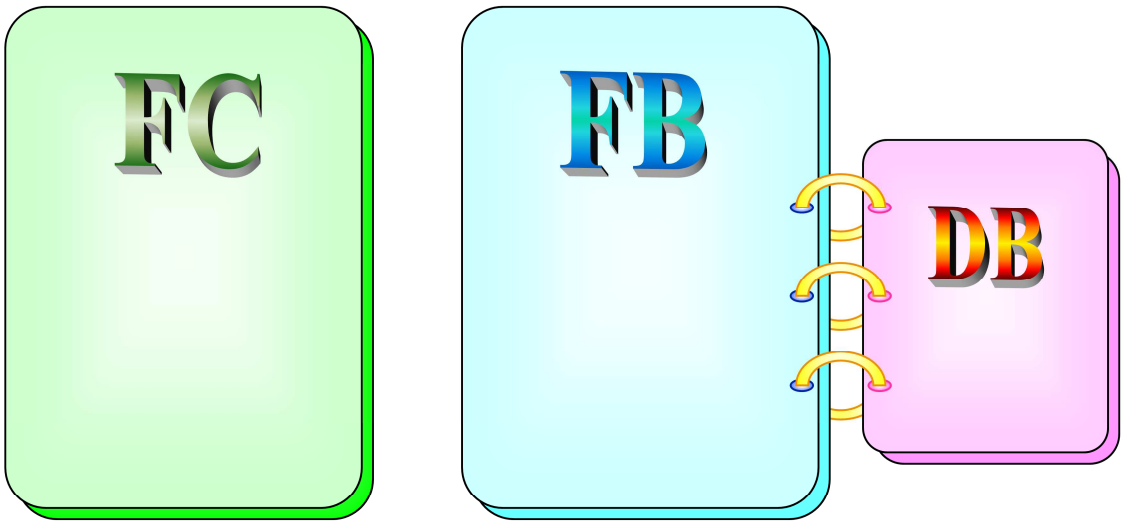
# Funções e Blocos de Funções



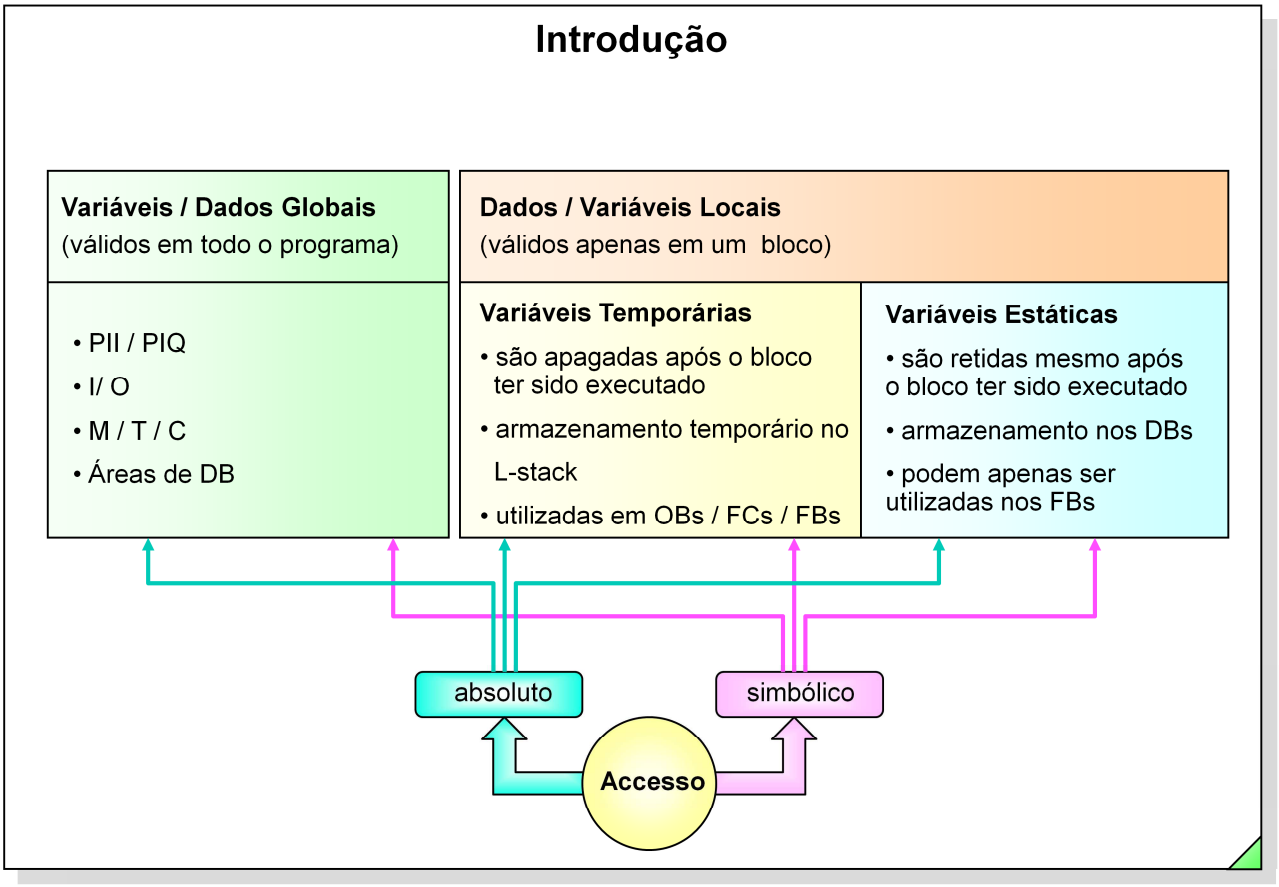

SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.1

sitrain

**Conteúdo**








**Geral**

Até agora, as entradas e saídas utilizadas no sistema transportador foram endereçadas com seus parâmetros atuais. Não foram associados parâmetros para os blocos.

Esse procedimento pode ser escolhido, por exemplo, na criação de um programa que será usado apenas em uma máquina especial.

Para funções utilizadas freqüentemente em sistemas maiores podem ser criados blocos parametrizáveis de uso universal (FC, FB). Eles possuem parâmetros formais de entrada e saída, aos quais são atribuídos parâmetros atuais quando a chamada é feita.

A associação da funcionalidade do bloco ao hardware é feita na parametrização, quando da chamada do bloco; a lógica do bloco não é alterada.

**Variáveis Locais**

Até agora foram utilizadas variáveis globais (bit memories e blocos de dados) para armazenar os dados de produção, por exemplo. Nesse capítulo serão dadas mais informações sobre o o armazenamento de dados em variáveis locais.

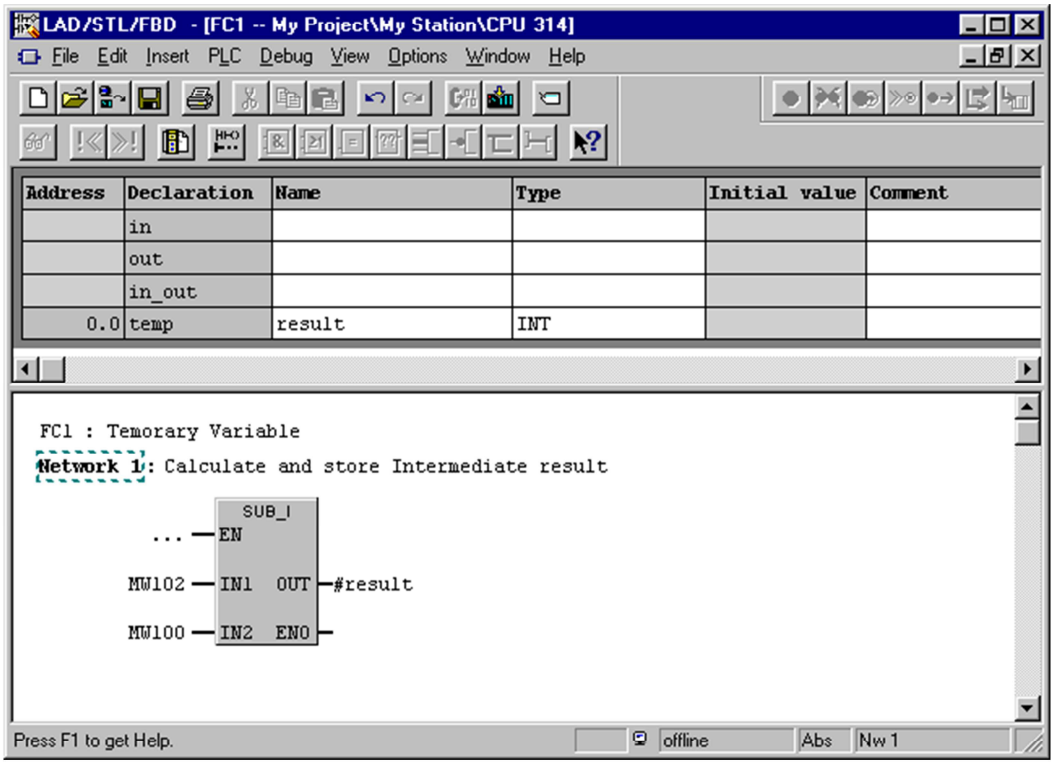
**Variáveis Temp**

Variáveis temporárias são variáveis armazenadas apenas durante a execução do bloco. Elas podem ser utilizadas em todos os blocos (OB, FC, FB).

**Variáveis Estáticas**

Se os dados tiverem de continuar armazenados mesmo após a execução do bloco deverão ser usadas as variáveis estáticas.
As variáveis estáticas só podem ser utilizadas em blocos de função.

# Variáveis Temporárias




SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.3

sitrain

**Geral**

Variáveis temporárias podem ser utilizadas em todos os blocos (OB, FC, FB). Elas são utilizadas para armazenar informações temporariamente enquanto o bloco está sendo executado. Os dados são perdidos quando é finalizada a execução do bloco.

Os dados são armazenados no L-stack (local data stack). O L-Stack é uma memória separada dentro da CPU.

**Declaração**

As variáveis são definidas na tabela de declaração do bloco. Na linha "temp" são preenchidos o nome e o tipo do dado da variável.

Não é possível predefinir um valor inicial neste caso.

