Informační a řídicí systémy I.
Programování PLC I. – IEC 61131-3

Pavel Balda
ZČU v Plzni, FAV, KKY

Osnova přednášky

- Společné rysy jazyků normy IEC 61131-3
- Stručný přehled jazyků
- Jednoduché příklady
- Úvod do sekvenčních funkčních grafů (SFC)

Doporučená literatura

- Bonfatti, F.; Monari, D.P.; Sampieri, U.: IEC1131-3 Programming Methodology. CJ International/Groupe ALterSys, France, 2001.
- IEC-61161-3 2nd Ed. Final Draft, 2001-04-16
- PLC open: **http://www.plcopen.org**
- ISaGraf: **http://www.isagraf.com**

Norma IEC 61131-3

- Norma IEC 61131-3 se zabývá programováním PLC (Programmable Logic Controller, programovatelný automat)
- Společné prvky (Common elements)
  - Softwarový model
  - Literály
  - Identifikátory
  - Datové typy
  - Proměnné
- Programovací jazyky
  - Sekvenční funkční grafy SFC (Sequential Function Charts)
  - Strukturovaný text ST (Structured Text)
  - Seznam instrukcí IL (Instruction List)
  - Funkční bloky FBD (Function Block Diagram)
  - Liniové schéma LD (Ladder diagram)

Ideje programovacích jazyků IEC 61131-3

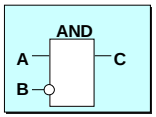
Seznam instrukcí (IL)

```
LD    A
ANDN  B
ST    C
```

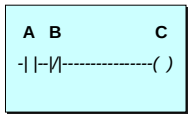
Strukturovaný text (ST)

```
C:= A  AND  NOT B
```

Funkční bloky (FBD)

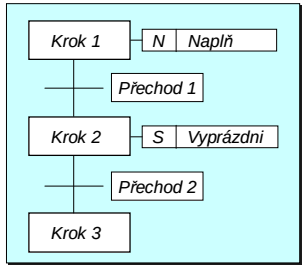


Liniové schéma (LD)

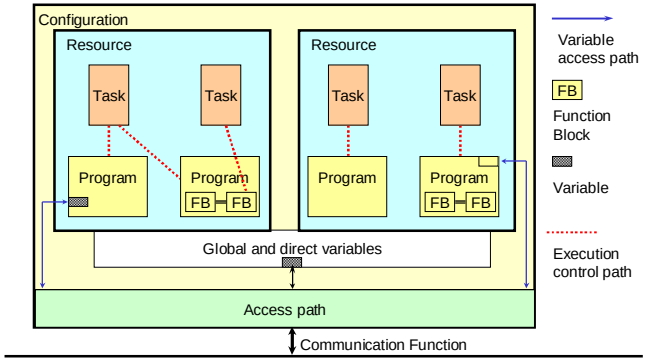


Idea sekvenčních funkčních grafů

- Mocná grafická technika pro popis sekvenčních řídicích programů
- Podobné technice stavových diagramů
- Používá se pro dekompozici složitých řídicích algoritmů
- Přehledné znázornění, vhodné i pro rychlý návrh diagnostiky



Softwarový model v IEC 61131-3 (1/2)



Softwarový model v IEC 61131-3 (2/2)

- Konfigurace (Configuration)
  - Odpovídá programovatelnému systému (programmable controller system) podle IEC 61131-1
  - Konfigurace obsahuje jeden nebo několik zdrojů
- Zdroj (Resource)
  - Provádí zpracování signálů (signal processing function), jeho uživatelské rozhraní (HMI – Human Machine Interface) rozhraní pro senzory a akční členy (sensor and actuator interface)
  - Každý zdroj obsahuje jeden nebo několik programů
- Program
  - Základní jednotka vykonávající kód
  - Může obsahovat nulu, jeden nebo více funkčních bloků nebo jiných prvků jazyků obsažených v této normě
  - Vykonávání programů může být řízeno nulou, jednou nebo několika úlohami (Task)

