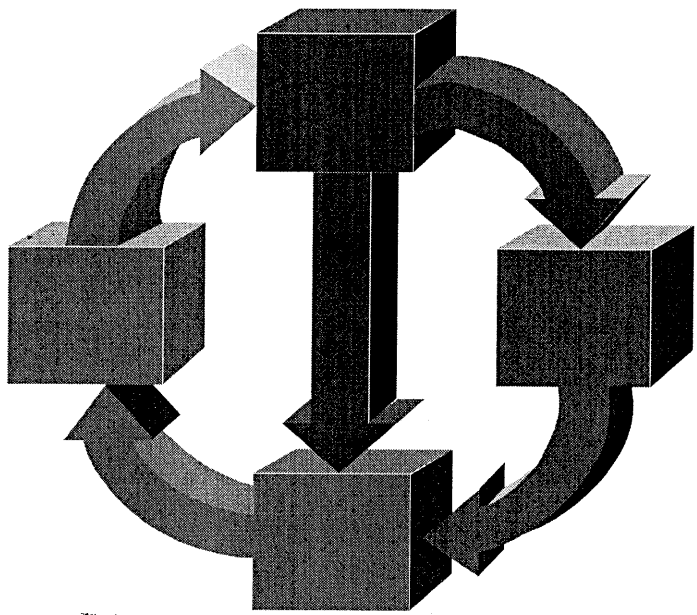
# Instrukce podmíněné hodnotou stavového bitu



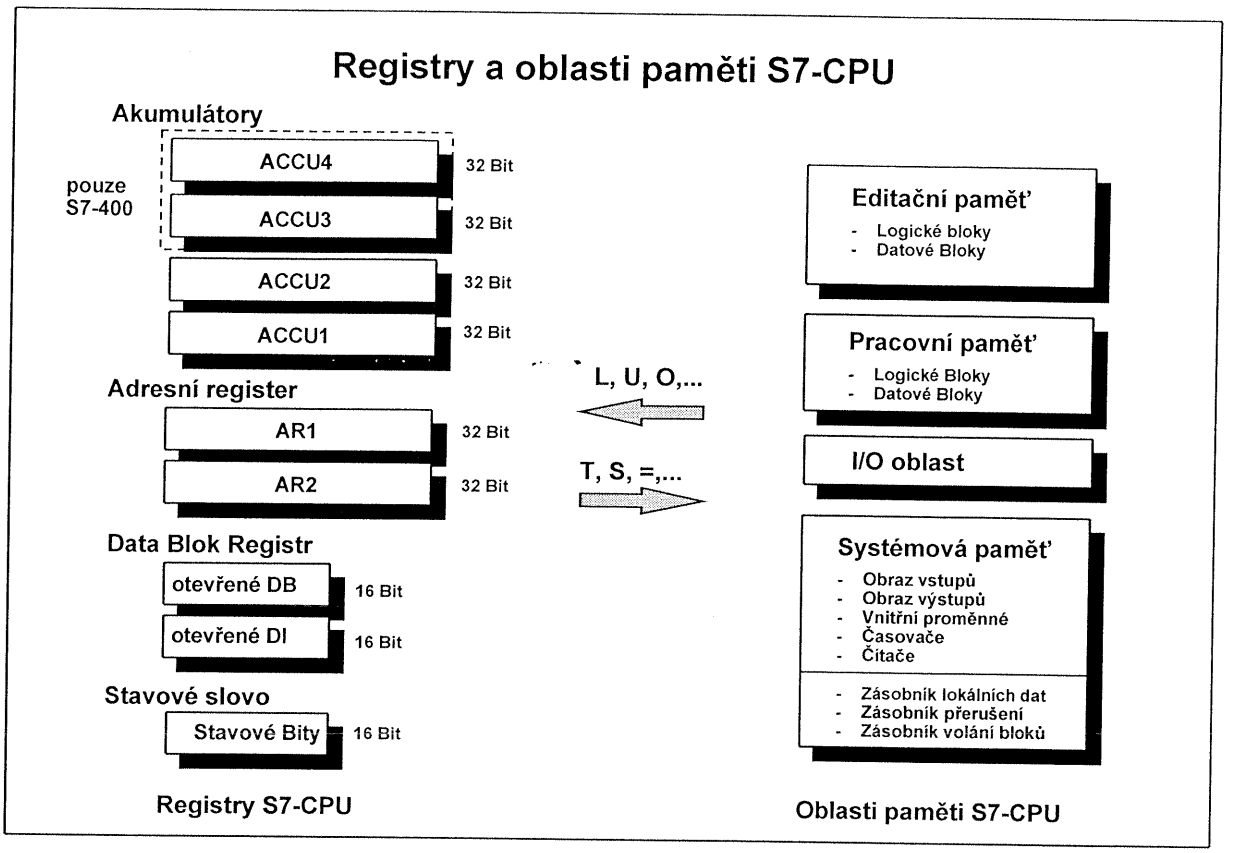
SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_01cz.1

Školicí středisko
firmy E&A spol. s r.o.

## Obsah








**Přehled**

S7-CPU obsahuje různé registry a oblasti paměti, které zajišťují správné provádění uživatelského programu.

**Registry CPU**

Registry CPU slouží ke zpracování adres nebo procesních údajů. S použitím, k tomu určených instrukcí (L, T,...),mohou být tato data přenášena mezi registry a oblastmi paměti.

- Akumulátory: Ke zpracování aritmetických, porovnávacích a jiných bytových, slovních nebo dvouslovních instrukcí jsou použity dva (u S7-300) nebo čtyři (u S7-400) akumulátory.
- Adresní registr: Dva adresní registry slouží ke zpracování ukazatelů při nepřímé adresaci paměti.
- Registr datových bloků: Tento registr obsahuje číslo otevřeného (aktivního) datového bloku. S7 umožňuje současné otevření 2 datových bloků. Jeden datový blok je otevřen s využitím DB registru, k otevření druhého (instančního) datového bloku slouží DI registr.
- Stavové slovo je složeno z bitů, které obsahují výsledky nebo stavy jednotlivých instrukcí během zpracování uživatelského programu.

**Oblasti paměti**

Paměť CPU S7 je rozdělena do 4 oblastí:

- *LOAD Memory* - editační paměť slouží k uložení uživatelského programu (bez symbolických adres a komentářů). Tato paměť může být typu RAM nebo FEPROM (Flash-EPROM).
- *WORKING Memory* (integrovaná RAM paměť) slouží k uložení částí programu, které jsou nezbytné pro zpracování uživatelského programu.
- I/O oblasti umožňují přímý přístup k periferním jednotkám a připojeným signálním modulům s jejich vstupy a výstupy nebo periferními zařízeními.
- Systémová RAM paměť obsahuje oblasti jako je např. tabulka obrazu vstupů a výstupů, vnitřní proměnné (M-bity), časovače, čítače a zásobníky lokálních dat, přerušení a volání bloků.

# Stavové slovo

## Význam jednotlivých bitů ve stavovém slovu

| Bit | Význam | Hodnota | Poznámka |
|---|---|---|---|
| 0 | /FC | $2^0$ | První kontrola |
| 1 | RLO | $2^1$ | Výsledek logické operace |
| 2 | STA | $2^2$ | Stav |
| 3 | OR | $2^3$ | Logický součet |
| 4 | OS | $2^4$ | Pamatované přetečení |
| 5 | OV | $2^5$ | Přetečení |
| 6 | CC0 | $2^6$ | Výsledkový bit 0 |
| 7 | CC1 | $2^7$ | Výsledkový bit 1 |
| 8 | BR | $2^8$ | Binární výsledek |
| 9...15 | bez významu | $2^9$ ..$2^{10}$ | |





**Stavové slovo**

Jednotlivé bity stavového slova obsahují informace o výsledku nebo stavu instrukcí jako je např. zachycená chyba.

Tyto stavové bity lze přímo využít v uživatelském programu a pomocí binárních instrukcí tak např. určovat běh programu.

**První kontrola**

Stavový bit 0 signalizuje první kontrolu. Logická hodnota "0" v tomto bitu signalizuje, že následující logická instrukce zahajuje nový logický řetězec v uživatelském programu. Lomítko (/) před názvem bitu znamená, že jeho logická hodnota je negovaná.

**Výsledek logické operace**

Stavový bit 1 je pojmenován RLO (RLO= *Result of Logic Operation*). Slouží jako dočasná paměť pro binární logické operace.
Umožňuje logickou kombinaci mezi výsledkem předchozí a následující logické operace. Výsledek - hodnota této logické kombinace je opět uložena do RLO bitu.

**Stavový bit**

Stavový bit 2 obsahuje hodnotu adresované bitové proměnné.

**Logický součet**

Stavový bit OR je používán v případě, že uživatelský program provádí operaci AND před OR s pomocí instrukce *O*. OR bit signalizuje, že výsledek předchozí instrukce AND je *1*, čímž je dáno, že výsledek operace OR je také *1*.

**Přetečení**

Bit OV (overflow - přetečení) signalizuje chybu při zpracování matematické nebo porovnávací instrukce při práci s číslem s plovoucí desetinnou čárkou. Tento bit je nastaven podle výsledku provedené operace.

## Testování stavových bitů

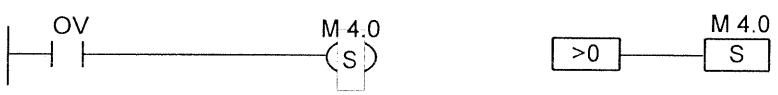
- **Testování v STL**

| | |
|---|---|
| A OV | Test přetečení |
| A OS | Test pamatovaného přetečení |
| A BR | Test *Binary Result* bitu |

- **Test výsledkových bitů (CC0, CC1)**

| | |
|---|---|
| A ==0 | Výsledek je roven 0 |
| A > 0 | Výsledek je větší než 0 |
| A <>0 | Výsledek je různý od 0 |
| A =<0 | Výsledek je menší nebo roven 0 atd. |
| A UO | Operace není přípustná |

**Testování v LAD a v FBD**




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_01cz.4

Školicí středisko firmy E&A spol. s r.o.

**OS - pamatované přetečení**

Bit OS (pamatované přetečení - overflow stored) je nastaven společně s bitem OV. Bit OS však nemění svojí hodnotu s výsledkem další matematické nebo porovnávací operace. Tento je resetován pouze použitím instrukcí JOS (skok je-li OS=1), volání bloků a konec bloku.

**Bity CC1 a CC0**

Bity CC1 a CC0 (*condition codes*) informují o výsledku :

- matematických instrukcí
- porovnávacích instrukcí
- slovních logických instrukcí
- instrukcí a funkcí posunu.

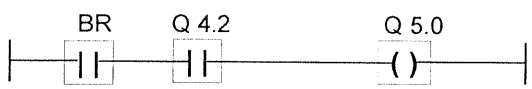
Bity CC1 a CC0 mohou být testovány nepřímo na kombinaci obou bitů instrukcemi podmíněných skoků. Testovány mohou být následující kombinace :

| CC1 | CC0 | Význam | |
|---|---|---|---|
| 0 | 0 | A ==0 | Výsledek = 0 (ACCU2 = ACCU1) |
| 1 | 0 | A >0 | Výsledek > 0 (ACCU2 > ACCU1) |
| 0 | 1 | A <0 | Výsledek < 0 (ACCU2 < ACCU1) |
| 1 | 1 | A UO | Nepřípustná operace (tj. dělení nulou). |

**LAD/FBD**

Instrukce pro testování kombinace CC bitů jsou v zobrazeních LAD a FBD k dispozici v *Katalogu elementů* ve skupině *Status Bits*.

## Instrukce pro zpracování stavových bitů

| Instrukce | Význam | Příklad |
|---|---|---|
| SET | Nastaví RLO do *1* | SET<br>= M 0.1 |
| CLR | Nastaví RLO do *0* | CLR |
| NOT | Neguje RLO | O Manual<br>O Automatic<br>NOT;<br>= operating modes<br>= M0.0 |
| SAVE | Uloží RLO do BR bitu |  |
| A BR | Test bitu ***B**inary **R**esult* |  |







**Instrukce L STW/T STW**

STEP 7 umožňuje také uložení stavového slova do proměnné a jeho opětovné nahrání zpět pro pozdější zpracování.

- L STW — Nahraj stavové slovo
- T MW 114 — Ulož do *memory word* 114

Instrukce T STW umožňuje obnovení dříve uložené hodnoty stavového slova.

**Změna hodnoty RLO**

STEP7 obsahuje také instrukce pro změnu hodnoty RLO bitu.
Instrukce SET umožňuje nepodmíněné nastavení bitu RLO do 1, instrukce CLR naopak nastaví RLO do 0. Současně s bitem RLO je nastavena také logická hodnota bitu STA.
Instrukce SET a CLR také resetují stavové bity OR a /FC, tzn. následující logická instrukce zahajuje nový logický řetězec.

Instrukce NOT nastaví bit RLO do opačné hodnoty, než jakou momentálně má.

**Bit BR**

BR bit představuje vnitřní paměťový bit, do kterého může být uložena hodnota RLO bitu předtím, než bude změněna ovlivňující instrukcí.

V grafických režimech zobrazení (LAD,FBD) odpovídá bit BR výstupnímu parametru ENO.

**Nastavení BR**

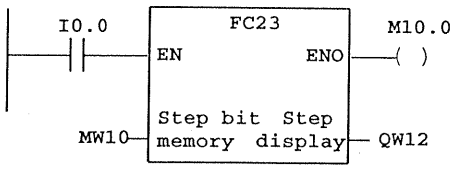
Instrukce SAVE umožňuje uložení hodnoty RLO bitu do bitu BR.

SAVE je zpracováván nezávisle na vnějších podmínkách (nepodmíněně) a neovlivňuje žádným způsobem ostatní stavové bity.

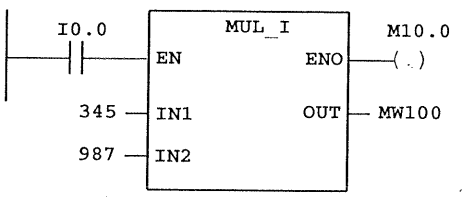
## Bit BR a parametr ENO ve volání bloků nebo komplexních funkcí

### LAD

Network 1: Cyklický program




Network 2: ???




### STL

```
Network 1: Cyklický program
      A     I    0.0
      JNB   _001
      CALL  FC   21
       Step bit memory     :=MW10
       Step display        :=AW12
_001: A     BR
      =     M    10.0
```

```
Network 2: ???
      L     345
      L     987
      +I
      T     MW   100
      AN    OV
      SAVE
      CLR
      A     BR
      =     M    10.0
```





**EN = Enable input**

„Vstupní" parametr *EN* umožňuje definovat volání bloku jako podmíněné obdobně, jako je tomu ve STEPu 5.

- Jestliže EN není aktivní, tj. stav signálu = 0, potom volaný blok není zpracován.V důsledku toho není nastaven ani výstupní parametr ENO.
- Je-li EN aktivní, tj. stav signálu = 1, potom je vyvolání bloku provedeno a blok je odpovídajícím způsobem zpracován. Podle výsledku zpracování je také nastaven výstupní parametr ENO.

**ENO = Enable output**

„Výstupní" parametr *ENO* signalizuje do okolního uživatelského programu zpracování volaného bloku. Oznamuje, zda byl volaný blok zpracován s chybou nebo bez chyby. Reprezentuje hodnotu stavového bitu *BR*.
Stavový bit *BR* je při zahájení zpracování prioritní třídy nastaven na hodnotu 1. Následně může být bit *BR* změněn pouze uživatelským programem ve volaném bloku, již ne systémem.

Je-li při zpracování bloku zachycena chyba, může být tento chybový stav signalizován okolnímu uživatelskému programu vynulováním stavového bitu *BR*. Po zpracování volaného bloku (v zobrazení LAD/FBD) je hodnota *BR* bitu zkopírována do výstupního parametru *ENO*.

**Poznámka**

„Parametr" *EN* není pravým vstupním parametrem. Je-li definován, jsou do uživatelského programu automaticky vloženy dvě další instrukce skoku na návěští.

Podobně také parametr *ENO* není pravým výstupním parametrem bloku. Je-li definován, jsou do uživatelského programu automaticky vloženy instrukce , které zkopírují hodnotu *BR* bitu do přiřazené proměnné.

## Instrukce skoků podmíněné stavovými bity

| | |
|---|---|
| JU Label[1)] | Nepodmíněný skok |
| JC Label[1)] | Skok, je-li *RLO* =1 |
| JCN Label[1)] | Skok, je-li *RLO* = 0 |
| JCB Label[1)] | Skok, je-li *RLO* = 1 a uložení *RLO* do *BR* |
| JNB Label[1)] | Skok, je-li *RLO* = 0 a uložení *RLO* do *BR* |
| JBI Label[1)] | Skok, je-li *BR* = 1 |
| JBNI Label[1)] | Skok, je-li *BR* = 0 |
| JO Label[1)] | Skok, je-li *OV* =1 |
| JOS Label[1)] | Skok, je-li *OS* =1 |

[1)] Návěští může obsahovat až 4 alfanumerické znaky

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_01cz.7

Školicí středisko
firmy E&A spol. s r.o.

**Význam**

Logické řídící funkce umožňují přerušit lineární zpracování uživatelského programu a pokračovat ve zpracování na jiném místě bloku. Větvení uživatelského programu může být prováděno nepodmíněně - tj. za každé situace, nebo podmíněně - tj. při splnění definovaných logických podmínek.

**Nepodmíněný skok**

Instrukce *JU* je provedena vždy nezávisle na vnějších podmínkách. Tato instrukce přeruší lineární zpracování uživatelského programu a pokračuje ve zpracování od místa definovaného návěštím. *JU* žádným způsobem nemění stavové bity.

**Funkce skoku s RLO a BR**

Větvení S7-programu lze realizovat také jako podmíněné a to hodnotou stavového bitu *RLO* nebo *BR*. Dalším rozšířením možností je uložení hodnoty *RLO* bitu do bitu *BR* v okamžiku provedení skoku.
Na *RLO* závislé instrukce skoku (*JC*, *JCN*) mění hodnotu některých stavových bitů. Stavové bity *STA* a *RLO* jsou nastaveny do 1 a bity *OR* a */FC* do 0 nejenom při splnění logické podmínky ale také i při nesplnění této podmínky.

*RLO* ukládající skoky (*JCB*, *JNB*) uloží vždy hodnotu bitu *RLO* do bitu *BR*. Zbývající bity STA, OR a /FC jsou zpracovány stejným způsobem jako u instrukcí, které hodnotu bitu RLO neukládají.

Instrukce skoku závislé na hodnotě bitu *BR* (JBI, JNBI) nastavují *STA* bit do 1 nejenom při splnění logické podmínky ale také při nesplnění této podmínky tj. vždy. Stejně tak jsou do 0 nastavovány bity *OR* a */FC*. Bity *RLO* a *BR* si zachovávají svoji původní hodnotu.

## Instrukce skoků podmíněné stavovými bity CC0 a CC1

| | |
|---|---|
| JZ Label[1)] | Skok, je-li stavový bit *CC1* =0 a *CC0* =0 (Výsledek operace = 0) |
| JN Label[1)] | Skok, je-li stavový bit *CC1* různý od *CC0* (Výsledek operace <> 0) |
| JP Label[1)] | Skok, je-li stavový bit *CC1* =1 a *CC0* =0 (Výsledek operace > 0) |
| JM Label[1)] | Skok, je-li stavový bit *CC1* =0 a *CC0* =1 (Výsledek operace < 0) |
| JPZ Label[1)] | Kombinuje skoky *JZ* a *JP* (Výsledek operace >= 0) |
| JMZ Label[1)] | Kombinuje skoky *JM* a *JZ* (Výsledek operace <= 0) |
| JUO Label[1)] | Skok, je-li operace nepřípustná např. neplatné reálné číslo nebo dělení nulou. |

[1)] Návěští může obsahovat až 4 alfanumerické znaky





**Skoky podmíněné bity OV a OS**

Skoky JO a JOS jsou provedeny, jestliže je zachyceno přetečení. Má-li být řetězec matematických operací ukončen úspěšně, musí být po provedení každé matematické instrukce vyhodnocena hodnota stavového bitu OV. Následující matematická instrukce v řetězci výpočtu, jejíž výsledek je v povoleném rozsahu číselného formátu totiž stavový bit OV nastaví do 0.
K vyhodnocení možného přetečení číselného rozsahu na konci výpočtového řetězce je určen stavový bit OS(*Overflow Stored* - pamatované přetečení). Stavový bit je nulován pouze při volání bloku, konci bloku nebo při zpracování instrukce skoku JOS.

Ostatní bity stavového slova nejsou při zpracování instrukce JO nebo JOS ovlivněny.

**Skoky podmíněné bity CC0 a CC1**

Zpracování jednotlivých částí uživatelského programu lze také podmínit kombinací logických hodnot bitů CC0 a CC1. Lze tak např. kontrolovat, zda výsledek mat. operace je kladný, záporný nebo roven nule. Skokové instrukce závislé na hodnotách bitů CC0 a CC1 žádným způsobem hodnoty těchto bitů nemění.

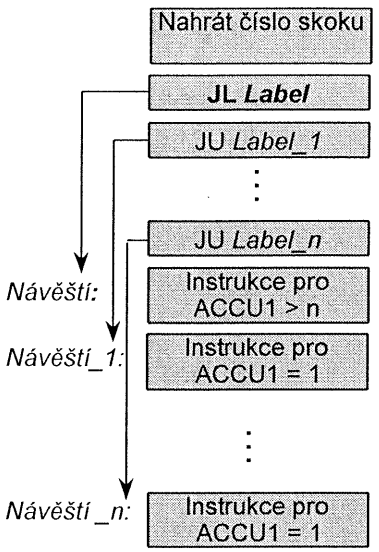
**Příklad**

Odčítání dvou celých čísel s mezivyhodnocením výsledku:

```
      L MW2
      L MW8
      -I
      JZ ZERO     // skok, je-li výsledek = 0
                  // činnost, je-li výsledek nenulový
ZERO: NOP 0       // činnost, je-li výsledek odečítání = 0
```

## Instrukce JL - distributor

**Režim činnosti**




**Příklad: Distibutor pro 3 varianty**

```
        L RecNo     // nahrát číslo receptu
        JL OVER     // distributor 3 možností
        JU Rec0     // skok na rec. č. 0 (ACCU1 = 0)
        JU Rec1     // skok na rec. č. 1 (ACCU1 = 1)
        JU Rec2     // skok na rec. č. 2 (ACCU1 = 2)
OVER:   JU ERROR    // skok pro ACCU1 > 2
        ...
Rec0:   L DBW4
        ...
Rec1:   L DBW6
        ...
Rec2:   L DBW8
        ...
ERROR:  CLR
        ...
```

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_01cz.9

Školicí středisko firmy E&A spol. s r.o.

**Instrukce JL**

*JL* distributor umožňuje cílené skoky na části programového bloku v závislosti na čísle skoku. Instrukce *JL* pracuje se seznamem instrukcí *JU*.

Tento seznam následuje bezprostředně po instrukci *JL* a může obsahovat maximálně 255 položek.

Mezi instrukce *JL* <návěští> a <návěští>: <instrukce> smí být umístěna pouze instrukce *JU*. Jestliže obsah akumulátoru *ACCU1-L-L* je *0*, je provedena první instrukce skoku, je-li obsah *1* , provede se druhá instrukce skoku atd. Je-li obsah akumulátoru větší než délka seznamu *JU* instrukcí je použito návěští uvedené spolu s instrukcí *JL*.

Instrukce *JL* je nepodmíněná a je provedena bez ohledu na vnější podmínky a pro jednotlivé části programu - varianty nemění stavové bity.

**Poznámka**

Skoky *JU* mohou být provedeny v rámci celého programového bloku (i z jednoho segmentu do druhého) a proto je nutné, aby návěští skoků byla jedinečná v rámci celého bloku programu.

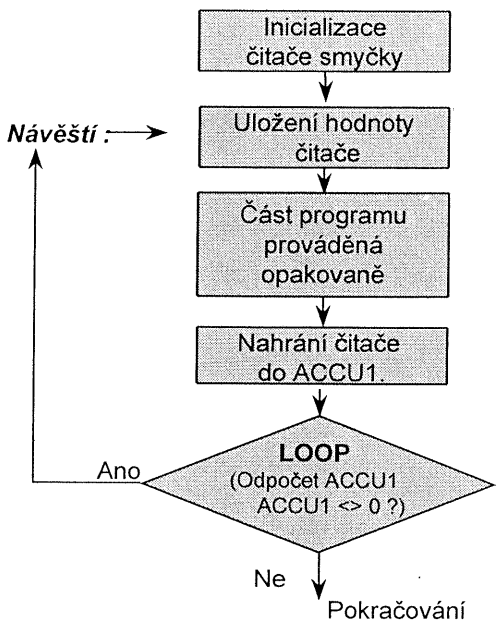
Skoky je možné provádět pouze uvnitř programového bloku. Délka návěští skoku je omezena na 4 alfanumerické znaky, s podmínkou, že první znak návěští smí být pouze písmeno. STEP 7 rozlišuje malá a velká písmena v názvu návěští.

Za znakem ":" oddělujícím název návěští musí být uvedena nějaká instrukce(minimálně instrukce NOP 0).

Maximální délka skoku je omezena na -32768 nebo +32767 slov programového kódu MC7. Aktuální délka skoku ( počet přeskočených instrukcí) závisí na jednotlivých instrukcích, které mohou být jedno-, dvou- nebo tříslovní.

## Instrukce Loop - smyčka

### Režim činnosti




### Příklad: instrukce Loop se 4 průběhy

```
        L  +4          // Nahrání počtu opakování
NEXT:   T  MW10        // Začátek smyčky
        .              // program
        .              // program
        .              // program
        L  MW10        // nahrání čitače smyčky
        LOOP NEXT      // Odpočet čitače (-1) a
                       // skok na návěští NEXT, je-li
                       // Accu 1> 0
```

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_01cz.10

Školicí středisko
firmy E&A spol. s r.o.

**Smyčka**

Instrukce smyčky *LOOP* umožňuje snadnou tvorbu programových smyček.

Počet opakování smyčky je definován v ACCU1-L. *LOOP* interpretuje pravé slovo akumulátoru 1 jako 16-bitové číslo bez znaménka v rozsahu od 0 do 65535.

Při každém zpracování instrukce *LOOP*, je hodnota v *ACCU1-L* snížena o *1*. Následně je porovnána tato hodnota s *0*. Je-li hodnota nenulová je proveden skok na návěští uvedené u instrukce *LOOP*. Pokud je hodnota v *ACCU1-L* rovna nule, skok se neprovede a zpracování programu pokračuje instrukcí, která je uvedena za instrukcí *LOOP*.

**Poznámka**

Čítač smyčky *LOOP nesmí být inicializován na 0.* V tomto případě by byla instrukce smyčky zpracována 65535 krát.

## Instrukce ukončení bloku

- **BE** — **B**lock **E**nd - konec bloku
- **BEU** — **B**lock **E**nd **U**nconditional (nepodmíněný, uvnitř bloku)
- **BEC** — **B**lock **E**nd **C**onditional (podmíněný, závislý na hodnotě RLO)

—(RET) **v zobrazení LAD**

—[RET] **v zobrazení FBD**

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_01cz.11

Školicí středisko
firmy E&A spol. s r.o.

**Funkce ukončení bloku**

Zpracování bloku lze ukončit pomocí instrukce BEC v závislosti na hodnotě bitu *RLO,* nebo nepodmíněně, pomocí instrukcí *BEU* a *BE.*

**BE**

Instrukce *BE* ukončuje zpracování aktivního programového bloku. Tato instrukce je vždy poslední instrukcí programového bloku. Je automaticky generována STEPem 7 při ukládání programového bloku.

Operační systém pokračuje ve zpracování nadřízeného bloku od instrukce, která bezprostředně následuje po instrukci skoku do ukončeného programového bloku. Současně je aktivována rezervovaná oblast lokálních dat.

**BEU**

Instrukce *BEU* ukončuje zpracování programového bloku podobně jako *BE.*

Na rozdíl od instrukce *BE,* ale smí být použita opakovaně, podle potřeby a to i uvnitř programového bloku.

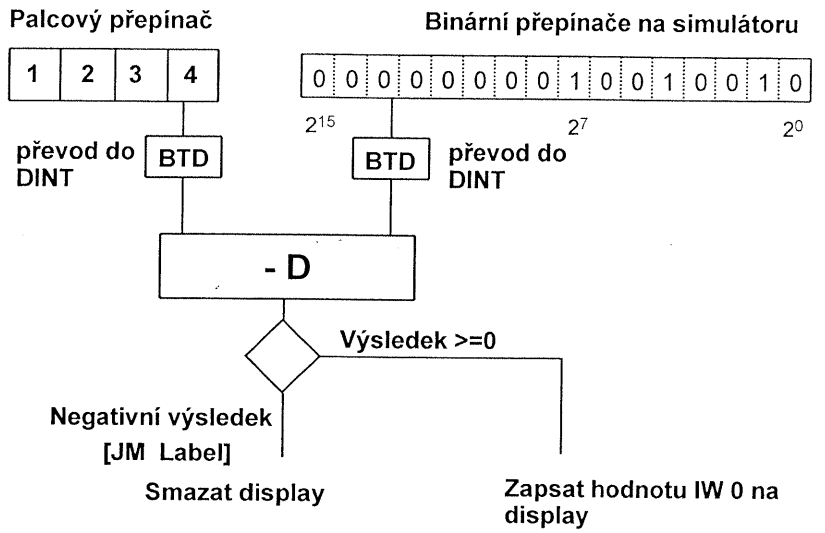
**BEC**

Tato instrukce umožňuje ukončit zpracování programového bloku v závislosti na hodnotě stavového bitu *RLO.* Je-li *RLO=1* zpracování bloku se ukončí a CPU pokračuje ve zpracování nadřízeného programového bloku instrukcí, která následuje za instrukcí volání právě ukončeného programového bloku.

Rezervovaná oblast lokálních dat je opět zpřístupněna.

Je-li *RLO=0*, BEC instrukce není zpracována. CPU nastaví *RLO* do *1* a pokračuje zpracováním instrukce následující po *BEC.*

# Cvičení 1.1: Skok po odečítání




**Palcový přepínač:** **S7-300: IW4 (IW2, 32-Bit. verze)**
**S7-400: IW30**

**Binární přepínače:** **S7-300: IW0 (IW0)**
**S7-400: IW28**

**Display:** **S7-300: QW12 (QW6, 32-Bit. verze)**
**S7-400: QW38**





**Přehled** Skokové instrukce umožňují přerušit lineární zpracování programu a pokračovat na jiném místě.

**Cíl cvičení** Naprogramovat instrukci skoku, která bude provedena v závislosti na výsledku odečítání.

**Zadání** Vytvořit projekt PRO2 a následně složku S7-programu s názvem EXERCISE a vytvořit FC 11 s následující funkcí:

1. Nahrát vstupní slovo palcového přepínače jako BCD hodnotu do akumulátoru.
2. Provést převod z BCD kódu do DINT. Pro převod použít instrukci BTD (BCD_TO_DINT). Totéž provést pro hodnotu načtenou z binárních přepínačů simulátoru.
3. Odečíst od hodnoty z palcového přepínače hodnotu z simulátoru.
4. Podle výsledku operace provést následující akce:

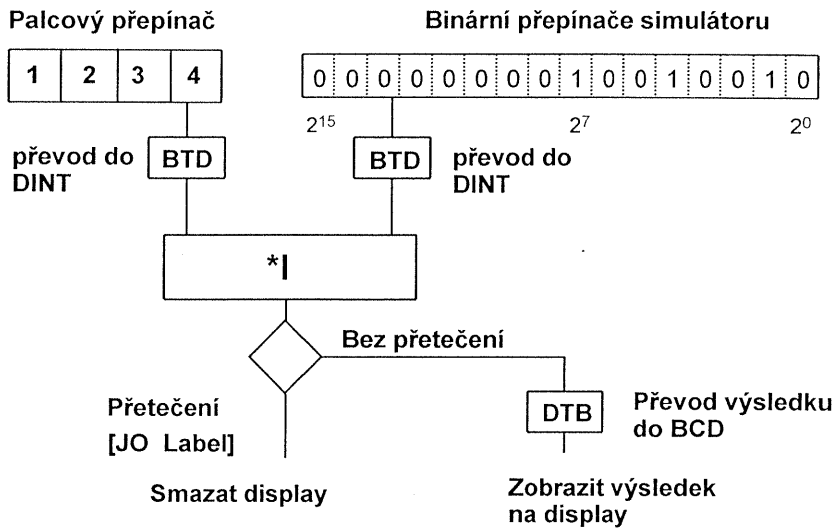
   Výsledek < 0: Vymazat display na simulátoru, tj. zapsat na display hodnotu 0.

   Výsledek >= 0: Zobrazit na display hodnotu z palcového přepínače.

   Poznámka: Pro vyhodnocení výsledku lze použít instrukci *JM [Label]* .

   Pro vymaskování chyb vyplývajících z převodu číselných formátů lze naprogramovat blok OB121 s instrukcí *NOP 0*.
5. Vyvolat FC11 v OB1 a nahrát bloky (OB1, OB121 a FC11) do CPU.
6. Otestovat funkčnost programu.

# Cvičení 1.2: Skok po násobení




**Palcový přepínač:** **S7-300: IW4 (IW2, 32-Bit. verze.)** **S7-400: IW30**
**Binární přepínače:** **S7-300: IW0 (IW0)** **S7-400: IW28**
**Display:** **S7-300: QW12 (QW6, 32-Bit. verze)** **S7-400: QW38**





**Cíl cvičení**

Naprogramovat instrukci skoku, která bude provedena v závislosti na výsledku násobení.

**Definice zadání**

Vytvořit FC 12 s následující funkcí:

1. nahrát do akumulátorů hodnoty z palcového přepínače a binárních přepínačů .
2. Převést BCD hodnoty do DINT s použitím instrukce BTD (BCD_TO_DINT).
3. Provést 16 bitové násobení.
4. Zkontrolovat výsledek operace na přetečení a provést následující akce :

Při přetečení: Smazat display.

Bez přetečení: Převést výsledek do BCD kódu a zobrazit na display.

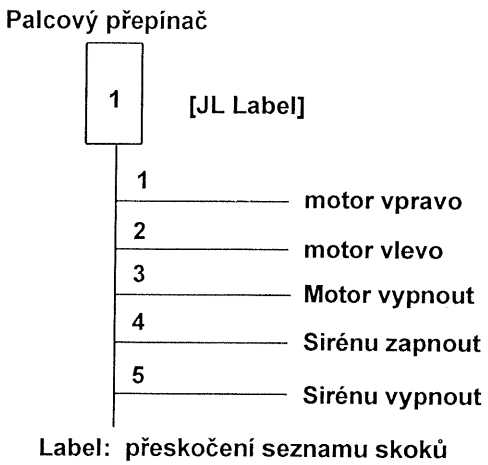
Poznámka: K vyhodnocení přetečení slouží instrukce *JO [Label]*.

Pro vymaskování chyb, které pramení z převodu mezi číselnými formáty, slouží blok *OB121* s instrukcí *NOP 0*.

5. Vyvolat FC12 z OB1 a nahrát program do CPU.
6. Otestovat funkčnost programu.

# Cvičení 1.3: Programování distributoru

**Funkce:**




| Adresy: | S7-300 (16-Bit) | S7-300 (32-Bit) | S7-400 |
|---|---|---|---|
| Motor_vpravo: | Q20.5 | Q8.5 | Q40.5 |
| Motor_vlevo: | Q20.6 | Q8.6 | Q40.6 |
| Siréna: | Q20.7 | Q8.7 | Q40.7 |







**Cíl cvičení**

Důvěrně se seznámit s instrukcí *JL*.

**Definice zadání**

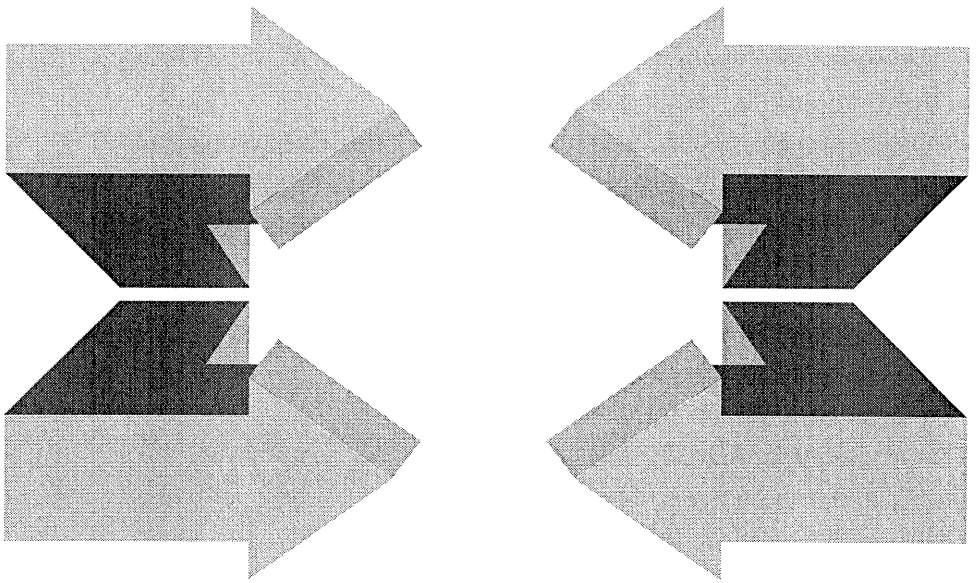
Snahou je vytvořit FC 13 s následující funkcí:

- Přes vstupní parametr SELECT zadat *INT* číslo v rozsahu od 1 do 5.
- Podle zadané hodnoty provést následující akce :
  - 1: spustit dopravník ve směru konečné montáže
  - 2: spustit dopravník v opačném směru.
  - 3: zastavit dopravník.
  - 4: spustit sirénu.
  - 5: vypnout sirénu.
- všechna ostatní čísla interpretovat jako chybu, tj. nastavit výstupní parametr *ENO* na hodnotu *FALSE*.

**Provedení**

1. Vytvořit FC13 s popsanou funkcí.
2. Vyvolat FC13 v OB1 podle hodnoty bitu I 28.0.
   Hodnota vstupního parametru SELECT je převzata s náběžnou hranou I 28.0.
4. V případě chybného zadání (SELECT > 5 nebo < 1, je Q36.0 v OB1 nastaven pomocí parametru *ENO* do *1*.
5. Nahrát program do CPU a otestovat jeho funkci.

# Instrukce pro práci s akumulátory



SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_02cz.1

Školicí středisko
firmy E&A spol. s r.o.

**Obsah** **Strana**

## Přehled instrukcí pro práci s akumulátory

- **Instrukce ovlivňující více akumulátorů**
  - TAK: záměna obsahu *ACCU1* a *ACCU2*
  - PUSH: kopírování *ACCU1* do dalších *ACCU*
  - POP: kopírování posledního *ACCU* do předchozích akumulátorů
  - ENT: kopírování *ACCU2* do dalších *ACCU,* bez *ACCU1*
  - LEAVE: kopírování posledního *ACCU* do předchozích akumulátorů, bez *ACCU2*
  - Aritmetické a slovní logické instrukce
- **Instrukce, ovlivňující pouze *ACCU1***
  - INC: zvýšení obsahu *ACCU 1-L-L*
  - DEC: snížení obsahu *ACCU 1-L-L*
  - CAW: záměna pořadí bytů v *ACCU1-L*
  - CAD: záměna pořadí bytů v *ACCU1*
  - INVI, INVD: tvorba prvního doplňku
  - NEGI, NEGD, NEGR: tvorba druhého doplňku (Negace)
  - RLDA, RRDA: Rotace obsahu *ACCU1* doleva nebo doprava s pomocí stavového bitu *CC1*

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_02cz.2

Školicí středisko
firmy E&A spol. s r.o.

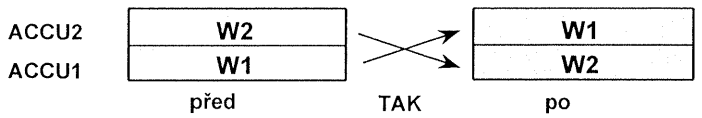
**Přehled**

Instrukce pro práci s akumulátory umožňují přesun obsahu akumulátorů mezi sebou nebo záměnu pořadí bytů v *ACCU1.* Zpracování čistě akumulátorových instrukcí je nezávislé na výsledku logické operace nebo na stavových bitech.

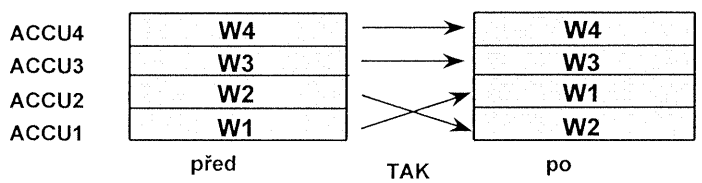
Stejně tak není zpracováním akumulátorové instrukce ovlivněn žádný stavový bit.

## Instrukce TAK (záměna ACCU1 a ACCU2)

**S7-300:**




**S7-400:**






Datum: 24.11.2002
Soubor: PRO2_02cz.3



**TAK**

TAK zamění obsah *ACCU 1* obsahem *ACCU 2* a naopak. Instrukce je zpracována bez ohledu na hodnoty stavových bitů a jejich hodnoty neovlivňuje. Obsahy ostatních akumulátorů nejsou instrukcí TAK ovlivněny (u S7-400).

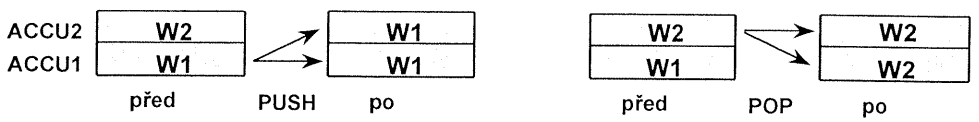
**Příklad**

Odečítání dvou hodnot:

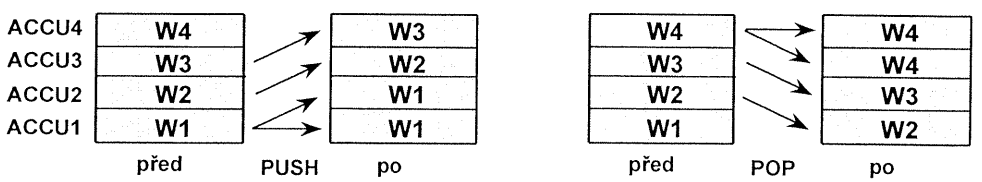
```
      L   MW10    // nahrát obsah MW10 do AKKU1-L.
      L   MW12    // nahrát obsah ACCU1-L do ACCU2-L a
                  // obsah MW12 do ACCU1-L.
      >I          // Je ACCU2-L (MW10) větší než ACCU1-L (MW12) ?
      JC NEXT     // Ano : skok na návěští NEXT.
      TAK         // záměna obsahů ACCU 1 a ACCU 2.
NEXT: -I          // ACCU2-L - ACCU1-L --> ACCU1-L
      T   MW14    // uložit výsledek (= větší hodnota - menší hodnota) do
                  MW14
```

## Instrukce PUSH a POP

**S7-300:**




**S7-400:**




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_02cz.4

Školicí středisko
firmy E&A spol. s r.o.

**PUSH**

Instrukce PUSH posune vždy obsah akumulátorů do vyššího *ACCU.* PUSH se obvykle používá k získání kopie obsahu *ACCU1* bez ztráty hodnot uložených v *ACCU2* a *ACCU3* (platí pouze pro S7-400).

- PUSH (S7-300): kopíruje obsah *ACCU1* do *ACCU 2. ACCU 1* zůstane bez změn.
- PUSH (S7-400): kopíruje obsah *ACCU3* do *ACCU4*, *ACCU2* do *ACCU3* a *ACCU1* do *ACCU2. ACCU1* zůstane bez změn.

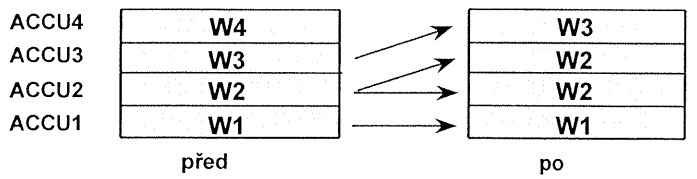
**POP**

Instrukce POP umožňuje přístup k hodnotám, které jsou uloženy v *ACCU2* až *ACCU4.* Obvykle se tato instrukce používá po instrukci T, v okamžiku, kdy obsah ACCU1 již není podstatný ale naopak je nutné pracovat s hodnotami, které jsou ve vyšších akumulátorech.

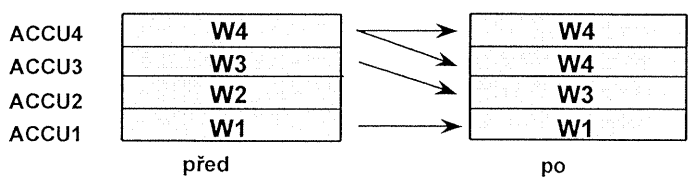
- POP (S7-300): kopíruje obsah *ACCU2* do *ACCU1. ACCU 2* zůstane nezměněn.
- POP (S7-400): kopíruje obsah *ACCU2* do *ACCU1*, *ACCU3* do *ACCU2* a *ACCU4* do *ACCU3. ACCU4* zůstane nezměněn.

## Instrukce ENT a LEAVE (pouze S7-400)

**ENT:**




**LEAVE:**




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_02cz.5

Školicí středisko
firmy E&A spol. s r.o.

**ENT**

Instrukce ENT posune obsah *ACCU2* a *ACCU3* do vyšších akumulátorů. Obsah *ACCU1* a *ACCU2* zůstane nezměněn.

ENT je obvykle používán ve spojení s instrukcí L:

```
ENT
L ...
```

Instrukce ENT je zpracována nepodmíněně a hodnoty stavových bitů neovlivňuje.

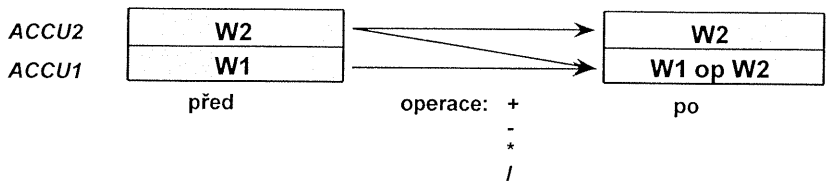
**LEAVE**

Instrukce LEAVE posune obsah *ACCU3* a *ACCU4* do nižších akumulátorů. Obsah *ACCU1* a *ACCU4* zůstane nezměněn.

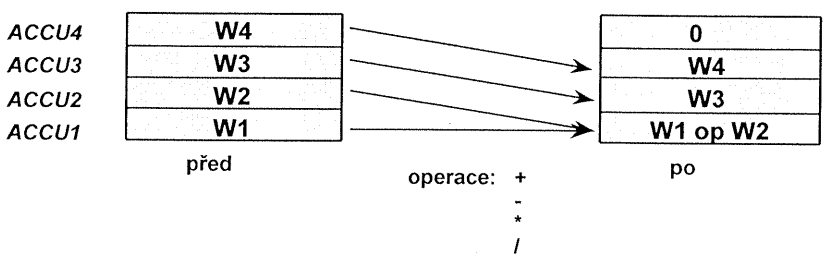
Instrukce LEAVE použitá po digitální logické operaci, přesune obsah akumulátorů 3 a 4 do akumulátorů 2 a 3. Výsledek digitální operace zůstane nezměněn v ACCU1.

## Aritmetické instrukce

**S7-300:**




**S7-400:**








**Aritmetické instrukce**

Aritmetické instrukce kombinují dvě číselné hodnoty obsažené v akumulátorech 1 a 2 podle základních operací aritmetiky. Výsledek výpočtu je po provedení operace obsažen v akumulátoru 1.

Stavové bity *CC0*, *CC1*, *OV* a *OS* poskytují informace o výsledku nebo mezivýsledku výpočtu.

**S7-300**

U S7-300 zůstává obsah *ACCU2* po aritmetické operaci nezměněn.

**S7-400**

U S7-400 je obsah *ACCU2* přepsán obsahem *ACCU3*. Obsah *ACCU4* je přenesen do *ACCU3*.

**Příklad**

Následující část programu dává různé výsledky podle toho, jestli je program spuštěn na S 7-300 nebo na S7-400:

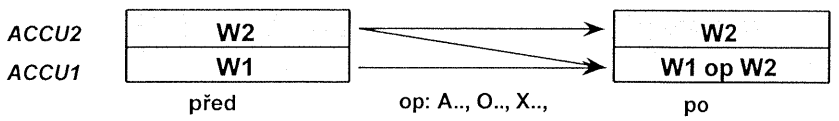
```
L     0     // nahrát 0 do ACCU1
L     5     // nahrát 5 do ACCU1, 0 do ACCU2
PUSH        // přesun 5 (ACCU1) do ACCU2; (S7-400: ACCU2 -> ACCU3)
*I          // násobení ACCU1 s ACCU2; (S7-400: ACCU3 -> ACCU2)
*I          // násobení ACCU1 s ACCU2; (S7-400: ACCU3 -> ACCU2)
```

Výsledek:

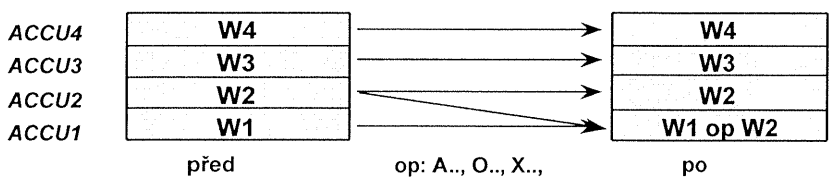
S7-300: ACCU1 = 125
S7-400: ACCU1 = 0

## Slovní logické instrukce

S7-300:




S7-400:








**Slovní logické instrukce**

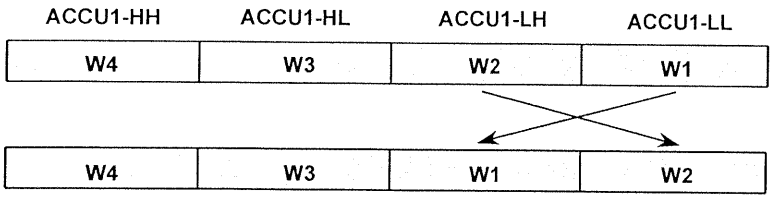
Slovní logické instrukce kombinují hodnoty v *ACCU1* a v *ACCU2* bit po bitu a výsledek logické operace ukládají do *ACCU1*.

Obsah ostatních akumulátorů (*ACCU2* u S7-300, nebo *ACCU2*, *ACCU3* a *ACCU4* u S7-400) zůstává po operaci nezměněn. Logické operace mohou být provedeny na šířce slova nebo dvojitého slova.

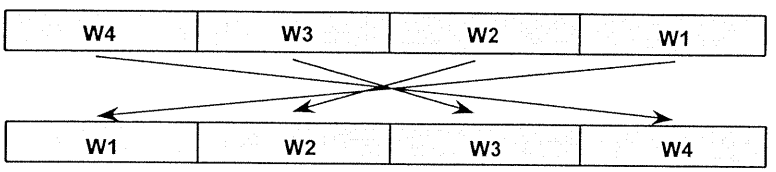
Jako slovní jsou přístupné logické operace AND, OR a Exclusive OR.

## Instrukce záměny v ACCU1

CAW:




CAD:




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_02cz.8

Školicí středisko
firmy E&A spol. s r.o.

**CAW** Instrukcí CAW lze zaměnit obsah jednotlivých bytů v *ACCU1. ACCU1-LH* je přenesen do *ACCU1-LL* a opačně..

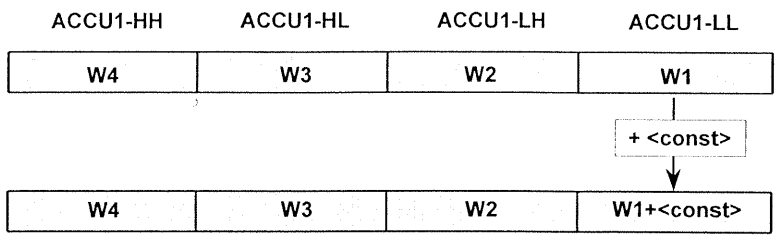
Tato instrukce umožňuje konverzi 16-bitových čísel (INT a WORD) mezi reprezentací SIMATIC a INTEL pro přenos dat mezi mezi SIMATICem S7 a PC.

**CAD** Instrukcí CAD lze zaměnit obsah jednotlivých bytů v celém akumulátoru, tzn. obsah ACCU1-HH je přenesen do ACCU1-LL a naopak a obsah ACCU1-HL je přenesen do ACCU1-LH (a naopak).

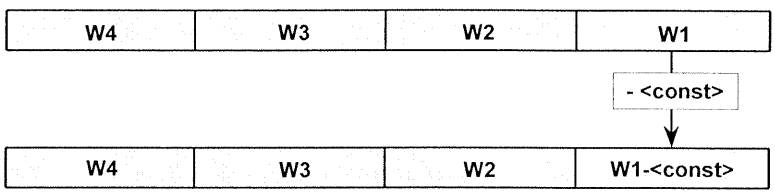
Instrukce CAD umožňuje konverzi 32 bitových čísel (DINT, DWORD a REAL) mezi reprezentací SIMATIC a INTEL pro přenos dat mezi SIMATICem a PC.

## Instrukce inkrementace v ACCU1

**INC <const>:**




**DEC <const>:**




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_02cz.9

Školicí středisko
firmy E&A spol. s r.o.

**INC** Instrukce INC <8-bit integer> přičte 8-mi bitové celé číslo k obsahu *ACCU1-LL* a uloží výsledek do *ACCU1-LL*.

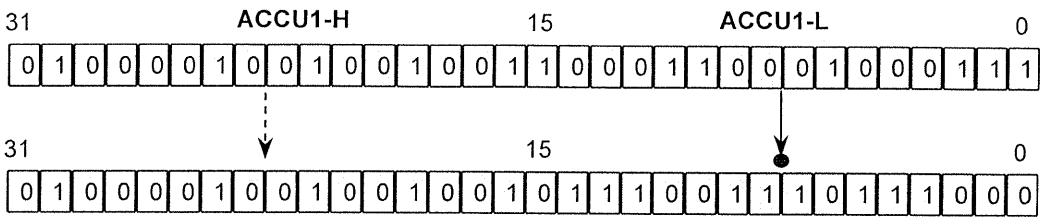
*ACCU1-LH, ACCU1-H* a *ACCU2*, nebo *ACCU3* a *ACCU4* zůstanou po zpracování instrukce bez změny.

**DEC** Instrukce DEC <8-bit integer> odečte 8-mi bitové celé číslo od *ACCU1-LL* a uloží výsledek do *ACCU1-LL*.

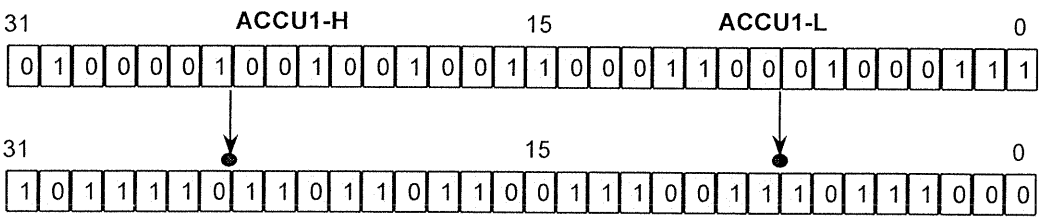
*ACCU1-LH*, *ACCU1-H* a *ACCU2*, nebo *ACCU3* a *ACCU4* zůstává bez změny.

## Tvorba prvního doplňku

**INVI (První doplněk ACCU1-L):**




**INVD (První doplněk ACCU1):**








**INVI**

Instrukce INVI neguje jednotlivé bity (0 až 15) obsažené v pravém slově *ACCU1*. Hodnotu 1 zamění za hodnotu 0 a naopak. Obsah levého slova (bity 16 až 31) zůstávají bez změny.

Instrukce INVI neovlivňuje žádný stavový bit.

**INVD**

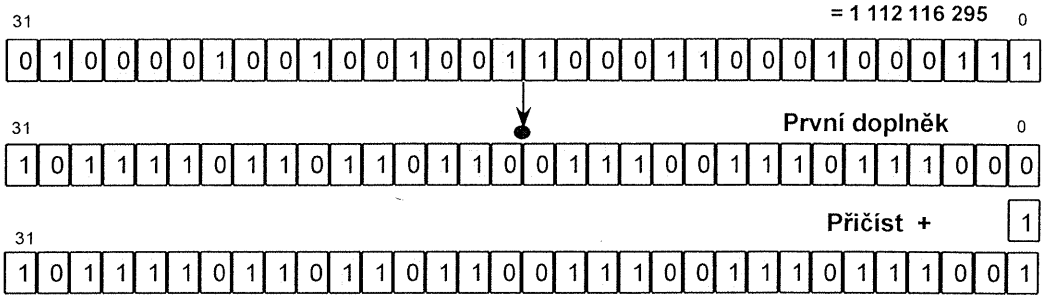
Instrukce INVD neguje jednotlivé bity ( 0 až 31) *ACCU1*. Hodnotu 1 zamění za hodnotu 0 a naopak.

Instrukce INVD neovlivňuje žádný stavový bit.

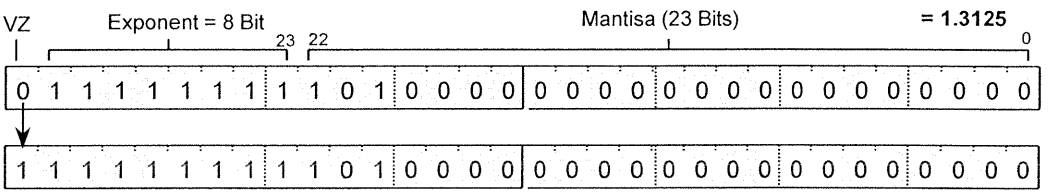
## Negace čísel (druhý doplněk)

**NEGI (Negace INT čísel)**

**NEGD (Negace DINT čísel):**




**NEGR (Negace REAL čísel):**




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_02cz.11

Školicí středisko
firmy E&A spol. s r.o.

**NEGI** Instrukce NEGI interpretuje obsah pravého slova (bity 0 až 15) *ACCU1* jako INT číslo a vytvoří z něho druhý doplněk s opačným znaménkem.

Tato instrukce je ekvivalentní násobení číslem -1. Levé slovo *ACCU1* (bity 16 až 31) zůstávají nezměněny.

Stavové bity *CC1*, *CC0*, *OS* a *OV* jsou nastaveny podle výsledku operace.

**NEGD** Instrukce NEGD interpretuje ACCU1 jako REAL číslo a násobí toto číslo číslem -1.

Druhý doplněk lze také vytvořit z prvního doplňku přičtením čísla -1.

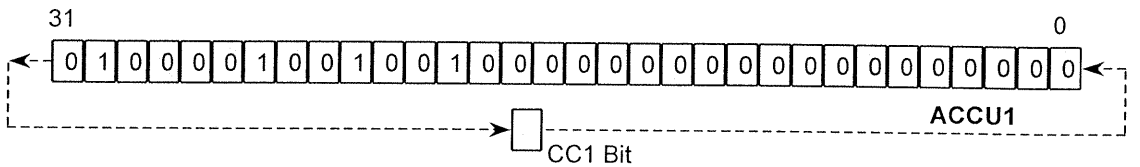
Stavové bity *CC1*, *CC0*, *OS* a *OV* jsou nastaveny podle výsledku operace.

**NEGR** Instrukce NEGR interpretuje *ACCU1* jako REAL číslo (32 bitové, IEEE-FP) a násobí toto číslo číslem'-1'. Instrukce invertuje bit 31 v *ACCU1* (znaménko mantisy).

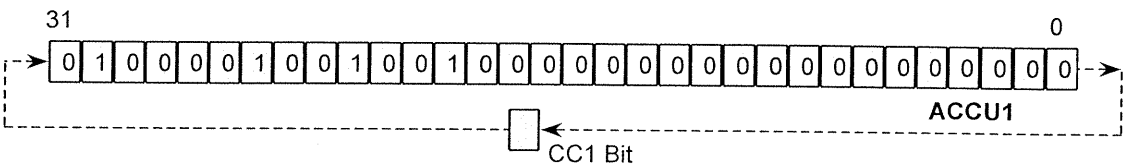
Instrukce NEGR neovlivňuje stavové bity.

## Instrukce pro 32-bitovou rotaci s bitem CC1

**RLDA (Rotace vlevo přes stavový bit CC1):**




**RRDA (Rotace vpravo přes stavový bit CC1):**




SIMATIC S7 OUT Datum: 24.11.2002 Soubor: PRO2_02cz.12



Školicí středisko firmy E&A spol. s r.o.

**RLDA**

Instrukce RLDA posune obsah *ACCU1* o 1 bit vlevo. Bit 0 je nastaven podle hodnoty stavového bitu *CC1*. Bit *CC1* obsahuje logickou hodnotu z "vyřazeného" bitu 31.

Stavové bity *CC0* a *OV* jsou nastaveny na *0*.

- Příklad:

  *ACCU1*: 0100 0100 1100 0100
  *CC1*: 1

  RLDA

  *ACCU1*: 1000 1001 1000 1001
  *CC1*: 0

**RRDA**

Instrukce RRDA posune obsah *ACCU1* o 1 bit vpravo. Bit 31 je nastaven podle hodnoty stavového bitu *CC1*. Bit *CC1* je nastaven na hodnotu, která "vypadla" z bitu 0.

Stavové bity *CC0* a *OV* jsou nastaveny na 0.

- Příklad:

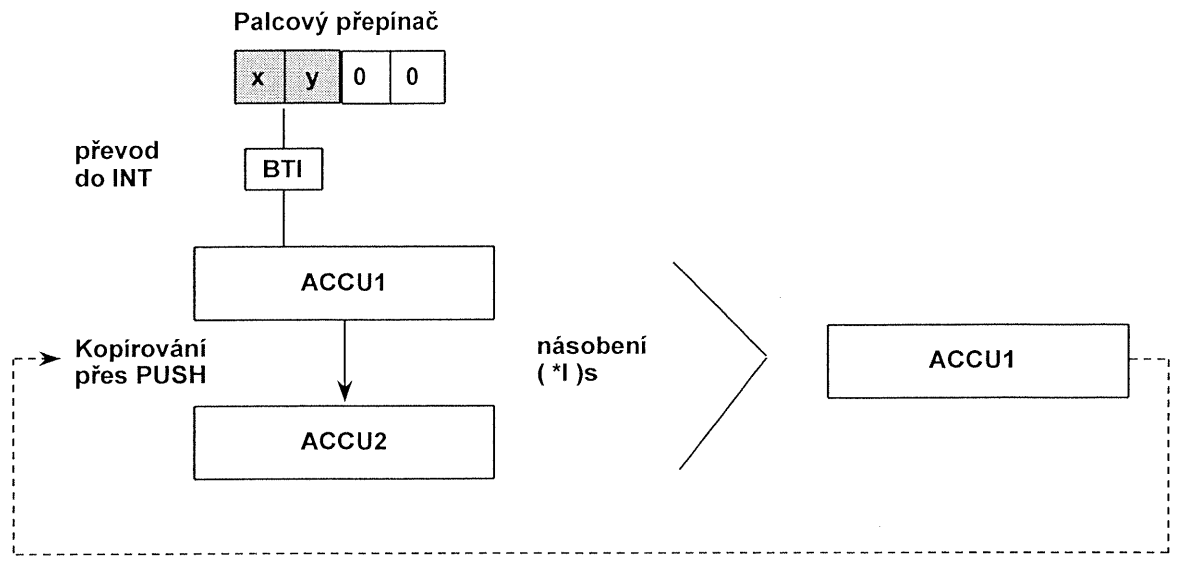
  *ACCU1*: 0100 0100 1100 0100
  *CC1*: 1

  RRDA

  *ACCU1*: 1010 0010 0110 0010
  *CC1*: 0

# Cvičení 2.1: Výpočet mocniny

**Cvičení : Výpočet 6-té mocniny celého čísla s pomocí PUSH a *I**




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_02cz.13

Školicí středisko firmy E&A spol. s r.o.

**Cíl cvičení** Důvěrně se seznámit s instrukcemi pro práci akumulátory pomocí výpočtu 6-té mocniny celého čísla.

**Definice úkolu** Vytvořit blok FC21 s následující funkcí:

- načte hodnotu levého bytu palcového přepínače a provede převod z BCD kódu do INT s použitím funkce BTI.
- vypočte 6. mocninu z načtené hodnoty.

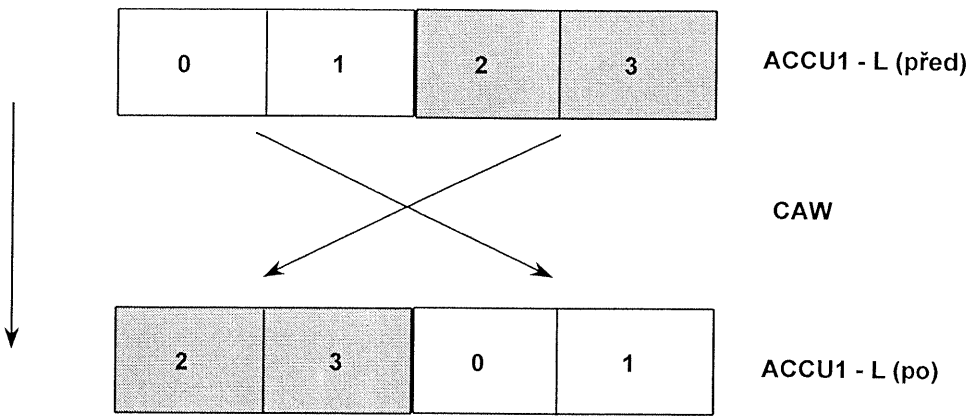
**Provedení**

1. Zkopírovat obsah *ACCU1* do *ACCU2* s pomocí instrukce PUSH.
2. Vynásobit *ACCU1* s *ACCU2*.
3. Zkopírovat obsah *ACCU1* do *ACCU2* s pomocí instrukce PUSH.
4. atd., atd., atd.

   Otázka: Kolikrát se musí kroky 1-3 opakovat, aby bylo dosaženo na S7-300 správného výsledku stejně tak jako na S7-400?
5. Zobrazit výsledek na display.
6. Vyvolat FC21 z OB1 a nahrát program do CPU.
7. Otestovat program.

**Poznámka** Aby výsledná hodnota nebyla příliš velká, může být použita pouze levá dekáda přepínače.

# Cvičení 2.2: Záměna hodnot v ACCU1




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_02cz.14

Školicí středisko
firmy E&A spol. s r.o.

**Cíl cvičení**

Důvěrně se seznámit s instrukcemi pro záměnu bytů v *ACCU1*.

Použití: Převod čísel z kódování používaného řídícím systémem SIMATIC do kódování používaného procesory INTEL (80486, Pentium,...).

Tento převod se **musí** provádět při jakékoliv výměně číselných údajů mezi SIMATICem a PC.

**Definice úkolu**

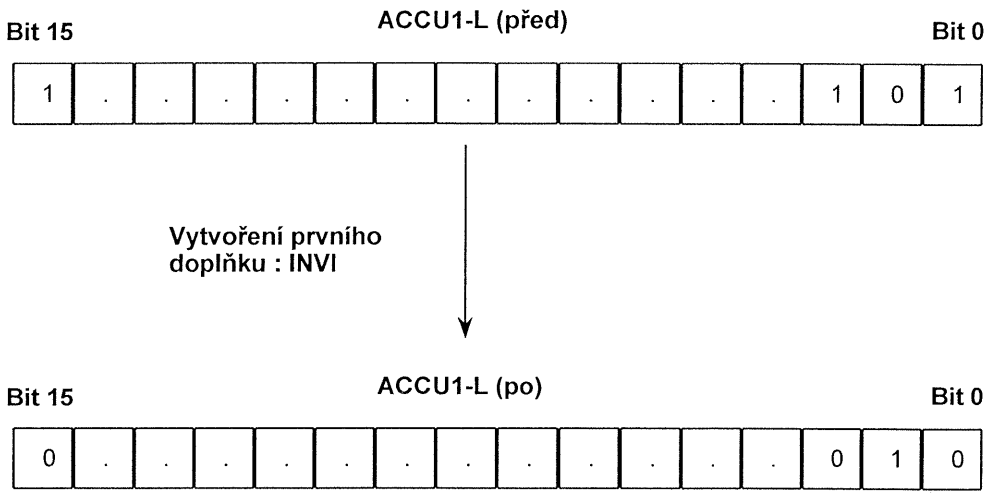
Vytvořit blok FC22 s následující funkcí:

- nahraje hodnotu z palcového přepínače do *ACCU1*.
- v *ACCU1 - L* zamění 2 byty s pomocí příkazu CAW.
- výsledný obsah *ACCU1* zobrazí na display.

**Provedení**

1. vytvořit blok FC22.
2. vyvolat tento blok (FC22) z OB1.
3. nahrát program do CPU.
4. otestovat program.

# Cvičení 2.3: Tvorba doplňků




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_02cz.15

Školicí středisko
firmy E&A spol. s r.o.

**Cíl cvičení**

Důvěrně se seznámit s instrukcí pro tvorbu doplňku.

Využití: Převod mezi zápornou a kladnou logikou.

Tento převod je potřebný vždy, když jsou načítány signály se zápornou logikou a uživatelský program používá kladnou logiku, nebo když je potřeba poslat na výstupní karty signály se zápornou logikou.

**Definice úkolu**

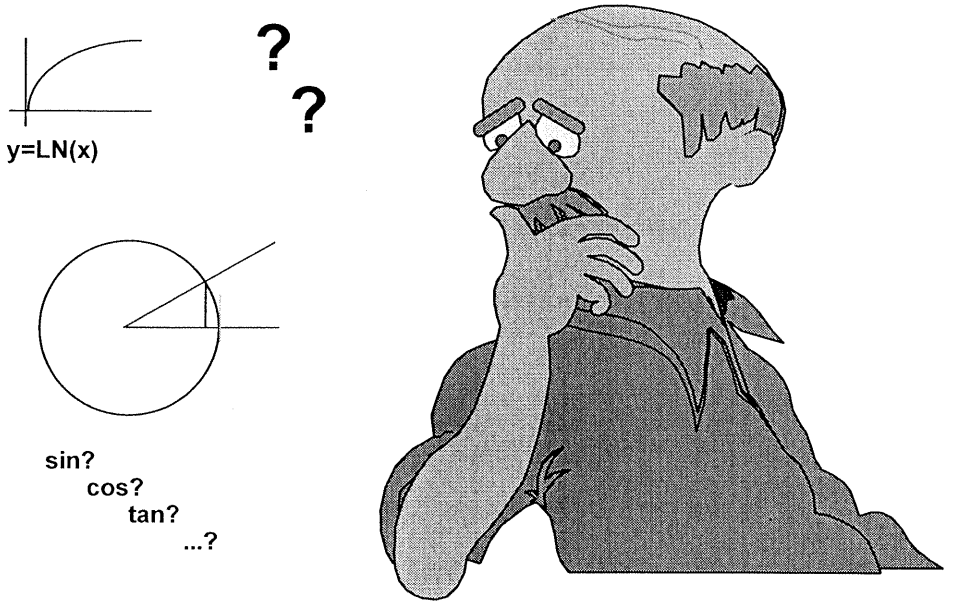
Vytvořit blok FC23 s následující funkcí:

- nahraje hodnotu z přepínačů simulátoru do *ACCU1*.
- vytvoří první doplněk.
- výslednou hodnotu zobrazí na LED diodách simulátoru.

**Postup**

1. vytvořit blok FC23
2. vyvolat blok FC23 z OB1
3. nahrát program do CPU.
4. otestovat funkčnost programu s využitím funkce *Variable Status.*

# Instrukce pro práci s REAL čísly




SIMATIC S7



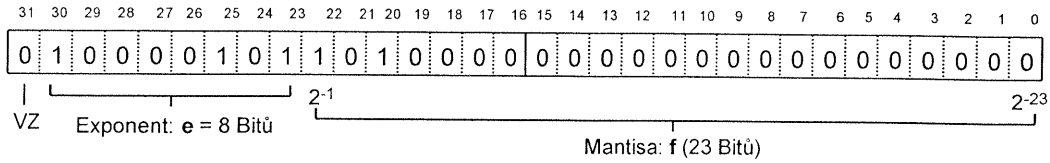
Datum: 24.11.2002
Soubor: PRO2_03cz.1

Školicí středisko
firmy E&A spol. s r.o.

**Obsah** Strana

# Reprezentace REAL čísel v SIMATICu S7

## ❑ Reprezentace formátu REAL čísla (IEEE FP 32-bit):




## ❑ Reprezentace normovaného REAL čísla:

**VZ x (1.f) x 2** $^{(e-127)}$

VZ = znaménkový bit, (0 odpovídá +, 1 odpovídá -)
f = 23 bit Mantisa s MSB = $2^{-1}$ a LSB = $2^{-23}$
e = mocnina čísla 2 ($0 < e < 255$)

## ❑ Příklad:

VZ = 0
e = 1000 0101 = 133
f = 1010 0000... = 0.5 + 0.125

⇨ R = +1.625 x 2(133-127) = 1.625 x 64 = **104.0**

## ❑ Rozsah hodnot normovaných REAL čísel:

- 3,402 823 x $10^{+38}$ ... -1,175 494 x $10^{-38}$, 0, 1,175 494 x $10^{-38}$ ... 3,402 823 x $10^{+38}$




Datum: 24.11.2002
Soubor: PRO2_03cz.2



**REAL číslo**

REAL (desetinná) čísla umožňují implementaci komplexních matematických výpočtů pro řízení procesů a pro uzavřené řídící smyčky.

Datový typ REAL obsahuje interně 3 části: znaménko, 8-bitový exponent základu 2 a 23-bitovou mantisu.

***Znaménkový bit s hodnotou 0 označuje číslo kladné, hodnota 1 označuje číslo záporné.***

Mantisa reprezentuje desetinnou část. Celá část mantisy nemusí být uložena, protože pro normované číslo je vždy 1 a pro nenormované číslo je 0.

**Rozsah**

| Označení | Exponent e | Mantisa f | Hodnota | CC1 | CC0 | OV | OS |
|---|---|---|---|---|---|---|---|
| Nedesetinné č | 255 | <>0 | [qNaN] | 1 | 1 | 1 | 1 |
| Přetečení | 255 | 0 | >$(2-2^{-23})\ 2^{127}$<br><$(-2+2^{-23})\ 2^{127}$ | 1<br>0 | 0<br>1 | 1<br>1 | 1<br>1 |
| Normované číslo | 1.. 254 | | $(1.f)\ 2^{e-127}$<br>$(-1.f)\ 2^{e-127}$ | 1<br>0 | 0<br>1 | 0<br>0 | -<br>- |
| Nenormované číslo | 0 | <>0 | $(0.f)\ 2^{-126}$<br>$(-0.f)\ 2^{-126}$ | 0<br>0 | 0<br>0 | 1<br>1 | 1<br>1 |
| Nula | 0 | 0 | +0 | 0 | 0 | 0 | - |

**Poznámka**

CPU počítají s desetinnými čísly v plném rozsahu přesnosti. Hodnota zobrazená na PG může vykazovat odchylku v důsledku chyby při zaokrouhlování.
REAL čísla jsou zaokrouhlována nahoru na 6 desetinných míst.

# Základní instrukce pro práci s REAL čísly

## ❑ Sčítání

```
L     MD10
L     MD20
+R              // MD10 + MD20
T     MD30
```

## ❑ Odečítání

```
L     MD10
L     MD20
-R              // MD10 - MD20
T     MD30
```

## ❑ Násobení

```
L     MD10
L     MD20
*R              // MD10 * MD20
T     MD30
```

## ❑ Dělení

```
L     MD10
L     MD20
/R              // MD10 / MD20
T     MD30
```






**Přehled**

Instrukce +R, -R, *R, /R interpretují hodnoty uložené v*ACCU1* a *ACCU2* jako REAL čísla. Po provedení instrukce (+R, -R, *R a /R) je výsledek uložen v *ACCU1*.

Stavové bity *CC0* a *CC1* indikují zda je výsledek záporný (CC1=0, CC0=1), nula (CC1=0; CC0=0) nebo kladný (CC1=1, CC0=0).

Stavové bity *OV* a *OS* indikují opuštění povoleného rozsahu hodnot.

**Neplatná REAL čísla**

Po neplatném výpočtu, tj. je-li jedna ze dvou hodnot interpretována jako chybné REAL číslo, je výsledkem uloženým v *ACCU1* také neplatné REAL číslo. Neplatné REAL číslo je v *ACCU1* uloženo jako výsledek zpracování neplatných hodnot instrukcemi:

Sčítání: sčítání s + nekonečnem a - nekonečnem.

Odečítání: odečítání + nekonečna a - nekonečna

Násobení: násobení nuly nekonečnem

Dělení: dělení nuly nulou, dělení nekonečnem nebo nulou.

Výsledek dělení platného REAL čísla nulou je + nekonečno nebo - nekonečno (závisí na znaménku čísla).

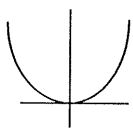
**Poznámka**

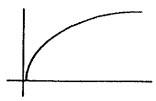
Hexadecimální číslo D#16#FFFF FFFF představuje například neplatné REAL číslo.

## Rozšiřující matematické instrukce

❑ **Matematické funkce:**

| | |
|---|---|
| SQR | 2. mocnina |
| SQRT | 2. odmocnina |
| EXP | $e^X$ |
| LN | Přirozený logaritmus $\log_e x$ (e=2,718282) |

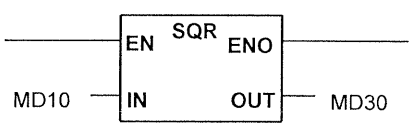




❑ **Příklad:**

```
L     MD10
SQR               // výpočet čtverce
T     MD30
```

**(STL)**




**(LAD)**







**Přehled**

Uvedené matematické funkce mění pouze obsah *ACCU1*. Obsah *ACCU2*, nebo *ACCU3* a *AKKU4* u S7-400zůstává nezměněn.

Podle hodnoty výsledku operace jsou nastaveny stavové bity *CC0*, *CC1*, *OV* a *OS*.

Je-li operace provedena s neplatným REAL číslem v *ACCU1*, je výsledkem také neplatné REAL číslo a stavové bity jsou nastaveny odpovídajícím způsobem.

**SQR**

Instrukce SQR vypočte 2.mocninu obsahu *ACCU1*.

**SQRT**

Instrukce SQRT vypočte 2.odmocninu z hodnoty uložené v *ACCU1*. Je-li číslo v *ACCU1* záporné, SQRT nastaví stavové bity *CC0*, *CC1*, *OV* a *OS* do 1 a uloží do *ACCU1* neplatné REAL číslo.

Je-li v *ACCU1* hodnota -0 , je vrácena hodnota -0.

**EXP**

Instrukce EXP vypočte mocninu *e* (=2.71828) a výsledek ($e^{ACCU1}$) uloží do ACCU1.

**LN**

Instrukce LN vypočte přirozený logaritmus hodnoty, uložené v *ACCU1*, ze základu *e*. Je-li v *ACCU1* číslo záporné nebo nula, instrukce nastaví stavové bity *CC0*, *CC1*, *OV* a *OS* do 1 a v *ACCU1* vrátí neplatné REAL číslo.

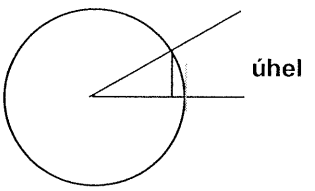
Přirozený logaritmus je inverzní funkcí k exponenciální funkci $e^x$.

je -li : $y = e^x$
potom: $x = \ln y$

## Trigonometrické funkce a funkce inverzní

❑ **Trigonometrické funkce:**

| | |
|---|---|
| SIN | Sinus |
| COS | Cosinus |
| TAN | Tangens |



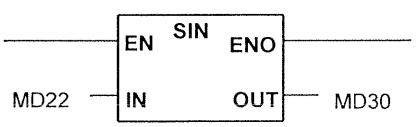

❑ **Arc funkce:**

| | |
|---|---|
| ASIN | Arc sinus |
| ACOS | Arc cosinus |
| ATAN | Arc tangens |

❑ **Příklad:**

```
L     MD10
SIN                 // výpočet sinus
T     MD30
```

**(STL)**




**(LAD)**








**Trigonometrické funkce**

Trigonometrické funkce očekávají v *ACCU1* úhel zadaný v radiánech jako REAL číslo. Uživatel je proto nucen před výpočtem trigonometrické funkce provést, je-li to nezbytné, převod z $0^0$ ... $360^0$ na 0 ... 2 pi ( pi=3,141593).

Pokud vstupní hodnota není v rozsahu 0...2pi, je automaticky přičten nebo odečten násobek 2pi tak, aby hodnota úhlu byla v rozsahu 0...2pi.

**Arc funkce**

Arc funkce jsou inverzní k funkcím trigonometrickým. Očekávají v *ACCU1* REAL číslo v definovaném rozsahu a vrací úhel v radiánech. :

| Funkce | povolený rozsah | výsledný rozsah |
|---|---|---|
| ASIN | -1 až +1 | - pi/2 až + pi/2 |
| ACOS | -1 až +1 | 0 až pi |
| ATAN | celý rozsah | - pi/2 až + pi/2 |

Při překročení povoleného rozsahu vrací arc funkce neplatné REAL číslo a stavové bity *CC0*, *CC1*, *OV* a *OS* jsou nastaveny do 1.

## Ostatní instrukce pro práci s REAL čísly

❑ **Převod z REAL do DINT:**

| | |
|---|---|
| RND+ | zaokrouhlení na nejbližší vyšší DINT číslo |
| RND- | zaokrouhlení na nejbližší nižší DINT číslo |
| RND | zaokrouhlení na nejbližší vyšší INT číslo |
| TRUNC | celá část |

❑ **Převod z DINT do REAL:**

| | |
|---|---|
| DTR | převod se zaokrouhlením |

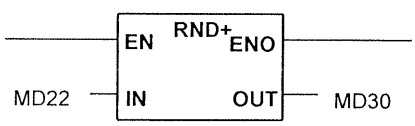
❑ **Ostatní instrukce pro práci s REAL čísly:**

| | |
|---|---|
| ABS | vytvoření absolutní hodnoty |
| NEGR | negace REAL čísla |

❑ **Příklad:**

```
L     MD10
RND+                  // zaokrouhlení na nejbližší vyšší DINT číslo
T     MD30
```

**(STL)**




**(LAD)**



Datum: 24.11.2002
Soubor: PRO2_03cz.6

Školicí středisko
firmy E&A spol. s r.o.

**Přehled** Konverzní instrukce převádí hodnotu, která je obsažena v *ACCU1* na jiný datový typ a výsledek uloží opět do *ACCU1*. Obsah ostatních akumulátorů zůstane nezměněný.

Jestliže při zpracování instrukcí RND+, RND-, RND nebo TRUNC není hodnota v *ACCU1* v povoleném rozsahu DINT čísla, nebo výchozí hodnota není v požadovaném REAL formátu, instrukce nastaví stavové bity OV a OS do *1* a převod se neprovede.

**RND+** převede REAL číslo v *ACCU1* na číslo DINT se stejnou nebo nejbližší vyšší hodnotou.

**RND-** převede REAL číslo v *ACCU1* na číslo DINT se stejnou nebo nejbližší nižší hodnotou.

**RND** převede REAL číslo v *ACCU1* na nejbližší DINT číslo. Je-li výsledek přesně mezi sudým a lichým číslem, instrukce vrátí liché číslo.

**TRUNC** vrací jako výsledek celočíselnou část převáděného čísla. Zlomková část čísla je zapomenuta.

**DTR** převede DINT číslo na číslo REAL. Protože DINT číslo má vyšší přesnost než číslo REAL, může během převodu dojít k zaokrouhlení na nejbližší číslo.

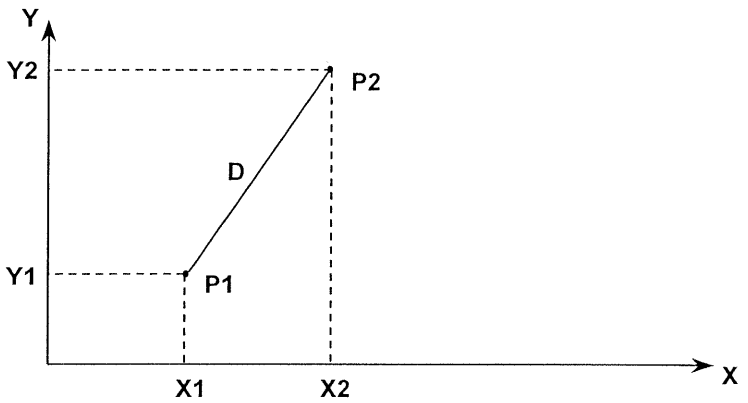
**ABS** vytvoří z REAL čísla obsaženého v *ACCU1* absolutní hodnotu nastavením znaménkového bitu (bit 31) na hodnotu 0.

**NEGR** neguje REAL číslo v *ACCU1* , tak, že invertuje znaménkový bit (bit 31).

Instrukce DTR, ABS a NEGR neovlivňují stavové bity.

# Cvičení 3.1: Výpočet vzdálenosti

**Cvičení : Výpočet vzdálenosti D mezi dvěma body v pravoúhlém souřadnicovém systému**




**Funkce: FC31** $D = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_03cz.7

Školicí středisko
firmy E&A spol. s r.o.

**Cíl cvičení** Použití matematických funkcí pro výpočet vzdálenosti mezi dvěma body v rovině.

**Definice úkolu** Vytvořit blok FC31 s následující funkcí:

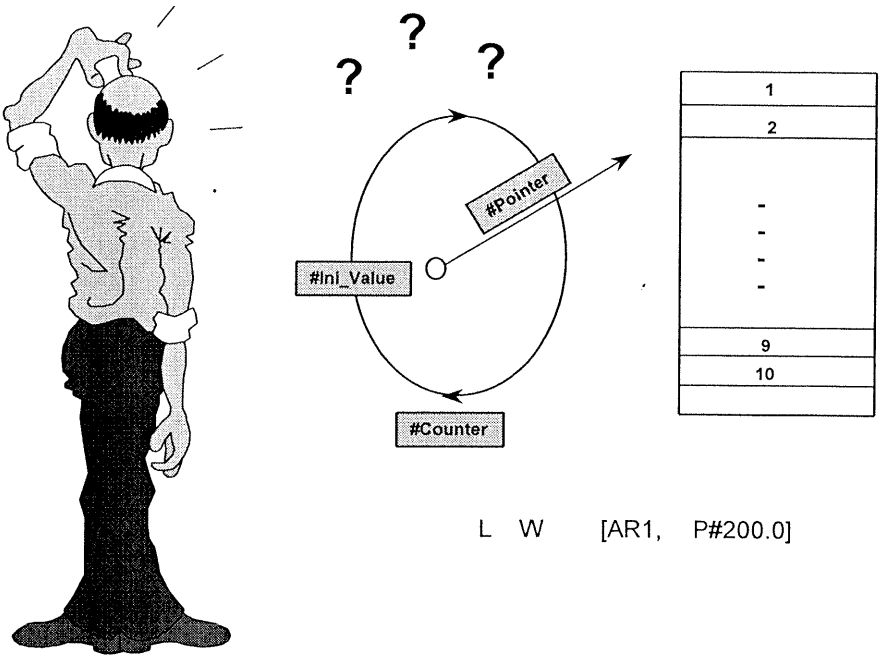
- parametrické zadání souřadnic (X1, Y1) a (X2, Y2) pro dva body P1 a P2.
- vrací vzdálenost mezi oběma body ve výstupním parametru RET_VAL.
- FC31 musí být použitelný jak v systému S7-300 stejně tak jako v systému S7-400.

**Postup**

1. vytvořit blok FC31 s výše uvedenou funkcí
2. vyvolat FC31 z OB1 a definovat následující vstupní a výstupní parametry :
   X1 = MD0, Y1 = MD4
   X2 = MD8, Y2 = MD12
   RET_VAL = MD16
3. nahrát program do CPU.
4. otestovat funkčnost pomocí funkce *Monitor/Modify Variable*.

**Doplňkový úkol** Vytvořit optimalizovanou verzi bloku FC31 pro S7-400.

# Nepřímá adresace a instrukce adresních registrů



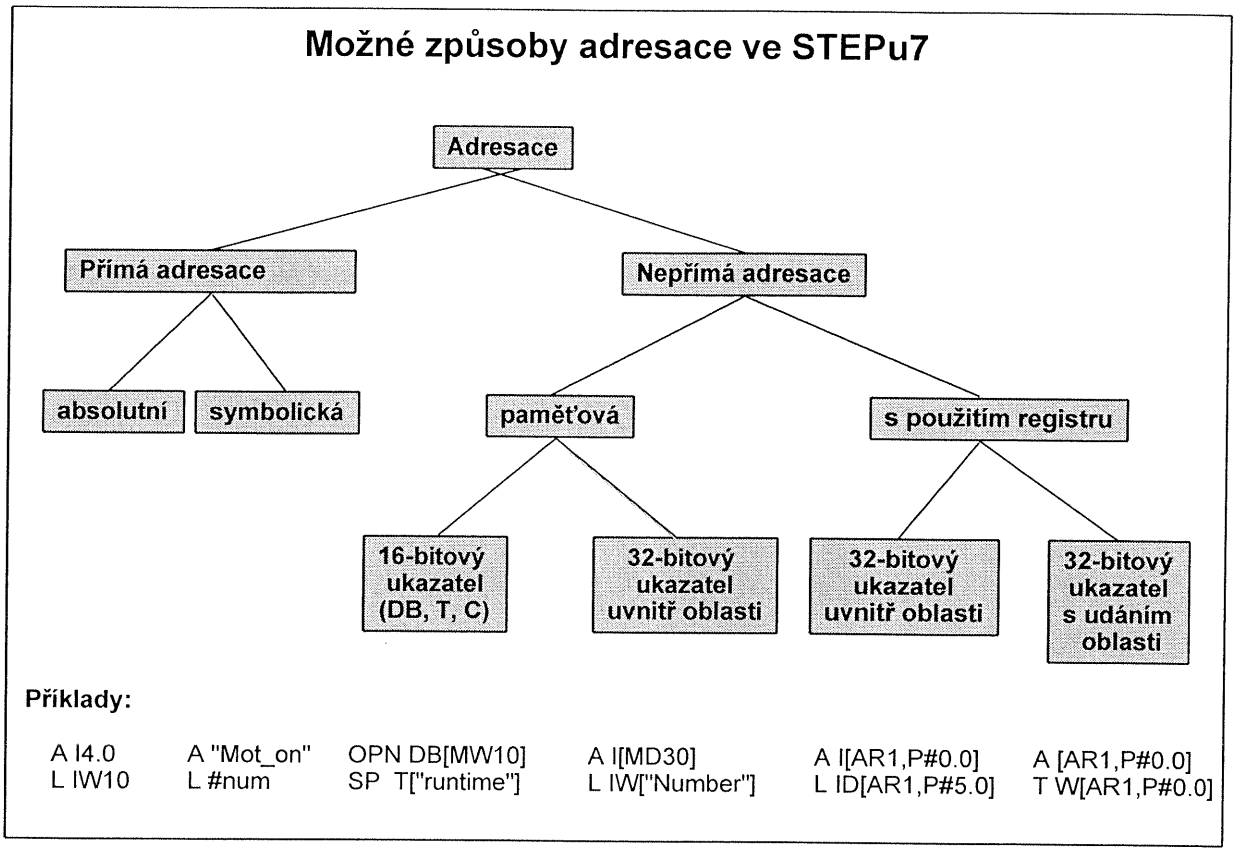




Datum: 24.11.2002
Soubor: PRO2_04cz.1



**Obsah** **Strana**




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_04cz.2

Školicí středisko
firmy E&A spol. s r.o.

**Přímá adresace**

Při použití přímé adresace je použitá paměťová oblast definována v instrukci, tzn. adresa zpracovávané proměnné je definována při tvorbě programu uživatelem.

**Symbolická adresace**

V uživatelském programu může být adresa proměnné definována absolutně (např I12.3) nebo symbolicky (např. "start signal"). Symbolická adresa použitého jména odpovídá přitom absolutní adrese operandu.

Program vytvořený s použitím symbolické adresace je snáze čitelný. Při symbolické adresaci jsou také odlišeny lokální symbolické názvy parametrů (definované v deklarační části bloku) a globální symbolické názvy (definované v tabulce symbolických názvů).

**Nepřímá adresace**

Při použití nepřímé adresace je adresa proměnné definována až během zpracování uživatelského programu. Nepřímá adresace umožňuje např. tvorbu programových smyček, s tím, že při každém průchodu smyčkou je zpracována proměnná s jinou adresou.

Při nepřímé adresaci je třeba rozlišovat mezi :

- paměťovou nepřímou adresací: adresa proměnné je uložena v uživatelské paměti (např. MD30).

  V tomto případě může mít proměnná, ve které je uložena adresa jiné proměnné, definován symbolický název.
- nepřímá adresace registrem: adresa proměnné je uložena v jednom ze dvou adresních registrů *AR1* nebo *AR2*.

**Varování**

**Protože při nepřímé adresaci jsou použité adresy vypočteny až při zpracování uživatelského programu, je zde nebezpečí, že paměť, ve které jsou adresy uloženy bude přepsána a to může mít za následek nepředvídatelné chování řízené technologie.**

# Přímá adresace proměnných

| Operand | Adresa operandu | Délka proměnné | Typ proměnné |
|---|---|---|---|
| I | 37.4 | Byte, word, double word | vstupy |
| Q | 27.7 | Byte, word, double word | výstupy |
| PIB | 655 | Byte, word, double word | periferní vstupy |
| PQB | 653 | Byte, word, double word | periferní výstupy |
| M | 55.0 | Byte, word, double word | vnitřní proměnné |
| T | 114 | -- | časovače |
| C | 13 | -- | čitače |
| DBX | 2001.6 | Byte (DBB), word (DBW), double word (DBD) | Data adresovaná přes DB registr |
| DIX | 406.1 | Byte (DIB), word (DIW), double word (DID) | Data adresovaná přes DI registr |
| L | 88.5 | Byte (LB), word (LW), double word (LD) | Lokální proměnné |





**Přímá adresace proměnných**

S použitím přímé adresace mohou být oslovovány jednoduché (základní) proměnné, tj. proměnné o délce maximálně 4 byty. Adresa základní proměnné obsahuje :

- identifikátor oblasti (např.: *IB* pro oblast vstupů)
- přesnou adresu (bytovou nebo i bitovou ) uvnitř definované oblasti, která jednoznačně určuje požadovanou proměnnou.

Adresy nebo proměnné mohou být také adresovány přes globální symbolické názvy definované v globální tabulce symbolických názvů.

**Periferie**

Při adresaci periferií je nyní nutné, na rozdíl od S5, odlišovat vstupní periferie a výstupní periferie. Je samozřejmostí, že periferní vstupy lze pouze číst (L PIW) a periferní výstupy pouze zapisovat (T PQW).

**Lokální data**

STEP 7 umožňuje také absolutní přístup do oblasti lokálních dat např.:

- A L 12.6 (testuje lokální datový bit s adresou 12.6 na logickou hodnotu 1)
- L LW 12 (nahraje obsah lokálního datového slova do *ACCU1*)

**DBX/DIX**

Uživatel má také přímý přístup k proměnným v datových blocích:

- A DBX 12.6 (testuje datový bit s adresou 12.6 v DB na logickou hodnotu 1. DB je nutno nejprve otevřít).
- L DB5.DBW10 (nahraje obsah DW10 v DB5 do *ACCU1*)

**Komplexní proměnné**

Přístup ke komplexním datovým proměnným (např. struktura nebo pole) je přes symbolické názvy.

Absolutní přístup je možný pouze ke složkám komplexních proměnných, a to pouze v případě, že tyto složky jsou tvořeny elementárními datovými typy.

## Přímá adresace proměnných v datových blocích

| Otevření DB bloku | Přenos dat v DB bloku | |
|---|---|---|
| OPN DB 19<br>OPN "Values" | L DBB 1 | nahrát datový byte 1 |
| | L DBW 2 | nahrát datové slovo 2 (byty 2/3) |
| | L 5 | nahrát číslo 5 |
| | T DBW 4 | uložit do slova 4 |
| OPN DI 20 | L 'A' | nahrát ASCII znak A |
| | L DIB28 | nahrát datový byte 28 |
| | ==I | porovnat |
| | A DBX 0.0 | testovat bit 0 z bytu 0 |
| Kombinace instrukcí (včetně OPN DB..) | L DB19.DBW4 | nahrát datové slovo 4 z DB 19 |
| | L "Values".Number_1 | Symbolický přístup k proměnné Number_1v DB19, který má symbolické jméno "Values" |
| | A DB10.DBX4.7 | testovat bit 7 z bytu 4 v DB 10 |

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_04cz.4

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

S7-CPU obsahuje 2 registry, které umožňují zpracování dat uložených v datových blocích. Čísla datových bloků, které jsou momentálně platné, jsou uloženy právě v těchto registrech a dovolují přistupovat k hodnotám v otevřených datových blocích. Před zpracováním hodnot z datového bloku musí být tento datový blok "otevřen" a jeho adresa obsažena v jednom ze dvou registrů otevřených datových bloků.

Otevření datového bloku lze provést explicitně pomocí instrukcí:

- **Opn DBx** nebo **OPN DIx**

nebo implicitně s použitím kombinované adresace DB operandu:

- **L DBx.DBWy** (L DIx.DIWy není platná instrukce!)

I v tomto případě je do DB registru uloženo číslo otevíraného datového bloku.

**Adresace**

Datové bloky jsou v S7 organizovány bytově ( na rozdíl od slovní organizace datových bloků v S5).

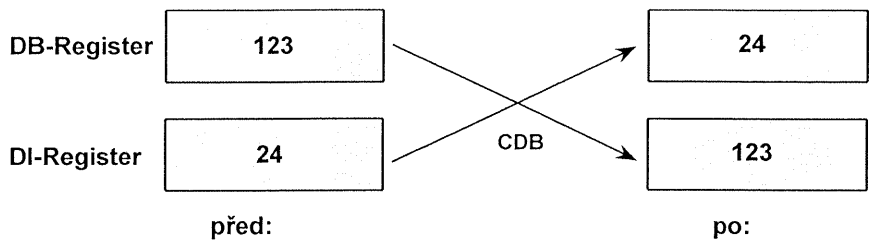
**Symbolický přístup**

K jednotlivým hodnotám v datovém bloku lze přistupovat také pomocí symbolických názvů jednotlivých proměnných. K tomu je nutné definovat v tabulce symbolických názvů název příslušného datového bloku. Symbolické názvy jednotlivých hodnot jsou definovány v tabulce rozhraní datového bloku v *DB Editoru*.

Po definici obou symbolických názvů lze použít např. instrukci: ***L "Values".Number_1***. Tato instrukce nahraje do *ACCU1* obsah symbolické proměnné *Number_1*, která je obsažena v datovém bloku se symbolickým názvem *Values*.

## Programové vyhodnocení vlastností DB bloků

- **Instrukce pro práci s DB registry:**
  - **CDB: záměna DB registrů**




  - **Nahrání DB registru do *ACCU1***
    - L DBNO (nahraje číslo otevřeného DB bloku do *ACCU1*)
    - L DINO (nahraje číslo otevřeného DI bloku do *ACCU1*)
  - **Nahrání délky datového bloku**
    - L DBLG (nahraje délku otevřeného DB bloku do *ACCU1*)
    - L DILG (nahraje délku otevřeného DI bloku do *ACCU1*)





**DB, DI registry**

Tyto registry obsahují čísla momentálně platných otevřených datových bloků. Současně mohou být otevřeny až 2 datové bloky současně.

STL používá první DB registr k definování přístupu k globálnímu - sdílenému datovému bloku DB a druhý DB registr k definování přístupu k instančnímu datovému bloku DI. Tyto registry jsou také nazývány jako DB a DI registr.

Z hlediska CPU jsou tyto registry rovnocenné, tj. každý datový blok může být otevřen ve kterémkoliv DB registru. Vyjímku tvoří současné otevření jediného DB bloku v obou registrech najednou.

**CDB**

CDB zamění obsah DB a DI registrů. Obsah DB registru je přenesen do DI registru a naopak. Tato instrukce neovlivňuje obsah *ACCU1* ani nastavení stavový bitů.

**L DBLG, L DILG:**

Tyto instrukce umožňují načtení délky ( počtu bytů ) datového bloku před tím než bude proveden přístup k požadovaným hodnotám. S jejich pomocí lze kontrolovat, zda datový blok vůbec může obsahovat požadované proměnné. Tato kontrola je žádoucí především v případě absolutní nepřímé adresace proměnných uložených v datových blocích.

**L DBNO, L DINO:**

S pomocí těchto instrukcí lze kontrolovat číslo právě platného otevřeného datového bloku.

# Paměťová nepřímá adresace

## ❑ 16-bitový slovní ukazatel (pro adresaci DB, T, C)

```
L 11
T MW 60
```

OPN DB[MW 60] ⟹ OPN DB 11

## ❑ 32-bitový dvouslovní ukazatel (pro adresaci I, Q, M, ...)

```
L  P#24.0
T  MD 50
```

L I W [MD50] ⟹ L IW 24

oblast šířka adresa

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_04cz.6

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

Při použití tohoto způsobu nepřímé adresace, je adresa proměnné uložena v uživatelské paměti.

Prováděný příkaz musí obsahovat následující údaje:

- instrukci (např.: OPN, A, L, atd.)
- identifikátor oblasti (typ proměnné = DB, C, T, I, QW, MD, atd.)
- a adresu proměnné = [proměnná].
  Tato proměnná obsahuje adresu ( ukazatel ) zpracovávané proměnné.

V závislosti na typu proměnné musí být ukazatel uložen buď v proměnné o šířce slovo nebo dvojité slovo.

**Instrukce pro 16-bitový ukazatel**

K adresaci časovačů, čitačů a bloků (DB, FC, FB) je používán 16-bitový ukazatel.

Všechny instrukce pro práci s čítači a časovači mohou být adresovány nepřímo. Datové bloky mohou být otevírány přes DB registr stejně tak jako přes DI registr.

Pokud je při nepřímém otevírání datových bloků (DB, DI) hodnota ukazatele nula, je provedena instrukce NOP 0.

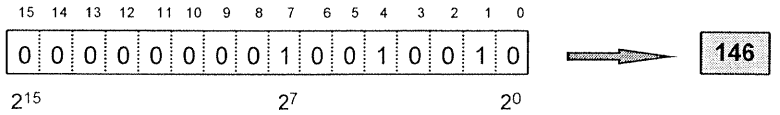
Programové bloky lze volat nepřímo pomocí instrukcí *UC* nebo *CC* (ne s pomocí instrukce *CALL*). Takto volané bloky nesmí obsahovat parametry nebo statické proměnné.

Při použití identifikátoru typu T, C, DB, DI, FB, FC je adresa operandu uložena v proměnné slovo.

Tento ukazatel je interpretován jako celé číslo v rozsahu 0 ... 65 535.

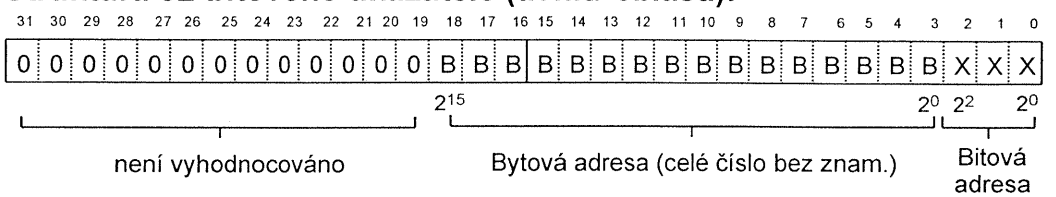
## Struktura ukazatele v paměťové nepřímé adresaci

### ❑ Struktura 16-bitového ukazatele:




Interpretována jako celé číslo bez znaménka v rozsahu 0 ... 65 535

### ❑ Struktura 32-bitového ukazatele (uvnitř oblasti):




### ❑ Nahrání 32-bitového ukazatele (uvnitř oblasti):

L P#25.3 (P = ukazatel - pointer, Bytová adresa = 25, Bitová adresa: 3)





**Instrukce s 32-bitovým ukazatelem**

Pomocí 32-bitového ukazatele mohou být adresovány následující proměnné:

- Bity pro bitové logické operace.
  Jako identifikátor typu lze použít I, Q, M, L, DIX nebo DBX
- Byty, slova a dvojitá slova pro nahrání nebo uložení hodnoty. Lze použít identifikátor typu IB, IW, ID, DBB, DBW, DBD, DIB, DIW, DID, PIB, PIW, PID.

Tato adresa operandu je chápána jako 32-bitový ukazatel. Ve dvojitém slově jsou nejnižší 3 bity (bit 0... bit 2) chápány jako bitová adresa, dalších 16 bitů (bit 3 ... bit 18) je interpretováno jako bytová adresa operandu. Bity 19 ... 31 nejsou vyhodnocovány.

**Poznámka**

Při použití nepřímé adresace u instrukcí *L a T* je nutné, aby bitová adresa ukazatele byla 0.

V opačném případě bude CPU při zpracování takovéto instrukce vyhlášena *run-time* chyba.

**Nahrání 32-bitového ukazatele**

Instrukce pro nahrání 32-bitového ukazatele má následující syntaxi:

L P#<Bytová adresa>.<Bitová adresa>

**Uložení ukazatele**

16-bitový a 32-bitový ukazatel může být uložený v následujících proměnných:

- M - M-bity
- L - Lokální data
- D - Datové bloky (DB nebo DI)

## Vlastnosti paměťové nepřímé adresace

- **Oblasti pro ukládání 16-bitového a 32-bitového ukazatele:**
  - M-Bity (s možností absolutní nebo symbolické adresace,
    t.j.: *OPN DB[MW30]*, OPN *DI["Motor_1"]*, atd.
    *A I[MD30]*, *T QD["Speed_1]*, atd.)
  - Lokální data (s možností absolutní nebo symbolické adresace,
    t.j.: *OPN DB[LW10]*, *OPN DI[#DB_NO]*, atd.
    *A I[LD10]*, *T QD[#Pointer]*, atd.)
  - Sdílené datové bloky (pouze absolutní adresace, DB musí být napřed otevřeno,
    t.j..: *OPN DB[DBW0]* (přepíše DB registr !!!), *OPN DI[DBW22]*, atd.
    *A I[DBD10]*, *T QD[DBD22]*, atd.)
  - Instanční datové bloky (pouze absolutní adresace, DI musí být napřed otevřen,
    t.j.: *OPN DB[DIW20]*, *OPN DI[DIW0]* (přepíše DI registr !!!), atd.
    *A I[DID10]*, *T QD[DID22*, atd.)

- **Vlastnosti zpracování ukazatele v FB a FC blocích**
  - Ukazatele přiřazené parametrům nemohou být zpracovány přímo
  - Tyto ukazatele musí být nejprve uloženy do lokálních proměnných a teprve potom mohou být použity pro nepřímou adresaci proměnných





**Oblasti pro ukazatele**

Při nepřímé adresaci může být do paměti uložen 16-bitový nebo 32-bitový ukazatel. Hodnota ukazatele může být uložena v následujících oblastech:

- M-bity : vnitřní proměnné CPU, odkaz na proměnnou lze definovat absolutně nebo symbolickým názvem z globální tabulky symbolických názvů.
- Lokální data: tyto lokální proměnné mohou být adresovány absolutně nebo symbolickým názvem definovaným v deklarační tabulce bloku.
- Sdílený datový blok: může být adresován pouze absolutně. Pokud je ukazatel uložen v datovém bloku, musí být před jeho zpracování otevřen příslušný datový blok v DB registru instrukcí OPN DBn.
- Instanční datový blok: může být adresován také pouze absolutně. Při použití DI bloku musí být dodrženy následující podmínky:

  OB a FC: V těchto blocích může být ukazatel, uložený v DI bloku, zpracován stejně, jako by byl uložen v DB bloku. Je potřeba si pouze uvědomit, že je používán registr DI místo DB registru.

  FB: v tomto případě nelze použít u instančních dat (tj. parametry nebo statické proměnné) symbolickou adresaci k paměťové nepřímé adresaci.

Absolutní přístup k lokálním datům uvnitř FB bloku je principielně možný přes "adresu" uvedenou v deklarační tabulce FB bloku. Je ale třeba mít na paměti, že v případě multiinstančního FB bloku není tato adresa absolutní v rámci DI, ale relativní vůči aktuální adrese v AR2.

Z tohoto důvodu nelze v multiinstančím FB bloku přistupovat k instančním hodnotám pomocí absolutních adres.

**Poznámka**

Je-li nutné ukládat hodnotu ukazatele do statické proměnné, je před použitím této hodnoty nutné zkopírovat ji do dočasné proměnné typu *temp* a teprve tuto proměnnou lze použít k nepřímé adresaci.

# Příklad pro nepřímou adresaci

```
FC30: Příklad pro nepřímou adresaci

Network 1:     nepřímé otevření DB
      L    #dbnumber            // uložit číslo DB do MW100
      T    MW   100             //
      OPN  DB[MW   100]         // Otevřít DB


Network 2:     Smyčka mazání

      L    P#18.0               // uložit koncovou adresu(DBW18)jako ukazatel
      T    MD   40              // do MD 40;
      L    10                   // nastavit čitač smyčky na 10
anf:  T    MB   50              // a uložit do MB 50;
      L    0                    // nahrát počáteční hodnotu
      T    DBW[MD   40]         // a uložit do DB;
      L    MD   40              // nahrát ukazatel
      L    P#2.0                // a snížit o 2 byty
      -D                        // a uložit zpět
      T    MD   40              // do MD 40;
      L    MB   50              // nahrát čítač smyčky
      LOOP anf                  // zpracovat smyčku;
```

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_04cz.9

Školicí středisko
firmy E&A spol. s r.o.

**Popis**

Tento příklad ukazuje funkci, která inicializuje vstupní hodnoty datového bloku na hodnotu 0. Číslo datového bloku je zadáno ve vstupním parametru.

Datový blok je otevřen v segmentu 1. V tomto případě je číslo datového bloku (vstupní parametr: #dbnumber) zkopírováno do proměnné MW100 a datový blok je nepřímo otevřen přes tuto proměnnou.

V segmentu 2 má být 10 datových slov v DB nastaveno na 0 pomocí smyčky. Čitač smyčky LOOP používá,v tomto případě, proměnnou MB50.

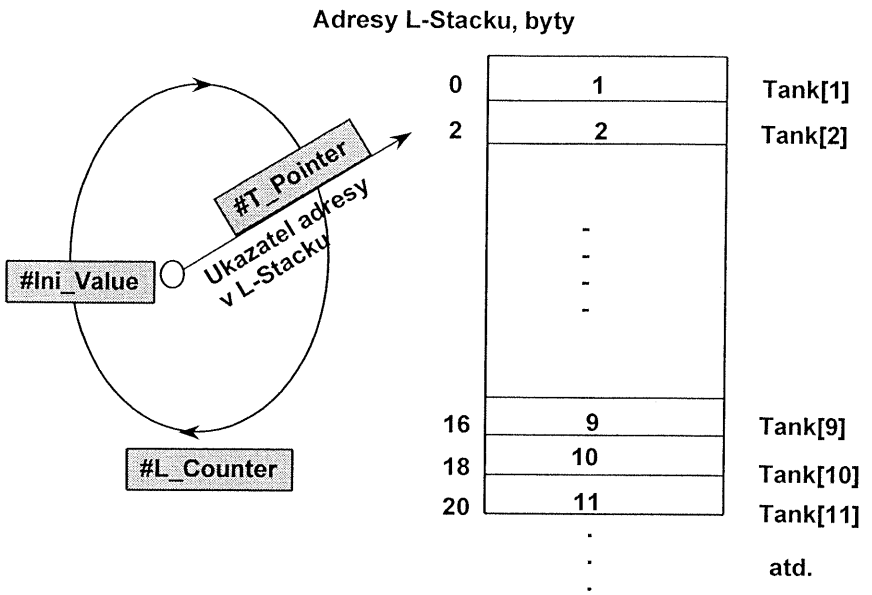
Zápis hodnoty 0 do jednotlivých datových slov je proveden nepřímou adresací přes proměnnou MD 40.

Před vstupem do smyčky je do MD40 uložen ukazatel na aktuální datové slovo. Při každém průběhu smyčkou je adresa v MD40 snížena o P#2.0, protože je prováděn přenos hodnot po slovech a ne po bytech.

**Poznámka**

Pro praktické použití je vhodné doplnit ukázku o kontrolu zadaného čísla datového bloku, parametrické zadání inicializační hodnoty a kontrolu požadované délky datového bloku. Další vhodná úprava spočívá v odstranění proměnných z oblasti M-bitů.

# Cvičení 4.1: Programová smyčka s nepřímou adresací






Datum: 24.11.2002
Soubor: PRO2_04cz.10

Školicí středisko
firmy E&A spol. s r.o.

**Cíl cvičení**

Důvěrně se seznámit s nepřímou adresací na praktickém příkladu s programovou smyčkou.

**Zadání**

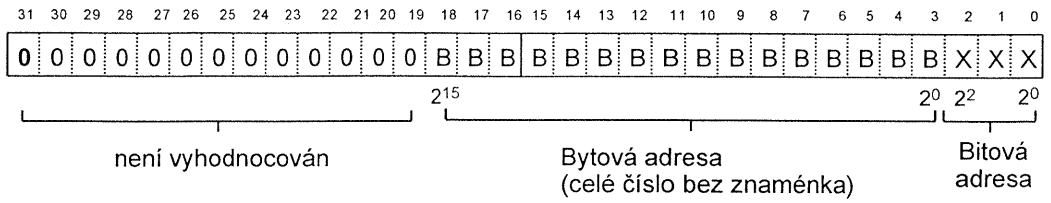
Při tvorbě programové smyčky použít nepřímou adresaci. Do proměnných zapsat číselnou řadu od 1 do 100.

1. Vytvořit FC41.
2. V deklarační části FC41 definovat pole "Tank" se 100 prvky typu "INT".
3. Prvky Tank[1] až Tank[100] naplnit vzestupnou číselnou řadou 1-100.
   K tomuto účelu lze použít programovou instrukci LOOP.

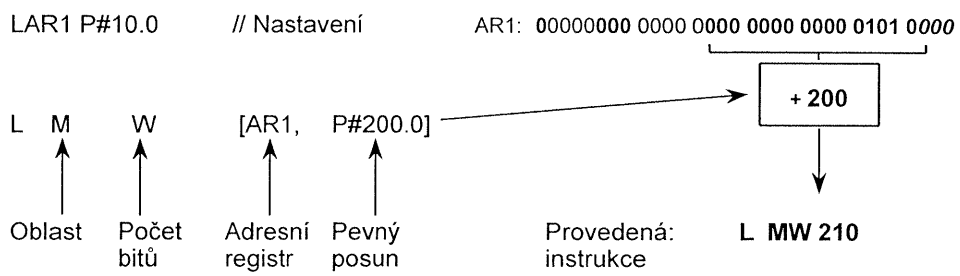
   - nejprve je nutno deklarovat dočasné proměnné *#Counter* ,*#Ini_Value* typu *INT* a *#Pointer* typu *DWORD*.
   - uložit hodnotu čítače smyčky do proměnné *#Counter* a nastavit hodnotu proměnné *#Ini_Value*, která bude zapisována do proměnných *#Tank[..]*.
   - Jednotlivé proměnné *Tank[..]* adresovat nepřímo s použitím proměnné *#Pointer*.
4. Vyvolat FC41 z OB1 a otestovat program s použitím funkce *Debug -> Monitor*.

## Nepřímá adresace uvnitř typové oblasti

### ❑ ukazatel uvnitř oblasti v AR 1 nebo AR2:




### ❑ Syntaxe:








**Přehled**

K uložení ukazatele proměnné, která je uvnitř definované oblasti lze využít také adresní registry *AR1* a *AR2*).

Tyto adresní registry obsahují v tomto případě 32-bitový ukazatel se stejným významem, jako v paměťové nepřímé adresaci.

**Syntaxe**

Instrukce registrové nepřímé adresace obsahuje:

- instrukci (t.j.: A, L, T, atd.)
- adresní identifikátor (I, MB, QD, atd.), který je kombinací identifikátoru oblasti (I, Q, M, DB, DI, atd.) a identifikátoru šířky proměnné (B=Byte, W=WORD, D=DWORD).
- adresní registr spolu s pevným posunem, který je uzavřen v hranatých závorkách. Tento pevný posun je před zpracováním instrukce přičten k obsahu adresního registru.

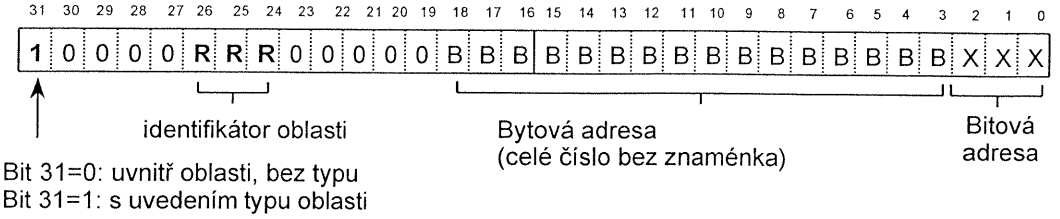
Adresní registr obsahuje bytovou a bitovou adresu nepřímo adresované proměnné.

**Poznámka**

- Při nepřímé adresaci proměnné *byte*, *word* nebo *double word* musí být bitová adresa nastavena na 0. V opačném případě vyhlásí CPU při zpracování této instrukce chybu.
- Je-li při nepřímé adresaci uvnitř oblasti v *AR1* nebo *AR2* definována adresa s uvedením typu oblasti, není tento typ oblasti při zpracování instrukce vyhodnocen.

## Nepřímá adresace s uvedením typové oblasti

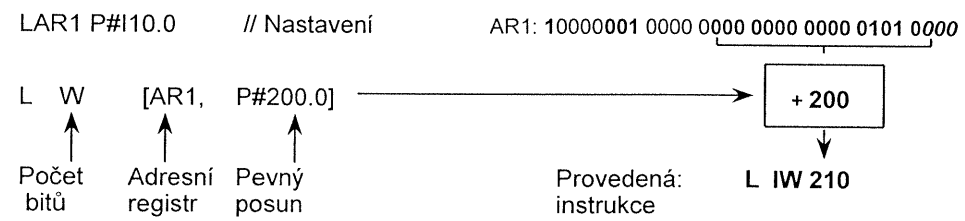
### ❑ Ukazatel s oblastí v AR 1 nebo AR2:




### ❑ Identifikátor oblasti:

| | | | |
|---|---|---|---|
| 000 | Periferie | 001 | Vstupy (PII) |
| 010 | Výstupy (PIQ) | 011 | M-Bity(vnitřní proměnné) |
| 100 | Data v DB registru | 101 | Data v DI registru |
| 110 | Lokální data | 111 | LD volaného bloku |

### ❑ Syntaxe:




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_04cz.12

Školicí středisko
firmy E&A spol. s r.o.

**Přehled** Při použití nepřímé adresace s uvedením typu lze v adresním registru definovat také typ oblasti, ve které leží adresovaná proměnná.

**Syntaxe** Při použití adresního registru je složení instrukce následující:

- instrukci (t.j.: A, L, T, atd.)
- identifikátor šířky proměnné (B=BYTE, W=WORD, D=DWORD).
- v hranatých závorkách uvedený adresní registr a pevný posun .

  Adresní registry musí obsahovat v tomto případě kromě adresy ve stylu *byte.bit* také identifikátor oblasti, ve které leží adresovaná proměnná.

  Pevný posun je ve tvaru *byte.bit* a je před zpracováním instrukce přičten k obsahu adresního registru.

**Poznámka**

- Při nepřímé adresaci proměnné šířky *byte, word* nebo *double word* musí být bitová adresa 0, v opačném případě vyhlásí CPU při zpracování této instrukce chybu.
- Adresace proměnné, ležící v oblasti s identifikátorem 110 vyvolá při zpracování chybové hlášení "*unknown area identifer*"(neznámá oblast).

  Nepřímá adresace proměnných ležících v této oblasti je možná pouze při nepřímé adresaci uvnitř oblasti - bez udání typu.

## Instrukce pro nahrání adresních registrů

### ❑ Nahrání hodnoty do adresního registru

| | |
|---|---|
| ○ LARn (n =1 nebo 2): | ulož obsah *ACCU1* do *ARn* |
| ○ LARn Var | ulož obsah proměnné Var do *ARn* |
| ○ LARn P#<Adresa> | ulož do *ARn* hodnotu ukazatele |

**Var:**

| | |
|---|---|
| ○ CPU registr: | AR1, AR2 (tj. *LAR1 AR2* a *LAR2 AR1*) |
| ○ 32-bitová proměnná: | MDn, LDn, DBDn, DIDn (tj. *L DBD5*, atd.) |
| ○ symbol. 32-bit proměnná: (sdílené a lokální) | 32-bitová sdílená proměnná (t.j.. *LAR1 "Index"*, atd.) a lokální proměnná v OB, FB a FC blocích (t.j. *LAR1 #Address*, atd..) |

**P#<Adresa>**

| | |
|---|---|
| ○ Ukazatel na absolutní logickou adresu: | En.m, An.m, Mn.m, Ln.m, DBDn.m, DIDn.m (t.j. *LAR1 P#M5.3*, *LAR2 P#I3.6*, atd.) |
| ○ Ukazatel na lokální, symbolickou adresu | OB: *temp* proměnné (t.j.: *LAR1 P##Pointer*, atd.)<br>FB: IN-, OUT-, INOUT-, STAT- a TEMP- proměnné.<br>FC: TEMP proměnné (*LAR1 P##Loop*, atd.) |

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_04cz.13

Školicí středisko
firmy E&A spol. s r.o.

**Nahrání adresy**

Adresní registr lze jednoduše inicializovat s použitím instrukce pro nahrání hodnoty ***L**oad.*
Instrukce LARn (n=1, 2) nahraje ukazatel do adresního registru ARn. Jako zdroj nahrané hodnoty může být použit *ACCU1, ARn* nebo dvojité slovo z oblasti M-bitů, lokálních proměnných, sdíleného datového bloku nebo instančního datového bloku. Přístup k dané proměnné může být proveden absolutně nebo symbolicky.