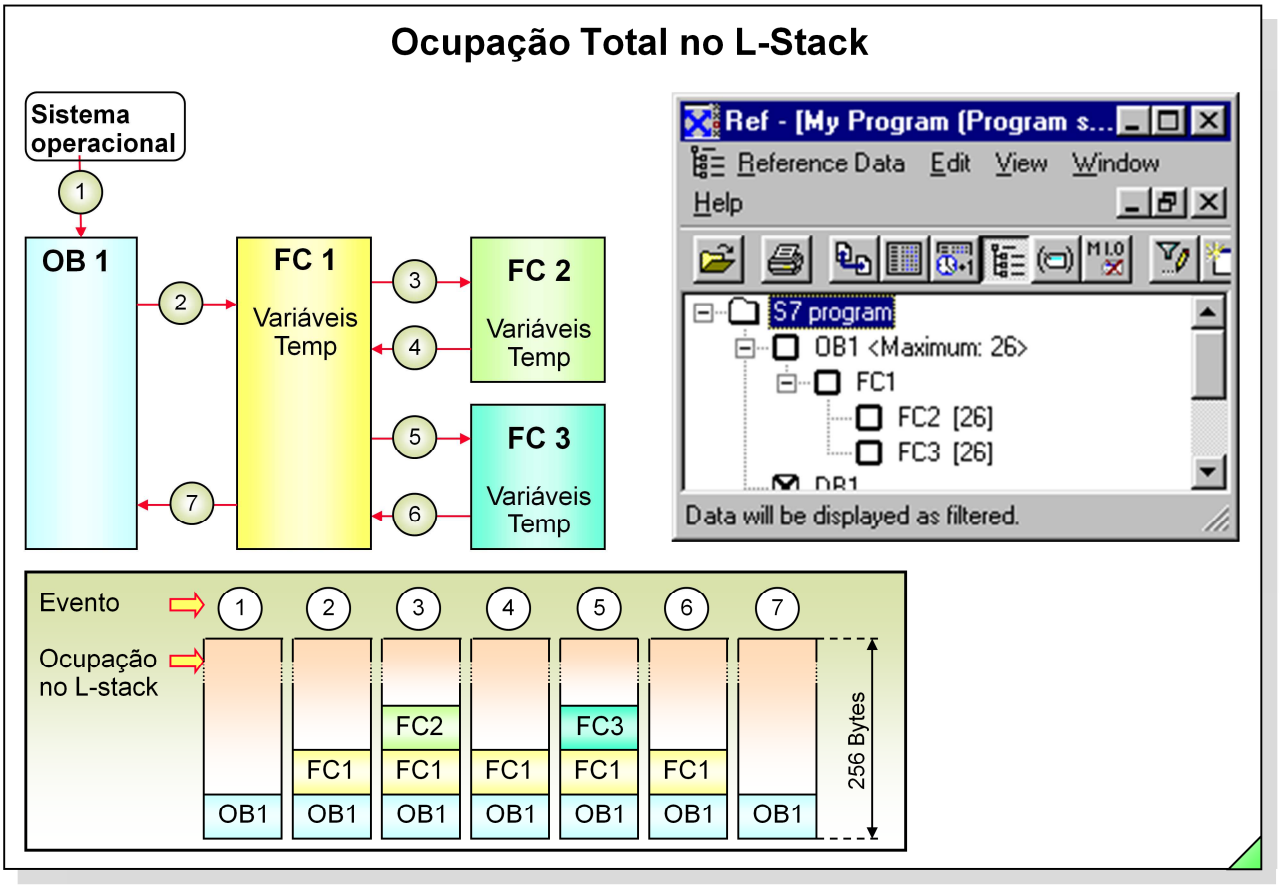
Após salvo o bloco, a locação de memória no L-Stack é exibida na coluna "Address".

**Acesso**

No Network 1 é exibido um exemplo do acesso simbólico a uma variável temporária. O resultado da subtração é armazenado na variável temporária "result".
É possível também fazer o acesso absoluto à variável (T LW0). Deve-se, contudo, evitar isso, já que o programa torna-se de difícil leitura.

**Nota**

Nomes de variáveis que começam com o caractere especial # são variáveis locais apenas válidas dentro do bloco onde foram declaradas na tabela de declarações. O Editor de Programa automaticamente cria o caractere especial.




SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.4

sitrain

**Ocupação Total no L-Stack**

É possível exibir o número de bytes que o programa todo necessita na pilha local de dados com a ferramenta "Reference Data". Essa ferramenta será melhor explorada no capítulo “Procura de Defeitos".

A ocupação total da pilha de dados local e o número de bytes necessário por chamada é exibido na tela.

**Ativando o Reference Data**

No SIMATIC Manager selecione a pasta Blocks e as opções de menu *Options -> Reference Data -> Display.*

**Nota**

Se o máximo número de dados locais for excedido durante a execução do programa, a CPU vai para o modo Stop. A mensagem “STOP causado por erro na alocação de dados locais” ("STOP caused by error when allocating local data“) é reportada como a causa do erro no buffer de diagnóstico.

## Tamanho do L-Stack

Tamanho total:
1.5 Kbyte
(CPU 313..316)

| Execução | | Para o S7-300: Classe de Prioridade | Para o S7-300: Tamanho do L-stack |
|---|---|---|---|
| Startup (execução única) | | 27 | 256 bytes |
| Execução cíclica | | 1 | |
| Execução controlada por tempo | Time-of-Day Interrupt | 2 | 256 bytes |
| | Time-Delay Interrupt | 3 | 256 bytes |
| | Cyclic Interrupt | 12 | 256 bytes |
| Execução por evento | Hardware Interrupt | 16 | 256 bytes |
| | Tratamento de Erros no startup | 28 | 256 bytes |
| | Tratamento de Erros no ciclo de scan | 26 | |





**Local Data Stack**

A pilha de dados locais (local data stack ou L stack) é uma área de memória que contém as variáveis temporárias (em substituição às memórias de rascunho da linha SIMATIC S5) dos blocos.

**Tamanho do L Stack**

Quando o sistema operacional chama um OB, uma área de L stack de 256 bytes é aberta enquanto o OB e os blocos chamados estão em execução.
Para cada classe de prioridade são reservados 256 bytes.
O L stack das CPUs 313..316 tem um total de 1536 bytes (1.5kByte).

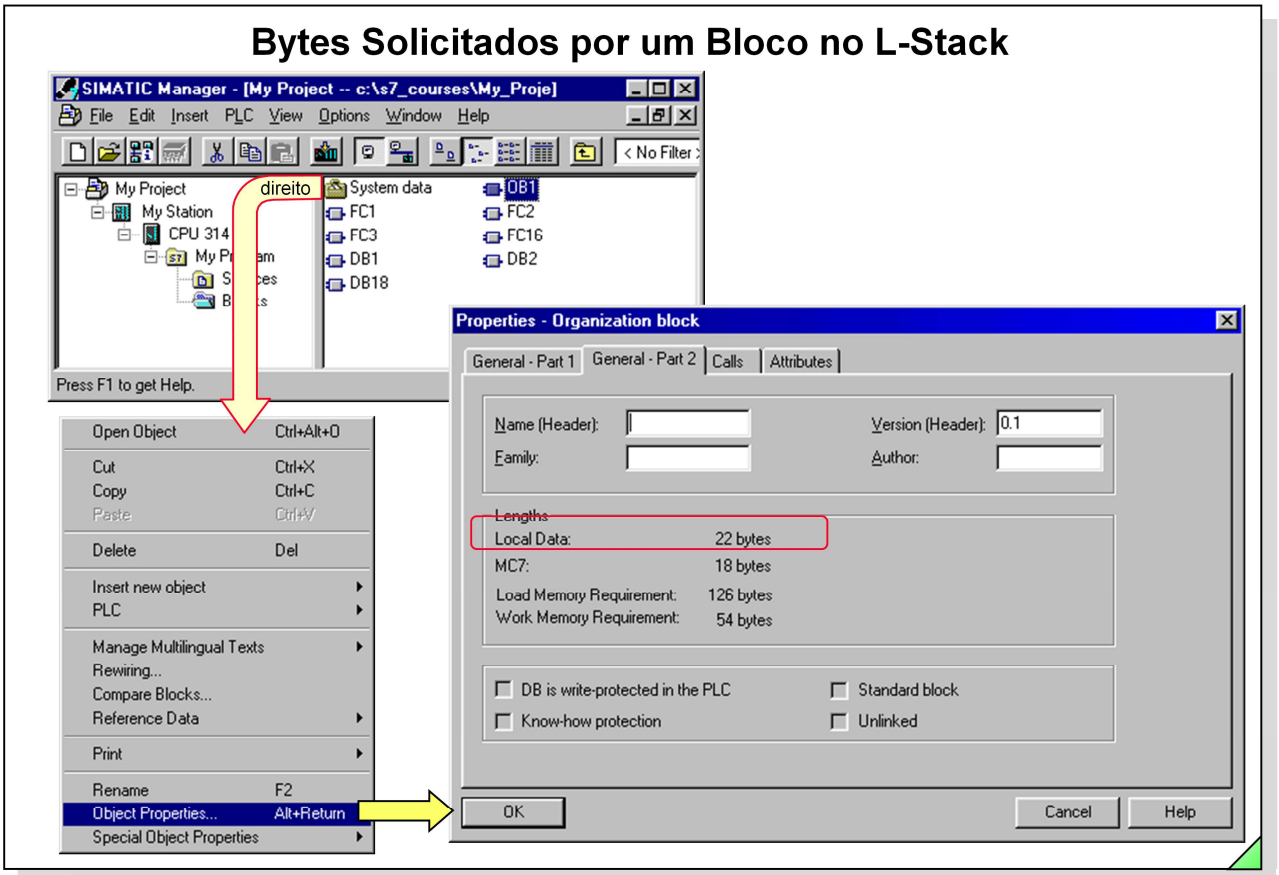
**Classes de Prioridade**

Há um total de oito classes de prioridade no S7-300. Contudo, não mais do que 6 classes de prioridade podem estar ativas ao mesmo tempo. Se, por exemplo, o OB 100 estiver ativo (com classe de prioridade 27), então o OB 1 (classe de prioridade 1) nunca pode estar ativo. Além disso, os OBs de 80 a 87 para erros assíncronos podem apenas ter classe de prioridade 28, se a falha ocorrer no programa de startup. Em outras palavras, para que interrompam o OB 100. Maiores informações serão vistas no capítulo "Blocos de Organização".

**S7-400**

Com as CPUs do S7-400 é possível escolher o tamanho da pilha de dados locais para as classes de prioridade individuais (Ferramenta: HW Config.).
É possível desativar as classes de prioridade que não são necessárias. Dessa forma, pode-se disponibilizar maior área de dados para as outras classes de prioridade.

# Bytes Solicitados por um Bloco no L-Stack







Data: 26/08/2011
Arquivo: S7-Bas-09.6

sitrain

**Exibindo o tamanho em Bytes**

É possível visualizar o número exato de bytes que um bloco necessita na pilha local de dados acessando as propriedades do bloco.

**Ativando**

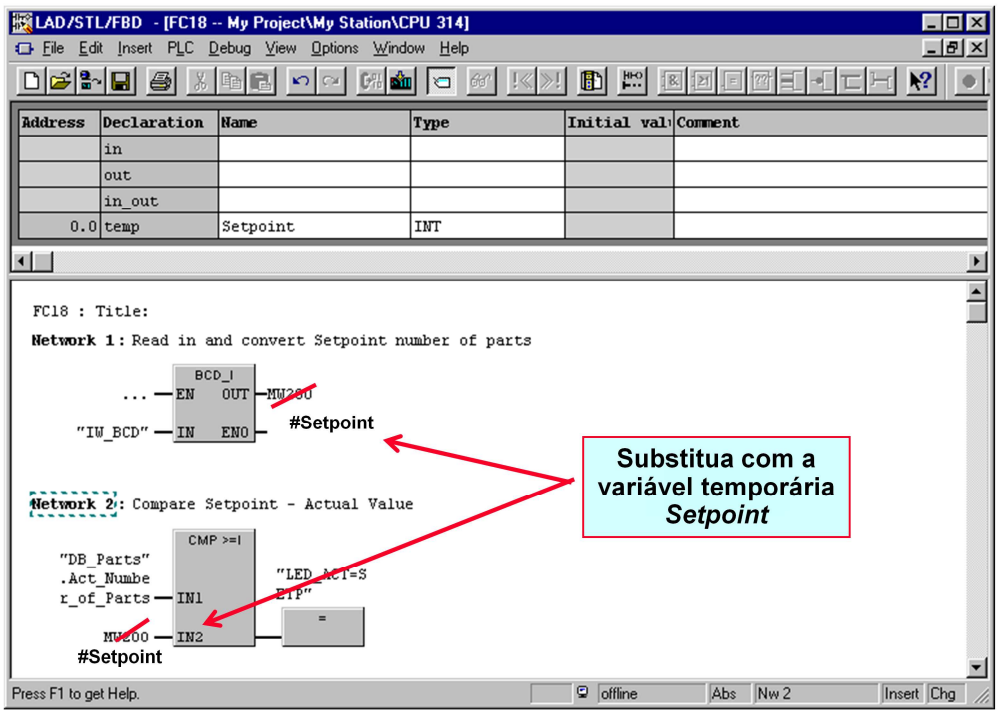
1. No SIMATIC Manager, selecione o bloco com o botão direito do mouse e depois -> *Object Properties.* ou
2. No SIMATIC Manager, selecione o bloco com o botão esquerdo do mouse e depois as opções de menu *Edit -> Object Properties.*

**Notas**

A soma de dados locais para um nível de execução (OB) é no máximo 256 bytes com o S7-300. Cada OB toma para si próprio sempre cerca de 20 ou 22 bytes. Isso significa que um máximo de 234 bytes pode ser utilizado por um FC ou FB.