Identifikátory a komentáře

- Identifikátor
  - Řetězec složený písmen, číslic a znaků '_' (podtržítko) začínající písmenem nebo podtržítkem, např. `MAX_U_24`, `ahoj`, `a_hoj`
  - Nerozlišují malá a velká písmena, tj. `ahoj`, `AHOJ` a `Ahoj` jsou stejné identifikátory
  - Podtržítko nesmí být na konci a v ostatních případech nesmí jich být několik přímo za sebou, např. `__MAX_U`, `_MAX__U` nebo `MAX_U_` nejsou správné identifikátory!
- Klíčová slova
  - Vyhrazené identifikátory pro syntaktické prvky jazyka
  - Neměla by být používána pro označování uživatelských proměnných
- Komentáře
  - Posloupnosti znaků mezi speciálními kombinacemi `(*` a `*)`, např. `(* toto je komentář *)`
  - Kromě IL a řetězcových literálů jsou dovoleny všude, kde může být mezera
  - Nesmí se vnořovat, není povoleno např. `(* (* vnořený *) *)`

9

Číselné literály

- Číselné (numerické literály) slouží pro zápis číselných konstant
  - Mohou uvnitř obsahovat znak '_' (podtržítko) !
  - Celočíselné
    - Dekadické – mohou zapisovat běžným způsobem čísla v desítkové soustavě a mohou mít znaménko '+' nebo '–', např. `-12`, `0`, `123_456`, `+789`
    - V soustavách o základech `2`, `8` nebo `16` – zapisují se s prefixem číselné soustavy následované znakem '#' a nesmějí obsahovat znaménko, např. `2#1110_0000` (240 dekadicky), `8#377` (255 dekadicky), `16#e5` (245 dekadicky)
    - Dále mohou explicitně obsahovat typ uvedený jako prefix a oddělený znakem '#', např. `DINT#5`, `UINT#16#FE`
  - Reálné
    - obsahují desetinnou tečku a mohou mít exponent (i se znaménkem), např. `-12.0`, `3.14159_26`, `1.234e-6`
  - Booleovské – `0` nebo `FALSE` a `1` nebo `TRUE`

10

Řetězcové literály

- Řetězce jednobajtových znaků
  - Uzavřené do jednoduchých uvozovek (apostrofů) '', např. `''`, `'A'`, `' '`, `'"'`
  - Náhradní posloupnosti dvouznakové začínající '$' – viz tabulka, např. `'$''`, `'$R$L'`, `'$$1.00'` ($1.00)
  - Tříznakové kombinace začínající znakem '$' – druhé dva znaky se interpretují jako hexadecimální kód znaku kterým je nahrazena v textu, např. `'$0A'` (LF), `'$C4$CB'` (`'ÄË'`)
- Řetězce dvoubajtových znaků
  - Uzavřené do dvojitých uvozovek '""'
  - Lze užívat 5-znakové kombinace začínající '$' – další 4 znaky jsou kódem tištěného znaku

Kombinace	Při tisku
`$$`	Znak dolar
`$'`	Jednoduchá uvozovka
`$L` nebo `$l`	Konec řádku (LF)
`$N` nebo `$n`	Nová řádka
`$P` nebo `$p`	Nová stránka (FF)
`$R` nebo `$r`	Návrat vozíku (CR)
`$T` nebo `$t`	Tabulátor
`$"`	Dvojitá uvozovka

- Mohou explicitně obsahovat prefix typu odděleného znakem '#', např. `STRING#'OK'`, `WSTRING#"OK"`

