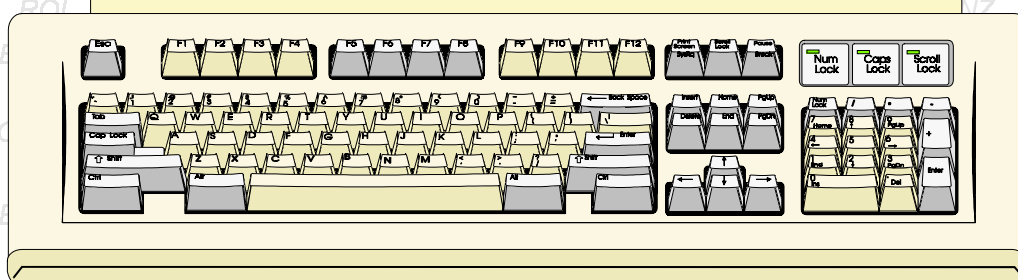
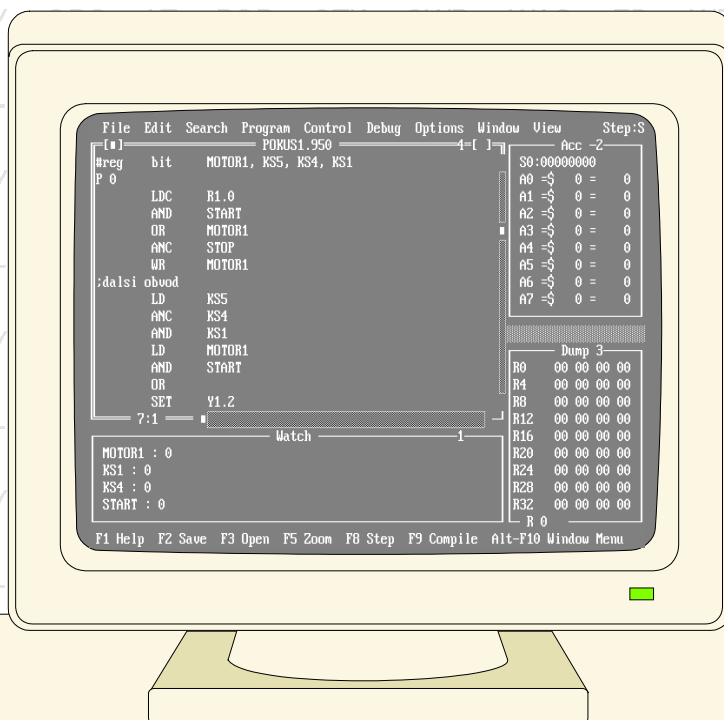


tecomat®

PROGRAMOVATELNÉ AUTOMATY



PŘÍKLADY PROGRAMOVÁNÍ PLC TECOMAT - MODEL 16 BITŮ

PŘÍKLADY PROGRAMOVÁNÍ PLC TECOMAT

MODEL 16 BITŮ

6. vydání - listopad 2003

OBSAH

ÚVOD	4
1. INSTRUKCE PRO ČTENÍ A ZÁPIS DAT	5
1.1. Čtení a zápis dat s přímým adresováním (LD, LDL, LDC, WR, WRC, PUT)	5
1.2. Zápis dat s alternací nejvyššího bitu (WRA)	7
2. LOGICKÉ INSTRUKCE.....	8
2.1. Logické instrukce s operandem (AND, ANC, OR, ORC, XOR, XOC)	8
2.2. Logické instrukce bez operandu (AND, ANL, OR, ORL, XOR, XOL, NEG, NGL)...	11
2.3. Instrukce SET, RES, LET, BET	14
2.4. Instrukce FLG	16
2.5. Posuvy a rotace hodnoty (ROL, ROR, SWP, SWL, MUL, DIV, ADX)	20
2.6. Sklopení zásobníku (STK)	28
3. ČÍTAČE, POSUVNÉ REGISTRY, ČASOVAČE, KROKOVÝ ŘADIČ	30
3.1. Čítače (CTU, CTD, CNT)	31
3.2. Posuvné registry (SFL, SFR)	33
3.3. Časovače (TON, TOF, RTO, IMP)	36
3.4. Krokový řadič (STE)	41
4. ARITMETICKÉ INSTRUKCE	54
4.1. Sčítání a odčítání (ADX, ADD, ADL, SUX, SUB, SUL, INR, DCR)	55
4.2. Násobení a dělení (MUL, MUD, DIV, DID)	59
4.3. Porovnání hodnot (CMP, CML, EQ, LT, GT)	64
4.4. Použití znaménka a dvojkového doplňku	71
4.5. Příklady výpočtů v pevné řádové čárce	74
4.6. Převody číselných soustav (BIN, BIL, BCD, BCL)	76
5. OPERACE SE ZÁSObNÍKY	78
5.1. Posun zásobníku (POP)	78
5.2. Operace s několika zásobníky (NXT, PRV, CHG, CHGS, LAC, WAC)	78
6. INSTRUKCE SKOKŮ A VOLÁNÍ.....	81
6.1. Instrukce skoku (JMP, JMD, JMC, JMI, JZ, JNZ, JC, JNC, JS, JNS, L)	81
6.2. Realizace podprogramů (CAL, CAD, CAC, CAI, RET, RED, REC, L)	85
7. ORGANIZAČNÍ INSTRUKCE.....	88
7.1. Podmíněný konec procesu (EC, ED)	88
7.2. Podmíněné přerušení procesu (SEQ)	89

8. TABULKOVÉ INSTRUKCE	91
8.1. Čtení a zápis do tabulek T (LTB, WTB)	91
8.2. Čtení a zápis do tabulek v zápisníku (LTB, WTB)	106
8.3. Hledání hodnot v tabulkách (FTB, FTM).....	110
8.4. Třídění podle tabulek (FTS).....	130
9. BLOKOVÉ OPERACE	136
9.1. Přesuny bloků dat (SRC, MOV).....	136
9.2. Kopírování tabulky do zápisníku a naopak (MTN, MNT)	137
9.3. Plnění zápisníku konstantou (FIL)	141
10. OPERACE SE STRUKTUROVANÝMI TABULKAMI	142
10.1. Čtení a zápis do strukturované tabulky (LDS, WRS, FIS, FIT)	142
10.2. Hledání ve strukturovaných tabulkách (FNS, FNT)	145
11. ARITMETICKÉ INSTRUKCE V PLOVOUCÍ ŘÁDOVÉ ČÁRCE	147
11.1. Sčítání, odčítání, násobení a dělení (ADF, SUF, MUF, DIF)	147
11.2. Porovnání (CMF)	148
11.3. Matematické funkce (CEI, FLO, ABS, LOG, EXP, LN, POW, SQR, SIN, ASN, COS, ACS, TAN, ATN, HYP).....	151
11.4. Převody čísel mezi formáty (UWF, IWF, ULF, ILF, UFW, IFW, UFL, IFL)	152
12. INSTRUKCE REGULÁTORU PID	155
13. OPERACE SE ZNAKY ASCII	156
13.1. Operace se znaky ASCII (BAS, ASB, STF, FST)	156
14. SYSTÉMOVÉ INSTRUKCE	159
14.1. Ovládání odezvy komunikací (HPE, HPD).....	159
14.2. Čtení a zápis do obvodu reálného času (RDT, WRT).....	160
14.3. Práce s pamětí DataBox (IDB, RDB, WDB).....	165
REJSTŘÍKY	168
Rejstřík problémových okruhů v příkladech	168
Rejstřík direktiv překladače.....	170
Rejstřík systémových registrů	171
Rejstřík instrukcí PLC	172
SEZNAM PŘÍKLADŮ	176

ÚVOD

Tato příručka obsahuje příklady ilustrující použití jednotlivých instrukcí PLC TECOMAT a naznačující řešení některých dílčích problémů pro centrální jednotky se zásobníkem šířky 16 bitů.

Členění kapitol odpovídá kapitolám příručky Soubor instrukcí PLC TECOMAT - model 16 bitů (TXV 001 05.01) z důvodu snadnějšího vyhledávání podrobnějších informací o jednotlivých instrukcích, které nejsou v této příručce uváděny.

Informace o struktuře PLC TECOMAT, poskytovaných systémových službách a direktivách překladače xPRO jsou uvedeny v Příručce programátora PLC TECOMAT (TXV 001 09.01).

Instrukce, klíčová slova a direktivy použité v příkladech

Pokud není uvedeno jinak, využívají příklady instrukční soubor implementovaný v centrálních jednotkách řad B a D. Alternativy pro instrukční soubory centrálních jednotek řad E, M a S zde nejsou uváděny. Příklady programování centrálních jednotek se zásobníkem šířky 32 bitů obsahuje příručka Příklady programování PLC TECOMAT - model 32 bitů (TXV 004 04.01).

Operandy jsou převážně zapisovány pomocí symbolických jmen a jejich typy jsou určeny deklaracemi pomocí direktiv překladače (viz Příručka programátora PLC TECOMAT).

Absolutní adresy jsou psány s uvozujícím znakem %, který sice není při programování centrálních jednotek se zásobníkem šířky 16 bitů povinný, ale doporučuje se používat s ohledem na přenositelnost uživatelských programů do centrálních jednotek se zásobníkem šířky 32 bitů.

Totéž platí o způsobu psaní prefixů (*indx*, *bitpart*, *bitcnt*, *offset*, *sizeof*), které jsou psány s dvěma podtržítky na začátku (*__indx*, *__bitpart*, *__bitcnt*, *__offset*, *__sizeof*) a následující objekt je uzavřen do závorek. Tato opatření jsou nutná z důvodu zamezení kolize se symbolickými jmény vyššího jazyka.

Příklady v prostředí Mosaic

Všechny příklady uvedené v této příručce jsou součástí instalace vývojového prostředí Mosaic pro Windows dodávané s každým PLC, takže lze snadno zde uvedené algoritmy použít ve svém programu. Příklady se po instalaci prostředí Mosaic na pevný disk nacházejí v adresáři *C:\TecoApp\Priklady16*.

Příklady v této příručce jsou uvedeny ve formě mnemonického kódu použitelné v základní instalaci prostředí Mosaic. Na začátku každého příkladu je kurzivou uveden název souboru **.mos*, ve kterém je tento příklad uložen v adresáři *Priklady16*.

K zobrazení příkladu v reléovém liniovém schématu použijte prostředí Mosaic.

Rejstříky příkladů

K nalezení řešení nějakého konkrétního dílčího problému nebo použití konkrétní instrukce slouží rejstříky na konci příručky.

Rejstřík problémových okruhů v příkladech slouží k lepší orientaci v příkladech a ke snazšímu nalezení několika různých řešení jednoho problémového okruhu (např. sekvenční řadiče pomocí instrukce **STE**, pomocí tabulkových instrukcí, nebo pomocí instrukce **SEQ**). Rejstříky direktiv překladače a instrukcí PLC umožňují nalézt příklad, ve kterém je použita konkrétní direktiva nebo instrukce. Rejstřík systémových registrů umožňuje nalézt příklad, ve kterém je použit konkrétní systémový registr S.

1. ČTENÍ A ZÁPIS DAT

1.1. Čtení a zápis dat s přímým adresováním (LD, LDL, LDC, WR, WRC, PUT)

Instrukce s přímým adresováním nesou v operandu adresu paměti, ze které čtou, nebo do které zapisují. Adresa je tedy pevně daná uživatelským programem.

Vstupní instrukce čtou informace šířky bit, byte, word, long nebo float ze zdroje v přímé (**LD**, **LDL**) nebo negované (**LDC**) hodnotě a ukládají je na vrchol zásobníku předtím posunutého vpřed.

Výstupní instrukce ukládají přímou (**WR**) nebo negovanou (**WRC**) hodnotu zásobníku na uvedenou cílovou adresu šířky bit, byte, word, long nebo float. Instrukce **PUT** navíc podmiňuje zápis hodnotou příznaku S1.0 = log.1.

Příklad 1.1.1

Předpokládejme, že R4 = %01110110 (\$76, resp. 118), R5 = %10011000 (\$98, resp. 152), R6 = %00000110 (6) a R7 = %00011000 (\$18, resp. 24). Pak jednotlivé instrukce ukládají na vrchol zásobníku A0 následující hodnoty:

```
LD    %R4.1      ;A0 = %11111111 11111111 bit
LD    %R4.0      ;A0 = %00000000 00000000 bit
LDC   %R4.0      ;A0 = %11111111 11111111 bit
LD    %R4        ;A0 = %00000000 01110110 byte
LD    %R5        ;A0 = %00000000 10011000 byte
LD    %RW4       ;A0 = %10011000 01110110 word
LDC   %RW6       ;A0 = %11100111 11111001 word
LD    %RL4       ;A0 = %10011000 01110110 long
                ;A1 = %00011000 00000110
LDC   %RL4       ;A0 = %01100111 10001001 long
                ;A1 = %11100111 11111001
```

V praxi ale používáme symbolické vyjádření registrů. Formát je dán deklarací proměnné.

```
#reg bit    registr1
#reg byte   registr2
#reg word   registr3
#reg long   registr4
#reg float  registr5
;
P 0
    LD    registr1      ;bit
    LDC   registr2      ;byte
    LD    registr3      ;word
    LDC   registr4      ;long
    LD    registr5      ;float
E 0
```

Konstanty zapisujeme způsobem popsaným v Příručce programátora. Nezapomínejme, že zápis konstanty ve formátu float vždy vyžaduje desetinnou tečku!

```
LD    $3456      ;A0 = %00110100 01010110 word
LDC   $3456      ;A0 = %11001011 10101001 word
LDL   $123456    ;A0 = %00110100 01010110 long
                ;A1 = %00000000 00010010
```

1. Čtení a zápis dat

```
LDL 1      ;A0 = %00000000 00000001 long
          ;A1 = %00000000 00000000
LDL 1.0    ;A0 = %00000000 00000000 float
          ;A1 = %00111111 10000000
```

Příklad 1.1.2

Předpokládejme stav vrcholu zásobníku A0 = %01100111 10001001 (\$6789), A1 = %00100011 01000101 (\$2345). Pak jednotlivé instrukce zapisují na adresovaná místa následující hodnoty:

```
WRC %Y1.2   ;Y1.2 = log.0           bit
WR  %Y1.3   ;Y1.3 = log.1           bit
WR  %R5      ;R5 = %10001001         byte
WRC %R5      ;R5 = %01110110         byte
WR  %YW2     ;Y3 = %01100111, Y2 = %10001001 word
WRC %YW2     ;Y3 = %10011000, Y2 = %01110110 word
WR  %RL0     ;R1 = %01100111, R0 = %10001001, long
          ;R3 = %00100011, R2 = %01000101
WRC %RL3     ;R4 = %10011000, R3 = %01110110, long
          ;R6 = %11011100, R5 = %10111010
```

Symbolický zápis je analogický podle předchozího příkladu.

Příklad 1.1.3

Požadujeme, aby vstupní proměnná *vstup* zůstávala v PLC po dobu 1 s beze změny.

Pomocí sekundového příznaku S20.2 budeme zapisovat *vstup* do proměnné *hodnota* jednou za sekundu.

```
B01103.mos
#reg byte vstup, hodnota
;
P 0
    LD  %S20.2   ;příznak 1 s
    WR  %S1.0    ;podmínka
    LD  vstup    ;vstupní hodnota
    PUT hodnota  ;aktualizace
E 0
```

Příklad 1.1.4

Realizujeme převodník kódu 2 z 5 na dvojkové číslo. Výsledek zapisujeme do proměnné *cislo*. Pokud bude zadána kombinace, která nemá smysl, nechme v proměnné *cislo* původní hodnotu.

```
B01104.mos
#reg byte cislo,kod
;
#table byte tab = %00110,%10001,%01001,%11000,%00101,
                  %10100,%01100,%00011,%10010,%01010
;
P 0
    LD  kod      ;kombinace 2 z 5
    FTB tab
    PUT cislo    ;zápis, pokud bylo hledání úspěšné
E 0
```

1.2. Zápis dat s alternací nejvyššího bitu (WRA)

Instrukce **WRA** provádí zápis dat s alternací nejvyššího bitu. Tato funkce nachází uplatnění při řízení inteligentních periferních modulů, které nejsou ovládány stavem nějaké trvale zapisované proměnné, ale jednorázovým zápisem příkazového kódu. Protože se zápis dat do všech periferních modulů z centrální jednotky provádí v každém cyklu, je třeba pomocí příznaku modulu sdělit, že tento příkaz je nový. K tomu se obvykle používá nejvyšší bit, který při změně kódu vždy změní hodnotu (tzv. alternace). Tento princip umožňuje zápis dvou stejných příkazových kódů za sebou (lišit se mohou až data předávaná spolu s příkazem).

Příklad 1.2.1

Zapišme do výstupního registru příkazový kód s alternací nejvyššího bitu.

```
B01201.mos
#def kod1  $21          ;kód 1. příkazu
#def kod2  $22          ;kód 2. příkazu
...
#reg byte  cont, datas[10]
;
P 0
    LD      kod1          ;nový příkaz
    WRA     cont          ;zápis do řídicího bytu modulu
                        ;s alternací nejvyššího bitu
E 0
```

2. LOGICKÉ INSTRUKCE

2.1. Logické instrukce s operandem (AND, ANC, OR, ORC, XOR, XOC)

Logické instrukce s operandem **AND**, **ANC**, **OR**, **ORC**, **XOR**, **XOC** zapisují na vrchol zásobníku výsledek logické operace šířky bit, byte, word.

Příklad 2.1.1

Realizujeme logický výraz $a = b + c$

```
B02101.mos
#reg bit  va, vb, vc
;
P 0
    LD    vb
    OR    vc
    WR    va
E 0
```

Příklad 2.1.2

Realizujeme osmici logických výrazů $a.0 = b.0 \cdot c.0$, $a.1 = b.1 \cdot c.1$, ..., $a.7 = b.7 \cdot c.7$

```
B02102.mos
#reg byte  va, vb, vc
;
P 0
    LD    vb
    AND   vc
    WR    va
E 0
```

Příklad 2.1.3

Porovnejme obsahy a a b . V c budou příznaky neshody, tj. na místě, kde nedošlo ke shodě obsahů, je jednička.

```
B02103.mos
#reg byte  va, vb, vc
;
P 0
    LD    va
    XOR   vb
    WR    vc
E 0
```

Příklad 2.1.4

Doplňme příklad 2.1.3 o celkový výsledek porovnání v bitové proměnné d .

```
B02104.mos
#reg byte  va, vb, vc
#reg bit   vd
;
```



```

P 0
    LD    va
    XOR   vb
    WR    vd          ;log.0, je-li A0 = 0 (obsahy shodné), jinak log.1
    WR    vc
E 0

```

Příklad 2.1.5

Realizujeme osmici výrazů $a.0 = \bar{b} + c.0$, $a.1 = \bar{b} + c.1$, ..., $a.7 = \bar{b} + c.7$

```

B02105.mos
#reg byte  va, vc
#reg bit   vb
;
P 0
    LD    vc
    ORC   vb
    WR    va
E 0

```

Příklad 2.1.6

Realizujeme logický výraz $a = \overline{\bar{b} \cdot (c.0 + c.1 + c.2 + \dots + c.7)}$

```

B02106.mos
#reg bit   va, vb
#reg byte  vc
;
P 0
    LD    vc
    ANC   vb
    WRC   va
E 0

```

Příklad 2.1.7

Provedme přesun dolních tří bitů proměnné a do proměnné b , vyšší bity v b budou nulové.

```

B02107.mos
#reg byte  va, vb
;
P 0
    LD    va
    AND   7
    WR    vb
E 0

```

Příklad 2.1.8

Provedme přesun horních čtyř bitů proměnné a do proměnné b , dolní čtyři bity v b budou jedničkové.

```

B02108.mos
#reg byte  va, vb
;
P 0
    LD    va

```

2. Logické instrukce

```
OR    $0F          ;totožné s OR 15 nebo OR %1111
WR    vb
E 0
```

Příklad 2.1.9

Proveďme přesun bitů z proměnné a do proměnné b tak, že dolních pět bitů bude negovaných a horní tři ne.

Platí:

$$a \oplus 1 = \bar{a}$$

$$a \oplus 0 = a$$

kde \oplus je symbol součtu mod 2 (pro dvě proměnné je shodný s funkcí XOR).

```
B02109.mos
#reg byte va, vb
;
P 0
LD    va
XOR   $1F          ;totožné s XOR 31 nebo XOR %11111
WR    vb
E 0
```

Příklad 2.1.10

Kombinací předchozích postupů provedme operaci tak, aby jednotlivé bity proměnné a vypadaly takto:

$$\begin{array}{llll} a.0 = 1 & a.1 = 0 & a.2 = 0 & a.3 = \bar{b.3} \\ a.4 = b.4 & a.5 = b.5 & a.6 = b.6 & a.7 = b.7 \end{array}$$

```
B02110.mos
#reg byte va, vb
;
P 0
LD    vb
OR    %111          ;nebo AND %11111000
XOR   %1110         ;      XOR %1001
WR    va
E 0
```

Příklad 2.1.11

Přesuňme do proměnné a výsledky následujících výrazů:

$$\begin{array}{ll} a.0 = b.0 + c.0 & \dots & a.7 = b.7 + c.7 \\ a.8 = b.8 & \dots & a.11 = b.11 \\ a.12 = \overline{b.12} & \dots & a.15 = \overline{b.15} \end{array}$$

```
B02111.mos
#reg word va, vb
#reg byte vc
;
P 0
LD    vb
OR    vc
XOR   $F000
WR    va
E 0
```

Příklad 2.1.12

Realizujeme čítač událostí v rozsahu 0 - 131 071 (\$1FFFF). Po dosažení maxima bude čítač vynulován a bude čítat opět od začátku.

Protože rozsah čítače je hodnota 2^{17} , tedy číslo šíře 17 bitů, můžeme použít k zacyklení čítače instrukci **ANL** k zamaskování nepotřebných horních 15 bitů. Tento příklad je obecný pro meze čítače 2^k , kde k je 17 až 31 (pro k = 1 až 15 použijeme místo instrukce **ANL** instrukci **AND** pro formát word).

```

B02112.mos
#reg long citac
#reg bit  udalost
;
P 0
    LD    udalost
    JMC    nic          ;čítáme počet výskytu udalost = log.1
    INR    citac
    LD    citac
    ANL    $1FFFF      ;zacyklení čítače
    WR    citac
nic:
E 0

```

2.2. Logické instrukce bez operandu (AND, ANC, ANL, ANLC, OR, ORC, ORL, ORLC, XOR, XOC, XOL, XOLC, NEG, NGL)

Logické instrukce se zásobníkem (bezoperandové) **AND**, **ANL**, **OR**, **ORL**, **XOR**, **XOL**, **NEG**, **NGL** pracují s šířkou word nebo long.

Příklad 2.2.1

Realizujeme logický výraz $a = (b + c) \cdot (d + e)$

```

B02201.mos
#reg bit  va, vb, vc, vd, ve
;
P 0
    LD    vb
    OR    vc          ;(b+c)
    LD    vd
    OR    ve          ;(d+e)
    AND          ;().()
    WR    va
E 0

```

Příklad 2.2.2

Realizujeme osmici logických výrazů $a.0 = (b.0 + c.0) \cdot (d.0 + e.0)$,
 $a.1 = (b.1 + c.1) \cdot (d.1 + e.1)$, ..., $a.7 = (b.7 + c.7) \cdot (d.7 + e.7)$

```

B02202.mos
#reg byte va, vb, vc, vd, ve
;
P 0
    LD    vb
    OR    vc          ;(b+c)

```

```
LD    vd
OR     ve          ;(d+e)
AND                ;().()
WR     va
```

E 0

Příklad 2.2.3

Realizujeme šestnáct logických výrazů $a.0 = (b.0 + c.0) \cdot (d.0 + e.0)$,
 $a.1 = (b.1 + c.1) \cdot (d.1 + e.1)$, ..., $a.15 = (b.15 + c.15) \cdot (d.15 + e.15)$

B02203.mos

```
#reg word  va, vb, vc, vd, ve
```

```
;
```

P 0

```
LD     vb
OR     vc          ;(b+c)
LD     vd
OR     ve          ;(d+e)
AND                ;().()
WR     va
```

E 0

Příklad 2.2.4

Porovnejme obsahy a a b . V c budou příznaky neshody, tj. na místě, kde nedošlo ke shodě obsahů, je jednička.

B02204.mos

```
#reg long  va, vb, vc
```

```
;
```

P 0

```
LD     va
LD     vb
XOL
WR     vc
```

E 0

Příklad 2.2.5

Typickým použitím instrukce **NEG** je patrně negování mezivýsledku při realizaci závorkovaných výrazů. Požadujeme například realizovat bytovou logickou operaci

$$a = \overline{b + c + b + c}.$$

B02205.mos

```
#reg byte va, vb, vc
```

```
;
```

P 0

```
LDC    vb
OR      vc
NEG
LD      vb
ORC     vc
NEG
OR
WR      va
```

E 0

Poznámka: V tomto ukázkovém příkladu realizujeme fakticky operaci $a = b \oplus c$, což lze provést následujícím postupem.

```
P 0
    LD    vb
    XOR   vc
    WR    va
E 0
```

Příklad 2.2.6

Nastavme byte a shodně s b , byte c inverzně.

```
B02206.mos
#reg byte va, vb, vc
;
P 0                                nebo                                P 0
    LD    vb                      LD    vb
    WR    va                      WR    va
    NEG                                       WRC   vc
    WR    vc
E 0                                E 0
```

Příklad 2.2.7

Provedme logickou operaci $a = \overline{b \cdot c} + d$ pro skupiny 32 proměnných (long).

```
B02207.mos
#reg long va, vb, vc, vd
;
P 0
    LD    vb
    LD    vc
    ANL                                ;b.c
    NGL                                ;negace
    LD    vd
    ORL                                ;b.c + d
    WR    va
E 0
```

Příklad 2.2.8

Požadujeme, aby na bitu a byl uložen podélný logický součet (OR) bitů $b.0$ až $b.7$ a na bitu c negace tohoto součtu. Možný je pouze následující postup.

```
B02208.mos
#reg bit va, vc
#reg byte vb
;
P 0
    LD    vb
    WR    va
    WRC   vc
E 0
```

Následující postupy jsou nesprávné:

```
P 0
    LD    vb
    WR    va
```

```

    NEG
    WR    vc
E 0

```

Tento postup nastaví vždy jedničkový obsah bitu *c* (původně nulový obsah vrcholu zásobníku A0 se negací změnil na \$FFFF).

```

P 0
    LD    vb
    WR    va
    XOR    %1111111111111111
    WR    vc
E 0

```

Tento postup nenastavuje bit *c* na požadovanou funkci negace logického součtu (NOR), ale na funkci součtu negovaných proměnných, tedy NAND:

$$c = \overline{b.0} + \overline{b.1} + \dots + \overline{b.7} = \overline{b.0 \cdot b.1 \cdot \dots \cdot b.7}$$

Této skutečnosti můžeme výhodně využít.

Příklad 2.2.9

Požadujeme, aby bit *a* byl nastaven na hodnotu podélného součinu všech bitů proměnné *b* formátu word.

$$a = \overline{b.0} + \overline{b.1} + \dots + \overline{b.15} = \overline{b.0 \cdot b.1 \cdot \dots \cdot b.15}$$

```

B02209.mos
#reg bit    va
#reg word   vb
;
P 0
    LDC    vb
    WRC    va
E 0

```

2.3. Instrukce SET, RES, LET, BET

Logické instrukce s cílovým místem **SET**, **RES**, **LET**, **BET** pracují ve formátu bit, byte, word. Instrukce **SET** a **RES** realizují pro adresované místo paměťovou funkci. Lze je však použít k nepodmíněnému vynulování, k nastavení obsahu nebo k realizaci logického součtu nebo součinu adresovaného místa s vrcholem zásobníku. **LET** generuje na vrcholu zásobníku impuls od náběžné hrany zapisované proměnné, **BET** generuje impuls od obou hran.

Příklad 2.3.1

Předpokládejme, že na vstup *rozepnuto* je přiveden rozpínací (klidový) kontakt a na vstup *sepnuto* spínací (pracovní) kontakt spínače. Předpokládá se, že spínač bude využíván k čítání počtu sepnutí a je tedy žádoucí, aby program byl necitlivý k zákmitům tohoto kontaktu.

```

B02301.mos
#reg bit    spinac, rozepnuto, sepnuto
;
P 0
    LD      rozepnuto
    RES     spinac
                                nebo
P 0
    LD      sepnuto
    SET     spinac

```

F 0 **F 0**

```
#reg byte spinace, klid, prac
```

i

nebo obecně dvou různých událostí.

B02303.mos

B02304.mos

```
RES    cislo
E 0
```

Příklad 2.3.5

Spojme příklady 2.3.3 a 2.3.4 tak, že budeme nulovat proměnnou *cislo* při obou hranách téhož vstupního signálu.

```
B02305.mos
#reg word cislo
#reg bit  vstup
#reg bit  stav                ;stavová proměnná
;
P 0
    LD    vstup
    BET   stav
    RES   cislo
E 0
```

Příklad 2.3.6

Změňme zadání předchozích příkladů tak, že budeme požadovat, aby proměnná *cislo* byla vynulována při náběžné hraně kteréhokoliv vstupu *vstup.0* až *vstup.3* nebo při závěrné hraně na některém ze vstupů *vstup.4* až *vstup.7*. Mohli bychom postupně vytvářet impulzy hran pro jednotlivé vstupy. S použitím bytových instrukcí však můžeme celou operaci provést naráz.

```
B02306.mos
#reg byte vstup
#reg word cislo
#reg byte stav                ;stavové proměnné
#reg bit  pomoc              ;pomocná proměnná
;
P 0
    LD    vstup
    XOR   $F0
    LET   stav
    WR    pomoc
    LD    pomoc
    RES   cislo
E 0
```

Instrukcí **XOR \$F0** negujeme hodnoty ze vstupů *vstup.4* až *vstup.7*, bytová instrukce **LET stav** realizuje hrany všech osmi proměnných. Bitové instrukce **WR pomoc** a **LD pomoc** jsou pomocné a slouží pouze k logickému sečtení všech hran - lze je nahradit podmíněným přeskokem.

2.4. Instrukce FLG

Instrukce **FLG** provádí soubor logických funkcí nad vrcholem zásobníku.

Příklad 2.4.1

Předpokládejme, že na *vstupy* jsou připojeny fotobuňky, počítající součástky na osmi postech. Chceme znát celkový počet součástek.


```

B02401.mos
#reg byte vstup
#reg byte stav
#reg long pocet
;
P 0
    LD    vstup
    LET    stav
    FLG
    LD    %S1
    AND    %1111
    ADX    pocet
    WR    pocet
E 0

```

Po instrukci **AND** %1111 je na vrcholu zásobníku počet náběžných hran vstupů. Pokud nás zajímá, zda je toto číslo liché, stačí využít informaci nejnižšího bitu této hodnoty (bit 0 vrcholu zásobníku nebo příznak S1.0). Pokud chceme zjistit, zda na všech vstupech byly náběžné hrany, stačí otestovat, zda bit S1.3 je roven jedné. Je-li třeba prověřit, zda právě na jednom vstupu byla náběžná hrana, stačí porovnat vrchol zásobníku po instrukci **AND** %1111 s hodnotou 1.

Příklad 2.4.2

Máme zadáno realizovat čítač s osmi vstupy *vstup.0* až *vstup.7*. Každá náběžná hrana na *vstup.0* až *vstup.3* zvýší jeho obsah o jedničku a každá sestupná hrana na *vstup.4* až *vstup.7* zmenší jeho obsah o jedničku.

```

B02402.mos
#reg byte vstup
#reg byte stavn, stavs
#reg word citac
;
P 0
    LD    vstup
    AND    %1111        ;vstup.0 - .3
    LET    stavn        ;náběžné hrany
    FLG
    LD    %S1
    AND    %1111
    ADD    citac        ;přičíst obsah čítače
    LDC    vstup        ;posune zásobník vpřed
    AND    %11110000    ;vstup.4 - .7
    LET    stavs        ;sestupné hrany
    FLG
    POP    2            ;posune zásobník vpřed
    LD    %S1
    AND    %1111
    SUB                 ;odečíst od obsahu čítače
    WR    citac        ;uložit čítač
E 0
;
P 63
    LD    $F0
    WR    stavs        ;ošetření prvního cyklu
E 63

```

Porovnáním s tradičními postupy vynikne účinnost tohoto postupu. Je nutno upozornit, že nelze použít postup, při kterém bychom nejprve sečetli vstupy a jejich negace s výsledky pak použili jako řídicí proměnné instrukce čítání **CNT**. Pokud nelze zaručit výskyt nejvýše jedné hrany v jednom cyklu, nelze na vstup čítače přivést ani logicky posčítané impulzy od hran. Při tradičním postupu bychom museli programovat instrukci čítače zvlášť pro každý vstup.

Příklad 2.4.3

Předpokládejme, že pro zabezpečení zvýšené spolehlivosti vstupní strany systému je každý vstupní signál realizován s trojnásobnou nadbytečností. Například na vstupech *vstup.0*, *vstup.1*, *vstup.2* by za normálního stavu měly být shodné hodnoty. Při chybě požadujeme většinové rozhodovací pravidlo a výsledná informace má být uložena v bitu *hodnota*.

```
B02403a.mos
#reg byte vstup
#reg bit  hodnota
;
P 0
    LD    vstup
    AND   7          ;rovnocenné s AND %111
    FLG
    LD    %S1.1
    WR    hodnota
E 0
```

Pokud je (po oddělení sledovaných vstupů) počet jedničkových hodnot 2 a 3 (většina), je jedničkový bit S1.1 po instrukci **FLG**. Pokud je nulový, pak byl jedničkový pouze jeden nebo žádný vstup (většina vstupů byla nulová). Tato hodnota je tedy jako výsledek nastavena do bitu *hodnota*.

Pokud není technicky možné vstupní signály přivést na vstupy stejného bytu, lze předcházející postup upravit takto:

```
B02403b.mos
#reg bit  vstup0, vstup1, vstup2
#reg bit  hodnota
;
P 0
    LD    vstup0      ;na pořadí nezáleží
    LD    vstup1
    LD    vstup2
    STK           ;sklopení načtených bitů do bytu
    AND   7
    FLG
    LD    %S1.1
    WR    hodnota
E 0
```

Příklad 2.4.4

Předchozí příklad chceme rozšířit o požadavek, aby bit *rozdil* byl jedničkový, pokud jeden ze vstupů má jinou hodnotu než ostatní („byl přehlasován“).

```

B02404.mos
#reg byte vstup
#reg bit  hodnota, rozdíl
;
P 0
    LD    vstup
    AND   7
    FLG
    LD    %S1.1
    WR    hodnota
    LD    %S1.0
    XOR   %S1.1
    WR    rozdíl
E 0

```

V bezporuchovém stavu musí být počet jedniček 3 nebo 0, dvojkově 11 nebo 00. Dolní dva bity S1 (po první instrukci **FLG**) musí být tedy navzájem shodné.