Není-li definována proměnná, jejíž obsah se má uložit do *Arn,* je automaticky převzata hodnota uložená v *ACCU1.* Obsah *ACCU1* nebo definované proměnné musí být ve formátu ukazatele tj. *pointer.*

**Nahrání ukazatele**

Hodnota ukazatele může být získána v "přirozené" podobě pomocí instrukce:

- L P#<identifikátor oblasti>n.m

Pomocí této instrukce může být do adresního registru uložena adresa obsahující také identifikátor oblasti proměnné. Při použití této metody je možná pouze absolutní adresace proměnné.

K adresaci lokální proměnné lze použít následující variantu instrukce:

- LARn P##*Address* (n=1, 2),

kde *n* udává číslo adresního registru, do kterého se má hodnota ukazatele uložit. V tomto případě je do *ARn* uložena adresa prvního bytu lokální proměnné. Tento způsob adresace lze použít všech lokálních proměnných definovaných v OB, FB a FC blocích stejně tak jako u statických proměnných definovaných v FB blocích.

**Poznámka**

Uložení hodnoty ukazatele IN, OUT a INOUT parametru (*#Name*) není možné provést přímo. Je nejprve nutné uložit hodnotu ukazatele parametru do *ACCU1* a teprve potom tuto hodnotu lze uložit do *ARn:*

- L P##*Param* (uloží ukazatel na parametr *#Name* do *ACCU1*)
  *LARn* (uloží obsah *ACCU1* do *ARn*)

## Ostatní instrukce pro práci s AR registry

- **Načtení hodnoty adresního registru**
  - TARn (n =1 nebo 2): Přenese obsah ARn do ACCU1
  - TARn <Adresa> Přenese obsah ARn do proměnné <Address>

  <Adresa>:
  - CPU registr: AR2 (t.j. *TAR1 AR2* )
  - 32-bitová absolutní proměnná:MDn, LDn, DBDn, DIDn (tj. *TAR2 MD5*, atd.)
  - symbolická 32-bitová proměnná(globální nebo lokální) 32-bitová globální proměnná (t.j. *TAR1 "Index"*, atd.) a lokální proměnná bloků OB, FB a FC (t.j. TAR1 #Adresa,atd.)
- **Záměna adresních registrů**
  - TAR zamění ("prohodí") obsahy registrů AR1 a AR2
- **Přičítání do adresního registru**
  - +ARn přičte ACCU1-L do ARn
  - +ARn P#n.m přičte ukazatel P#n.m do AR1 nebo AR2





**Načtení hodnoty AR registru**

Instrukce *TARn* umožňuje načtení hodnoty adresního registru *ARn* do definované proměnné. Touto proměnnou může být dvojité slovo v oblasti M-bitů, lokálních proměnných, globálního datového bloku nebo instančního datového bloku.
Není-li definována žádná proměnná, je obsah adresního registru uložen do *ACCU1*. Předchozí obsah je přesunut do *ACCU2*. Obsah *ACCU2* je zapomenut.

Obsah *ACCU3* a *ACCU4* (u S7-400) zůstává nezměněn.

**Záměna adresních registrů**

Instrukce *TAR* zamění obsah AR1 za AR2 a naopak.

**Přičítání do adresních registrů**

Obsah adresních registrů lze měnit také pomocí matematické operace sčítání, např. při programování smyčky. Přičítanou hodnotou může být konstanta (ukazatel bez udání typové oblasti proměnné) nebo obsah pravého slova *ACCU1-L*.

Instrukce +AR1 a +AR2 interpretují hodnotu obsaženou v *ACCU1* jako číslo ve formátu INT, rozšíří toto číslo na 24 bitů se správným znaménkem a přičte tuto hodnotu k obsahu adresního registru. Touto cestou lze provést i odečítání. Při překročení povoleného rozsahu bytové adresy (0 ... 65 535) jsou přebývající bity jednoduše ignorovány.

Instrukce +ARn P#n.m přičte do adresního registru konstantu ve tvaru ukazatele. Ukazatel může mít maximálně hodnotu P#4095.7.

Žádná z uvedených instrukcí nemění hodnoty stavových bitů.

## Charakteristika nepřímé adresace s registry

- **Interní použití AR1 editorem STL/LAD/FBD.**
  - Při přístupu k parametrům FC bloku, je hodnota **AR1** a **DB registru** přepsána, pokud je parametr komplexního typu *ARRAY*, *STRUCT*nebo *DATE_AND_TIME* .
  - Při přístupu k INOUT parametrům FB bloků je hodnota **AR1** a **DB registru** přepsána, pokud je parametr komplexního typu *ARRAY*, *STRUCT* nebo *DATE_AND_TIME*.

**Mezi instrukcemi nahrání ukazatele do AR1 a zpracování nepřímo registrově adresované proměnné nesmí být proveden přístup k lokální proměnné.**

- **Interní použití AR2 editorem STL/LAD/FBD.**
  - Registry **AR2** a **DI** jsou použity pro základní registrovou adresaci všech parametrů, lokálních a statických proměnných definovaných v blocích **FB**.

Bez zálohování obsahu **AR2** a **DI** nelze uživatelsky přepisovat uvnitř FB bloků tyto registry. V opačném případě dojde ke ztrátě možnosti oslovení parametrů daného FB bloku.

  - V FC blocích nejsou žádná omezení pro práci s těmito registry.(AR2 ,DI)





**Adresní registr AR1**

Editor STEPu7 používá adresní registr *AR1* pro přístup ke komplexním parametrům bloku. Při symbolickém přístupu ke komplexním parametrům typu *ARRAY* nebo *STRUCT* je obsah registrů AR1 a DB přepsán. Symbolický přístup k lokálním proměnným FB a FC bloků nepřepisuje obsah ani *AR1* ani *DB* registru.

**Adresní registr AR2**

STEP7 používá pro symbolický přístup ke statickým proměnným a všem parametrům registrovou nepřímou adresaci bez udání typu oblasti.
Pokud po přepsání obsahu těchto registrů není jejich obsah obnoven, není možné přistupovat k instančním proměnným a parametrům. Je-li nutné uvnitř FB bloků používat tyto registry pro uživatelské účely, je nutné obsah registrů *AR2 a DI* zálohovat a po provedení uživatelských výpočtů jejich obsah obnovit. Je doporučen následující postup :

1. Zálohování obsahu *DI* a *AR2* v proměnných typu *DWORD*:

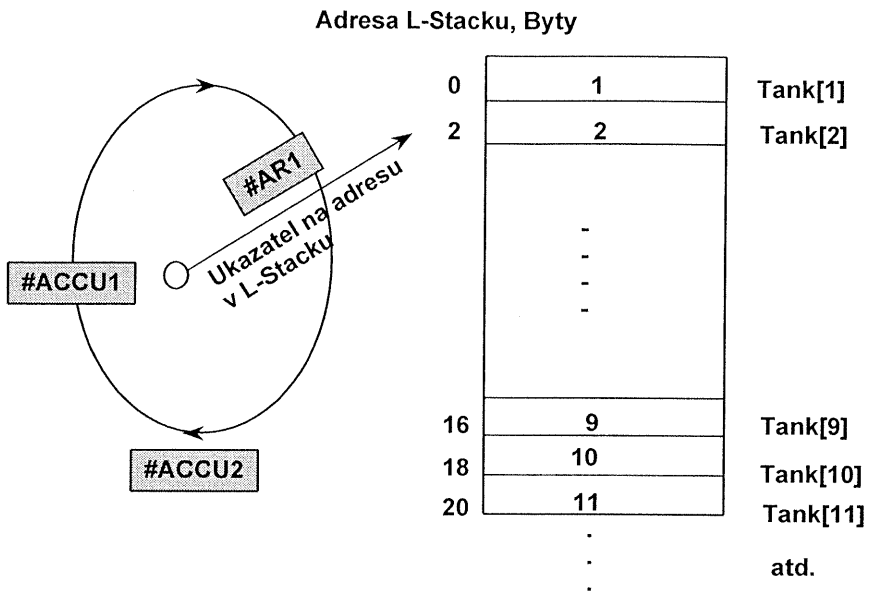
```
TAR2 #AR2_REG   // uložit AR2 do lokální proměnné #AR2_REG
L DINO          // nahrát obsah DI do ACCU1
T #DI_REG       // uložit ACCU1 do proměnné #DI_REG
```

2. Použití *DI* a *AR2* registrů pro uživatelský záměr. Při těchto akcích nelze přistupovat k parametrů nebo statickým proměnným FB bloku.
3. Obnovení obsahu *DI* a *AR2* registrů:

```
LAR2 #AR2_REG   // nahrání obsahu proměnné #AR2_REG do AR2
OPN DI[#DI_REG] // obnovení DI registru
```

Po provedení těchto instrukcí lze k parametrům a statickým proměnným FB bloku opět přistupovat symbolicky.

# Cvičení 4.2: Programová smyčka s nepřímou adresací s registrem




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_04cz.16

Školicí středisko
firmy E&A spol. s r.o.

**Cíl cvičení**

Důvěrně se seznámit s nepřímou adresací pomocí adresních registrů při použití programové smyčky.

**Definice zadání**

S použitím nepřímé registrové adresace v programové smyčce zapsat do 100 proměnných hodnoty od 1 do 100. Program řešit bez použití lokálních proměnných.

Nastavit čítač programové smyčky a inicializovat hodnoty akumulátorů.

K adresaci proměnných Tank[..] použít adresní registr *AR1* a metodu nepřímé registrové adresace bez udání typu oblasti proměnné.

**Postup**

1. Vytvořit blok FC42.
2. V deklarační části FC42 definovat pole se jménem Tank o 100 proměnných datového typu *INT* .
3. Přednastavit hodnoty proměnných Tank[1] až Tank[100] na hodnotu 1 až 100.
   - Použít programovou smyčku (instrukce LOOP)
   - Použít adresní registr *AR1* pro adresování jednotlivých prvků pole Tank[..].
4. Vyvolat blok FC42 v OB1 a otestovat program.

## Typy ukazatelů ve STEPu7

- **16-bitový ukazatel pro paměťovou nepřímou adresaci**
  - Pro nepřímý přístup k časovačům, čitačům a otevírání datových bloků pomocí hodnoty uložené v proměnné
- **32-bitový ukazatel pro paměťovou a registrovou nepřímou adresaci**
  - 32 bitový ukazatel bez udání typu oblasti pro paměťovou a registrovou nepřímou adresaci proměnných v PI, PQ, I, Q, M, DB, DI a L (lokální proměnné)
  - 32 bitový ukazatel s udáním typu oblasti proměnné pro registrovou nepřímou adresaci proměnných v PI, PQ, I, Q, M, DB, DI, L a V (lokální proměnná v L-Stacku volajícího bloku)
- **48-bitový ukazatel (datový typ POINTER)**
  - Datový typ pro předání parametrů blokům (FB a FC)
  - Obsahuje, na doplnění 32-bitového ukazatele s udáním typu oblasti proměnné, deklaraci DB čísla
- **80-bitový ukazatel (datový typ ANY)**
  - Datový typ pro předání parametrů blokům (FB a FC)
  - doplněk k 32-bitovému ukazateli s udáním typu oblasti proměnné







**Typy ukazatelů ve STEPu 7**

Kromě ukazatelů popsaných v předchozích kapitolách (16-bitový, 32-bitový bez udání typu oblasti a 32-bitový s udáním typu oblasti proměnné) rozeznává STEP 7 ještě další dva doplňkové typy ukazatelů:

- 48-bitový ukazatel (datový typ *POINTER* )
- 80-bitový ukazatel(datový typ *ANY*)

16 a 32-bitový ukazatel může být přímo nahrán do akumulátoru nebo adresního registru a použit k nepřímé adresaci.

Ukazatele typu *POINTER* a *ANY* (větší než 32 bitů) nemohou být přímo nahrány do akumulátoru. Slouží výhradně ke kompletní adresaci aktuálních parametrů přiřazených formálním parametrům volaného bloku.

Například lze deklarovat parametr bloku jako datový typ *POINTER* nebo *ANY* a při volání tohoto bloku lze tomuto parametru přiřadit adresu aktuálního parametru.

**POINTER**

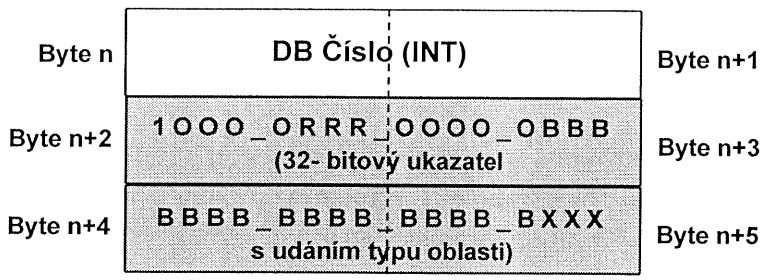
Datový typ POINTER nemá pro uživatele velký význam. Je používán editorem STL/LAD/FBD hlavně k přiřazení aktuálního parametru typu *ARRAY*, *STRUCT*, atd. formálnímu parametru bloku FB nebo FC.

**ANY**

Datový typ ANY je použit ve STEPu 7 k přiřazení parametrů systémových funkcí (SFC) a systémových funkčních bloků (SFB).

## Struktura a použití ukazatele typu POINTER

### ❑ Stavba datového typu *POINTER*




### ❑ Přiřazení parametru typu *POINTER*

- **Zobrazení ukazatele**

  P#DBn.DBX x.y — kde: n= číslo DB , x= číslo bytu, y= číslo bitu
  P#DIn .DIX x.y — (t.j.: P#DB5.DBX3.4, P#DI2.DIX10.0, atd.)
  P#Zx.y — kde: Z= oblast, t.j.: P, I, Q, M a L
  (t.j.: P#I5.3, P#M10.0, atd.)

- **Deklarace adresy proměnné:**

  MD30 — (v tomto případě jsou čísla DB a
  #Motor_on — bitová adresa zadány automaticky)
  "Motor_1".speed

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_04cz.18

Školicí středisko
firmy E&A spol. s r.o.

**Datový typ POINTER** Parametr datového typu POINTER obsahuje, kromě údajů obsažených v ukazateli s udáním oblasti proměnné, také číslo datového bloku jako celé číslo bez znaménka v rozsahu 0 ... 65,535. Tento údaj je definován, odkazuje-li ukazatel s udáním oblasti proměnné na globální nebo instanční datový blok.

Ve všech ostatních případech, kdy ukazatel odkazuje na ostatní oblasti (P, I, Q, M, L), je v prvních dvou bytech typu *POINTER* uložena hodnota 0.

**Přiřazení parametrů** Během volání bloku (FC nebo FB) musí být parametru typu *POINTER* přiřazen aktuální operand.

**Ukazatel** V tomto případě je přiřazena hodnota typu *Pointer* (P#...):

- P#DB10.DBX2.0 (datový bit 2.0 v DB10, oblast "DB")
- P#I5.3 (I5.3, DB číslo = 0, oblast "PII")

**Deklarace adresy** Druhou možností definice adresy je udání proměnné t.j. bez deklarace typu P#. Proměnná může být deklarována absolutně (se zadáním čísla DB bloku, identifikátoru proměnné a bytové a bitové adresy této proměnné):

- DB5.DBW10 (Bit 10.0, DB číslo = 5, oblast "DB")

nebo symbolicky:

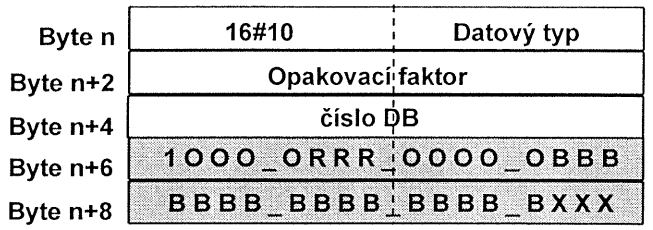
- #Motor_on, "Motor_1".speed

V obou případech *STL/LAD/FBD Editor* jednoznačně identifikuje číslo datového bloku, typ proměnné, bytovou a bitovou adresu proměnné a přiřadí výslednou hodnotu parametru typu *POINTER*.

**Poznámka** Výsledná hodnota je po uložení bloku zobrazena v obou případech.

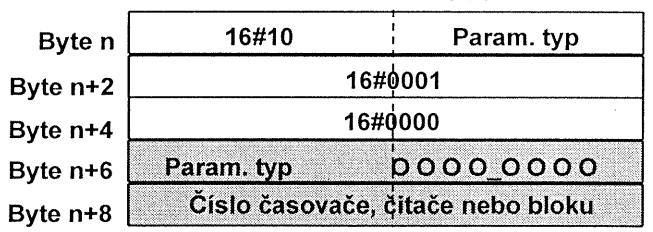
# Konfigurace ukazatele typu ANY

- Datový typ *ANY*




| Datový typ | Identifikátor |
|---|---|
| VOID | 00 |
| BOOL | 01 |
| BYTE | 02 |
| CHAR | 03 |
| WORD | 04 |
| INT | 05 |
| DWORD | 06 |
| DINT | 07 |
| REAL | 08 |
| DATE | 09 |
| TOD | 0A |
| TIME | 0B |
| S5TIME | 0C |
| DT | 0E |
| STRING | 13 |

- Ukazatel *ANY* pro parametrické typy




| Param. typ | Identifikátor |
|---|---|
| BLOCK_FB | 17 |
| BLOCK_FC | 18 |
| BLOCK_DB | 19 |
| BLOCK_SDB | 1A |
| COUNTER | 1C |
| TIMER | 1D |

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_04cz.19

Školicí středisko
firmy E&A spol. s r.o.

**Datový typ ANY**

Ukazatel *ANY* obsahuje, na doplnění ukazatele s udáním typu oblasti, číslo DB bloku,identifikátor datového typu proměnné a opakovací faktor. To umožňuje identifikovat nejenom jednotlivou proměnnou ale celou datovou oblast.

STEP 7 rozeznává dvě varianty datového typu *ANY* :

- proměnné s datovým typem : ukazatel má naznačenou strukturu a obsahuje identifikátor 16#10 (v STL), identifikátor datového typu, opakovací faktor, číslo DB bloku a ukazatel s udáním typu oblasti proměnné.
- proměnné s parametrickým typem: v tomto případě ukazatel *ANY* obsahuje pouze identifikátor 16#10 (v STL), identifikátor parametrického typu a 16-bitové číslo bez znaménka v bytech n+8 a n+9, které odpovídá číslu bloku. Byty n+2 až n+7 jsou doplněny nulami.

**Deklarace ukazatele ANY**

Proměnné datového typu ANY mohou být obecně deklarovány jako parametry typu IN, OUT a INOUT.
FB bloky nabízí doplňkovou možnost deklarovat proměnnou typu *ANY* jako lokální proměnnou. Použití lokální proměnné umožňuje variabilní definování hodnoty ukazatele za běhu programu.

**Identifikátor oblasti (RRR):**

| | | | |
|---|---|---|---|
| 000 | I/O | 001 | Vstupy (PII) |
| 010 | Výstupy (PIQ) | 011 | M- Bity |
| 100 | Data v globálním DB bloku | 101 | Data v instančním DI bloku |
| 110 | Lokální data | 111 | Lokální data volajícího bloku |

## Přiřazení parametru typu ANY

- **Zadání přímé hodnoty:**
  - **P#[Data block.]Bitová adresa typ počet**

    P#DB10.DB12.0 REAL 20 — Ukazatel na pole v DB10, začínající bytem 12, které obsahuje 20 adres datového typu REAL (ARRAY[1..20] OF REAL)

    P#I 10.0 BOOL 8 — Ukazatel na pole 8 bitů v IB10
- **Deklarace adresy:**
  - **absolutní:**

    DB5.DBD10 — Datový typ: DWORD, opakovací faktor (RF): 1
    DB číslo: 5, ukazatel: P#DB5.DBX10.0

    IW32 — Typ: WORD, RF: 1, DB-No: 0, Ukazatel: P#I32.0

    T35 — Type: TIMER, No.: 35
  - **symbolická:**

    #Motor_1.speed
    "Pump:Start" — u elementárních datových typů, překladač určí odpovídající datový typ, nastaví RF=1 a určí hodnotu ukazatele
- **Poznámka**

  při symbolické deklaraci (ARRAY, STRUCT, STRING, UDT), je překladačem určena pouze oblast bytů a je předána v parametru *ANY*.





**Přiřazení**

Parametru typu *ANY* lze přiřadit jak adresu proměnné tak přímo hodnotu ukazatele.

**Přímá hodnota**

Při přiřazení přímé hodnoty (t.j.: P#DB5.DBX10.0 INT 8) nastaví *STL/LAD/FBD Editor* ukazatel *ANY* na hodnotu odpovídající typu a číslu v deklaraci.

Je-li uvnitř volaného bloku požadována kompletní informace o datovém typu odkazované proměnné (např.ARRAY[1..8] OF REAL) a opakovací faktor, musí být použita absolutní hodnota ukazatele.

**Zadání adresy**

Parametru typu *ANY* může být také přiřazena adresa, na kterou má odkazovat. Tato deklarace může být absolutní nebo symbolická, udáním symbolického jména proměnné.
Při absolutním zadání adresy *STL/LAD/FBD Editor* automaticky určí připojený datový typ (BOOL, BYTE, WORD, DWORD,), nastaví opakovací faktor=1, číslo DB bloku definuje stejně jako u ukazatele s udáním oblasti s prvním bitem adresy převede vše do struktury ukazatele.
Při zadání proměnné elementárního datového typu jejím symbolickým jménem, *STL/LAD/FBD Editor* určí stejným způsobem hodnotu ukazatele a předá ji parametru typu *ANY*.

**Poznámka**

Je-li proměnná komplexního typu (např. ARRAY[1..8] OF REAL), potom *STL/LAD/FBD Editor* předá pouze velikost proměnné v bytech (t.j.: opakovací faktor: 32, datový typ: BYTE)

## Nepřímé přiřazení parametru typu ANY

- **Přiřazení lokální aktuální adresy datovému typu ANY**
  - **deklarace lokální proměnné typu ANY**

    t.j.: temp rozšiřující ukazatel *ANY*
  - **zapsání hodnoty ukazatele do lokální proměnné *auxpointer* typu *ANY***

    t.j.:

```
LAR1  P##auxpointer          // uložení adresy proměnné auxpointer
L     B#16#01                // nahrání identifikátoru 01
T     LB [AR1,P#0.0]         // a přenos s posunem 0
L     ...
...
```

  - **Přiřazení parametru typu *ANY* bloku**

    t.j.:

```
CALL FC   111
     Targetfield:=#auxiliarypointer
```

- **Výhoda**
  - **dynamická změna parametru typu ANY za běhu programu**



Datum: 24.11.2002
Soubor: PRO2_04cz.21



**Nepřímé přiřazení**

Volající blok může přiřadit parametru typu ANY lokální proměnnou typu *ANY* Tato lokální proměnná může být uložena v oblasti lokálních dat volajícího bloku.

V tomto případě *STL Editor* neuloží do lokální proměnné ukazatel na lokální proměnnou, ale převezme její hodnotu ukazující na požadovanou proměnnou.

Editor převezme v tomto případě ukazatel typu *ANY* obsažený v lokální proměnné volaného FC nebo FB bloku.

**Výhoda**

Tato adresace umožňuje přiřadit ukazatel typu *ANY* parametru typu *ANY* a tak lze měnit dynamicky obsah parametru. Toto je výhodné zvláště při použití systémových funkcí (např. SFC 20 "BLKMOV" nebo SFC 21 "FILL" , viz.cvičení 7.5).

# Zpracování parametru typu ANY

| Adresa | Deklarace | Název | Typ | Počáteční hodnota | Komentář |
|---|---|---|---|---|---|
| 0.0 | in | Pointer | ANY | | |
| | out | | | | |
| | in out | | | | |
| 0.0 | temp | Data type | BYTE | | |
| 2.0 | temp | WF | WORD | | |
| 4.0 | temp | DB_No | WORD | | |
| 6.0 | temp | Be_Pointer | DWORD | | |

```
Network 1: Definování datového typu, opakovacího faktoru, DB-No a Be-
Pointer

     L    P##Pointer           // uložit adresu #Pointer do ACCU1
     LAR1                      // a nahrát ji do AR1;
     L    B [AR1,P#1.0]        // definovat datový typ z ukazatele
     T    #Datatype            // a uložit do lokální proměnné;
     L    W [AR1,P#2.0]        // definovat opakovací faktor
     T    WF                   // a uložit do  lokální proměnné;
     L    W [AR1,P#4.0]        // definovat číslo DB
     T    #DB_No               // a uložit do lokální proměnné;
     L    D [AR1,P#6.0]        // definovat ukazatel oblasti
     T    #Be_Pointer          // a uložit do lokální proměnné;
```





**Přehled**

Hodnota ukazatele typu POINTER nebo ANY může být ve volaném bloku (FB nebo FC) načtena a odpovídajícím způsobem zpracována a použitím registrové nepřímé adresace.

**Postup zpracování**

Vyhodnocení informace předané ukazatelem typu *ANY* je provedeno v následujících krocích.

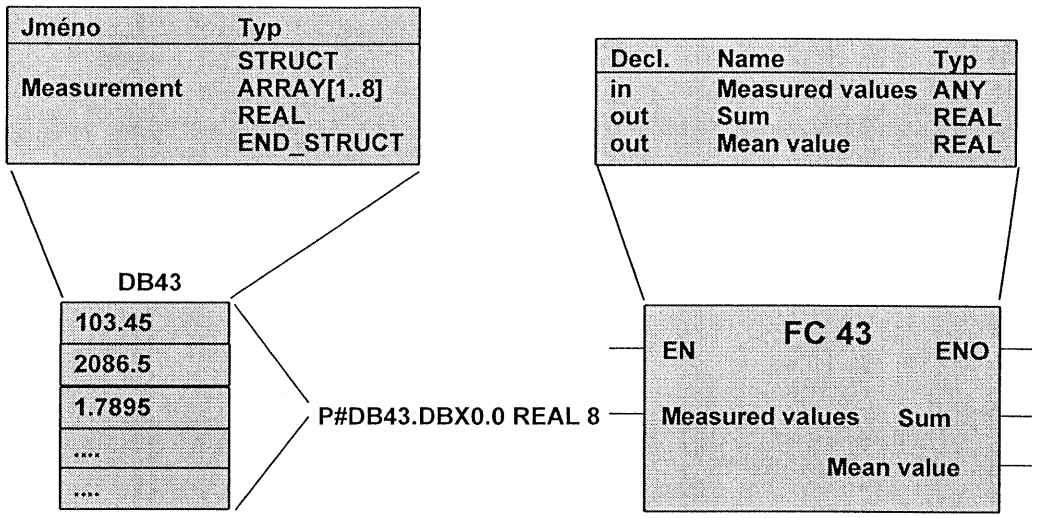
1. Nejdříve ze všeho je ukazatel s udáním oblasti předaný parametrem *ANY* nahrán do adresního registru *AR1* s pomocí následujících instrukcí:
   - LAR1 P##Pointer (v FB) nebo
   - L P##Pointer (v FC, adresu je nejprve nutno nahrát do *ACCU1*
     LAR1 a teprve potom ji lze uložit do *AR1* registru)

   Předaný ukazatel *ANY* je uložen v automaticky otevřeném instančním datovém bloku (u FB) nebo u FC v zásobníku lokálních proměnných volajícího bloku.

2. Při použití registrové nepřímé adresace může být hodnota předaná ukazatelem typu *ANY* přečtena a např. uložena do lokální proměnné pro pozdější zpracování.
   - L B[AR1,P#1.0] (načti identifikátor datového typu aktuálního parametru do *ACCU1*)
   - L W[AR1,P#2.0] (načti opakovací faktor do *ACCU1*)
   - L W[AR1,P#4.0] (načti do *ACCU1* číslo DB bloku ve kterém je aktuální parametr, nebo načti 0, je-li aktuální parametr v oblasti P, PII, PIQ, M, L)
   - L D[AR1,P#6.0] (načti ukazatel s udáním oblasti aktuálního parametru do *ACCU1*)

Informace uložená v pointeru "ANY" musí být následně zpracována v souladu s definicí řídícího algoritmu.

# Cvičení 4.3: Funkce pro výpočet celkového součtu a průměru








**Přehled**

Obecné FC nebo FB bloky mohou být vytvořeny s pomocí parametrů typu *ANY*. Takto vytvořené bloky nejsou závislá na konkrétních datových typech parametrů.

**Cíl**

Vytvořit FC43 s následující funkcí:

- funkce očekává hodnoty typu REAL ve vstupním parametru *Measured_values* (typ *ANY*).
- výsledek funkce - celkový součet je předán výstupním parametrem *Sum* (typ: REAL) a průměrná hodnota je ve výstupním parametru *Mean_value* (typ: REAL).
- je - li předán jiný datový typ, je hlášena chyba (parametr ENO , tj. BR bit=0)

**Postup**

1. Vytvořit FC43 a deklarovat výše uvedené parametry. Dále deklarovat odpovídající lokální proměnné pro uložení opakovacího faktoru, čísla DB bloku a datové oblasti proměnné.
2. Nejprve načíst datový typ proměnné z ukazatele *ANY* a ukončit FC43, pokud se bude jednat o jiný datový typ než *REAL*.
3. S pomocí programové smyčky (LOOP) vypočítat hodnoty jednotlivých parametrů a předat je.
4. Vytvořit DB43. Deklarovat proměnnou *Measurement* typu ARRAY[1..8] v DB43 a zadat hodnoty do DB bloku.
5. Vyvolat FC43 v OB1. Přiřadit vstupní a výstupní parametry a definovat hodnoty parametrů.
6. Nahrát program do CPU a otestovat ho.

# Správa dat v uživatelském programu

SIMATIC S7

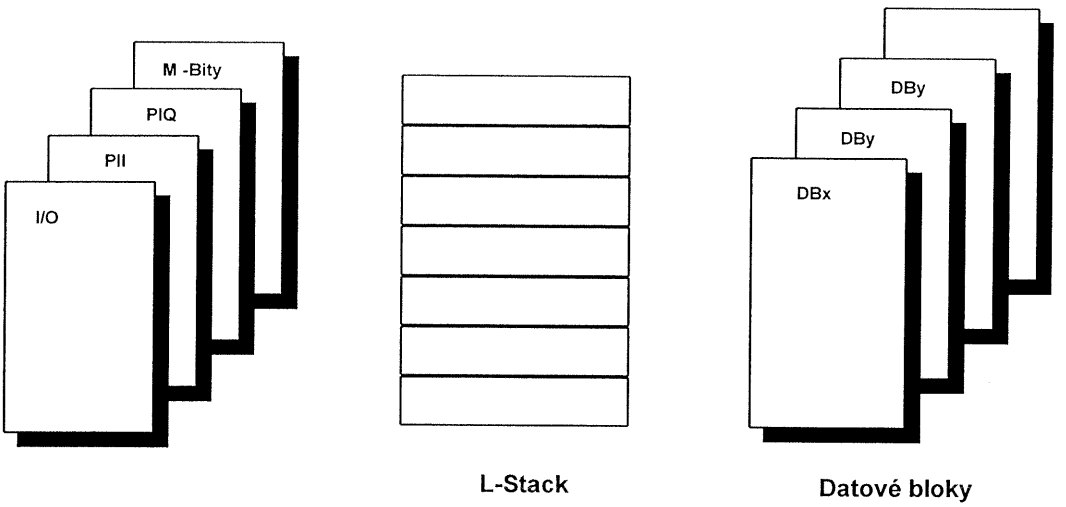


Datum: 24.11.2002
Soubor: PRO2_05cz.1

Školicí středisko
firmy E&A spol. s r.o.

**Obsah** **Strana**

## Oblasti pro ukládání dat




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_05cz.2

Školicí středisko firmy E&A spol. s r.o.

**Přehled**

Vedle logických bloků s definovaným uživatelským algoritmem může program také obsahovat technologické a jiné hodnoty nutné pro správnou funkci uživatelského programu.

Tyto hodnoty jsou obvykle uloženy v proměnných, které musí být definovány:

- alokací v paměti (např. P, PII, PIQ, M-bity, L-Stack, DB)
- datovým typem (základní nebo komplexní datový typ, parametrický typ)

Dále je nutné u proměnných rozlišovat způsob přístupu:

- globální (sdílené) proměnné, tj. takové, které jsou definovány v globální tabulce symbolických názvů nebo v globálním (sdíleném) datovém bloku
- lokální proměnné,tj. takové, které jsou definovány v deklarační tabulce OB, FC a FB bloků.

Proměnné mohou být alokovány v permanentní paměti v obrazu vstupů a výstupů, v periferiích, vnitřních proměnných (M-bitech) nebo v datových blocích nebo mohou být alokovány dynamicky během zpracování programu v oblasti *L-STACK*u.

**L-STACK**

Lokální zásobník dat (L-Stack) je oblast paměti určená k ukládání:

- lokálních proměnných logického bloku včetně startovních informací OB bloků.
- aktuálních parametrů při předávání parametrů při volání FC bloků.
- logických mezivýsledků v *LAD* zobrazení uživatelského programu

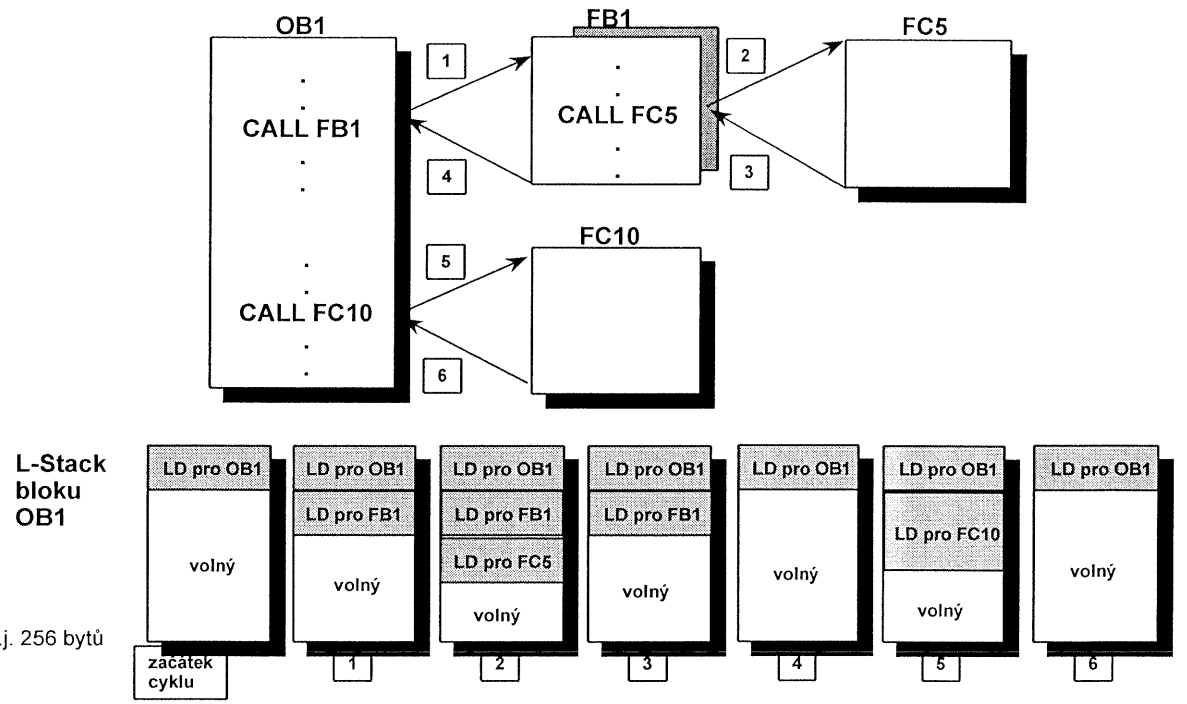
Každý OB blok si definuje vlastní L-STACK do kterého jsou ukládány požadované lokální proměnné.

U S7-300 je velikost L-STACKu pevně definována (256bytů/OB). U S7-400 je velikost L-STACKu uživatelsky definovatelná pro každý OB blok.

**Datové bloky**

Datové bloky používají logické bloky k ukládání hodnot proměnných. Na rozdíl od lokálních proměnných nejsou hodnoty uložené v datových blocích smazány po ukončení zpracování bloku nebo při zavření DB bloku.

## Funkce lokálního datového zásobníku




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_05cz.3

Školicí středisko firmy E&A spol. s r.o.

**Zásobník lokálních dat**

Každá prioritní třída, tj. každý blok OB má přiřazen L-STACK pro ukládání lokálních proměnných nebo volání vnořených bloků.
Před vyvoláním vnořeného bloku (OB, FB nebo FC) je systémem rezervována nová dynamická paměť pro ukládání lokálních proměnných definovaných v deklarační tabulce volaného bloku. Takto alokovaná paměť je uvolněna po zpracování instrukce BE tohoto bloku.

**Pořadí**

Na obrázku je zobrazena typická sekvence cyklického zpracování bloku OB1. Před zahájením cyklického zpracování operační systém alokuje potřebnou paměť pro lokální proměnné bloku OB1. Vedle těchto proměnných, deklarovaných uživatelem, vyhradí a inicializuje systém 20 bytů pro startovní informace.

**1**

Před zpracováním bloku FB1 rezervuje operační systém paměť potřebnou pro lokální proměnné definované v bloku FB1. Tato paměť je alokována bezprostředně za pamětí, která je vyhrazena lokálním proměnným bloku OB1.

**2**

Před zpracováním bloku FC5 rezervuje operační systém paměť potřebnou pro lokální proměnné bloku FC5. Tato paměť je alokována ihned za pamětí určenou proměnným bloku FB1.

**3**

Po dokončení bloku FC5 je alokovaná paměť uvolněna pro další použití.

**4**

Po dokončení bloku FB1 je alokovaná paměť uvolněna pro další použití.

**5**

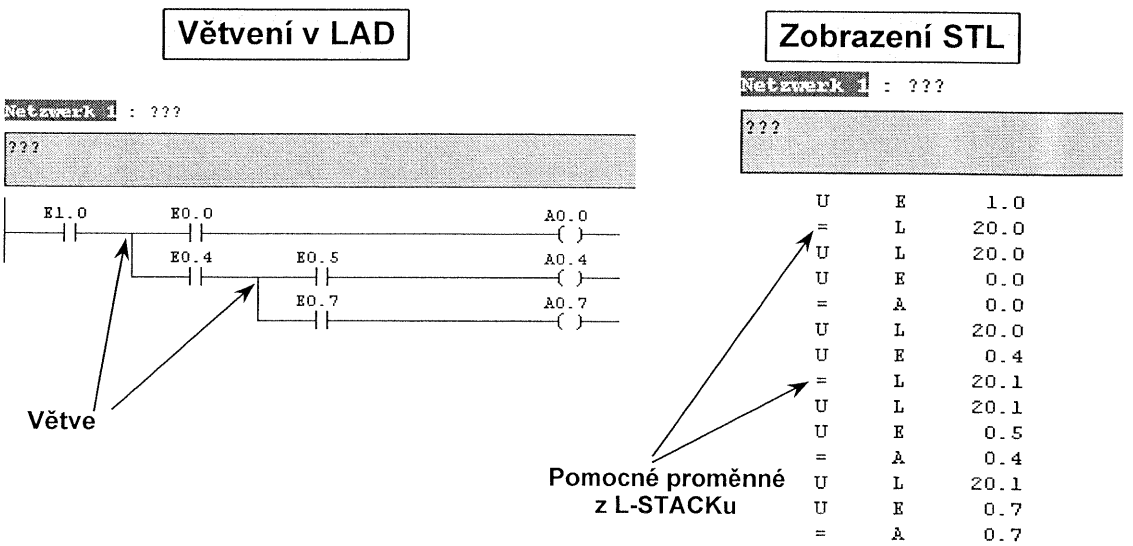
Před zahájením zpracování bloku FC10 alokuje operační sytém paměť potřebnou pro lokální proměnné tohoto bloku. Tato paměť je alokována ihned za oblastí určenou pro lokální proměnné bloku OB1.

Takto alokovaná paměť byla při zpracování programu původně určena blokům FB1 a FC5. Původní hodnoty zapsané do této oblasti paměti jsou nyní přepsány hodnotami lokálních proměnných definovaných v bloku FC10.

**Výhody**

Správa paměti lokálních proměnných je řízena operačním systémem a nelze ji řídit z uživatelského programu. Pokud je určitá prioritní třída přerušena blokem OB s jinou prioritou, lokální proměnné není nutné zálohovat. Různé OB bloky mají přiřazenu vlastní paměť pro lokální proměnné.

## Příklad: Přepisování pracovních hodnot




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_05cz.4

Školicí středisko
firmy E&A spol. s r.o.

**Větvení v LAD Editoru**

Obrázek ukazuje příklad větvení logických podmínek vytvořené v LAD Editoru přidáním dodatečných výstupů Q 0.7 a Q 0.4.

**Pracovní paměť a konektory**

Ve STEPu 5 nebylo možné programovat větvení přímo.Uživatel musel použít pomocnou proměnnou, zpravidla M-bit jako konektor na úrovni výstupu ze segmentu.

V dalším segmentu ( pomyslné větvi ) byl tento konektor použit v pozici vstupní proměnné do logického řetězce.

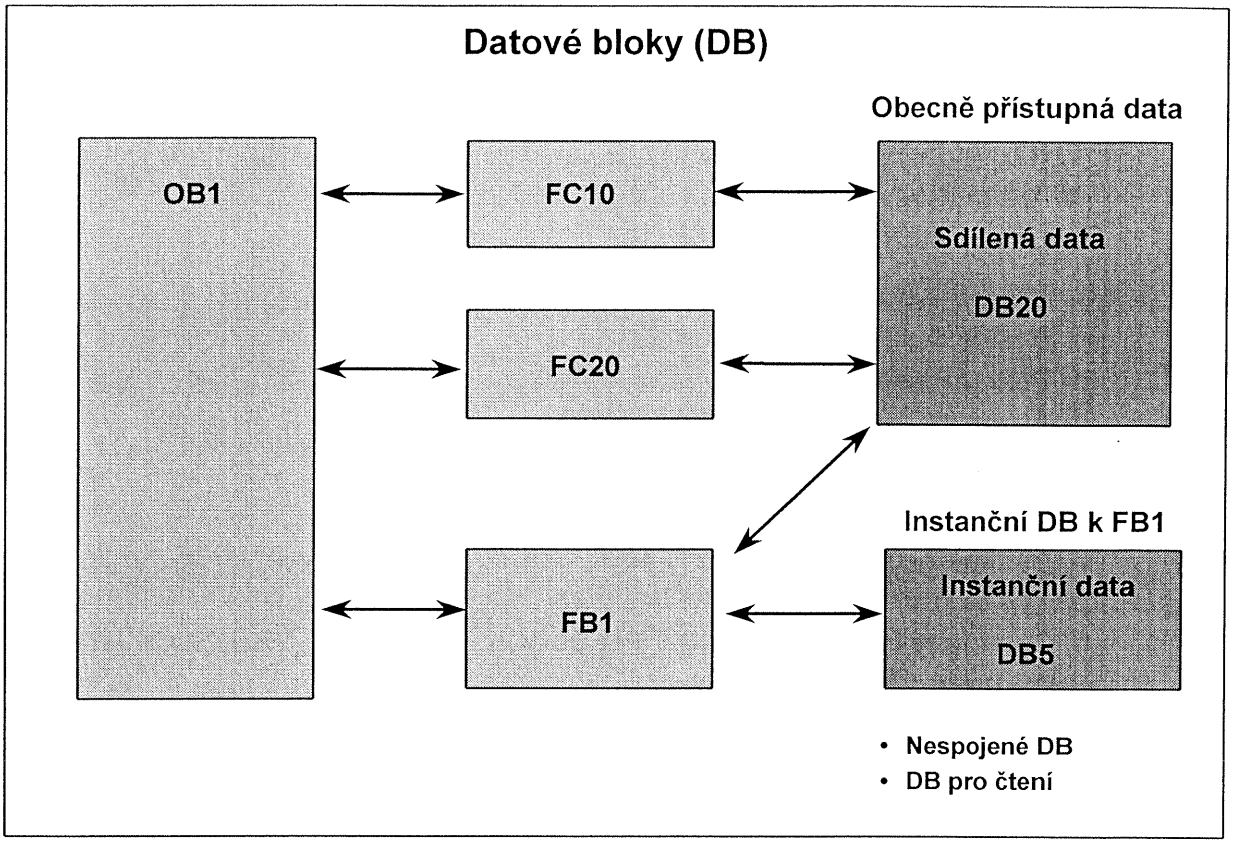
**LAD Editor**

S použitím LAD Editoru ve STEPu7 je možné programovat větve přímo. Interně je k této akci použita lokální proměnná - bitová proměnná z L-STACKu.

Použití proměnné z L-STACKu (ve výše uvedeném případě L 20.0 a L 20.1) zajišťuje, že při zpracování bloku nedojde k přepsání hodnot.

**Lokální proměnné**

Uživatel může také definovat lokální proměnné v deklarační části bloku a přistupovat k nim absolutně nebo symbolicky - přes symbolické jméno definované v deklarační tabulce.




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_05cz.5

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

Datové bloky slouží k ukládání uživatelských hodnot. Stejně jako logické bloky, zabírají datové bloky místo v uživatelské paměti. Proměnné hodnoty (např. číselné hodnoty) s nimiž program pracuje, jsou obsaženy v datových blocích.

Uživatelský program přistupuje k těmto hodnotám po bitech , bytech, slovech nebo dvojitých slovech. Tento přístup může být absolutní nebo pomocí symbolických jmen definovaných v globální tabulce symbolických názvů.

**Aplikační oblast**

Datové bloky, v závislosti na jejich obsahu, mohou být zřízeny dvojím možným způsobem.

- Globální datové bloky: obsahují hodnoty přístupné všem logickým blokům uživatelského programu.
- Instanční datové bloky: jsou přiřazeny pouze jednomu bloku FB. Tyto datové bloky mohou být zpracovány pouze přiřazeným FB blokem. Multi-instanční DB jsou zvláštní variantou instančního DB bloku. Multi-instanční DB je přiřazen opět jednomu bloku FB, ale podle určitých pravidel ukládá hodnoty vnořených FB bloků.

**Tvorba DB**

Globální DB bloky jsou generovány pomocí DB Editoru nebo s použitím předdefinovaných uživatelských datových typů UDT.

Instanční nebo multi-instanční DB jsou generována vždy s pomocí přiřazených FB bloků.

**Možnosti**

Pomocí dialogového okna "Properties" lze nastavit tyto vlastnosti:

*Unlinked DB* : Datové bloky sestavené s touto vlastností jsou při transferu do CPU uloženy pouze v editační paměti a nejsou přeneseny do pracovní části paměti.Jsou-li hodnoty uložené v těchto blocích potřeba pro zpracování programu, obsah příslušného bloku musí být přenesen do volné pracovní paměti systémovým blokem SFC20.

*Write-protected*: Datové bloky jsou v tomto případě chráněny před přepsáním hodnot ze strany uživatelského programu.

## Datové typy ve STEPu7

- Základní datové typy (do 32bitů)
  - Bitové datové typy (BOOL, BYTE, WORD, DWORD, CHAR)
  - Aritmetické datové typy (INT, DINT, REAL)
  - Časové typy (S5TIME, TIME, DATE, TIME_OF_DAY)
- Komplexní datové typy
  - Reálný čas (DATE_AND_TIME)
  - Pole (ARRAY[....] OF....)
  - Struktura (STRUCT... END_STRUCT)
  - STRING (řetězec znaků)
- Parametrické typy
  - Parametry časovač a čítač (TIMER, COUNTER)
  - Typ ukazatel (POINTER, ANY)
  - Parametry typu blok (BLOCK_FB, BLOCK_FC, BLOCK_DB, BLOCK_SDB)
- Vlastní definované datové typy
  - Uživatelem definovaný datový typ UDT

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_05cz.6

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

Datové typy zavádí vlastnosti dat, tj. způsob prezentace hodnoty, povolený rozsah a instrukce pro zpracování těchto hodnot.

Ve STEPu 7 jsou na výběr 3 třídy datových typů :

- Základní datové typy
- Komplexní datové typy
- Parametrické typy

**Použití datových typů**

Datové typy jsou použity v STL programu nejenom implicitně při zpracování instrukce v CPU, ale také explicitně při přiřazení parametrů bloku.

Zpracování instrukce: vstupní adresy STEP7 příkazů jsou implicitně přiřazeny datovým typům uvnitř STL programu. Pomocí digitálních příkazů se obsah akumulátoru spojuje s jiným a je na programátorovi, aby naplnil akumulátory hodnotami korektních datových typů.

Jestli že výstupní data jedné instrukce jsou použity jako vstupní hodnoty do instrukce následující(řeťezový výpočet), musí si být programátor jist, že se jedná o kompatibilní datové typy. Pokud tomu tak není, musí programátor explicitně zajistit konverzi mezi příslušnými datovými typy.

Volání bloků: STL/LAD/FBD Editor kontroluje kompatibilitu předávaných dat. Datové typy aktuálních parametrů musí být kompatibilní s datovými typy parametrů formálních, jinak není přiřazení možné a kompilátor (překladač) vyhlásí chybu.

**Deklarace datových typů**

Proměnné použité v uživatelském programu STEPu 7 a jejich adresy jsou přiřazeny datovým typům v některém z těchto míst :

- globální tabulka symbolických názvů programu
- deklarační tabulka logického bloku
- zobrazení deklarací při tvorbě globálního datového bloku.

## Základní datové typy

| Klíčové slovo | Šířka (v bitech) | Příklad konstanty |
|---|---|---|
| BOOL<br>BYTE<br>WORD<br>DWORD<br>CHAR | 1<br>8<br>16<br>32<br>8 | 1 nebo 0<br>B#16#A9<br>W#16#12AF<br>DW#16#ADAC1EF5<br>' w ' |
| S5TIME | 16 | S5T#5s_200ms |
| INT<br>DINT<br>REAL | 16<br>32<br>32 | 123<br>65539 nebo L#-123<br>1.2 nebo 34.5E-12 |
| TIME<br>DATE<br>TIME_OF_DAY | 32<br>16<br>32 | T#2D_1H_3M_45S_12_MS<br>D#1993-01-20<br>TOD#12:23:45.12 |






**BOOL, BYTE, WORD DWORD, CHAR** Proměnné datového typu BOOL reprezentují bit, proměnná typu BYTE, WORD, a DWORD je tvořena sekvencí 8, 16 nebo 32 bitů. V případě těchto datových typů není možné individuální vyhodnocování hodnot jednotlivých bitů.

Zvláštní variantou jsou BCD čísla a proměnná čítače *count* (hodnota), použitá při definici čítače, stejně jako datový typ CHAR reprezentující znak v ASCII reprezentaci (viz: ST7-PRO1).

**INT, DINT, REAL** Proměnné těchto datových typů reprezentují čísla a s nimi definované aritmetické instrukce (viz.: ST7-PRO1 a kap. 3).

**S5TIME** Proměnná datového typu S5TIME je použita ve spojení se specifickými časovými funkcemi. Časový údaj je zadán ve tvaru hodiny, minuty, sekundy a milisekundy. Interní reprezentace zabírá místo jako BCD číslo od 000 do 999 násobeno časovou základnou (0=10ms, 1=100ms, 2=1s, 3=10s), která je definována ve 4 nejvýznamnějších bitech.

Funkce FC 33 a FC40 z knihovny *IEC-Library* v *StdLib30* provádí převod mezi formáty S5TIME a TIME.

**DATE** Proměnná typu DATE je uložena ve slově jako celé číslo bez znaménka. Obsah proměnné odpovídá počtu dní od 01.01.1990 (tj.: D#2168-12-31 = W#16#FF62).

**TIME** Proměnná typu TIME zabírá v paměti dvojité slovo (*double word)*. Obsah proměnné je interpretován jako celé číslo na 32 bitech udávající počet milisekund a může být kladné i záporné (tj.: T#1s=L#1 000, T#24d20h31m23s647ms = L#214748647).

**TIME_OF_DAY** Také proměnná typu TIME_OF_DAY zabírá v paměti dvojité slovo. Obsahuje počet milisekund od počátku dne(0:00 hodin) jako celé číslo bez znaménka (tj.: TOD#23:59:59.999 = DW#16#0526_5B77).

## Komplexní datové typy

| Klíčové slovo | Šířka (v bitech) | Příklad |
|---|---|---|
| **DATE_AND_TIME** | 64 | DT#97-09-24-12:14:55.0 |
| **STRING**<br>(Řetězec max. 254 znaků) | 8 * (#znak +2) | 'This is a string'<br>'SIEMENS' |
| **ARRAY**<br>(Pole, skupina komponent stejného datového typu) | uživatelská definice | Mes_Val.: ARRAY[1..20] OF INT; |
| **STRUCT**<br>(Struktura, skupina komponent různého datového typu) | uživatelská definice | Motor: STRUCT<br>Speed: INT;<br>POWER: REAL;<br>END_STRUCT; |
| **UDT**<br>(Uživatelský datový typ, "vzor" ze základních nebo komplexních datových typů | uživatelská definice | Motor: STRUCT<br>Speed: INT;<br>POWER: REAL;)<br>END_STRUCT; |

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_05cz.8

Školicí středisko firmy E&A spol. s r.o.

**Komplexní datové typy**

Komplexní datové typy (pole a struktury) jsou výsledkem seskupení základních nebo komplexních datových typů.

Uživatel má možnost tvorby přizpůsobeného datového typu potřebného k řešení určitého problému.

Komplexní datové typy nelze zpracovávat celé najednou přímou instrukcí STEPu 7(jsou větší než 32 bitů), ale musí se zpracovávat část po části.

Komplexní datové typy jsou předdefinované tak, že délka typu DATE_AND_TIME je 64 bitů. Délka typu ARRAY , STRUCT a STRING je určena uživatelem.

Proměnné komplexních datových typů mohou být deklarovány pouze v globálních datových blocích nebo jako parametry nebo lokální proměnné logických bloků.

**Uživatelské datové typy**

Komplexní datové typy nemají vlastní identifikátor a nemohou být proto použity opakovaně při deklaraci proměnných.

Strukturovaný datový typ může být přiřazen vlastnímu „bloku“ (UDT1 ... UDT65535) s pomocí uživatelem definovaných datových typů.

Tyto „bloky“ potom mohou být použity podle potřeby v deklaraci části logického bloku nebo při tvorbě globálního datového bloku.

## Parametrické typy

| Parametr | Šířka (v bitech) | Popis |
|---|---|---|
| TIMER | 16 | definuje časovač, který bude použit volaným blokem<br>Formát: T1 |
| COUNTER | 16 | definuje čítač, který bude použit volaným blokem<br>Formát: C10 |
| BLOCK_FB<br><br>BLOCK_DB<br>BLOCK_SDB | 16<br>BLOCK_FC | definuje blok použitý volaným blokem<br>Formát: FC101, DB12 |
| POINTER | 48 | ukazatel obsahuje místo hodnoty adresu<br>Formát: P#M50.0 |
| ANY | 80 | tento typ se používá, je-li volba datového typu součástí volby parametru<br>Formát: P#M50.0 BYTE 10 |





**Parametrické typy**

Jako doplněk základních a komplexních datových typů, lze definovat také parametrické typy parametrů. Tyto formální operandy jsou potom zpracovány pomocí stejných instrukcí, jako odpovídající aktuální operandy.

Při volání tohoto parametrického bloku musí být potom tyto parametry přiřazeny aktuálním operandům.

**TIMER a COUNTER**

Tento parametrický typ definuje formální operand typu časovač (TIMER) a čítač (COUNTER).

**BLOCK_xx**

S pomocí parametrů typu BLOCK_FB nebo Block_FC lze volaným blokům předat jako parametr FB nebo FC blok, který bude při zpracování takto parametrizovaného bloku také zpracován. Jako aktuální operand lze v tomto případě předat pouze bloky, které neobsahují parametry nebo statické proměnné.

Takto předaný blok může být zpracován pouze pomocí instrukcí UC nebo CC (instrukci CALL nelze použít).

Pro parametricky předané datové nebo systémové datové bloky nejsou dána žádná omezení.

**POINTER**

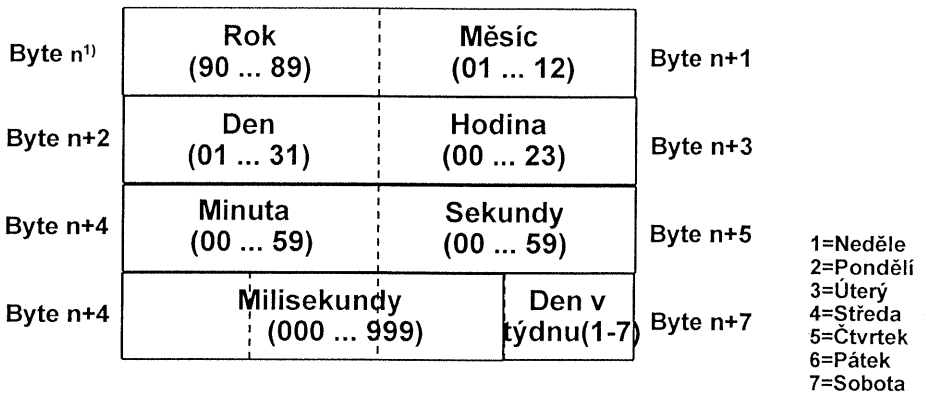
Parametr typu ukazatel neobsahuje hodnotu, ale adresu. Při přiřazení je tomuto parametru přiřazena jako aktuální operand adresa proměnná, která obsahuje požadovanou hodnotu. Např. P#M50.0. V tomto případě je hodnota uložena v paměti počínaje M-bitem M50.0.

**ANY**

ANY parametr se používá v případě, že součástí přiřazení aktuálního operandu je také datový typ aktuálního operandu. Např.P# M10.0 Byte 10. V tomto případě je jako aktuální operand předána oblast M-bitů MB 10 až MB 19.

## Datový typ: DATE_AND_TIME

- **Struktura:**




- **Všechny hodnoty jsou uloženy v BCD formátu**
- **Formát proměnné:**
  **DT#Rok-Měsíc-Den-Hodina:Minuta:Sekunda.[Milisekunda]**
  **Příklad: DT#1998-03-21-17:23:00:00**
- **Zpracování probíhá přes IEC-knihovnu funkcí**

[1] n = sudý

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_05cz.10

Školicí středisko
firmy E&A spol. s r.o.

**Přehled** Datový typ DATE_AND_TIME reprezentuje datum a čas v rámci dne. Mimo klíčového slova DATE_AND_TIME lze použít také zkrácenou formu zápisu DT. Obě klíčová slova mohou být zapsány také malými znaky.

**Přednastavení** Proměnou lze při deklaraci nastavit na počáteční hodnotu. Počáteční hodnota musí být ve formátu této proměnné typu DT. Při definici počáteční hodnoty lze počet milisekund vynechat.

**Zpracování** Proměnná typu DT může být zpracována pomocí symbolické nebo absolutní adresace s použitím odpovídajících funkcí v IEC-knihovně.

**Poznámka** Aktuální čas systémových hodin CPU lze číst pomocí SFC1 (READ_CLK). Načtená hodnota je ve formátu DATE_AND_TIME.

## Funkce pro zpracování proměnných typu DT

- IEC-knihovna v *StdLib30*
  - **FC1 (AD_DT_TM):** Funkce FC 1 přičte k proměnné typu DT časový interval ve formátu TIME a výsledek vrátí ve formátu DT.
  - **FC34 (SB_DT_DT):** Funkce FC 34 odečte dva objekty formátu DT a vrátí výsledek (časový interval) ve formátu TIME.
  - **FC35 (SB_DT_TM):** Funkce FC 35 odečte od objektu ve formátu DT časový interval formátu TIME a vrátí výsledek ve formátu DT.
  - **FC3 (D_TOD_DT):** Funkce FC 3 kombinuje proměnné DATE a TIME_OF_DAY (TOD) do formátu DATE_AND_TIME (DT).
  - **FC6 (DT_DATE):** Funkce FC 6 oddělí z proměnné DT datum DATE.
  - **FC7 (DT_DAY):** Funkce FC 7 vrací z proměnné DT den v týdnu.
  - **FC8 (DT_TOD):** Funkce FC 8 vrací z proměnné DT čas TIME_OF_DAY.
  - **Porovnávací funkce pro DT#proměnné:** FC9 (EQ_DT), FC12 (GE_DT), FC14 (GT_DT), FC18 (LE_DT), FC23 (LT_DT), FC28 (NE_DT)





**Přehled**

Při instalaci STEPu 7 je také nainstalována knihovna funkcí *StdLib30* s podknihovnou funkcí určenou pro zpracování IEC datových typů.

Obrázek ukazuje funkce pro zpracování proměnné typu DATE_AND_TIME.

**Poznámka k FC1, FC35**

Při použití FC1, FC3 a FC34 je nutné dodržet následující podmínky: Časový okamžik (parametr T) musí být v rozsahu DT#1990-01-01-00:00:00.000 až DT#2089-12-31-23:59:59.999. Funkce neprovádí žádnou kontrolu vstupní hodnoty.

Jestliže při sčítání nebo odečítání je výsledná hodnota mimo platný rozsah, je výsledek převeden do povoleného rozsahu a stavový bit *BR* je nastaven na hodnotu "0".

**FC34**

Hodnota proměnné musí být v rozsahu DT#1990-01-01-00:00:00.000 až DT#2089-12-31-23:59:59.999. Funkce neprovádí žádnou kontrolu vstupní hodnoty. Je-li první hodnota (parametr T1) větší (mladší) než druhá hodnota (parametr T2) je výsledná hodnota kladná. V opačném případě je výsledek záporný.

Je-li při odečítání výsledná hodnota mimo platný rozsah datového typu TIME, je výsledek opraven do rozsahu datového typu a stavový bit BR je nastaven na hodnotu 0.

**FC3, FC6, FC7, FC8**

Tyto funkce nesignalizují žádné chybové stavy. Uživatel je sám zodpovědný za to, že zadané vstupní parametry představují smysluplné hodnoty.

**Porovnávací funkce**

Porovnávací funkce také neprovádí vyhodnocení chyb. Výstupní parametr RET_VAL vrací pouze logický výsledek porovnání - platný (RET_VAL=TRUE), nebo neplatný (RET_VAL=FALSE).

## Datový typ: ARRAY

- **Proměnné typu pole ARRAY :**
  - **pevný počet prvků jednoho datového typu (základní, DT#,STRUCT, UDT)**
  - **Deklarace:**
    - jednorozměrné:
      *Jméno*: ARRAY[*minIndex..maxIndex*] OF *datový typ;*
    - vícerozměrné:
      *Jméno*:ARRAY[*mindex1..maxindex1,mindex2..maxindex2,...*] OF *datový typ;*
  - ***Index*: Datový typ INT (-32768...32767)**
- **Příklad:**
  - **Deklarace proměnné:**
    - jednorozměrné: Measured value: ARRAY[1..10] OF REAL;
    - vícerozměrné: *Result*: ARRAY[1..24,2..6] OF DINT;
  - **Přístup k proměnné:**
    - jednorozměrné.: L Measured value[5] // nahraj do ACCU1 prvek č. 5 z pole
      // "Measured value"
    - vícerozměrné: T Result[10,5]

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_05cz.12

Školicí středisko
firmy E&A spol s r.o.

**Přehled**

Datový typ ARRAY představuje pole s pevným počtem prvků stejného datového typu. Pole může být maximálně 6ti-rozměrné. Pro prvky datového typu ARRAY platí následující omezení:

- základní (žádná omezení)
- komplexní (pouze datový typ DATE_AND_TIME, STRUCT, UDT)
- žádné parametrické typy.

Pole nemohou být vnořovány. Limitní rozsah indexů je dán rozsahem datového typu INT, t.j., od -32768 do 32767.

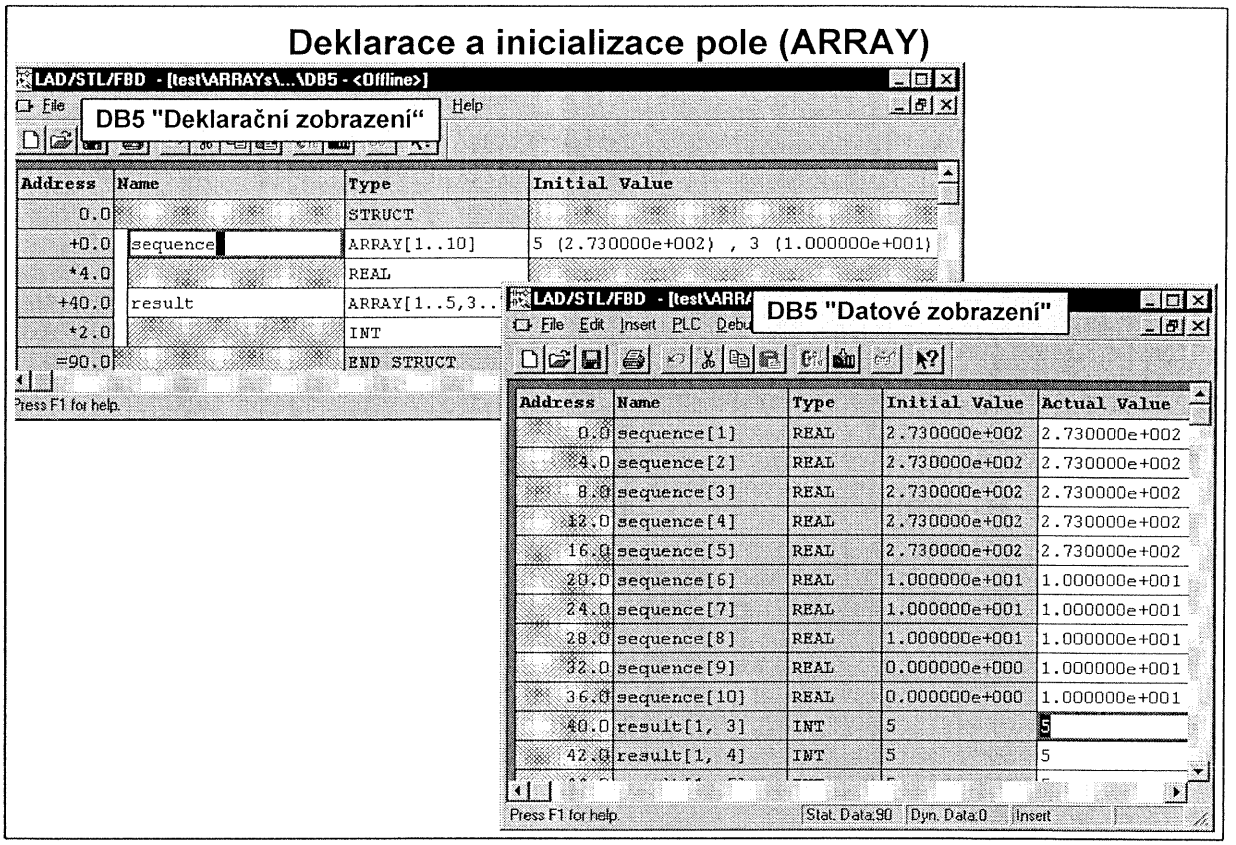
**Přístup**

STL instrukce mohou být použity k přístupu k prvkům pole základního datového typu. Prvek pole je adresován názvem pole a hodnotou indexu v hranatých závorkách.

Hodnota indexu musí být pevně daná konstantou. Variabilní adresování za běhu programu není v STL možné.

**Poznámka**

Variabilní adresování jednotlivých prvků pole je možné pouze s použitím S7-SCL. Variabilní adresování v STL může být implementováno pouze s použitím registrové nepřímé adresace.




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_05cz.13

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

V uvedeném příkladu, jsou v DB 5 deklarovány 2 pole ARRAY:

- sequence: ARRAY[1..10] OF REAL
- result: ARRAY[1..5,3..7] OF INT

**Inicializace polí ARRAY**

Jednotlivé prvky pole mohou být inicializovány na počáteční hodnotu.
(ne s FC parametry, in/out parametry FB nebo lokálními proměnnými).
Inicializační hodnota musí být kompatibilní s datovým typem prvku.
Tyto inicializační hodnoty, oddělené čárkou, jsou zadány ve sloupci *Initial Value*.
Jestliže má být inicializováno více prvků na stejnou hodnotu lze použít opakovací faktor. Opakovací faktor je umístěn za počáteční hodnotou v kulatých závorkách.

**Příklad**

5 (1.23467e+002) (následujících 5 prvků je inicializováno na hodnotu 123.467 )

5 (7,2,3) (následujících 15 prvků je inicializováno postupně s hodnotami 7, 2 a 3.)

Výsledek inicializace lze zkontrolovat nebo změnit v hodnotovém zobrazení datového bloku (*View -> Data View*). Je-li počet inicializačních hodnot menší než počet prvků, jsou zbývající prvky inicializovány na hodnotu 0.

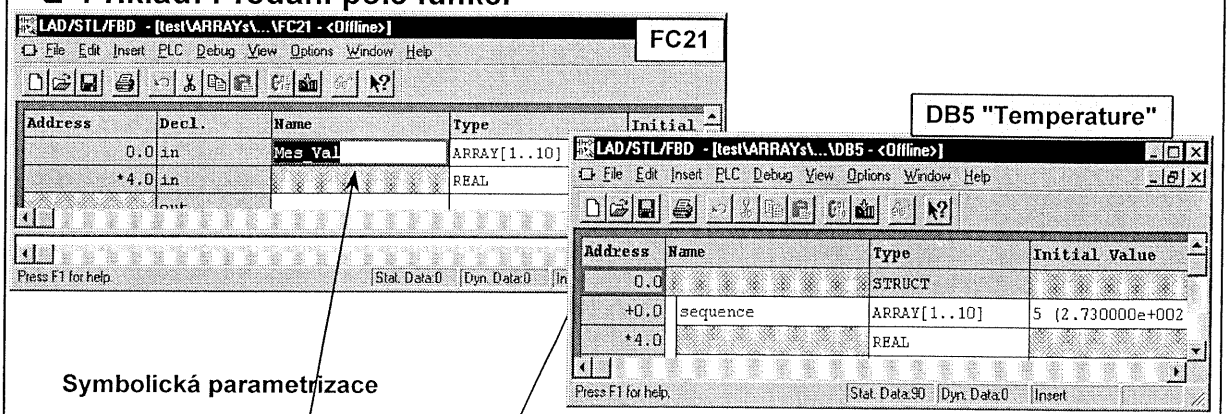
**Převzetí inicializačních hodnot**

Při inicializaci prvků v deklaračním zobrazení DB bloku jsou hodnoty převzaty do datového zobrazení.

V deklaraci vstupních a výstupních parametrů FB jsou hodnoty prvků pole akceptovány jako vstupní hodnoty při generování instančního DB.

# Předávání parametrů ARRAY

## ❑ Příklad: Předání pole funkci




**Symbolická parametrizace**

```
Network 1: Mes_Val je deklarována jako ARRAY v FC21

      CALL  FC    21
       Mes_Val:="Temperature".sequence
```

**Absolutní zobrazení**

```
Network 1: Mes_Val je deklarována jako ARRAY v FC21

      CALL  FC    21
       Mes_Val:=P#DB5.DBX0.0
```

**nelze zadat tímto způsobem**



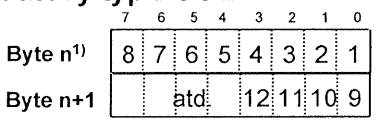

Datum: 24.11.2002
Soubor: PRO2_05cz.14



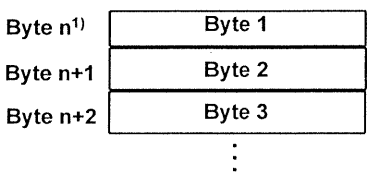
| | |
|---|---|
| **Přehled** | Pole lze předat jako kompletní proměnnou parametru bloku, jestliže je stejného datového typu nebo jedná-li se o datový typ POINTER nebo ANY. Stejně tak lze obsah pole zkopírovat do jiné cílové oblasti s pomocí např. systémové funkce SFC 20 BLKMOV.<br>Přiřadit parametru lze i jednotlivé prvky pole jestliže se jedná o identické datové typy. |
| **Proměnná pole** | Parametry pole lze předat také symbolicky. Aktuální parametr - pole musí mít v tomto případě stejnou strukturu (rozměr, datový typ prvků, atd.) jako formální operand - parametr.<br>Aktuální parametr - předávané pole může být alokován v datovém bloku (globálním nebo instančním) nebo v oblasti lokálních proměnných definovaných ve volajícím bloku. |
| **Poznámka** | Velikost použité paměti pro lokální proměnné lze určit přes *View -> Block Properties*. |
| **Interní předání parametru FC** | Vlastnímu předání parametru předchází kontrola typové kompatibility STL/LAD/FBD Editorem. Vlastní předání je provedeno *odkazem, tzn. při předání parametru je předán pouze ukazatel na počáteční proměnnou.* |

## Uložení ARRAY proměnných v paměti

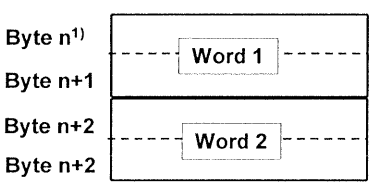
- jednorozměrné pole
  - datový typ BOOL




  - datový typ BYTE, CHAR

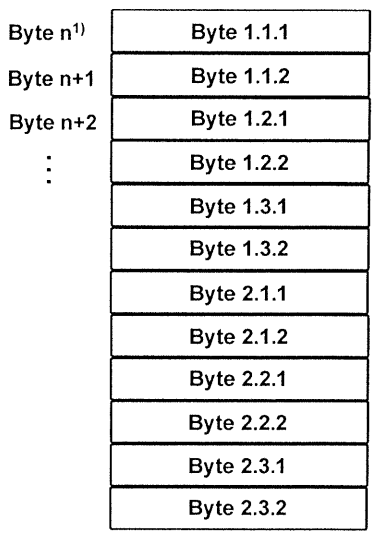



  - datový typ WORD, DWORD,...




1) n = sudé číslo

- vícerozměrné pole
  - Příklad.
    ARRAY[1..2,1..3,1..2] OF BYTE




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_05cz.15

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

Výše uvedený obrázek ukazuje uložení proměnné typu pole v paměti.

Přesná znalost uložení pole v paměti je nezbytně nutná pro případ nepřímé adresace jednotlivých prvků pole.

**Uložení proměnné**

Proměnná typu ARRAY vždy začíná na sudé adrese a zabírá celá paměťová slova.

Ve vícerozměrném poli jsou proměnné uloženy řádek po řádku počínaje první dimenzí. Následující dimenze typu bit nebo byte začíná na dalším bytu, ostatní datové typy začínají na dalším slově.

**Poznámka**

Adresy jednotlivých prvků pole v DB jsou zobrazeny v datovém zobrazení ve sloupci *Addressss*.

## Datový typ: STRUCT

- **Proměnná typu STRUCT (Structure):**
  - **pevný počet prvků různých datových typů (základní, DT#,ARRAYs, STRUCT, UDT)**
  - **Deklarace:**

```
StructName:    STRUCT
  Comp1Name: datový typ;
  Comp2Name: datový typ;
  ...
END_STRUCT
```

    - *StructName* je název proměnné typu STRUCT
    - *Comp1Name*, atd. jsou názvy jednotlivých prvků

- **Příklad:**
  - **Deklarace proměnné :**

```
MotorControl    : STRUCT
  ON            : BOOL;
  OFF           : BOOL;
  SetpointSpeed: INT;
  ActualSpeed : INT;
END_STRUCT;
```

**Přístup k proměnné:**

```
S MotorControl.ON
L MotorControl.ActualSpeed
T MotorControl.SetpointSpeed
    ...
```

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_05cz.16

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

Datový typ STRUCT (Structure) představuje pevný počet prvků, které mají ve stejném okamžiku různý datový typ. Strukturu lze vnořovat až do úrovně 8.

Struktura může být deklarována v deklarační části bloku, globálním nebo instančním datovém bloku nebo v uživatelem definovaném datovém typu (UDT).

Pro typ struktura platí následující omezení:

- základní datové typy (bez omezení)
- komplexní (pouze typy DATE_AND_TIME, ARRAY, STRUCT, UDT)
- nelze použít parametrický datový typ

**Přístup k prvkům**

K přístupu k základním typům prvků struktury lze použít STL instrukce. Jednotlivé prvky struktury jsou adresovány následně:

- *NázevStruktury.Názevprvku*

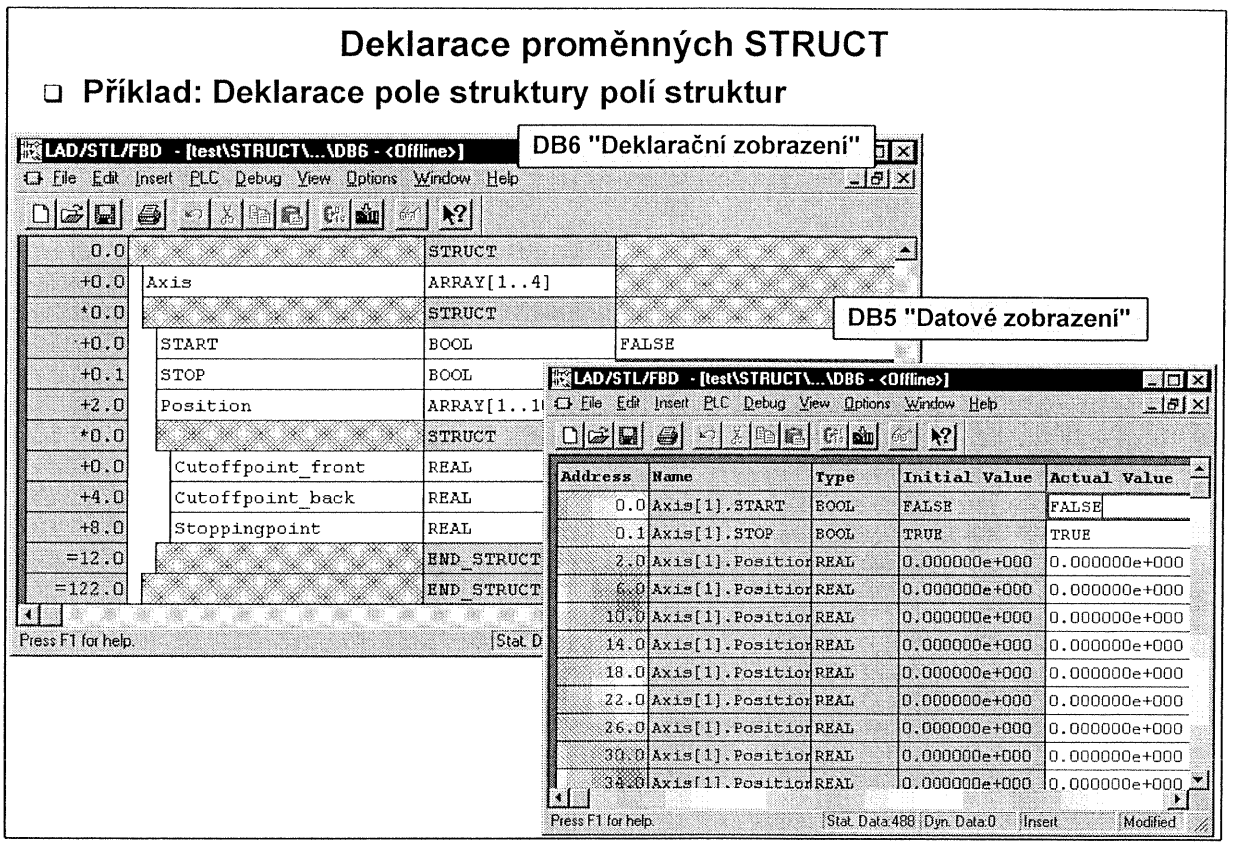
Mezi názvem struktury a názvem prvku musí být tečka.

Jestliže je použita vnořená struktura, musí být jednotlivé názvy struktury, substruktury a prvku substruktury odděleny tečkou:

- *StructureName.ComponentName.SubcomponentName. ...*

# Deklarace proměnných STRUCT

## ❑ Příklad: Deklarace pole struktury polí struktur




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_05cz.17

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

Uvedený příklad obsahuje jednorozměrné pole ARRAY [1..4] s prvky typu STRUCT v DB6 ("Turn_6") .

Struktura obsahuje 2 prvky datového typu BOOL ( "START" a "STOP"). Třetí prvek je komplexní datový typ ARRAY[1..10].

Tento prvek (ARRAY[1..10]) obsahuje prvky typu STRUCT s prvky typu REAL ("Cutoffpoint_front", "Cutoffpoint_ back" a "Stoppingpoint") .

**Přístup**

Jednotlivé prvky jsou adresovány např. následujícím způsobem:

- L "Turn_6".Axis[3].Position[7].Cutoffpoint_back
- S "Turn_6".Axis[2].START, atd.

**Inicializace STRUCT**

Jednotlivé prvky struktury mohou být inicializovány při deklaraci, ne však s FC parametry, in/out parametry FB nebo lokální proměnnou. Datový typ inicializační hodnoty musí být kompatibilní s datovým typem prvku.

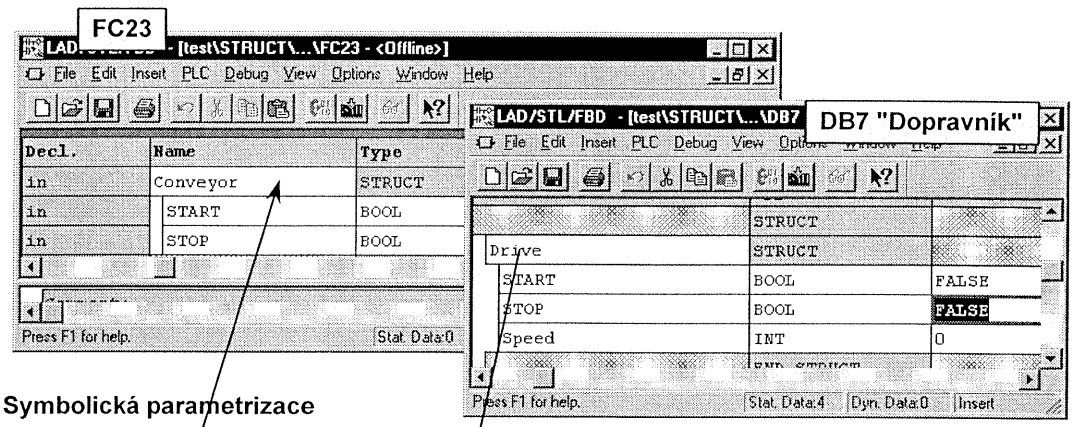
**Převzetí inicializační hodnoty**

Inicializační hodnoty zadané v deklaračním zobrazení DB jsou převzaty až po zvolení *Edit -> Initialize Data Block*

Při deklaraci vstupních nebo výstupních parametrů v FB jsou počáteční hodnoty struktury převzaty jako aktuální při generování instančního datového bloku.

## Předávání parametrů STRUCT

- **Příklad: předání STRUCT funkci**




**Symbolická parametrizace**

```
Network 1: Dopravník je deklarován jako STRUCT v FC23

      CALL  FC    21
       Conveyor:="Conveyor_belt".Drive
```

**Absolutní zobrazení**

```
Network 1: Dopravník je deklarován jako STRUCT v FC23

      CALL  FC    21
       Conveyor:=P#DB7.DBX0.0
```

**nepřípustný zápis**

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_05cz.18

Školicí středisko firmy E&A spol. s r.o.

**Přehled**

Struktura muže být předána jako kompletní proměnná parametru stejného typu nebo jako typ POINTER nebo ANY. Nebo lze kompletní obsah proměnné zkopírovat do jiné cílové oblasti např. s pomocí systémové funkce SFC 20 BLKMOV.

Jestliže je parametr stejného datového typu jako prvek struktury lze ho také předat jako parametr.

**Proměnná typu STRUCT**

Předání parametru, který je definován jako struktura lze provést pouze symbolicky. Je nezbytně nutné aby aktuální parametr byl deklarován jako datový typ deklarovaný v parametru.

Aktuální parametry, tj. proměnné typu struktura mohou být alokovány v oblasti DB (globálních nebo instančních) nebo v oblasti lokálních proměnných volajícího bloku.

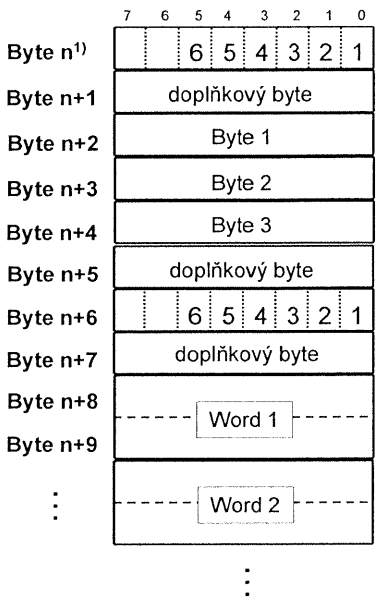
**Interní předání parametru FC**

Při psaní programu STL/LAD/FBD Editor zkontroluje kompatibilitu datového typu aktuální proměnné a parametru a předá pouze odkaz tj. ukazatel na aktuální proměnnou typu struktura.

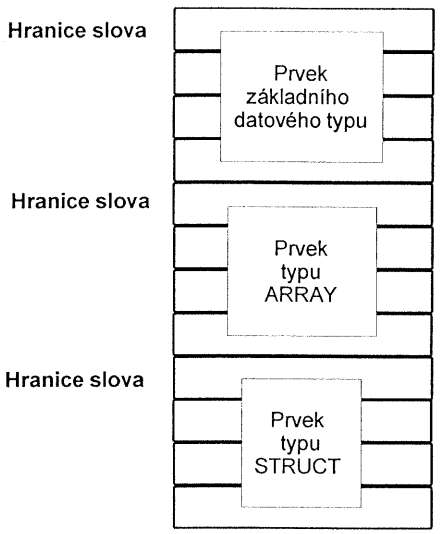
Tento odkaz je zobrazen při absolutním zobrazení programu, který je otevřen v „online" režimu *Accessible Nodes*. V tomto případě nejsou k dispozici informace nutné pro zpětný překlad programu do symbolické podoby.

# Uložení STRUCT proměnných v paměti

- **Struktura ze základních datových typů**




**Struktura z komplexních datových typů**




1) n = sudé

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_05cz.19

Školicí středisko firmy E&A spol. s r.o.

**Přehled**

Obrázek nahoře ukazuje detailní uložení proměnné typu STRUCT v paměti CPU.

Přesná znalost tohoto uložení je nezbytně nutná pro nepřímou adresaci jednotlivých prvků struktury.

**Uložení proměnné**

Proměnná typu STRUCT začíná vždy na začátku slova, tj. na bytu se sudou adresou. Jednotlivé prvky struktury jsou uloženy v pořadí deklarace ve struktuře. Proměnná typu STRUCT zabírá paměť po slovech.

Prvky typu BOOL začínají na bytu se sudou adresou od nejméně významného bitu, proměnné typu BYTE a CHAR začínají také na bytu se sudou adresou.

Prvky ostatních datových typů začínají na slově.

## Datový typ: STRING

- **Proměnné typu STRING (řetězec znaků):**
  - **Datový typ STRING představuje řetězec maximálně 254 znaků**
  - **Použití: zpracování textových hlášení**
  - **Deklarace:**
    - *StringName: STRING[maxNo]: 'Initialtizationtext'*
      (Textový řetězec s max. *maxNo* znaky, *maxNo*: 0... 254)
    - *StringName:* STRING: *'Initializationtext'*
      (Textový řetězec s max. 254 znaky)
- **Příklady:**
  - **Deklarace proměnné:**
    - Fault signal : STRING 'Chyba_motoru_4'
      (Proměnná *Fault signal* je inicializována s uvedeným textem)
    - Warning : STRING[50] ' '
      ("prázdná" proměnná *Warning,* může obsahovat až 50 znaků)
  - **Zpracování:**
    - základní přístup:
      L Fault signal[5] (nahraje 5. znak z *Fault signal*)
    - Zpracování funkcemi IEC-knihovny





**Přehled**

Datový typ STRING je určen k ukládání řetězce znaků tj. textových hlášení. Datový typ STRING představuje řetězec až 254 znaků.

Číslo v hranatých závorkách (1..254) při deklaraci udává maximální počet znaků, které mohou být uloženy v dané proměnné. Není-li tento údaj definován, předpokládá STL/LAD/FBD Editor délku 254 znaků.

**Přístup k STRING proměnné**

Jednotlivé znaky STRING proměnné jsou zpracovatelné pomocí základních instrukcí :

- L *StringName[5]*(nahraje 5. znak uložený v proměnné)

Zpracování proměnné STRING se jinak provádí pomocí funkcí IEC-knihovny.

**Inicializace**

Při deklaraci mohou být proměnné STRING inicializovány na počáteční hodnotu (ale ne jako parametry FC bloku, in/out parametry FB nebo lokální proměnné).

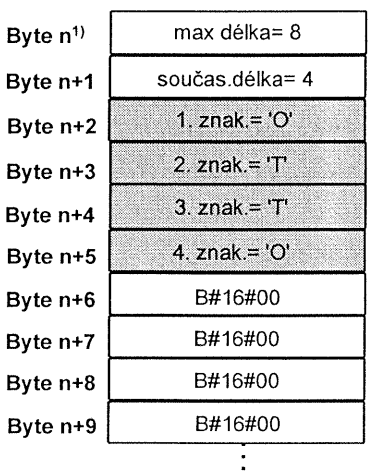
Inicializační hodnota v ASCII kódu je uzavřena v apostrofech. Při inicializaci na řídící znaky, jsou tyto znaky uvozeny dolarem.

Použitelné řídící znaky jsou:

- §§ dolar
- §L, §l line feed (LF)
- §P, §p page feed
- §R, §r carriage return
- §T, §t tabulator

## Uložení STRING proměnných v paměti

- **Příklad:**
  - **Deklarace s inicializací**
    - Given name: STRING[8]: 'OTTO'
  - **Uložení proměnné STRING "Given name"**




**Udává max. počet uložitelných znaků, t.j. délku zadanou při deklaraci**

**Udává okamžitý počet znaků proměnné STRING**

- **Maximální a okamžitá délka je vyhodnocována funkcemi IEC-knihovny.**

1) n = sudé

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_05cz.21

Školicí středisko firmy E&A spol. s r.o.

**Přehled**

Proměnná typu STRING je max. 256 bytů dlouhá, tj. může obsahovat max. 254 bytů uživatelských dat.

**Uložení proměnné**

STRING proměnná začíná vždy na slově se sudým počátečním bytem.

V prvním bytu je uložena maximální délka proměnné zadaná v deklaraci proměnné. V druhém bytu je uložena okamžitá délka , tj. počet znaků inicializačního řetězce.Obě tyto informace jsou nezbytně nutné pro zpracování textu pomocí funkcí IEC-knihovny.

Za okamžitou délkou následuje sequence ASCII znaků tvořící vlastní text. Neobsazené byty jsou při inicializaci přepsány hodnotou B#16#00.

**Předávání parametru**

Proměnná typu STRING může být předávána stejně jako typy ARRAY nebo STRUCT parametru stejného datového typu, tj. parametru STRING se stejnou délkou řetězce.

Parametrům FC nebo FB bloků lze STRING předávat také jako typ POINTER nebo ANY.

## Funkce pro zpracování proměnných typu STRING

- **IEC-knihovna v StdLib30**
  - **FC2 (CONCAT):** FC2 spojí dvě proměnné STRING do jednoho textového řetězce
  - **FC4 (DELETE):** FC 4 smaže v řetězci IN, L znaků počínaje pozicí P.
  - **FC11 (FIND):** FC 11 hledá v řetězci podřetězec znaků.
  - **FC17 (INSERT):** FC 17 vloží do textu IN2 od pozice P text IN1.
  - **FC20 (LEFT):** FC 20 vrací prvních L znaků textu.
  - **FC21 (LEN):** FC 21 vrací okamžitou délku, tj. skutečný počet znaků řetězce.
  - **FC26 (MID):** FC 26 vrací L znaků textu počínaje pozicí P.
  - **FC31 (REPLACE):** FC 31 nahradí L znaků textu IN1 od pozice P (včetně) textem IN2.
  - **FC32 (RIGHT):** FC 32 vrací posledních L znaků textu.
  - **Porovnávací funkce pro STRING proměnné:** FC10 (EQ_STRING), FC13 (GE_STRING), FC15 (GT_STRING), FC19 (LE_STRING), FC24 (LT_STRING), FC29 (NE_STRING)

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_05cz.22

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

Při instalaci STEPu 7 je spolu s knihovnou *StdLib30* instalována také IEC-knihovna pro zpracování IEC datových typů.

Výše uvedené funkce slouží ke zpracování proměnných typu STRING.

**Poznámka**

Obecně provádí funkce vyhodnocení chyby s použitím maximální a aktuální délky textového řetězce.Stejně tak obecně, pokud funkce vyhodnotí chybu, nastaví BR bit do nuly.

Detailní popis jednotlivých funkcí IEC-knihovny je v on-line nápovědě.

**Porovnávací funkce**

Porovnávací funkce provádí lexikografické porovnání znaků textového řetězce. Při porovnávání funkce porovnává ASCII kódy jednotlivých znaků zleva doprava.

První nalezený odlišný znak určuje výsledek porovnávání. Funkce nesignalizují jakoukoliv chybu. Vrací pouze stav porovnání - platný x neplatný.

## Uživatelsky definované datové typy: UDT

- **Použití UDT:**
  - **Tvorba nových datových typů s rozšířenými nebo upravenými vlastnostmi (např. inicializační hodnotou)**
  - **Výkonnější řešení automatizačních úloh**
  - **Globálně platné pro všechny bloky v daném programu**
  - **V často definovaných proměnných nebo datových blocích**
- **Příklady:**
  - **Definice nového datového typu (Structure):**

```
UDT5            STRUCT
   ON           : BOOL;
   OFF          : BOOL;
   SetpointSpeed: INT;          ...
   ActualSpeed  : INT;
END_STRUCT;
```

  - **Deklarace proměnné:**
    - Motor_1: UDT5;
      Motor_2: UDT5;
  - **Přístup k proměnné:**
    - L Motor_1.ActualSpeed





**Přehled**

Jestliže se v uživatelském programu často opakuje např. datová struktura, dovoluje STEP 7 pojmenovat tuto strukturu samostatným názvem - umožňuje vytvoření vlastního datového typu podobně jako např. v jazyce „C“ lze deklarovat typy s pomocí *typedef*.

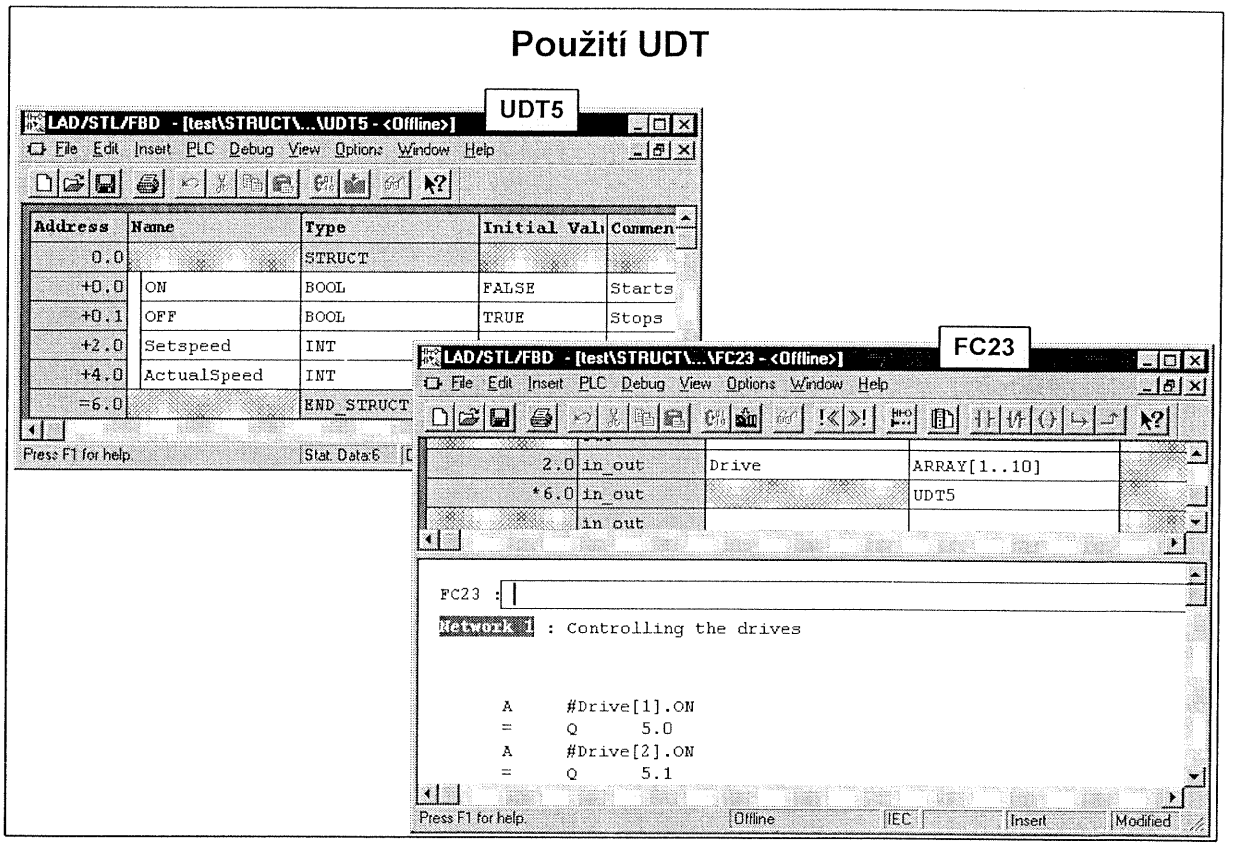
Uživatel tak může vytvořit datové typy, které umožňují snadnější a rychlejší řešení automatizační úlohy.

**Tvorba UDT**

UDT jsou vytvářeny v STL/FBD/LAD editoru v rozsahu UDT1 ... UDT65535.

UDT mohou mít také symbolický název definovaný v globální tabulce symbolických názvů.

# Použití UDT




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_05cz.24

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

V uvedeném příkladu je UDT5 tvořena 4 prvky (ON, OFF, SetpointSpeed a ActualSpeed) a je použita v FC23 při deklaraci in/out parametrů.

V FC 23 je deklarováno pole 10ti prvků typu UDT5.

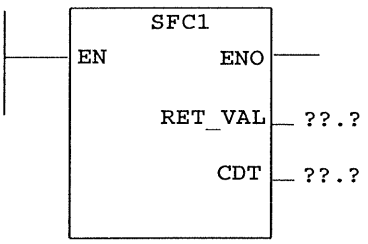
**Inicializace UDT**

UDT lze inicializovat a používat stejně jako typ STRUCT. UDT představuje vlastně šablonu pro deklaraci nových proměnných. Stejně jako u struktury lze definovat počáteční hodnoty v UDT. Je-li tato UDT potom použita k definici proměnné, jsou tyto inicializační hodnoty převzaty touto proměnnou( nelze však použít pro parametry FC, pro in/out parametry FB a lokální proměnné.

**Tvorba DB**

UDT lze použít také k vytvoření globálního DB bloku ( Dialog: *New Data Block*). V tomto případě bude mít DB stejnou strukturu a počáteční hodnoty jako UDT použitá při jeho tvorbě.

# Cvičení 5.1: Čtení Time-of-Day pomocí SFC 1 (READ_CLK)




| Parametr | Deklarace | Datový typ | Oblast | Popis |
|---|---|---|---|---|
| CDT | OUTPUT | DATE_AND_TIME (DT) | D, L | Aktuální datum a čas |
| RET_VAL | OUTPUT | INT | E, A, M, D, L | SFC Returnvalue – vracená hodnota |

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_05cz.25

Školicí středisko
firmy E&A spol. s r.o.

**Cíl cvičení:** Seznámit se s komplexním datovým typem DATE_AND_TIME a s IEC funkcemi pro zpracování proměnných tohoto typu.

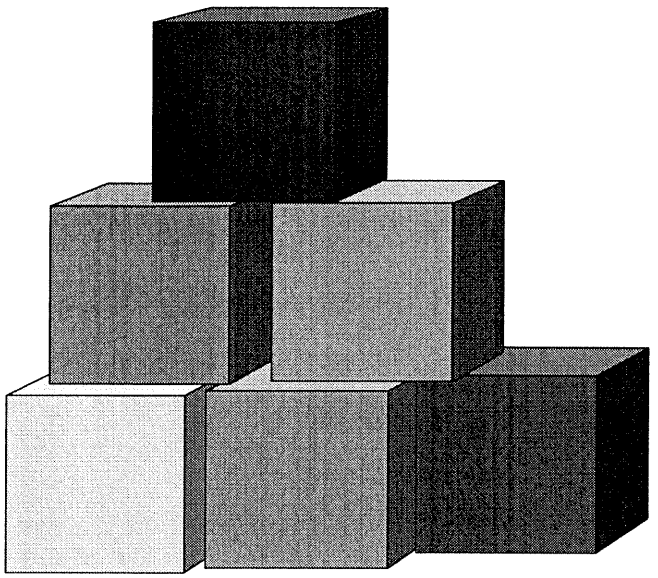
**Úkol:** Vytvořit FC51 s následující funkcí:

- FC51 načte aktuální denní čas pomocí SFC51 a zobrazí hodiny a minuty na digitální display.

**Provedení**

1. Vytvořit blok FC51 s popsanou funkcí
2. Vyvolat FC51 v OB1.
3. S pomocí funkce *PLC -> Set Time and Date* SIMATIC Manageru zkontrolovat nastavení hodin CPU
4. Nahrát program do CPU
5. Otestovat program.

# Volání bloků a multi-instanční model



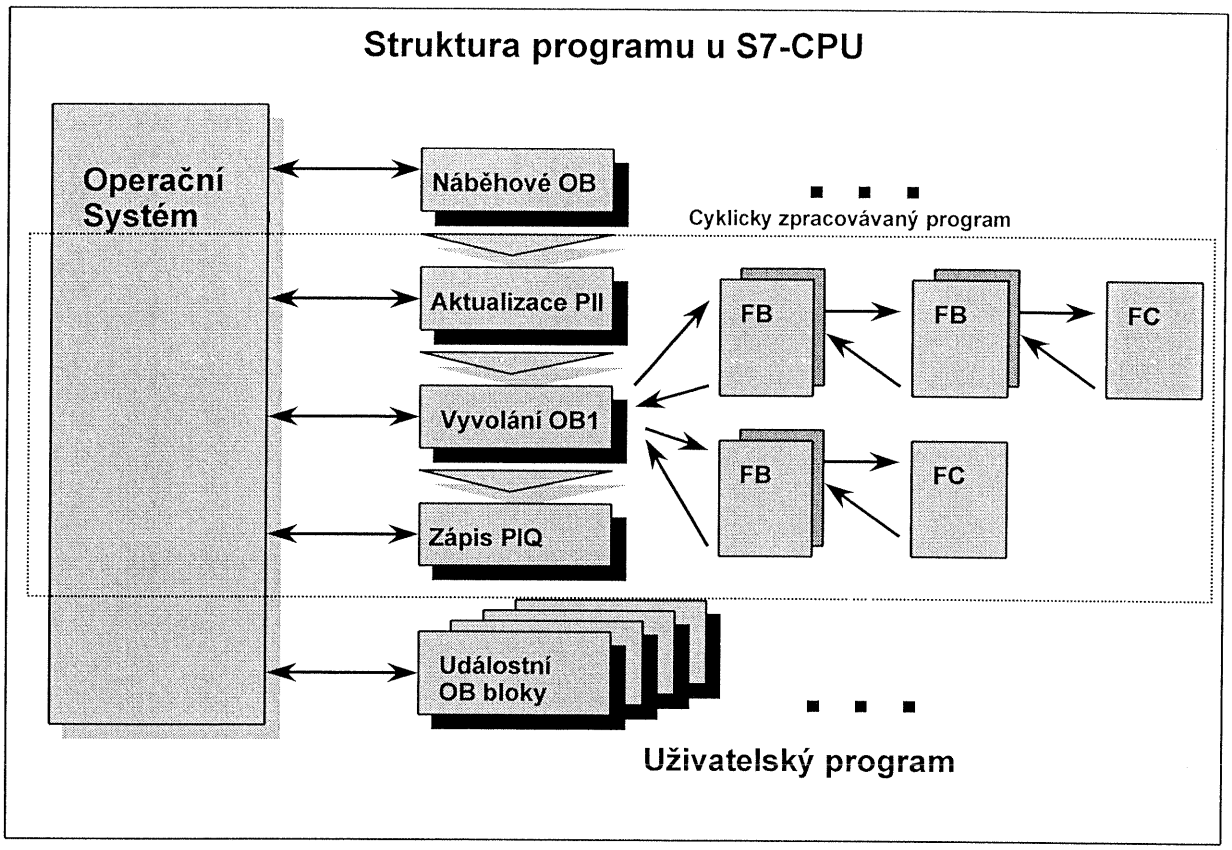
SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_06cz.1

Školicí středisko
firmy E&A spol. s r. o.

**Obsah** **Strana**




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_06cz.2

Školicí středisko
firmy E&A spol. s r. o.

**Přehled**

Řídící systémy S7-300/400 zpracovávaji informace z řízeného procesu. Tyto informace jsou „čteny" pomocí „vstupů", vyhodnoceny, zpracovány a výsledek je poskytnut procesu pomocí „výstupů".

**Programová struktura**

Program v CPU řídícího systému S7-300/400 je rozdělen na:

- operační systém
- uživatelský program

Organizační bloky (OB) tvoří rozhraní mezi operačním systémem a uživatelským programem.

**Operační systém**

Operační systém organizuje všechny funkce CPU a kroky, které nejsou spojeny s uživatelem definovaným úkolem. Jedná se o:

- kompletní náběh nebo náběh CPU,
- zpracování obrazu technologie (vstupů a výstupů),
- volání cyklického a událostních OB bloků,
- rozpoznávání chybových stavů
- komunikace např. s PG a se vstupy a výstupy řízeného procesu

Výkon operačního systému lze ovlivnit definování hodnot systémových parametrů. (tj. max. doba cyklu, priority událostních OB, atd.).

**Uživatelský program**

Uživatelský program lze vytvořit v programovacím prostředí STEP 7. Tento program umožňuje tvorbu jednotlivých částí - bloků, uživatelského řídícího algoritmu, který určuje chování řízeného procesu.

**Bloky**

Organizační bloky jsou volány operačním systémem v souvislosti s výskytem odpovídajících událostí. Všechny ostatní bloky ( FC, FB ) musí být volány z těchto organizačních bloků OB.

# Přehled bloků

| Typy Bloků | Vlastnosti |
|---|---|
| Organizační blok (**OB**) | - uživatelské rozhraní pro operační systém<br>- vzrůstající priority (0...27)<br>- specifické náběhové informace v *L-STACKu* |
| Funkční blok (**FB**) | - možnost definice parametrů<br>- s vlastní pamětí |
| Funkce (**FC**) | - možnost definice parametrů (parametry musí být přiřazeny při volání bloku)<br>- vrací výsledek algoritmu<br>- bez vlastní paměti |
| Datový blok (**DB**) | - strukturovaná oblast pro ukládání dat (Instanční DB)<br>- strukturovaná oblast pro ukládání dat (sdílená) (platná pro celý program) |
| Systémový funkční blok (**SFB**) | - FB (s pamětí) uložený v operačním systému a použitelný uživatelem |
| Systémová funkce (**SFC**) | - funkce (bez paměti) uložená v operačním systému použitelná uživatelem |
| Systémový datový blok (**SDB**) | - Datový blok pro konfigurační data a parametry |





**Úvod**

Komplexní automatizační úlohu lze snadněji členit, je-li rozdělena na jednodušší úlohy představované technologickými funkcemi systému nebo procesu.

**Třídy bloků**

Bloky jsou stavební částí uživatelského programu. Podle obsahu je lze rozdělit do dvou tříd:

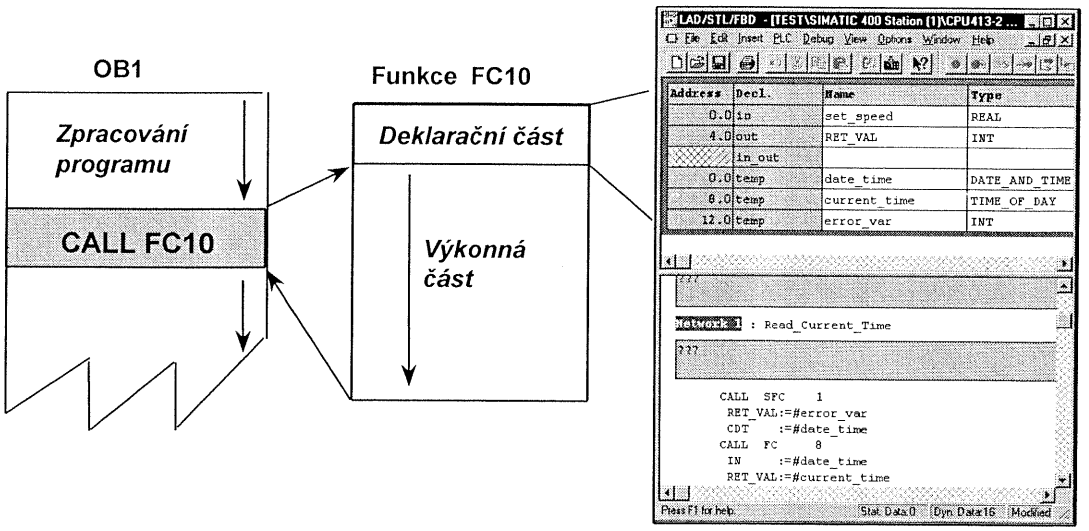
- logické bloky (OB, FB, FC, SFB a SFC) - jsou bloky , ve kterých je uložen uživatelský Step 7 program.
- datové bloky (DB a SDB) - jsou určeny pro ukládání hodnot.

**Použití**

Rozdělení uživatelského programu na jednotlivé bloky má následující výhody:

- ucelená automatizační úloha může být rozdělena na menší, jednodušší úlohy
- parametrické bloky jako FB, SFB, FC a SFC lze použít vícenásobně, stačí pouze definovat odlišnou sadu aktuálních operandů.
- zvláštní bloky (FB a SFB) mají "paměť" a jsou použitelné zvláště pro složité a náročné výpočty.
- bloky lze editovat, nahrávat a testovat jednotlivě. Touto cestou lze měnit jednotlivé části programu blok po bloku během zpracování programu CPU.
- pro mnoho úloh jsou vytvořeny skupiny bloků, uložené ve standardní knihovně a stačí je podle potřeby pouze připojit do uživatelského programu a použít.
- bloky pro často používané standardní úlohy lze integrovat ( dodavatelem operačního systému) jako systémové funkční bloky (SFB) nebo jako systémové funkce (SFC) do operačního systému CPU.

# Volání funkcí (FC)







Datum: 24.11.2002
Soubor: PRO2_06cz.4



**Přehled**

Funkce jsou části uživatelského programu a v souladu s normou IEC-Standard 1131-3 představují bloky bez paměti. Funkce jsou logické bloky se vstupními, výstupními nebo vstupně/výstupními parametry.

Funkce nemají paměť, neexistuje žádná oblast, do které by bylo možné ukládat natrvalo výsledky operací. Dočasně platné výsledky operací je možné ukládat pouze do oblasti lokálních proměnných (L-Stacku).

**Volání FC**

Jestliže má být FC blok zpracován, musí být zavolán. Volání FC bloku se provádí instrukcí volání (CALL FC10) která současně vyvolá seznam parametrů.

Po zpracování instrukce volání, pokračuje CPU ve zpracování volaného bloku (např. FC10). Volaný blok je zpracován až do okamžiku, kdy CPU zpracuje instrukci BE, BEU nebo BEC.

Po zpracování některé z těchto instrukcí pokračuje CPU ve zpracování volajícího bloku (např. OB1) instrukcí, která následuje za instrukcí volání bloku CALL.

**Použití**

Funkce jsou používány pro realizaci „doplňkových" služeb, které vracejí výsledek volajícímu bloku.

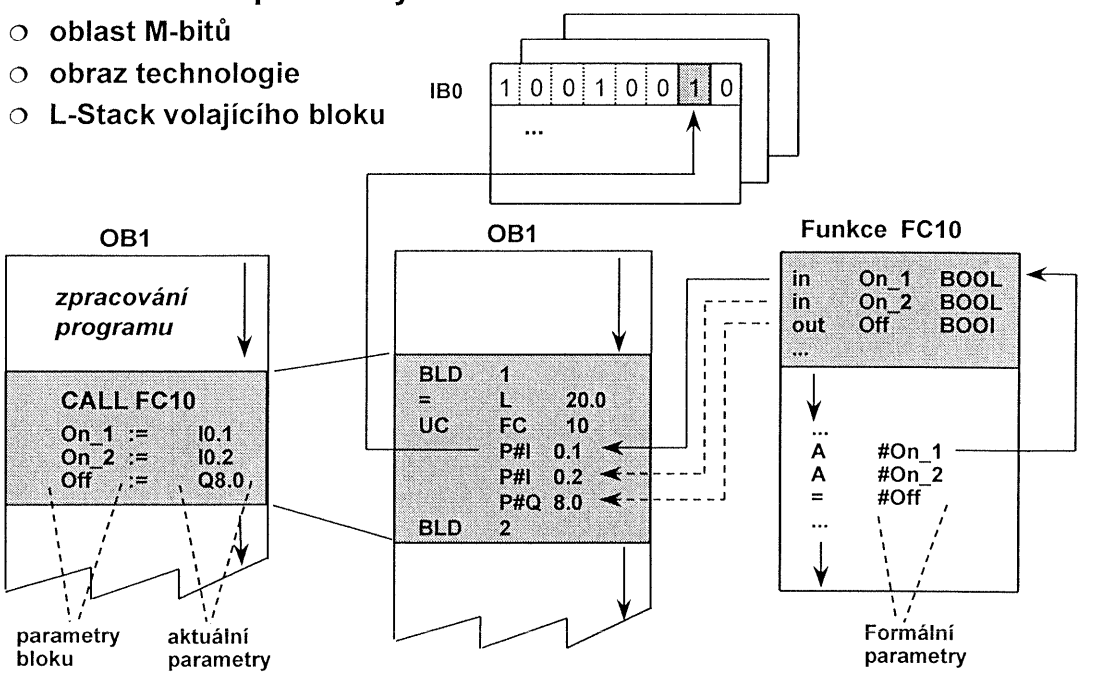
**Výstupní parametr RET_VAL**

Jestliže program odpovídá normě IEC, smí blok obsahovat pouze jeden výstupní parametr *RET_VAL*.

Při zadávání funkce v textovém režimu, kdy je datový typ přímo zadáván při definici funkce, např. *Function FC4: INT*. Není definice výstupního parametru *RET_VAL* nezbytná.

## Předávání parametrů při volání FC

- základní aktuální parametry:
  - oblast M-bitů
  - obraz technologie
  - L-Stack volajícího bloku







Datum: 24.11.2002
Soubor: PRO2_06cz.5



**Parametry FC bloku**

Data určená ke zpracování mohou být předána volané funkci. Tato data se předávají přes seznam parametrů, jejichž jména a datový typ je definován při tvorbě bloku v deklarační části bloku.

V deklarační části bloku lze definovat vstupní (*decl. in,* pouze pro čtení), výstupní (*decl. out,* pouze pro zápis) a vstupně/výstupní (*decl. in_out,* pro čtení i zápis)parametry.

Počet parametrů je omezen pouze velikostí paměti. Jméno parametru smí obsahovat maximálně 24 znaků. Parametry může doprovázet detailnější komentář.

**Mechanismus předání**

Při volání bloku CALL, STL/LAD/FBD Editor nejprve určí hodnotu ukazatele s definicí oblasti pro aktuální parametry a uloží ji.

Jestliže je uvnitř FC použit formální operand (např: *A On_1*), určí CPU z B-Stacku místo volání bloku. Ze seznamu parametrů parametrů je určen odpovídající operand a hodnota ukazatele s udáním oblasti na aktuální operand. Blok má přístup k hodnotě aktuálního operandu pomocí tohoto ukazatele.

Tento mechanismus odpovídá „předávání odkazem".

Tento mechanismus předávání přes ukazatel má následující důsledky:

- všechny parametry bloku musí být při volání bloku přiřazeny aktuálním operandům.
- parametry bloku nemohou být při deklaraci inicializovány.

**Poznámka**

Je-li parametru bloku přiřazen operand z oblasti DB nebo komplexní aktuální operand, je předání hodnoty parametru složitější.

## Ostatní instrukce volání bloků

- **instrukce CALL**
  - **instrukce je makro**
    - obsah registrů může být přepsán, včetně DB registrů
    - kolize s obsahem B-Stacku
    - po volání je otevřen chybný DB blok
    - doba zpracování CALL není přesně zjistitelná
  - **správné předání aktuálních operandů parametrům bloku**
  - **příklad:**
    - CALL FC10<br>On_1 := I0.1<br>On_2 := I0.2<br>Off := Q8.0
- **instrukce UC a CC**
  - **na RLO nezávislé (UC) nebo závislé (CC) volání bloku**
    - příklad: UC FC20 nebo CC FC20
  - **použitelné pouze s FC bez parametrů**

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_06cz.6

Školicí středisko
firmy E&A spol. s r. o.

**Instrukce CALL**

Instrukci (macro) CALL lze použít k volání bloků FC, SFC, FB a SFB.

Pouze instrukce CALL umožňuje přímou výměnu dat - informací mezi volajícím a volaným blokem. Zajišťuje také správné přiřazení aktuálních operandů odpovídajícím formálním parametrům.

Z faktu, že instrukce CALL je macro, skládající se z několika STL instrukcí STEPu 7, vyplývá několik zvláštních vlastností.

Jestliže je formální parametr přiřazen aktuálnímu operandu, který leží v oblasti DB, je předání parametru provedeno s pomocí DB registru. Proto :

- DB platný uvnitř volaného FC není DB platný před provedením volání CALL.
- jestliže při zpracování volaného FC bloku přejde CPU do STOP stavu, potom je hodnota DB registru zobrazená v B-Stacku přepsána na hodnotu potřebnou pro předání parametrů.

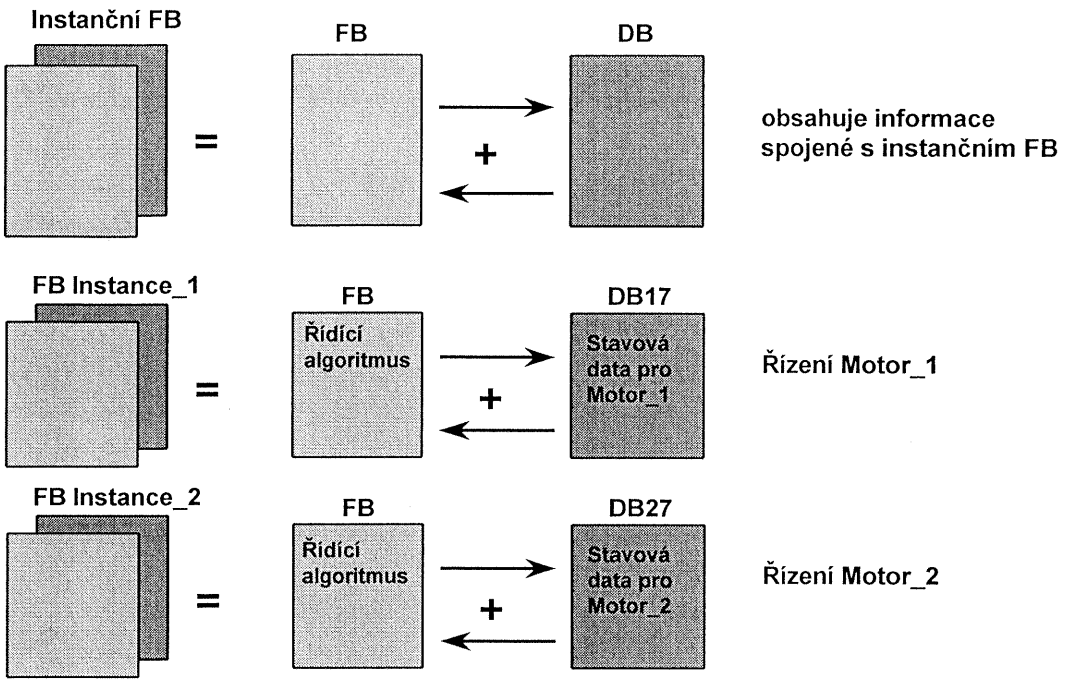
**Instrukce UC, CC**

Volání bloků lze provést také pomocí instrukcí UC a CC. Instrukce UC volá blok nepodmíněně, tj. bez ohledu na vnější podmínky. Instrukce CC volá blok podmíněně s ohledem na hodnotu stavového bitu RLO. Je-li hodnota tohoto bitu „1", CPU provede volání a zpracuje volaný blok. Je-li hodnota bitu „0" volání se neprovede a program pokračuje instrukcí, která následuje za instrukcí volání (CC).

**Důležité**

Instrukce UC a CC lze použít pouze k **volání** bloků **bez parametrů.**

## Instanční model funkčních bloků (FB)




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_06cz.7

Školící středisko
firmy E&A spol. s r. o.

**Přehled**

Funkční bloky jsou částí uživatelského programu a v souladu s normou IEC Standard 1131-3 představují logické bloky s pamětí. Mohou být volány z bloků typu OB, FB nebo FC.

Na rozdíl od FC bloků mohou FB bloky vystupovat jako instanční bloky, tj. bloky s pamětí vyhrazenou pouze pro jejich potřebu. V jednoduché formě (*single-instance mode)* je volanému FB bloku vyhrazen jeden DB blok. Tento vyhrazený blok je nutné specifikovat explicitně při každém volání FB bloku.