11

Časové literály

- Literály pro dobu trvání
  - Pro měření uplynulého času od určitého okamžiku
  - Udávají čas ve dnech, hodinách, minutách, vteřinách a milisekundách,
  - Nejméně významná jednotka může být zapsána jako reálné číslo bez exponentu
  - Prefixy `T#`, `TIME#`, `t#` nebo `time#`
  - Příklady: `T#5d14h12m18s3.5ms`, `TIME#14.7h`, `time#25h15m`
- Literály pro absolutní čas a datum
  - Pro synchronizaci začátku a konce dané události s absolutním časem
  - Datum se udává ve tvaru rok-měsíc-den, čas ve tvaru hodina:minuta:vteřina
  - Prefixy `TIME_OF_DAY#`, `TOD#`, `DATE#`, `D#`, `DATE_AND_TIME#`, `DT#`
  - Příklady: `DATE#2006-03-20`, `time_of_day#10:42:23.78`, `DT#2006-03-20-10:42:23.78`

12

Jednoduché datové typy

Název	Datový typ	Bitů	Inic.	Název	Datový typ	Bitů	Inic.
BOOL	Boolean	1	0	TIME	Duration (doba trvání)	*	T#0s
SINT	Short integer	8	0	DATE	Date (only)	*	D#0001-01-01
INT	Integer	16	0	TOD	Time of day (only)	*	TOD#00:00:00
DINT	Double integer	32	0	DT	Date and time of day	*	jako D#+TOD#
LINT	Long integer	64	0	STRING	Variable-lenght char string	*	''
USINT	Unsigned short int	8	0	WSTRING	Var.-length 2-byte char str	*	""
UINT	Unsigned int	16	0	BYTE	Bit string of length 8	8	
UDINT	Unsign. double int	32	0	WORD	Bit string of length 16	16	
ULINT	Unsigned long int	64	0	DWORD	Bit string of length 32	32	
REAL	Real number	32	0.0	LWORD	Bit string of length 64	64	
LREAL	Long real number	64	0.0				

13

Hierarchie jednoduchých typů

- ANY
  - ANY_DERIVED
  - ANY_ELEMENTARY
    - ANY_MAGNITUDE
      - ANY_NUM
        - ANY_REAL
          - LREAL, REAL
        - ANY_INT
          - LINT, DINT, INT, SINT
          - ULINT, UDINT, UINT, USINT
      - TIME
    - ANY_BIT
      - LWORD, DWORD, WORD, BYTE, BOOL
    - ANY_STRING
      - STRING, WSTRING
    - ANY_DATE
      - DATE_AND_TIME, DATE, TIME_OF_DAY
- Generické datové typy – identifikované prefixem '**ANY**'
  - Používají se pro specifikaci přetížených vstupů a výstupů standardních funkcí a funkčních bloků

14

Odvozené datové typy (1/2)

- Odvozené datové typy (derived data types)
  - Specifikovány uživatelem nebo výrobcem PLC
  - Deklarují se pomocí konstrukce **TYPE**...**END_TYPE**
  - `TYPE Teplota : REAL := 10.0; END_TYPE`
- Vyjmenované typy (enumerated types)
  - Hodnoty datových prvků mohou nabývat pouze hodnot z deklarovaného seznamu
  - Počáteční hodnotou je hodnota prvního prvku nebo hodnota nastavená operátorem přiřazení
  - `TYPE PracovniRezim : (Rucni, Auto, Vyp) := Vyp; END_TYPE`
- Deklarace rozsahu (subrange)
  - Určuje přípustný rozsah všech hodnot daného typu, zadávaný spodní a horní mezí. Není li použit přiřazovací operátor, je počáteční hodnotou první (spodní) mez
  - `TYPE Teplota : REAL (-1.0..+200.0):= 10.0; END_TYPE`

15

Odvozené datové typy (2/2)