Příklad 2.4.5

Vytvoříme součin bitů *vstup.0* až *vstup.7* a součin bitů *vstup.8* až *vstup.15*. Jejich logický součet uložíme do bitu *vysledek*.

```

B02405.mos
#reg word vstup
#reg bit  vysledek
;
P 0                                nebo                                P 0
    LD    vstup                    LD    vstup
    FLG                                     FLG
    LD    %S1                        LD    %S1.4
    AND   %110000                    OR    %S1.5
    WR    vysledek                    WR    vysledek
E 0                                E 0

```

Příklad 2.4.6

Je požadováno vytvořit výraz $a = b.7 \cdot b.5 \cdot b.4 \cdot b.1 \cdot b.0 + b.14 \cdot b.11 \cdot b.9$

Namísto tradičního postupu lze upravit postup z předchozího příkladu.

```

B02406.mos
#reg word vb
#reg bit  va
;
P 0
    LD    vb
    OR    %1011010101001100      ;masky bitů b
    FLG
    LD    %S1.4
    OR    %S1.5
    WR    va
E 0

```

Instrukce **OR** oddělí nevýznamné bity (zapiše na jejich pozice jedničky) a další postup je shodný s předchozím příkladem.

Příklad 2.4.7

Zkomplikujme si zadání a požadujeme výraz

$$a = \overline{b.7} \cdot \overline{b.5} \cdot \overline{b.4} \cdot \overline{b.1} \cdot \overline{b.0} + b.14 \cdot b.11 \cdot b.9$$

Postup z předchozího příkladu stačí pouze upravit.

```
B02407.mos
#reg word vb
#reg bit va
;
P 0
    LD    vb
    OR    %1011010101001100    ;masky bitů b
    XOR    %0000100010100001    ;negace bitů
    FLG
    LD    %S1.4
    OR    %S1.5
    WR    va
E 0
```

Druhá maska je volena tak, že jedničky jsou na pozicích bitů, které jsou ve výrazu negovány.

Příklad 2.4.8

Požadujeme vytvoření součinu všech bitů proměnné *vstup* typu word, přičemž bity 13, 12, 11, 9, 7, 6, 3, 2 a 1 v něm budou negované.

```
B02408.mos
#reg word vstup
#reg bit vysledek
;
P 0
    LD    vstup
    XOR    %0011101011001110    ;negace bitů
    FLG
    WR    vysledek
E 0
```

Tento postup je jednoznačně úspornější než tradiční.

2.5. Posuvy a rotace hodnoty (ROL, ROR, SWP, SWL, MUL, DIV, ADX)

K realizaci posuvů a rotací máme k dispozici několik prostředků:

- instrukce **ROL** a **ROR**
- instrukce **SWP** a **SWL** (záměna bytů či wordů vrcholu zásobníku, tj. rotace o 8, resp. 16 bitů)
- instrukce **MUL**, **DIV**
- sečtení shodných obsahů (**ADX**)

Příklad 2.5.1

Realizujte posuvný registr v šíři 32 bitů (long) nad registrem *posuv*. Použijeme instrukci **ADX**.

```
B02501.mos
#reg long posuv
;
P 0
    LD    posuv
    ADX   posuv
    WR    posuv
E 0
```

Příklad 2.5.2

Realizujeme v registru *citac* čtyřstupňový čítač v Johnsonově kódu. Johnsonův kód má pro 4 stavy periodu 8:

```
0000←
0001
0011
0111
1111
1110
1100
1000—
```

Nejčastěji se používá ke generování vícefázových periodických průběhů. Nejsnáze se realizuje jako kruhový posuvný registr, který má ve zpětné vazbě vřazenu negaci a vychází z nulového počátečního stavu.

Nejprve budeme realizovat tradičním postupem zkrácení cyklu posuvného registru na 4 stupně.

```
B02502a.mos
#reg byte citac
;
P 0
    LD    citac
    ROL   1           ;nebo MUL 2
    WR    citac
    AND   %10000
    WRC   citac.0     ;rotace
E 0
```

Stejného výsledku však dosáhneme s osmistupňovým registrem bez negace ve zpětné vazbě a s počátečním obsahem %11110000.

```
B02502b.mos
#reg byte citac
;
P 0
    LD    citac
    ROL   1
    WR    citac
    SWP
    SET   citac
E 0
```

Můžeme použít i šestnáctistupňový kruhový registr s počátečním obsahem %11110000 11110000. Pak se program zjednoduší.

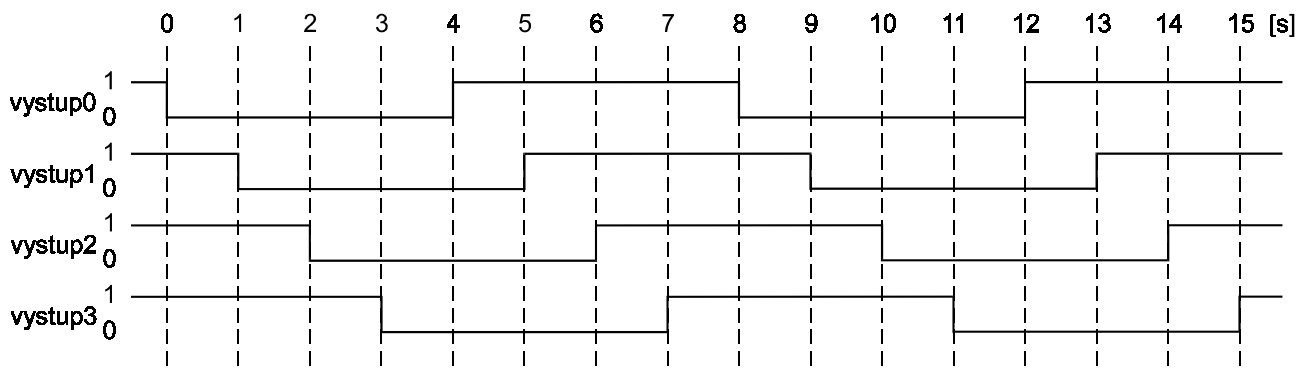
2. Logické instrukce

```
B02502c.mos
#reg word citac
;
P 0
    LD    citac
    ROL   1
    WR    citac
E 0
```

Prvé řešení (zkrácení cyklu) je obecné a lze jej použít pro libovolnou délku kruhového registru (do 16 stupňů). Druhá dvě řešení jsou použitelná pouze v případech, kdy počet stupňů je dvojkovou mocninou.

Příklad 2.5.3

Na výstupech *vystup0* až *vystup3* realizujte čtyřfázový časový průběh:



Pokud nechceme použít podmíněný přeskok, použijeme postup s násobením:

```
B02503a.mos
#reg byte vystupy      ;bity 0 až 3 .. vystup0, vystup1, vystup2, vystup3
#reg byte pomoc
;
P 0
    LD    %S20.2        ;náběžná hrana 1 s
    WR    %S1.0          ;příznak pro podmíněný zápis
    AND   2
    MUL   pomoc
    PUT   pomoc          ;podmíněný zápis
    SWP
    SET   pomoc
    LD    vystupy
    AND   %11110000
    LD    pomoc
    AND   %1111
    OR
    WR    vystupy
E 0
;
P 63
    LD    %11110000
    WR    pomoc          ;inicializace pomocného registru
E 63
```

Prvé dvě instrukce promění impuls od časové jednotky 1 s na číslo 0 nebo 2, kterým násobíme pracovní obsah registru *pomoc* (obdobně jako v předchozím příkladu) a výsledek pak přesuneme na výstupy. Počáteční obsah registru *pomoc* je přednastaven na %11110000 (zde v rámci restartu).

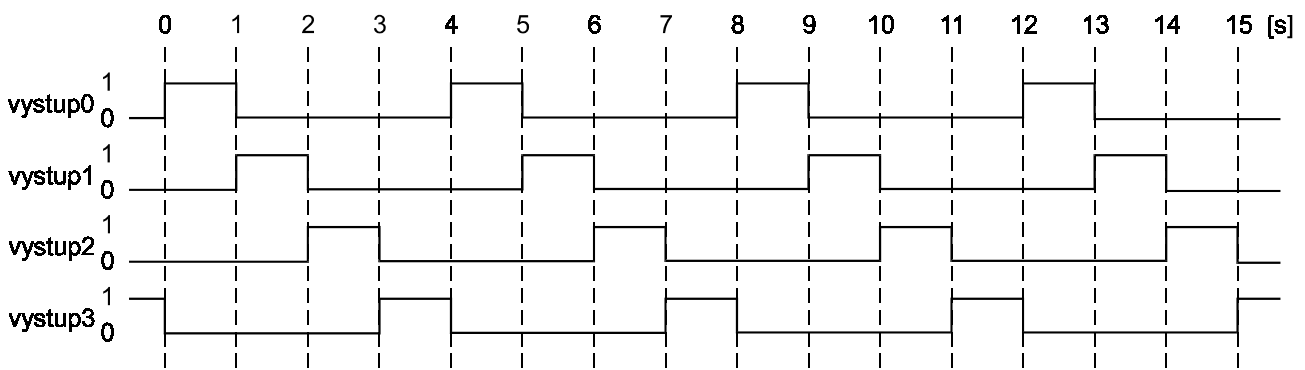
S využitím instrukcí **SET** a **RES** lze přesun na výstupy provést úsporněji:

B02503b.mos

```
#reg byte vystupy      ;bity 0 až 3 .. vystup0, vystup1, vystup2, vystup3
#reg byte pomoc
;
P 0
    LD    %S20.2
    WR    %S1.0      ;příznak pro podmíněný zápis
    AND   %1111
    RES   vystupy
    AND   2
    MUL   pomoc
    PUT   pomoc      ;podmíněný zápis
    SWP
    OR    pomoc
    PUT   pomoc      ;podmíněný zápis
    AND   %1111
    SET   vystupy
E 0
;
P 63
    LD    %11110000
    WR    pomoc      ;inicializace pomocného registru
E 63
```

Příklad 2.5.4

Na výstupech *vystup0* až *vystup3* realizujte čtyřfázový časový průběh:

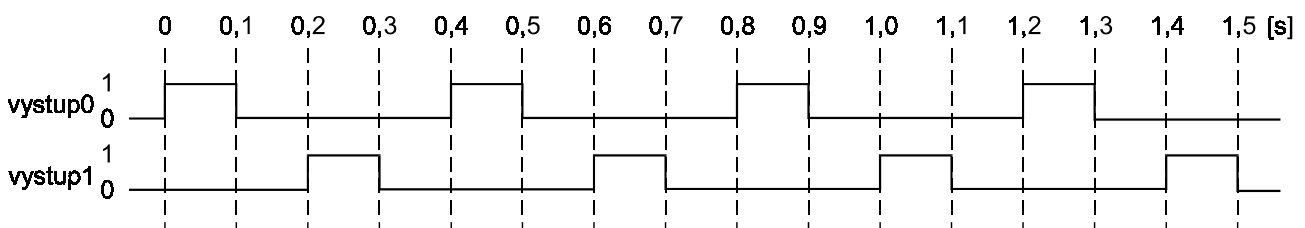


Můžeme použít přesně stejný postup, jako v předchozím příkladu. Pouze musíme zajistit, aby počáteční obsah registru *pomoc* byl %00010001.

Popsaným postupem lze realizovat libovolné průběhy dvou - čtyř nebo osmifázových výstupních nebo vnitřních proměnných.

Příklad 2.5.5

Na výstupech *vystup0* a *vystup1* realizujte dvoufázový průběh:



Požadované průběhy odpovídají první a třetí fázi z předchozího příkladu, můžeme postupovat obdobně.

B02505.mos

#reg bit vystup0, vystup1

#reg byte pomoc

;

P 0

```
LD    %S20.0    ;impulz 100 ms
WR    %S1.0      ;příznak pro podmíněný zápis
AND    2
MUL    pomoc
PUT    pomoc      ;podmíněný zápis
SWP
SET    pomoc
LD    pomoc.0
WR    vystup0
LD    pomoc.2
WR    vystup1
```

E 0

;

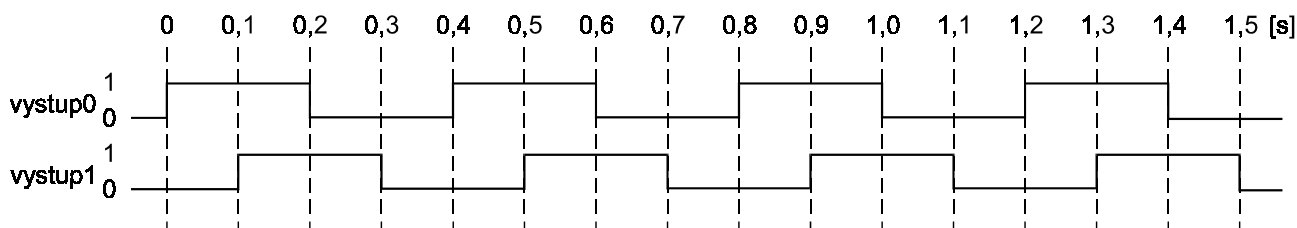
P 63

```
LD    %00010001
WR    pomoc      ;inicializace pomocného registru
```

E 63

Příklad 2.5.6

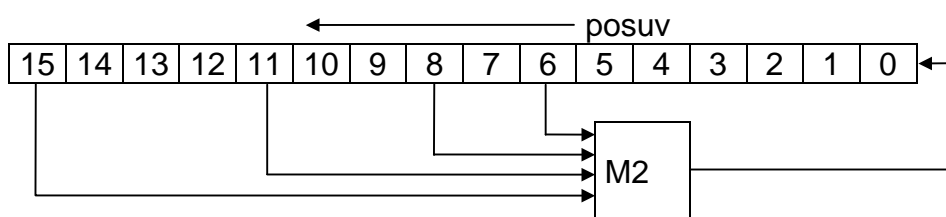
Na výstupech *vystup0* a *vystup1* realizujte dvoufázový průběh:



Úlohu řeší program z příkladu 2.5.5, ale s počátečním obsahem registru *pomoc* = %11001100. Generování vícefázových průběhů je účelné zejména pro ovládání periodických procesů (řízení toku materiálu a postupu operací, světelné a zvukové efekty, „vlnivý pohyb“ při přepravě materiálu), k synchronizaci signálů a procesů s cílem vyloučit hazardní stavy a zvýšit spolehlivost, případně v některých případech rozlišení směru, rozpoznání polohy nebo stavu, rozpoznání chybových stavů. Posouvání je nejčastěji odvozeno od počtu cyklů programu, od časových jednotek nebo od určených signálů nebo událostí.

Příklad 2.5.7

Zajistěte, aby hodnota bitu *vystup* se jevila jako náhodná. Je více způsobů generování náhodných proměnných nebo čísel. Patrně nejjednodušší realizaci umožňují pseudonáhodné generátory se zpětnovazebními registry.



Znázorněný zpětnovazební registr je používán v příznakových analyzátoch (symbol M2 je funkce mod 2 = lichá parita).

Obsah kteréhokoliv stupně registru lze považovat za pseudonáhodnou proměnnou - uvnitř cyklu 65 535 se jeví jako náhodná, opakuje se s periodou právě 65 535 stavů. Za pseudonáhodné číslo lze považovat i obsah celého registru nebo jeho části - pseudonáhodná posloupnost čísel má opět periodu 65 535 hodnot. Podmínkou však je, že počáteční stav registru nesmí být nulový.

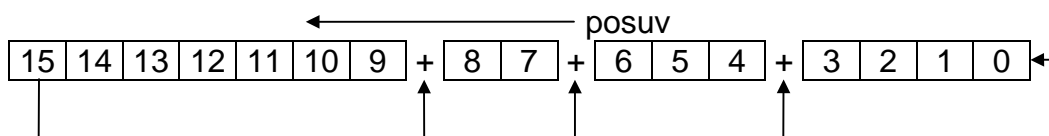
Úlohu lze řešit programem:

B02507a.mos

```
#reg bit   vystup
#reg word registr
;
P 0
    LD     registr
    ROL    1
    WR     registr
    LD     %0001001010000000    ;výběr bitů
    AND
    FLG
    LD     %S1.0
    XOR    registr.0
    WR     registr.0
    WR     vystup
E 0
;
P 63
    LD     1                    ;libovolná nenulová hodnota
    WR     registr
E 63
```

Po rotaci proměnné *registr* o jeden stupeň byly vybrány bity na pozicích, odkud se mají odebírat zpětné vazby (pozice určené jedničkami masky). Pozice jsou oproti schématu posunuty o jeden stupeň vlevo, protože testování provádíme až po posuvu. Vybrané hodnoty jsou sečteny mod 2 instrukcí **FLG** a výsledek je operací mod 2 přičten k nové hodnotě nejnižšího stupně (před posuvem hodnota nejvyššího stupně). Výsledná hodnota je nastavena na výstup.

Stejně statistické vlastnosti (ne však sled obsahů registru) má i posloupnost generovaná následujícím zpětnovazebním registrem.



Jeho struktuře odpovídá program:

B02507b.mos

```
#reg bit   vystup
#reg word registr
;
P 0
    LD     registr
    ROL    1
    LD     %S0.1
    WR     vystup
    LD     %0000001010010000
    AND
```

```
        XOR
        WR    registr
E 0
;
P 63
        LD    1                ;libovolná nenulová hodnota
        WR    registr
E 63
```

V obou případech jsou změny na výstupu synchronizovány s dobou cyklu uživatelského programu.

Příklad 2.5.8

Proveďme logickou operaci $a = b.0 \cdot b.4 + b.1 \cdot b.5 + b.2 \cdot b.6 + b.3 \cdot b.7$

Tradiční bitový postup nebudeme uvádět, obsahuje 12 instrukcí. Úlohu vyřešíme jednodušeji:

```
B02508.mos
#reg bit  va
#reg byte vb
;
P 0
        LD    vb
        MUL   16                ;posun o 4 bity vlevo
        AND   $FF00            ;vynulování zbytku
        SWP                   ;posun o 8 bitů vpravo
        AND   vb                ;4 součiny
        WR    va                ;součet
E 0
```

Posunem byla skupina horních bitů *b.4* až *b.7* vysunuta do dolní části A0H, instrukcí **SWP** překlopena do A0L. Logické součiny pak je možno provést paralelně nad stejno-
lehlými bity.

Příklad 2.5.9

Zaměňme navzájem dolní a horní polovinu bytu *cislo*, výsledek uložme do *vysledek*.

```
B02509a.mos
#reg byte cislo, vysledek
;
P 0
        LD    cislo
        MUL   16                ;posun o 4 bity vlevo
        WR    vysledek          ;odložení A0L
        SWP                   ;záměna A0L a A0H
        OR    vysledek          ;sestavení
        WR    vysledek          ;uložení
E 0
```

Nejprve jsme původní obsah rozdělili do dvou bytů A0, jednu část jsme si odložili do *vysledek*, druhou část překlopili a znovu složili.

S využitím instrukce **SET** lze postup ještě zjednodušit:

```
B02509b.mos
#reg byte cislo, vysledek
;
P 0
        LD    cislo
```

```

      MUL    16                ;posun o 4 bity vlevo
      WR     vysledek          ;odložení A0L
      SWP                    ;záměna A0L a A0H
      SET     vysledek          ;sestavení a uložení
E 0

```

Stejný postup pro obecnou mocninu dvou je rovnocenný rotaci obsahu bytu vlevo.

Příklad 2.5.10

Požadujeme přesun bytů *vstup0* do *vysledek0*, *vstup1* do *vysledek1*.

```

B02510a.mos
#reg byte vstup0, vstup1
#reg byte vysledek0, vysledek1
;
P 0
      LD     vstup0
      WR     vysledek0
      LD     vstup1
      WR     vysledek1
E 0

```

Předcházející klasický postup lze při stejném rozmístění vstupů nahradit:

```

B02510b.mos
#reg word vstup
#reg byte vysledek0, vysledek1
;
P 0
      LD     vstup
      WR     vysledek0
      SWP
      WR     vysledek1
E 0

```

Příklad 2.5.11

Požadujeme vzájemně zaměnit obsahy bytů proměnné *cislo* šířky word.

Na tento požadavek můžeme narazit při zpracování dat z jiného zařízení, které ukládá data v opačném pořadí.

```

B02511.mos
#reg word cislo
;
P 0
      LD     cislo
      SWP
      WR     cislo
E 0

```

Příklad 2.5.12

Podobně jako v předchozím příkladu požadujeme zrcadlově převrátit pořadí bytů proměnné *cislo* šířky long.

```

B02512.mos
#reg long cislo
;
P 0
      LD     cislo      ;$1234

```

```
SWP          ;$1243
SWL          ;$4312
SWP          ;$4321
WR    cislo
```

E 0

2.6. Sklopení zásobníku (STK)

Instrukce **STK** provede sklopení logických hodnot všech vrstev zásobníku do dolních osmi bitů jeho vrcholu.

Příklad 2.6.1

Požadujeme, aby na bitech *vysledek.0* až *vysledek.3* byla provedena operace **SET** s bitovými proměnnými *vstup0*, *vstup1*, *vstup2*, *vstup3* v uvedeném pořadí.

Tradičně budeme postupovat takto:

```
B02601.mos
#reg byte vysledek
#reg bit  vstup0, vstup1, vstup2, vstup3
;
P 0
    LD    vstup0
    SET   vysledek.0
    LD    vstup1
    SET   vysledek.1
    LD    vstup2
    SET   vysledek.2
    LD    vstup3
    SET   vysledek.3
E 0
```

S využitím instrukce **STK** můžeme použít tento postup:

```
P 0
    LD    vstup3
    LD    vstup2
    LD    vstup1
    LD    vstup0
    STK
    AND   %1111
    SET   vysledek
E 0
```

Příklad 2.6.2

Požadujeme uložit do bitu *vysledek* součet všech bitů bytů *vstup0*, *vstup1* a *vstup2*.

```
B02602.mos
#reg bit  vysledek
#reg byte vstup0, vstup1, vstup2
;
P 0
    LD    vstup0
    LD    vstup1
    LD    vstup2
    STK
    AND   %111
```

```
      WR    vyledek
E 0
```

Tento příklad lze však řešit jednodušeji.

```
P 0
      LD    vstup0
      OR    vstup1
      OR    vstup2
      WR    vyledek
E 0
```

Příklad 2.6.3

Požadujeme, aby byly nejprve vytvořeny součty všech bitů bytů *vstup0*, *vstup1*, *vstup2* a součin jejich hodnot pak uložen do bitu *vyledek*.

B02603.mos

```
#reg bit  vyledek
#reg byte vstup0, vstup1, vstup2
;
P 0
      LD    vstup0
      LD    vstup1
      LD    vstup2
      STK
      LD    %1111111111111000
      OR
      NEG
      WRC   vyledek
E 0
```

3. ČÍTAČE, POSUVNÉ REGISTRY, ČASOVAČE, KROKOVÝ ŘADIČ

Tyto instrukce realizují ucelené funkce sekvenčních logických členů (čítače, časovače, posuvné registry). V grafických programovacích jazycích bývají souhrnně označovány jako funkční bloky. Mají operand v prostoru R šířky word (65 536 stavů).

Instrukce čítačů umožňují čítání nahoru (**CTU**), dolů (**CTD**) i obousměrně (**CNT**). Instrukce posuvných registrů umožňují posouvat bity v registru vlevo (**SFL**) a vpravo (**SFR**). Instrukce časovačů umožňují časování od sepnutí vstupu (**TON**), od rozepnutí vstupu (**TOF**), integrující časování (**RTO**) a generování pevných impulzů (**IMP**). Časová jednotka se zadává v instrukci (10 ms, 100 ms, 1 s, 10 s). Instrukce **STE** definuje sekvenční šestnáctikrokový řadič (stepper).

Funkce čítačů a posuvných registrů lze chápat jako podmíněné čítání nebo posuv. Podmínkou je zde náběžná hrana řídicí proměnné. Pro vyhodnocení náběžné hrany má systém pro každý objekt (RW0 až RWmax-1) ve vnitřní paměti vyhrazeny proměnné, které si pamatují hodnotu proměnné z minulého cyklu (tyto proměnné nejsou dostupné z uživatelského programu). Podobně jako u instrukcí **LET**, **BET** je i zde nutné zajistit, aby se uvnitř jednoho cyklu uživatelského programu obsluhovala pro daný objekt a danou vstupní proměnnou pouze jediná vnější proměnná. Nad jedním objektem tedy smí být v jednom cyklu aktivovány instrukce **CTU**, **CTD**, **CNT**, **SFL** a **SFR** s libovolnými parametry, nesmí však být použity opakovaně shodné instrukce (např. dvakrát **CTU**) s různými řídicími proměnnými nad společným objektem nebo kombinace instrukcí **CNT** a **CTU** nebo **CNT** a **CTD**.

Za těchto podmínek smí být kterýkoliv objekt obsluhován instrukcemi **CTU**, **CTD**, **CNT**, **SFL**, **SFR**, aniž by se prováděla inicializace objektu. Podobně nedochází k inicializaci časovačů, pokud jsou obsluhovány stejným typem instrukce se stejnou časovou jednotkou. Při každé změně objektu z režimu časovače na čítač či posuvný registr nebo naopak dochází k inicializaci objektu. Při ní je obsah obsluhovaného slova vynulován a odezvy dynamických řídicích proměnných (z nichž se vyhodnocují náběžné hrany) se nastaví do jedničky (tím se potlačí falešné náběžné hrany způsobené náhodným počátečním stavem řídicí vstupní proměnné). Stejná inicializace se provádí při změně režimu časovače nebo změně časové jednotky.

Při zapnutí sítě nebo při restartu systému se opět nastaví dynamické řídicí proměnné do jedniček a tím se zabrání náhodnému vzniku náběžných hran a nechtěné aktivaci objektů. Všechny objekty se přednostně nastavují do režimu čítače a posuvného registru. Neprovádí se však paušální inicializace objektů - obsah objektů umístěných v remanentní zóně zápisníku zůstává zachován (při dodržení pravidel pro zachování obsahu remanentní zóny - neporušenost obsahu, nezměněný uživatelský program, není studený restart), objekty umístěné mimo remanentní zónu jsou nulovány.

Instrukce funkčních bloků očekávají vstupní proměnné v aktivním zásobníku, v něm předávají i výstupní parametry. Pořadí vrstev a změněné obsahy výstupních parametrů jsou přizpůsobeny možnosti kaskádování funkčních bloků. Dvouhodnotové vstupní parametry se vyhodnocují podle stejných zásad, jako u bitových instrukcí - hodnotou logické proměnné je logický součet (OR) všech šestnácti bitů odpovídající úrovni zásobníku (tj. log.0 při nulové hodnotě, log.1 při nenulové hodnotě). Obdobně jsou nastavovány hodnoty dvouhodnotových výstupních proměnných stejně jako je nastavují bitové instrukce (tj. buď 0 nebo 65 535 - samé jedničky).

Obsah datového objektu (slova programovaného v instrukci) se automaticky aktualizuje. Navíc instrukce čítačů a posuvných registrů vynášejí jeho obsah na vrchol aktivního

zásobníku, protože je předpokládáno jeho další využití. Při kaskádování stačí posunout zásobník zpět instrukcí **POP** 1 a dále použít instrukci odpovídající vyššímu stupni kaskády.

3.1. Čítače (CTU, CTD, CNT)

Instrukce čítačů umožňují čítání nahoru (**CTU**), dolů (**CTD**) i obousměrně (**CNT**) s operandem v prostoru R šířky word.

Příklad 3.1.1

V registru *citac* střádejme počet změn signálu *vstup1* do jedniček v době, kdy je *vstup2* = log.1.

```
B03101.mos
#reg bit  vstup1, vstup2
#reg word citac
;
P 0
    LD    vstup1      ;vstup UP
    LDC    vstup2      ;vstup RESET
    CTU    citac
E 0
```

Příklad 3.1.2

Počet náběžných hran na *vstup* střádejme bez časového omezení. Oproti předchozímu řešení stačí neutralizovat vstup RESET čítače:

```
B03102.mos
#reg bit  vstup
#reg word citac
;
P 0
    LD    vstup      ;vstup UP
    LD    0
    CTU    citac
E 0
```

Příklad 3.1.3

Počet náběžných hran na *vstup* přepočítávejme mod 10 000 (stav čítače se mění v rozmezí 0 až 9 999). Vždy při dosažení mezní hodnoty nastavme *vystup* na log.1.

```
B03103.mos
#reg bit  vstup, vystup
#reg word citac
;
P 0
    LD    vstup      ;vstup UP
    LD    vystup      ;vstup RESET
    CTU    citac
    EQ    10000
    WR    vystup
E 0
```

Příklad 3.1.4

Počet náběžných hran na *vstup* střádejme bez časového omezení na rozsahu long.

```
B03104.mos
#reg bit vstup
#reg long citac
;
P 0
    LD    vstup          ;vstup UP
    LD    0              ;vstup RESET
    CTU   word citac
    POP   1              ;posun zásobníku zpět
    CTU   word citac+2
E 0
```

Příklad 3.1.5

Předpokládejme nulový výchozí stav registru *citac*. Každou náběžnou hranou na *vstup1* zvětšeme obsah o 1, každou náběžnou hranou na *vstup2* zmenšeme jeho obsah o 1. Nulovou hodnotu (vyvážený stav) indikujeme na *vystup1* = log.1, převahu kladných pulzů indikujeme na *vystup2* = log.1 a převahu záporných pulzů na *vystup3* = log.1.

```
B03105.mos
#reg bit vstup1, vstup2, vystup1, vystup2, vystup3
#reg word citac
;
P 0
    LD    vstup1          ;vstup UP
    LD    0              ;vstup RESET
    CTU   citac           ;čítání vpřed
    LD    vstup2          ;vstup DOWN
    LD    0              ;vstup RESET
    CTD   citac           ;čítání zpět
    WRC   vystup1         ;zápis nulovosti
    LD    citac.15
    OR    vystup1         ;nula není kladná
    WR    vystup3         ;záporné
    WRC   vystup2         ;kladné
E 0
```

nebo

```
P 0
    LD    vstup1          ;vstup UP
    LD    vstup2          ;vstup DOWN
    LD    0              ;vstup RESET
    CNT   citac           ;čítání obousměrné
    WRC   vystup1         ;nulovost
    LD    citac.15
    OR    vystup1         ;nula není kladná
    WR    vystup3         ;záporné
    WRC   vystup2         ;kladné
E 0
```

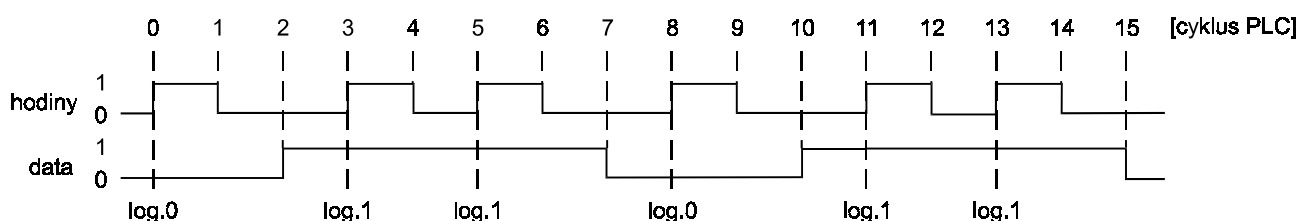
Mlčky jsme předpokládali zobrazení znaménka ve dvojkovém doplňku a omezení rozsahu čísla na 15 bitů - bit *citac.15* je tedy znaménkový.

3.2. Posuvné registry (SFL, SFR)

Instrukce posuvných registrů umožňují posouvat bity v registru vlevo (**SFL**) a vpravo (**SFR**) s operandem v prostoru R šířky word.

Příklad 3.2.1

Realizujeme sériový přenos dat pomocí dvou binárních vstupů. Na vstupu *hodiny* se přenáší synchronizační pulzy, na vstupu *data* jsou přenášena data v pořadí od nejnižšího k nejvyššímu bitu.



Čtení hodnoty na vstupu *data* budeme provádět vždy při náběžné hraně signálu *hodiny*. Střádat je budeme po bytech pomocí instrukce **SFR** (pokud bychom přenášeli data v pořadí od nejvyššího k nejnižšímu bitu, pak bychom použili instrukci **SFL**). Délka dat je vždy 5 bytů.

Program pro příjem bude vypadat takto:

```

B03201pa.mos
#reg bit  hodiny, datax
#reg word shift
#reg byte index, datar[5]
#reg bit  zprava
;
P 0
    LD    hodiny      ;CLCL
    LD    datax       ;DATAI
    SFR   shift
    POP   1           ;DATAO
    JMC   nic         ;test zarážky, log.0 - nic nedělat
    LD    4           ;mez tabulky
    LD    index       ;index do pole data
    LD    shift       ;přijato 8 bitů
    SWP                   ;přesun do dolní poloviny
    WTB   datar       ;zápis do indexovaného pole
    LD    $80         ;zápis zarážky na pozici 8 bitů
    WR    shift       ;před koncem registru
    INR   index       ;zvýšení indexu
    LD    index
    CMP   5           ;test zaplnění
    JC    nic
    LD    0
    WR    index       ;vynulování indexu
    LD    1
    WR    zprava      ;příznak přijaté zprávy
nic:
E 0
;
P 63
    LD    $80         ;zápis zarážky na pozici 8 bitů
    WR    shift       ;před koncem registru
E 63

```

Místo abychom zavedli počítadlo přijatých bitů, využili jsme funkci posuvného registru tak, že jsme zapsali log.1 na pozici, která bude z registru vysunuta po osmi posunech. Stačí pak testovat hodnotu vysouvaného bitu. V případě použití instrukce **SFL** bude zarážka na zrcadlově opačné pozici (zapisovaná hodnota \$100) a odpadne instrukce **SWP**.

Pokud bude mít délka dat sudou hodnotu (např. 6), můžeme zpracovávat najednou 16 bitů.