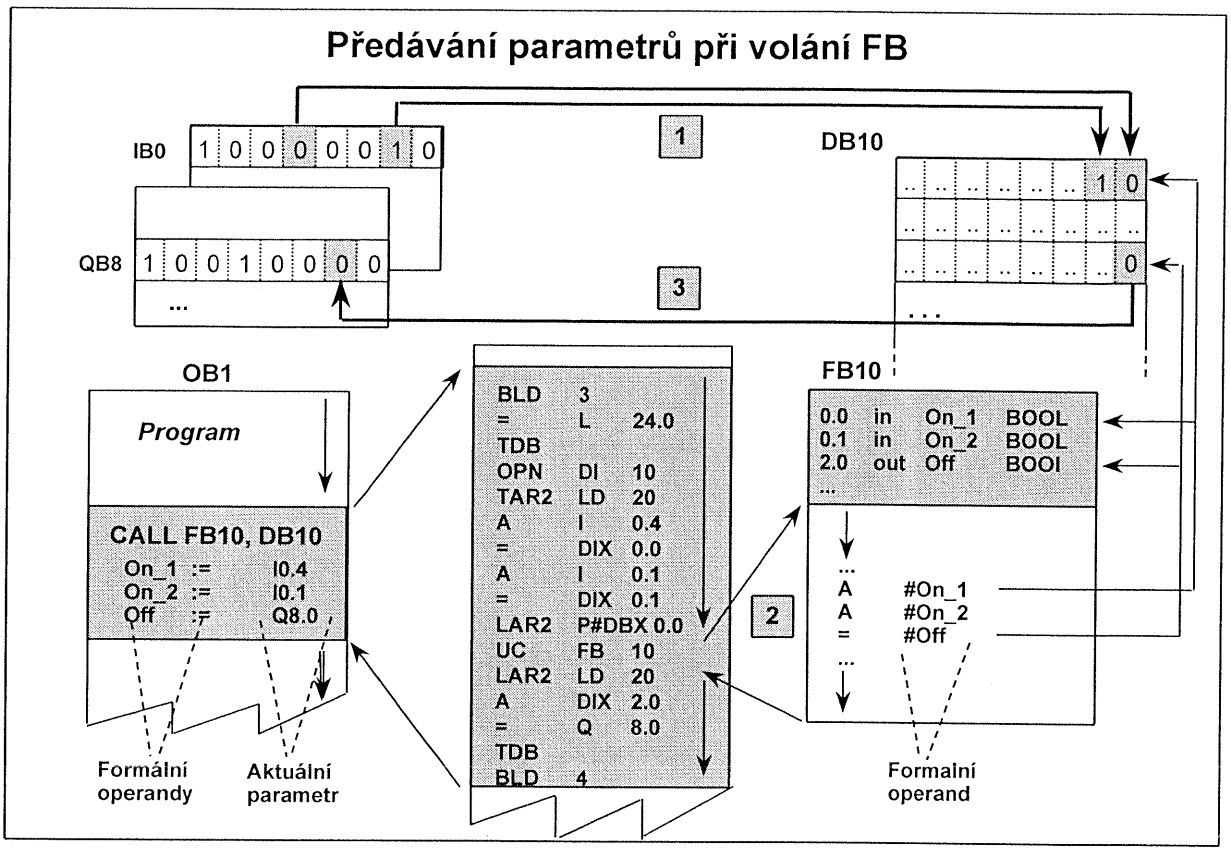
**Co je Instance?**

Možnost vytvářet instanční volání FB bloku je hlavním odlišujícím prvkem od FC bloků. FB blok se statickými parametry představuje obecný algoritmus, který realizuje libovolnou řídící úlohu. Aplikace tohoto algoritmu - FB bloku na konkrétní část technologie je *instancí* - konkrétním případem. Statické proměnné definované při deklaraci algoritmu - FB bloku existují pouze v daném konkrétním případě aplikace FB, jsou to instanční proměnné.

Podobně jako proměnné mohou být jako instanční použity i celé FB bloky - multi-instanční model volání FB bloků. Tento model umožňuje uchování instančních proměnných z vícero volání v jednom datovém bloku multi-instančním DB.

**Použití**

Na rozdíl od FC mají instanční FB paměť. Tato paměť slouží např. k uchování hodnot čitačů, časovačů, výsledků zpracování v minulých cyklech.




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_06cz.8

Školicí středisko
firmy E&A spol. s r. o.

**Parametr při volání FB**

V programu zpracovávané hodnoty lze předat volanému instančnímu FB bloku. Tyto hodnoty mohou být předány přes seznam parametrů zobrazený po zapsání instrukce CALL. Typ parametru (*in, out, in_out* ), jeho název a datový typ je definován při deklaraci parametru v deklarační tabulce bloku.

Na rozdíl od volání FC bloku, nemusí být volání instančního FB bloku vstupním, výstupním parametrům stejně tak jako vstupně/výstupním parametrům přiřazen aktuální operand.

**Předání parametrů**

Jestliže je DB blok vytvořen jako instanční k určitému FB bloku, STL/LAD/FBD Editor automaticky rezervuje oblast paměti pro parametry *(in, out, in_out)* bloku a pro statické proměnné definované v deklarační části bloku.

Struktura instančního DB s přiřazením proměnných je automaticky předdefinována v deklarační tabulce instančního FB bloku.

Při volání instančního FB bloku pomocí macro-instrukce CALL je instanční DB otevřen přes DI registr a aktuální hodnoty *in* a *in_out* parametrů jsou zkopírovány do proměnných v instančním DB, před vlastním voláním FB.

Je-li uvnitř FB požadován přístup k formálnímu operandu, má to za následek přístup k parametru, který náleží instančnímu DB. Tento přístup je interně proveden přes registry DI a AR2.

Po zpracování FB jsou hodnoty formálních *out* a *in_out* parametrů zkopírovány do aktuálních parametrů definovaných ve volání bloku instrukcí CALL. Teprve potom pokračuje zpracování programu instrukcí ležící za instrukcí CALL.

## Zvláštnosti CALL při volání FB

- **Předávání parametru hodnotou (kopírováním hodnoty):**
  - **Inicializace:**
    - Parametry FB bloku mohou být inicializovány při deklaraci
    - Vyjímka: vstupní/výstupní parametry komplexního datového typu (STRUCT, ARRAY, STRING a DATE_AND_TIME)
  - **Přiřazení FB parametrů při volání:**
    - FB parametry nemusí být přiřazeny
    - Přístup lze realizovat "zvenku" (z volajícího bloku)
    - Vyjímka: vstupní/výstupní parametry komplexního datového typu (STRUCT, ARRAY, STRING a DATE_AND_TIME)
- **Přístup k formálním parametrům je proveden přes registry DI a AR2**
  - **Přepsání DI nebo AR2 registrů:**

    ➡ **přístup k instančním proměnným již není možný**

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_06cz.9

Školicí středisko
firmy E&A spol. s r. o.

**Vyjímka**
U vstupních nebo výstupních parametrů komplexního typu (ARRAY, STRUCT, STRING, DT) není do instančního DB kopírována hodnota proměnné, ale ukazatel na aktuální parametr.

**Inicializace**
Parametry bloku a statické proměnné mohou být inicializovány při deklaraci v FB. Je-li potom generován instanční DB blok, jsou inicializační hodnoty zkopírovány do vytvořených instančních proměnných. Vyjímku tvoří parametry komplexního datového typu (viz. výjimka).

Instanční DB lze editovat stejně jako globální DB v režimu zobrazení "Data View".

**Přiřazení parametrů bloku**
Parametry bloku nemusí být při volání bloku přiřazeny. V tomto případě nejsou do nebo z instančního DB kopírovány žádné hodnoty. Parametry v instančním DB si zachovají hodnotu, která byla definována při posledním uložení.

Výjimka: při volání bloku musí být přiřazeny komplexní vstupní nebo výstupní parametry.

**Přístup "zvenku"**
K parametrům v instančním DB lze přistupovat stejným způsobem jako k parametrům v globálním DB.
To je zvlášť užitečné, jestliže mají být přiřazeny např. samotné prvky komplexního datového typu.

Vyjímka: Vstupní nebo výstupní parametry komplexního datového typu nemohou být přiřazeny "zvenku".

**Důležité**
Jestliže je během zpracování FB bloku přepsán obsah registrů DI a AR2, není možné dále přistupovat k instančním proměnným (*in, out, in/out* parametry a *stat* proměnné).

## Instrukce pro volání funkcí

- **Instrukce CALL**
  - **Instrukce je Macro**
    - Obsah registrů je přepsán, zvláště DB registrů
    - Po volání je otevřen jiný DB blok
    - Doba zpracování CALL není přesně určitelná
  - **Instrukce CALL zajišťuje správné přiřazení aktuálních dat parametrům bloku**
  - **Příklad:**
    - CALL FB10, DB10
      On_1 := I0.1
      On_2 := I0.2
      Off := Q8.0
- **Instrukce volání UC a CC**
  - **na RLO nezávislé (UC) nebo RLO podmíněné volání bloku**
    - Příklad: UC FB20 nebo CC FB20
  - **použitelné pouze pro FB bez instančních proměnných (tj. bez parametrů a statických proměnných)**

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_06cz.10

Školící středisko
firmy E&A spol. s r. o.

**Instrukce CALL**

Instrukci (Macro) CALL lze použít pro volání bloků FC, SFC, FB a SFB.

CALL zajišťuje správné přiřazení formálním operandům bloku. Při použití makro-instrukce CALL, která se skládá z několika instrukcí STEPu 7 je nutné vzít v úvahu několik zvláštností.

Uvnitř sekvence volání makra je pomocí instrukce CDB zaměněn obsah registrů DB a DI :

- uvnitř volané instance FB je otevřený, platný DB blok jiný, než který byl otevřen před voláním CALL.
- jestliže při zpracování instance FB přejde CPU do STOP stavu, je v B-STACKu v DB registru zobrazena hodnota, která byla původně uložena v registru DI.
- po ukončení zpracování bloku a návratu do volajícího bloku již není otevřen datový blok DB, který byl otevřen před zpracováním instančního FB bloku.

**Instrukce UC, CC**

Volání bloků lze provést také pomocí instrukcí UC a CC.

Instrukce UC provádí volání bloku nepodmíněně, tj. bez ohledu na hodnotu stavového bitu RLO. Blok volaný pomocí této instrukce je zpracován vždy.

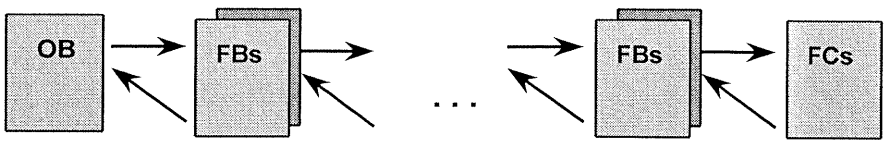
Instrukce CC volá blok podmíněně, tj. s ohledem na hodnotu stavového bitu RLO. Volání - zpracování bloku se provede pouze tehdy, je-li hodnota stavového bitu RLO = 1. Je-li bit RLO = 0, volání se neprovede, stavový bit RLO se nastaví do 1 a pokračuje se zpracováním instrukce, která leží za instrukcí CC.

**Důležité**

Instrukce UC a CC lze použít pouze k volání FB bloků bez parametrů a instančních proměnných, které se definují v deklarační tabulce bloku.

## Předávání parametrů "Passing on"

- **Hloubka vnoření:**
  - **S7-300: max. 8  S7-400: max. 16**




- **Předávání je závislé na typu bloku, dat a typu parametru:**

| Volání | FC volá FC | FB volá FC | FC volá FB | FB volá FB |
|---|---|---|---|---|
| Datový typ | E C P | E C P | E C P | E C P |
| Input -> Input | x - - | x x - | x - x | x x x |
| Output -> Output | x - - | x x - | x - - | x x - |
| In/out -> Input | x - - | x - - | x - - | x - - |
| In/out -> Output | x - - | x - - | x - - | x - - |
| In/out -> In/out | x - - | x - - | x - - | x - - |

E: Základní datový typ
C: Komplexní datový typ
P: Parametrický typ (Timer, Counter, Block_x)

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_06cz.11

Školicí středisko
firmy E&A spol. s r. o.

**Přehled**

Tento způsob předávání parametrů je zvláštní formou přístupu nebo přiřazení parametrů. "Passing on" znamená předání formálního operandu volajícího bloku jako aktuální operand volanému bloku.

**Omezení**

I zde platí obecné pravidlo, že aktuální operand musí být stejného typu jako formální operand. Dále platí, že vstupní parametry volajícího bloku mohou být použity pouze se vstupními parametry volaného bloku, výstupní parametry volajícího lze přiřadit pouze výstupním parametrům volaného bloku.
Vstupně/výstupní parametry volajícího bloku lze principielně přiřadit vstupním, výstupním i vstupně/výstupním parametrům volaného bloku.

**Omezení datových typů**

Při předávání datových typů, jsou omezení závislá na rozdílném uložení parametrů při volání FC nebo FB bloků.
Parametry základních datových typů lze předávat bez omezení. Vstupní a výstupní parametry komplexního datového typu lze předávat pouze volanému FB bloku.

Parametry parametrického typu: TIMER, COUNTER a BLOCK_x lze předat pouze ze vstupního parametru na vstupní parametr volaného FB bloku.

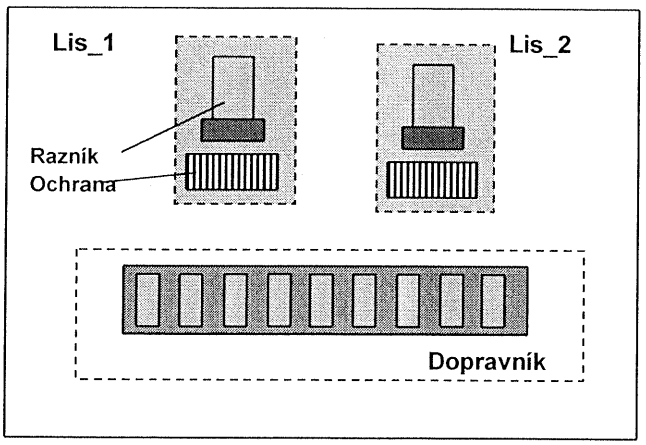
**Poznámka**

Předání parametrů typu TIMER, COUNTER a BLOCK_x lze u FC bloku realizovat s použitím nepřímé adresace. Číslo požadovaného TIMERu, COUNTERu nebo BLOCKu lze např. předat v proměnné typu WORD.

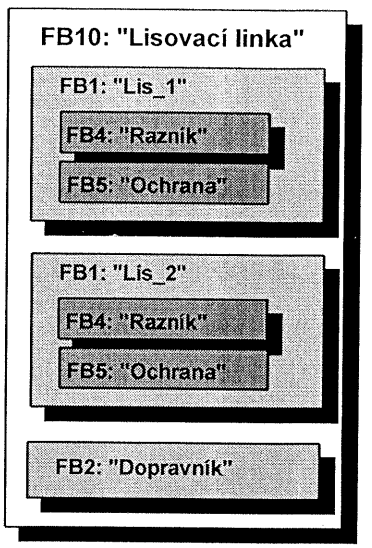
Uvnitř posledního volaného bloku lze hodnotu tohoto parametru uložit do lokální proměnné a přes tuto proměnnou zpracovat požadovaný TIMER, COUNTER nebo BLOCK.

## Multi-instanční model procesních jednotek

- Příklad: Lisovací linka




Technologické rozdělení




Struktura programu používajícího multi-instanční model





**Technologické části**

Technologické jednotky představují fyzické objekty - části technologie, např. dopravník, jednotlivá pracoviště nebo celé stroje nebo části strojů. Technologické jednotky jsou použity jako logické identifikační kriteria umožňující rozdělení řídícího algoritmu do jednotlivých bloků. S použitím těchto kriterií lze dosáhnout hierarchického uspořádání programu. Díky tomuto principu lze algoritmus pro řízení výše uvedené úlohy rozdělit na např. na jednotky "lis", které obsahují podjednotky "razník" a "ochrana".

Touto cestou lze technologické celky dělit na podcelky (agregace).

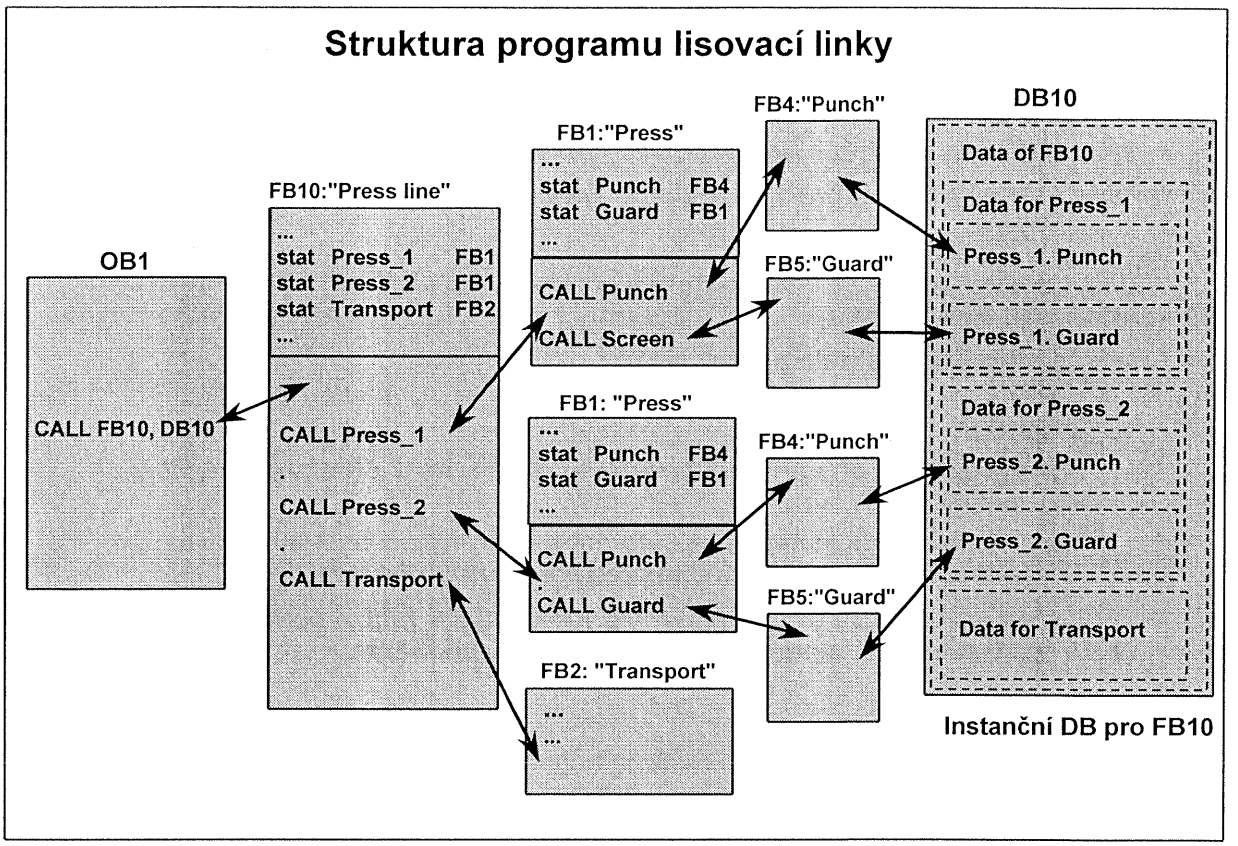
**"Multi-instance"**

Programovací jazyk STEP7 také podporuje princip "agregace" a znovu použití již jednou hotové části programu. Využívá k tomu FB bloky a myšlenku "multi-instance".
Popis technologických jednotek a podjednotek je programově řešen v S7 FB. Rozdělení jednotek na podjednotky je dosaženo deklarací podřízených FB bloků v části "stat Var" jako instancí uvnitř nadřízeného FB bloku.

**Opakované použití**

Opakované použití již hotové části programu je možné díky následující koncepci :

- Kdykoliv výrobce vytvoří procesní podjednotku (ventil, motor, válec, atd.), vytvoří současně také FB pro řízení této podjednotky.
- Současně se začleněním podjednotky do technologie, deklaruje v FB nadřízeného celku instanční FB pro řízení této podjednotky.




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_06cz.13

Školicí středisko
firmy E&A spol. s r. o.

**Multiinstanční model** Jestliže je použit multi-instanční model, obsahuje instanční DB data pro několik FB bloků volaných a deklarovaných uvnitř multi-instančního FB bloku.

Hierarchicky nejvyšší procesní jednotka (v tomto příkladu: FB10 "Press line") je volána absolutně nebo symbolicky s uvedením instančního DB (v tomto příkladu : DB10).

**Deklarace** V části "stat_var" v deklarační tabulce FB10 ("Press line") jsou definovány 2 instance (proměnné) datového typu "FB1" s názvy #Press_1 a #Press_2 a 1 instance datového typu "FB2" s názvem #Transport.

V deklarační části "FB1" jsou dále deklarovány instance typu "FB4" s názvem #Punch a typu "FB5" s názvem #Guard.

Takto deklarovaný instanční FB blok lze potom v programu volat jeho symbolickým názvem uvedeným v deklarační tabulce při definici instance.

**Poznámka** Definici instance v deklarační tabulce bloku lze provést pouze tehdy, jestliže FB blok, jehož instance se deklaruje, existuje. Při tvorbě jakéhokoliv programu ve STEPu7 platí zásada, že použít lze pouze to co existuje.

**Multi-instanční DB** Multi-instanční DB má strukturu složenou z instančních proměnných použitých v jednotlivých instančních FB. Volaná instance (FB) má automaticky přístup do odpovídající oblasti multi-instančního DB (DB10).

## Zvláštnosti multi-instančních modelů

- **Výhody multi-instančního modelu:**
  - **Pouze jeden DB blok je potřebný pro více instancí**
  - **Není nutná žádná další správa pro nastavení přístupu do privátní datové oblasti instance**
  - **Multi-instanční model umožňuje používat styl objektově-orientovaného programování**
  - **Maximální hloubka vnoření: 8**
- **Omezení:**
  - **Přímý (I, Q) přístup k procesním signálům není z instančního FB možný**
  - **Přístup k procesním signálům nebo komunikaci s jinými procesními jednotkami je možný pouze přes parametry FB bloku.**
  - **Stav - hodnoty proměnných lze v FB ukládat pouze do statických proměnných, ne do globálního DB nebo do M-bitů.**
- **Poznámka:**
  - **Přístup k instančním proměnným je možný také "zvenku" např.. v OB1: L "Press line".Press_2.Punch.<VarName>**







**Výhody**

Multi-instanční model umožňuje ukládat data více z instancí do jediného DB bloku. Přitom není nutná žádná další "údržba" těchto hodnot nebo přístupu k těmto proměnným.
Řídící program může mít strukturu, která odráží mechanickou konstrukci stroje.
STEP7 podporuje multi-instanční model až do 8 úrovní.

**Předpoklady**

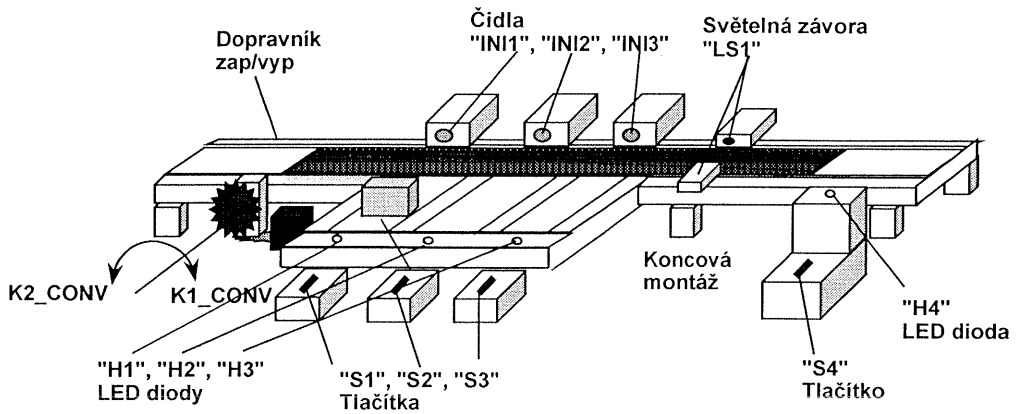
Pro bezproblémové použití multi-instančního FB je třeba dodržet několik zásad:

- Jakýkoliv přímý přístup k procesním signálům odporuje principu multi-instancí
- Komunikace s procesem nebo dalšími řídícími bloky musí probíhat pouze pomocí parametrů.
- Stavy nebo jiné informace o řízené jednotce je nutné ukládat pouze do statických proměnných definovaných v instančním FB bloku.

# Cvičení 6: Model dopravníku dokončovací linky

- **Činnost pracoviště**
  - zpracování dílu na pracovišti
  - dokončení, umístění na pás
  - čekání na další polotovar

**Činnost dopravníku**
- čekání na hotový díl
- doprava ke konečné montáži
- konečná montáž,založení polotovaru
- doprava na pracoviště




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_06cz.15

Školicí středisko
firmy E&A spol. s r. o.

**Cíl**

Model dopravníku je určen k praktickému použití FB v řídícím algoritmu.Pro řízení pracoviště je použito samostatné FB stejně tak jako pro řízení dopravního pásu. FB pro řízení pracoviště je navíc použito jako multi-instanční.

V následujícím cvičení je linka rozšířena o pracoviště 2 a 3 s použitím multi-instančního modelu.

**Princip funkce modelu**

Pro procvičení programování FB je stanovena následující funkce modelu dopravníku (obecně pro jedno z pracoviště):

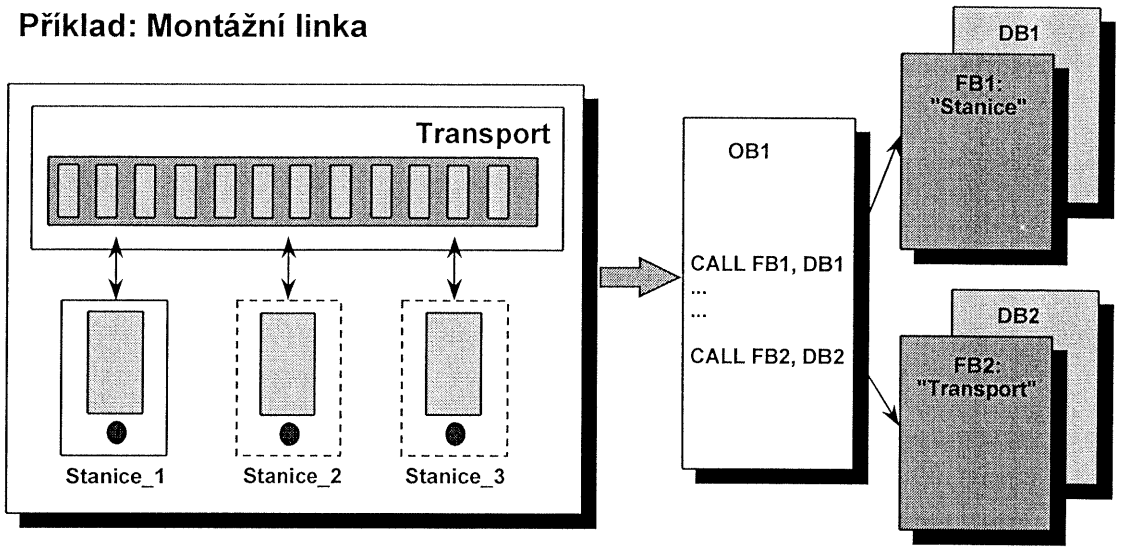
1. Systém je ve výchozím stavu, tj. pracoviště 1 zpracovává výrobek a tato situace je signalizována "H1" .

   Čidla "INI1" a "LS1" nejsou aktivována, motor pásu je vypnutý.
2. Po dokončení dílu na pracovišti operátor potvrdí dokončení operace tlačítkem "S1". Signalizace obsazení pracoviště "H1" následně zhasne. d.

   Po dokončení operace je díl umístěn operátorem na volné místo v pozici detekované "INI1".
3. Hotový díl je z pracoviště dopraven pásem na konečnou montáž v pozici detekované "LS1". Dosažení této pozice je signalizováno "H4".
4. Na stanici konečné montáže operátor odebere hotový díl a na jeho místo založí nový polotovar. Odebrání výrobku a založení polotovaru potvrdí operátor tlačítkem "S4", následně signalizace "H4" zhasne.
5. Po stisku tlačítka "S4" dopraví pás nový polotovar zpět na pracoviště 1. Dosažení pracoviště je detekováno čidlem "INI1" a signalizováno "H1".
6. Nový polotovar operátor převezme na na pracoviště na zahájí zpracování.
7. Pokračuje další cyklus popsaný v kroku 1.

# Cvičení 6.1: Programová struktura pracoviště

❑ **Příklad: Montážní linka**




**Technologické rozdělení montážní linky**

**Rozdělení programu podle technologie**




Datum: 24.11.2002
Soubor: PRO2_06cz.16



**Přehled**

STEP 7 nabízí dvě varianty tvorby instancí:

- Instanční FB je volán z různých míst v programu. Při každém volání je definován různý instanční DB blok.
- Instanční FB je volán několikrát z jednoho FB a pro uložení statických proměnných používá jeden DB blok společný pro všechny instance (multi-instanční model).

**1. krok**

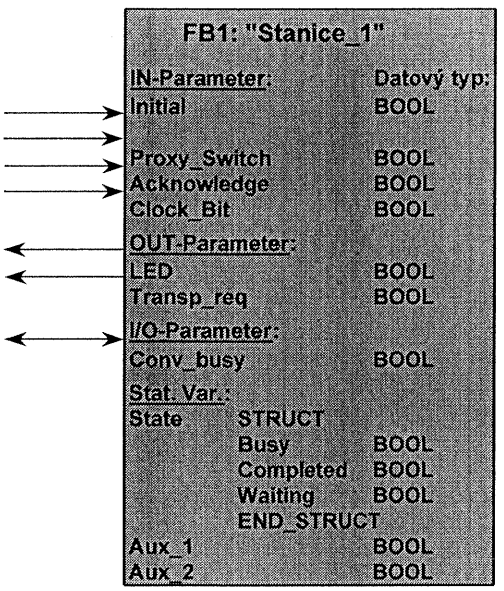
V tomto kroku je do programu začleněno pouze pracoviště 1. Řízení takovéto technologie je rozděleno na dvě části :

- Stanice: řízení jednoho pracoviště je realizováno v FB1 s globálním symbolickým názvem "Station" (globální tabulka symboliky)
- Transport: řízení dopravního pásu je realizováno v FB2 s globálním symbolickým názvem "Transport"

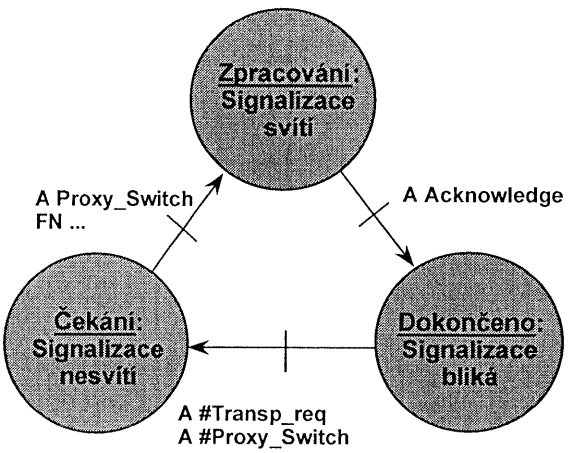
Pro dokončení řízení celé linky musí být oba FB bloky volány s příslušným instančním DB. (jednoduchý instanční model).

# Cvičení 6.1: Tvorba FB1 pro pracoviště

❑ **Rozhraní FB1 :**




❑ **Stavový model:**







Datum: 24.11.2002
Soubor: PRO2_06cz.17



**Přehled**

Nejprve je nutno vytvořit FB1, s řídící funkcí pro pracoviště. Proces je popsán pomocí jednoduchého stavového modelu se třemi stavy ("Zpracování", "Dokončeno" a "Čekání").

S ohledem na budoucí využití v multi-instančním modelu není možné přistupovat k signálům přímo. K procesním signálům lze přistupovat pouze přes parametry.

**Funkce FB1**

Blok FB1 popisuje reakci na příslušné kombinace stavů technologických signálů. Toto chování je popsáno v následující části.
Na dalších stránkách je popsán princip programování stavového modelu:

**"Zpracování"**

Pracoviště je ve stavu, kdy operátor zpracovává polotovar. Signalizace pracoviště "H1" trvale svítí. Pomocí výstupního parametru #Transp_req=0 signalizuje pracoviště, že je obsazeno a díl je zpracováván.

Přechodovou podmínkou do stavu "Dokončeno" je potvrzení operátora tlačítkem "S1".

**"Dokončeno"**

V tomto stavu - kdy pás je volný (Conv_busy=0) lze díl umístit na pás před čidlo "INI". Tím se nastaví *in/out* parametr #Belt_occupied=1 a také *out* parametr #Trans_req=1.

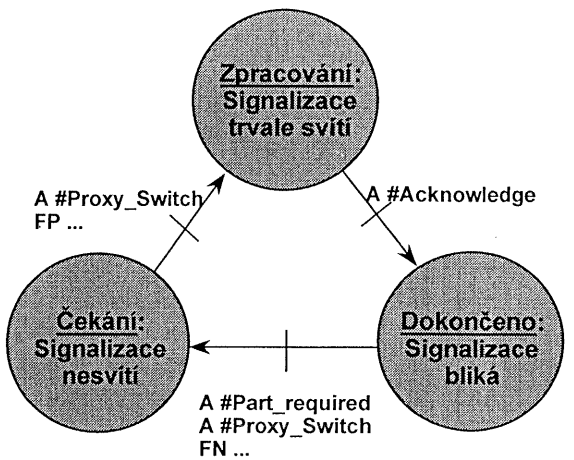
Tyto signály identifikují pás jako "obsazený" a žádná další stanice nesmí požadovat dopravu a na druhou stranu pás může být odstartován.

## Cvičení 6.1: Struktura FB1

### ❑ Struktura FB1 :

```
Network: 1   Initializing
        A    #Initial
        S    #State.Busy
        R    #State.Completed
        R    State.Waiting
        ...  // Set/Reset Parameters
Network 2    State: Busy
        AN   #State.Busy
        SPB  REDY
        ...  // Execution of Actions
        ...  // Scanning the progression conditions
        R    State.Busy
        S    State.Completed
Network 3    State: Completed
REDY:   AN State.Completed
        SPB  Waiting
        ...  // Execution of Actions
        ...  // Scanning the progression conditions
        R    State.Waiting
        S    State.Busy
Network: 4   State: Waiting
WAIT:   AN   #state.Waiting
        SPB  END
        ...  // Execution of Actions
        ...  // Scanning the progression conditions
        R    State.Waiting
        S    State.Busy
END:    BEU
```

### ❑ Stavový model:




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_06cz.18

Školicí středisko
firmy E&A spol. s r. o.

Dokud je stanice ve stavu "Dokončeno" signalizace bliká s frekvencí danou *in* parametrem #Clock_Bit.

Přechod do stavu "Čekání", je podmíněn spádovou hranou čidla "INI" při současném běhu pásu (#Transp_req=1).

**"Čekání"**

V tomto stavu čeká pracoviště na nový polotovar.Signalizace pracoviště je vypnuta.Dodání nového polotovaru je detekováno čidlem "INI".

Jakmile je detekováno dodání nového polotovaru je pás zastaven (#Transp_req=0) a dodaný díl je převzat na pracoviště.

Spádová hrana čidla "INI" uvolňuje pás (#Conv_busy=0) pro vyřízení požadavků ostatních pracovišť.

Spádová hrana čidla - signalizující převzetí dílu na pracoviště také nastavuje stav "Zpracování" na tomto pracovišti.

**Inicializace**

Kromě "normálního" chodu technologie je požadováno, aby technologii bylo možné inicializovat pomocí signálu "Initial" ve stavu "Zpracování". Při volání FB1 lze parametru #Initial přiřadit vstup I0.0.

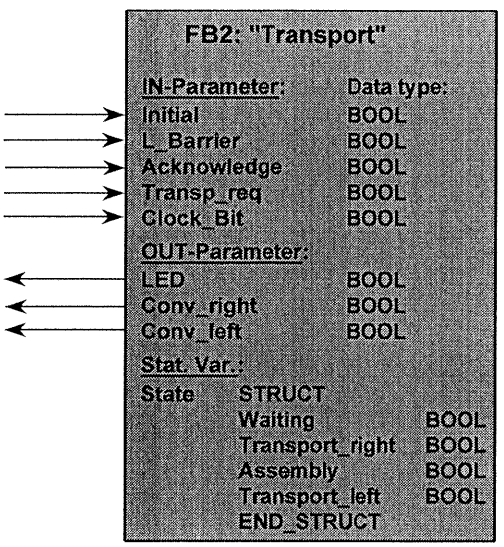
**Zadání 6.1**

Úkolem tohoto cvičení je

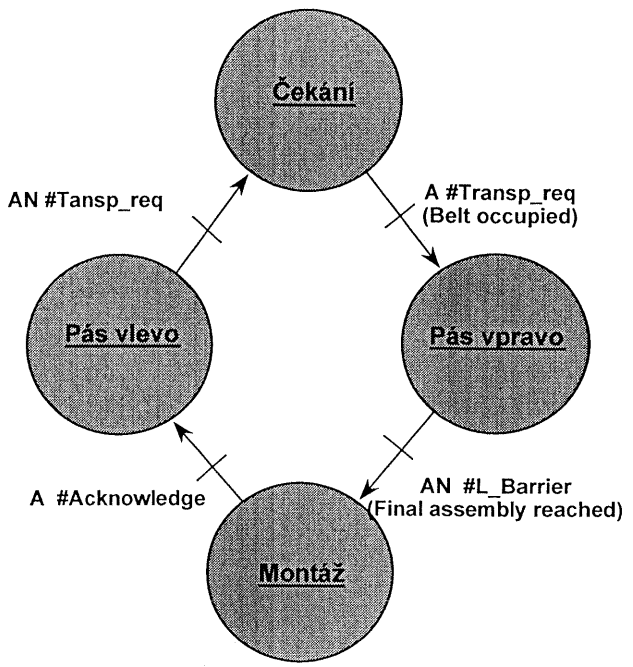
1. Vytvořit v projektu PRO2-Project nový program "Belt".
2. Vytvořit FB1 podle uvedeného zadání
3. Pro blikání použít v parametrizaci CPU "Clock memory byte" MB10.
4. Z OB1 vyvolat FB1 s instančním DB DB1 a přiřadit parametrům odpovídající technologické signály. Parametru #Clock_Bit přiřadit M10.1.
5. Nahrát program do CPU a otestovat jeho funkčnost.

# Cvièení 6.2: Tvorba FB2 pro dopravní pás

## ❑ Rozhraní FB2 :




## ❑ Stavový model:




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_06cz.19

Školicí středisko
firmy E&A spol. s r. o.

**Definice zadání**

Nejprve je nutné vytvořit FB 2 ve kterém bude uložen algoritmus řízení pásu. Činnost je popsána stavovým modelem se 4 stavy ("Čekání", "Pás vpravo", "Montáž", "Pás vlevo").

Při tvorbě algoritmu je nutné vzít v úvahu nezávislost funkce pásu na počtu operujících pracovišť.

**Princip funkce FB2**

Funkce FB2 je dána stavem a vztahy mezi jednotlivými signály. Vlastní funkce je popsána dále:

**"Čekání"**

V tomto stavu pás čeká na dokončení dílu na jednom z pracovišť. Za této situace je signalizace pásu "H4" zhasnuta a pás je v klidu.

Je-li některým z pracovišť nastaven bit #Transport_req přejde pás do stavu "Pás vpravo".

**"Pás vpravo"**

V tomto stavu je díl dopravován z pracoviště na místo konečné montáže. Po dobu dopravy bliká signalizace "H4" v taktu daném vstupním parametrem.

Dosažení místa konečné montáže je detekováno čidlem "LS" a následně je pás přepnut do stavu "Montáž".

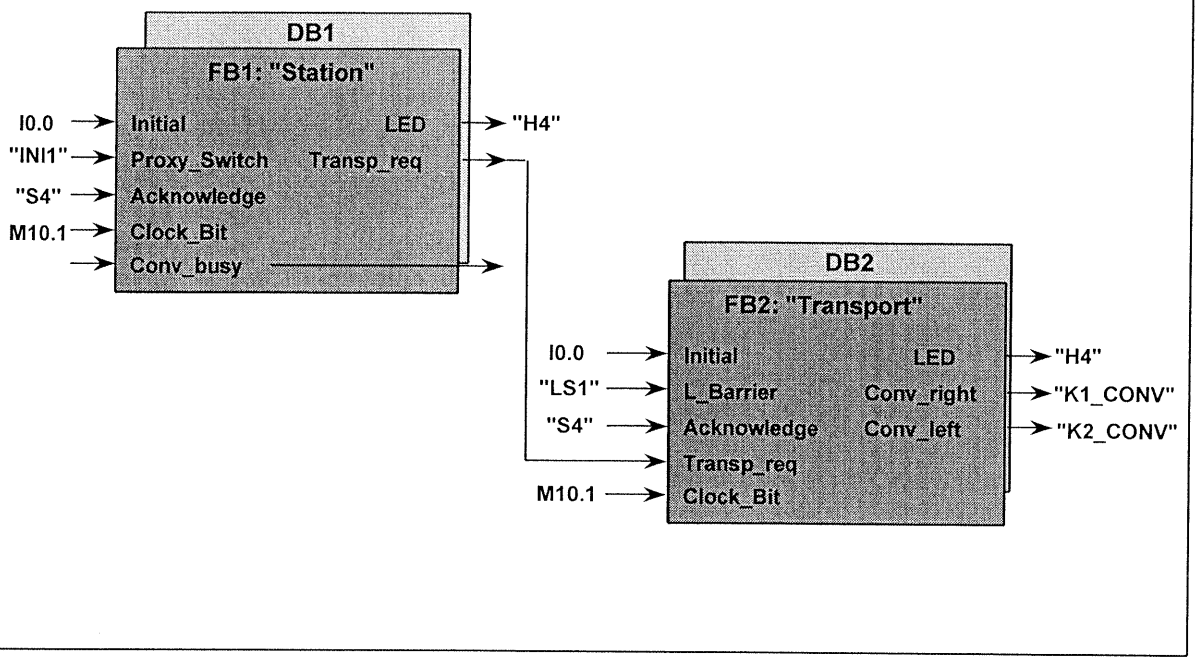
**"Montáž"**

V tomto stavu operátor odebere hotový díl a na jeho místo uloží nový polotovar. Po dobu kdy je pás v tomto stavu trvale svítí signalizace pásu. Dokončení operace operátor signalizuje tlačítkem "S4".

Stisknutí tlačítka "S4" přepne pás do stavu "Pás vlevo".

# Cvičení 6.2: Propojení parametrů FB1 a FB2 v OB1

## ❑ Obsah OB1:








**"Pás vlevo"**

V tomto stavu je dopravován nový polotovar na pracoviště. Po dobu dopravy bliká signalizace "H4" v taktu daném parametrem #Clock_Bit.
Pás je zastaven jakmile je signál #Transp_req =0.

**Poznámka**

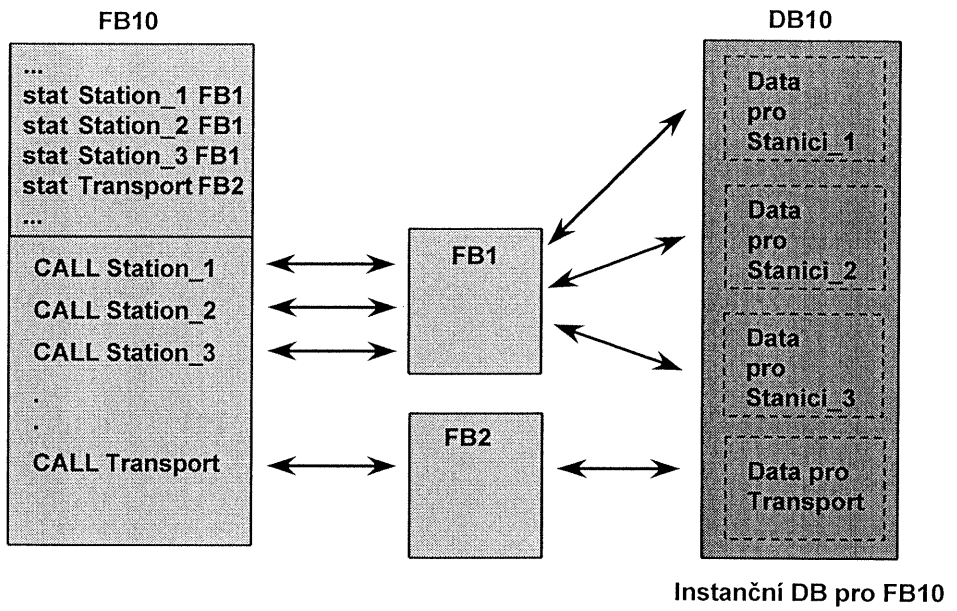
Do stavu "Čekání" lze pás přepnout také pomocí inicializačního vstupu #Initial. Tomuto parametru lze přiřadit signál I0.0.

**Zadání 6.2**

Cílem tohoto cvičení je:

1. Vytvořit FB2 s funkcí popsanou výše.
2. V OB1 zavolat FB2 (jako instanční DB lze použít DB2) po volání FB1.
   Propojení FB bloků parametry proveďte podle schématu.
3. Nahrát bloky do CPU a otestovat funkčnost vytvořeného programu.

# Cvičení 6.3: Rozšíření na 3 stanice pomocí multi-instančního modelu




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_06cz.21

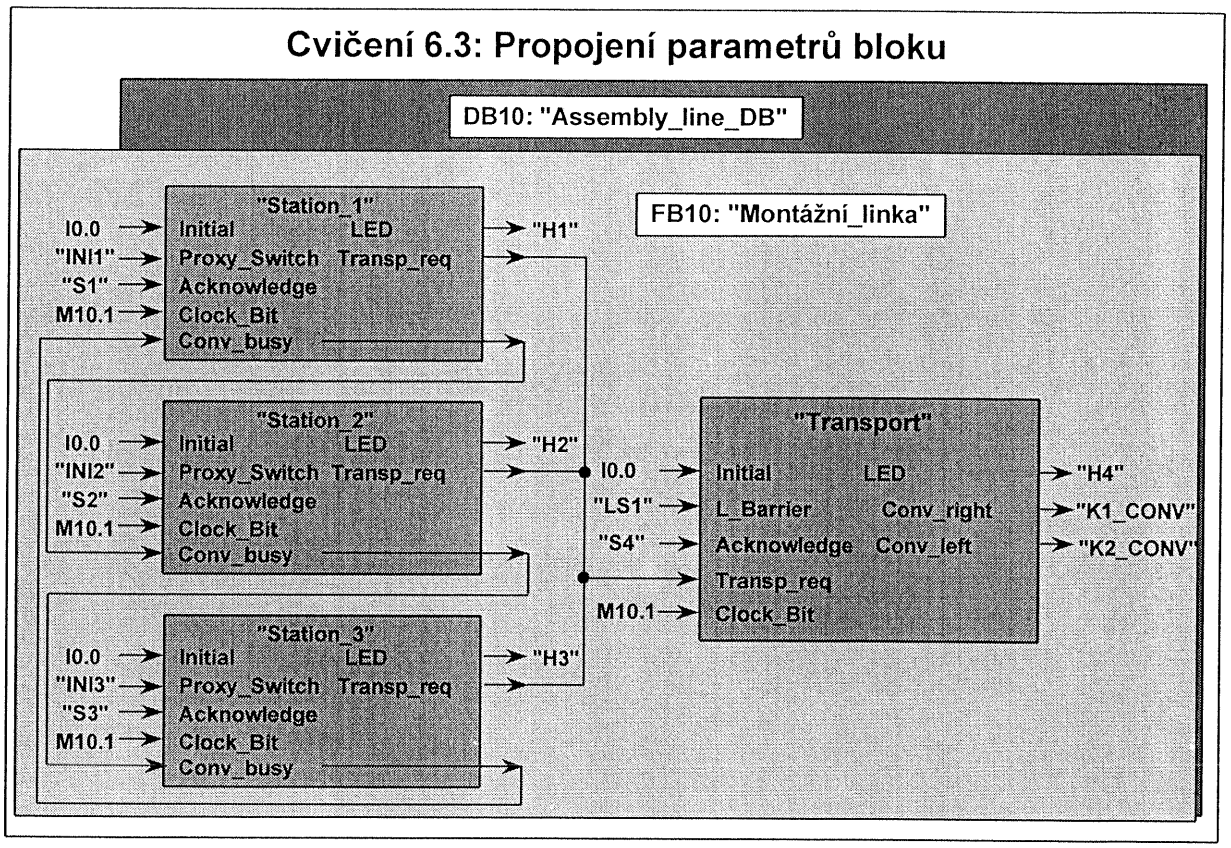
Školicí středisko
firmy E&A spol. s r. o.

**Struktura programu**

V závěrečné části cvičení je dosaženo plné funkčnosti modelu pro 3 pracoviště. Pro dosažení tohoto cíle je řízení modelu (3 pracoviště a 1 pás) přesunuto do jednoho funkčního bloku(FB10).

Uvnitř FB10 je řízení 3 pracovišť implementováno jako samostatné instance FB1 a řízení pásu jako instance FB2.

# Cvičení 6.3: Propojení parametrů bloku




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_06cz.22

Školicí středisko
firmy E&A spol. s r. o.

**Postup**

1. Nejprve je nutno vytvořit FB10. V části *stat. Var.,* deklarovat 3 instance FB1 s názvy : Station_1, Station_2 a Station_3 a jednu instanci FB2 s názvem Transport.
2. V FB10 postupně vyvolat jednotlivé instance a propojit je přes parametry podle výše uvedeného schématu.

   Poznámka pro propojení *in/out* parametru #Belt_occupied.
   Jakou proměnnou lze použít k přiřazení tomuto parametru? *Temp*orary nebo *stat*ickou?

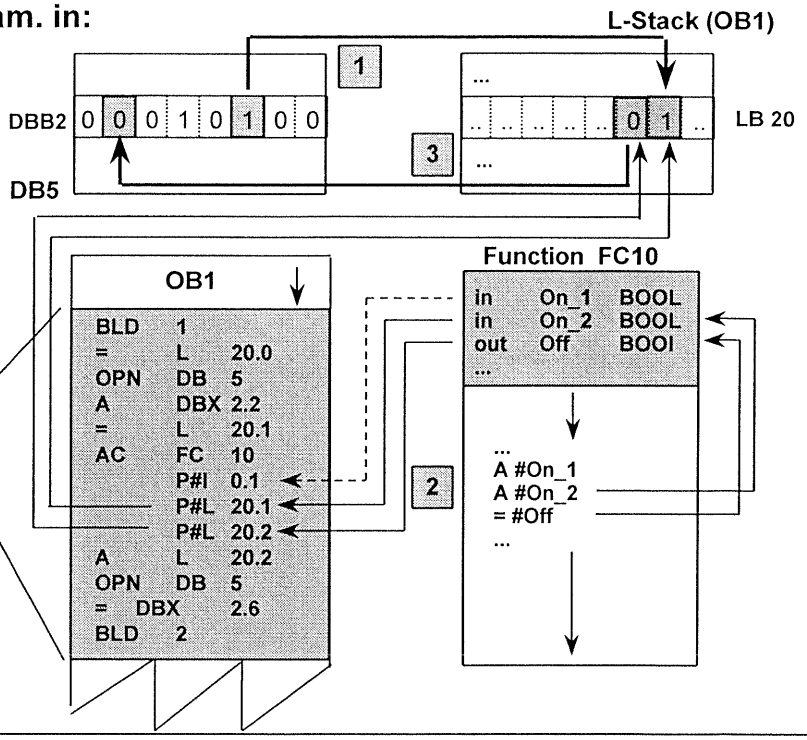
   Stejnou pozornost je třeba věnovat propojení výstupního parametru #Transp_req (logický součet "Or") na výstupní parametr #Transp_req bloku FB2. Jak lze realizovat toto propojení?
3. Vytvořit instanční DB10 pro FB 10. Editovat DB10 a zkontrolovat jeho strukturu v zobrazení deklarace i hodnot.
4. V OB1 zavolat FB10 s instančním DB- DB10.
5. Nahrát jednotlivé bloky do CPU a ověřit funkčnost programu.

**Otázka**

Jakých výhod a nevýhod bylo dosaženo tímto uspořádáním?

## Předání parametrů při volání FC (2)

- základní aktuální param. in:
  - konstantách
  - datových blocích







Datum: 24.11.2002
Soubor: PRO2_06cz.23



**Předávání parametrů** Jestliže je *in*, *out* nebo *in/out* parametru bloku přiřazena konstanta nebo parametr, uložený v DB, STL/LAD/FBD Editor nejprve reservuje nezbytně nutnou paměť v L-STACKu volajícího bloku a zkopíruje do ní hodnoty *in* a *in/out* parametrů - hodnoty aktuálních parametrů.

Pro *out* parametry se reservace paměti také provede, ale není inicializována.

Teprve potom se provede přepnutí do volaného FB a STL/LAD/FBD Editor předá ve všech případech ukazatel s udáním oblasti na L-Stack volajícího bloku.

Po skoku zpět do volajícího bloku jsou výsledky zkopírovány zpět do *out* a *in/out* parametrů - aktuálních operandů.

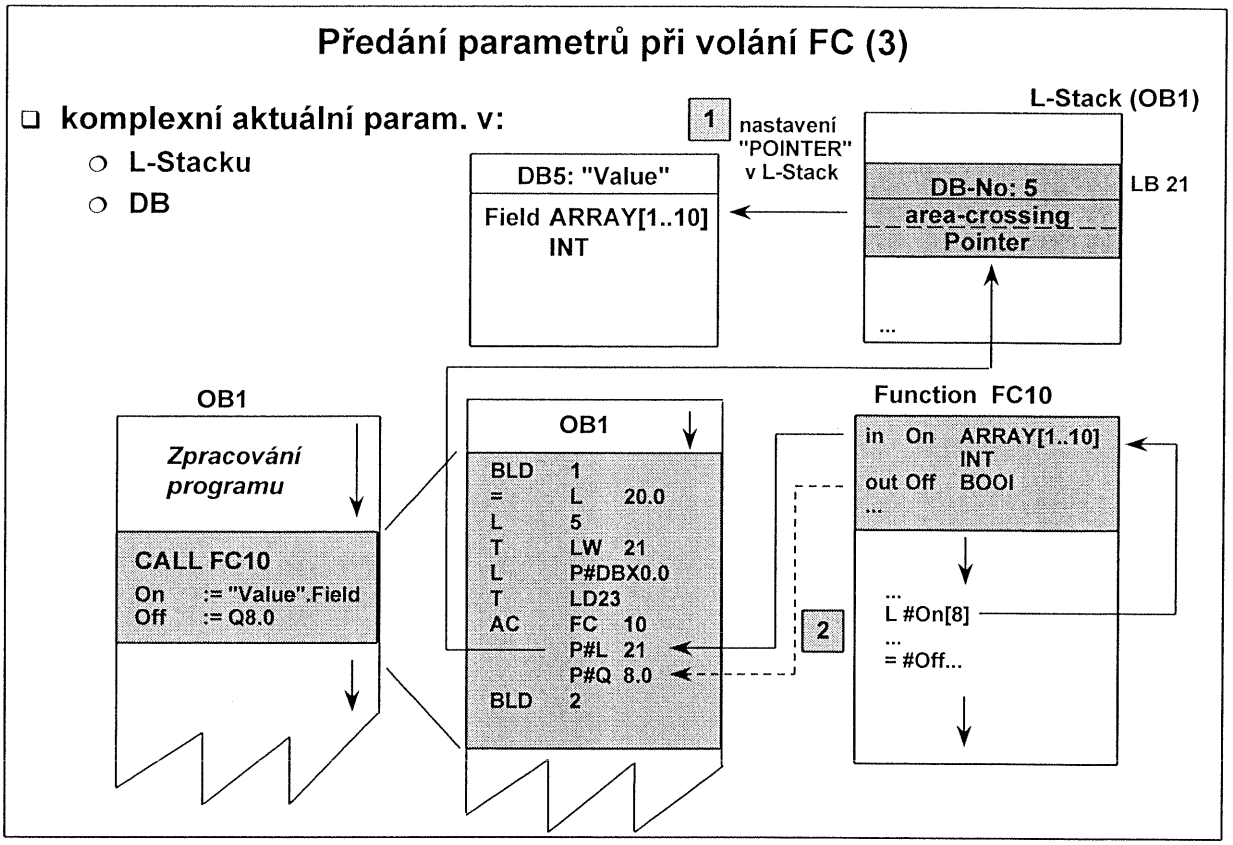
**Důsledky** Díky tomuto mechanismu předávání lze vstupní parametry pouze testovat a výstupní parametry pouze zapisovat.

Jestliže je proveden zápis do vstupního parametru, odpovídající hodnota se samozřejmě uloží do L-STACKu, ale po ukončení zpracování bloku nebude přenesena do aktuálního operandu.

Z toho samého důvodu nelze výstupní parametry číst. Z L-STACKu je díky inicializaci načtena nesmyslná hodnota. *In/out* parametry mají nejméně problémů. Hodnota aktuálního operandu je jim přiřazena před voláním, stejně jako po volání.

**Důležité** Výstupní parametry musí být zapsány uvnitř volaného FB bloku, jinak je do aktuálního operandu zkopírována z L_STACKu chybná hodnota.

Jestliže nelze zaručit bezchybné zapsání hodnoty do výstupního parametru, je výhodnější použít parametr typu *in/out*.










**Předávání parametrů** Aktuální parametry komplexního datového typu (DT, STRING, ARRAY, STRUCT a UDT) jsou uloženy v datovém bloku nebo v L-Stack volajícího bloku (V-oblast).

Protože 32-bitový ukazatel s udáním oblasti (area-crossing pointer) nedosahuje k aktuálním parametrům uloženým v DB, STL/LAD/FBD Editor ukládá do L-Stacku volaného bloku 48-bit "ukazatel" na aktuální parametr.

Během volání je na "ukazatel" předán 32-bitový *area-crossing* ukazatel.

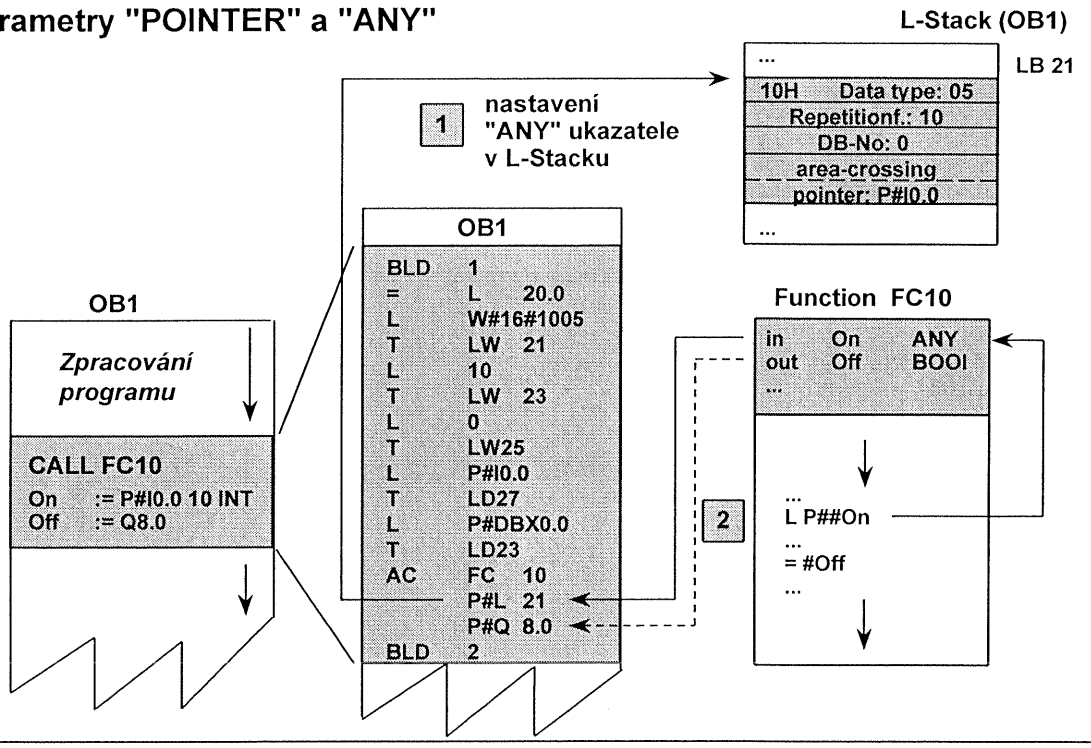
Nastavení "ukazatele" v L-Stacku volajícího bloku je provedeno před přepnutím do volaného bloku.

**Důsledky** Parametry komplexního datového typu jsou předávány snadněji než základní datové typy. Vstupní parametry komplexního typu lze uvnitř volaného FC bez dalších úvah zapisovat

Stejně tak lze výstupní parametry bez dalších úvah dotazovat.

# Předání parametrů při volání FC (4)

## ❑ Parametry "POINTER" a "ANY"







Datum: 24.11.2002
Soubor: PRO2_06cz.25



**Předání parametrů**

Je-li FC bloku předáván parametr typu "POINTER" nebo "ANY", STL/LAD/FBD Editor nastaví odpovídající datovou strukturu v L-STACKu volajícího bloku.

Při volání FC je potom předán 32-bitový *area-crossing* ukazatel na tuto strukturu.

Uvnitř volaného FC nelze k parametrům přistupovat přímo, protože "POINTER" nebo "ANY" ukazatel odkazují na chybný typ dat.

Vyhodnocení obsahu "POINTER" nebo "ANY" musí být provedeno uživatelem s použitím základních STL instrukcí uvnitř volaného bloku (viz cvičení 3.3).

Nastavení "POINTER" nebo "ANY" struktury v L-Stacku volajícího bloku je provedeno před skokem do volaného FC.

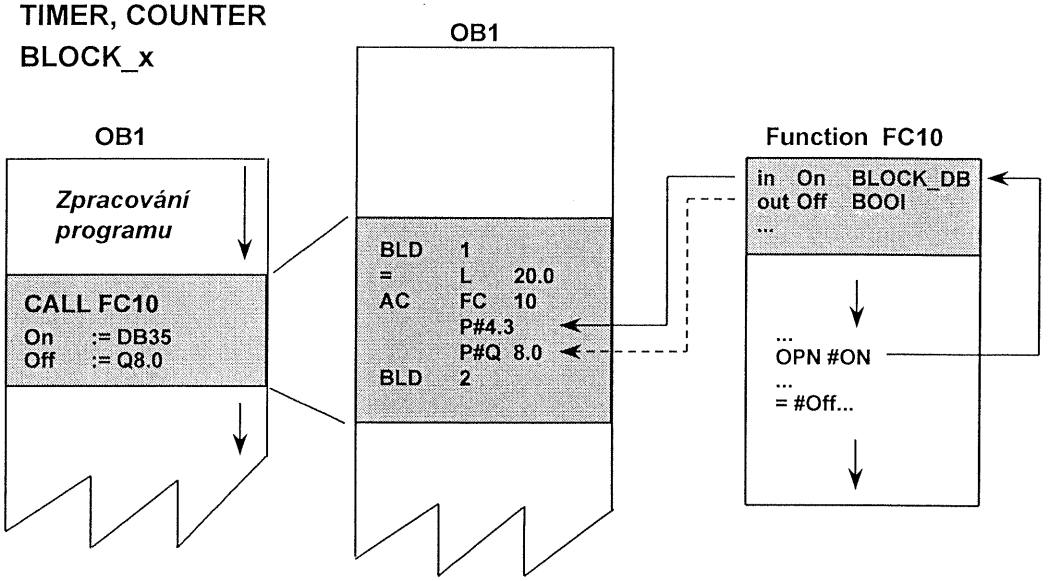
**Vyjímka**

Vyjímku z tohoto pravidla dělá STL/LAD/FBD Editor tehdy, když je parametru "ANY" přiřazen aktuální operand typu "ANY ", který je již uložen v L-Stacku volajícího bloku.

V tomto případě STL/LAD/FBD Editor nenastavuje další strukturu "ANY" ukazatele, ale do FC předává přímo 32-bit *area-crossing* ukazatel na již existující "ANY" ukazatel (v L-Stacku volajícího bloku).

## Předání parametrů při volání FC (5)

- **Parametry:**
  - TIMER, COUNTER
  - BLOCK_x





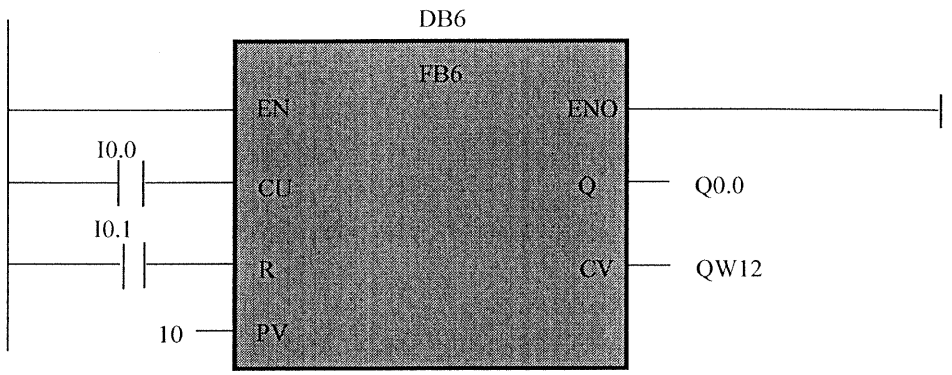

Datum: 24.11.2002
Soubor: PRO2_06cz.26



**Předání parametrů**

Předání parametrů typu TIMER, COUNTER a BLOCK_x je nejjednodušší. V tomto případě je předáno, místo 32-bitového *area-crossing* ukazatele, číslo aktuálního TIMERu nebo COUNTERu nebo bloku.

# Doplňkové cvičení 6.4: Tvorba vlastního bloku čitače




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_06cz.27

Školicí středisko
firmy E&A spol. s r. o.

**Definice**

Vytvořit 16-bitový "čítač" FB6 "CU" s následujícími vlastnostmi:

- S náběžnou hranou na vstupu "CU" je obsah "čítače" zvýšen o 1. Pokud již bylo dosaženo horního limitu 32 767 impuls se ignoruje.
- Je-li vstup "R" =1 je " čitač" nastaven na 0 bez ohledu na stav signálu "CU"
- Výstup "Q" signalizuje, zda je obsah "čítače" menší než přednastavená ; hodnota PV (Q=0) nebo roven či větší než PV (Q=1)

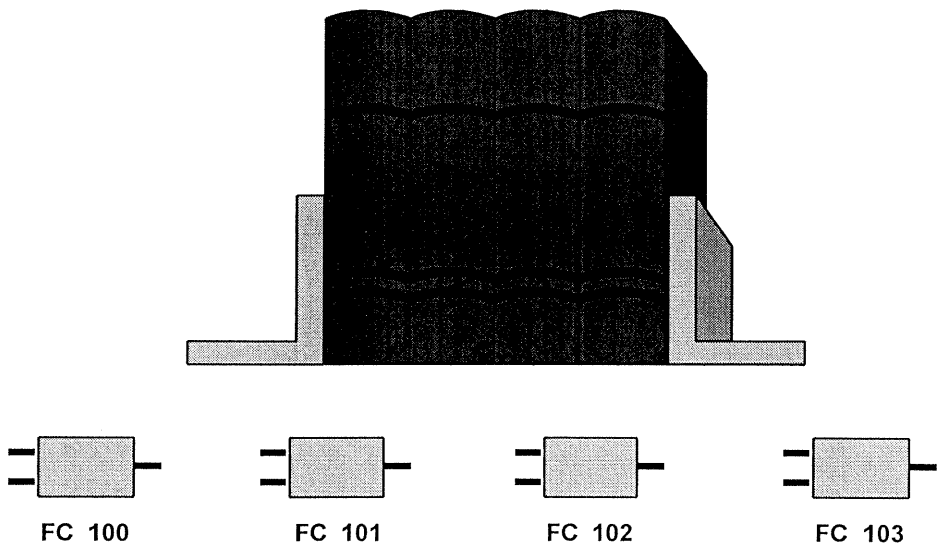
**Parametry**

| Parametr | Deklarace | Datový typ | Popis |
|---|---|---|---|
| CU | INPUT | BOOL | čitací impuls |
| R | INPUT | BOOL | Reset |
| PV | INPUT | INT | Přednastavená hodnota |
| Q | OUTPUT | BOOL | Stav čitače: Q = 1--> CU >PV |
| CV | OUTPUT | INT | Okamžitá hodnota |

**Postup**

1. Vytvořit FB6 s popsanými vlastnostmi. Při implementaci nepoužívat globální adresy CPU.
3. V OB1 zavolat FB 6 s instančním DB 6 a přiřadit následující operandy:
   - CU = I0.0
   - R = I0.1
   - PV = IW4 (palcový přepínač Simulátoru)
   - Q = Q8.0
   - CV = QW12 (display Simulátoru)
5. Nahrát program do CPU a otestovat.

# Knihovny




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_07cz.1

Školicí středisko
firmy E&A spol. s r.o., MB

**Obsah** **Strana**

## Vlastnosti knihoven

- **Vlastnosti a smysl:**
  - **uschování částí programu pro další použití**
  - **přímý přenos do CPU a testování není možné**
- **Konfigurace:**
  - **knihovny mohou obsahovat více programových složek**
  - **knihovny nemohou obsahovat složku "Hardware"**
  - **programová složka vždy obsahuje:**
    - složky "Blocks", "Sources", "Symbols"
    - složku "Charts" (pouze při instalovaném doplňku: S7-CFC)
- **Použití:**
  - **se SIMATIC Managerem:**
    - zakládání knihoven (ale ne se stejným jménem jako je jméno projektu)
    - bloky lze kopírovat mezi projekty a knihovnami
    - knihovny lze archivovat

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_07cz.2

Školicí středisko
firmy E&A spol. s r o., MB

**Přehled**

Knihovny slouží k uložení částí programů, které lze opakovaně použít. Tyto části programu lze kopírovat z projektu do knihovny nebo je přímo vytvářet v knihovně.

Tvorba bloku v knihovně má stejné zásady jako tvorba v projektu. Až na jedinou výjimku a tou je testování bloku.

**Konfigurace**

Stejně jako projekty mají i knihovny hierarchické uspořádání:

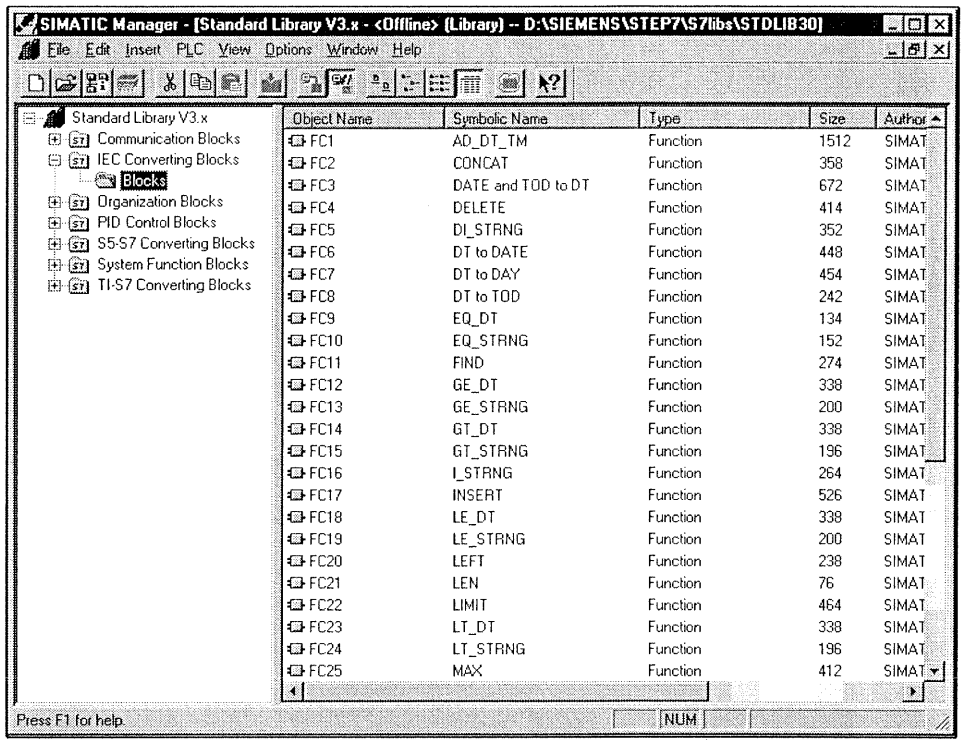
- knihovny obsahují složku S7-Program.
- tato složka (S7-Program) obsahuje dále objekty "Block" ("User Program"), "Sources", "Charts" a "Symbols".
- složka "Blocks" obsahuje jednotlivé bloky, které mohou být nahrány do S7-CPU. Dále může obsahovat objekty typu VAT a UDT, které se do CPU nenahrávají.
- složka "Sources" obsahuje zdrojové programy vytvořené pomocí jiných programovacích jazyků.
- složka "Charts" obsahuje CFC-grafy (pouze při instalovaném doplňku S7-CFC)

Při zakládání nové složky S7-Program, jsou automaticky také vytvořeny složky “Block” a “Sources” , stejně jako objekt “Symbols”.

**Použití knihoven**

Knihovny slouží k uložení jakékoliv části programu (vyjma HW konfigurace), kterou lze kdykoliv použít v jiném programu nebo projektu.

# Konfigurace a obsah standardní knihovny




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_07cz.3

Školicí středisko
firmy E&A spol. s r.o., MB

**Úvod**

Při instalaci STEPu 7 jsou automaticky instalovány také 2 knihovny:

- standardní knihovna stdlibs(V2) pro verzi 2 a
- Standard Library V3.x pro verzi 3.

Bloky obsažené v těchto knihovnách lze kdykoliv kopírovat do uživatelského projektu.

**Otevření knihovny**

K otevření knihovny slouží nabídka: *File -> Open* nebo příslušné tlačítko na nástrojové liště.

Vyvolaný dialog umožňuje uživateli určit která knihovna bude otevřena (případně který projekt).

**Standardní knihovna**

Knihovna StdLib30 obsahuje následující S7-Programy:

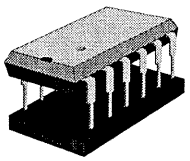
- komunikační bloky: funkce pro obsluhu distribuovaných periferií s použitím S7-300 Profibus CP
- IEC konverzní bloky: funkce pro práci s datovým typem DATE_AND_TIME a STRING.
- organizační bloky: obsahem jsou všechny systémové funkce S7-300/400
- PID řídící bloky: funkce pro PID řízení
- S5-S7 konverzní bloky: standardní bloky nutné pro převod S5 programu na S7 program
- systémové FB: obsahuje všechny systémové funkce S7-300/400.
- TI-S7 konverzní bloky:obecně použitelné standardní funkce např. zpracování analogové hodnoty apod.

**Poznámka**

Během instalace programovacích doplňků jsou instalovány další doplňkové knihovny.

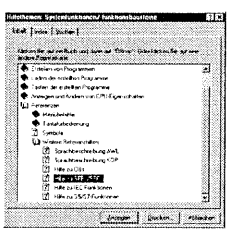
Popis knihoven "PID" a "TI - S7 Converting Blocks" je umístěn pod nabídkou: *Taskbar -> SIMATIC -> S7 manuals -> PID Control, Standard Functions 2*

## Vlastnosti systémových funkcí



**Systémové funkce (SFC a SFB) jsou součástí operačního systému CPU**

**Popis těchto funkcí je v manuálech:**
**"System Software Reference Manual for S7-300/400" a**
**"System Functions and Standard Functions"**



**Další popis těchto funkcí je také v "on-line" nápovědě**

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_07cz.4

Školicí středisko
firmy E&A spol. s r.o., MB

**Úvod**

Funkce, které nejsou implementovány jako instrukce STEPu7 (např.tvorba DB, komunikace s jiným PLC, atd.) lze do programu implementovat s použitím systémových funkcí (SFC) nebo systémových funkčních bloků (SFB).

Bloky SFC a SFB jsou uloženy jako součást operačního sytému CPU. Do knihoven je importována pouze jejich deklarační část, kterou lze načíst z CPU.

S použitím STL/LAD/FBD Editoru lze tyto načtené bloky otevřít a zobrazit tak jejich deklarační část. Zpětný přenos do CPU není samozřejmě možný.

V uživatelském programu mohou být tyto bloky volány pomocí instrukce CALL stejně jako bloky FC a FB. Při volání SFB bloků je nutné stejně jako u FB bloků definovat instanční DB.

Dostupnost jednotlivých SFC a SFB bloků je závislá na použitém systému (S7-300 nebo S7-400) a instalovaném CPU. Tyto bloky mají bez ohledu na systém a CPU stejné číslo, funkci a rozhraní.

**Návod/popis**

Detailnější popis jednotlivých systémových funkcí je v manuálu:

- "The System Software Reference Manual for S7-300/400, System Functions and Standard Functions".

**Online nápověda**

Tento detailnější popis je také v elektronické podobě v nápovědě:

- *Help topics -> Block help -> Help with SFBs/SFCs .*

## Přehled systémových funkcí (část 1.)

| Skupina funkcí | Funkce | Blok | S7-300 | S7-400 |
|---|---|---|---|---|
| Kopírování a blokové funkce | Přesun bloku | SFC 20 | X | X |
| | Nastavení pole | SFC 21 | X | X |
| | Generování DB | SFC 22 | X | X |
| | Mazání DB | SFC 23 | - | X |
| | Testování DB | SFC 24 | - | X |
| | Komprimace | SFC 25 | - | X |
| | Substitute value in ACCU 1 | SFC 44 | X[1)] | X |
| Programové řízení | Víceprocesorové přerušení | SFC 35 | - | X[2)] |
| | Kontrola doby cyklu | SFC 43 | X | X |
| | Stop stav | SFC 46 | X | X |
| | Prodleva (čekání) | SFC 47 | X[)] | X |
| Hodiny | Nastavení hodin | SFC 0 | X | X |
| | Čtení hodin | SFC 1 | X | X |
| | Synchronizace | SFC 48 | - | X |
| Čítač hodin | Nastavení čítače | SFC 2 | X[1)] | X |
| | Start a stop | SFC 3 | X[1)] | X |
| | Načtení čítače | SFC 4 | X[1)] | X |
| | Čtení systémových hodin | SFC 64 | X | X |

1) ne u CPU 312IFM 2) pouze modernizovaná CPU



**Kopírovací a blokové funkce**

- SFC 20 kopíruje obsah paměti z oblasti zdroje do oblasti cíle
- SFC 21 vyplní oblast cílového pole obsahem zdrojového pole
- SFC 22 vytvoří DB bez přednastavené hodnoty v pracovní oblasti
- SFC 23 maže DB z pracovní eventuelně z editační paměti.
- SFC 24 detekuje zda je DB přítomný v paměti (jaká je jeho délka).
- SFC 25 provádí kompresi paměti.
- SFC 44 (volaný z OB 122) ukládá hodnoty chybného vstupního modulu v Accu

**Řízení programu**

- SFC 35 ve víceprocesorovém režimu synchronně spouští OB60 na všech CPU
- SFC 43 spouští nové odměřování doby cyklu
- SFC 46 přepne CPU do Stop stavu
- SFC 47 umožňuje prodloužení doby cyklu až na 32767 µs.

**Zpracování hodin**

- SFC 0 nastavuje reálný datum a čas na hodinách CPU
- SFC 1 čte současné datum a čas ze systémových hodin
- SFC 48 synchronizuje všechny podřízené hodiny na daném segmentu sběrnice. CPU vyvolávající tuto funkci musí být nastaven jako "master".

**Zpracování čítače hodin**

Každé CPU má definovaný počet čitačů hodin, které mohou odměřovat dobu provádění operace.

- SFC 2 nastavuje čitač hodin na žádanou hodnotu
- SFC 3 spouští a zastavuje čitač hodin
- SFC 4 čte okamžitý stav čitače hodin
- SFC 64 čte systémový čas. Systémový čitač hodin je aktualizován každých 10 ms ( u S7-300) nebo 1 ms (S7-400).

## Přehled systémových funkcí (část 2.)

| Skupina | Funkce | Blok | S7-300 | S7-400 |
|---|---|---|---|---|
| Přenos dat | Zápis aktivních parametrů. | SFC 55 | X | X |
| | Zápis definovaných param. | SFC 56 | X | X |
| | Parametrizace modulu. | SFC 57 | X | X |
| | Zápis dat. | SFC 58 | X | X |
| | Čtení dat. | SFC 59 | X | X |
| Časové přerušení | Nastavení | SFC 28 | X[1)] | X |
| | Zrušení | SFC 29 | X[1)] | X |
| | Aktivace | SFC 30 | X[1)] | X |
| | Snímání | SFC 31 | X[1)] | X |
| Zpožděné přerušení | Start | SFC 32 | X[1)] | X |
| | Zrušení | SFC 33 | X[1)] | X |
| | Snímání | SFC 34 | X[1)] | X |
| Synchronní chyby | Maskování chyby | SFC 36 | X | X |
| | Odmaskování chyby | SFC 37 | X | X |
| | Čtení stavového registru | SFC 38 | X | X |
| Chybové přerušení a asynchronní chyby | Zrušit nové přerušení | SFC 39 | X | X |
| | Povolit nové přerušení | SFC 40 | X | X |
| | Prodleva nového přerušení | SFC 41 | X | X |
| | Povolení vyšší priority přerušení | SFC 42 | X | X |

1)ne pro CPU 312IFM





**Přenos datových záznamů**

V CPU je systémová oblast pro parametrizaci a diagnostiku parametrizovatelných modulů. Tato oblast obsahuje záznamy 0 až 255, které mohou být čteny nebo zapisovány.

- SFC 55 přenese aktivní parametry do adresovaného modulu. Parametry v SDB v CPU nejsou přepsány.
- SFC 56 přenese parametry (datový záznam RECNUM) do modulu.
- SFC 57 přenese všechny datové záznamy z SDB do modulu.
- SFC 58 přenese záznam RECORD do modulu.
- SFC 59 čte záznam RECORD z modulu.

**Časové přerušení (time-of-date interrupt)**

Tyto bloky slouží k obsluze časových přerušení (OB 10 až 17). Výchozí okamžik lze definovat pomocí STEPu 7 nebo s použitím těchto funkcí.

- SFC 28 nastavuje výchozí datum a čas pro příslušný OB.
- SFC 29 maže datum a čas OB (OB 10 až OB 17).
- SFC 30 aktivuje časové přerušení OB.
- SFC 31 čte stav časového přerušení OB.

**Zpožděné přerušení (delay interrupt)**

- SFC 32 startuje přerušení (OB 20 až 27).
- SFC 32 ruší přerušení.
- SFC 34 čte stav přerušení.

**Synchronní chyby**

- SFC 36 maskuje synchronní chybu, tj. chybná instrukce nevyvolá odpovídající chybový OB
- SFC 37 odmaskuje synchronní chybu
- SFC 38 čte chybový registr.

**Přerušení a asynchronní chyby**

- SFC 39 zakazuje zpracování přerušení a asynchronních chyb.
- SFC 40 znovu povoluje zpracování přerušení a asynchronních chyb.
- SFC 41 zpozdí zpracování přerušení a asynchronních chyb.
- SFC 42 uvolňuje opět zpracování odloženého přerušení nebo asychronních chyb

## Přehled systémových funkcí (část 3.)

| Skupina | Funkce | Blok | S7-300 | S7-400 |
|---|---|---|---|---|
| Systémová diagnostika | Čte startovní informace. | SFC 6 | - | X |
| | Čte systémový stav | SFC 51 | X | X |
| | Zapisuje do diag. Bufferu | SFC 52 | X | X |
| Obraz procesu, I/O oblast | Aktivace PII vstupů. | SFC 26 | - | X |
| | Aktivate PIQ výstupů. | SFC 27 | - | X |
| | Setuje bit v I/O. | SFC 79 | - | X |
| | Resetuje bit v I/O. | SFC 80 | - | X |
| Adresace modulů | Určuje logickou adresu. | SFC 5 | - | X |
| | Určuje slot. | SFC 49 | X | X |
| | Určuje všechny logické adresy. | SFC 50 | X | X |
| Distribuované I/O | Spouští zpracování přerušení | SFC 7 | 1) | 1) |
| | Synchronizace DP stanic | SFC 11 | 1) | 1) |
| | Čte diagnostická data. | SFC 13 | 1) | 1) |
| | Čte uživatelská data. | SFC 14 | 1) | 1) |
| | Zapisuje uživatelská data. | SFC 15 | 1) | |
| Globální datová komunikace | Vyšle GD package. | SFC 60 | - | X |
| | Přijme GD package. | SFC 61 | - | X |

1) pouze u CPU s DP rozhraním tj. CPU 315-2 DP





**Systémová diagnostika**

- SFC 6 čte startovní informace naposledy volaného OB a startovních OB
- SFC 51 čte část záznamů systémového stavu. Seznam obsahuje : systémová data, diagnostická stavová data, diagnostická data, a diagnostický buffer.
- SFC 52 umožňuje uživatelský zápis do diagnostického bufferu

**Obraz procesu, I/O oblast**

- SFC 26 aktualizuje část nebo celý obraz vstupů
- SFC 27 aktualizuje výstupy podle části nebo celého obrazu výstupů
- SFC 79/ 80 umožňují setování a resetování pole bitů v oblasti I/O ve spojení s funkcí *Master Control Relay.*

**Adresace modulů**

- SFC 5 převádí logické adresy na geografické
- SFC 49 určuje geografickou adresu z logické adresy
- SFC 50 zjišťuje všechny logické adresy modulů

**Decentrální periferie**

- SFC 7 spouští uživatelský program inteligentní *slave*-stanice (CPU 315-2DP) a hardwarové přerušení na DP *master.*
- SFC 11 synchronizuje jednu nebo více skupin DP *slave*
- SFC 13 čte diagnostická data DP *slave*
- SFC 14 čte konzistentní data z DP *slave.*
- SFC 15 zapisuje konzistentní data na DP *slave*

**Globální datová komunikace**

Globální data jsou zapisována cyklicky (např. každý 8. cyklus) bez použití SFC. Pomocí SFC 60 a 61 lze spouštět vysílání a příjem globálních dat uživatelsky.

- SFC 60 vysílá GD package
- SFC 61 přijímá GD package

## Přehled systémových funkcí (část 4.)

| Skupina | Funkce | Blok | S7-300 | S7-400 |
|---|---|---|---|---|
| Datová výměna přes SFB, konfigurovaná spojení | Sledování stavu | SFC 62 | - | X |
| | Nekoordinované vysílání | SFB 8 | - | X |
| | Nekoordinovaný příjem | SFB 9 | - | X |
| | Vysílání bloku | SFB 12 | - | X |
| | Čtení bloku | SFB 13 | - | X |
| | Čtení ze vzdáleného CPU | SFB 14 | - | X |
| | Zápis dat do vzdáleného CPU | SFB 15 | - | X |
| | Vysílání na tiskárnu | SFB 16 | - | X |
| | Provedení kompletního restartu | SFB 19 | - | X |
| | Stop stav | SFB 20 | - | X |
| | Provedení restartu | SFB 21 | - | X |
| | Sledování stavu zařízení | SFB 22 | - | X |
| | Příjem stavu zařízení | SFB 23 | - | X |
| Datová výměna přes SFC, nekonfigurovaná spojení | Vysílání dat na externího partnera | SFC 65 | 1) | 1) |
| | Příjem dat od externího partnera | SFC 66 | 1) | 1) |
| | Čtení externích dat | SFC 67 | 1) | 1) |
| | Zápis externích dat | SFC 68 | 1) | 1) |
| | Zrušit externí spojení | SFC 69 | 1) | 1) |
| | Čtení dat z interního partnera | SFC 72 | 1) | 1) |
| | Zápis dat do interního partnera | SFC 73 | 1) | 1) |
| | Zrušit interní spojení | SFC 74 | 1) | 1) |

1) pouze inovovaná CPU





**Datová výměna přes SFB**

SFB slouží k výměně dat a správě programů přes konfigurované spojení. Podle toho zda je nutné volat CFB bloky na jedné nebo obou stranách lze hovořit o jednostranné nebo obou stranné komunikaci. SFB bloky jsou k dispozici pouze na S7-400.

- SFC 62 určuje stav lokální instance CFB a stav přidruženého spojení
- SFB 8 nekoordinované vysílání na vzdáleného(remote) partnera
- SFB 9 nekoordinovaný příjem ze vzdáleného partnera
- SFB 12 vysílání max. 64 Kbyte dat s potvrzením od vzdáleného.
- SFB 13 příjem dat od vzdáleného partnera s potvrzením
- SFB 14 čte data ze vzdáleného CPU (jednostranná komunikace)
- SFB 15 zápis dat na vzdálené CPU (jednostranná komunikace)
- SFB 16 vysílání formátovaných dat na vzdálenou tiskárnu
- SFB 19 spustí kompletní náběh na vzdáleném CPU
- SFB 20 vyvolá STOP stav na vzdáleném CPU
- SFB 21 spustí náběh vzdáleného CPU
- SFB 22 zjistí stav (režim činnosti, chybové informace) na vzdáleném CPU.
- SFB 23 příjem stavových informací vzdáleného partnera.

**Datová výměna přes SFC**

Tato komunikace je možná jak u S7-300 tak i u S7-400. V porovnání s komunikací pomocí SFB bloků jsou zde následující rozdíly:

- není nutná konfigurace spojení
- nejsou potřeba instanční DB
- maximální délka uživatelských dat: 76 bytes
- aktivní konfigurace spojení
- komunikace přes MPI nebo K bus

# Přehled systémových funkcí (část 5.)

| Skupina | Funkce | Blok | S7-300 | S7-400 |
|---|---|---|---|---|
| Integrované řízení uzavřené smyčky | Plynulé řízení | SFB 41 | 3) | - |
| | Krokové řízení | SFB 42 | 3) | - |
| | Tvarování pulsů | SFB 43 | 3) | - |
| Umělé hmoty | volání kompletačního bloku | SFC 63 | 1) | - |
| Integrované funkce | Vysokorychlostní čítač | SFB 29 | 2) | - |
| | Měřič frekvence | SFB 30 | 2) | - |
| | A/B čítač | SFB 38 | 3) | - |
| | Polohování | SFB 39 | 3) | - |
| IEC časovače a IEC čítače | Impuls | SFB 3 | x | x |
| | Zpožděné zapnutí | SFB 4 | x | x |
| | Zpožděné vypnutí | SFB 5 | x | x |
| | Inkrementace | SFB 0 | x | x |
| | Dekrementace | SFB 1 | x | x |
| | Inkrement/dekrement | SFB 2 | x | x |
| Blokově orientovaná hlášení | Hlášení bez potvrzení | SFB 36 | - | x |
| | Hlášení s potvrzením | SFB 33 | - | x |
| | Hlášení s 8 připojenými hodnotami | SFB 35 | - | x |
| | Hlášení bez připojených hodnot | SFB 34 | - | x |
| | Vysílání archivních datdata | SFB 37 | - | x |
| | Zákaz hlášení | SFC 10 | - | x |
| | Uvolnění hlášení | SFC 9 | - | x |

1) pouze u CPU 614 2) pouze u CPU 312 IFM 3) pouze u CPU 314IFM





**Řízení uzavřené smyčky**

Tyto bloky budou integrovány do pozdějších verzí CPU

**Technologie umělých hmot**

U CPU 614 (S7-300) lze vytvářet jednotlivé bloky také v jazyku "C". SFC 63 umožňuje volání takto vytvořených bloků.

**Integrované funkce**

Tyto bloky jsou pouze u CPU 312 IFM (S7-300).
Detailní popis je v manuálu *Integrated Functions*.

- SFB 29 počítá impulsy na integrovaných vstupech.
- SFB 30 umožňuje určit frekvenci signálu na integrovaných vstupech.

**IEC časovače a čítače**

Tyto časové a čítačové funkce odpovídají standardu IEC 1131-3. Zbývající funkce zajišťují zpětnou kompatibilitu s PLC SIMATIC S5.

**Blokově orientovaná hlášení**

Tyto bloky umožňují zpracování hlášení pro systémy MMI. Tato hlášení jsou vytvářena v CPU pomocí těchto procedur a následně vyslána na display. Hlášení jsou centrálně potvrzována, tj. jestliže je hlášení na displeji potvrzeno, je odpověď odeslána také do CPU.

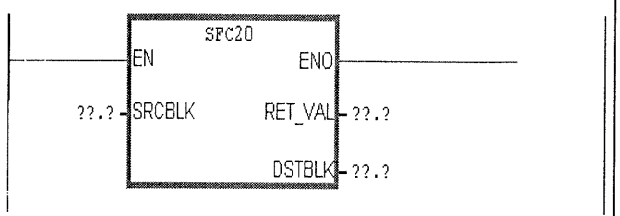
Hlášení jsou spouštěna hranou signálu.

# Volání systémových funkcí

**Volání v STL**

```
CALL  SFC   20
 SRCBLK :=
 RET_VAL:=
 DSTBLK :=
```

**Volání v LAD**








**Úvod**

Je-li volána systémová funkce, je tato funkce automaticky zkopírována do uživatelského programu.

Všechny systémové funkce jsou uloženy ve standardní knihovně StdLib30. Lze také zkopírovat SFC a SFB bloky z této knihovny do uživatelského programu.

V knihovně je uložena kompletní symbolická tabulka.Symboly zkopírovaných bloků jsou automaticky zkopírovány do globální symbolické tabulky v programu.

**Volání v STL**

Po zadání instrukce CALL SFC .., je zobrazen seznam parametrů, které je nutno přiřadit.

Detailní popis parametrů a vracených chybových hlášení je v „on-line" nápovědě.

**Volání v LAD/FBD**

Po vybrání odpovídajícího místa v segmentu lze blok vložit z nabídky *Insert -> LAD Element --> SFC* nebo *SFB* bloky.

## Vyhodnocení chybových hlášení

- **Stavový bit BR informuje o způsobu zpracování, RLO=0 signalizuje chybné zpracování, RLO=1 signalizuje bezchybné zpracování.**
  - **Test BR v STL instrukcí A BR**
  - **Test v LAD pomocí výstupního parametru ENO**
- **Většina systémových funkcí vrací ve výstupním parametru RET_VAL (INT) chybový kód:**
  - **RET_VAL=W#16#8 X Y Z**

    **třída chyby, individuální chybové číslo (SFC specifické) nebo číslo události (obecné)**

    **X>0: obecná chyba, X= číslo chybujícího parametru**
    **X=0: SFC specifická chyba**

    **"8" : nastala chyba**
  - **Příklad:**
    - W#16#8081 je specifický chybový kód SFC.
    - W#16#823A je obecný chybový kód, příčinnou je parametr č.2





**Chybová informace**

SFC bloky signalizují do okolního programu, zda jejich zpracování proběhlo s chybou nebo bez chyby. Průběh zpracování je signalizován dvojím způsobem:

- bitem BR ve stavovém slově a
- hodnotou výstupního parametru RET_VAL.

**Poznámka**

Zjištění skutečného způsobu zpracování je vhodné provádět dvoustupňově:

- nejprve vyhodnotit stavový bit BR.
- a následně testovat výstupní parametr RET_VAL.

**Obecné chyby**

Obecná chyba může být hlášena při zpracování všech systémových funkcí. Obecný chybový kód obsahuje 2 čísla:

- číslo parametru (1 až 127), kde 1 představuje 1. parametr, 2 druhý parametr volaného SFC bloku, atd..
- číslo události (0 až 127) - synchronní chyby.

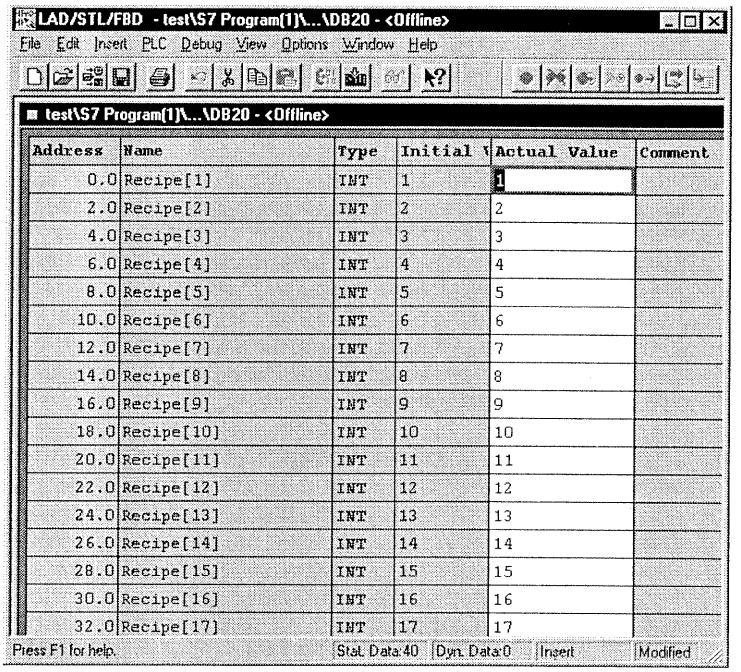
Detailní popis obecných chybových kódů je v manuálu "System Functions and Standard Functions" nebo v "on-line" nápovědě.

**Specifické chyby**

Některé systémové funkce (SFC) poskytují specifické chybové kódy odpovídající příslušným chybám, ke kterým došlo při zpracování bloku.

Detailní popis těchto kódů je také v "on-line" nápovědě pro systémové funkce.

# Cvičení 7.1: Tvorba "Unlinked" DB







Datum: 24.11.2002
Soubor: PRO2_07cz.12



**Cíl cvičení**

Vytvořit datový blok s atributem "UNLINKED".

**Definice**

Protože pracovní paměť CPU není neomezená, lze datové bloky ukládat do editační paměti.

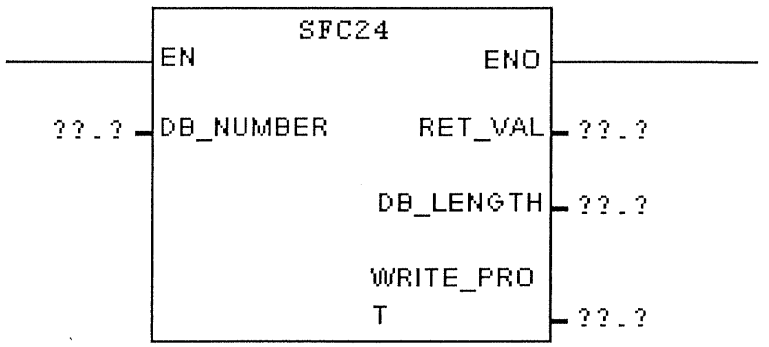
V pracovní paměti tak mohou zůstat pouze "pracovní" datové bloky - s aktuálními hodnotami. Při změně sady technologických parametrů je potom nutné zkopírovat příslušný datový blok z editační paměti do pracovní.

S použitím atributu "UNLINKED" lze dosáhnout toho, že datový blok bude při přenosu do CPU nahrán pouze do editační paměti a nebude automaticky přenesen do pracovní paměti.

**Postup**

1. Založit DB20.
2. Deklarovat proměnnou "Recipe" typu ARRAY[1..20] typu "INT" .
3. Aktivací nabídky *View -> Data View,* přepnout do datového zobrazení a nastavit hodnoty proměnných
4. V SIMATIC Manageru vybrat příslušný datový blok, pravým tlačítkem vyvolat zkrácenou nabídku (*short-menu*) > *Object properties* a aktivovat volbu "UNLINKED".
5. Přenést datový blok do CPU.
6. Co se stane, jestliže např. v uživatelském programu bude použita instrukce L DB20.DBW0?

# Cvičení 7.2: Testování DB (SFC 24: pouze S7-400)




| Parametr | Deklarace | Datový typ | Oblast | Popis |
|---|---|---|---|---|
| DB_NUMBER | INPUT | WORD | I,Q,M,D,L | Konstanta, číslo kontrolovaného DB |
| RET_VAL | OUTPUT | INT | I,Q,M,D,L | chybový kód |
| DB_LENGTH | OUTPUT | WORD | I,Q,M,D,L | délka kontrolovaného DB (v bytech) |
| WRITE_PROT | OUTPUT | BOOL | I,Q,M,D,L | 1 signalizuje aktivovaný atribut "Write protect" |





**Cíl cvičení**

S pomocí SFC 24 určit, zda je datový blok uložen v pracovní paměti CPU.

**Úkol**

Vytvořit FC72, které pomocí SFC 24 bude zjišťovat, zda DB je v editační paměti, pracovní paměti nebo v paměti vůbec není uložen:

- FC 72 očekává ve vstupním parametru "DB_NUM" (WORD) číslo testovaného DB.
- ve výstupním parametru RET_VAL (INT) vrací místo uložení DB:
  - 1: DB je v editační paměti
  - 0: DB je v pracovní paměti
  - -1: DB není nikde v paměti

**Poznámka**

Výstupní parametr SFC 24 RET_VAL vrací následující informace:

- w#16# 0000 žádná chyba nezachycena
- w#16# 80A1 neplatné číslo DB v parametru DB_NUMBER (0 nebo > max. DB číslo)
- w#16# 80B1 DB v paměti CPU neexistuje
- w#16# 80B2 DB byl vytvořen s atributem "unlinked" (a je uložen v editační paměti)

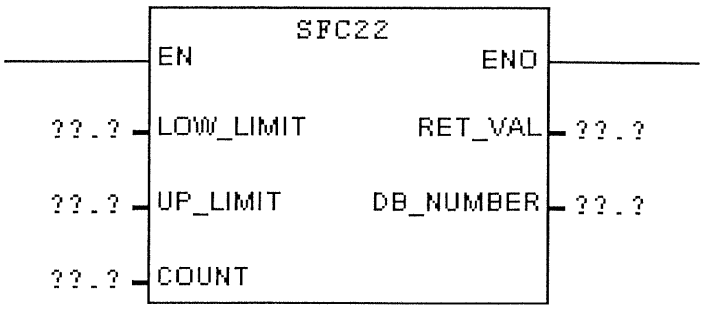
**Postup**

1. Vytvořit FC 72
2. Vytvořit OB1, který s pomocí FC 72 kontroluje existenci DB 20. Výsledek kontroly bude zobrazen na displeji simulátoru.
3. Nahrát bloky do CPU a otestovat funkci programu.

**Poznámka**

Systémová funkce je dostupná pouze v S7-400!

# Cvičení 7.3: Tvorba DB (SFC 22)




| Parametr | Deklarace | Datový typ | Oblast | Popis |
|---|---|---|---|---|
| LOW_LIMIT | INPUT | WORD | I,Q,M,D,L, konst. | číslo 1.vytvářeného DB |
| UP_LIMIT | INPUT | WORD | I,Q,M,D,L, konst. | číslo posledního DB |
| COUNT | INPUT | WORD | I,Q,M,D,L, konst. | počet bytů v DB, sudé číslo |
| RET_VAL | OUTPUT | INT | I,Q,M,D,L | hlášení SFC bloku |
| DB_NUMBER | OUTPUT | WORD | I,Q,M,D,L | počet vytvořených DB |

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_07cz.14

Školicí středisko
firmy E&A spol. s r.o., MB

**Cíl cvičení** Seznámit se s programovou tvorbou nového DB.

**Úkol** Startovním blokem OB100, vytvořit v pracovní paměti DB 10.

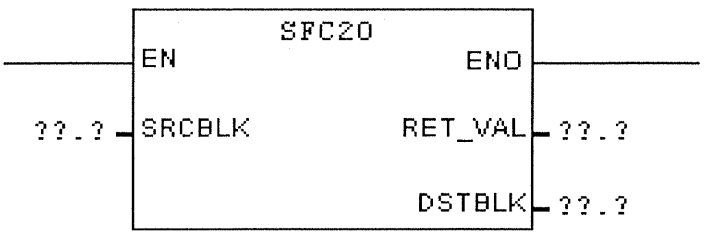
**Postup**

1. Vytvořit OB 100.
2. S použitím SFC22 a OB100 vytvořit DB 10 o délce 20 slov. Výstupní parametr RET_VAL vyhodnocovat prostřednictvím MW 0 a hodnotu parametru "DB_NUMBER" zobrazit na displeji
3. Nahrát bloky do CPU a otestovat funkci programu.

**Poznámka** SFC 22 vrací v RET_VAL následující hlášení:

- W#16# 0000 bez chyby
- W#16# 8091 překročena hloubka vnoření
- W#16# 8092 komprese paměti momentálně aktivní
- W#16# 80A1 chybné číslo DB
- W#16# 80A2 chybná délka
- W#16# 80B1 nedostupné číslo DB, DB již existuje
- W#16# 80B2 nedostatek paměti
- W#16# 80B3 nedostatek volné paměti, nutná komprese paměti

# Cvičení 7.4: Kopírování DB z editační do pracovní paměti (SFC 20)




| Parametr | Deklarace | Datový typ | Oblast | Popis |
|---|---|---|---|---|
| SRCBLK | INPUT | ANY | I,Q,M,D,L | zdrojová oblast |
| RET_VAL | OUTPUT | INT | I,Q,M,D,L | hlášení SFC |
| DSTBLK | OUTPUT | ANY | I,Q,M,D,L | cílová oblast |




Datum: 24.11.2002
Soubor: PRO2_07cz.15



**Cíl cvičení**

Seznámit se s možností přenosu bloku, s blokem SFC20.

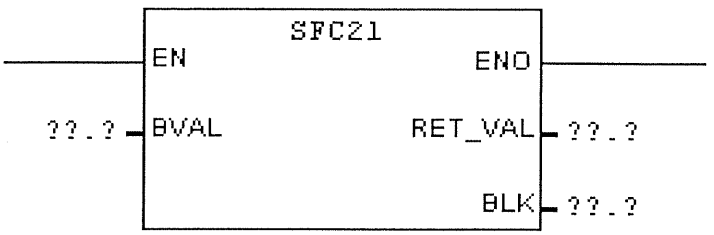
**Definice**

Technologické hodnoty (DW0-DW38) přenést z DB 20 (v editační paměti) do DB 10 (DW0-DW38) v pracovní paměti. Kopírování aktivovat náběžnou hranou signálu I1.0.

**Postup**

1. Vytvořit OB1, které zkopíruje hodnoty z DB 20 do DB 10 s pomocí SFC20 (Block transfer).
2. Vrácenou hodnotu v RET_VAL zobrazit na display simulátoru.
3. Nahrát program do CPU a otestovat funkčnost programu.

# Cvičení 7.5: Inicializace DB (SFC 21: FILL)




| Parametr | Deklarace | Datový typ | Oblast | Popis |
|---|---|---|---|---|
| BVAL | INPUT | ANY | I,Q,M,D,L | Inicializační hodnota |
| RET_VAL | OUTPUT | INT | I,Q,M,D,L | hodnota vracená SFC |
| BLK | OUTPUT | ANY | I,Q,M,D,L | cílová oblast vyplněná hodnotou BVAL |







**Cíl cvičení:** Seznámení se s touto systémovou funkcí.

**Definice**

Vytvořit FC 75 který inicializuje definovaný DB.
Funkce FC 75 je následující:

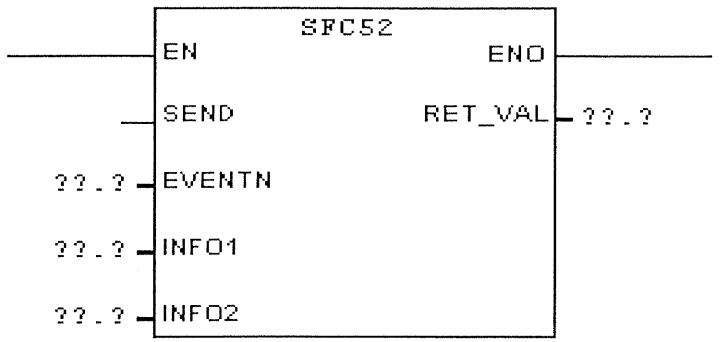
- FC 75 má deklarovány následující vstupní parametry:
  - DB_No (WORD): číslo inicializovaného DB
  - Ini_Byte (BYTE): inicializační hodnota, kterou bude DB vyplněn
- FC 75 nejprve zkontroluje, zda DB existuje v pracovní paměti CPU.Jestliže existuje je určena jeho délka

  Následně FC75 inicializuje DB definovanou hodnotou.
- FC 75 vrací v RET_VAL (BOOL) výsledek operace
  - 1: DB byl úspěšně inicializován.
  - 0: DB nebyl inicializován, tzn. DB není v pracovní paměti CPU.

**Postup**

1. Vytvořit FC 75 s popsanou funkcí.
2. Začlenit FC 75 do OB1 tak, aby DB 10 byl inicializován hodnotou "0" s náběžnou hranou vstupu I1.1.
3. Nahrát program do CPU a otestovat funkci programu.

# Cvičení 7.6: Zápis hlášení do diagnostického bufferu (SFC 52)




| Parametr | Deklarace | Datový typ | Oblast | Popis |
|---|---|---|---|---|
| SEND | INPUT | BOOL | I,Q,M,D,L, konst. | uvolnění odeslání |
| EVENTN | INPUT | WORD | I,Q,M,D,L, konst. | číslo, typ události (event ID) |
| INFO1 | INPUT | ANY | I,Q,M,D,L | zapisovaná informace , 2 byty |
| INFO2 | INPUT | ANY | I,Q,M,D,L | -----//----, 4 byty |
| RET_VAL | OUTPUT | INT | I,Q,M,D,L | hodnota vracená SFC |

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_07cz.17

Školicí středisko
firmy E&A spol. s r.o., MB

**Cíl cvičení**

Zpřístupnit uživatelský zápis do diagnostického bufferu.

**Definice**

Vytvořit FC 76 s následující funkcí:

- Při zachycení systémové chyby (simulované hranou I1.2) je do diagnostického bufferu zapsáno hlášení. Toto hlášení může být přečteno pomocí PG.

**Postup**

1. Vytvořit FC 76, který zapíše hlášení do diagnostického bufferu. Systémová chyba je simulována hranou I 1.2.
2. V SIMATIC Manageru aktivovat funkci "CPU Messages"
3. Z OB1 vyvolat FC 76 a otestovat funkci programu.

**Poznámka**

SFC 52 má následující parametry:

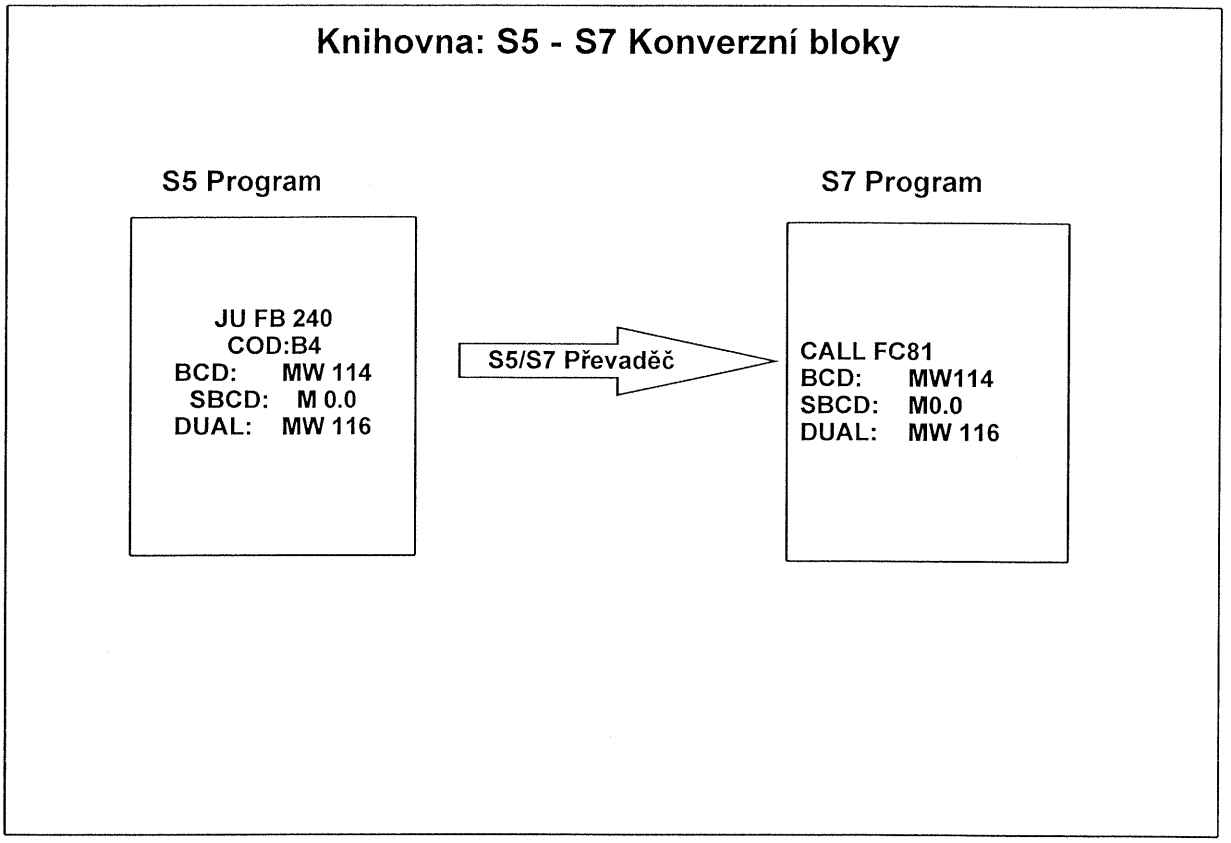
- EVENTN W#16# 9B0A (stavové nesrovnalosti, zachycená událost, externí chyba, vstup do diagnostického bufferu)
- INFO1 W#16# 8 (např. přepínač)
- INFO2 DW#16# 1 (např. přepínač)

Identifikátor Event ID číslo 9 je dostupný pro uživatele.
(Detailní popis je v manuálu "System Functions and Standard Functions manual").

**Chybové kódy**

SFC 52 vrací v RET_VAL následující chyby :

- 8083 nepřípustný datový typ v INFO1
- 8084 nepřípustný datový typ v INFO2
- 8085 nepřípustná hodnota EVENTN
- 8086 nepřípustná délka INFO1
- 8087 nepřípustná délka INFO2
- 8091 žádný uzel nepřihlášen
- 8092 zápis není možný, diagnostický buffer je plný







Datum: 24.11.2002
Soubor: PRO2_07cz.18



**Úvod**

Tato knihovna obsahuje standardní S7 bloky nutné k převodu programu S5 na program S7. Tím je myšleno, že např. jestliže je v S5 programu použit FB 240 , je v S7 programu nahrazen blokem FC 81.

Protože převaděč pouze převádí volání na volání bloku FC81, musí být blok ručně zkopírován z knihovny do projektu.

**Obsah knihovny**

Bloky knihovny jsou rozděleny do následujících skupin:

- aritmetika v plovoucí čárce, např. sčítání, odečítání
- signální funkce
- integrované funkce např. převod BCD --> Dual
- základní logické funkce např. LIFO

**Popis**

Detailní popis převodu je popsán v "Converting from STEP 5 Programs".

**Popis knihovny**

Popis jednotlivých bloků je v "on-line" nápovědě *Help --> Help topics --> References --> Additional reference aids --> Help with S5/S7 functions.*

**Poznámka**

Tyto bloky stejně tak jako v S5 programu používají "šmíráky".

.

# Knihovna: TI - S7 Konverzní bloky (část 1.)

| Blok | Symbol | Popis |
|---|---|---|
| FC 80 | TONR | odstartování časovače jako pamatované zpožděné zapnutí |
| FC 81 | IBLKMOV | nepřímý přenos datové oblasti |
| FC 82 | RSET | reset oblasti M-bitů nebo I/O oblasti |
| FC 83 | SET | set oblasti M-bitů nebo I/O oblasti. |
| FC 84 | ATT | zápis hodnoty do tabulky. |
| FC 85 | FIFO | výstup první hodnoty z tabulky |
| FC 86 | TBL_FIND | hledání hodnoty v tabulce |
| FC 87 | LIFO | výstup poslední hodnoty z tabulky |
| FC 88 | TBL | tabulka |
| FC 89 | TBL_WRD | kopírování hodnoty z tabulky |
| FC 90 | WSR | uložení dat v posuvném registru |
| FC 91 | WRD_TBL | logická kombinace hodnoty s prvkem tab. a uložení |
| FC 92 | SHRB | posun bitu v posuvném registru |
| FC 93 | SEG | tvorba bitového vzoru pro 7 -segment display. |
| FC 94 | ATH | převod ASCII znaku --> hexačíslo |
| FC 95 | HTA | převod hexačísla --> ASCII znak |
| FC 96 | ENCO | set definovaný bit ve slově |
| FC 97 | DECO | čtení čísla nejméně významného bitu |
| FC 98 | BCDCPL | tvorba desítkového doplňku |
| FC 99 | BITSUM | počet setovaných bitů |





**FC 80** Funkce startuje čas jako pamatované zpožděné zapnutí. Funkce počítá čas, dokud aktuální hodnota ET nedosáhne přednastavené hodnoty PV nebo ji nepřekročí.

**FC 81** Tato funkce umožňuje přenos oblasti dat z jedné oblasti do druhé.
Parametry S_DATA a D_DATA typu "POINTER" definují obě oblasti.Délka oblastí je definována přes samostatné parametry.

**FC 82/83** Setuje bity v definované oblasti do "1" (FC 83) nebo do "0" (FC 82), jestliže je MCR bit =1. Je-li MCR bit = 0 stav bitů se nezmění.

**FC 84-FC92** Tyto funkce implementují tabulkové funkce, např. FIFO. Hodnoty jsou ve slovech a délka je nastavitelná

**FC 93-FC 99** Tato skupina poskytuje různé převodní funkce.

# Knihovna: TI - S7 Konverzní bloky (část 2.)

| Blok | Symbol | Popis |
|---|---|---|
| FC 100 | RSETI | Okamžitý reset výstupní oblasti. |
| FC 101 | SETI | Okamžitý set výstupní oblasti. |
| FC 102 | DEV | Standardní odchylka |
| FC 103 | CDT | Korelační datová tabulka |
| FC 104 | TBL_TBL | Tabulka logických operací |
| FC 105 | SCALE | Přepočet hodnoty |
| FC 106 | UNSCALE | Inverzní funkce k FC105 |
| FB 80 | LEAD_LAG | Lead/Lag algoritmus |
| FB 81 | DCAT | Diskrétní řízení přerušení |
| FB 82 | MCAT | Hlídání zpětné vazby |
| FB 83 | IMC | Porovnávání indexové matice |
| FB 84 | SMC | Scanner matice |
| FB 85 | DRUM | DRUM (sekvenční procesor) |
| FB 86 | PACK | Komprese datové tabulky |








**FC 100-FC 101** Je-li MCR bit =1 funkce (re)setují bity v definovaném rozsahu. Je-li bit MCR=0 funkce se neprovede.

**FC 102** Standardní odchylka (DEV) počítá standardní odchylku ze skupiny hodnot uložených v tabulce (TBL). Výsledek je uložen do parametru OUT. Standardní odchylka je určena podle vzorce :

$$\text{Standardní odchylka} = \sqrt{\frac{(N * SqSum) - Sum^2}{N * (N - 1)}}$$

kde:

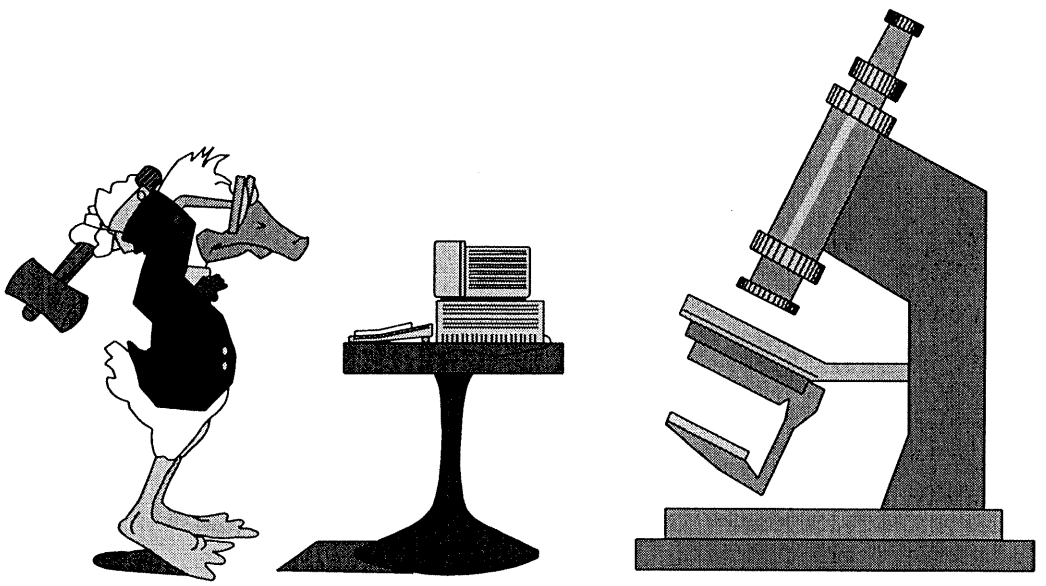
- Sum = součet hodnot v tabulce TBL
- N = počet hodnot v tabulce TBL
- SqSum = druhá mocnina součtu tabulky TBL

**FC 103** Korelační datová tabulka (CDT) porovnává vstupní hodnotu (IN) s existující tabulkou.

**FC 104-FC 105** Slouží ke zpracování analogových hodnot.

**FB 80- FB 86** viz. manuál.

# Zpracování synchronních a asynchronních chyb



SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_08cz.1

Školicí středisko
firmy E&A spol. s r.o., MB

**Obsah** **Strana**

## Zpracování asynchronních chyb

- **Asynchronní chyby nejsou přiřaditelné konkrétnímu místu v programu, při jehož zpracování byla chyba zachycena.**

| Typ chyby | Příklad | Chybové OB |
|---|---|---|
| Časová chyba | Překročení max. doby cyklu | OB 80 |
| Chyba napájení | Chyba zálohovací baterie | OB 81 |
| Diagnostické přerušení | Přerušení vedení u diagnostikovatelného modulu | OB 82 |
| Přerušení z odebrání/vložení | Odebrání signálového modulu v RUN režimu CPU S7-400 | OB 83 [1] |
| Chyba CPU | Chybná úroveň signálu MPI rozhraní | OB 84 [1] |
| Chyba programové sekvence | Chyba při aktualizaci obrazu processu (chybý modul) | OB 85 |
| Chyba nosiče modulů | Chyba napájecího zdroje rozšiřujícího nosiče modulů | OB 86 [1] |
| Komunikační chyba | Chybný identifikátor hlášení | OB 87 |

[1] pouze S7-400





**Úvod** Přehled zachycuje seznam asynchronních chyb, které nejsou přiřazené konkrétnímu místu v programu.

**Časová chyba** Standardně je max. doba cyklu nastavena na 150ms. Pokud zpracování cyklu trvá déle než max. doba cyklu, je vyhlášena časová chyba. Pokud je v jednom cyklu zachycena tato chyba dvakrát, přejde CPU do STOP stavu.