- Struktura (structure)
  - Deklarace struktury určuje datové typy vložených prvků (sub-elements), které jsou dostupné svými jmény
  - Každý z vložených prvků může být jednoduchý typ nebo struktura
  - ```
    TYPE CidloTeploty
      Tepl : Teplota;
      Kalibrace : DATE;
      HorniMez : REAL := 180.0;
    END_TYPE
    ```
- Pole (array)
  - Deklarace specifikuje typ každého prvku a rozsah indexů
  - Prvky daného pole mohou být inicializovány různými hodnotami
  - `TYPE CelaCisla : ARRAY [1..10] OF INT; END_TYPE`
  - ```
    TYPE TeplotyPece : ARRAY [1..20] OF Teplota :=
      [10(-1.0), 10(1.0)];
    END_TYPE
    ```
- Řetězec (string)
  - Maximální délka stringu je závislá na implementaci
  - `TYPE STR10 : STRING[10] := 'ABCDEF'; END_TYPE`

16

Proměnné (1/3)

- Proměnné (variables) uchovávají data, jejichž obsah se může měnit
  - Data přidružená ke vstupům (inputs), výstupům (outputs) a vnitřní paměti (memory) programovatelného automatu
  - Proměnné se deklarují v deklarační části na začátku programu, funkce nebo funkčního bloku
- Deklarační část (declaration part) specifikuje typ a případně fyzické nebo logické umístění proměnné v dané programové jednotce (Program Organization Unit, POU)
- Rozsah platnosti (scope) proměnných je lokální v rámci dané POU, kde je proměnná deklarována. Výjimku tvoří jen globální proměnné (viz dále)
- **VAR** – klíčové slovo pro deklaraci vnitřních (internal) proměnných
  - Používají se interně v dané POU k ukládání výsledků výpočtů
- **VAR_INPUT** – klíčové slovo pro deklaraci vstupních proměnných
  - Používají se pro proměnné, jejichž zdroj pochází z vnějšku dané POU
  - Pro program odpovídají vstupům z čidel
  - Pro funkci nebo FB jsou vstupními parametry

Proměnné (2/3)

- **VAR_OUTPUT** – klíčové slovo pro deklaraci výstupních proměnných
  - Používají se pro proměnné nastavované v dané POU pro vnější entity
  - Pro program odpovídají výstupům na akční členy (actuators) generovanými programem
  - Pro funkci nebo FB jsou výstupními parametry; mohou být dále používány v dané POU
- **VAR_IN_OUT** – klíčové slovo pro deklaraci vstupně-výstupních proměnných
  - Získávají svou hodnotu mimo danou POU, mohou však v ní být modifikovány a tyto nové hodnoty lze používat vně dané POU.
  - Lze jimi realizovat „volání odkazem“ známé z jazyků jako Pascal nebo C.
- **VAR_GLOBAL** – klíčové slovo pro deklaraci globálních proměnných
  - Lze k nim přistupovat ze všech POU v daném programu
  - V daném POU je třeba uvést klíčové slovo **VAR_EXTERNAL**
  - Typ uvedený ve **VAR_GLOBAL** daného programu a **VAR_EXTERNAL** dané POU musí souhlasit!
- **VAR_ACCESS** – klíčové slovo pro proměnné, které mohou být předávány prostřednictvím komunikace a přístupové cesty (access path) mezi programy

Proměnné (3/3)

- Proměnné ponechávající si hodnotu (retentive variables)
  - V některých implementacích může být počáteční hodnota proměnné nastavena z uchované hodnoty (retained value) uložené při ukončení předchozího běhu
  - Tím je umožněno provádět tzv. teplý restart (warm restart)
- Jednoprvkové proměnné (single-element variables)
  - Proměnné tvořené jedním datovým prvkem jednoduchého, výčtového nebo rozsahového typu, nebo odvozeného jednoduchého typu od uvedených typů
  - Mohou být reprezentovány symbolicky pomocí identifikátorů nebo přímo (directly) pomocí znaku '**%**' (procento) následovaného prefixem umístění (location) a prefixem velikosti (size), viz další stránku
- Víceprvkové proměnné (multiple element variables)
  - Proměnné typů pole nebo struktura
  - K prvkům struktur se přistupuje pomocí znaku '**.**' (tečka)
  - K prvkům pole pomocí indexů v hranatých závorkách '**[**' a '**]**'
  - Nelze je používat v jazyku seznam instrukcí (IL)
- **AT** – klíčové slovo přiřazující fyzickou nebo logickou adresu symbolicky reprezentované proměnné

