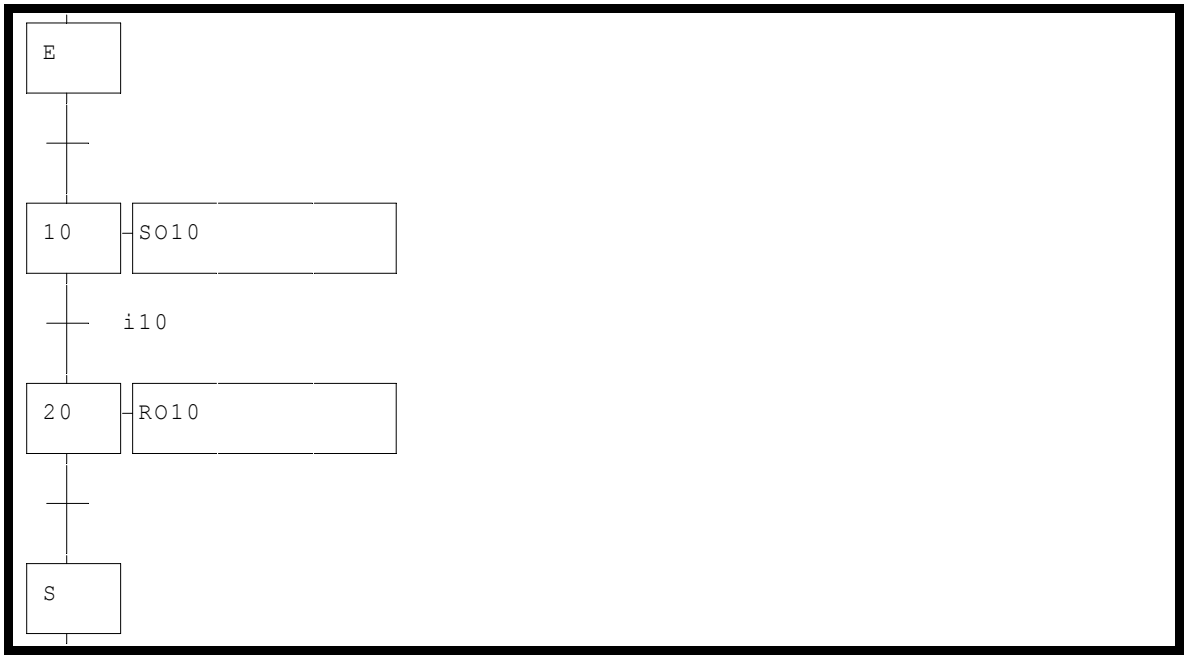
parada urgencia	i0
bp inicio de ciclo	i1
bp fin de ciclo	i2
MOTOR	o1
TESTIGO INIC	o0

Este ejemplo es una variante del programa anterior. La orden « G100:100 » de la etapa 1 memoriza el estado del Grafcet de producción antes de que se fuerce a cero. Al retomar, el Grafcet de producción se colocará de nuevo en el estado en que estaba antes del corte, con la orden « F100:100 ». El estado del Grafcet de producción se memoriza a partir del bit 100 (es el segundo parámetro de las órdenes « F » y « G » el que precisa esta ubicación), la directiva de compilación « #B200 » reserva los bits u100 a u199 para este tipo de utilización. Notemos que una directiva

« #B102 » habría bastado, ya que el Grafcet de producción sólo necesita dos bits para memorizarse (un bit por etapa).

2.1.10. Grafcet y macro-etapas

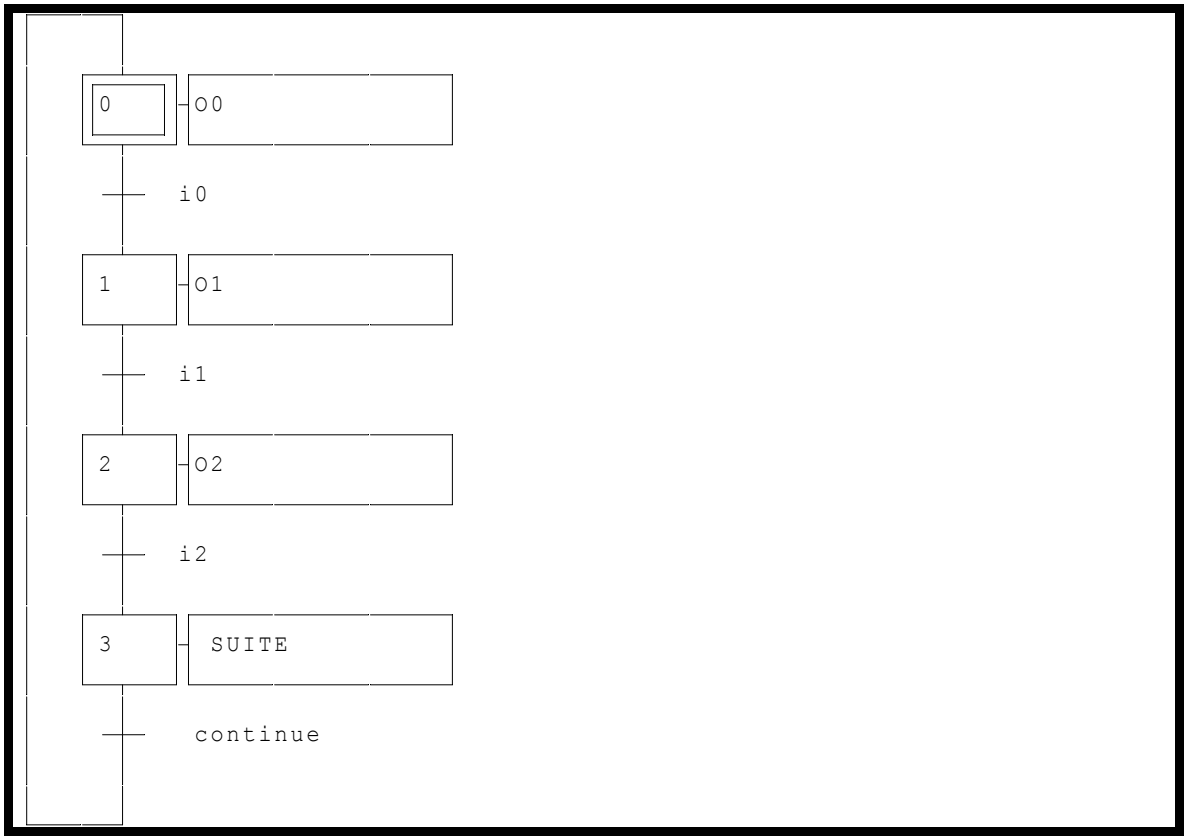


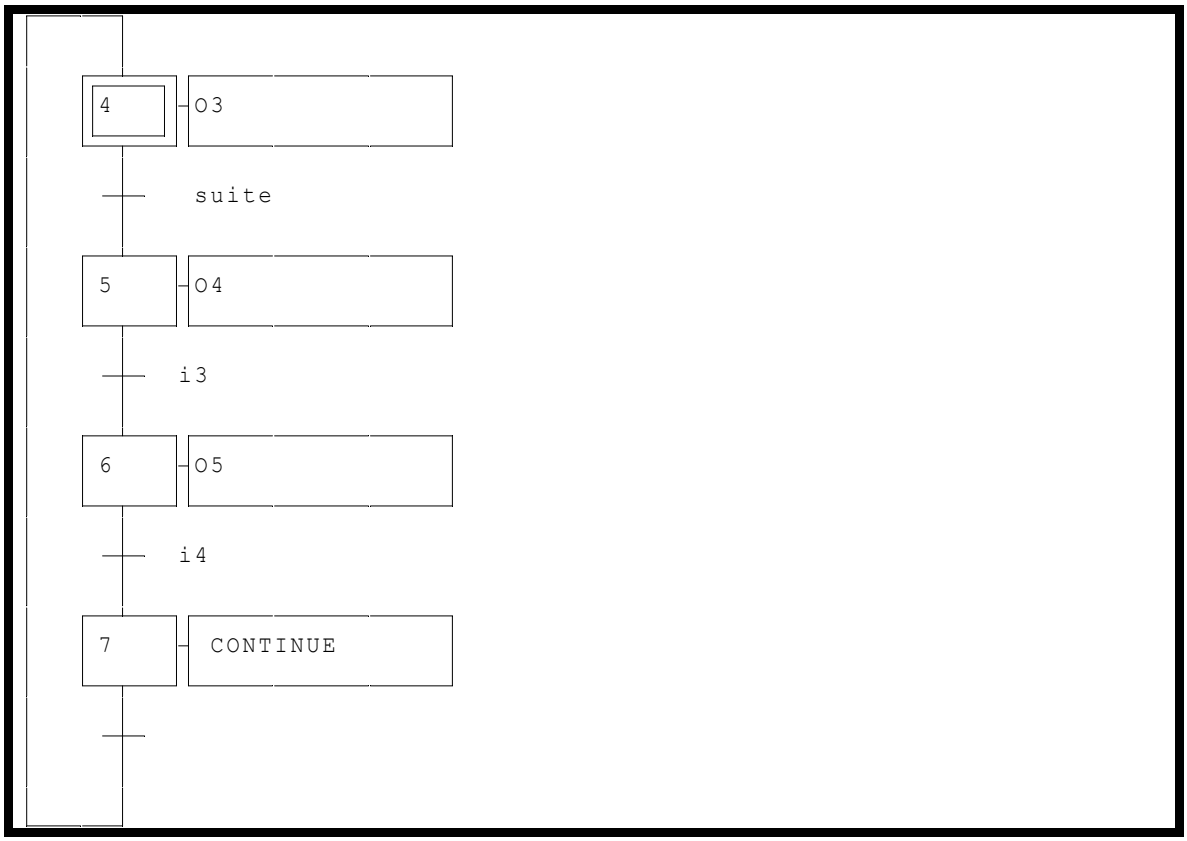


exemple\grafcet\sample11.agn

Este ejemplo ilustra la utilización de las macro-etapas. Los folios « Macro etapa 1 » y « Macro etapa 3 » representan la expansión de las macro-etapas con las etapas de entradas y de salidas. Las etapas 1 y 3 del folio « Programa principal » están definidas como macro-etapas. A las expansiones de macro-etapas en visualización se accede haciendo clic con el botón izquierdo del ratón en las macro-etapas.

2.1.11. Folios en cadena

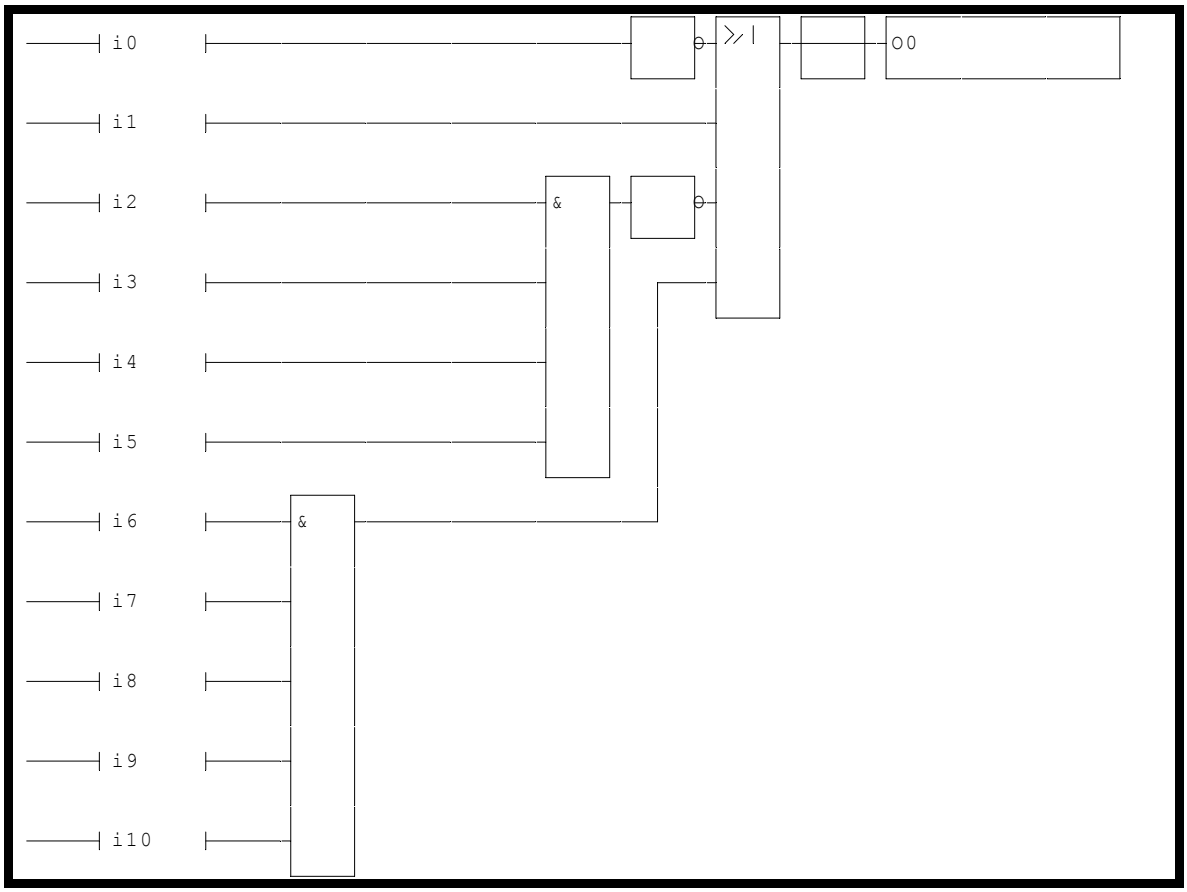




exemple\grafcet\sample12.agn

En este ejemplo se han utilizado dos folios para escribir el programa. Los símbolos « _CONTINUACIÓN_ » y « _CONTINÚA_ » se han declarado como bits (ver el fichero de símbolos) y permiten vincular los dos Grafcets (es otra técnica de sincronización que se puede utilizar con AUTOMGEN).

