Vložené řídicí systémy
Programování PLC II. – IEC 61131-3

Pavel Balda
ZČU v Plzni, FAV, KKY

Osnova přednášky

- Strukturovaný text (ST)
- Liniová (kontaktní) schémata (LD)
- Schémata složená z funkčních bloků (FBD)

Strukturovaný text (ST)

- Strukturovaný text (Structured text, ST) je textový programovací jazyk
  - Vyšší programovací jazyk podobný jazyku Pascal (a částečně C) navržený pro programování řídicích algoritmů
  - Používá se zejména tam, kde není snadné použít grafické programovací jazyky, např. při implementaci složitých procedur
  - Implicitní jazyk pro popis podmínek přechodu a akcí v SFC
- Pro textové programovací jazyky (druhým je seznam instrukcí, IL) definuje norma IEC 61131-3 řadu konstrukcí, např:
  - Deklarace typů
  - Deklarace proměnných
  - Deklarace kroků, přechodů a akcí pro SFC
  - Deklarace funkcí a funkčních bloků

Program ve strukturovaném textu

- Program ve strukturovaném textu (ST)
  - Posloupnost příkazů ST
  - Každý příkaz je ukončen oddělovačem '`;`' (středník)
  - Oddělovače – oddělují identifikátory, literály a klíčová slova
    - Bílé znaky (white spaces) – mezera, tabulátor, nový řádek, apod. Chovají se jako mezera (znak s dekadickým kódem 32)
    - Aktivní oddělovače – zejména operátory a závorky
    - Bílé znaky a komentáře (tj. text mezi `(*` a `*)`) se mohou vyskytovat mezi aktivními oddělovači identifikátory a literály
  - Příkazy lze rozdělit do následujících kategorií
    - Přiřazovací příkaz (assignment statement), např. `Prom:=vyraz;`
    - Příkazy pro výběr (selection statements) – `IF`, `THEN`, `ELSE`, `CASE`,...
    - Iterační příkazy (iteration statements) – `FOR`, `WHILE`, `REPEAT`,...
    - Řídicí příkazy funkcí a funkčních bloků
    - Řídicí příkazy – `RETURN`, `EXIT`,...

Výrazy (Expressions)

- Výraz – konstrukce, která po svém vyhodnocení získá hodnotu jednoduchého nebo odvozeného typu
  - Skládají se z operátorů a operandů
  - Maximální délka výrazu závisí na implementaci
- Operand – literál, hodnota výčtového typu, proměnná, volání funkce nebo jiný výraz
- Operátory – viz následující tabulku, udávající jejich prioritu (precedence)
- Vyhodnocení výrazu – aplikování operátorů na operandy podle priority
  - Nejprve se vyhodnotí operátor s nejvyšší prioritou, pak operátor s nižší prioritou, atd., dokud není vyhodnocení výrazu dokončeno
  - Operátory se stejnou prioritou se ve výrazu vyhodnocují zleva doprava
  - Příklad:
    `A`, `B`, `C`, `D` jsou typu INT s hodnotami `1`, `2`, `3`, `4` (v uvedeném pořadí). Pak
    `A+B-C*ABS(D)` se vyhodnotí na `-9`
    `(A+B-C)*ABS(D)` se vyhodnotí na `0`

5

Priority operátorů

Operace	Symbol	Operace	Symbol
Nejvyšší priorita		Sčítání	`+`
Uzávorkování	`(`výraz`)`	Odčítání	`-`
Vyhodnocení funkce	jmeno_fn`(`argumenty`)`	Porovnávání	`<`, `>`, `<=`, `>=`
Negace	`-`	Rovnost	`=`
Doplněk (complement)	`NOT`	Nerovnost	`<>`
Umocňování	`**`	Booleovský AND	`&`, `AND`
Násobení	`*`	Booleovký exklusivní OR	`XOR`
Dělení	`/`	Booleovský OR	`OR`
Modulo	`MOD`	*Nejnižší priorita*	

6

Přiřazovací příkaz

- Přiřazovací příkaz nahrazuje aktuální hodnotu jednoprvkové nebo víceprvkové proměnné výsledkem vyhodnocení výrazu
  - Na levé straně je odkaz na přiřazovanou proměnnou
  - Pak následuje přiřazovací operátor '`:=`'
  - Na pravé straně je vyhodnocovaný výraz
- Příklad: `A := B;`
  Lze použít jak pro proměnné jednoduchých typů např. typu INT, tak i pro proměnné odvozených typů. V případě přiřazení proměnných typů struktur (stejného typu) se přiřazují hodnoty odpovídajících si prvků
- Přiřazovací příkaz se používá též pro přiřazení hodnoty funkci uvnitř jejího těla.