Se mais do que 256 bytes de dados locais são definidos em um bloco, o bloco não pode ser transferido para a CPU. A transferência é interrompida com a mensagem de erro "The block could not be copied“ (O bloco não pôde ser copiado). Dentro desta mensagem de erro existe um botão de detalhes (“Details”). Clicando nele, uma caixa de mensagem aparece com uma explicação "Incorrect local data length“ (comprimento de dados locais incorreto).

# Exercício: Uso de Variáveis Temporárias








**Situação Atual: no FC 18**

- As peças transportadas no modo AUTO são contadas (por adição na variável "DB_Parts".ACTUAL_Number_of_Parts), assim que elas atingirem a posição de Controle Final ou atravessarem a barreira luminosa.
- O número de peças transportadas (*número de peças ATUAL*) é exibido no display digital BCD.
- É possível escolher o *SETPOINT número de peças,* de quantas peças devem ser transportadas, usando a chave BCD de pré-seleção. Quando o dado setpoint é atingido, ele é exibido no LED na posição de Controle Final (LED Q 20.4 / Q 8.4).
- O número de peças transportadas (variável "DB_Parts".ACTUAL_Number_of_Parts) é zerado através da chave não retentiva na posição de Controle Final quando o sistema é desligado (Q 8.1 / 4.1 = 0) ou quando a mensagem de setpoint atingido (LED) é reconhecida.
- Tão logo essa mensagem apareça (LED), nenhuma outra função de transporte pode ser iniciada (intertravamento no FC 16).

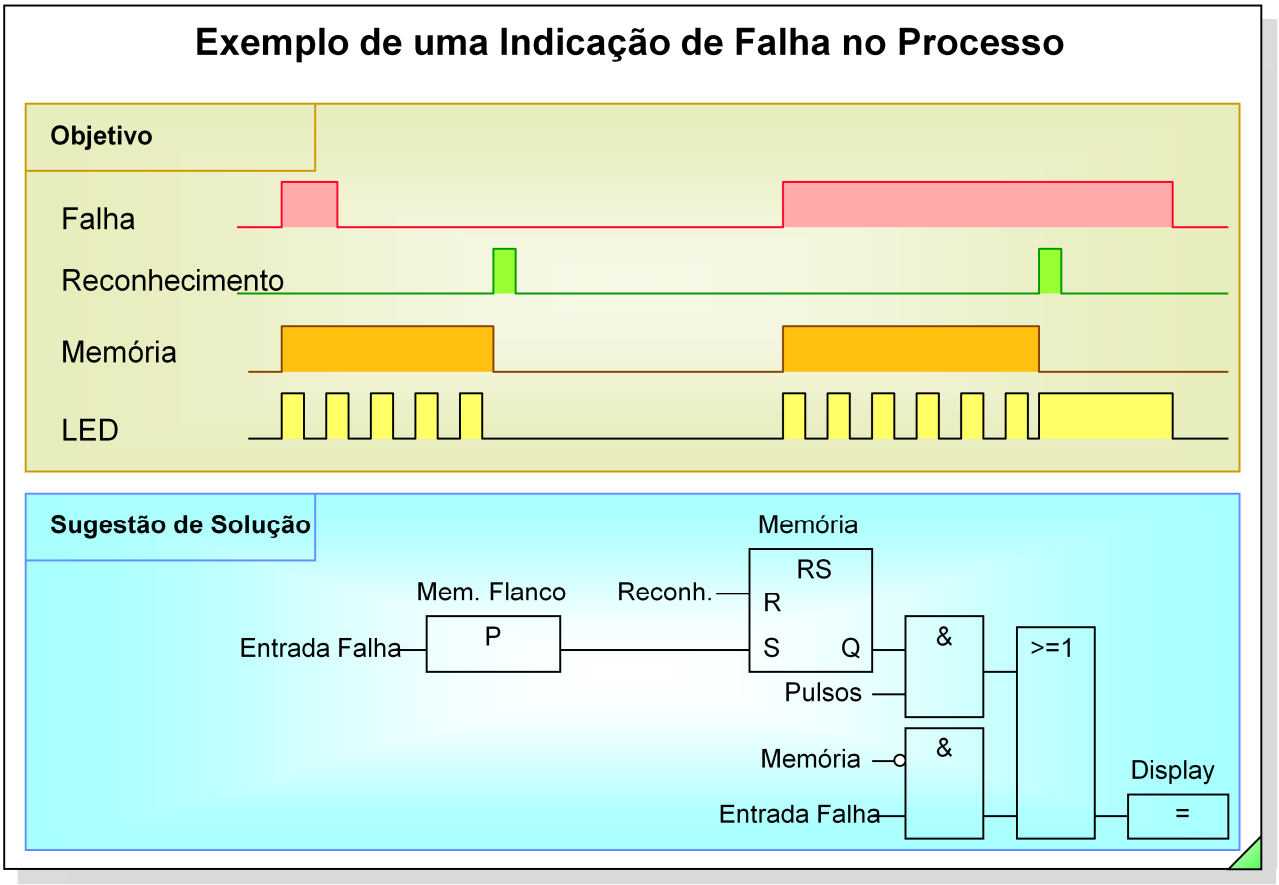
**Objetivo:**

A funcionalidade programada no FC 18 deve permanecer inalterada. Contudo, use a variável local, temporária *Setpoint* para o armazenamento intermediário do *SETPOINT número de peças* convertido de BCD para INT.

**O Que Fazer**:

- No FC 18, declare a variável temporária *Setpoint* como do tipo INT.
- Substitua o memory marker auxiliar utilizado para o armazenamento intermediário com a nova variável *Setpoint*.

# Exemplo de uma Indicação de Falha no Processo




SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.8

sitrain

**Descrição**

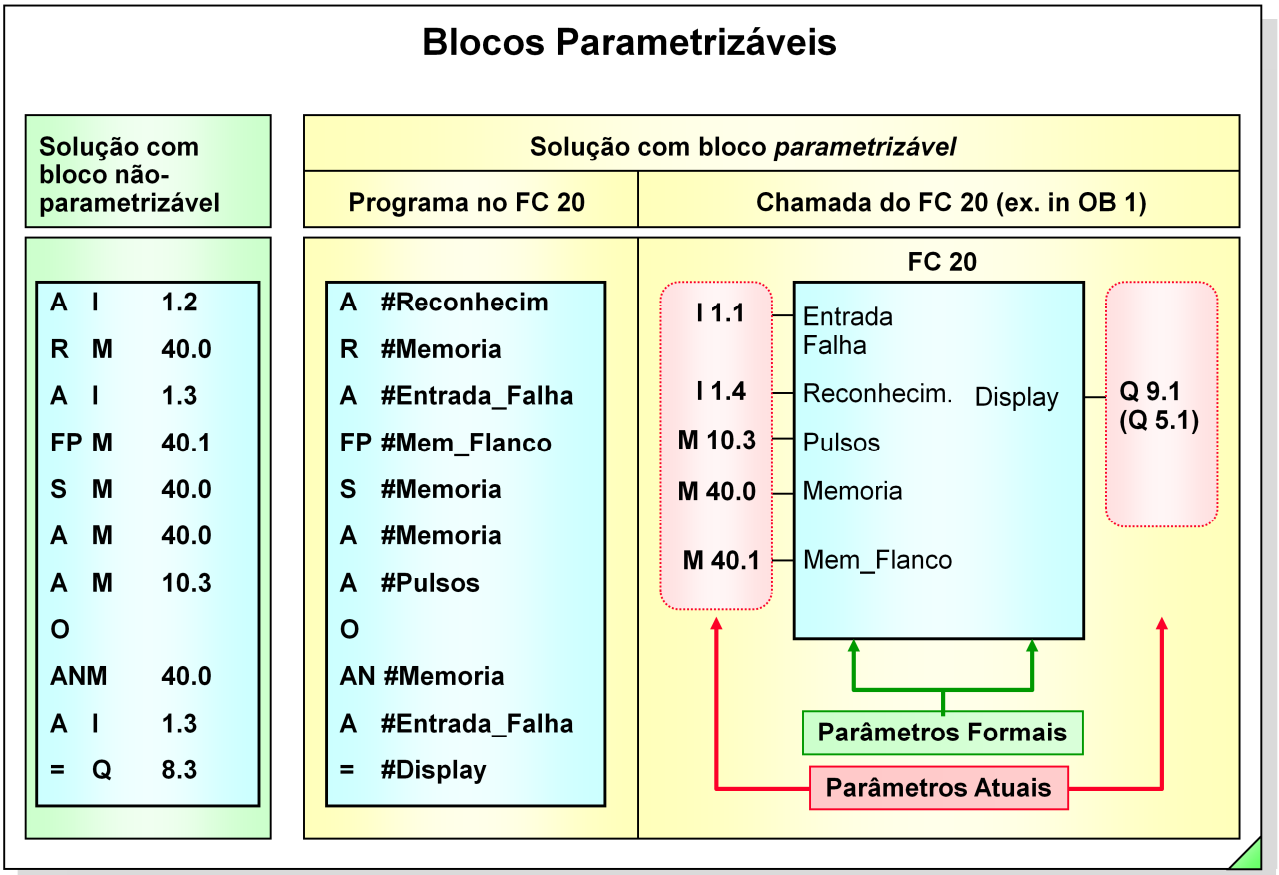
Problemas (falhas) que ocorrem devem ser exibidas por um LED no púlpito do operador. Quando o problema (I 1.3) ocorre, o LED (Q 8.3 ou Q 4.3) deve piscar com 2Hz. O problema é reconhecido na entrada I 1.2. Se o problema for corrigido, o LED pára de piscar. Se o problema continuar, o LED muda para o estado aceso permanente até que o problema esteja corrigido.

**Programa**

Para que mesmo os problemas existentes por curtos períodos de tempo não sejam perdidos é utilizado um flip flop com set dominante (M40.0).
Uma detecção de flanco do RLO do sinal da mensagem é utilizado, para que a memória possa sofrer reset quando a falha é reconhecida.

Se a memória for setada (a falha ainda não foi reconhecida), a lógica AND superior faz com que o LED pisque. Com isso, o bit memory M10.3, definido como clock memory durante a parametrização da CPU, é avaliado.

A operação lógica AND inferior é utilizada para provocar o estado aceso permanente caso a falha seja reconhecida e o problema persistir.






Data: 26/08/2011
Arquivo: S7-Bas-09.9


**Aplicação**

Podem ser programados blocos parametrizáveis para funções freqüentemente acessadas. Isso traz os seguintes benefícios:

- o programa tem apenas que ser criado uma vez, o que reduz significativamente o tempo de programação.
- o bloco é armazenado na memória apenas uma vez, o que reduz significativamente a quantidade total requerida.
- o bloco ou a funcionalidade implementada através do bloco pode ser chamado quantas vezes for necessário, com endereços diferentes em cada uma das chamadas. Para isso, os parâmetros formais (parâmetros de entrada, saída, ou de entrada / saída) são preenchidos com parâmetros atuais cada vez que ele é chamado.