B03201pb.mos

```
#reg bit  hodiny, datax
#reg word shift, datar[3]
#reg byte index
#reg bit  zprava
;
P 0
    LD    hodiny    ;CLCL
    LD    datax     ;DATAI
    SFR   shift
    POP   1         ;DATAO
    JMC   nic       ;test zarážky, log.0 - nic nedělat
    LD    2         ;mez tabulky
    LD    index     ;index do pole data
    LD    shift     ;přijato 16 bitů
    WTB   datar     ;zápis do indexovaného pole
    LD    $8000     ;zápis zarážky na pozici 16 bitů
    WR    shift     ;před koncem registru
    INR   index     ;zvýšení indexu
    LD    index
    CMP   3         ;test zaplnění
    JC    nic
    LD    0
    WR    index     ;vynulování indexu
    LD    1
    WR    zprava    ;příznak přijaté zprávy
nic:
E 0
;
P 63
    LD    $8000     ;zápis zarážky na pozici 16 bitů
    WR    shift     ;před koncem registru
E 63
```

V případě použití instrukce **SFL** bude zarážka na zrcadlově opačné pozici (zapisovaná hodnota 1).

Při generování zprávy budeme změnu hodnoty na výstupu *data* provádět vždy při nulové úrovni signálu *hodiny*. Instrukce **SFR**, resp. **SFL**, se pro tento účel nehodí. Rozklad bytů na bity provedeme pomocí instrukce **ROR** (pokud bychom přenášeli data v pořadí od nejvyššího k nejnižšímu bitu, pak bychom použili instrukci **ROL** - po instrukci **LTB datas** by musela být ještě instrukce **SWP**). Délka dat je vždy 5 bytů.

Program pro vysílání může vypadat takto:

B03201va.mos

```
#reg bit  hodiny, datax
#reg word shift
#reg byte index, indexb, datas[5]
#reg bit  hotovo, zprava, pomoc, vysilani
;
```

```

P 0
    LD    zprava           ;začít vysílat zprávu?
    SET    vysilani
    SET    hodiny          ;falešný první pulz
    RES    zprava          ;smazání příznaku
    LD    vysilani
    CAD    komunikace      ;obsluha vysílání jen při příznaku
E 0
;
P 60
komunikace:
    LD    hotovo
    JMD    skok2           ;odvysílaná zpráva?
    LD    indexb
    JMD    skok1           ;všechny bity odvysílány?
    LD    4                ;mez tabulky
    LD    index            ;index do pole data
    CMP    5               ;test ukončení
    JNC    dost            ;celá zpráva?
    LTB    datas           ;čtení z indexovaného pole
    WR     shift           ;dalších 8 bitů
    INR    index
    LD     8
    WR     indexb          ;8 bitů
skok1:
    LDC    hodiny          ;nová data po sestupné hraně
    LET    pomoc
    JMC    konec
    LD     shift
    ROR    1
    WR     shift
    LD     %S0.1
    WR     datax           ;zapis hodnoty do výstupu
    DCR    indexb
    RET
skok2:
    LDC    hodiny          ;čekání na dokončení posledního
    LET    pomoc          ;hodinového pulzu
    JMC    konec
    RES    vysilani        ;vysílání dokončeno
    RET
dost:
    LD     0
    WR     index           ;vynulování indexu
    LD     1
    WR     hotovo          ;příznak odvysílané zprávy
konec:
    LD     hodiny
    WRC    hodiny          ;negace signalu hodiny
    RET
E 60

```

3.3. Časovače (TON, TOF, RTO, IMP)

Instrukce časovačů umožňují časování od sepnutí vstupu (**TON**), od rozepnutí vstupu (**TOF**), integrující časování (**RTO**) a generování pevných impulzů (**IMP**) s operandem v prostoru R šířky word. Časová jednotka se zadává v instrukci (10 ms, 100 ms, 1 s, 10 s).

Upozornění: Pokud je časová jednotka časovače přibližně stejná nebo menší než doba cyklu PLC, je funkce příznaků S0.0 a S0.5 nespolehlivá (hodnota časovače narůstá po větších skocích a předvolba, resp. rozsah časovače, je rovnou překročena, takže její dosažení nemusí být detekováno). Příznaky S0.0 a S0.5 lze nahradit testem náběžné hrany příznaků S0.2 a S0.4.

Příklad 3.3.1

Realizujeme časovou funkci ze *vstup*, která potlačí jedničkové impulzy do 150 ms, delší jedničkové impulzy předává na *vystup* se zpožděným náběhem o 150 ms.

```

B03301.mos
#reg bit vstup, vystup
#reg word casovac
;
P 0
    LD    vstup          ;vstup XT
    LD    15             ;vstup VAL
    TON   casovac.0      ;nebo jen TON casovac
    WR    vystup
E 0

```

Příklad 3.3.2

Realizujeme časovou funkci, která na *vystup* předává jedničkové impulzy odvozené od jedničkového stavu *vstup*. Je-li tento vstup jedničkový po dobu kratší než 250 ms, pak je kopírován na *vystup*, delší jedničkový impulz je zkrácen na 250 ms (vynulován po uplynutí 250 ms).

```

B03302.mos
#reg bit vstup, vystup
#reg word casovac
;
P 0
    LD    vstup          ;vstup XT
    LD    25             ;vstup VAL
    TON   casovac.0      ;nebo jen TON casovac
    NEG
    AND
    WR    vystup
E 0

```

Po instrukci časovače je proveden součin:

vystup = „nedočasoval“ AND „je aktivní“

Příklad 3.3.3

Na *vstup1* a *vstup2* jsou přivedena tlačítka (log.1 = stisknuto). Ošetřeme jejich stavy tak, aby tlačítka měla význam bezpečnostního dvouručního ovládání. Jako výstupní informaci nastavme *signal* = log.1 za předpokladu, že obě tlačítka byla nejprve v klidové poloze, potom byla stisknuta v rozmezí do 1,5 s a potom současně držena minimálně 250 ms.

```

B03303.mos
#reg bit  vstup1, vstup2
#reg word casovac1, casovac2, casovac3
#reg bit  signal
;
P 0
    LD    vstup1           ;prvé tlačítko
    LD    150
    TON   casovac1
    NEG
    AND
    AND           ;tisknuto do 1,5 s
    LD    vstup2
    LD    150
    TON   casovac2
    NEG
    AND           ;tisknuto do 1,5 s
    AND           ;obě současně tisknuta do 1,5 s
    LD    25
    TON   casovac3         ;současně držet 250 ms
    SET   signal           ;výstup
    LDC   vstup1
    ORC   vstup2           ;shození výstupu uvolněním
    RES   signal           ;libovolného tlačítka
E 0

```

Příklad 3.3.4

Realizujeme časovou funkci, která potlačí krátkodobé nulové impulzy (kratší než 1,5 s) na *vstup* a delší nulové impulzy předává *vystup* se zpožděním přechodu log.1 → log.0 o 1,5 s.

```

B03304.mos
#reg bit  vstup, vystup
#reg word casovac
;
P 0                                nebo                                P 0
    LD    vstup                LD    vstup
    LD    150                  LD    15
    TOF   casovac              TOF   casovac.1
    WR    vystup              WR    vystup
E 0                                E 0

```

V prvním řešení byla použita časová jednotka 10 ms, ve druhém 100 ms.

Příklad 3.3.5

Předpokládejme, že řídíme vyvrtávačku a že pro *vystup1* je přiřazen pohyb v ose vřetena (log.1 - pohyb do řezu, log.0 - návrat z řezu). Pokud je součtová doba vrtání (součet dob, kdy byl aktivován pohyb do řezu) delší než 70 minut, nastavme *vystup2*, který signalizuje nutnost přebroušení nástroje.

Vynulování časovače bylo provedeno po zapnutí (provádí PLC automaticky). V uživatelském programu pak stačí psát:

```

B03305.mos
#reg bit  vstup, vystup1, vystup2
#reg word casovac
;
P 0
    LD    vystup1           ;řídící proměnná

```

3. Čítače, posuvné registry, časovače, krokový řadič

```
LD vstup ;nulovací proměnná
LD 4200 ;70 min = 70 x 60 s
RTO casovac.2 ;časovač v sekundách
WR vystup2
```

E 0

Na *vstup* předpokládáme připojené tlačítko, kterým obsluhující potvrdí výměnu nebo přeostržení nástroje.

Příklad 3.3.6

V registru *delka* uchovíme délku posledního jedničkového impulsu proměnné *vstup* v sekundách. V zapínací sekvenci je obsah *casovac* vynulován.

B03306.mos

```
#reg bit vstup
#reg word casovac, delka
#reg bit impuls
;
P 0
LD vstup ;řídící proměnná XT
LDC vstup
LET impuls ;její sestupná hrana = RT
JMC navesti
LD casovac ;odložení doby minulého impulsu
WR delka
POP 1
navesti:
LD 0 ;na předvolbě nezáleží
RTO casovac.2 ;měření času
```

E 0

Čítač necháváme pracovat vždy po dobu jediného impulsu. V čekacím stavu jeho hodnotu neměníme a při příchodu nového impulsu teprve přepíšeme jeho časový údaj do *delka* a nastavíme nulovací proměnnou. Zde nepotřebujeme výsledek porovnání s předvolbou, takže hodnota předvolby může být libovolná (tedy i nulová). V tomto případě je změřená doba impulsu zkrácena o dobu prvního cyklu při náběžné hraně impulsu - s ohledem na rozlišovací schopnost měření je však tato chyba zanedbatelná.

Příklad 3.3.7

V registru *delka* uchovávejme hodnotu nejdelšího jedničkového impulsu proměnné *vstup*, měřeného v sekundách. I zde budeme předpokládat, že zapnutím je obsah *delka* nulován. Po dobu překročení dosud nejdelší doby impulsu („překonávání rekordu“) nastavíme *rekord* = log.1.

Mohli bychom vyjít z minulého postupu a doplnit podmíněný přesun časového údaje do *delka* o podmínku překonávání dosavadní uložené hodnoty. Uvedeme však jiný postup, který více využívá možností instrukce **RTO**:

B03307.mos

```
#reg bit vstup
#reg word casovac, delka
#reg bit rekord
;
P 0
LD vstup ;řídící proměnná
LDC vstup ;nulovací proměnná = skončil impuls
LD delka ;předvolba = dosavadní maximum
RTO casovac.2
```

```

WR    rekord           ;překonání dosavadního rekordu
JMC    navesti
LD     casovac
WR     delka

```

navesti:

E 0

Zde jsme jako nulovací proměnnou využili negaci řídicí proměnné - po konci impulzu se časovač vynuluje. Jako předvolbu jsme použili hodnotu dosavadního „rekordu“ - při jeho překonání jej aktualizujeme současnou hodnotou časovače.

Pokud se budeme pak vracet k některé vrstvě zásobníku, je třeba přidat před *navesti* instrukci **POP** 1 k vyrovnání úrovně zásobníku. Posun byl způsoben předcházející instrukcí **LD casovac**.

Příklad 3.3.8

Nastavme *vystup* = log.1 při překročení 40 hodin. Čítač nulujeme při jedničkové hodnotě proměnné *reset*.

Použijeme kaskádování instrukcí **RTO** s **CTU**.

40 hodin = 144 000 s = 2 * 65 536 + 12 928

B03308.mos

```

#reg bit  vstup, reset, vystup
#reg long casovac
#reg bit  pomoc
;
P 0
    LD     vstup           ;řídící proměnná
    LD     reset           ;nulovací proměnná
    RES    vystup          ;předběžné nulování
    LD     12928            ;předvolba nižšího stupně kaskády
    RTO    word casovac.2
    LD     %S0.2            ;předvolba dosažena nebo překročena
    WR     pomoc           ;výsledek porovnání v nižším stupni
    POP    2
    CTU    word casovac+2
    GT     1               ;porovnání ve vyšším stupni
    AND    pomoc
    SET    vystup

```

E 0

Příklad 3.3.9

Realizujeme časovou funkci, která normalizuje šířku jedničkových impulzů na *vstup* na jednotnou délku 3,5 s (kratší pulzy prodlouží, delší zkrátí na 3,5 s). Po skončení normalizovaného impulzu ponechme minimální pauzu 1,5 s, kdy je výstup nulový. Takto zpracovaný průběh zapišme na *vystup*.

B03309.mos

```

#reg bit  vstup, vystup
#reg word casovac1, casovac2
;
P 0
    LD     vstup
    LD     500             ;3,5 s + 1,5 s
    IMP    casovac1        ;impulz + mezera
    WR     vystup          ;mezivýsledek
    LD     350             ;3,5 s

```

```

IMP  casovac2      ;pouze impuls
AND  vystup        ;nebo NEG
WR   vystup        ;      RES vystup ;součin obou

```

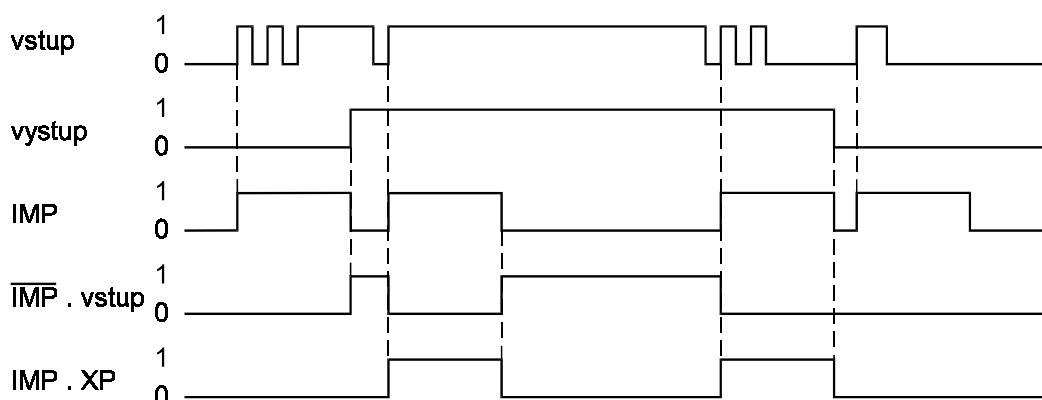
E 0

V programu realizujeme dva impulzní časovače. Prvý má jako hodnotu předvolby součtovou dobu impulsu i povinné pauzy, tedy 5 s a je aktivován od náběžné hrany řídicí proměnné *vstup*. Aktivace druhého je odvozena od počátku aktivace prvního a jeho předvolba odpovídá požadované délce výstupního impulsu (3,5 s). Oba časovače jsme zvolili s časovou jednotkou 10 ms. Bit *vystup* je nastavován na hodnotu logického součinu výstupních proměnných obou časovačů - po dobu výstupního pulzu a povinné mezery, je program necitlivý na změny proměnné *vstup*, další impuls může být vyhodnocen až při první náběžné hraně po ukončení povinné mezery.

Příklad 3.3.10

Předpokládejme, že informace o stavu procesu, přivedená na *vstup* má tu vlastnost, že každá změna stavu je provázána sérií náhodných zákmitů, které se ustálí do 1 s. Navíc se v signálu *vstup* mohou vyskytovat krátkodobé ojedinělé zákmity (zlomky sekundy), které nenesou informaci o změně stavu. Stav se mění poměrně zvolna - typická doba setrvání ve stavu log.1 nebo log.0 je desítky sekund (např. plovákový snímač hladiny objemné nádrže). Realizujeme časový filtr, který potlačí krátkodobé rušivé impulzy (kratší než 1 s) a potlačí přechodný děj při změně stavu, za platný stav bude považována hodnota *vstup* za 1 s po první změně. Výsledek bude uložen do *vystup*. Jedná se tedy o funkci zpožděného přitahu a odpadu.

Použití instrukcí **TON** a **TOF** není optimální, protože oba časovače jsou během svého aktivního stavu citlivé na změny vstupní proměnné. Situaci přibližuje časový diagram:



Časovač se zaktivuje vždy prvou změnou hodnoty *vstup* s hodnotou předvolby 1 s. Výstupní proměnnou *vystup* můžeme definovat takto:

- pokud časovač není aktivní, je výstup roven současnému vstupu
- pokud je časovač aktivní, je výstup roven hodnotě vstupu před aktivací časovače.

Tuto definici můžeme vyjádřit výrazem:

$$vystup = \overline{IMP} \cdot vstup + IMP \cdot XP$$

kde *XP* představuje hodnotu *vstup* před aktivací časovače (před hranou, která jej aktivovala).

Výraz můžeme realizovat programem:

B03310.mos

#reg bit vstup, vystup

#reg word casovac

#reg bit zaloha, minuly, impulz, odklad

;

P 0

LD zaloha

WR minuly ;minulý vstup

LD vstup

WR zaloha ;současný vstup

XOR ;změna vstup

LD 100 ;1 s

IMP casovac

LET impulz ;impulz = současný IMP

JMC navesti

AND minuly ;začíná-li IMP, odložení minulé hodnoty vstup

WR odklad ;do odklad

navesti:

LD impulz

AND odklad ;IMP . XP

LDC impulz

AND vstup ;IMP . vstup

OR

WR vystup

E 0

3.4. Krokový řadič (STE)

Instrukce **STE** provádí komplex činností. Je určena především pro realizaci krokového řadiče, lze ji však použít i k jiným účelům, kde využíváme vedlejších účinků. Nejprve popíšeme vlastní provedení instrukce, pak teprve rozvedeme možnosti použití.

Byte adresovaný instrukcí **STE** má význam stavu krokového řadiče. Význam mají hodnoty 0 až 15, ale vyšší hodnoty nejsou na závalu. Dolní polovina tohoto bytu (bity 0 až 3) je převedena na masku 1 z 16 („maska stavových proměnných“, „stavová maska“) podle předpisu:

stav (bity 3 - 0)	bitová maska
0	00000000 00000001
1	00000000 00000010
:	:
14	01000000 00000000
15	10000000 00000000

Po převedení čísla stavu na stavovou masku se testuje, zda podmínkový vektor má na pozici jedničky v masce i jedničkový podmínkový bit. Pokud ano, pak:

- zvýší se číslo stavu v adresovaném bytu o jedničku
- v kruhu se posune stavová maska vlevo o jednu pozici
- nastaví se příznak S1.0 = log.1
- pokud při rotaci došlo k přenosu (stav se měnil z 15 na 16), nastaví se navíc S1.1 = log.1

Při nesplnění podmínky se nemění číslo stavu ani stavová maska, registr S1 = 0.

Aktualizovaná hodnota stavové masky je zapsána na vrchol zásobníku. Aktualizované číslo stavu je uloženo do adresovaného bytu - jeho hodnota není korigována přepočtem mod 16 (jeho hodnota může být libovolně větší do 255). Do dalšího bytu (na adrese

o jednu vyšší, než je operand instrukce) je uložen obsah horní poloviny předchozího bytu (obsah 0 až 15), která má význam přenosu nebo přetečení a může být využita ke kaskádování nebo segmentování řadičů.

Příklad 3.4.1

Realizujeme dekodér číslo → maska 1 z n. Číslo je uloženo v dolních čtyřech bitech bytu *cislo*.

Pokud je podmínkový vektor nulový (nedojde k přechodu), pak instrukce **STE** převádí číslo stavu (jeho dolní byte) na masku, která má v šestnácti bitech jedinou jedničku. Pro masku 1 z 16 můžeme psát přímo:

```
B03401a.mos
#reg byte cislo, prenos
#reg word maska
;
P 0
    LD    0
    STE   word cislo
    WR    maska
E 0
```

Pokud předem přepočteme obsah stavu mod n ($n < 16$), můžeme převádět číslo stavu na masku 1 z n ($n < 16$) např. postupem:

```
B03401b.mos
#reg byte cislo, prenos
#reg word maska
;
P 0
    LD    cislo
    AND   7           ;maska 1 z 8
    WR    cislo
    LD    0
    STE   word cislo
    WR    maska
E 0
```

nebo

```
P 0
    LD    %11111000
    RES   cislo
    LD    0
    STE   word cislo
    WR    maska
E 0
```

V prvním postupu jsme přepočet mod 8 provedli odmaskováním horních pěti bitů vrcholu zásobníku A0, ve druhém postupu jsme stejnou operaci provedli přímo v bytu *cislo* instrukcí **RES**.

Číslo stavu můžeme převést na masku 1 z 10 následujícím postupem:

```
B03401c.mos
#reg byte cislo, prenos
#reg word maska
;
P 0
    LD    cislo
    DIV   10           ;přepočet
    SWP
    WR    word cislo    ;uložení zbytku po dělení
    LD    0
    STE   word cislo
    WR    maska
E 0
```

Můžeme však realizovat i masku pro $n > 16$, např. pro $n = 32$:

```

B03401d.mos
#reg byte cislo, prenos
#reg word pomoc
#reg long maska
;
P 0
    LD    0
    STE   word cislo
    WR    pomoc           ;odlož masku 1 z 16
    ANC   prenos.0
    WR    word maska      ;není-li přenos, uložit masku do spodního
                          ;wordu, jinak 0
    LD    pomoc           ;přečti masku
    AND   prenos.0
    WR    word maska+2    ;je-li přenos, uložit masku do horního
                          ;wordu, jinak 0
E 0

```

Příklad 3.4.2

Realizujeme nepodmíněnou rotaci masky.

Pokud obsahuje podmínkový vektor samé jedničky (65 535), pak se při každé aktivaci instrukce provede přechod. To má za následek, že číslo stavu se vždy inkrementuje a odpovídající maska je vždy rotována. V předchozím příkladu 3.4.1 stačí vždy zaměnit instrukci **LD 0** za **LD 65535**.

Příklad 3.4.3

Realizujeme rotaci masky se společnou podmínkou.

Jde o kombinaci obou předchozích případů, pokud hodnotu podmínkového vektoru získáme jako výsledek instrukce **LD** s bitovým operandem. Např. následujícím postupem čítáme v registru *cislo* počet náběžných hran proměnné *vstup* (mod 256) a při každé náběžné hraně rotujeme masku 1 z 16 o jednu pozici.

```

B03403.mos
#reg bit vstup
#reg byte cislo, prenos
#reg word maska
#reg bit impulz
;
P 0
    LD    vstup
    LET   impulz
    STE   word cislo
    WR    maska
E 0

```

Příklad 3.4.4

Realizujeme krokový řadič.

Nejsnáze je instrukce **STE** použitelná v případech, kdy řízený proces má cyklický charakter a je tvořen sledem na sebe navazujících akcí, přičemž nová akce je podmíněna vykonáním akce předešlé. Pro ilustraci uvažujeme zjednodušený příklad poloautomatického vrtacího pracoviště. Výměna polotovarů se provádí ručně, na jedno upnutí je vyvrtána jediná díra. Postup operací je následující:

- výměnu polotovaru nahlásí obsluha tlačítkem START
- vřetenem (jehož otáčení se nevypíná) sjede rychloposuvem nad rovinu materiálu)

- vrtání se provádí pracovním posuvem
- po dosažení dna díry se vřeteno vrací do výchozí polohy rychloposuvem
- uvolní se materiál

Zjednodušení spočívá zejména v tom, že byl vybrán příklad triviální jednoduché technologické operace a byly zjednodušeny logické podmínky a kontroly (nekontroluje se otáčení vřetene, zařazení rychloposuvu nebo pracovního posuvu, překročení bezpečnostních mezních poloh na stroji, přetížení vřetena apod.).

K přesnému popisu sekvenčních úloh je výhodné použít metodiku grafického jazyka GRAFCET, který se stává světovou normou. Pro náš případ vystačíme se třemi prvky:

stav - Určuje nějakou význačnou situaci v našem programu nebo v našem modulu řízeného procesu. Může být ve dvou stavech: aktivní - pasivní. Aktivní stav způsobí provedení akcí, které jsou mu přiřazeny, pasivní stavy žádné akce nevyvolávají. Stavy budeme označovat obdélníkem s vypsáním číslem stavu. Přiřazené akce nebo hodnoty výstupů se mohou zapsat vedle symbolu vpravo.



AKCE - komentář

počáteční stav - Má stejný význam jako ostatní stavy. Navíc má tu výsadu, že proces se začíná vyvíjet vždy z tohoto stavu. Značí se dvojité orámovaným obdélníkem:



AKCE - komentář

přechod - Značí se jako vodorovná čárka, která přerušuje spojnice mezi dvěma stavy, vedle níž se uvádí jednobitová podmínka přechodu. Aktivují se pouze ty přechody, které vycházejí z aktivního stavu. Pokud není splněna podmínka žádného z přechodů, nemění se aktivace stavu. Při splnění podmínky je proveden přechod - dosud aktivní stav se pasivuje a aktivuje se stav za přechodem. Aktivní stav zůstává aktivním do té doby, než splněná podmínka aktivního přechodu „nepropustí“ aktivaci do následného stavu. Podmínky přechodů můžeme chápat jako „ventily“, které ovládají tok aktivace mezi stavy a tím i vývoj systému (programu).

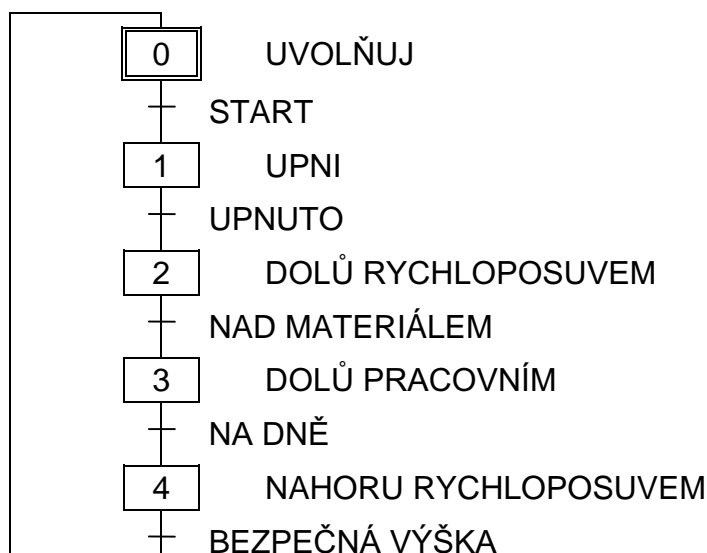
† PODMÍNKA PŘECHODU

Je nutné dodržet základní pravidla:

- každý výstup z libovolného stavu musí končit na vstupu nějakého přechodu
- výstup z libovolného přechodu musí ústít na vstup nějakého stavu

Nesmí tedy existovat posloupnost stavů, které nejsou odděleny přechody a nesmí existovat posloupnost přechodů, které nejsou odděleny stavy. Nesmí ani existovat stav nebo přechod, jehož výstup nikam nevede, nebo na jehož vstup nic nepřichází (s výjimkou počátečního stavu).

Graf, který podle těchto pravidel vytvoříme, budeme nazývat „přechodovým grafem“ nebo též „stavovým nebo vývojovým grafem“ systému (na rozdíl od vývojového diagramu programu). Náš problém lze popsat následujícím přechodovým grafem:



Má 5 stavů, ovládá jej 5 vstupů („start“, „nad materiálem“, „na dně“, „bezpečná výška“, „upnuto“) a nastavuje 5 výstupů („upni“, „uvolňuj“, „dolů rychloposuvem“, „dolů pracovním“, „nahoru rychloposuvem“). Je mnoho způsobů, jak tuto úlohu řešit. Výhodné je použití instrukce **STE**.

V zapínací sekvenci je nastaven počáteční stav č. 0.

B03404.mos

```

#reg byte  vstupy      ;.0 start
                        ;.1 upnuto
                        ;.2 nad materiálem
                        ;.3 na dně
                        ;.4 bezpečná výška

#reg byte  vystupy     ;.0 uvolňuj
                        ;.1 upni
                        ;.2 dolů rychloposuvem
                        ;.3 dolů pracovním
                        ;.4 nahoru rychloposuvem

#reg byte  stav, prenos, maska
;
P 0
  LD  %11111
  RES vystupy      ;předběžné vynulování výstupů
  LD  vstupy
  AND                               ;oddělení bezvýznamných vstupů
  STE word stav    ;ošetření přechodu
  WR  maska        ;odložení
  LD  maska.5      ;ošetření přechodu - 0
  RES maska.5      ;vynulování výstupu .5
  RES stav         ;vynulování stavu
  SET maska.0      ;nastavení výstupu .0
  LD  maska        ;nastavení výstupů
  SET vystupy
  
```

Nejprve jsme vynulovali prostor pro 5 výstupů krokového řadiče. Pak jsme z bytu *vstupy* oddělili 5 vstupů, které krokový řadič ovládají a instrukcí **STE** jsme ošetřili případný přechod. Při přechodech do stavů 1 až 4 je vše v pořádku a mohli bychom hodnotu stavové masky přičíst k obsahu bytu *vystupy*. Při přechodu ze stavu 4 je však v masce navíc jednička na pozici č. 5 a na pozici č. 0 chybí. Proto je třeba tento přechod zvlášť ošetřit. Současně je nastaven i *stav* = 0. Mlčky jsme předpokládali, že všechny funkce stroje jsou „samosvorné“, že dosažený stav se bude sám udržovat. To patrně nebude

splněno např. při upínání obrobku. V tomto případě je nutné ještě doplnit na konec programu:

```
AND    %11110           ;ve stavech 1 až 4 je obrobek trvale upínán
SET    upni
```

E 0

Domníváme se, že tradičním programem nelze vytvořit úspornější program. Výhodnost instrukce **STE** se však projevuje zejména při složitějších úlohách. Je třeba přiznat, že v tomto ilustrativním případě jsme si trochu pomohli vhodným uspořádáním vstupů a výstupů.

Viz též příklady 7.2.1, 8.1.6.

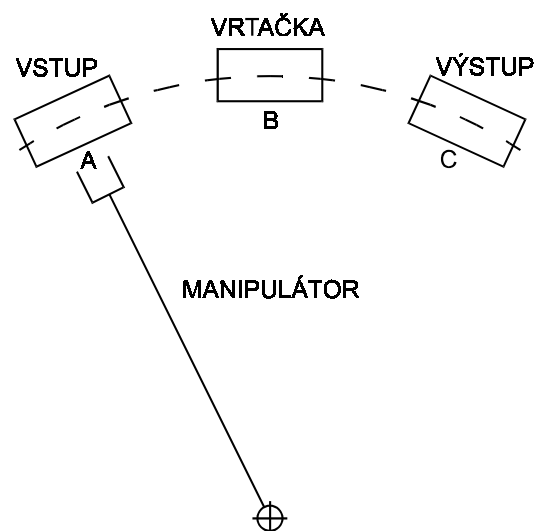
Příklad 3.4.5

Budeme předpokládat obdobnou technologickou operaci jako v předchozím příkladu 3.4.4, pouze s tím rozdílem, že ji zautomatizujeme následovně:

Na vstupní místo A přicházejí polotovary (např. válečkovým dopravníkem), zde je uchopí rameno manipulátoru, přemístí pod vřeteno vrtačky (na místo B), po dobu vrtání se zaaretuje a zpevní, pak přemístí nad výstupní místo C, kde jej uvolní (na přepravník, do bedny).

Proces můžeme rozdělit na části:

- vstupní místo A: dává jen výstup: „připraven“
- rameno manipulátoru:
 - otočný pohyb: 3 polohy: „místo A“
„místo B“
„místo C“
 - řídí se povely: „vlevo“
„vpravo“
 - svislý pohyb: 2 polohy: „nahore“
„dole“
 - řídí se povely: „nahoru“
„dolů“
 - čelisti: 2 polohy: „sevřeno“
„rozevřeno“
 - řídí se povely: „sevři“
„rozevři“
- pracovní místo B (vrtání):
 - aretační zařízení: stav: „aretováno“
 - povel: „aretuj“
 - vřeteno: 3 polohy: „bezpečná výška“
„pracovní výška“
„na dně“
 - řídí se povely: „dolů rychloposuvem“
„dolů pracovním“
„nahoru rychloposuvem“

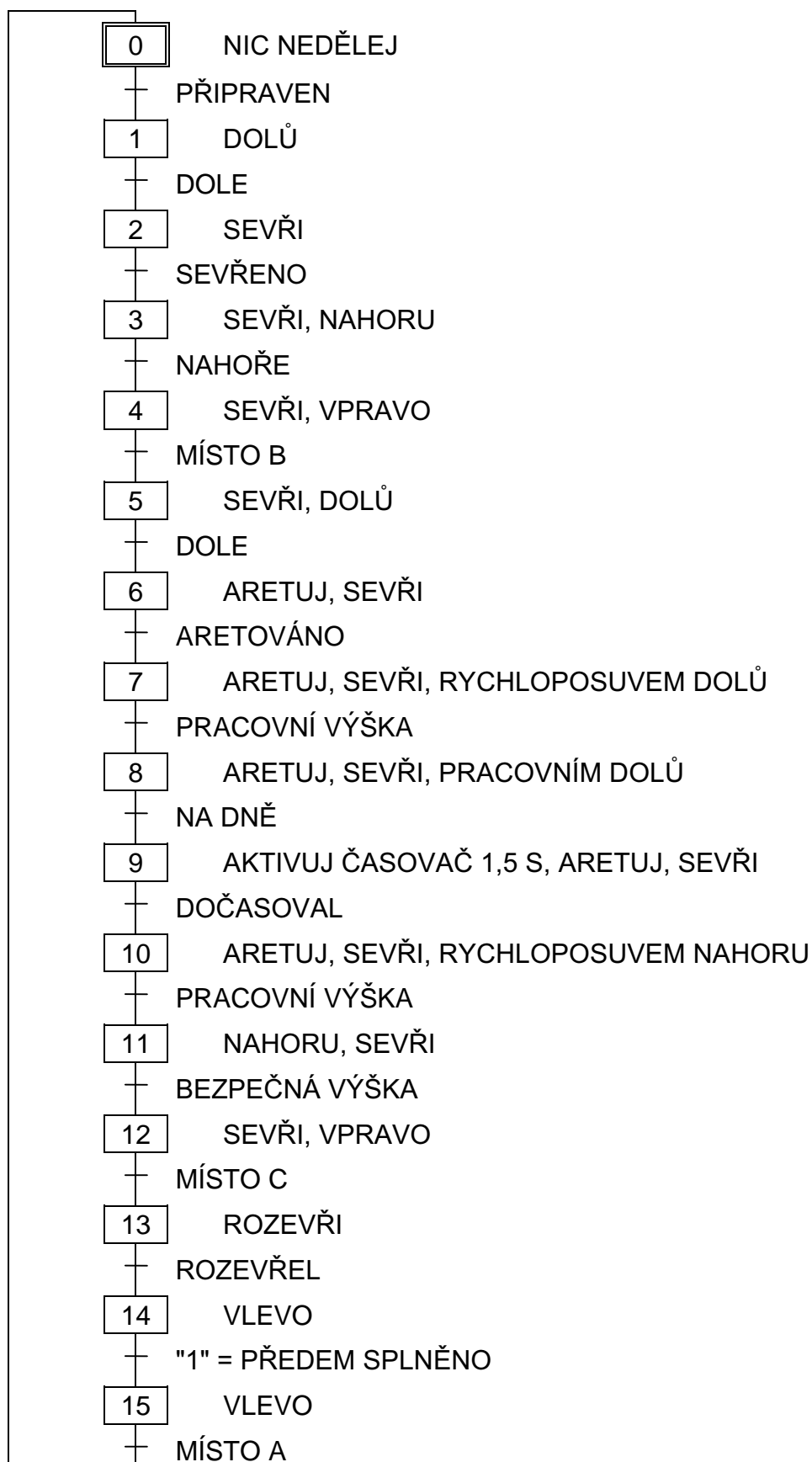


Předpokládáme, že obrobek na místě A i B je uložen a obráběn v dolní poloze ramena manipulátoru, přemísťován v horní poloze. U čelisti rozlišujeme dva mezní stavy (sevřeno, rozevřeno), zatímco u aretačního zařízení rozlišujeme stav jediný (aretováno) a jeho nepřítomnost považujeme za stav nearetováno. Oproti předchozímu postupu jsme zde předepsali prodlevu na dně díry v délce 1,5 s.