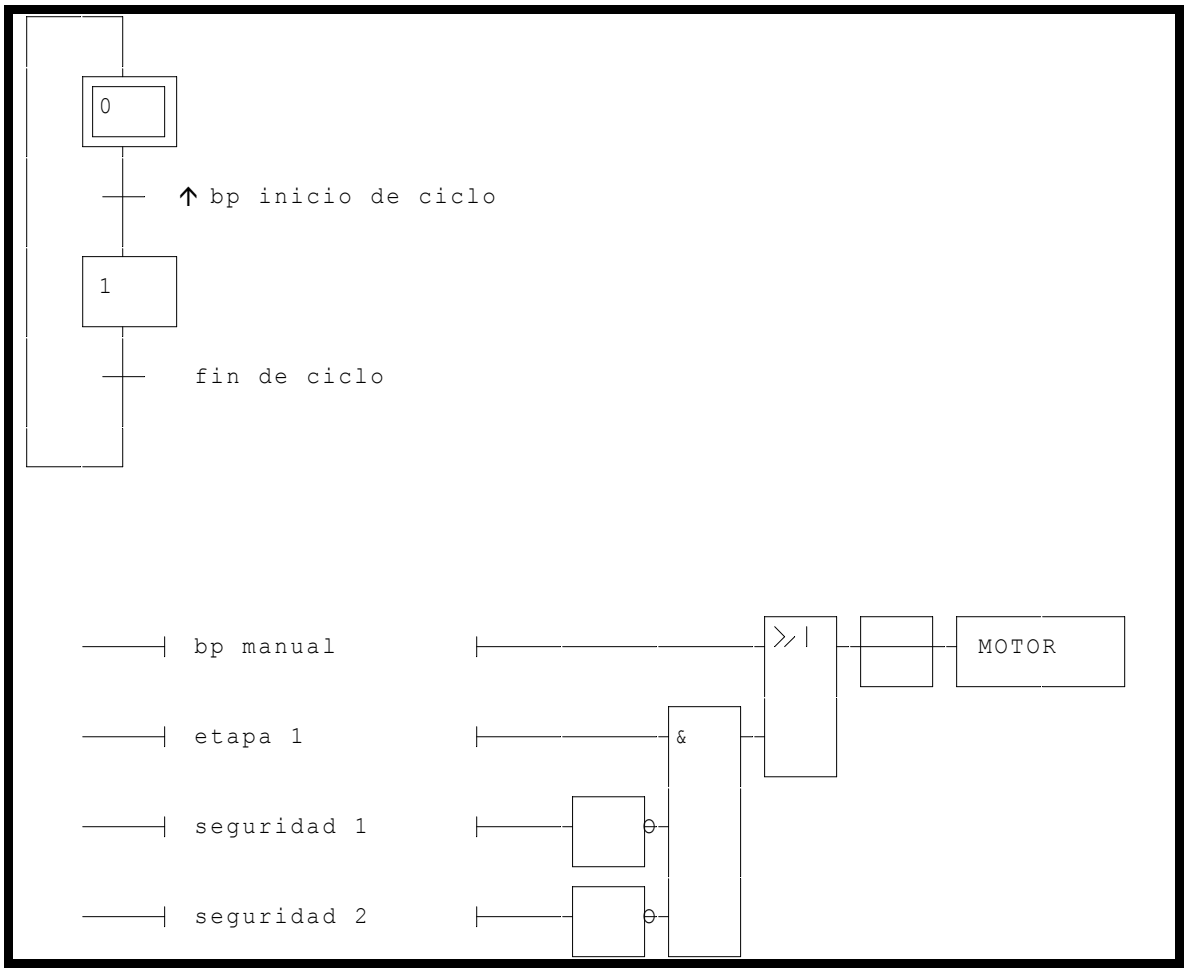
2.1.12. Logigrama



exemple\logigramme\sample14.agn

Este ejemplo desarrollado en logigramas muestra la utilización de los diferentes bloques: el bloque de asignación asociado a la tecla [0] a la izquierda del rectángulo de acción, el bloque « no » asociado a la tecla [1] que complementa una señal, los bloques de anclaje de tests y las funciones « Y » y « O ».

2.1.13. Grafcet y Logigrama

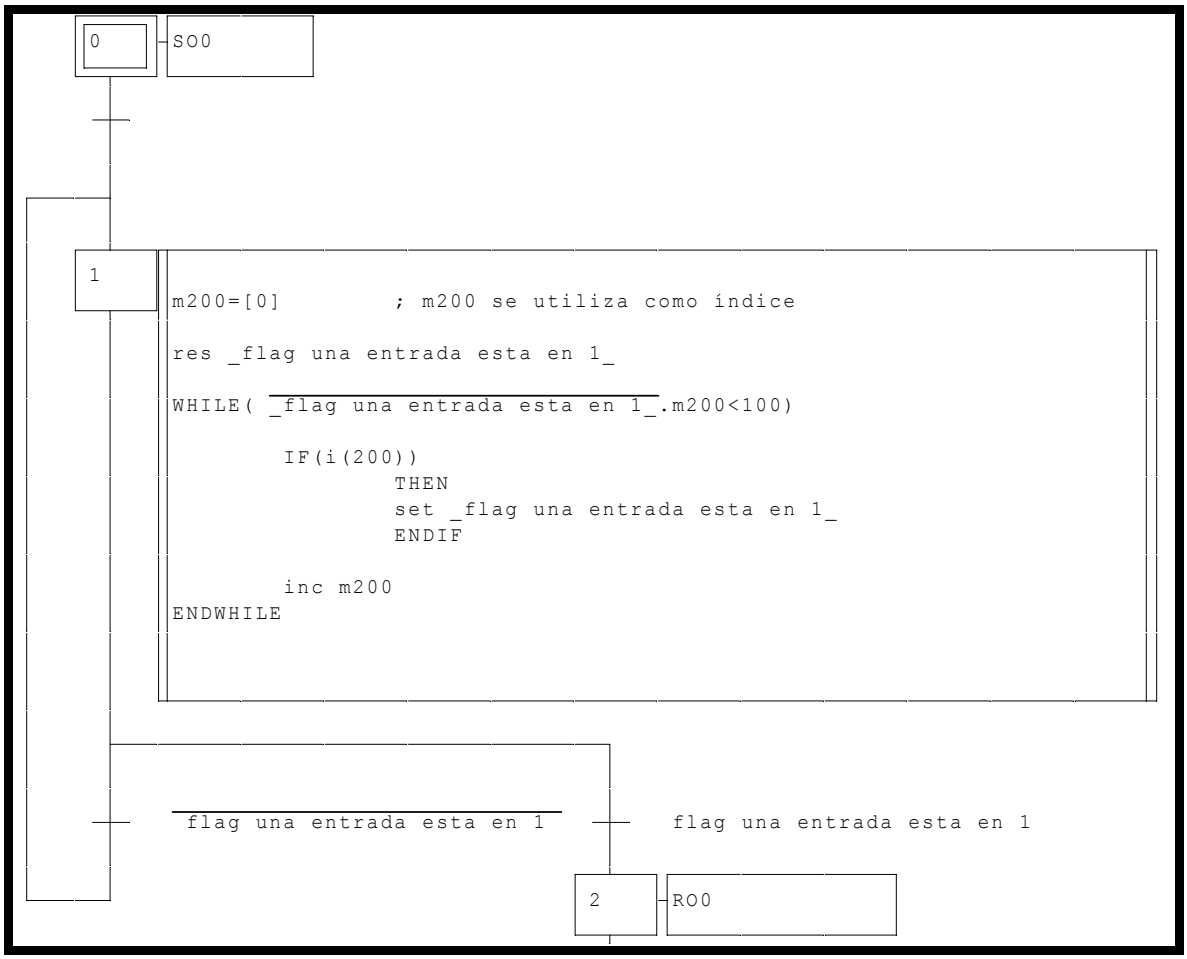


exemple\logigramme\exempl15.agn

En este ejemplo se utilizan conjuntamente un Grafcet y un Logigrama. El símbolo « _etapa 1_ » utilizado en el logigrama está asociado a la variable « x1 ».

Este tipo de programación presenta claramente las condiciones de activación de una salida.

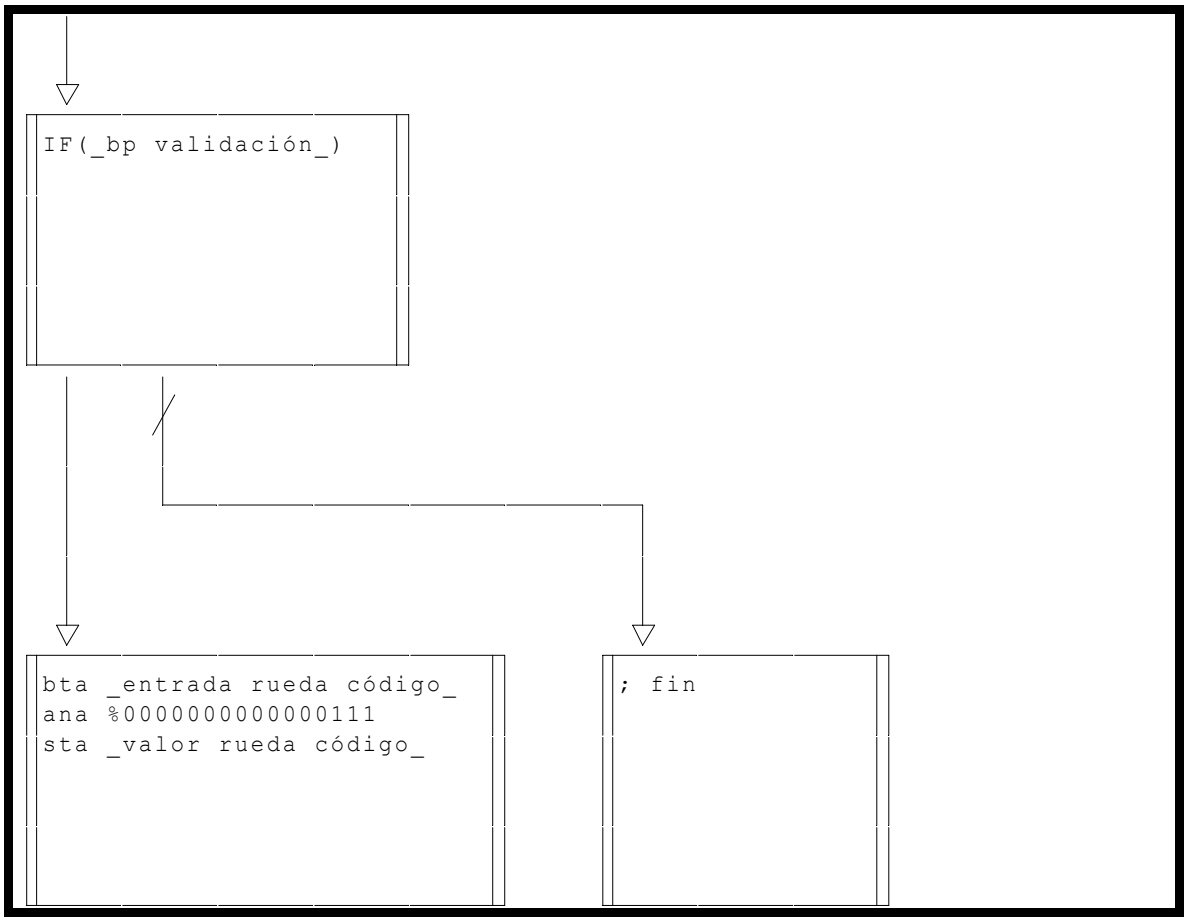
2.1.14. Caja de lenguaje literal



exemple\lit\sample16.agn

Este programa que asocia Grafcet con caja de lenguaje literal tiene por objeto testear las entradas i0 a i99. Si una de estas entradas está en uno, la etapa 2 está activada y el Grafcet se encuentra en un estado que impide cualquier evolución. El símbolo « _flag una entrada está en uno_ » está asociado al bit u500. Se ha utilizado un direccionamiento indexado para explorar las 100 entradas. Notemos también el empleo simultáneo del lenguaje literal bajo nivel y extendido.

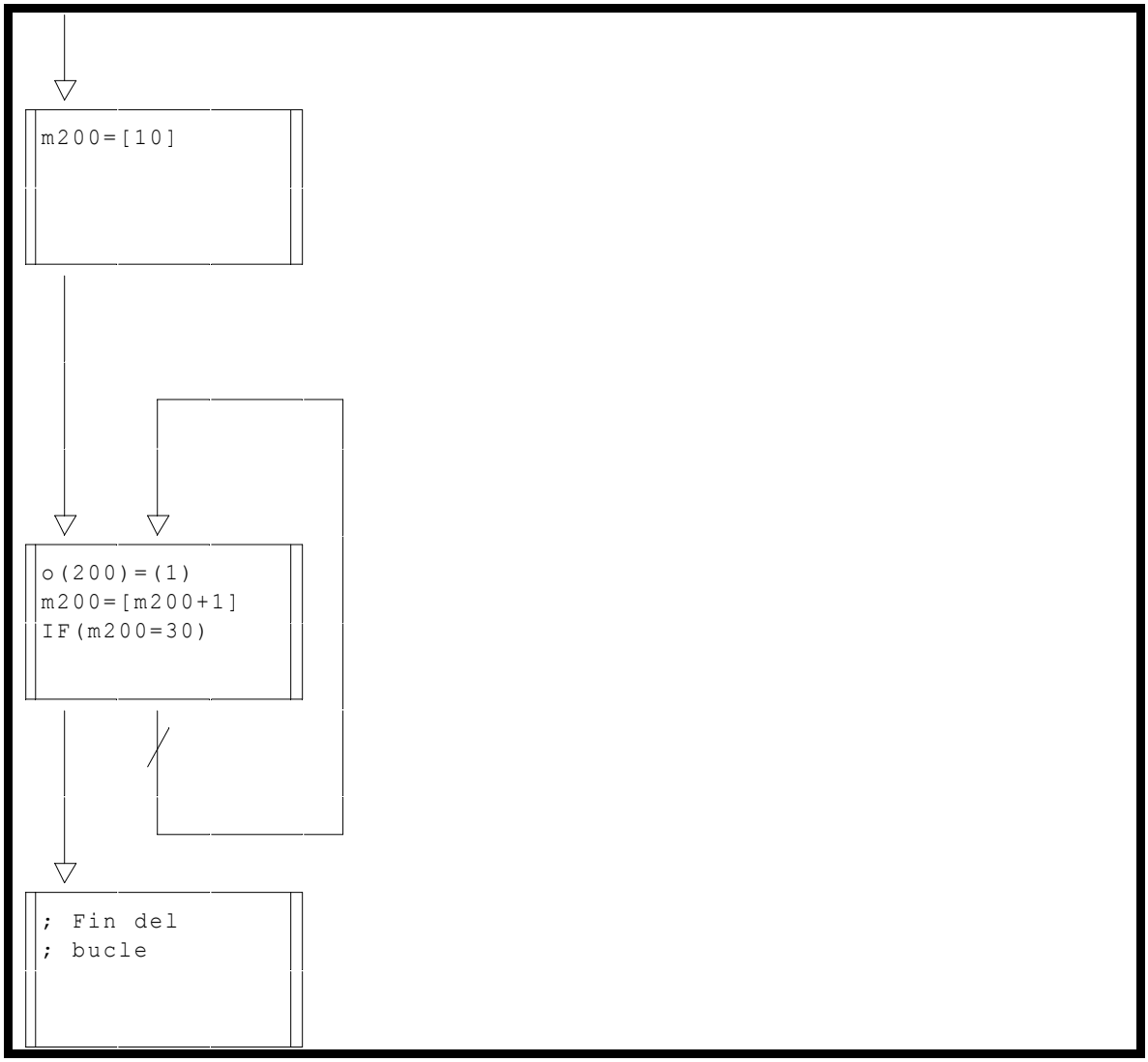
2.1.15. Organigrama



exemple\organigramme\sample18.agn

Este ejemplo ilustra la utilización de un organigrama para efectuar un tratamiento algorítmico y numérico. Tres entradas provenientes de una rueda de código son leídas y almacenadas en una palabra si una entrada de validación está activa.