**Execução**

Quando o bloco é executado e aparece a instrução *"A #Entrada_Falha"*, o parâmetro é substituído pelo parâmetro atual transferido durante a chamada. Se a entrada I 1.4 for o parâmetro atual para o parâmetro *Entrada_Falha* na chamada do bloco, então a instrução "A I 1.4" é executada como a instrução "A #Entrada_Falha" .

**Blocos Parametrizáveis**

Pode-se programar um bloco FC ou FB parametrizável. Não é possível programar blocos de organização parametrizáveis, já que eles são chamados pelo sistema operacional, e além disso não é possível transferir endereços atuais durante a chamada.

**Exemplo**

Mesmo se a função de avaliação e exibição de falha for requerida duas vezes no sistema será necessário programar o FC 20 parametrizável apenas uma vez. O FC 20 pode ser usado então três vezes para três diferentes avaliações de falha, e são transferidos diferentes endereços atuais em cada chamada.

# Declarando os Parâmetros Formais no FC 20

**Parâmetros Formais**

| Tipo de parâmetro | Declaração | Uso | Graphic Display |
|---|---|---|---|
| Parâmetro de entrada | in | Apenas leitura | À esquerda do bloco |
| Parâmetro de saída | out | Apenas escrita | À direita do bloco |
| Parâmetro In/out | In_out | Leitura / escrita | À esquerda do bloco |

| Address | Declaration | Name | Type | Initial value | Comment |
|---|---|---|---|---|---|
| 0.0 | in | Disturbance_Input | BOOL | | |
| 0.1 | in | Acknowledge | BOOL | | |
| 0.2 | in | Flash_frequency | BOOL | | |
| 2.0 | out | Display | BOOL | | |
| 4.0 | in_out | Report_Memory | BOOL | | |
| 4.1 | in_out | Edge_Memory_bit | BOOL | | |
| | temp | | | | |





**Endereços Formais** Antes de criar o programa para o bloco parametrizável é necessário definir os parâmetros formais na tabela de declarações.

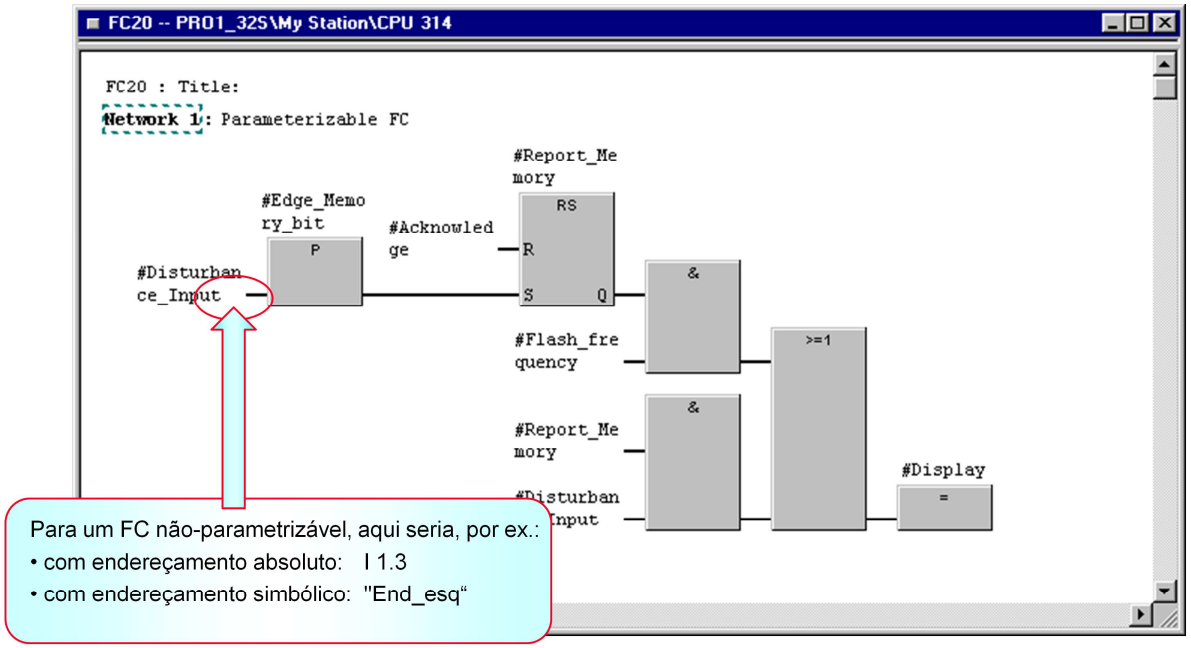
**Tipo de Parâmetro** Na tabela da figura podem ser observados três possíveis tipos de parâmetros e suas utilizações. É importante ter certeza de que os endereços formais que possuem acesso de leitura (utilizados em operações A, O, L) e acesso de escrita (associados com operações S, R, T) sejam declarados como parâmetros in/out.

**Exemplo FC20** Na parte inferior da figura pode ser observada a tabela de declarações do bloco FC 20 para a exibição da mensagem de falha (consulte a página anterior). Como os parâmetros formais #Memoria e #Mem_Flanco são acessados tanto para leitura como para escrita com a operação FP, deve-se declará-los como parâmetros in/out.

**Notas** Existe apenas uma linha para cada tipo de parâmetro na tabela de declarações. Após completada a declaração do parâmetro formal, pressionando-se Enter uma linha adicional automaticamente aparece, para esse tipo de parâmetro. Também é possível inserir uma linha adicional de declaração utilizando as opções de menu *Insert -> Declaration Row -> Before Selection /After Selection*.

**Atenção!** Os parâmetros formais declarados de um bloco são sua interface externa, isto é, eles são “visíveis” ou relevantes nos outros blocos que o chamam. Se a interface de um bloco for modificada posteriormente adicionando-se ou retirando-se parâmetros formais, isso resultará que as chamadas tenham de ser atualizadas ou corrigidas em todos os outros blocos nos quais a chamada deste bloco já foi programada.

# Editando um Bloco Parametrizável








**Notas**

Não importa se os nomes dos parâmetros formais são escritos com letras maiúsculas ou minúsculas. O caractere “#” na frente do nome é automaticamente inserido pelo PG. Ele serve para indicar que trata-se de uma variável local, definida na tabela de declarações do bloco.
É possível, ao editar o programa em LAD / FBD, que o nome não seja completamente exibido em uma linha. Isso depende da configuração do Editor de Programa (*Options -> Customize -> "LAD/FBD"* tab -> Width of address field).

**Símbolos**

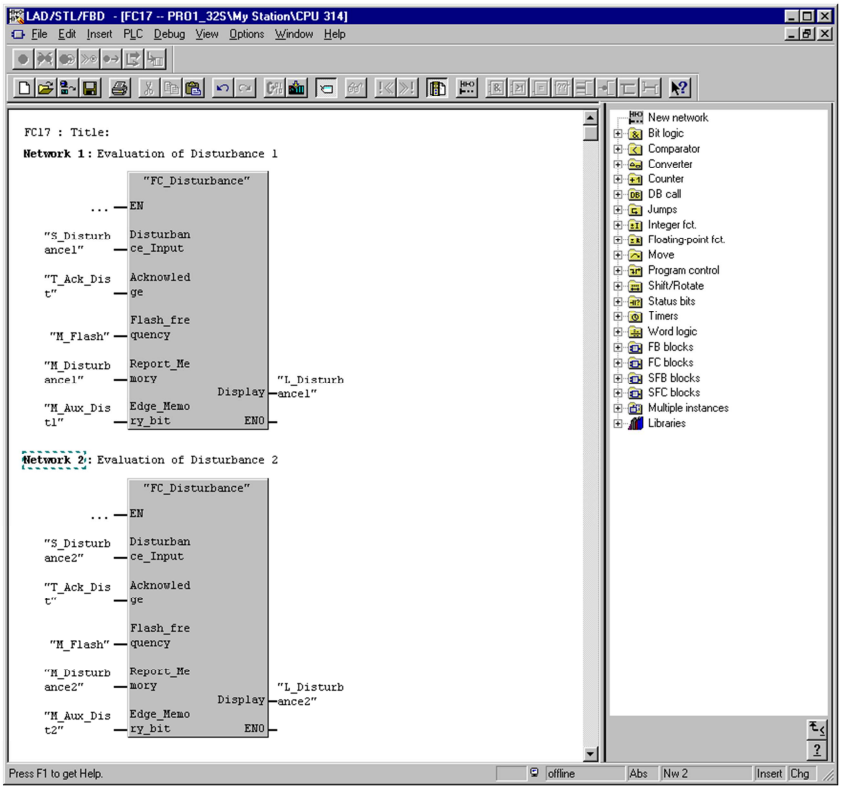
1. Se for utilizado um nome simbólico ao editar um bloco, o Editor busca na tabela de declarações.

   Se ele existir, o símbolo com o caractere # em frente é aceito no programa como uma variável local.
2. Se ele não puder ser encontrado como variável local, o Editor busca por um símbolo global na tabela de símbolos.

   Se ele for encontrado ele é colocado entre aspas e aceito no programa.
3. Se for especificado um mesmo nome simbólico global (na tabela de símbolos) que um local (na tabela de declaração de variáveis), o Editor irá inserir sempre a variável local.

   Se, contudo, for necessário trabalhar com o símbolo global, deve-se digitá-lo no programa entre aspas.

# Chamando um Bloco Parametrizável




SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.12

sitrain

**Programando uma Chamada de Bloco**

A chamada de um bloco parametrizável pode ser programada arrastando com o mouse o símbolo do bloco desejado para a seção de programa. O símbolo se encontra na pasta "FC Blocks" ou "FB Blocks" do Catálogo de Elementos de Programa do Editor LAD/FBD/STL. Os campos de parâmetros formais do bloco chamado aparecem com pontos de interrogação automaticamente, para que sejam preenchidos.

**Nota**

Basicamente, quando uma função (FC) parametrizável é chamada, um parâmetro atual deve ser associado a cada um dos parâmetros formais.

*Exceção:*

Nas linguagens de programação gráfica LAD e FBD, a parametrização dos parâmetros EN e ENO, os quais são automaticamente criados pelo Editor, é opcional.

**Parametrização**

Todos os endereços globais e locais cujos tipos de dados correspondem aos parâmetros do bloco chamado podem ser usados como parâmetros atuais.
Os parâmetros atuais podem ser transferidos com um endereço absoluto ou um nome simbólico – da mesma forma que na tabela de símbolos globais ou na tabela de declarações do bloco.

**Transferência de Parâmetros**

Basicamente, a transferência de parâmetros também é possível, isto é, parâmetros formais do bloco que faz a chamada são transferidos como parâmetros atuais para o bloco chamado. Para parâmetros do tipo de dado complexo isso é limitado. Isso é explorado com maiores detalhes no curso avançado.

# Exercício: Editando um Bloco FC Parametrizável

**1.** **Tabela de declaração do bloco FC 20**

| Address | Declaration | Name | Type | Initial value | Comment |
|---|---|---|---|---|---|
| 0.0 | in | Disturbance_Input | BOOL | | |
| 0.1 | in | Acknowledge | BOOL | | |
| 0.2 | in | Flash_frequency | BOOL | | |
| 2.0 | out | Display | BOOL | | |
| 4.0 | in_out | Report_Memory | BOOL | | |
| 4.1 | in_out | Edge_Memory_bit | BOOL | | |
| | temp | | | | |

**Bloco não-parametrizável**

```
A  I   1.1
R  M   40.0
A  I   1.4
FP M   40.1
S  M   40.0
A  M   40.0
A  M   10.3
O
AN M   40.0
A  I   1.3
=  Q   9.1(Q5.1)
```

**2.**

**Bloco FC 20 parametrizável**

```
A  #Reconhecim.
R  #Memoria
A  #Entrada...
:
:
:
:
:
:
:
:
```

SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.13

sitrain

**Função da Avaliação de Falha**

Problemas (falhas) que ocorrem (chave do simulador) são exibidas através de um LED no simulador com uma freqüência de 2 Hz. As falhas podem ser reconhecidas utilizando uma chave não retentiva (pushbutton) no simulador. Ao reconhecer uma falha ainda existente (a chave no simulador ainda estiver ligada), o LED muda para o estado aceso permanente. O estado aceso permanente desaparece assim que a falha não existe mais. Ao reconhecer uma falha que não existe mais (a chave no simulador está desligada), o LED apaga imediatamente.