Smyslem těchto příkladů je ilustrovat možnosti použití a výhody instrukce **STE**, nikoliv prezentovat optimální návrhy automatizovaných technologických pracovišť a jsme si vědo-

mi mnohých zjednodušení a slabých míst (přejezdy manipulátoru, nedostatečná kontrola a blokování chybových a mezních stavů).

Průběh procesu opět můžeme znázornit přechodovým grafem:



V tomto řešení má proces 15 stavů (0 až 14). Abychom nemuseli programovat zkrácení cyklu řadiče ze stavu 14 do stavu 0, zařadili jsme stav č. 15 jako prázdný (žádná akce, jedničková podmínka). Do procesu vstupuje 12 vstupních proměnných a vystupuje z něho 10 výstupních proměnných. Navíc je od přechodu do stavu 9 aktivován časovač (např. **TON**) a jeho výstupní proměnná je využita jako podmínka přechodu do stavu 10.

I zde přizpůsobíme obsazení vstupů a výstupů řešenému úkolu. Lze oprávněně namítnout, že tímto způsobem nutíme řešitele přizpůsobovat způsob kabeláže, tedy technické vybavení podřídít eleganci řešení programového vybavení - přizpůsobujeme tedy hardware potřebám softwaru. Obecně je tento postup proti „duchu doby“ a proti „výdobytkům“ éry programovatelných systémů. Lze odpovědět řečnickou otázkou: „Jsme-li v situaci, že obsazení svorek může být libovolné, proč by nemělo být takové, aby optimalizovalo řešení klíčové úlohy projektu?“ Dosti toho, že je mnoho případů, kdy obsazení svorek není libovolné, kdy se na činnosti krokového řadiče účastní vnitřní proměnné, vazby mezi časovači, čítači, kombinované podmínky a podobně. Obecně je třeba vzít v úvahu, že „optima se dosáhne ústupky na obou stranách“.

Na neobsazených pozicích vstupů a výstupů mohou být signály, které nesouvisejí s naší úlohou. Můžeme ji řešit následujícím programem (*stav* a *casovac* jsou vynulovány v zapínací sekvenci):

```
B03405.mos
#reg word vstupy, vystupy
#reg byte stav, prenos
#reg word podminky, casovac, maska
#reg bit cas
;
#def pripraven      podminky.0
#def dole           podminky.1
#def sevreno        podminky.2
#def nahore         podminky.3
#def misto_B        podminky.4
#def dole2          podminky.5      ;sw kopie bitu podminky.1
#def aretovano      podminky.6
#def prac_vyska     podminky.7
#def na_dne         podminky.8
#def docasoval      podminky.9      ;výstup ze sw časovače
#def prac_vyska2    podminky.10     ;sw kopie bitu podminky.7
#def bezp_vyska     podminky.11
#def misto_C        podminky.12
#def rozevrel       podminky.13
#def jedna          podminky.14     ;sw zapisuje log.1
#def misto_A        podminky.15
;
#def dolu           maska.1
#def sevri          maska.2
#def nahoru         maska.3
#def vpravo         maska.4
#def aretuj         maska.6
#def rych_dolu      maska.7
#def prac_dolu      maska.8
#def rozevri        maska.9
#def rych_nahoru    maska.10
#def vlevo          maska.11
;
P 0
    LD    vstupy
    WR    podminky
```



```

LD      1
WR      jedna                ;splněná podm. pro přechod do 15
LD      dole
WR      dole2                ;„dole“ pro přechod do 6
LD      cas
WR      docasoval            ;výstup časovače z minulého cyklu
LD      prac_vyska
WR      prac_vyska2          ;pracovní výška - pro přechod do 11
LD      %000011111011110
RES     vystupy              ;vynulování používaných výstupů
LD      podminky
STE     word stav            ;aktivace přechodů řadiče
WR      maska                ;odlož masku - předobraz výstupů
AND     %0001111111111100
WR      sevri                ;ve stavech 2 až 12 „sevři“
AND     %0000011111000000
WR      aretuj              ;ve stavech 6 až 10 „aretuj“
LD      maska.11
SET     nahoru              ;ve stavech 3 a 11 „nahoru“
LD      maska.5
SET     dolu                ;ve stavech 1 a 5 „dolů“
LD      maska.15
OR      maska.14
WR      vlevo                ;ve stavech 14 a 15 „vlevo“
LD      maska.12
SET     vpravo              ;ve stavech 12 a 4 „vpravo“
LD      rozevri
LD      150
TON     casovac              ;ve stavu 9 čekej 1,5 s
WR      cas                  ;odložit pro příští cyklus
LD      maska.13
WR      rozevri              ;ve stavu 13 „rozevři“
LD      maska                ;hodnoty výstupů
AND     %000011111011110    ;vynulovat neobsluhované
SET     vystupy              ;nastavení výstupů Y

```

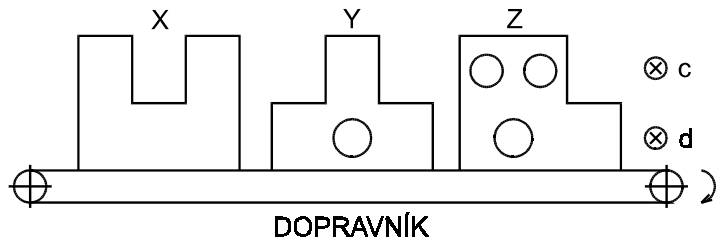
E 0

Program je podrobně komentován - stačí stručné vysvětlení. V úvodní části jsou sejmuty *vstupy* do proměnné *podminky* a zde jsou korigovány hodnoty některých podmínkových bitů (opakující se podmínkové proměnné, výstup časovače, splněná podmínka). Pak jsou vynulovány pozice výstupů, které obsluhuje náš sekvenční řadič. Po aktivaci řadiče (instrukce **STE**) je hodnota stavové masky odložena do *maska* jako prototyp hodnot výstupů, které jsou postupně korigovány (některé výstupy jsou přemístěny, některé jsou nastaveny na hodnoty logického součtu), je aktivován časovač **TON** pro měření prodlevy ve stavu 9. Na závěr jsou v obsahu *maska* vynulovány pozice, které řadič neovládá a výsledek je přičten k obsahu *vystupy*. Nejdelší na programu je transformace vstupních a výstupních proměnných řadiče, přestože jsme maximálně přizpůsobili rozmístění vstupů a výstupů systému. Při náhodném rozmístění vstupů je nejúčelnější střídat osmici vstupních podmínek v zásobníku (A0 až A7) a pak instrukcí **STK** sklopit do bytu. Při náhodném rozmístění výstupů nebo při komplikovanějších vztazích mezi stavy a výstupy je účelnější transformovat číslo stavu na hodnoty výstupů, případně na návěští podprogramu akce tabulkou (viz kap.8.).

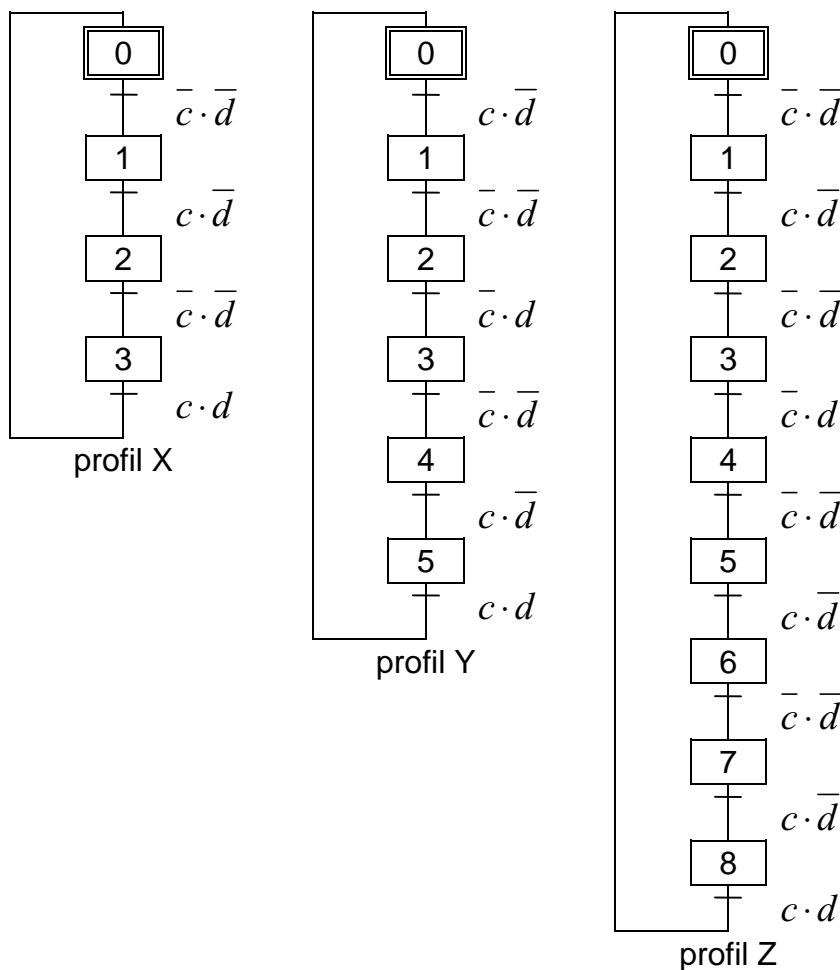
Program není krátký, ale pro porovnání doporučujeme, aby si čtenář vyřešil různé varianty programů pro zadanou úlohu.

Příklad 3.4.6

Pásový dopravník se pohybuje konstantní rychlostí a přemísťuje součásti různých tvarů. Nad ním jsou umístěna dvě světelná čidla *c*, *d* dle nákresu, jejich vývody jsou připojeny na vstupy *fotoc* a *fotod* (osvětlená fotonka = log.1, zastíněná = log.0). Realizujeme program, který při rozpoznání profilů X, Y, Z nastaví proměnné *profil_X*, *profil_Y*, *profil_Z*. Jiné profily, jejich překrývání nebo jiná orientace jsou vyloučeny. Rychlost posuvu je konstantní, za dobu proběhnutí jednoho dílce vykoná uživatelský program minimálně 50 cyklů, během mezery mezi díly vykoná minimálně 10 cyklů.



Střídání signálů na vstupech je popsáno přechodovými grafy. Stavům neodpovídají žádné vnější výstupy, pouze koncové stavy nastavují vnitřní proměnné *profil_*. Stavů řadičů A, B, C jsou v každé mezeře (*c* - *d*) nulovány - tím není nutné řešit zkrácení jejich délky.



B03406.mos

```
#reg bit  fotoc, fotod
#reg byte stav_X, prenos_X, stav_Y, prenos_Y, stav_Z, prenos_Z
#reg bit   profil_X, profil_Y, profil_Z
;
P 0

LD      fotoc           ;řadič pro rozpoznání profilu X
XOR     %101           ;čidlo c
LD      fotod           ;negované c
XOR     %111           ;čidlo d
XOR     %111           ;negované d
```

```

AND          ;podmínkový vektor
STE word stav_X      ;řadič X
AND %10000
SET profil_X        ;rozpoznal X
                ;řadič pro rozpoznání profilu Y
LD fotoc           ;čidlo c
XOR %1110          ;negované c
LD fotod           ;čidlo d
XOR %11011         ;negované d
AND             ;podmínkový vektor
STE word stav_Y      ;řadič Y
AND %1000000
SET profil_Y        ;rozpoznal Y
                ;řadič pro rozpoznání profilu Z
LD fotoc           ;čidlo c
XOR %01011101      ;negované c
LD fotod           ;čidlo d
XOR %11110111      ;negované d
AND             ;podmínkový vektor
STE word stav_Z      ;řadič Z
AND %1000000000
SET profil_Z        ;rozpoznal Z
LD fotoc
AND fotod           ;mezera = 1
RES stav_X          ;vynuluj stavy
RES stav_Y
RES stav_Z
E 0
;
P 63
LD 1
WR fotoc           ;přednastavení fotobuněk c, d
WR fotod           ;(mezera)
E 63

```

Viz též příklad 8.3.9.

Příklad 3.4.7

Na vstupy *vstup0* až *vstup15* jsou připojeny spínací kontakty šestnácti nezávislých tlačítek (log.1 = stisknuto tlačítko). Pokud není žádné stisknuto, nastaví se v registru *stav* kód = 16, jinak se do *stav* uloží pořadové číslo nejnižší pozice se sepnutým kontaktem (0 až 15). Způsobů řešení je mnoho, uvedeme ten, který pracuje s instrukcí **STE**.

```

B03407a.mos
#reg word  vstupy
#reg byte  stav, prenos
;
P 0
LD 0
WR stav      ;počáteční stav
cyklus:      ;začátek cyklu
LDC vstupy   ;negované vstupy
STE word stav
LD %S1.1
JMD konec   ;při otočce skonči
JS cyklus    ;opakuj do nalezení
konec:
E 0

```

Jako podmínkový vektor jsou použity negované vstupy. Instrukce **STE** je cyklicky vykonávána až do té doby než se zastaví na nesplněné podmínce na pozici nejnižšího stisknutého tlačítka. Pokud není žádné tlačítko stisknuto, skončí cyklus při přechodu ze stavu 15 do 16.

Elegantnější je v tomto případě řešení s bitovou tabulkovou instrukcí **FTB**.

```
B03407b.mos
#reg word  vstupy
#reg byte stav
;
P 0
    LD    15          ;počet tlačítek - 1
    LD    1           ;hledání log.1
    FTB   vstupy.0     ;bitový přístup
    WR    stav
E 0
```

Poznámka ke krokovým řadičům

Obecně bývá problém realizace krokových řadičů nebo automatů složitější. Často bývá požadováno, aby systém v určitých stavech setrval nějakou minimální dobu. Pokud se naopak do určité maximální doby neuskuteční nový přechod, je to příznakem nějaké poruchy v řízené soustavě. Někdy je požadováno, aby po splnění podmínky přechodu zařadil systém časovou prodlevu a teprve po jejím odeznění přechod uskutečnil (čas na uklidnění). Řídící systémy, které jsou specializovány na realizaci krokových řadičů, mají již ve své základní výbavě zabudován soubor časovačů, které lze přiřadit jednotlivým přechodům s různě nastavenými hodnotami. Pokud tato možnost chybí, je nutno časové závislosti doplnit a to buď na úkor objemu instrukcí, které ošetřují jednotlivé časovače a doplňují o jejich výstupy podmínky přechodu nebo na úkol rozšíření řadiče o čekací stavy. Podrobněji se k této problematice vrátíme v souvislosti s tabulkovou realizací krokových řadičů a automatů v kap.8.

Specializované programové řadiče (sekvencery) poskytují uživateli určitý komfort při ovládání. Typicky pracují ve čtyřech režimech:

- automatický - nepřerušené vykonávání cyklů řadiče
- stop po konci cyklu - po přechodu do stavu 0 se řadič zastaví a čeká na ruční odstartování (např. po kontrole výrobku nebo po ručním založení dílce apod.)
- krok po kroku - před každým přechodem se systém zastaví a čeká na odstartování - tento režim je využíván při odkódování či kontrole technologie nebo uživatelského programu
- ruční řízení - uživatel přímo ovládá výstupy. Tento režim je použitelný při seřizování technologie, při opravách, při ručním dokončení nestandardní operace, při ošetření chybných stavů nebo havarijních situací.

Většina z těchto možností je uživateli PLC TECOMAT dostupná, musí si ji však sám naprogramovat.

Při úlohách rozpoznávání profilů podle sledu signálů z čidel polohy jsme předpokládali ideální souběhy hran bez zámkitů a fázových posuvů - to v reálných situacích obvykle nelze zaručit, takže je nutné úloze rozpoznávání předřadit úlohu předzpracování (filtrace) vstupních signálů. Jedním ze způsobů je časová filtrace, jejíž jednu možnost jsme naznačili v příkladu 3.3.10. Možná a účelná je však i majoritní filtrace, kdy v každém cyklu uživatelského programu sejmeme vzorky současných vstupů a přidáme k vzorkům z minulých cyklů (zásobník, kruhový registr). Za platnou úroveň pak budeme považovat tu, pro kterou „hlasuje“ většina vzorků z několika posledních cyklů (např. 3, 5, 7). Rozhodovací pravidla mohou však být jiná - např. za změnu úrovně můžeme považovat situaci, kdy

všechny vzorky se „shodnou“ na stejné hodnotě, do té doby bude uvažován původní stav, apod. Ke všem těmto úlohám lze velmi výhodně využívat instrukci **FLG** (počet jedniček).

4. ARITMETICKÉ INSTRUKCE

Aritmetické instrukce pracující s vrcholem zásobníku a vstupní proměnnou umožňují sčítání, odčítání, násobení, dělení a porovnání šířky byte, word, long.

Tytéž instrukce existují též jako bezoperandové, pracující na vrstvách 0 a 1 aktivního zásobníku (šířka word), resp. dvojvrstvách 01 a 23 (šířka long).

Další instrukce pracující se zásobníkem umožňují převody čísel, inkrementaci a dekrementaci hodnoty na vrcholu zásobníku i v registru a rotaci vrcholu zásobníku.

Pořadí operandů

Pro jistotu shrňme, v jakém pořadí jsou zpracovávány operandy aritmetických instrukcí. Pro instrukce s vyjádřeným operandem se provádí operace

$$A0 = (\text{původní } A0) @ (\text{operand})$$

kde @ je typ operace

operandem je buď adresovaný obsah nebo bezprostřední data.

Například instrukce **SUX** R2, **SUX** RW2, **SUX** RL2 provádějí

$$A0 = (\text{původní } A0) - (\text{obsah } R2)$$

$$A0 = (\text{původní } A0) - (\text{obsah } RW2)$$

$$A01 = (\text{původní } A01) - (\text{obsah } RL2)$$

Instrukce **SUB** 21971 provádí

$$A0 = (\text{původní } A0) - 21971 + CI$$

U instrukcí s nevyjádřeným operandem (bezoperandových instrukcí) se nejprve posune zásobník zpět, původní A0 (resp. A01) se vysune do fiktivního registru operandu a předepsaná operace se provede jako v předchozích případech. Operandy do A1 a A0 (resp. A23 a A01) plníme obvykle instrukcí **LD**. Typický postup

```
LD    operand 1
LD    operand 2
SUB
```

provádí operaci

$$A0 = (\text{operand 1}) - (\text{operand 2}) + CI$$

Pořadí, v jakém jsou zpracovávány operandy, je tedy stejné jako pořadí, ve kterém byly programovány. Například postup

LD	operand1	je rovnocenný postupu	LD	operand1
LD	operand2		SUB	operand2
SUB				

a na vrcholu A0 uloží

$$A0 = (\text{obsah operand1}) - (\text{obsah operand2}) + CI$$

Příznaky v S0

- S0.0 - ZR - příznak nulovosti výsledku
 - je-li ZR = 1, výsledek aritmetické operace byl nulový
- S0.1 - CO - výstupní přenos
 - při aritmetické operaci došlo k přenosu

- S0.2 - \leq - výsledek nerovnosti
 S0.3 - CI - vstupní přenos
 - přičítá se k operandu instrukce, po vykonání instrukce je nulový
 - o nastavení CI se musí postarat uživatel v případě kaskádování postupem
- ```
LD S0.1
WR S0.3
```
- je tedy zajištěno, že se kaskádování neprovede samovolně, ale vždy je nutné jej zvlášť vyžádat.
- S0.4-S0.6- - nejvyšší číslice výsledku převodu do formátu BCD (instrukce **BCD**)

#### 4.1. Sčítání a odčítání (ADX, ADD, ADL, SUX, SUB, SUL, INR, DCR)

Aritmetické instrukce pracující s vrcholem zásobníku a vstupní proměnnou umožňují sčítání a odčítání šířky byte, word, long (**ADX, ADD, ADL, SUX, SUB, SUL**), inkrementaci a dekrementaci hodnoty na vrcholu zásobníku i v registru (**INR, DCR**).

Instrukce **ADD, SUB, INR** bez operandu a **DCR** bez operandu pracují s příznaky S0.0 až S0.3 podle výsledků provedené aritmetické operace, příznaky S0.4 až S0.7 vždy nulují. Výsledek aritmetické operace je dosažitelný na vrcholu zásobníku. Instrukce **DCR** s operandem nastavuje příznak S0.0.

V centrálních jednotkách řady B a D pro běžné operace sčítání a odčítání upřednostňujeme instrukce **ADX** a **SUX** před instrukcemi **ADD** a **SUB**. Instrukce **ADX** a **SUX** nenastavují příznaky v registru S0 a jsou tedy rychlejší.

##### Příklad 4.1.1

Proveďme operaci  $a = b - c$ , přenos zanedbejme.

```
B04101.mos
#reg word va, vb, vc ;možné šířky byte a long beze změny programu
;
P 0
 LD vb
 SUX vc
 WR va
E 0
```

##### Příklad 4.1.2

Proveďme operaci  $a = b - c$ , kde proměnné  $a, b$  mají šířku word, proměnná  $c$  má šířku byte.

```
B04102wb.mos
#reg word va, vb
#reg byte vc
;
P 0
 LD vb
 SUX vc
 WR va
E 0
```

Šířku long nelze jednoduše kombinovat s jinými šířkami z důvodu dvojnásobné velikosti používaných vrstev na zásobníku. Pokud bychom zadání upravili tak, že proměnné  $a, b$  mají šířku long, proměnná  $c$  má šířku word, musíme upravit program takto:

```
B041021w.mos
#reg long va, vb
#reg word vc
;
P 0
 LD vb
 LD 0 ;roztažení proměnné c na šířku long
 LD vc
 SUL
 WR va
E 0
```

Tuto konstrukci však nedoporučujeme používat, protože není jednoduše přenositelná do centrálních jednotek se zásobníkem šířky 32 bitů a takto napsaný program bychom v budoucnu při přechodu na nový typ centrální jednotky museli opravovat. Nejčistším způsobem je zajistit, aby všechny proměnné měly stejnou šířku.

### Příklad 4.1.3

Proveďme operaci  $a = b + c$ , kde proměnná  $a$  má šířku word, proměnné  $b, c$  mají šířku byte.

```
B04103wb.mos
#reg word va
#reg byte vb, vc
;
P 0
 LD vb
 ADX vc
 WR va
E 0
```

Šířku long nelze jednoduše kombinovat s jinými šířkami z důvodu dvojnásobné velikosti používaných vrstev na zásobníku. Pokud bychom zadání upravili tak, že proměnná  $a$  má šířku long, proměnné  $b, c$  mají šířku word, musíme upravit program takto:

```
B041031w.mos
#reg long va
#reg word vb, vc
;
P 0
 LD 0 ;roztažení proměnné b na šířku long
 LD vb
 LD 0 ;roztažení proměnné c na šířku long
 LD vc
 ADL
 WR va
E 0
```

Tuto konstrukci však nedoporučujeme používat, protože není jednoduše přenositelná do centrálních jednotek se zásobníkem šířky 32 bitů a takto napsaný program bychom v budoucnu při přechodu na nový typ centrální jednotky museli opravovat. Nejčistším způsobem je zajistit, aby všechny proměnné měly stejnou šířku.

### Příklad 4.1.4

Obsah proměnné *cislo* zvětšeme o konstantu 13 541.



```
B04104w.mos
#reg word cislo
;
P 0
 LD cislo
 ADD 13541
 WR cislo
E 0
```

```
B04104l.mos
#reg long cislo
;
P 0
 LD cislo
 ADL 13541
 WR cislo
E 0
```

#### **Příklad 4.1.5**

Obsah proměnné *cislo* šířky byte zvětšeme o konstantu 115.

```
B04105.mos
#reg byte cislo
;
P 0
 LD cislo
 ADD 115
 WR cislo
E 0
```

#### **Příklad 4.1.6**

Obsah proměnné *cislo* zvětšeme o konstantu 1.

```
B04106.mos
#reg byte cislo
;
P 0
 INR cislo
E 0
```

#### **Příklad 4.1.7**

Nad obsahem proměnné *citac* realizujeme čítač, počítající minuty od začátku procesu nebo od posledního vynulování či přepsání stavu. Systémová proměnná S20.4 se mění při každé náběžné hraně časové jednotky s periodou 1 minuta. Stačí počítat tyto impulzy.

```
B04107.mos
#reg word citac
;
P 0
 LD %S20.4
 WR %S0.3 ;nastavení přenosu CI
 LD citac
 ADD 0 ;inkrementace o CI
 WR citac
E 0
```

nebo

```
P 0
 LD 1
 AND %S20.4
 ADX citac
 WR citac
E 0
```

V prvním případě jsme změnu časové jednotky nastavili do vstupního přenosu CI a přičtením hodnoty 0 ji připočetli k obsahu čítače. Ve druhém případě jsme hodnotu časového impulzu proměnili na číslo 0 nebo 1 a pak přičetli k obsahu čítače. V těchto jednoduchých příkladech jsme se obešli bez instrukce čítačů, zejména díky existenci změnových impulzů časových jednotek.

### **Příklad 4.1.8**

Nad obsahem proměnné *citac* realizujeme vratný čítač, který po 1 s snižuje svůj stav. Můžeme postupovat obdobně, jako v předchozím příkladu. Zdroj časových impulzů je nyní v S20.2.

```
B04108a.mos
#reg word citac
;
P 0
 LD %S20.2
 WR %S0.3 ;nastavení přenosu CI
 LD citac
 SUB 0 ;dekrementace o CI
 WR citac
E 0
```

Jednodušší však bude postup:

```
B04108b.mos
#reg word citac
;
P 0
 LD %S20.2 ;na vrchol zásobníku se zapíše 0 nebo 65 535
 ADX citac
 WR citac
E 0
```

Číslo 65 535 je dvojkovým doplňkem čísla 1 a jeho přičítání je rovnocenné přičítání -1, tedy odečtení 1. Chceme-li realizovat čítač šířky long, musíme naplnit celou dvojrstvu A01.

```
B04108c.mos
#reg long citac
;
P 0
 LD %S20.2 ;na vrchol zásobníku se zapíše 0
 LD %S20.2 ;nebo $FFFFFFFF
 ADX citac
 WR citac
E 0
```

### **Příklad 4.1.9**

Realizujeme 5x prováděný cyklus.

Pro realizaci různých čítačů cyklů jsou vhodné instrukce **INR** a **DCR**.

*B04109a.mos*

```
#reg byte pocitadlo
;
P 0
 LD 0
 WR pocitadlo
cyklus: ;začátek cyklu
:
 INR pocitadlo
 LD pocitadlo
 CMP 5
 JNZ cyklus ;test počtu opakování
E 0 ;konec cyklu, pocitadlo = 5
```

Postup je sice v pořádku, ale výhodnější je nastavit počítadlo cyklů na žádaný počet průchodů a odčítat až k nule. Instrukce **DCR** při vynulování registru nastaví příznak S0.0 (ZR).

*B04109b.mos*

```
#reg byte pocitadlo
;
P 0
 LD 5
 WR pocitadlo
cyklus: ;začátek cyklu
:
 DCR pocitadlo
 JNZ cyklus ;test počtu opakování
E 0 ;konec cyklu, pocitadlo = 5
```

## 4.2. Násobení a dělení (MUL, MUD, DIV, DID)

Instrukce **MUL** násobí hodnoty šířky byte (do hodnoty 255) a výsledek má šířku word (s rezervou postačuje pro maximální hodnotu výsledku, tj. pro 65 025). Instrukce není určena pro realizaci složitých výpočetních algoritmů, ale pouze k příležitostným výpočtům a pomocným operacím.

Instrukce **MUD** má dvojnásobný rozsah, násobí hodnoty šířky word (do hodnoty 65 535) a výsledek má šířku long (maximální hodnota výsledku 4 294 836 225).

Instrukce **DIV** (word / byte = byte) je určena především pro jednoduché přepočty a pomocné operace. Není určena k realizaci rozsáhlejších výpočtů. Je mnoho problémů, které lze převést na operaci dělení a účinně provádět instrukci **DIV** (přepočty „mod n“, posuvy vpravo, oddělení skupiny bitů, přepočty časových údajů).

Pro výpočty s většími rozsahy je určena instrukce **DID** (long / word = word).

Je třeba zajistit, aby dělitel byl nenulový. Systém sice tuto situaci rozpozná, ošetří jako nezávažnou chybu a jako výsledek uloží maximální hodnotu daného formátu (interpretace symbolu  $\infty$  - nekonečno, přetečení, chybná hodnota). Představy uživatele a systému na tuto chybu se však mohou různit. Proto je lepší, pokud uživatel nulovost dělitele předem vyloučí nebo se s ní vyrovná podle svých představ.

Úlohy s většími rozsahy operandů a výsledku je třeba řešit ve formátu float (viz kap.11).

## Příklad 4.2.1

Do proměnné *a* uložíme součin proměnných *b* a *c*.

```
B04201a.mos
#reg word va
#reg byte vb, vc
;
P 0
 LD vb
 MUL vc
 WR va
E 0
```

```
B04201b.mos
#reg long va
#reg word vb, vc
;
P 0
 LD vb
 MUD vc
 WR va
E 0
```

## Příklad 4.2.2

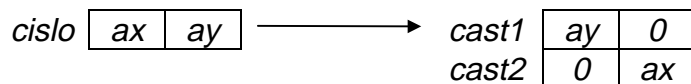
Do proměnné *a* uložíme součin dolních čtyř bitů proměnných šířky byte *b* a *c*.

```
B04202.mos
#reg byte va, vb, vc
;
P 0
 LD vb
 AND %1111
 LD vc
 AND %1111
 MUL
 WR va
E 0
```

## Příklad 4.2.3

Pokud je jeden z činitelů dvojková mocnina (např. čísla 2, 4, 8, 16, 32, 64, 128), pak je násobení rovnocenné s posunem vlevo o tolik míst, jaký byl mocnitel (tj. o 1, 2, 3, 4, 5, 6, 7). Posun bytu se uskutečňuje v rámci celého vrcholu zásobníku A0, zleva i zprava od posunutého čísla jsou nuly.

Požadujeme například rozdělení bytu *cislo* na dvě části a uložíme je do *cast1* a *cast2* dle schématu:



Pokud obě skupiny *ax*, *ay* mají šířku 4 bity, pak stačí prostý postup:

```
B04203.mos
#reg byte cislo, cast1, cast2
;
P 0
 LD cislo
 MUL 16 ;rovnocenné s MUL %00010000
```

```

WR word cast1 ;musí být dodržena deklarace
E 0 ;cast1, cast2 v daném pořadí

```

Pokud bychom požadovali nesoúměrné rozdělení, např.  $ax = 3$  bity,  $ay = 5$  bitů, stačí změnit instrukci na **MUL 8**.

#### Příklad 4.2.4

Přepočteme údaje systémových registrů S7, S8, S9 (čítače min., hod., den v týdnu) na počet minut od začátku týdne a uložíme v proměnné *cas*.

```

B04204.mos
#reg word cas
;
P 0
 LD %S9 ;dny v týdnu
 DCR ;přepočet pořadového čísla na index
 MUL 24 ;přepočet na hodiny
 ADX %S8 ;hodiny
 MUL 60 ;přepočet na minuty
 ADX %S7 ;minuty
 WR cas
E 0

```

#### Příklad 4.2.5

Zařídíme, aby se na bitu *vystup* každých 16 s objevil jedničkový pulz trvající 1 s.

Úlohu lze řešit mnoha způsoby. Budeme ji řešit použitím čítače mod 16. V příkladech 4.1.7, 4.1.8 byly uvedeny možnosti jednoduché realizace čítačů.

```

B04205a.mos
#reg bit vystup
#reg byte citac
;
P 0
 LD 1
 AND %S20.2
 ADX citac ;inkrementace čítače
 DIV 16
 AND $FF00 ;vynulování podílu
 SWP
 WR citac ;zbytek = stav čítače
 WRC vystup ;při nule nastavit výstup
E 0

```

Prvé tři instrukce podmíněně inkrementují obsah čítače, zbytek po dělení má význam zkorigovaného stavu čítače - stav 16 je zobrazen jako 0, hodnoty 16 jsou nezměněny. Každý stav čítače trvá do další změny, tedy 1 s. Pro nastavení bitu *vystup* je nejjednodušší vyhodnotit stav 0. Hodnotu podílu jsme v našem případě nevyužili.

Uvedený postup je obecný a mohl by realizovat libovolný čítač v modulu 2 až 255 - stačí změnit operand instrukce **DIV**. Místo předem zadaného modulu (zadaného jako bezprostřední operand) můžeme využít i modul dynamicky se měnící v registru R nebo na vstupech X (např. proměnný kmitočet blikání nebo akustického signálu, proměnné dávky, proměnné časové intervaly) - stačí uvést instrukce **DIV predvolba**, kde proměnná *predvolba* bude deklarovaná v příslušné oblasti.

V našem případě a ve všech případech, kdy je modul dvojkovou mocninou, lze použít jednodušší postup, např.:

```

B04205b.mos
#reg bit vystup
#reg byte citac
;
P 0
 LD 1
 AND %S20.2
 ADX citac ;inkrementace čítače
 AND 15 ;maska
 WR citac ;zbytek = stav čítače
 WRC vystup ;při nule nastavit výstup
E 0

```

V těchto případech lze použít i vratný čítač (u obecného modulu nikoliv!):

```

B04205c.mos
#reg bit vystup
#reg byte citac
;
P 0
 LD %S20.2 ;hodnota 0 nebo -1 (65 535)
 ADX citac ;dekrementace čítače
 AND 15 ;maska
 WR citac ;zbytek = stav čítače
 WRC vystup ;při nule nastavit výstup
E 0

```

### Příklad 4.2.6

Předchozí případ upravme v tom smyslu, že budeme požadovat interval 20 s a navíc v tomto intervalu ještě nastavovat jedničkový impuls na bitu *impulz* po dobu jednoho cyklu programu:

```

B04206.mos
#reg bit vystup, impulz
#reg byte pomoc, citac
;
P 0
 LD 1
 AND %S20.2
 LD citac
 WRC vystup ;při nule nastavit výstup
 ADD
 DIV 20
 WR word citac-1 ;pomoc = podíl, citac = čítač
 AND 255
 WR impulz
E 0

```

Podíl má v našem případě význam přenosu a je jedničkový pouze v cyklu, kdy došlo k přetečení, a je tedy (jeho nenulovost) použit k nastavení bitu *impulz*. Bit *vystup* je nyní nastavován podle stavu čítače z minulého cyklu, pokud nevadí, že je o cyklus zpožděn - jinak by bylo nutné použít stejné zakončení, jako v předchozím případě. Byte *pomoc* slouží jako pomocný k možnosti uložení hodnoty čítače bez nutnosti provádět prohození bytů vrcholu zásobníku instrukcí **SWP**.