Přímo reprezentované proměnné

- Přímá reprezentace proměnné se skládá ze znaku '**%**', prefixu umístění (první část tabulky), prefixu velikosti (druhá část tabulky) a jednoho nebo několika celých čísel bez znaménka oddělených tečkami. Počet čísel oddělených tečkami je implementačně závislý parametr daný výrobcem

Prefix	Význam	Datový typ
I	Vstup (Input location)	
Q	Výstup (Output location)	
M	Paměť (Memory location)	
X	Jeden bit	**BOOL**
žádný	Jeden bit	**BOOL**
B	Byte (8 bitů)	**BYTE**
W	Word (16 bitů)	**WORD**
D	Double Word (32 bitů)	**DWORD**
L	Long (quad) Word (64 bitů)	**LWORD**

Proměnné – příklady (1/2)

- Přímo reprezentované, ne-retentivní proměnné
  - ```
    VAR
      AT %IW6.2 : WORD;      (* 16 bitový řetězec *)
      AT %MW6   : INT;       (* 16 bit integer inicializovaný na 0 *)
      AT %QX5.1 : BOOL := 1; (* Boolean inicializovaný na 1 *)
      AT %MW7   : INT := 8;  (* 16 bit integer inicializovaný na 8 *)
    END_VAR
    ```
- Přímo reprezentované, retentivní proměnné
  - ```
    VAR RETAIN
      AT %QW4 : WORD;
      AT %QW5 : WORD := 16#FF00;
    END_VAR
    ```
  - Po studeném startu je `%QW4` inicializována na 0, `%QW5` má nejvyšších 8 bitů inicializováno na 1, ostatní na 0
- Umístění symbolických proměnných
  - ```
    VAR
      Lim5 AT %IX27 : BOOL;   (* Lim5 je 27. vstupní bit *)
      Teplota AT %IW28 : INT; (* Teplota je 28. vstupní slovo*)
      PolohaVentilu AT %QW29 : INT := 100; (* 29. Výstupní WORD je
          přiřazen INT proměnné PolohaVentilu inicializované na 100 *)
    END_VAR
    ```


Proměnné – příklady (2/2)

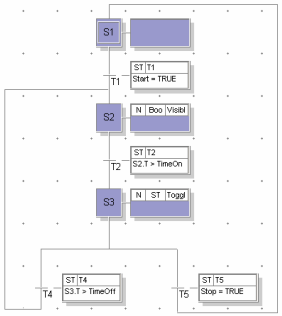
- Umístění a inicializace polí
  - ```
    VAR
      Outs AT %QW6 : ARRAY [0..9] OF INT := [10(1)];
      Bits : ARRAY [0..7] OF BOOL := [1,1,0,0,0,1,0,0];
      Tbt : ARRAY [1..2][1..3] OF INT := [1,2,3(4),6];
    END_VAR
    ```
  - `Outs` je pole 10 celých čísel inicializovaných na 1 souvisle alokovaných od pozice `%QW6`
  - Pole `Tbt` je pole 2*3 celých čísel: `Tbt[1,1]:=1; Tbt[1,2]:=2; Tbt[1,3]:=4; Tbt[2,1]:=4; Tbt[2,2]:=4; Tbt[2,3]:=6;`
- Deklarace a inicializace retentivního pole
  - ```
    VAR RETAIN
      RTbt : ARRAY [1..2][1..3] OF INT := [1,2,3(4)];
    END_VAR
    ```
  - Po studeném startu je `RTbt` inicializováno na stejné hodnoty jako `Tbt` v předchozím příkladu, kromě `RTbt[2,3]:=0;`
- Automatická alokace symbolických proměnných
  - ```
    VAR
      Den : WORD;               (* Den je 16 bit. string *)
      Awd, Bwd, Cwd : INT;      (* 3 celočíselné proměnné *)
      Okay : STRING[10] := 'OK'; (* zinicializovaný řetězec *)
    END_VAR
    ```