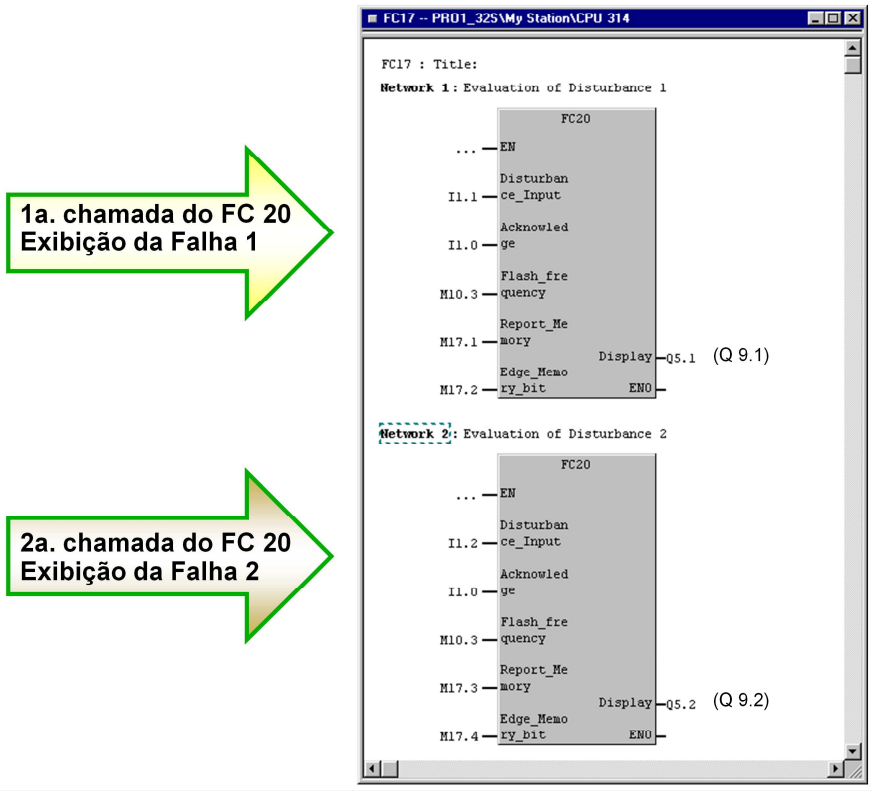
**Objetivo**

Escreva o programa para a análise de falha no bloco parametrizável FC 20. Na figura pode-se observar a tabela de declarações do FC 20, com os parâmetros formais e o início do programa no qual os parâmetros formais são utilizados.

**O Que Fazer**

- Insira o bloco FC 20.
- Declare os parâmetros formais como exibido na figura.
- Escreva o programa no FC 20 de acordo com o objetivo utilizando os parâmetros formais declarados.
- Salve o bloco e transfira-o para a CPU.

# Exercício: Chamando um Bloco FC Parametrizável




SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.14

sitrain

**Objetivo**

Duas falhas de processo (duas chaves no simulador) devem ser avaliadas e exibidas através dos LEDs no simulador. Sendo assim, programe duas chamadas do FC 20 e associe parâmetros a ele utilizando os parâmetros atuais exibidos na figura.

**O Que Fazer**

- Programe o FC 20 – chame-o em dois novos networks no bloco *Avaliação de Falha* FC 17;
- Salve o FC 17 modificado e transfira-o para a CPU.

**Nota**

O memory byte MB 10 foi parametrizado como clock memory nas propriedades da CPU através da ferramenta HW-Config. Se tiver sido feito um reset de memória por algum motivo o *system data* gerado pelo *HW Config* deve ser carregado novamente na CPU para que o bit memory M10.3 pulse.

# Blocos de Funções (FBs)

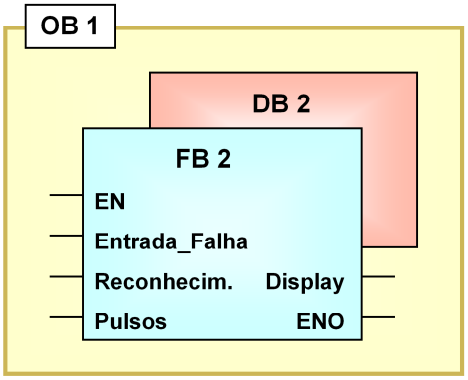



**Tabela de declaração do bloco de função**

| Address | Declaration | Name | Type | Initial value | Comment |
|---|---|---|---|---|---|
| 0.0 | in | Disturbance_Input | BOOL | FALSE | |
| 0.1 | in | Acknowledge | BOOL | FALSE | |
| 0.2 | in | Flash_frequency | BOOL | FALSE | |
| 2.0 | out | Display | BOOL | FALSE | |
| | in_out | | | | |
| 4.0 | stat | Report_Memory | BOOL | FALSE | |
| 4.1 | stat | Edge_Memory_bit | BOOL | FALSE | |
| | temp | | | | |





**Características Especiais dos FBs**

Ao contrário da funções (FCs), os blocos de função (FBs) possuem uma (recall) memória de recorrência. Isso significa que um bloco de dados local está associado ao bloco de função, que é o chamado **bloco de dados instance** (instance data block). Ao fazer a chamada de um FB, deve ser especificado também o número do DB instance, o qual é automaticamente aberto.

Um DB instance é utilizado para salvar **variáveis estáticas**. Estas varáveis locais podem apenas ser utilizadas no FB, declaradas em sua tabela de declaração. No momento em que o bloco é finalizado elas são armazenadas.

**Parâmetros**

Na chamada do bloco de função, os valores dos parâmetros atuais são armazenados no DB instance.

Se um parâmetro atual não for associado a um parâmetro formal na chamada do bloco o último valor armazenado no DB instance para esse parâmetro é utilizado na execução do programa.

Podem ser especificados diferentes parâmetros atuais em cada chamada de FB.

Quando o bloco de função é finalizado, os dados no bloco de dados são retidos.

**Vantagens do FB**

- Ao escrever um programa utilizando um FC, é necessário buscar por endereços de memória disponíveis ou áreas de dados, e a organização destes é de responsabilidade do programador. As variáveis estáticas de um FB, por outro lado, são mantidas pelo software STEP 7.
- Utilizando as variáveis estáticas não se corre o risco de utilizar áreas de endereçamento de bit memories ou de dados duas vezes.
- Ao invés dos parâmetros formais “Memória" e “Mem_Flanco" do FC20, utilizam-se as variáveis estáticas “Memoria" e “Mem_Flanco” no FB. Isso torna a chamada do bloco mais simples, já que dois parâmetros não são mais necessários.

# Bloco de Função para Exibição de Mensagem

| Address | Declaration | Name | Type | Initial value | Comment |
|---|---|---|---|---|---|
| 0.0 | in | Disturbance_Input | BOOL | FALSE | |
| 0.1 | in | Acknowledge | BOOL | FALSE | |
| 0.2 | in | Flash_frequency | BOOL | FALSE | |
| 2.0 | out | Display | BOOL | FALSE | |
| | in_out | | | | |
| 4.0 | stat | Report_Memory | BOOL | FALSE | |
| 4.1 | stat | Edge_Memory_bit | BOOL | FALSE | |
| | temp | | | | |

Tabela de declarações do bloco de função

DB20 -- PRO1_32S\Chapter13

| Address | Declaration | Name | Type | Initial val | Comment |
|---|---|---|---|---|---|
| 0.0 | in | Disturbance_Input | BOOL | FALSE | |
| 0.1 | in | Acknowledge | BOOL | FALSE | |
| 0.2 | in | Flash_frequency | BOOL | FALSE | |
| 2.0 | out | Display | BOOL | FALSE | |
| 4.0 | stat | Report_Memory | BOOL | FALSE | |
| 4.1 | stat | Edge_Memory_bit | BOOL | FALSE | |

Bloco de dados Instance

SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.16

sitrain

**Exibição de Messagem**

No exercício prévio foi criado o bloco parametrizável FC 20 para a exibição de uma mensagem (indicação de um problema).

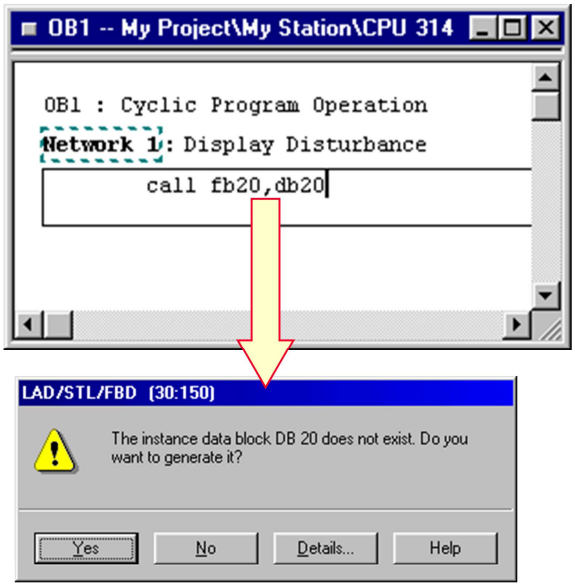
Ao invés de bit memories, utilizados no FC20 para salvar o sinal e sua detecção de flanco do RLO, pode-se utilizar as variáveis estáticas em um FB. Elas são armazenadas no DB instance referenciado ao FB.

**Estrutura do DB Instance**

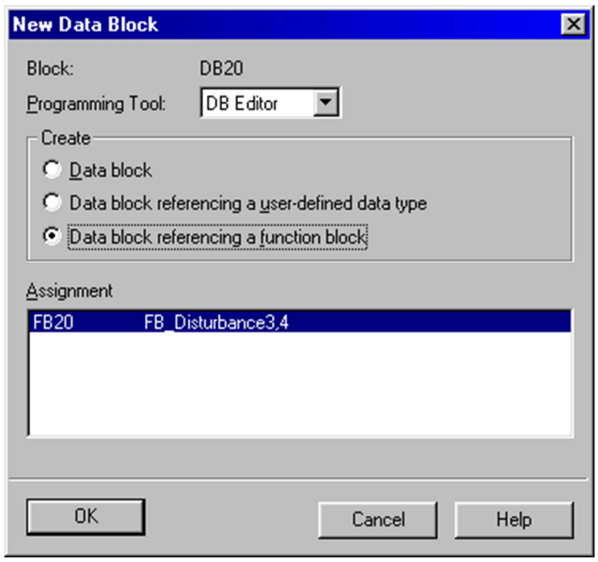
Quando um DB é criado e referenciado a um FB, o STEP7 cria a estrutura de dados do bloco de dados utilizando a estrutura especificada na tabela de declaração local do bloco de função. Após salvar o DB, o bloco de dados é criado e pode então ser utilizado como um DB instance.