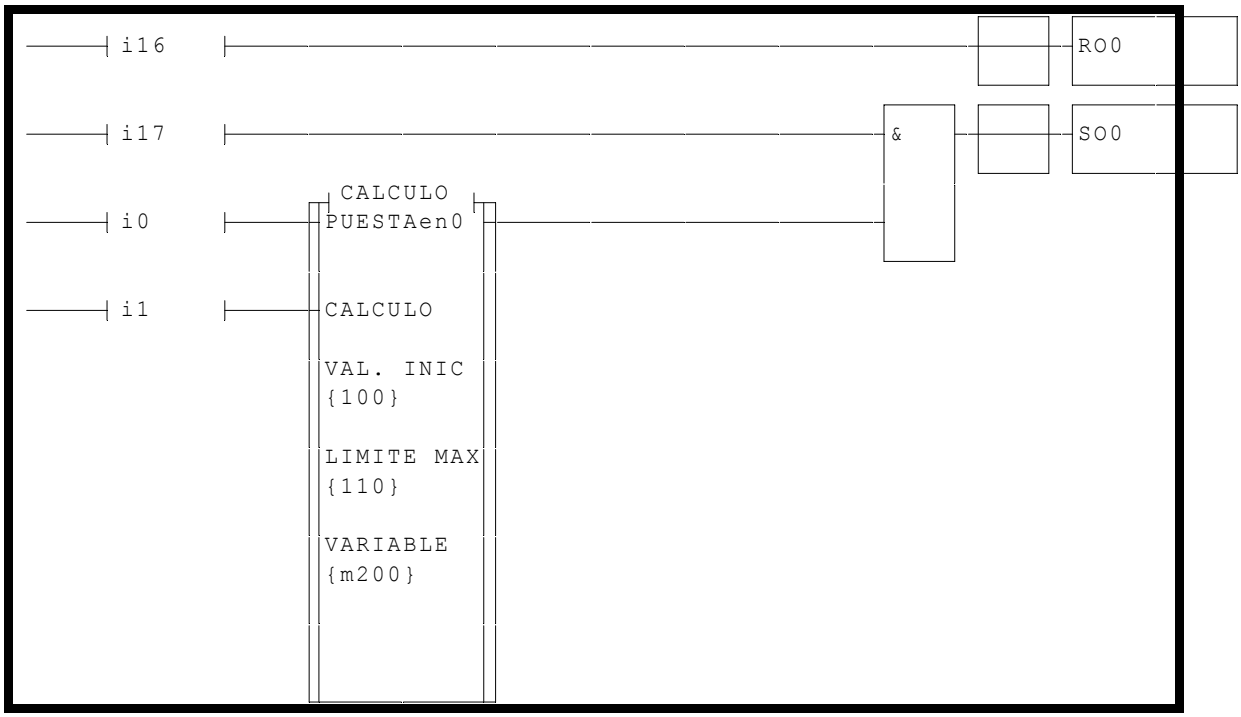
2.1.16. Organigrama



exemple\organigramme\sample19.agn

Este segundo ejemplo de organigrama realiza una estructura de bucle que permite forzar a uno una serie de salidas (o10 a o29) con un direccionamiento indirecto (« o(200) »).

2.1.17. Bloque funcional



exemple\bf\sample20.agn

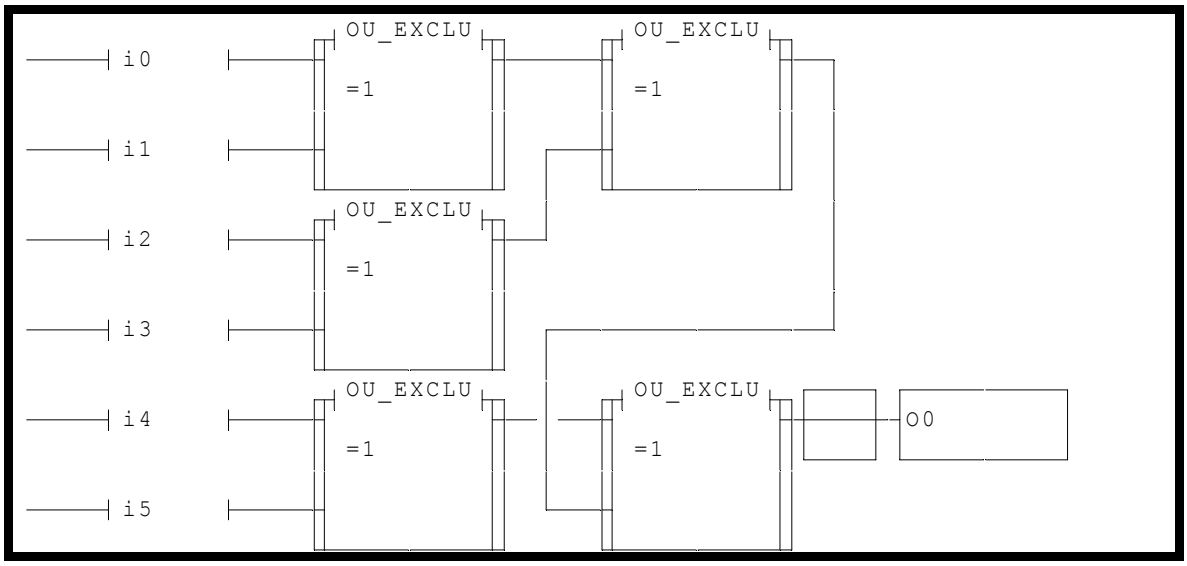
```
; Gestión de la entrada de RAZ
IF({I0})
        THEN
        {?2}=[{?0}]
        ENDIF

; Gestión de la entrada de cálculo

IF(#{I1})
        THEN
        {?2}=[{?2}+1]
        ENDIF

; Testea el límite máx


IF({?2}={?1})
        THEN
        {O0}=(1)
        {?2}=[{?0}]
        ENDIF
        ELSE
        {O0}=(0)
        ENDIF
```

comptage.lib (incluido en los recursos del proyecto)

Este ejemplo ilustra la utilización de un bloque funcional. Las funciones del bloque « COMPTAGE » que hemos definido son las siguientes:

- ⇨ el cálculo se hará partiendo de un valor de inicio y se terminará en un valor límite máximo,
- ⇨ cuando la variable de cálculo alcance el límite máximo será forzada al valor de inicio y la salida del bloque pasará a uno durante un ciclo de programa,
- ⇨ el bloque poseerá una entrada booleana de RAZ y una entrada de cálculo en el frente ascendente.

2.1.18. Bloque funcional



exemple\bf\sample21.agn

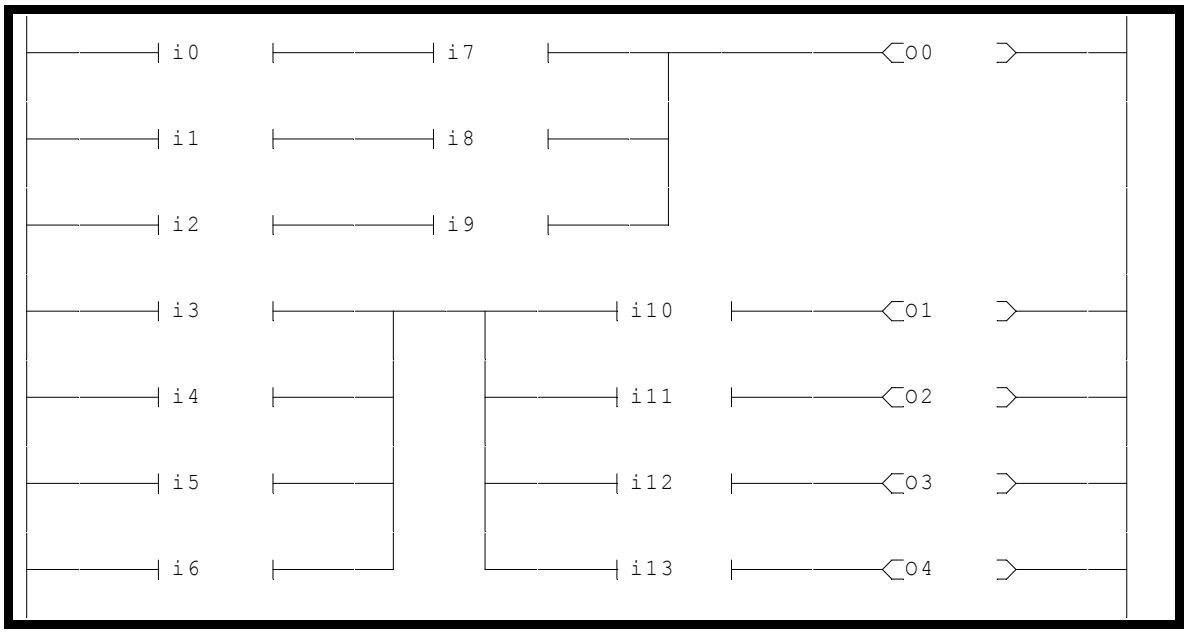
```
; O exclusivo
neq {o0} orr /{i0} eor {i1} orr {i0} eor /{i1}
```

ou_exclu.lib (incluido en los recursos del proyecto)

Este segundo ejemplo de bloque funcional ilustra la utilización múltiple de un mismo bloque. El bloque « OU_EXCLU » realiza un O exclusivo entre las dos entradas booleanas. Este ejemplo utiliza 5 bloques para realizar un O exclusivo entre 6 entradas (i0 a i5). El archivo « OU_EXCLU.LIB » indicado abajo rige el

funcionamiento del bloque. La ecuación booleana del O exclusivo es la siguiente: « (i0./i1)+(/i0.i1) ». La forma equivalente utilizada aquí permite codificar la ecuación en una sola línea de lenguaje literal bajo nivel sin utilizar variables intermedias.