**Příklad 4.2.7**

Registr *citac* má význam čítače (např. výrobků, časových jednotek) a máme zjistit, zda dosáhl nebo překročil hodnotu 75. Tradičně bychom použili komparaci:

```
B04207a.mos
#reg byte citac
#reg bit vysledek
;
P 0
 LD citac
 GT 74
 WR vysledek
E 0
```

Vrchol A0 nese informaci právě (a pouze) o výsledku porovnání. Můžeme ale použít i dělení:

```
B04207b.mos
#reg byte citac
#reg bit vysledek
;
P 0
 LD citac
 DIV 75
 AND 255
 WR vysledek
E 0
```

Dolní byte vrcholu A0L (podíl) nese potřebnou informaci (A0L = 0 nedosáhl, A0L > 0 dosáhl nebo překročil), ale jeho číselný obsah navíc říká, kolikrát překročil stanovený rozsah. Horní byte (A0H - zbytek) může být využit, pokud by bylo potřebné obsah čítače přepočítat modulo 75 (pak nebude bezprostředně použita instrukce **AND** 255, která hodnotu zbytku maže).

**Příklad 4.2.8**

Systémový registr S7 má význam čítače času v minutách (mod 60). Máme realizovat cyklický algoritmus s periodou 5 minut. Budeme potřebovat pořadové číslo cyklů v rámci hodiny, počet minut (0 až 4) a počet sekund (0 až 299) od počátku cyklu.

```
B04208.mos
#reg byte cas[4]
;
P 0
 LD %S7
 DIV 5 ;kolikrátá pětiminuta (cyklus) v hodině
 WR word cas
 SWP ;cas[0] = počet cyklů, cas[1] = počet minut
 MUL 60
 ADX %S6
 WR word cas+2 ;cas[2], cas[3] = počet sek. od začátku cyklu
E 0
```

Počet minut v rámci hodiny jsme rozdělili na počet intervalů a pořadí minuty v intervalech, které násobením 60 a přičtením počtu sekund převedeme na počet sekund od začátku intervalu.

Obdobným postupem můžeme rozdělit hodinu, minutu, sekundu na čtvrtinu, třetinu, polovinu, den rozdělit na dopoledne - odpoledne, na směny, nebo jiné celistvé díly.

### Příklad 4.2.9

Oddělme horní a dolní polovinu bytu *cislo* tak, aby každá část zabírala dolní polovinu bytů *cast1* a *cast2* a horní byly nulové.

*B04209.mos*

```
#reg byte cislo, cast1, cast2
```

```
;
```

```
P 0
```

```
LD cislo
```

```
DIV 16
```

```
WR word cast1 ;cast1 a cast2 deklarovány za sebou
```

```
E 0
```

V registru *cast1* je uložena horní polovina bytu (podíl), v registru *cast2* dolní polovina (zbytek). Obdobně lze rozdělit byte na nesouměrné části pouhou volbou dělitele, který musí být dvojkovou mocninou (2 až 128).

Dělení dvojkovou mocninou lze chápat jako posun bytu vpravo (počet posuvů vpravo je roven mocnině). Posunutým údajem je dolní část výsledku (podíl), horní část (zbytek) má význam vysunutého obsahu.

### Příklad 4.2.10

Systémový registr SW18 má význam dvojkového časového údaje v desítkách sekund. Přepočteme jej na minuty a uložíme do registru *podil*. K vyřešení stačí vydělit jej číslem 6. Dále podle hodnoty zbytku zaokrouhleme výsledek (souměrně - 0 až 2 dolů, 3 až 5 nahoru).

*B04210.mos*

```
#reg word podil
```

```
;
```

```
P 0
```

```
LD 0 ;převod formátu na long
```

```
LD %SW18
```

```
DID 6
```

```
WR podil ;výsledek je maximálně word
```

```
POP 2 ;posun zbytku na vrchol zásobníku
```

```
ADD 3 ;6 : 2 = 3
```

```
DIV 6
```

```
AND 1
```

```
ADX podil
```

```
WR podil
```

```
E 0
```

Obecně lze souměrné zaokrouhlování provádět tak, že k mezivýsledku přičteme polovinu jednotky ( $\text{modul} / 2$ ) a výsledek znovu přepočteme v původním modulu.

## 4.3. Porovnání hodnot a limitní funkce (CMP, CML, EQ, LT, GT)

Instrukce porovnání (**CMP**, **CML**) provedou porovnání dvou hodnot a podle výsledku nastaví hodnoty příznaků v S0. Zásobník zůstává beze změn.

Instrukce porovnání s testem (**EQ**, **LT**, **GT**) provedou porovnání dvou hodnot, podle výsledku nastaví hodnoty příznaků v S0 a výsledek testu zapíšou na vrchol zásobníku.



### Příklad 4.3.1

Nastavte bit *vysledek*, pokud *cislo1*  $\neq$  *cislo2*.

Zde nezáleží na pořadí operandů, můžeme rovnocenně psát:

*B04301.mos*

#reg word *cislo1*, *cislo2*

#reg bit *vysledek*

;

|     |     |                 |      |     |     |                 |
|-----|-----|-----------------|------|-----|-----|-----------------|
| P 0 |     |                 | nebo | P 0 |     |                 |
|     | LD  | <i>cislo1</i>   |      |     | LD  | <i>cislo2</i>   |
|     | EQ  | <i>cislo2</i>   |      |     | EQ  | <i>cislo1</i>   |
|     | WRC | <i>vysledek</i> |      |     | WRC | <i>vysledek</i> |
| E 0 |     |                 |      | E 0 |     |                 |

Místo instrukce **EQ** lze použít **SUB** a testovat nenulovost výsledku instrukcí **WR**.

```

P 0
 LD cislo1
 SUB cislo2
 WR vysledek
E 0

```

K určení neshody lze použít i instrukci **XOR**.

```

P 0
 LD cislo1
 XOR cislo2
 WR vysledek
E 0

```

Oproti instrukci **EQ** nebo **SUB** však porovnání s instrukcí **XOR** poskytuje navíc informaci o místu neshody, tam kde jsou jedničky ve výsledku, tam jsou odlišnosti.

### Příklad 4.3.2

Nastavme bit *vysledek*, pokud *cislo* = 23 115.

Zde vyhoví typický postup.

*B04302.mos*

#reg word *cislo*

#reg bit *vysledek*

;

```

P 0
 LD cislo
 EQ 23115
 WR vysledek
E 0

```

### Příklad 4.3.3

Nastavme bit *vysledek0*, pokud *cislo* = 23 115, bit *vysledek1*, pokud *cislo* < 23 115 a bit *vysledek2*, pokud *cislo* ≤ 23 115.

Při tradičním postupu programujeme postupně jednotlivé úkoly.

*B04303a.mos*

#reg word *cislo*

#reg bit *vysledek0*, *vysledek1*, *vysledek2*

;

```

P 0
 LD cislo

```

```
EQ 23115
WR vysledek0
LD cislo
LT 23115
WR vysledek1
OR
WR vysledek2
```

E 0

Po instrukci **LT** je na A0 výsledek ostré nerovnosti a na A1 výsledek instrukce **EQ**. Logickým sečtením získáme výsledek neostré nerovnosti:

$$(\leq) = (<) \text{ OR } (=)$$

V našem případě by stačila jediná instrukce porovnání s využitím informací v příznakovém registru S0, např.

```
B04303b.mos
#reg word cislo
#reg bit vysledek0, vysledek1, vysledek2
;
P 0
 LD cislo
 EQ 23115
 WR vysledek0
 LD %S0.1
 WR vysledek1
 LD %S0.2
 WR vysledek2
```

E 0

V našem případě mají bity *vysledek0*, *vysledek1*, *vysledek2* stejný význam jako bity v S0. Pokud budou tyto bity součástí jednoho bytu *vysledek*, stačí obsah S0 přesunout najednou.

```
B04303c.mos
#reg word cislo
#reg byte vysledek
;
P 0
 LD cislo
 CMP 23115
 LD %S0
 WR vysledek
```

E 0

```
B04303cl.mos
#reg long cislo
#reg byte vysledek
;
P 0
 LD cislo
 CML 23115
 LD %S0
 WR vysledek
```

E 0

### Příklad 4.3.4

Nastavme bit *vysledek*, pokud *cislo*  $\geq 15\,734$ . Tradičním řešením je postup:

```
B04304.mos
#reg word cislo
#reg bit vysledek
;
P 0
 LD cislo
 GT 15734
 OR %S0.0
 WR vysledek
E 0
```

nebo při změně pořadí operandů:

```
P 0
 LD 15734
 LT cislo
 OR %S0.0
 WR vysledek
E 0
```

Pokud si uvědomíme, že negace ( $\geq$ ) je ( $<$ ), tedy ( $\geq$ ) je negace ( $<$ ), můžeme psát:

```
P 0
 LD cislo
 LT 15734
 WRC vysledek
E 0
```

Neostrou nerovnost na ostrou můžeme změnit také pouhým posunutím meze. Původní nerovnost  $cislo \geq 15\,734$  je rovnocenná nerovnosti  $cislo > 15\,733$ , přípustný je tedy i postup:

```
P 0
 LD cislo
 GT 15733
 WR vysledek
E 0
```

#### Příklad 4.3.5

Nastavme bit *vysledek*, pokud *cislo* = 1. Je možné tradiční porovnání:

```
B04305.mos
#reg word cislo
#reg bit vysledek
;
P 0
 LD cislo
 EQ 1
 WR vysledek
E 0
```

Stačí však dekrementovat obsah a testovat nulovost:

```
P 0
 LD cislo
 DCR
 WRC vysledek
E 0
```

### Příklad 4.3.6

Nastavme bit *vysledek* na log.1, pokud *cislo* = 0. Zde již není nutná žádná komparace, stačí testovat nulovost.

```
B04306.mos
#reg word cislo
#reg bit vysledek
;
P 0
 LD cislo
 WRC vysledek
E 0
```

### Příklad 4.3.7

Nastavme bit *vysledek* na log.1, pokud *cislo* > 154.

```
B04307.mos
#reg byte cislo
#reg bit vysledek
;
P 0
 LD cislo
 GT 154
 WR vysledek
E 0
```

Zde je rozdíl pouze v tom, že instrukce **LD** pracuje s bytovým operandem.

### Příklad 4.3.8

Nastavme bit *vysledek* na log.1, pokud *cislo1* = *cislo2*.

```
B04308a.mos
#reg byte cislo1, cislo2
#reg bit vysledek
;
P 0
 LD cislo1
 LD cislo2
 EQ
 WR vysledek
E 0
```

Zde nelze použít instrukci **EQ** s vyjádřeným operandem, protože pracuje s operandem šířky word. Nebylo by možné jednoduše oddělit programované byty od nežádoucích „doplňkových bytů“. Zde bytovými instrukcemi **LD** uložíme potřebné byty do zásobníku doplněné na šířku word nulami.

Při testování rovnosti může být místo instrukce **EQ** použita instrukce **SUB** a při ukládání bitovou instrukcí **WRC** automaticky testujeme nulovost rozdílu.

```
B04308b.mos
#reg byte cislo1, cislo2
#reg bit vysledek
;
P 0
 LD cislo1
 LD cislo2
 SUB
```

```
WRC vysledek
E 0
```

Logické operace s vyjádřeným operandem však pracují i ve formátu byte, takže zde můžeme použít postup:

```
B04308c.mos
#reg byte cislo1, cislo2
#reg bit vysledek
;
P 0
 LD cislo1
 XOR cislo2
 WRC vysledek
E 0
```

Tento postup je pro dané zadání nejúspornější. Navíc umožňuje najít místa neshody (na pozicích jedniček výsledku se operandy různí).

#### Příklad 4.3.9

Nastavme bit *vysledek* na log.1, pokud je splněna dvojitá nerovnost  $19 \leq cislo \leq 34$  (*cislo* je uvnitř uzavřeného intervalu). Splněny musí být současně obě dílčí nerovnosti (tedy AND obou dílčích výsledků).

Bez velkého uvažování můžeme sestavit tradiční postup:

```
B04309.mos
#reg byte cislo
#reg bit vysledek
;
P 0
 LD cislo
 GT 19
 OR %S0.0
 LD cislo
 LT 34
 OR %S0.0
 AND
 WR vysledek
E 0
```

Pokud posuneme meze, můžeme stejnou podmínku zapsat ostrými nerovnostmi  $18 < cislo < 35$  a realizovat úspornějším postupem:

```
P 0
 LD cislo
 GT 18
 LD cislo
 LT 35
 AND
 WR vysledek
E 0
```

Lze ovšem použít ještě úspornější postup:

```
P 0
 LD cislo
 SUB 19
 LT 16 ;16 = 35 - 19
 WR vysledek
E 0
```

Tento postup je zjevně nejúspornější a nejrychlejší. Odečtením původní dolní meze (19) se celý testovaný interval posune k nulové mezi:  $0 \leq \text{cislo} \leq 15$ , což je rovnocenné jediné nerovnosti  $\text{cislo} \leq 15$  nebo  $\text{cislo} < 16$ . Hodnoty, které původně ležely pod dolní mezí (0 až 18) se nyní posunuly pod hranici rozsahu, tedy na 65 517 až 65 535.

### Příklad 4.3.10

Při určité události (např. při startu topení) byl do proměnné *cas* okopírován časový údaj ze systémového registru SW16 (binární čítač času v sekundách). Po uplynutí 550 s je třeba nastavit bit *akce* na log.1 (do té doby byl nulový). Z různých možností vybíráme postup, který je jednoduchý a spolehlivý.

```
B04310.mos
#reg word cas, udalost
#reg bit akce
;
P 0
 LD %SW16 ;současný čas
 SUX cas ;počáteční čas
 WR udalost ;čas od události
 GT 549 ;nebo LT 550
 WR akce ; WRC akce
E 0
```

Nejprve je od aktuálního časového údaje (SW16) odečten počáteční čas z proměnné *cas* a výsledek má význam relativního času vůči počáteční události (doba zapnutí topení). Odložení tohoto údaje do proměnné *udalost* nebylo požadováno, ale v praktických případech se k tomuto údaji budeme vícekrát vracet. Dále již jen porovnáváme se zadanou mezí (posunutou tak, aby již po 550 s byl nastaven jedničkový výstup *akce*).

Je užitečné si uvědomit, že údaje časoměrných registrů se cyklicky mění od 0 k maximální hodnotě (255 nebo 65 535), a pak znova od nuly. Protože počáteční událost je obvykle nezávislá na stavu těchto registrů, nelze vyloučit situaci, kdy údaj počátečního času je vyšší než aktuální časový údaj. Náš postup se i v tomto případě zachová správně, pouze je třeba zajistit, aby doba, kdy sledujeme proces, nebyla delší než 65 534 s. Při nepozornosti lze však vytvořit postupy, které budou selhávat. Nedoporučuje se zasahovat během procesu do stavu časoměrných registrů!

### Příklad 4.3.11

Výsledkem instrukcí porovnání je logická proměnná, která je nastavena na všech šestnácti bitech A0. Byla by škoda ji využívat pouze k nastavování jedné bitové proměnné, jak jsme požadovali v dosavadních případech.

Stav A0 lze kombinovat s obsahy bytů a slov, ať již mají význam souboru bitových proměnných, kódových kombinací nebo číselných hodnot.

Požadujeme, aby obsah proměnné *vysledek* byl nulový tak dlouho, dokud je obsah  $\text{cislo} < 550$  (viz předchozí příklad), od té doby ( $\text{cislo} = 550$ ) má být obsah *vysledek* nastaven na 179.

```
B04311.mos
#reg word cislo
#reg byte vysledek
;
P 0
 LD cislo
 GT 549
 AND 179
 WR vysledek
```

E 0

Oproti variantám s větvením programu je toto řešení úspornější a přehlednější.

#### Příklad 4.3.12

Požadujeme realizovat omezovač maximální hodnoty obsahu proměnné *cislo*. Pokud je obsah  $cislo \leq 178$ , zůstává zachován, při překročení této meze je touto mezí nahrazen, tj.  $cislo = 178$ .

```
B04312.mos
#reg byte cislo
;
P 0
 LD cislo
 LT 178
 AND cislo
 LDC %S0.1
 AND 178
 OR
 WR cislo
```

E 0

Místo tohoto klasického postupu můžeme použít uživatelskou instrukci **LIM**.

#### 4.4. Použití znaménka a dvojkového doplňku

Dosud jsme se v aritmetických operacích nezabývali zobrazením znaménka a předpokládali jsme, že všechny číselné hodnoty jsou celá kladná (nezáporná) čísla.

Existuje několik způsobů zobrazení znaménka. Pro uživatele by byl patrně nej názornější způsob „znaménko a hodnota“, který je současně málo vhodný pro zpracování v systému (obzvláště pokud převažují instrukce sčítání a odčítání). Spočívá v tom, že znaménko je uloženo odděleně od číselné hodnoty. Obvykle se používá přiřazení:  $\log.0 = +$ ,  $\log.1 = -$  (stačí 1 bit, může však být zobrazeno nadbytečně). Pokud se uživatel rozhodne pro tento systém, musí si vše vyřešit a obsloužit sám (systém tento způsob zobrazení nepodporuje).

Naproti tomu, práce s doplňkovými kódy vyplývá přímo z mechanismu vykonávání aritmetických instrukcí.

Pro jednoduchost výkladu předpokládejme, že zpracováváme data v šíři jednoho bytu. Předpokládejme, že budeme od nuly odčítat postupně čísla 1, 2, ..., 255. Budeme dostávat výsledky:

|               |           |
|---------------|-----------|
| 0 – 1 = 255   | %11111111 |
| 0 – 2 = 254   | %11111110 |
| 0 – 3 = 253   | %11111101 |
| 0 – 4 = 252   | %11111100 |
| .....         |           |
| 0 – 126 = 130 | %10000010 |
| 0 – 127 = 129 | %10000001 |
| 0 – 128 = 128 | %10000000 |
| 0 – 129 = 127 | %01111111 |
| 0 – 130 = 126 | %01111110 |
| .....         |           |
| 0 – 254 = 2   | %00000010 |
| 0 – 255 = 1   | %00000001 |

Normálně bychom tyto výsledky považovali za chybné. Zkusme však přijmout pravidla, která nám systém nabízí. Pak můžeme výsledek 255 považovat za  $-1$ , 254 za  $-2$ , atd. Pokud se omezíme na rozsah hodnot od  $-128$  do  $+127$ , pak podle nejvyššího bitu výsledku můžeme spolehlivě rozlišit znaménko:  $\log.0 = +$ ,  $\log.1 = -$ .

Popsaný způsob vytvoření a zobrazení záporného čísla nazýváme dvojkovým doplňkem nebo přesněji dvojkovým doplňkovým kódem. Tento způsob kódování záporných čísel dnes používají všechny vyšší jazyky, proměnné jsou zpravidla označovány jako signed (znaménkové) nebo unsigned (bez znaménka).

Dvojkový doplněk k zápornému číslu (ke dvojkovému doplňku) vytvoříme stejně, tedy opět odečtením doplňku od nuly, například:

$$-(-5) = 0 - 251 = +5$$

Stejně můžeme zavést dvojkový doplňkový kód pro libovolný rozsah čísla. Stačí, abychom šíři bitů, potřebnou pro zobrazení nejvyšší absolutní hodnoty čísla rozšířili o alespoň jeden bit navíc a při sčítání nebo odčítání s ním zacházeli jako s bitem výsledku. Pak hodnota tohoto bitu ponese informaci o znaménku výsledku.

Nadbytečných bitů může být i více (např. celý byte). Pokud nedojde k překročení maximální číselné hodnoty (přetečení), budou všechny tyto přídavné bity nastaveny shodně a kterýkoliv z nich může být využit jako znaménkový. V případě přetečení rozsahu budou hodnoty nižších bitů různé od hodnot na vyšších pozicích a podle této neshody lze indikovat přetečení, případně rekonstruovat správný výsledek - jako znaménkový by pak měl být využit nejvyšší z přídavných bitů.

V praxi postupujeme nejčastěji tak, že z rozsahu bitů, které máme k dispozici na zobrazení čísla (8, 16 nebo 32 bitů) vyhradíme nejvyšší bit jako znaménkový a zbytek pak zůstává pro zobrazení číselné hodnoty. Při šířce byte je číselný rozsah omezen na  $-128$  až  $+127$ , při šířce word se zvýší na  $-32\,768$  až  $+32\,767$  a při šířce long dosáhne  $-2\,147\,483\,648$  až  $+2\,147\,483\,647$ .

Formát float má zvláštní strukturu, která znaménko obsahuje také na nejvyšším bitu, nicméně výpočty je třeba provádět se speciálními instrukcemi (viz kap.11.). Dále se budeme věnovat pouze formátům v pevné řádové čárce (byte, word, long).

Za předpokladu, že hodnoty operandů ani výsledků nepřekročí uvedené rozsahy (nepřetečou), můžeme operace sčítání i odčítání nebo inkrementace a dekrementace provádět nezávisle na znaménku a hodnotě operandů a výsledek ponese vždy správnou informaci o znaménku a hodnotě - opět bude zobrazen ve dvojkovém doplňku.

Pokud je to potřebné, můžeme operaci odčítání nahradit přičtením menšitele, kterému jsme změnili znaménko - vytvořili jeho doplněk. V obou případech bude výsledek stejný, pouze přenosový bit CO (S0.1) bude při přičítání doplňku nastaven opačně než při odčítání původního menšitele.

Je třeba upozornit, že význam přenosových bitů CI a CO (a tedy i  $\geq$ ) tak, jak byly popsány, platí výhradně pro případ operací s kladnými čísly. Vztahy přenosových bitů CI a CO při kaskádování se nemění ani při práci s operandy ve dvojkovém doplňkovém kódu. Pouze je nutné si uvědomit, že v případech, kdy operaci odčítání nahrazujeme sčítáním doplňku menšitele, pak bity CI a CO mají opačný (negovaný) význam.

Kromě popsaneého dvojkového doplňku se někdy používá tzv. jedničkový doplněk. Lze jej získat odečtením převáděného čísla od nejvyššího zobrazitelného čísla na daném počtu bitů (u bytu od 255, u wordu od 65 535, u longu od 4 294 967 295), tedy od samých jedniček. Dvojkový doplněk z jedničkového získáme inkrementací. Počítání v jedničkovém kódu není příliš výhodné pro nutnost provádět kruhový přenos - po každé operaci sčítání nebo odčítání je třeba případný přenos znovu přičíst nebo odečíst od mezivýsledku. Zajímavou vlastností jedničkového doplňku je skutečnost, že jej získáme prostou negací



všech bitů převáděného čísla. S jeho pomocí pak lze dvojkový doplněk získat jako negaci všech bitů a následným přičtením jedničky (inkrementací).

Ještě uvedme, jak lze rozpoznat přetečení výsledku přes vyhrazený rozsah. Jako nejjednodušší prostředek lze doporučit určitou velkorysost při vyhrazování prostoru pro zobrazení čísel - např. při zpracování bytových údajů se vyplatí vyhradit pro ně rozsah word. Ukazuje se, že přílišná úzkostlivost a malicherná šetrnost se obvykle prodrazí (nejméně v programování, zejména komplikovanějším programem a prodloužením jeho doby odezvy).

Již bylo uvedeno, že pokud je více znaménkových bitů, pak neshoda jejich hodnot je příznakem „přetečení hodnoty do znaménka“. Za hodnotu znaménka je nejbezpečnější považovat nejvyšší ze znaménkových bitů, z hodnoty dolního poškozeného lze rekonstruovat správný obsah výsledku.

Pokud pracujeme s jediným znaménkovým bitem, pak v normálních situacích musí být přenos vstupující do znaménkového bitu shodný s přenosem vystupujícím ze znaménkového bitu. V případě neshody přenosů došlo k přetečení. Tuto situaci lze převést na vyhodnocení součtu mod 2 znaménkového bitu obou operandů, znaménkového bitu výsledku a výsledného výstupního přenosu (CO) vystupujícího ze znaménkového bitu. Pokud je tento součet nulový (byl sudý počet jedniček), je výsledek v rozsahu. Jedničková hodnota součtu (lichý počet jedniček) indikuje přetečení.

K násobení a dělení se znaménkem musíme použít speciální instrukce, protože jinak bychom museli pracovat s operandy v absolutní hodnotě a znaménko výsledku dopočítat zvlášť podle vztahů:

|                   |                   |
|-------------------|-------------------|
| $(+) * (+) = (+)$ | $(+) / (+) = (+)$ |
| $(+) * (-) = (-)$ | $(+) / (-) = (-)$ |
| $(-) * (+) = (-)$ | $(-) / (+) = (-)$ |
| $(-) * (-) = (+)$ | $(-) / (-) = (+)$ |

Výsledné znaménko je výsledkem funkce XOR znamének obou operandů.

Pokud použijeme instrukce pro znaménkovou aritmetiku pro centrální jednotky řady D a B dostupné jako uživatelské instrukce **MUDS**, **MUL32S**, **DIDS**, **DIV32S**, nemusíme tento problém řešit.

Další úskalí nastává v případě, že do zásobníku ukládáme ze zápisníku proměnnou, která má menší šířku než zápisník. Pokud totiž pro načtení proměnné šířky byte na vrchol zásobníku, který má šířku word, použijeme instrukci **LD**, dojde k tomu, že v dolním bytu je načtená proměnná a v horním bytu je 0, takže jsme přišli o informaci o znaménku hodnoty proměnné.

Pokud se těmto problémům chceme vyhnout, používáme pro znaménkové proměnné zásadně takovou šířku, se kterou na zásobníku počítáme, tj. word nebo long.

#### Příklad 4.4.1

Vynásobíme proměnné *a* a *b* šířky word se znaménkem. Výsledek uložíme do proměnné *c* šířky long se znaménkem.

Při využití instrukčního souboru v centrálních jednotkách řady D a B bychom museli postupovat takto:

```

B04401a.mos
#reg word va, vb
#reg long vc
;
P 0
 LD va

```

```

LD va.15 ;znaménko a
WR %S0.3 ;CI
XOR
ADD 0 ;a v doplňku
LD vb
LD vb.15 ;znaménko b
WR %S0.3 ;CI
XOR
ADD 0 ;b v doplňku
MUD
 ;a . b = c
WR vc
LD va.15
XOR vb.15 ;výpočet znaménka c
JMC konec
LDC vc ;minus
ADL 1 ;c v doplňku
WR vc

```

konec:

E 0

Problém vyřešíme nejlépe uživatelskou instrukcí.

```

B04401b.mos
#reg word va, vb
#reg long vc
#usi umuds = muds
#def MUDS usi umuds
;
P 0
 LD va
 LD vb
 MUDS
 WR vc
E 0

```

### 4.5. Příklady výpočtů v pevné řádové čárce

Dosud jsme počítali pouze s celými čísly v rozsazích daných použitými formáty proměnných. Pokud je požadavek počítat s čísly reálnými (tedy s čísly, která mají platné číslice za desetinnou čárkou), lze použít dva přístupy.

Prvním přístupem je převod čísel na formát float používající pohyblivou řádovou čárku (tj. číslo vyjádřené mantisou a exponentem, pevný je počet platných číslic). Výhodou tohoto přístupu je velký rozsah při dostatečné přesnosti, možnost složitých matematických výpočtů. Nevýhodou je časová náročnost výpočtů a převodů formátu float na ostatní formáty a naopak. Podrobněji se formátem float budeme zabývat v kap.11.

Druhým přístupem je použití tzv. pevné řádové čárky. Znamená to, že je předem určen počet desetinných míst, se kterými se počítá. Výhodou tohoto přístupu je možnost použití běžných aritmetických instrukcí, tedy rychlost výpočtů a převodů. Nevýhodou je omezený rozsah.

V uživatelském programu budeme pevnou řádovou čárku interpretovat tak, že jednotkou se stane místo prvního řádu nejnižší uvažovaný řád za desetinnou čárkou. Zadáme-li, že budeme počítat s přesností na 3 desetinná místa, budeme všechna čísla zadávat v tisícinách (budou tedy 1000 x větší). Je to vlastně totéž, jako kdybychom při měření vzdálenosti přešli z měření v metrech na milimetry.

Tab.4.1 Rozsahy v pevné řádové čárce

| Počet desetinných míst | Rozsah hodnot                  |                                   |                                                  |
|------------------------|--------------------------------|-----------------------------------|--------------------------------------------------|
|                        | byte                           | word                              | long                                             |
| 1 0,1                  | 0 až 25,5<br>-12,8 až +12,7    | 0 až 6553,5<br>-3276,8 až +3276,7 | 0 až 429496729,5<br>-214748364,8 až +214748364,7 |
| 2 0,01                 | 0 až 2,55<br>-1,28 až +1,27    | 0 až 655,35<br>-327,68 až +327,67 | 0 až 42949672,95<br>-21474836,48 až +21474836,47 |
| 3 0,001                | 0 až 0,255<br>-0,128 až +0,127 | 0 až 65,535<br>-32,768 až +32,767 | 0 až 4294967,295<br>-2147483,648 až +2147483,647 |
| atd.                   |                                |                                   |                                                  |

Pro proměnné šířky byte, word a long lze tedy použít rozsahy uvedené v tab.4.1. Větší rozsahy lze řešit kaskádováním instrukcí.

Sčítat a odčítat můžeme pouze čísla se stejným počtem desetinných míst. Tentýž počet desetinných míst bude mít i výsledek. Např. součet

$$0,01 + 0,1 = 0,11$$

realizujeme při 2 desetinných místech takto:

$$1 + 10 = 11$$

Násobit a dělit můžeme čísla s různým počtem desetinných míst. Při násobení dojde k sečtení počtu desetinných míst obou činitelů. Pokud např. pracujeme s dvěma desetinnými místy, pak výpočet

$$1 * 1 = 1$$

je interpretován jako

$$0,01 * 0,01 = 0,0001 \quad (2 + 2 = 4)$$

Výsledkem je číslo s počtem čtyř desetinných míst. U dělení je situace opačná, počty desetinných míst se odčítají.

$$0,0001 : 0,01 = 0,01 \quad (4 - 2 = 2)$$

$$0,01 : 0,01 = 1 \quad (2 - 2 = 0)$$

$$0,01 : 0,0001 = 100 \quad (2 - 4 = -2)$$

Na tyto změny je třeba dávat bedlivý pozor.

#### Příklad 4.5.1

Proveďme výpočet výrazu  $a = \frac{b+c \cdot d}{e+f}$

Všechny vstupní proměnné mají šířku word a mají dvě desetinná místa. Proměnná a má šířku long a má také dvě desetinná místa.

*B04501.mos*

#reg long va

#reg word vb, vc, vd, ve, vf

;

P 0

LD vb

MUD 100

;převod na long, čtyři desetinná místa

LD vc

MUD vd

;výsledek je long, čtyři desetinná místa

ADL

;b + c.d

LD ve

|  |     |    |                                        |
|--|-----|----|----------------------------------------|
|  | ADX | vf | ;e + f (word)                          |
|  | DID |    | ;long / word (4 - 2 desetinná místa)   |
|  | WR  | va | ;výsledek je long, dvě desetinná místa |

E 0

### 4.6. Převody číselných soustav (BIN, BIL, BCD, BCL)

Drtivá většina informací je v PLC uložena v binárním, tedy dvojkovém kódu. Pro snadnější interpretaci těchto čísel se používá tzv. hexadecimální, tj. šestnáctková číselná soustava, jejíž jedna číslice obsazuje 4 bity, tedy právě půl bytu. Jeden byte je tak interpretován dvěma číslicemi, z nichž každá může nabývat hodnot 0 - 9, A - F (jednoznaková interpretace hodnot 0 - 15). Pokud ale chceme hodnotu proměnné zobrazit na nějakém zobrazovači, zpravidla požadujeme zobrazení v běžně používané desítkové soustavě. Pro tu se používá kód BCD (binárně dekadický kód), kdy čtveřice bitů nabývá hodnot 0 - 9. Jeden byte má pak rozsah 0 - 99. S takovými čísly se prakticky nedá počítat (nebo jen velmi těžko), ale pro zobrazení nebo naopak vstupování hodnot z klávesnice jsou nejvhodnější.

Pro převod čísel z binární soustavy do kódu BCD jsou určeny instrukce **BCD** a **BCL** a pro převod opačný instrukce **BIN** a **BIL**.

#### Příklad 4.6.1

Na *vstupy* jsou připojeny vývody dvou kotoučových desítkových přepínačů („contravesů“) v kódu BCD - nižší váhy na nižších bitech. Tento údaj je třeba převést na dvojkový a uložit do proměnné *cislo*, aby s ní bylo možné provádět aritmetické operace a porovnávání. Stačí jednoduchý postup:

```

B04601.mos
#reg byte vstupy, cislo
;
P 0
 LD vstupy
 BIN
 WR cislo
E 0

```

#### Příklad 4.6.2

Proměnná *obraz* je zobrazována na operačním panelu. Je požadováno zpracovat dvojkový obsah proměnné *cislo* tak, aby byl správně zobrazován jeho obsah, pokud je v rozsahu 0 - 99, jinak je třeba nastavit bit *pretečení*.

```

B04602.mos
#reg word obraz
#reg byte cislo
#reg bit preteceeni
;
P 0
 LD cislo
 BCD ;převod na BCD
 WR obraz
 AND $FF00
 WR preteceeni ;příznak přetečení
 LD obraz
 SWP ;přesun čísla na horní byte
 BAS ;převod na ASCII

```

```

 WR obraz ;první dvě číslice (další dvě nejsou využity)
E 0

```

### Příklad 4.6.3

Úlohu 4.6.2 změníme tím, že místo signálky, hlásící přetečení výsledku přes dvě desítkové číslice, přidáme třetí pozici na operačním panelu pro indikaci proměnné *cislo* (rozsah 0 až 255).

```

B04603.mos
#reg byte obraz[3]
#reg byte cislo
;
P 0
 LD cislo
 BCD ;převod na BCD
 BAS ;převod na ASCII
 SWP ;1. pozice je navíc
 WR obraz ;1. číslice
 SWL
 WR word obraz+1 ;2. a 3. číslice
E 0

```

### Příklad 4.6.4

Nad obsahem proměnné *citac* formátu word je realizován čítač (instrukcí **CTU**, **CTD**, **CNT** nebo aritmetickými instrukcemi). Přepočtíme jeho obsah tak, aby nevybočil z rozsahu 0 - 9999.

```

B04604.mos
#reg word citac
;
P 0
 LD citac
 BCD
 BIN
 WR citac
E 0

```

V případě potřeby lze v S0.4 až S0.6 detekovat překročení meze. Obdobně lze omezit rozsah hodnoty na 0 - 9, 0 - 99 nebo 0 - 999, případně na jiné hodnoty. Je však třeba zvážit časovou náročnost těchto instrukcí.