Sekvenční funkční grafy (SFC)

- Sekvenční funkční grafy (Sequential Function Charts, SFC) jsou určeny pro návrh sekvenčního řízení
  - Výrazově bohatý grafický formalismus podobný stavovým diagramům
  - Umožňují rozčlenit složitý program na množinu kroků (stavů) a přechodů mezi nimi
  - Ke každému kroku je přidružena množina akcí
  - Ke každému přechodu je přidružena množina podmínek

SFC – Jednoduchý příklad

- Příklad „blikač“
  - **`S1`** – výchozí stav
  - **`T1`** – při nastavení signálu **`Start`** na **`TRUE`** se přechází do stavu **`S2`**
  - **`S2`** – Nastaví se proměnná **`Visible`** na **`TRUE`**
  - **`T2`** – po setrvání ve stavu **`S2`** po dobu větší než **`TimeOn`** se přechází do stavu **`S3`**
  - **`S3`** – invertuje se proměnná **`Visible`**
  - **`T4`** – po uplynutí času většího než **`TimeOff`** se ze stavu **`S3`** přechází do **`S2`** a postup se opakuje
  - **`T5`** – je-li nastavena proměnná **`Stop`**, přechází se ze stavu **`S3`** do výchozího stavu **`S1`**



Krok


```
STEP Si:
  (* deklarace akcí *)
END_STEP
```

- Krok (Step) určuje situaci, kdy chování vstupů a výstupů dané POU je definováno přidruženými akcemi
  - Krok lze reprezentovat graficky nebo textově, viz obrázek
- **Si.X** – příznak kroku (step flag)
  - Udává, zda je krok aktivní nebo neaktivní a je reprezentován booleovskou proměnnou **Si.X**, kde **Si** je název kroku
- **Si.T** – uplynulý čas (elapsed time)
  - Udává čas po který setrvává SFC v daném kroku
  - Tento čas v proměnné typu **TIME** je po opuštění kroku ponechán na hodnotě, jakou měl v okamžiku opuštění, při aktivaci kroku je nastaven na hodnotu **t#0s**

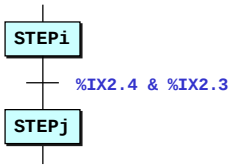
Přechod


```
TRANSITION Tij:
  (* podmínka přechodu *)
END_TRANSITION
```

- Přechod (Transition) představuje podmínku, za které se řízení předává z jednoho nebo více předchozích kroků do jednoho nebo více následujících kroků
  - Přechod lze reprezentovat graficky nebo textově, viz obrázek.
  - Při splnění podmínky vyhodnocované jako booleovský výraz dojde k přechodu ve směru shora dolů
- Podmínka přechodu může být k daném přechodu přidružena následovně (viz příklady na následujících stránkách):
  1. Jako booleovský výraz v ST
  2. Jako liniové schéma (LD) jehož výstup přiléhá (adjacent) k vertikální lince
  3. Jako FBD jehož výstup přiléhá k vertikální lince
  4. Jako LD nebo FBD připojený k vertikální lince pomocí konektoru
  5. Jako konstrukce **TRANSITION**...**END_TRANSITION** užívající ST
  6. Jako konstrukce **TRANSITION**...**END_TRANSITION** užívající IL
  7. Jménem přechodu přidruženého ke konstrukci **TRANSITION**...**END_TRANSITION**, v jejímž těle je kód v LD, FBD, IL nebo ST vracející hodnotu booleovské proměnné

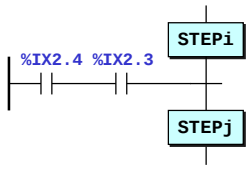
Příklady přechodů (1/3)

Ad 1.