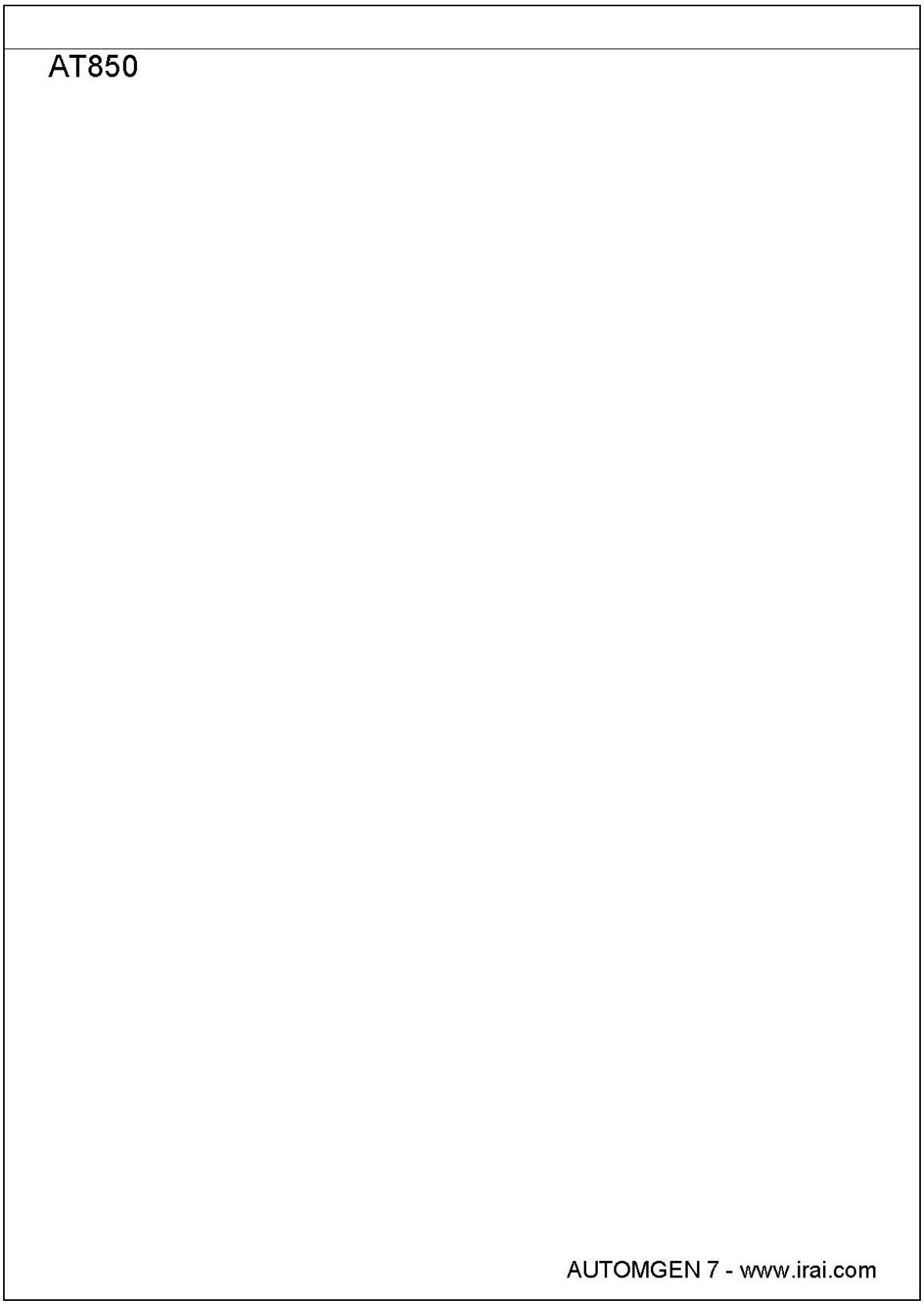
2.1.19. Ladder



exemple\laddersample22.agn

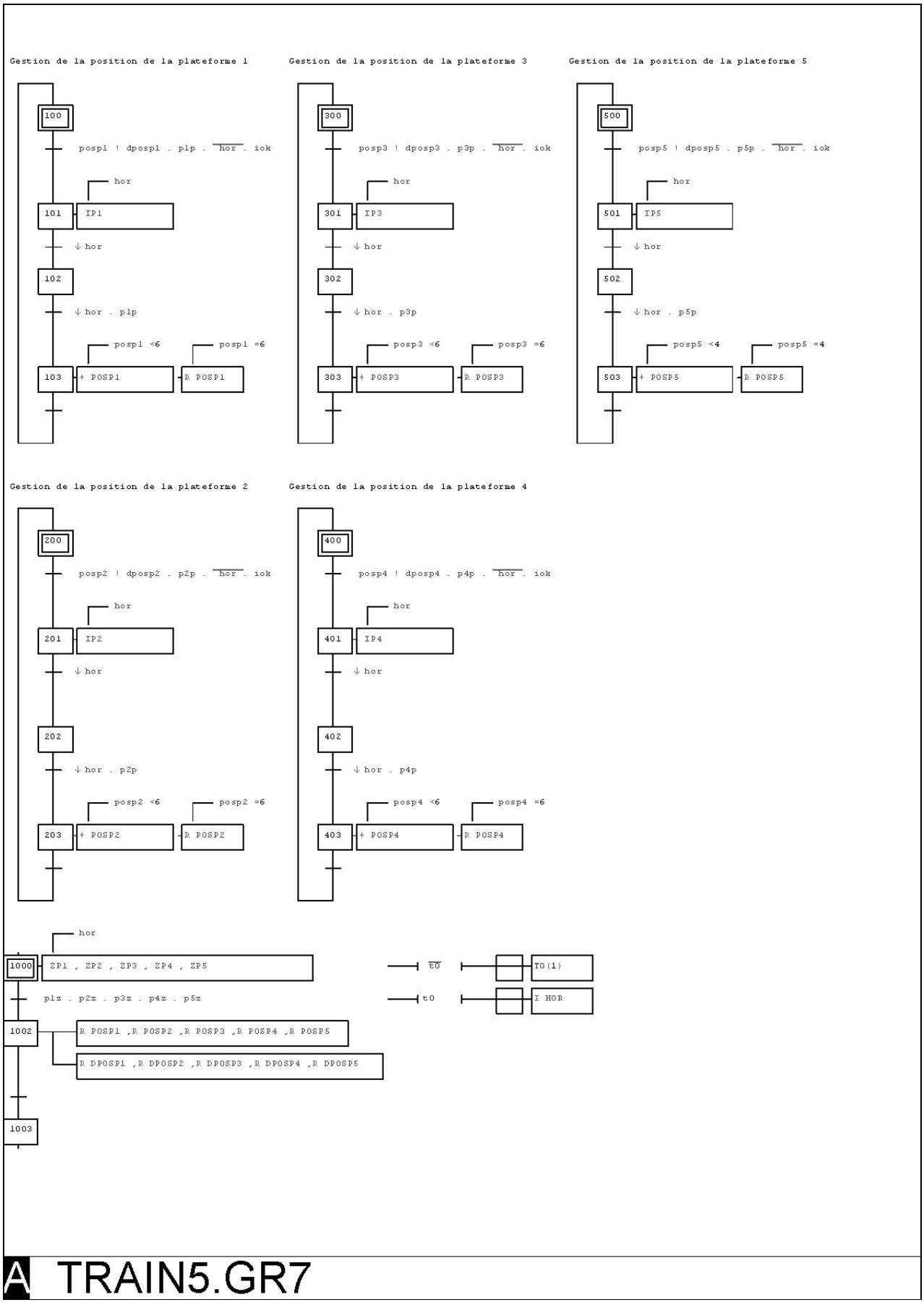
Este ejemplo ilustra la utilización de la programación en ladder.

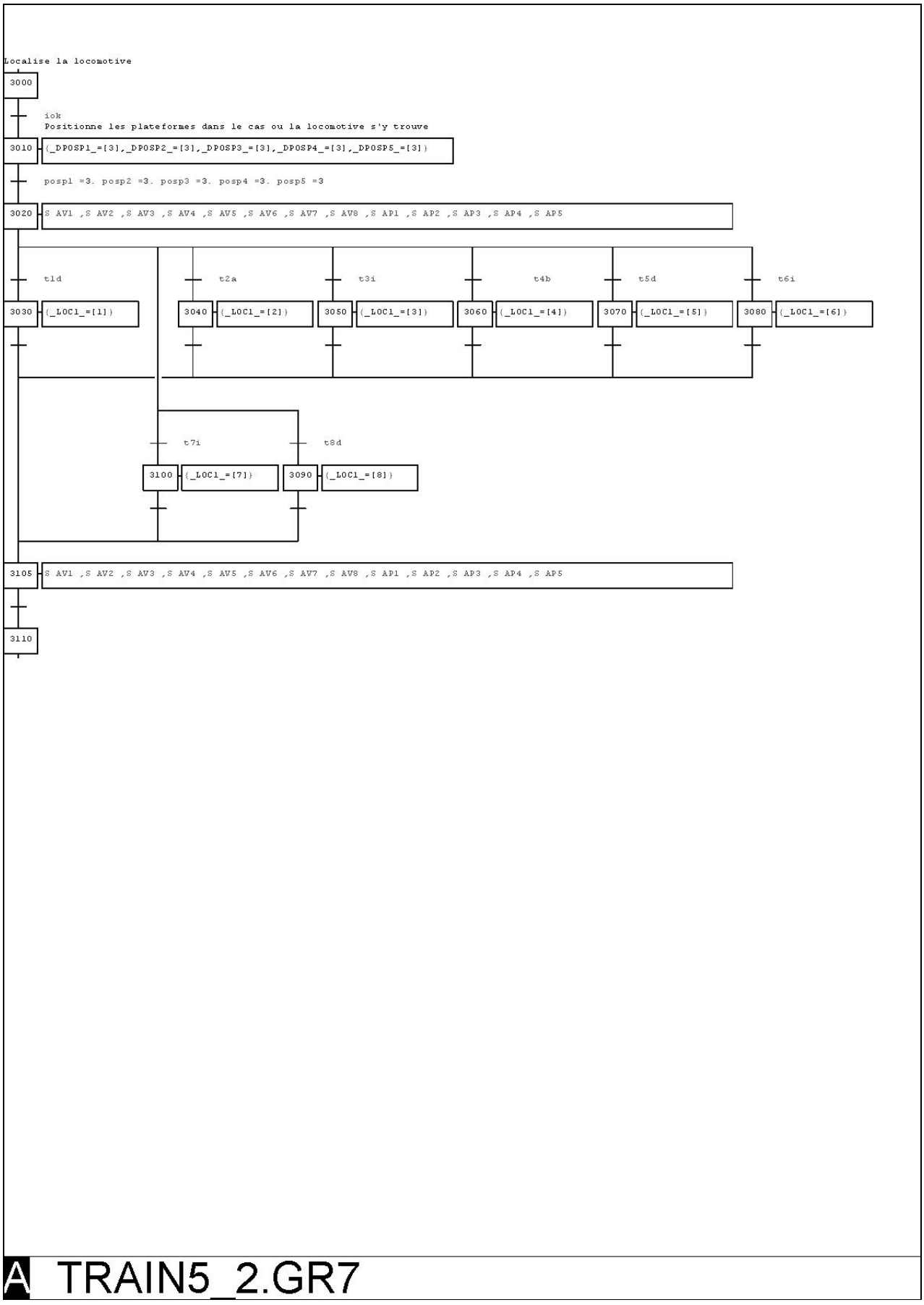
2.1.20. Ejemplo desarrollado sobre una maqueta de tren



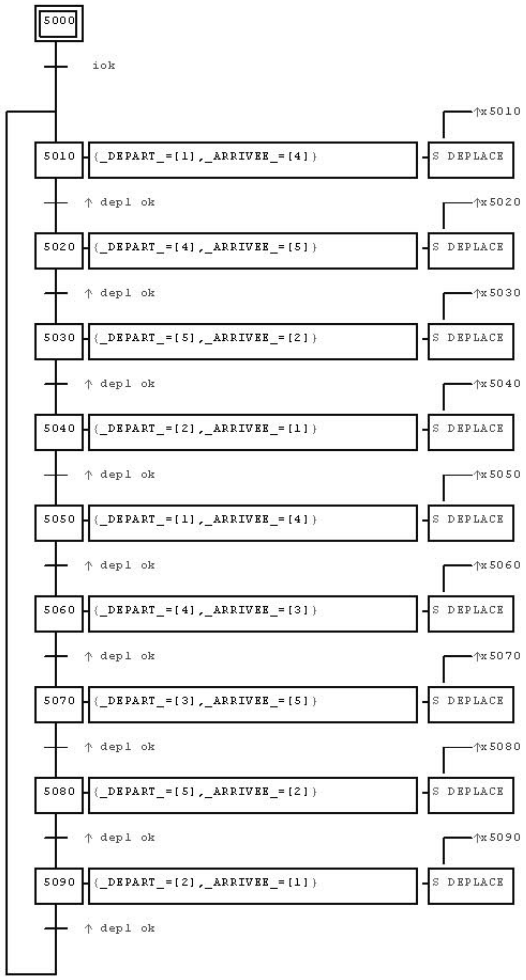
AV1	O0	alimentation voie 1
AV2	O1	alimentation voie 2
AV3	O2	alimentation voie 3
AV4	O3	alimentation voie 4
AV5	O4	alimentation voie 5
AV6	O5	alimentation voie 6
AV7	O6	alimentation voie 7
AV8	O7	alimentation voie 8
AP1	O8	alimentation plateforme 1
AP2	O9	alimentation plateforme 2
AP3	O10	alimentation plateforme 3
AP4	O11	alimentation plateforme 4
AP5	O12	alimentation plateforme 5
IP1	O13	rotation plateforme 1
IP2	O14	rotation plateforme 2
IP3	O15	rotation plateforme 3
IP4	O16	rotation plateforme 4
IP5	O17	rotation plateforme 5
ZP1	O18	initialisation plateforme 1
ZP2	O19	initialisation plateforme 2
ZP3	O20	initialisation plateforme 3
ZP4	O21	initialisation plateforme 4
ZP5	O22	initialisation plateforme 5
DV1	O23	direction voie 1
DV2	O24	direction voie 2
DV3	O25	direction voie 3
DV4	O26	direction voie 4
DV5	O27	direction voie 5
DV6	O28	direction voie 6
DV7	O29	direction voie 7
DV8	O30	direction voie 8
S1D	O31	feu droit voie 1
S1I	O32	feu gauche voie 1
S2A	O33	feu haut voie 2
S2B	O34	feu bas voie 2
S3D	O35	feu droit voie 3
S3I	O36	feu gauche voie 3
S4A	O37	feu haut voie 4
S4B	O38	feu bas voie 4
S5D	O39	feu droit voie 5
S5I	O40	feu gauche voie 5
S6D	O41	feu droit voie 6
S6I	O42	feu gauche voie 6
S7D	O43	feu droit voie 7
S7I	O44	feu gauche voie 7
S8D	O45	feu droit voie 8
S8I	O46	feu gauche voie 8
T1D	i0	train droit voie 1
T1I	i1	train gauche voie 1
T2A	i2	train haut voie 2
T2B	i3	train bas voie 2
T3D	i4	train droit voie 3
T3I	i5	train gauche voie 3
T4A	i6	train haut voie 4
T4B	i7	train bas voie 4
T5D	i8	train droit voie 5

1/1 Symboles


Gestion de la position de la plateforme 1
100
posp1 ! dposp1 . p1p . hor . iok
hor
101
IP1
↓ hor
102
↓ hor . p1p
posp1 <6
posp1 =6
103
+ POSP1
R POSP1
Gestion de la position de la plateforme 3
300
posp3 ! dposp3 . p3p . hor . iok
hor
301
IP3
↓ hor
302
↓ hor . p3p
posp3 <6
posp3 =6
303
+ POSP3
R POSP3
Gestion de la position de la plateforme 5
500
posp5 ! dposp5 . p5p . hor . iok
hor
501
IP5
↓ hor
502
↓ hor . p5p
posp5 <4
posp5 =4
503
+ POSP5
R POSP5
Gestion de la position de la plateforme 2
200
posp2 ! dposp2 . p2p . hor . iok
hor
201
IP2
↓ hor
202
↓ hor . p2p
posp2 <6
posp2 =6
203
+ POSP2
R POSP2
Gestion de la position de la plateforme 4
400
posp4 ! dposp4 . p4p . hor . iok
hor
401
IP4
↓ hor
402
↓ hor . p4p
posp4 <6
posp4 =6
403
+ POSP4
R POSP4
hor
1000
ZP1 , ZP2 , ZP3 , ZP4 , ZP5
t0
T0(1)
p1z . p2z . p3z . p4z . p5z
t0
I HOR
1002
R POSP1 ,R POSP2 ,R POSP3 ,R POSP4 ,R POSP5
R DPOSP1 ,R DPOSP2 ,R DPOSP3 ,R DPOSP4 ,R DPOSP5
1003
A TRAIN5.GR7