## 5. OPERACE SE ZÁSObNÍKY

### 5.1. Posun zásobníku (POP)

Instrukce **POP** provádí posun zásobníku zpět o zvolený počet úrovní.

#### Příklad 5.1.1

Uložme obsah vrstvy A3 zásobníku do proměnné *cislo* beze změny obsahu zásobníku.

```
B05101.mos
#reg word cislo
;
P 0
 POP 3 ;posun obsahu A3 do A0
 WR cislo ;uložení obsahu
 POP -3 ;srovnání zásobníku do původního stavu
E 0
```

### 5.2. Operace s několika zásobníky (NXT, PRV, CHG, CHGS, LAC, WAC)

Instrukce **NXT**, **PRV**, **CHG**, **CHGS** provádějí aktivaci následujícího, předcházejícího nebo vybraného zásobníku včetně zálohování registrů S0 a S1 (kromě **CHG**).

Instrukce **LAC** a **WAC** umožňují čtení a zápis hodnoty z a na vrchol zvoleného zásobníku.

#### Příklad 5.2.1

Instrukce **CHG**, **CHGS**, **NXT** a **PRV** lze použít pro odložení rozpracovaného zásobníku při volání podprogramu pro nějaký dílčí výpočet, který způsobí nežádoucí přepsání současného obsahu zásobníku jinými hodnotami.

```
B05201.mos
;
P 0
 :
 CHGS 1 ;přepnutí na zásobník B,
 ;se zásobníkem A jsou odloženy i S0 a S1
 CAL podprogram ;zavolání podprogramu
 CHGS 0 ;návrat na zásobník A s původními S0 a S1
 :
E 0
```

Stejně můžeme použít i instrukce **NXT** a **PRV**. Jejich výhoda spočívá v tom, že umožňují různou úroveň „vnoření“ zásobníků, tzn. že je jedno, jestli těmito instrukcemi procházíme při aktivním zásobníku A nebo jiném. Jediné, co si musíme ohlídat, je počet vnoření, který je 8. Poté se nám začne přepisovat opět zásobník A. Tento přístup je vhodný obzvlášť pro vícenásobně volané podprogramy.

```
P 60
podprogram:
 :
 NXT ;přepnutí na následující zásobník v řadě,
 ;odloženy jsou i S0 a S1
```

```
CAL podprogram2 ;zavolání podprogramu 2
PRV ;návrat na původní zásobník s S0 a S1
:
RET
```

E 60

### Příklad 5.2.2

Potřebujeme realizovat nepřímé čtení a zápis do prvků pole.

Protože se nám tento problém bude v programu jistě opakovat vícekrát, vytvoříme si makroinstrukce LDMI a WRMI pro čtení a zápis prvků pole. Makroinstrukce LDMI načte hodnotu prvku homogenního pole na vrchol aktivního zásobníku, pohyb zásobníku je stejný jako u instrukce **LD**. Makroinstrukce WRMI zapíše hodnotu z vrcholu aktivního zásobníku do prvku homogenního pole, obsah zásobníku zůstane nezměněn.

```
B05202.mos
#def maxR 8191
#reg word pole1[10], index1
#reg word pole2[10], index2
;
#macro LDMI (pole, index)
 LD maxR
 LD index
 LTB pole
 SWL
 POP 1
 SWL
 POP 1
#endm
;
#macro WRMI (pole, index)
 WAC 7
 LD maxR
 LD index
 LAC 7
 WTB pole
 POP 3
#endm
;
P 0
 LDMI (pole1, index1) ;čtení prvku pole1
 :
 WRMI (pole2, index2) ;zápis prvku do pole2
E 0
```

Instrukce **WAC** a **LAC** v tomto případě slouží k odložení hodnoty bez nutnosti vytvoření pomocné proměnné v zápisníku.

### Příklad 5.2.3

Realizujme aritmetický výraz  $a = b + c \cdot d + e \cdot f$ , kde jednotlivé proměnné představují výsledky složitějších operací prováděných během cyklu uživatelského programu. Tyto výsledky můžeme odkládat pomocí instrukce **WAC** na nepoužívaný zásobník a pak zpracovat najednou.

B05203.mos

#reg long va

;

P 0

|     |    |                   |
|-----|----|-------------------|
| :   |    | ;výpočet b        |
| WAC | 7  | ;uložení          |
| :   |    | ;výpočet c        |
| WAC | 7  | ;uložení          |
| :   |    | ;výpočet d        |
| WAC | 7  | ;uložení          |
| :   |    | ;výpočet e        |
| WAC | 7  | ;uložení          |
| :   |    | ;výpočet f        |
| LAC | 7  | ;vyjmutí e        |
| MUD |    | ;e.f              |
| LAC | 7  | ;vyjmutí d        |
| LAC | 7  | ;vyjmutí c        |
| MUD |    | ;c.d              |
| ADL |    | ;c.d + e.f        |
| LD  | 0  | ;doplnění na long |
| LAC | 7  | ;vyjmutí b        |
| ADL |    | ;b + c.d + e.f    |
| WR  | va |                   |

E 0

Výraz lze samozřejmě řešit i pomocí odkládání mezivýsledků do zápisníku, ale to již závisí na konkrétním případě.



## 6. INSTRUKCE SKOKŮ A VOLÁNÍ

### 6.1. Instrukce skoku

Instrukce **JMP**, **JMD**, **JMC**, **JMI**, **JZ**, **JNZ**, **JC**, **JNC**, **JS**, **JNS** provádějí skoky v uživatelském programu na zadané návěští.

Návěští označuje místo v programu, které slouží jako cíl instrukcí skoků nebo volání. Návěštím může být navíc označeno libovolné místo v programu, pokud je to potřebné v zájmu lepší přehlednosti při prohlížení, monitorování nebo ladění uživatelského programu. K jeho označení slouží instrukce **L n**, kde *n* je číslo v rozsahu 0 až 8191.

Použijeme-li v uživatelském programu návěští **L** s nejvyšší hodnotou parametru *n*, překladač vytvoří tabulku adres návěští pro všechna návěští od **L0** do **Ln** včetně těch, které nejsou využity (mají nulovou adresu). Z toho vyplývá, že použijeme-li v programu pouze návěští **L8191**, zabere nám tabulka návěští zbytečně 16 KB paměti, přičemž adresy všech ostatních návěští jsou nulové! Proto doporučujeme používat symbolická jména návěští, překladač **xPRO** návěštím přiděluje čísla vzestupně od 0. Pokud potřebujeme návěštím přidělit pevná čísla (např. z důvodů použití instrukcí **JMI** nebo **CAI**), použijeme direktivu *#label*.

#### Příklad 6.1.1

Pokud má proměnná *cislo* hodnotu 10, zvýšme hodnotu proměnné *citac* o 1.

```
B06101a.mos
#reg byte cislo, citac
;
P 0
 LD cislo
 EQ 10
 JMC nic ;A0 = log.0 -> skok
 INR citac

nic:
E 0
```

Návěští *nic* má automaticky přidělené číslo 0. Protože je nám v tomto a ve většině ostatních případů jedno, jaké číslo bude návěští přiděleno, nemusíme jej deklarovat direktivou *#label*.

Můžeme také použít instrukci **CMP**, která je rychlejší.

```
B06101b.mos
#reg byte cislo, citac
;
P 0
 LD cislo
 CMP 10
 JNZ nic ;S0.0 = log.0 -> skok
 INR citac

nic:
E 0
```

Viz též příklad 7.1.1.

**Příklad 6.1.2**

Má-li proměnná *cislo* hodnotu vyšší než 50, uložíme tuto hodnotu do proměnné *vrchol* a zvýšíme hodnotu proměnné *citacv* o 1. V ostatních případech zvýšíme o 1 hodnotu proměnné *citac*.

*B06102a.mos*

```
#reg byte cislo, citac, citacv, vrchol
;
P 0
 LD cislo
 GT 50
 JMC neni ;A0 = log.0 -> skok
 LD cislo
 WR vrchol
 INR citacv
 JMP konec
;
neni:
 INR citac
konec:
E 0
```

Zde je jednoznačně výhodnější instrukce **CMP**, která nepřepisuje *vrchol* zásobníku.

*B06102b.mos*

```
#reg byte cislo, citac, citacv, vrchol
;
P 0
 LD cislo
 CMP 51
 JC neni ;S0.1 = log.1 -> skok
 WR vrchol ;S0.1 = log.0 - cislo ≥ 51
 INR citacv
 JMP konec
;
neni:
 INR citac
konec:
E 0
```

Viz též příklad 7.1.2.

**Příklad 6.1.3**

Má-li proměnná *cislo* hodnotu vyšší než 50, uložíme tuto hodnotu do proměnné *vrchol* a zvýšíme hodnotu proměnné *citacv* o 1. Má-li proměnná *cislo* hodnotu nižší než 50, uložíme tuto hodnotu do proměnné *propad* a zvýšíme hodnotu proměnné *citacn* o 1. V případě *cislo* = 50 zvýšíme o 1 hodnotu proměnné *citac*.

Případ trojnásobného větvení vede na instrukci **CMP**.

*B06103.mos*

```
#reg byte cislo, citac, citacv, citacn, vrchol, propad
;
P 0
 LD cislo
 CMP 50
 JZ rovno ;S0.0 = log.1 -> skok (cislo = 50)
 JC mensi ;S1.0 = log.1 -> skok (cislo < 50)
 WR vrchol ;cislo > 50
```

```

 INR citacv
 JMP konec
;
mensi:
 WR propad ;cislo < 50
 INR citacn
 JMP konec
;
rovno:
 INR citac ;cislo = 50
konec:
E 0

```

#### Příklad 6.1.4

Vyhledejte v tabulce *seznam* hodnotu uloženou v proměnné *hodnota*. Pokud nebude hodnota nalezena, zvýšte hodnotu proměnné *chyby* o 1.

Tabulkové instrukce nastavují příznak S1.0 v případě správného výsledku. Této skutečnosti můžeme využít.

```

B06104.mos
#reg word hodnota
#reg byte chyby, index
#table word seznam = 1,2,5,10,20,50,100,200,500,1000,2000,5000
;
P 0
 LD hodnota
 FTB seznam ;hledání v seznamu hodnot
 WR index ;index hodnoty v seznamu
 JS konec
 INR chyby ;hodnota nebyla nalezena
konec:
E 0

```

#### Příklad 6.1.5

V proměnné *cinnost* je uložen kód následně prováděnné činnosti. Realizujme skok na provádění příslušné činnosti podle tohoto kódu.

Použijeme instrukci **JMI** a začátky činností označíme návěštími pevně svázanými deklarací *#label* (analogický postup s instrukcí **CAI** je uveden v příkladu 6.2.1).

```

B06105a.mos
#label 0,cin0,cin1,cin2,cin3,cin4
#reg byte cinnost
;
P 0
 LD cinnost
 CMP 5 ;test platné hodnoty kódu
 JNC chyba ;překročení platné hodnoty -> skok
 JMI ;skok podle kódu
;
cin0:
 JMP konec
;
cin1:
 JMP konec
;
cin2:
 JMP konec

```

```

;
cin3:
 JMP konec
;
cin4:
 JMP konec
;
chyba:
 : ;ošetření chyby
konec:
E 0

```

Pokud nemůžeme začít číslovat návěští od 0 (např. proto, že používáme instrukce **JMI** a **CAI** víckrát), musíme číslo prvního návěští přičíst ke kódu.

```

B06105b.mos
#def posun 13 ;cin0 bude odpovídat L 13
#label posun,cin0,cin1,cin2,cin3,cin4
#reg byte cinnost
;
P 0
 LD cinnost
 CMP 5 ;test platné hodnoty kódu
 JNC chyba ;překročení platné hodnoty -> skok
 ADD posun ;posunutí hodnoty kódu
 JMI ;skok podle kódu, při neplatné hodnotě
; ;skok na návěští chyba
cin0:
 JMP konec
;
cin1:
 JMP konec
;
cin2:
 JMP konec
;
cin3:
 JMP konec
;
cin4:
 JMP konec
;
chyba:
 : ;ošetření chyby
konec:
E 0

```

V případě, že více kódů vede na jednu činnost, bude na začátku této činnosti více návěští, nebo kód převedeme pomocí tabulkových instrukcí na redukovaný kód odpovídající výlučně jediné činnosti. Příklady jsou uvedeny v rámci příkladů tabulkových instrukcí příkladem 8.3.3 počínaje.

Nepodceňujme kontrolu platných hodnot kódu. Skok „do neznáma“ zpravidla končí v lepším případě závažnou chybou PLC (80 13 PC PC - návěští neexistuje, nebo 80 14 PC PC - překročeno maximální číslo návěští), v horším případě nevypočitatelným chováním, kdy procesor z instrukce **JMI** skáče kamsi do hlubin uživatelského programu.

## 6.2. Realizace podprogramů

Instrukce **CAL**, **CAD**, **CAC**, **CAI** provádějí volání podprogramu začínajícího na zadaném návěští.

Podprogram je úsek uživatelského programu, který je zahájený instrukcí **L** a je zakončený některou z instrukcí návratu (**RET**, **RED**, **REC**). Obvykle se předpokládá jeho aktivace z několika míst uživatelského programu. Forma podprogramu je výhodná k popisu standardizovaných úloh a stavebních úloh, které by se jinak vyskytovaly rozepsané v různých místech uživatelského programu nebo pokud je žádoucí přehledně a systematicky vytvořený uživatelský program (zejména u složitějších aplikací).

Přechod do podprogramu je obdobou skoku. Navíc je však do pomocného zásobníku návratových adres uložena adresa instrukce, která je zapsána za instrukcí volání (návratová adresa) - na této adrese pokračuje uživatelský program po návratu z podprogramu. Podprogram tedy můžeme chápat jako „odbočku“, ze které se uživatelský program automaticky vrací za místo, z kterého odbočil. Stejný podprogram tedy může být volán z různých míst uživatelského programu a vždy se správně vrátí (za předpokladu, že uživatel neudělal v podprogramu chybu).

Zásobník návratových adres má 8 úrovní a jeho obsah není dostupný z uživatelského programu. Při každém volání se zvýší jeho úroveň a na vrchol se uloží návratová adresa tohoto volání. Každý návrat snižuje úroveň tohoto zásobníku. Je tedy možné vnořování podprogramů do sebe: program volá podprogram, který volá podprogram, který ..., atd. až do osmi úrovní.

Překročení osmé úrovně je hlášeno jako chyba 80 10 PC PC - nejčastěji k této chybě dochází zacyklením, kdy podprogram volá sám sebe (přímo nebo zprostředkovaně).

K chybě může dojít i v jiných případech nevyváženého použití zásobníku návratových adres (např. výskyt instrukce návratu v hlavním programu - chyba 80 11 PC PC, nebo opomenutí instrukce návratu v podprogramu - chyba 80 12 PC PC).

### Příklad 6.2.1

Pokud má proměnná *cislo1* hodnotu 10, zvýšme hodnotu proměnné *citac* o 1. Totéž provedme, pokud má proměnná *cislo2* hodnotu 50.

*B06201a.mos*

```
#reg byte cislo1, cislo2, citac
;
P 0
 LD cislo1
 EQ 10
 JMC test2 ;A0 = log.0 -> skok
 CAL inkrement
test2:
 LD cislo2
 EQ 50
 JMC konec ;A0 = log.0 -> skok
 CAL inkrement
konec:
E 0
;
P 60
inkrement:
 INR citac
 RET
E 60
```

V tomto případě je výhodné použít podmíněné volání podprogramu.

```

B06201b.mos
#reg byte cislo1, cislo2, citac
;
P 0
 LD cislo1
 EQ 10
 CAD inkrement ;A0 = log.1 -> podprogram
 LD cislo2
 EQ 50
 CAD inkrement ;A0 = log.1 -> podprogram
E 0
;
P 60
inkrement:
 INR citac
 RET
E 60

```

Zde uvedený podprogram s jednou výkonnou instrukcí má samozřejmě symbolický charakter. Cílem je demonstrovat výhodu podmíněného volání.

### Příklad 6.2.2

V proměnné *cinnost* je uložen kód následně prováděnné činnosti. Realizujme skok na provádění příslušné činnosti podle tohoto kódu.

Zadání je shodné s příkladem 6.1.5, kde byl realizován pomocí instrukce **JMI**.

```

B06202.mos
#label 0,cin0,cin1,cin2,cin3,cin4,chyba
#reg byte cinnost
;
P 0
 LD cinnost
 CMP 5 ;test platné hodnoty kódu
 JC platny
 LD 5 ;překročení platné hodnoty - oprava
platny:
 CAI ;skok do podprogramu podle kódu, při chybném
E 0 ;kódu zavolání podprogramu chyba
;
P 60
cin0:
 RET
;
cin1:
 RET
;
cin2:
 RET
;
cin3:
 RET
;
cin4:
 RET
;
chyba:
 RET
E 60

```

Použití instrukce **CAI** vede na rozdíl od instrukce **JMI** k přehlednějšímu uspořádání uživatelského programu. Kontrolu platnosti kódů je nutné provést pomocí instrukcí porovnání a v případě chyby generování opraveného kódu pro zavolání ošetření chyby.

Nepodceňujeme kontrolu platných hodnot kódu. Skok „do neznáma“ zpravidla končí v lepším případě závažnou chybou PLC (80 13 PC PC - návěští neexistuje, nebo 80 14 PC PC - překročeno maximální číslo návěští), v horším případě nevypočitatelným chováním, kdy procesor z instrukce **CAI** skáče kamsi do hlubin uživatelského programu.

### **Příklad 6.2.3**

Pokud detekujeme na signálu *vstup* náběžnou hranu, pak provedme následující činnost. Má-li proměnná *cislo* hodnotu vyšší než 50, zvýšme hodnotu proměnné *citac* o 1. V ostatních případech nedělejme nic.

Místo podmíněného skoku **JMC** na návěští před koncem podprogramu **RET** použijeme instrukci podmíněného návratu **REC**.

*B06203.mos*

```
#reg bit vstup
#reg byte cislo, citac
#reg bit stav
;
P 0
 LD vstup
 LET stav
 CAD akce ;při náběžné hraně provedena akce
E 0
;
P 60
akce:
 LD cislo
 GT 50
 REC ;A0 = log.0 -> konec podprogramu
 INR citac
 RET
E 60
```

## 7. ORGANIZAČNÍ INSTRUKCE

### 7.1. Podmíněný konec procesu (EC, ED)

Organizační instrukce zpřehledňují stavbu uživatelského programu a jeho částí.

Instrukce **P** označuje začátek uživatelského procesu, instrukce **E**, **ED**, **EC** provádějí konec uživatelského procesu.

#### Příklad 7.1.1

Pokud má proměnná *cislo* hodnotu 10, zvýšme hodnotu proměnné *citac* o 1.

Zadání je totožné s příkladem 6.1.1, který jsme v první variantě realizovali s pomocí instrukce **EQ** a skoku **JMC** *nic*. Pokud návěští *nic* bezprostředně předchází konec procesu **E 0**, pak můžeme použít podmíněný konec procesu.

```
B07101.mos
#reg byte cislo, citac
;
P 0
 LD cislo
 EQ 10
 EC ;A0 = log.0 -> konec
 INR citac
E 0
```

#### Příklad 7.1.2

Má-li proměnná *cislo* hodnotu vyšší než 50, uložíme tuto hodnotu do proměnné *vrchol* a zvýšme hodnotu proměnné *citacv* o 1. V ostatních případech zvýšme o 1 hodnotu proměnné *citac*.

Zadání je totožné s příkladem 6.1.2. Zde se program dělí na dvě větve, které jsou ukončeny skokem na návěští *konec*. Protože nemůžeme napsat dvakrát instrukci **E 0**, vícenásobný konec procesu realizujeme takto:

```
B07102.mos
#reg byte cislo, citac, citacv, vrchol
;
P 0
 LD cislo
 CMP 51
 JC neni ;S0.1 = log.1 -> skok
 WR vrchol ;S0.1 = log.0 - cislo ≥ 51
 INR citacv
 ED ;konec pro A ≠ 0
 EC ;konec pro A = 0
;
neni:
 INR citac
E 0
```

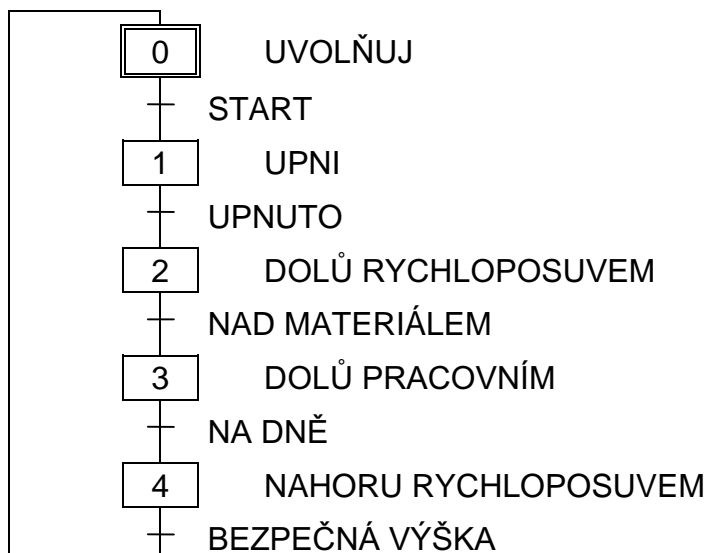


## 7.2. Podmíněné přerušení procesu (SEQ)

Instrukce **SEQ** umožňuje podmíněně ukončit proces a současně definovat místo, odkud se proces začne v příštím cyklu vykonávat. Toho můžeme využít v sekvenčním programování.

### Příklad 7.2.1

Realizujeme sekvenční řadič podle příkladu 3.4.4.  
Sekvenční řadič je dán přechodovým grafem:



*B07201.mos*

```
#reg byte vstupy
#def start vstupy.0 ;start
#def upnuto vstupy.1 ;upnuto
#def nahore vstupy.2 ;nad materiálem
#def dole vstupy.3 ;na dně
#def bezpeci vstupy.4 ;bezpečná výška
;
#reg byte vystupy
#def uvolnuj vystupy.0 ;uvolňuj
#def upni vystupy.1 ;upni
#def dolu vystupy.2 ;dolů rychloposuvem
#def prac vystupy.3 ;dolů pracovním
#def nahoru vystupy.4 ;nahoru rychloposuvem
;
P 0
 LD start
 SET %S25.1 ;zařadit proces P10 s krokovým řadičem
E 0
;
P 10
 LD 1
 RES uvolnuj ;stav 1 (v A0 je log.1)
 SET upni
prechod12:
 LD upnuto ;podmínka přechodu do stavu 2
 SEQ prechod12 ;při upnuto = log.0 konec P10,
 ;nový proces P10 začne na prechod12
 SET dolu ;stav 2 (v A0 je log.1)
```

```

prechod23:
 LD nahore ;podmínka přechodu do stavu 3
 SEQ prechod23 ;při nahore = log.0 konec P10,
 ;nový proces P10 začne na prechod23
 RES dolu ;stav 3 (v A0 je log.1)
 SET prac
prechod34:
 LD dole ;podmínka přechodu do stavu 4
 SEQ prechod34 ;při start = log.0 konec P10,
 ;nový proces P10 začne na prechod34
 RES prac ;stav 4 (v A0 je log.1)
 SET nahoru
prechod40:
 LD bezpeci ;podmínka přechodu do stavu 0
 SEQ prechod40 ;při start = log.0 konec P10,
 ;nový proces P10 začne na prechod40
 RES nahoru ;stav 0 (v A0 je log.1)
 RES upni
 SET uvolnuj
 RES %S25.1 ;vyřadit proces P10
E 10
;
P 63
 LD 1
 WR vystupy ;počáteční stav výstupů
E 63

```

Celý řadič jsme umístili do samostatného procesu, jehož spuštění interpretujeme jako přechod ze stavu 0 do stavu 1 (start). Řadič po přechodu ze stavu 4 do stavu 0 sám provede vyřazení procesu.

Porovnáme-li tento způsob řešení s řešením pomocí instrukce **STE** (viz příklad 3.4.4), dojdeme k závěru, že toto řešení sice zabírá více paměti, ale na druhou stranu je přehledné a obecnější (zejména v případech, kdy akce odpovídající jednotlivým stavům jsou různé, nespočívající jen v nastavování různých hodnot výstupů). Bližší analýzou dokonce zjistíme, že v každém cyklu se prochází méně instrukcí (5 až 14), než při variantě s instrukcí **STE** (vždy 16 instrukcí).

Viz též příklad 8.1.6.

## 8. TABULKOVÉ INSTRUKCE

Tabulky T jsou součástí uživatelského programu a mají vždy předepsanou strukturu, je to vždy řada hodnot stejné šířky (bit, byte, word, long nebo float) s přidávanými kontrolními údaji. Každé položce (hodnotě z této řady) je přiřazeno pořadové číslo - index. Nejnížší položka má nulový index, index poslední položky nazýváme mezí (počet položek = mez + 1). S každou tabulkou je uložena i její mez (jednotně měřená v počtu bytů).

Tabulky v zápisníku mají stejnou strukturu jako tabulky T. Mez se zadává jako parametr v zásobníku.

Aparát tabulkových instrukcí dovoluje realizovat jednoduchým způsobem i velmi složité funkce, přičemž řešení bývá úspornější a rychlejší oproti tradičnímu (někdy výrazně), vždy je však pružnější a přizpůsobivější. Program je názorný a přehledný.

Každou tabulku lze interpretovat ve formátech bit, byte, word, long nebo float podle použité instrukce. Všechny tabulky jsou uloženy bytově nezávisle na tom, jak byly v uživatelském programu zadány. Z toho vyplývá následující:

- bitově zadaná tabulka je doplněna na celý počet bytů nulovými bity
- pokud interpretujeme ve formátu word tabulku s lichým počtem bytů, je poslední byte nedostupný, obdobná situace nastane u nezarovnaných tabulek formátu long nebo float

### 8.1. Čtení a zápis do tabulek T (LTB, WTB)

Instrukce **LTB** umožňuje čtení z tabulky T a instrukce **WTB** zápis do tabulky T ve formátech bit, byte, word.

Pozor! Zápis do tabulek T při zapnuté uživatelské paměti EEPROM je možný, pokud při překladu uživatelského programu překladačem xPRO označíme volbu *Chráněné tabulky T*, jinak budou po vypnutí a opětovném zapnutí PLC všechny změny ztraceny (z EEPROM se zkopírují hodnoty deklarované ve zdrojovém textu uživatelského programu).

#### Příklad 8.1.1

Předpokládejme, že pro výstupy *vystup0* až *vystup4* máme realizovat čtyři kombinační logické funkce proměnných *vstup0* až *vstup3*, zadané pravdivostní tabulkou:

| <i>vstup3</i> | <i>vstup2</i> | <i>vstup1</i> | <i>vstup0</i> | <i>vystup4</i> | <i>vystup3</i> | <i>vystup2</i> | <i>vystup1</i> | <i>vystup0</i> |
|---------------|---------------|---------------|---------------|----------------|----------------|----------------|----------------|----------------|
| 0             | 0             | 0             | 0             | 0              | 1              | 0              | 0              | 1              |
| 0             | 0             | 0             | 1             | 1              | 0              | 0              | 0              | 1              |
| 0             | 0             | 1             | 0             | 1              | 1              | 0              | 0              | 0              |
| 0             | 0             | 1             | 1             | 0              | 1              | 1              | 0              | 0              |
| 0             | 1             | 0             | 0             | 1              | 1              | 0              | 0              | 1              |
| 0             | 1             | 0             | 1             | 1              | 0              | 1              | 0              | 1              |
| 0             | 1             | 1             | 0             | 0              | 0              | 1              | 0              | 1              |
| 0             | 1             | 1             | 1             | 0              | 1              | 1              | 1              | 1              |
| 1             | 0             | 0             | 0             | 0              | 1              | 0              | 0              | 0              |
| 1             | 0             | 0             | 1             | 0              | 0              | 1              | 0              | 1              |
| 1             | 0             | 1             | 0             | 1              | 1              | 1              | 0              | 0              |
| 1             | 0             | 1             | 1             | 1              | 1              | 1              | 1              | 0              |
| 1             | 1             | 0             | 0             | 1              | 0              | 1              | 0              | 0              |
| 1             | 1             | 0             | 1             | 0              | 0              | 1              | 1              | 1              |
| 1             | 1             | 1             | 0             | 1              | 0              | 1              | 1              | 0              |
| 1             | 1             | 1             | 1             | 0              | 1              | 1              | 1              | 0              |

Tradiční postup spočívající v rozepsání všech pěti funkcí do logických výrazů, jejich úpravě a přepisu do posloupnosti instrukcí zde neuvádíme - doporučujeme však čtenáři, aby jej pro porovnání vyzkoušel. Při použití instrukce **LTB** (přímým překladem) existují dva přístupy. Prvý přiřazuje každé skalární logické funkci *vystup4* až *vystup0* jednu bitovou tabulku:

```
B08101a.mos
#reg byte vstupy
#reg bit vystup0, vystup1, vystup2, vystup3, vystup4
;
#table bit tab0 = 1,1,0,0,1,1,1,1,0,1,0,0,0,1,0,0
#table bit tab1 = 0,0,0,0,0,0,0,1,0,0,0,1,0,1,1,1
#table bit tab2 = 0,0,0,1,0,1,1,1,0,1,1,1,1,1,1,1
#table bit tab3 = 1,0,1,1,1,0,0,1,1,0,1,1,0,0,0,1
#table bit tab4 = 0,1,1,0,1,1,0,0,0,0,1,1,1,0,1,0
;
P 0
 LD vstupy
 LTB tab0 ;funkce 0
 WR vystup0
 POP 1
 LTB tab1 ;funkce 1
 WR vystup1
 POP 1
 LTB tab2 ;funkce 2
 WR vystup2
 POP 1
 LTB tab3 ;funkce 3
 WR vystup3
 POP 1
 LTB tab4 ;funkce 4
 WR vystup4
E 0
```

Oproti tradičnímu je toto řešení nesrovnatelně úspornější, rychlejší, nesrovnatelně snáze můžeme respektovat změnu zadání funkcí. Prvou instrukcí sejmem vstupní vektor, pak postupně realizujeme jednotlivé skalární funkce 0 až 4 a zapisujeme do výstupních proměnných *vystup0* až *vystup4*. Instrukce **POP 1** nahrazuje opětné sejmutí vstupního vektoru (**POP** je rychlejší). Tabulky *tab0* až *tab4* jsou deklarovány bitově a překladač implicitně přiřazuje tento formát k instrukci s příslušnou tabulkou. Pokud bychom tabulky zadali ve formátu byte, word nebo long museli bychom provést u instrukcí **LTB** přetypování.

```
B08101b.mos
#reg byte vstupy
#reg bit vystup0, vystup1, vystup2, vystup3, vystup4
;
#table word tab0 = %0010001011110011
#table word tab1 = %1110100010000000
#table word tab2 = %1111111011101000
#table word tab3 = %1000110110011101
#table word tab4 = %0101110000110110
;
P 0
 LD vstupy
 LTB tab0.0 ;funkce 0
 WR vystup0
 POP 1
```

```

LTB tab1.0 ;funkce 1
WR vystup1
POP 1
LTB tab2.0 ;funkce 2
WR vystup2
POP 1
LTB tab3.0 ;funkce 3
WR vystup3
POP 1
LTB tab4.0 ;funkce 4
WR vystup4

```

E 0

Ještě rychlejší je následující přístup. Tabulkou, v níž každá položka obsahuje informace funkcí 0 až 4 odpovídající společnému indexu (nejvyšší 3 bity jsou nevyužity), definujeme vektorovou funkci proměnných *vstup0*, *vstup1*, *vstup2* a *vstup3*. Vektorem zde nazýváme uspořádaný soubor bitových (skalárních) funkcí, tedy (*vystup4*, *vystup3*, *vystup2*, *vystup1*, *vystup0*).

*B08101c.mos*

#reg byte *vstupy*

#reg byte *vystupy*

;

```

#table byte tab = %01001,%10001,%11000,%01100,%11001,%10101,
 %00101,%01111,%01000,%00101,%11100,%11110,
 %10100,%00111,%10110,%01110

```

;

P 0

```

LD %11111
RES vystupy ;předběžné nulování výstupů
LD vstupy
LTB tab
AND %11111
SET vystupy ;nastavení výstupů

```

E 0

Toto řešení je úspornější oproti předchozímu, přestože tabulka *tab* je ze 3/8 nevyužita. Tento volný prostor však můžeme obsadit dalšími třemi bitovými funkcemi stejného počtu proměnných (stejných proměnných *vstup0*, *vstup1*, *vstup2*, *vstup3* nebo úplně jiných).

Obecně platí, že použití bitového formátu tabulek je výhodnější pro ojedinělé použití dlouhých tabulek (velkého počtu vstupních proměnných). Pro větší počet krátkých bitových tabulek stejného počtu proměnných nebo pro realizaci vektorových funkcí je pak výhodnější typ byte nebo word.