Podmínka přechodu zapsaná pomocí ST

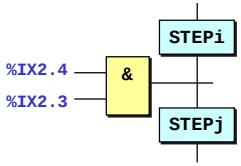
Ad 2.



Podmínka přechodu zapsaná pomocí LD

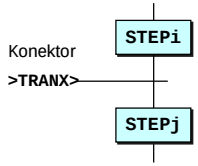
Příklady přechodů (2/3)

Ad 3.

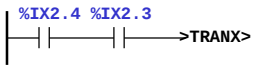


Podmínka přechodu zapsaná pomocí FB

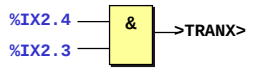
Ad 4.



4a. Podmínka přechodu zapsaná pomocí LD



4b. Podmínka přechodu zapsaná pomocí FBD



Příklady přechodů (3/3)

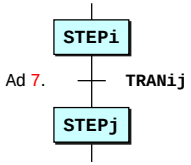
Ad 5. Textový ekvivalent případu 1. využívající jazyk ST

```
STEP STEPi: END_STEP
TRANSITION FROM STEPi TO STEPj
  := %IX2.4 & %IX2.3;
END_TRANSITION
STEP STEPj: END_STEP
```

Ad 6. Textový ekvivalent případu 1. využívající jazyk IL

```
STEP STEPi: END_STEP
TRANSITION FROM STEPi TO STEPj:
  LD  %IX2.4
  AND %IX2.3;
END_TRANSITION
STEP STEPj: END_STEP
```

Ad 7.



7c. Podmínka přechodu zapsaná pomocí IL

```
TRANSITION TRANij FROM STEPi TO STEPj:
  LD  %IX2.4
  AND %IX2.3;
END_TRANSITION
```

7d. Podmínka přechodu zapsaná pomocí ST

```
TRANSITION TRANij FROM STEPi TO STEPj
  := %IX2.4 & %IX2.3;
END_TRANSITION
```

29

Akce (1/2)

- Akce jsou v SFC přidruženy ke krokům
  - Ke každému kroku může být přidružena žádná, jedna nebo několik akcí
  - Krok, který nemá žádnou akci se považuje za „čekací“ do té doby, než nastane podmínka jeho opuštění
- Deklarace akce se skládá ze jména akce a z těla akce
  - Rozsah platnosti dané akce (scope) je v rámci dané POU obsahující deklaraci
- Tělem akce může být:
  - Booleovská proměnná
  - Posloupnost instrukcí v IL
  - Posloupnost příkazů v ST
  - Posloupnost příček v LD
  - Propojení bloků ve FBD
  - Další SFC
- Akce se přidružují ke krokům buď v grafických blocích nebo textově

30

Akce (2/2)

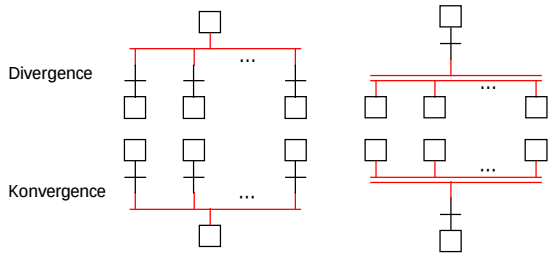
- Chování akce je určeno tzv. kvalifikátorem akce, viz tabulku
  - Podrobněji v samostatné přednášce o SFC