# Gerando Blocos de Dados Instance

**1. Gerando o DB instance via chamada de FB**




**2. Criando um novo DB instance**








**Gerando um DB Instance**

Existem duas maneiras de se gerar um novo DB instance:

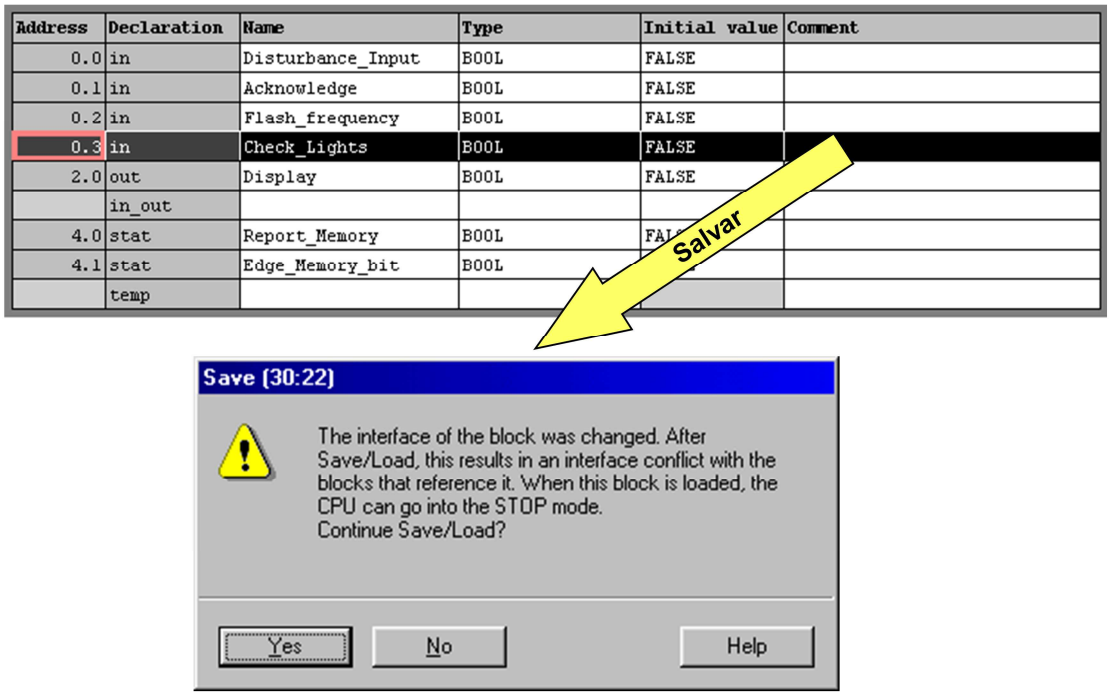
- Especificando na chamada do FB com qual DB instance ele irá trabalhar. Aparece a seguinte mensagem:
  "The instance data block DB x does not exist. Do you want to generate it?“
  (“O bloco de dados instance DB x não existe. Você deseja criá-lo?”).
- Selecionando, ao criar um novo DB, a opção "Data block referencing a function block“ (“Bloco de dados referenciando um bloco de função”).

**Notes**

Um DB instance pode apenas referenciar um FB. Contudo, um FB pode ser referenciado por diferentes DB instance cada vez que for chamado.

Se o FB for modificado (adicionando parâmetros ou variáveis estáticas), o DB instance deve ser gerado novamente.

# Atualizando (Inserindo / Apagando) Parâmetros de um Bloco








**Problema**

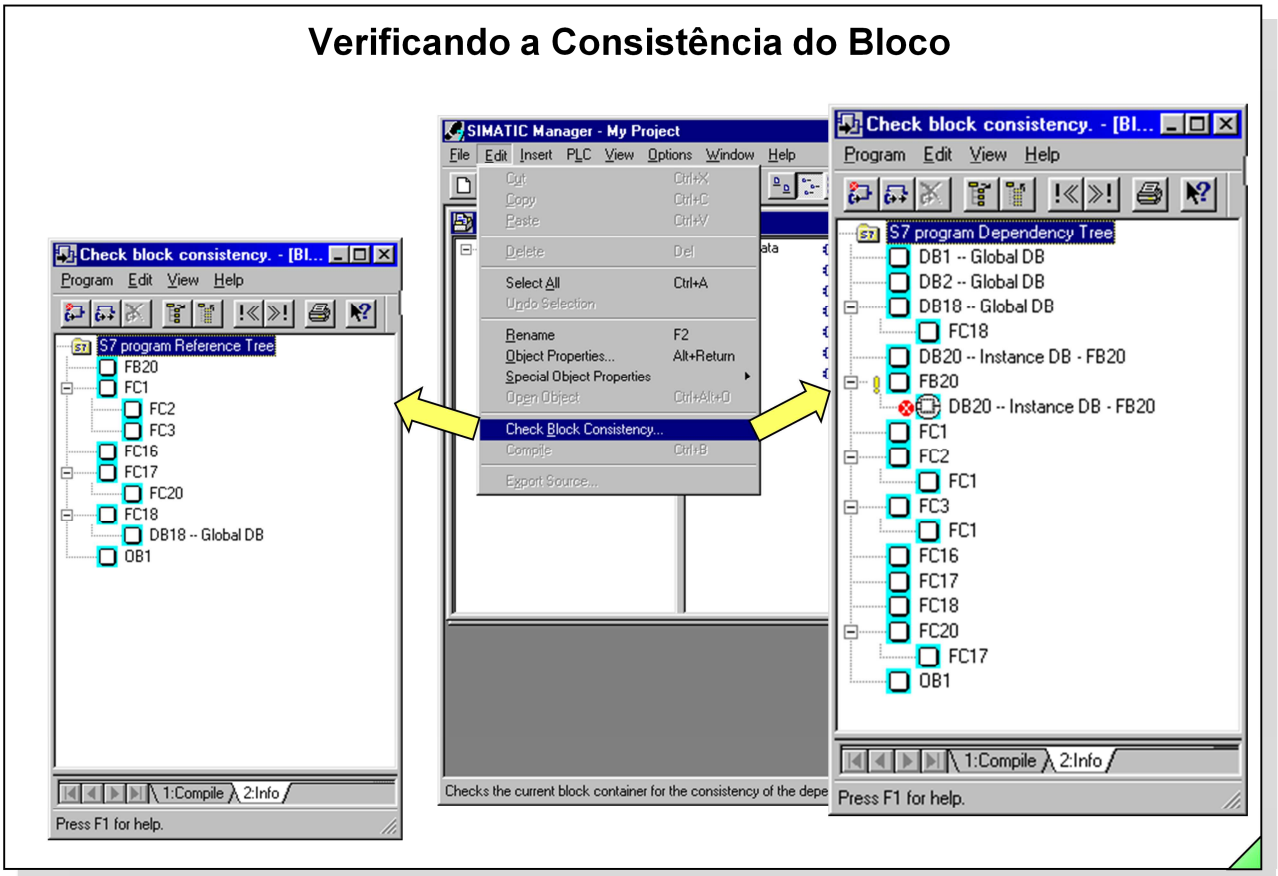
As modificações feitas nos blocos durante ou depois da criação do programa podem gerar conflitos (“time stamp conflicts”). Os conflitos podem, por sua vez, levar à inconsistências entre as chamadas e o bloco chamado ou o bloco referenciado, e portanto a um alto grau de correção.

Se parâmetros de um bloco forem adicionados ou apagados, após já ter sido feita a chamada no programa, torna-se necessário atualizar as chamadas dele nos outros blocos. Se isso não for feito, a CPU pode ir para STOP ou a execução correta do bloco de função não poderá mais ser garantida, já que os parâmetros formais adicionais não estão associados a parâmetros atuais na chamada.

No exemplo, o parâmetro de entrada adicional "Check_lights" foi inserido e deve ser associado a um parâmetro atual em todas as chamadas do bloco.

Ao salvar um bloco cuja interface foi modificada pela adição ou remoção de parâmetros formais, aparece uma mensagem de advertência à respeito de possíveis problemas.

# Verificando a Consistência do Bloco




SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.19

sitrain

**Área de Utilização**

A função *Check block consistency -> Compile* elimina uma grande parte de todos os conflitos e inconsistências de blocos.

Os conflitos de interface ocorrem quando a interface do bloco parametrizável é modificada, após feitas as chamadas do bloco nos outros blocos. As inconsistências de bloco também ocorrem, por exemplo, quando endereços são acessados simbolicamente, e a associação Símbolo <-> Endereço absoluto é modificada posteriormente na tabela global de símbolos ou nos blocos de dados.

Os blocos cujas inconsistências não puderem ser eliminadas automaticamente (por ex. por conflitos de interface), são indicados com símbolos (consulte o help online) e podem ser abertos e corrigidos pelo usuário utilizando o Editor usando o botão direito do mouse (por favor observe a página seguinte).

**Tree View...**

A visualização em formato de árvore (“tree view”) exibe a lógica / dependências de interface ou referências dos blocos da pasta de bloco selecionada. A árvore pode ser exibida tanto como uma Árvore de Dependência (“Dependency Tree”) ou como uma Árvore de Referência (“ReferenceTree”) utilizando *View -> Reference Tree / Dependency Tree*.

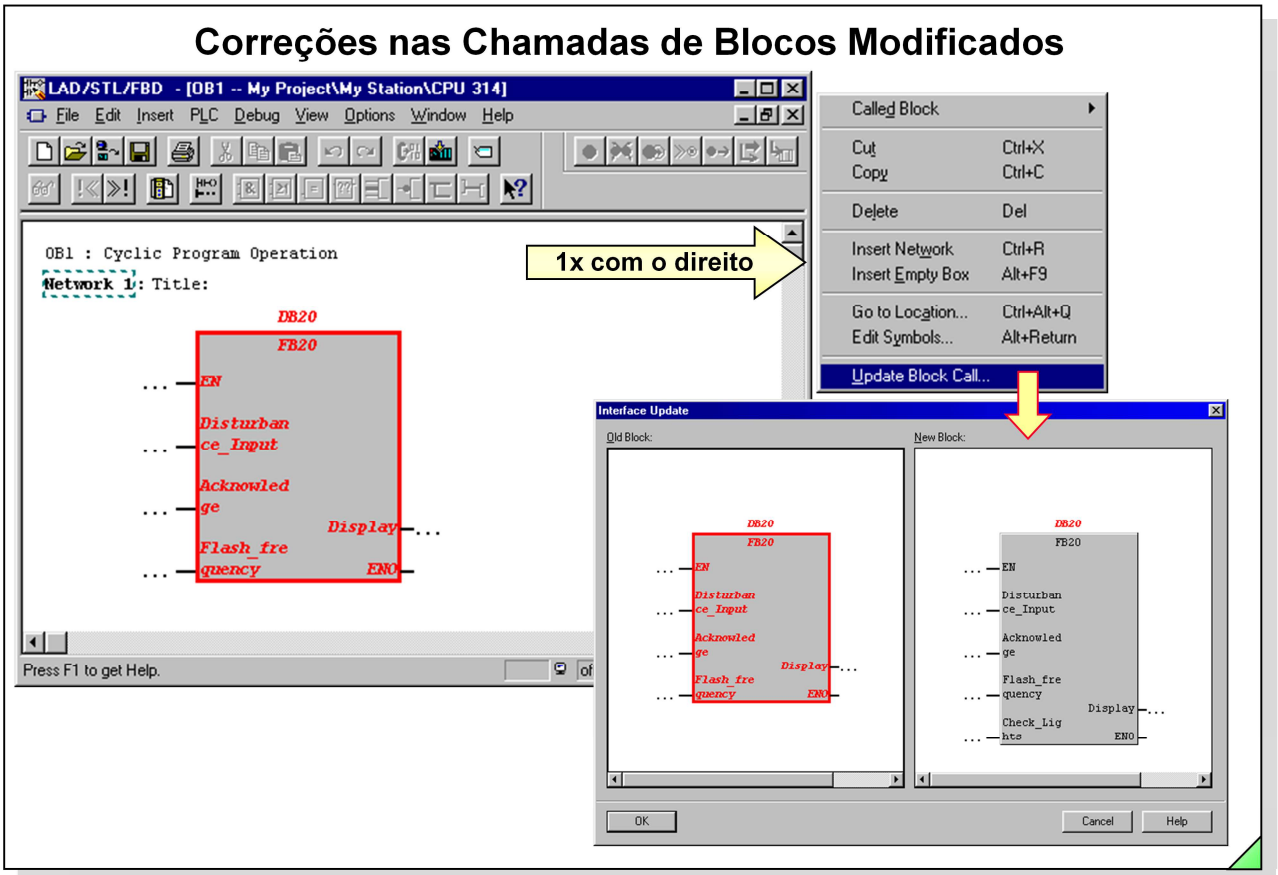
**...as Reference Tree**

A árvore de referência exibe em níveis, da esquerda para a direita, todas as dependências de seus blocos ou seu aninhamento. Assim como com o comando *Reference data / Program structure,* o caminho das chamadas é exibido da esquerda para a direita, iniciando do nível de aninhamento 1. Além disso, a árvore de referência fornece uma visão geral da profundidade do aninhamento nos níveis individuais de execução do programa.