7

Příkaz IF

- Příkaz `IF` umožní vykonávat skupinu příkazů pouze tehdy, má-li odpovídající booleovský výraz hodnotu `1` (`TRUE`)
- Není-li podmínka splněna, nevykoná se žádný příkaz nebo se vykoná skupina příkazů za klíčovým slovem `ELSE` nebo se vykoná skupina příkazů za klíčovým slovem `ELSIF` pokud je splněna jemu příslušná booleovská podmínka
- Příklad:

```
D := B*B - 4*A*C;
IF D < 0.0 THEN Nroots := 0;
ELSIF D = 0.0 THEN
  Nroots := 1;
  X1 := -B/(2.0*A);
ELSE
 Nroots := 2;
 X1 := (-B + SQRT(D))/(2.0*A);
 X2 := (-B - SQRT(D))/(2.0*A);
END_IF
```

8

Příkaz CASE

- Příkaz **CASE** se skládá z výrazu, který se vyhodnotí na typ **INT** (selektor), a ze seznamu skupin příkazů, z nichž každá je označena jedním nebo několika celými čísly nebo intervaly celých čísel
- Pro danou vypočtenou hodnotu selektoru se spustí první skupina příkazů, která tuto hodnotu obsahuje
- Pokud hodnota selektoru není obsažena v žádném výše uvedeném rozsahu, spustí se skupina příkazů za klíčovým slovem **ELSE**, vyskytuje-li se v daném příkazu **CASE**. Není-li klíčové slovo **ELSE** uvedeno, neprovede se žádný příkaz
- Příklad:

```
sel := BcdToInt(predvolba);
chyba := 0;
CASE sel OF
  1, 5:     Displej := TeplotaPece;
  2:        Displej := RychlostMotoru;
  3:        Displej := SpotrebaPlynu;
  4, 6..10: Displej := Stav[sel-4];
  ELSE      Displej := 0;
            chyba := 1;
END_CASE
```


Příkaz FOR (1/2)

- Příkaz **FOR** se používá pro cykly, kdy je dopředu znám počet iterací (v opačném případě se používají příkazy **WHILE** a **REPEAT**, viz dále)
- Vykonává posloupnost příkazů od klíčového slova **FOR** až k **END_FOR** po dobu, než řídicí proměnná dosáhne koncové hodnoty
- Řídicí proměnná je inkrementována (dekrementována) o zadanou hodnotu za klíčovým slovem **BY** (není-li uvedeno je hodnota implicitně rovna **1**)
- Počáteční a koncová hodnota (před a za klíčovým slovem **TO**) se získají vyhodnocením výrazů stejného celočíselného typu (**SINT**, **INT** nebo **DINT**) a neměla by se měnit výrazy uvnitř cyklu
- Test na ukončení cyklu se provádí na začátku cyklu, takže cyklus nemusí proběhnout ani jednou
- Hodnota řídicí proměnné cyklu po ukončení cyklu je implementačně závislá!

Příkaz FOR (2/2)

- Iterační příkazy (**FOR**, **WHILE** a **REPEAT**) lze předčasně ukončit příkazem **EXIT**, kterým se předá řízení na první příkaz za ukončovací klíčové slovo **END_FOR**, **END_WHILE** nebo **END_REPEAT** nejvnitřnějšího cyklu, v němž je **EXIT** obsažen
- Příklad
  - V cyklu **FOR** se určuje první výskyt řetězce **'heslo'** v lichých indexech pole řetězců **Slova** v rozsahu indexů **[1..100]**. Není-li řetězec obsažen, bude mít proměnná **J** hodnotu **101**.
  - V případě nalezení zadaného slova se ukončí vykonávání cyklu příkazem **EXIT**, aby se neprováděly zbytečné iterace

```
J := 101;
FOR I := 1 TO 100 BY 2 DO
  IF Slova[I] = 'heslo' THEN
    J := I;
    EXIT;
  END_IF;
END_FOR;
```