**Chyba zdroje** Tato chyba je hlášena, pokud dojde k výpadku zálohovací baterie a současně k výpadku 24V zdroje u CPU S7-400. Na rozdíl od ostatních chyb nejde CPU do STOP stavu.

**Diagnostické přerušení** Diagnostikovatelné moduly (např. analogové) mohou vyvolat, v případě chyby, přerušení. Tyto moduly však musí být vhodně parametrizovány.

**Přerušení odebráním nebo vložením** Toto přerušení je vyvoláno odebráním nebo vložením nového modulu do systému S7-400 při zpracování cyklu. Po vložení nového modulu si systém zkontroluje správnost typu vloženého modulu. Tato funkce umožňuje odebírat staré a vkládat nové moduly za běhu programu.

**Chyba CPU** Tatc chyba je hlášena u S7-400 v případě chyby na MPI rozhraní K-sběrnice nebo rozhraní rozšiřujícího I/O modulu.

**Chyba programové sekvence** Toto hlášení je výsledkem chyby přístupu při aktualizaci obrazu procesu nebo např. při chybějícím OB při parametrizaci "time-of-day" přerušení.

**Chyba nosiče modulů** je detekována při poruše nosiče modulů, sítě PLC nebo slave-stanice PROFIBUS-DP sítě.

**Komunikační chyba** Tuto chybu vyvolá chybný identifikátor hlášení v přijaté oblasti sdílených dat nebo když je přijímací datový blok příliš krátký pro uložené stavové informace. U S7-400 má tato chyba za následek znemožnění vysílání synchronizačního hlášení.

## Zpracování chybového organizačního bloku

- pro potlačení přechodu CPU do STP stavu lze programovat prázdný chybový OB blok.
- je-li to nutné, lze do chybového OB programovat požadovanou reakci na vzniklou chybu. Pro přechod do STOP stavu lze použít SFC 46.
- doplňkové informace o chybě jsou uloženy ve startovních informacích chybového OB bloku a mohou být programově vyhodnoceny.
- popis chybových OB bloků je v "on-line" nápovědě nebo v manuálu "System and Standard Function manual".





**Zpracování Start-Info**

Pro použití chybových OB platí několik pravidel.
Pro každý OB blok jsou v deklarační části definovány lokální proměnné.
Do těchto lokálních proměnných ukládá operační systém startovní informace.
Např:

| Name | Type | Initial Value | Comment |
|---|---|---|---|
| OB81_EV_CLASS | BYTE | | 16#39, Event class 3, Entering ev |
| OB81_FLT_ID | BYTE | | 16#XX, Fault identifcation code |
| OB81_PRIORITY | BYTE | | 26/28 (Priority of 1 is lowest) |
| OB81_OB_NUMBR | BYTE | | 81 (Organization block 81, OB81) |
| OB81_RESERVED_1 | BYTE | | Reserved for system |
| OB81_RESERVED_2 | BYTE | | Reserved for system |
| OB81_MDL_ADDR | INT | | Reserved for system |
| OB81_RESERVED_3 | BYTE | | Reserved for system |
| OB81_RESERVED_4 | BYTE | | Reserved for system |
| OB81_RESERVED_5 | BYTE | | Reserved for system |
| OB81_RESERVED_6 | BYTE | | Reserved for system |
| OB81_DATE_TIME | DATE_AND_TIME | | Date and time OB81 started |

Chybové identifikátory:

- B#16#21: Nejméně jedna záložní baterie centrálního nosiče je vadná (BATTF)
- B#16#22: Zálohovací napětí v centrálním nosiči chybuje (BAF).
- B#16#23: Chyba zdroje 24V v centrálním nosiči modulů.
- B#16#31: Nejméně jedna záložní baterie rozšiřujícího nosiče modulů je vadná.
- B#16#32: Záložní baterie jednoho z rozšiřujících modulů je vadná.
- B#16#33: Chyba zdroje 24V v rozšiřujícím nosiči modulů.

# Příklad asynchronního chybového OB

```
OB81:    Error OB: Power supply failure

Network 1:    Battery failure, coming event

      L     #OB81_FLT_ID          // Load error identifier
      L     B#16#22               // Identifier: Battery failure in CR
      ==I
      =     M     81.1            // Set auxiliary memory marker
      L     #OB81_EV_CLASS        // Identifier: coming, leaving
      L     B#16#39               // Identifier: coming event
      ==I
      =     M     81.2            // Aux. mem. marker coming event
      A     M     81.1            // Battery failure and
      A     M     81.2            // coming event
      S     M     81.0            // Set aux. mem. marker for error
                                  // display

Network 2:    Reset auxiliary memory marker, when battery O.K.

      L     #OB81_EV_CLASS        // Identifier: coming, leaving
      L     B#16#38               // Identifier: leaving
      ==I
      R     M     81.0            // Reset auxiliary memory marker
```





**Definice úlohy**

Výpadek záložní baterie má být zobrazen na operátorském displeji. Po výměně baterie má automaticky toto hlášení zhasnout.

**Popis**

Při zachycení chyby záložní baterie je volán chybový OB blok. Po odstranění chyby je tento blok volán podruhé.

V příkladu je vyhodnocována proměnná OB81_FLT_ID pro zachycení chyby baterie. tato situace je hlášena kódem 22H. Zachycení této hodnoty nastaví M 81.1.

Zobrazení hlášení je spuštěno příchodem události, zhasnutí je vyvoláno odchodem události vyhodnocené v proměnné OB81_EV_CLASS:

- B#16#38 příchozí událost
- B#16#38 odchozí událost.

Setování a resetování bitu M 81.0 je realizováno vyhodnocením typu události.

V cyklickém programu lze M81.0 použít k např. rozblikání signalizace chyby.

## Zpracování synchronní chyby

- Synchronní chyby jsou vyvolány zpracováním části uživatelského programu
- Chyby v aritmetických instrukcích (přetečení, neplatné REAL číslo )

  ⇒ Setuje stavové bity

- Chyby při zpracování STL instrukcí (synchronní chyby)

  ⇒ Vyvolá chybový OB blok

| Typ chyby | Příklad | Chybový OB blok |
|---|---|---|
| Programová chyba | Volání neexistujícího bloku | OB 121 |
| Přístupová chyba | Přímý přístup na vadný nebo neexistující signálový modul | OB 122 |





**Synchronní chyby**

Operační systém CPU vyhlásí synchronní chybu, když je spojena se zpracováním programu. OB121 je volán při programové chybě, OB122 při chybě přístupu. Není-li v CPU nahrán příslušný chybový blok, přejde CPU při zachycení synchronní chyby do STOP stavu.

Chybový OB blok má stejnou prioritu jako blok v němž došlo k chybě. Lze proto vyhodnocovat obsah registrů chybujícího bloku a pokud je to nutné, lze obsah registru, před návratem do přerušeného bloku, změnit.

**Maskování synchronních chyb**

S7 obsahuje SFC určené k maskování a odmaskování startovních informací OB 121:

- SFC36 "MSK_FLT": maskuje daný chybový kód
- SFC37 "DMSK_FLT": odmaskuje chybu maskovanou SFC36
- SFC38 "READ_ERR": čte chybový registr

# Startovní informace pro programovou chybu OB121

| Proměnná | Datový typ | Popis, přiřazení |
|---|---|---|
| OB121_EV_CLASS | BYTE | B#16#25 = volání programové chyby OB121 |
| OB121_SW_FLT | BYTE | Chybový kód |
| OB121_PRIORITY | BYTE | Prioritní třída |
| OB121_OB_NUMBER | BYTE | Číslo OB (B#16#79) |
| OB121_BLK_TYPE | BYTE | Typ přerušeného bloku(pouze S7-400)<br>OB:B#16#88, DB:B#16#8A, FB:B#16#8E, FC:B#16#8C |
| OB121_RESERVED_1 | BYTE | doplňkový chybový kód (viz. text) |
| OB121_FLT_REG | WORD | OB121: zdroj chyby |
| OB121_BLK_NUM | WORD | Číslo bloku ve kterém byla chyba zachycena |
| OB121_PRG_ADDR | WORD | Adresa chyby v bloku (pouze S7-400) |
| OB121_DATE_TIME | DT | Časová značka, kdy došlo k chybě |





**Chybové kódy OB121_SW_FLT**

B#16#21: chyba konverze BCD. OB121_FLT_REG obsahuje identifikátor registru (W#16#0000: ACCU 1).
B#16#22: chybná délka oblasti při čtení
B#16#23: chybná délka oblasti při zápisu
B#16#28: chyba při nepřímo adresovaném čtení typu BYTE, WORD nebo DWORD s nenulovou adresou.
B#16#29: chyba při nepřímo adresovaném zápisu do typu BYTE, WORD nebo DWORD s nenulovou adresou.

V tomto případě obsahuje OB121_FLT_REG chybnou adresu a OB121_RESERVED_1 obsahuje identifikátor chybující oblasti:

| Bit 7 až 4 (typ přístupu): | Bit 3 až 0 (oblast) | |
|---|---|---|
| 0: Bit přístup | 0: I/O oblast | 4: globální DB |
| 1: Byte přístup | 1: PII | 5: instanční DB |
| 2: Word přístup | 2: PIQ | 6: vlastní lokální data |
| 3: Double word přístup | 3: M-Bity | 7: lokální data volajícího |

B#16#24: chyba rozsahu při čtení
B#16#25: chyba rozsahu při zápisu

OB121_FLT_REG obsahuje identifikátor B#16#86: vlastní lokální data.

B#16#26: chybné číslo časovače (chybné číslo je v OB121_FLT_REG)
B#16#27: chybné číslo čitače (chybné číslo je v OB121_FLT_REG)

B#16#30: zápis do "write-protect" globálního DB (číslo je v OB121_FLT_REG)
B#16#31: zápis do "write-protect" instančního DB (číslo je v OB121_FLT_REG)
B#16#32: chybné číslo globálního DB (číslo je v OB121_FLT_REG)
B#16#33: chybné číslo instančního DB (číslo je v OB121_FLT_REG)
B#16#34: chybné číslo volaného FC (číslo je v OB121_FLT_REG)
B#16#35: chybné číslo volaného FB (číslo je v OB121_FLT_REG)
B#16#3A: přístup k neexistujícímu DB (číslo je v OB121_FLT_REG)
B#16#3C: přístup k neexistujícímu FC (číslo je v OB121_FLT_REG)
B#16#3D: přístup k neexistujícímu SFC (číslo je v OB121_FLT_REG)
B#16#3E: přístup k neexistujícímu FB (číslo je v OB121_FLT_REG)
B#16#3F: přístup k neexistujícímu SFB (číslo je v OB121_FLT_REG)

# Startovní informace pro chybu přístupu OB122

| Proměnná | Datový typ | Popis, přiřazení |
|---|---|---|
| OB122_EV_CLASS | BYTE | B#16#29=chyba při volání |
| OB122_SW_FLT | BYTE | Chybový kód ( B#16#42, B#16#43, B#16#44, B#16#45) |
| OB122_PRIORITY | BYTE | Prioritní třída, kde došlo k chybě |
| OB122_OB_NUMBR | BYTE | Číslo OB (B#16#80) |
| OB122_BLK_TYPE | BYTE | Typ přerušeného bloku (pouze S7-400)<br>OB: B#16#88, DB: B#16#8A, FB: B#16#8E, FC: B#16#8C |
| OB122_MEM_AREA | BYTE | Doplňkový chybový kód (viz. text) |
| OB122_FLT_REG | WORD | OB122: Adresa chybujícího operandu |
| OB122_BLK_NUM | WORD | Číslo chybujícího bloku |
| OB122_PRG_ADDR | WORD | Adresa chyby v bloku (pouze S7-400) |
| OB122_DATE_TIME | DT | Časová známka chyby |



Datum: 24.11.2002
Soubor: PRO2_08cz.7



**Chybové kódy** OB122_SW_FLT má následující význam:

B#16#42 S7-300: chyba při čtení I/O
S7-400: první čtení po zachycení chyby

B#16#43: S7-300: chyba při zápisu I/O
S7-400: první zápis po zachycení chyby

B#16#44: pouze S7-400: chyba při n-tém (n>1) čtení po výskytu chyby

B#16#45: pouze S7-400: chyba při n-tém (n>1) zápisu po výskytu chyby .

**OB122_MEM_AREA** Tato proměnná obsahuje informace o typu přístupu a typu oblasti:

Bit 7 až 4 typ přístupu:

0: Bit přístup
1: Byte přístup
2: Word přístup
3: Double word přístup

Bit 3 až 0 oblast:

0: I/O oblast
1: Obraz vstupů PII
2: Obraz výstupů PIQ

## Maskování synchronních chyb

- **Nevýhody synchronních chybových OB:**
  - Program pro zpracování chyb a řízení procesu je minimálně ve dvou blocích
  - Problémy se změnami nebo s údržbou
- **Výhody maskování:**
  - Program pro proces a vyhodnocení chyby jsou ve stejném bloku
- **Maskování synchronních chyb:**
  - Před "nebezpečnými" instrukcemi:
    SFC 36 MSK_FLT: maskování synchronních chyb
    (zablokování volání OB12x)
  - Provedení "nebezpečných" instrukcí
  - Vyhodnocení zpracování
    SFC 38 READ_ERR: čtení chybového registru
  - Povolení volání OB12x:
    SFC 37 DMSK_FLT: demaskování synchronních chyb





**Nevýhody synchron. chybových OB**

Zpracování synchronních chyb pomocí chybových synchronních OB bloků má několik nevýhod:

- Propracované zpracování chyb musí zachytit všechny varianty synchronních chyb v celém programu - ve všech blocích.

  Synchronní chybové OB bloky musí tedy nejprve lokalizovat místo synchronní chyby a potom odpovídajícím způsobem zareagovat.
- Každá změna v existujícím bloku tedy musí být doprovázena změnou v v synchronním chybovém OB.
- Bloky nemohou být začleněny do programu bez odpovídajícího chybového OB.

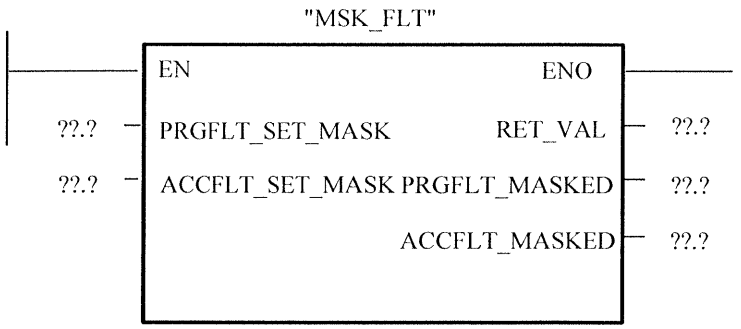
**Varianta k chybovým synchronním OB**

S7 nabízí pro řešení tohoto problému funkci "Masking Synchronous Errors" - maskování synchronních chyb, mechanismus, který umožňuje sloučit program pro řízení procesu a zpracování chyby do jednoho bloku.
Postup je následující:

1. Před zpracováním "kritických" instrukcí (např. otevření DB nebo přístup k DB s neznámou délkou ) lze odpovídající synchronní chyby maskovat použitím SFC 36 MSK_FLT.

   Pokud dojde při zpracování instrukcí k chybě, nedojde k vyvolání chybového OB bloku.
2. Po zpracování kritických instrukcí lze uživatelsky vyhodnotit pomocí SFC 38 READ_ERR, zda nedošlo k chybě.
3. Po uzavření této části programu lze synchronní chyby odmaskovat a uvolnit tak opět volání synchronního chybového OB.

## SFC 36 pro maskování synchronních chyb




| Parametr | Deklarace | Datový typ | Oblast | Popis |
|---|---|---|---|---|
| PRGFLT_SET_MASK | INPUT | DWORD | I, Q, M, D, L, konst. | doplňkový programový chybový filtr |
| ACCFLT_SET_MASK | INPUT | BYTE | I, Q, M, D, L, konst. | doplňkový přístupový chybový filtr |
| RET_VAL | OUTPUT | INT | I, Q, M, D, L | výsledek zpracování SFC, W#16#0001: nový filtr překryl filtr stávající |
| PRGFLT_MASKED | OUTPUT | DWORD | I, Q, M, D, L | dokončený programový chybový filtr |
| ACCFLT_MASKED | OUTPUT | DWORD | I, Q, M, D, L | dokončená přístupový chybový filtr |

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_08cz.9

Školicí středisko
firmy E&A spol. s r.o., MB

**Maskování synchronních chyb**

S pomocí SFC 36 MSK_FLT lze zakázat volání synchronního chybového OB aktivací chybových filtrů. "1" v tomto filtru zakazuje volání OB při výskytu dané chyby - chyba je maskována.

Požadované maskování je provedeno ve spojení s maskováním uloženým v operačním systému (logický OR mezi bity filtru). SFC36 signalizuje pomocí hodnoty v RET_VAL, zda pro požadovaný filtr je nastaven nejméně jeden bit (W#16#0001).

SFC36 vrací skutečný stav masky filtru ve výstupním parametru.

**Reakce CPU**

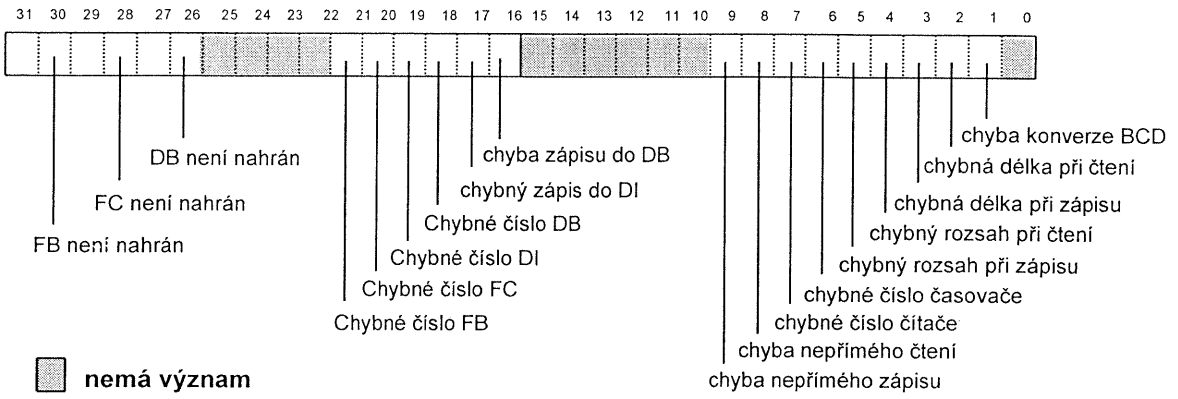
Jestliže je programová nebo přístupová chyba maskována, CPU reaguje následovně:

1. při programové nebo přístupové chybě není chybový OB volán
2. chyba je zapsána do chybového registru. Stav registru lze číst pomocí SFC38 READ_ERR.
3. operační systém zaznamená chybu do diagnostického bufferu bez ohledu na stav maskování.

**Maskovací podmínka**

Maskování platí pouze pro tu prioritní třídu, ve které je SFC36 volán. Jestliže např. je maskování provedeno z hlavního programu - OB1, synchronní chyba vzniklá při zpracování alarmového programu bude stále vyvolávat synchronní chybový OB blok.

## Struktura programového chybového filtru




**Poznámka: Odpovídající bity ve výstupním parametru PRGFLT_MASKED mají následující význam:**
**"1": chyba maskována**
**"0": chyba není maskována**
**Nepoužité bity jsou nastaveny do "1".**



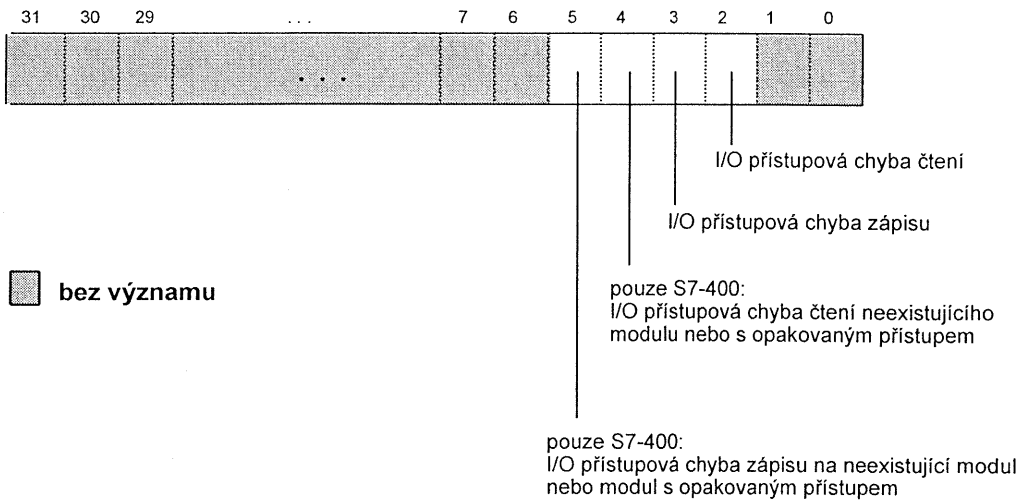
Datum: 24.11.2002
Soubor: PRO2_08cz.10



**Programový chybový filtr**

Uživatel může řídit zpracování synchronních chyb pomocí maskovacích filtrů. Pomocí těchto filtrů lze chyby maskovat nebo odmaskovat.

## Struktura přístupového chybového filtru




**Poznámka: Význam bitů na výstupu ACCFLT_MASKED je následující:**

**"1": chyba maskována.**

**"0": chyba nemaskována**

**Nepřiřazené bity jsou nastaveny do "1".**

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_08cz.11

Školicí středisko
firmy E&A spol. s r.o., MB

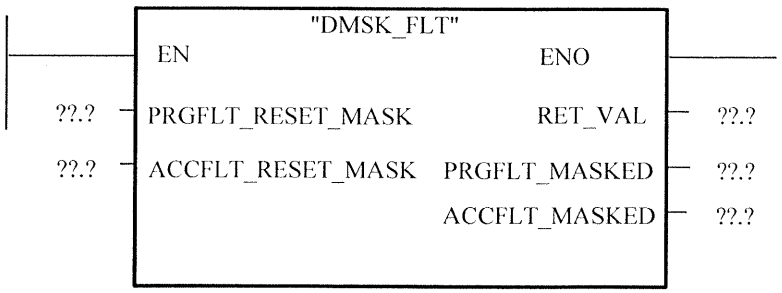
**Přístupový chybový filtr**

S7-400 rozlišuje 2 typy přístupových chyb. Chybný přístup k existujícímu modulu a chybu přístupu k neexistujícímu modulu.

Jestliže modul chybuje při přístupu, je detekován "time-out" (QVZ). Pokud modul není v systému přítomen, každý pokus o přístup vyvolá chybu PZF.

Přístupová chyba je hlášena jak při přímém přístupu tak při přístupu nepřímou adresací.

# SFC 37 - odmaskování synchronních chyb




| Parametr | Deklarace | Datový typ | Oblast | Popis |
|---|---|---|---|---|
| PRGFLT_RESET_MASK | INPUT | DWORD | I, Q, M, D, L, Const. | Programový chybový filtr pro reset |
| ACCFLT_RESET_MASK | INPUT | BYTE | I, Q, M, D, L, Const. | Přístupový chybový filtr pro reset |
| RET_VAL | OUTPUT | INT | I, Q, M, D, L | Výsledek SFC, W#16#0001: nový filtr obsahuje bity nenastavené ve filtru |
| PRGFLT_MASKED | OUTPUT | DWORD | I, Q, M, D, L | Stará hodnota programového filtru |
| ACCFLT_MASKED | OUTPUT | DWORD | I, Q, M, D, L | Stará hodnota přístupového filtru |

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_08cz.12

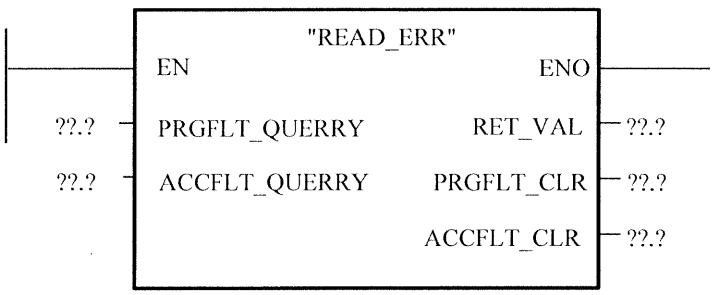
Školicí středisko
firmy E&A spol. s r.o., MB

**Odmaskování synchronních chyb**

SFC37 DMSK_FLT uvolňuje opět volání chybového OB. "1" definuje odmaskovanou chybu, při jejímž výskytu bude opět volán chybový OB. Obsah chybového registru je pro danou chybu vymazán.

Na výstupních parametrech vrací SFC37 současný stav maskování - maskovaná událost má ve filtru "1".

# SFC 38 - čtení chybového registru




| Parametr | Deklarace | Datový typ | Oblast | Popis |
|---|---|---|---|---|
| PRGFLT_QUERRY | INPUT | DWORD | I, Q, M, D, L, konst. | Programový filtr pro kontrolu |
| ACCFLT_QUERRY | INPUT | BYTE | I, Q, M, D, L, konst. | Přístupový filtr pro kontrolu |
| RET_VAL | OUTPUT | INT | I, Q, M, D, L | Výsledek SFC , W#16#0001: filtr obsahuje nenastavené bity |
| PRGFLT_CLR | OUTPUT | DWORD | I, Q, M, D, L | Programový filtr s chybovým hlášením |
| ACCFLT_CLR | OUTPUT | DWORD | I, Q, M, D, L | Přístupový filtr s chybovým hlášením |



Datum: 24.11.2002
Soubor: PRO2_08cz.13



**Čtení chybového registru**

SFC38 READ_ERR čte stav chybového registru.
"1" ve filtru určuje kontrolovanou chybu

SFC38 vrací v parametrech hodnotu "1" u zachycené události, která je touto kontrolou vymazána z chybového registru.

# Příklad: testování datového bloku

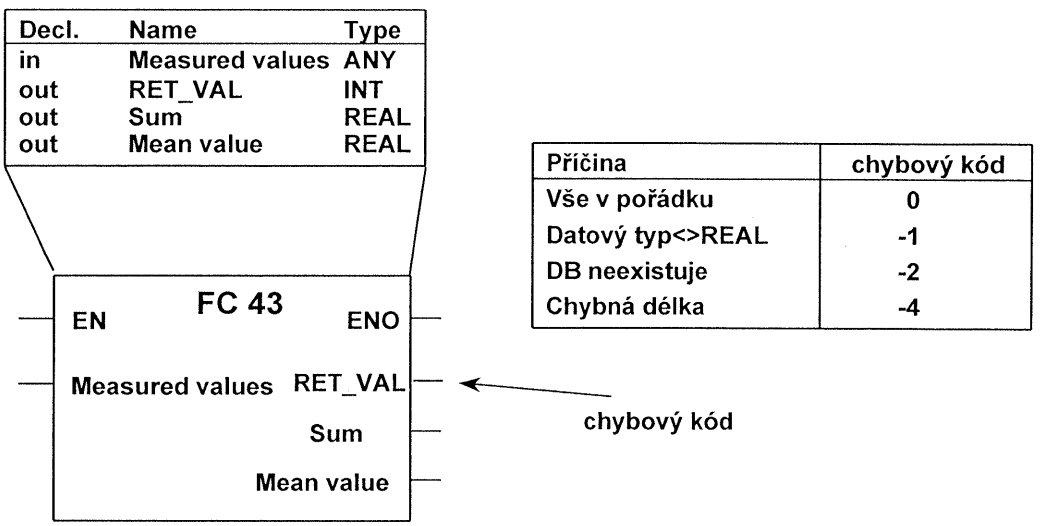
```
Network 1: Masking, Testing, Demasking
// Mask "DB does not exist"
      CALL SFC  36(
        PRGFLT_SET_MASK  := DW#16#4000000,        // Identifier: DB does not exist
        ACCFLT_SET_MASK  := DW#16#0,      // no masking for access errors
        RET_VAL          := SFC36Error,
        PRGFLT_MASKED    := Prog36Mask,
        ACCFLT_MASKED    := Zugr36Mask);
// Test call
      OPN DB[DB_NO];
// Check programming error
      CALL SFC  38(
        PRGFLT_QUERRY    := DW#16#4000000,        // Identifier: DB does not exist
        ACCFLT_QUERRY    := DW#16#0,      // no masking for access errors
        RET_VAL          := SFC38Error,
        PRGFLT_MASKED    := Prog38Mask,
        ACCFLT_MASKED    := Zugr38Mask);

// Evalute result
      L    Prog38Mask
      L    DW#16#4000000
      ==D
      =    DB_NOT_THERE                 // Set auxiliary variable DB not there

// Demask "DB does not exist"
CALL  SFC  37(
        PRGFLT_RESET_MASK          := DW#16#4000000,      // Identifier: DB does not
exist
        ACCFLT_RESET_MASK          := DW#16#0,            // no masking for access
errors
        RET_VAL          := SFC37ERROR,
        PRGFLT_MASKED    := Prog37Mask,
        ACCFLT_MASKED    := Zugr37Mask);
```



Datum: 24.11.2002
Soubor: PRO2_08cz.14

Školicí středisko
firmy E&A spol. s r.o., MB

# Cvičení 8.1: Zpracování chyby v FC43




| Příčina | chybový kód |
|---|---|
| Vše v pořádku | 0 |
| Datový typ<>REAL | -1 |
| DB neexistuje | -2 |
| Chybná délka | -4 |





**Přehled**

Ve cvičení 4.3 byl vytvořen blok FC43 určující celkový součet a průměrnou hodnotu z pole REAL čísel. Tento FC blok realizuje základní kontrolu datového typu.

Zpracování chyby je nyní rozšířeno tak aby chybná parametrizace nevolala chybový OB blok.

**Cíl**

Vytvoří FC43 s následující funkcí:

- je-li datový typ různý od REAL skončí FC s chybovým kódem "-1".
- je-li zadáno neplatné číslo DB (mimo rozsah nebo neexistujícího bloku) skončí FC s kódem "-2".
- je-li detekován přístup na neexistující adresu skončí FC s kódem "-4".
- ve všech ostatních chybných případech nastaví FC 43 stavový bit BR-Bit na hodnotu "0" a vrací neplatné REAL číslo v parametrech "Sum" a "Mean Value".

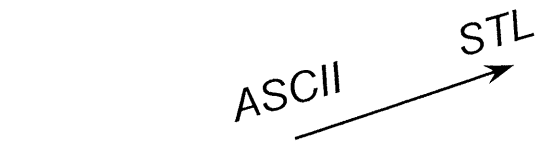
**Postup**

1. Rozšířit FC43 o výstupní parametr RET_VAL (chybový kód).
2. V FC43 provést odpovídající zpracování chyby
3. FC43 vyvolat z OB1.
4. Nahrát bloky do CPU a otestovat funkci programu.

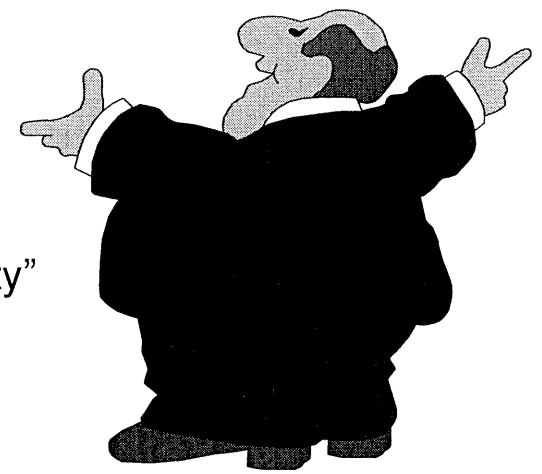
**Otázka?**

Které možné synchronní chyby nejsou v tomto programu vzaty do úvahy?

# Tvorba programu textovým editorem




"Ochranné atributy"



SIMATIC S7

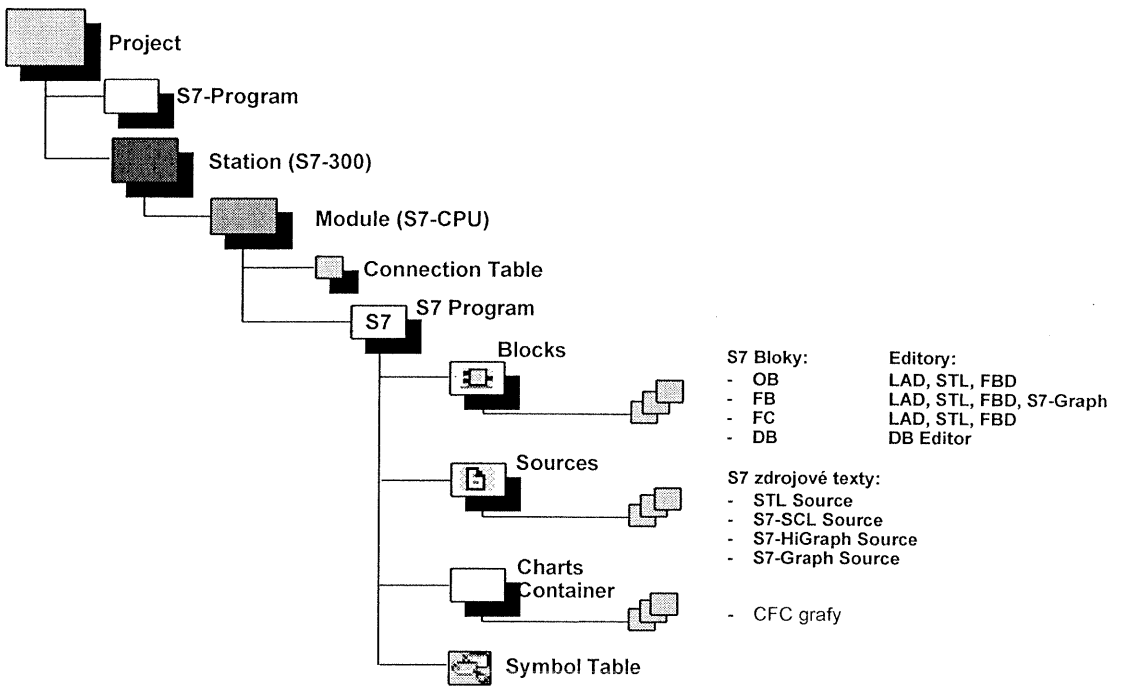


Datum: 24.11.2002
Soubor: PRO2_09cz.1

Školicí středisko
firmy E&A spol. s r.o., MB

**Obsah** **Strana**

## S7 Programy ve struktuře projektu




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_09cz.2

Školicí středisko
firmy E&A spol. s r.o., MB

**Přehled**

Aby bylo možné řádně vytvořit nový S7-program, musí být nejprve vytvořen v SIMATIC Manageru projekt. Následně lze vytvořit složku S7-programu

- nezávislou na modulu: v tomto případě je programová složka vytvořena jako přímo podřízená projektu. Program lze později podřídit konkrétnímu modulu CPU.
- závislou na modulu: v tomto případě je nejprve nutno vytvořit složku stanice (SIMATIC 300/400) s definovaným CPU. Složka S7-programu je automaticky vytvořena také, jako podřízená složka.

**Bloky, Zdroje a Grafy**

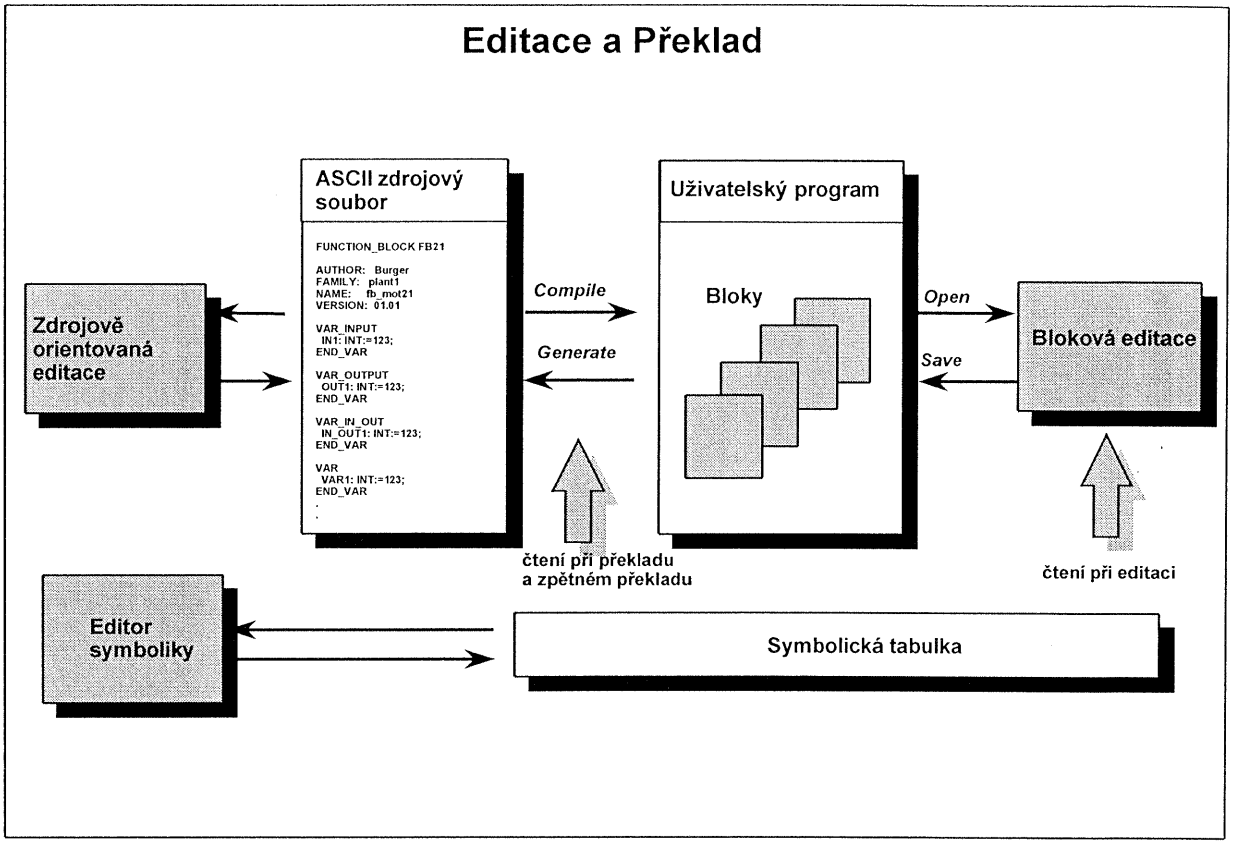
Uživatelský program lze uložit ve formě bloku, zdrojového textu nebo grafu Zdrojové texty a grafy jsou přitom použity pouze jako zdroj pro vytvoření S7-bloků. Pouze bloky mohou být nahrány do CPU.

Zda je vytvořen blok, zdrojový text nebo graf záleží na vybraném programovacím jazyku a použitém editoru.

**Uživatelský program**

Do CPU lze nahrát pouze bloky uživatelského programu, který podle potřeby obsahuje organizační bloky (OB), funkce (FC, funkční bloky (FB) a datové bloky (DB).

Bloky typu UDT, usnadňující tvorbu programu, nelze do CPU nahrát, stejně jako bloky typu VAT, které obsahují seznam adres sledovaných funkcí *Monitor/Modify Variables*.




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_09cz.3



**Editační možnosti**

V závislosti na programovacím jazyku lze volit mezi blokově orientovanou editací a zdrojově orientovanou editací.

- Bloková editace (STL, LAD, FBD, S7-Graph, S7-HiGraph, CFC)

  Každý prvek vkládaný do programu je ihned kontrolován na syntaktickou chybu. Pokud je zjištěna chyba, je daný prvek označen červeně a před uložením je nutné syntaktickou chybu odstranit.

  Syntakticky správně zadané prvky jsou ihned přeloženy a zpětně zobrazeny. Při použití symbolických názvů prvků musí být tyto názvy nejprve definovány v symbolické tabulce, jinak je symbol označen červeně a ve stavovém řádku je zobrazeno odpovídající chybové hlášení.

- Zdrojově orientovaná editace (STL, S7-SCL)

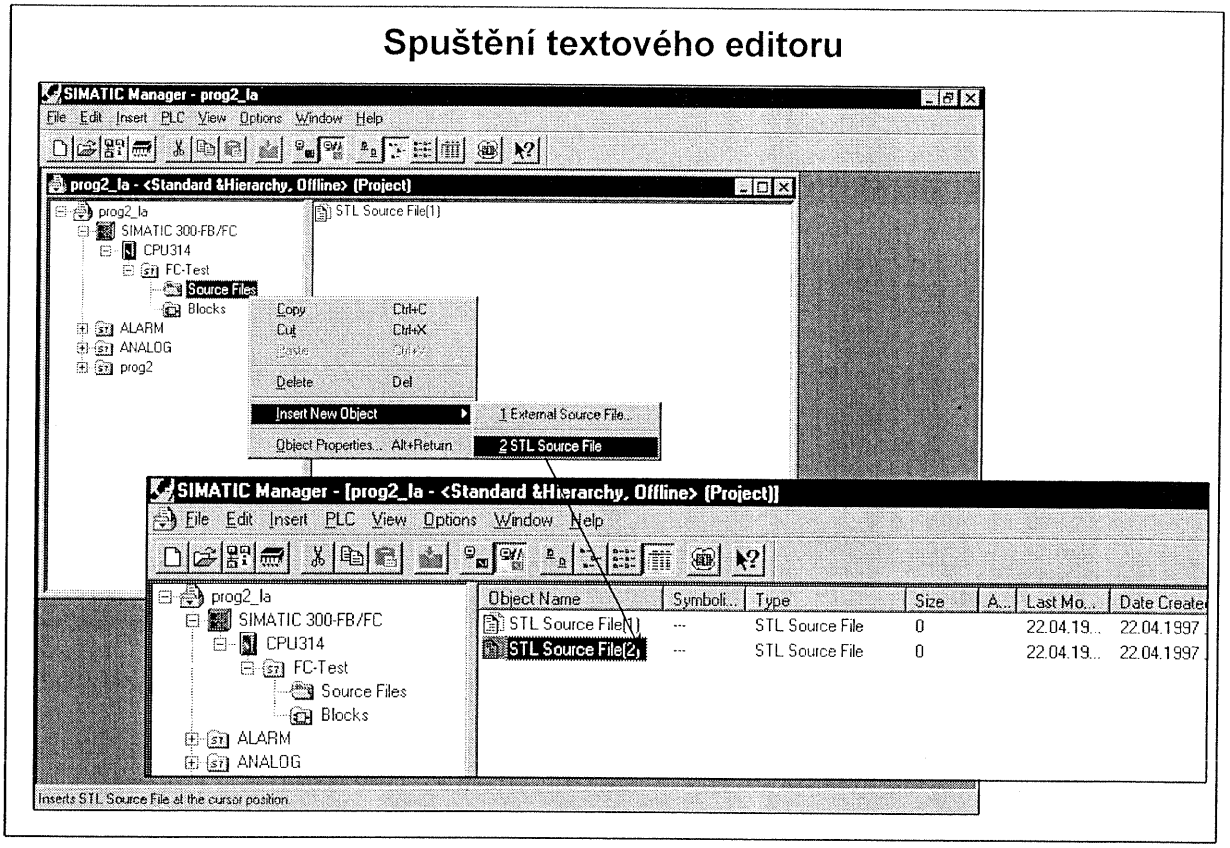
  Při zdrojové editaci je blok nebo program nejprve vytvořen pomocí příslušného editoru a uložen jako zdrojový text. Před začleněním této části do programu je nejprve nutné provést překlad pomocí příslušného překladače. Při překladu jsou kontrolována syntaktická pravidla a pokud je zjištěna chyba je signalizována a je nutné ji odstranit.

  Při použití symbolických operandů ve zdrojovém textu musí být před provedením překladu definovány příslušné symboly v tabulce symboliky. Zdrojové soubory mají tu výhodu, že mohou být exportovány a následně zpracovávány pomocí jiného textového editoru. Výsledek editace lze snadno importovat zpět do S7-programu.

**Výhody zdrojových textů**

- ve jednom zdrojovém textu může být uloženo více bloků i se syntaktickými chybami.
- zdrojový soubor může být vytvořen pomocí oblíbeného textového editoru, následně importován do projektu a přeložen.
- ochranné atributy bloku lze definovat pouze ve zdrojové podobě bloku.
- změny (např. přidání parametru bloku) jsou snadněji proveditelné než v blokově orientovaném editoru.
- hotový program lze zpětně přeložit do formy zdrojového textu a exportovat do souboru se kterým se snadněji manipuluje.

# Spuštění textového editoru







Datum: 24.11.2002
Soubor: PRO2_09cz.4



**Spuštění ze SIMATIC Manageru**

Spuštění textového editoru lze provést i ze SIMATIC Manageru. Předpokladem je, že je vytvořen projekt se složkou S7-Program. Je jedno zda je program vytvořený jako (ne)závislý na modulu.

Pomocí textového editoru lze vytvořit textový soubor, který bude teprve následně přeložen a vytvořené bloky budou umístěny do složky *Blocks*.

**Vytvoření souboru**

Vytvoření nového zdrojového souboru lze provést jak ze SIMATIC Manageru tak z Text Editoru. K tvorbě zdrojových textů je ale nezbytně nutná složka *Source Files*. Postup tvorby je následující:

- V SIMATIC Manageru vybrat složku *Source Files,* vyvolat pravým tlačítkem myši short-menu a aktivovat nabídku *Insert New Object -> STL Source File.* V pravé části okna projektu se objeví nový objekt se zvoleným jménem.
- V Text Editoru stačí k vytvoření nového zdrojového souboru vyvolat nabídku *File ->New* a v následujícím dialogovém okně vyplnit potřebné údaje.

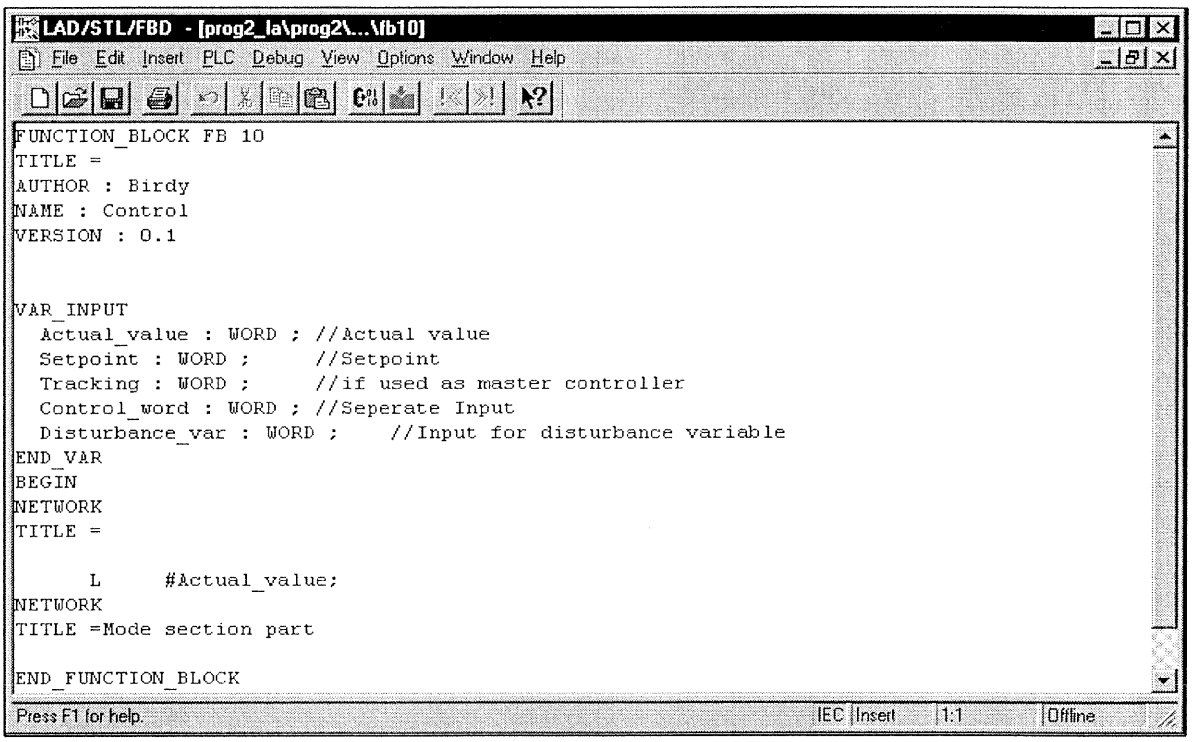
**Otevření souboru**

K otevření souboru SIMATIC Manageru slouží nabídka *Edit -> Open* nebo stačí dvojklik *na zvoleném objektu.*

**Generování souboru**

Zdrojový souboru lze vytvořit i z již hotového bloku. K zpětnému překladu slouží v Text Editoru nabídka *File -> Generate Source.* Po zadání potřebných údajů bude vytvořen zdrojový text se zvolenými bloky.

# Tvorba programu textovým editorem







Datum: 24.11.2002
Soubor: PRO2_09cz.5



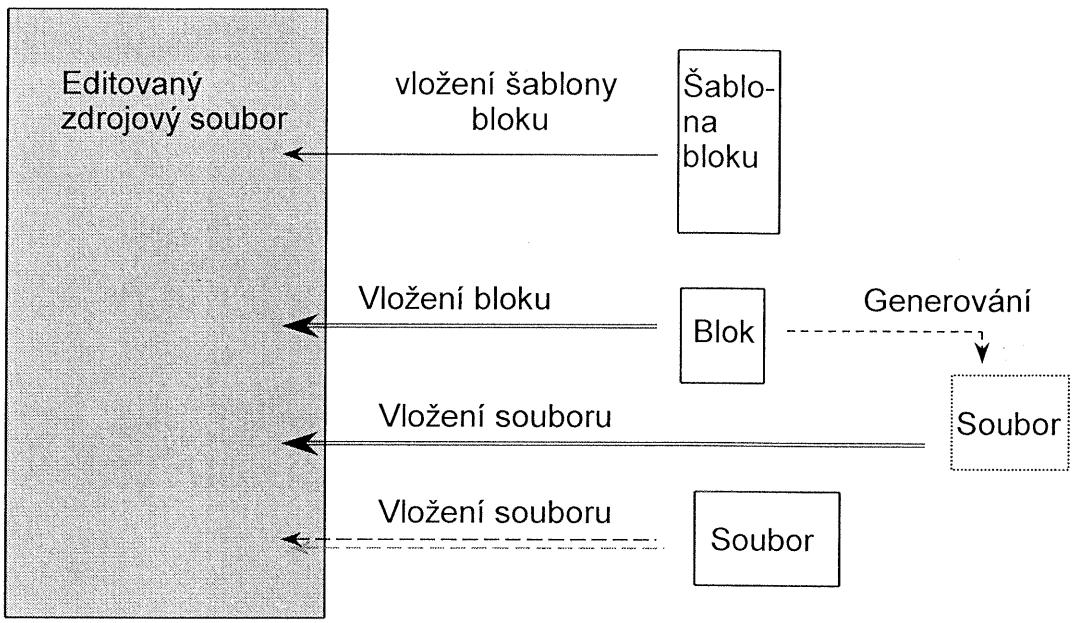
**Text Editor**

Místo programování v STL lze vytvořit uživatelský program také s použitím integrovaného Text Editoru a vytvořit tak zdrojový soubor. Při editaci lze do jednoho zdrojového souboru vložit zdrojový text i několika bloků. Kontrola syntaktických pravidel v tomto případě uživatele nezdržuje. Bude provedena až při překladu.

**Nastavení**

Před zahájením práce s Text Editorem je vhodné provést nastavení editoru z nabídky *Options Settings*. Tato nastavení usnadňují práci a umožňují přizpůsobení editoru osobním požadavkům uživatele.

## Vkládání šablon bloků, bloků a zdrojových souborů




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_09cz.6

Školicí středisko
firmy E&A spol. s r.o., MB

**Vložení šablony**

Šablony bloků OB, FB, FC, DB, Instančních DB, DB z UDT a UDT jsou přímo integrovány v Text Editoru. Tyto šablony obsahují všechna potřebná klíčová slova v požadovaném pořadí. Vložení lze jednoduše provést pomocí nabídky *Insert -> Block Template -> OB/FB/FC/DB/IDB/ DB from UDT/UDT.*

**Vložení bloků**

Do zdrojového textu lze samozřejmě vložit také zdrojovou podobu již existujícího bloku. Import se provede z nabídky *Insert -> Object -> Block.* V následujícím dialogu stačí definovat blok.

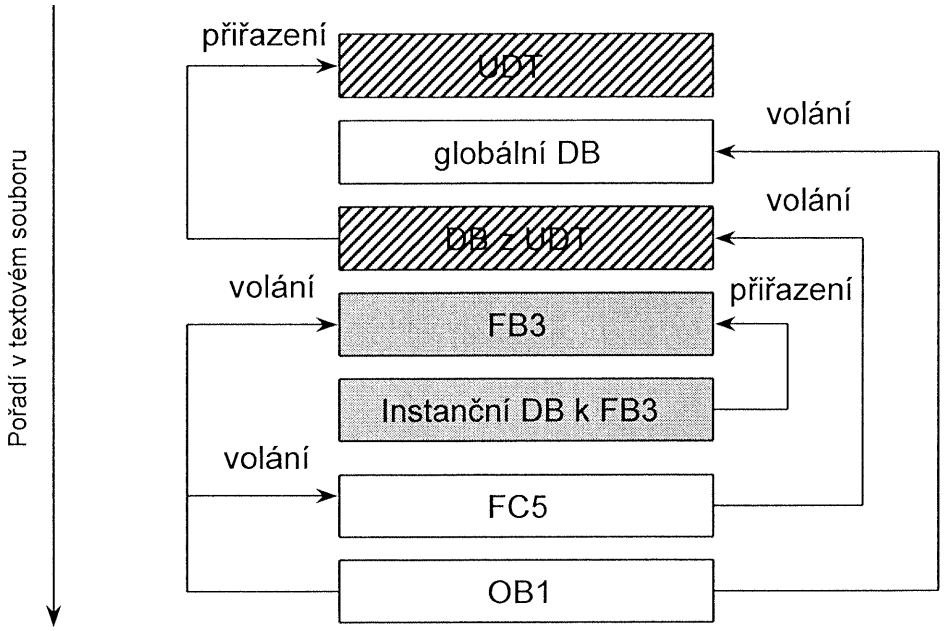
**Vložení zdrojového souboru**

Do editovaného textového souboru lze vložit také text z jiného textového souboru. Stačí aktivovat nabídku *Insert -> Object -> File* a definovat importovaný soubor.

**Poznámka**

Jakýkoliv text lze také vložit pomocí Windows schránky - *clipboardu.*

## Obecná editační pravidla a struktura







Datum: 24.11.2002
Soubor: PRO2_09cz.7



**Pravidla**

Pro uživatelský program ve formě zdrojového textu platí následující pravidla:

- Syntaxe instrukcí STL je stejná jako v blokovém editoru, výjimku tvoří pouze deklarace datových typů *Array* a *Structure*.
- Při zadávání názvů a textů je nutné rozlišovat velká a malá písmena.
- Každá instrukce nebo deklarace proměnné musí být ukončena ";", na jednom řádku může být i víc instrukcí než jedna.
- Každý komentář musí být oddělen znaky "//".

**Pořadí v souboru**

Všechna pravidla definující pořadí textů v souboru lze shrnout do jediného:

**Lze použít pouze to, co je definováno a deklarováno.**

Znamená to, že deklarace a definice bloků, proměnných a UDT musí předcházet jejich použití.

## Syntaxe logických bloků

| Konfigurace | Příklad klíčového slova |
|---|---|
| Začátek bloku | ORGANIZATION_BLOCK *OB 1*<br>FUNCTION_ BLOCK *FB 6*<br>FUNCTION FC 1 : *int* |
| Titulek bloku (volitelný) | TITLE = *Block title* |
| Komentář bloku (volitelný) | *//Block comment* |
| Atributy bloku (volitelné) | KNOW_HOW_PROTECT<br>AUTHOR: *Müller*<br>FAMILY: *Motors*<br>NAME: *Motorone*<br>VERSION: *0815* |
| Deklarace proměnných<br>(deklarované typy, závisí na typu bloku)<br><br><br><br>Ukončení deklarace pro všechny typy | VAR_IN<br>VAR_OUT<br>VAR_IN_OUT<br>VAR<br>VAR_TEMP<br>..<br>END_VAR |
| Programová část obsahuje:<br>Segmenty<br>Titulek segmentu<br>Komentář segmentu | BEGIN<br>NETWORK<br>TITLE=*first network*<br>//<br>..<br>NETWORK<br>.. |
| Konec bloku | END_ ORGANIZATION _BLOCK<br>END_FUNCTION_BLOCK<br>END_FUNCTION |





**Pravidla**

Při editaci logického bloku musí být dodržena následující pravidla:

- mezi definici typu bloku (např.ORGANIZATION_BLOCK) a identifikátorem bloku (např.*OB 1)* musí být mezera. Pokud je použito symbolické označení bloku, je symbolický název bloku v uvozovkách a musí být jedinečný.
- při definici funkce FC musí být deklarován také typ vracené hodnoty - datový typ proměnné RET_VAL. Povolen je pouze základní nebo komplexní datový typ. V případě, že funkce nevrací žádnou hodnotu, musí být uvedeno klíčové slovo "VOID".
- definice čísla segmentu není dovolena.

**Volání bloků "CALL"**

Syntaxe volání FC a FB instrukcí CALL se liší od volání v blokovém editoru. Definice aktuálních operandů je uzavřena závorkami, v závorce jsou jednotlivé parametry odděleny čárkou:

CALL FC1 (param1 := I 0.0, param2 := I0.1);

**Komentáře**

Pro zachování stejného zobrazení komentářů i v blokovém editoru je nutné dodržet následující pravidla:

- volání bloků: ve zdrojovém textu zachovat stejné pořadí přiřazení aktuálních operandů parametrům v jakém jsou deklarovány v rozhraní bloku.
- při překladu části, ve které je realizován přístup k datovému bloku pomocí instrukce "OPN", může dojít ke ztrátě komentáře instrukce "OPN" a následného komentáře. Je vhodnější použít kompaktní podobu přístupu nebo použít instrukci "NOP" za instrukcí "OPN"

Příklad : L DB5.DBW20; //Comm.

```
OPN DB5; //Comment1
NOP 0;
L DBW20; //Comment2).
```

# Syntaxe datových bloků

| Konfigurace | Příklad klíčového slova |
|---|---|
| Začátek bloku | DATA_BLOCK *DB 26* |
| Titulek bloku (volitelný) | TITLE = *Block title* |
| Komentář bloku (volitelný) | *//Block comment* |
| Atributy bloku (volitelné) | KNOW_HOW_PROTECT<br>AUTHOR: *Müller*<br>FAMILY: *Motors*<br>NAME: *Motorone*<br>VERSION: *0815* |
| Deklarační část - závislá na typu DB: | |
| globální DB:<br>deklarace proměnných<br>(volitelně s počáteční hodnotou) | STRUCT<br>..<br>..<br>END_STRUCT |
| DB pomocí UDT:<br>definice UDT | UDT *16* |
| Instanční DB<br>definice FB | FB *20* |
| Instrukční část s aktuálními hodnotami | BEGIN<br>..<br>.. |
| Konec bloku | END_ DATA_BLOCK |

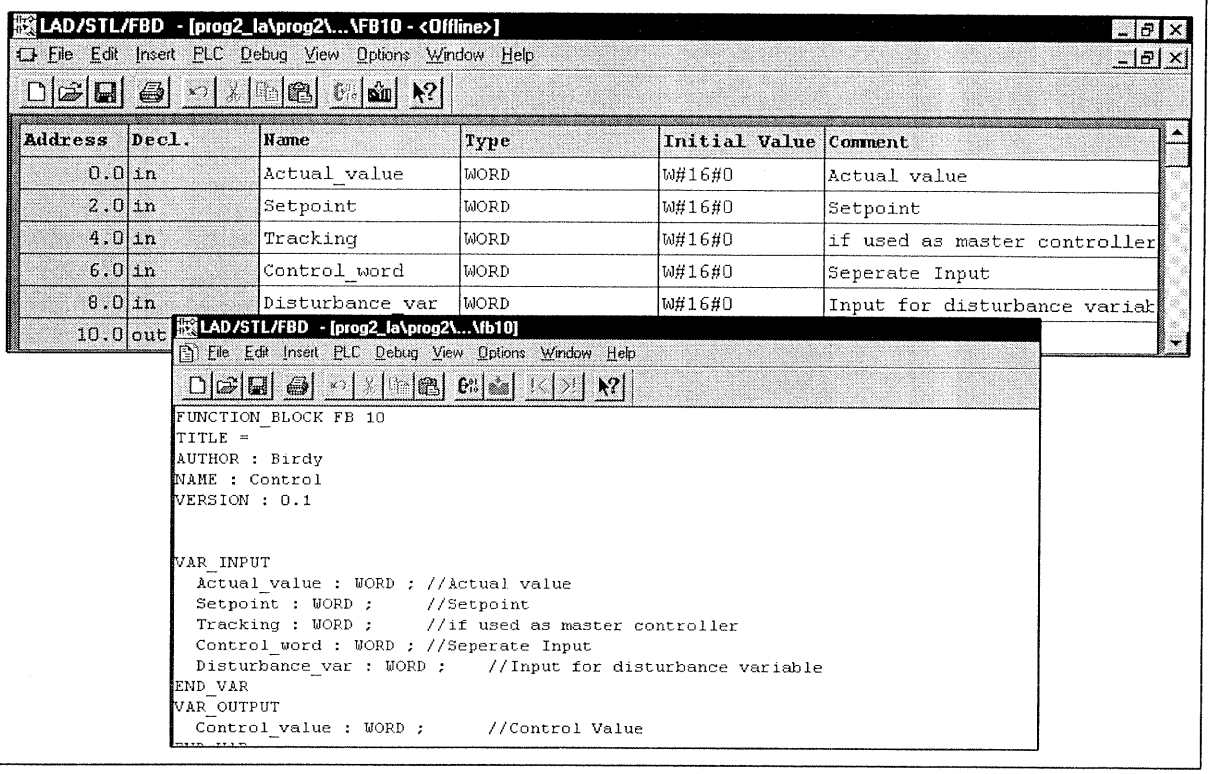




**Pravidla**

- nelze vytvořit DB 0
- volitelně lze definovat hodnoty všech nebo některých proměnných. U proměnných s nedefinovanou počáteční hodnotou bude použita implicitní hodnota příslušného datového typu.
- po překladu do bloku bude zobrazen pouze komentář uvedený v deklarační části.

# Pravidla deklarace proměnných









**Typy proměnných**

Typ proměnné je definován klíčovým slovem, které zahajuje deklarační část proměnných. Podle typu bloku lze deklarovat omezený počet typů proměnných. (viz. tabulka)

| Deklarovaný typ | Klíčové slovo | OB | FB | FC |
|---|---|---|---|---|
| vstupní parametr | VAR_INPUT | - | x | x |
| výstupní parametr | VAR_OUTPUT | - | x | x |
| vstupně/výstupní parametr | VAR_IN_OUT | - | x | x |
| statická proměnná | VAR | - | x | - |
| lokální proměnná | VAR_TEMP | x | x | x |
| konec deklarace | END_VAR | | | |

**Pravidla**

- Proměnné musí být deklarovány v pořadí deklarovaných typů. Všechny proměnné jednoho typu musí být deklarovány najednou.
- Jméno proměnné musí začínat písmenem a nesmí odpovídat některému z rezervovaných klíčových slov.
- Datový typ je od názvu proměnné oddělen dvojtečkou. Povoleny jsou základní, komplexní a uživatelské datové typy.
- Pole je definováno klíčovým slovem ARRAY. V hranatých závorkách je uveden rozměr pole a datový typ.
- Struktura je definována klíčovými slovy STRUCT a END_STRUCT.
- Deklarace každé proměnné je zakončena ";"
- Komentář je od deklarační části oddělen "//".

## Umístění atributů bloku

| Atribut | Logické bloky (OB,FC,FB) | Datové bloky | UDT |
|---|---|---|---|
| KNOW_HOW_PROTECT | YES | YES | NO |
| AUTHOR | YES | YES | NO |
| FAMILY | YES | YES | NO |
| NAME | YES | YES | NO |
| VERSION | YES | YES | NO |
| UNLINKED | NO | YES | NO |
| READ_ONLY | NO | YES | NO |





**Systémové atributy** Lze definovat systémové atributy bloků např. pro procesní diagnostiku nebo pro konfiguraci řídícího systému. Atributy jsou umístěny v kulatých závorkách za komentářem bloku.

**Vlastnosti bloku** Lze definovat jméno bloku, skupinu, verzi a autora. Je nutné dodržet pravidla:

- Vlastnosti jsou definovány před deklarací proměnných
- **Nejsou zakončeny** ";" na konci řádku.

**Ochrana bloku** Bloky lze "uzamknout" pomocí klíčového slova "KNOW_HOW_PROTECT" :

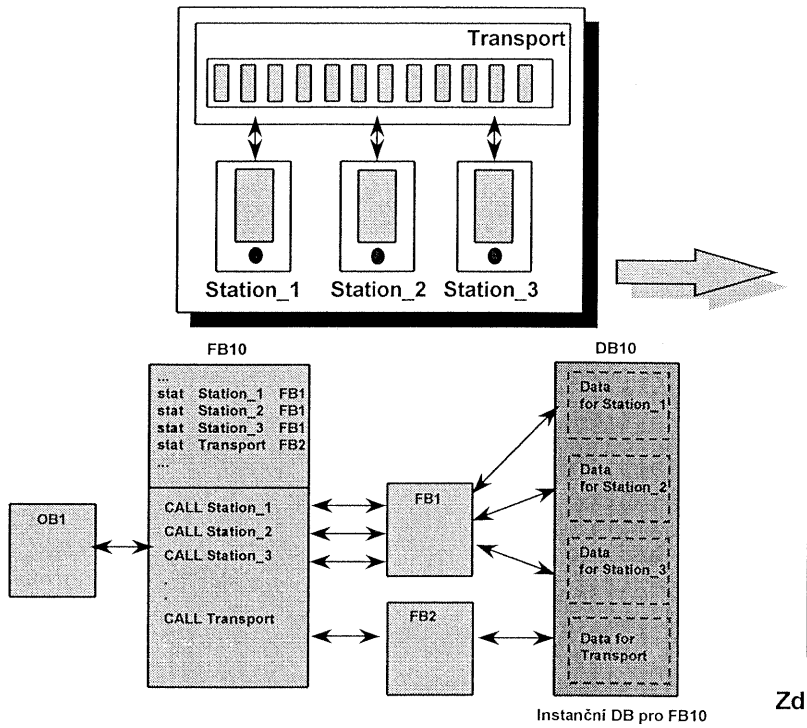
- po překladu nelze v STL/FBD/LAD Editoru zobrazit instrukční část bloku
- jsou zobrazeny pouze *in*, *out* a *in/out* parametry z deklarační části. Lokální a instanční proměnné nejsou zobrazeny.
- zpětným překladem se získá opět pouze deklarační část bloku.

"KNOW_HOW_PROTECT" musí být uvedeno před ostatními atributy bloku.

**Ochrana proti zápisu** U datového bloku lze zabránit přepsání hodnot proměnných definovaných ve zdrojovém tvaru bloku uvedením klauzule "READ_ONLY" . Musí být uvedena před deklarací proměnných.

**"UNLINKED"** Takto ošetřený datový blok nebude po nahrání do editační paměti CPU překopírován do pracovní paměti.

# Cvičení 9.1: Tvorba zdrojového souboru




```
FUNCTION_BLOCK "Station"
TITLE =
VERSION : 0.1

VAR_INPUT
  Initial : BOOL ;
  Proxy_Switch : BOOL ;
  Acknowledge : BOOL ;
  Clock_Bit : BOOL ;
END_VAR
VAR_OUTPUT
  LED : BOOL ;
  Transp_req : BOOL ;
END_VAR
VAR_IN_OUT
  Conv_busy : BOOL ;
END_VAR
VAR
  State : STRUCT
   Busy : BOOL ;
   Completed : BOOL ;
   Waiting : BOOL ;
  END_STRUCT ;
  Aux_1 : BOOL ;
  Aux_2 : BOOL ;
END_VAR
BEGIN
NETWORK
TITLE =Initialization
//"Busy" is set with the
// initialization
      U      #Initial;
...
```

**Zdrojový text: "montážní linka"**

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_09cz.12

Školicí středisko
firmy E&A spol. s r.o., MB

**Přehled**

Nejdříve bude vygenerován z programu cvičení 6.3 zdrojový soubor. Následně bude tento soubor doplněn u funkci počítání.

**Cíl cvičení**

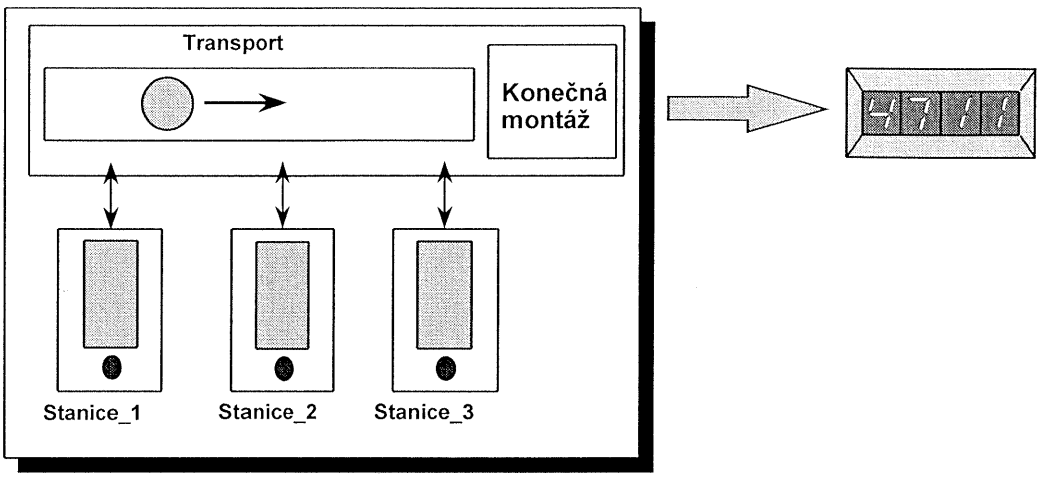
Do složky *Source Files* projektu PRO2-Project vygenerovat zdrojový text z již hotového programu, rozšířit funkčnost tohoto programu a následně bezchybně přeložit do blokové formy.

**Postup**

1. z požadovaného programu otevřít nějaký blok v STL/LAD/FBD Editoru.
2. aktivovat nabídku *File -> Generate Source*. Zobrazí se okno pro zadání jména nového zdrojového souboru.
3. zadat jméno souboru a potvrdit "OK". Zobrazí se okno "Select STEP7 Blocks".
4. Vybrat požadovaný blok (nebo bloky) a potvrdit "OK"

   Poznámka: Následně se zobrazí hlášení "Program structure (XREF) sorted" a generování zdrojového souboru je zahájeno.
5. V Text Editoru otevřít vytvořený zdrojový text a doplnit funkci programu.
6. Aktivací nabídky *File -> Check Consistency* zkontrolovat bezchybnost zadání.

# Cvičení 9.2: Počítání hotových dílů




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_09cz.13

Školicí středisko
firmy E&A spol. s r.o., MB

**Cíl cvičení**

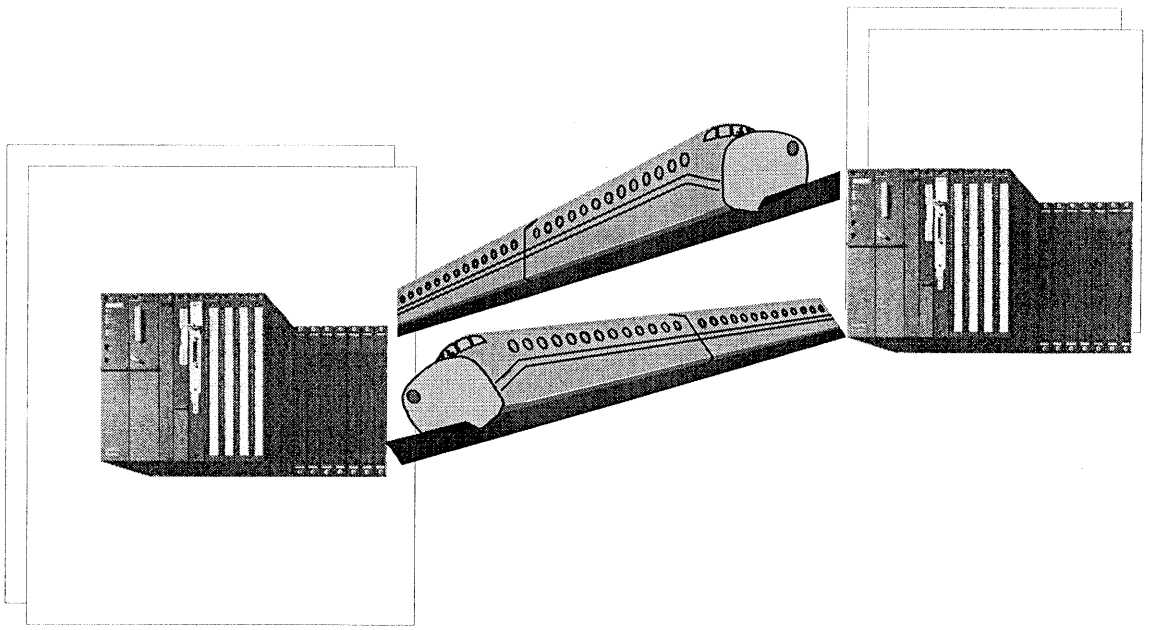
Do funkčního bloku "Transport" integrovat čitač počítající dokončené díly:

- čitač realizovat pomocí IEC 1131-3 čitače (SFB 0 "CTU").
- s každou spádovou hranou světelné závory inkrementovat čitač.
- čitač je resetován vstupním signálem-parametrem "Initial".
- aktuální hodnota čitače je předávána volajícímu bloku výstupním parametrem #Countvalue (INT).
- aktuální hodnotu zobrazovat na display simulátoru.
- požadované rozšíření provést ve formě zdrojového textu
- do všech FB a DB vložit klauzuli "KNOW_HOW_PROTECT" .

**Postup**

1. zkopírovat SFB 0 z knihovny StdLib30 do programu.
3. otevřít zdrojová text.
2. v FB "Transport" deklarovat statickou proměnnou "Counter" typu SFB 0 a výstupní parametr #Countervalue typu INT.
3. rozšířit FB "Transport" o nezbytné instrukce.
4. FB10 rozšířit o zobrazení hodnoty na display.
5. přeložit zdrojový text do blokové podoby.
6. nahrát bloky do CPU a otestovat funkci programu.
7. doplnit zdrojový text o ochranu bloků.

# Základní a rozšířená S7 komunikace



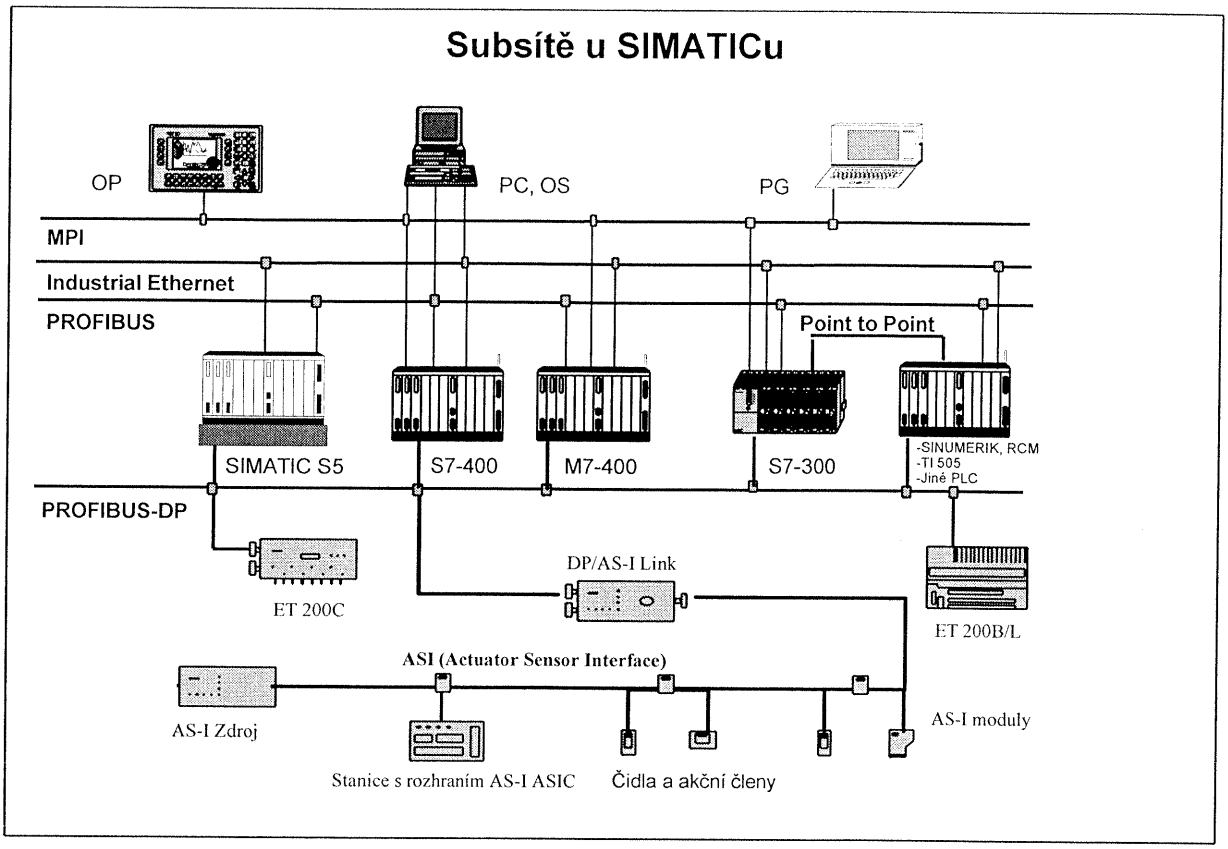
SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_10cz.1

Školicí středisko
firmy E&A spol. s r.o., MB

**Obsah** **Strana**




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_10cz.2



**Přehled**

SIEMENS nabízí, v závislosti na komunikační požadavky, následující subsítě:

**MPI**

MPI síť je navržena pro časově nekritickou komunikaci. MPI vícebodové komunikační rozhraní pro SIMATIC S7, M7 a C7.

Je navrženo také jako rozhraní u PG, tj. pro spojení PG a OP s řídícím systémem. Na druhou stranu umožňuje propojení pouze malého počtu CPU do sítě. Výstavba této sítě je však velmi jednoduchá.

**Industrial Ethernet**

*Industrial Ethernet* - průmyslový ethernet je otevřený, na výrobci dílů nezávislý, komunikační systém určený pro manažerskou a správní úroveň.

*Industrial Ethernet* je navržen pro časově nekritické vysílání většího objemu dat a komunikaci několika sítí pomocí bran - *gateway*.

**PROFIBUS**

*PROFIBUS* je také otevřený, na výrobci nezávislý, komunikační systém. Tento systém je navržen ve dvou verzích, které se liší svými vlastnostmi:

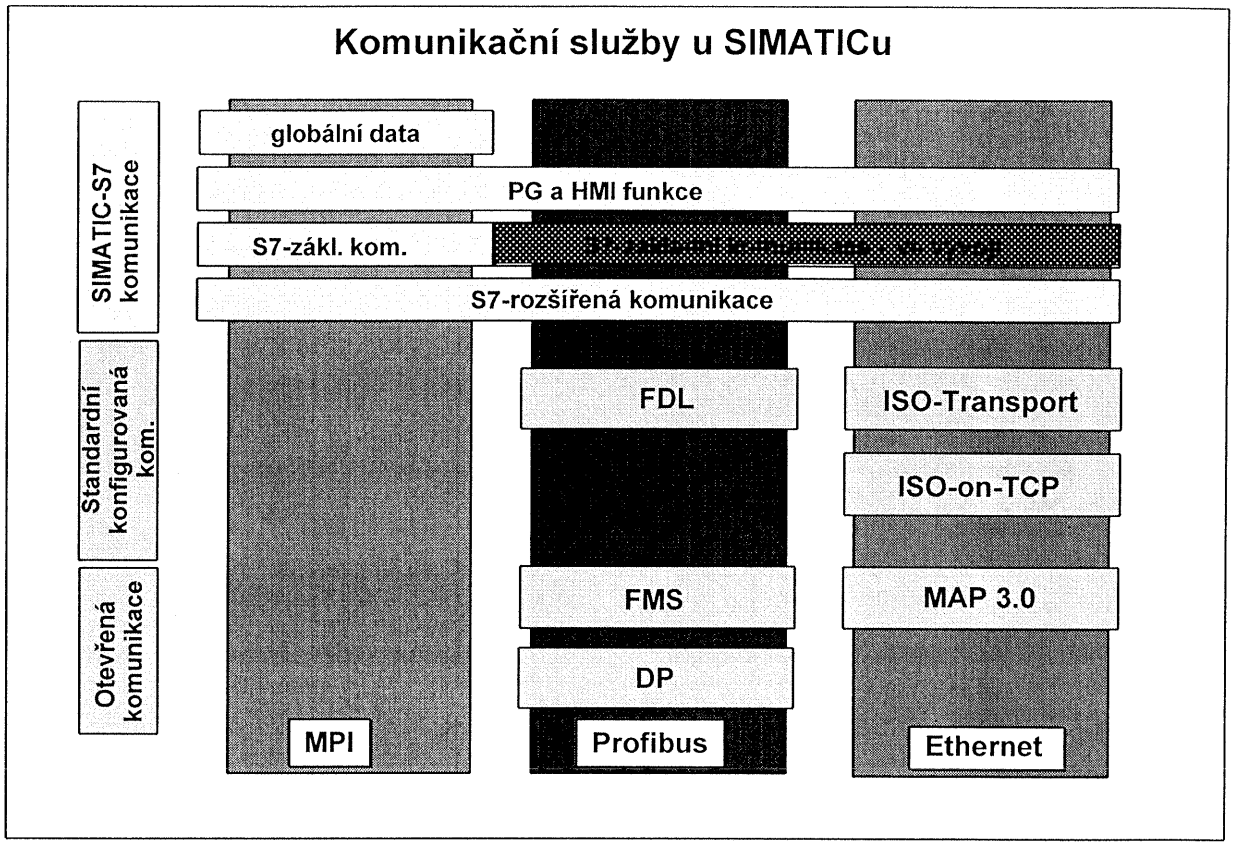
- PROFIBUS - pro časově nekritickou komunikaci mezi rovnocennými partnery.
- PROFIBUS DP - pro časově kritickou, cyklickou výměnu dat mezi inteligentní řídící stanicí - *master* a podřízenou stanicí - *slave*.

**PtP-Connection**

*Point-to-Point* spojení se používá hlavně pro časově nekritickou komunikaci mezi dvěmi stanicemi nebo pro spojení stanice s datovým terminálem (OP, tiskárna, čtečka čárového kódu, magnetických karet atd.)

**AS-Interface**

AS-I rozhraní je komunikační síť určená pro nejnižší úroveň komunikace mezi PLC a jednotlivými čidly a akčními členy.



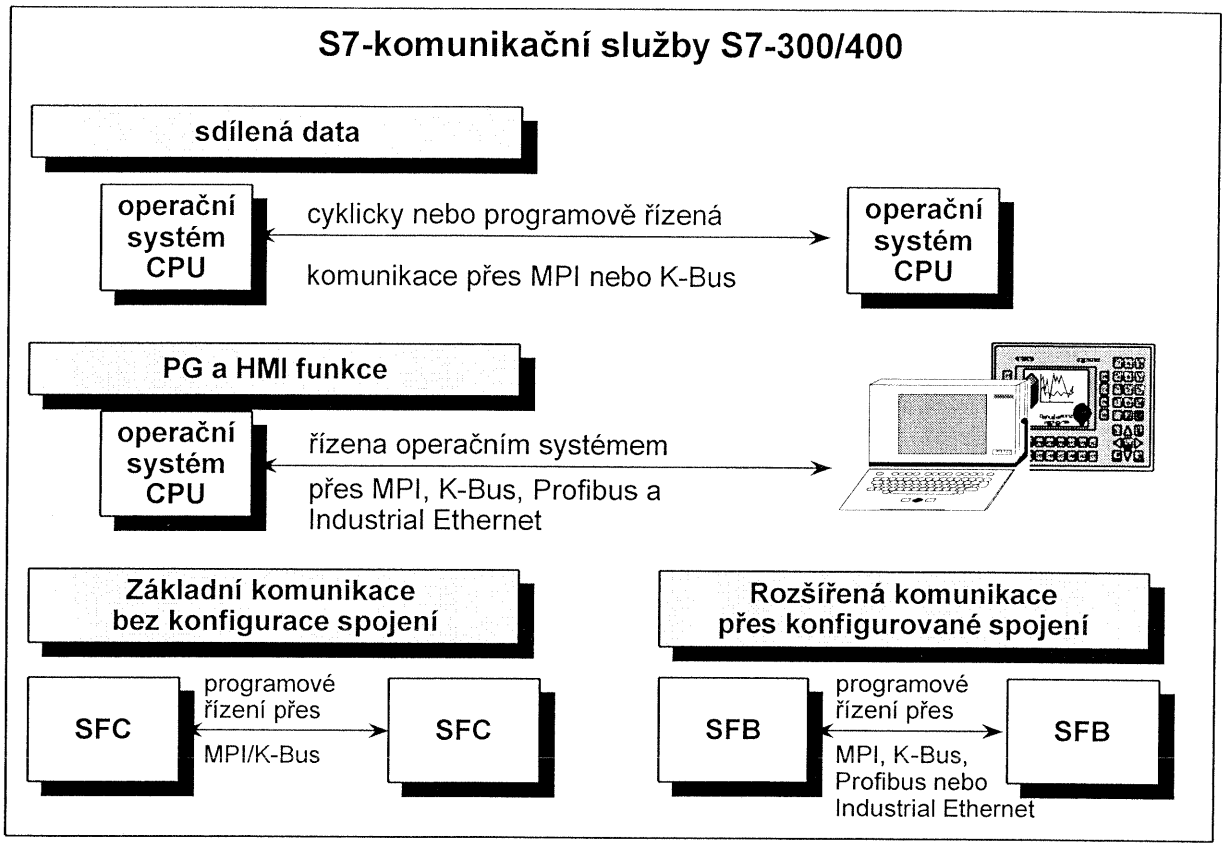

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_10cz.3



| | |
|---|---|
| **Služby** | Komunikační služby popisují komunikační funkce - jejich výkonové charakteristiky, jako je např. výměna dat, řídící služby, kontrolní služby atd. |
| **Globální data** | GD (***G***lobal ***D***ata) jsou určena pro cyklickou výměnu malého objemu dat (u S7-400 navíc řiditelnou událostmi). |
| **S7 komunikace** | Tyto komunikační funkce jsou optimalizovány pro komunikaci S7, PG a TD/OP panelů v jednoduchém komunikačním systému SIMATIC S7.<br>• PG funkce; umožňují připojení PG bez nutnosti složité konfigurace.<br>• HMI funkce; umožňují jednoduché připojení TD/OP.<br>• základní nekonfigurovaná komunikace je implementována pomocí SFC bloků obsažených v operačním systému CPU.<br>• rozšířená komunikace pomocí SFB (S7-400 Client/Server; S7-300 Server). Pro tuto komunikaci je nejprve nutné konfigurovat příslušné spojení. |
| **FDL (SDA)** | Tato vrstva je určena pro bezpečný přenos průměrného množství dat mezi SIMATIC S7 a S5. Odpovídá 2.vrstvě *Fieldbus Data Link* (FDL) pro *Profibus*. |
| **ISO Transport** | Tato komunikace slouží k přenosu (až 240bytů) dat mezi SIMATIC S5 a S7. |
| **ISO-on-TCP** | Slouží k bezpečnému přenosu dat mezi SIMATIC S7 a PC nebo jinými systémy pomocí sítě TCP/IP.<br>FDL, ISO a ISO-on-TCP funkce jsou přístupné přes volání funkcí AG-SEND/AG-RECEIVE. |
| **FMS** | ***F***ieldbus ***M***essage ***S***pecification (FMS) je objektově orientovaná komunikace. Funkce podporované FMS (proměnné, doménové služby, atd.) jsou definovány v normě *EN 50170 Vol. 2.* |
| **MAP** | Komunikace původně vyvinutá americkou společností General Motors pro objektově orientovanou komunikaci mezi PLC systémy (MAP= Manufacturer Automation Protocol). |
| **DP** | DP (Distributed I/O) je zvlášť optimalizovaná komunikace. Je určena pro časově kritickou výměnu mezi inteligentní řídící jednotkou (DP Master) a průmyslovou stanicí (DP Slave). Je definována v normě *EN 50170 Vol. 3.* |




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_10cz.4



**Sdílená data** Tato komunikace umožňuje cyklickou výměnu dat mezi CPU přes MPI rozhraní bez nutnosti programování. Datová výměna se provádí v cyklu, společně s aktualizací obrazu procesu.

**PG a HMI funkce** Systémové služby PG a HMI funkcí tvoří základ pro analýzu rozšířené S7-komunikace. Předpokladem pro spojení PG nebo HMI stanice s S7-300/400 je dostupné volné spojení na systém (S7-CPU, M7-CPU, M7-FM, atd.).

**Základní komunikace** Tyto komunikační služby umožňují přenos dat mezi CPU S7-300/400 pomocí sítě MPI nebo vnitřně pomocí K sběrnice. Na straně vysilače je volána funkce (SFC) X_SEND , na straně příjemce je potom volána funkce X_RCV. Objem dat na jedno volání je max. 76 bytů.
Spojení je při volání bloků aktivně vytvořeno a po dokončení i zrušeno.

**Rozšířená komunikace** Tyto komunikační služby jsou určeny pro všechny CPU S7-400. Umožňují přenos až 64KB dat pomocí jedné z subsítí (MPI, Profibus, Industrial Ethernet). Jako programové rozhraní slouží systémové funkční bloky (SFB) Tyto bloky jsou součástí pouze operačního systému S7-400 a nejsou dostupné u CPU S7-300. Tyto služby kromě výměny dat poskytují také řídící funkce (např "stop" , "start" partnera).
Tuto komunikaci je třeba konfigurovat (tabulka spojení) a definovaná spojení jsou trvale vytvořena při zapnutí systémů.

# Komunikační možnosti S7-CPU

| S7 Modul | Počet spojení | System Functions | Popis |
|---|---|---|---|
| CPU 313, 314, 315 | max. 4 pro SFB<br>max. 8 pro SFC<br>(4 pro CPU 313) | SFC bloky<br>GET/PUT/START/STOP<br>jsou integrovány v<br>operačním systému | všechny SFC=76 Bytů uživatelských dat<br>GET = 222 Bytů uživatelských dat<br>PUT = 212 Bytů uživatelských dat<br>STOP/START = Serverové nástroje |
| CPU412 | max. 8 pro SFB<br>max. 6 pro SFC | SFC bloky<br>SFB bloky | all SFC's=76 Bytů uživatelských dat<br>GET = 462 Bytů uživatelských dat<br>PUT = 452 Bytů uživatelských dat<br>USEND/URCV= 454 Bytů uživatelských dat<br>BSEND/BRCV=64Kbytů uživatelských dat |
| CPU413 | max. 16 pro SFB<br>max. 14 pro SFC | --- // --- | --- // --- |
| CPU414 | max. 32 pro SFB<br>max. 30 pro SFC | --- // --- | --- // --- |
| CPU416 | max. 64 pro SFB<br>max. 62 pro SFC | --- // --- | --- // --- |
| PROFIBUS CP | max. 32 (S7-400) | Komunikační kanál | |
| ETHERNET CP | max. 48 (S7-400) | Komunikační kanál | |






**Přehled**

Stanice, které se účastní spojení musí mít volné prostředky pro toto spojení. Velikost zdrojů - počet navázaných spojení, závisí na CPU/CP.

Jsou-li použity všechny zdroje stanice, nelze již vytvořit nové spojení.

**PG/OP**

Každé PG nebo OP/TD spojení vyžaduje komunikační zdroje CPU SIMATICu S7/M7. Standardně jsou všechny komunikační zdroje S7/M7-CPU určené pro spojení s PG a OP/TD.

**CPU komunikace**

Je-li komunikace S7/M7-300/400-CPU vedena pomocí SFC a SFB bloků přes integrované MPI-/PROFIBUS-DP rozhraní, jsou komunikační zdroje použity na všechna S7 spojení.

**Komunikace přes CP**

Při S7 komunikaci přes externí CP rozhraní jsou komunikační zdroje použity na propojení CPU a CP.
Jestliže je konfigurováno jednoduché propojení přes PROFIBUS nebo Industrial Ethernet CP nejsou nutné žádné další konfigurační údaje pro parametrizaci CP.

**FM komunikace**

Jestliže je vedena komunikace FM přes integrované MPI-/PROFIBUS-DP rozhraní na S7/M7-CPU, potom na FM je použit jeden zdroj a na CPU dva zdroje. Totéž platí také ve víceprocesorovém režimu, kde jsou CPU propojeny nepřímo přes K-Bus s MPI subsítí.

## SFC komunikace: přehled

- Datová výměna přes MPI subsíť nebo uvnitř stanice
- Ve srovnání s SFB komunikací není nutná žádná konfigurace spojení
- Spojení s partnerem je dynamicky navazováno a ukončováno.
- Přenos max. 76 Bytů
- Použitelné u všech procesorů S7-300/400
- Data lze také zapisovat nebo číst přes PROFIBUS-DP z S7-200 pomocí funkcí X_GET, X_PUT
- Partner komunikace může existovat i v jiném S7 projektu





**Přehled**

Pomocí SFC nekonfigurované komunikace lze přenášet malý objem dat mezi S7/M7-300/400-CPU a dalšími komunikujícími moduly.

Partner musí být připojen na stejné MPI subsíti nebo být dosažitelný uvnitř stanice přes K-Bus nebo PROFIBUS-DP.

Konfigurace spojení není nutná.

**Spojení**

Při zavolání SFC je dynamicky vytvořeno spojení s partnerem o udané adrese a po ukončení přenosu je toto spojení v závislosti na parametru CONT uzavřeno. Při konfigurovaném spojení jsou všechny dostupné komunikační zdroje požadovány u všech partnerů.

Nejsou-li při zavolání SFC dostupné žádné komunikační zdroje, vrací RET_VAL odpovídající chybové hlášení.

Existující spojení vytvořené pomocí SFB nelze použít. Jestliže aktivní CPU přejde při přenosu dat do STOP stavu, je spojení zrušeno.

V RUN stavu CPU nelze komunikační SFC smazat, protože to ohrožuje dostupnost použitých komunikačních zdrojů. Programové změny lze provádět pouze ve STOP stavu.

**Objem dat**

Pro všechny CPU S7/M7/C7 je velikost přenášeného balíku dat omezena na max. 76 bytů.

# SFC komunikace : přehled bloků

| SFC | Název | Popis |
|---|---|---|
| SFC 65 | X_SEND | Vysílání dat příjemci X_RCV |
| SFC 66 | X_RCV | Příjem dat od odesilatele X_SEND |
| SFC 67 | X_GET | Čtení dat z partnerského PLC |
| SFC 68 | X_PUT | Zápis dat na partnerské PLC |
| SFC 69 | X_ABORT | Zrušení stávající komunikace |
| SFC 72 | I_GET | Interní čtení dat (z partnera ve stejné stanici např.FM) |
| SFC 73 | I_PUT | Interní zápis dat (do partnera ve stejné stanici) |
| SFC 74 | I-ABORT | Zrušení interního komunikačního spojení. |





**Přehled**

Komunikační SFC bloky nabízí potvrzovaný přenos dat po nekonfigurovaném spojení.

SFC bloky (X_...) umožňují adresovat ( a komunikovat) partnery ve stejné MPI subsíti, SFC bloky (I_...) umožňují adresovat partnery ve stejné stanici (např. FM moduly)

Komunikace přes MPI subsíť umožňuje také komunikaci s partnerem, který existuje v jiném S7-projektu.

Počet po sobě jdoucích dosažitelných komunikačních uzlů není omezen.

**Adresace**

Při externí komunikaci (X_...) přes MPI subsíť je adresa partnera dána MPI adresou, při interní komunikaci (I_...) uvnitř stejné stanice je adresa dána počáteční logickou adresou modulu.

Pokud je modul vybaven samostatnou základní adresou pro vstupy (I-adresa) stejně tak jako samostatnou adresou pro výstupy (Q adresa), udává se při volání SFC bloku nižší číslo.

**Datová konzistence**

Maximální velikost datové oblasti, kterou lze operačním systémem přečíst (X_PUT, I_PUT) nebo zapsat (X_GET, I_GET) jako souvislý blok je označována jako datová konzistence:

- S7-300-CPU: 8 Bytů
- S7-400-CPU: 32 Bytů

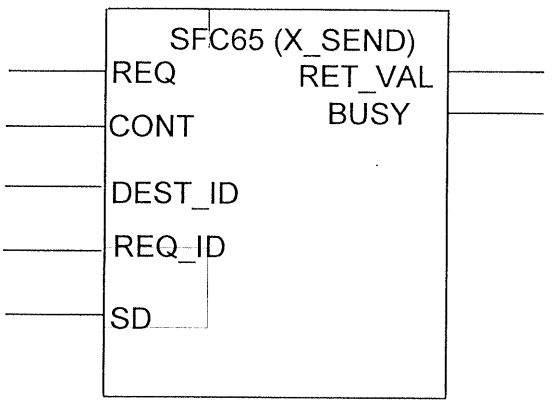
# SFC komunikace : blok X_SEND (SFC 65)

## STL zobrazení

s příkladem parametrizace

```
CALL SFC 65
REQ:=      M4.0//Trigger
CONT:=     FALSE//zruš. spoj.
DEST_ID:=  W#16#4/MPI_adresa.
REQ_ID:=   DW#16#1//Identifikátor
SD:=       P#M20.0 BYTE 10
RET_VAL:=  MW40//chybový kód
BUSY:=     M 4.1//přenos aktivní
```

## LAD/FBD zobrazení







Datum: 24.11.2002
Soubor: PRO2_10cz.8



**Popis**

SFC 65 "X_SEND" umožňuje odeslání dat externí stanici, která je mimo aktuální stanici. Příjem dat "na druhé straně" lze provést pomocí bloku SFC 66 "X_RCV".

Vstupní parametr REQ_ID umožňuje identifikaci odeslaných dat, je odeslán spolu s daty a lze ho po příjmu vyhodnotit a určit tak původce přijatých dat.

Vlastní odeslání se aktivuje parametrem REQ=1.

Při komunikaci musí být zajištěno, že vysílaná datová oblast (definované parametrem SD) je stejně velká nebo menší než přijímací oblast (definovaná parametrem RD).

Parametry SFC 65 X_SEND

| Parametr | Deklarace | Typ | Význam |
|---|---|---|---|
| REQ | INPUT | BOOL (I,Q,M,D,L const.) | aktivace přenosu dat (= 1) |
| CONT | INPUT | WORD (I,Q,M,D,L Const.) | CONT=0 zrušit spojení<br>CONT=1 spojení zachovat |
| DEST_ID | INPUT | WORD (I,Q,M,D,L const.) | MPI adresa příjemce |
| REQ_ID | INPUT | DWORD (I,Q,M,D,L, const.) | Identifikátor původu dat |
| SD | INPUT | ANY (I,Q,M,D) | Odesílaná datová oblast |
| RET_VAL | OUTPUT | INT (I,Q,M,D,L) | výsledek přenosu - chybový kód |
| BUSY | OUTPUT | BOOL (I,Q,M,D) | BUSY=1 Přenos aktivní - „jede“<br>BUSY=0 Přenos ukončen |

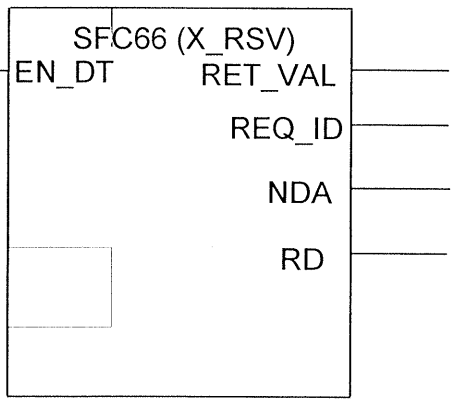
# SFC komunikace : blok X_RCV (SFC 66)

STL zobrazení

s příkladem parametrizace

```
CALL SFC 66
EN_DT:=     TRUE//aktivace přenosu
RET_VAL:=   MW 50//chybový kód
REQ_ID:=    MD52//datový ID
NDA   :=    M40.0//Data přijata
RD:=        P#M20.0 BYTE 10
```

LAD/FBD zobrazení







Datum: 24.11.2002
Soubor: PRO2_10cz.9



**Popis**

SFC 66 "X_RCV" zajišťuje příjem dat odeslaných pomocí SFC 65 "X_SEND". Komunikační partner je mimo aktuální stanici.

SFC 66 "X_RCV" umožňuje:

- určit, zda v daném okamžiku jsou dostupná vysílaná data a umístit je do interního zásobníku operačního systému.
- kopírovat blok dat z interního zásobníku do definované přijímací oblasti

Výběr funkce se provádí pomocí vstupního parametru EN_DT (***EN***able ***D***ata ***T***ransfer).

Parametry X_RCV SFC 66

| Parametr | Deklarace | Type | Význam |
|---|---|---|---|
| EN_DT | INPUT | BOOL (I,Q,M,D,L, constant) | EN_DT=0 kontrola bloku dat<br>EN_DT=1 přenos dat do paměti |
| RET_VAL | OUTPUT | INT (I,Q,M,D,L) | Výsledek přenosu - chybový kód |
| REQ_ID | OUTPUT | DWORD (I,Q,M,D,L,) | Identifikátor odesilatele X_SEND SFC 66, jehož data jsou první ve frontě |
| NDA | OUTPUT | BOOL (I,Q,M,D,L) | NDA=0 žádná data<br>NDA=1 nejméně 1 blok dat přijat (EN_DT=0) nebo blok dat byl přenesen do paměti (EN_DT=1) |
| RD | OUTPUT | ANY (I,Q,M,D) | Příjmová oblast dat |

## SFB komunikace : přehled

- Datová výměna pomocí MPI, K-Bus, Profibus nebo Industrial Ethernet
- Konfigurovaná spojení přes tabulku spojení
- Spojení jsou konfigurována při kompletním náběhu a existují trvale - včetně STOP stavu CPU
- Přenos velkého objemu dat
- Řídící komunikační služby pro obsluhu partnera (Stop, Start)
- SFB existují ve všech S7-400
- Data lze číst a zapisovat i přes S7-300 (GET/PUT)
- Jedno spojení umožňuje zpracování několika různých úloh





**Přehled**

SFB bloky jsou dostupné na všech CPU řady S7-400 a umožňují datovou výměnu mezi S7/M7-300/400. Lze přenášet až 64 Kbytů pomocí různých subsítí.

**Spojení**

Komunikační SFB bloky nabízí možnost ochrany dat přenášených pomocí konfigurovaných spojení. Konfigurace těchto spojení se provádí pomocí programu NETPRO, integrovaného do SIMATIC Manageru.

Konfigurovaná spojení jsou navázána při **kompletním** náběhu stanic a existují trvale i když stanice přejde do STOP stavu. Při pouhém náběhu nejsou tato spojení navazována.

Účastníci komunikace musí být připojeni na společnou MPI, PROFIBUS nebo Industrial Ethernet subsíť

**SFB**

Programové rozhraní v uživatelském programu tvoří zvláštní systémové bloky SFB, které splňují normu ISO/IEC 1131-5 a zajišťují jednotné uživatelské rozhraní.

**Objem dat**

Objem přenášených dat závisí na použitém SFB bloku a účastnících komunikace:

- PUT/GET 160 byte u S7-300 a 400 byte u S7-400/M7
- USEND/UREC až 440 byte
- BSEND/BRCV až 64KByte

# SFB komunikace : přehled bloků

| SFB/SFC | Název | Typ komun. | Popis |
|---|---|---|---|
| SFB 8 | USEND | 2-stranná | odeslání dat URCV klientovi |
| SFB 9 | URCV | 2-stranná | příjem dat od USEND klienta |
| SFB 12 | BSEND | 2-stranná | odeslání až 64KB dat příjemci BRCV |
| SFB 13 | BRCV | 2-stranná | příjem až 64KB dat od BSEND |
| SFB 14 | GET | 1-stranná | čtení dat z partnera (PLC) |
| SFB 15 | PUT | 1-stranná | zápis dat do partnera (PLC) |
| SFB 16 | PRINT | 1-stranná | odeslání dat na vzdálenou tiskárnu |
| SFB 19 | START | 1-stranná | provede kompletní náběh PLC |
| SFB 20 | STOP | 1-stranná | vyvolá STOP stav partnera |
| SFB 21 | RESUME | 1-stranná | vyvolá náběh partnera |
| SFB 22 | STATUS | 1-stranná | kontrola stavu partnera (RUN,STOP, náběh) |
| SFB 23 | USTATUS | 1-stranná | příjem stavových hlášení partnera |
| SFC 62 | CONTROL | --- | kontrola vnitřního stavu spojení a SFB |





**SFB: S7- 400**

Tyto komunikační SFB jsou integrovány do operačního systému CPU. Všechny uvedené bloky jsou dostupné ve všech S7-400.

Pro začlenění do uživatelského programu jsou v knihovně "STDLIB30" ve složce "Builtin" uvedeny záhlaví těchto bloků.

**SFB: S7 - 300**

PLC S7-300 neobsahuje SFB pro rozšířenou komunikaci. Na druhou stranu, ale podporuje řídící funkce 1-stranných komunikačních služeb, tzn. že např. CPU 3xx lze přečíst nebo do něho zapsat data z CPU 4xx pomocí bloků GET a PUT.

**Třídy funkcí**

Bloky jsou rozděleny do 4 funkčních tříd:

- vysílací a přijímací funkce
- řídící funkce
- sledovací funkce
- testovací funkce

**SFB pro datovou výměnu**

SFB bloky umožňují datovou výměnu mezi stanicemi (S7/M7-CPU, M7-FM):

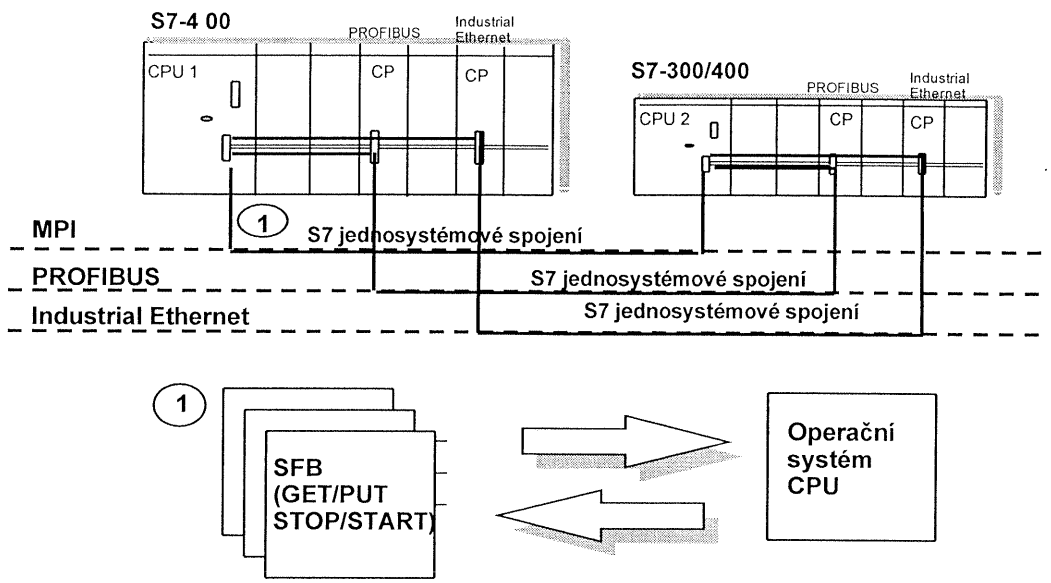
- GET, PUT (1-stranné čtení nebo zápis proměnných)
- USEND/URCV (2-stranné nekoordinované vysílání/příjem)
- BSEND/BRCV (2-stranné blokové vysílání/příjem)

**SFB pro správu programu**

SFB pro správu programu umožňují řízení a vyhodnocování režimu činnosti partnera nebo spojení - komunikačního kanálu.

- START/STOP/RESUME (řídící funkce)
- STATUS/USTATUS (monitorovací funkce)
- CONTROL (sledovací funkce)

## SFB komunikace přes jednostranné spojení




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_10cz.12



**S7 jednosystémové spojení**

Aby komunikační SFB bloky mohly komunikovat s ostatními, musí být nejprve konfigurováno jednosystémové spojení.

Maximální počet konfigurovaných spojení závisí na vlastním komunikačním uzlu.

S7 jednosystémová spojení mohou být konfigurována pro MPI, Industrial Ethernet a PROFIBUS síť.

**Jedno-stranné S7 jednosystémové spojení**

Protože S7-300 neobsahuje potřebné SFB bloky, lze pro spojení S7-400 a S7-300 konfigurovat pouze jednostranné spojení.

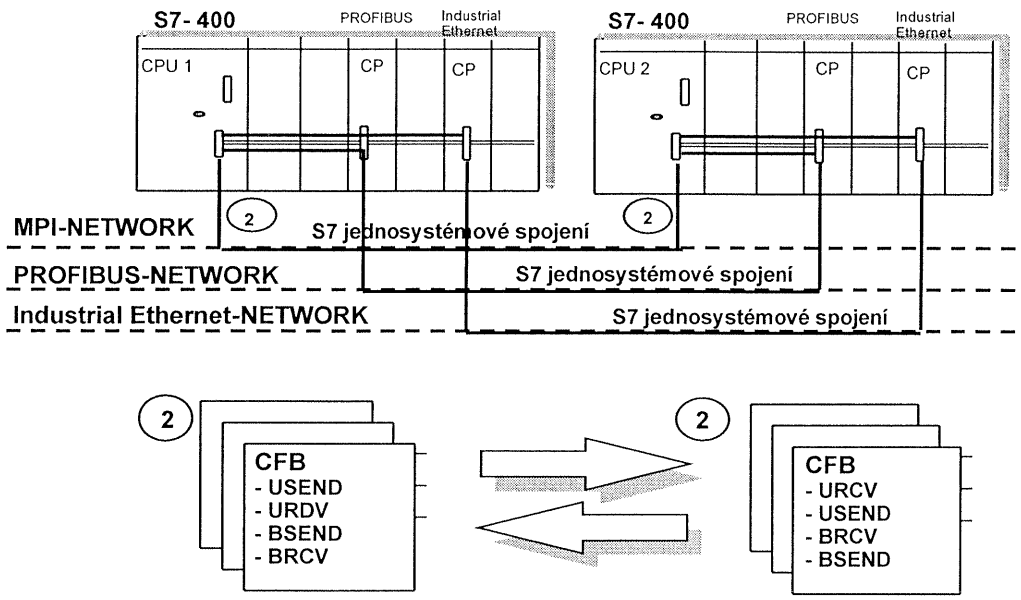
U jednostranné komunikace je potřeba vytvořit uživatelský program pouze na straně iniciátora komunikace - klienta. Na druhé straně (server) je komunikace obsluhována plně operačním systémem a není nutné provádět jakékoliv programovací práce. Pro jednostranné spojení se také zadává identifikátor pouze na straně klienta.

**"Jednostranné" SFB**

- GET, PUT
- STOP, START, RESUME
- STATUS, USTATUS

jsou považovány za jednostranné komunikační SFB bloky.

# SFB komunikace přes oboustranné spojení







Datum: 24.11.2002
Soubor: PRO2_10cz.13



**Oboustranné S7 jednosystémové spojení**

Jestliže jsou použity oboustranné komunikační služby, musí být konfigurováno oboustranné S7 jednosystémové spojení na obou stranách komunikace. Na rozdíl od jednostranné komunikace musí být tato konfigurace provedena jak na straně klienta tak na straně serveru.

Při konfiguraci je vytvořen identifikátor spojení, pomocí kterého se jak server tak klient odkazují na toto spojení při volání SFB bloků.

S7 jednosystémová spojení vytvořená pomocí CP pro Ind. Ethernet nebo PROFIBUS, nevyžadují další konfigurační data. Při oboustranné jednosystémové komunikaci musí uživatel určit která strana (které PLC) zahajuje komunikaci.

**"Oboustranná" SFB**

- BSEND=vysilač (Client) ==> BRCV přijímač (Server)
- USEND=vysilač (Client) ==> URCV přijímač (Server)

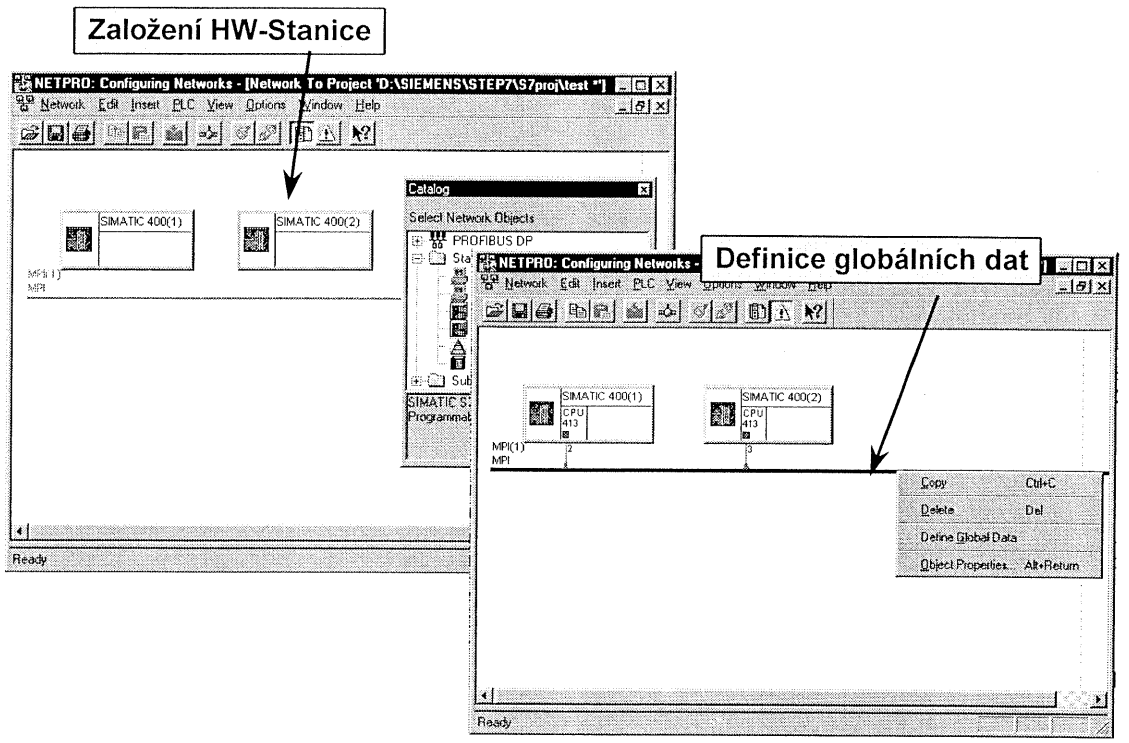
jsou předpokládána na obou stranách komunikace, tj. tyto bloky musí být instalovány v páru.

Přijímač (Server) může pomocí volání bloku URCV nebo BRCV určovat, kdy je připraven přijmout data od vysilače k dalšímu zpracování. Na druhou stranu může přijímač pomocí parametru "NDR" (***N***ew ***D***ata ***R***eceived) testovat, zda nebyla přijata nějaká nová data ke zpracování.

U jednostranné komunikace není program na straně serveru informován o příjmu nových dat.

Jednostranné komunikační služby lze také zpracovávat pomocí oboustranného spojení.

# Konfigurace sítě pomocí NETPRO






**Úvod** Program *NETPRO* umožňuje grafické konfigurování MPI, Profibus nebo Industrial Ethernet sítě. Výhodou je přehlednost a dokumentovatelnost konfigurace.

**Spuštění** Program se spouští dvojklikem na symbolu sítě, např. MPI v SIMATIC Manageru.

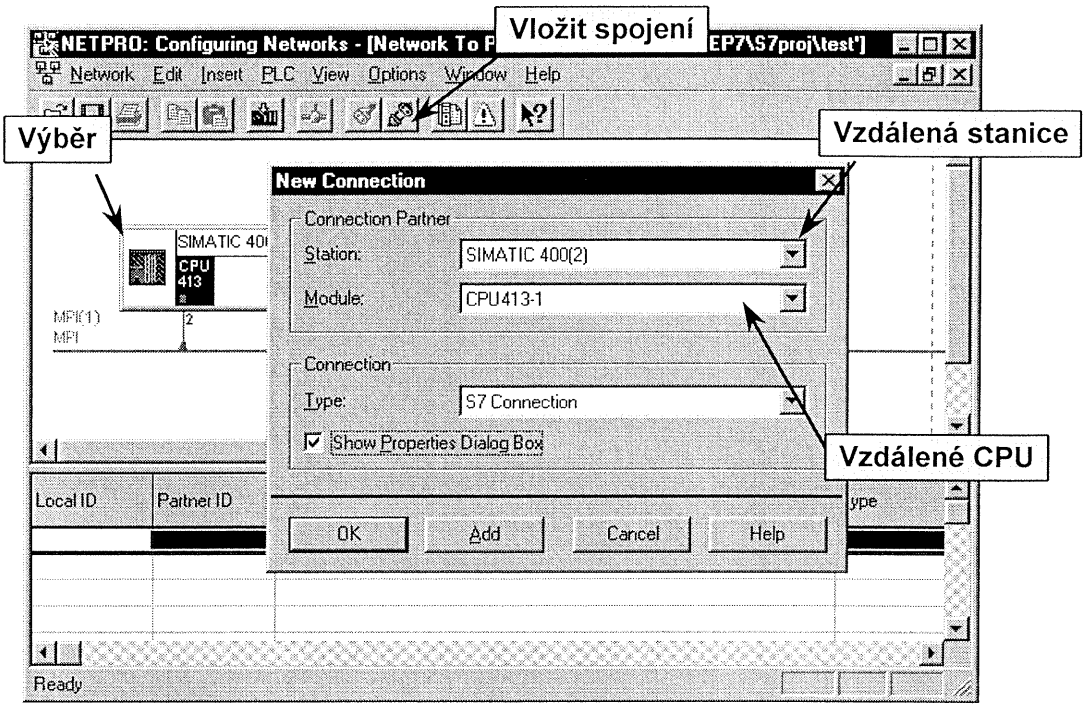
**Vložení HW-Stanice** V katalogu lze vybrat nezbytné díly (subsiť, stanici) a metodou *drag and drop* je umístit na požadované místo ve schématu.

**Konfigurace HW** Po umístění dílu na ploše lze dvojklikem na CPU spustit program HWConfig a definovat konfiguraci stanice ( výběr MPI adresy, připojené subsítě, atd.)

Před konfigurací jednotlivých spojení musí být nakonfigurovány všechny stanice připojené do dané subsítě.

**Globální data** Pomocí pravého tlačítka myši lze kliknutím na subsíti vyvolat *short-menu* a výběrem nabídky "Define Global Data" spustit program pro definici globálních dat.

# Konfigurace S7 jednosystémového spojení







Datum: 24.11.2002
Soubor: PRO2_10cz.15



**Přehled**

Předpokladem pro programově řízenou výměnu dat pomocí SFB je vytvoření požadovaného komunikačního spojení.

**Vytvoření spojení**

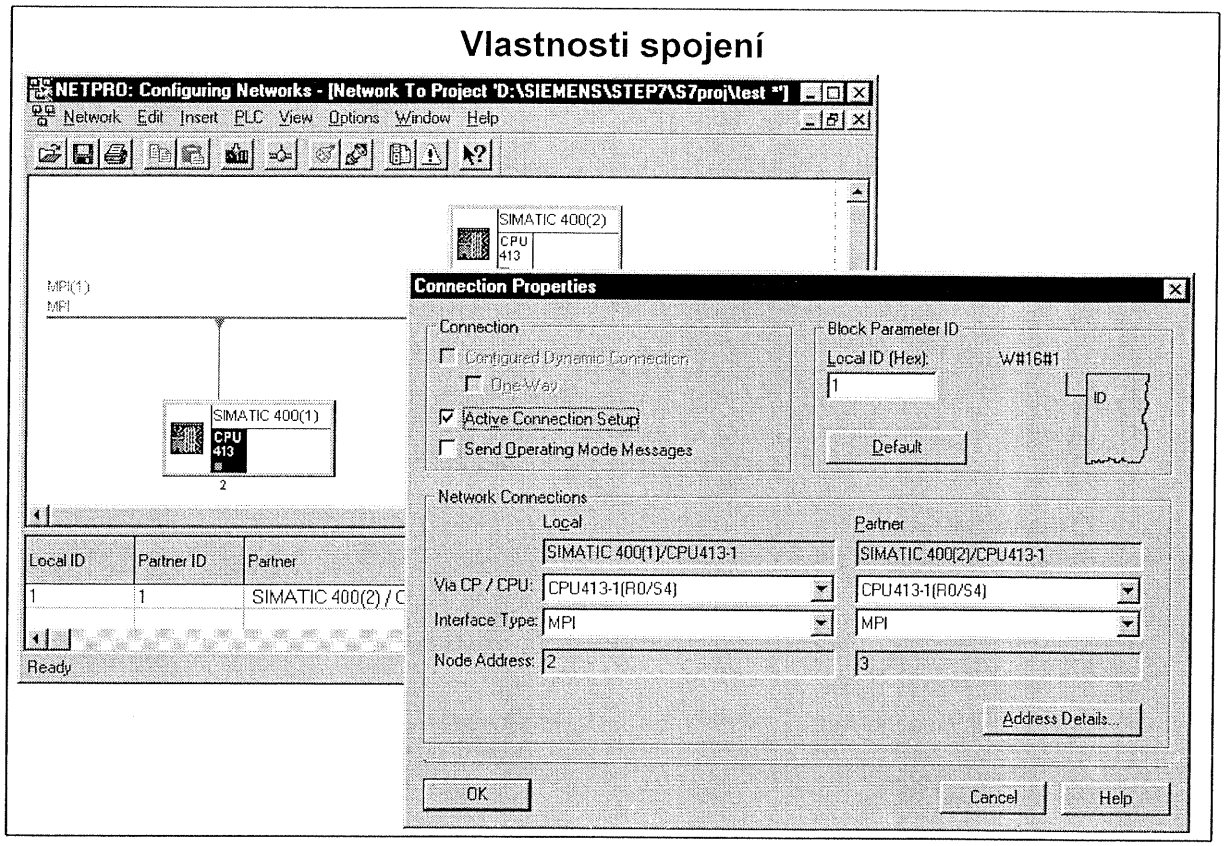
Spojení mezi lokální a vzdálenou stanicí může být vytvořeno pouze tehdy, jsou-li obě stanice připojeny do stejné subsítě.