**...as Dependency Tree**

A árvore de dependência exibe em níveis da esquerda para a direita as dependências de todos os blocos ou seu aninhamento. Neste caso os caminhos exibidos não iniciam-se do nível de aninhamento 1, mas sim dos blocos individuais. Assim, todos os blocos da pasta de blocos são listados no maior nível à esquerda. Os níveis seguintes (à direita) exibem as dependências ou os blocos a partir dos quais eles são chamados. Assim como com o comando *Reference data / Cross reference list,* a árvore de dependência fornece informação sobre quais outros blocos chamam cada bloco.

## Correções nas Chamadas de Blocos Modificados







Data: 26/08/2011
Arquivo: S7-Bas-09.20

sitrain

**Atualizando uma Chamada**

As chamadas inconsistentes do bloco (na figura o FC 20) são marcadas em vermelho no bloco aberto que faz a chamada (na figura o FC 17).

Clicando com o botão direito do mouse sobre a chamada inconsistente pode-se escolher a função *Update Block Call* na lista de opções. Uma janela é exibida na qual a chamada do bloco antiga (inconsistente) e a nova (na figura com o parâmetro adicional "Check_Lights") aparecem. Após confirmar com OK, é possível completar o parâmetro formal "Check_Lights" com o parâmetro atual restante.

O DB Instance é gerado novamente para blocos de função.

# Exercício: Editando um Bloco de Função

**1.** **Tabela de declarações do bloco FB 20**

| Address | Declaration | Name | Type | Initial value | Comment |
|---|---|---|---|---|---|
| 0.0 | in | Disturbance_Input | BOOL | FALSE | |
| 0.1 | in | Acknowledge | BOOL | FALSE | |
| 0.2 | in | Flash_frequency | BOOL | FALSE | |
| 2.0 | out | Display | BOOL | FALSE | |
| | in_out | | | | |
| 4.0 | stat | Report_Memory | BOOL | FALSE | |
| 4.1 | stat | Edge_Memory_bit | BOOL | FALSE | |
| | temp | | | | |

**2.** **Seção de programa do FB 20**

```
A   #Acknowledge
R   #Report memory
A   #Disturb...
:
:
```





**Objetivo**

Uma falha adicional de processo (chave do simulador) deve ser monitorada. A maneira mais fácil de fazer isso seria programar outra chamada do FC20.

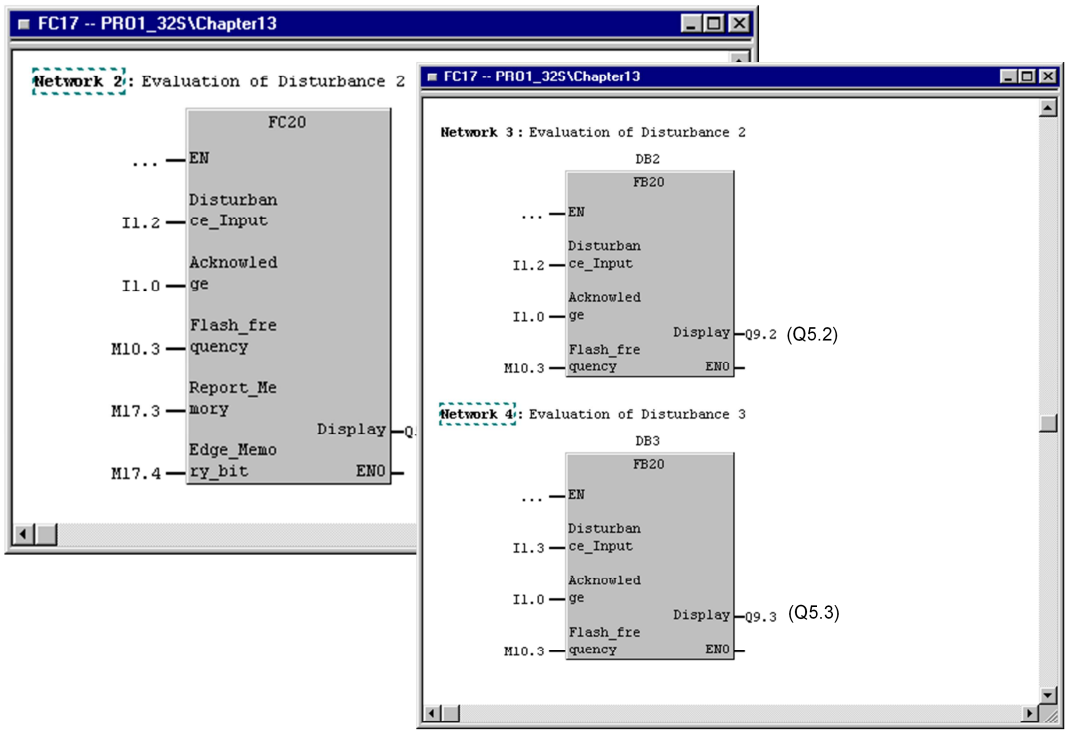
Contudo, para aproveitar as vantagens da solução utilizando FB programe o FB20 parametrizável para monitoração dessa falha.

Para o armazenamento dos bits *mem_flanco* e *memoria,* utilize variáveis estáticas armazenadas no DB instance do FB. Na figura pode ser observada a tabela de declaração do FB20 com os parâmetros de entrada e saída e o início do programa.

**O Que Fazer**

- Insira o bloco FB 20 dentro do programa S7 "Programa_1" .
- Declare os parâmetros formais e as variáveis estáticas do bloco como exibido na figura.
- Escreva um programa para o FB 20, copiando as partes de programa úteis do já programado FC 20.
- Salve o bloco e transfira-o para a CPU.

# Exercício: Chamando um Bloco de Função e Testando-o




SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.22

sitrain

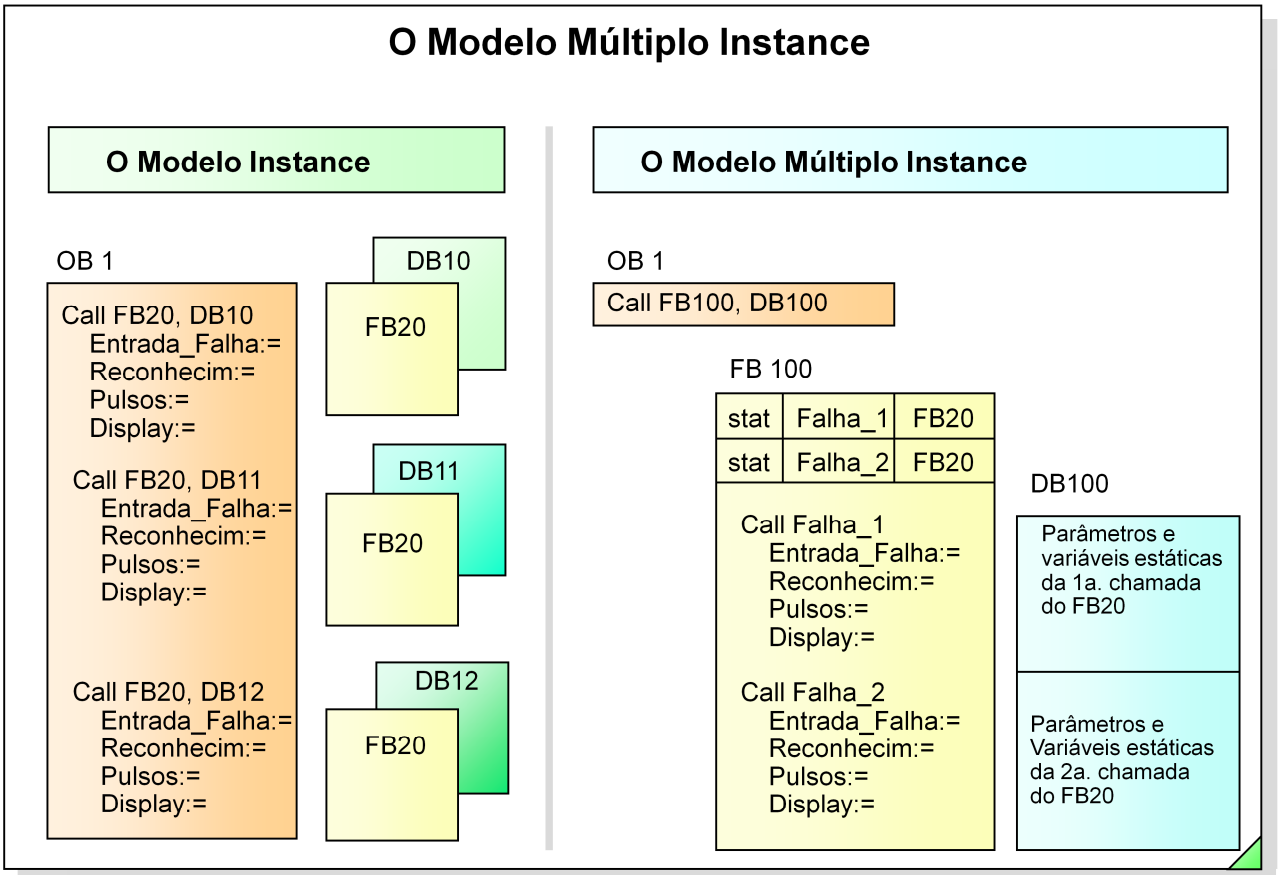
**Objetivo:**

A monitoração da Falha 2 (programada até agora através da chamada do FC20) e a monitoração da nova Falha 3 devem ser implementadas com o novo bloco FB 20 criado.

O bloco parametrizável FB 20 deve ser chamado duas vezes, cada vez com um diferente bloco de dados instance, no FC 17.

**O Que Fazer**

- No FC 17 apague a segunda chamada do FC 20, já que a monitoração da segunda falha deve ser implementada com o FB 20.
- Programe ambas chamadas do FB 20 como exibido na figura em dois novos networks no FC 17. Deixe o Editor gerar os DBs instance 2 e 3.
- Salve apenas o bloco modificado FC 17, por enquanto.
- Primeiro transfira os DBs instance 2 e 3 através do SIMATIC Manager para a CPU e depois o FC 17 modificado.
- Teste o funcionamento do programa.




SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.23

sitrain

**O Modelo Múltiplo Instance**

Até agora foram utilizados diferentes blocos de dados instance para cada. chamada de um bloco de função. Contudo o número de blocos de dados é limitado, e por essa razão existe um método que permite utilizar um DB instance comum para várias chamadas de FB.