Příkaz WHILE

- Příkaz **WHILE** provádí opakovaně posloupnost příkazů až ke klíčovému slovu **END_WHILE** dokud je splněna daná podmínka
- Není-li splněna podmínka už od začátku, neprovede se posloupnost příkazů ani jednou
- Příklad (upravený příklad z předchozí stránky pro cyklus **WHILE**)

```
J := 1;
WHILE J <= 100 & Slova[J] <> 'heslo' DO
  J := J+2;
END_WHILE;
```

Příkaz REPEAT

- Příkaz **REPEAT** provádí opakovaně posloupnost příkazů až ke klíčovému slovu **END_REPEAT** dokud je splněna daná podmínka
- Posloupnost příkazů se provede vždy alespoň jednou
- Příklad z předchozí stránky upravený pro cyklus **REPEAT**)

```
J := -1;
REPEAT
  J := J+2;
UNTIL J = 101 OR Slova[J] = 'heslo'
END_REPEAT;
```

- Příkazy WHILE a REPEAT by se neměly používat pro synchronizaci procesů, např. jako „čekací“ smyčka s externě nastavovanou ukončovací podmínkou.
  - K tomu účelu slouží „současně běžící sekvence“ v SFC (dvojité divergence)

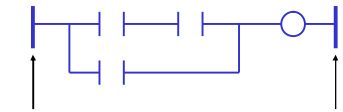
Volání funkcí a funkčních bloků

- Z ST lze volat funkce a funkční bloky
- Funkce mohou být volány jako součást výrazů
- Funkční bloky se volají příkazem skládajícím se ze jména FB následovaného v závorkách uzavřeným seznamem přiřazení hodnot vstupním parametrům
  - Příklad:

```
(* Volání funkčního bloku *)
Monitor (In := %IX5, Pt := T#300ms);
(* Použití výsledku funkčního bloku *)
A := Monitor.Q;
```


Liniová (kontaktní) schémata (LD)

- Liniové (kontaktní) schéma (Ladder Diagram, LD) je grafická reprezentace Booleovských výrazů kombinujících kontakty (vstupní argumenty) s cívkami (coils) (výstupní argumenty)
- V LD se používají grafické symboly zorganizované podobně jako příčky na žebříku
- LD je omezen po obou stranách levou a pravou vodicí tyčí (power rail), viz obr.
- Propojovací linie (spojení) spojují vodicí tyče. Mohou být horizontální i vertikální



Levá vodicí tyč (Left power rail)

Pravá vodicí tyč (Right power rail)

Základní grafické symboly LD

Levá vodicí tyč
(Left power rail)

Pravá vodicí tyč
(Right power rail)

Vodorovná spojovací čára
(Horizontal connection line)

Svislá spojovací čára
(Vertical connection line)

Vícenásobné propojení –
všechny propojené dohromady

Kontakt svázaný s nějakou proměnnou

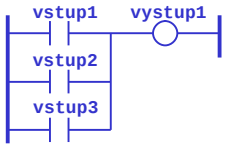
Cívka svázaná s výstupní nebo s vnitřní proměnnou

Stav prvků a jejich spojení v LD

- Každý prvek na spojovací čáře má svůj Booleovský stav – **1** (**TRUE**) nebo **0** (**FALSE**)
- Každé spojení má svůj stav podle stavu jednotlivých prvků, které obsahuje. Tento stav odpovídá „průtoku elektrického proudu“ ve spojení
- Levá vodicí tyč je trvale zapnuta (pod napětím), stav pravé tyče není předem definován
- Vodorovné spojení (Horizontal link) přenáší stav z prvku nejblíže vlevo na prvek nejblíže vpravo. Každá vodorovná čára připojená na levou vodicí tyč má stav **TRUE**
- Svislé spojení tvoří svislá čára připojená k jednomu nebo více vodorovným elementům na každé straně. Stav svislého spojení je určen logickým součtem (**OR**) stavů vodorovných elementů připojených zleva. Tedy stav svislého spojení je Off (**FALSE**), je-li stav všech vodorovných připojení přicházejících zleva Off (**FALSE**). Aby byl tento stav On (**TRUE**) stačí, aby stav alespoň jednoho připojení zleva byl On (**TRUE**)