Localise la locomotive
3000
iok
Positionne les plateformes dans le cas ou la locomotive s'y trouve
3010
{_DPOSP1_=[3],_DPOSP2_=[3],_DPOSP3_=[3],_DPOSP4_=[3],_DPOSP5_=[3]}
posp1 =3. posp2 =3. posp3 =3. posp4 =3. posp5 =3
3020
S AV1 ,S AV2 ,S AV3 ,S AV4 ,S AV5 ,S AV6 ,S AV7 ,S AV8 ,S AP1 ,S AP2 ,S AP3 ,S AP4 ,S AP5
t1d
t2a
t3i
t4b
t5d
t6i
3030
{_LOC1_=[1]}
3040
{_LOC1_=[2]}
3050
{_LOC1_=[3]}
3060
{_LOC1_=[4]}
3070
{_LOC1_=[5]}
3080
{_LOC1_=[6]}
t7i
t8d
3100
{_LOC1_=[7]}
3090
{_LOC1_=[8]}
3105
S AV1 ,S AV2 ,S AV3 ,S AV4 ,S AV5 ,S AV6 ,S AV7 ,S AV8 ,S AP1 ,S AP2 ,S AP3 ,S AP4 ,S AP5
3110
A
TRAIN5_2.GR7

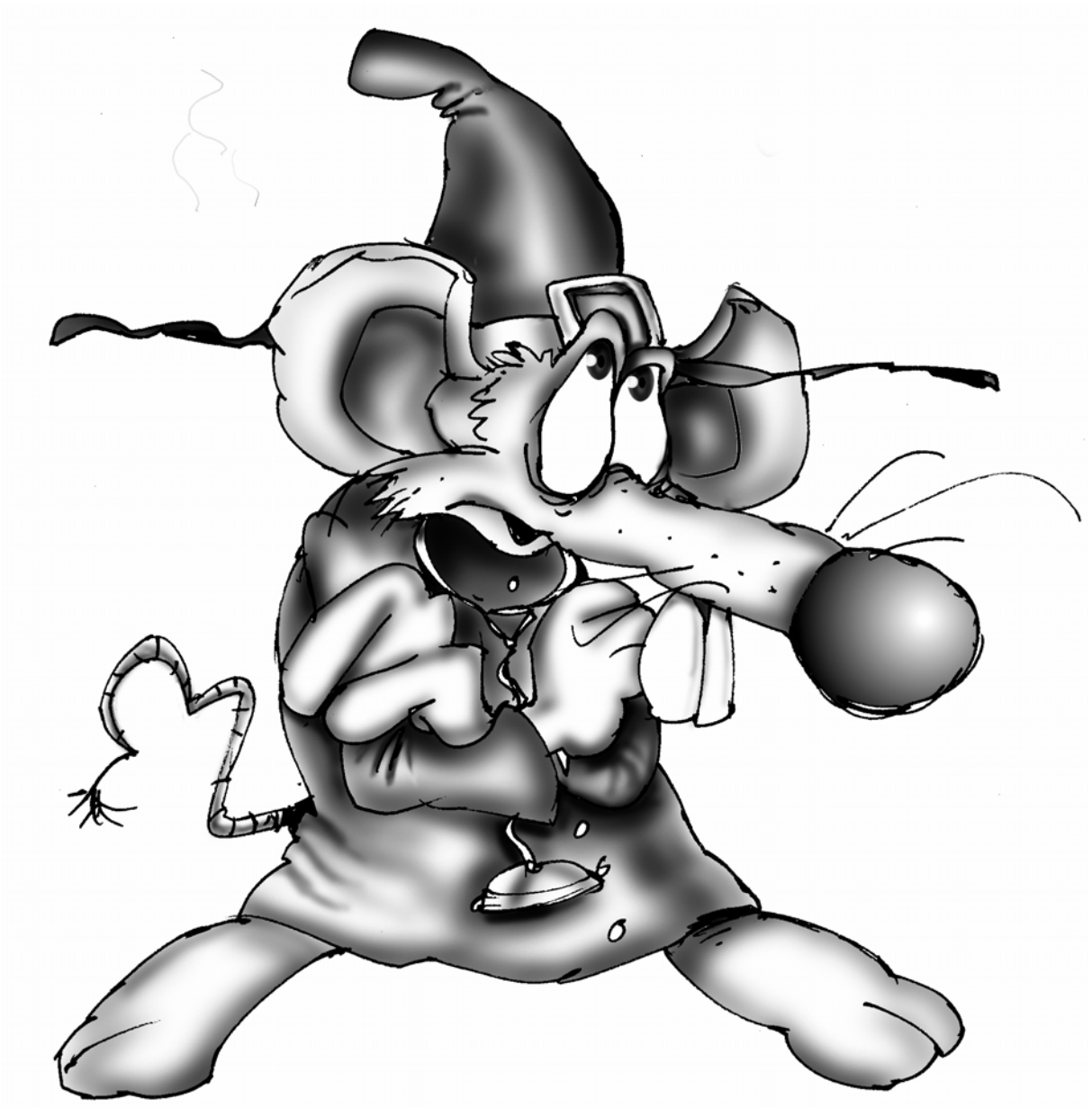

sous-programme : déplace la locomotive de la plateforme (_depart_) à la plateforme (_arrivee_) déplacement élémentaire
#$_rotation plateforme_=0,?_dposp1_,?_dposp2_,?_dposp3_,?_dposp4_,?_dposp5_
#$_alimentation plateforme_=0,?_ap1_,?_ap2_,?_ap3_,?_ap4_,?_ap5_
#$_direction depart_=1,4,3,2,1, 4,3,3,2,4, 3,2,3,2,3, 2,1,3,2,2, 5,1,0,1,6, 5,2,3,1,7, 4,5,4,3,5, 3,5,4,0,8, 0,0,0,0,0
#$_alimentation voie_=0,?_av1_,?_av2_,?_av3_,?_av4_,?_av5_,?_av6_,?_av7_,?_av8_
#$_presence plateforme_=0,?_tp1_,?_tp2_,?_tp3_,?_tp4_,?_tp5_
#$_presence voie_=0,?_td1_,?_t2a_,?_t3i_,?_t4b_,?_t5i_,?_t6i_,?_t7i_,?_t8I_
↑ deplace
600
(_INDEX_=[?_DIRECTION DEPART_]),R DEPLACE ,R DEPL OK ,R DEPL ERR
depart ! arrivee
601
(_TEST DEPART_=[M(_INDEX_)],INC _INDEX_)
(_TEST ARRIVEE_=[M(_INDEX_)],INC _INDEX_)
(_DIRECTION_=[M(_INDEX_)],INC _INDEX_)
(_DIRECTION2_=[M(_INDEX_)],INC_INDEX_)
(_VOIE_=[M(_INDEX_)],INC _INDEX_)
voie !0. depart = test depart . arrivee = test arrivee
voie !0.(depart ! test depart + arrivee ! test arrivee)
602
index2=[_depart_+?_rotation plateforme_]
index=[m(_index2_)]
m(_index_)=[_direction_]
dec _index_
pos depart=[m(_index_)]
index2=[_arrivee_+?_rotation plateforme_]
index=[m(_index2_)]
m(_index_)=[_direction2_]
dec _index_
pos arrivee=[m(_index_)]
pos depart = direction
603
index2=[_arrivee_+?_presence plateforme_]
index3=[M(_index2_)]
index2=[_voie_+?_presence voie_]
fin voie=[M(_index2_)]
index2=[_depart_+?_alimentation plateforme_]
alimp1=[M(_index2_)]
index2=[_arrivee_+?_alimentation plateforme_]
alimp2=[M(_index2_)]
index2=[_voie_+?_alimentation voie_]
alimv=[M(_index2_)]
?(I(_index3_)) ; Train sur la plateforme d'arrivee ?
->(
set _depl ok_
res 0(_alimp1_)
res 0(_alimp2_)
res 0(_alimv_)
)<-
->/(
set 0(_alimp1_)
set 0(_alimp2_)
pos arrivee=[m(_index_)]
equ 0(_alimv_) orr I(_fin voie_) eor _pos arrivee_=_direction2_
)<-
depl ok
voie =0
604
S DEPL ERR
A TRAIN5_3.GR7



A TRAIN5_4.GR7

Las aventuras del Doctor R.

Manual pedagógico del usuario de AUTOMGEN



Dibujo de Taban

Distribución

Doctor R. …… Señor R.

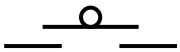
El Doctor R. en el reino de la domótica

Vamos a abordar diferentes ejemplos aplicables directamente en un proyecto de domótica. Con un primer enfoque simple, estos ejemplos nos permitirán asimilar diferentes aspectos de la base de los automatismos y del aprendizaje de AUTOMGEN e IRIS.

Este símbolo evocará una parte comando (un autómata programable por ejemplo).

¿Hay necesidad de precisar que esto

evocará una bombilla,

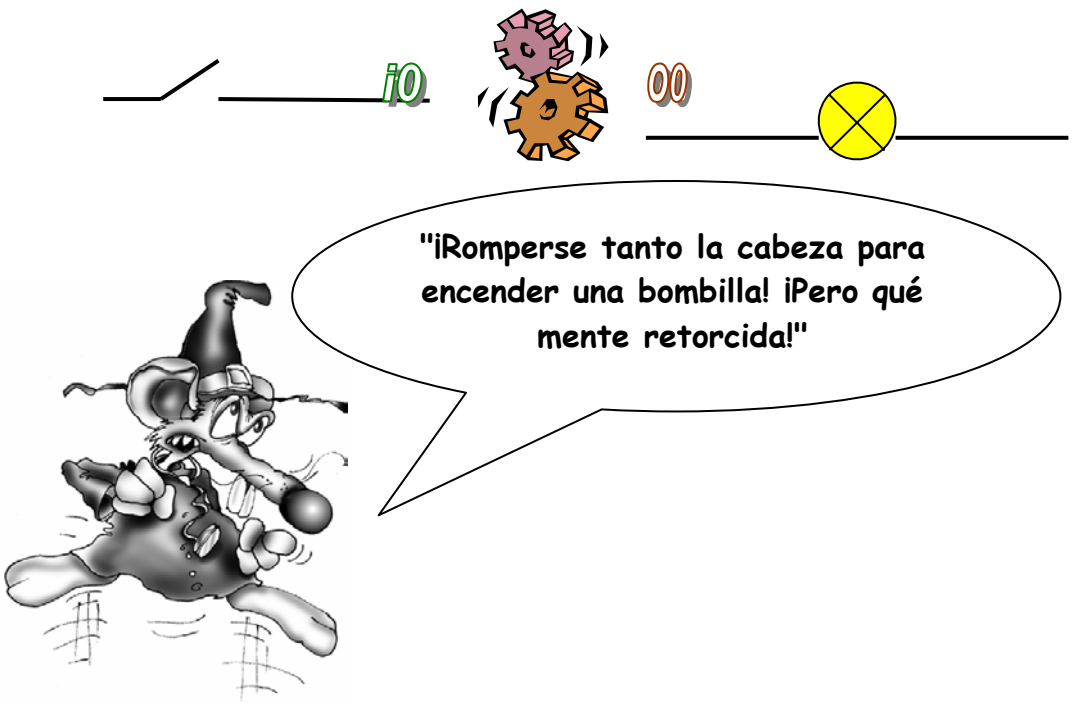


un botón pulsador

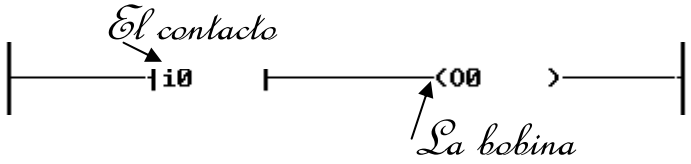
y un interruptor?

Primer ejemplo: « quién fue el primero, el interruptor o la bombilla … »

Un simple interruptor y una simple bombilla: el interruptor está cableado en la entrada i0, la bombilla en la salida o0. Si el interruptor se cierra, la bombilla se enciende; si el interruptor se abre, la bombilla se apaga. Más simple, imposible.