O modelo múltiplo instance permite utilizar apenas um DB para várias chamadas. Para isso é necessário um FB adicional para gerenciar estes instances.
Para cada chamada de FB (FB 20), é definida uma variável estática no FB de maior nível (FB 100). Sendo assim na chamada do bloco Call Falha_1, não é necessário especificar um DB instance.

O FB de maior nível (FB 100) é chamado, por exemplo, no OB1, e o DB instance (DB 100) é gerado apenas uma vez.

**Nota**

Múltiplos instances são discutidos no curso de programação avançada.

# Exercício: Reconhecendo Tipos de Variáveis

| Address | Declaration | Name | Type | Initial val | Comment |
|---|---|---|---|---|---|
| 0.0 | in | Number_1 | WORD | W#16#0 | |
| 2.0 | in | Number_2 | WORD | W#16#0 | |
| 4.0 | out | Result | WORD | W#16#0 | |
| | in_out | | | | |
| 6.0 | stat | Max_Value | INT | 0 | |
| 0.0 | temp | Intermediate_result | INT | | |

TYPE OF VARIABLE

| Instrução | Global | Local | Absoluto | Simbólico | Tempor. | Estática | Parâmetro |
|---|---|---|---|---|---|---|---|
| L #Number_1 | | | | | | | |
| L #Number_2 | | | | | | | |
| T #Max_value | | | | | | | |
| L #Intermediate_result | | | | | | | |
| L “Number_1" | | | | | | | |
| T MW 40 | | | | | | | |
| T #Number_2 | | | | | | | |





**Objetivo**

Na seção de instruções da figura pode ser observado um programa com várias variáveis. Na tabela ao lado, associe as propriedades correspondentes às variáveis.

**O Que Fazer**

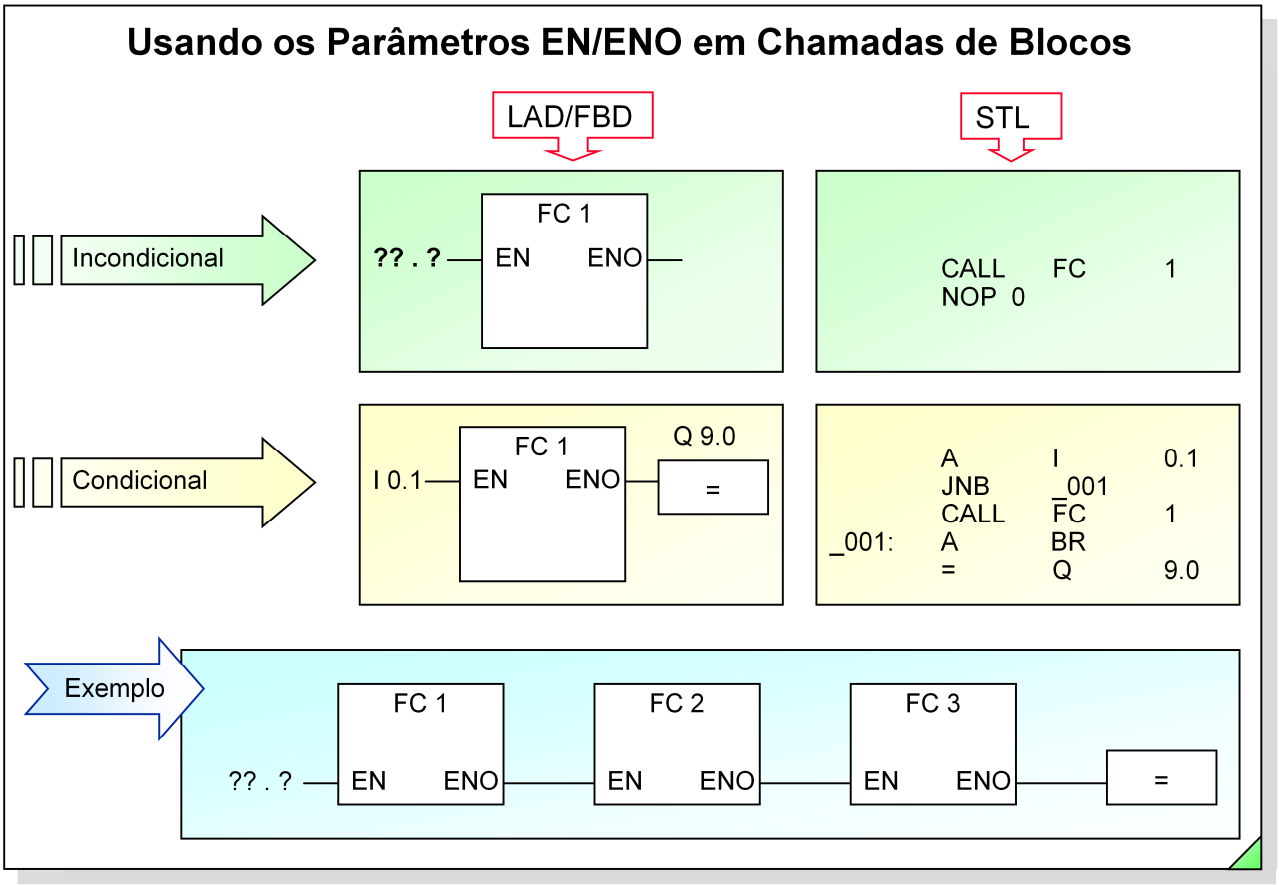
Na tabela, marque o tipo de dados correspondente com um X.

Responda a seguinte pergunta:
O que está incorreto na instrução T#Number_2 ?

.....

.....

.....




**FCs Standard**

Existem as seguintes regras para a execução dos FCs standard:

- Se EN=0, o bloco não é executado e o ENO também é =0.
- Se EN=1, o bloco é executado e se for executado sem erros o ENO também é =1.

  Se um erro ocorrer durante a execução do bloco, o ENO se torna =0.

**FCs do Usuário**

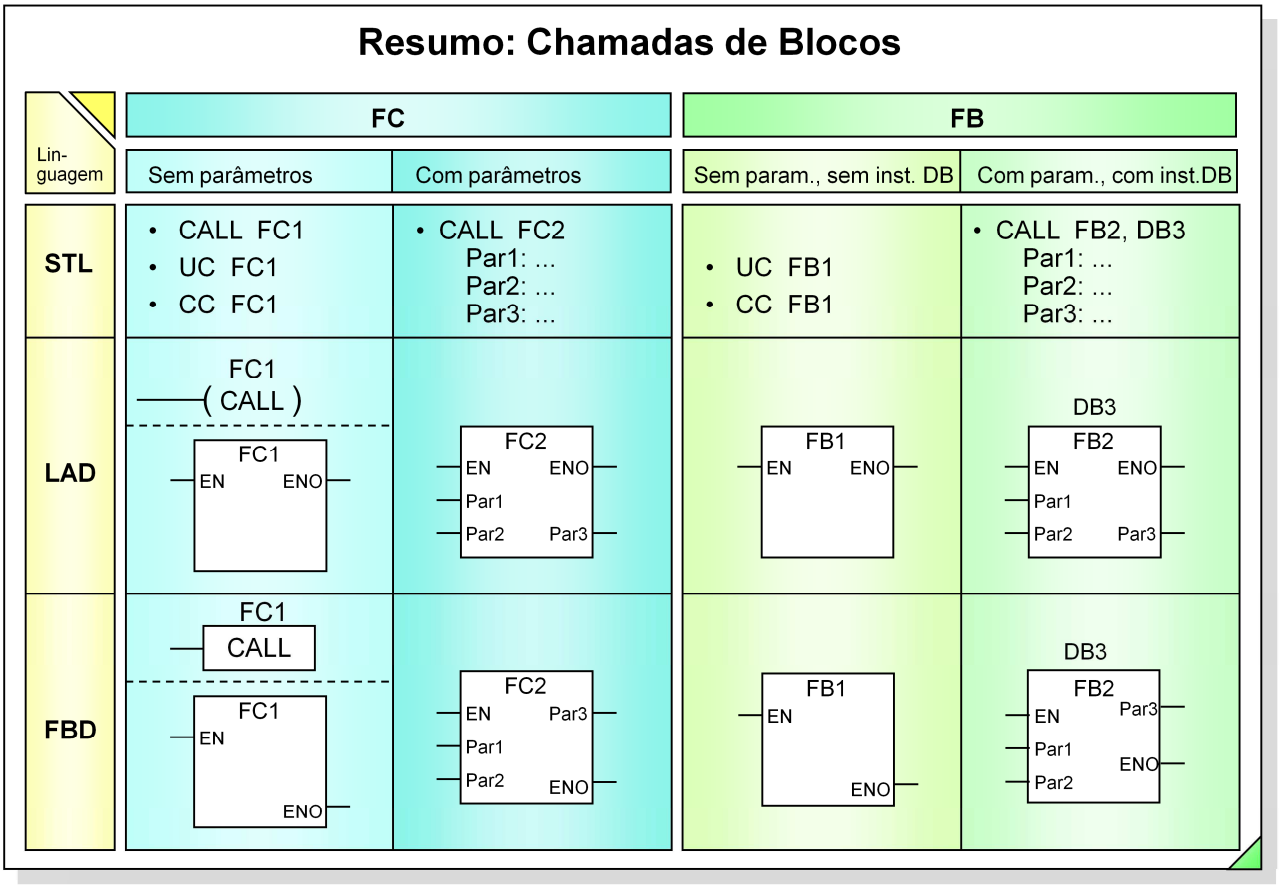
Não importa se o bloco do usuário foi escrito em LAD, FBD ou STL; ao ser chamado em LAD/FBD os parâmetros EN e ENO são adicionados de qualquer forma. Sendo assim é possível utilizar o RLO.

EN/ENO não existe em STL. Pode-se, contudo, emular os parâmetros.

Deve-se programar – independente da linguagem de programação – uma análise de erros.

**Interconexão**

Em LAD/FBD vários blocos podem ser agrupados, e conectados um ao outro logicamente utilizando os parâmetros EN / ENO.




SIMATIC S7



Data: 26/08/2011
Arquivo: S7-Bas-09.26

sitrain

**CALL**

As instruções "CALL" são utilizadas para chamadas de blocos de programa (FC, FB, SFC, SFB). Nas linguagens de programação gráfica LAD e FBD, a chamada de bloco pode ser feita dependente de uma condição (RLO) utilizando a entrada EN do bloco CALL. Na linguagem de programação STL a chamada de bloco é absoluta, isto é, dependente do RLO.

Ao chamar um FB ou SFB com "CALL", deve-se especificar o instance DB relevante. Pode ser utilizado tanto o nome absoluto quanto simbólico do bloco.

Por exemplo: "CALL FB2, DB2" ou "CALL valve, level".

**UC**

A instrução "UC" faz uma chamada incondicional do bloco FC ou FB. A operação UC é permitida apenas para blocos FC ou FB não parametrizáveis. Assim, nenhuma variável estática pode ser declarada num FB chamado com UC.

**CC**

A instrução "CC" faz uma chamada condicional do bloco FC ou FB. A operação CC é permitida apenas para blocos FC ou FB não parametrizáveis. Assim, nenhuma variável estática pode ser declarada num FB chamado com CC.

**Parâmetros**

Os parâmetros formais declarados na tabela de declaração de um bloco são a interface do bloco. Quando um FC parametrizável é chamado, um parâmetro atual deve ser transferido para cada parâmetro formal. Essa transferência de parâmetros não é obrigatória quando um FB é chamado.

Variáveis estáticas e temporárias não são parâmetros e por isso não fazem parte da interface do bloco. Sendo assim, não existe transferência de parâmetros para variáveis estáticas ou temporárias na chamada do bloco.