Postup pro vytvoření nového spojení je následující:

1. V polích "Station" a "Module" vybrat CPU, pro které má být vytvořeno spojení (lokální stanice).
2. Dvojklikem na prázdném řádku v tabulce spojení nebo z nabídky *Insert -> Connection...* otevřít dialog "New Connection".
3. V polích "Station" a "Module" vybrat vzdálené CPU se kterým má být navázáno spojení (komunikační partner, vzdálená stanice)
4. V poli “Type” vybrat typ spojení (např. *S7 Single-System Connection*).
5. Vlastnosti spojení lze měnit aktivací volby "Show Properties Dialog Box".
6. Potvrdit zadané údaje tlačítkem "OK".

**Výsledek:**

STEP 7 vytvořil v tabulce spojení spojení mezi lokální stanicí (s definovaným Local ID) a vzdálenou stanicí ( s definovaným Partner ID). Uvedené identifikátory jsou potřebné při parametrizaci volaných SFB bloků.







Datum: 24.11.2002
Soubor: PRO2_10cz.16



**Přehled**

Vedle vytvoření spojení s partnerem a definice typu spojení lze, v závislosti na typu spojení, definovat i další vlastnosti.

**Vlastnosti objektu**

Postup pro definici vlastností objektu má tyto kroky:

1. Označit spojení, jehož vlastnosti budou definovány.
2. Aktivací nabídky *Edit -> Object Properties* otevřít dialogové okno "Object Properties" , ve kterém lze měnit tyto vlastnosti:

***Active Connection Set Up***

Umožňuje definovat, který z partnerů vytvoří spojení po kompletním náběhu.

***Send Operating Status Messages***

Při aktivaci lokálního uzlu je partnerovi nebo do SFB 23: USTATUS v partnerském CPU odesláno hlášení o režimu činnosti (STOP,START,HOLD,...).

***Local ID***

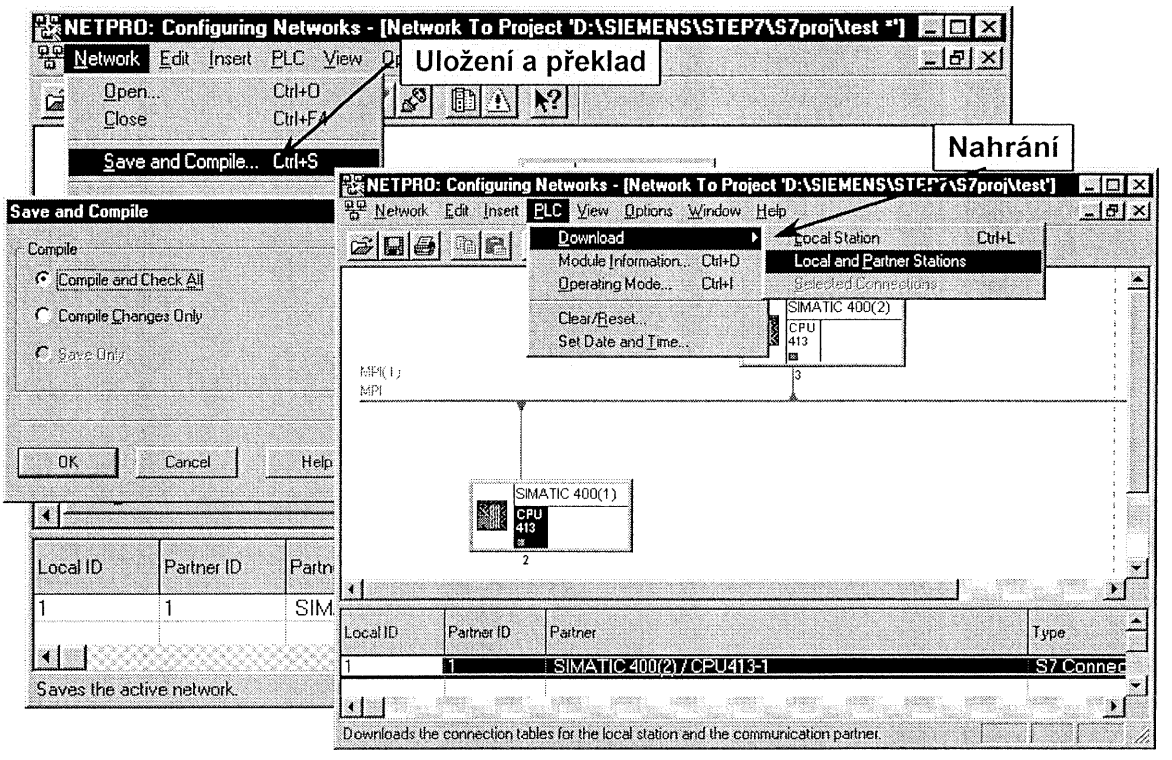
V tomto poli je zobrazeno identifikátor *local ID* modulu, jehož spojení jsou zobrazena. Tento identifikátor lze podle potřeby změnit.

***Network Connections***

Toto pole zobrazuje "cestu" datové výměny.

Jestliže existuje více možností, jak provést datovou výměnu, tzn. CPU jsou propojena pomocí více subsítí, lze touto volbou definovat cestu přes kterou bude datová výměna prováděna.

# Překlad a nahrání konfiguračních dat







Datum: 24.11.2002
Soubor: PRO2_10cz.17



**Překlad a uložení**

Před nahráním komunikačních parametrizačních dat do jednotlivých stanic je nejprve nutné provést překlad a uložit vytvořenou konfiguraci. Tato akce se provede z nabídky *File -> Save and Compile*.
Následně otevřené okno poskytuje výběr ze dvou variant překladu:

*Compile All and Check*: uloží všechna definovaná spojení a provede jejich kontrolu konzistence v rámci projektu. Všechna spojení jsou přeložena a uložena do systémových dat. V případě detekce chyby je zobrazeno příslušné chybové hlášení.

Tato volba se aktivuje, pokud byly v síti prováděny úpravy adres uzlů, rušeny celé subsítě atd.

*Compile Changes*: uloží všechna spojení v rámci projektu a přeloží pouze ta spojení, která byla od posledního překladu změněna.

Při ukončení konfigurace spojení je zobrazen dotaz, zda se změny mají uložit. Při odpovědi "Yes" budou všechna data uložena a přeložena do systémových dat.

**Nahrání konfigurace**

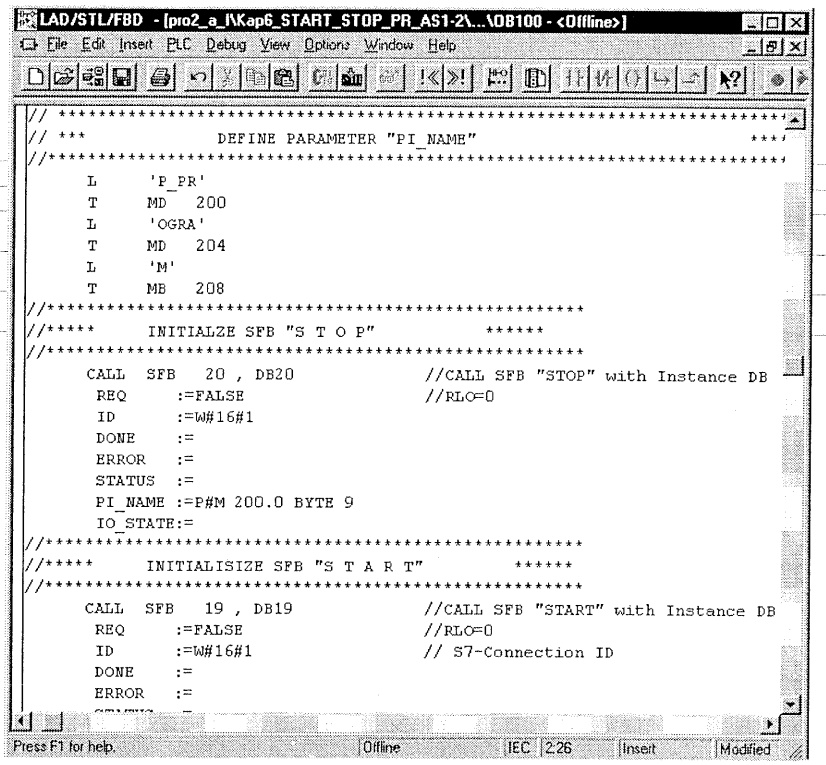
Po překladu a uložení dat je nutné tato konfigurační data nahrát do příslušných modulů. Nahrání lze provést přes MPI, PROFIBUS nebo Industrial Ethernet.

Existují 3 varianty nahrání do PLC:

- Download, Local Station (menu PLC) - do lokální stanice
- Download, Local and Partner Stations (menu PLC) - do obou partnerů
- Download, Marked Connections - podle označeného spojení

(bližší informace jsou uvedeny v elektronické nápovědě)

# Inicializace SFB v OB100






**Přehled**

Před zahájením komunikace pomocí odpovídajících SFB musí být všechny instalované SFB inicializovány pomocí OB100.

**Předpoklady**

Před začleněním jednotlivých SFB do OB100 musí být splněny následující podmínky:

- musí existovat fyzické spojení mezi partnery (kabeláž)
- musí existovat "On-line" spojení mezi partnery (test pomocí funkce "Accessible Nodes" v SIMATIC Manageru)
- data konfigurovaného spojení musí být nahrány do příslušných CPU.

**Vyvolání SFB**

Každý SFB, který se podílí na komunikaci musí být inicializován nepodmíněným voláním v OB100 s definovaným instančním DB.

Při inicializačním volání musí být definovány tyto parametry:

- REQ (aktivace vysílání s RLO = 0 )
- ID (číslo spojení)
- PI_NAME (ukazatel na text 'P_PROGRAM')
- R_ID (číslo párového bloku)
- SD_1 (ukazatel na počátek vysílané oblasti)
- RD_1 (ukazatel na počátek příjmového bufferu)
- LEN (délka přenášených dat)

**Vymazání CPU**

Vymazání paměti vede vždy ke ztrátě všech spojení. Po vymazání paměti je nutné provést kompletní náběh z důvodu inicializace OB100.

SFB v ostatních CPU rozpoznají ztrátu spojení.

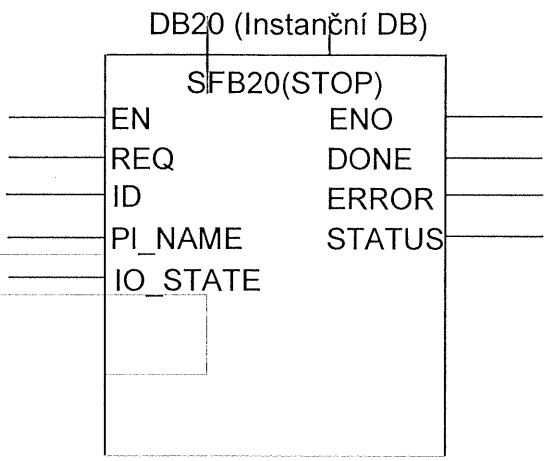
# SFB komunikace : blok STOP (SFB 20)

## STL zobrazení

s příkladem parametrizace

```
CALL STOP, I_STOP(Instanční DB)

  REQ:= I 0.0      //Startovací hrana
  ID:= W#16#1 //číslo spojení
  PI_NAME:= P#M100.0 Byte 9
  IO_STATE:=
  DONE:= #DONE_FLAG_20
  ERROR:= #ERROR_FLAG_20
  STATUS:= #STATUS_WORD_20
```

## LAD/FBD zobrazení









**Přehled**

Náběžná hrana řídícího vstupu REQ, aktivuje SFB20 (STOP) a následně STOP stav na vzdálené stanici s identifikátorem ID. Změna režimu činnosti je možná pouze , je-li vzdálená stanice v režimu RUN, HOLD nebo náběhu.

Úspěšné provedení úlohy je signalizováno 1 v parametru DONE. Případná chyba je signalizována v parametrech ERROR a STATUS.

Režim činnosti lze obnovit pouze ze vzdálené stanice, která volala SFB20.

| Parametr | Deklarace | Typ | Význam |
|---|---|---|---|
| REQ | INPUT | BOOL | nábì žná hrana aktivuje STOP na stanici definované ID |
| ID | INPUT | WORD (I,Q;M,D,L , constant) | číslo spojení definované v tabulce spojení |
| PI_NAME | IN_OUT | ANY | ukazatel na textový řetězec - jméno startovaného programu. U S7 je jméno P_PROGRAM. |
| IO_STATE | IN_OUT | BYTE | bez významu |
| DONE | OUTPUT | BOOL | náběžná hrana = provedeno |
| ERROR | OUTPUT | BOOL | náběžná hrana = chyba |
| STATUS | OUTPUT | WORD | obsahuje detailní kód chyby nebo varování |

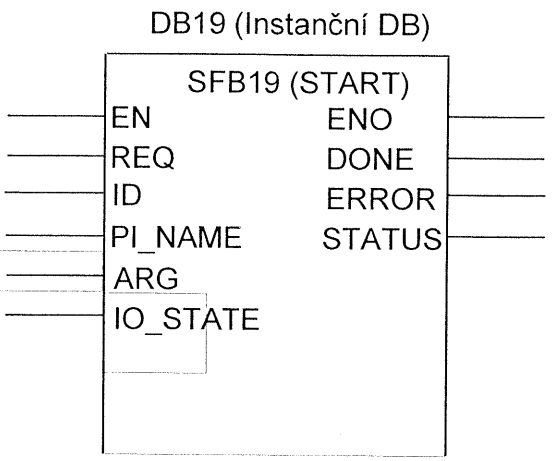
# SFB komunikace : blok START (SFB 19)

## STL zobrazení

s příkladem parametrizace

```
CALL START, I_START (Instanční DB)

  REQ:=I 0.1          //Start edge
  ID:=W#16#1          //Connection no.
  PI_NAME:=P#M100.0 Byte 9 //*
  ARG:=
  IO_STATE:=
  DONE:= #DONE_FLAG_19
  ERROR:= #ERROR_FLAG_19
  STATUS:= #STATUS_WORD_19
```

## LAD/FBD zobrazení




* umístění textu 'P_PROGRAM"




Datum: 24.11.2002
Soubor: PRO2_10cz.20



**Přehled**

Náběžná hrana REQ, aktivuje SFB19 (START) a vyvolá na vzdálené stanici identifikované adresou ID kompletní náběh. Musí být splněny tyto podmínky:

- vzdálené CPU musí být ve STOP stavu.
- přepínač režimů CPU musí být v poloze "RUN" nebo "RUN-P".

Po provedení kompletního náběhu přejde vzdálené CPU do RUN režimu a odešle potvrzení o provedení. Toto provedení je signalizováno 1 v parametru DONE. Chyba je signalizována v parametrech ERROR a STATUS.

| Parametr | Deklarace | Typ | Význam |
|---|---|---|---|
| REQ | INPUT | BOOL | náběžná hrana aktivuje kompletní náběh na stanici ID |
| ID | INPUT | WORD (I,Q,M,D,L, constant) | číslo spojení z tabulky spojení |
| PI_NAME | IN_OUT | ANY | ukazatel na jméno programu P_PROGRAM |
| ARG | IN_OUT | ANY | bez významu |
| IO_STATE | IN_OUT | ANY | bez významu |
| DONE | OUTPUT | BOOL | náběžná hrana = provedeno |
| ERROR | OUTPUT | BOOL | náběžná hrana = chyba |
| STATUS | OUTPUT | WORD | chybové nebo varovné hlášení |

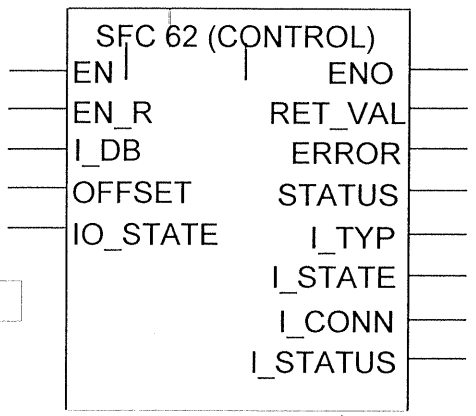
# SFB komunikace : blok CONTROL (SFC 62)

## STL zobrazení

s příkladem parametrizace

```
CALL "CONTROL"
EN_R:= I 0. 2  //Start
I_DB:= W#16#F //Instance DB-NR
OFFSET:= W#16#0
RET_VAL:= MW4 //Error
ERROR:=Q0.4 //Error-Bool
STATUS:= MW 4 //Status display
I_TYP:= MB 52 //SFB-TYPE
I_STATE:= MB 53 //SFB state
I_CONN:= M 54.0 //Connection state
I_STATUS:= MW102 //Status of the SFB
```

## LAD zobrazení







Datum: 24.11.2002
Soubor: PRO2_10cz.21



**Přehled**

Pomocí SFC62 "CONTROL" lze kontrolovat stav spojení vytvořeného pomocí instance SFB.

Úroveň 1 na vstupu EN_R aktivuje kontrolu spojení realizovaného SFB pomocí instančního DB I_DB.

| Parametr | Deklarace | Typ | Význam |
|---|---|---|---|
| EN_R | INPUT | BOOL | aktivace fukce |
| I_DB | INPUT | BLOCK_DB (I,Q,M,D,L,konst.) | číslo instančního DB |
| OFFSSET | INPUT | WORD (I,Q,M,D,L, konst.) | offset - posun v multiinstančním DB (bez multiinst. =0) |
| RET_VAL | OUTPUT | INT (I,Q,M,D,L) | 8000H chyba SFC62 |
| ERROR | OUTPUT | BOOL (I,Q,M,D,L) | chyba při zpracování SFC62 →RLO=1 |
| STATUS | OUTPUT | WORD (I,Q,M,D,L) | chybové hlášení SFC62 |
| I_TYP | OUTPUT | BYTE (I,Q,M,D,L) | identifikátor typu CFB |
| I_STATE | OUTPUT | BYTE (I,Q,M,D,L) | současný stav CFB |
| I_CONN | OUTPUT | BOOL (I,Q,M,D,L) | stav spojení: 0 = spojení zrušeno 1 = spojení existuje |
| I_STATUS | OUTPUT | WORD (I,Q,M,D,L) | chybové hlášení nebo stav SFB |

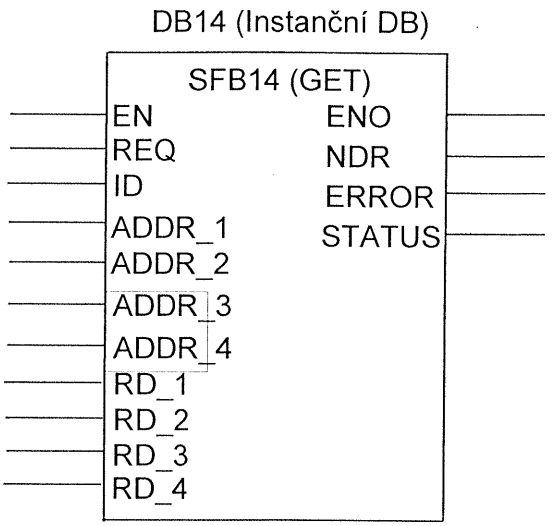
# SFB komunikace : blok GET (SFB 14)

## STL zobrazení

s příkladem parametrizace

```
CALL GET, I_GET (Instanční DB)
REQ:=I 0.2
ID:=W#16#1          //Connect. No.
NDR:=#NDR_FLAG
ERROR:= #ERROR_FLAG
STATUS:= #STATUS_WORD
ADDR_1:=P#I 0.0 BYTE 1
ADDR_2:=P#I 4.0 BYTE 2
ADDR_3:=
ADDR_4:=
RD_1:=P#Q 0.0 BYTE 1
RD_2:=P#Q 4.0 BYTE 2
RD_3
RD_4
```

## LAD zobrazení







Datum: 24.11.2002
Soubor: PRO2_10cz.22



**Přehled**

SFB14 (GET) umožňuje číst data ze vzdálené stanice

Náběžná hrana REQ odešle úlohu čtení na vzdálenou stanici a ta vrátí data

Po bezchybném provedení jsou přijatá data uložena do oblastí definovaných při volání (RD_i) SFB. Dokončení přenosu je signalizováno hodnotou 1 v parametru NDR.

| Parametr | Deklarace | Typ | Význam |
|---|---|---|---|
| REQ | INPUT | BOOL (I,Q,M,D,L constant) | náběžná hrana aktivuje přenos |
| ID | INPUT | WORD (I,Q,M,D,L constant) | číslo spojení definované v tabulce spojení |
| ADDR_1 ... ADDR_4 | IN_OUT | ANY (I,Q,M,D) | ukazatele na zdrojové (čtené) oblasti ve vzdáleném CPU. |
| RD_1 ... RD_4 | IN_OUT | ANY (I,Q,M,D) | ukazatele na cílové oblasti kam budou načtená data po přijetí uložena. (data z oblasti **ADDR_1** ==>budou uložena do oblasti definované **RD_1**, atd.) |
| NDR | OUTPUT | BOOL (I,Q,M,D,L) | náběžná hrana signalizuje bezchybný příjem nových dat |
| ERROR | OUTPUT | BOOL (I,Q,M,D,L) | náběžná hrana signalizuje chybu. |
| STATUS | OUTPUT | WORD (I,Q,M,D,L) | obsahuje detailní kód chyby nebo varování. |

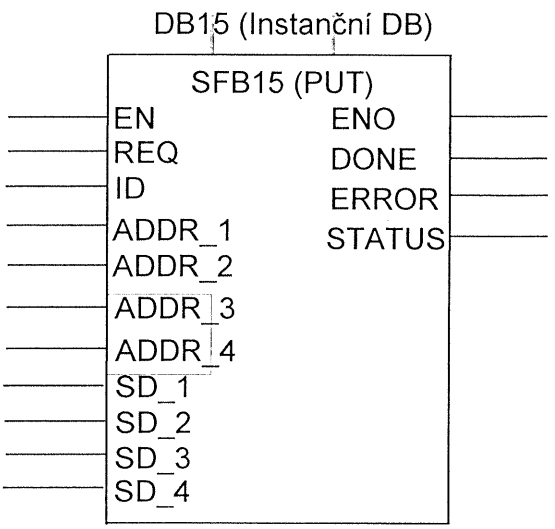
# SFB komunikace : blok PUT (SFB 15)

## STL zobrazení

s příkladem parametrizace

```
CALL PUT, I_PUT(Instanční DB)
REQ:=I 0.3
ID:=W#16#1          //Connect. No.
DONE:= #DONE_FLAG
ERROR:= #ERROR_FLAG
STATUS:= #STATUS_FLAG
ADDR_1:=P#Q 12.0 WORD 1
ADDR_2:=
ADDR_3:=
ADDR_4:=
SD_1:=P#I 2.0 WORD 1
SD_2:=
SD_3:=
SD_4:=
```

## LAD zobrazení









**Přehled**

SFB15 (PUT) umožňuje zápis dat do vzdálené stanice.

Náběžná hrana REQ aktivuje zápis z oblasti SD_i do vzdálené stanice do oblasti ADDR_i. Po bezchybném uložení dat potvrdí vzdálená stanice provedení akce.

| Parametr | Deklarace | Typ | Význam |
|---|---|---|---|
| REQ | INPUT | BOOL (I,Q,M,D,L constant) | náběžná hrana aktivuje přenos |
| ID | INPUT | WORD (I,Q,M,D,L constant) | číslo spojení uvedené v tabulce spojení |
| ADDR_1 ... ADDR_4 | IN_OUT | ANY (I,Q,M,D) | ukazatel na cílovou oblast vzdáleného partnera - kam se data mají zapsat |
| SD_1 ... SD_4 | IN_OUT | ANY (I,Q,M,D) | ukazatel na odesílaná data **(SD_1 → ADDR_1, atd.)** |
| DONE | OUTPUT | BOOL (I,Q,M,D,L) | náběžní hrana signalizuje bezchybý přenos |
| ERROR | OUTPUT | BOOL (I,Q,M,D,L) | náběžná hrana signalizuje chybu při přenosu |
| STATUS | OUTPUT | WORD (I,Q,M,D,L) | detailní kód chyby nebo varování |

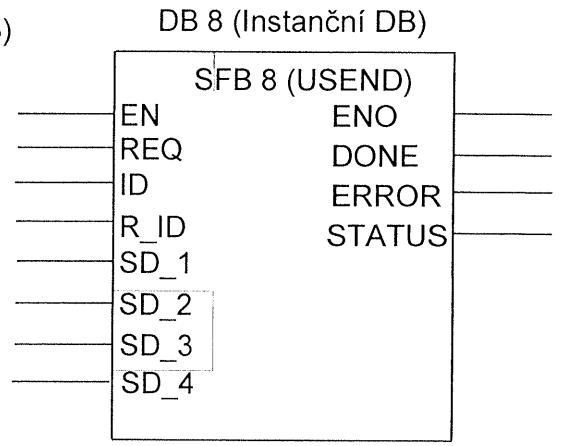
# SFB komunikace : blok USEND (SFB 8)

## STL zobrazení

s příkladem parametrizace

```
CALL USEND, I_USEND(Instanční DB)
REQ:= I 0.4
ID:=W#16#3          //Connect. No.
R_ID:=DW#16#B1 //Block pair
DONE:= #DONE_FLAG
ERROR:= #ERROR_FLAG
STATUS:= #STATUS_WORD
SD_1 :=P#DB3.DBX0.0 BYTE 100
SD_2 :=P#DB3.DBX100.0 BYTE 100
SD_3 :=P#DB3.DBX200.0 BYTE 100
SD_4 :=P#DB3.DBX300.0 BYTE 154
```

## LAD zobrazení







Datum: 24.11.2002
Soubor: PRO2_10cz.24



**Přehled**

SFB8 (USEND) odvysílá data partnerovi s SFB - URCV a identifikátorem definovaným v R_ID. Data budou odeslána s náběžnou hranou REQ. Vysílání není koordinováno se vzdálenou stanicí.

Přenášená data jsou definována ukazatelem v SD_1 až SD_4, ale není nutné použít všechny oblasti.

| Parametr | Deklarace | Typ | Význam |
|---|---|---|---|
| REQ | INPUT | BOOL (I,Q,M,D,L constant) | Náběžná hrana aktivuje přenos dat |
| ID | INPUT | WORD (I,Q,M,D,L constant) | Číslo spojení definované v tabulce spojení |
| R_ID | INPUT | WORD (I,Q,M,D,L constant) | Identifikátor příjemce, musí být shodný u obou bloků (USEND a URCV). |
| DONE | OUTPUT | BOOL (I,Q,M,D,L) | náběžná hrana signalizuje bezchybné ukončení přenosu |
| ERROR | OUTPUT | BOOL (I,Q,M,D,L) | náběžná hrana signalizuje chybu přenosu |
| STATUS | OUTPUT | BOOL (I,Q,M,D,L) | chybové hlášení p ři detekci chyby ( ERROR = 1) |
| SD_1 ... SD_4 | IN_OUT | ANY (I,Q,M,D) | Ukazatel na odesílanou oblast dat (Data z obiasti **SD_1** budou přijata do oblasti RD_1 na vzdálené stanici, musí souhlasit po čet ,délka a typ dat) |

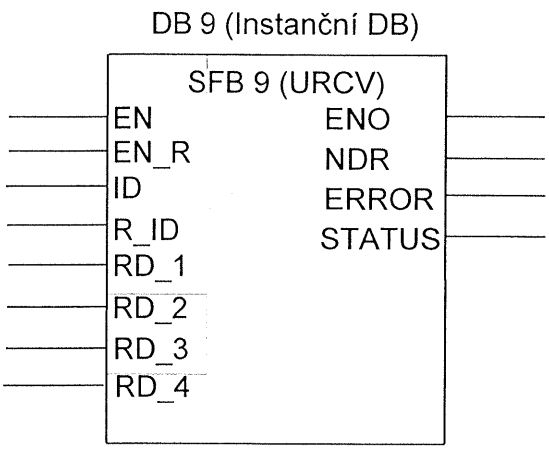
# SFB komunikace : blok URCV (SFB 9)

## STL zobrazení

s příkladem parametrizace

```
CALL URCV, I_URCV(Instanční DB)
EN_R:= I 0.5
ID:= W#16#3          //S7 connection
R_ID:= DW#16#B1   //Block pair
NDR:= #NDR_FLAG
ERROR:= #ERROR_FLAG
STATUS:= #STATUS_WORD
RD_1:=P#DB3.DBX0.0 BYTE 100
RD_2:=P#DB3.DBX100.0 BYTE 100
RD_3:=P#DB3.DBX200.0 BYTE 100
RD_4:=P#DB3.DBX300.0 BYTE 154
```

## LAD zobrazení




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_10cz.25

Školicí středisko
firmy E&A spol. s r.o., MB

**Přehled**

SFB9 (URCV) zajistí asynchronní příjem dat odeslaných ze vzdálené stanice pomocí SFB "USEND" se stejným R_ID, který je definován při volání tohoto bloku. Aktivace EN_R =1 zkopíruje přijatá data do oblastí definovaných v parametrech RD_1 až RD_4.

Při prvním volání je vytvořen přijímací buffer, do kterého jsou vždy umístěna přijatá data.

| Parametr | Deklarace | Typ | Význam |
|---|---|---|---|
| EN_R | INPUT | BOOL (I,Q,M,D,L constant) | s RLO = 1 jsou přijatá data přenesena do definovaných oblastí |
| ID | INPUT | WORD (I,Q,M,D,L constant) | číslo spojení |
| R_ID | INPUT | DWORD (I,Q,M,D,L constant) | identifikátor příjemce (shodný na obou stranách USEND a URCV). |
| NDR | OUTPUT | BOOL (I,Q,M,D,L) | náběžná hrana siganlizuje příjem nových dat |
| ERROR | OUTPUT | BOOL (I,Q,M,D,L) | náběžná hrana = chyba |
| STATUS | OUTPUT | BOOL (I,Q,M,D,L) | stavové hlášení je-li ERROR = 1 |
| RD_1<br>...<br>RD_4 | IN_OUT | ANY (I,Q,M,D) | ukazatele na přijímací oblasti (u SD_i a RD_i musí souhlasit počet, délka a datový typ proměnných) |

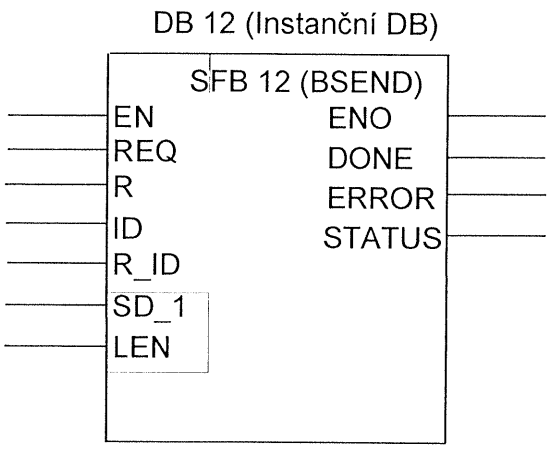
# SFB komunikace : blok BSEND (SFB 12)

## STL zobrazení

s příkladem parametrizace

```
CALL BSEND, I_BSEND(Instanční DB)
REQ:= I 0.4
R:= I0.5
ID:=W#16#3          //S7 Connection
R_ID:=DW#16#B2
DONE:= #DONE_FLAG
ERROR:= #ERROR_FLAG
STATUS:= #STATUS_WORD
SD_1:=P#DB1.DBX0.0 BYTE 40000
LEN:= #DB_LEN
```

## LAD zobrazení







Datum: 24.11.2002
Soubor: PRO2_10cz.26



**Přehled**

SFB12 (BSEND) odešle data vzdálené stanici s SFB "BRCV". (Identifikátor příjemce R_ID musí být na obou stranách shodný. Tento blok umožňuje přenos až 64KB dat ( pro všechny CPU). Přenos je aktivován náběžnou hranou REQ. Přenos dat je asynchronní vůči cyklickému zpracování programu.

| Parameter | Deklarace | Typ | Význam |
|---|---|---|---|
| REQ | INPUT | BOOL (I,Q,M,D,L constant) | náběžná hrana aktivuje přenos |
| R | INPUT | BOOL (I,Q,M,D,L constant) | náběžná hrana inicializuje BSEND do výchozího stavu |
| ID | INPUT | WORD (I,Q,M,D,L constant) | číslo spojení |
| R_ID | INPUT | DWORD (I,Q,M,D,L) | identifikátor příjemce, shodný na obou stranách komunikace |
| SD_1 | IN_OUT | ANY (I,Q,M,D,L) | ukazatel na odesílaná data, délka není vyhodnocována |
| LEN | IN_OUT | WORD (I,Q,M,D,L) | délka přenášené datové oblasti |
| DONE | OUTPUT | BOOL (I,Q,M,D,L) | náběžná hrana signalizuje bezchybný přenos |
| ERROR | OUTPUT | BOOL (I,Q,M,D,L) | náběžná hrana signalizuje chybu při přenosu |
| STATUS | OUTPUT | WORD (I,Q,M,D,L) | detailní kód chyby nebo varování |

# SFB komunikace : blok BRCV (SFB 13)

## STL zobrazení

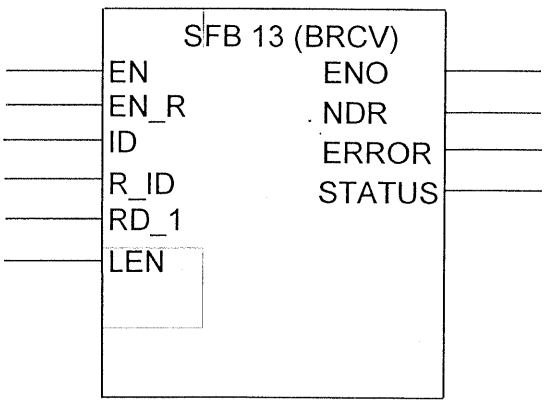
s příkladem parametrizace

CALL BRCV, I_BRCV(Instanční DB)

```
EN_R:= I 0.4
ID:=W#16#3   //S7 oboustranné spojení
R_ID:=DW#16#B2
NDR:= #NDR_FLAG
ERROR:= #ERROR_FLAG
STATUS:= #STATUS_WORD
RD_1:=P#DB2.DBX0.0 BYTE 40000
LEN:= #DB_LEN
```

## LAD zobrazení

DB 13 (Instanční DB)









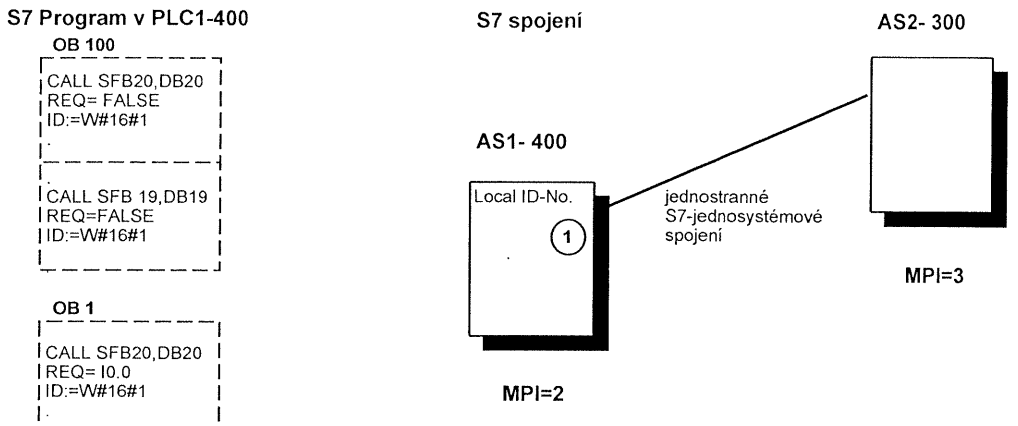
**Přehled**

SFB13 (BRCV) přijme data odeslaná ze vzdálené stanice SFB "BSEND". Identifikátor příjemce R_ID musí být ale na obou stranách shodný. EN_R = 1 povoluje příjem dat do oblasti definované RD_1.

Následně přijaté segmenty jsou potvrzeny na stranu odesilatele a parametr LEN je aktualizován. Opětovné vyvolání během asynchronního přenosu vede k signalizaci varování v parametru STATUS. Jestliže je opětovné volání provedeno s EN_R=0, je přenos zrušen a SFB přejde do inicializačního stavu. Bezchybně ukončený přenos všech datových segmentů je signalizován NDR = 1.

| Parametr | Deklarace | Typ | Význam |
|---|---|---|---|
| EN_R | INPUT | BOOL (I,Q,MD,Lconst.) | RLO = 1 uvolňuje příjem<br>RLO = 0 příjem přerušen - zablokován |
| ID | INPUT | WORD (I,Q,MD,Lconst.) | číslo spojení |
| R_ID | INPUT | DWORD (I,Q,MD,L const.) | identifikátor příjemce - stejný na obou stranách |
| RD_1 | IN_OUT | ANY | ukazatel příjmové oblasti. Délka udává maximální možnou délku přijaté oblasti |
| LEN | IN_OUT | WORD | délka přijímaných dat (v bytech) |
| NDR | OUTPUT | BOOL | náběžná hrana signalizuje přijetí nových dat |
| ERROR | OUTPUT | BOOL | náběžná hrana signalizuje chybu |
| STATUS | OUTPUT | WORD | detailní kód chyby nebo varování |

# Cvičení 10.1: Komunikace s SFB START/STOP







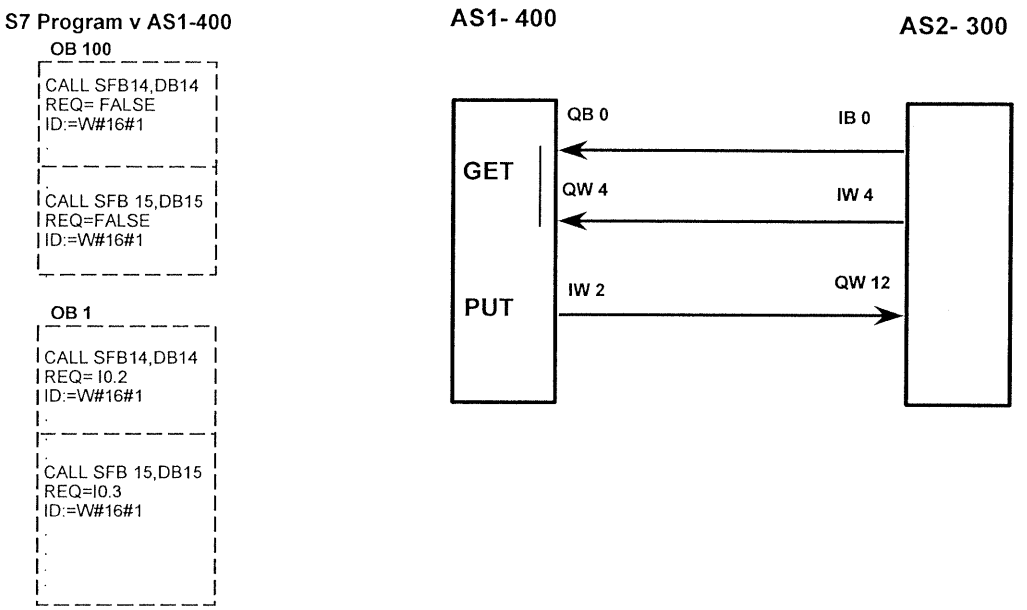
Datum: 24.11.2002
Soubor: PRO2_10cz.28



Cvičení 10.1

| Krok | Postup | Poznámka |
|---|---|---|
| 1 | Vytvořit nový projekt „SFB-Comm“ | |
| 2 | - Vytvořit 2 HW stanice HW Stations: AS1-400 a AS2-300<br>- Propojit HW Stanice přes MPI síť<br>- Zkontrolovat propojení pomocí funkce „Accessible Nodes“<br>- Přejmenovat CPU na CPU413-AS1 a CPU314-AS2 | |
| 3 | - Vytvořit mezi oběma CPU S7 jednosystémové propojení<br>- Nahrát tabulku propojení do PLC S7-400 | |
| 4 | - V projektu vytvořit složku S7 program<br>- Vytvořit OB100<br>- V OB100 volat inicializaci CFB „START“ a „STOP“ | |
| 5 | - Editovat OB1<br>- Vytvořit segment “SFB_STOP“ a volat CFB “STOP“:<br>- Vytvořit segment “SFB_START“ a volat CFB “START“: | - Start přes I0.0<br>- Start přes I0.1 |
| 6 | - Výsledek volání zobrazovat na display QW2 | |

# Cvičení 10.2: Komunikace s SFB GET/PUT







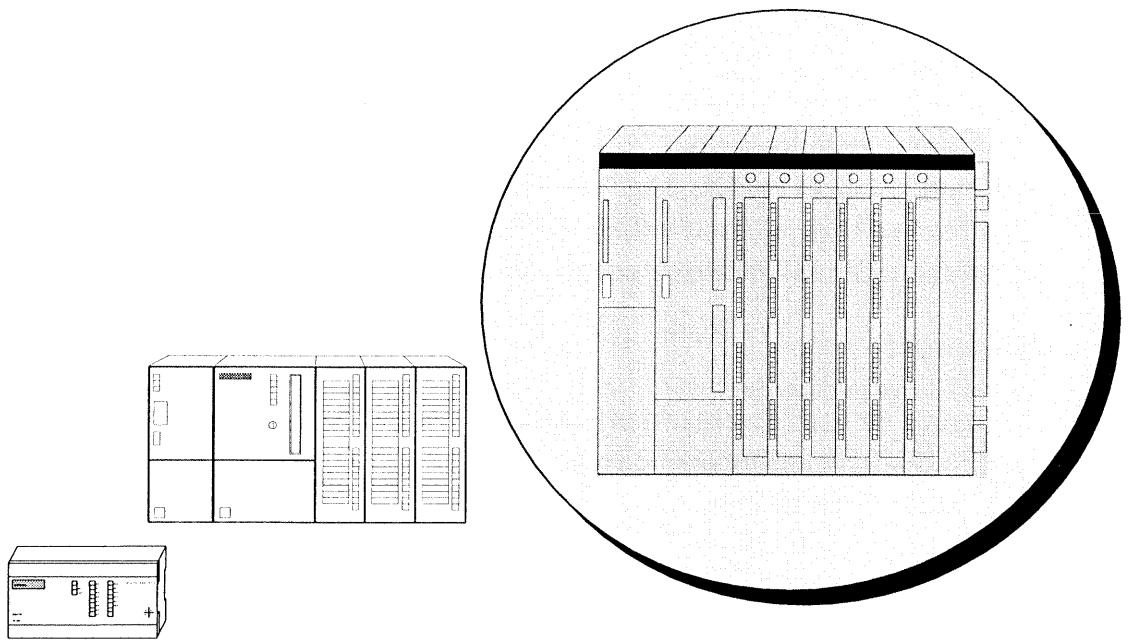
Datum: 24.11.2002
Soubor: PRO2_10cz.29



Cvičení 10.2

| Krok | Postup | Poznámka |
|---|---|---|
| 1 | - v projektu vytvořit složku S7 program<br>- vytvořit OB100<br>- v OB100 volat inicializaci SFB „GET" a „PUT" | |
| 2 | -editovat OB1<br>- vytvořit segment „SFB-GET" a volat v něm SFB „GET"<br>- pomocí SFB „GET" číst IB0 z PLC2-300 a uložit ho do QB 0 v PLC1-400<br>- číst IW4 z PLC2-300 a uložit do QW4 PLC1-400 | - aktivovat I0.2 |
| 3 | - vytvořit segment „SFB-PUT" a volat SFB „PUT"<br>- pomocí SFB „PUT" zapsat IW2 z PLC1-400 do QW12 PLC2-300 | - aktivovat I0.3 |
| 4 | výsledek komunikace zobrazit na display QW2 na PLC1-400 | |

# PLC S7-400



SIMATIC S7



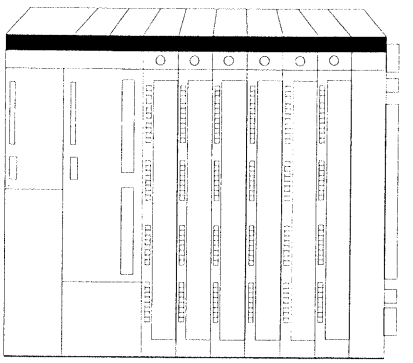
Datum: 24.11.2002
Soubor: PRO2_11cz.1

Školicí středisko
firmy E&A spol. s r.o.

**Obsah** **Strana**

## SIMATIC S7-400: Přehled

- **Otevřený systém**
  - **výkonný**
  - **odstupňovaný výkon CPU**
  - **víceprocesorový režim**
  - **připojení až 21 rozšiřujících jednotek**
  - **široké spektrum modulů (SM, FM, CP)**
  - **široké možnosti síťových propojení**
- **Výkonnost**
  - **výkonné procesory (pouze 80 nsec na binární příkaz)**
  - **až 1.6 MByte uživatelské paměti**
  - **výkonná komunikace**
- **Možnosti**
  - **speciální funkce**
  - **zvláštní možnosti rozšíření přes PLC S5**



SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.2

Školicí středisko firmy E&A spol. s r.o.

**SIMATIC S7-400**

SIMATIC S7-400 je PLC středního až velkého výkonu. Modulární, vysoce otevřený, robustní s širokou možností komunikace nabízí řešení ve všech oblastech řízení procesů.

**Otevřený systém**

Za pozornost stojí u S7-400:

- jednoduchost montáže modulů, v porovnání s S5 byl montážní prostor redukován o 54%.
- S7-400 nabízí výkonnostně odstupňované CPU moduly, symetrický a asymetrický víceprocesorový režim.
- široké spektrum modulů pro všechny aplikace (signální moduly, funkční moduly, komunikační moduly)
- rozšiřitelnost až o 21 rozšiřujících nosičů modulů a o distribuované jednotky přes PROFIBUS-DP
- možnost síťového propojení přes MPI, PROFIBUS a Industrial Ethernet.

**Výkon**

- rychlé procesory s 80 ns na binární instrukci a až 1.6 MByte uživatelské paměti umožňují řešit i ty nejsložitější automatizační úlohy.
- výkonná komunikační sběrnice (10.5 MBaud) poskytuje rychlou komunikaci pro datovou výměnu.

**Všestrannost**

- univerzální funkce, např. restart, odebírání a vkládání modulů v režimu RUN, atd.
- pro doplnění, S7-400 umožňuje použití existujících modulů určených pro bS5:
  - S5-IP nebo WF v S7 centrálním nosiči
  - připojení S5 rozšiřujících nosičů k S7 centrálnímu nosiči.

## S7-400: základní odlišnosti od S7-300

- větší paměť a větší počet I/O/M/T/C
- parametrizovatelné adresované vstupy a výstupy
- lze připojit ER - S5 a použít moduly S5 CP/IP
- větší počet systémových funkcí, např. komunikační bloky
- velikost bloku až 64KB
- kompletní náběh a náběh ("studený" a "teplý" restart)
- porovnávání nastavené a aktuální konfigurace při náběhu
- odebírání modulů pod napětím
- několik částí obrazu procesu
- parametrizace priorit OB
- větší počet OB pro stopky - watchdog, HW a time-of-day přerušení
- hloubka vnoření až 16 úrovní
- nastavitelná velikost L-Stacku pro každou prioritní třídu
- víceprocesorový režim s až 4 CPU

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_11cz.3

Školicí středisko
firmy E&A spol. s r.o.

**Odlišnosti**

Obrázek ukazuje základní rozdíly mezi S7-400 a S7-300.
Jednotlivé body jsou detailně rozebrány na následujících stránkách.

## Technická data S7-400 CPU (1)

| | CPU 412-1 | CPU 413-1 | CPU 413-2 DP | CPU 414-1 | CPU 414-2 DP | CPU 416-1 | CPU 416-2 DP |
|---|---|---|---|---|---|---|---|
| Doba zpracování binární instrukce | 200 nsec | 200 nsec | 200 nsec | 100 nsec | 100 nsec | 80 nsec | 80 nsec |
| Load/Transfer (Word) | 200 nsec | 200 nsec | 200 nsec | 100 nsec | 100 nsec | 80 nsec | 80 nsec |
| 16-Bit fixed point (+/-) | 200 nsec | 200 nsec | 200 nsec | 100 nsec | 100 nsec | 80 nsec | 80 nsec |
| IEEE floating point (+/-) | 1200 nsec | 1200 nsec | 1200 nsec | 600 nsec | 600 nsec | 480 nsec | 480 nsec |
| Uživatelská paměť | | | | | | | |
| pracovní | 48 KB | 72 KB | 72 KB | 128 KB | 128/384 KB | 512 KB | 0.8/1.6 MB |
| editační (integr.) | 8 KB | 8 KB | 8 KB | 8 KB | 8 KB | 16 KB | 16 KB |
| editační (extern.) | 15 MB | 15 MB | 15 MB | 15 MB | 15 MB | 15 MB | 15 MB |
| Adresy | | | | | | | |
| M-Bity | 4096 | 4096 | 4096 | 8192 | 8192 | 16384 | 16384 |
| Hodiny | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| Časovače | 256 | 256 | 256 | 256 | 256 | 512 | 512 |
| Čitače | 256 | 256 | 256 | 256 | 256 | 512 | 512 |
| Bloky | | | | | | | |
| FB | 256 | 256 | 256 | 512 | 512 | 2048 | 2048 |
| FC | 256 | 256 | 256 | 1024 | 1024 | 2048 | 2048 |
| DB | 511 | 511 | 511 | 1023 | 1023 | 4095 | 4095 |
| Obraz technologie (vstupy/výstupy) | p. 128 Byte | p. 128 Byte | p. 128 Byte | p. 256 Byte | p. 256 Byte | p. 512 Byte | p. 512 Byte |
| max. I/O adresní prostor | p.0.5KByte*) | p.1 KByte*) | p. 1 KByte*) | p. 2 KByte*) | p. 4 KByte*) | p. 4 KByte*) | p. 8 KByte*) |
| Rozhraní integrované | MPI | MPI | MPI, DP | MPI | MPI, DP | MPI | MPI, DP |

*) 1 Byte = 8 digital I/Os
2 Byte = 1 analog I/O

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_11cz.4

Školicí středisko
firmy E&A spol. s r.o.

### Typy CPU

Pro každý výkonnostní rozsah je k dispozici odpovídající CPU s odpovídající dobou cyklu, velikostí paměti a počtem bloků.

### Programování

Tvorba uživatelského programu je v souladu s normou IEC 1131-3. V S7 jsou integrovány následující charakteristiky:

- Programové instrukce pro zpracování analogové hodnoty
  - aritmetika s pevnou a plovoucí desetinnou čárkou (32 bit)
  - druhá mocnina a odmocnina
  - logaritmické funkce
  - trigonometrické funkce
- nové datové typy:
  - ARRAY (pole)
  - STRUCT (struktura)
  - POINTER (ukazatel)
- objektově orientovaný přístup :
  - FB s instančním DB
  - Multi-instanční model
- integrované systémové bloky, např. pro komunikaci, atd.

### Procesní I/O

Logické adresy I/O modulů jsou lineárně definovatelné v adresním prostoru přiměřené velikosti. Neexistují P, Q periferie.

Adresy DP-slave stanic jsou také zobrazeny v lineárním adresním prostoru. Díky tomu mohou být distribuované periferie adresovány v uživatelském programu stejným způsobem jako centrální I/O.

Parametrizace centrálních i distribuovaných I/O se provádí pomocí STEPu 7.

# Technická data CPU S7-400 (2)

| | CPU 412-1 | CPU 413-1 | CPU 413-2 | CPU 414-1 | CPU 414-2 | CPU 416-1 | CPU 416-2 |
|---|---|---|---|---|---|---|---|
| Organizační bloky | | | | | | | |
| cyklické zpracování | OB 1 | OB 1 | OB 1 | OB 1 | OB 1 | OB 1 | OB 1 |
| Time-of-Day přerušení | OB 10,11 | OB 10,11 | OB 10,11 | OB 10-13 | OB 10-13 | OB 10-17 | OB 10-17 |
| Time-delay přerušení | OB 20,21 | OB 20,21 | OB 20,21 | OB 20-23 | OB 20-23 | OB 20-23 | OB 20-23 |
| Watchdog přerušení | OB 32,35 | OB 32,35 | OB 32,35 | OB 32-35 | OB 32-35 | OB 30-38 | OB 30-38 |
| Hardware přerušení | OB 40,41 | OB 40,41 | OB 40,41 | OB 40-43 | OB 40-43 | OB 40-47 | OB 40-47 |
| Multicomputing přerušení | OB 60 | OB 60 | OB 60 | OB 60 | OB 60 | OB 60 | OB 60 |
| Start-up | OB100,101 | OB100,101 | OB100,101 | OB100,101 | OB100,101 | OB 00,101 | OB 100,101 |
| Error, asynchronous | OB 80-87 | OB 80-87 | OB 80-87 | OB 80-87 | OB 80-87 | OB 80-87 | OB 80-87 |
| Error, synchronous | OB121,122 | OB121,122 | OB121,122 | OB121,122 | OB121,122 | OB121,122 | OB 121,122 |
| Lokální data | 4 KB | 4 KB | 4 KB | 8 KB | 8 KB | 16 KB | 16 KB |
| Max. délka bloku | 64 KB | 64 KB | 64 KB | 64 KB | 64 KB | 64 KB | 64 KB |
| Hloubka vnoření | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| Programové řízení komunikace, max. počet kanálů MPI, K bus | 8 | 16 | 16 | 32 | 32 | 64 | 64 |
| Cyklická datová komunikace přes MPI, Global Data objem GD okruhu | 54 Byte Uživ. dat | 54 Byte Uživ. dat | 54 Byte Uživ. dat | 54 Byte Uživ. dat | 54 Byte Uživ. dat | 54 Byte Uživ. dat | 54 Byte Uživ. dat |
| GD okruhy / CPU | 8 | 8 | 8 | 8 | 8 | 16 | 16 |
| Vysílané GD Pakety / GD okruh | 1 | 1 | 1 | 1 | 1 | 1 | 1x 16 x 54 Byte= 864 Byte |
| Přijímané GD Pakety / GD okruh | 2 | 2 | 2 | 2 | 2 | 2 | 2 x 16 x 54 Byte= 1728 Byte |

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.5

Školicí středisko firmy E&A spol. s r.o.

**Komunikace**

S7-400 poskytuje několik různých možností komunikace:

1. integrované rozhraní Multi-Point-Interface (MPI), několika bodové komunikační rozhraní pro připojení PG/PC, HMI systémů, M7-300/400 a dalších S7-300/400 systémů
2. integrované rozhraní PROFIBUS-DP u CPU 413-2/414-2/416-2 pro připojení DP Slave systémů (např.. ET200) k CPU.
3. komunikační procesory (např. CP443) pro připojení na PROFIBUS a Industrial Ethernet.
4. komunikační procesory (např. CP441) pro *point-to-point* (PtP) komunikaci se systémem S7, S5 nebo externím řídícím systémem.

**S7 funkce**

S7 komunikační funkce lze rozdělit do dvou skupin:

S7 základní komunikace: pomocí MPI lze mezi stanicemi (S7-300/400) nebo uvnitř stanice přenášet až 76 bytů.

Příslušné SFC bloky jsou integrována do operačního systému CPU. Tato komunikace nevyžaduje konfiguraci spojení, komunikační zdroje a adresa partnera je definována přímo při volání SFC bloku.

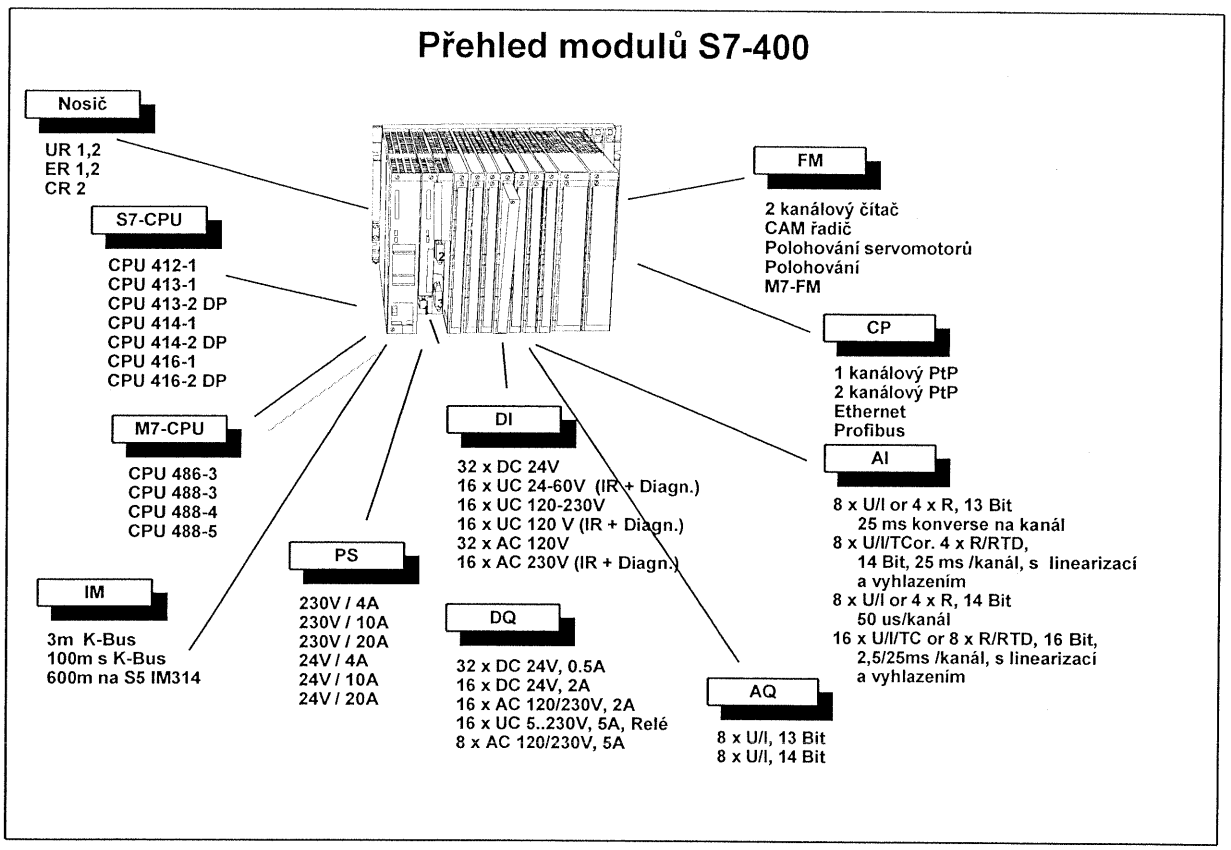
S7 rozšířená komunikace: umožňuje přenos většího objemu dat (až 64KB) nezávisle na typu použité sítě.

Příslušné SFB bloky jsou integrovány do operačního systému CPU S7-400. Tato komunikace vyžaduje konfigurované spojení.

Spojení lze realizovat přes MPI, PROFIBUS, Industrial Ethernet a PtP.

**Sdílená data (GD)**

Sdílená data umožňují cyklickou výměnu malého objemu dat mezi více stanicemi (max. 15 CPU) pomocí GD komunikace.




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.6

Školicí středisko
firmy E&A spol. s r.o.

**Nosiče modulů**

Pro S7-400 jsou dostupné následující nosiče modulů:

UR1/UR2 je navržen jako univerzální nosič použitelný jako centrální i jako rozšiřující nosič. Obsahuje 18/9 jednoduchých pozic s P a K sběrnicí.

ER1/ER2 je navržen jako rozšiřující nosič bez K sběrnice.

Pro asymetrické víceprocesorové systémy je určen segmentový nosič CR2.

**S7-CPU**:

Všechny S7-400 CPU jsou zpětně kompatibilní pro všechny STEP 7 uživatelské programy. Jsou dostupné ve dvou verzích: jednoduché šířky a dvojité šířky s DP-Master rozhraním.

Přes integrované DP rozhraní může být připojeno až 64 DP-Slave stanic adresovaných přes DP rozhraní. Maximální rychlost je 12 MBaud.

**FM moduly**

Spektrum S5-IP modulů pro polohování, řízení a čítání je u S7 nahrazeno FM moduly. Navíc je dostupný M7-FM jako volně C-programovatelný funkční modul pro řízení procesu.

**IM**

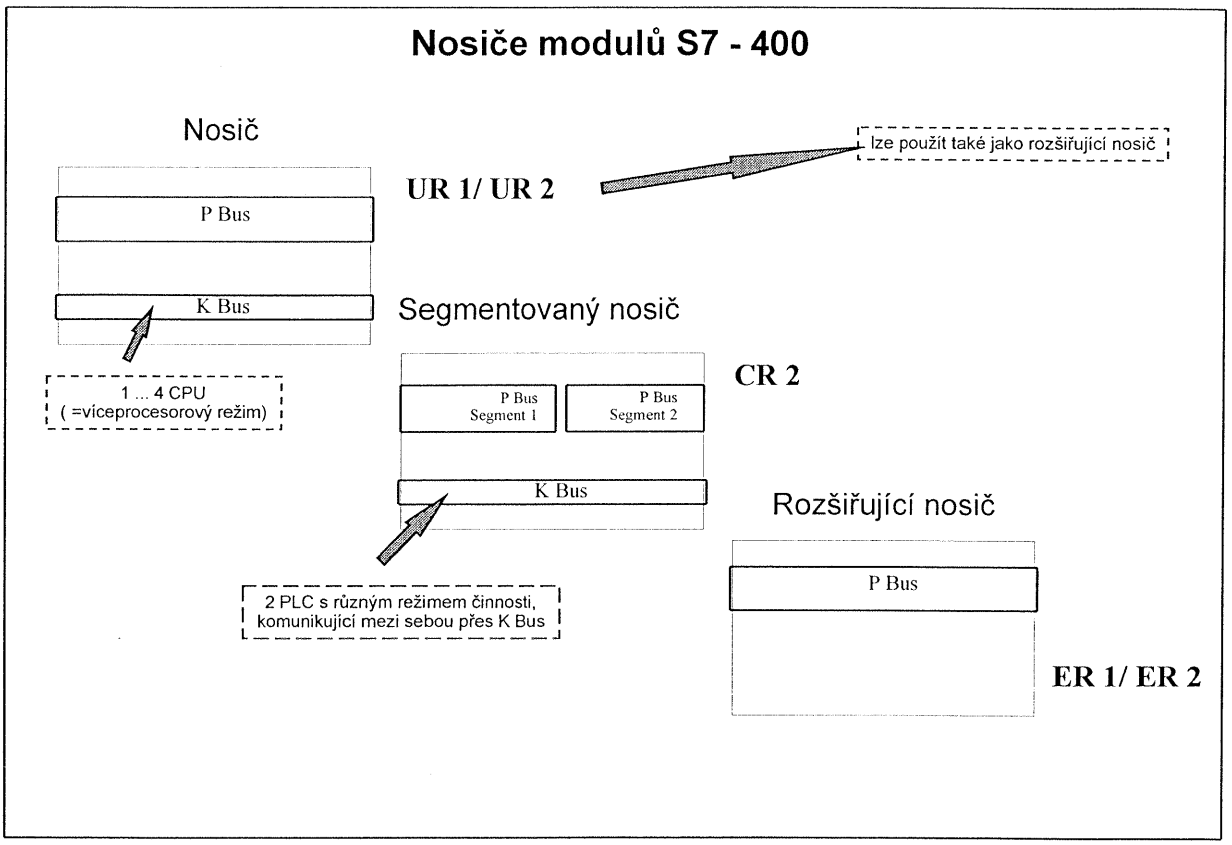
IM moduly umožňují připojení SIMATIC S7 a SIMATIC S5 rozšiřujících nosičů k centrálnímu nosiči S7-400.

**CP**

CP moduly umožňují připojení CPU do následujících sítí:

- Industrial Ethernet (CP 443-1)
- PROFIBUS (CP 443-5)
- Point-to-Point (CP441-1 a CP441-2)

Navíc všechna CPU umožňují připojení CPU pomocí MPI rozhraní do MPI sítě. Maximálně lze do MPI sítě připojit 32 stanic.








**UR 1 / UR 2**

UR1/UR2 lze použít jako centrální nosič modulů i jako rozšiřující nosič modulů. Tento nosič je vybaven vysokorychlostní sběrnicí Peripheral Bus (P-Bus, 1.5 microsec./ Byte) pro rychlou výměnu I/O signálů a časově kritický přístup k signálním modulům.

Na doplnění, UR1 (18 pozic) / UR2 (9 pozic) mají výkonnou sériovou sběrnici Communication Bus (K-Bus) pro rychlou výměnu dat (10.5 MBaud) mezi K-Bus stanicemi (S7/M7-CPU, FM, CP ).

Jednotlivé sběrnice (P-bus a K-bus) jsou navzájem oddělené.

**CR2**

Segmentovaný nosič CR 2 má I/O sběrnici rozdělenou na dvě části - 10 a 8 pozic. CPU lze umístit do každého segmentu. Každé CPU zastává funkci master pro daný segment P-Busu a má přístup pouze k SM modulům v tomto segmentu.

Režim činnosti obou CPU není synchronizován, tj. každé CPU může být v jiném režimu činnosti. Komunikace mezi oběma CPU probíhá přes nedělenou komunikační sběrnici K-Bus.

**Proč CR2?**

U synchronního víceprocesorového režimu mají všechny zúčastněné CPU (max. 4) stejný režim činnosti (např. STOP).

**ER 1 / ER 2**

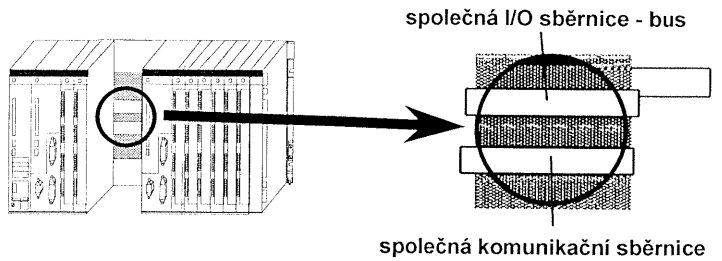
ER1 (18 pozic)/ER2 (9 pozic) nemá K-Bus, přerušovací - alarmové vedení, 24 V napájení pro moduly ani zálohovací baterii.

**Pravidla pozic**

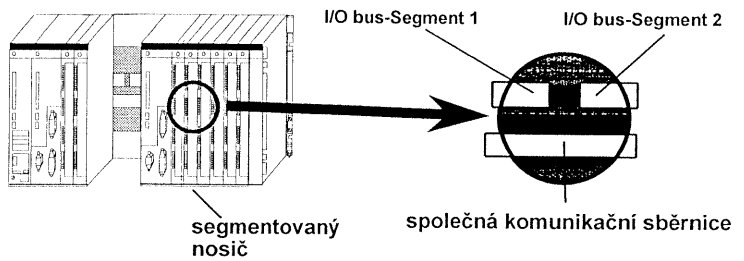
Pro umístění modulů do jednotlivých pozic nosiče neplatí žádná pravidla. *Vyjímka: pouze napájecí zdroj je v pozici úplně vlevo a přijímač IM v rozšiřujícím nosiči je úplně vpravo.*

## Symetrický a asymetrický víceprocesorový režim

- **Symetrický víceprocesorový režim**




- **Asymetrický víceprocesorový režim**




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_11cz.8

Školicí středisko firmy E&A spol. s r.o.

**Víceprocesorový režim**

Víceprocesorový režim umožňuje postupné zvyšování výkonu celého řídícího systému. Vložením dalšího CPU lze zvýšit velikost paměti, počet čítačů, časovačů, atd.). Zpracování složitého algoritmu lze rozdělit na několik částí, z nichž každá bude zpracovávána jiným CPU, které mezi sebou budou komunikovat.

**Symetrický**

V tomto režimu pracují všechny CPU (až 4) na společné P a K sběrnici. V rámci toho existuje pouze jeden adresní prostor, ve kterém se zobrazují signální moduly.

Každý modul založený do UR1 nebo UR2 musí být samozřejmě přiřazen během konfigurace konkrétnímu CPU. CPU, které je ve funkci "Master" pro tento modul může :

- přijímat přerušení od modulu
- parametrizovat modul
- přistupovat k modulu pomocí L PBxx, T Wxx, atd.

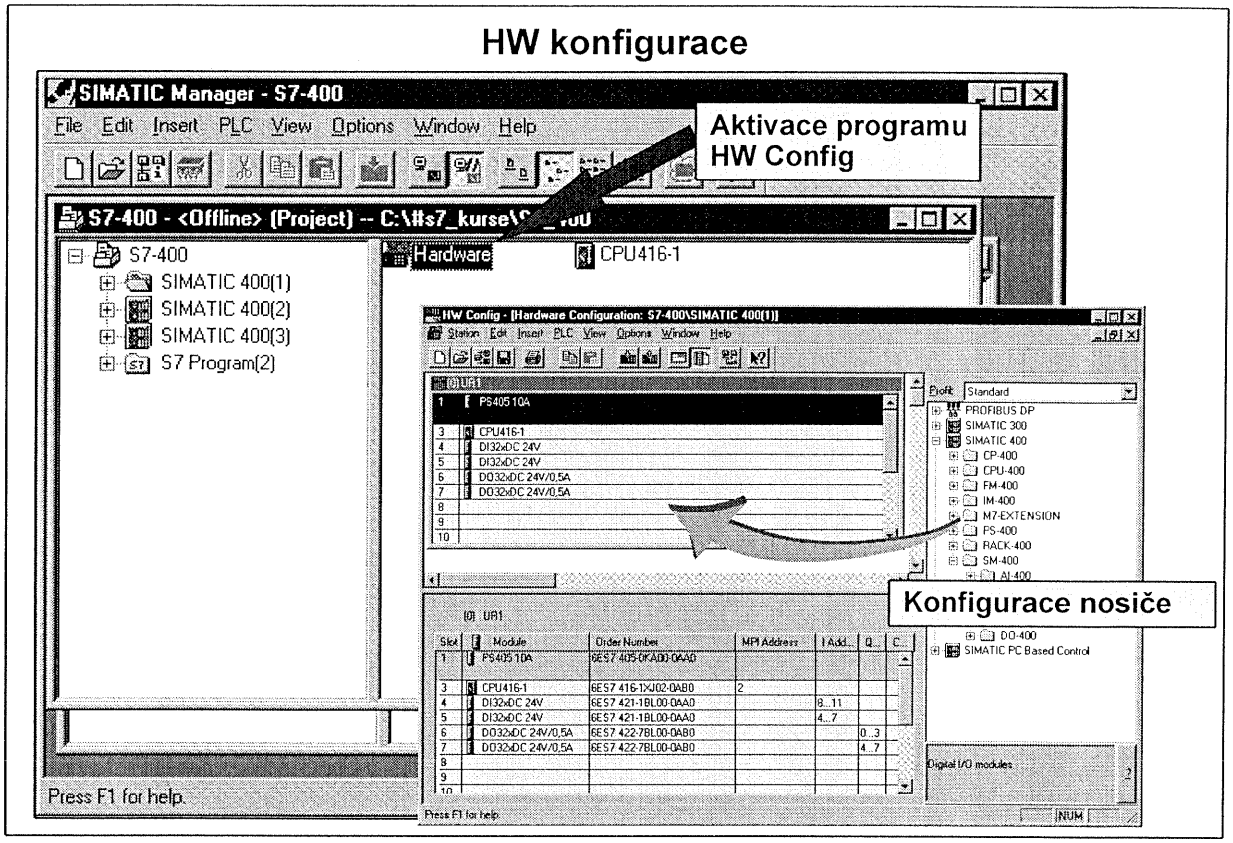
Režim činnosti všech CPU je synchronizován, tzn. všechny CPU jsou ve stejném režimu. Z vnějšího pohledu se celá stanice chová jako jeden velký řídící systém.

**Asymetrický**

Asymetrický režim je možný pouze s nosičem CR2. Tento nosič má P-sběrnici rozdělenou na dvě samostatné části.

Do každé části se instaluje jeden CPU. Oba CPU moduly pracují nezávisle na druhém, každý CPU modul má samostatný I/O adresní prostor. Společná K-sběrnice umožňuje vzájemnou komunikaci CPU bez dalšího doplňkového HW. Z vnějšího pohledu se jedná o dva samostatné řídící systémy,které spolu komunikují po sběrnici. Výhody:

- úspora místa v rozvaděči
- finanční úspora, stačí pouze jeden nosič a jeden napájecí zdroj.







Datum: 24.11.2002
Soubor: PRO2_11cz.9



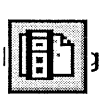
**Konfigurace**

Centralizovaná konfigurace využívá definování modulů do nosiče s CPU a následně definování modulů do rozšiřujících nosičů.

**Vytvoření konfigurace**

Pro otevření konfigurace stanice je nutné provést následující kroky:

1. vybrat myší požadovanou HW stanici
2. aktivací *Edit -> Object* nebo dvojklikem v pravém okně ( na objektu *Hardware)* otevřít okno HW konfigurace.
3 z otevřeného hardwarového katalogu metodou *drag&drop* vybrat požadovaný nosič modulů a jednotlivé moduly.



Kliknutím na symbol "+" otevírá jednotlivé subsložky a zobrazuje moduly dané skupiny. Po označení požadovaného modulu jsou ve spodní části katalogu zobrazeny detailní informace. Kliknutí na symbol "-" ukončí zobrazení subsložek a modulů.

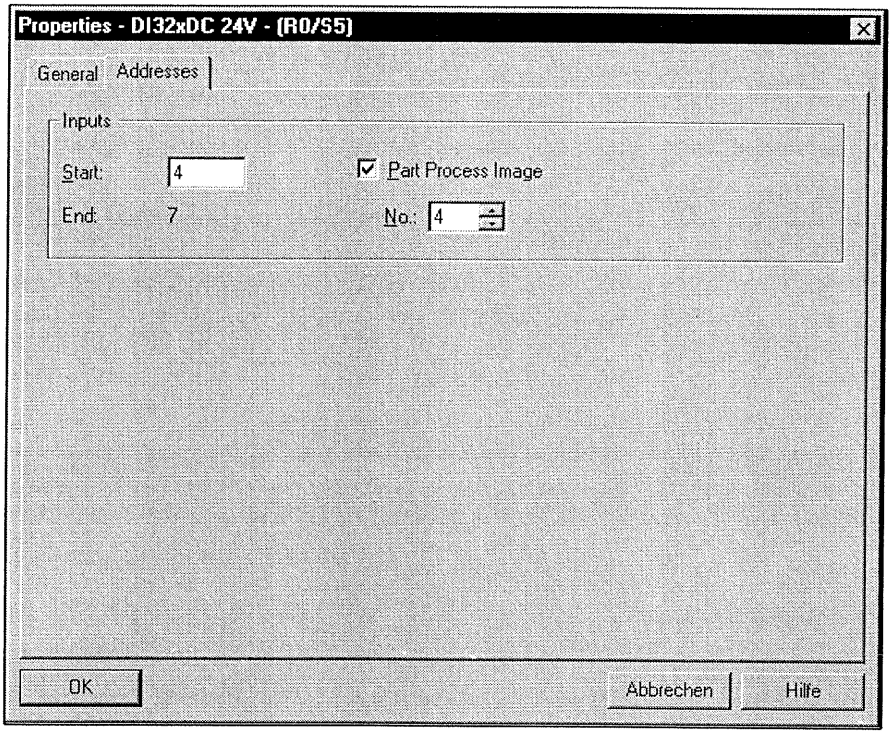
**Výběr nosiče**

Jestliže je otevřeno okno stanice a HW katalogu lze

1. vybrat typ stanice (S7-300/S7-400).
2. po určení typu stanice otevřít složku RACK a umístit požadovaný nosič modulů do levé části okna stanice. Po umístění nosiče bude v této části okna zobrazena prázdná tabulka.

   Nosič je prezentován jako prázdná konfigurační tabulka do které se umisťují na požadované pozice jednotlivé moduly. Každý řádek tabulky odpovídá jedné pozici v nosiči modulů.
3. metodou *drag&drop* lze nyní umístit jednotlivé moduly do příslušných pozic v prázdné konfigurační tabulce.

# Parametry modulů: logická adresa, obraz procesu







Datum: 24.11.2002
Soubor: PRO2_11cz.10



**Obecně**

U systému S7 -400 je k dispozici standardní adresace I/O modulů. Tento způsob adresace je aktivní, dokud do CPU nejsou nahrány systémové konfigurační datové bloky.

**Standardní adresa**

Standardní - default adresa je závislá na:

- čísle nosiče modulů. Toto číslo je definováno na přijímači IM (1..21), číslo centrálního nosiče je vždy 0
- pozici modulu v nosiči. Standardní adresa se určuje podle vztahu:

Digital start adresa = [(číslo nosiče) x 18 + pozice -1] x 4

Analog start adresa= [(číslo nosiče) x 18 + pozice -1] x 64 + 512

**Část obrazu procesu**

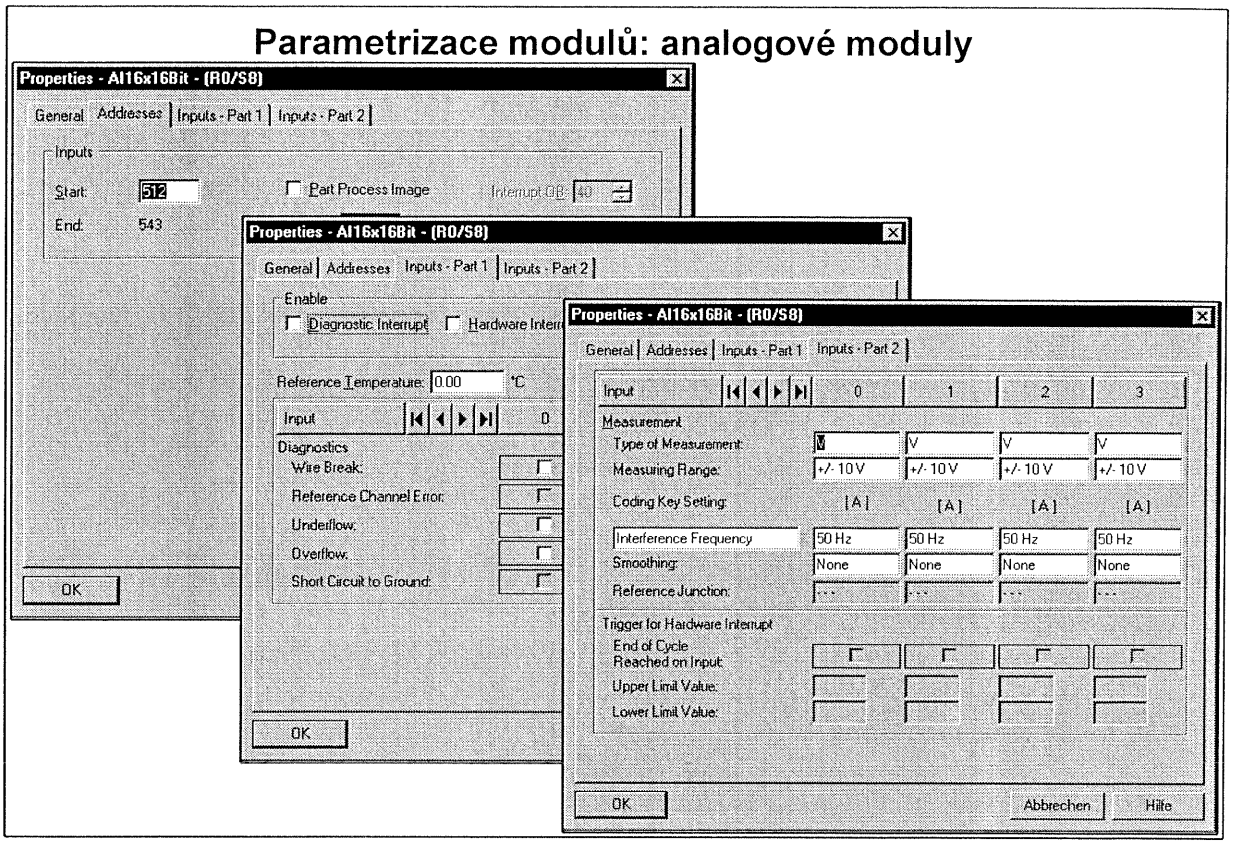
Vedle celkového obrazu procesu lze definovat parametry až pro 8 částí obrazu procesu. Tyto části mohou být aktualizovány z uživatelského programu pomocí systémových funkcí SFC 26/27.

Touto cestou má uživatel možnost používat obraz procesu, který je určen pro prioritní třídu OB1, také v jiných prioritních třídách např. OB35.

OB35 potom obsahuje následující sekvenci:

1. načtení aktuálních hodnot do příslušné části obrazu vstupů procesu pomocí SFC26.
2. volání řídícího algoritmu. Algoritmus zapíše nové požadované hodnoty do příslušné části obrazu výstupů procesu.
3. zápis části obrazu výstupů procesu do periferií pomocí SFC27.

# Parametrizace modulů: analogové moduly




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.11

Školicí středisko
firmy E&A spol. s r.o.

**Parametrizace modulů**

Všechny parametrizovatelné moduly lze parametrizovat pomocí programu HWConfig.

Analogové moduly obsahují několik registrů pro parametrizaci:

**Addresses**

- počáteční adresa modulu
- číslo části obrazu procesu
- OB hardwarového přerušení

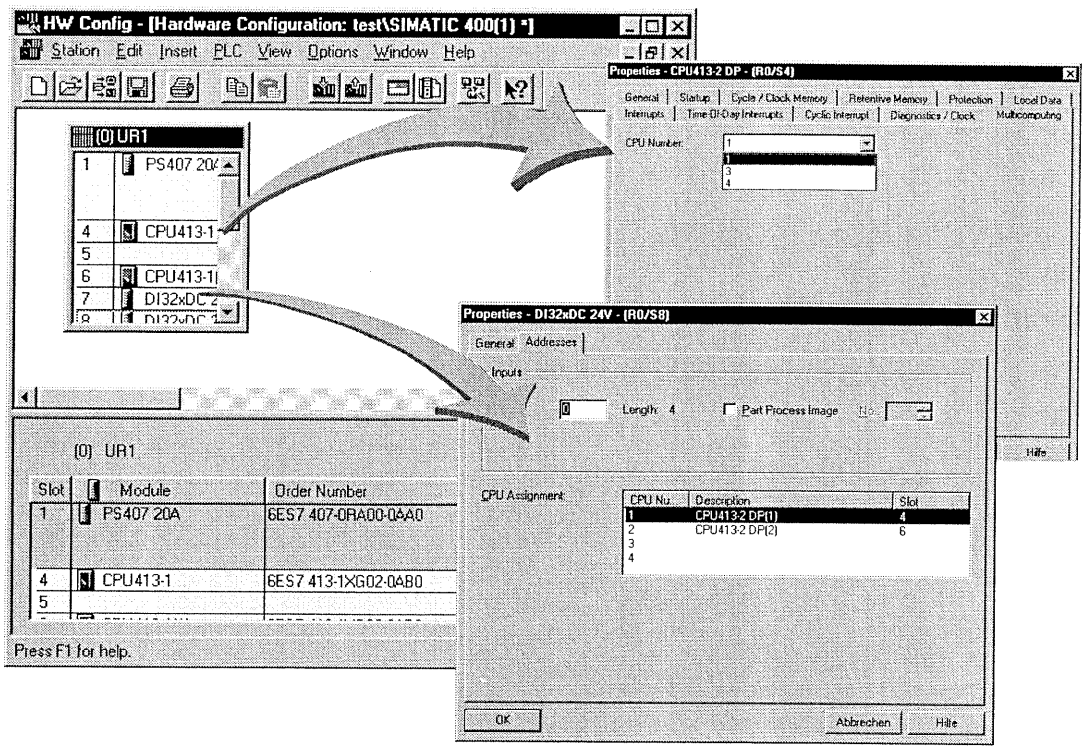
**Inputs - Part 1**

- uvolnění hardwarového a diagnostického přerušení
- uvolnění sledování diagnostiky
  - kontrola přerušení vedení
  - chyba srovnávacího kanálu
  - podtečení
  - přetečení
  - zkratování

**Inputs - Part 2**

- typ měření
- měřící rozsah
- potlačení interferencí
- vyhlazení
- konec cyklu převodu
- dolní a horní hranice pro HW přerušení

# Konfigurace víceprocesorového režimu




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.12

Školicí středisko
firmy E&A spol. s r.o.

**Přehled**

Víceprocesorový režim umožňuje synchronní práci většího počtu CPU (2 až 4) v centrálním nosiči modulů S7-400.

Všechna CPU mají společný režim činnosti - společně provádějí náběh, zpracování cyklu i přechod do STOP stavu.

**Nastavení**

Víceprocesorový režim nastává v okamžiku, kdy je do vhodného nosiče založeno několik CPU. Zda CPU umožňuje víceprocesorový režim je uvedeno v informativním textu v HW katalogu.

Jednotlivá CPU se "dělí" o společný adresní prostor, tzn. že adresní prostor modulu je vždy exklusivně přidělen konkrétnímu CPU.

**Postup**

Konfigurace víceprocesorového režimu probíhá v následujících krocích:

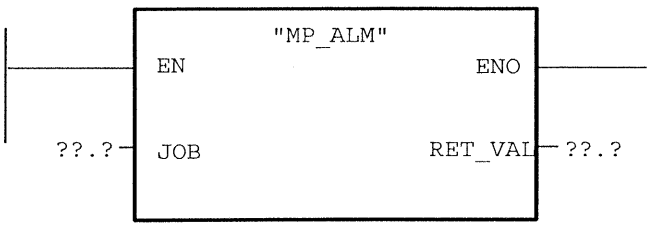
1. umístit všechny CPU moduly do nosiče.
2. dvojklikem na CPU otevřít okno pro nastavení čísla CPU ve víceprocesorovém režimu.
3. postupně přiřadit moduly jednotlivým CPU:
   - založit modul do nosiče
   - dvojklikem na modulu otevřít kartu "Addresses".
   - v poli "CPU No." definovat požadované CPU.

U modulů s přerušením je přiřazené CPU zobrazeno jako cílové CPU v kartě vstupů ("Inputs") nebo výstupů ("Outputs").

Nabídka *View -> Filter -> CPUx Modules* umožňuje zobrazit přiřazené moduly ve formě tabulky pro jednotlivá CPU.

Konfiguraci stanice lze do nahrát pouze jako celek do všech CPU najednou. Nahrání nedokončené konfigurace není možné.

# SFC 35 pro synchronizaci víceprocesorového režimu




| Parametr | Deklarace | Datový typ | Oblast | Popis |
|---|---|---|---|---|
| JOB | INPUT | BYTE | I, Q, M, D, L, Const. | identifikátor úlohy<br>rozsah: 1 až 15 |
| RET_VAL | OUTPUT | INT | I, Q, M, D, L | Pokud je při zpracování detekována chyba, je zde uveden chybový kód:<br>W#16#0000: bez chyby.<br>W#16#8090: parametr JOB neobsahuje platné číslo úlohy.<br>W#16#80A0: na tomto nebo jiném CPU není dokončen OB 60<br>W#16#80A1: chybný režim činnosti (START-UP místo RUN) |

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.13

Školicí středisko
firmy E&A spol. s r.o.

**Popis**

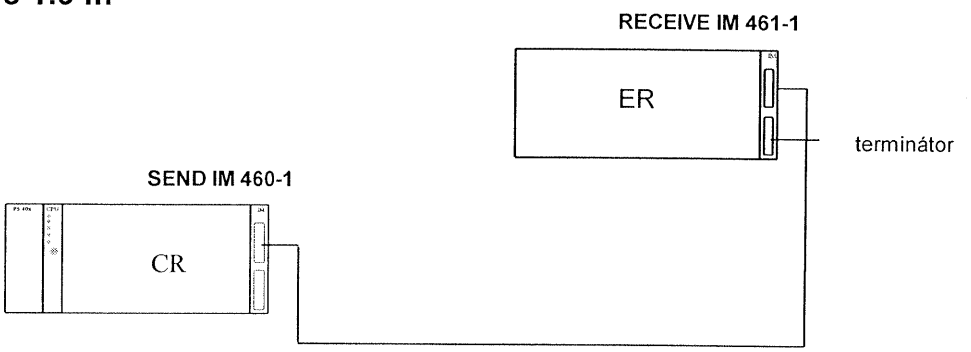
Vyvolání SFC 35 "MP_ALM" spouští víceprocesorové přerušení ve víceprocesorovém režimu. To umožňuje synchronizovaný start OB 60 na všech procesorech. V jednoprocesorovém režimu nebo v asynchronním víceprocesorovém režimu je OB60 spuštěn pouze na CPU, na kterém byl volán SFC35.

Vstupní parametr JOB umožňuje identifikovat příčinu požadovaného víceprocesorového přerušení. Tento identifikátor je odeslán na všechny CPU a lze ho v OB60 vyhodnotit.

SFC 35 "MP_ALM" lze volat z kteréhokoliv místa v uživatelském programu. Protože volání SFC35 má smysl pouze v RUN režimu, je v režimu náběhu víceprocesorové přerušení potlačeno.

## Centrální rozšiřování 1

- Je přenášeno pouze napájení a P bus
- rozšíření pouze o 1 ER
- maximální vzdálenost mezi CR a ER je 1.5 m




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_11cz.14

Školicí středisko
firmy E&A spol. s r.o.

**Centralizovaná konfigurace 1**

Propojení centrálního a rozšiřujícího nosiče lze realizovat dvojicí modulů rozhraní IM 460-1/IM 461-1, které se zapojí do nosičů UR1, UR2 a CR2. Maximální vzdálenost propojení je 1.5 m. Toto propojení poskytuje následující možnosti:

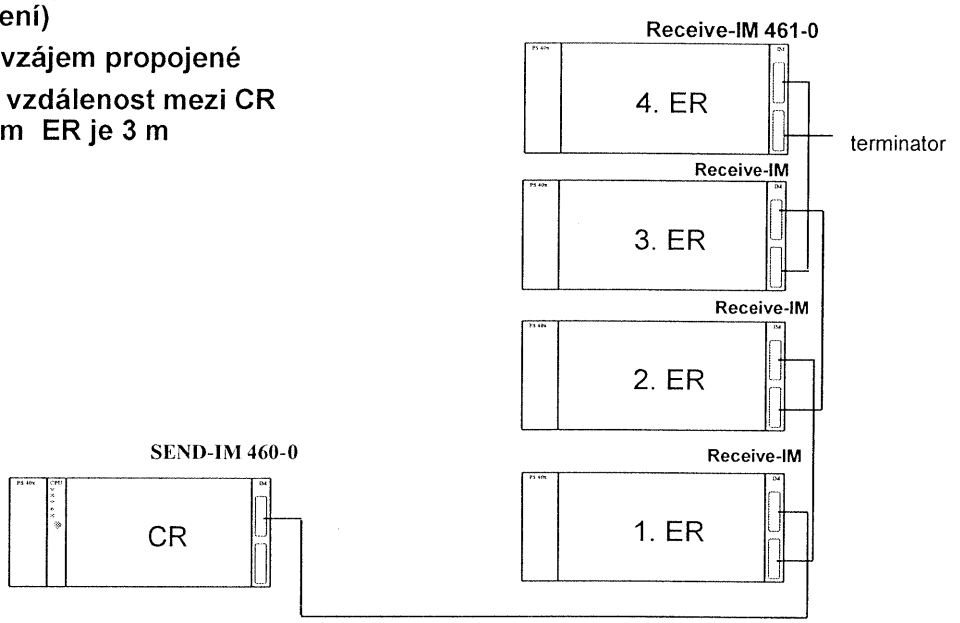
- připojení maximálně 2 rozšiřujících nosičů (1/rozhraní).
- maximální vzdálenost centrálního a rozšiřujícího nosiče 1.5 m.
- maximálně 2 vysílače SEND-IM 460-1 v jednom centrálním nosiči.
- modul Send-IM 460-1 propojuje do rozšiřujícího nosiče pouze P bus (ne K bus). Moduly v rozšiřujícím nosiči jsou napájeny 5 V (max. 5 A/pozice) přes propojovací kabel. Není nutný žádný další napájecí zdroj v rozšiřujícím nosiči.
- nepoužité rozhraní v Send-IM 460-1 nemusí být ukončeno, nepoužité rozhraní v Receive-IM 461-1 musí být zakončeno ukončovacím odporem - terminátorem.
- na modulu přijímače (Receive-IM 461-1) musí být pomocí přepínače nastaveno číslo rozšiřujícího nosiče pro identifikaci v systému.

**Poznámka**

Do jednoho centrálního nosiče lze připojit maximálně 21 rozšiřujících nosičů. Přijímač IM musí být v rozšiřujícím nosiči vždy v pozici úplně vpravo.

# Centrální rozšiřování 2

- jsou přenášeny sběrnice P a K (bez napájení)
- až 4 ER navzájem propojené
- maximální vzdálenost mezi CR a posledním ER je 3 m




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.15

Školicí středisko
firmy E&A spol. s r.o.

**Centralizovaná konfigurace 2**

Moduly rozhraní IM 460-0/IM 461-0 umožňují propojení centrálního a rozšiřujícího nosiče až na vzdálenost 3 m. Vlastnosti tohoto propojení jsou následující:

- připojení maximálně 8 rozšiřujících nosičů (4 na jedno rozhraní).
- maximální vzdálenost mezi centrálním a posledním nosičem 3 m.
- maximálně 6 ks vysílače Send-IM 460-0 v jednom centrálním nosiči
- Send-IM 460-0 rozhraní propojuje P bus a K bus na rozšiřující nosič. V rozšiřujícím nosiči **musí** být napájecí zdroj. Moduly nejsou napájeny přes propojovací kabel.
- nepoužité rozhraní Send-IM 460-0 nemusí být ukončeno, nepoužité rozhraní Receive-IM 461-0 musí být ukončeno terminátorem.
- na přijímači Receive-IM 461-0 musí být pomocí přepínače nastaveno číslo rozšiřujícího nosiče pro identifikaci v systému.

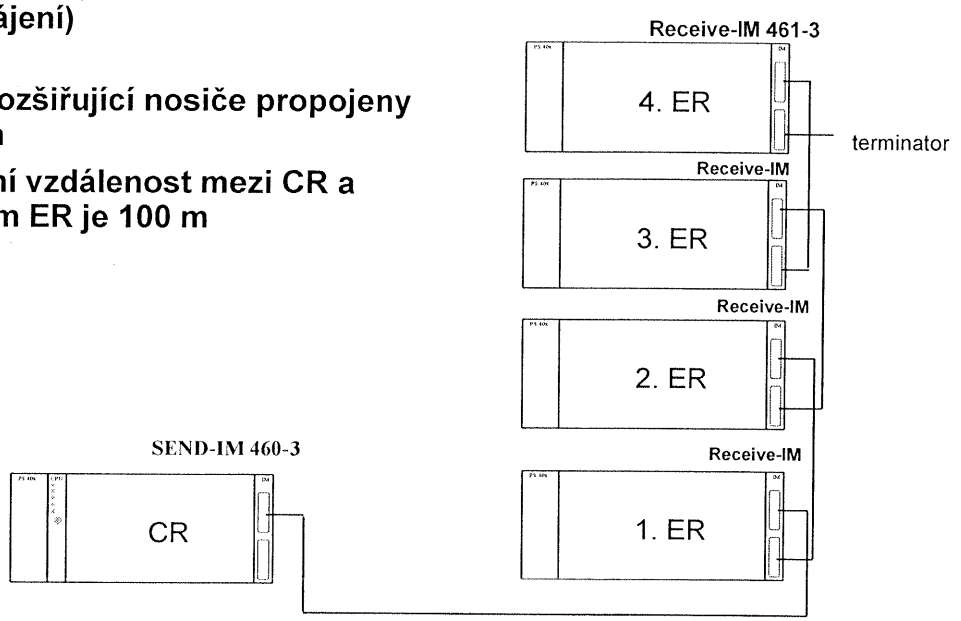
**Poznámka**

Do jednoho centrálního nosiče lze připojit maximálně 21 rozšiřujících nosičů.

K bus je propojen pouze na prvních 6 rozšiřujících nosičů, tzn. inteligentní moduly (CP, FM) lze umístit pouze do centrálního a prvních 6ti rozšiřujících nosičů.
Přijímač IM musí být v rozšiřujícím nosiči vždy v pozici úplně vpravo.

## Distribuované rozšiřování

- jsou přenášeny sběrnice P a K (bez napájení)
- až 4 ER rozšiřující nosiče propojeny navzájem
- maximální vzdálenost mezi CR a posledním ER je 100 m




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_11cz.16

Školicí středisko
firmy E&A spol. s r.o.

**Distribuovaná konfigurace**

Moduly rozhraní IM 460-3/IM 461-3 umožňují propojení centrálního nosiče s rozšiřujícími nosiči až na vzdálenost 100 m s následujícími vlastnostmi:

- lze připojit maximálně 8 rozšiřujících nosičů (4 na jedno rozhraní).
- maximální vzdálenost mezi centrálou a posledním rozšiřujícím nosičem je 100 m.
- maximálně 6 Send-IM 460-3 v jednom centrálním nosiči
- Send-IM 460-3 propojuje sběrnice P a K. V rozšiřujícím nosiči musí být napájecí zdroj
- nepoužité rozhraní Send-IM 460-3 nemusí být ukončeno, nepoužité rozhraní Receive-IM 461-3 musí být ukončeno terminátorem.
- na rozšiřujícím nosiči Receive-IM 461-3 musí být pomocí přepínače nastaveno číslo rozšiřujícího nosiče pro identifikaci v systému.

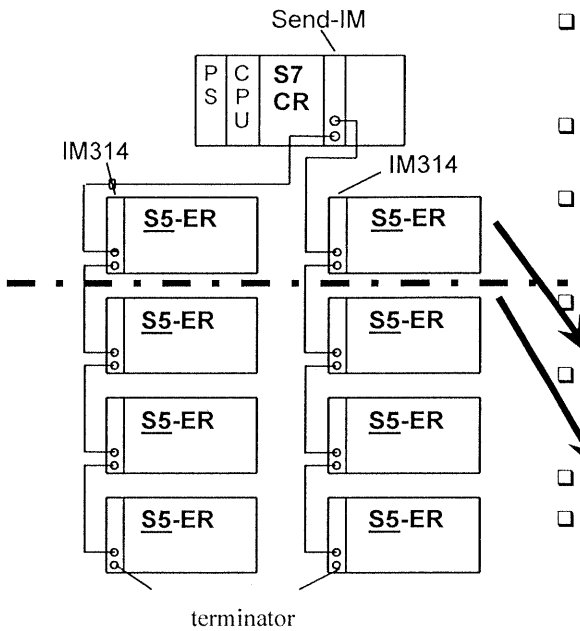
**Poznámka**

Do centrálního nosiče lze připojit maximálně 21 rozšiřujících nosičů.

K bus je propojena pouze na prvních 6 rozšiřujících nosičů, tzn. inteligentní moduly (CP,FM) lze zapojit pouze do centrálního a prvních 6ti rozšiřujících nosičů.

Receive-IM musí být v rozšiřujícím nosiči vždy v pozici úplně vpravo.

## Distribuované spojení S7 a S5




- lze připojit až 4 ks S5 rozšiřujících nosičů za sebou
- až 4 Send-IM v centrálním nosiči
- maximální vzdálenost CR a posledního ER: 600m
- je propojen paralelní S5 bus
- použití S5 rozšiřujících nosičů: ER 183 U, ER 185 U, ER 701-2, ER 701-3
- ostatní S5 ER nosiče
- až 32 S5 ER nosičů na S7-400 CR

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.17

Školicí středisko
firmy E&A spol. s r.o.

**Distribuovaná konfigurace pomocí S5-ER**

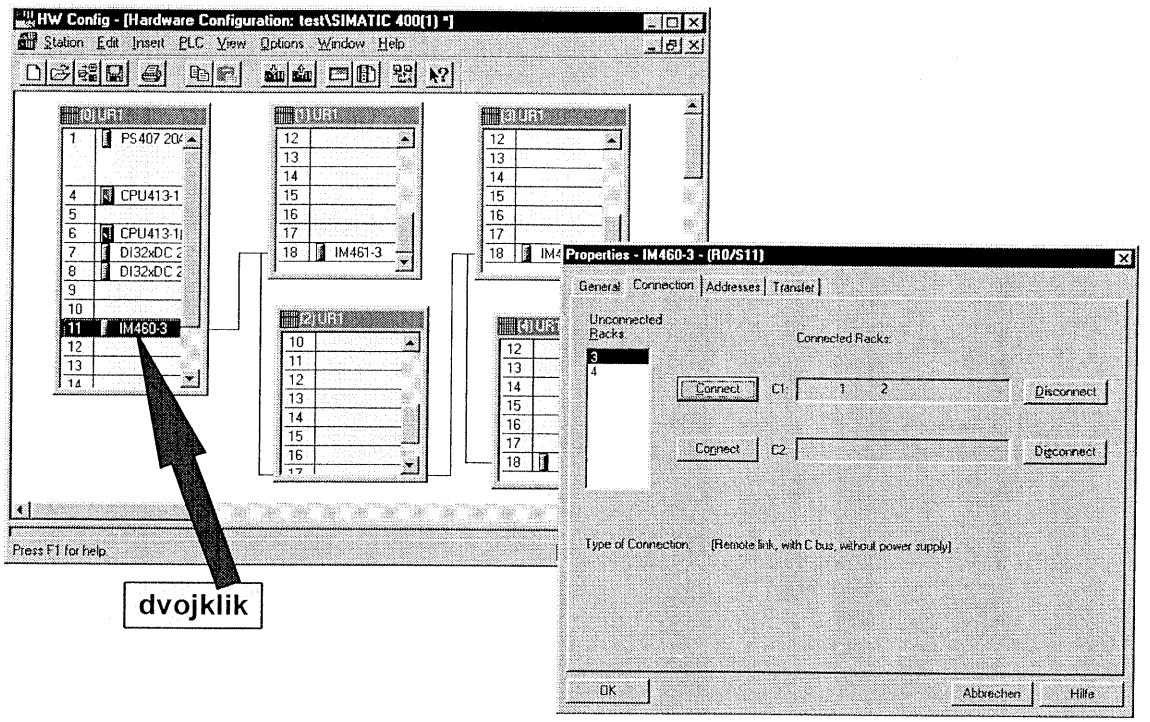
Použití IM463-2 (Send-IM) v centrálním nosiči S7-400 (UR1/UR2/CR2) a IM314 (Receive-IM v S5 ER) v rozšiřujícím nosiči S5 (ER 183U/185U, 701-2, 701-3) umožňuje připojení rozšiřujících nosičů řady S5 bez 5V napájení a K sběrnice.

V centrálním nosiči lze umístit až 4 IM463-2, propojení má tyto vlastnosti:

- max. 4 ER spojené za sebou.
- max. vzdálenost CR a posledního ER je 600 m
- jako první mohou být k S7-400 připojeny ER-183U/ER-185U (pro S5-135U/155U) nebo ER-701-2/ER-701-3 (pro S5-115U).

Za tyto rozšiřující nosiče (ER) lze připojit ostatní typy ER podle možností S5 PLC (viz katalog ST50).

# Rozšiřování centrální konfigurace







Datum: 24.11.2002
Soubor: PRO2_11cz.18



**Rozšiřování konfigurace**

Připojení rozšiřujícího nosiče se provádí v následujících krocích:

1. z HW katalogu vybrat požadovaný typ ER
2. metodou *Drag&Drop* umístit tento nosič do okna stanice (vlevo)
3. umístit jednotlivé moduly do nosiče

   Důležité: před připojením ER k CR musí být do ER umístěn příslušný přijímací modul ( v pozici úplně vpravo).
4. Pouze S7-400 - propojení mezi Send-IM v CR a Receive-IM v ER:
   - dvojklik na objektu Send-IM
   - vybrat kartu "Connection". Je zde uveden seznam nepřipojených rozšiřujících nosičů
   - označit jednotlivé nosiče a pomocí tlačítka "Connect" je propojit na příslušný konektor Send-IM (C1 a C2).

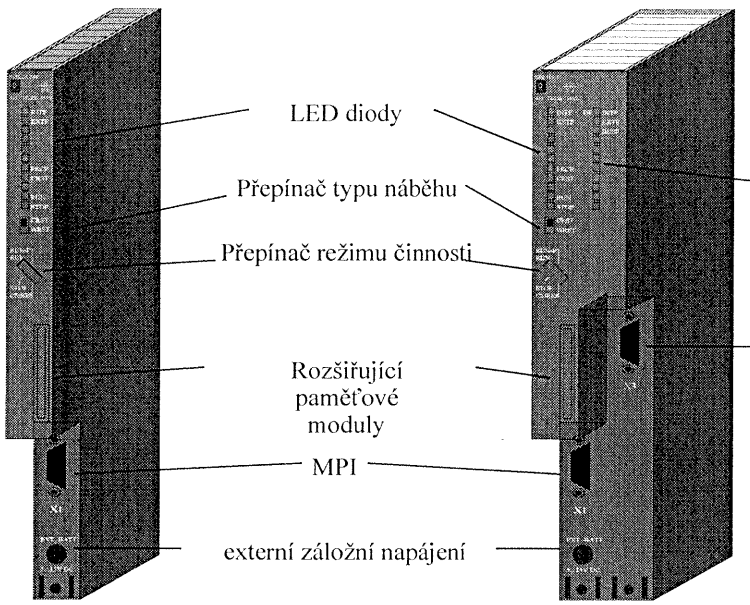
   V okně stanice jsou následně zobrazeny vytvořená propojení.

**Zvláštnosti CR2**

Při použití nosiče CR2 je nutné nejprve vytvořit propojení mezi prázdným ER (pouze s definovaným Receive-IM) a Send-IM modulem příslušného CPU. Teprve potom lze do ER vkládat jednotlivé moduly.

## Moduly CPU

CPU 412-1
CPU 413-1
CPU 414-1
CPU 416-1




CPU 413-2 DP
CPU 414-2 DP
CPU 416-2 DP




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_11cz.19

Školicí středisko
firmy E&A spol. s r.o.

**Ext. záložní napájení** Pomocí tohoto konektoru lze připojit externí záložní baterii (DC 5...15V) na zálohování RAM paměti během např. výpadku nebo výměny napájecího zdroje. Stejný účel má interní záložní baterie.

**MPI rozhraní** MPI rozhraní umožňuje připojení PG/OP pomocí sítě MPI.

**PROFIBUS-DP rozhraní** CPU 413-2/414-2/416-2 jsou vybaveny integrovaným rozhraním PROFIBUS-DP master pro připojení PROFIBUS- DP slave stanic. ( ET200M, ET200U (B/C), S7-300, atd.).

**Paměťové moduly** Jako rozšiřující paměťové moduly lze použít RAM nebo FLASH-EPROM karty.

- RAM karta 64KByte, 256KByte, 1MByte, 2MByte zálohovaná napájecím zdrojem s baterií.
- FLASH-EPROM karta 64KByte, 256KByte, 1MByte, 2MByte, 4MByte, 8MByte, 16MByte

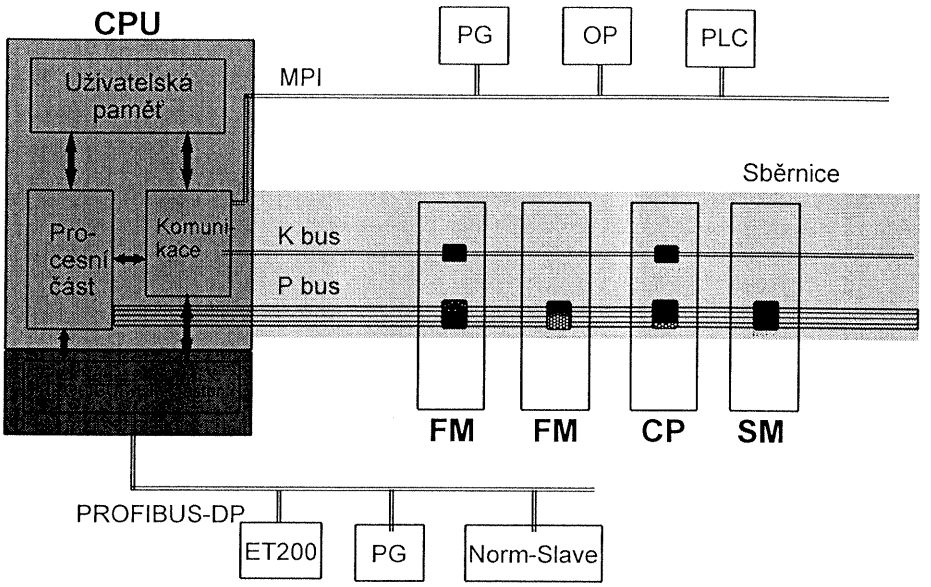
**Režim činnosti**

MRES = vymazání paměti (***M***emory ***RES***et)
STOP = STOP stav, tj. program není zpracováván
RUN = Program je zpracováván, pomocí PG lze v CPU pouze číst
RUN-P= Program je zpracováván, PG umožňuje čtení i zápis do paměti CPU

**Typ náběhu**

CRST= kompletní náběh CPU (Cold Restart) při startování CPU pomocí přepínače
WRST= náběh CPU (Warm Restart) při startování pomocí přepínače

## Systémová architektura




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.20

Školicí středisko
firmy E&A spol. s r.o.

**Konfigurace CPU** CPU S7-400 je funkčně rozděleno na dvě části:

- Procesní část
- Komunikační část

**P-Bus** Procesní část CPU komunikuje se signálními moduly pomocí P-sběrnice (P-Bus). Je optimalizována na rychlou výměnu malého objemu dat (typicky 4 byty).

Vlastnosti P bus (S7-400):

- šířka 8 bitů
- paralelní
- přístupová doba 1.5 µs

**K-Bus** Komunikační sběrnice (K bus) umožňuje komunikaci CPU s inteligentními moduly (např. CP,FM). Je navržena jako logické rozšíření MPI rozhraní pro výměnu velkého objemu dat.

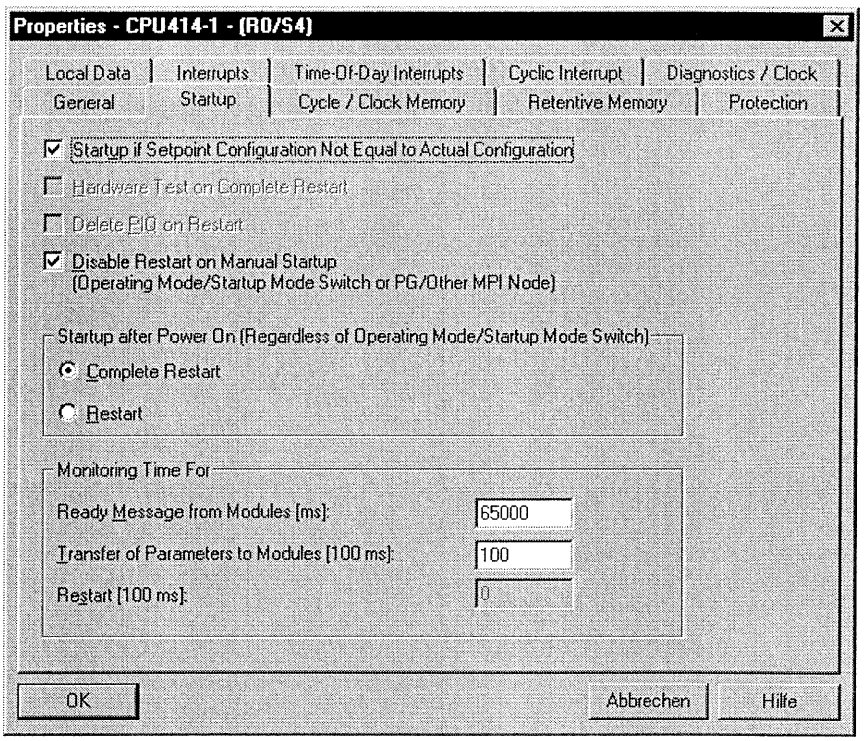
Vlastnosti

- sériová
- rychlost: 10.5 MBits/sec
- max. 127 komunikačních uzlů (teoreticky)

**MPI** Komunikace přes MPI rozhraní má následující vlastnosti:

- sériová
- rychlost: 187.5 KBits/sec
- max. 32 komunikačních uzlů

## Parametry CPU: náběhové vlastnosti








**Setpoint x Actual configuration** Aktivace této volby povoluje provedení náběhu CPU S7-400 i v případě, že aktuální konfigurace detekovaná CPU se liší od konfigurace definované programem *HW Config*.

**Hardware Test** Je-li aktivována tato volba, je při kompletním náběhu kontrolována interní RAM paměť CPU.

**Delete PIQ..** Standardně je při dokončení cyklu CPU (po náběhu) vymazán obraz technologie. Tato volba povoluje/zakazuje tuto vlastnost.

**Disable Restart** Zakazuje kompletní náběh v ručním režimu.

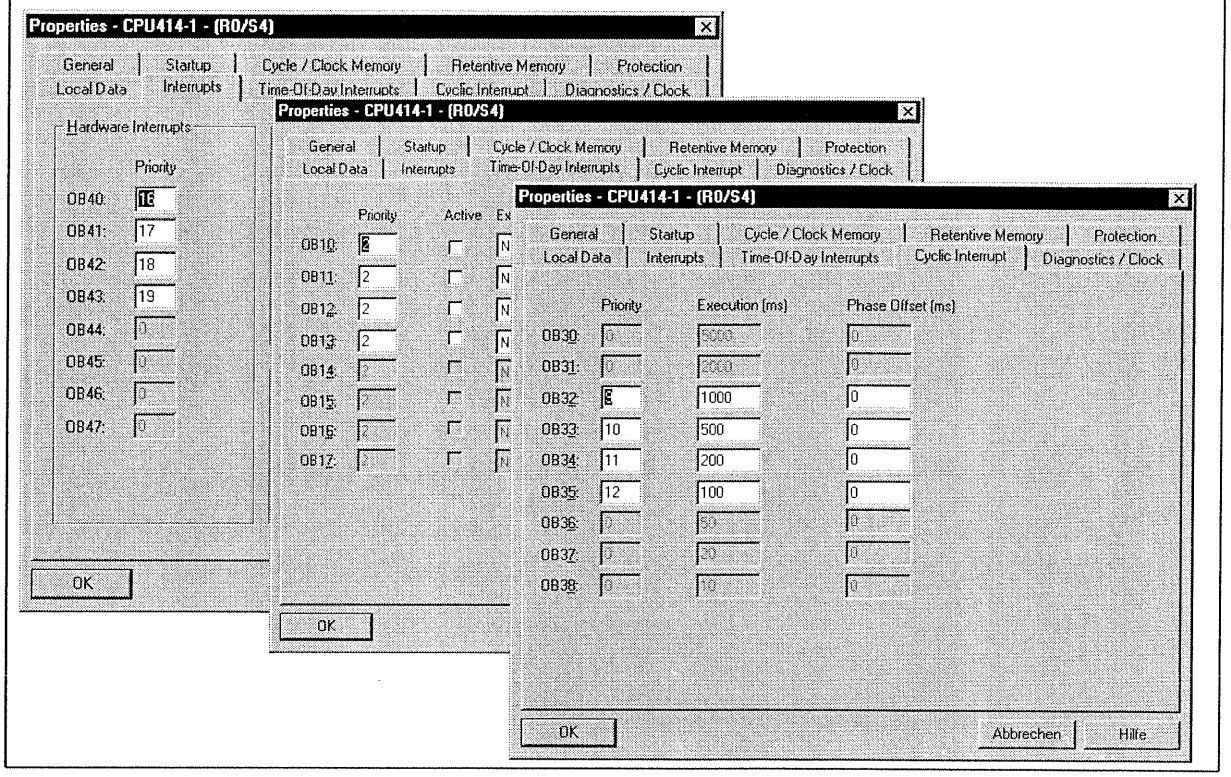
**POWER ON** Při zapnutí napájení může být u S7-400 proveden:

- kompletní náběh (vymazání neremanentních oblastí a zahájení programu od 1. instrukce OB1).
- náběh (všechny paměťové oblasti jsou zachovány a zpracování programu pokračuje od místa přerušení).

**Monitoring Times** U S7-400 lze definovat následující časy:

- maximální čekací dobu pro ohlášení se všech modulů CPU
- maximální čas, během kterého musí modul potvrdit převzetí parametrizačních dat
- maximální čas po který může být přerušeno napájení a po jeho obnovení dojde pouze k náběhu CPU

# Parametry CPU: přerušení







Datum: 24.11.2002
Soubor: PRO2_11cz.22



**Priority**

S7-400 umožňuje definovat vlastní priority zpracování bloků přerušení stanovit tak pořadí zpracování těchto bloků při současném výskytu několika přerušení.

Vyšší číslo znamená vyšší prioritu.

***Time-of-Day* přerušení**

Toto přerušení umožňuje nastavit kdy (rok, měsíc, den, hodina, minuta, vteřina) a jak často (jednou, každou minutu, hodinu, den, měsíc, rok) se provede příslušný OB blok a program v něm definovaný.

Lze tak např. automatizovat změnu letního/zimního času na hodinách reálného času v CPU.

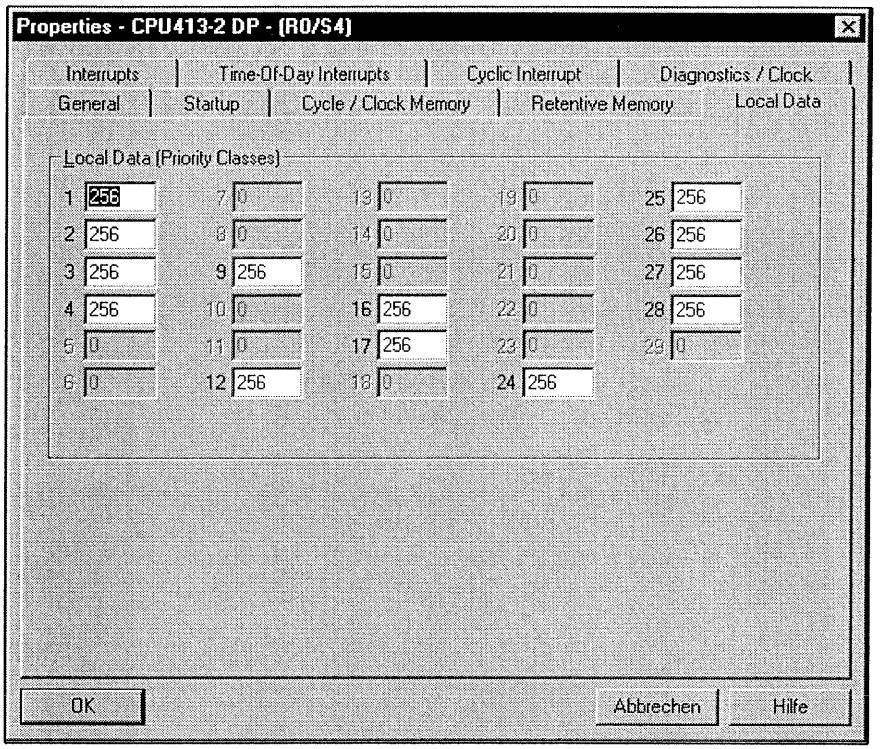
**Cyklické přerušení**

S7 umožňuje zpracovávat část programu v pevně daných časových intervalech. Tuto funkci poskytuje cyklické přerušení. S7-300 může tímto způsobem zpracovávat OB 35 (standardně každých 100 ms).

Časový interval se může pohybovat v rozsahu 1 až 60000 ms.

S7-400 nabízí 8 cyklických přerušení, každé s vlastním časovým intervalem. Protože v daném okamžiku může být zpracováváno pouze jedno cyklické přerušení lze v případě definice několika přerušení definovat také časový posun se kterým budou tato přerušení zpracována( "Phase Offset").

# Parametry CPU: lokální data







Datum: 24.11.2002
Soubor: PRO2_11cz.23



**Lokální data**

Tato položka umožňuje definovat požadavky na velikost oblasti lokálních dat pro každou prioritní třídu(každý OB blok).

Jestliže jsou programové požadavky na lokání paměť vetší než je maximální dostupná velikost lokálních dat, přejde CPU do STOP stavu.