### Příklad 8.1.2

Převeďme desítkovou číslici v proměnné *cislo* do kódu 2 z 5 dle tabulky a uložme do proměnné *kod*.

| <i>cislo</i> | <i>kod</i> |
|--------------|------------|
| 0 0000       | 00110      |
| 1 0001       | 10001      |
| 2 0010       | 01001      |
| 3 0011       | 11000      |
| 4 0100       | 00101      |

| <i>cislo</i> | <i>kod</i> |
|--------------|------------|
| 5 0101       | 10100      |
| 6 0110       | 01100      |
| 7 0111       | 00011      |
| 8 1000       | 10010      |
| 9 1001       | 01010      |

```

B08102.mos
#reg byte cislo, kod
;
#table byte tab = %00110,%10001,%01001,%11000,%00101,%10100,
 %01100,%00011,%10010,%01010
;
P 0
 LD %11111
 RES kod ;předběžné nulování kódu
 LD cislo
 LTB tab
 AND %11111
 SET kod
E 0

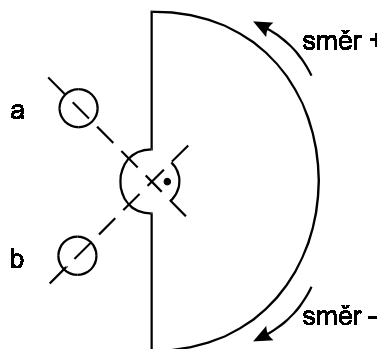
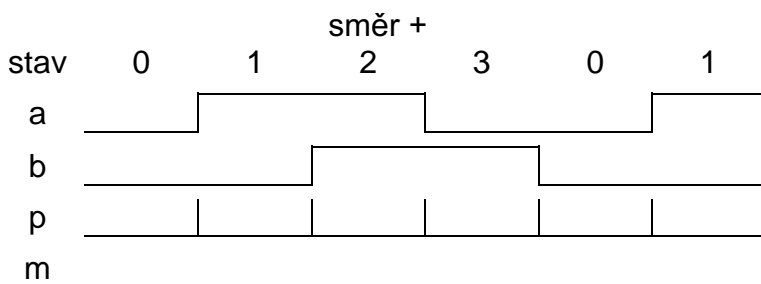
```

Opačný postup viz příklad 8.3.1.

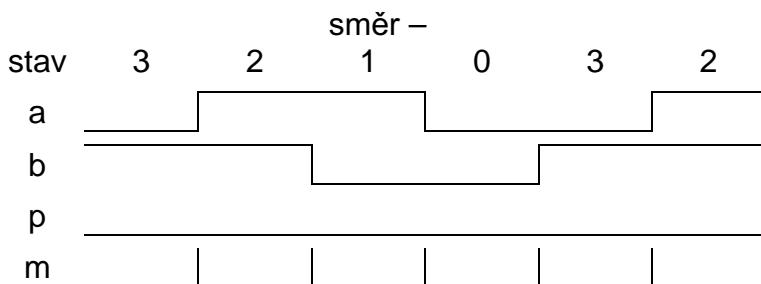
### Příklad 8.1.3

Předpokládejme jednoduchý snímač otáčení dle náčrtu.

K hřídeli je připojena půlkruhová clonka, která zatmívá dvě fotobuňky a, b, připojené ke vstupům *fotoa* a *fotob*. Máme vyhodnotit signály ze clonek tak, abychom rozlišili čtyři polohy během otáčky a čítali počet čtvrtotáček s rozlišením směru pohybu: osvětlené fotobuňce odpovídá hodnota log.1, zatacloněné log.0. Budeme-li předpokládat plynulé otáčení směrem +, pak na fotobuňkách a, b, dostaneme následující posloupnost:



Při opačném směru otáčení dostaneme posloupnost:



Impulzy p, m jsou vnitřní proměnné (*plus*, *minus*) vyhodnocené při každém rozlišení směru (při hranách a, b), které budeme přičítat nebo odečítat ke stavu čítače čtvrtotáček. Polohu v rámci otáčky určíme z kombinace úrovní a, b (čísla stavu). Během jedné otáčky lze rozlišit čtyři impulzy ve směru + a čtyři impulzy ve směru -. Tradičně bychom asi postupovali tak, že pomocí instrukcí **LET** bychom vygenerovali impulzy náběžné a sestupné hrany vstupů *fotoa*, *fotob* a pak v kombinaci s hodnotami *fotoa*, *fotob* vygenerovali impulzy *plus*, *minus* nebo přímo hodnotami aktualizovali stav čítače. Při tradičním bitovém programování je to postup rutinní, ale zdoluhavý. Při použití bytových instrukcí lze postup zrychlit, je však náročný na představivost.

Zde uvedeme postup pracující s tabulkou. K vyhodnocení hrany potřebujeme vždy znát současnou a minulou úroveň logické proměnné. Předpokládejme, že v každém cyklu

zpracováváme minulý vzorek proměnných *fotoa*, *fotob* (označíme je  $a_0$ ,  $b_0$ ) a současný vzorek (označíme jej  $a_1$ ,  $b_1$ ). Průběh proměnných při obou směrech otáčení můžeme soustředit do tabulek, z nichž první definuje součinné podmínky pro impulzy *plus*, druhá pro impulzy *minus*.

| směr + |       |       |       | směr – |       |       |       |
|--------|-------|-------|-------|--------|-------|-------|-------|
| $a_0$  | $b_0$ | $a_1$ | $b_1$ | $a_0$  | $b_0$ | $a_1$ | $b_1$ |
| 0      | 0     | 1     | 0     | 0      | 0     | 0     | 1     |
| 1      | 0     | 1     | 1     | 0      | 1     | 1     | 1     |
| 1      | 1     | 0     | 1     | 1      | 1     | 1     | 0     |
| 0      | 1     | 0     | 0     | 1      | 0     | 0     | 0     |

Obě tabulky můžeme sloučit do Karnaughovy mapy:

|  |       |              |              |              |              |
|--|-------|--------------|--------------|--------------|--------------|
|  |       | $a_0$        |              |              |              |
|  |       |              |              | $b_0$        |              |
|  |       | <i>nic</i>   | <i>plus</i>  | <i>chyba</i> | <i>minus</i> |
|  | $a_1$ | <i>minus</i> | <i>nic</i>   | <i>plus</i>  | <i>chyba</i> |
|  |       | <i>chyba</i> | <i>minus</i> | <i>nic</i>   | <i>plus</i>  |
|  |       | <i>plus</i>  | <i>chyba</i> | <i>minus</i> | <i>nic</i>   |
|  |       |              |              |              |              |
|  |       | $b_1$        |              |              |              |

Políčka označená *plus* a *minus* odpovídají jedničkové hodnotě impulzních proměnných *plus*, *minus*. Políčka označená *nic* odpovídají stavům, kdy se žádný ze vstupů nezměnil. Políčka označená *chyba* odpovídají stavům, kdy se současně mění dva signály, což je za normálních podmínek vyloučeno. Jedničkové políčko tedy signalizuje chybu (rušivý impulz, rychlost změn překročila rychlost jejich zpracování, apod.). Nyní máme několik možností, jak postupovat.

Tradičnímu postupu odpovídá transformace mapy na logické výrazy, které budeme realizovat programem (symbolu  $\oplus$  odpovídá součet mod 2).

$$plus = (b_0 \oplus a_1) \cdot (a_0 \oplus b_0 \oplus a_1 \oplus b_1) \quad \text{pulzy +}$$

$$minus = (a_0 \oplus b_1) \cdot (a_0 \oplus b_0 \oplus a_1 \oplus b_1) \quad \text{pulzy –}$$

$$chyba = (a_0 \oplus a_1) \cdot (a_0 \oplus b_0 \oplus a_1 \oplus b_1 \oplus 1) \quad \text{chyba (změna obou vstupů)}$$

Mapu však můžeme přetransformovat na pravdivostní tabulku (mohli jsme ji sestavit přímo bez mapy):

## 8. Tabulkové instrukce

| $a_0$ | $b_0$ | $a_1$ | $b_1$ | <i>plus</i> | <i>minus</i> | <i>chyba</i> | <i>nic</i> | <i>maska</i> |   |   |   |
|-------|-------|-------|-------|-------------|--------------|--------------|------------|--------------|---|---|---|
| 0     | 0     | 0     | 0     | 0           | 0            | 0            | 1          | 0            | 0 | 0 | 1 |
| 0     | 0     | 0     | 1     | 0           | 1            | 0            | 0          | 1            | 0 | 0 | 0 |
| 0     | 0     | 1     | 0     | 1           | 0            | 0            | 0          | 0            | 0 | 1 | 0 |
| 0     | 0     | 1     | 1     | 0           | 0            | 1            | 0          | 0            | 1 | 0 | 0 |
| 0     | 1     | 0     | 0     | 1           | 0            | 0            | 0          | 0            | 0 | 0 | 1 |
| 0     | 1     | 0     | 1     | 0           | 0            | 0            | 1          | 1            | 0 | 0 | 0 |
| 0     | 1     | 1     | 0     | 0           | 0            | 1            | 0          | 0            | 0 | 1 | 0 |
| 0     | 1     | 1     | 1     | 0           | 1            | 0            | 0          | 0            | 1 | 0 | 0 |
| 1     | 0     | 0     | 0     | 0           | 1            | 0            | 0          | 0            | 0 | 0 | 1 |
| 1     | 0     | 0     | 1     | 0           | 0            | 1            | 0          | 1            | 0 | 0 | 0 |
| 1     | 0     | 1     | 0     | 0           | 0            | 0            | 1          | 0            | 0 | 1 | 0 |
| 1     | 0     | 1     | 1     | 1           | 0            | 0            | 0          | 0            | 1 | 0 | 0 |
| 1     | 1     | 0     | 0     | 0           | 0            | 1            | 0          | 0            | 0 | 0 | 1 |
| 1     | 1     | 0     | 1     | 1           | 0            | 0            | 0          | 1            | 0 | 0 | 0 |
| 1     | 1     | 1     | 0     | 0           | 0            | 1            | 0          | 0            | 0 | 0 | 1 |
| 1     | 1     | 1     | 1     | 1           | 0            | 0            | 0          | 1            | 0 | 0 | 0 |
| 1     | 1     | 1     | 0     | 0           | 1            | 0            | 0          | 0            | 0 | 1 | 0 |
| 1     | 1     | 1     | 1     | 0           | 0            | 0            | 1          | 0            | 1 | 0 | 0 |

Tabulka má 16 bytových položek, které jsou z poloviny nevyužity. Můžeme se s tím smířit nebo můžeme zbylou polovinu využít pro jinou úlohu (např. pro obdobu příkladu 8.1.1). Můžeme si však „vymýšlet“ a rozšířit zadání tak, aby tabulka byla maximálně využita. Můžeme např. dekodovat proměnné  $a_1$ ,  $b_1$  na masku polohy typu „1 ze 4“ - viz proměnné *maska*.

Úlohu pak můžeme řešit krátkým obslužným programem a tabulkou v rozsahu 16 bytů.

*B08103a.mos*

```
#reg bit vstupý ;.1 - fotoa, .0 - fotob
#reg byte vzorek, polozka
#reg word citac
#reg bit prechod
#def minus polozka.6
#def plus polozka.7
;
#table byte tab = %00010001,%01001000,%10000010,%00100100,
 %10000001,%00011000,%00100010,%01000100,
 %01000001,%00101000,%00010010,%10000100,
 %00100001,%10001000,%01000010,%00010100
;
P 0
 LD vzorek ;minulý vzorek - a0, b0
 MUL 4 ;posunutí o 2 pozice vlevo
 LD vstupý ;současné vstupý
 LD 3
 RES vzorek ;vynulování starých a0, b0
 AND vzorek ;oddělení vstupů a1, b1
 SET vzorek ;uložení pro příští cyklus
 OR vzorek ;složení a0, b0, a1, b1
 LTB tab
 WR polozka ;odložení položky
 LD plus ;UP
 LD minus ;DOWN
 LD 0 ;RESET
 CNT citac ;čítání
```

Chybu a nezměněný stav jsme zatím neošetřovali. Z tabulky lze získat další informace. Pokud např. chceme ošetřit přechod do polohy 0 (nezávisle na směru), stačí pokračovat:

```
LD plus
OR minus
```



```
AND polozka.0
WR prechod
```

E 0

Pokud je žádoucí ošetřit přechod do polohy 0 při otočení ve směru +, doplníme pouze:

```
LD plus
AND polozka.0
WR prechod
```

E 0

Úlohu však můžeme řešit ještě jiným přístupem. Vyjdeme ze stejné Karnaughovy mapy. Symboly jejích políček však nyní nebudeme chápat jako jedničkové hodnoty odpovídajících bitových výstupních funkcí, ale jako symbolická jména akcí, které v jednotlivých stavech provádíme - např. *plus* představuje akci (např. podprogram), která ošetřuje impulz +, *chyba* symbolizuje akci ošetřující chybu, apod. Tabulka se změní pouze v tom, že její položky jsou čísla návěští, která zahajují odpovídající akce. Převod čísla položky na odpovídající akci provedeme instrukcí **JMI** nebo raději **CAI**. Po rozvětvení instrukcí **JMI** musíme každou akci zakončit skokem na návěští společného pokračování, zatímco akce aktivované instrukcí **CAI** stačí zakončit instrukcí návratu (**RET**) a program sám bude pokračovat na návratové adrese (za instrukcí **CAI**). Tento přístup je účelný zejména v případech, kdy převažují požadavky na mnohacestné větvení programu. Prvý (bitový) přístup je naopak účelný zejména tehdy, když lineárním (nevětveným) programem nastavujeme výstupy a vykonáváme přitom stále stejnou činnost.

*B08103b.mos*

```
#reg bit vstupý ;.1 - fotoa, .0 - fotob
```

```
#reg byte vzorek
```

```
#reg word citac
```

```
;
```

```
#label 0,nic,chyba,minus,plus ;přiřazení pevných návěští
```

```
#table byte tab = 0,2,3,1,3,0,1,2,2,1,0,3,1,3,2,0
```

```
;
```

P 0

```
LD vzorek ;minulý vzorek - a0, b0
MUL 4 ;posunutí o 2 pozice vlevo
LD vstupý ;současné vstupý
LD 3
RES vzorek ;vynulování starých a0, b0
AND ;oddělení vstupů a1, b1
SET vzorek ;uložení pro příští cyklus
OR ;složení a0, b0, a1, b1
LTB tab
CAI ;rozsok
```

E 0

```
;
```

P 60 ;soubor podprogramů

nic:

```
RET
```

```
;
```

chyba:

```
RET
```

```
;
```

minus:

```
DCR citac ;snížení počtu pulzů
```

```
RET
```

```
;
```

plus:

INR citac ; zvýšení počtu pulzů  
RET

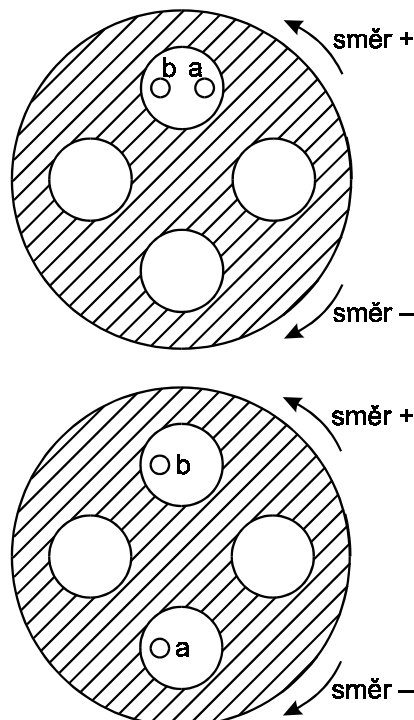
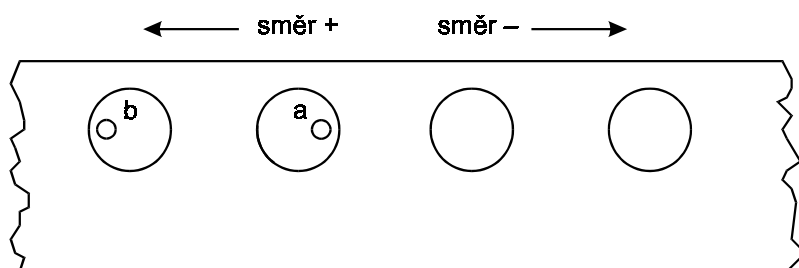
E 60

## Příklad 8.1.4

Předpokládejme, že je požadováno rozlišit 16 poloh za jednu otáčku hřídele a při každé změně polohy inkrementovat nebo dekrementovat stav čítače s ohledem na směr pohybu. Předpokládejme uspořádání kruhového snímače dle náčrtů vpravo.

Mechanické uspořádání snímače se od uspořádání z příkladu 8.1.3 liší jen zdánlivě, podstata problému zůstává stejná. Místo půlené clonky máme zde kotouč s otvory (na tvaru otvorů nezáleží - může to být např. i pastorek), jejichž šířka je stejná jako jsou šířky plných částí. Fotobuňky *a*, *b* jsou umístěny ve vzdálenosti čtvrtiny rozteče mezi otvory - případ dle náčrtu vpravo nahoře. Řešení dle náčrtu vpravo dole je funkčně rovnocenné, je však odolnější vůči vzájemnému ovlivňování fotobuněk *a* a je výhodné při jemnějším dělení kotouče. Průběh signálů i jejich zpracování je stejné jako v příkladu 8.1.3.

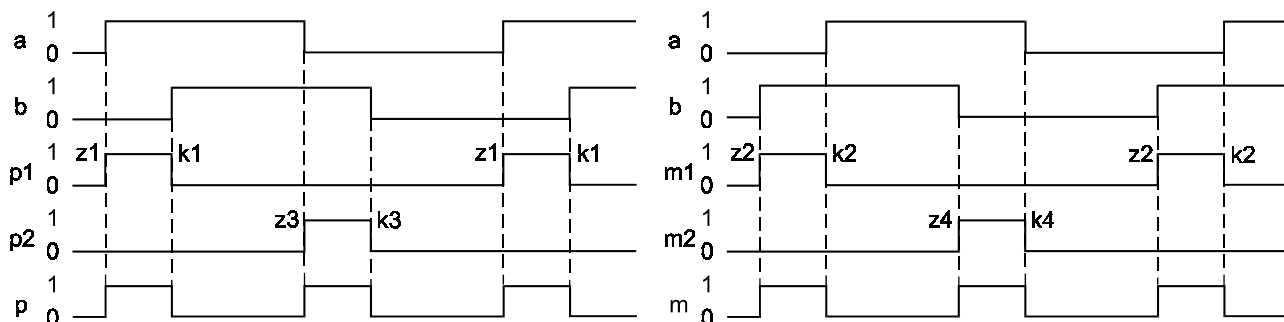
Na stejnou úlohu vede i podélný (lineární) snímač:



## Příklad 8.1.5

Předpokládejme, že na vstupy jsou přivedeny periodické logické proměnné *a*, *b* se stejným kmitočtem (dostatečně pomalým proti době cyklu). Máme rozlišit případy, kdy jsou ve fázi, kdy proměnná *a* předbíhá proměnnou *b* a naopak, kdy se *a* zpožďuje za *b*. Současně máme změřit dobu předbíhání nebo zpoždování. Tuto funkci můžeme nazvat fázovým diskriminátorem nebo fázovým diferenčním členem.

Situaci, kdy signál *a* předbíhá *b* znázorňuje časový průběh vlevo, opačnou situaci časový průběh vpravo:



Proměnné *p* odpovídají předbíhání *a* před *b*, proměnné *m* opačné situaci. Impulzy *p1*, *m1* odpovídají posuvům náběžných hran, *p2*, *m2* posuvům sestupných hran, impulzy *p*, *m* odpovídají posuvům obou hran. Hodnoty *z1*, *k1*, *z2*, *k2*, *z3*, *k3*, *z4*, *k4* ohraničují začátek a

konec impulzů  $p$  a  $m$ . Situaci, kdy vyhodnocujeme rozdíly náběžných i sestupných hran popisuje Karnaughova mapa:

|       |       |            |              |
|-------|-------|------------|--------------|
|       |       | $a_0$      |              |
|       |       | $b_0$      |              |
| $a_1$ | $b_1$ | <i>nic</i> | <i>k3</i>    |
|       |       | <i>sm</i>  | <i>k4</i>    |
|       |       | <i>z2</i>  | <i>nic</i>   |
|       |       | <i>z3</i>  | <i>chyba</i> |
|       |       | <i>sp</i>  | <i>k2</i>    |
|       |       | <i>nic</i> | <i>k1</i>    |
|       |       | <i>z1</i>  | <i>chyba</i> |
|       |       | <i>z4</i>  | <i>nic</i>   |

Mapu můžeme opět převést na tabulku a interpretovat ji programem. V našem případě bude patrně nejvýhodnější symbolům v mapě přiřadit akce - tabulka bude obsahovat číslo návěští, která zahajují podprogramy akcí. Symboly *sp* a *sm* označují situace „současná náběžná“ a „současná závěrná“, symbol *chyba* označuje chybový stav a *nic* označuje neměnné vstupní proměnné.

Například akce odpovídající *z1* až *z4* může aktivovat měření času (časovačem nebo s využitím časoměrných registrů), *k1* až *k4* končí měření času a vyhodnocují časový údaj. Akce *sp* a *sm* nastavují nulový časový odstup signálů, akce *nic* je prázdná, akce *chyba* indikuje chybu.

Při vyhodnocení posuvu obou hran se vyskytují dva problémy:

- pokud nejsou signály přesně souměrné, mohou být výsledky po měření náběžných a sestupných hran odlišné,
- může nastat stav (obzvláště po inicializaci nebo při překročení max. posuvu), kdy nelze rozlišit předbíhání od zpoždování.

Proto může být účelné vyhodnocovat pouze náběžné hrany. V tom případě se změní pouze mapa, tabulka a některé akce, obslužný program se nezmění:

|       |       |             |              |
|-------|-------|-------------|--------------|
|       |       | $a_0$       |              |
|       |       | $b_0$       |              |
| $a_1$ | $b_1$ | <i>klid</i> | <i>klid</i>  |
|       |       | <i>klid</i> | <i>klid</i>  |
|       |       | <i>z2</i>   | <i>nic</i>   |
|       |       | <i>klid</i> | <i>chyba</i> |
|       |       | <i>sp</i>   | <i>k2</i>    |
|       |       | <i>klid</i> | <i>k1</i>    |
|       |       | <i>z1</i>   | <i>chyba</i> |
|       |       | <i>klid</i> | <i>nic</i>   |

Symboły *z1*, *z2*, *k1*, *k2*, *chyba*, *nic*, *sp* mají nezměněný význam, symbol *klid* uvádí systém do klidového stavu, končí měření času a nuluje časový údaj.

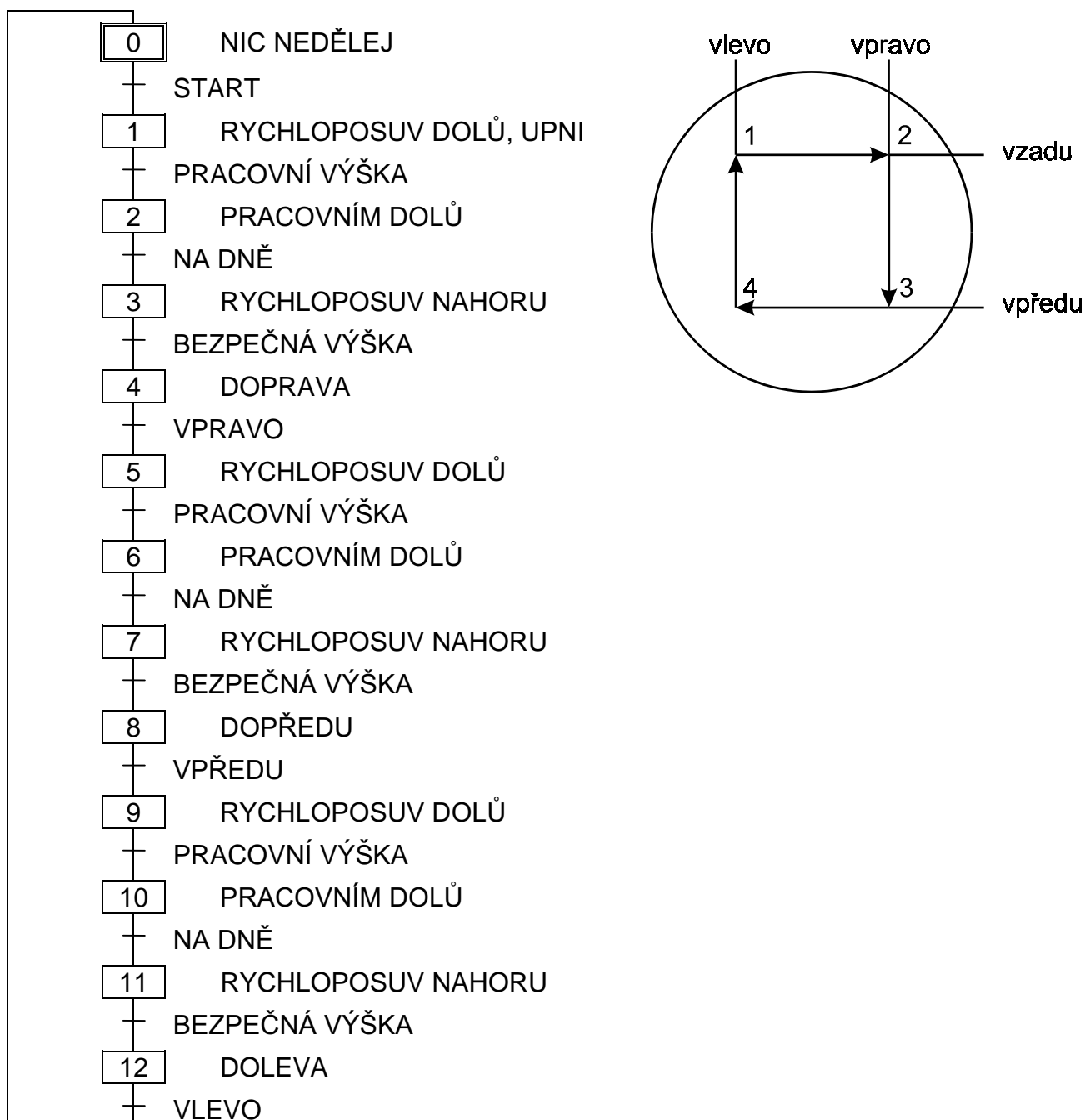
### Příklad 8.1.6

Máme řídit vrtačku, která po ručním založení dílce a po odstartování vyvrtá čtveřici děr. Postup vrtání je stejný jako v příkladu 3.4.4 k instrukci **STE**, resp. v příkladu 7.2.1 k instrukci **SEQ**, tedy

- rychloposuvem na pracovní výšku
- pracovním posuvem na dno
- rychloposuvem na bezpečnou výšku

Kruhový řadič má 8 vstupů a 8 výstupů - na jejich rozmístění nyní nezáleží, zvolíme je tedy náhodně.

Sekvenční řadič má 18 stavů a strukturu dle schématu:





Použití instrukce **STE** zde již není tak výhodné jako u typických příkladů pro tuto instrukci. Použijeme proto postup tabulkové realizace krokového řadiče, který je oproti instrukci **STE** obecnější. Tabulkovou realizaci podporují zejména tyto skutečnosti:

- počet stavů převyšuje 16 - bylo by nutné řešit kaskádování,
- podmínkové bity nelze nenásilně uspořádat tak, aby na stejnohlých pozicích jako jim odpovídající stavy - potřebná vstupní transformace by vyžadovala obsáhlý program a byla by nepřehledná,
- pozice výstupních bitů neodpovídají pozicím ve stavové masce - opět by byla nutná poměrně rozsáhlá transformace stavových bitů na výstupní - zde ji provedeme systematicky, přehledně a úsporně tabulkou.

Poznámka: Je třeba si uvědomit, že se stále jedná o „cvičné problémy“, které se snaží ilustrovat možnosti jednotlivých postupů - zde tabulkových. Pro praxi je tento případ příliš zjednodušen.

U skutečných případů je často účelné (nebo nutné) řešit rozklad složitého řadiče na jednodušší nebo specializované, které vedou na úspornější realizaci. I náš případ by bylo možné rozložit na dvě úrovně, které řeší zvlášť přestavování polohy obrobku a zvlášť řízení vřetena do řezu. Výsledný počet stavů by byl menší a patrně by instrukce **STE** mohla být užitečnou - tento případ zde nebudeme řešit.

Počáteční stav časovače byl vynulován po zapnutí systému. Řadič pak realizujeme následujícím programem.

```

B08106.mos
#reg byte vstupy
;.0: start - ruční start cyklu řadiče po založení dílce
;.1: vlevo - nad dírami č.1 nebo 4
;.2: vpravo - nad dírami č.2 nebo 3
;.3: vzadu - nad dírami č.1 nebo 2
;.4: vpředu - nad dírami č.3 nebo 4
;.5: bezpečná výška - bezpečná výška pro přesuny nástroje
;.6: pracovní výška - odtud se začíná vrtat pracovním posuvem
;.7: dno - hloubka dna díry
;
#reg byte vystupy
;.0: rychloposuvem dolů
;.1: pracovním dolů
;.2: rychloposuvem nahoru
;.3: doprava
;.4: doleva

```

## 8. Tabulkové instrukce

```

;.5: dopředu
;.6: dozadu
;.7: upni
;
#reg byte stav
#reg bit prechod, pretečení
;
#table byte masky = %00000001, ; 0 - start
 %01000000, ; 1 - pracovní výška
 %10000000, ; 2 - dno
 %00100000, ; 3 - bezpečná výška
 %00000100, ; 4 - vpravo
 %01000000, ; 5 - pracovní výška
 %10000000, ; 6 - dno
 %00100000, ; 7 - bezpečná výška
 %00010000, ; 8 - vpředu
 %01000000, ; 9 - pracovní výška
 %10000000, ;10 - dno
 %00100000, ;11 - bezpečná výška
 %00000010, ;12 - vlevo
 %01000000, ;13 - pracovní výška
 %10000000, ;14 - dno
 %00100000, ;15 - bezpečná výška
 %00001000, ;16 - vzadu
 %11111111 ;17 - splněná podmínka
;
#table byte hodnoty = %00000000, ; 0 - uvolněno, čeká na start
 %10000001, ; 1 - upni, rychloposuvem dolů
 %10000010, ; 2 - upni, pracovním dolů
 %10000100, ; 3 - upni, rychloposuvem nahoru
 %10001000, ; 4 - upni, doprava
 %10000001, ; 5 - upni, rychloposuvem dolů
 %10000010, ; 6 - upni, pracovním dolů
 %10000100, ; 7 - upni, rychloposuvem nahoru
 %10100000, ; 8 - upni, dopředu
 %10000001, ; 9 - upni, rychloposuvem dolů
 %10000010, ;10 - upni, pracovním dolů
 %10000100, ;11 - upni, rychloposuvem nahoru
 %10010000, ;12 - upni, doleva
 %10000001, ;13 - upni, rychloposuvem dolů
 %10000010, ;14 - upni, pracovním dolů
 %10000100, ;15 - upni, rychloposuvem nahoru
 %01000000, ;16 - dozadu
 %00000000 ;17 - nic
;
P 0
LD stav ;stav řadiče
LTB masky ;masky podmínek
AND vstupy ;podmínka přechodu
WR %S0.3 ;vstupní přenos CI
WR prechod ;vlastní příznak přechodu
POP 1 ;vyrovnání zásobníku
ADD 0 ;přechod
WR stav ;aktualizace stavu
LTB hodnoty ;hodnoty výstupů
WR vystupy
LDC %S1.0 ;příznak přetečení přes tabulku
WR pretečení ;vlastní příznak přetečení řadiče

```

RES stav ;při přetečení vynulování stavu  
E 0

Krokovému řadiči odpovídají dvě tabulky (*masky* a *hodnoty*). Indexem pro výběr položek je *stav* - obě tedy mají 18 položek, obě mají bytový formát.

Prvá tabulka *masky* přiřazuje k číslu stavu masku pro výběr podmínky přechodu z bytu *vstupy*. Jednička v položce označuje pozici proměnné, která rozhoduje o přechodu. V našem případě mají všechny položky právě jedinou jedničku. Obecně jich však může být více. V tom případě je podmínkou logický součet OR všech hodnot na označených pozicích. Hodnota podmínky se uloží do vstupního přenosu CI (S0.3) a (je-li to dále potřebné) do nějaké vnitřní proměnné - např. *prechod*. Vlastní přechod provede instrukce **ADD 0**, která při splnění podmínky inkrementuje *stav*.

Druhá tabulka *hodnoty* přiřazuje číslu stavu odpovídající hodnoty výstupů - celé obsahy bytu *vystupy*. Současně s výběrem položky z tabulky *hodnoty* je řešeno i „otáčení řadiče“, tj. přechod z posledního stavu (17) do počátečního (0) - přenos řadiče. Podmíněná inkrementace dovolí změnu do fiktivního stavu č. 18. Při pokusu o výběr této položky dostaneme obsah položky s indexem přepočteným mod 18, tedy správnou položku č. 0. Současně je však nastaven příznak S1.0 = 0, jehož negovanou hodnotu využijeme k vynulování paměti stavu a (pokud je pro další postup žádoucí) nastavení příznaku přenosu, např. *pretečení* - příznakový registr v S1 je k tomu nevhodný, protože jej ovládají tabulkové instrukce.

V našem „školském“ příkladu jsou položky tabulky výstupů *hodnoty* obsazeny poměrně řídké a určité posloupnosti položek se v ní opakují. Ve skutečných případech může být závislost výstupních vektorů na stavech komplikovanější a užitečnost tabulky výraznější.

Realizace krokových řadičů nebo složitějších sekvenčních systémů může být spojena s větším počtem tabulek, které např. obsahují:

- masky stavových proměnných (v našem příkladu jsme se bez nich obešli)
- návěští akcí stavů

Náš řadič je výrazně orientován na vnější prostředí (vyhodnocuje vstupy, nastavuje výstupy). Mohou být však případy vnitřní orientace, kdy každý stav je provázen jinou akcí, složitější než pouhé nastavení omezeného počtu výstupů (např. ovládání širokého souboru rozptýlených výstupů, řízení časových závislostí, ovládání různých procesů apod.). V takových případech je užitečné pracovat s tabulkou výstupních akcí, která obsahuje čísla návěští, jimiž tyto akce začínají. Pro náš příklad by to bylo účelné například v situaci, kdy by jednotlivé posuny nebyly realizovány „reléově“, ale programem, který ovládá impulzy krokových motorů.

- návěští akcí ošetřujících změny stavů

V našem případě jsme prováděli stejnou akci při přechodu do stavu jako v průběhu setrvávání ve stavu. V některých případech je třeba zvlášť jednorázově ošetřit okamžik přechodu (např. zrušit některé účinky předchozího stavu, připravit podmínky pro nový stav, aktivovat časovače, zrušit stavové masky minulého stavu), ošetřit přenos nebo jiné důležité přechody a zvlášť ošetřit ustálený stav setrvávání ve stavu. Oba typy větvení programu je užitečné provést tabulkami čísel návěští. Někdy postačuje paušálně ošetřit situaci, že došlo k nějakému přechodu nebo že došlo k přenosu.

- časové údaje doby maximálního setrvávání ve stavu (funkce watch dog), přičemž překročení této doby vede na ošetření chyby
- časové údaje minimálního setrvávání ve stavu  
Minimálně po tuto dobu by měl řadič setrvat v novém stavu, a to i tehdy, když podmínka přechodu byla předem splněna.