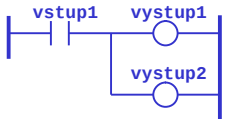
Vícenásobné připojení v LD

- Vícenásobné připojení vlevo
  - Stav prvku vpravo od připojení je **OR** stavu všech prvků vlevo (viz obr. vlevo)
- Vícenásobné připojení vpravo
  - Stav prvku vlevo se přenáší do všech prvků vpravo (viz obr. vpravo)



Ekvivalentně v ST:

```
vystup1 := vstup1 OR
    vstup2 OR vstup3;
```



Ekvivalentně v ST:

```
vystup1 := vstup1;
vystup2 := vstup1;
```


Kontakty a cívky

- Kontakt

  - Prvek, který nastavuje stav na vodorovném spojení své pravé straně jako logický součin (**AND**) stavu vodorovného spojení na své levé straně a odpovídající funkci proměnné, která jej reprezentuje (vstupní, výstupní, paměťová)
- Cívka

  - Prvek, kopírující stav spojení na levé straně na spojení na pravé straně bez modifikace a ukládá odpovídající funkci stavu do přidružené Booleovské proměnné
- Existují různé druhy kontaktů a cívek, viz dále

Statické kontakty

- Přímý (spínací) kontakt

Ekvivalentně v ST:

```
vystup1 := vstup1 AND vstup2;
```

- Invertovaný (rozpínací) kontakt

Ekvivalentně v ST:

```
vystup1 := NOT(vstup1) AND NOT(vstup2);
```

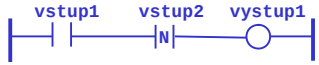
Kontakty s detekcí hrany

- Kontakt, detekující náběžnou (pozitivní) hranu

Ekvivalentně v ST:

```
vystup1 := vstup1 AND (vstup2 AND NOT vstup2predch);
(* vstup2predch je hodnota vstup2 v predchozim cyklu *)
```

- Kontakt, detekující sestupnou (negativní) hranu

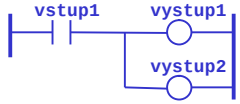


Ekvivalentně v ST:

```
vystup1 := vstup1 AND (NOT (vstup2) AND vstup2predch);
(* vstup2predch je hodnota vstup2 v predchozim cyklu *)
```


Cívky s okamžitým výstupem

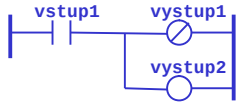
- Přímá cívka (direct coil)



Ekvivalentně v ST:

```
vystup1 := vstup1;
vystup2 := vstup1;
```

- Negovaná (inverzní) cívka

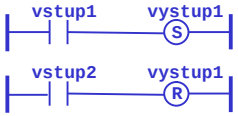


Ekvivalentně v ST:

```
vystup1 := NOT (vstup1);
vystup2 := vstup1;
```


Cívky přidržující výstup

- Set (latch) coil
  - Nastavuje přidruženou proměnnou na TRUE, pokud se stav levého spojení změní na TRUE
- Reset coil
  - Nastavuje přidruženou proměnnou na FALSE, pokud se stav levého spojení změní na TRUE



Ekvivalentně v ST:

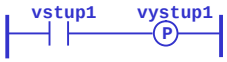
```
IF vstup1 THEN vystup1 := TRUE; END_IF;
IF vstup2 THEN vystup1 := FALSE; END_IF;
```


Cívky, ponechávající si hodnotu

- Cívky, ponechávající si hodnotu (retentive coils) jsou tří typů
  - Memory – odpovídá přímé cívce
  - Set memory – odpovídá cívce „set coil“
  - Reset memory – odpovídá cívce „reset coil“
- Jediný rozdíl je v tom, že se jejich stav ukládá do paměti jejíž stav není ovlivněn teplým startem.
  - Proměnné odpovídající těmto cívkám nemusí být deklarovány jako **VAR_RETAIN**

Cívky s detekcí hrany

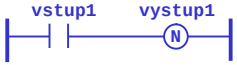
- Cívka, detekující náběžnou (pozitivní) hranu



Ekvivalentně v ST:

```
IF vstup1 AND NOT (vstup1predch) THEN vystup1 := TRUE;
   ELSE vystup1 := FALSE; END_IF;
(* vstup1predch je hodnota vstup1 v predchozim cyklu *)
```