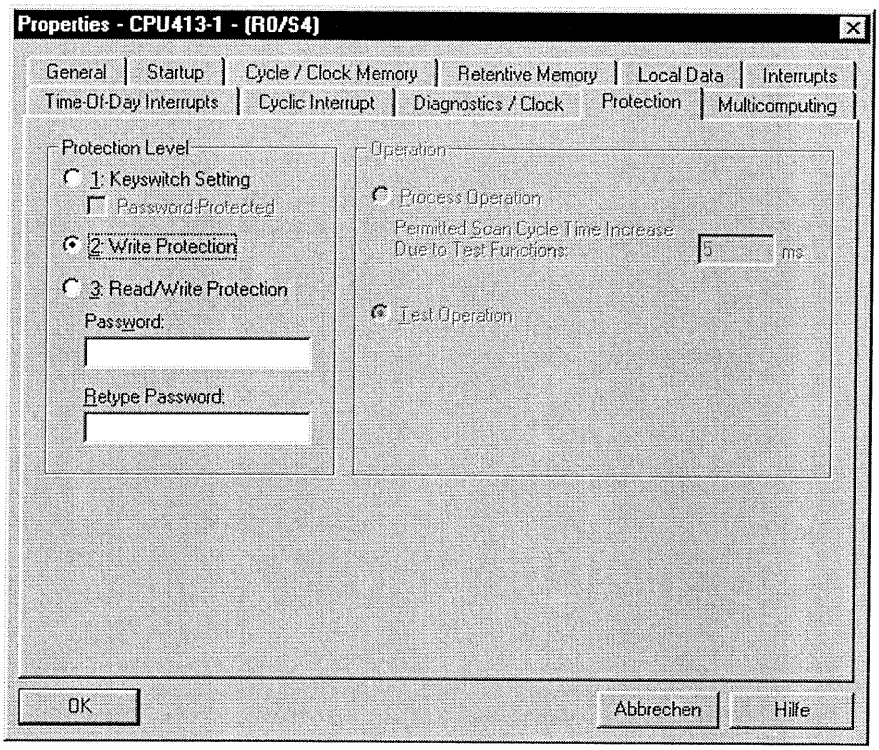
Pokud jsou lokální data adresována symbolicky (tento způsob práce s lokálními proměnnými lze jedině doporučit) přebírá operační systém jejich organizaci a správu.

**Velikost L-Stacku**

Velikost této paměti je závislá na typu CPU:

- CPU 412 - 4 KByte
- CPU 413 - 4 KByte
- CPU 414 - 8 KByte
- CPU 416 - 16 KByte

# Parametry CPU: ochrana přístupu




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.24

Školicí středisko
firmy E&A spol. s r.o.

**Funkce**

Tato funkce poskytuje výběr jedné ze tří možných úrovní ochrany CPU proti neoprávněnému přístupu.

**Vlastnosti**

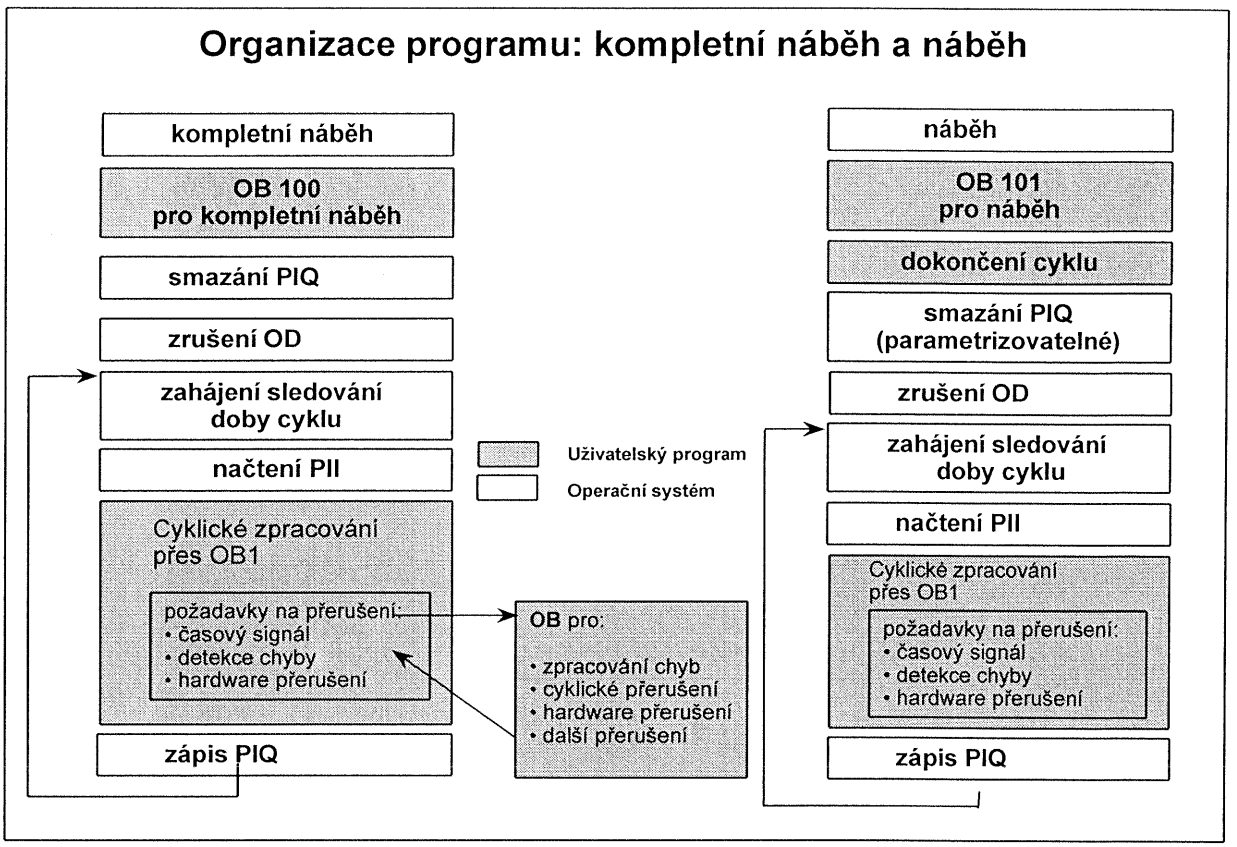
Úroveň 1 (bez hesla): ochrana CPU je definována polohou přepínače režimů činnosti:

- poloha RUN-P nebo STOP: nejsou uplatněna žádná omezení
- poloha RUN: jsou povoleny pouze funkce čtení

**Parametrizované úrovně ochrany**

Je-li aktivována ochranná úroveň s heslem:

- čtení a zápis je umožněn pouze po zadání hesla bez ohledu na polohu přepínače režimů.
- bez zadaní hesla jsou uplatněna následující omezení:
  - úroveň 1 : omezení viz výše.
  - úroveň 2:: je umožněno pouze četní bez ohledu na polohu přepínače režimů činnosti.
  - úroveň 3: bez ohledu na přepínač režimů činnosti je zablokováno jak čtení tak i zápis do paměti CPU.




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_11cz.25

Školicí středisko
firmy E&A spol. s r.o.

**Kompletní náběh (studený start)** Při kompletním náběhu je zpracován OB100, neremanentní paměťové oblasti (M-bitů, čítačů a časovačů) jsou vymazány a zpracování OB1 je zahájeno od první instrukce.

**Náběh (teplý start)** Při náběhu je zpracován OB101, cyklus OB1 je dokončen od místa přerušení s původními hodnotami.

**PII** vstupní tabulka zobrazení procesu (**P**rocess **i**mage **i**nput table)

**PIQ** výstupní tabulka zobrazení procesu (Process image output table)

**OD** zablokování výstupů (**O**utput **D**isable), odpovídá stejnému příkazu u S5.

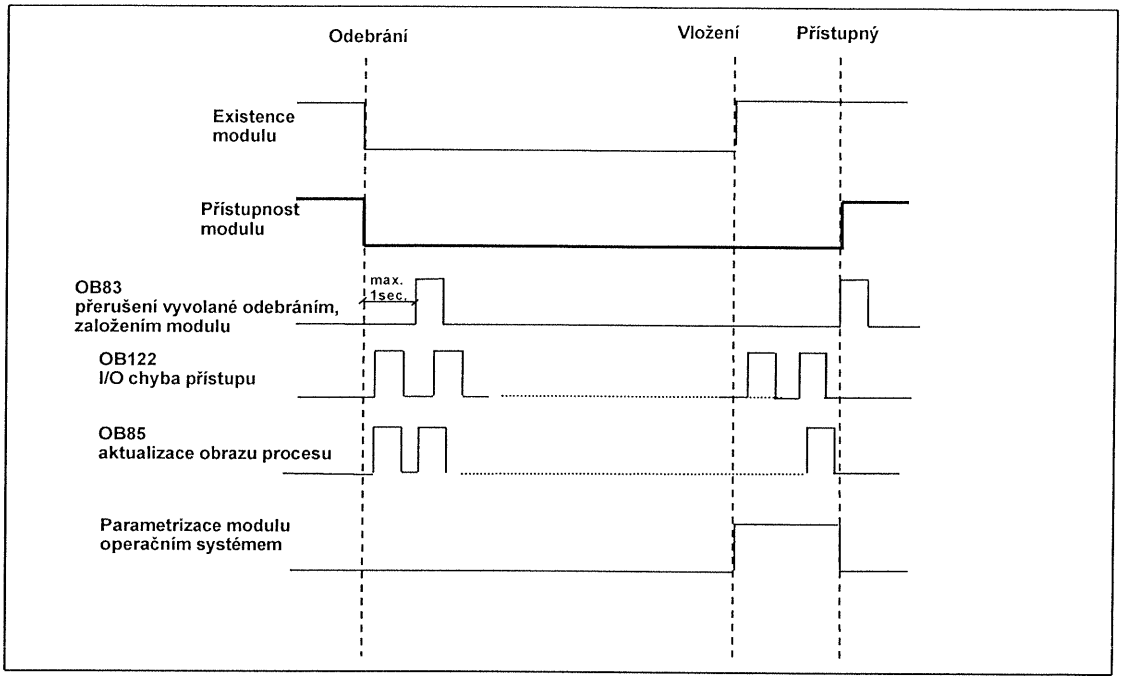
**Aktivace** Organizační bloky jsou spouštěny výhradně operačním systémem. Spuštění některých OB může být zablokováno uživatelem nebo může být změněna jeho priorita.

**Priority** Zpracování každého OB s programem může být přerušeno OB blokem s vyšší prioritou. Priority jsou odstupňovány od 0 do 28, přičemž 0 znamená nejnižší prioritu (OB zablokován) a 28 představuje nejvyšší prioritu.

S7-400 umožňuje volné přiřazení priorit jednotlivým OB. Vyjímkou jsou OB bloky zajišťující základní funkce systému (OB1-cyklické zpracování, OB100 - studený start, OB101-teplý start, OB60 zpracování na pozadí, reakční asynchronní chybové OB). V těchto případech je priorita určována systémem a nemůže být uživatelsky změněna.

OB bloky se stejnou prioritou nemohou být přerušeny. Jsou zpracovávány v pořadí, v jakém byly zachyceny požadavky na jejich zpracování.

# Přerušení od vložení/odebrání modulu do S7 - 400







Datum: 24.11.2002
Soubor: PRO2_11cz.26



**Přerušení vložením, odebráním - OB83**

S7-400 umožňuje odebrání nebo založení modulu během zpracování v režimu RUN nebo STOP. Vyjímku tvoří moduly CPU, PS, S5 moduly v adaptačním modulu a IM moduly.
Po odebrání modulu v RUN režimu mohou být volány některé z uvedených OB bloků ( v závislosti na okolnostech) :

- OB85-aktualizace obrazu procesu
- OB122-I/O přístupová chyba
- OB83- odebrání/založení modulu.

Je nutné vzít v úvahu, že OB83 je volán po cca. 1sec., zatímco ostatní OB mohou být vyvolány mnohem dříve.

Modul je po založení zkontrolován CPU a pokud není detekována chyba, jsou modulu předány parametrizační data. Po dokončení parametrizace je modul zpřístupněn.

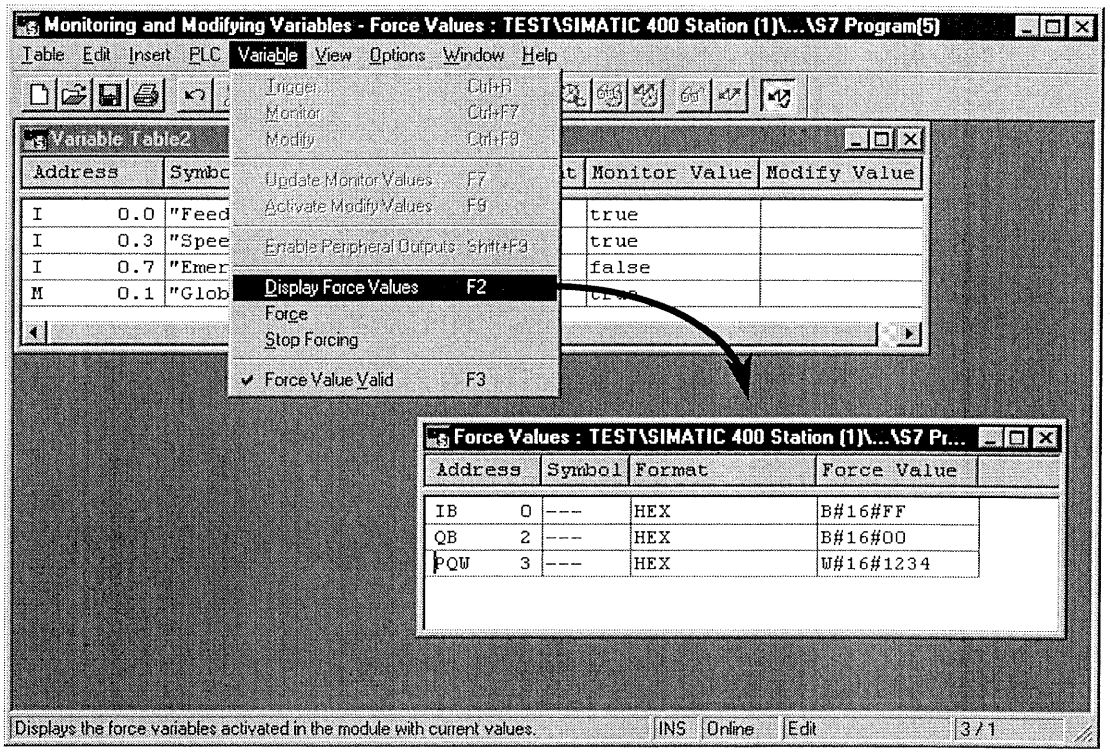
Pokud je při parametrizaci detekována chyba je automaticky vyvoláno diagnostické přerušení a OB82.

**Startovní data v OB83**

OB83 si ukládá do lokálních proměnných následující údaje:

- odebrání/založení modulu
- Logickou adresu modulu
- Aktuální typ modulu

# Ovládání výstupů v S7-400






***Forcing***

Funkce *Forcing* umožňuje u S7-400 zadávat připravené hodnoty do zvolených proměnných.

**Poznámka**

- Před spuštěním funkce je nutné se přesvědčit, že žádný jiný uživatel neaktivuje současně tuto funkci.
- Funkci lze ukončit nebo smazat pouze příkazem z nabídky *Variable -> Stop Forcing*.
- "Forcing" nelze "vzít zpět" pomocí nabídky *Edit -> Undo* .
- Zavření okna *Force Values* nebo ukončení programu "Monitor/Modify Variables" neukončuje funkci *Force*.
- V *On-line* nápovědě jsou uvedeny rozdíly mezi funkcemi *Forcing* a *Modifying Variables*.

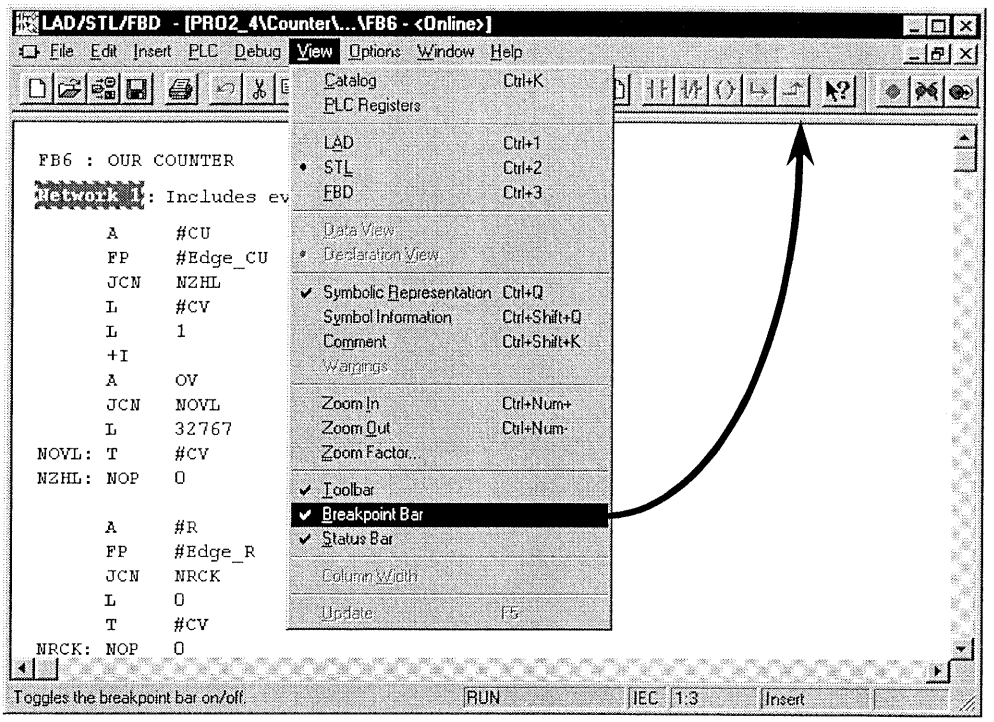
**Aktivace funkce "Force"**

K aktivaci funkce Force je nutné provést následující kroky:

1. Z nabídky *Table -> Open* nebo dvojklikem otevřít požadovanou tabulku proměnných.
2. Pomocí nabídky *PLC -> Connect* To navázat komunikaci s příslušným CPU.
3. Aktivací nabídky *Variable -> Display Force Values* otevřít okno s ovládanými proměnnými.

   Nabídku pro aktivaci funkce *Force* lze vybrat pouze je-li aktivní okno "Force Values". Pokud není úloha Force aktivní, je okno prázdné. Je-li úloha aktivní jsou proměnné zobrazeny tučným písmem včetně hodnot.
4. Ve sloupci "Address" se definují "forsované" proměnné, ve sloupci "Force Value" se definuje příslušná "forsovaná" hodnota.
5. Spuštění funkce se provede z nabídky *Variable -> Force*. Jestli že není úloha momentálně aktivní, jsou proměnným přiřazeny definované hodnoty.
6. Ukončení funkce *Force* se provede pomocí nabídky *Variable -> Stop Forcing*.

# Aktivace lišty bodů přerušení







Datum: 24.11.2002
Soubor: PRO2_11cz.28



**Body přerušení**

Tato testovací funkce umožňuje při zobrazení programu v režimu STL "zastavit" zpracování programu a vyhodnotit obsah registrů CPU.

V bloku může být definováno více bodů přerušení. Jejich počet závisí na typu CPU:

- CPU 412,413: 2 body
- CPU 414: 3 body
- CPU 416: 4 body

**Poznámka**

- Pro výběr funkce "Breakpoint" musí být blok otevřen v "on-line" režimu.
- Funkci "Breakpoints" lze vybrat - aktivovat, pouze tehdy, když je aktivována volba *Debug -> Operation -> Test Operation.*
- Nabídka *Execute Next Statement* nebo *Execute Call* vyžaduje pro správnou funkci volný bod přerušení.
- Pokud při zpracování programu dojde CPU k bodu přerušení, přejde z RUN režimu do HOLD režimu. V tomto režimu svítí LED dioda STOP a současně bliká LED dioda RUN.

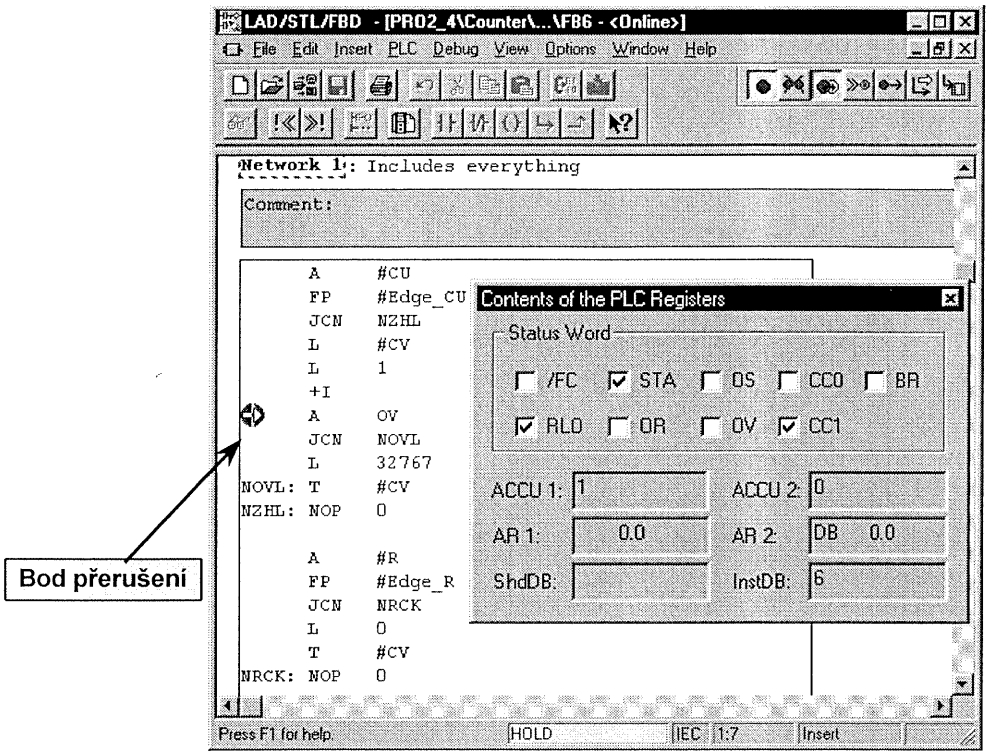
**Funkce bodů přerušení**

Funkce spojené s body přerušení lze aktivovat z editoru programu přes nabídku *Debug* nebo přes nástrojovou lištu.

**Nástrojová lišta**

Nástrojovou lištu s funkcemi bodů přerušení lze zobrazit a schovat pomocí nabídky *View -> Breakpoint Bar.*

# Zpracování programu s body přerušení (pouze S7-400)



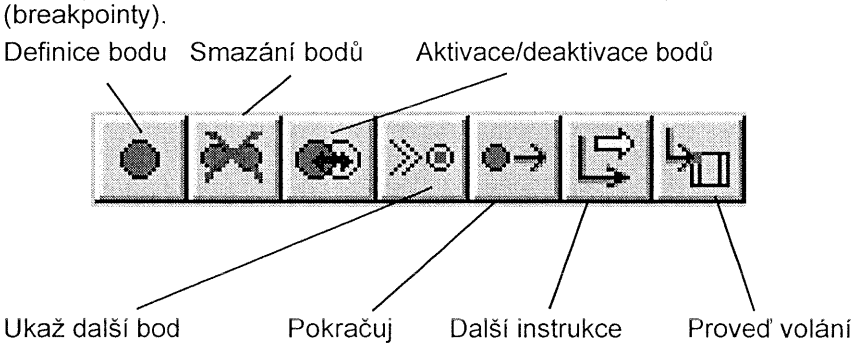




Datum: 24.11.2002
Soubor: PRO2_11cz.29



**Nástrojová lišta** Nástrojová lišta nabízí tlačítka pro snadnější práci s body přerušení (breakpointy).




**Nastavení bodu** Tato funkce definuje nové místo s bodem přerušení, kde bude zpracování programu zastaveno. Instrukce přerušení není zpracována.

**Smazání bodů** Všechny body přerušení jsou smazány.

**Aktivace/deaktivace** Aktivuje všechny body přerušení.

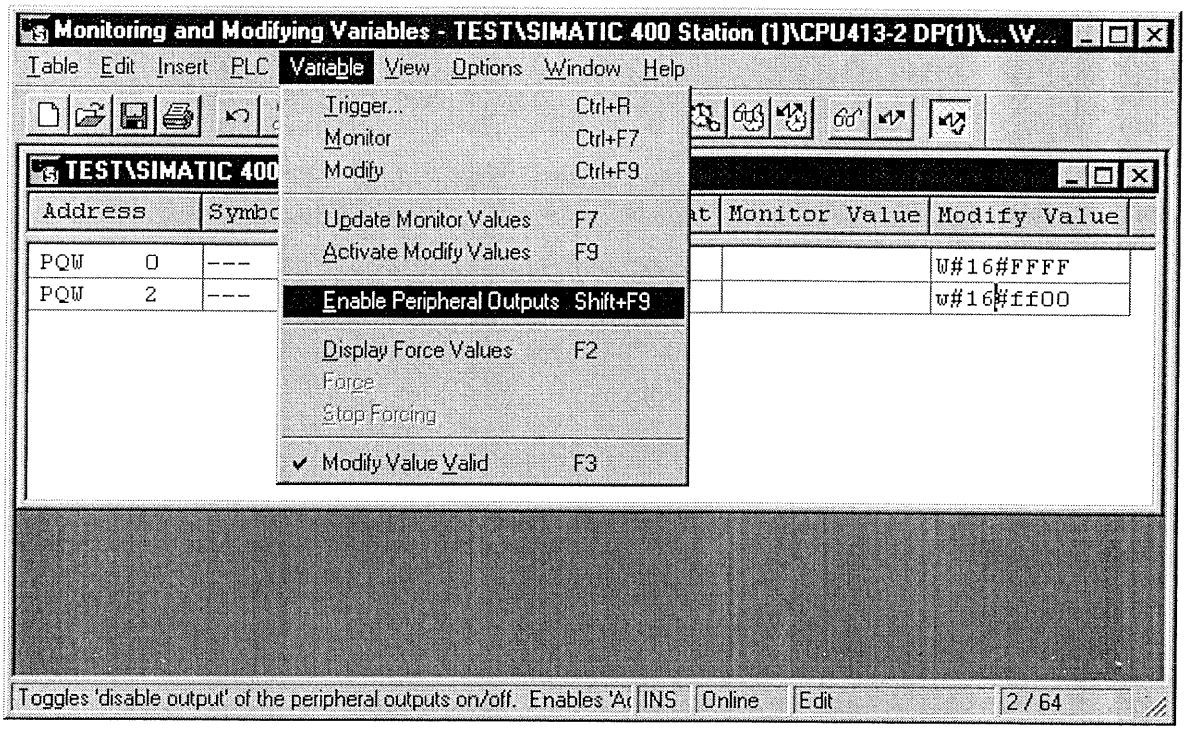
**Ukaž další bod** Zobrazuje část programu kde je definován další *breakpoint.* V editoru se zobrazí další část programu, aniž by došlo k jeho zpracování.

**Pokračuj** Uvolňuje zpracování programu až do dosažení dalšího aktivního *breakpointu.*

**Další instrukce** Umožňuje zpracování programu po jednotlivých instrukcích. Pokud je následující instrukcí volání bloku, provede se toto volání jako jedna instrukce (v jiných IDE je tato funkce někdy označována jako "step over")

**Proveď volání** Na rozdíl od předchozí funkce je při dosažení volání bloku umožněno "krokování" volaného bloku (v jiných IDE je tato funkce někdy označována jako "step into")

# Uvolnění periferních výstupů (pouze S7-400)







Datum: 24.11.2002
Soubor: PRO2_11cz.30



**Úvod**

Funkce "enable peripheral outputs" uvolňuje blokování periferních výstupů (PQ). To umožňuje měnit hodnoty periferních výstupů i ve STOP stavu CPU.

**Provedení**

Při aktivaci této funkce je nutné postupovat následovně:

1. Otevřít/vytvořit tabulku proměnných (VAT), která obsahuje požadované periferní výstupy.
2. Aktivovat nabídku *PLC -> Connect To* a navázat komunikaci s žádaným CPU.
3. Pomocí nabídky *PLC -> Operating Mode* přepnout CPU do STOP stavu.
4. Zadat žádané hodnoty periferních výstupů do sloupce "Modify Value".

   Příklad: PQB 7 Hodnota : 2#0001000011
   PQW 2 W#16#0027
   PQD 4 DW#16#0001
5. Z nabídky *Variable -> Enable Peripheral Output* uvolnit periferní výstupy.
6. Pomocí nabídky *Variable -> Activate Modify Values* přenést nově definované hodnoty do proměnných

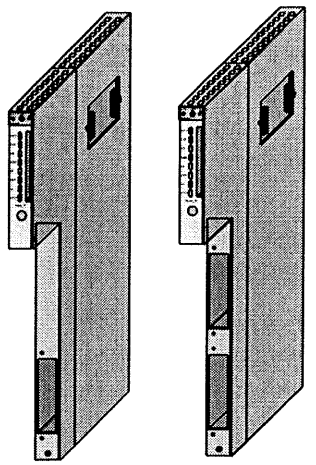
   Funkce *Enable Peripheral Output* zůstává aktivní, dokud není opět vypnuta pomocí nabídky *Variable -> Enable Peripheral Output.*
7. Pro definování nových hodnot proměnných opakovat krok 4.

**Poznámka**

- Jestliže CPU změní režim činnosti ze STOP stavu do STARTUP nebo RUN, zobrazí se příslušné hlášení.
- Stejně tak je zobrazeno hlášení, pokud je CPU v RUN režimu a je aktivována funkce "enable peripheral outputs".

# CP 441: Point-to-Point spojení

- Rozhraní
  - CP 441-1: 1 volitelný modul rozhraní
  - CP 441-2: 2 volitelné moduly rozhraní
- Signalizace:
  - LED-diody pro vysílání, příjem,chyby
- Rychlost
  - CP 441-1: max. 38.4 kBaud
  - CP 441-2: max. 76.8 kBaud
- Protokoly
  - standardní integrované protokoly
  - volitelné ne-Siemens protokoly (pouze CP 441-2)



CP 441-1 CP 441-2

- CP441-1: Cenově výhodný standard
- CP441-2: Vysoký výkon pro náročné úlohy

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_11cz.31

Školicí středisko
firmy E&A spol. s r.o.

**Popis**

*Point-to-point* spojení je cenově výhodná a velice výkonná alternativa sběrnicových systémů. Umožňuje jednoduché připojení např. čtečky čárových kódů, váhových systémů nebo jiných řídících systémů.

**Protokoly CP441-1**

- parametrizovatelný 3964 (R)
- parametrizovatelný ASCII protokol
- tiskárna

**Protokoly CP441-2**

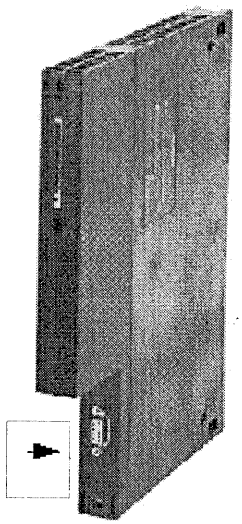
- parametrizovatelný 3964 (R)
- RK 512
- parametrizovatelný ASCII
- tiskárna
- Rozšiřující externí protokoly např.:
  - Modbus master / slave (Modicon)
  - Allen Bradley (protokol DF1)

**Moduly rozhraní**

- TTY-, 9-pin konektor, aktivní/pasivní module
- V.24 (RS232 C), 9-pin konektor
- RS 422/485, 15-pin konektor

# CP 443-5 : PROFIBUS spojení

- **Formát: S7, jednoduchá tloušťka**
- **Protokoly:**
  - **SEND/RCV**
  - **S7 Funkce**
  - **FMS (pouze CP 443-5 Basic)**
  - **DP Master (CP 443-5 Extended)**
- **Rychlost: 9.6 Kbps až 12 Mbps**
- **Spojení:**
  - **elektrický kabel: 9-pin konektor**
  - **světelný kabel: přes bus terminál**
- **Konfigurace:**
  **NCM S7 pro PROFIBUS**
  **obsahující FC a FB**



**CP 443-5 Basis**
**CP 443-5 Extended**

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_11cz.32

Školicí středisko
firmy E&A spol. s r.o.

**Popis**

Komunikační procesor CP 443-5 umožňuje připojení S7-400 do sítě PROFIBUS.

**Protokoly**

Pro CP443-5 jsou dostupné tyto protokoly:

S7 Funkce:

Protokol je určen pro základní komunikaci mezi SIMATIC S7/M7/C7 a PC na základě vrstvy 7, ISO/OSI referenčního modelu.

S7 protokol nabízí SFB rozhraní pro komunikaci S7-CPU s automaty řady SIMATIC S7. Jako doplněk poskytuje funkce pro programování, testování, správu a diagnostiku.

SEND/RCV:

Protokol založený na vrstvě 2 (FDL-Layer) pro komunikaci mezi SIMATIC S7, SIMATIC S5, PC/PG a ne-Siemens jednotkami. Rozhraní SEND/RCV poskytuje jednoduchou komunikaci pro výměnu nestrukturovaných bloků dat.

SEND/RCV rozhraní je implementováno v jednotlivých komponentech jako

- komunikační bloky SIMATIC S5
- Funkce SIMATIC S7
- C-funkce PG/PC

PROFIBUS FMS (*Fieldbus Message Specification*)

Protokol určený pro otevřenou komunikaci mezi systémy SIMATIC S5, S7, PC/PG, průmyslové stanice a ne-Siemens produkty. (EN 50170, Vol. 2, PROFIBUS).

Protokol FMS umožňuje objektově orientovanou komunikaci založenou na vrstvě 7 ISO/OSI referenčního modelu. Typický objem datového paketu je 240 bytů.

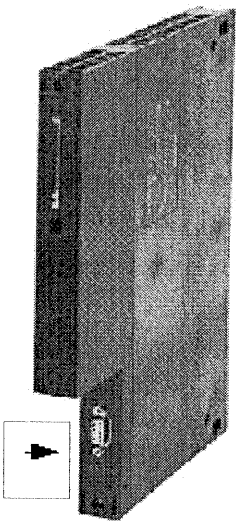
PROFIBUS DP (*Distributed Peripheral*)

Protokol určený pro otevřenou komunikaci mezi systémy SIMATIC S5, S7, PC/PG nebo ne-Siemens systémy a průmyslovými zařízeními. (EN 50170, Vol. 2, PROFIBUS).

DP protokol je navržen pro rychlou výměnu malého objemu dat mezi systémem nebo PC a průmyslovým zařízením s reakční dobou < 10 ms.

# IM 467: PROFIBUS-DP Master rozhraní

- Formát: S7, jednoduchá tloušťka
- Protokoly:
  - DP Master
  - S7 Funkce
- Rychlost: 9.6 Kbps až 12 Mbps
- Spojení:
  - elektrický kabel: 9-pin sub-D konektor
- Konfigurace:
  - Konfigurace a programování je možné přes PROFIBUS-DP.



IM 467

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_11cz.33

Školicí středisko
firmy E&A spol. s r.o.

**Popis**

Modul rozhraní IM 467 je určený pro operace v systémech S7-400. Umožňuje připojení S7-400 do sítě PROFIBUS-DP.

**Protokoly**

IM 467 nabízí dva protokoly:

PROFIBUS-DP

IM 467 je PROFIBUS-DP Master v souladu s EN 50 170. Konfigurace se provádí kompletně ze STEPu 7.Výkon je srovnatelný s integrovaným PROFIBUS-DP rozhraním ostatních CPU.

Pro DP komunikaci není nutné žádné volání funkcí v uživatelském programu.

S7 Funkce

S7 funkce zaručují optimální a jednoduchou komunikaci při řešení úloh pomocí SIMATIC S7/M7/C7. U IM467 jsou dostupné následující funkce:

- PG funkce přes PROFIBUS-DP
- HMI funkce přes PROFIBUS-DP

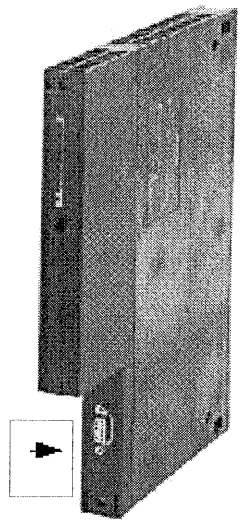
Komunikace probíhá bez nutnosti konfigurace IM modulu.

S7 funkce lze použít samostatně nebo paralelně s funkcemi PROFIBUS-DP protokolu.

Paralelní použití obou funkcí se však odráží na době cyklu PROFIBUS-DP sběrnice.

# CP 443-1: spojení po Industrial Ethernetu

- Formát: S7, jednoduchá tloušťka
- Protokoly:
  - SEND/RCV a S7 funkce na ISO transportním zásobníku (CP 443-1) a TCP/IP zásobníku (CP 443-1 TCP/IP)
  - MMS/MAP (v přípravě)
- Spojení:
  - S7 funkce: max. 48 účastníků
  - SEND/RCV: max. 64 účastníků
- Funkce:
  - Multiprotokolové možnosti
  - Vzdálené programování přes LAN a WAN (pouze CP443-1 TCP/IP)
- Spojení:
  - Automaticky přepínané mezi AUI a Twisted Pair spojením
- Konfigurace:
  - NCM-S7 pro Industrial Ethernet s voláním funkcí pro SEND/RCV



**CP 443-1**
**CP 443-1 TCP/IP**

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_11cz.34

Školicí středisko
firmy E&A spol. s r.o.

**Popis**

Komunikační procesory CP 443-1 a CP 443-1 TCP/IP umožňují spojení S7-400 po síti Industrial Ethernet.

**Protokoly**

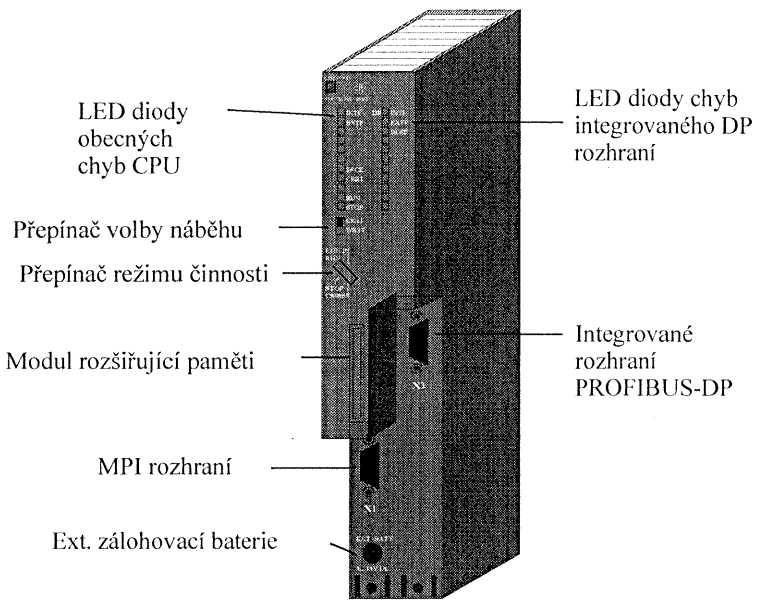
S7 Funkce

Protokol pro základní komunikaci mezi SIMATIC S7/M7/C7 a PC založený na vrstvě 7 ISO/OSI referenčního modelu.

SEND/RCV

Protokol pro komunikaci mezi SIMATIC S7, SIMATIC S5, PC/PG a ne-Siemens stanicemi MMS/MAP založený na vrstvě 4 (ISO-Transportní vrstva na CP 443-1 a TCP-Transportní vrstva na CP 443-1 TCP/IP).
Je určen pro otevřenou komunikaci na základě vrstvy 7 mezi SIMATIC S7, SIMATIC S5, PC/PG a ne-Siemens systémy.

# Parametrizace decentrálních periferií



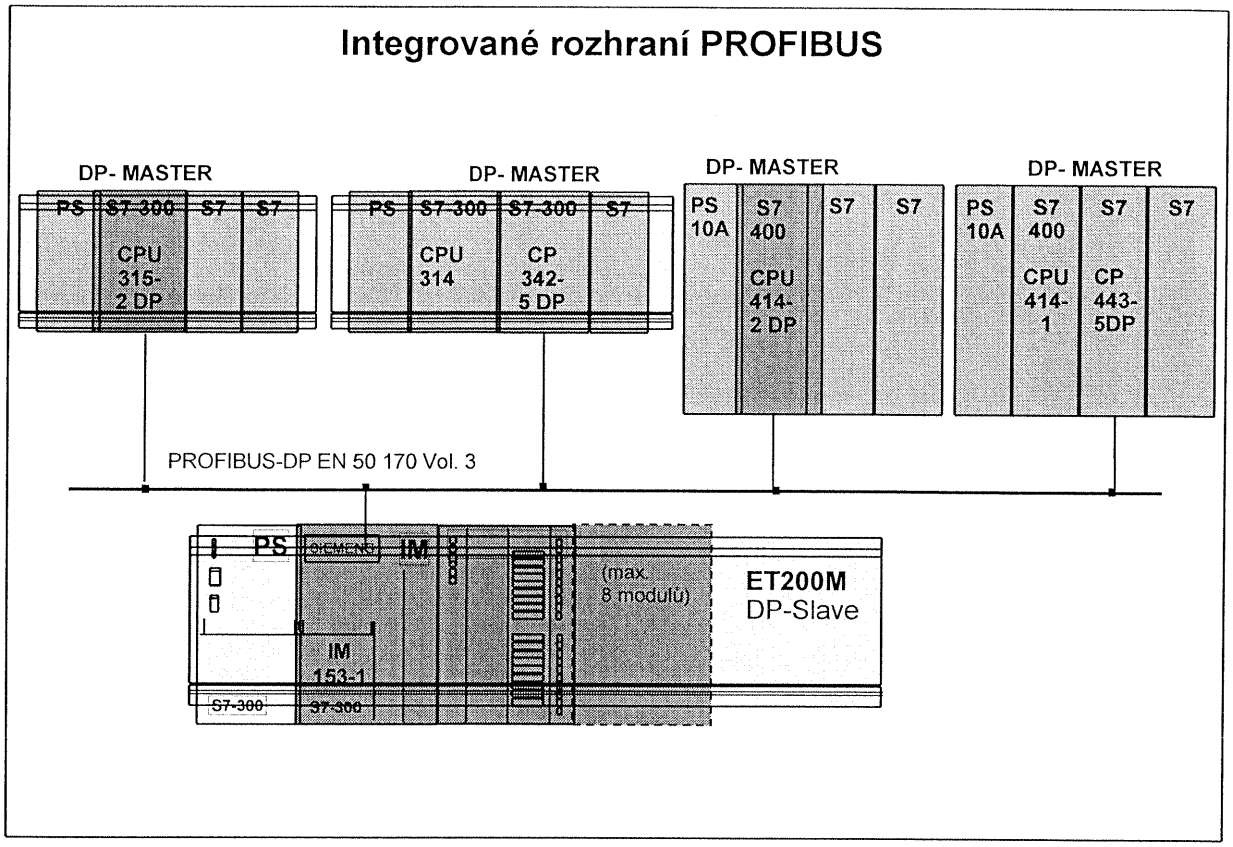

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_12cz.1

Školicí středisko
firmy E&A spol. s r.o.,

**Obsah** **Strana**






**Připojení**

Systémy S7-300 nebo S7-400 lze do sítě PROFIBUS připojit přes komunikační procesory nebo přes CPU s integrovaným rozhraním PROFIBUS Master.

CPU s integrovaným rozhraním PROFIBUS Master umožňují jednoduchou výstavbu decentrálních periferií s vysokou přístupovou rychlostí (až 12MBaud).

Decentrální periferie jsou přitom z hlediska uživatele obsluhovány stejně jako centrální periferie (konfigurace, adresace, programování).

**Master**

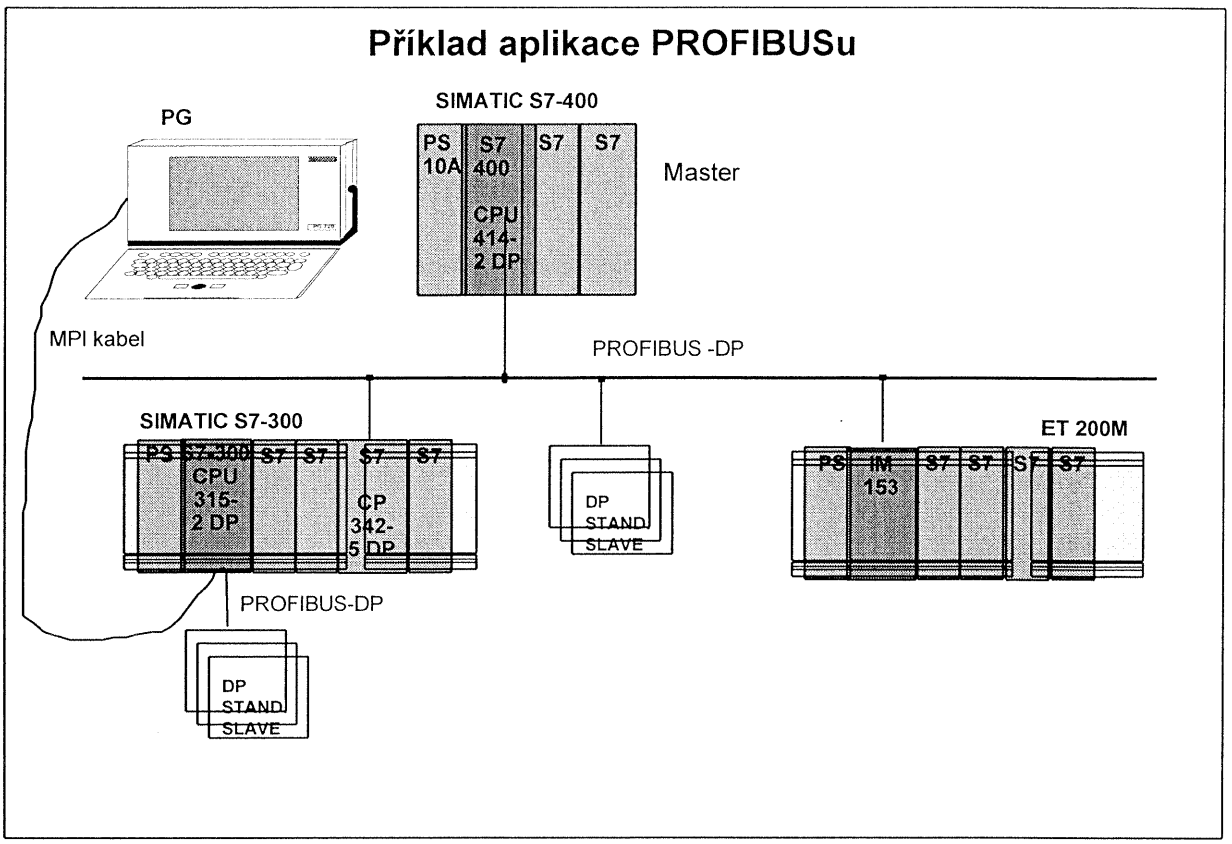
- S7-300 nebo S7-400 CPU s integrovaným PROFIBUS Master rozhraním
- CP v kombinaci s CPU
- SIMATIC S5

**DP Slave**

- Decentrální I/O zařízení ET 200
- DP-slave stanice jiných výrobců
- Operátorské panely (OP)

**Varování**

Nejpomalejší stanice sítě určuje celkovou maximální rychlost sítě, ve které je zařazena.






**DP Standard Slave**

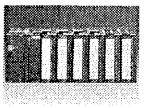
SIEMENS nabízí celou řadu standardních slave stanic pro síť PROFIBUS (DP standard EN 50170):

- ET200L, ET200B, ET200C
- S5-95U s DP-Slave rozhraním,
- S7-300 s CP342-5 jako slave
- DP/PLC Interface Link
- Operátorské panely OP5...OP45
- IM328-N a IM329-N pro připojení CNC řídícího systému 840C do Profibusu
- absolutní polohovací čidla (12 bitová jednootáčková/24bitová víceotáčková) s rozhraním PROFIBUS
- Digital SIMOREG Konvertor 6RA24
- SIMOVERT Master Drives
- SIMADYN D
- SIPART
- Identifikační systém ASM-440 (MOBY-I a MOBY-L)
- motorové ochrany a řídící jednotky SIMOCODE
- systémy TELEPERM M

**CP 342-5DP, CP 443-5DP**

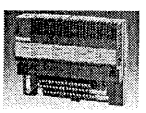
Pro parametrizaci, programování, testování a údržbu komunikačních procesorů je nutný doplňkový program NCM S7.

## Dostupné DP Slave-stanice

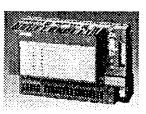


ET 200M — obsahuje modul rozhraní a moduly řady S7-300

ET 200U — obsahuje modul rozhraní a moduly řady S5

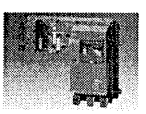


ET 200B



ET 200L — malé, kompaktní I/O stanice s integrovanými vstupy a výstupy

ET 200C — malá, kompaktní stanice s integrovanými vstupy a výstupy, s krytím IP 66/ IP 67



ET 200X — malá, kompaktní stanice s integrovanými vstupy a výstupy

SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_12cz.4

Školicí středisko firmy E&A spol. s r.o.,

**Modulární Slave ET 200M**

ET200M obsahuje modul rozhraní IM153-1, kterým se připojuje na S7/M7-PROFIBUS Master systém. Umožňuje použití všech modulů řady S7-300 s P-Bus sběrnicí.

ET 200M umožňuje připojení maximálně 128 bytů vstupů a výstupů při rychlosti až 12 MBaud

**ET 200L, ET 200B**

ET 200L a ET 200B obsahují terminál a elektronický blok a digitálními a analogovými kanály. ET 200L lze použít až do rychlosti 1.5 MBaud. ET 200B lze s výhodou použít tam, kde je malý montážní prostor. Maximální rychlost komunikace je 12Mbaud.

**ET 200C**

Kompaktní ET 200C má vysoký stupeň krytí IP66/IP77. Je proto použitelný i ve vnějším prostředí. Max. rychlost je 12Mbaud pro digitální signály a 1.5 MBaud pro analogové signály.

**ET 200X**

Kompaktní ET 200X má také vysoký stupeň ochrany IP 66/IP 67 a obsahuje základní a rozšiřující moduly.

**Inteligentní Slave stanice**

např. CP 342-5 nebo AG 95 PROFIBUS.

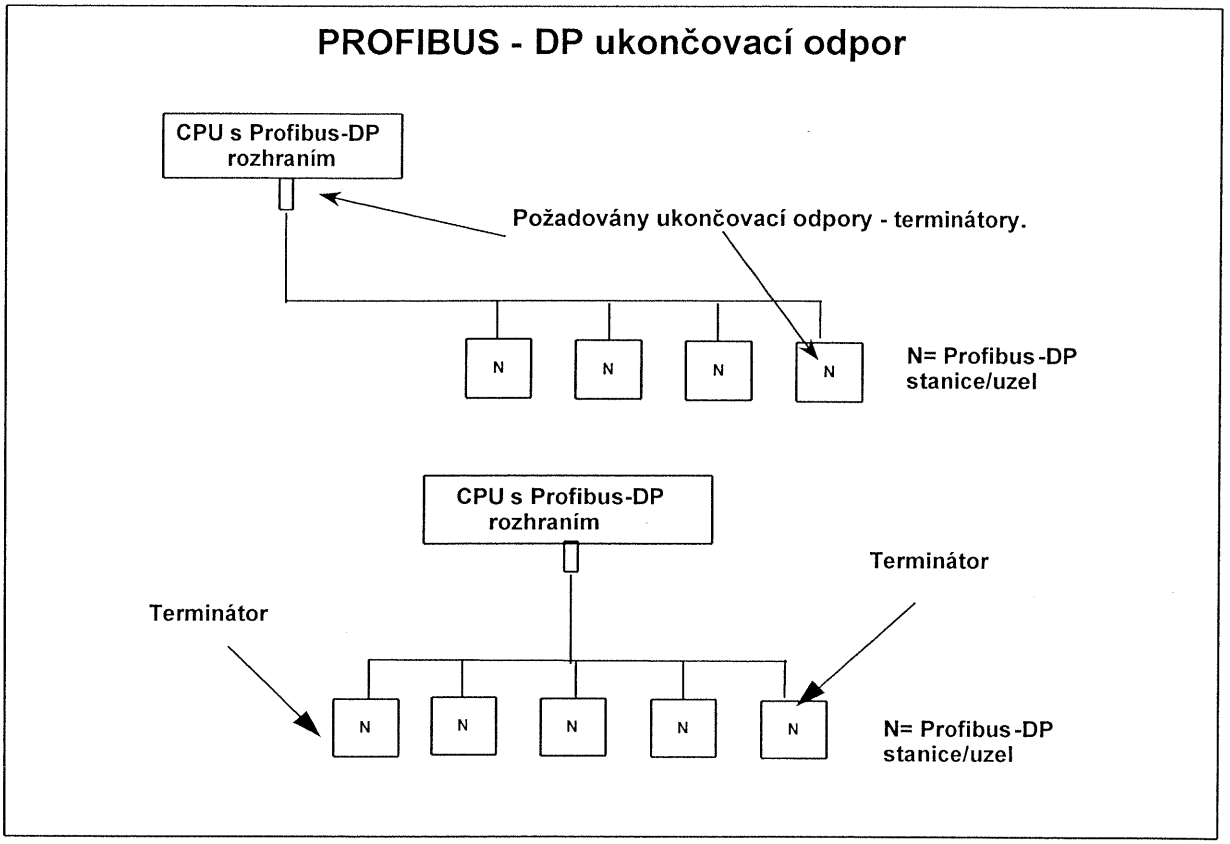
**Další stanice**

Seznam možných slave stanic je uveden v katalogu *ST PI for PROFIBUS*

**Slave stanice jiných výrobců**

Do sítě PROFIBUS lze připojit všechna zařízení, která splňují podmínky EN 50 170.

## PROFIBUS - DP ukončovací odpor




SIMATIC S7

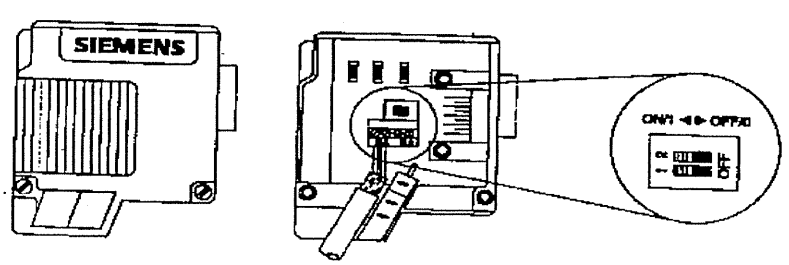


Datum: 24.11.2002
Soubor: PRO2_12cz.5



**Terminátor**

První a poslední uzel segmentu sítě musí mít zapnutý ukončovací odpor - terminátor. Tento terminátor je integrován v konektoru pro připojení do sítě. Zapnutí se provede přepnutím přepínače do polohy ON, který je podle provedení přístupný na krytu konektoru nebo po odkrytování.




**Délka kabelu**

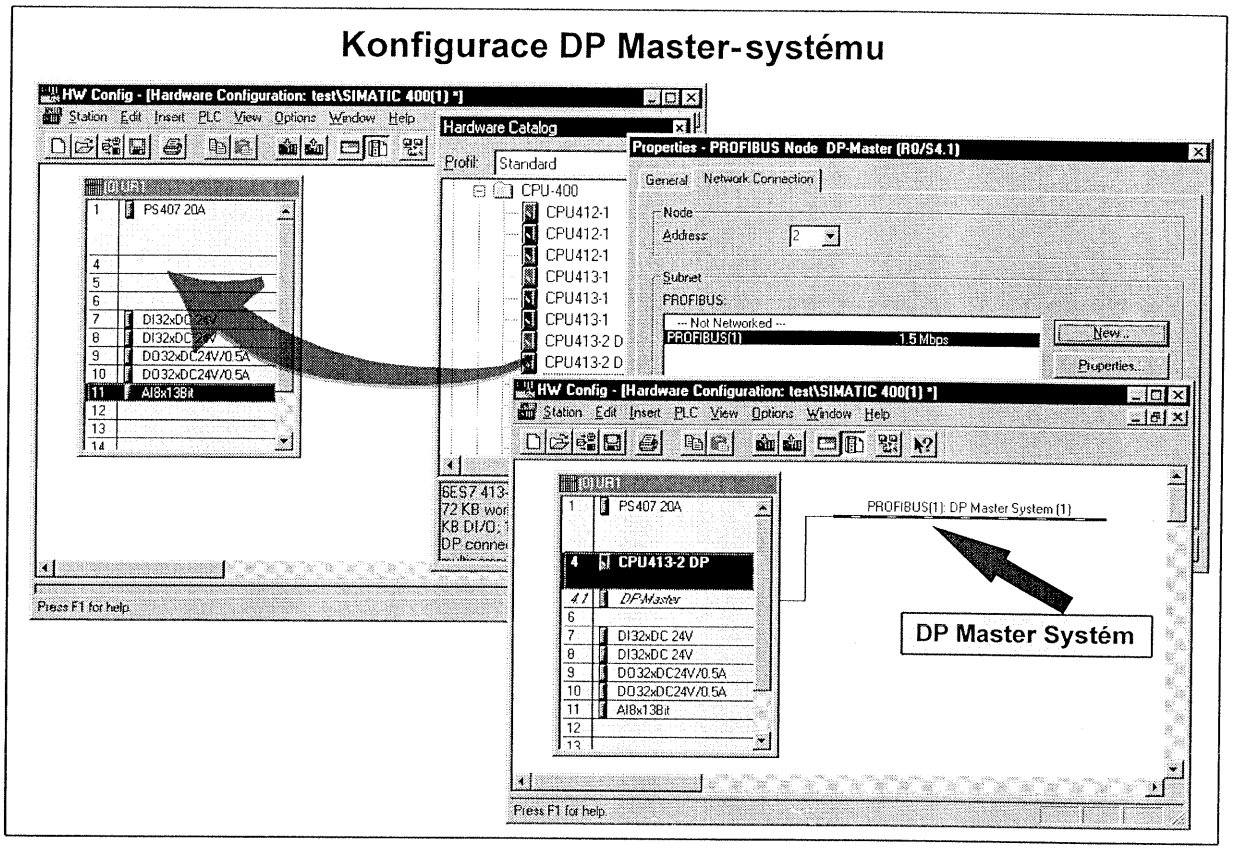
Maximální délka segmentu závisí na komunikační rychlosti:

| Rychlost | Délka segmentu |
|---|---|
| 9.6 to 187.5 kBaud | 10000 m |
| 500 kBaud | 400 m |
| 1.5 MBaud | 200 m |
| 3 to 12 MBaud | 100 m |

Maximální délka segmentu MPI sítě je 50 m.

**Repeater**

Celkově lze použít až 9 za sebou.







Datum: 24.11.2002
Soubor: PRO2_12cz.6



**Distribuované I/O**

Všechny systémy obsahující DP master a DP slave-stanice jsou označovány jako distribuované I/O, které jsou propojeny sběrnicovým kabelem a komunikují pomocí PROFIBUS-DP protokolu.

**DP Master**

Jako DP master lze instalovat:

- S7-CPU s integrovaným DP master rozhraním (např.. CPU 414-2, atd.)
- moduly rozhraní M7-CPU/M7-FM.
- CP ve spojení s CPU (např.. CP 443-5, atd.)

**Parametrizace DP Master**

Parametrizace se provádí pomocí programu *HWConfig,* postup je následující:

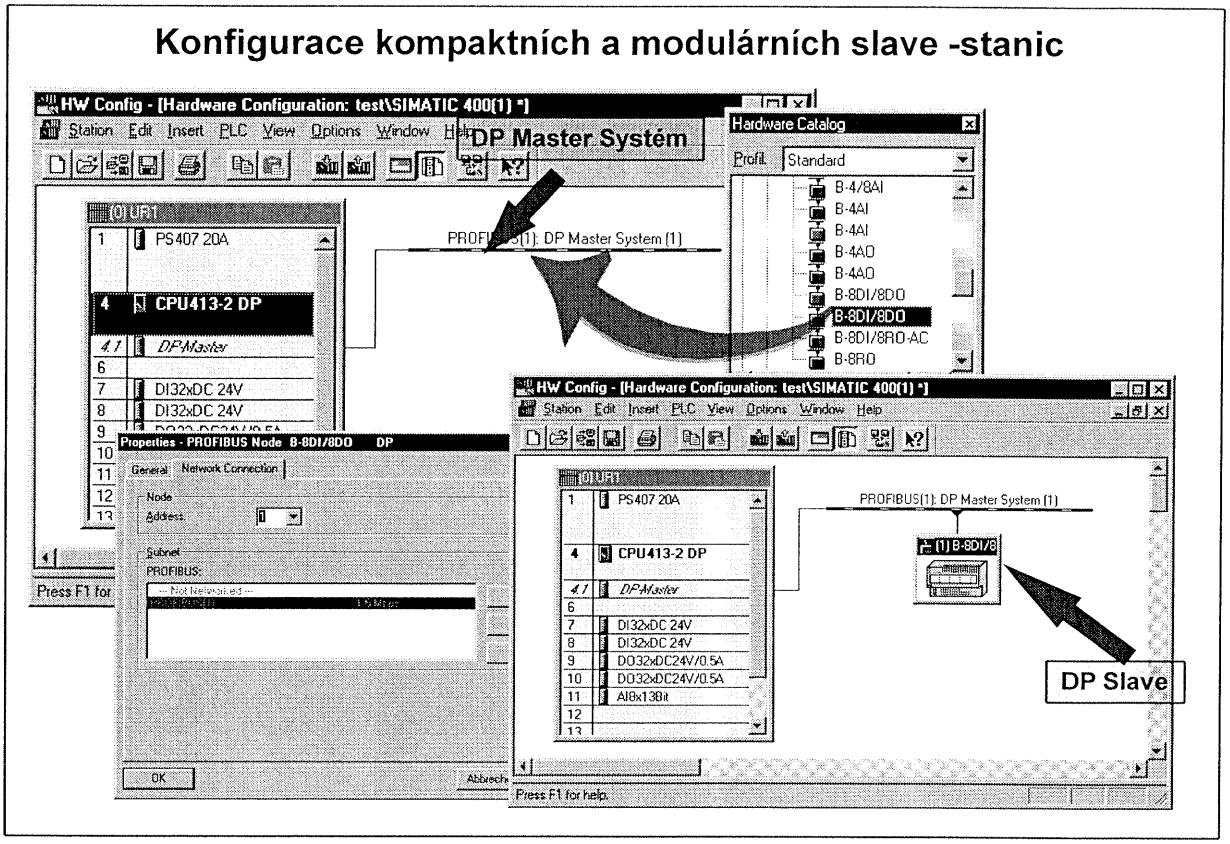
1. v okně *Hardware Catalog* vybrat DP master modul.
2. Metodou *Drag&Drop* umístit modul do nosiče modulů

   Otevře se dialogové okno "Properties - PROFIBUS Nodes".
   V tomto okně lze definovat následující vlastnosti:
   - definovat novou síť PROFIBUS nebo vybrat existující.
   - nastavit vlastnosti sítě PROFIBUS (rychlost, atd.).
   - definovat PROFIBUS adresy DP mastera.
3. Nastavení potvrdit tlačítkem "O.K.". Zobrazí se symbol sítě: 
   Na tento symbol se potom umisťují jednotlivé slave stanice.

**Poznámka**

Lze definovat síť s jedním master systémem nebo s několika master systémy. V multi-master síti je v jednom segmentu zapojeno několik master systémů.

# Konfigurace kompaktních a modulárních slave -stanic




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_12cz.7

Školicí středisko
firmy E&A spol. s r.o.,

**DP Slave**

- moduly s integrovanými digitálními/analogovými vstupy/výstupy (kompaktní DP slave, např. ET200B).
- moduly rozhraní s připojenými S5 nebo S7 moduly (modulární DP slave, např. ET200M).
- S7-200/300 stanice s moduly, které podporují funkci "Intelligent Slave" (např. CPU 215-DP, CPU 315-2).

**Výběr DP Slave**

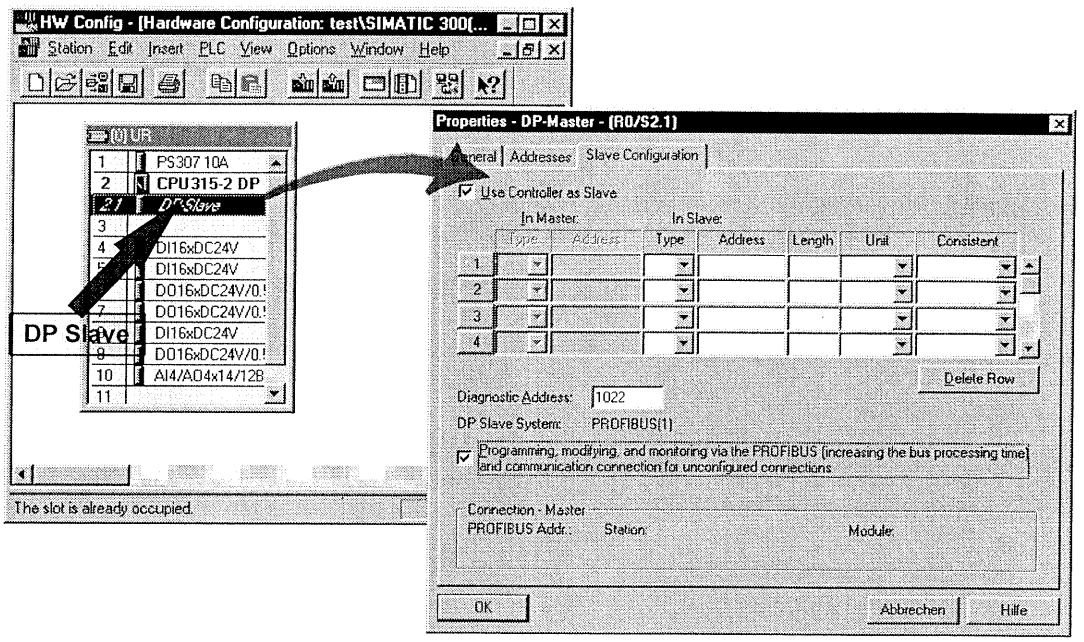
Konfigurace DP slave stanice se provádí následujícím způsobem:

1. z HW katalogu vybrat požadovaný kompaktní DP slave (např. ET200B) nebo modul rozhraní (např. IM153 pro ET200M).
2. metodou *Drag&Drop* umístit symbol do master systému DP sítě.

   Otevře se okno "Properties - PROFIBUS Nodes". Zde lze definovat:
   - vlastnosti segmentu sítě PROFIBUS (rychlost, atd.).
   - PROFIBUS adresu DP slave stanice.
3. Potvrdit nastavení tlačítkem "O.K.". K symbolu kabelu segmentu se připojí symbol představující příslušnou slave stanici.
4. U modulární stanice lze nyní vkládat jednotlivé moduly z HW katalogu do konfigurační tabulky.

   Adresace a parametrizace těchto modulů se provádí stejným způsobem, jako u modulů v centrální konfiguraci.

# Konfigurace inteligentních DP Slave stanice (např. CPU 315-2)








**Inteligentní Slave**

Základní vlastností inteligentních DP slave stanic je, že data nejsou master systému poskytována přímo, ale předzpracovaná CPU.

U inteligentní DP slave stanice nemá DP master přístup přímo na vstupy/výstupy slave stanice, ale do oblasti předzpracovaných dat. Uživatelský program inteligentní slave stanice musí zajišťovat datovou výměnu mezi vstupy/výstupy a přístupovou oblastí pro DP master.

**Poznámka**

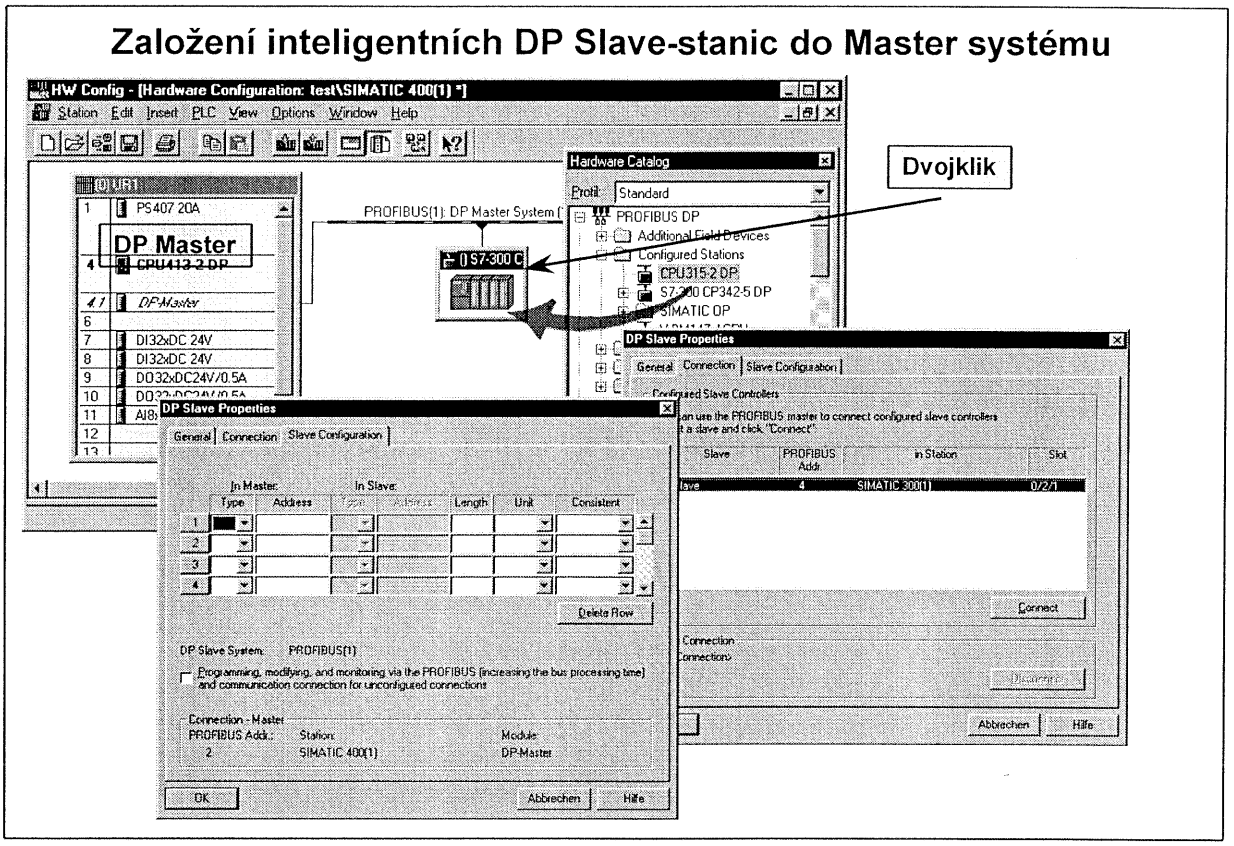
Inteligentní DP slave (např. CPU 315-2 DP) nemůže být současně konfigurován jako DP master a DP slave. CPU 315-2 DP konfigurované jako DP slave nemůže být současně pro jiné DP slave stanice master systémem.

**Konfigurace DP Slave**

Konfigurace CPU 315-2 DP jako inteligentní DP slave stanice se provádí následovně:

1. založit S7-300 Station v projektu.
2. spustit program *HWConfig*.
3. z HW katalogu založit do nosiče CPU 315-2 DP.
4. dvojklikem na řádku 2.1 v konfigurační tabulce otevřít okno "Properties - DP Master".
5. na kartě "Slave Configuration" aktivovat volbu "Use Controller as Slave".
6. definovat ostatní PROFIBUS parametry CPU 315-2-DP.
7. potvrdit nastavení tlačítkem "O.K.".

# Založení inteligentních DP Slave-stanic do Master systému







Datum: 24.11.2002
Soubor: PRO2_12cz.9



**Vložení int. DP Slave**

Postup při vkládání CPU 315-2 DP jako inteligentní slave stanice je následující:

1. do projektu vložit DP master stanici (např. S7-400).
2. spustit *HW Config.*
3. vložit DP master systém (např. CPU 414-2 DP) z katalogu do konfigurační tabulky.
4. dvojklikem na objektu "DP Master" otevřít okno "Properties - DP Master".
5. definovat všechny PROFIBUS parametry DP master systému a potvrdit je tlačítkem "OK".
6. metodou Drag&Drop umístit CPU 315-2 DP z "Hardware Catalog" do master systému.
7. dvojklikem na druhém řádku DP slave otevřít okno parametrů a vybrat kartu "Connection".

   Zde je uveden seznam všech propojení master - i.slave.
8. označit požadovaný inteligentní DP slave a stisknout tlačítko "Connect".
9. vybrat kartu "Slave Configuration" a definovat vzájemné master a slave adresy přístupových oblastí.

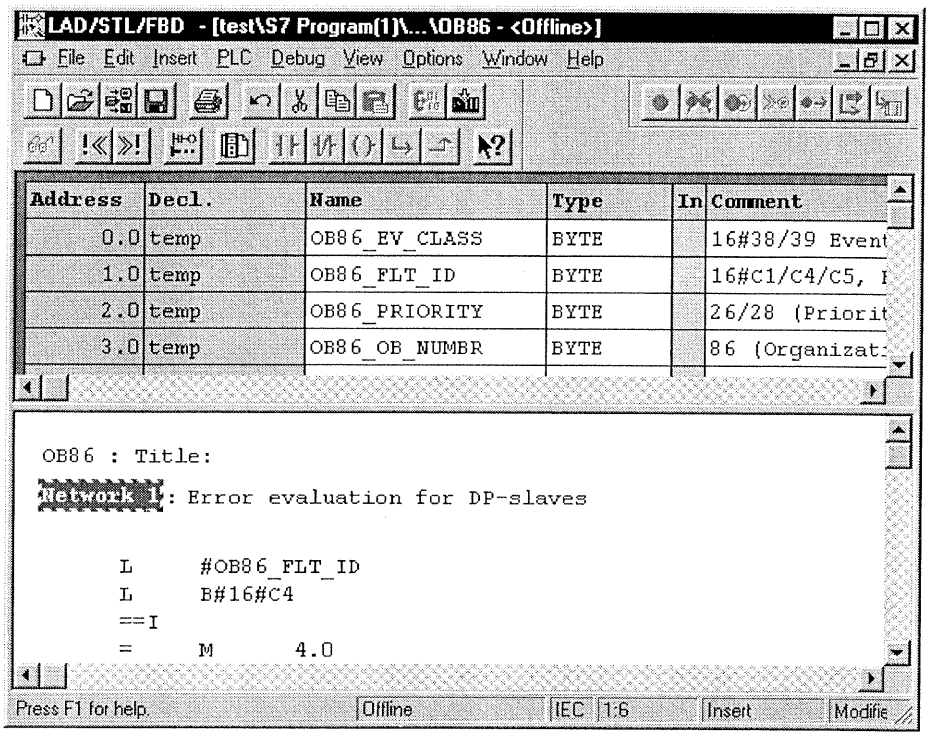
   Vstupní oblast DP master je výstupní oblastí DP slave a obráceně.
10. nastavení potvrdit tlačítkem "O.K.".

**Poznámka**

Pro správnou funkci DP master a slave systému je nutné nahrát do CPU příslušné chybové OB. (OB 86 ,OB 122, atd.).

Před založení int. slave do DP master je nutné, aby v projektu již byl definován systém použitelný jako int. slave, tzn. nejprve je nutno v projektu definovat konfiguraci CPU315-2DP a teprve potom konfiguraci S7-400.

# Chybová analýza v OB 86 při selhání slave-stanice







Datum: 24.11.2002
Soubor: PRO2_12cz.10



**Selhání stanice**

Operační systém CPU(CPU 315-2DP nebo S7-400) aktivuje OB86, jestliže rozeznal chybu nosiče, segmentu sítě nebo stanice distribuovaných I/O.

Jestliže není naprogramován OB86 a systém detekuje chybu, přejde CPU do STOP stavu.

**Proměnné OB86**

- OB86_FLT_ID: B#16#C4 //DP-Stanice porušena
- OB86_FLT_ID: B#16#C5 //DP-Station chybuje
- OB86_MDL_ADDR: Logické základní adresy DP-Mastera (diagnostické adresy)
- OB86_ RACKS_FLTD: ==> přejmenované datové typy v DWORD

  Obsah:

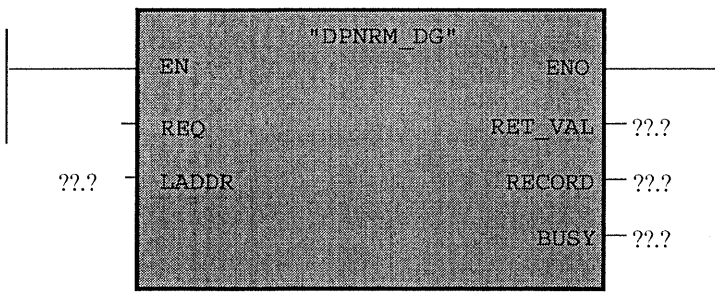
  Bit 0 až 7: číslo DP-Stanice (PROFIBUS adresa)

  Bit 8 až15: identifikátor DP sítě

  Bit 16 až 30: diagnostické adresy DP-Slave stanice

  Bit 31: I/O identifikátor

# Diagnostika slave stanice pomocí SFC 13 (DPNRM_DG)




| Parametr | Deklarace | Typ | Oblast | Popis |
|---|---|---|---|---|
| REQ | INPUT | BOOL | I, Q, M, D, L, konst. | REQ = 1: požadavek na čtení |
| LADDR | INPUT | WORD | I, Q, M, D, L, konst. | diagnostická adresa DP slave stanice |
| RET_VAL | OUTPUT | INT | I, Q, M, D, L | je-li při zpracování zachycena chyba, obsahuje chybový kód. Při bezchybném zpracování obsahuje počet přenesených dat |
| RECORD | OUTPUT | ANY | I, Q, M, D, L | cílová oblast pro diagnostická data. Povolen je pouze datový typ BYTE. Minimální délka oblasti je 6 bytů. |
| BUSY | OUTPUT | BOOL | I, Q, M, D, L | BUSY = 1: Funkce není dokončena - pracuje se. |






**Slave diagnostika**

SFC 13 "DPNRM_DG" umožňuje čtení diagnostických údajů z DP slave stanice. Načtené údaje jsou uloženy do oblasti definované parametrem RECORD (OUT2) a jejich struktura je v souladu s normou EN 50 170.

Aktivace čtení se provádí přiřazením hodnoty 1 parametru REQ (IN0) při volání SFC 13.

Přibližná struktura načtené oblasti je uvedena v tabulce, bližší podrobnosti o DP slave stanicích jsou uvedeny v příslušných manuálech (např.chybové kódy jsou v manuálu NCM -S7).

Přibližná struktura

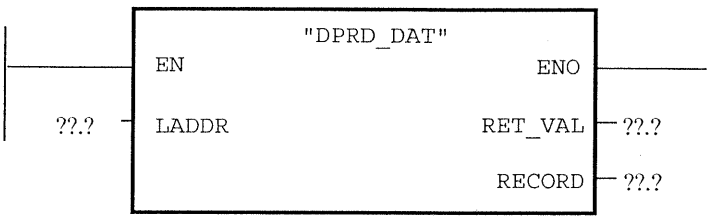
| Byte | Význam |
|---|---|
| 0 | stav stanice 1.část |
| 1 | stav stanice 2.část |
| 2 | stav stanice 3.část |
| 3 | číslo Master Stanice |
| 5 | ID výrobce (nižší byte) |
| 6... | doplňující slave-diagnostika |

**Poznámka**

U normálních slave stanic u kterých je délka diag. dat větší než 240 bytů a celkově 244 bytů, je prvních 240 bytů zapsáno do cílové oblasti a odpovídající bit přetečení je nastaven do 1.

# Čtení dat z DP Slave stanice pomocí SFC 14

- **Použití SFC 14 "DPRD_DAT" je nezbytné, pokud je požadováno čtení více než 4 po sobě jdoucích bytů.**




| Parametr | Deklarace | Typ | Oblast | Popis |
|---|---|---|---|---|
| LADDR | INPUT | WORD | I, Q, M, D, L, konst. | počáteční adresa vstupního modulu, který má být čten |
| RET_VAL | OUTPUT | INT | I, Q, M, D, L | kód chyby detekované při zpracování |
| RECORD | OUTPUT | ANY | I,Q, M, D, L | cílová oblast načtených dat. Délka musí být stejná jako modul konfigurovaný ve STEPu 7. Povolen je pouze datový typ BYTE. |





**Funkce**

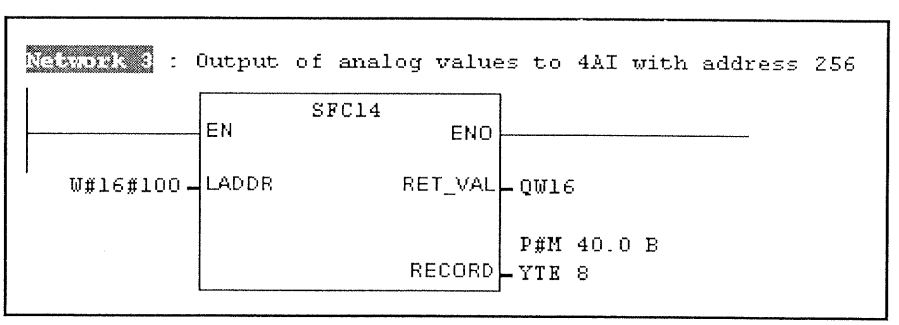
SFC 14 "DPRD_DAT" umožňuje čtení dat z DP slave stanice.

Počet načítaných bytů musí být 3 nebo více než 4. Maximální načítaná délka oblasti je dána konkrétním CPU. Není-li při zpracování zachycena chyba, jsou data uložena do oblasti definované parametrem RECORD.

Cílová oblast musí mít stejnou délku jako vybraný modul konfigurovaný ve STEPu 7.

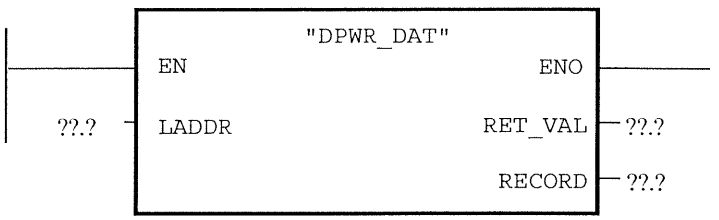
U modulárních DP slave stanic nebo u stanic s více DP identifikátory je počáteční adresou definován jeden modul/DP identifikátor.

**Příklad**



# Zápis dat do DP Slave stanice pomocí SFC 15

- **Použití SFC 15 "DPRD_DAT" je nutné v případě zápisu více jak 4 bytů za sebou.**




| Parametr | Deklarace | Typ | Oblast | Popis |
|---|---|---|---|---|
| LADDR | INPUT | WORD | I, Q, M, D, L, konst. | počáteční adresa výstupního modulu |
| RECORD | INPUT | ANY | I, Q, M, D, L | Zdrojová oblast dat, její délka musí být stejná jako délka zvoleného modulu konfigurovaného ve STEPu7. Povolen je pouze datový typ BYTE. |
| RET_VAL | OUTPUT | INT | I, Q, M, D, L | Kód chyby zachycené při zpracování SFC. |





**Funkce**

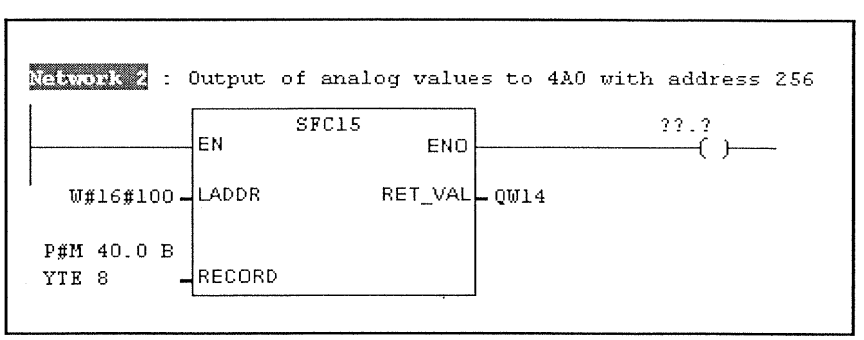
SFC 15 "DPWR_DAT" umožňuje odeslání nejméně 3 nebo více než 4 bytů z oblasti RECORD na DP slave stanici.

Maximální počet zapisovaných bytů je dán konkrétním CPU. Datový přenos je proveden synchronně, tj. po ukončení SFC je ukončena také funkce zápisu dat.

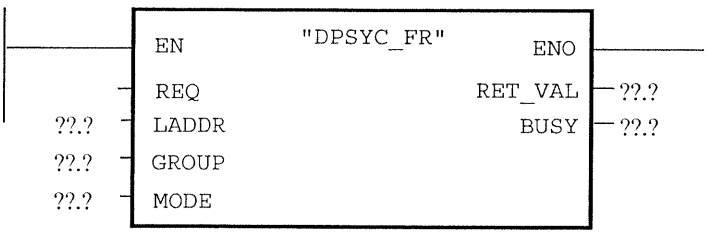
Zdrojová oblast musí mít stejnou délku jako vybraný modul konfigurovaný ve STEPu 7.

U modulárních slave stanic je v daném okamžiku přístup pouze na jeden modul stanice.

**Příklad**



# Synchronizace DP Slave stanic přes SFC 11 (DPSYC_FR)




| Parametr | Deklarace | Typ | Oblast | Popis |
|---|---|---|---|---|
| REQ | INPUT | BOOL | I, Q, M, D, L, konst. | úrovňový start<br>REQ=1: funkce SYNC-/FREEZE aktivována |
| LADDR | INPUT | WORD | I, Q, M, D, L, konst. | logická adresa DP Master systému |
| GROUP | INPUT | BYTE | I, Q, M, D, L, konst. | Výběr skupiny<br>Bit 0 = 1: skupina 1 vybrána<br>Bit 1 = 1: skupina 2 vybrána ...<br>Bit 7 = 1: skupina 8 vybrána<br>Lze vybrat několik skupin najednou. |
| MODE | INPUT | BYTE | I, Q, M, D, L, konst. | ID úlohy (kódován v souladu s EN 50 170 V 3)<br>Bit 0, 1, 6, 7:reservováno (hodnota 0)<br>Bit 2 = 1: UNFREEZE je proveden<br>Bit 3 = 1: FREEZE je proveden<br>Bit 4 = 1: UNSYNC je proveden<br>Bit 5 = 1: SYNC je proveden |
| RET_VAL | OUTPUT | INT | I, Q, M, D, L | Chybový kód vyhodnocovaný v každém cyklu |
| BUSY | OUTPUT | BOOL | I, Q, M, D, L | BUSY=1: Úloha se provádí. |





**Popis**

SFC 11 "DPSYC_FR" umožňuje synchronizovat jednu nebo více skupin DP slave stanic. Lze vyslat jeden z následujících řídících příkazů:

- SYNC (simultánní výstup a "zmrznutí" výstupů na DP slave stanici.
- UNSYNC (zruší řídící příkaz SYNC)
- FREEZE (zmrznutí stavu vstupů a jejich načtení)
- UNFREEZE (zruší příkaz FREEZE)

**Předpoklady**

Před vysláním zmíněných příkazů musí být DP slave stanice rozděleny ve STEPu7 do skupin SYNCH nebo FREEZE.

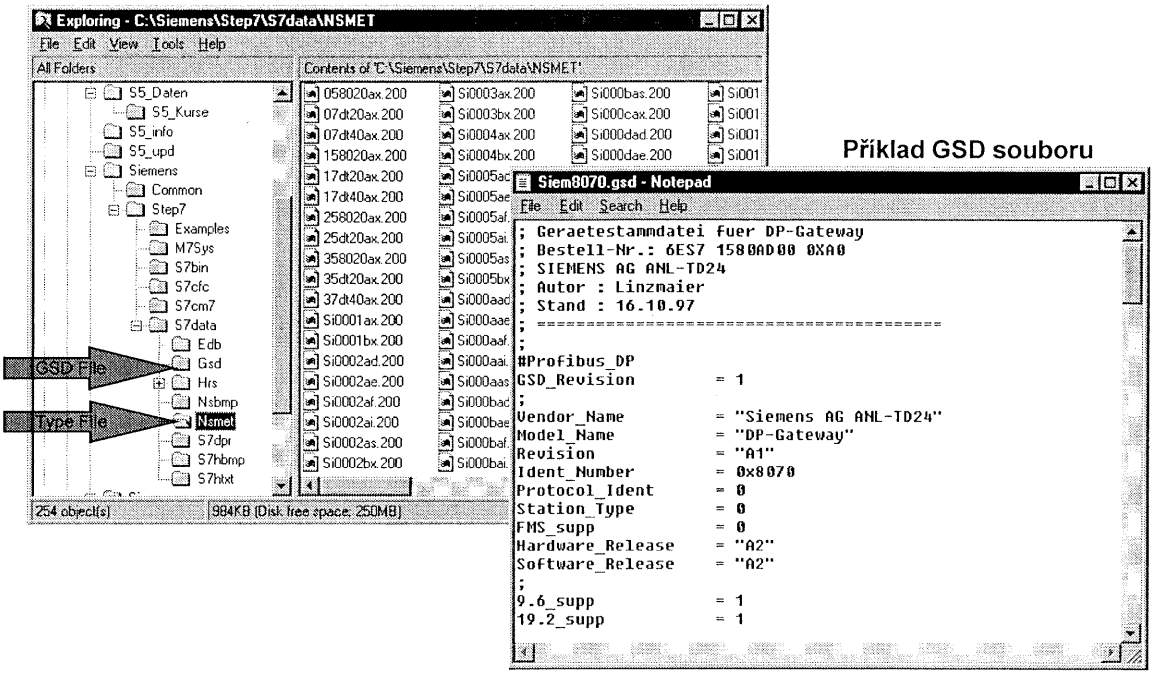
**Efekt SYNC**

Příkaz SYNC umožňuje přepnout definované skupiny do režimu *sync*, tj. DP Master odešle a "zmrazí" data na DP slave stanici. Následující nová data jsou zapsána do bufferu, ale výstupní hodnoty zůstanou nezměněny. Po každém příkazu jsou data z bufferu odeslána na výstupy najednou pro celou skupinu - simultánní aktualizace výstupních dat procesu.

**Efekt FREEZE**

Řídící příkaz FREEZE přepíná DP slave stanici do režimu *freeze*. Každý příkaz FREEZE uloží simultánně okamžitý stav vstupů a odešle ho do vstupní oblasti CPU.
Vstupy/ výstupy jsou cyklicky aktualizovány až po provedení příkazu UNSYNC/UNFREEZE.

# Pozdější instalace PROFIBUS modulů od jiného výrobce








**Typ souboru**

STEP 7 požaduje pro každý DP slave soubor DDB (device data base file) nebo typový soubor, aby mohl příslušnou slave stanici nabídnout v HW katalogu programu *HWConfig*.

Všechny vlastnosti DP slave stanice jsou uloženy v souboru DDB a odpovídají DP standardu. Typ souboru odpovídá specifikaci Siemens.

SIEMENS AG DP slave stanice mají definován každá vlastní typový soubor. Ostatní výrobci mají vlastní soubor DDB nebo typové soubory s vlastnostmi DP slave stanic.

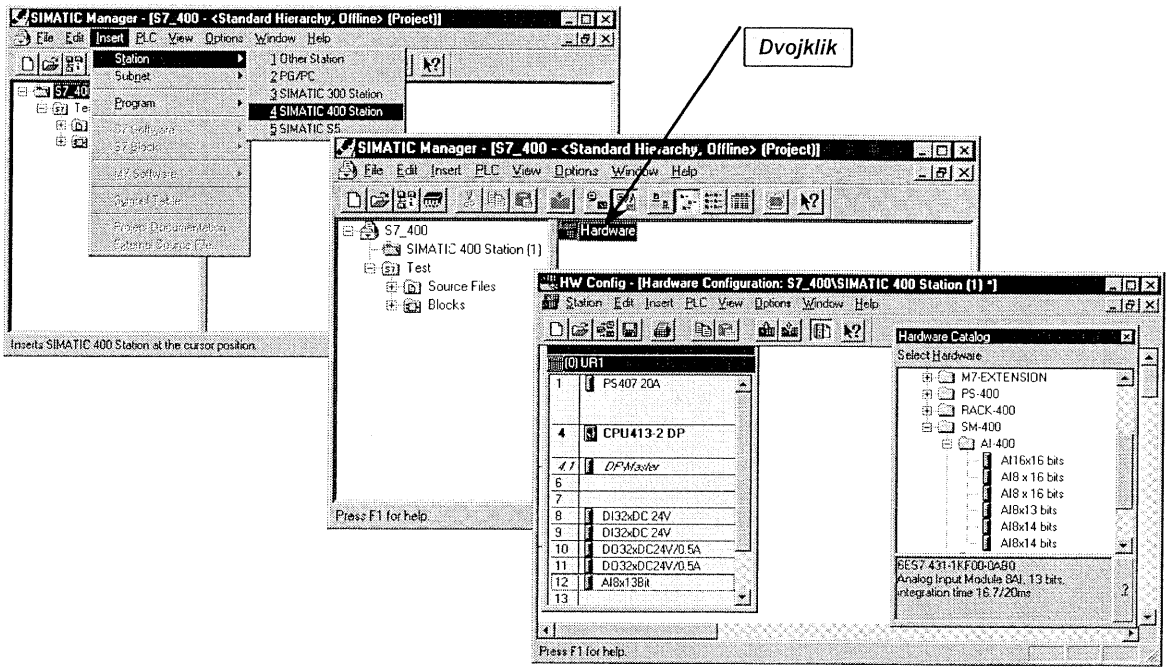
Pokud požadovaná DP slave stanice v HW katalogu chybí, je nutné doinstalovat odpovídající DDB nebo typový soubor do adresáře STEPu 7

**Instalace DP-Slave**

Nový DP-Slave lze instalovat následujícím postupem:

1. Po spuštění STEPu 7 je nutné zkopírovat příslušný GSD soubor do adresáře "..\STEP7\S7DATA\GSD.
2. Následná aktivace nabídky *Options -> Update DP Type Files* v HW Configu "přidá" novou stanici do skupiny "Additional Field Devices".

# Cvičení 12.1: Konfigurace modelu, parametrizace modulů







Datum: 24.11.2002
Soubor: PRO2_12cz.16



**Přehled**

SIMATIC Manager umožňuje správu programového a hardwarového vybavení S7. HWConfig umožňuje konfiguraci systému a parametrizaci modulů.

Vytvořenou konfiguraci lze potom nahrát do CPU. Při náběhu CPU tyto parametry přidělí jednotlivým modulům.

**Definice úlohy**

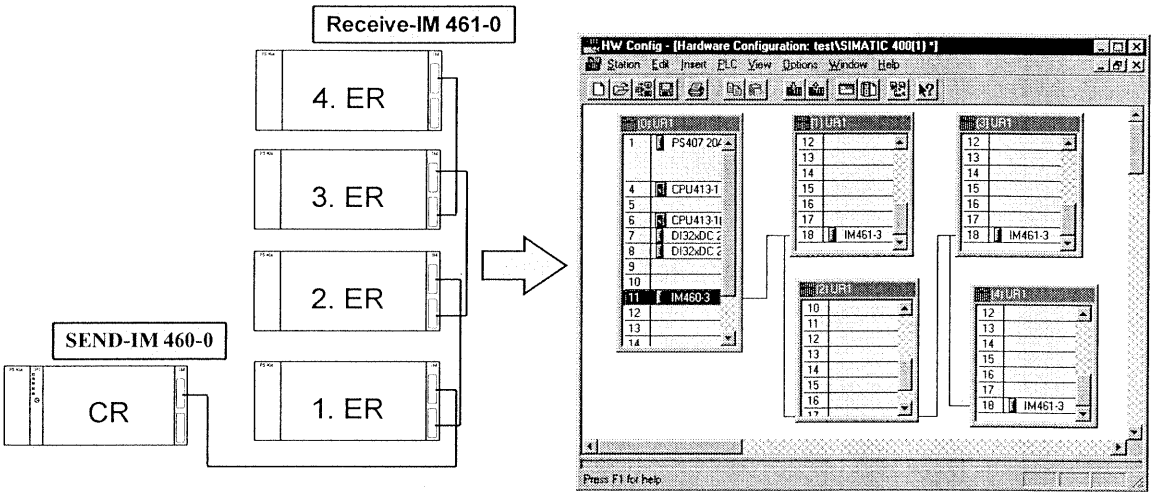
Založit v projektu stanici S7-400. Vytvořit a nahrát do CPU konfiguraci cvičné sestavy.

- digitální moduly adresovat od 0 do 7.
- adresa analogového modulu nastavit na 64.
- nastavit "taktovací" byte CPU na MB10.

**Postup**

1. vytvořit projekt "S7_400" a založit stanici (pomocí nabídky *Insert -> S7-400 Station).*
2. spustit program HW Config.
3. otevřít HW katalog a definovat nosič a jednotlivé moduly
4. definovat parametry CPU, vstupních a výstupních modulů
5. uložit konfiguraci.
6. nahrát tuto konfiguraci ve STOP stavu do CPU
7. provést studený start CPU.
8. pomocí funkce "Monitor/Modify Variables" zkontrolovat správnost nastavení parametrů I/O modulů.

# Cvičení 12.2: Tvorba rozšířené konfigurace







Datum: 24.11.2002
Soubor: PRO2_12cz.17



**Přehled** Pomocí HWConfigu konfigurovat komplexní systém s centrálním a několika rozšiřujícími nosiči.

**Definice** V projektu "S7-400" vytvořit stanici S7-400, která bude obsahovat jeden centrální nosič a 4 rozšiřující nosiče modulů (viz. obrázek).

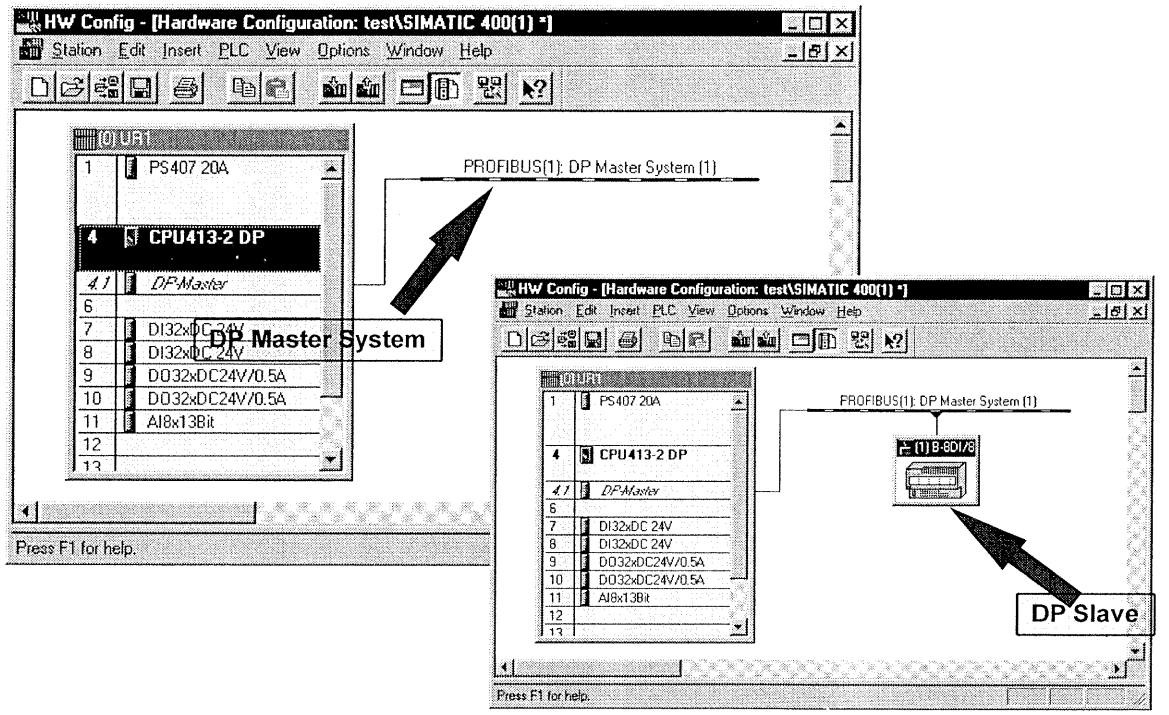
**Postup**

1. založit v projektu novou stanici S7-400 a otevřít její konfiguraci v HWConfigu
2. z HW katalogu vybrat požadované nosiče (UR1) a umístit je do okna stanice.
3. do centrálního nosiče vložit moduly podle konfigurace cvičné sestavy
4. do centrálního nosiče vložit modul Send-IM 460-0, do rozšiřujících nosičů vložit modul Receive-IM 461-0.

   Důležité: Před připojením rozšiřujících nosičů k SEND-IM musí být v rozšiřujících modulech založeny moduly RECEIVE-IM.
5. postupně vytvořit propojení mezi SEND-IM a RECEIVE-IM moduly jednotlivých nosičů:
   - dvojklik na Send-IM
   - vybrat kartu "Connection". Seznam obsahuje všechny nepřipojené nosiče.
   - označit jednotlivé nosiče a propojit je s žádaným SEND-IM přes tlačítko "Connect".

**Výsledek** V okně stanice se zobrazí odpovídající propojení mezi centrálním a rozšiřujícím nosičem modulů.

# Cvičení 12.3: Konfigurace Profibus-DP







Datum: 24.11.2002
Soubor: PRO2_12cz.18



**Přehled** HWConfig umožňuje také konfiguraci a parametrizaci DP slave stanic.

**Definice úlohy** V projektu "S7-400" vytvořit S7-400 stanici s CPU 413-2DP s integrovaným DP rozhraním. Jako slave stanici definovat a parametrizovat ET 200B s 8DI a 8DO.

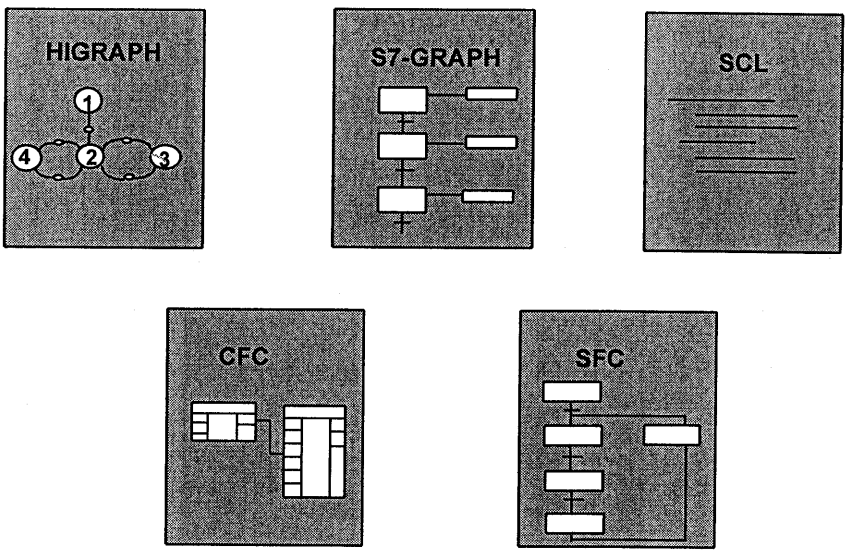
**Postup**

1. v HW Configu otevřít konfiguraci nově vytvořené stanice S7-400.
2. vybrat nosič UR1 a umístit ho do okna stanice.
3. do nosiče umístit CPU 414-2.

   Po umístění CPU se otevře okno "Properties - PROFIBUS Nodes" umožňující definici parametrů DP sítě.
4. potvrdit nastavení tlačítkem "OK". V okně stanice je zobrazen symbol sítě: ▬▬▬
5. vybrat z katalogu žádanou slave stanici (ET200B) a umístit ji do sítě.

   Je otevřeno okno "Properties - PROFIBUS Nodes" , kde lze definovat:
   - vlastnosti PROFIBUS sítě (rychlost, atd.).
   - adresu DP-slave stanice
6. potvrdit nastavení tlačítkem "OK". V konfigurační tabulce se zobrazí symbol představující příslušnou slave stanici.

# Vývojové nástroje pro S7/M7



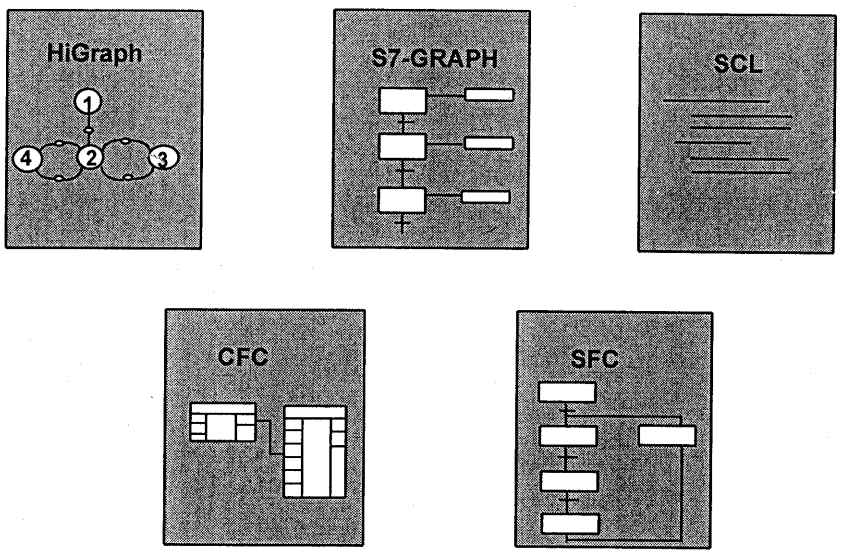

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.1

Školicí středisko
firmy E&A spol. s r.o., MB

**Obsah** **Strana**

# Vývojové nástroje pro S7/M7




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_13cz.2

Školicí středisko
firmy E&A spol. s r.o., MB

**Obsah** **Strana**

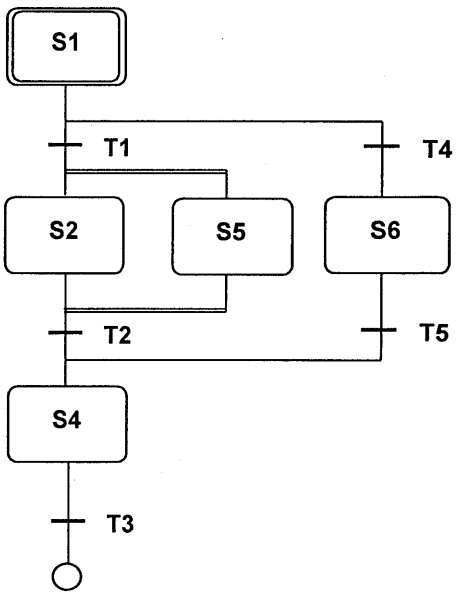
# S7-GRAPH (Sekvenční řídící systém)

**Definice:**

- **S7-GRAPH je nástroj podporující sekvenční řízení pro S7-300 (CPU 314 a vyšší) a S7- 400 podle normy IEC 1131-3.**

**Možnosti:**

- **S7-GRAPH usnadňuje následující fáze vývoje:**
  - **plánování, konfigurace**
  - **programování**
  - **odlaďování**
  - **údržba, diagnostika**
  - **změny a rozšiřování**




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_13cz.3

Školicí středisko
firmy E&A spol. s r.o., MB

**S7-GRAPH**

S7-GRAPH umožňuje rychlou a snadnou tvorbu a konfiguraci programových řetězců. Řízený proces je rozdělen do jednotlivých kroků, ve kterých se realizují akce a transicí - přechodových podmínek, které určují stavy, ve kterých lze akce provést. Celý řetězec je zobrazen v grafické podobě a lze ho doplnit o komentář.

Prováděné akce jsou definovány v krocích, transice řídí přechod z jednoho kroku do druhého. Obsah kroků a přechodových podmínek je definován pomocí standardních prostředků STEPu 7.

S7-GRAPH pro S7-300/400 je kompatibilní s programovacím jazykem definovaným v IEC 1131-3 standard.

**Vlastnosti**

S7-Graph podporuje následující funkce:

- více řetězců (max. 8) v jednom S7-GRAPH funkčním bloku
- volné číslování (1 až 999) kroků (max. 250 řetězci) a přechodových podmínek (max. 250)
- simultánní a alternativní větvení řetězce
- skoky (i do jiných řetězců)
- start/stop zpracování řetězce a aktivace a podržení kroku

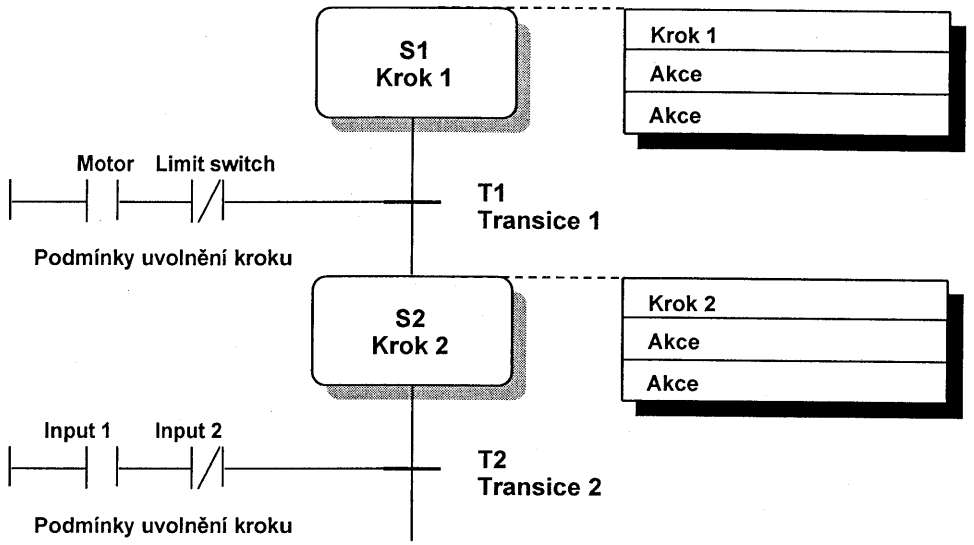
**Testovací funkce**

- zobrazení aktivního nebo chybujícího kroku
- Sledování a změna hodnot proměnných
- přepínání mezi režimy: manual, automat a krokování

**Uživatelské rozhraní**

- Zobrazení přehledové, jednotlivé stránky a jednotlivé kroky
- Graficky oddělené vazby(max. 32 podmínek), akce (max. 100 v jednom kroku) a kritické podmínky („supervision", max. 32 podmínek)

## Programová struktura sekvenčního řídícího systému




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_13cz.4

Školicí středisko
firmy E&A spol. s r.o., MB

**Přehled**

Sekvenčně řízený systém je systém, který po splnění potřebných podmínek přepne řetězec z jednoho kroku do kroku následujícího. Charakteristické pro tyto systémy je rozdělení řízené úlohy na

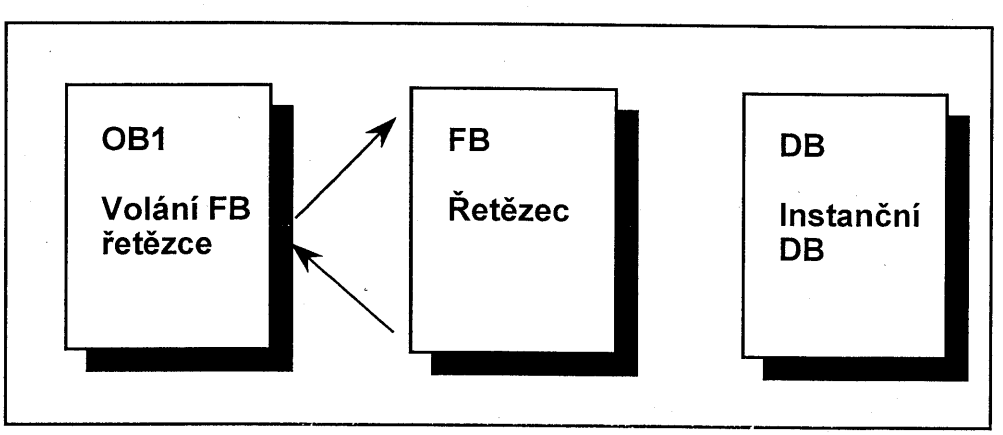
- Kroky a
- Transice (krok uvolňující podmínky)

**Řetězec**

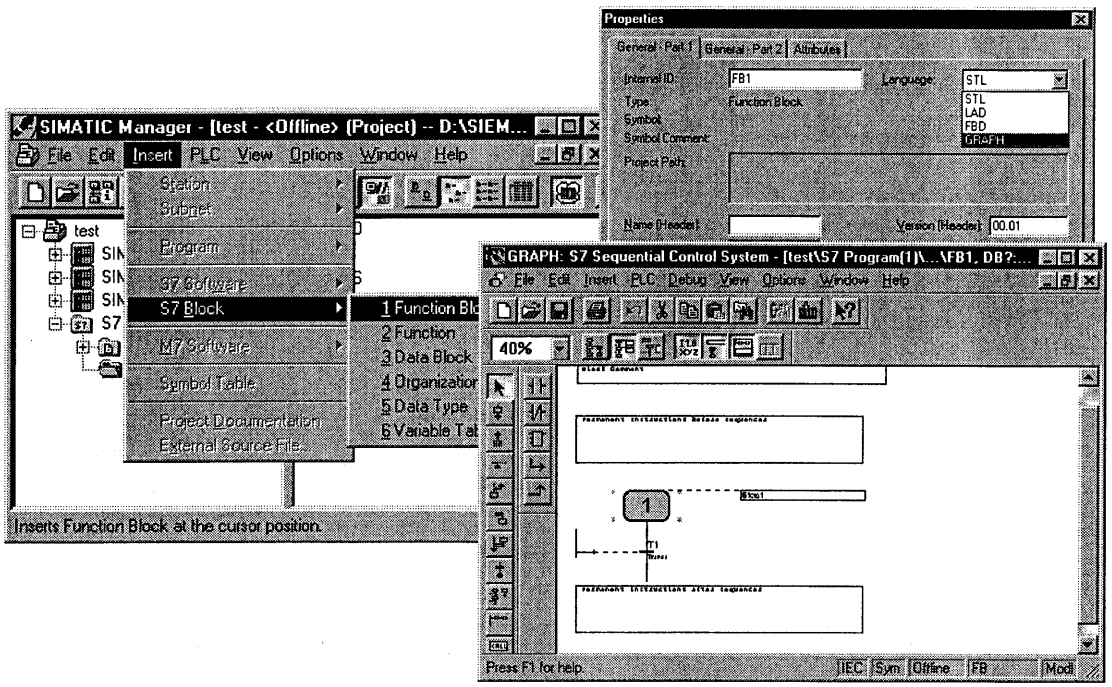
Kroky a transice tvoří dohromady řetězec. Tento řetězec je uložen v FB bloku a příslušný instanční DB obsahuje data pro řetězec.

Pro zpracování programu jsou nezbytné nejméně 3 bloky:

- FB blok, který obsahuje řetězec
- instanční DB
- OB, FB nebo FC, který realizuje volání FB s řetězcem Parametry a číslo instančního DB jsou předány při volání řetězce.



# Nastavení sekvenčního FB bloku




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_13cz.5

Školicí středisko
firmy E&A spol. s r.o., MB

**Přehled**

Založení nového S7-GRAPH řetězce lze provést buď z SIMATIC Manageru nebo z S7-GRAPH Editoru. V obou případech musí být nejprve otevřen odpovídající projekt v SIMATIC Manageru.

**Založení FB**

Založení S7-GRAPH řetězce v FB lze pomocí SIMATIC-Manageru provést následujícím způsobem:

1. otevřít uživatelský program ve kterém má být řetězec založen.
2. z hlavní nabídky aktivovat *Insert -> S7 Block -> Function Block*
3. ve vlastnostech FB zvolit programovací jazyk GRAPH a potvrdit "O.K."
4. Dvojklikem se spustí editace bloku v S7-GRAPH Editoru.

**S7-GRAPH Editor**

Obsahuje několik specifických funkcí přístupných přes nabídku *View -> Toolbar*:

Standard: obsahuje funkce pro práci se soubory (*Open, Save*, atd.) a pro editaci (*Copy, Insert*, atd.).

Sequencer: obsahuje funkce pro práci s prvky řetězce.

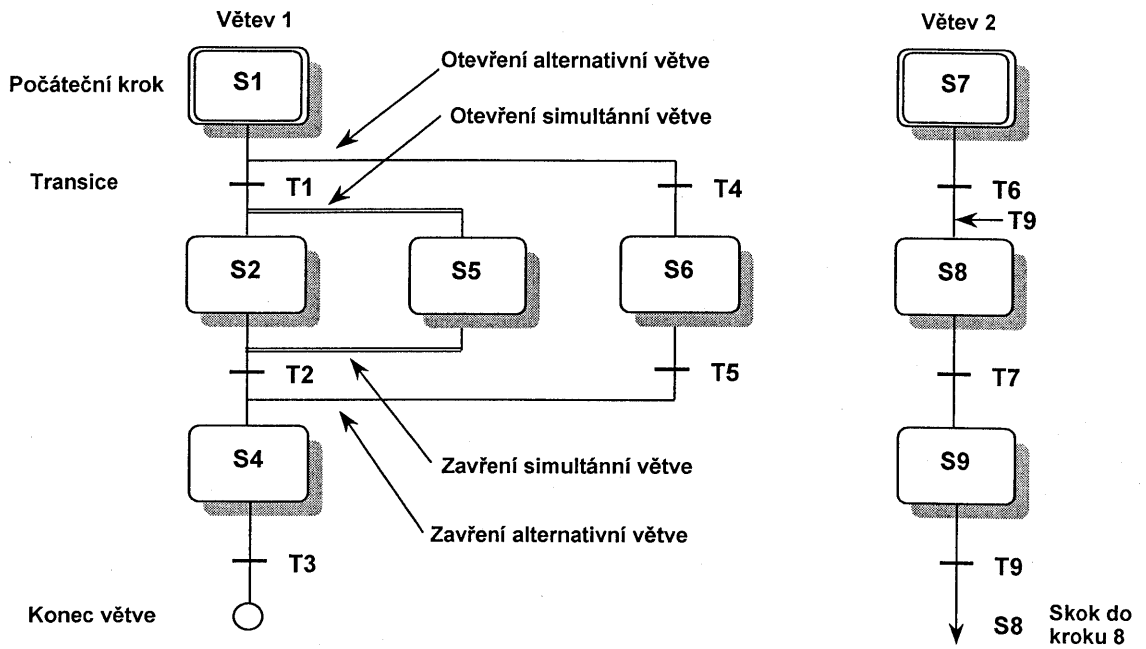
LAD: obsahuje funkce pro práci s LAD prvky v programu.

View: obsahuje funkce pro přepínání zobrazení

**Poznámka**

Nástrojovou lištu (*toolbar*) lze libovolně umisťovat po ploše S7-GRAPH Editoru.

## Prvky sequenceru




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_13cz.6

Školicí středisko
firmy E&A spol. s r.o., MB

**Přehled**

Při tvorbě řetězce lze s výhodou použít grafické programování. Jednoduchým umísťováním prvků řetězce lze tak vytvářet požadovaný řetězec.

**Struktura řetězce**

Řetězec je sled kroků a transic a může být buď lineární nebo větvený.

- Uvnitř kroků jsou formulovány instrukce pro řízení technologie.
- Transice obsahuje definici podmínek, které musí být splněny pro přechod z jednoho kroku do druhého.

Následující ikony v nástrojové liště umožňují vkládání jednotlivých prvků do řetězce :

 Pár krok/transice

 Skok

 Otevřít alternativní větev

 Větev zastavit

 Zavřít alternativní větev

 Otevřít simultánní větev

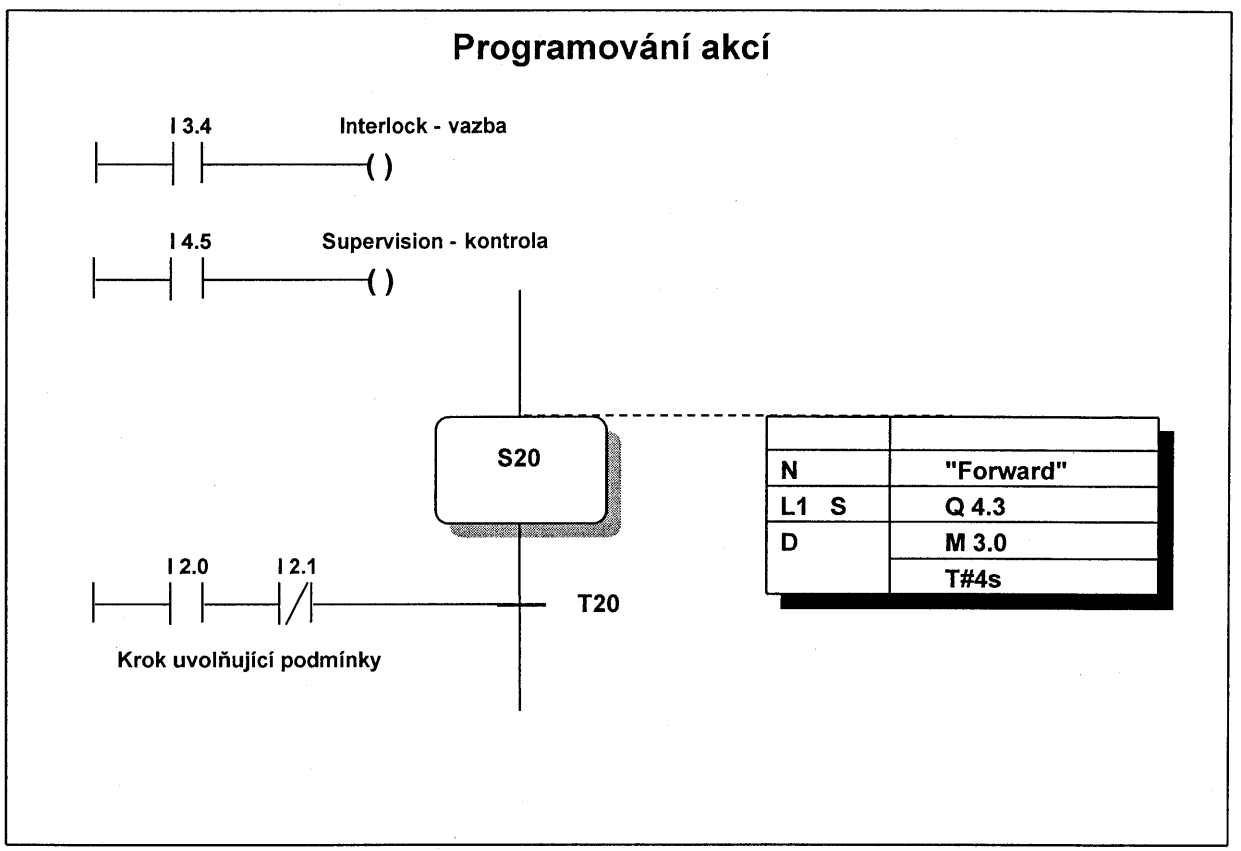
 Zavřít simultánní větev

 Vložit nový řetězec

**Pravidla**

Struktura sekvenčního řídícího systému - řetězce musí splňovat následující pravidla:

- Kroky a transice mohou být vkládány pouze v páru.
- Skákat lze v rámci jednoho řetězce nebo mezi řetězci
- Větev řetězce lze zastavit a deaktivovat její zpracování po tranzici.