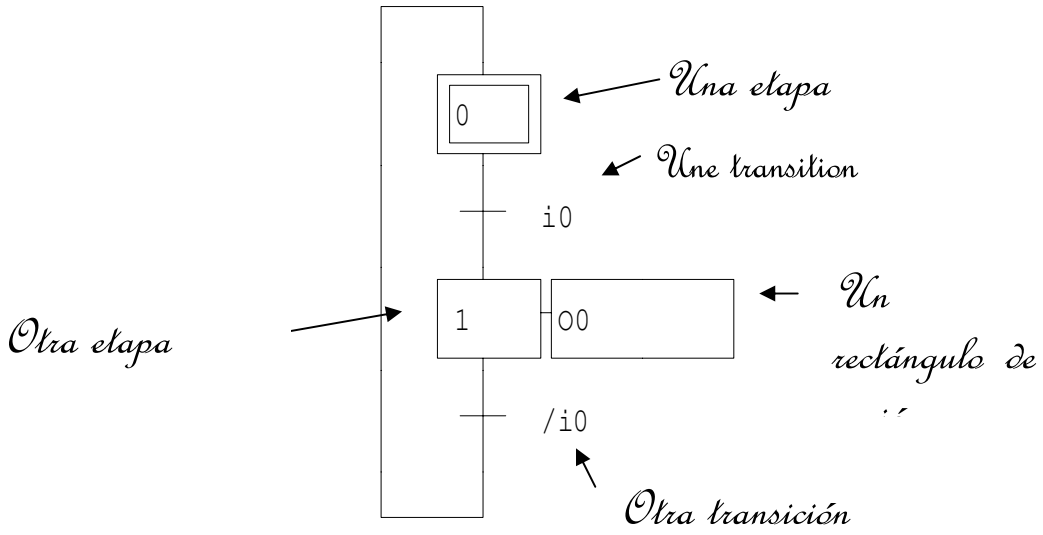


Solución 1: el lenguaje natural del electricista: ladder



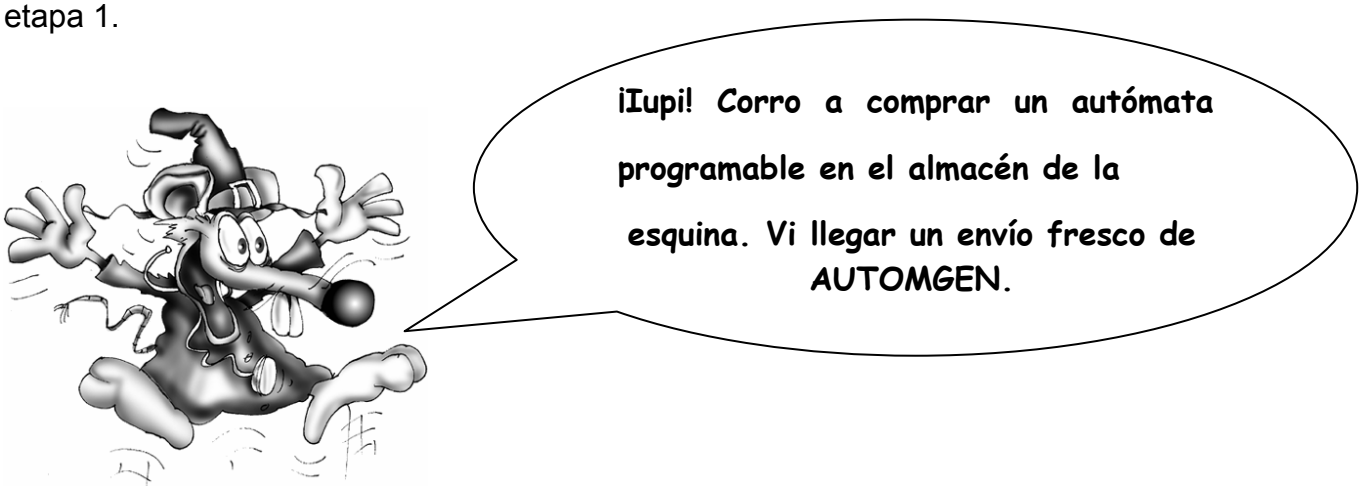
Ladder es la transcripción más directa del esquema eléctrico. El contacto recibe el nombre de la entrada donde está cableado el interruptor, la bobina el nombre de la salida donde está cableada la bombilla.

Solución 2: el lenguaje secuencial del automatista: Grafcet

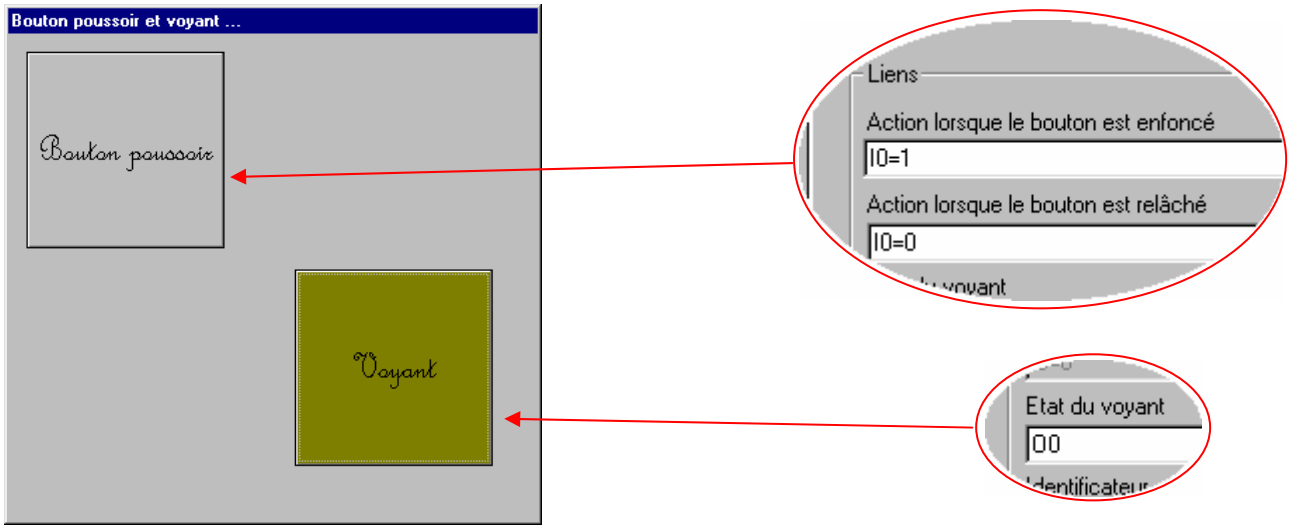


Grafcet se basa en la noción de estado. Para nuestro problema podemos decir que hay dos estados: el estado encendido y el estado apagado. Cada etapa representa un estado: aquí la etapa 0 representa el estado apagado y la etapa 1 el estado encendido. Queda por determinar la condición que hace evolucionar del

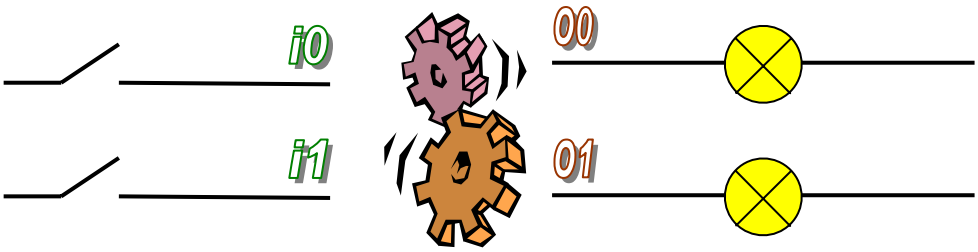
estado apagado al estado encendido: interruptor cerrado (indicado i0), y luego la condición que hace pasar del estado encendido al estado apagado: interruptor abierto (indicado / i0). Las condiciones están escritas a la derecha del elemento indicado como transición. El rectángulo asociado a la etapa 1 llamado rectángulo de acción contiene el nombre de la salida O0 (salida donde está cableada nuestra bombilla). Así, en todo momento, el estado de la bombilla es el mismo que el de la etapa 1.



Los afortunados posesores de IRIS pueden utilizar dos objetos BPVOYANT para simular este primer ejemplo.

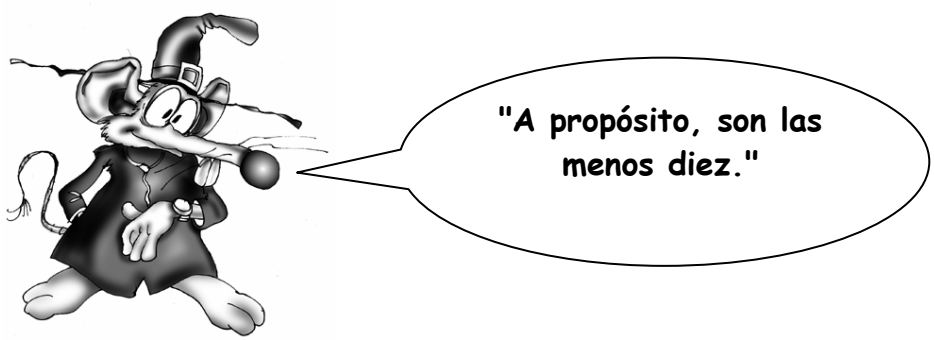


A jugar …



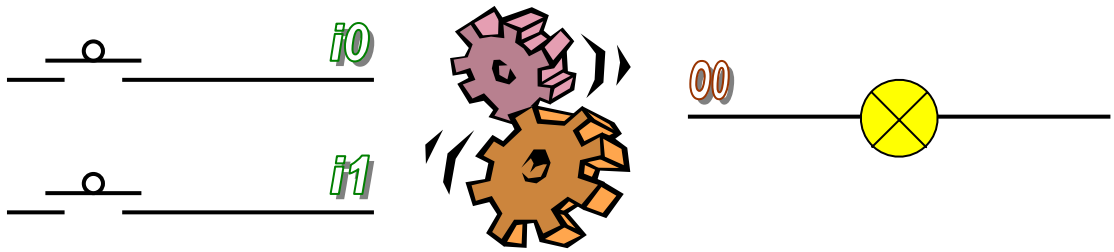
El interruptor 1 enciende la bombilla 1, el interruptor 2 la bombilla 2. Al final de este documento se propone una solución Grafcet.

Segundo ejemplo: « temporizaciones, minuteros y otras diversiones temporales… »

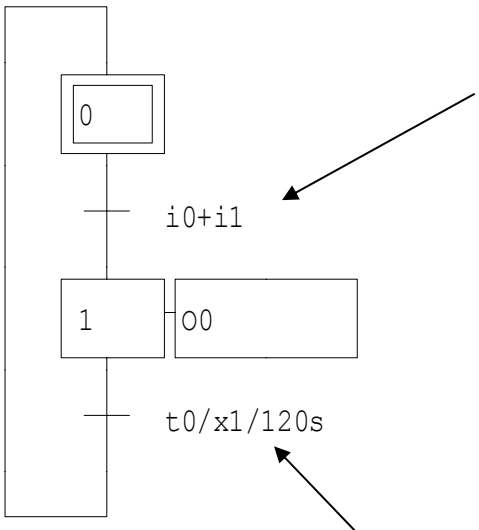


Como ustedes se imaginan, la noción de temporización se utiliza cuando un programa debe, de una manera u otra, efectuar acciones teniendo en cuenta un dato relativo al tiempo. Esperar cierto tiempo antes de llevar a cabo una acción, o bien llevar a cabo una acción durante cierto tiempo, por ejemplo.

Nuestro segundo ejemplo es el siguiente: un pasillo está equipado con una bombilla y dos botones pulsadores. Al presionar uno de los dos botones, la bombilla se enciende 2 minutos (según el Dr. R. ese plazo es más que suficiente para cruzar el pasillo).



Solución 1: la simplicidad

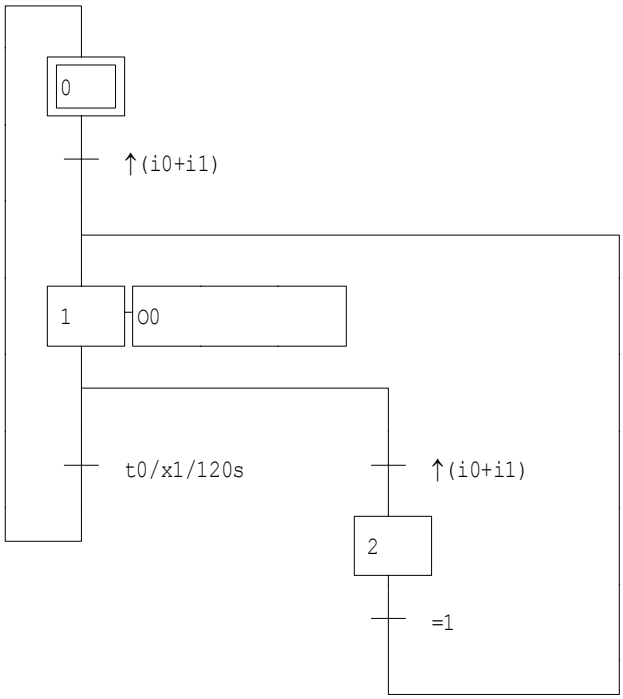


Encender la bombilla si se presiona el botón pulsador 1 o si se presiona el botón pulsador 2. "O" se escribe "+" en una transición.

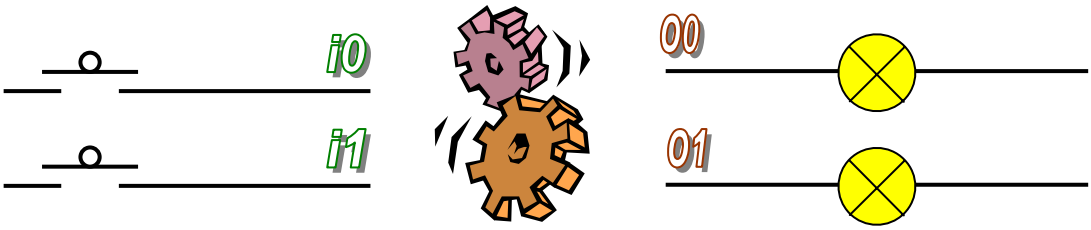
Esperar 2 minutos (120 segundos) utilizando la temporización 0, es la etapa 1 la que lanza la temporización.

Solución 2: mejora

La desventaja de esta solución es que si el botón pulsador se presiona mientras la bombilla está encendida, la temporización no se restablece. La semana pasada, el Dr. R. creyó haber restablecido el minutero y de buenas a primeras se encontró en la oscuridad.



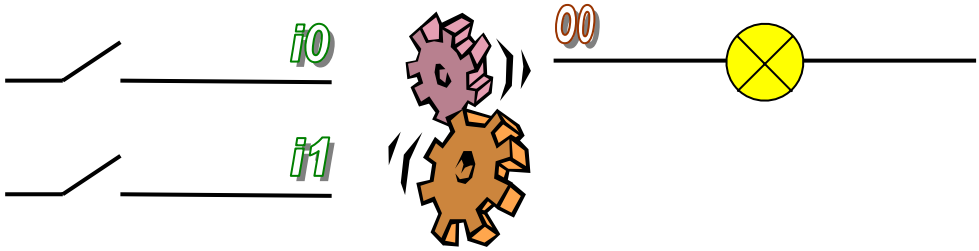
Y si nos lanzamos a escribir un programa de verdad ...