- Cívka, detekující sestupnou (negativní) hranu

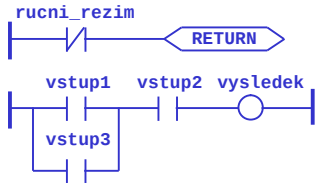


Ekvivalentně v ST:

```
IF NOT (vstup1) AND vstup1predch THEN vystup1 := TRUE;
   ELSE vystup1 := FALSE; END_IF;
(* vstup1predch je hodnota vstup1 v predchozim cyklu *)
```


Vyhodnocování LD a příkaz RETURN

- POU vytvořená v LD se vyhodnoduje následovně:
  - Schéma se vyhodnocuje shora dolů, není-li pořadí změněno příkazem **RETURN** nebo **JUMP**
  - Jednotlivé „příčky“ se vyhodnocují zleva doprava
- Příkaz **RETURN** umožňuje podmíněně ukončit vyhodnocování dané POU

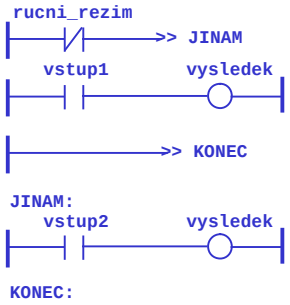


Ekvivalentně v ST:

```
IF NOT (rucni_rezim) THEN
  RETURN;
END_IF;
vysledek := (vstup1 OR vstup3)
   AND vstup2;
```


Příkaz JUMP

- Pro řízení pořadí zpracování sítě LD lze dále používat návěští, podmíněné i nepodmíněné skoky pomocí příkazu **JUMP**

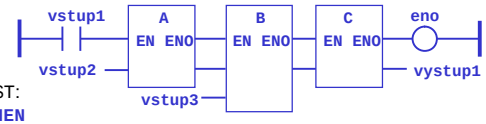


Ekvivalentně v IL (v ST nejsou skoky!):

```
        LDN  rucni_rezim
        JMPC JINAM
        LD   vstup1
        ST   vysledek
        JMP  KONEC
JINAM:
        LD   vstup2
        ST   vysledek
KONEC:
```


Funkce a Funkční bloky v LD

- Funkce a FB mohou být zapojeny do LD pokud mají Booleovské vstupy a výstupy
  - Vstupy jsou přímo připojeny do schématu
  - Výstupy jsou cívky už ve schématu existující nebo pro tento účel definované
  - Pokud funkce nebo FB nemá Booleovské vstupy (musí být připojeny ve schématu), existuje implicitně vstup EN, pomocí kterého „přitéká proud“ (power flow) do funkce
  - Je-li EN rovno TRUE, je blok povolen (enabled) a vykonává se
  - Obdobně existuje Booleovský výstup ENO, pomocí kterého „teče proud“ do další funkce nebo cívky. Výstup ENO je TRUE, pokud se funkce nebo FB vykonala úspěšně



Ekvivalentně v ST:

```
IF vstup1 THEN
  vystup1 := C(B(A(vstup2), vstup3));
END_IF;
```

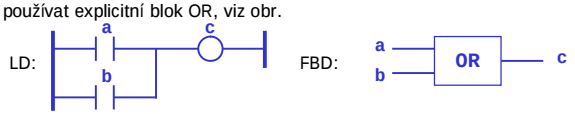
Schémata z funkčních bloků (FBD)

- Schémata z funkčních bloků (Function Block Diagrams, FBD) jsou grafickým programovacím jazykem
  - Interpretují tok signálu mezi jednotlivými prvky schématu
  - Analogické k toku signálu v elektronických obvodech
  - Popisují chování funkcí, funkčních bloků a programů jako množiny propojených grafických bloků (funkcí a funkčních bloků)
  - FBD lze použít pro detailní popis podmínek přechodů a akcí v SFC

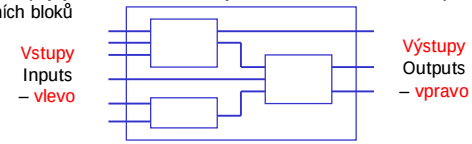
29

Propojení prvků ve schématu FBD (1/2)

- Jednotlivé prvky schémat jsou propojeny lomenými čarami s následujícími vlastnostmi
  - Čáry znázorňují tok signálu ve schématu
  - Výstupy bloků se nemohou přímo spojovat („zkratovat“)
  - Tj. není dovoleno spojování výstupů jako v jazyku LD. Místo něj je třeba používat explicitní blok OR, viz obr.