**Přehled**

Akce jsou aktivní částí kroku, které systém provádí v aktivním kroku. Příslušné instrukce jsou programovány v krocích kde nastavují výstupy nebo aktivují nebo deaktivují kaskádové kroky. Akce obsahují instrukce a mohou být připojeny k podmínkám nebo kombinovány s událostmi.

**Programování**

Postup při programování je následující:

1. Vybrat tabulku vpravo od kroku, stisknout klávesu TAB.
2. Nyní lze programovat akce zadáním příslušných instrukcí, které mají být provedeny, do tabulky :

- každá instrukce zabírá právě jeden řádek v tabulce
- v levém sloupci je zadána instrukce, v pravém sloupci je definována adresa
- jestliže je použita instrukce požadující časovou informaci (D nebo L), S7-GRAPH automaticky nastaví v pravém sloupci dva řádky. Požadovaná časová informace se zadává do spodního řádku.

3. Rozšíření tabulky o další akci se provede opětovným stisknutím klávesy TAB.

Jestliže jsou použity instrukce logické kombinace s podmínkami (všechny instrukce obsahující písmeno "C"), je nutné programovat tyto podmínky v režimu zobrazení jednotlivých kroků jako vazby - *interlock.*

**Vazby**
***Interlocks***

Vazby jsou určeny k podmínečnému uvolnění akcí v kroku. Je-li podmínka splněna, jsou všechny akce s "C" provedeny.Přechod do dalšího kroku se provádí bez ohledu na stav vazby.

**Kontrola**
***Supervision***

Podmínky definované v *supervision* umožňují detekci chybné funkce a následnou reakci na nastalou chybu (zastavení řetězce a zpracování potvrzení chyby operátorem). Přechod do dalšího kroku je možný pouze po odeznění chyby.

## Standardní akce v kroku

- Blok akcí s jednoduchými instrukcemi

| Action block _1 | |
|---|---|
| N | M1.1 |
| S | M1.2 |
| R | M1.3 |
| D | M1.4 |
| | T#1H2M3S |
| L | M1.5 |
| | T#4MS |
| CALL | FC1 |

  - D = Čas zpoždění( delayed), časově zpožděné nepamatované přiřazení
    L = Čas omezení (limited), časově omezené nepamatované přiřazení
  - CALL = volání bloku



Datum: 24.11.2002
Soubor: PRO2_13cz.8



**Přehled** Tyto identifikátory instrukcí jsou definovány v normě Standard IEC 1131-3 část 2.6.4.

**D, L, N** Přidružené akce jsou resetovány po dokončení kroku.

**D, L** Časový údaj lze definovat jako konstantu nebo proměnnou.

**S, R** Odpovídající akce zůstává po dokončení kroku aktivní.

**CALL** Volání bloku FBi.DBi, FCi, SFBi.DBi, SFCi. Zpracování GRAPH-programu pokračuje po dokončení zpracování volaného bloku.

**Poznámka** Adresace operandů je možné provádět absolutně i symbolicky.

**Časový údaj** Časový údaj je zadáván v souladu s IEC časovým formátem:

T#mDnHoMpSqMS

mD: m Dnů
nH: n Hodin
oM: o Minut
pS:p Sec.
qMS q Millisec.

Maximální časový interval je cca. 24 D20H.

# Akce závislé na vnitřních vazbách

- **Blok akcí s podmíněnými instrukcemi**

| Action block _2 | |
|---|---|
| NC | M1.1 |
| SC | M1.2 |
| RC | M1.3 |
| DC | M1.4 |
| | T#1H2M3S |
| LC | M1.5 |
| | T#4MS |
| CALLC | FB5.DB3 |

- **Podmínky**
  - **Akce s identifikátorem "C" (Condition = podmínka) je provedena pouze tehdy, je-li splněna kroková podmínka ("C" = 1).**
  - **Je-li podmínka = "0" jedná se o chybu vazby a akce není provedena. Krok je označen jako chybový a je vytvořeno hlášení "Chyba".**





**Zvláštní případ** Podmíněná akce pro kterou není programována vazba je provedena nepodmíněně. Identifikátor "C" je v tomto případě ignorován.

**Příklad:**

| Instrukce | Adresa | Význam |
|---|---|---|
| NC | Q1.0 | Jak dlouho je krok aktivní a podmínka vazby je splněna tak dlouho je Q1.0 = 1, jinak je nulový. |
| SC | Q1.0 | Je-li krok aktivní a podmínka splněna je Q1.0 nasetován do 1 zůstává v tomto stavu |
| RC | Q1.0 | Je-li krok aktivní a podmínka splněna je Q1.0 resetován do 0 a zůstává v tomto stavu. |
| DC | Q1.0<br>T#<const> | Je-li krok aktivní i po uplynutí času T# a je splněna podmínka , je Q1.0 = 1. Není-li krok aktivní nebo podmínka není splněna, je Q1.0=0. |
| LC | Q1.0<br>T#<const> | Je-li krok aktivní a podmínka splněna, je po dobu T# výstup Q1.0 =1. Není.li krok aktivní je Q1.0=0. |
| CALLC | FB5.DB3 | Je-li podmínka splněna a krok aktivní je volán příslušný blok. Zpracování GRAPH-programu pokračuje po ukončení volaného bloku. |

## Akce spouštěné událostí

- Blok akcí s událostními instrukcemi

| Action block _3 | |
|---|---|
| A1 N | M1.1 |
| L1 N | M1.2 |
| L0 N | M1.3 |
| S1 N | M1.4 |
| S0 N | M2.4 |
| V1 N | M2.5 |
| V0 N | M2.6 |

  - A1 = potvrzení
  - L1 = vazba,příchozí chyba vazby
  - L0 = vazba, odchozí chyba vazby
  - S1 = krok, krok aktivován
  - S0 = krok, krok deaktivován
  - V1 = kontrola, příchozí monitorovaná chyba
  - V0 = kontrola, odchozí monitorovaná chyba





**Přehled**

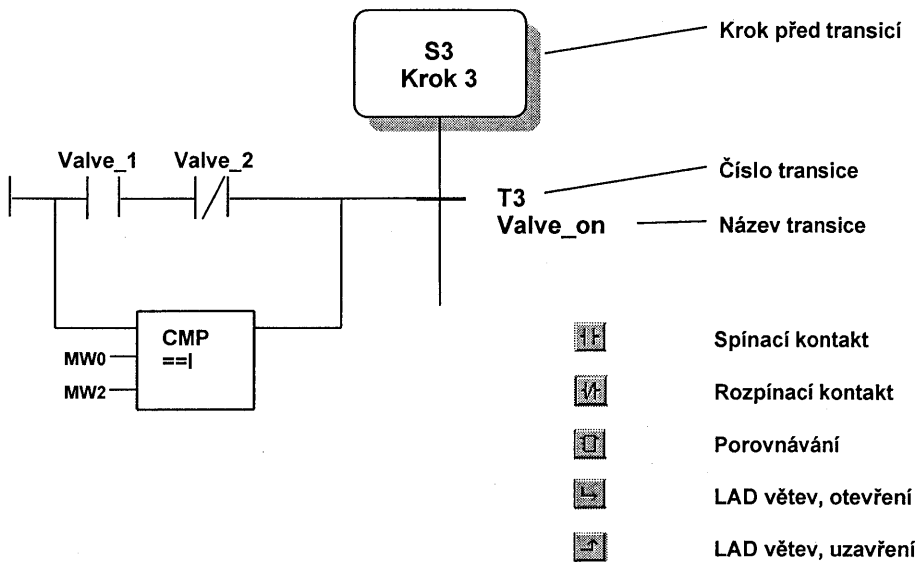
Událost lze detekovat a propojit s žádanou akcí. Monitorovat a ovlivňovat lze nejenom jednotlivé kroky ale i celý sekvenční řídící systém

**Aktivace a deaktivace kroků**

Pomocí instrukcí ON a OFF lze aktivovat a deaktivovat jiné kroky řetězce.

| Událost | Instrukce | Adresa | Popis |
|---|---|---|---|
| S1<br>S0<br>V1<br>V0 | ON<br><br>OFF | Si<br><br>i=číslo kroku | Závisí na události (de)aktivující krok |
| L1<br>L0<br>A1 | OFF | S_ALL | Závisí na události (de)aktivující všechny kroky, vyjímku tvoří krok obsahující akci |

# Kontrola podmínek v tranzicích, vazbách a kontrolách




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_13cz.11

Školicí středisko
firmy E&A spol. s r.o., MB

**Přehled**

Při programování podmínek je k dispozici LAD segment. Každou vazbu, tranzici a kontrolu lze programovat ve formě LAD segmentu.

**Podmínky**

S7-GRAPH rozlišuje následující typy podmínek:

Transice: popisuje podmínky, které umožňují přechod z jednoho kroku do druhého.

Trvalé podmínky: Trvalé podmínky jsou umístěny před nebo za řetězec a jsou vyhodnocovány jednou za cyklus.

Kontrola: popisuje podmínky, které vedou k signalizaci chyby kontroly a zabraňují přechodu do dalšího kroku.

Vazba: popisuje podmínky, jejichž splnění je nutné pro provedení akce. Není-li podmínka splněna, je signalizována chyba vazby.

**Prvky LAD**

Ze základních prvků LAD jsou dostupné následující: spínací kontakt, rozpínací kontakt, porovnávání, otevření a zavření LAD větve (logický OR).

**Editace**

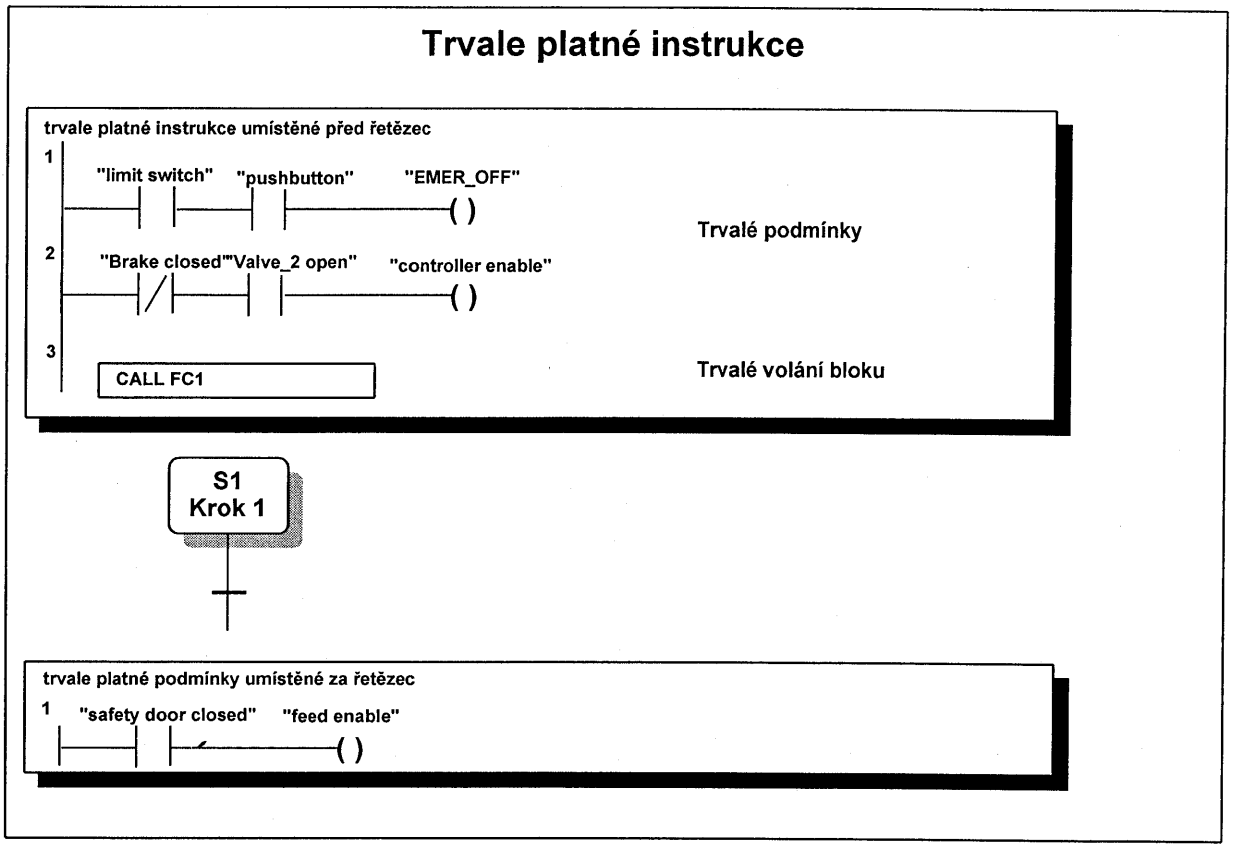
Nejjednodušší cestou programování podmínek je režim zobrazení jediného kroku. Zbylé dva režimy slouží pro vkládání prvků LAD. Výběr režimu se provádí z nabídky *Options -> Customize*:

**DRAG & DROP**

Umístění prvku se provede přetažením žádaného prvku z nástrojové lišty do žádané polohy.

**Přidání (ADD)**

Výběr prvku kontrolní podmínky je třeba provést před vložením nového prvku.







Datum: 24.11.2002
Soubor: PRO2_13cz.12



**Přehled**

Trvale platné instrukce jsou podmínky nebo volání bloků umístěné před nebo za řetězec a jsou zpracovávány jednou za cyklus.

S použitím trvalých instrukcí lze např.:

- volat bloky obsahující STL, LAD, FBD nebo SCL instrukce přímo z programu S7-GRAPH nebo
- programovat podmínky platné ve více krocích (musí být programovány pouze jednou)

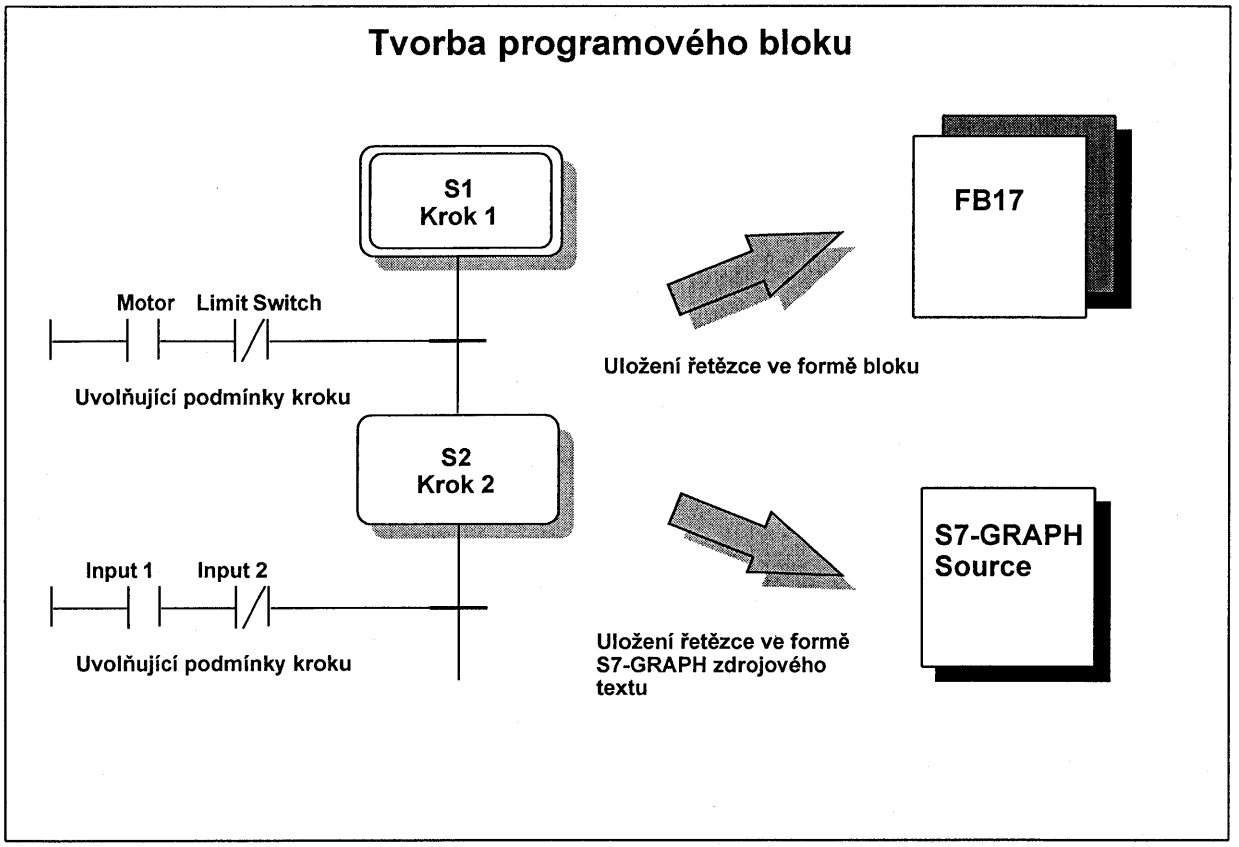
Počet trvalých instrukcí není omezen, je dán pouze potřebami uživatele.

**Trvalé instrukce**

Trvale platné instrukce jsou umístěny před a za řetězcem a jsou vyhodnocovány jednou za cyklus.
Postup:

1. Z nabídky vybrat *View -> Single Page*, nebo zvolit odpovídající ikonu v nástrojové liště.
2. Z nabídky zvolit *View -> Permanent Instructions*.
3. Vybrat *Insert -> Permanent Instruction -> Condition* nebo *Call*, nebo aktivovat odpovídající ikonu. Ukazatel myši se změní na LAD segment nebo CALL instrukci.
4. Umístit LAD segment nebo instrukci CALL před nebo za řetězec.
5. Stisknout levé tlačítko myši nebo opakovaně zvolit odpovídající nabídku nebo ikonu v nástrojové liště.
6. Jedná-li se o podmínky lze je nyní editovat použitím dostupných prvků LAD.

   Jedná-li se o volání bloku lze nyní doplnit název bloku a definovat aktuální parametry.








**Přehled**

Při ukládání S7-GRAPH funkčního bloku je automaticky spuštěn překlad.

Uložit lze pouze bezchybně přeložený blok. Pokud je při překladu zjištěna chyba, blok není uložen a je zobrazeno příslušné chybové hlášení.

Pokud je potřeba uložit blok obsahující chyby, lze použít nabídku *File -> Generate Source*.

**Volby**

S7-GRAPH lze přizpůsobit potřebám uživatele pomocí nabídky *Options -> Customize*:

- parametry dostupné při volání S7-GRAPH FB (minimální, standardní nebo maximální sada parametrů)
- zobrazení hlášení při detekci chyby během překladu v okně hlášení
- způsob uložení kroků a transic v instančním DB (v poli struktur nebo jako samostatné struktury)
- zda při ukládání analytických dat do instančního DB a aktivované funkci *Skip Mode* a současně detekované chybě kontroly musí být tato chyba potvrzována.
- zda se chyby vazeb a kontrol mají také zpracovávat pomocí SFC 52 (diagnostický buffer) nebo pomocí SFC 17, 18 (vysílání na HMI stanici).

**Instanční DB**

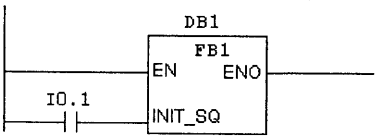
Uživatel má k dispozici dvě cesty, jak vytvořit instanční DB:

- je -li aktivována volba *Include Instance DB Automatically*, je uživatel v okamžiku ukládání FB vyzván k definování čísla instančního DB.
- není-li tato volba aktivována, lze instanční DB vytvořit z nabídky *File -> Create Instance DB*.

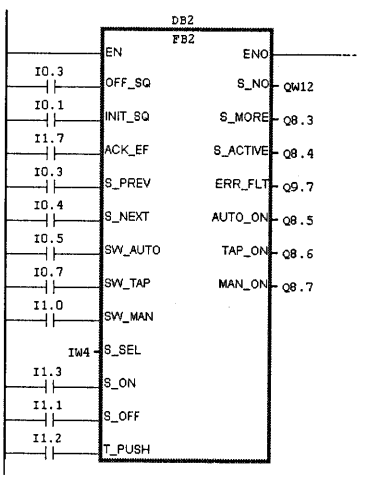
# Integrace FB do OB1

## ❑ Volání S7-Graph FB bloku

- ○ **Minimální parametry (default)**
  - 1 vstupní parametr pro řízení řetězce
- ○ **Standardní sada parametrů**
  - 12 vstupních parametrů pro řízení
  - 7 výstupních parametrů pro zobrazení režimu činnosti
- ○ **Maximální sada parametrů**
  - 17 vstupních parametrů pro řízení řetězce
  - 12 výstupních parametrů pro zobrazování




**Minimum parameter set**




**Standard parameter set**




Datum: 24.11.2002
Soubor: PRO2_13cz.14



**Sekvenční FB**

Pomocí S7-GRAPHu lze vytvořit sekvenční FB (včetně instančního DB), který je spustitelný na CPU 314 a CPU 4xx. Žádné další standardní FB bloky nejsou nutné. Jednotlivé funkce lze aktivovat pomocí parametrů vytvořeného FB a jsou zobrazovány v instančním DB.

Z nabídky *Options -> Customize* lze na kartě *Compile* definovat počet parametrů výsledného FB bloku.

FB reaguje na náběžné hrany signálů.

**Minimální parametry (Default)**

V minimální konfiguraci má FB pouze jeden vstupní parametr INIT_SQ, Řetězec je bezprostředně zpracováván v automatu.

Počáteční krok je aktivován náběžnou hranou INIT_SQ.

**Standardní parametry**

Rozšířené možnosti umožňují nastavení režimu činnosti. Nastavení jsou akceptována s náběžnou hranou signálů.

Řetězec zachovává poslední aktivovaný režim činnosti. Režim činnosti je změněn pouze při aktivaci jiného režimu činnosti. Parametry, které nejsou požadovány zůstávají nepoužity.

**Maximální parametry**

V této variantě jsou zobrazeny a mohou být přiřazeny všechny parametry volání FB řetězce.

## Aktivace testovací funkce

- **Postup**
  - **Nahrání sekvenčního FB a instančního DB**
    - Nahrání FB (a instančního DB) s řetězcem lze provést z nabídky ***PLC -> Download***
  - **Výběr instančního DB:**
    - Pro testování je nutné nejprve vybrat příslušný instanční DB pomocí nabídky ***Debug -> Test Environment***
  - **Spuštění funkce "Monitor":**
    - Vybrat žádanou oblast řetězce ( následně bude zobrazena stavová informace oblasti otevřené v okně).
    - Aktivovat nabídku ***Debug -> Monitor***
  - **Ukončení funkce "Monitor":**
    - Deaktivovat nabídku ***Debug -> Monitor***

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.15

Školicí středisko
firmy E&A spol. s r.o., MB

**Testovací funkce**

S7-GRAPH řetězec lze zpracovávat v testovacím režimu. V tomto režimu jsou zobrazeny stavy jednotlivých prvků řetězce a hodnoty jednotlivých adres.

**Nahrání do PLC**

Je-li nutné nahrávat do PLC všechny bloky programu, je nutné provést tuto operaci ve STOP stavu CPU, protože v RUN stavu může dojít k chybovým stavům.

V pořadí, v jakém mají být jednotlivé FB nahrány do PLC ( spolu s příslušným instančním DB) provést následující kroky:

1. S otevřeným FB aktivovat *PLC -> Download.*
2. V dialogovém okně "Download" vybrat instanční DB příslušející k otevřenému FB
3. Potvrdit nezbytná hlášení .

**Aktivace monitorování**

Tato funkce je prováděna paralelně pro všechna otevřená okna stejného sekvenčního FB. Jestli že je požadována aktualizace velkého množství stavových údajů, je zobrazeno varování, že v danou chvíli nelze aktualizovat všechny stavové informace synchronně. Po potvrzení tohoto hlášení je řetězec zobrazen v režimu *Status.*

**Deaktivace monitorování**

Sledování stavu řetězce musí být vždy vypnuto pokud se provádí změny ve struktuře řetězce nebo v obsahu jednotlivých kroků či tranzic. Provádí-li se úpravy řetězce, je výhodnější tyto změny nejprve uložit na harddisk a teprve potom je nahrávat do CPU.

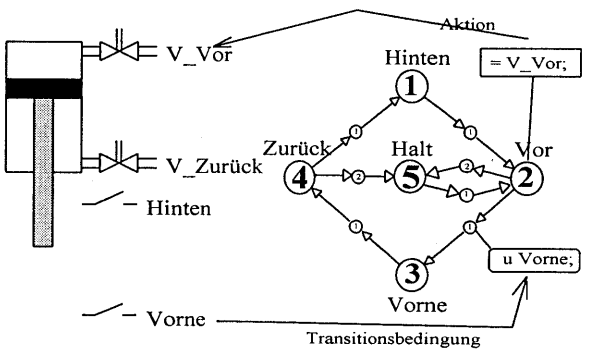
## Doplněk S7-HiGraph

**Definice:**

- **S7-HiGraph: nástroj pro programování stavových diagramů pro S7-300 (od CPU 314) a S7-400.**

**Výhody:**

- **S7-HiGraph umožňuje optimalizaci těchto fází automatizace procesu:**
  - **Plánování, konfigurace**
  - **Programování**
  - **Odladění**
  - **Údržba, diagnostika**
  - **Změny a úpravy**




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_13cz.16



**Proč stavové diagramy**

Řízení mechanických strojů, které je důležitou oblastí automatizace, se skládá z sekvenčního řetězce jednotlivých pohybů.
Běžné prostředky (STL, LAD, FBD) nebo sekvenční programovací jazyky (GRAPH5, S7-GRAPH, atd.) se v této oblasti běžně používají. Tyto programovací jazyky ale neumožňují vazbu mezi programátorem, který definuje řídící algoritmus a konstruktérem mechanických částí stroje.

Výsledkem je, že konstruktér elektrických částí a mechanických částí používají různé prostředky a vzájemná výměna informací je proto obtížná.

Cílem VDW (Association of German Machine Tool Manufacturers) bylo vytvoření univerzálního CASE programovacího prostředku, který by byl použitelný ve všech oblastech při vývoji a výrobě mechanických strojů.

Výsledkem činnosti VDW je definice popisu technologie pomocí metody stavových diagramů. Provádí se pomocí programové nadstavby S7-HiGraph a tento způsob programování je použitelný u řídících systémů S7-300 (od CPU 314) a S7-400.

**Výhody**

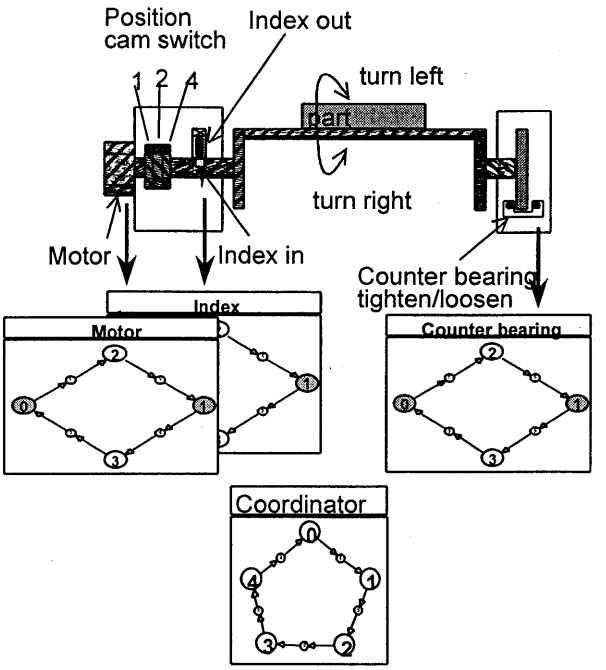
Tato "objektově-orientovaná" metoda popisu problému je dobře použitelná :

- konstruktéry mechanických částí stroje
- konstruktéry elektrických částí
- pracovníky revize i údržby.

Metoda popisu pomocí stavových diagramů tak zkracuje dobu vývoje strojů a jejich revizi.

# Princip metody stavových diagramů

- **Příklad: Otočný stůl pro frézu**
  - **Funkční jednotky (FU)**
    - Motor
    - Index
    - Čítač
  - **Stavové diagramy**
    - jeden diagram pro každou FU
    - navíc jeden koordinační diagram








**Metoda stavových diagramů**

Metoda stavových diagramů se zaměřuje na popis chování "objektů" reálného světa. Stroje nebo systémy, které mají být zautomatizovány, chápe jako kombinaci nezávislých prvků, funkčních jednotek, které jsou tvořeny kombinací mechanických a elektrických základních prvků.

**Funkční jednotky**

Funkční jednotka (FU) je nejmenší mechanická část stroje nebo systému. Ke každé funkční jednotce je definován stavový diagram, který popisuje elektro-mechanické chování této jednotky.

Automatizované objekty jsou rozděleny na jednotlivé mechanické funkční jednotky popsané metodou stavových diagramů.

**Stavový diagram**

Stavový diagram popisuje dynamické chování funkční jednotky. Toto chování je chápáno jako přechod funkční jednotky z jednoho stavu do druhého.

Funkce stroje nebo systému je chápána jako kombinace paralelně zpracovávaných stavových diagramů.

**Stavy a přechod**

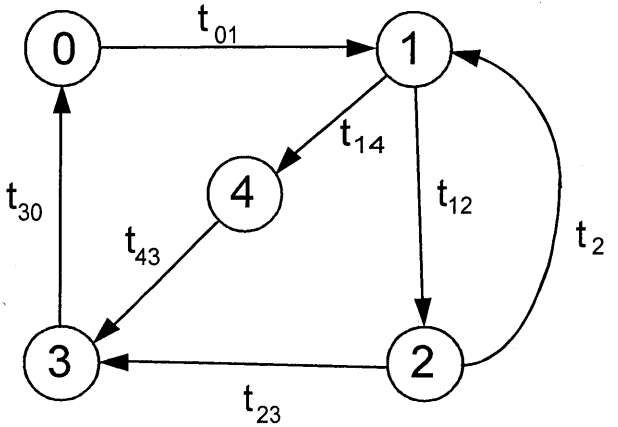
Ve stavovém diagramu představuje kroužek stav FU. Šipka představuje přechod z jednoho stavu do druhého. Přechod do druhého stavu je uskutečněn, jsou-li podmínky definované v tranzici splněny.

**Hlášení**

Koordinace a synchronizace většího počtu FU mezi sebou se provádí pomocí výměny hlášení.

## Prvky stavového diagramu

- **Stavy 0,1, ...**
  - zobrazovány jako kruh
  - Statické stavy
  - Pohyblivé stavy
  - Vždy je aktivní nějaký stav
  - Akce jsou definovány v daném stavu
- **Transice $t_{01}$, ...**
  - zobrazovány jako šipka
  - přechodové podmínky jsou definovány v transicích






Datum: 24.11.2002
Soubor: PRO2_13cz.18



**Stavový graf** Stavový graf zobrazuje všechny stavy FU jako kruh a transice mezi nimi jako šipky.

**Stavy** Stavy funkční jednotky mohou být statické (Dveře_otevřeny, Motor_vypnut, atd.) nebo dynamické, tj., pohyblivé stavy (Dveře_otevírány, Motor_doleva, atd).

V daném okamžiku může být subsystém, popsaný stavovým grafem, právě v jednom stavu.

Každý stavový graf musí mít definován stav s číslem 0 - počáteční stav.

**Akce** Akce mohou být přiřazeny stavům. Tyto akce lze rozdělit do tří skupin:

- Akce prováděné jednou - při vstupu/dosažení nového stavu.
- Akce prováděné cyklicky, po dobu, po kterou je FU v daném stavu.
- Akce prováděné jednou - při opuštění starého stavu.

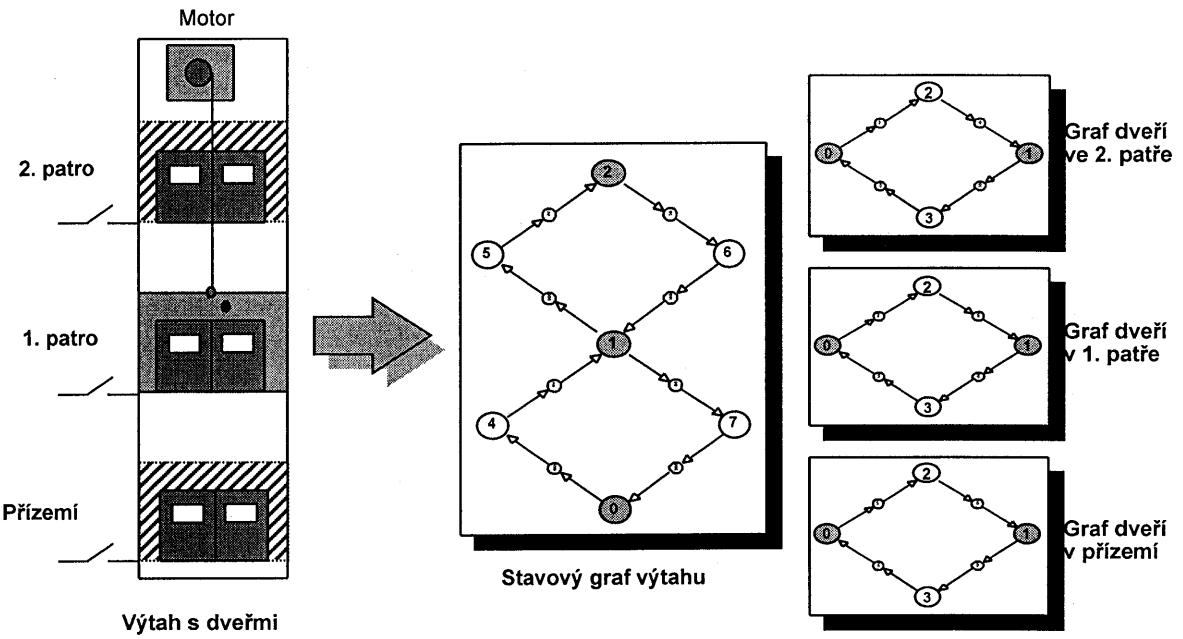
Obsah akcí je definován pomocí jazyka, který je podobný jazyku STL.

**Transice** Transice identifikují změnu FU z jednoho stavu do druhého. Tyto transice mezi stavy jsou závislé na podmínkách, které jsou nebo nejsou v daném okamžiku splněny.

Subsystém popsaný stavovým grafem mění svůj stav, jestli že jsou podmínky definované v transici splněny. V transici nelze provádět žádné akce.

Podmínky transic jsou definovány jazykem podobným STL.

# Příklad: Stavový diagram pro řízení výtahu




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_13cz.19



**Přehled**

Příklad ukazuje popis výtahu ve třípodlažní budově pomocí metody stavových grafů.

**Funkční jednotky**

Automatizovaný objekt (výtah s dveřmi) lze rozdělit do následujících funkčních jednotek s odpovídajícím stavovým grafem :

- klec výtahu
- dveře v každém patře.

**Stavový graf klece výtahu**

Ve stavovém grafu představují "kroužky" možné stavy FU a šipka představují přechody mezi jednotlivými stavy.

FU "řízení klece výtahu" má definovány stavy 0 ... 8. Stavy 0, 1 a 2 představují stojící výtah v každém patře.

Stavy 3, 4, 5, 6, 7 a 8 představují dynamický pohyb nahoru nebo dolu mezi jednotlivými patry.

**Stavové grafy dveří**

Stavy 0 a 1 funkční jednotky "Dveře" představují statický stav "dveře otevřeny/zavřeny", stavy 2 a 3 představují dynamické stavy "dveře se otevírají/zavírají".

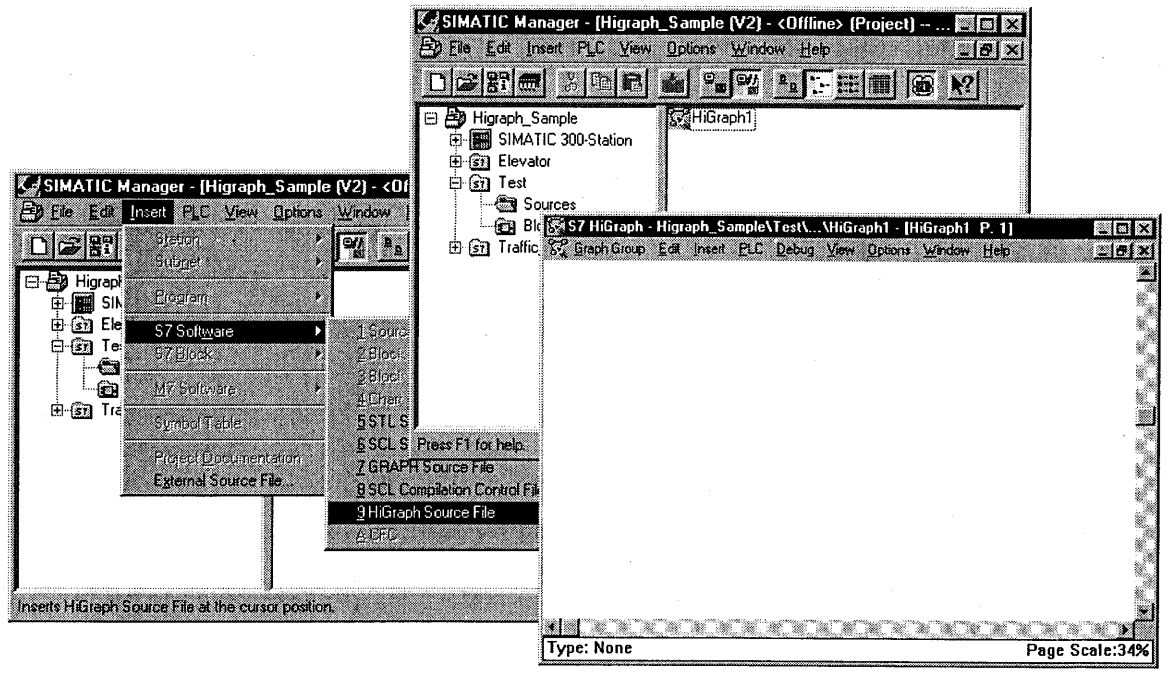
**Hlášení**

Koordinace nebo synchronizace jednotlivých stavových grafů dveří a klece výtahu lze principielně realizovat bez pomocí koordinačního stavového grafu.

Po dosažení žádaného patra vyšle stavový graf klece výtahu hlášení "Otevřít dveře" příslušnému stavovému grafu dveří.

Tento stavový graf přijme hlášení a "otevře" dveře. Po zavření dveří vyšle stavový graf dveří hlášení "dveře zavřeny". Stavový graf klece toto hlášení přijme a výtah se může rozjet do dalšího patra.

# Nastavení zdroje HiGraph







Datum: 24.11.2002
Soubor: PRO2_13cz.20



**Přehled**

Pro práci s S7-HiGraph stavovými grafy je nutné mít otevřený v Simatic Manageru S7 projekt.

**Editace zdrojového textu**

Nastavení S7-HiGraph zdroje v SIMATIC Manageru lze provést následovně:

1. Otevřít složku do které má být HiGraph zdroj vložen
2. Aktivovat nabídku *Insert -> S7-Software -> HiGraph Source.*

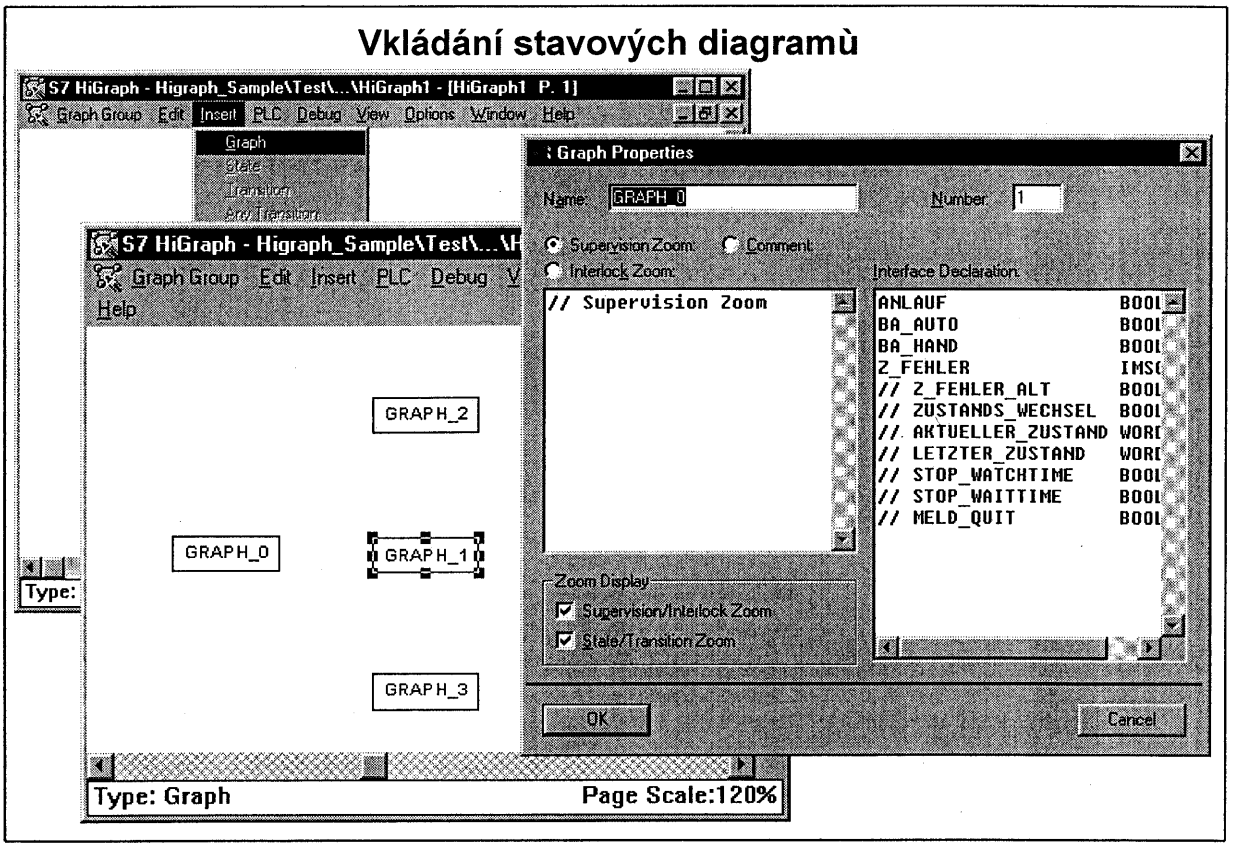
   HiGraph zdrojový soubor je založen pod standardním jménem ve složce *Sources*. Před další editací souboru lze jméno předefinovat.
3. Dvojklikem na objektu zdrojového souboru se spustí HiGraph Editor a otevře se požadovaný soubor.

**Aplikační okno**

Skupinové okno grafů zobrazuje aktuální strukturu stavových grafů včetně vyměňovaných hlášení.

Každý stavový graf je ve skupinovém okně zobrazen jako čtverec, každé hlášení jako šipka.

# Vkládání stavových diagramù







Datum: 24.11.2002
Soubor: PRO2_13cz.21



**Přehled**

Nabídka *Insert* obsahuje funkce pro tvorbu a editaci grafů, stavů a transic. V závislosti na otevřeném okně jsou dostupné pouze přípustné prvky, které lze do příslušného okna vložit.

**Vložení stavového grafu**

Pro vložení nového stavového grafu je nutné provést následující:

1. Aktivovat nabídku *Insert -> Graph*.
   Automaticky se aktivuje skupinové okno a kursor změní podobu na čtverec.
2. Umístit kursor do požadované polohy a kliknout levým tlačítkem myši.

V požadovaném místě je vložen čtverec stavového grafu se standardním jménem GRAPH_0 atd. Opakováním lze takto vložit všechny požadované stavové grafy do skupinového okna.

3. Vkládací režim se ukončí kliknutím pravého tlačítka myši. Kursor změní tvar na kříž.

**Poznámka**

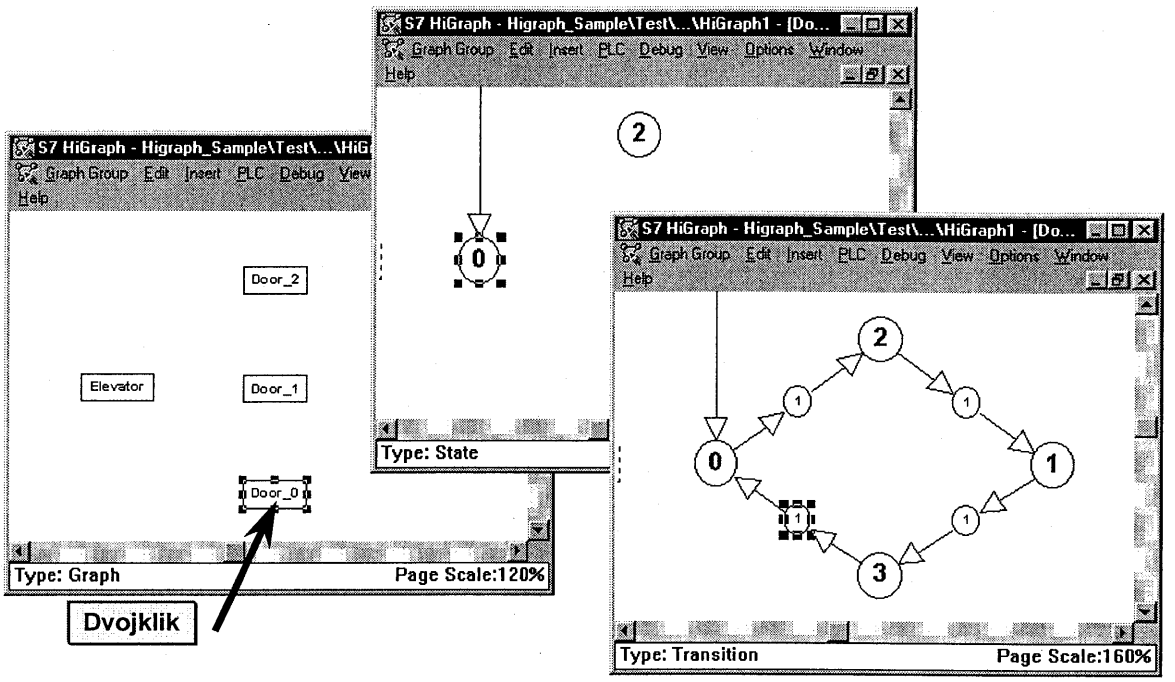
Nabídka *Options -> Align Horizontally/Vertically* umožňuje zarovnání pozice jednotlivých symbolů grafů.

**Změna názvu grafu**

Pro provedení změny standardního názvu grafu je potřeba provést následující kroky:

1. myší vybrat symbol žádaného grafu.
2. pomocí nabídky *Edit -> Graph Properties* otevřít dialog "Graph Properties".
3. v poli *Name* definovat požadovaný název grafu a potvrdit tuto volbu tlačítkem OK.

# Vkládání stavů a přechodů







Datum: 24.11.2002
Soubor: PRO2_13cz.22



**Výběr editačního okna grafu**

Otevření editačního okna grafu lze provést dvojklikem na objektu grafu.

**Vložení stavu**

Vložení nového stavu do stav. grafu lze provést následujícím způsobem:

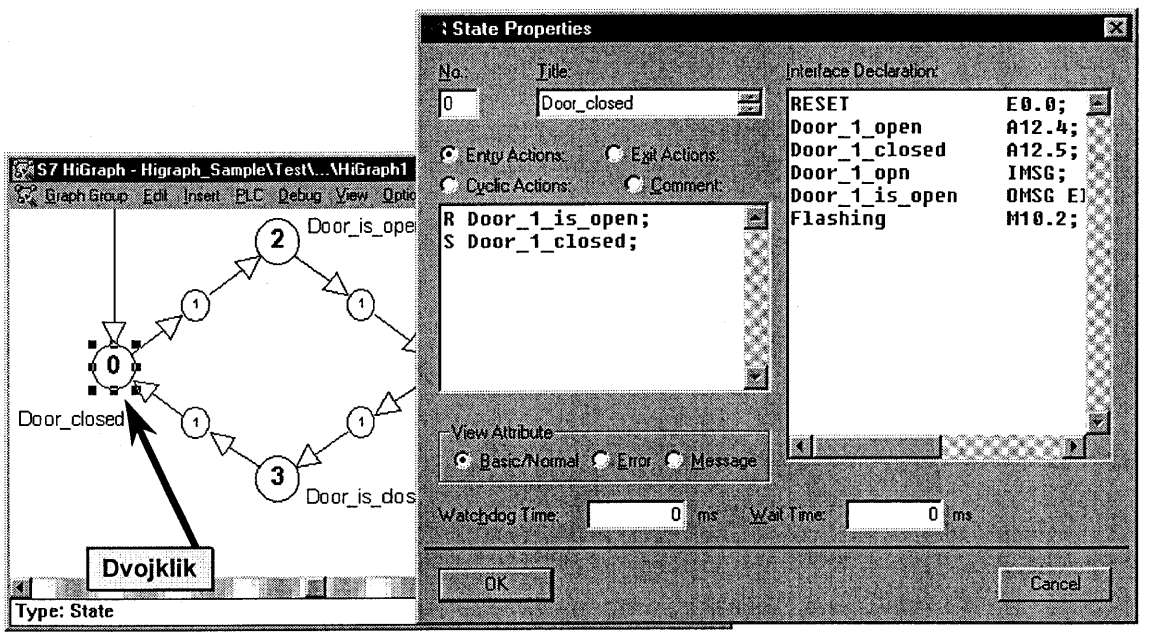
1. ve zvoleném editačním okně vybrat nabídku *Insert -> State*. Kurzor změní svůj tvar na kruh.
2. umístit kurzor do zvolené polohy a stisknout levé tlačítko myši.
   Ve zvolené pozici je vytvořen kruh s automaticky definovaným číslem 0, 1, 2, atd. Opakováním uvedeného postupu lze definovat všechny potřebné stavy stavového grafu.
3. ukončení vkládacího režimu se provede stisknutím pravého tlačítka myši. Kurzor opět změní svůj tvar na kříž.

**Vložení transice**

Vložení transice - přechodové podmínky se provede následujícím způsobem:

1. aktivovat nabídku *Insert -> Transition*. Kurzor změní svůj tvar na šipku.
2. kliknout levým tlačítkem myši na počáteční stav a se stisknutou myší táhnout do koncového stavu. V koncovém stavu tlačítko myši uvolnit.
   Po uvolnění tlačítka je mezi stavy vytvořena požadovaná transice. Opakováním tohoto postupu lze definovat všechny potřebné transice.
3. Ukončení vkládacího režimu se provede stisknutím pravého tlačítka myši.

# Programování akcí







Datum: 24.11.2002
Soubor: PRO2_13cz.23



**Akce**

Ve všech stavech lze definovat akce, které se mají provést. Tyto akce lze rozdělit na :

vstupní akce: akce je provedena pouze jednou, při vstupu FU do daného stavu.

cyklické akce: cyklické akce jsou prováděny v každém cyklu, kdy je RLO=1, tak dlouho, dokud je FU v daném stavu. Při opouštění stavu jsou akce provedeny s RLO=0.

výstupní akce: výstupní akce jsou provedeny pouze jednou, při opouštění stavu.

Textový editor umožňuje definování prováděných instrukcí pomocí tlačítek. Syntaktická kontrola instrukcí se provádí při ukončení editace příslušného dialogového okna.

**Deklarace rozhraní**

Rozhraní je tvořeno seznamem adres všech signálů, které budou použity ve stavovém grafu. Lze definovat všechny použité vstupy, výstupy, M-bity, hlášení a podmínky. Formální adresa je popsána :

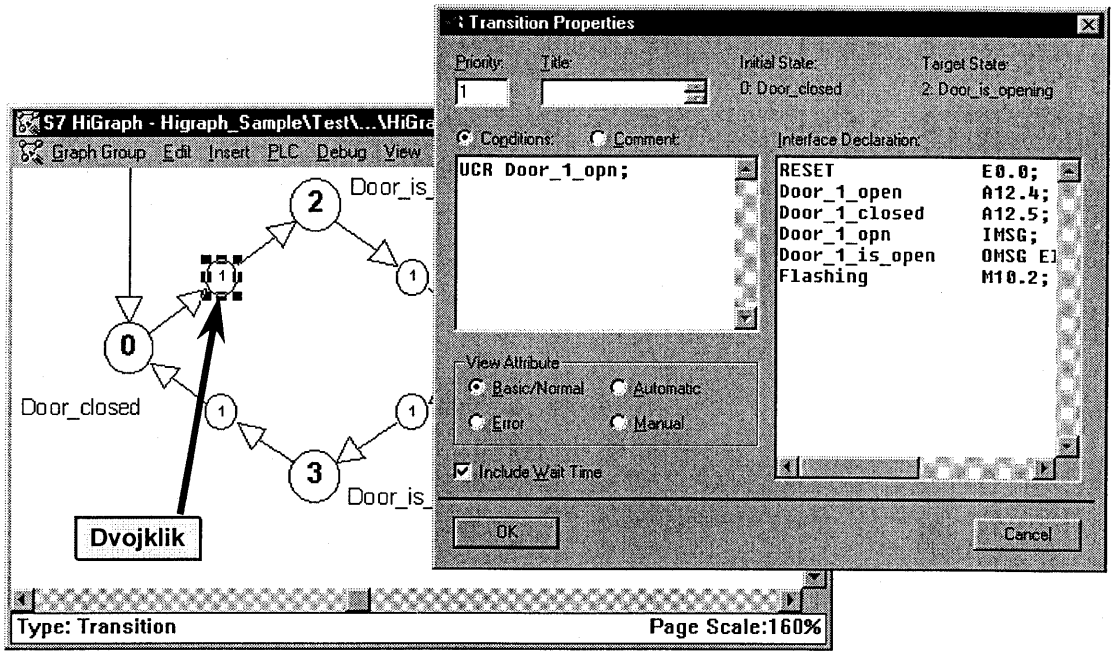
- symbolickým jménem, použitým ve stavovém grafu.
- S7 adresou, která může být zadána absolutně nebo symbolicky (tabulka symbolů)
- identifikátorem datového typu(t.j.: BOOL, BYTE, WORD, INT, ..., IMSG, OMSG)

**Deklaračí seznam**

Při definování akcí a podmínek stavového grafu lze použít pouze symbolické adresy uvedené v deklaračním seznamu.

Tento postup umožňuje také opakované použití stavových grafů prostřednictvím šablon, bez nutnosti změny akcí nebo transic.

# Definice podmínek přechodů







Datum: 24.11.2002
Soubor: PRO2_13cz.24



**Transice**

Přechod z jednoho stavu do druhého je popsán přechodovými podmínkami.

V transici jsou podmínky zpracovávány v souladu se zákony Boolean logiky. Lze definovat adresy bez hodnoty, vyjímku tvoří pouze hlášení.

Programování podmínek transicí lze provést v Higraphu také pomocí jazyka podobného STL. Syntaktická kontrola je potom provedena při ukončení dialogového okna.

**Priority**

Transice, které začínají v jednom stavu lze jednoznačně rozlišit přiřazením priorit. Je-li potom splněno několik transic najednou, je zahájena transice s nižším prioritním číslem.

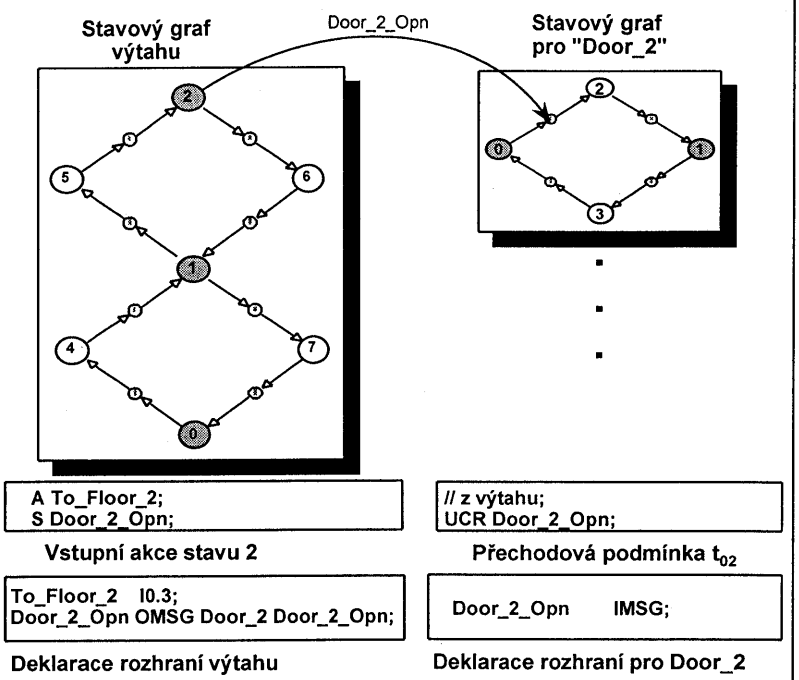
**Poznámka**

V dialogovém okně "Graph Group Properties" (nabídka *Graph Group -> Properties*) lze volit zobrazení jednotlivých objektů pomocí zaškrtávacích polí.

V editačním okně grafu jsou instrukce stavů a transic zobrazeny vně jednotlivých objektů.

## Komunikace mezi diagramy

- Vyměňovaná hlášení
  - vnitřní hlášení
  - vnější hlášení








**Přehled** Stavové grafy se mohou navzájem ovlivňovat výměnou hlášení. Tato hlášení jsou jedním grafem vyslána a cílovým grafem přijata a vyhodnocena.

**Hlášení** Hlášení je binární proměnná vyslaná akcí a vyhodnocená buď transicí nebo akcí.

Hlášení je vždy spojeno s určitým stavem (interní hlášení) nebo s proměnnou.

**Interní hlášení** Interní hlášení slouží k synchronizaci uvnitř jedné skupiny grafů. Jsou tvořena bity v připojeném datovém bloku DB.

Při deklaraci interního hlášení je vždy definován cílový graf -diagram.

**Externí hlášení** Externí hlášení slouží k synchronizaci grafů v různých skupinách (FC). Při deklaraci hlášení není definován název grafu, ale adresa globální proměnné (např. M-bitu).

**Deklarace** Při deklaraci hlášení musí být definován typ hlášení - zda se jedná o vstupní (IMSG) nebo výstupní (OMSG) hlášení.

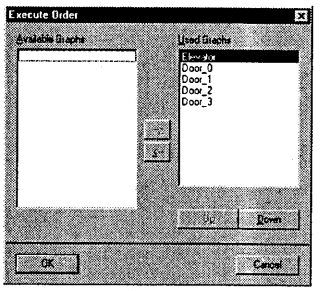
Navíc u interního výstupního hlášení musí být definován název cílového grafu-diagramu a symbolický název hlášení, pod kterým bude vyhodnocováno.

**Příklad:**

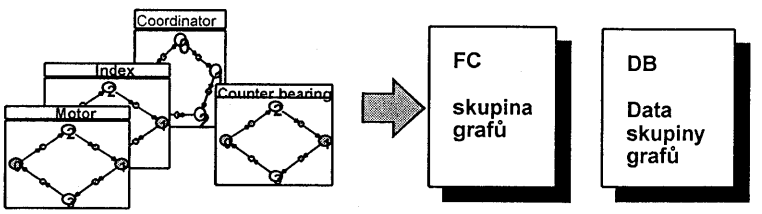
```
Door_0_Open      IMSG;                          //interní vstupní hlášení
Door_1_Is_Open   OMSG Elevator Door_1_Is_Open;  //interní
                                                 výstupní hlášení
Door_1_Close     IMSG DB100.DBx2.2;             //externí vstupní hlášení
Door_1_Is_Closed OMSG M100.0;                   //externí výstupní hlášení
```

## Tvorba a zařazení výsledného kódu

- **Pořadí provedení**
  - **Nabídka:**
    ***Graph Group -> Execute Order***




- **Překlad**
  - **Nabídka:**
    ***Graph***
    ***Group -> Compile***




- **Začlenění do OB1**
  - **Přiřazení parametru "anlauf"**

```
OB1 : ???

Network 1: ???

      CALL  "Elevator"
       anlauf:=I1.0
```





**Tvorba kódu** Výsledný program vytvořený překladačem HiGraph je závislý na cílovém řídícím systému. Překladačem vytvořený *S7-Destination Code* je uložen v závislosti na S7 datech ve formě bloků FC a DB.

**Zpracování** Zpracování systému grafů je prováděno cyklicky a je definováno přes nabídku *Graph Group -> Execution Sequence...* .

**Cyklické zpracování** Cyklické zpracování grafu má následující vlastnosti:

- každý stavový graf (proces) se při náběhu systému nachází v definovaném počátečním stavu (stav 0).
- stavy jednotlivých stavových grafů jsou měněny v závislosti na splněných transicích na cílové stavy.
- každý graf může být v jednom cyklu pouze v jednom stavu.
- stavová data grafu jsou uložena v DB

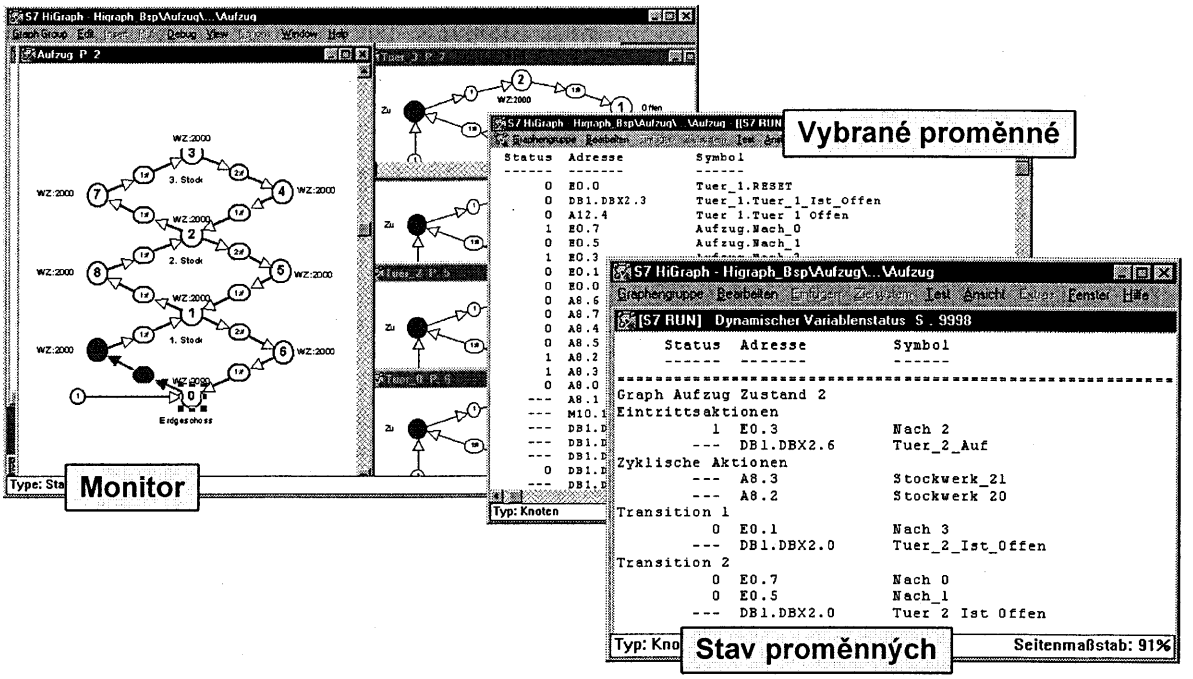
**Velikost cílového kódu** Uživatel může ovlivnit velikost výsledného kódu následujícím způsobem:

- smazání formálních adres diagnostických údajů ("Diagnosis Info")
- smazání všech kontrolních časů a chybových pohledů v grafu.
- redukování počtu grafů (lepší málo velkých než hodně malých grafů)
- věnování pozornosti nepřerušeného číslování stavů (vzestupně od nuly).

**Náběh a restart** Při volání skupinového FC bloku musí být přiřazen OB1 lokální bit určující náběh/restart řídícího systému. Při náběhu nebo restartu PLC jsou všechny stavové grafy uvedeny do počátečního stavu 0.

**Nahrání bloků** Bloky lze nahrát do CPU pomocí nabídky *PLC -> Download...* nebo pomocí SIMATIC Manageru.

# Testovací funkce S7-HiGraphu




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.27



**Přehled**

Testovací funkce lze, po nahrání bloků do CPU, spustit a zastavit přes nabídku *Debug ->Monitor.*

**Stavové zobrazení**

HiGraph umožňuje sledování stavu jednotlivých grafů-diagramů. Momentálně platný stav nebo transice je barevně zvýrazněna.

**Stav programu**

Je-li funkce "Monitor" zapnuta, lze dvojklikem vybrat stav nebo transici, která bude detailně zobrazena. Otevře se okno s STL programem a kursor bude umístěn na první řádek.

**Stav proměnných**

Nabídka *Debug ->Select Variable....* umožňuje výběr symbolů, které mají být zobrazeny v okně. Lze zobrazit:

Stav: hodnotu proměnné.

Adresu: absolutní adresu proměnné.

Symbol: symbolické jméno proměnné.

**Dynamický stav proměnné**

Toto okno umožňuje zobrazení stavu proměnných grafu, které jsou v lokálním prostředí aktivního stavu.

**Změna programu**

Změnu - editaci programu lze provést pouze při vypnuté funkci sledování. Změna se provádí stejně jako tvorba nového programu. Po uložení, překladu a nahrání nové verze programu do CPU lze pokračovat v testování algoritmu.

## S7- SCL (vyšší programovací jazyk)

### Výhody

- snadnější a rychlejší programování
- vysoká kvalita programu
- přehlednost, čitelnost programu
- snadnější testování

### Nevýhody

- doplňkové údaje pro překladač

### Příklad

```
FUNCTION_BLOCK FC22: REAL
        VAR_INPUT
                x1: REAL;
                x2: REAL;
                y1: REAL;
                y2: REAL;
        END_VAR
        VAR_OUTPUT
                Q2: REAL;
        END_VAR
        BEGIN
        FC22:=SQRT((x2 - x1)**2 + (y2 - y1)**2);
        // FC22: = Return value of the function

        Q2 := x1;
END_FUNCTION
```

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.28

Školicí středisko
firmy E&A spol. s r.o., MB

**S7-SCL**

*Structured Control Language* je v souladu s IEC EN 6 1131 Part 3 (Strukturovaný Text)

SCL je PASCALu podobný vyšší programovací jazyk určený pro jednoduché programování matematických algoritmů, správy dat a organizačních úloh na systémech S7 - 300/400 a C7.

Systémy S7 lze použít také pro komplexní úlohy, např. uzavřené řídící smyčky.

SCL program je vytvořen a uložen v SCL kódu ve zdrojovém souboru. Proveditelný STL blok je vytvořen během překladu zdrojového textu.

**Možnosti**

SCL nabízí možnosti vyššího programovacího jazyka:

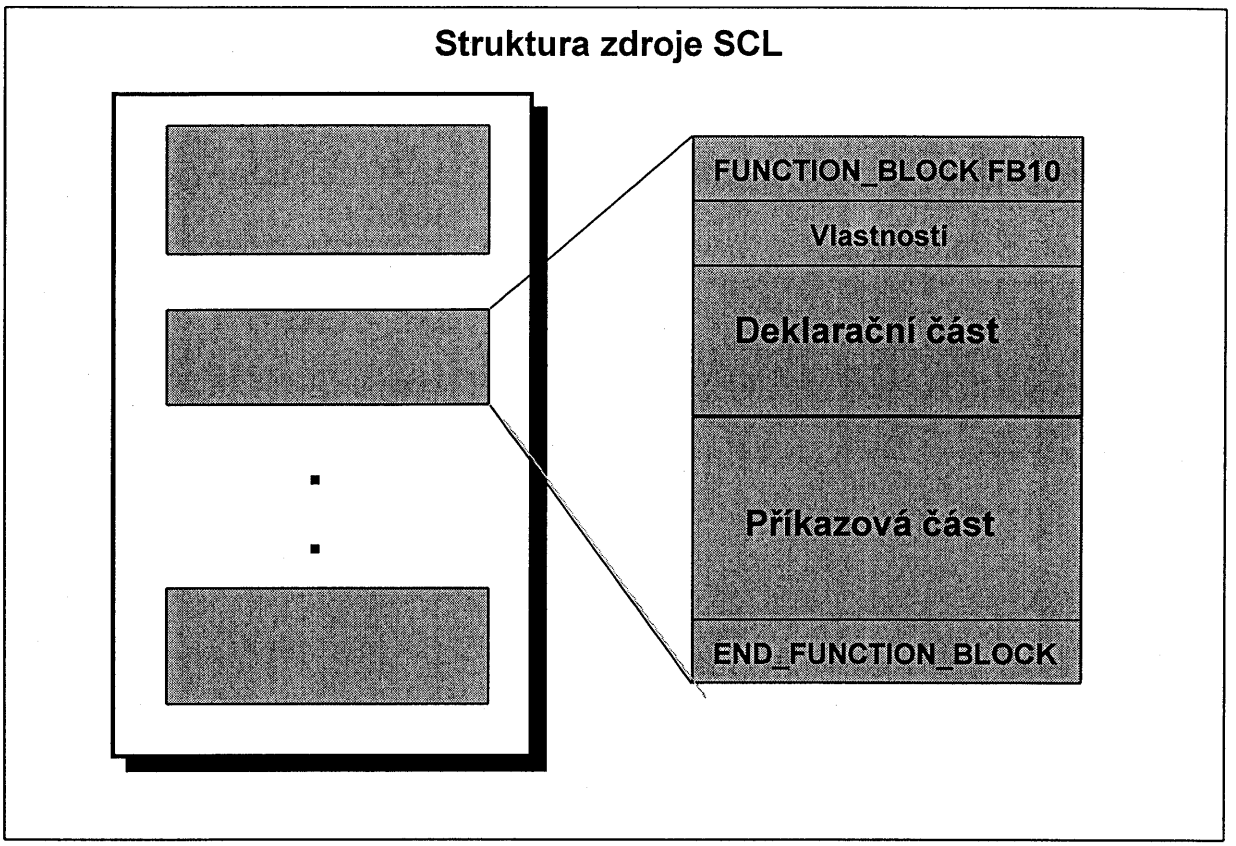
- smyčky
- rozhodování
- větvení, atd.

kombinované se specifickými řídícími funkcemi:

- bitový přístup do oblasti I/O, M-bitů, časovačů, čítačů
- přístup k tabulce symbolů
- přístup k STEP7 blokům

**Výhody SCL**

- jednoduchá výuka programování, zvláště pro začátečníky
- "čitelnost" a přehlednost programu
- jednoduché programování komplexních algoritmů a zpracování komplexních datových struktur
- integrovaný debugger pro symbolické odlaďování zdrojového kódu programu.
- systémová integrace do prostředí STEPu 7.
- snadno pochopitelný pro PLC techniky díky podobnosti s jazyky S7.




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.29



**Struktura SCL textu** SCL zdrojový text může obsahovat libovolný počet bloků (OB, FB, FC, DB a UDT).

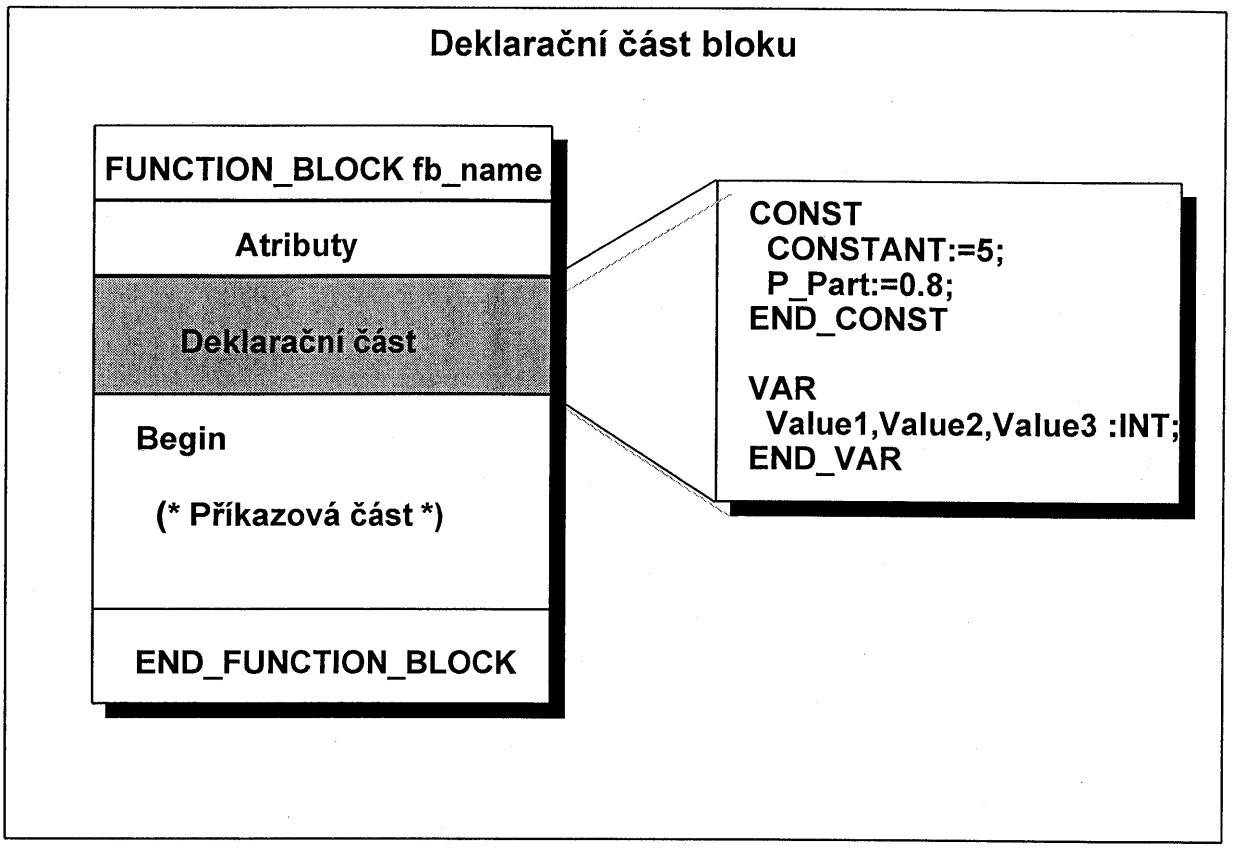
**Struktura bloku** Uvnitř zdrojového textu je každý blok, v závislosti na typu, ohraničen standardními identifikátory. Tělo bloku obsahuje deklarační a výkonnou část. Vlastnosti -atributy bloku jsou definovány v části mezi identifikátorem počátku bloku a deklarační částí bloku.

**Vlastnosti** Atributy definují vlastnosti bloku, které mohou být zobrazeny po překladu SIMATIC Managerem přes nabídku *Edit -> Object Properties*.

**Deklarační část** V deklarační části bloku jsou definovány lokální proměnné, parametry bloku, konstanty a návěští skoků.

**Příkazová část** Příkazová část obsahuje jednotlivé příkazy, které mají být provedeny.

**Pořadí bloků** Při překladu SCL programu do STL jazyka je nutné mít na paměti, že **volaný** blok **musí být definován před volajícím** blokem.







Datum: 24.11.2002
Soubor: PRO2_13cz.30



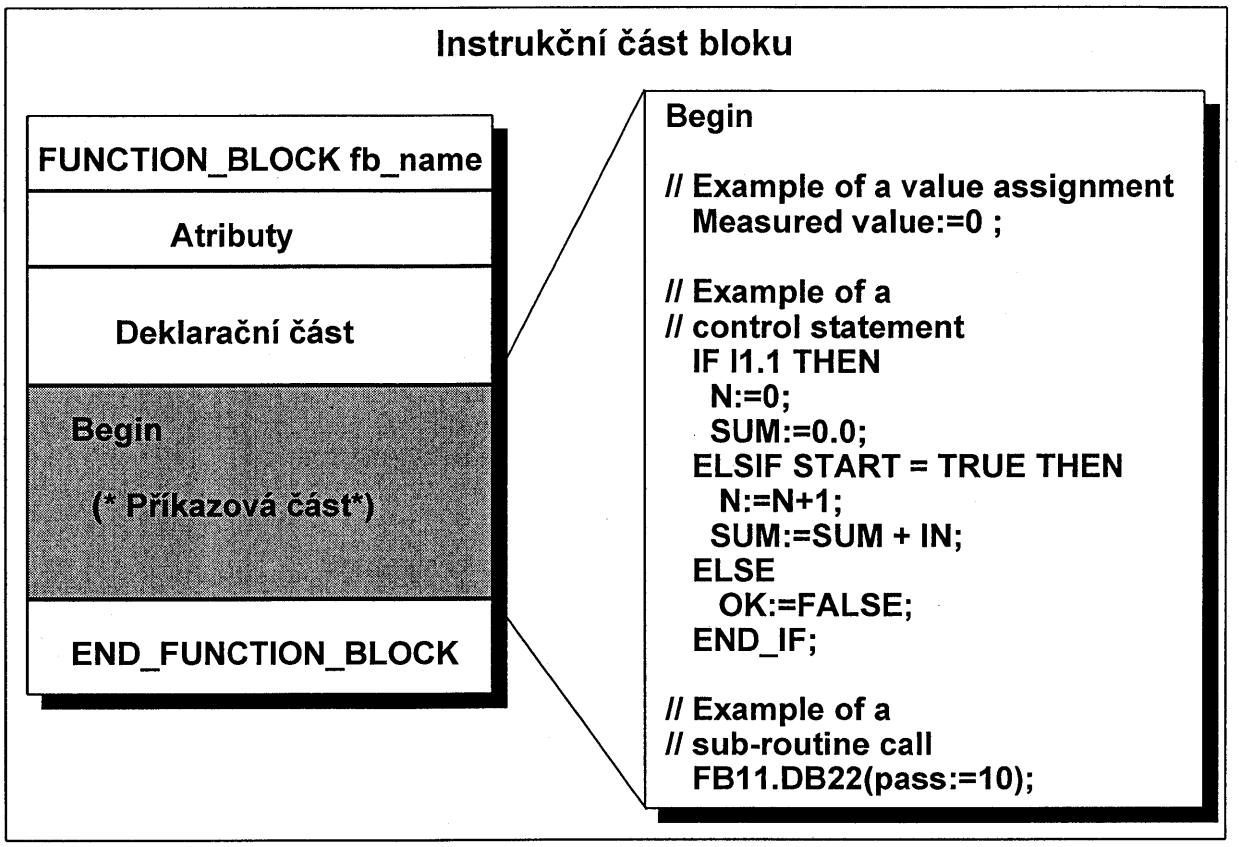
**Vzhled**

Deklarační část je určena pro definici lokálních a globálních proměnných, parametrů bloku, konstant a návěští skoků. Je rozdělena do jednotlivých deklaračních částí definovaných klíčovými slovy.

Jednotlivé části mohou být vloženy do SCL textu pomocí nabídky *Insert -> Block Template -> Constant, Parameter.*

**Bloky**

| Data | Syntaxe | FB | FC | OB | DB | UDT |
|---|---|---|---|---|---|---|
| Konstanty | CONST<br>*Declaration list*<br>END_CONST | X | X | X | | |
| Návěští skoku | LABEL<br>*Declaration list*<br>END_LABEL | X | X | X | | |
| Lokální proměnná | VAR_TEMP<br>*Declaration list*<br>END_VAR | X | X | X | | |
| Statická proměnná | VAR (STRUCT)<br>*Declaration list*<br>END_VAR | X | | | (X) | (X) |
| Vstupní parametr | VAR_INPUT<br>*Declaration list*<br>END_VAR | X | X | | | |
| Výstupní parametr | VAR_OUTPUT<br>*Declaration list*<br>END_VAR | X | X | | | |
| Vstupně/výstupní parametr | VAR_IN_OUT<br>*Declaration list*<br>END_VAR | X | X | | | |




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.31

Školicí středisko
firmy E&A spol. s r.o., MB

**Příkazová část** Tato část obsahuje instrukce, které jsou zpracovány při volání bloku.

**Dělení** Jednotlivé instrukce lze rozdělit do 3 skupin:

- přiřazení hodnot: tato skupina umožňuje vyhodnocení aritmetického výrazu nebo přiřazení hodnoty proměnné
- řídící příkazy: slouží k větvení programu nebo k opakování skupin instrukcí.
- volání podprogramů: umožňují volání funkcí a funkčních bloků.

**Poznámka** Při programování příkazové části bloku je nutné vzít v úvahu tato pravidla:

- příkazová část začíná klíčovým slovem BEGIN a končí klíčovým slovem pro konec daného typu bloku.
- každý příkaz musí být zakončen středníkem.
- všechny identifikátory (jména) použité v příkazové části musí být předem deklarovány.

**Šablony** Pomocí nabídky *Insert -> Control Structure ->IF, CASE, FOR, WHILE, REPEAT* lze do zdrojového textu vložit šablony řídících struktur programu.

## Výrazy, operátory a adresy v S7-SCL

- **Výrazy**
  - Matematické výrazy — ((3+CONST_INT) * (VAR_INT ** 37) / 3.14)
  - Porovnávací výrazy — Q >=9
  - Logické výrazy — (n >5) AND (n < 20)
- **Operátory**
  - přiřazení — :=
  - Matematické operátory — *, /, MOD, DIV, +, -, **
  - Porovnávací operátory — <, >, <=, >=, = <>
  - Logické operátory — NOT, AND nebo &, XOR, OR
- **Adresy**
  - Konstanty — 30. 0, FACTOR, 'SIEMENS'
  - Rozšířené proměnné — Status, IB5, DB10.DW5, Motor.Power, FC12(Q:=On)
  - Výrazy v (...) — ((3+CONST_INT) * (VAR_INT ** 37))



Datum: 24.11.2002
Soubor: PRO2_13cz.32

Školicí středisko
firmy E&A spol. s r.o., MB

**Výrazy**

Výrazy obsahují adresy, operátory nebo kulaté závorky.

SCL dovoluje formulování standardních výrazů, tj. matematických, logických a porovnávacích. Při tvorbě výrazu lze použít proměnné z datových bloků, struktur, polí, vstupů a výstupů.

**Operátory a Adresy**

Výrazy obsahují operátory a adresy. Většina SCL operátorů spojuje dvě adresy (např. A +B) a jsou označovány jako binární operátory. Ostatní pracují pouze s jednou adresou a jsou označovány jako unární operátory.

Výsledek výrazu leží na jeho druhé straně.

- přiřazení proměnné (A := B + C;)
- podmínka řídícího příkazu (IF A< B DO ... )
- aktuální parametr při volání funkce nebo funkčního bloku (FB20 (Input := A + B) )

## Příkazy v S7-SCL

- **Přiřazení hodnoty**
  - **příklad:** `A := B + C;`
- **Řídící příkazy**
  - **příkaz IF** `IF I1.1 THEN ... ELSIF ... ELSE ... END_IF`
  - **příkaz CASE** `CASE SELECTOR OF 1: ...; 2: ... ELSE: ... END_CASE`
  - **příkaz FOR** `FOR INDEX := 1 TO 49 BY 2 DO ... END_FOR`
  - **příkaz WHILE** `WHILE INDEX <= 50 DO ... END_WHILE`
  - **příkaz REPEAT** `REPEAT ... UNTIL INDEX:= 51 ... END_REPEAT`
  - **příkaz CONTINUE** `WHILE BOOL_1 DO ... CONTINUE ... END_WHILE`
  - **příkaz EXIT** `WHILE BOOL_1 DO ... EXIT ... END_WHILE`
  - **příkaz GOTO** `IF INDEX <23 THEN GOTO MARK; ...`
  - **příkaz RETURN** `IF ENABLED THEN RETURN; ...`
- **Volání funkcí a funkčních bloků**
  - **volání FB nebo SFB** `FB11.DB20(IN:=VALUE1, THROUGH:=VALUE2);`
  - **volání FC nebo SFC** `RETURN := FC32(IN:=VALUE1,OUT:=VALUE2);`

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.33



**Příkazy**

V příkazové části jsou definovány akce, které se mají provádět s proměnnými definovanými v deklarační části bloku. Normálně jsou příkazy prováděny v pořadí v jakém jsou uvedeny v programu.

**Přiřazení hodnot**

Tato instrukce umožňuje přiřazení nové hodnoty proměnné. Stará hodnota je přitom ztracena.

**Řídící příkazy**

Sled zpracování instrukcí může být pomocí těchto instrukcí změněn.

Umožňují variabilní zpracování programu, podle toho, které podmínky jsou splněny (instrukce IF a CASE).
Instrukce smyčky umožňují opakované zpracování skupiny instrukcí (instrukce FOR, WHILE a REPEAT).

Instrukce skoku (CONTINUE, EXIT a GOTO) umožňují přerušení sledu zpracovávaných instrukcí a pokračování ve zvoleném místě.

**Volání FB a FC**

V souladu se principy strukturovaného programování, lze z SCL bloku volat jiné funkce (FC, SFC) nebo funkční bloky (FB, SFB). Lze volat:

- funkce a funkční bloky vytvořené v SCL nebo jiném S7 programovacím jazyce.
- standardní funkce a funkční bloky dodané spolu s SCL..
- systémové funkce a funkční bloky obsažené v operačním systému CPU.

## Přiřazení hodnot v S7-SCL

- **Lokální proměnné**
  - **základní datové typy** — `COUNTER := (5 + RUNVAR) * 2;`
  - **struktury**
    - kompletní struktura — `STRUCT_1 := STRUCT_2;`
    - složky struktury — `STRUCT_1.COMP3 := STRUCT_2.COMP1;`
  - **pole**
    - celé pole — `ARRAY_1 := ARRAY_2;`
    - prvky pole — `ARRAY_1[I] := ARRAY_2 [J];`
- **Globální proměnné**
  - **paměťové oblasti CPU**
    - absolutní přístup — `VALUE := IW10;`
    - symbolický přístup — `VALUE := INPUT;` // "INPUT" v symbolické tabulce
    - indexovaný přístup — `VALUE := IW[INDEX];`
  - **datové bloky**
    - absolutní přístup — `VALUE := DB11.DW5;`
    - symbolický přístup — `VALUE := MOTOR.POWER;` // MOTOR a POWER musí
    - indexovaný přístup — `VALUE := MOTOR.DW[Index];` // být v tabulce symbolů
    - přes vstupní parametry — `VALUE := I_PAR.DW[Index];` // I_PAR je VAR_IN





**Princip**

Přiřazení hodnoty umožňuje přepsat starou hodnotu uloženou v proměnné hodnotou novou, danou vyhodnocením výrazu. Tento výraz může obsahovat také identifikátor FC bloku, který je volán a vrací odpovídající hodnotu.

Hodnota výrazu musí být kompatibilní s datovým typem proměnné.

**Přiřazení hodnoty komplexní proměnné**

Komplexní proměnná odkazuje na kompletní proměnnou (kompletní struktura, kompletní pole, řetězec) nebo na složku komplexní proměnné. Jsou dvě možnosti přiřazení komplexní proměnné:

- přiřazení celé komplexní proměnné jiné celé komplexní proměnné (struktura, pole, řetězec).

  Toto přiřazení lze provést pouze tehdy, jsou-li v cílové struktuře komponenty stejného názvu a typu jako ve zdrojové struktuře.

  Kompletní pole lze přiřadit pouze poli se stejným datovým typem položek a velikosti.
- přiřazení jednotlivých komponent struktury jednotlivým komponentám cílové struktury stejného datového typu.

# Příkaz IF v S7-SCL

## Syntaxe

```
IF      <výraz> THEN <příkazy>
[ELSIF  <výraz> THEN < příkazy>] (volitelně)
.
.
.
[ELSIF  <výraz> THEN < příkazy>] (volitelně)
[ELSE   <příkazy>]                          (volitelně)
END_IF
```

## Příklad

```
IF INPUT_OK THEN
   N := 0;
   SUM := 0.0;
   OK := FALSE;                  // nastav OK-Flag na FALSE
ELSIF START_OK THEN
   N := N + 1;
   SUM := SUM + IN;
ELSE
   OK := FALSE;
END_IF;
```





**Princip**

Častým problémem programování je to, že část instrukcí má být provedena v závislosti na definovaných podmínkách. Vyžaduje to větvení programu do alternativních větví programu.

Instrukce IF je podmíněnou instrukcí. Umožňuje definování jedné nebo několika možností a v závislosti na splněné podmínce bude zpracována pouze jedna (nebo také žádná) větev programu.

**Zpracování**

Instrukce IF je provedena podle následujících pravidel:

1. je-li hodnota prvního výrazu TRUE (pravda), je zpracována část příkazů uvedená po klauzuli THEN, v opačném případě jsou vyhodnoceny výrazy uvedené v klauzulích ELSIF.
2. pokud není splněn žádný logický výraz definovaný v klauzulích ELSIF, je provedena (pokud je definována) skupina příkazů definovaná v klauzuli ELSE.

Počet klauzulí ELSIF závisí na uživateli. Klauzule ELSIF a ELSE nemusí být vůbec definovány.

# Příkaz WHILE v S7-SCL

## Syntaxe

```
WHILE <výraz> DO <příkazy>
END_WHILE
```

## Příklad

```
FUNCTION_BLOCK SEARCH          // SEARCH je deklarován v tabulce symbolů
VAR
   INDEX        : INT;
   KEYWORD      : ARRAY[1..50] OF STRING;
END_VAR

BEGIN
INDEX := 1;
WHILE INDEX <= 50 AND KEYWORD[INDEX] <> 'KEY'
DO
   INDEX := INDEX + 2;
END_WHILE;

END_FUNCTION_BLOCK
```





**Princip**

Instrukce WHILE umožňuje opakované zpracování části programu na základě definované podmínky. Zpracování podmínky se řídí pravidly zpracování logických výrazů.

Příkazy definované za klauzulí DO jsou zpracovávány tak dlouho, dokud je hodnota vyhodnocované podmínky TRUE (pravda).

**Zpracování**

Instrukce WHILE je zpracovávána podle následujících pravidel:

1. vyhodnocení podmínky je provedeno před každým zpracováním části DO
2. je-li hodnota logického výrazu TRUE (pravda) je klauzule DO zpracována.
3. v opačném případě je zpracování instrukce WHILE ukončeno. Tato situace může nastat také v případě prvního vyhodnocení podmínky.

## Volání funkčních bloků

- **Volání globální instance**
  - **Absolutní volání**

```
FB10.DB20(X1 := 5, X2 := IW12, ...);      (*  volá FB10 s instančním DB20 *)
```

  - **Symbolické volání**

```
DRIVE.ON(X1 =5, X2 := IW12,...);          (*  DRIVE a ON jsou deklarovány v
                                              tabulce symbolů *)
```

- **Volání lokální instance**
  - **Volání přes identifikátor**

```
VAR
   MOTOR          : FB10;
END_VAR
BEGIN
.
MOTOR(X1 := 5, X2 := IW12,...);           (*  Volání lokální instance je možné
                                              pouze uvnitř jiných funkčních bloků*)
```

SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.37

Školicí středisko
firmy E&A spol. s r.o., MB

**Globální a lokální instance**

Myšlenka multiinstancí je zohledněna také v SCL. V SCL lze realizovat jak globální instance tak i lokální instance.
Volání FB jako lokální instance se od volání globální instance liší způsobem ukládání hodnot. Data v tomto případě nejsou ukládána do samostatného DB, ale jsou součástí instanční datové oblasti volajícího FB bloku.

**Volání globální instance**

Volání globální instance musí obsahovat:
- jméno funkčního bloku nebo systémového funkčního bloku
- identifikátor instančního datového bloku DB
- přiřazení aktuálních parametrů

Volání globální instance může být provedeno absolutně nebo symbolicky.
Volání FB jako globální instance je možné ve všech typech logických bloků (OB, FB, FC).

**Volání lokální instance**

Při volání lokální instance musí být uvedeno:
- jméno lokální instance (identifikátor)
- přiřazení parametrů.

Volání FB jako lokální instance je možné pouze z bloků FB. Proměnná typu FB musí být definována v deklarační části (VAR ...END_VAR) volajícího blok.

Volání FB je provedeno jako jméno proměnné.

## "OK"-Flag a vyhodnocení chyb

- **Globální bit identifikace chyby (po skončení bloku je zkopírován do bitu BR)**

- **Příklad:**

```
                    // nastavení OK proměnné na TRUE, umožňuje kontrolovat,
                    // zda následující instrukce byly provedeny
                    // bezchybně
OK := TRUE;
SUM := SUM + IN;
IF OK THEN          // součet ukončen úspěšně
    ...
ELSE                // přetečení při sčítání
    ...
END_IF;
```





**Popis**

SCL nabízí logickou globální proměnnou "OK-Flag" , která umožňuje detekci chyb při zpracování logických bloků. Tato proměnná slouží k identifikaci úspěšného nebo neúspěšného provedení příkazů a k provedení následné odpovídající reakce.

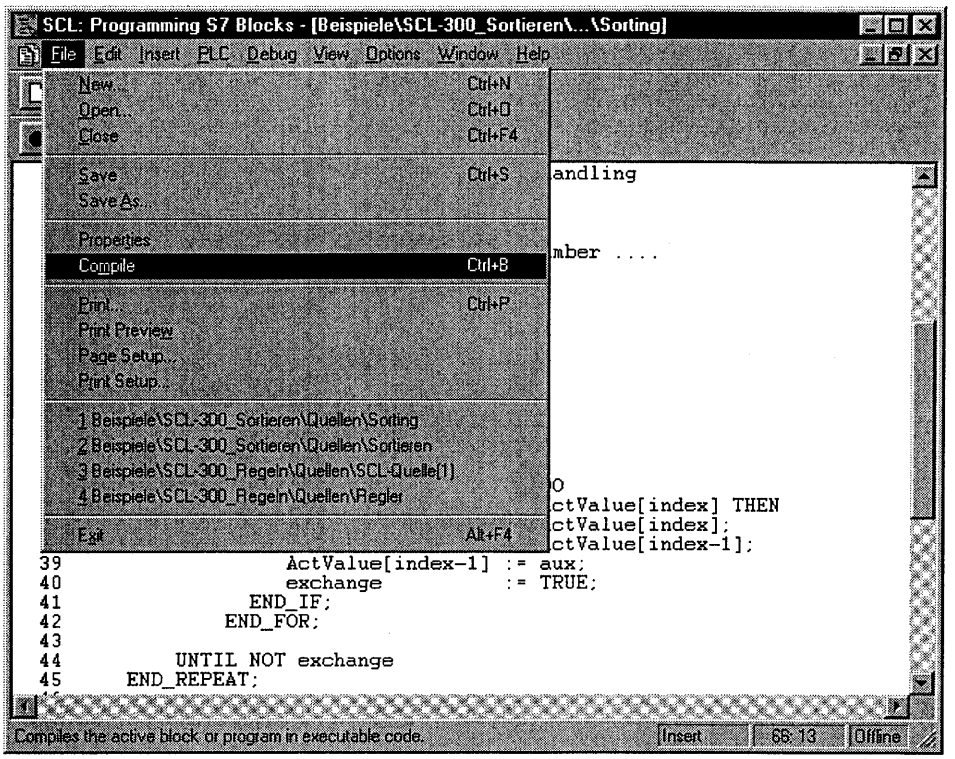
**Zpracování**

Je-li při zpracování příkazů detekována chyba (např. přetečení) je OK-Flag nastaven operačním systémem na hodnotu logické 0 (FALSE). Po ukončení bloku je OK-Flag zkopírován do systémového parametru ENO a může být vyhodnocen jiným blokem.
OK-Flag je při zahájení zpracování programu nastaven na hodnotu TRUE a může být kdekoliv vyhodnocen pomocí instrukcí SCL nebo může být nastaven na hodnotu TRUE nebo FALSE.

**Platnost**

Proměnná OK-Flag je dostupná ve všech CPU, její deklarace není nutná.

# Překlad SCL zdrojového textu








**Překlad**

Před spuštěním nebo zahájením testování je nutné provést překlad SCL programu. Tuto funkci lze spustit z nabídky *File --> Compile* nebo pomocí odpovídající ikony v nástrojové liště.

Překladač má následující vlastnosti:

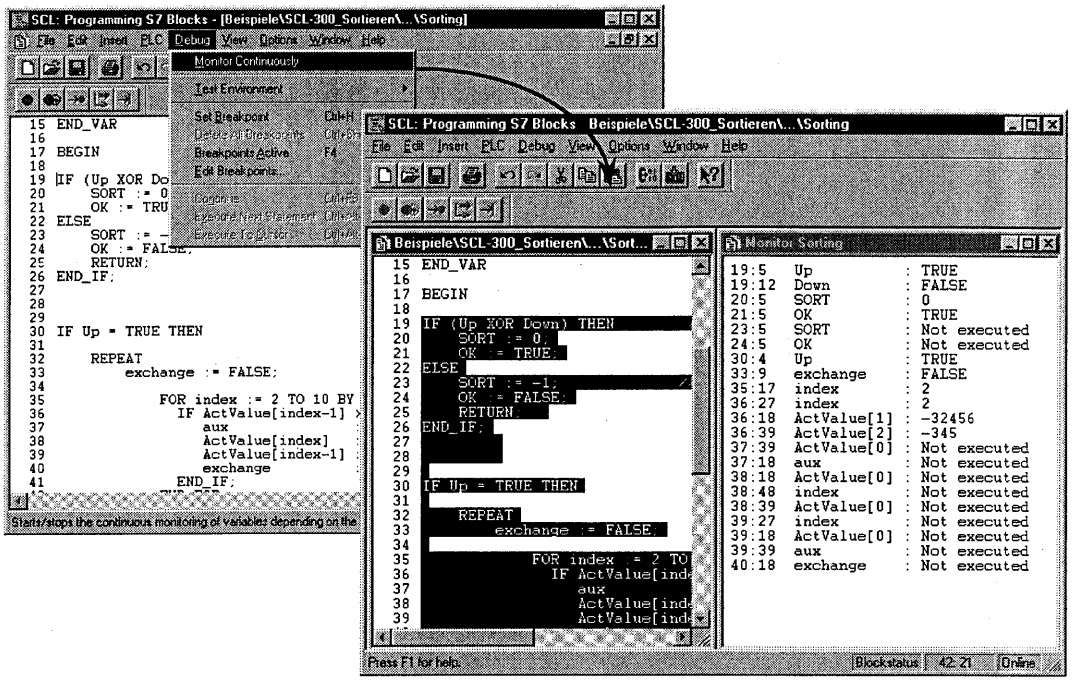
- pracuje v dávkovém režimu, tj. překládá zdrojový text jako celek. Překlad pouze vybraných bloků není možný.
- při překladu kontroluje syntaxi zdrojového textu a zobrazuje všechny zachycené chyby.
- při bezchybném překladu vytváří podle nastavených voleb buď výsledný programový kód nebo poskytuje testovací informace. Je-li požadováno testování ve vyšším programovacím jazyce je nutné aktivovat volbu *Debug Info* pro každý program samostatně.
- překladač vytváří pro každý volaný funkční blok definovaný instanční DB.

**Nastavení**

Uživatel má možnost ovlivnit chování překladače pomocí voleb přístupných přes nabídku Options - > Customize -> Compiler:

- *Maximum number of errors*: maximální počet chyb po jejichž detekci překladač přeruší překlad
- *Create Object Code*: povoluje vytvoření kódu spustitelného na PLC.
- *Optimize*: vytvoří kratší program
- *Range Check*: během zpracování jsou indexy kontrolovány na povolený rozsah
- *Debug Information*: vytvoří doplňkové informace pro testování pomocí debbugeru ve vyšším programovacím jazyce.
- *OK Flag*: při zpracování výsledného programu je při zachycení chyby proměnná OK nastavena na *FALSE*.

# Sledování zpracování programu







Datum: 24.11.2002
Soubor: PRO2_13cz.40



**Výběr**

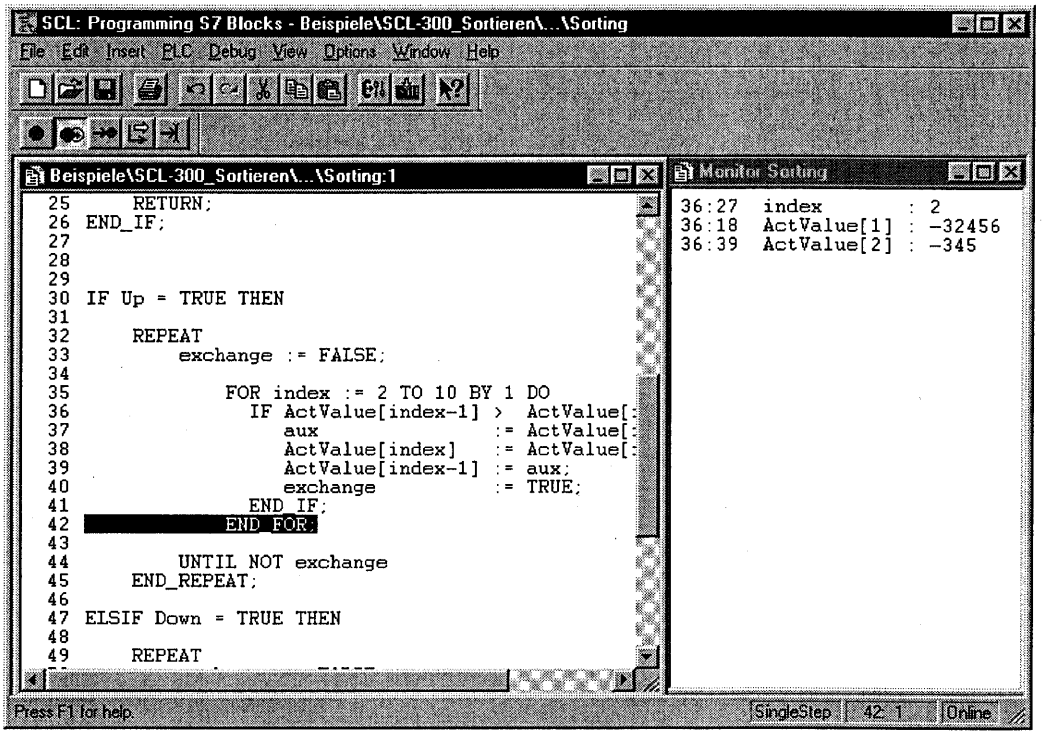
Testovací funkci *Monitor Continously* lze aktivovat následovně:

1. ujistit se, že všechny předpoklady jsou splněny.
2. vybrat okno zdrojového textu a umístit kursor do bloku SCL zdrojového textu.
3. aktivovat funkci z nabídky *Debug -> Monitor Continously.* Zobrazí se prázdné okno "Watch" s tlačítkem "overlay values".
4. v okně editoru umístit kursor na začátek sledované oblasti. Délka sledované oblasti je definována automaticky a je v okně editoru zvýrazněna.
5. funkci lze zrušit pomocí nabídky *Debug -> Monitor Continously* a uzavřením okna "Watch".

**Testovací režim** Sledovanou oblast lze ovlivnit pomocí nabídky *Debug -> Test Environment*:

- *Process*: SCL debugger zredukuje monitorovanou oblast v testovacím prostředí tak, aby doba cyklu byla minimálně ovlivněna.
- *Laboratory*: sledovaná oblast je omezena pouze výkonností CPU v prostředí *Laboratory*. Sledovaná oblast je větší než v režimu prostředí *Process* a díky tomu je doba cyklu odpovídajícím způsobem prodloužena.

# Nastavení a editace bodů přerušení (breakpointů)






**Nastavení bodů přerušení**

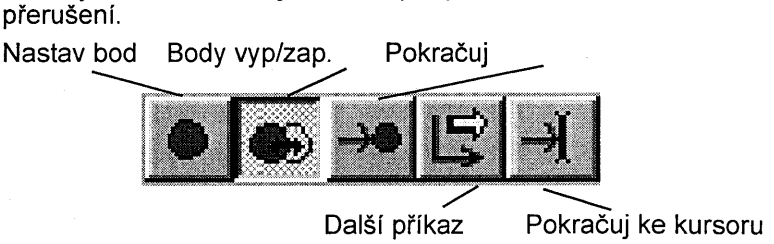
Body přerušení umožňují testování SCL programu. Je možné nastavit několik bodů přerušením. Postup je následující:

1. vybrat okno s testovaným SCL programem.
2. umístit kursor do polohy, kde bude nastaven bod přerušení a aktivovat nabídku *Debug -> Set Breakpoint*:

Při nastavení prvního bodu přerušení je otevřeno okno "Watch" a je zobrazena pozice nastaveného bodu přerušení.

**Nástrojová lišta**

Nástrojová lišta obsahuje tlačítka pro práci v testovacím režimu s body přerušení.



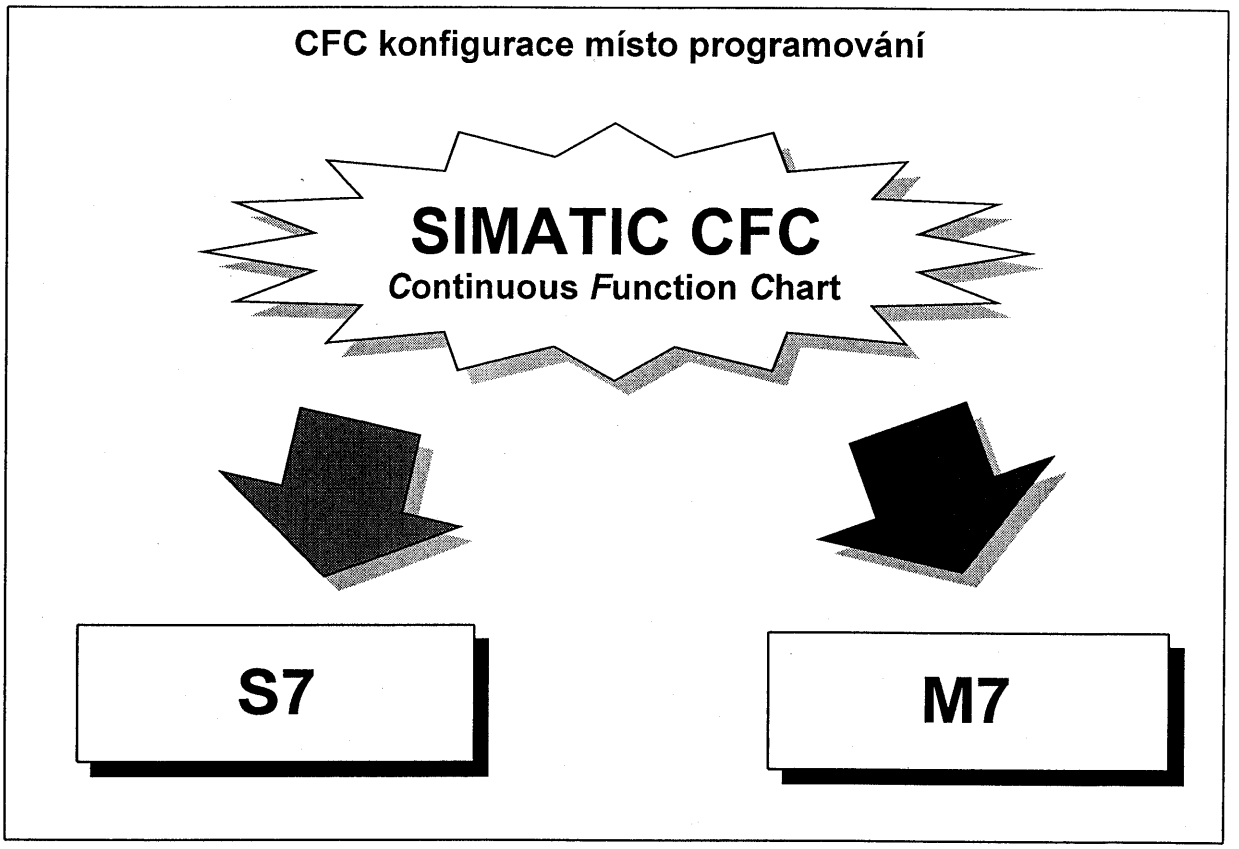

**Editace bodů**

Nastavené body mohou být pomocí nabídky *Debug -> Edit Breakpoints* aktivovány nebo deaktivovány dvojklikem v zobrazeném dialogovém okně.

**Počet bodů**

Počet aktivních, tj. zohledněných bodů přerušení závisí na typu CPU:

- CPU 416: max. 4 aktivní body
- CPU 414: max. 2 aktivní body
- CPU 314: neumožňuje aktivaci bodů přerušení




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.42



**Co je CFC ?**

Programovací doplněk SIMATIC CFC poskytuje možnost technologické a funkčně orientované konfigurace programovatelných řídících systémů S7 a M7.

Uživatelský program lze kompletně vytvářet v prostředí CFC.

Předem připravené bloky mohou být parametrizovány a propojeny při řešení zadané úlohy.

**Výhody**

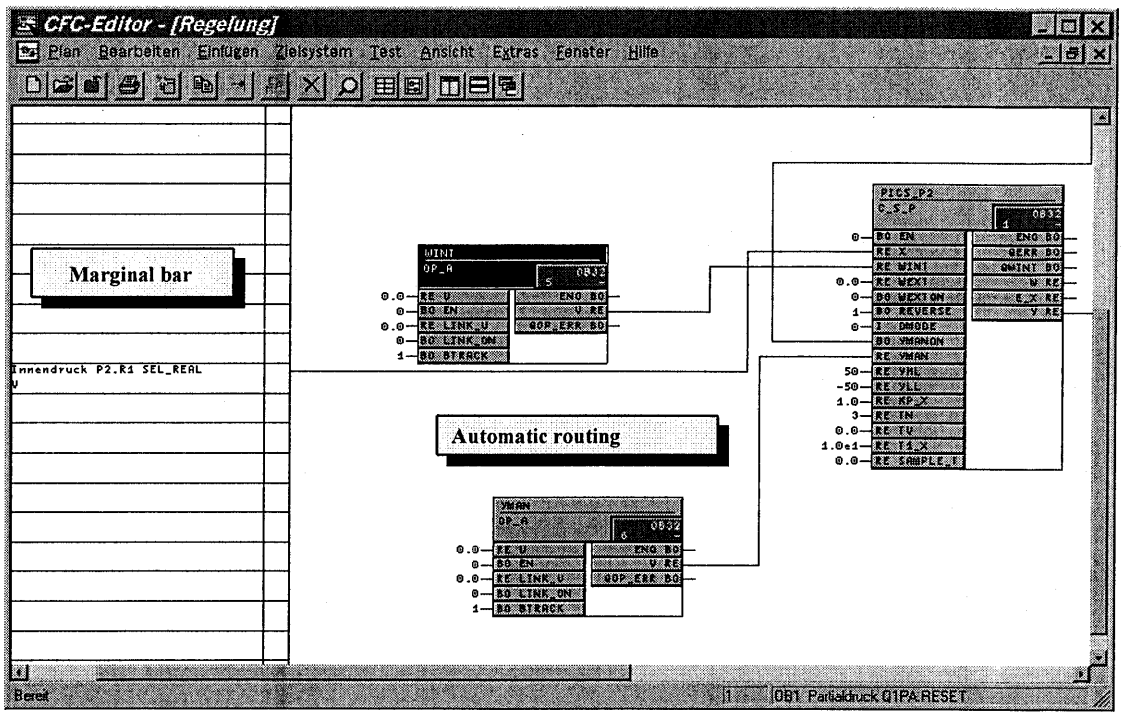
Volitelný doplněk CFC je integrován do prostředí STEPu 7. Je snadno pochopitelný a použitelný.

- CFC může být použit při řešení jednoduchých i komplexních řídících úloh.
- jednoduchý způsob propojování komunikujících bloků poskytuje snadnou konfiguraci programu.
- snadná testování a odlaďování programu.

**Souhrn**

SIMATIC CFC nabízí všem tvůrcům uživatelských programů vysoký standard vytvořených programů při minimálním časovém intervalu.

## CFC schéma







Datum: 24.11.2002
Soubor: PRO2_13cz.43



**CFC schéma**

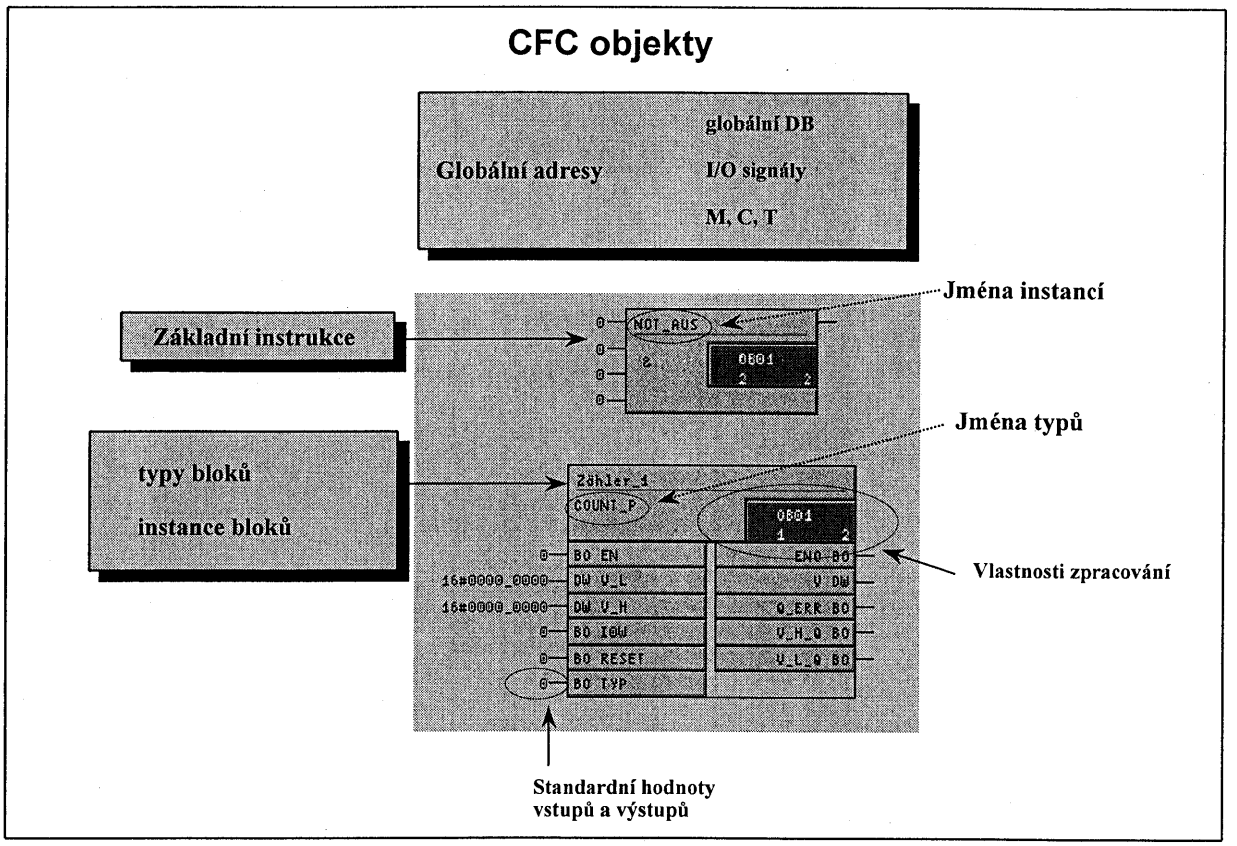
Bloky potřebné k řešení technologické úlohy mohou být rozděleny do několika plánů.

Plán nemá pro generování výsledného kódu nebo jeho zpracování žádný význam.

Plány slouží pouze ke strukturalizaci a dokumentaci programu.

CFC plán obsahuje 6 stran

- 1 strana obsahuje pracovní oblast a dva okrajové sloupce.
- automatickou správu okrajových sloupců
- uživatelsky příjemné rozvržení signálů
- automatické směrování
- kompletní správu zdrojů
- dokumentaci 1 : 1








**CFC objekty**

Nejdůležitější objekty CFC jsou popsány v následujících odstavcích.

**Typy bloků**

Typ bloku představuje šablonu instancí a popisuje jejich vnitřní strukturu.

**Instance bloků (bloky)**

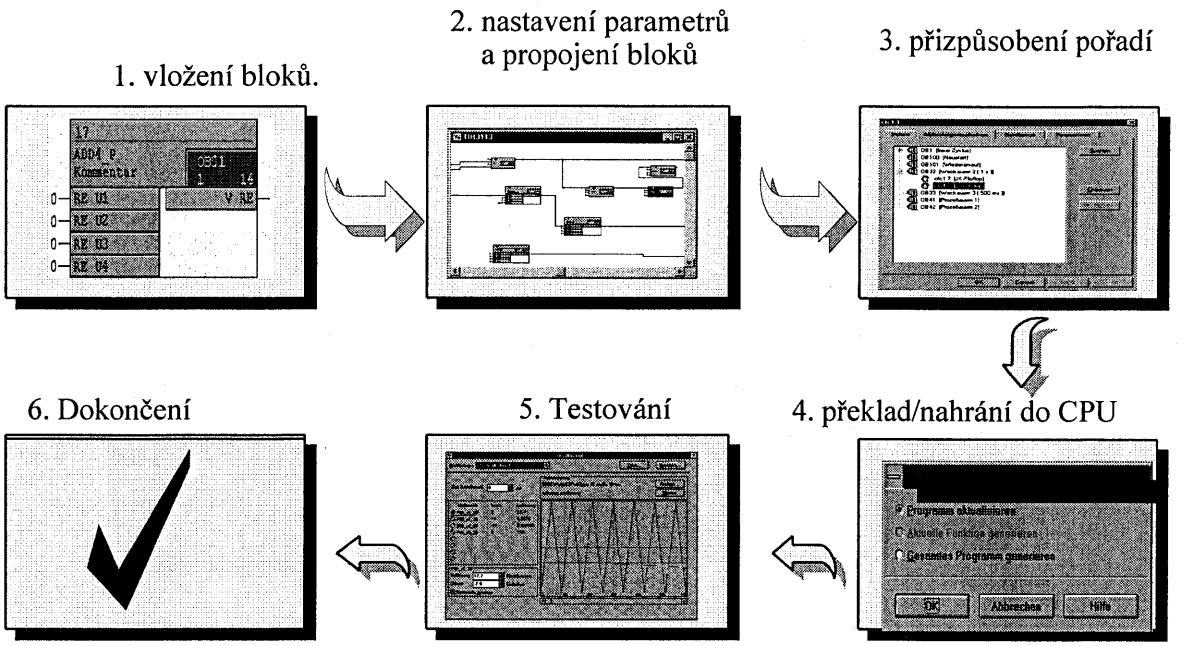
Instance bloku je konkrétní objekt vytvořený v souladu s popsaným typem. Každá instance má jasný identifikátor umožňující rozlišitelnost instancí. Identifikátor je tvořen jednoznačným názvem plánu, tečkou a jednoznačným názvem bloku (např. conveyorcontroller.motor). Maximální délka je 24 znaků.

**Bloky**

Bloky ve STEPu 7 tvoří část programu a mají definovanou funkci, strukturu, použití. Jedná se logické bloky (FB, FC,...), datové bloky, a bloky uživatelsky definovaných datových typů.Tyto bloky obsahují:

- základní instrukce:
  Funkce jako je např. AND, SUM, atd.
- Globální adresy:
  I/O signály, M-bity, čítače, časovače a globální datové bloky.

## CFC programy - konfigurace místo programování




SIMATIC S7



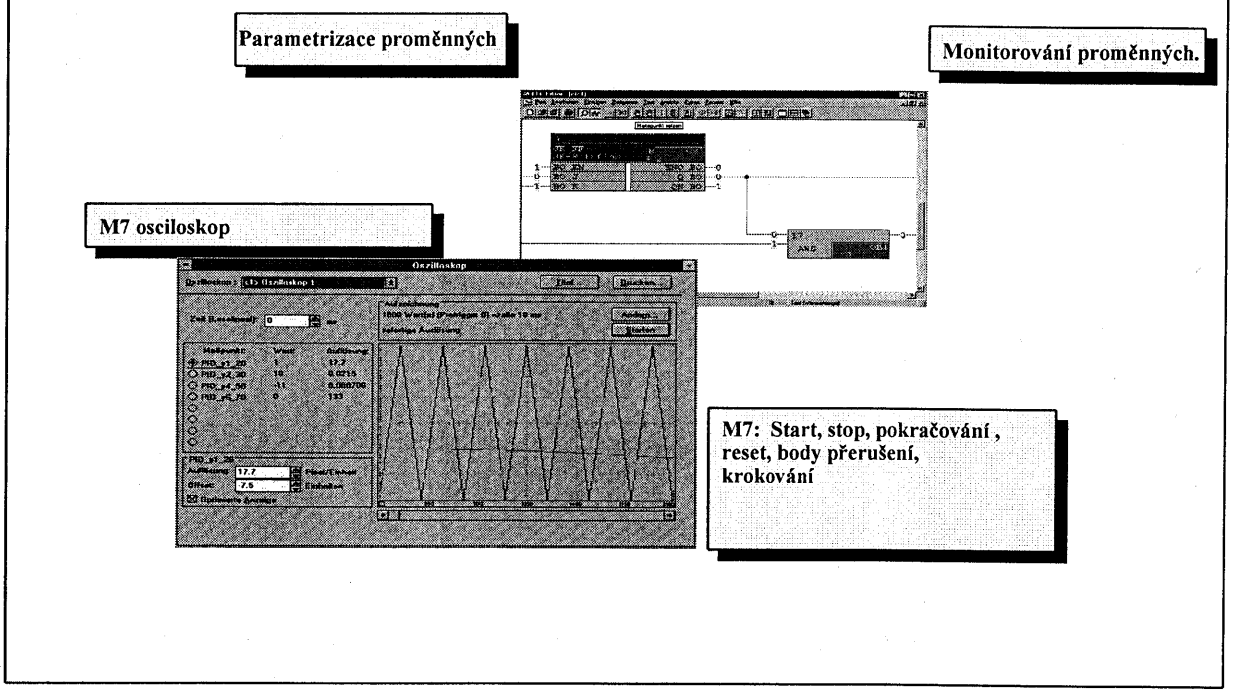
Datum: 24.11.2002
Soubor: PRO2_13cz.45

Školicí středisko
firmy E&A spol. s r.o., MB

**Konfigurování pomocí CFC**

- tvorba projektu v S7/M7 programu.
- tvorba nebo kopírování typu bloků.
- tvorba CFC plánu.
- import typů bloků do CFC.
- vkládání bloků.
- nastavování parametrů a propojování bloků
- přizpůsobení sekvenčních vlastností.
- překlad.
- nahrání.
- testování.
- dokončení.