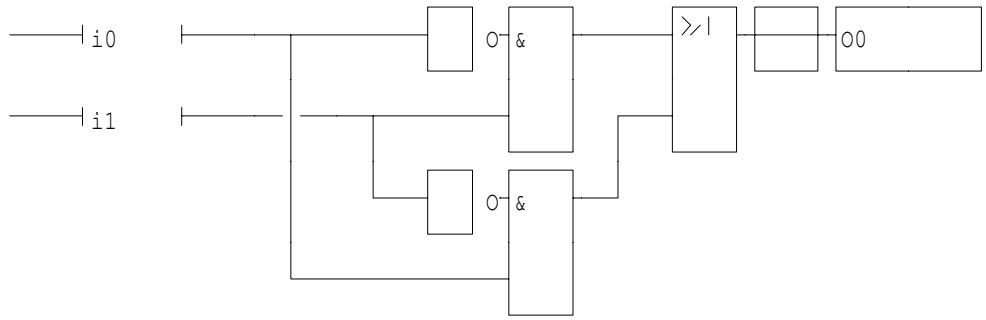
Una gestión inteligente de la iluminación del pasillo: se ha colocado una bombilla en cada extremo. Cuando se presiona un interruptor, las dos bombillas se encienden; la bombilla del lado del interruptor presionado se apaga al cabo de 30 segundos, y la otra al cabo de un minuto.

Tercer ejemplo: « variación sobre el tema del conmutador… »

Recordemos el genial principio del conmutador: dos interruptores permiten encender o apagar la misma bombilla.

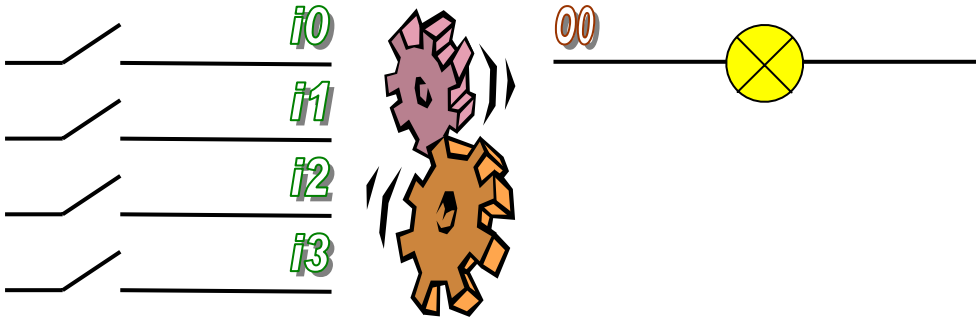


He aquí una solución en logigrama:

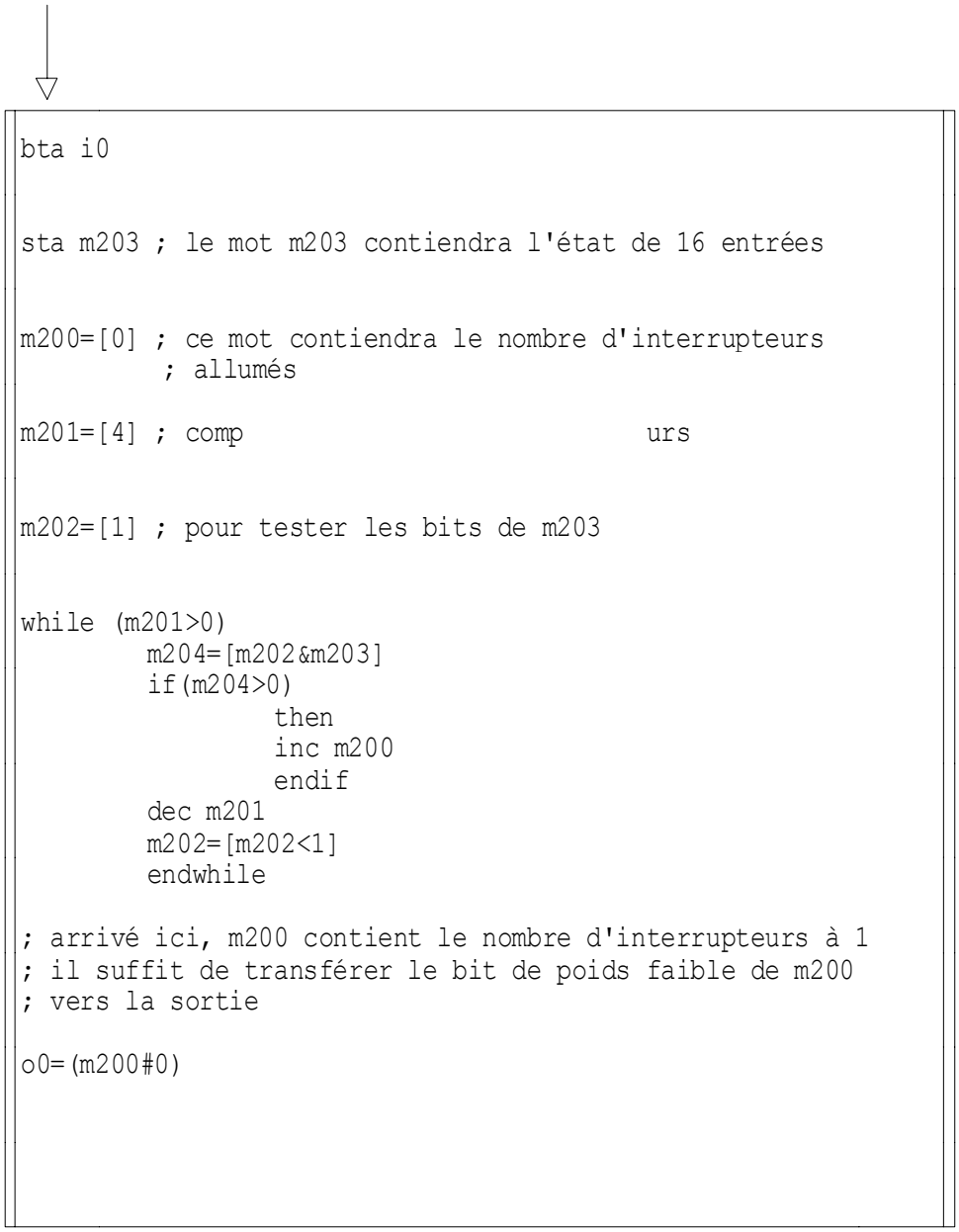


Los puristas habrán reconocido la ecuación booleana del O exclusivo.

Las cosas se ponen realmente interesantes si deseamos conservar las propiedades del conmutador con un número de interruptores mayor que 2.

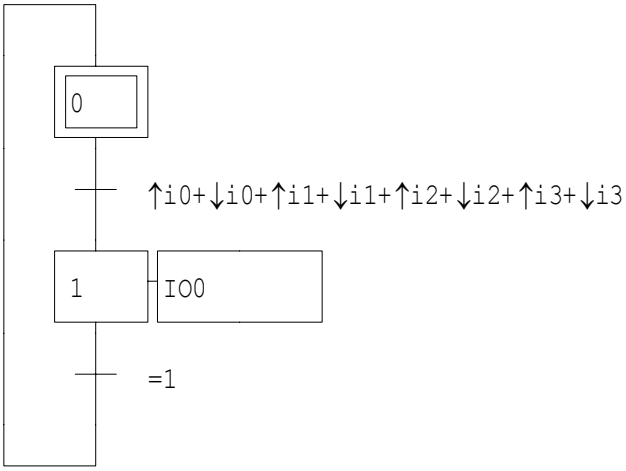


Una solución que utiliza el lenguaje literal de AUTOMGEN.



```
bta i0


sta m203 ; le mot m203 contiendra l'état de 16 entrées


m200=[0] ; ce mot contiendra le nombre d'interrupteurs
         ; allumés

m201=[4] ; comp                          urs


m202=[1] ; pour tester les bits de m203


while (m201>0)
        m204=[m202&m203]
        if(m204>0)
                then
                inc m200
                endif
        dec m201
        m202=[m202<1]
        endwhile

; arrivé ici, m200 contient le nombre d'interrupteurs à 1
; il suffit de transférer le bit de poids faible de m200
; vers la sortie

o0=(m200#0)
```

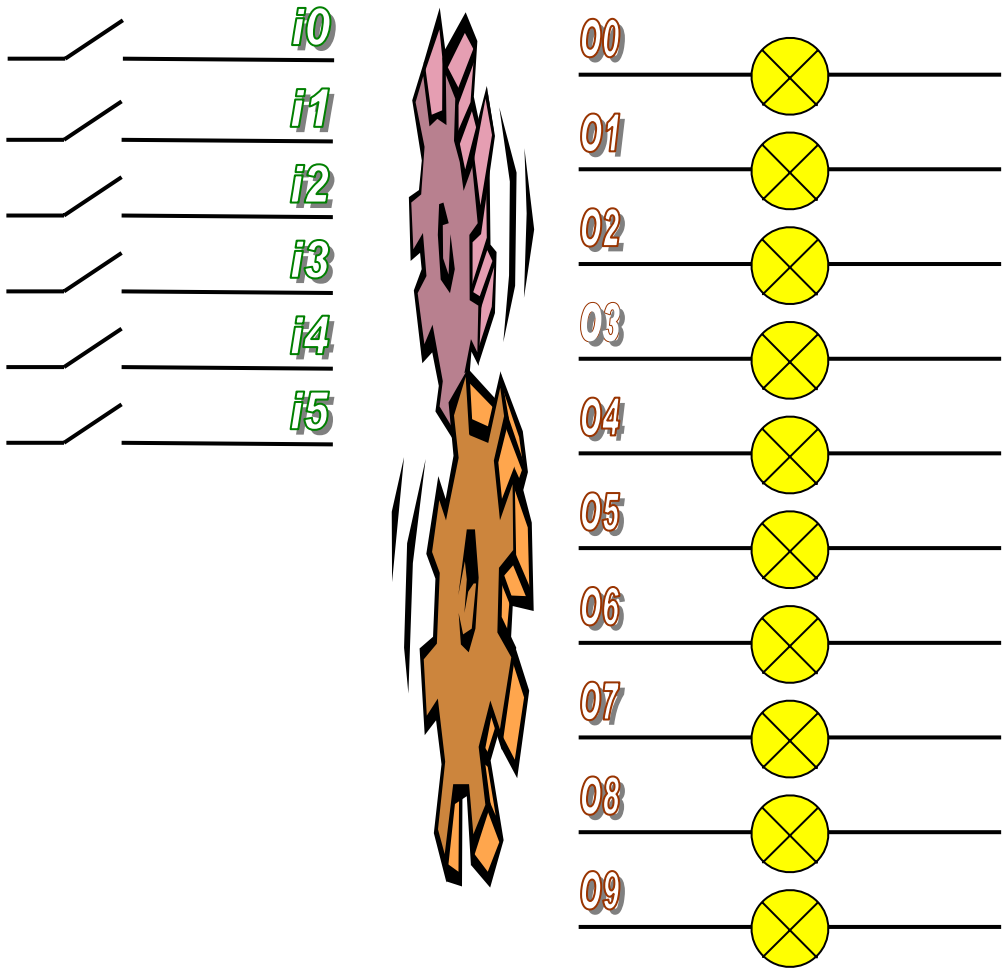
Otra más astuta:



« IO0 » significa « invertir el estado de la salida 0 ».

Prueben esto:

Un amplio cuarto con 6 interruptores y 10 bombillas. Cada interruptor permite iluminar más o menos el cuarto (pasando del estado en que todo está encendido al estado en que se enciende una bombilla, luego dos, etc...).

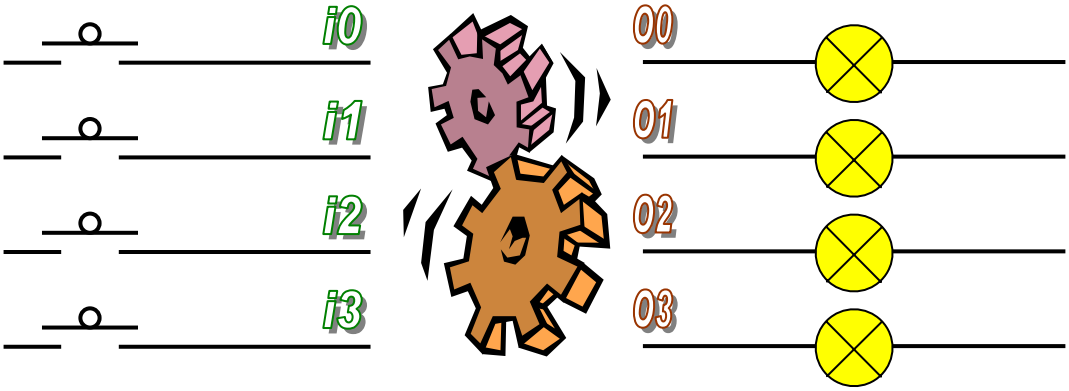


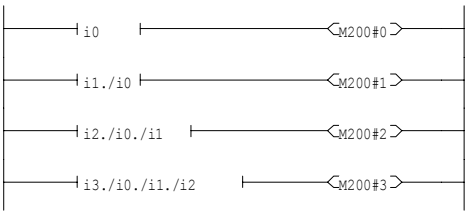
Cuarto ejemplo: « Y el botón pulsador se vuelve inteligente … »