Kvalif.	Definice	Chování akce
žádný	Non-stored (null quailier)	Jako kvalifikátor **N**
N	Non-stored	Provádí se, když je daný krok aktivní
R	overriding Reset	Ukončuje provádění akcí s kvalifikátorem **S**, **SD**, **SL**
S	Set (Stored)	Akce se provádí, dokud není dosažen stav v němž má kval. **R**
L	time Limited	Akce se provádí po dobu zadanou parametrem v kvalifikátoru
D	time Delayed	Akce se spustí za čas daný parametrem v kvalifikátoru
P	Pulse	Provede se, když je daný krok aktivován
SD	Stored and time Delayed	Po uplynutí zadaného zpoždění spouští akci jako při **S**
DS	Delayed and Stored	Akce se začne provádět, trvá-li daný stav alespoň zadaný čas
SL	Stored and time Limited	Akce se provádí jako při **S**, ale jen do uplynutí zadaného času
P1	Pulse (rising edge)	Akce se provede pouze jednou po náběžné hraně pulsu
P0	Pulse (falling edge)	Akce se provede pouze jednou po sestupné hraně pulsu

31

Divergence a konvergence (1/3)

- Prostředky větvení programu v SFC
- Divergence – vícenásobné spojení z jednoho symbolu (kroku nebo přechodu) na několik symbolů opačného typu
- Konvergence – vícenásobné spojení z několika symbolů (kroků nebo přechodů) na jeden symbol opačného typu
- Divergence a konvergence mohou být jednoduché nebo dvojité, viz obr.



32

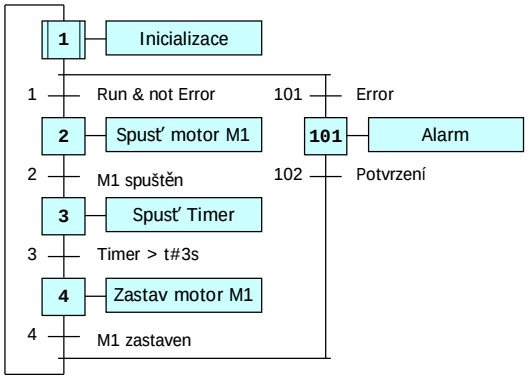
Divergence a konvergence (2/3)

- Jednoduchá divergence – spojení jednoho kroku s několika přechody
  - Z přidružených podmínek přechodu může být splněna nejvýše jedna (detaily viz samostatnou přednášku k SFC)
  - Při splnění jedné podmínky se začne vykonávat po ní následující krok
  - Může tedy běžet vždy jen jedna z větví vzniklých touto divergencí, proto se jí někdy říká OR-divergence
- Jednoduchá konvergence – spojení několika přechodů s jedním krokem
  - Používá se pro spojení větví vzniklých jednoduchou divergencí
- Speciální případy:
  - Tzv. přeskočení sekvence (sequence skip), kdy některá větev neobsahuje ani kroky ani přechody
  - Sekvenční smyčka (sequence loop), kdy některá nebo několik větví se vrací do některého z předchozích stavů

Divergence a konvergence (3/3)

- Dvojitá divergence – spojení jednoho přechodu s několika kroky
  - Odpovídá paralelnímu běhu procesů reprezentovaných jednotlivými větvemi, nazývanými současně běžící sekvence (simultaneous sequences)
  - Proto se ji někdy říká AND-divergence
  - Spustí se, je-li aktivní předchozí krok a splní-li se podmínka vycházející z něj do této divergence. Pak se zinicializují všechny počáteční kroky všech současně běžících sekvencí
- Dvojitá konvergence – spojení několika kroků s jediním přechodem
  - Používá se pro spojení větví vzniklých jednoduchou divergencí
  - Spouští se, když jsou aktivní všechny kroky k ní připojené a je splněna z ní vycházející podmínka
  - Za konvergencí jsou všechny předchozí aktivní kroky deaktivovány a je aktivován jediný krok následující po podmínce vycházející z divergence

Příklad jednoduché divergence a konvergence



Příklad dvojité divergence a konvergence