# Testování a odlaďování





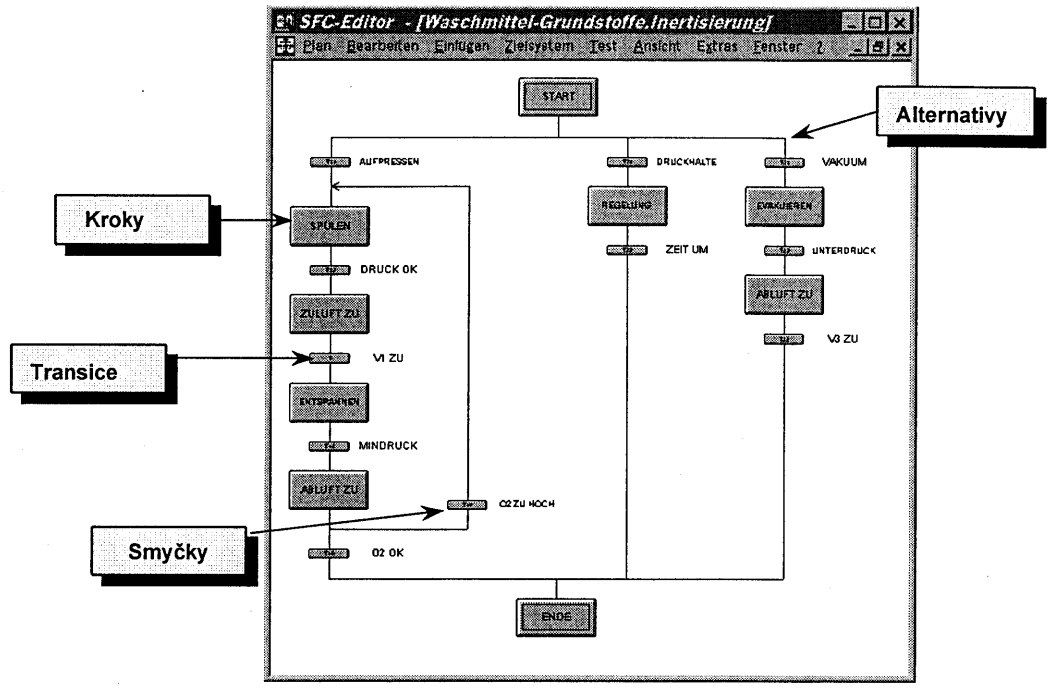



**CFC testování a odlaďování**

CFC nabízí následující testovací a odlaďovací funkce:

- sledování a nastavování parametrů proměnných v plánu
- kontrola zpracování programu pomocí bodů přerušení (pouze M7)
- Cyklické archivování hodnot proměnných (pouze M7)

# SFC = Sekvenční řídící systém




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.47

Školicí středisko
firmy E&A spol. s r.o., MB

**SFC (Sequential Function Chart)**

SFC je sekvenční řídící systém, který přepíná z jednoho stavu do druhého v závislosti na splněných podmínkách

Typickou oblastí aplikace tohoto systému jsou dávkové výrobní procesy.Sekvenční řídící systém může být také nasazen na kontinuální výrobu, např.pro náběh nebo odstavení, změnu pracovních bodů atd.

SFC např. umožňuje definovat výrobní proces jako proces řízený událostmi (event-driven system).

**Princip**

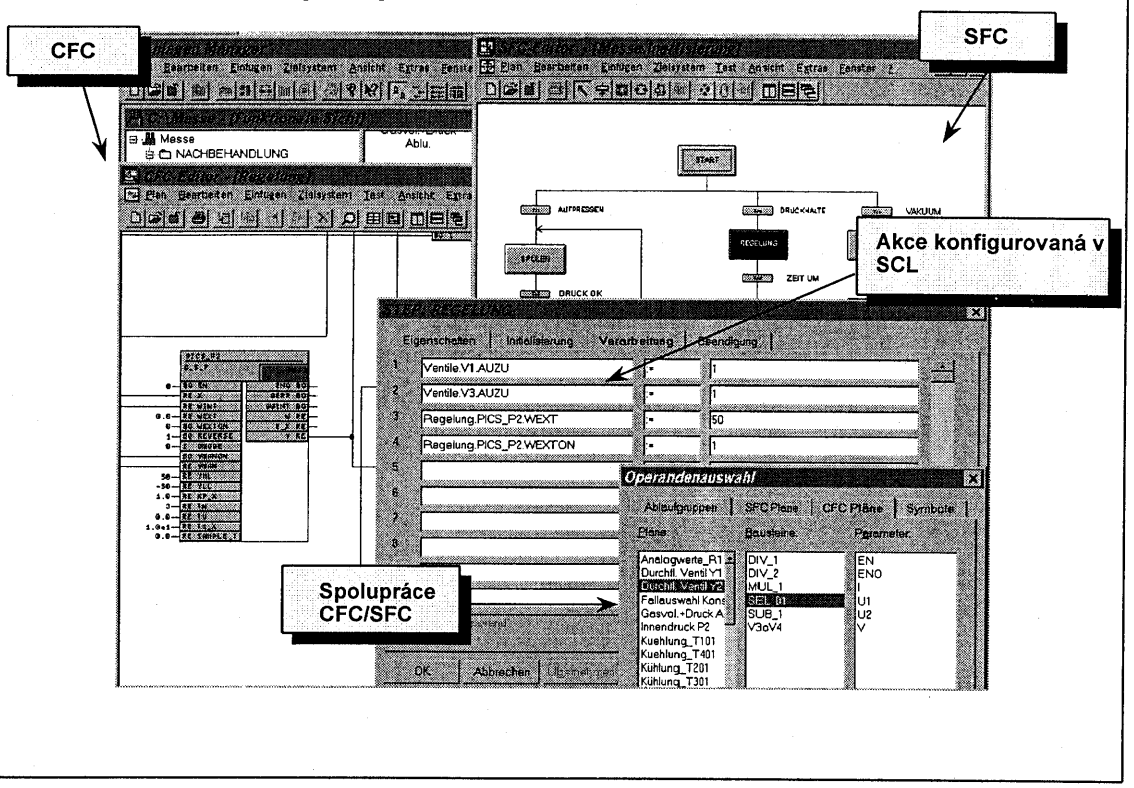
SFC-Editor umožňuje grafickou tvorbu plánu programu. Uživatel se nemusí zabývat algoritmy nebo možnostmi zařízení ale může se soustředit na technologické aspekty konfigurace.
Po vytvoření plánu lze přepnout do detailního zobrazení konfigurace a lze definovat parametry jednotlivých prvků, tj. lze konfigurovat akce (kroky) a podmínky (transice).

Po konfiguraci je vytvořen proveditelný kód. Tento kód lze po nahrání do CPU testovat pomocí SFC.

**Omezení plánu**

| | |
|---|---|
| • řetězců v plánu | 1 |
| • kroků v plánu | 2 ... 255 |
| • transic v plánu | 1 ... 255 |
| • instrukcí v kroku | <= 50 |
| • podmínek v transici | <= 10 |

## Spolupráce CFC/SFC a SCL




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_13cz.48

Školicí středisko
firmy E&A spol. s r.o., MB

**SFC**

- akce a transice lze konfigurovat v jazyce SCL
- přímý přístup k instancím bloků CFC
- společná správa dat a tvorba kódu s CFC
- integrace do prostředí STEP7 SIMATIC Manageru

# Řešení cvičení

SIMATIC S7

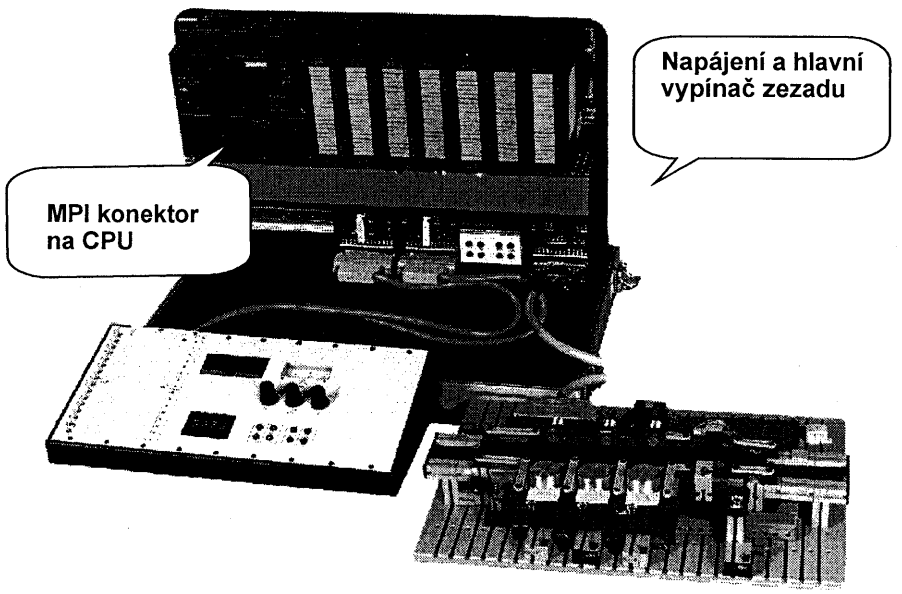


Datum: 24.11.2002
Soubor: PRO2_14cz.1

Information and Training Center
Knowledge for Automation

**Obsah** **Strana**

## Konfigurace cvičné sestavy




SIMATIC S7


Datum: 24.11.2002
Soubor: PRO2_14cz.2



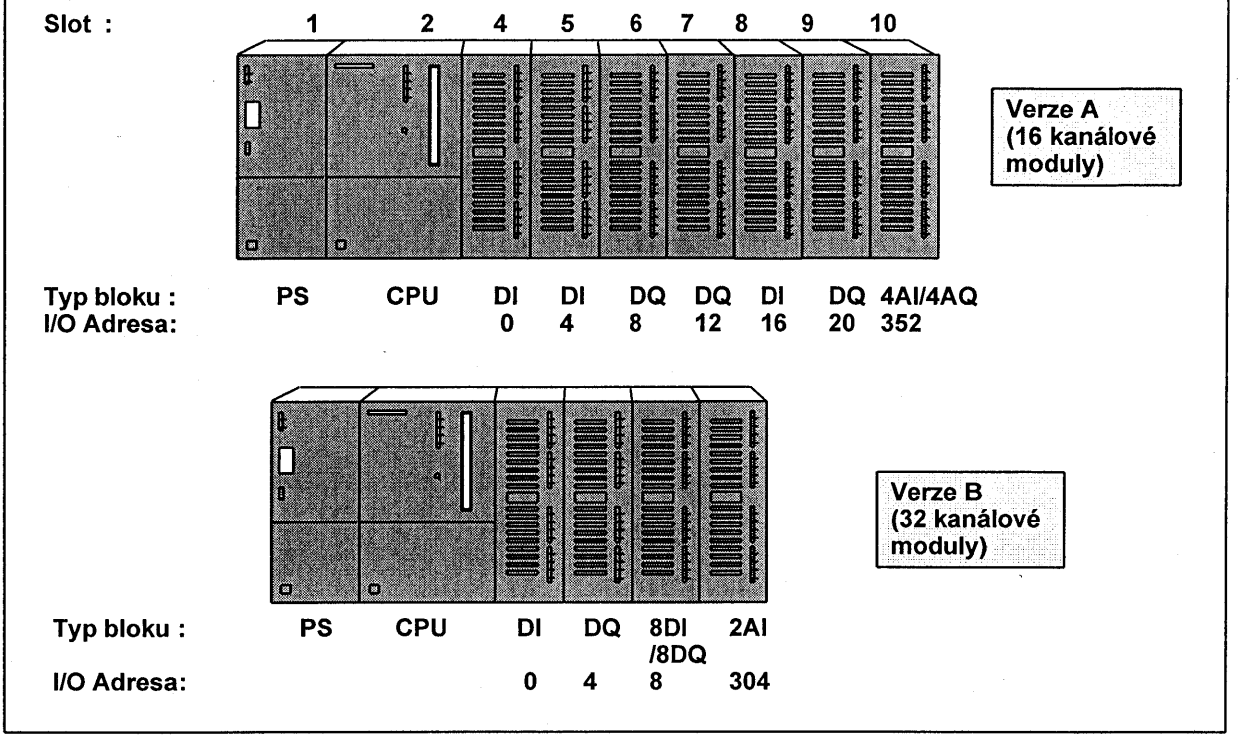
**Obsah cvičné sestavy**

Cvičná sestava obsahuje následující části:
- řídící systém S7-300 s CPU 314
- vstupní a výstupní digitální a analogové moduly
- simulátor s digitální a analogovou částí
- model dopravníku

**Poznámka**

Cvičná sestava je dodávána také s řídícím systémem S7-400.

## Cvičná sestava S7-300




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_14cz.3

Information and Training Center
Knowledge for Automation

**Konfigurace** Cvičný model je dostupný ve dvou variantách:

**Verze A**

| | | |
|---|---|---|
| Slot 1: | Napájecí zdroj 24V/5A | |
| Slot 2: | CPU 314 | |
| Slot 4: | Digitální vstupy 16x24V | Přepínače simulátoru |
| Slot 5: | Digitální vstupy 16x24V | Palcový přepínač |
| Slot 6: | Digitální výstupy 16x24V 0.5A | Výstupní LED diody simulátoru |
| Slot 7: | Digitální výstupy 16x24V 0.5A | Display |
| Slot 8: | Digitální vstupy 16x24V | Vstupy dopravníku |
| Slot 9: | Digitální výstupy 16x24V 0.5A | Výstupy dopravníku |
| Slot 10: | Analogový modul 4 AI/4 AO | Analogový simulátor |

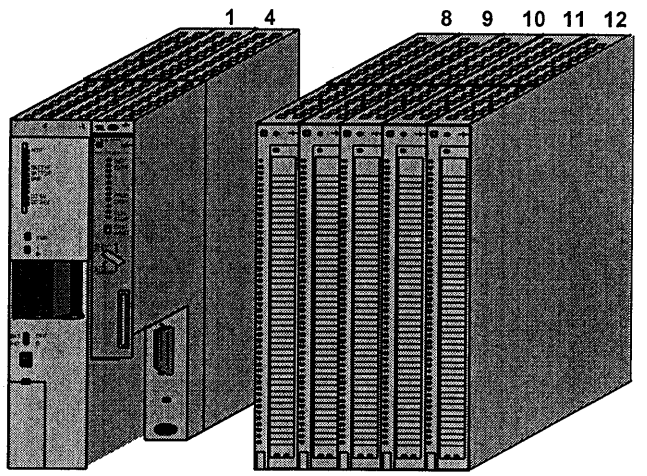
**Verze B**

| | | |
|---|---|---|
| Slot 1: | Napájecí zdroj 24V/5A | |
| Slot 2: | CPU 314 | |
| Slot 4: | Digitální vstupy 32x24V | Přepínače simulátoru a palcový přepínač |
| Slot 5: | Digitální výstupy 32x24V/0,5A | LED diody simulátoru a display |
| Slot 6: | Digitální modul 8X24V/ 8x24V 0.5A | model dopravníku |
| Slot 7: | Analogový modul | Analogový simulátor |

**Adresy** CPU 312-314 používá pevnou adresaci tj. bytová adresa modulů je odvozena od pozice modulu vůči CPU.

Ostatní CPU (od CPU 315-2 a S7-400) mají adresu modulu definovánu při parametrizaci CPU programem *HWConfig*.

## Cvičná sestava S7-400






Datum: 24.11.2002
Soubor: PRO2_14cz.4



**Konfigurace**

Řídící systém je dodán v následující konfiguraci:

| | | |
|---|---|---|
| Slot 1,2 a 3: | Napájecí zdroj 220V/20A | |
| Slot 4: | CPU 412, 414 nebo CPU 416 | |
| Slot 5, 6 a 7: | nepoužitý (použitelný pouze s M7) | |
| Slot 8: | Digitální vstupy 32x24V | Vstupy simulátoru |
| Slot 9: | Digitální vstupy 32x24V | Vstupy modelu |
| Slot 10: | Digitální výstupy 32x24V 0.5A | Výstupy simulátoru |
| Slot 11: | Digitální výstupy 16x24V 0.5A | Výstupy dopravníku |
| Slot 12: | Analogový modul | Adresovaný ze simulátoru |

**Adresy**

U S7-400 jsou uvedené základní adresy platné pouze při implicitní adresaci jestliže CPU neobsahuje konfigurační parametrická data (SDB).

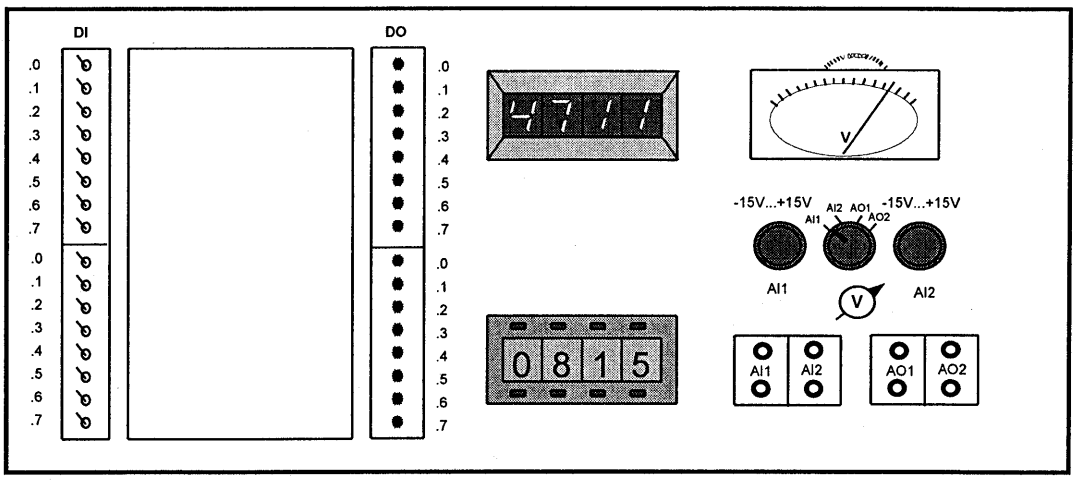
Implicitní adresy modulů v centrálním nosiči S7-400 jsou určeny podle následujícího vztahu:

Digitální moduly: (Slot - 1) x 4

Analogové moduly: (Slot - 1) x 64 + 512

Počáteční adresa modulů je u CPU S7-400 volně parametrizovatelná pomocí programu *HWConfig*.

# Konfigurace simulátoru







Datum: 24.11.2002
Soubor: PRO2_14cz.5



**Konfigurace**

Simulátor je k řídícímu systému připojen pomocí dvou kabelů s konektory a je rozdělen na 3 části:

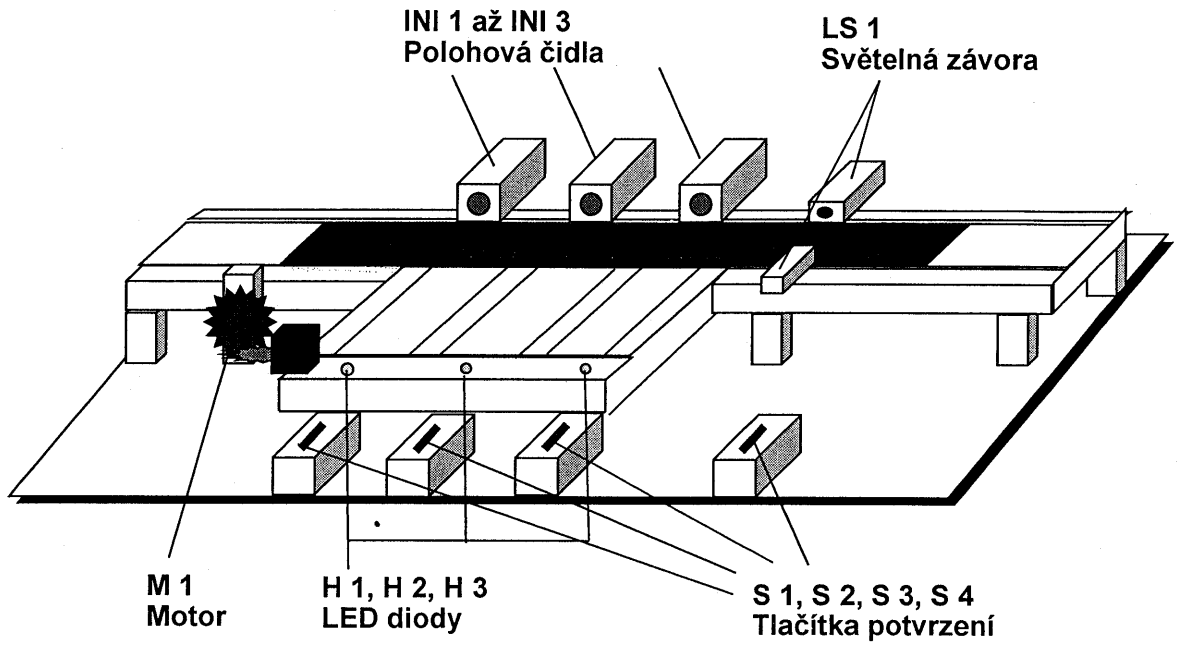
- binární část s 16 přepínači a 16 LED diodami
- digitální část s 4 místným palcovým přepínačem a digitálním displayem v BCD kódu
- analogová část s voltmetrem na zobrazení analogových vstupních kanálů 0 a 1 nebo výstupních analogových kanálů 0 a 1. Výběr zobrazovaného kanálu se provádí pomocí přepínače. K nastavování analogových vstupů slouží 2 samostatné potenciometry.

**Adresace**

V uživatelském programu lze použít následující adresy:

| Vstupy/Výstupy | S7-300 A | S7-300 B | S7-400 |
|---|---|---|---|
| Přepínače | IW 0 | IW 0 | IW 28 |
| LED diody | QW 8 | QW 4 | QW 36 |
| Palcový přepínač | IW 4 | IW 2 | IW 30 |
| Display | QW 12 | QW 6 | QW 38 |
| Analogové kanály | PIW 352/354 | PIW 304/306 | PIW 1216/1218 |

## Konfigurace modelu dopravníku




SIMATIC S7



Datum: 24.11.2002
Soubor: PRO2_14cz.6



**Konfigurace**

Obrázek nahoře ukazuje konfiguraci modelu dopravníku s jeho čidly a výstupy.

Adresy u modelu S7-400 odpovídají implicitním adresám jednotlivých modulů. Změny v konfiguraci adres lze provést s pomocí *HWConfigu*.

Pro změnu adres jednotlivých modulů si stačí zapamatovat, že vstupní signály z dopravníku jsou připojeny na druhý vstupní modul (slot 9) a výstupní signály jsou připojeny na druhý výstupní modul (slot 11).

Ve všech případech jsou signály ze simulátoru připojeny na první vstupní a první výstupní modul.

**Adresy**

| S7-300 A | S7-300 B | S7-4000 | Vstupy/Výstupy | Symbol |
|---|---|---|---|---|
| I 16.0 | I 8.0 | I 32.0 | Světelná závora | LS1 |
| I 16.1 | I 8.1 | I 32.1 | Potvrzení prac. 1 | S1 |
| I 16.2 | I 8.2 | I 32.2 | Potvrzení prac. 2 | S2 |
| I 16.3 | I 8.3 | I 32.3 | Potvrzení prac. 3 | S3 |
| I 16.4 | I 8.4 | I 32.4 | Konečné potvrzení | S4 |
| I 16.5 | I 8.5 | I 32.5 | Pracoviště 1 | INI1 |
| I 16.6 | I 8.6 | I 32.6 | Pracoviště 2 | INI2 |
| I 16.7 | I 8.7 | I 32.7 | Pracoviště 3 | INI3 |
| Q 20.1 | Q 8.1 | Q 40.1 | LED dioda pracoviště 1 | H1 |
| Q 20.2 | Q 8.2 | Q 40.2 | LED dioda pracoviště 2 | H2 |
| Q 20.3 | Q 8.3 | Q 40.3 | LED dioda pracoviště 3 | H3 |
| Q 20.4 | Q 8.4 | Q 40.4 | LED linky | H4 |
| Q 20.5 | Q 8.5 | Q 40.5 | Dopravník dopředu | K1_LAUF |
| Q20.6 | Q 8.6 | Q 40.6 | Dopravník dozadu | K2_LAUF |
| Q 20.7 | Q 8.7 | Q 40.7 | Houkačka | TUT1 |

# Řešení cvičení 1.1: Skok po odečítání:

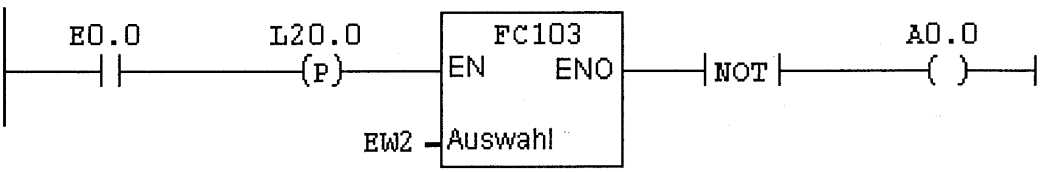
```
FUNCTION FC 11 : VOID
TITLE =Exercise 1.1 : Skok po odečítání
//Version for 16Bit-SM
AUTHOR : PT41
FAMILY : A4_0
NAME : ST7PRO2
VERSION : 0.0

BEGIN
NETWORK
TITLE =

      L    IW   4;          //Palcový přepínač
      BTD ;                 //BCD --> Integer
      L    IW   0;          //Input word 0
       BTD;
       -D;
      JN   NEG;             //Skok, je-li výsledek negativní
      L    IW   0;
      JU   END;
NEG:  L    0;
END:  T    QW  12;          //display
      END_FUNCTION
```

# Řešení cvičení 1.2: Skok po násobení :

```
FUNCTION FC 12 : VOID
TITLE =Exercise 1.2 : Skok po násobení
//Version for 16Bit-SM
AUTHOR : PT41
FAMILY : A4_0
NAME : ST7PRO2
VERSION : 0.0

BEGIN
NETWORK
TITLE =

      L    IW   4;          //Palcový přepínač
      BTD;                  //BCD --> Integer
      L    IW   0;          //Přepínače simulátoru
      BTD;
      *I;
      JO   OVL;             //Skok při přetečení (overflow)
      DTB;                  //Integer --> BCD
      JU   END;
OVL:  L    0;
END:  T    QW  12;          //display
      END_FUNCTION
```

# Řešení cvičení 1.3: Selektor:

## OB1




FUNCTION FC 13: VOID

```
VAR_INPUT
Select: INT;
END_VAR
BEGIN
L    #Select;
OW   W#16#FF00;          // kontrola select >255 nebo
JCN  Err;                // záporný
JL   GT5;                // skok na návěští je-li Accu1-L-L >5
JU   Err;                // je-li select = 0 (nepřípustné)
JU   Dr_1;               // dopravník vpravo (select=1)
JU   Dr_2;               // dopravník vlevo (select=2)
JU   Dr_3;               // dopravník stop (select=3)
JU   Ho_1;               // Houkej
JU   Ho_2;               // zmlkni
GT5: JU   Err;
Dr_1: S   Q    4.5;      // dopravník vpravo
      R   Q    4.6;
      JU  End;
Dr_2: S   Q    4.6;      // dopravník vlevo
      R   Q    4.5;
      JU  End;
Dr_3: R   Q    4.5;      // dopravník stop
      R   Q    4.6;
      JU  End;
Ho_1: S   Q    4.7;      // houkej
      JU  End;
Ho_2: R   Q    4.7;      // zmlkni
      JU  End;
Err:  CLR;
      SAVE;
End:  BE;
END_FUNCTION
```

# Řešení cvičení 2.1: Výpočet mocniny

```
FUNCTION FC 21 : VOID
TITLE =Exercise 2.1: Calculation of Exponents
//Version for 16Bit-SM
AUTHOR : PT41
FAMILY : A4_0
NAME : ST7PRO2
VERSION : 0.0

BEGIN
NETWORK
TITLE =

L     IB    4;          //nahraj BCD číslo
BTI;                    //BCD --> Integer
 PUSH;                  //ACCU1 --> ACCU2
*D;                     //ACCU1 *ACCU2 --> ACCU1
PUSH;                   //ACCU1 --> ACCU2
PUSH;                   //nezbytné u S7-400: ACCU2 -> ACCU3
*D;                     // ACCU1 *ACCU2 --> ACCU1
*D;                     // ACCU1 *ACCU2 --> ACCU1
 DTB;                   //DINT--> BCD (podle potřeby)
T     QW  12;           //Display
END_FUNCTION
```

# Řešení cvičení 2.2: Výměna dat v ACCU1:

```
FUNCTION FC 22 : VOID
TITLE =Exercise 2.2: Data exchange in ACCU1
//Exercise 2.2: Data exchange in ACCU1
//Version for 16Bit-SM
AUTHOR : PT41
FAMILY : A4_0
NAME : ST7PRO2
VERSION : 0.0

BEGIN
NETWORK
TITLE =

L     IW   4;          //palcový přepínač - BCD kód
CAW;                   //záměna 2 bytů v ACCU1-L
T     QW  12;          //display
END_FUNCTION
```

# Řešení cvičení 2.3: Tvorba doplňků:

```
FUNCTION FC 23 : VOID
TITLE =Exercise 1.6: Forming the Ones Complement
//Exercise 2.3: Forming Complements
//Version for 16Bit-SM
AUTHOR : PT41
FAMILY : A4_0
NAME : ST7PRO2
VERSION : 0.0

BEGIN
NETWORK
TITLE =Ones complement in STL

L     IW   0;          //nahraj vstupní slovo v BCD kódu
INVI;                  //vytvoř první doplněk
T     QW   0;          //výsledek do výstupního slova
END_FUNCTION
```

# Řešení cvičení 3.1: Výpočet vzdálenosti:

```
FUNCTION FC 31 : REAL
TITLE =Exercise 3.1: Calculating the Distance
//Exercise 3.1: Mathematical Functions
AUTHOR : PT41
FAMILY : A4_0
NAME : ST7PRO2
VERSION : 0.0

VAR_INPUT
  X1:  REAL;
  Y1:  REAL;
  X2:  REAL;
  Y2:  REAL ;
END_VAR
VAR_TEMP
  XSquare : REAL;
END_VAR
BEGIN
  NETWORK
  TITLE =
     L    X1;              // X-souřadnice P1
     L    X2;              // X-souřadnice P2
     -R;                   // odečti (X1-X2)
     SQR;                  // (X1-X2)^2
     T    XSquare;         // ulož do lokální proměnné
     L    Y1;              // Y-souřadnice P1
     L    Y2;              // Y-souřadnice P2
     -R;                   // odečti (Y1-Y2)
     SQR;                  // (Y1-Y2)^2
     L    XSquare;         // nahraj lokální proměnnou
     +R;                   // součet
     SQRT;                 // 2. odmocnina
     T    RET_VAL;         // výsledek

END_FUNCTION
```

# Řešení cvičení 4.1: Smyčka s nepřímou adresací (1)

```
FUNCTION FC 41 : VOID
TITLE =Exercise 4.1:Loop programming with memory-indirect addressing
//Exercise 4.1: Loop programming with memory-indirect addressing

VAR_TEMP
  Tank : ARRAY  [1 .. 100 ] OF INT ;
  Counter : INT ;
  Ini_Value : INT ;
  Pointer : DWORD ;
END_VAR
BEGIN
NETWORK
TITLE = Loop programming

    L    P#0.0;                  // nahraj adresu 1. tanku
    T    #Pointer;                 // a ulož do ukazatele
    L    1;                        // nahraj 1
    T    #Ini_Value;               // a ulož do proměnné Ini_Value
    L    100;                      // počet cyklů 100
BEGN: T    #Counter;               // ulož do čitače smyčky
    L    #Ini_Value;
    T    LW [#Pointer];            // ulož Ini_Value do Tank[i]
    L    1;                        // zvyš obsah ACCU1(Ini_Value)
    +I   ;                         // o 1
    T    #Ini_Value;               // a ulož do Ini_Value
    L    #Pointer;                 // nahraj ukazatel do Accu1
    L    P#2.0;                  // zvedni adresu bytu o 2
    +D   ;                         // a ulož
    T    #Pointer;                 // do ukazatele
    L    #Counter;                 // načti čitač smyčky,
    LOOP  BEGN;                    // odečti 1 a pokračuj ve smyčce

NETWORK
TITLE =Network for the status display of the Local Data

    L    #Tank[1];                 // zobrazení hodnoty proměnné
    L    #Tank[2];
    L    #Tank[3];                 //          "
    L    #Tank[4];
    L    #Tank[5];
    L    #Tank[6];
    L    #Tank[7];
    L    #Tank[8];
    L    #Tank[9];
    L    #Tank[10];
    L    #Tank[64];
    L    #Tank[100];
END_FUNCTION
```

# Řešení cvičení 4.2: Smyčka s nepřímou adresací (2)

```
FUNCTION FC 42 : VOID
TITLE =Exercise 4.2:Loop programming with register indirect addressing
//Exercise 4.2: Loop programming with register indirect addressing

VAR_TEMP
  Tank : ARRAY  [1 .. 100 ] OF INT ;
  Loop counter : INT ;
  Ini_Value : INT ;
END_VAR
BEGIN
NETWORK
TITLE =Loop programming

    LAR1  P#0.0;              // nahraj adresu 1. tanku do AR1
    L     1;                  // 1 --> Accu1 (Ini_W.)
    L     100;                // 100 --> Accu1 (loopc.); 1 --> Accu2 (Ini_Value)
BEGN: TAK   ;                 // zaměň ACCU1/2, Loop counter v Accu2, Ini_V. v Accu1
    T     LW [AR1,P#0.0];     // Ini_Value --> Tank[i]
    INC   1;                  // přičti 1 k Ini_Value
    +AR1  P#2.0;              // zvedni AR1 o 2 byty
    TAK   ;                   // Loop counter --> Accu1, Ini_V. --> Accu2
    LOOP  BEGN;               // odečti loop counter a pokračuj ve smyčce

NETWORK
TITLE =Network for the status display of the Local Data

    L     #Tank[1];
    L     #Tank[2];
    L     #Tank[3];
    L     #Tank[4];
    L     #Tank[5];
    L     #Tank[6];
    L     #Tank[7];
    L     #Tank[8];
    L     #Tank[9];
    L     #Tank[10];
    L     #Tank[64];
    L     #Tank[100];
END_FUNCTION
```

# Řešení cvičení 4.3: Výpočet celk.součtu a průměru

```
FUNCTION FC 43 : VOID
TITLE =Exercise 4.2: Calculation of sum and mena value
//Exercise 4.2: Calculation of sum and mean value
VAR_INPUT
 Measured_values : ANY ;
END_VAR
VAR_OUTPUT
  Sum : REAL ;
  Mean_value : REAL ;
END_VAR
VAR_TEMP
  Loop_counter : WORD ;
  DB_No : WORD ;
END_VAR
BEGIN
    L    P##Meas_values;      // nahraj ukazatel "ANY"
    LAR1  ;                   // ukazatel do AR1
    L    B [AR1,P#1.0];       // načti identifikátor datového typu
    L    8;                   // načti identifikátor typu REAL (16#08)
    ==I   ;
    JC   REAL;                // skok je-li to datový typ REAL
    NOP   0;                  // není-li to REAL
    CLR   ;                   // RLO=0
    SAVE  ;                   // BR =0
    L    L#-1;                // nahraj neplatné Real číslo
    T    #Sum;
    T    #Mean_value;
    BEU   ;
REAL: NOP   0;                // je-li to REAL
    L    W [AR1,P#2.0];       // nahraj počet prvků
    T    #Loop_counter;       // inicializuj loop counter
    L    W [AR1,P#4.0];       // nahraj číslo DB nebo 0
    T    #DB_No;              // je-li DB_No=0 potom OPN DB[DB_No]=NOP
    OPN   DB [#DB_No];        // Run time error!!, pokud DB neexistuje
    L    D [AR1,P#6.0];       // nahraj ukazatel oblasti
    LAR1  ;                   // do AR1, chyba !! pokud je identifikátor "DI"
    L    0.000000e+000;       // 0 do Accu1 (Sum =0.0)
    L    #Loop_counter;       // Counter do ACCU1; Sum=0 do ACCU2
BEGN: TAK   ;                 // Loopc v ACCU2, Sum v ACCU1
    ENT   ;                   // Loopc v ACCU3, Sum v ACCU2
    L    D [AR1,P#0.0];       // prvek pole v ACCU1
    +R    ;                   // Sum v ACCU1, Loopc v ACCU2
    +AR1  P#4.0;              // zvedni AR1 o 4 byty
    TAK   ;                   // Loopc. v ACCU1, Sum v ACCU2
    LOOP  BEGN;               // sniž loopc a pokračuj ve smyčce
    TAK   ;                   // Sum v ACCU1
    T    #Sum;                // Sum v #Sum
    L    #Loop_counter;       // Sum v ACCU2, počet v ACCU1
    DTR   ;                   // konverze INT(16-Bit) --> REAL
    /R    ;                   // průměrná hodnota v ACCU1
    T    #Mean_value;         // průměrná hodnota do parametru #Mean value
    SET   ;                   // Set RLO na 1
    SAVE  ;                   // RLO --> BR
END_FUNCTION
```

# Řešení cvičení 5.1: Čtení systémových hodin pomocí SFC 1(READ_CLK)

```
FUNCTION FC 51 : VOID
TITLE =Exercise 5.1: Read system clock with SFC 1(READ_CLK)
//Exercise 5.1: Read system clock with SFC 1(READ_CLK)
//Version for 16Bit-SM
AUTHOR : PT41
FAMILY : A2_0
NAME : ST7PRO2
VERSION : 0.0


VAR_TEMP
  Date_Time : DATE_AND_TIME ;          //aktuální datum a čas
  RET_VAL_SFC1 : INT ;                 //výsledek SFC 1
END_VAR
BEGIN
NETWORK
TITLE =SFC 1 (READ_CLK) call

    CALL "READ_CLK" (
       RET_VAL          := #Ret_Val_SFC1,
       CDT              := #Date_Time);

    NOP   0;
NETWORK
TITLE =Display Hour, Minute

    LAR1  P##Date_Time       // určení adresy proměnné
    L     LB [AR1, P#3.0];   // načti hodiny
    T     QB     12;         // a zobraz
    L     LB [AR1, P#4.0];   // načti minuty
    T     QB     13;         // a zobraz

END_FUNCTION
```

# Řešení cvičení 6.1: Tvorba FB1 pro pracoviště (část 1.)

```
FUNCTION_BLOCK "Station"
TITLE =
VERSION : 0.1
VAR_INPUT
  Initial : BOOL ;
  Proxy_Switch : BOOL ;
  Acknowledge : BOOL ;
  Clock_Bit : BOOL ;
END_VAR
VAR_OUTPUT
  LED : BOOL ;
  Transp_req : BOOL ;
END_VAR
VAR_IN_OUT
  Conv_busy : BOOL ;
END_VAR
VAR
  State : STRUCT
   Busy : BOOL ;
   Completed : BOOL ;
   Waiting : BOOL ;
  END_STRUCT ;
  Aux_1 : BOOL ;
  Aux_2 : BOOL ;
END_VAR

BEGIN
NETWORK
TITLE =Initializing
//With initializing, the basic state, i.e. Station is set in state
//"Busy"


     A    #Initial;
     R    #State.Waiting;
     S    #State.Busy;
     R    #State.Completed;
     R    #Conv_busy;

NETWORK
TITLE =State: Busy
//Processing runs in this state. Processing ends when the operator
//acknowledges, with the workplace acknowledgement button, the
//finishing of the piece.

     AN   #State.Busy;
     JC   REDY;
     S    #LED;
     R    #Transp_req;
     A    #Acknowledge;
     R    #State.Busy;
     R    #LED;
     S    #State.Completed;
```

// (Continued next page)

# Řešení cvičení 6.1: Tvorba FB1 pro pracoviště (část 2.)

```
NETWORK
TITLE =State: Completed
//In the finished state, waiting takes place until the finished part can be laid on the
// transport belt. The input signal #Band_frei signals if the transport belt is free

REDY: AN   #State.Completed;
      JC   WAIT;
      A    #Clock_Bit;
      =    #LED;
      AN   #Conv_busy;
      A    #Proxy_Switch;
      S    #Conv_busy;
      S    #Transp_req;
      A    #Transp_req;
      A    #Proxy_Switch;
      FN   #Aux_1;
      R    #State.Completed;
      S    #State.Waiting;

NETWORK
TITLE =State: Waiting
//Waits for a new raw part. Arrival of a new part is signalled by
//the transport belt's proximity switch

WAIT: AN   #State.Waiting;
      JC   ENDE;
      R    #LED;
      A    #Proxy_Switch; //new raw material has arrived
      R    #Transp_req; // stop belt
      A    #Proxy_Switch;
      FN   #Aux_1; //after removal, processing commences
      R    #Conv_busy; enable conveyor once again
      R    #State.Waiting;
      S    #State.Busy;
ENDE: BEU  ;

END_FUNCTION_BLOCK
```

# Řešení cvičení 6.1: Tvorba FB1 pro pracoviště, OB1

```
ORGANIZATION_BLOCK OB 1
TITLE =
VERSION : 0.1


VAR_TEMP
  OB1_EV_CLASS : BYTE ;    //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
  OB1_SCAN_1 : BYTE ; //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
  OB1_PRIORITY : BYTE ;    //1 (Priority of 1 is lowest)
  OB1_OB_NUMBR : BYTE ;  //1 (Organization block 1, OB1)
  OB1_RESERVED_1 : BYTE ;  //Reserved for system
  OB1_RESERVED_2 : BYTE ;  //Reserved for system
  OB1_PREV_CYCLE : INT ;   //Cycle time of previous OB1 scan (milliseconds)
  OB1_MIN_CYCLE : INT ;    //Minimum cycle time of OB1 (milliseconds)
  OB1_MAX_CYCLE : INT ;    //Maximum cycle time of OB1 (milliseconds)
  OB1_DATE_TIME : DATE_AND_TIME ;  //Date and time OB1 started
END_VAR
BEGIN
NETWORK
TITLE =Control: Station_1

    CALL "Station" , "Station_DB" (
       Initial              := I 0.0,
       Proxy_Switch         := "INI1",
       Acknowledge:= "S1",
       Clock_Bit            := M  10.1,
       LED                  := "H1");

END_ORGANIZATION_BLOCK
```

# Řešení cvičení 6.2: Tvorba FB2 pro dopravník (část 1.)

```
FUNCTION_BLOCK "Transport"
TITLE =Controlling the conveyor
VERSION : 0.1
VAR_INPUT
  Initial : BOOL ;
  L_Barrier : BOOL ;
  Acknowledge : BOOL ;
  Transp_req : BOOL ;
  Clock_Bit : BOOL ;
END_VAR
VAR_OUTPUT
  LED : BOOL ;
  Conv_right : BOOL ;
  Conv_left : BOOL ;
END_VAR
VAR
  State : STRUCT
   Waiting : BOOL ;
   Conv_right : BOOL ;
   Assembly : BOOL ;
   Conv_left : BOOL ;
  END_STRUCT ;
END_VAR
BEGIN
NETWORK
TITLE =Initialization
      A    #Initial;
      S    #State.Waiting;
      R    #State.Conv_right;
      R    #State.Assembly;
      R    #State.Conv_left;
NETWORK
TITLE =State: Waiting
//The belt waits in this state for a finished part.
      AN   #State.Waiting;
      JC   RECH;
      R    #Conv_right;
      R    #Conv_left;
      R    #LED;
      A    #Transp_req;
      R    #State.Waiting;
      S    #State.Conv_right;
NETWORK
TITLE =State: Conv_right
//This state describes the finished part's transport in the direct
//final assembly
RECH: AN   #State.Conv_right;
      JC   ENDM;
      S    #Conv_right;
      A    #Clock_Bit;
      =    #LED;
      AN   #L_Barrier;
      R    #Conv_right;
      R    #State.Conv_right;
      S    #State.Assembly;
```

(Continued on next page)

# Řešení cvičení 6.2: Tvorba FB2 pro dopravník (část 2.)

```
NETWORK
TITLE =State: Assembly
//In this state, the finished part is removed and a new raw piece is laid
//on the belt. After that, the raw piece's transport in the direction of the empty
//processing station is started with S4.
//
ENDM: AN    #State.Assembly;
      JC    LINK;
      S     #LED;
      A     #Acknowledge;
      R     #LED;
      R     #State.Assembly;
      S     #State.Conv_left;



NETWORK
TITLE =State: Conv_left
//In the state, the raw piece's transport to the station takes place, that delivered
//the finished piece.
LINK: AN    #State.Conv_left;
      JC    ENDE;
      S     #Conv_left;
      A     #Clock_Bit;
      =     #LED;
      AN    #Transp_req;
      R     #Conv_left;
      R     #State.Conv_left;
      S     #State.Waiting;
ENDE: BEU   ;

END_FUNCTION_BLOCK
```

# Řešení cvičení 6.2: Tvorba FB2 pro dopravník - OB1

```
ORGANIZATION_BLOCK OB 1
TITLE =
VERSION : 0.1


VAR_TEMP
  OB1_EV_CLASS : BYTE ;      //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
  OB1_SCAN_1 : BYTE ;        //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
  OB1_PRIORITY : BYTE ;      //1 (Priority of 1 is lowest)
  OB1_OB_NUMBR : BYTE ;      //1 (Organization block 1, OB1)
  OB1_RESERVED_1 : BYTE ;    //Reserved for system
  OB1_RESERVED_2 : BYTE ;    //Reserved for system
  OB1_PREV_CYCLE : INT ;     //Cycle time of previous OB1 scan (milliseconds)
  OB1_MIN_CYCLE : INT ;      //Minimum cycle time of OB1 (milliseconds)
  OB1_MAX_CYCLE : INT ;      //Maximum cycle time of OB1 (milliseconds)
  OB1_DATE_TIME : DATE_AND_TIME ;  //Date and time OB1 started
END_VAR
BEGIN
NETWORK
TITLE =Controlling: Station_1

    CALL "Station" , "Station_DB" (
        Initial                 := I     0.0,
        Proxy_Switch            := "INI1",
        Acknowledge             := "S1",
        Clock_Bit               := M     10.1,
        LED                     := "H1",
        Transp_req              := "Transport_DB".Transp_req);

NETWORK
TITLE =Controlling: Conveyor


    CALL "Transport" , "Transport_DB" (
        Initial                 := I     0.0,
        L_Barrier               := "LS1",
        Acknowledge             := "S4",
        Clock_Bit               := M     10.1,
        LED                     := "H4",
        Conv_right              := "K1 CONV",
        Conv_left               := "K2_CONV");

END_ORGANIZATION_BLOCK
```

# Řešení cvičení 6.3: Tvorba FB10 (část 1.)

```
FUNCTION_BLOCK FB 10
TITLE =
VERSION : 0.1


VAR
  Station_1 : "Station";
  Station_2 : "Station";
  Station_3 : "Station";
  Transport : "Transport";
  Conv_busy : BOOL ;
END_VAR
VAR_TEMP
  Trans_1 : BOOL ;
  Trans_2 : BOOL ;
  Trans_3 : BOOL ;
  Trans : BOOL ;
END_VAR
BEGIN
NETWORK
TITLE =Station_1

    CALL #Station_1 (
        Initial             := I     0.0,
        Proxy_Switch        := "INI1",
        Acknowledge         := "S1",
        Clock_Bit           := "CLOCK_BIT",
        LED                 := "H1",
        Transp_req          := #Trans_1,
        Conv_busy           := #Conv_busy);

NETWORK
TITLE =Station_2


    CALL #Station_2 (
        Initial             := I     0.0,
        Proxy_Switch        := "INI2",
        Acknowledge         := "S2",
        Clock_Bit           := "CLOCK_BIT",
        LED                 := "H2",
        Transp_req          := #Trans_2,
        Conv_busy           := #Conv_busy);

NETWORK
TITLE =Station_3

    CALL #Station_3 (
        Initial             := I     0.0,
        Proxy_Switch        := "INI3",
        Acknowledge         := "S3",
        Clock_Bit           := "CLOCK_BIT",
        LED                 := "H3",
        Transp_req          := #Trans_3,
        Conv_busy           := #Conv_busy);

// (Continued on next page)
```

# Řešení cvičení 6.3: Tvorba FB10 - OB1

```
NETWORK
TITLE =Logic: Transp_req
//Forming the logic for #Transp_req
    O    #Trans_1;
    O    #Trans_2;
    O    #Trans_3;
    =    #Trans;


NETWORK
TITLE =Transport

    CALL #Transport (
       Initial                 := I     0.0,
       L_Barrier               := "LS1",
       Acknowledge             := "S4",
       Transp_req              := #Trans,
       Clock_Bit               := "CLOCK_BIT",
       LED                     :=  "H4",
       Conv_right              := "K1_CONV",
       Conv_left               := "K2_CONV");

END_FUNCTION_BLOCK


ORGANIZATION_BLOCK OB 1
TITLE =
VERSION : 0.1

VAR_TEMP
  OB1_EV_CLASS : BYTE ;       //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
  OB1_SCAN_1 : BYTE ;         //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
  OB1_PRIORITY : BYTE ;       //1 (Priority of 1 is lowest)
  OB1_OB_NUMBR : BYTE ;       //1 (Organization block 1, OB1)
  OB1_RESERVED_1 : BYTE ;     //Reserved for system
  OB1_RESERVED_2 : BYTE ;     //Reserved for system
  OB1_PREV_CYCLE : INT ;      //Cycle time of previous OB1 scan (milliseconds)
  OB1_MIN_CYCLE : INT ;       //Minimum cycle time of OB1 (milliseconds)
  OB1_MAX_CYCLE : INT ;       //Maximum cycle time of OB1 (milliseconds)
  OB1_DATE_TIME : DATE_AND_TIME ;  //Date and time OB1 started
END_VAR

BEGIN
NETWORK
TITLE =

    CALL FB    10 , DB    10 ;

END_ORGANIZATION_BLOCK
```

# Řešení cvičení 6.4: Tvorba vlastního bloku čitačů - DB6, FB6

```
DATA_BLOCK DB 6
 FB 1
BEGIN
  CU := FALSE;
  R := FALSE;
  PV := 0;
  Q := FALSE;
  CV := 0;
  Edge_memory_bit_CU := FALSE;
  Edge_memory_bit_R := FALSE;
END_DATA_BLOCK

FUNCTION_BLOCK FB 6

VAR_INPUT
  CU : BOOL ;
  R : BOOL ;
  PV : INT ;
END_VAR
VAR_OUTPUT
  Q : BOOL ;
  CV : INT ;
END_VAR
VAR
  Edge_memory_bit_CU : BOOL ;
  Edge_memory_bit_R : BOOL ;
END_VAR
BEGIN
    A   #CU;                    // Start count up
    FP   #Edge_memory_bit_CU;   // Set edge memory bit
    JCN  NZHL;                  // Jump if no edge
    L    #CV;                   // Load current value
    L    1;                     // Load one
    +I   ;                      // 16 -Bit-Addition
    A    OV;                    // Overflow ?
    JCN  NOVL;                  // Jump if no overflow
    L    32767;                 // Otherwise: load maximum value
NOVL: T    #CV;                 // Transfer resul to CV
NZHL: NOP  0;                   // End count up
                                //
    A    #R;                    // Start reset
    FP   #Edge_memory_bit_R;          // Set edge memory bit
    JCN  NRCK;                  // Jump if no edge
    L    0;                     // Load 16-Bit const. 0
    T    #CV;                   // Transfer in current value
NRCK: NOP  0;                   // End reset
                                //
    L    #CV;                   // Start handling output Q
    L    #PV;                   // Load preset value
    >=I  ;                      // CV >= PV ?
    =    #Q;                    // Assign RLO the output Q
END_FUNCTION_BLOCK
```

# Řešení cvičení 7.2: Testování datového bloku (SFC 24: pouze S7 400)

```
FUNCTION FC 72 : INT
TITLE =
VERSION : 0.1


VAR_INPUT
  DB_No : WORD ;
END_VAR
VAR_TEMP
  I_DB_Length : WORD ;
  I_RET_VAL : INT ;
  I_Write_Protect : BOOL ;
END_VAR
BEGIN
NETWORK
TITLE =Testing DB

     CALL "TEST_DB" (
        DB_NUMBER    := #DB_No,
        RET_VAL      := #I_RET_VAL,
        DB_LENGTH    := #I_DB_Length,
        WRITE_PROT   := #I_Write_Protect);
     L    #I_RET_VAL;
     L    W#16#0;
     ==I  ;
     JC   DBOK;                    // DB v pracovní paměti
     TAK  ;
     L    W#16#80A1;
     ==I  ;
     JC   NODB;                    // DB není v CPU
     TAK  ;
     L    W#16#80B1;
     ==I  ;
     JC   NODB;                    // DB v pracovní paměti
     TAK  ;
     L    W#16#80B2;
     ==I  ;
     JC   DBLM;                    // DB pouze v editační paměti
NODB: L    -1;
     T    #RET_VAL;             // DB není v CPU
     BEU  ;
DBLM: L    1;
     T    #RET_VAL;             // DB pouze v editační paměti - unlinked
     BEU  ;
DBOK: L    0;
     T    #RET_VAL;             // DB v pracovní paměti

END_FUNCTION
```

# Řešení cvičení 7.3: Tvorba DB bloku (SFC 22)

```
ORGANIZATION_BLOCK OB 100
TITLE =
VERSION : 0.1


VAR_TEMP
  OB100_EV_CLASS : BYTE ;        //16#13, Event class 1, Entering event state, Event
                                     // logged in diagnostic buffer
  OB100_STRTUP : BYTE ;          //16#81/82/83/84 Method of startup
  OB100_PRIORITY : BYTE ;        //27 (Priority of 1 is lowest)
  OB100_OB_NUMBR : BYTE ;        //100 (Organization block 100, OB100)
  OB100_RESERVED_1 : BYTE ;      //Reserved for system
  OB100_RESERVED_2 : BYTE ;      //Reserved for system
  OB100_STOP : WORD ;            //Event that caused CPU to stop (16#4xxx)
  OB100_STRT_INFO : DWORD ;      //Information on how system started
  OB100_DATE_TIME : DATE_AND_TIME ;   //Date and time OB100 started
END_VAR

BEGIN
NETWORK
TITLE =Creating DB10

    CALL "CREAT_DB" (
       LOW_LIMIT      := W#16#A,      // číslo DB bloku (DB10)
       UP_LIMIT       := W#16#A,      //  "
       COUNT          := W#16#28,     // velikost 40 ( 40Bytů)
       RET_VAL        := MW    0,
       DB_NUMBER      := QW    12);

END_ORGANIZATION_BLOCK
```

# Řešení cvičení 7.4: Kopírování DB z Editační do Pracovní paměti (SFC 20)

```
ORGANIZATION_BLOCK OB 1
TITLE =
//Copying a DB from Load into Working Memory
VERSION : 2.10


VAR_TEMP
  OB1_EV_CLASS : BYTE ;      //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
  OB1_SCAN_1 : BYTE ;        //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
  OB1_PRIORITY : BYTE ;      //1 (Priority of 1 is lowest)
  OB1_OB_NUMBR : BYTE ;      //1 (Organization block 1, OB1)
  OB1_RESERVED_1 : BYTE ;    //Reserved for system
  OB1_RESERVED_2 : BYTE ;    //Reserved for system
  OB1_PREV_CYCLE : INT ;     //Cycle time of previous OB1 scan (milliseconds)
  OB1_MIN_CYCLE : INT ;      //Minimum cycle time of OB1 (milliseconds)
  OB1_MAX_CYCLE : INT ;      //Maximum cycle time of OB1 (milliseconds)
  OB1_DATE_TIME : DATE_AND_TIME ;  //Date and time OB1 started
END_VAR
BEGIN
NETWORK
TITLE =

    CALL "BLKMOV" (
       SRCBLK              := P#DB10.DBX 0.0 BYTE 40,
       RET_VAL             := QW   12,
       DSTBLK              := P#DB20.DBX 0.0 BYTE 40);
END_ORGANIZATION_BLOCK
```

# Řešení cvičení 7.5: Inicializace DB s "0" (SFC 21: FILL)

```
FUNCTION FC 75 : BOOL
TITLE =
// Exercise 7.5: Initializing DB (only S7-400)
VERSION : 0.1
VAR_INPUT
  DB_NUM : WORD ;
  INI : BYTE ;
END_VAR
VAR_TEMP
  I_RET_VAL : INT ;
  I_DB_Length : WORD ;
  I_WRITE_PROT : BOOL ;
  I_ANY : ANY ;
  DB_No : WORD ;
  I_INI : BYTE ;
  I_RET_VAL1 : INT ;
END_VAR
BEGIN
NETWORK
TITLE =
//Check if DB is in work memory and if necessary initialize
     CALL "TEST_DB" (
          DB_NUMBER          := #DB_NUM,
          RET_VAL            := #I_RET_VAL,
          DB_LENGTH          := #I_DB_Length,
          WRITE_PROT         := #I_WRITE_PROT);

     L    #I_RET_VAL;
     L    W#16#0;
     ==I  ;                          // DB v pracovní paměti
     AN   #I_WRITE_PROT;
     JC   OK;
     CLR  ;                          // Inicializace není možná
     =    #RET_VAL; // Return FALSE
     BEU  ;
OK:  LAR1 P##I_ANY;                  // nahraj ukazatel
     L    B#16#10;                   // Identifikátor ANY
     T    LB [AR1,P#0.0];            // do Byte-Offset 0
     L    B#16#2;                    // Identifikátor datového typu BYTE
     T    LB [AR1,P#1.0];            // do Byte-Offset 1
     L    #I_DB_Length;              // délka DB v bytech
     T    LW [AR1,P#2.0];            // do Byte-Offset 2
     L    #DB_NUM;                   // číslo DB
     T    LW [AR1,P#4.0];            // do Byte-Offset 4
     L    P#DBX 0.0;                 // ukazatel na DBX0.0
     T    LD [AR1,P#6.0];            // do Byte-Offset 6
     L    #INI;                      // inicializační byte
     T    #I_INI;                    // uložit do lokální proměnné

     CALL "FILL" (
          BVAL               := #I_INI,// only with temp. Var.
          RET_VAL            := #I_RET_VAL,
          BLK                := #I_ANY);
     SET  ;
     =    #RET_VAL;
     BE   ;

END_FUNCTION
```

# Řešení cvičení 7.6: Zápis hlášení do Diagnostického Bufferu (SFC 52)

```
FUNCTION FC 76 : VOID
TITLE =
//Exercise 7.6: Writing a Message in the Diagnostics Buffer (SFC 52)
VERSION : 0.0
VAR_TEMP
  I_RET_VAL : INT ;
  info1 : WORD ;
  info2 : DWORD ;
END_VAR

BEGIN
NETWORK
TITLE =


    L    W#16#8;
    T    #info1;
    L    W#16#1;
    T    #info2;

    CALL "WR_USMSG" (
        SEND            := TRUE,
        EVENTN          := W#16#9B0A,
        INFO1           := #info1,
        INFO2           := #info2,
        RET_VAL         := #I_RET_VAL);

END_FUNCTION
```

# Řešení cvičení 8.1: Zpracování chyb FC43 (část 1.)

```
FUNCTION FC 81 : INT
TITLE =Exercise 8.2: Calculation of sum, mean value with error handling
// Solution for S7-400
VERSION : 0.0

VAR_INPUT
  Measured_values : ANY ;
END_VAR
VAR_OUTPUT
  Sum : REAL ;
  Mean_value : REAL ;
END_VAR
VAR_TEMP
  Loop_counter : WORD ;
  DB_No : WORD ;
  Sum_1 : REAL ;
  sfc_ret_val : INT ;
  sfc_prgflt : DWORD ;
  sfc_accflt : DWORD ;
END_VAR
BEGIN
NETWORK
TITLE =

     L    P##Measured_values;       // nahraj ukazatel "ANY"
     LAR1  ;                        // do AR1
     L    B [AR1,P#1.0];            // načti identifikátor datového typu
     L    8;                         // načti identifikátor REAL (16#08)
     ==I   ;
     L    -1;                       // Identifikátor datového typu není REAL
     JCN  ERRO;                     // není-li to REAL skoč

// Maskování synchronní chyby: volané DB neexistuje,
// chybné číslo globálního DB
// chybné číslo instančního DB,
// chyba oblasti při čtení,
// chybná délka čtené oblasti
     CALL "MSK_FLT" (
        PRGFLT_SET_MASK  := DW#16#40C0014,
        ACCFLT_SET_MASK  := DW#16#0,
        RET_VAL          := #sfc_ret_val,
        PRGFLT_MASKED    := #sfc_prgflt,
        ACCFLT_MASKED    := #sfc_accflt);


     L    W [AR1,P#2.0];            // nahraj  počet prvků
     T    #Loop_counter;            // inicializuj čitač smyčky
     L    W [AR1,P#4.0];            // nahraj číslo DB number nebo 0
     T    #DB_No;                   // je-li: DB_No=0 potom : OPN DB[DB_No]=NOP
     OPN   DB [#DB_No];             // Run-time error!!, pokud DB neexistuje
     L    D [AR1,P#6.0];            // ukazatel aktuální proměnné
     LAR1  ;                        // chyba!! je-li identifikátor oblasti "DI"
     L    0.000000e+000;            // 0 --> Accu1 (Sum =0.0)
     L    #Loop_counter;            // čitač smyčky --> ACCU1; Sum=0 --> ACCU2

// (další strana ..)
```

# Řešení cvičení 8.1: Zpracování chyb FC43 (část .2)

```
BEGN: TAK  ;                    // Loopc --> ACCU2, Sum --> ACCU1
      ENT  ;                    // Loopc --> ACCU3, Sum --> ACCU2
      L    D [AR1,P#0.0];       // prvek oblasti v ACCU1
      +R   ;                    // Sum --> ACCU1, Loopc --> ACCU2
      +AR1 P#4.0;               // zvedni AR1 o 4 byty
      TAK  ;                    // Loopc. --> ACCU1, Sum --> ACCU2
      LOOP BEGN;                // sniž čitač smyčky a pokračuj v ní
      TAK  ;                    // Sum --> ACCU1
      T    #Sum_1;              // Sum --> #Sum
// vyhodnocení chyby
      CALL "READ_ERR" (
           PRGFLT_QUERY         := DW#16#40C0014,
           ACCFLT_QUERY         := DW#16#0,
           RET_VAL              := #sfc_ret_val,
           PRGFLT_CLR           := #sfc_prgflt,
           ACCFLT_CLR           := #sfc_accflt);
      L    #sfc_prgflt;         // Check for faulty DB
      L    DW#16#40C0000;
      AD   ;
      L    -2;                  // chybový kód pro neexistující DB
      JZ   ERRO;                // je-li chyba skoč
      L    #sfc_prgflt;         // kontrola oblasti a délky oblasti
      L    DW#16#14;
      AD   ;
      L    -4;                  // chybný identifikátor oblasti nebo délka oblasti
      JZ   ERRO;                // skoč, je-li chyba
// není-li chyba "normálně" pokračuj
      L    #Sum_1;
      L    #Loop_counter;       // Sum --> ACCU2, Number --> ACCU1
      DTR  ;                    // DINT (32-Bit) --> REAL
      /R   ;                    // průměrná hodnota v ACCU1
      T    #Mean_value;         // průměrná hodnota --> #Mean_value
      SET  ;                    // BR <-- 1
      SAVE ;
      L    0 ;                  // hlášení : vše O.K.
      T    RET_VAL;
      JU   DMSK;                // skok na odmaskování synchronních chyb
ERRO: CLR  ;                    // Instrukce v případě chyby, RLO=0
      SAVE ;                    // BR =0
      T    #RET_VAL;            // zapiš chybový kód do RET_VAL
      L    L#-1;                // zapiš neplatné Real číslo
      T    #Sum;
      T    #Mean_value;
DMSK: NOP  0;                   // odmaskuj synchronní chyby
      CALL "DMSK_FLT" (
           PRGFLT_RESET_MASK    := DW#16#40C0014,
           ACCFLT_RESET_MASK    := DW#16#0,
           RET_VAL              := #sfc_ret_val,
           PRGFLT_MASKED        := #sfc_prgflt,
           ACCFLT_MASKED        := #sfc_accflt);
      BEU  ;

END_FUNCTION
```

# Řešení cvičení 9.2: Počítání dokončených dílů (část 1., FB1)

```
FUNCTION_BLOCK "Station"
TITLE =
VERSION : 0.1


VAR_INPUT
  Initial : BOOL ;
  Proxy_Switch : BOOL ;
  Acknowledge : BOOL ;
  Clock_Bit : BOOL ;
END_VAR
VAR_OUTPUT
  LED : BOOL ;
  Transp_req : BOOL ;
END_VAR
VAR_IN_OUT
  conv_busy : BOOL ;
END_VAR
VAR
  State : STRUCT
   Busy : BOOL ;
   Completed : BOOL ;
   Waiting : BOOL ;
  END_STRUCT ;
  Aux_1 : BOOL ;
  Aux_2 : BOOL ;
END_VAR
BEGIN
NETWORK
TITLE =Initializing
//With initializing, the basic state, i.e. Station is set in state
//"Busy"

     A    #Initial;
     R    #State.Waiting;
     S    #State.Busy;
     R    #State.Completed;
     R    #conv_busy;



NETWORK
TITLE =State: Busy
//Processing runs in this state. Processing ends when the operator
//acknowledges, with the workplace acknowledgement button, the
//finishing of the piece.

     AN   #State.Busy;
     JC   REDY;
     S    #LED;
     R    #Transp_req;
     A    #Acknowledge;
     R    #State.Busy;
     R    #LED;
     S    #State.Completed;

// (pokračování...)
```

# Řešení cvičení 9.2: Počítání dokončených dílů (část 2., FB1 pokračování)

```
NETWORK
TITLE =State: Completed
//In the finished state, waiting takes place until the finished part can be laid on the
// transport belt. The input signal #Band_frei signals if the transport belt is free
REDY: AN    #State.Completed;
      JC    WAIT;
      A     #Clock_Bit;
      =     #LED;
      AN    #Conv_busy;
      A     #Proxy_Switch;
      S     #Conv_busy;
      S     #Transp_req;
      A     #Transp_req;
      A     #Proxy_Switch;
      FN    #Aux_1;
      R     #State.Completed;
      S     #State.Waiting;

NETWORK
TITLE =State: Waiting
//Waits for a new raw part. Arrival of a new part is signaled by
//the transport belt's proximity switch
WAIT: AN    #State.Waiting;
      JC    ENDE;
      R     #LED;
      A     #Proxy_Switch; // new raw material has arrived
      R     #Transp_req; // stop belt
      A     #Proxy_Switch;
      FN    #Aux_1; // after removal, processing commences
      R     #Conv_busy; // enable conveyor once again
      R     #State.Waiting;
      S     #State.Busy;
ENDE: BEU   ;
```

# Řešení cvičení 9.2: Počítání dokončených dílů(část 3., FB2)

```
FUNCTION_BLOCK "Transport"
TITLE =Controlling the conveyor
VERSION : 0.1

VAR_INPUT
  Initial : BOOL ;
  L_Barrier : BOOL ;
  Acknowledge : BOOL ;
  Transp_req : BOOL ;
  Clock_Bit : BOOL ;
END_VAR
VAR_OUTPUT
  LED : BOOL ;
  Conv_right : BOOL ;
  Conv_left : BOOL ;
  Count_Value :INT ;
END_VAR
VAR
  State : STRUCT
   Waiting : BOOL ;
   Transport_right : BOOL ;
   Assembly : BOOL ;
   Transport_left : BOOL ;
  END_STRUCT ;
  Counter: SFB0;
END_VAR
BEGIN
NETWORK
TITLE =Initialization

     A    #Initial;
     S    #State.Waiting;
     R    #State.Transport_right;
     R    #State.Assembly;
     R    #State.Transport_left;
                call  #Counter (R:= #Initial);    // Reset only if necessary

NETWORK
TITLE =State: "Waiting"
//The conveyor waits for a completed part in this state.
     AN   #State.Waiting;
     JC   RECH;
     R    #Conv_right;
     R    #Conv_left;
     R    #LED;
     A    #Transp_req;
     R    #State.Waiting;
     S    #State.Transport_right;

// (Continued on the next page)
```

# Řešení cvičení 9.2: Počítání dokončených dílů (část 4., FB2 pokračování)

```
NETWORK
TITLE =State: Transport_right
//This state describes the finished part's transport in the direct
//final assembly
RECH: AN    #State.Conv_right;
      JC    ENDM;
      S     #Conv_right;
      A     #Clock_Bit;
      =     #LED;
      AN    #L_Barrier;
      R     #Conv_right;
      R     #State.Conv_right;
      S     #State.Assembly;
                  AN  #L_Barrier;
                  =   #L_Barrier;
CALL #Counter (   CU        := #L_Barrier,
                  CV        := #Current_Value);

NETWORK
TITLE =State: Assembly
//In this state, the finished part is removed and a new raw piece is laid
//on the belt. After that, the raw piece's transport in the direction of the empty
//processing station is started with S4.
//
ENDM: AN    #State.Assembly;
      JC    LINK;
      S     #LED;
      A     #Acknowledge;
      R     #LED;
      R     #State.Assembly;
      S     #State.Conv_left;

NETWORK
TITLE =State: Conv_left
//In the state, the raw piece's transport to the station takes place, that delivered
//the finished piece.
LINK: AN    #State.Conv_left;
      JC    ENDE;
      S     #Conv_left;
      A     #Clock_Bit;
      =     #LED;
      AN    #Transp_req;
      R     #Conv_left;
      R     #State.Conv_left;
      S     #State.Waiting;
ENDE: BEU   ;

END_FUNCTION_BLOCK
```

# Řešení cvičení 9.2: Počítání dokončených dílů (část 5., FB10)

```
FUNCTION_BLOCK FB 10
TITLE =
VERSION : 0.1


VAR
  Station_1 : "Station";
  Station_2 : "Station";
  Station_3 : "Station";
  Transport : "Transport";
  Conv_busy : BOOL ;
END_VAR
VAR_TEMP
  Trans_1 : BOOL ;
  Trans_2 : BOOL ;
  Trans_3 : BOOL ;
  Trans : BOOL ;
END_VAR
BEGIN
NETWORK
TITLE =Station_1

    CALL #Station_1 (
        Initial             := I     0.0,
        Proxy_Switch        := "INI1",
        Acknowledge         := "S1",
        Clock_Bit           := "CLOCK_BIT",
        LED                 := "H1",
        Transp_req          := #Trans_1,
        Conv_busy           := #Conv_busy);

NETWORK
TITLE =Station_2


    CALL #Station_2 (
        Initial             := I     0.0,
        Proxy_Switch        := "INI2",
        Acknowledge         := "S2",
        Clock_Bit           := "CLOCK_BIT",
        LED                 := "H2",
        Transp_req          := #Trans_2,
        Conv_busy           := #Conv_busy);

NETWORK
TITLE =Station_3

    CALL #Station_3 (
        Initial             := I     0.0,
        Proxy_Switch        := "INI3",
        Acknowledge         := "S3",
        Clock_Bit           := "CLOCK_BIT",
        LED                 := "H3",
        Transp_req          := #Trans_3,
        Conv_busy           := #Conv_busy);


// (Continued on next page)
```

# Řešení cvičení 9.2: Počítání dokončených dílů (část 6., FB10 pokračování)

```
NETWORK
TITLE =Logic: Transp_req
//Forming the logic for #Transp_req
    O    #Trans_1;
    O    #Trans_2;
    O    #Trans_3;
    =    #Trans;


NETWORK
TITLE =Transport

    CALL #Transport (
        Initial                 := I     0.0,
        L_Barrier               := "LS1",
        Acknowledge             := "S4",
        Transp_req              := #Trans,
        Clock_Bit               := "CLOCK_BIT",
        LED                     :=  "H4",
        Conv_right              := "K1_CONV",
        Conv_left               := "K2_CONV");

                L    #Transport.Current_value ;
                ITD ;                  // Expand to DINT
                DTB ;                     // Convert to BCD
                T    QW2 ;

END_FUNCTION_BLOCK
```

# Řešení cvičení 9.2: Počítání dokončených dílů(část 7., DB10)

```
DATA_BLOCK DB 10
TITLE =
VERSION : 0.0

 FB 10
BEGIN
  Station_1.Initial := FALSE;
  Station_1.Proxy_Switch := FALSE;
  Station_1.Acknowledge := FALSE;
  Station_1.Clock_Bit := FALSE;
  Station_1.LED := FALSE;
  Station_1.Transp_req := FALSE;
  Station_1.Conv_busy:= FALSE;
  Station_1.State.Busy := FALSE;
  Station_1.State.Completed := FALSE;
  Station_1.State.Waiting := FALSE;
  Station_1.Aux_1 := FALSE;
  Station_1.Aux_2 := FALSE;
  Station_2.Initial := FALSE;
  Station_2.Proxy_Switch := FALSE;
  Station_2.Acknowledge := FALSE;
  Station_2.Clock_Bit := FALSE;
  Station_2.LED := FALSE;
  Station_2.Transp_req := FALSE;
  Station_2.Conv_busy := FALSE;
  Station_2.State.Busy := FALSE;
  Station_2.State.Completed := FALSE;
  Station_2.State.Waiting := FALSE;
  Station_2.Aux_1 := FALSE;
  Station_2.Aux_2 := FALSE;
  Station_3.Initial := FALSE;
  Station_3.Proxy_Switch := FALSE;
  Station_3.Acknowledge := FALSE;
  Station_3.Clock_Bit := FALSE;
  Station_3.LED := FALSE;
  Station_3.Transp_req := FALSE;
  Station_3.Conv_busy := FALSE;
  Station_3.State.Busy := FALSE;
  Station_3.State.Completed := FALSE;
  Station_3.State.Waiting := FALSE;
  Station_3.Aux_1 := FALSE;
  Station_3.Aux_2 := FALSE;
  Transport.Initial := FALSE;
  Transport.L_Barrier := FALSE;
  Transport.Acknowledge := FALSE;
  Transport.Transp_req := FALSE;
  Transport.Clock_Bit := FALSE;
  Transport.LED := FALSE;
  Transport.Conv_right := FALSE;
  Transport.Conv_left := FALSE;
  Transport.State.Waiting := FALSE;
  Transport.State.Transport_right := FALSE;
  Transport.State.Assembly := FALSE;
  Transport.State.Transport_left := FALSE;
  Conv_busy := FALSE;
END_DATA_BLOCK
```

# Řešení cvičení 9.2: Počítání dokončených dílů(část 8., OB1)

```
ORGANIZATION_BLOCK OB 1
TITLE =
VERSION : 0.1


VAR_TEMP
  OB1_EV_CLASS : BYTE ;    //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
  OB1_SCAN_1 : BYTE ; //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
  OB1_PRIORITY : BYTE ;    //1 (Priority of 1 is lowest)
  OB1_OB_NUMBR : BYTE ;  //1 (Organization block 1, OB1)
  OB1_RESERVED_1 : BYTE ;  //Reserved for system
  OB1_RESERVED_2 : BYTE ;  //Reserved for system
  OB1_PREV_CYCLE : INT ;  //Cycle time of previous OB1 scan (milliseconds)
  OB1_MIN_CYCLE : INT ;    //Minimum cycle time of OB1 (milliseconds)
  OB1_MAX_CYCLE : INT ;    //Maximum cycle time of OB1 (milliseconds)
  OB1_DATE_TIME : DATE_AND_TIME ;  //Date and time OB1 started
END_VAR
BEGIN
NETWORK
TITLE =

    CALL FB    10 , DB    10 ;

END_ORGANIZATION_BLOCK
```

# Řešení cvičení 10.1: Komunikace s SFB bloky START/STOP OB100, OB1 (část 1.)

```
ORGANIZATION_BLOCK OB 100
TITLE =Initializing the SFBs "START" and "STOP"
VERSION : 0.1


VAR_TEMP
  OB100_EV_CLASS : BYTE ; //16#13, Event class 1, Entering event state, Event logged in
                diagnostic buffer
  OB100_STRTUP : BYTE ;    //16#81/82/83/84 Method of startup
  OB100_PRIORITY : BYTE ;  //27 (Priority of 1 is lowest)
  OB100_OB_NUMBR : BYTE ;  //100 (Organization block 100, OB100)
  OB100_RESERVED_1 : BYTE ;   //Reserved for system
  OB100_RESERVED_2 : BYTE ;   //Reserved for system
  OB100_STOP : WORD ;      //Event that caused CPU to stop (16#4xxx)
  OB100_STRT_INFO : DWORD ;  //Information on how system started
  OB100_DATE_TIME : DATE_AND_TIME ;   //Date and time OB100 started
  P_NAME : ARRAY  [1 .. 9 ] OF CHAR ;
END_VAR
BEGIN
NETWORK
TITLE =
//Enter STRING "P_PROGRAM" in PI_NAME
// ***************************************************************************
// ***          DEFINE THE PARAMETER "PI_NAME" OF THE CFBs          ****
                *
//***************************************************************************
//

    L    'P_PR';
    T    MD  100;
    L    'OGRA';
    T    MD  104;
    L    'M';
    T    MB  108;
NETWORK
TITLE =

    CALL SFB  20 , DB   20 (// Initialization of SFB "STOP"
       REQ              := FALSE,
       ID               := W#16#3,
       PI_NAME              := P#M 100.0 BYTE 9);

NETWORK
TITLE =

    CALL SFB  19 , DB   19 (// Initialization of SFB "START"
       REQ              := FALSE,
       ID               := W#16#3,
       PI_NAME              := P#M 100.0 BYTE 9);

END_ORGANIZATION_BLOCK
```

# Řešení cvičení 10.1: Komunikace s SFB bloky START/STOP: OB100, OB1 (část 2.)

```
ORGANIZATION_BLOCK OB 1
TITLE =
VERSION : 0.1


VAR_TEMP
 OB1_EV_CLASS : BYTE ;    //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
 OB1_SCAN_1 : BYTE ; //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
 OB1_PRIORITY : BYTE ;     //1 (Priority of 1 is lowest)
 OB1_OB_NUMBR : BYTE ;  //1 (Organization block 1, OB1)
 OB1_RESERVED_1 : BYTE ;  //Reserved for system
 OB1_RESERVED_2 : BYTE ;  //Reserved for system
 OB1_PREV_CYCLE : INT ;  //Cycle time of previous OB1 scan (milliseconds)
 OB1_MIN_CYCLE : INT ;     //Minimum cycle time of OB1 (milliseconds)
 OB1_MAX_CYCLE : INT ;    //Maximum cycle time of OB1 (milliseconds)
 OB1_DATE_TIME : DATE_AND_TIME ;  //Date and time OB1 started
 DONE_FLAG_20 : BOOL ;
 ERROR_FLAG_20 : BOOL ;
 DONE_FLAG_19 : BOOL ;
 ERROR_FLAG_19 : BOOL ;
 STATUS_WORD_20 : WORD ;
 STATUS_WORD_19 : WORD ;
END_VAR
BEGIN
NETWORK
TITLE ="SFB_STOP"

    CALL SFB   20 , DB    20 (
       REQ               := I     0.0,
       DONE               := #DONE_FLAG_20,
       ERROR              := #ERROR_FLAG_20,
       STATUS             := #STATUS_WORD_20);

NETWORK
TITLE ="SFB_START"

    CALL SFB   19 , DB    19 (
       REQ               := I     0.1,
       DONE               := #DONE_FLAG_19,
       ERROR              := #ERROR_FLAG_19,
       STATUS             := #STATUS_WORD_19);
```

// (Continued on the next page)

# Řešení cvičení 10.1: Komunikace s SFB bloky START/STOP, OB100, OB1 (část 3)

```
NETWORK
TITLE =

    A(    ; // Returned STATUS word from SFB "START" in QW2
    O     #DONE_FLAG_19;
    O     #ERROR_FLAG_19;
    )     ;
    JNB _001;
    L     #STATUS_WORD_19;
    T     QW    2;
_001: NOP   0;
NETWORK
TITLE =

    A(    ; // Returned STATUS word from SFB "STOP" in QW2
    O     #DONE_FLAG_20;
    O     #ERROR_FLAG_20;
    )     ;
    JNB _002;
    L     #STATUS_WORD_20;
    T     QW    2;
_002: NOP   0;
NETWORK
TITLE =

    A     I     0.0; // Otherwise write FFFF in QW2
    BEC   ;
    A     I     0.1;
    BEC   ;
    L     W#16#FFFF;
    T     QW    2;
END_ORGANIZATION_BLOCK
```

# Řešení cvičení 10.2: Komunikace s SFB bloky GET/PUT, OB100, OB1 (část 1.)

```
ORGANIZATION_BLOCK OB 100
TITLE =Complete Restart
//Exercise : S7-400 reads out of S7-300 and writes in S7-300
AUTHOR : AUT95
FAMILY : A2_0
NAME : ST7PROG3
VERSION : 0.0


VAR_TEMP
 OB100_EV_CLASS : BYTE ; //16#13, Event class 1, Entering event state, Event logged in
              diagnostic buffer
 OB100_STRTUP : BYTE ;    //16#81/82/83/84 Method of startup
 OB100_PRIORITY : BYTE ;  //27 (Priority of 1 is lowest)
 OB100_OB_NUMBR : BYTE ;  //100 (Organization block 100, OB100)
 OB100_RESERVED_1 : BYTE ;   //Reserved for system
 OB100_RESERVED_2 : BYTE ;   //Reserved for system
 OB100_STOP : WORD ;      //Event that caused CPU to stop (16#4xxx)
 OB100_STRT_INFO : DWORD ;  //Information on how system started
 OB100_DATE_TIME : DATE_AND_TIME ;   //Date and time OB100 started
 PI_NAME : ARRAY  [1 .. 9 ] OF CHAR ;
END_VAR
BEGIN
NETWORK
TITLE =Initialize CFB "GET"

   CALL SFB   14 , DB    14 (
      REQ                 := FALSE,
      ID                  := W#16#3);

   NOP   0;
NETWORK
TITLE =Initialize CFB "PUT"

   CALL SFB   15 , DB    15 (
      REQ                 := FALSE,
      ID                  := W#16#3);

END_ORGANIZATION_BLOCK
```

# Řešení cvičení 10.2: Komunikace s SFB bloky GET/PUT, OB100, OB1 (část 2.)

```
ORGANIZATION_BLOCK OB 1
TITLE =Cycle
//Exercise: S7400 writes in S7-300 and reads out of S7-300
AUTHOR : AUt95
FAMILY : A2_0
NAME : ST7PROG3
VERSION : 0.0


VAR_TEMP
  OB1_EV_CLASS : BYTE ;    //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
  OB1_SCAN_1 : BYTE ; //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
  OB1_PRIORITY : BYTE ;    //1 (Priority of 1 is lowest)
  OB1_OB_NUMBR : BYTE ;  //1 (Organization block 1, OB1)
  OB1_RESERVED_1 : BYTE ;  //Reserved for system
  OB1_RESERVED_2 : BYTE ;  //Reserved for system
  OB1_PREV_CYCLE : INT ;  //Cycle time of previous OB1 scan (milliseconds)
  OB1_MIN_CYCLE : INT ;    //Minimum cycle time of OB1 (milliseconds)
  OB1_MAX_CYCLE : INT ;    //Maximum cycle time of OB1 (milliseconds)
  OB1_DATE_TIME : DATE_AND_TIME ;  //Date and time OB1 started
  NDR_FLAG_14 : BOOL ;
  ERROR_FLAG_14 : BOOL ;
  DONE_FLAG_15 : BOOL ;
  ERROR_FLAG_15 : BOOL ;
  STATUS_WORD_14 : WORD ;
  STATUS_WORD_15 : WORD ;
END_VAR
BEGIN
NETWORK
TITLE ="SFB_GET"

    CALL SFB   14 , DB    14 (// Call of "GET"
       REQ                 := I     0.2,
       ID                  := W#16#3,
       NDR                 := #NDR_FLAG_14,
       ERROR               := #ERROR_FLAG_14,
       STATUS              := #STATUS_WORD_14,
       ADDR_1              := P#I 0.0 BYTE 1,
       ADDR_2              := P#I 4.0 WORD 1,
       RD_1                := P#Q 0.0 BYTE 1,
       RD_2                := P#Q 4.0 WORD 1);
```

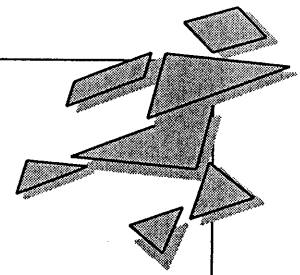
// (Continued on the next page)

# Řešení cvičení 10.2: Komunikace s SFB bloky GET/PUT, OB100, OB1 (část 3.)

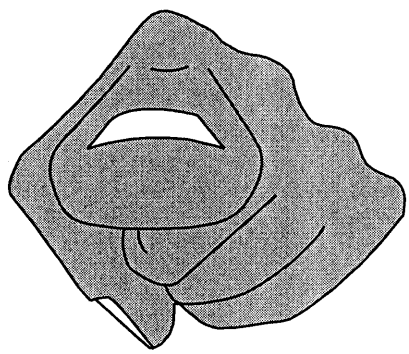
```
NETWORK
TITLE ="SFB_PUT"

    CALL SFB   15 , DB    15 (//Call of "PUT"
       REQ                 := I     0.3,
       ID                  := W#16#3,
       DONE                := #DONE_FLAG_15,
       ERROR               := #ERROR_FLAG_15,
       STATUS              := #STATUS_WORD_15,
       ADDR_1              := P#Q 12.0 WORD 1,
       SD_1                := P#I 2.0 WORD 1);

NETWORK
TITLE =

    A(    ; // Returned STATUS word from SFB "GET" in QW2
    O     #NDR_FLAG_14;
    O     #ERROR_FLAG_14;
    )     ;
    JNB _002;
    L     #STATUS_WORD_14;
    T     QW    2;
_002: NOP   0;
NETWORK
TITLE =

    A(    ; // Returned STATUS word from SFB "PUT" in QW2
    O     #DONE_FLAG_15;
    O     #ERROR_FLAG_15;
    )     ;
    JNB _001;
    L     #STATUS_WORD_15;
    T     QW    2;
_001: NOP   0;
NETWORK
TITLE =

    A     I     0.2; // Otherwise write FFFF in QW2
    BEC   ;
    A     I     0.3;
    BEC   ;
    L     W#16#FFFF;
    T     QW    2;
END_ORGANIZATION_BLOCK
```

# Řešení cvičení 10.2: Komunikace s SFB bloky GET/PUT, OB100, OB1 (část 3)

```
NETWORK
TITLE ="SFB_PUT"

    CALL SFB   15 , DB    15 (//Call of "PUT"
       REQ                 := I     0.3,
       ID                  := W#16#3,
       DONE                  := #DONE_FLAG_15,
       ERROR                 := #ERROR_FLAG_15,
       STATUS                := #STATUS_WORD_15,
       ADDR_1                := P#Q 12.0 WORD 1,
       SD_1                := P#I 2.0 WORD 1);

NETWORK
TITLE =

    A(    ; // Returned STATUS word from SFB "GET" in QW2
    O     #NDR_FLAG_14;
    O     #ERROR_FLAG_14;
    )     ;
    JNB _002;
    L     #STATUS_WORD_14;
    T     QW    2;
_002: NOP   0;
NETWORK
TITLE =

    A(    ; // Returned STATUS word from SFB "PUT" in QW2
    O     #DONE_FLAG_15;
    O     #ERROR_FLAG_15;
    )     ;
    JNB _001;
    L     #STATUS_WORD_15;
    T     QW    2;
_001: NOP   0;
NETWORK
TITLE =

    A     I     0.2; // Otherwise write FFFF in QW2
    BEC   ;
    A     I     0.3;
    BEC   ;
    L     W#16#FFFF;
    T     QW    2;
END_ORGANIZATION_BLOCK
```



# A co dál?






Datum : 24.11.2002
Soubor: 7Prog2_Fcz

Školící středisko
firmy E&A spol. s r.o., MB

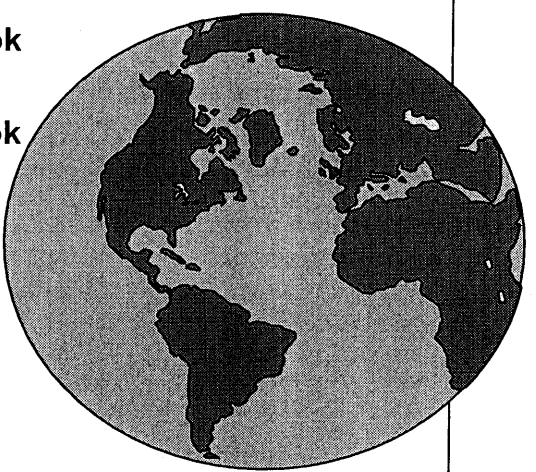
## Rádi bychom Vám řekli několik slov ....

Obsah:

- A co dál ?
- Znalosti pro automatizaci
- SIMATIC školení
- Systematické školení na SIMATIC S5
- přechod z SIMATIC S5 na SIMATIC S7
- systém školení na SIMATIC S7
- školení na SIMATIC S7-200
- doplňkové programy pro SIMATIC S7/M7
- SIMATIC NET
- SIMATIC WinCC

## Znalosti pro automatizaci

- **51 školících středisek v Německu a dalších 140 ve více jak 55 zemích**
- **V Německu**
  **24,000 účastníků / 101,000 školících dní za rok**
- **V zbytku světa**
  **29,600 účastníků / 120,000 školících dní za rok**
- **školení je určeno pro každého, pro všechny oblasti automatizačních technologií**



se školením od A&D.....





### Jaké výhody Vám poskytuje náš systém školení SIMATIC?

- rychlé a efektivní získávání vědomostí
- snížení prostojů při řešení poruch ve Vašem závodě
- zajištěná kvalita výuky
- vysoce motivovaný školící personál
- zjednodušení a zkrácení rozhodovacích procesů

**Poznámka**

Následující stránky Vám předkládají informace o našem rozsahu SIMATIC kurzů. Na poslední stránce lze nalézt faxový formulář, pomocí kterého lze u naší firmy některý z nabízených kurzů objednat.

Můžete nás také vyhledat na Internetu:

http:**www.eaamlbol.cz**

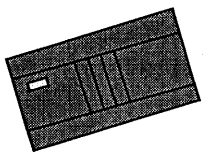
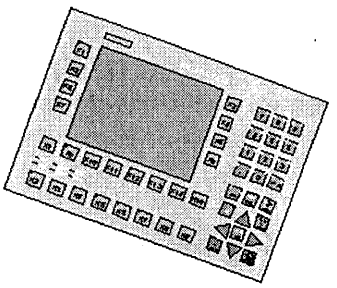
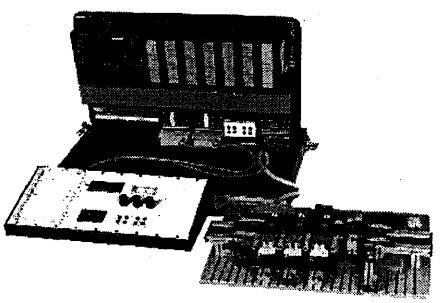
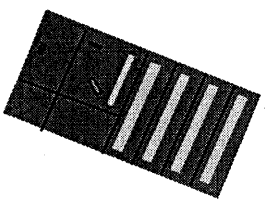
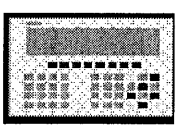
E-mail: skoleni**@eaa.siemens.cz**

nebo zavolat na naši INFO-linku:

Tel: **0326/713 811**
Fax: **0326/713 954,953**

## SIMATIC školení

- SIMATIC S5
- SIMATIC S7
- SIMATIC M7
- SIMATIC C7
- SIMATIC HMI (COROS, ProTool, WinCC)
- SIMATIC NET (PROFIBUS, Ethernet)

SIMATIC S7


Datum : 24.11.2002
Soubor: 7Prog2_Fcz

Školící středisko firmy E&A spol. s r.o., MB

Právě jste ukončili jeden z kurzů uceleného systému školení. Doufáme, že tento kurz splnil Vaše očekávání.

Také doufáme, že informace získané v tomto kurzu Vám pomohou ulehčit a zefektivnit Vaši práci na pracovišti.

Rádi bychom byli i nadále Vašimi partnery při Vašem dalším vzdělávání v oboru řídících systému SIMATIC.

Z tohoto důvodu jsme pro Vás připravili několik následujících stránek, na kterých Vám chceme předložit základní informaci o kurzech pořádaných naší firmou a systému vzdělávání pro průmyslovou automatizaci nabízeným koncernem SIEMENS AG.

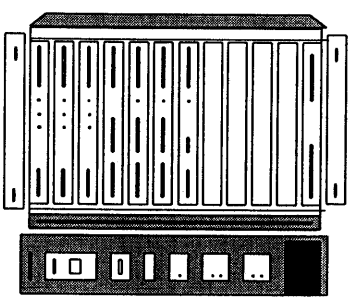
# Systematické vzdělávání pro SIMATIC S5

- PLC systém SIMATIC S5 si již dávno vydobil své jméno na trhu s automatizační technikou a vytvořil standard pro PLC systémy ostatních výrobců.
  V současné době je nabízeno široké spektrum školení pro práci s PLC SIMATIC S5 :

  - **systémové kurzy SIMATIC S5**

  - **doplňující kurzy SIMATIC S5**
    - algoritmizace automatizačních úloh, řízení v otevřené smyčce
    - zpětnovazební regulace, polohování
    - průmyslové sítě, komunikace

  - **Strukturovaná školení splňující požadavky VDMA/ZVEI (Asociace německých výrobců strojů a zařízení a asociace německého elektrotechnického a elektronického průmyslu)**



SIMATIC S7


Datum : 24.11.2002
Soubor: 7Prog2_Fcz

Školící středisko firmy E&A spol. s r.o., MB

**Systémové kurzy**

Systémová školení
Servisní školení
Kurzy programování

Firma E&A spol s r.o. v současnosti organizuje všechny výše uvedené systémové kurzy pro práci na PLC systému SIMATIC S5..

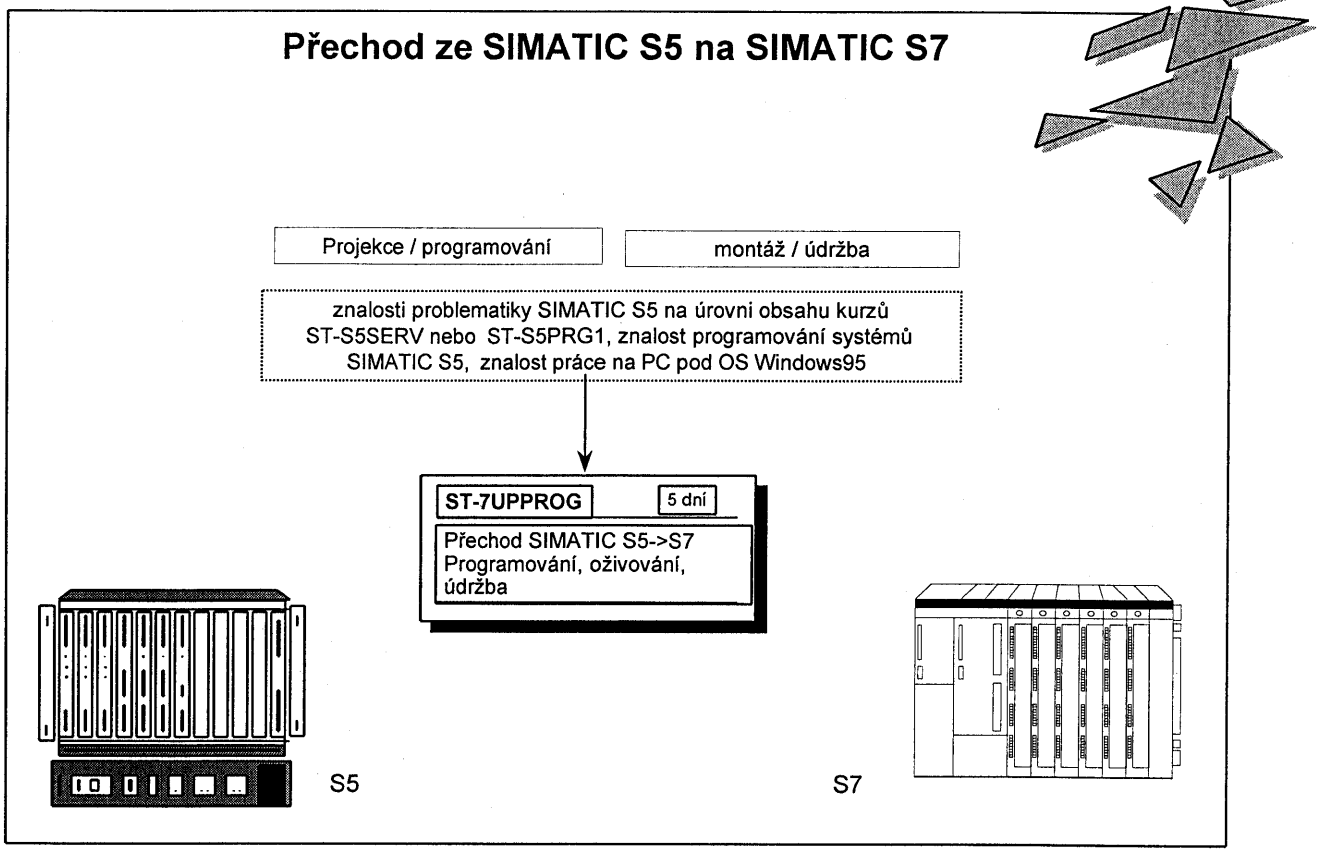
**Doplňující kurzy**

Plánování / návrh projektů
Sekvenční řízení s GRAPHem 5
Systémy se zvýšenou odolností proti výskytu poruch (F řada)
Redundantní systémy (H řada)
Číslicové řízení
Programově řešená zpětnovazební regulace
Polohování s IP246 / 266
Přímé propojení (Point-to-point)
Komunikace přes síť SINEC L1
Profibus pro SIMATIC S5
Seminář CP 5431-FMS
Seminář S5-95/PROFIBUS
Industrial Ethernet pro SIMATIC S5

Vybrané doplňující kurzy jsou firmou E&A s.r.o. organizovány příležitostně podle konkrétních přání zákazníků.

**Obsahy kurzů**

Detailní informace o obsahu všech kurzů jsou obsaženy v katalogu ITC, který lze na vyžádání získat ve všech školících střediscích fy SIEMENS.
Potřebné informace lze získat také na Internetovské adrese :
**http://www.ad.siemens.de/training**
Informace o kurzech pořádaných firmou E&A s.r.o. lze získat na adrese :
**http://www.eaamlbol.cz**







Datum : 24.11.2002
Soubor: 7Prog2_Fcz



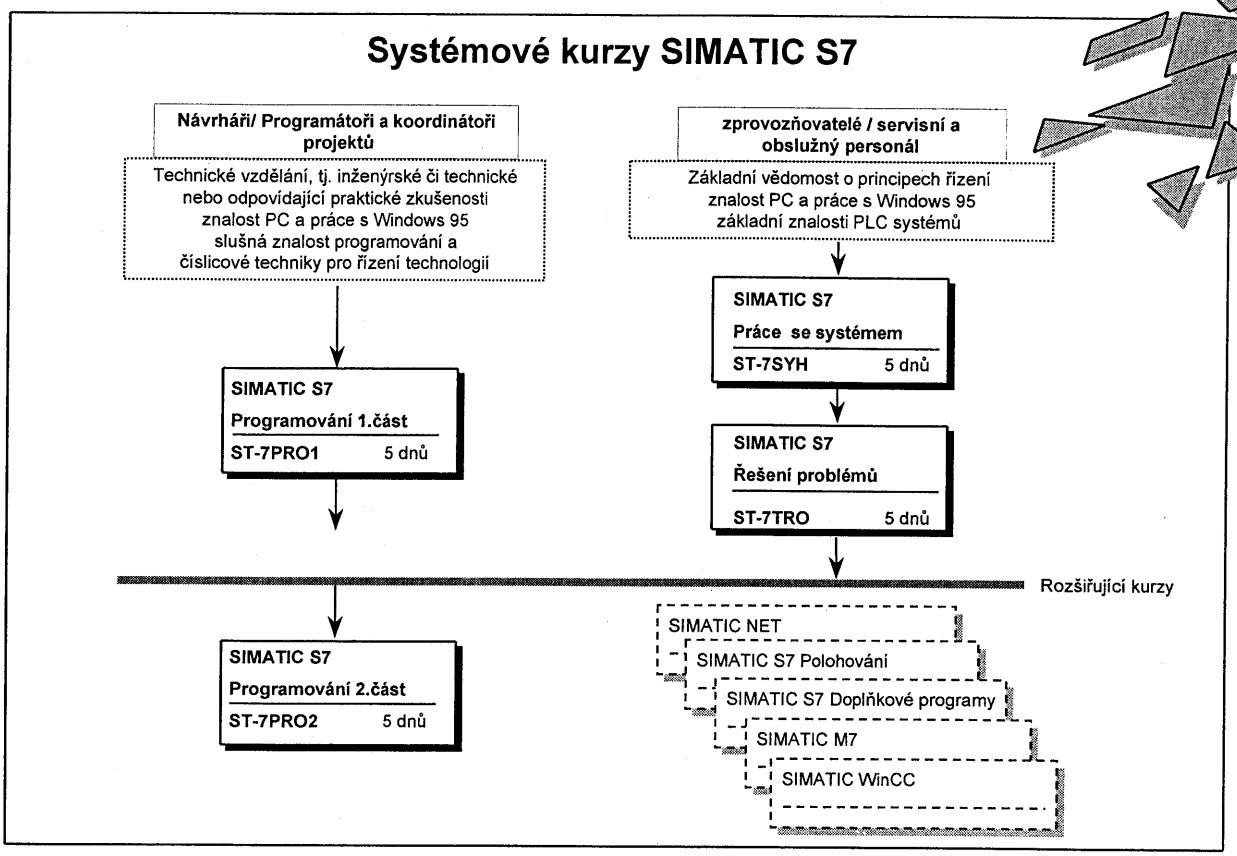
**ST-7UPPROG**

Tento kurz je určen pro získání informací potřebných pro práci se systémem SIMATIC S7. Kurz je koncipován jako přeškolení ze systému S5 na systém S7.

| | | |
|---|---|---|
| cílové skupiny | : | Programátoři, návrháři , projektanti |
| Doba trvání | : | 5 dní |
| Vstupní požadavky | : | práce pod OS Windows 95<br>základní znalost problematiky SIMATIC S5 |

**Předmětem kurzu jsou :**

- Přehled a hlavní výkonové rysy systému
- Konfigurace SIMATIC S7 hardware
- Dokumentační nástroje STEPu 7, ladění uživatelského programu a chybová diagnostika
- konfigurace komunikace (Globální data)
- Do kurzu jsou zahrnuta tato témata patřící do obsahu kurzu ST-7PRO1 :
  - včlenění komponent SIMATICu S5 do systému SIMATIC S7-400
  - Programování - rozdílnosti a nové programové konstrukce
  - Konverze S5 uživatelských programů
  - Nepřímá adresace
  - Speciální rysy systému S7-400
  - Praktická cvičení určená pro ověření teoretických vědomostí získaných v průběhu kurzu




SIMATIC S7



Datum: 24.11.2002
Soubor: 7Prog2_Fcz

Školící středisko firmy E&A spol. s r.o., MB

## Úvod

Systémové kurzy pro PLC SIMATIC S7 tvoří podobně jako kurzy pro SIMATIC S5 ucelený systém vzdělávání organizovaný do tzv. linií kurzů, které jsou navrženy jako optimální doporučné posloupnosti jednotlivých kurzů pro pracovníky různé kvalifikace a pracovního zaměření.

## Výhodná organizace kurzů

- Modulární struktura kurzů
  - přizpůsobení kurzů schopnostem uživatele
  - individuální přístup
  - pružný výběr termínů
  - možnost organizování kurzů u zákazníka

- Cílově orientované zaměření kurzů
  - kurzy zaměřené na obor činnosti
  - kurzy obsahově organizované podle požadavků zákazníka

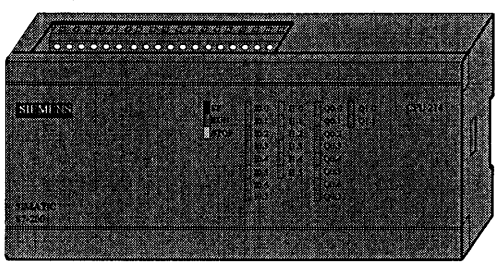
# školení SIMATIC S7-200

**Návrháři, programátoři, oživovači a servisní personáll**

Základní vědomosti o principech řízení
znalost PC techniky a práce pod Windows95

**SIMATIC S7**
**seminář S7-200**

**ST-7MICRO** 2 dny



SIMATIC S7


Datum : 24.11.2002
Soubor: 7Prog2_Fcz

Školící středisko firmy E&A spol. s r.o., MB

**ST-7MICRO** seminář SIMATIC S7-200

Cílová skupina : začátečníci

Doba trvání : 2 dny

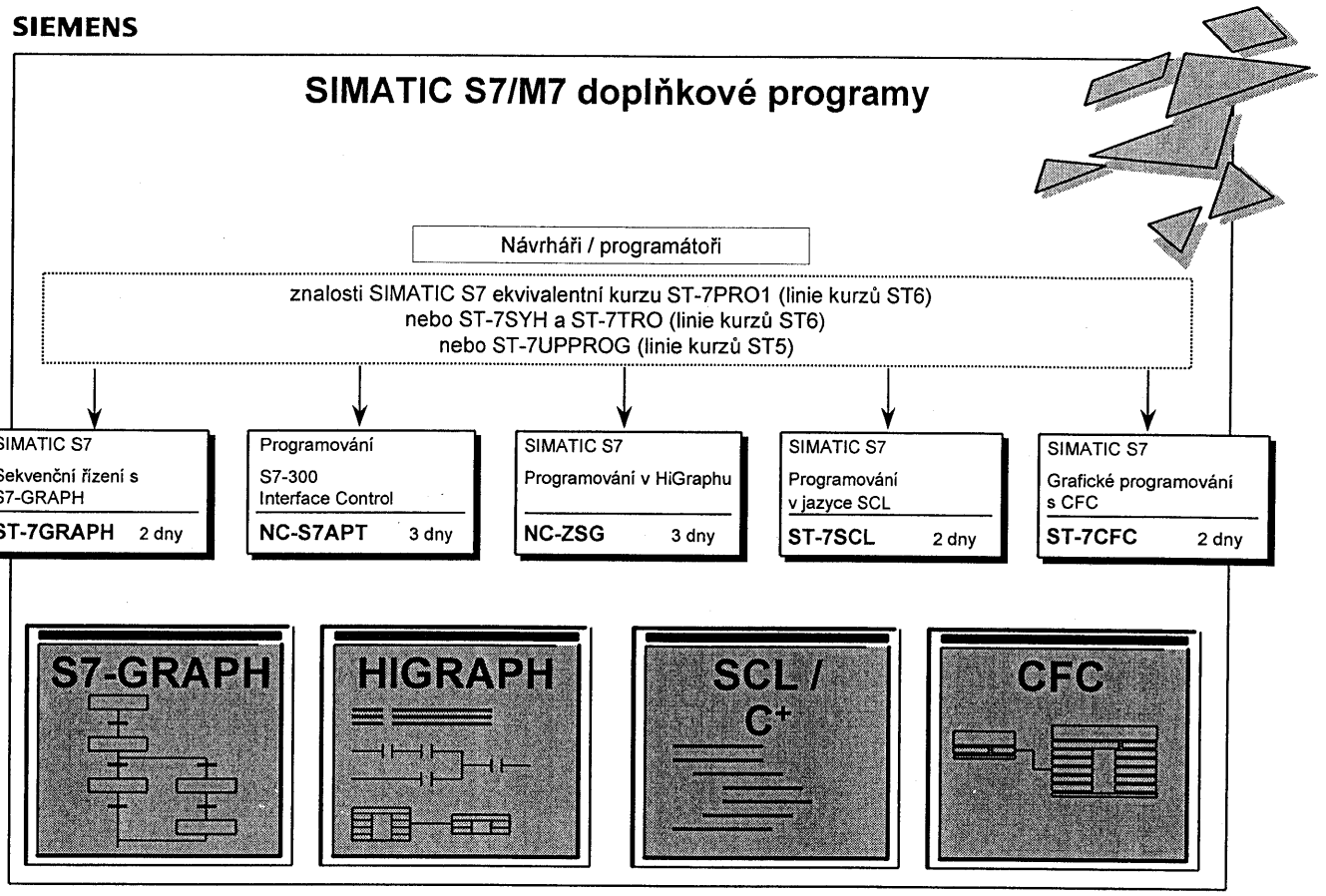
Vstupní požadavky :Žádná či pouze minimální znalost PLC techniky

**Popis kurzu**

Kurz je určen začátečníkům, kteří chtějí rychle získat potřebné vědomosti pro práci se systémem SIMATIC S7-200.

Hlavními probíranými tématy kurzu jsou :

- Výkonové rysy PLC systému SIMATIC S7-200 a programovacích přístrojů
- Možnosti rozšíření a adresace S7-200
- Možnosti sestavení, napsání, zdokumentování a spuštění jednoduchého řídícího programu pro PLC SIMATIC S7-200 řešícího jednoduchou automatizační úlohu pro SIMATIC
- Úvod do struktury a zpracování uživatelského programu pro SIMATIC S7-200
- Práce s programem STEP 7 Micro/WIN
- Možnosti použití základních operací a standardních či speciálních funkcí poskytovaných PLC přístrojem




SIMATIC S7



Datum : 24.11.2002
Soubor: 7Prog2_Fcz

Školící středisko firmy E&A spol. s r.o., MB

**ST-7GRAPH**

- Programování sekvencerů
- Sestavování programů pomocí S7-GRAPHu
- Porovnání GRAPHu 5 a S7-GRAPHu
- Možnosti testování a diagnostiky, dokumentace k programu
- Praktická cvičení

**NC-ZSG**

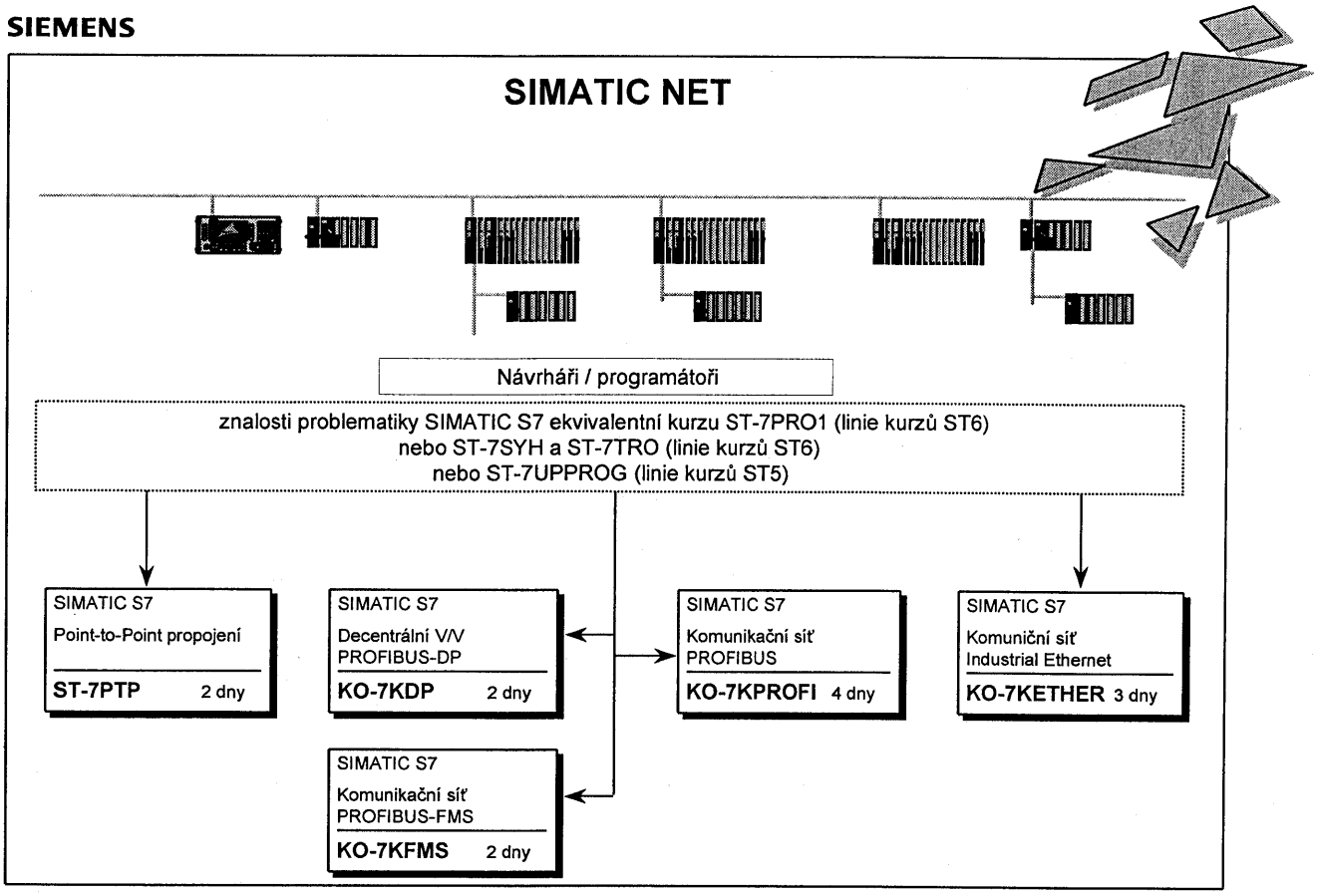
- Programování řídícího systému pomocí HiGRAPHu.
- Programovací nástroje a jejich užití
- Možnosti testování a diagnostiky
- Dokumentace k programu
- Praktická cvičení

**ST-7SCL**

- Vyšší programovací jazyk S7-SCL pro PLC přístroje SIMATIC S7
- Komponenty S7-SCL
- Struktura programu, syntaxe a semantika jazyka
- Možnosti testování a diagnostiky
- Praktická cvičení

**ST-7CFC**

- Programovací nástroje pro grafické programování PLC SIMATIC S7/M7
- Konfigurační programy
- Propojovací bloky
- Programování uživatelských bloků v STEPu 7
- Možnosti testování a diagnostiky
- Praktická cvičení




SIMATIC S7



Datum : 24.11.2002
Soubor: 7Prog2_Fcz

Školící středisko firmy E&A spol. s r.o., MB

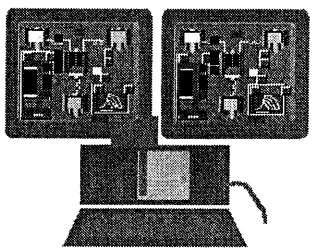
**ST-7PTP** Kurz je určen technikům všech zaměření, kteří chtějí realizovat nejjednodušší komunikační propojení PLC přístrojů S7-300 a S7-400.

**KO-7KDP** Kurz je zaměřen na získání teoretických vědomostí a praktických zkušeností s realizací datové komunikace mezi přístroji pomocí průmyslové sítě PROFIBUS-DP umožňující cyklickou výměnu dat mezi SIMATICem S7 jako řídící jednotkou (master) a pasivními jednotkami (slaves).
Systém decentrálních periferních jednotek ET200 je navržen pro rychlou cyklickou výměnu dat mezi přístroji s možností optimální volby konfigurace do různých průmyslových prostředí.

**KO-7KFMS** Cílem kurzu je seznámit účastníky s principy heterogenní datové komunikace mezi PLC přístroji SIMATIC S7 prostřednictvím průmyslové sítě PROFIBUS protokolem FMS.

**KO-7KPROFI** Tento kurz kombinuje kurzy KO-7KDP a KO-7KFMS. Jeho obsah je identický s obsahy obou kurzů. Jedná se o komplexní kurz pro uživatele datové sítě standardu PROFIBUS.

**KO-7KETHER** Kurz je určen pro všechny pracovníky, kteří potřebují v uživatelském programu pro SIMATIC S7 přijímat resp. vysílat data do průmyslové sítě Industrial Ethernet. Je také vhodný pro tzv.oživovače řídících systémů, kteří potřebují začlenit PLC přístroj SIMATIC S7 do síťových systémů standardu Ethernet.



## SIMATIC WinCC

Návrháři/ programátoři a koordinátoři projektů,
servisní personál, pracovnící odpovědní za koncepční rozvoj

Zkušenost s grafickým uživatel. rozhranním na PC,t.j. Windows 3.11 - 95/NT nebo OS/2

Systémový kurz

**SIMATIC WinCC**
systémové školení Human-Machine Interface
**ST-BWINCCS** 5 dnů

Dobrá znalost „C", základní znalost relačních databází a práce pod Windows 95/NT

Základní znalost „C", základní znalost problematiky relačních databází

seminář

**SIMATIC WinCC**
Otevřený systém E
**ST-BWINCCE** 2 dny

**SIMATIC WinCC**
Otevřený systém N
**ST-BWINCCN** 1 den

SIMATIC S7


Datum : 24.11.2002
Soubor: 7Prog2_Fcz

Školící středisko firmy E&A spol. s r.o., MB

**ST-BWINCCS**

- přehled systému SIMATIC WinCC
- nastavení Windows95, použití standardních rozhranní Windows
- vytvoření projektu, propojení mezi dvěma PLC přístroji, simulace proměnných, grafické objekty
- obrazovka hlášení, archivace hlášení
- okno křivek, archivace snímaných (měřených) hodnot, uživatelské archívy
- systém hlášení, procesy na pozadí
- otevřené uživatelské rozhranní API (použití a struktura)
- praktická cvičení

**ST-BWINCCE**

Tento seminář je určen pro uživatele, kteří chtějí integrovat další aplikační programy do otevřeného systému WinCC:

- Úvod do architektury WinCC (otevřená rozhranní a možnosti integrace dalších aplikací, databáze, kanál DLL, povely „Global Scripts")
- Úvod do jazyka Visual C++, ODK (vývojové prostředí pro API), strukturování WinCC API, použití API funkcí
- Praktická cvičení

**ST-BWINCCN**

Tento seminář je určen pro pracovníky, kteří budou pracovat a aplikacemi integrovanými do otevřeného systému WinCC :

- Stručný úvod do architektury vizualizačního systému WinCC (otevřená rozhranní a možnosti integrace dalších aplikací, databáze, kanál DLL),
- Obecný úvod do povelů „Global Scripts", přístup k databázím WinCC pomocí programu Excel, OLE funkce
- Praktická cvičení

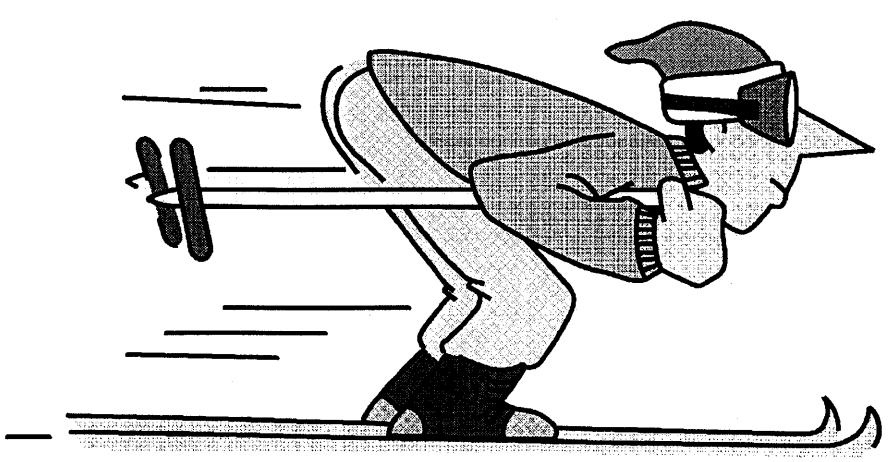
## Ještě stále máte otázky ?

- **Jsme tady pro Vás**

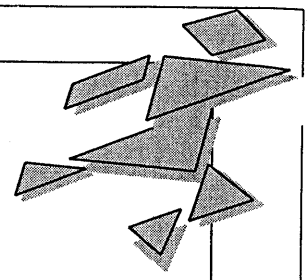




Teď je však řada na Vás.....



# Závazná přihláška
## na kurs řídících systémů SIMATIC



**Název kursu :** **Termín konání :**

**Jméno a příjmení účastníka :**

**Rezervace ubytování :** ☐ **Ano** ☐ **od neděle** ☐ **od pondělí**

☐ **Ne**

---

**Název firmy :**

**Adresa firmy :** **DIČ :**

**Kontaktní osoba :** **IČO :**

**Telefon :**

**Fax :**

**Souhlasíme s objednacími podmínkami školení.**

.....

**Razítko, podpis**

---

zde odstřihnout

Podmínky přihlášky :

Zvolený termín je nutné vždy telefonicky konzultovat s pracovníky školícího střediska..

Odesláním přihlášky se objednatel zavazuje uhradit zálohovou fakturu, která mu bude zaslána firmou E&A spol. s r.o. po obdržení objednávky jako její potvrzení. Její úhrada musí být provedena před zahájením daného kursu.

Zákazník má možnost zrušit potvrzenou objednávku nejpozději 10 kalendářních dnů před zahájením kursu.

Vyplněnou objednávku je nutné zaslat na adresu

E&A spol. s r.o.
P.O.BOX 27
29301 Mladá Boleslav

nebo faxem na telefonní číslo 0326 –713 954, 953