  - Bloky se spojují do schémat složených z elementárních bloků (funkcí nebo funkčních bloků



30

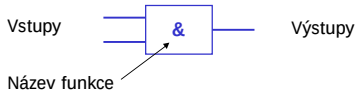
Propojení prvků ve schématu FBD (2/2)

- Vstupní proměnné programu ve FBD
  - Musí být připojeny ke vstupům bloků
  - Typ každé proměnné musí být odpovídat typu připojeného vstupu
  - Vstupem FBD může být konstantní výraz, vnitřní, vstupní nebo výstupní proměnná
- Výstupní proměnné programu ve FBD
  - Musí být připojeny k výstupům bloků
  - Typ každé proměnné musí být odpovídat typu připojeného výstupu
  - Výstupem FBD může být vnitřní nebo výstupní proměnná
- Jednotlivá propojovací čára může být použita k propojení:
  - Vstupní proměnné a vstupu bloku
  - Výstupu bloku a vstup jiného bloku
  - Výstupu bloku a výstupní proměnné
- Propojení má následující vlastnosti:
  - Je orientované, ve směru zleva doprava
  - Levé a pravé zakončení propojovací čáry je stejného typu
  - Lze použít několik pravých zakončení, která značí, že informace z levého zakončení je přenášena na několik dalších zakončení (všechny musí být stejného typu)

31

Schématická značka bloku

- Funkce bloku je vyznačena uvnitř obdélníkového symbolu bloku



- Propojovací čára může být zakončena symbolem negace
  - Malé kolečko na pravém konci propojovací čáry, viz příklad

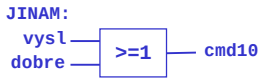
Ekvivalentně v ST:

```
c := a AND NOT(b);
```

32

Příkaz JUMP

- Pro řízení pořadí zpracování sítě FBD lze dále používat návěští a skok na něj pomocí příkazu **JUMP**
  - Při splnění Booleovské podmínky (**TRUE**) se vykonávání programu přenese za symbol návěští, viz příklad

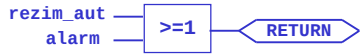


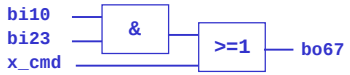
Ekvivalentně v IL (v ST nejsou skoky!):

```
        LD   rucne
        AND  b1
        JMPC JINAM
        ...  ...
JINAM:
        LD   vysl
        OR   dobre
        ST   cmd10
```


Příkaz RETURN a vyhodnocování FBD

- Příkaz **RETURN** umožňuje podmíněně ukončit vyhodnocování dané POU
  - Nabude-li připojená Booleovská hodnota **TRUE**, ukončí se daný program a zbývající část se nevykoná





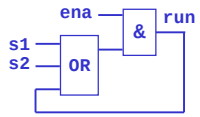
Ekvivalentně v ST:

```
IF rezim_aut OR alarm THEN
  RETURN;
END_IF;
bo67 := (bi10 AND bi23)
   OR x_cmd;
```

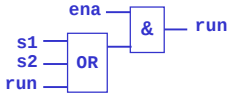
- Pořadí vyhodnocování sítě FBD se řídí následujícím pravidlem:
  - Vyhodnocení sítě je ukončeno před započetím vyhodnocování jiné sítě, využívající jeden nebo několik výstupů předchozí vyhodnocované sítě
  - Pořadí vyhodnocování dané sítě je implementačně závislé (!!!), obvykle shora dolů a zleva doprava

Zpětnovazební spojení bloků ve FBD

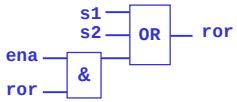
- Ve schématu existuje zpětná vazba (zpětnovazební cesta, feedback path), pokud je výstup nějaké funkce nebo funkčního bloku použit jako vstup funkce nebo funkčního bloku, který jej předchází (je dříve vyhodnocován)
  - Explicitní zpětná vazba – propojení výstupu daného bloku se vstupem dříve vyhodnocovaného bloku (obr. a)
  - Implicitní zpětná vazba – případ, kdy je výstup přiřazen do proměnné, použité pro vstup dříve vyhodnocovaného bloku (obr. b, c)
  - V případě explicitní zpětné vazby (obr. a)) není jednoznačně určeno pořadí, jak ji vykonávat (zda jako případ b nebo c)
  - Poznámka: v LD lze používat jen implicitní zpětnou vazbu



a)



b)



c)