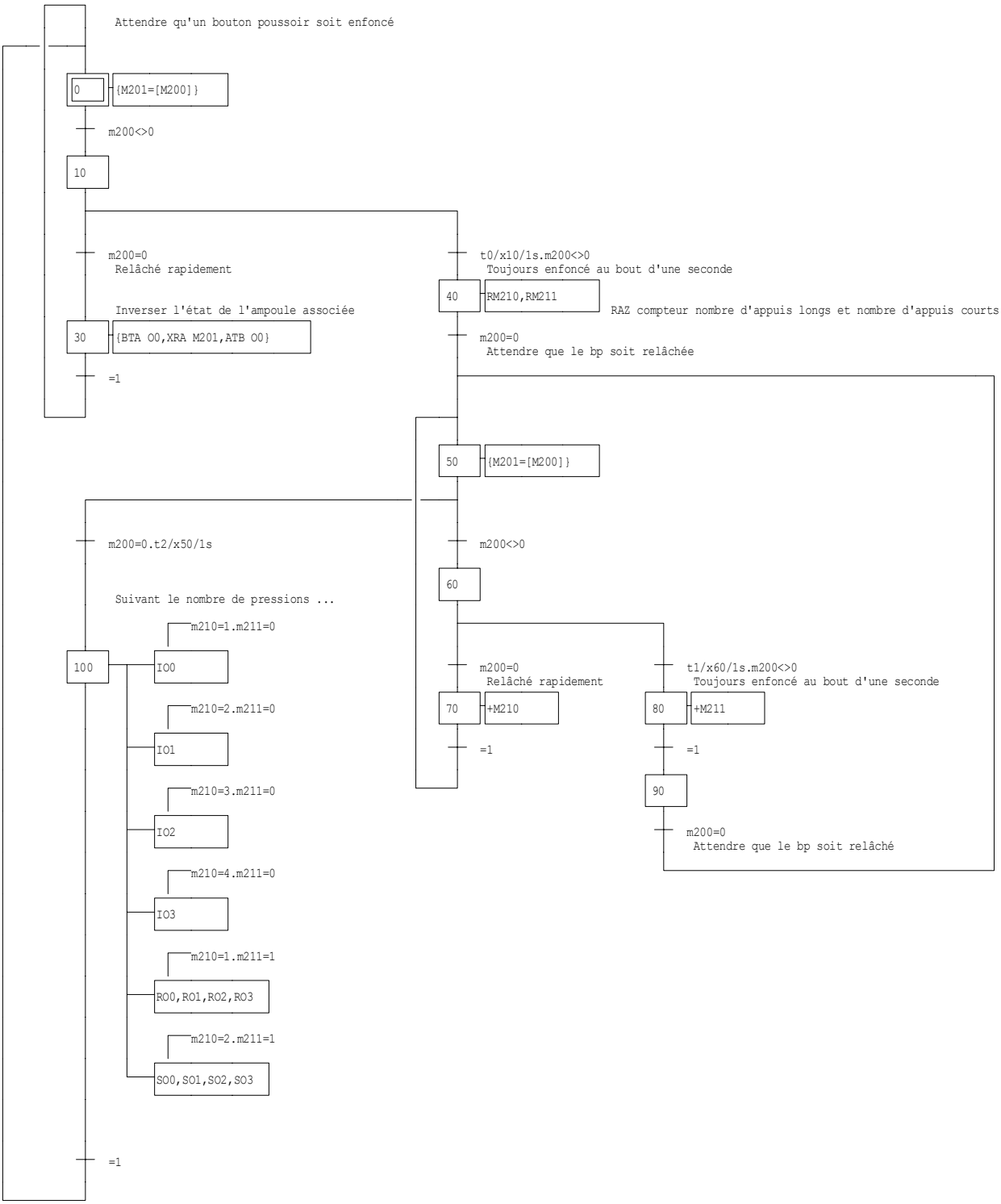
En todos los ejemplos anteriores, los botones pulsadores realizan una sola función. Es decir, la persona que los maneja tiene sólo dos opciones: presionar o no presionar para obtener una función (encender o apagar). Imaginemos un botón pulsador « más rendidor » capaz de recibir dos tipos de presión: una presión corta (arbitrariamente menos de 1 segundo) o una presión larga (arbitrariamente 1 segundo o más).

Para este ejemplo, cuatro botones pulsadores y cuatro bombillas. En forma predeterminada, con el uso normal, cada uno de los botones pulsadores está asociado a una bombilla. Una presión corta enciende o apaga la bombilla asociada. Cada botón pulsador debe permitir pilotear cada bombilla o la totalidad de las bombillas. La tabla siguiente resume el funcionamiento.

Acción sobre los botones pulsadores	**Resultado**
Una presión corta	La bombilla asociada cambia de estado
Una presión larga y una presión corta	La bombilla número 1 cambia de estado
Una presión larga y dos presiones cortas	La bombilla número 2 cambia de estado
Una presión larga y tres presiones cortas	La bombilla número 3 cambia de estado
Una presión larga y cuatro presiones cortas	La bombilla número 4 cambia de estado
Dos presiones largas y una presión corta	Todas las bombillas se apagan
Dos presiones largas y dos presiones cortas	Todas las bombillas se encienden




i0
M200#0
i1./i0
M200#1
i2./i0./i1
M200#2
i3./i0./i1./i2
M200#3


Attendre qu'un bouton poussoir soit enfoncé
0
{M201=[M200]}
m200<>0
10
m200=0
Relâché rapidement
Inverser l'état de l'ampoule associée
30
{BTA O0,XRA M201,ATB O0}
=1
t0/x10/1s.m200<>0
Toujours enfoncé au bout d'une seconde
40
RM210,RM211
RAZ compteur nombre d'appuis longs et nombre d'appuis courts
m200=0
Attendre que le bp soit relâchée
50
{M201=[M200]}
m200=0.t2/x50/1s
Suivant le nombre de pressions ...
m210=1.m211=0
100
IO0
m210=2.m211=0
IO1
m210=3.m211=0
IO2
m210=4.m211=0
IO3
m210=1.m211=1
RO0,RO1,RO2,RO3
m210=2.m211=1
SO0,SO1,SO2,SO3
=1
m200<>0
60
m200=0
Relâché rapidement
70
+M210
=1
t1/x60/1s.m200<>0
Toujours enfoncé au bout d'une seconde
80
+M211
=1
90
m200=0
Attendre que le bp soit relâché

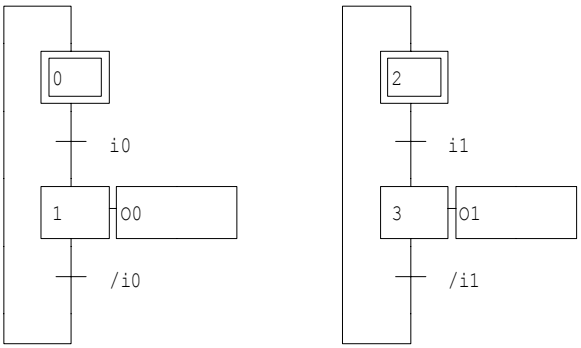
Con esto concluye este manual pedagógico. Esperamos les haya permitido descubrir las posibilidades de AUTOMGEN.

Proponemos un último ejercicio.

Automatizar el apartamento de la tía Hortensia respetando su gusto desmedido por los interruptores niquelados.

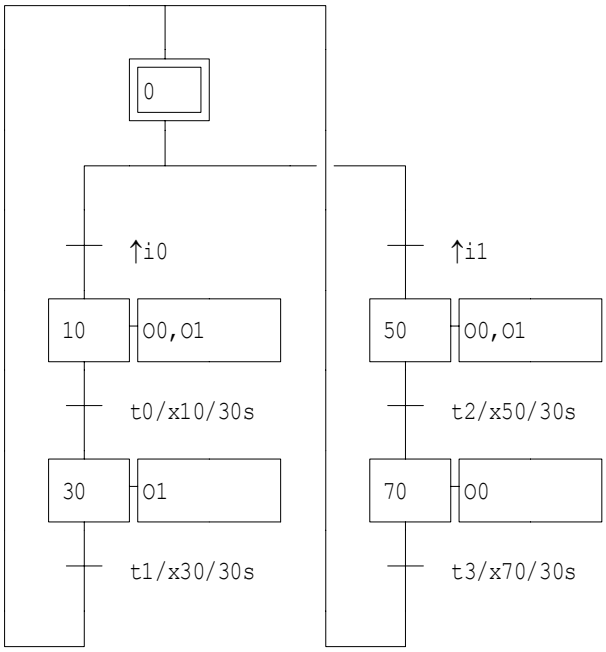
Las soluciones …

« quién fue el primero, el interruptor o la bombilla … »

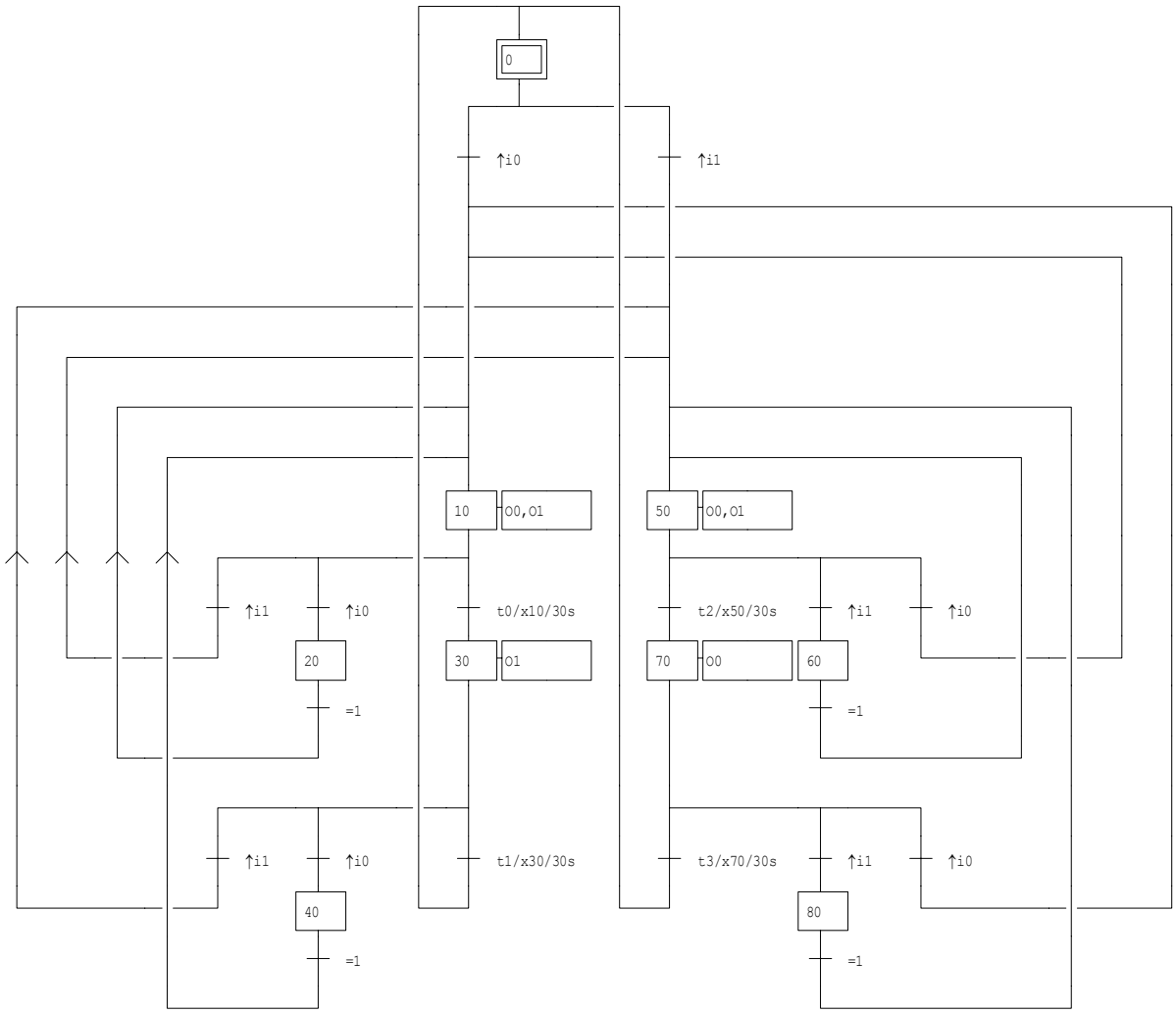


Es suficiente escribir dos Grafcets idénticos. Cada uno se ocupa independientemente de un interruptor y de una bombilla.

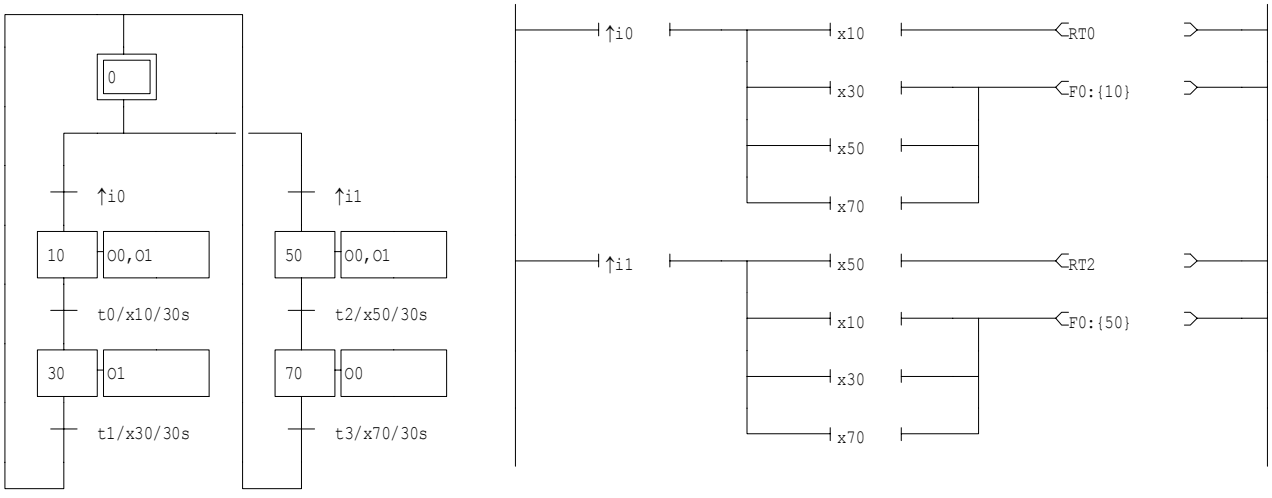
« temporizaciones, minuteros y otras diversiones temporales… »



Una versión simple sin la gestión del restablecimiento del minutero.



El restablecimiento del minutero hace que el programa sea muy complejo.



Una tercera solución con Grafcet, lenguaje ladder y forzados de Grafcet. El programa permanece legible.

« variación sobre el tema del conmutador …»