## 8. Tabulkové instrukce

- časové údaje prodlevy mezi přechody

Po splnění podmínky čeká systém po nastavenou dobu, než provede skutečný přechod a s ním spojené akce. Jde tedy o zdržení okamžiku přechodu. Důvodem je obvykle čas na uklidnění řízeného procesu, na ustálení dojezdů, teplot, apod.

- masky pro součinné podmínky přechodu

V našem příkladu byla podmínkou přechodu jedničková hodnota vstupní proměnné nebo jedničková hodnota logického součtu OR více proměnných. V mnoha případech je požadováno definovat podmínky ve formě součinu označených proměnných, přičemž libovolné z nich mohou být negované - k tomu je třeba dvojice masek pro každý přechod. V našem programu se pak změni začátek:

P 0

```
LD stav ;stav řadiče
LTB polarita ;tabulka masek polarity:
 ;log.1 = přímé, log.0 = negované

XOR vstupy
WR pomoc ;mezivýsledek
POP 1
LTB vyber ;tabulka výběrových masek:
 ;log.1 = uvažuj, log.0 = ignoruj

AND pomoc ;A0 = 0: splněná podmínka
WRC %S0.3 ;vstupní přenos CI
WRC prechod ;vlastní příznak přechodu
POP 1 ;vyrovnání zásobníku
ADD 0 ;přechod
WR stav ;aktualizace stavu
LTB hodnoty ;hodnoty výstupů
WR vystupy
LDC %S1.0 ;příznak přetečení přes tabulku
WR preteceeni ;vlastní příznak přetečení řadiče
RES stav ;při přetečení vynulování stavu
```

E 0

- masky pro kombinované nastavení výstupů

V našem příkladu jsme uvažovali jednoznačné nastavení výstupních proměnných v každém stavu (buď log.0 nebo log.1). Může však být požadováno realizovat „paměťovou funkci“ některých výstupů. To znamená, že některé výstupy mají být nastaveny do log.0, některé do log.1, některé se mají zachovat, jiné negovat, přičemž toto přiřazení se mění stav od stavu.

Obecně k realizaci tohoto požadavku stačí opět dvojice masek (ze dvou tabulek). V našem programu pak změníme zakončení.

P 0

```
LD stav ;stav řadiče
LTB masky ;masky podmínek
AND vstupy ;podmínka přechodu
WR %S0.3 ;vstupní přenos CI
WR prechod ;vlastní příznak přechodu
POP 1 ;vyrovnání zásobníku
ADD 0 ;přechod
WR stav ;aktualizace stavu
LTB vyber ;tabulka masek výběru:
 ;log.1 = pevné nastavení,
 ;log.0 = paměťová funkce

RES vystupy
LDC %S1.0 ;příznak přenosu
WR preteceeni ;vlastní příznak přetečení řadiče
```



```
RES stav ;při přetečení vynulování stavu
LTB akce ;tabulka masek polarity, nebo pravdivosti:
;log.1 = nastav log.1 nebo negace
;log.0 = nastav log.0 nebo neměň

XOR vystupy
WR vystupy
```

E 0

Je pochopitelné, že používání tabulek by nemělo být samoučelným. Nemá smysl dělat z nich povinnost nebo prestižní záležitost. Skutečností však zůstává, že ve většině případů poskytují tabulky optimální řešení. Pouze v případech blízcích se triviálním, bývá výhodnější tradiční postup.

Za zvláštní případ krokového řadiče můžeme považovat:

- časový řadič  
V krajním případě vystačí bez vnější podmínky - všechny podmínky jsou předem splněny. Každému stavu je přiřazena pouze hodnota časovače. Časový řadič potom cyklicky realizuje naprogramovanou posloupnost časových intervalů. Tak jak bylo dosud popsáno, můžeme jednotlivým stavům nebo přechodům přiřadit tabulkami hodnoty výstupů, čísla akcí, apod.
- bubnový řadič, generátor cyklických posloupností:  
Obvykle pracuje úplně autonomně (bez vnějších podmínek) nebo s omezeným počtem proměnných - např.: „o krok vpřed“, „o krok vzad“, „do počátečního stavu“ nebo s výstupem časovače. Podle vnějšího požadavku nebo autonomně mění hodnoty stavů a provádí jim odpovídající akce nebo nastavuje odpovídající posloupnosti výstupů.

Je užitečné si uvědomit, že naopak popsaný krokový řadič je zvláštním případem složitějších sekvenčních systémů. Nejčastěji se setkáváme s konečnými automaty různých typů (Moore, Mealy, zádržový, kombinační). Pro ně je typické, že nemají jednoznačně určenou posloupnost stavů, ale že z jednoho stavu existuje více možných přechodů - o novém stavu rozhoduje hodnota současného stavu a kombinace vstupních proměnných (podmínek). Hodnoty výstupních proměnných, případně typ akce, hodnoty časovačů apod. mohou opět záviset na současném stavu a kombinaci vstupů (Mealyho automat) nebo jen na hodnotě stavu (Mooreův automat).

Se zádržovými automaty jsme se setkali v příkladech s vyhodnocením sledu hran při rozlišení polohy a směru nebo u fázového diskriminátoru. Zde je informace o stavu nahrazena hodnotou minulého vzorku vstupních proměnných, případně i hodnotami starších vzorků.

Kombinační automat naopak určuje hodnoty výstupů nebo akcí, jednoznačně podle kombinace vstupních proměnných. Není tedy nutno pracovat s vnitřním stavem (viz příklad pravdivostní tabulky).

Reálné systémy jsou kombinacemi různých typů. Je třeba ještě zdůraznit, že rozsahy tabulek rostou exponenciálně se složitostí úlohy (s šířkou dat, tj. s počtem bitů vstupních a stavových proměnných). Tabulková realizace je užitečná pouze do určité složitosti, dále narůstají tabulky do „obludných rozměrů“. V těchto případech však pomůže rozklad složitějšího systému na několik jednodušších. Počet tabulek vzroste, ale jejich souhrnný objem může být výrazně menší.

Při řízení reálných soustav se často setkáváme s případem, kdy systém nebo program se vyvíjí s větším počtem současně aktivních stavů. Typickým případem je řízení několika nezávislých pracovišť, obsluhování toku materiálu mezi nimi, včetně synchronizace a čekání na kritických místech apod. Pro popis jsou velmi výhodné tzv. Petriho sítě. Jejich modifikací je pak grafický programovací jazyk GRAFCET. Při vlastní realizaci na úrovni jazyka mnemokódů pak obvykle postupujeme obdobně, jako při realizaci sekvenčních řadičů nebo automatů, aktivujeme je však vícekrát s různými hodnotami stavů. Navíc pak

musíme vyřešit úlohu pamatování měnícího se počtu stavů, zachování jejich priorit, vzájemných vazeb mezi přechody, zmnožování nebo redukování počtu aktivních stavů.

### 8.2. Čtení a zápis do tabulek v zápisníku (LTB, WTB)

Instrukce **LTB** umožňuje čtení z tabulky v zápisníku a instrukce **WTB** zápis do tabulky v zápisníku ve formátech bit, byte, word.

#### Příklad 8.2.1

Uživatelský program obsluhuje stejným způsobem čtyři stejné procesy č. 0, 1, 2, 3. Vstupní proměnné procesů jsou soustředěny v bytech *vstup0*, *vstup1*, *vstup2*, *vstup3*. Každý proces má jedinou výstupní proměnnou, *vystup0* až *vystup3*, přičemž pro proces *n* platí:

$$vystupn = vstupn.7 + vstupn.6 + vstupn.5 + vstupn.4 + vstupn.2 + vstupn.1$$

Předpokládejme, že číslo obsluhovaného procesu je v proměnné *proces*, jejíž obsah byl nastaven dříve. Vstupy a výstupy jsou řazeny za sebou vzestupně, takže tvoří tabulku.

*B08201.mos*

```
#reg byte vstup0, vstup1, vstup2, vstup3
#reg bit vystup0, vystup1, vystup2, vystup3
#reg byte proces
;
P 0
 LD 3 ;MEZ = poslední proces
 LD proces ;INDEX = číslo procesu
 LTB vstup0 ;odpovídající vstupy
 XOR %00110000 ;log.1 = negované vstupy
 AND %11110110 ;log.1 = zastoupené vstupy
 WTB vystup0 ;zapiš příslušný výstup
E 0
```

V typických případech bychom tento úsek programu uzavřeli do smyčky se čtyřmi průchody (s obsahem *proces* = 0 až 3). V jednom cyklu uživatelského programu jsou tedy obslouženy všechny procesy. Pokud doba odezvy není kritická, je možné zvolit takový způsob aktivace, že v každém cyklu uživatelského programu je obsloužen jeden proces. Pro čtyři procesy (obecně pro jakýkoliv počet =  $2^k$ ) můžeme pak místo *proces* pracovat se systémovým registrem S4, který čítá cykly uživatelského programu mod 256.

#### Příklad 8.2.2

Předpokládejme předchozí příklad s tím rozdílem, že funkce výstupu je popsána pravdivostní tabulkou *funkce* (256 bitových položek, tedy 32 bytů).

*B08202.mos*

```
#reg byte vstup0, vstup1, vstup2, vstup3
#reg bit vystup0, vystup1, vystup2, vystup3
#reg byte proces
#reg bit pomoc
;
#table byte funkce = %00011100,%00110011,%01100011,%10001101,%00111100,
 %11011100,%00001001,%11111100,%00110100,%10001110,
 %10010001,%10011001,%10111100,%01011100,%10001001,
 %11011001,%01101100,%11001110,%11010001,%10001001,
 %00110100,%11111100,%00101001,%11110001,%00110000,
```

```

 %10011110,%10000001,%10011101,%00111110,%11010100,
 %00000001,%11111011
;
P 0
 LD 3 ;mez
 LD proces ;číslo procesu
 LTB vstup0 ;vstupní vektor
 LTB funkce.0 ;výstupní funkce
 WR pomoc ;odložení výstupu
 LD 3 ;mez
 LD proces ;číslo procesu
 LD pomoc ;výstup
 WTB vystup0 ;uložení
E 0

```

Zatímco v předchozím příkladu tvořily instrukce **LTB** vstupy pro **WTB**, v našem případě je tato návaznost porušena. Musíme proto výstup z tabulky *funkce* odložit (např. do bitu *pomoc*), znovu nastavit mez, číslo procesu a výstupní hodnotu do úrovně zásobníku A2 až A0 a teprve potom provést zápis do indexované výstupní proměnné.

### Příklad 8.2.3

Opět předpokládejme, že postupně obsluhujeme čtveřici procesů č. 0 až 3. Nyní však jsou každému přiřazeny 3 vstupní proměnné a 8 výstupních se vzestupným přiřazením. Výstupní vektorová funkce je popsána bytovou tabulkou *funkce*.

č.0: vstup00 až vstup20, vystup0  
 č.1: vstup01 až vstup21, vystup1  
 č.2: vstup02 až vstup22, vystup2  
 č.3: vstup03 až vstup23, vystup3

```

B08203.mos
#reg bit vstup00, vstup10, vstup20
#reg bit vstup01, vstup11, vstup21
#reg bit vstup02, vstup12, vstup22
#reg bit vstup03, vstup13, vstup23
#reg byte vystup0, vystup1, vystup2, vystup3
#reg byte proces, pomoc
;
#table byte funkce = %00011100,%00110011,%01100011,%10001101,
 %00111100,%11011100,%00001001,%11111001
;
P 0
 LD 11 ; (počet procesů * 3) - 1
 LD proces
 MUL 3
 LTB vstup00 ;bit 0
 WR pomoc.0
 POP 1
 INR
 LTB vstup00 ;bit 1
 WR pomoc.1
 POP 1
 INR
 LTB vstup00 ;bit 2
 WR pomoc.2
 LD pomoc ;vstupní vektor procesu
 LTB funkce ;výstupní funkce

```

```

WR pomoc ;odložení
LD 3 ;počet procesů - 1
LD proces
LD pomoc
WTB vystup0 ;uložení výstupního vektoru
E 0

```

**Příklad 8.2.4**

Číselný údaj v proměnné *cislo* (rozsah 5 bitů) máme převést na masku 1 z 32 v proměnné *kod*.

```

B08204.mos
#reg byte cislo
#reg long kod
;
P 0
 LD 0
 WR kod ;vynulování
 LD 31
 LD cislo ;číslo pozice
 LD 1 ;zápis jedničkové masky
 WTB kod.0
E 0

```

Prostor pro masku nejprve vynulujeme a pak na odpovídající pozici zapíšeme jedničku. Zde využíváme skutečnosti, že bitová instrukce **WTB** (ale i **LTB**) převádí číslo odpovídající pozici bitu. Postup můžeme použít např. k zápisu stavové masky ke stavu krokového řadiče nebo obecnějšího sekvenčního automatu. Podobně můžeme realizovat funkci demultiplexeru s tím rozdílem, že místo konstanty 1 bychom zapisovali odpovídající proměnnou.

**Příklad 8.2.5**

Předpokládejme realizaci sekvenčního řadiče s tím, že je přípustné, aby současně bylo několik stavů aktivních. Máme při každém přechodu aktualizovat hodnoty stavových bitů, odpovídajících aktivním stavům.

Zde si nemůžeme dovolit všechny stavové bity vynulovat a nastavit do jedničky jedinou novou stavovou proměnnou. Při přechodu musíme nejprve vynulovat stavovou proměnnou odpovídající opuštěnému stavu - provedeme to analogicky jako v předchozím příkladu, pouze se zápisem hodnoty 0. Potom zapíšeme hodnotu 1 na pozici odpovídající nově aktivovanému stavu.

**Příklad 8.2.6**

Při realizaci sekvenčních úloh často potřebujeme pracovat s minulým vzorkem proměnných, případně se vzorky staršími. Například budeme požadovat, aby v každém cyklu byl ukládán minulý vzorek vektoru X0. Kromě tradičního postupu, kdy vše přeskládáme na nová místa, např.

```

LD hodnota ;starý
WR zaloha
LD vstup ;nový
WR hodnota

```

Můžeme ponechat obsahy na svých místech, pouze jednou očekáváme starý obsah v jedné pozici, zatímco příště na druhé pozici, atd.

```

B08206.mos
#reg byte vstupý
#reg byte vzorky[4], index
;
P 0
 LD 3 ;MEZ - 4 položky
 LD index
 LD vstupý
 WTB vzorky ;nejnovější vzorek
 POP 1
 INR
 WR index ;uložení zvýšeného indexu
 LTB vzorky ;nejstarší vzorek
 JS konec
 LD 0 ;zacyklení tabulky
 WR index
konec:
E 0

```

Tímto způsobem zapisujeme nový vzorek postupně do pole registrů *vzorky* (které tvoří bytovou tabulku o čtyřech položkách) a z následující položky je čtena hodnota nejstaršího vzorku.

Postup je obecný a může být výhodný ve složitějších případech - např. při ukládání indexovaných vzorků pro indexované proměnné, nebo při ukládání vzorků do více vrstev.

### Příklad 8.2.7

Je požadováno realizovat zpožďovací linku pro vzorky vektoru *vstup0* do hloubky 16 úrovní. Zde již bude tradiční postup s přesouváním neefektivní oproti předchozímu, který nechává vzorky na svých místech a cyklicky mění hodnoty indexů, kterými ovládá zápis a čtení. Namísto rolování obsahů roluje indexy.

```

B08207.mos
#reg byte vstupý
#reg byte vzorky[16], index, zpozdeni
;
P 0
 LD 15 ;MEZ
 LD index
 LTB vzorky ;16 úrovní zpět
 WR zpozdeni ;nejstarší hodnota
 POP 1
 LD vstupý ;současný vzorek
 WTB vzorky
 JS konec
 LD 0 ;zacyklení tabulky
 WR index
konec:
 INR index
E 0

```

Program postupně zapisuje vzorky *vstupý* do pole registrů *vzorky* stále dokola. Položka adresovaná současným stavem čítače *index* obsahuje (před uložením současného vzorku) hodnotu vzorku, starého 16 cyklů. Podle potřeby můžeme ze záznamu vybírat vzorky různého stáří. Například při inkrementaci indexu postupně adresujeme vzorek starý 15, 14, 13, ... cyklů. Pokud výchozí hodnotu *index* dekrementujeme, získáváme vzorky staré 1, 2, 3, ... cykly. Podle potřeby tedy můžeme realizovat různé typy zásobníku. V prvním případě

je typu FIFO (first in - first out, první tam - první ven), ve druhém případě je typu LIFO (last in - first out, poslední tam - první ven).

### 8.3. Hledání hodnot v tabulkách (FTB, FTM)

Tabulkové instrukce **FTB**, **FTM** umožňují prohledávání obsahu tabulek T a tabulek v zápisníku ve formátech bit (jen **FTB**), byte a word.

#### Příklad 8.3.1

Realizujeme převodník kódu 2 z 5 na dvojkové číslo.

Můžeme zadat tabulku o délce 32 bytů, která přiřazuje deseti obsazeným kódům 2 z 5 hodnotu odpovídajícího dvojkového čísla. Zbývajících 22 položek, které odpovídají mimokódovým kombinacím, mohou mít náhodný obsah - tedy např. 0 nebo kód (např. 255 nebo 10), který signalizuje chybu.

Pokud však nespěcháme, nemusíme zadávat žádnou novou tabulku a použít tu, kterou jsme již zadali v příkladu 8.1.2 pro řešení opačné úlohy. Pak stačí jednoduchý postup:

```
B08301.mos
#reg byte cislo, kod
#table byte tab = %00110,%10001,%01001,%11000,%00101,
 %10100,%01100,%00011,%10010,%01010
;
P 0
 LD kod ;kombinace 2 z 5
 FTB tab
 WR cislo ;výstup
```

Pokud bylo hledání neúspěšné (byla zadána mimokódová kombinace) je výstup 10. Kdybychom tuto situaci chtěli ošetřit jinak, např. *cislo* = 255, museli bychom doplnit další dvě instrukce.

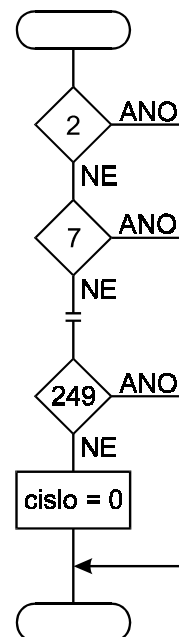
```
 LDC %S1.0
 SET cislo
E 0
```

#### Příklad 8.3.2

Pokud *cislo* nabývá některou z hodnot: 2, 7, 15, 27, 95, 153, 249, pak *cislo* neměňme, jinak jej vynulujeme.

Tradičně se tato úloha řeší větvením programu dle vývojového diagramu vpravo. Tomu odpovídá program:

```
B08302a.mos
#reg byte cislo
;
P 0
 LD cislo
 CMP 2
 JZ konec
 CMP 7
 JZ konec
 :
 :
 CMP 249
 JZ konec
 LD 0
```



```

WR cislo
konec:
E 0

```

Stejnou úlohu můžeme úsporněji, přehledněji a rychleji řešit programem s tabulkovou instrukcí **FTB**:

```

B08302b.mos
#reg byte cislo
#table byte tab = 2,7,15,27,95,153,249
;
P 0
 LD cislo
 FTB tab
 LDC %S1.0
 RES cislo
E 0

```

Položky tabulky *tab* tvoří soubor testovaných číselných hodnot - v našem případě je tabulka bytová, pokud bychom testovali čísla nad 255, byla by formátu word.

Porovnáme-li oba vztahy na délku a vezmeme-li v úvahu, že i tabulka, s kterou nepracujeme, zabírá 4 byty (prázdná tabulka), dojdeme k paradoxnímu závěru, že druhé řešení je úspornější vždy - i při jediném testu. V praxi patrně nebudeme tak přehnaně úsporní, abychom definovali tabulku s jedinou položkou. Z hlediska doby vykonávání programu bude hranice výhodnosti ležet výše. Přesné doby výkonu instrukcí jsou dány typem centrální jednotky a jsou uvedeny v uživatelských příručkách k jednotlivým typům PLC. Lze však odhadnout, že tabulkové řešení bude rychlejší od třech testovaných hodnot.

### Příklad 8.3.3

Je požadováno větvení programu podle hodnoty *cislo*. Platí následující pořadové přiřazení návěstí k hodnotám:

```

cislo = 2 je2
 = 7 je7
 = 15 je15
 = 27 je27
 = 95 je95
 = 153 je153
 = 249 je249
jinak nic

```

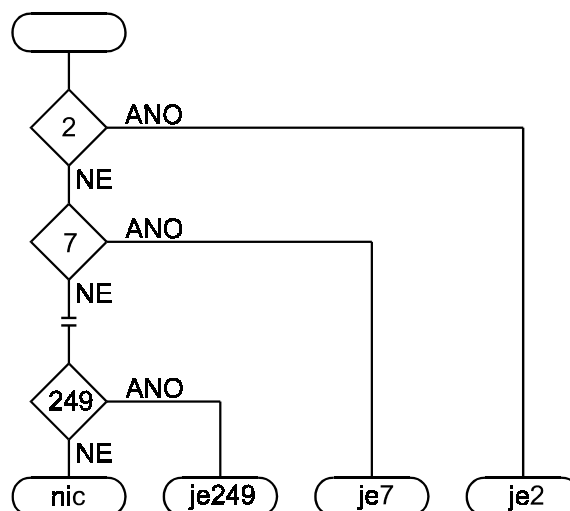
Strukturu programu lze popsat vývojovým diagramem uvedeným vpravo.

Tradičně bychom problém řešili postupem:

```

B08303a.mos
#reg byte cislo
;
P 0
 LD cislo
 CMP 2
 JZ je2
 CMP 7
 JZ je7
 :
 :
 CMP 249
 JZ je249

```



```
 JMP nic
;
je2:
:
je7:
:
:
je249:
:
nic:
:
konec:
E 0
```

Stejnou úlohu můžeme řešit s pomocí tabulky:

```
B08303b.mos
#reg byte cislo
#label 0,je2,je7,je15,je27,je95,je153,je249,nic
#table byte tab = 2,7,15,27,95,153,249
;
P 0
 LD cislo
 FTB tab
 JMI
;
je2:
:
je7:
:
:
je249:
:
nic:
:
konec:
E 0
```

Instrukce **FTB** vynese pořadové číslo položky, pokud byla nalezena, na vrchol zásobníku (hodnotě 2 přiřadí 0, hodnotě 7 přiřadí 1, ..., hodnotě 249 přiřadí 6). Pokud není nalezena žádná z testovaných hodnot, uloží na vrchol zásobníku hodnotu 7 (mez + 1). Protože návěští cílů skoků (nebo volání) jsou číslována pořadově od 0 (deklarace *#label*), lze tento výsledek použít jako vstup pro instrukci **JMI** nebo **CAI**. Pokud by bylo nutno návěští deklarovat pořadově od jiné hodnoty než 0 (např. 13), stačí k hodnotě na vrcholu zásobníku před instrukcí **JMI** nebo **CAI** přičíst toto číslo počátečního návěští (např. **ADX 13**).

#### **Příklad 8.3.4**

Zadání předchozího příkladu změňme tak, že návěští, na která přechází program podle testovaných hodnot nejsou číslována pořadově. Důvodem může být vícenásobné použití stejných návěští.

Tradiční postup se nezmění. Tabulkový postup rozšíříme o instrukci **LTB**, která k indexu nalezené položky přiřadí číslo návěští.



```

B08304.mos
#reg byte cislo
#label je2,je7,je15,je27,je95,je153,je249,nic
#table byte tab = 2,7,15,27,95,153,249
#table byte navesti = __indx (je2), __indx (je7), __indx (je15),
 __indx (je27), __indx (je95), __indx (je153),
 __indx (je249), __indx (nic)

;
P 0
 LD cislo
 FTB tab
 LTB navesti
 CAI
E 0
;
P 60
je2:
 RET
;
:
nic:
 RET
E 60

```

Oproti předchozímu příkladu jsme navíc zaplatili pouze doplněním transformační tabulky *navesti*. Všimněme si, že tato tabulka je o položku delší než *tab*. Přirazuje totiž hodnotu i pro případ nenalezené položky. Návěští, jejichž seznam tabulka obsahuje, je nutné předem nadeklarovat pomocí direktivy *#label*, není však nutné zadávat číslo prvního návěští, ani dodržet pořadí návěští.

### Příklad 8.3.5

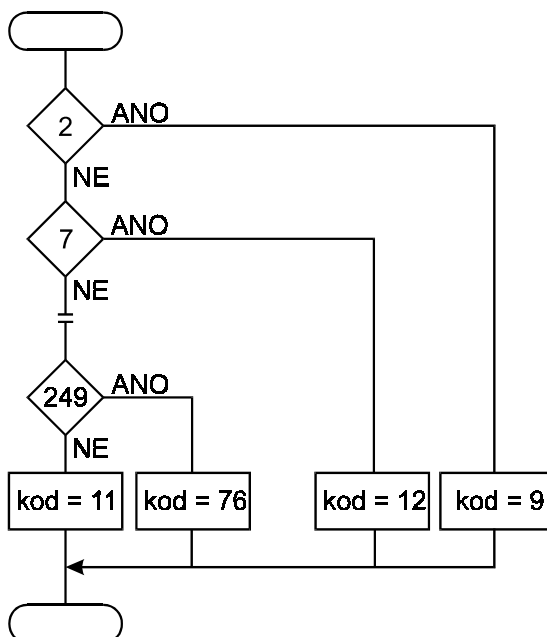
Řešme úlohu popsanou vývojovým diagramem vpravo.

Je obdobou úloh z příkladů 8.3.2, 8.3.3 a 8.3.4. Tradičním postupem ji lze převést na větvení programu. Po rozvětvení některým ze způsobů popsaných v příkladech 8.3.3, 8.3.4 můžeme doplnit jednotlivé akce:

```

B08305a.mos
#reg byte cislo, kod
#label 0,je2,je7,je15,je27,je95,je153,je249,nic
#table byte tab = 2,7,15,27,95,153,249
;
P 0
 LD cislo
 FTB tab
 CAI
 WR kod
E 0
;
P 60
je2:
 LD 9
 RET
;
je7:
 LD 12
 RET
;

```



```

:
;
je249:
 LD 76
 RET
;
nic:
 LD 11
 RET
E 60

```

Následující postup je v tomto případě podstatně výhodnější:

```

B08305b.mos
#reg byte cislo, kod
#table byte tab = 2,7,15,27,95,153,249
#table byte seznam = 9,12,17,159,231,67,76,11
;
P 0
 LD cislo
 FTB tab
 LTB seznam
 WR kod
E 0

```

### Poznámka k metodice:

Uvědomíme-li si podstatu problému (často bývá užitečné si uvědomit skutečnou podstatu problému), pak celý složitě strukturovaný (a přehlednost postrádající) program neřeší nic víc, než pouhou transformaci dat - tu lze řešit velmi efektivně dvojicí tabulek a program zůstane zcela lineární (nevětvený). O výhodách tabulkového řešení zde není třeba diskutovat.

Je třeba upozornit, že podobné situace se vyskytují často, a to nejenom při programování logických systémů, ale při programování libovolného počítače. Analyzujeme-li podstatu složitě větvených programů, pak téměř vždy dojdeme k závěru, že spočívá v transformaci dat - pro tu existují daleko efektivnější prostředky ve všech programovacích jazycích. S jejich využitím vyřešíme úlohu buď zcela bez větvení nebo s větvením výrazně redukováným. Nemusí být vždy použita tabulka. Transformací dat jsou i logické nebo numerické funkce. Ve většině případů je větvení programu tím nejméně vhodným prostředkem pro transformaci dat.

Lze jej doporučit pouze pro nejjednodušší případy strukturování programu, např. podmíněný přeskok části programu, dvou nebo vícecestné větvení do větví (nebo podprogramů), z nichž každá provádí jiný postup operací - nic nelze vytknout nebo by sjednocování bylo násilné.

V každém případě je třeba varovat před programy, jejichž vývojový diagram je velkoplošný výkres, v němž jsou spoje vedeny všemi směry, odkudkoliv kamkoliv. Není šikovným programátorem ten, který dovede vytvořit takto složitý program (a možná jej i odladí a bude udržovat), ale ten, který složitou úlohu realizuje lineárním nebo dobře strukturovaným programem. Obdobné tvrzení lze vyslovit i o složitosti a struktuře logických nebo reléových schémat pro pevnou logiku. Nestrukturovaným postupem (vytvářením složitě propletených struktur) jsme nejvíce ohroženi zejména v případech, kdy nám jako předloha programu slouží logické nebo reléové schéma existujícího nebo obdobného zařízení nebo pokud nejsme schopni se oprostit od stereotypů návrháře pevné logiky.

Před programováním se vyplatí věnovat určitý čas analýze úlohy, oprostit se od způsobu „jak se to dělalo tradičně“, uvědomit si podstatu úlohy („co to má dělat“, „jak to má pracovat“), navrhnout vlastní algoritmus řešení a popsat jej obecnými prostředky - přecho-

dovým diagramem sekvenčního řadiče nebo automatu, jazykem GRAFCET, pravdivostními tabulkami kombinačních funkcí, dvojicemi přiřazení vstup - výstup, časovými diagramy, slovními poznámkami, některým z vyšších programovacích jazyků a podobně. Rozhodně není tato přípravná etapa (byť stála sebevíce času) zbytečnou. Obvykle se projeví slabá místa v původním zadání, nedostatečné pochopení problematiky nebo možnosti vylepšeného nebo zobecněného zadání. Teprve po precizním zadání má smysl začít uvažovat o podrobnostech programu, o rozdělení úkolů mezi více programátorů. Zní to paradoxně, ale právě v situaci časové tísně se vyplatí obětovat část pracovního času solidní analýze problému, vzdor nevraživým pohledům vedoucích a dotěrným otázkám zadavatele typu „kdy to bude?“. Ušetří se tím mnoho času „zaplétáním struktury“ během programování a při jejím „rozplétání“ během ladění, při každé změně zadání, rozšíření funkcí nebo při inovaci. Někdy se tím šetří čas a náklady spojené s obětováním neúspěšné varianty řešení.

### Příklad 8.3.6

Výstup *kontakt* ovládáme denně podle následujících pravidel:

*kontakt* = log.1 (zapnuto)  
 od 5.50 do 7.30,  
 od 7.45 do 9.25,  
 od 9.55 do 10.30,  
 od 11.41 do 13.50,  
 od 15.49 do 17.15,  
 od 21.13 do 23.17,  
 jinak je *kontakt* = log.0 (vypnuto).

Doufejme, že po předchozí metodické poznámce, odolá čtenář pokušení použít tradiční postup s větvením programu. Odoláme i pokušení měřit čas některou z instrukcí časovačů a využijeme údaje systémového reálného času v systémových registrech S7 (minuty) a S8 (hodiny).

Můžeme použít postup s dvojicí tabulek - *zap* obsahuje zapínací časy, *vyp* obsahuje vypínací časy.

*B08306a.mos*

```
#reg bit kontakt
#table byte zap = 50,5,45,7,55,9,41,11,49,15,13,21
#table byte vyp = 30,7,25,9,30,10,50,13,15,17,17,23
;
P 0
 LD %SW7 ;A0H = hodiny, A0L = minuty
 FTB word zap ;zapínací časy
 LD %S1.0
 SET kontakt ;když našel, zapne
 LD %SW7
 FTB word vyp ;vypínací časy
 LD %S1.0
 RES kontakt ;když našel, vypne
E 0
```

Tabulky *zap* a *vyp* jsou sice v programu interpretovány ve formátu word, ale zadávání časových položek v šířce word by bylo zbytečně pracné - hodinový údaj bychom museli násobit 256 a přičíst minutový. Proto je deklarujeme bytově, v prvním bytu minuty, v následujícím hodiny. Časy však můžeme sestavit (proložit) do jediné tabulky.

## 8. Tabulkové instrukce

```

B08306b.mos
#reg bit kontakt
#table byte cas = 50,5,30,7,45,7,25,9,55,9,30,10,41,11,50,13,49,15,
 15,17,13,21,17,23
;
P 0
 LD %SW7 ;A0H = hodiny, A0L = minuty
 FTB word cas
 LD %S1
 SET kontakt ;když našel, předběžně zapne
 AND
 RES kontakt ;když našel a je lichý index, vypne
E 0

```

### Příklad 8.3.7

Předpokládejme, že máme realizovat složitější časový program zadaný tabulkovou formou - pro tento typ úlohy se někdy používá termín časový procesor:

| čas (den, hod., min.) | vystup.7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 | akce   |
|-----------------------|----------|----|----|----|----|----|----|----|--------|
| po, 5.15              | 1        | D  | D  | 1  | D  | D  | 0  | 1  | akce1  |
| po, 5.45              | 1        | 1  | 0  | 1  | C  | D  | 0  | 0  | -      |
| po, 6.10              | 1        | 1  | 1  | 1  | 0  | 1  | 0  | 1  | -      |
| po, 6.31              | 0        | 1  | 0  | 1  | 1  | 0  | 1  | D  | akce2  |
| .....                 |          |    |    |    |    |    |    |    | .....  |
| ne, 17.30             | 0        | 0  | D  | 1  | 0  | D  | 1  | 0  | akce15 |
| ne, 18.59             | 0        | 0  | D  | 0  | 0  | 0  | 0  | C  | -      |
| ne, 23.14             | 0        | 0  | D  | 0  | 0  | 0  | 0  | D  | akce16 |
| v mezičasech          | D        | D  | D  | D  | D  | D  | D  | D  | -      |

Symbody ve sloupci výstupů mají význam:

- 0, 1 - hodnoty přímo zapisované na výstupy
- D - nemění se hodnota výstupu („direct“)
- C - hodnota výstupu se neguje („complement“)

Hodnota log.1 v kombinaci s paměťovou funkcí odpovídá instrukci **SET**, hodnota log.0 odpovídá **RES**. Symbody C se uplatní pouze jednorázově při novém souhlasu s dalším časovým údajem. Údaje ve sloupci „akce“ mají význam návěští, kterým je ošetřena akce v odpovídajícím okamžiku. Proškrtnutí má význam paušálního ošetření akcí *nic* (např. prázdná akce). Obsahy položek jsou sestaveny náhodně, pouze pro ilustraci, neodpovídají žádnému reálnému procesu.

Přiřazení dnů v týdnu je následující:

- po - S9 = 1
- út - S9 = 2
- .....
- ne - S9 = 7

Při možnosti práce s tabulkami není úloha tak složitá, jak se zdá. Všimněme si, že k zobrazení hodinového údaje (0 - 23) postačuje 5 bitů. Zbývající horní bity v horní části položky můžeme využít k zobrazení dnů v týdnu (1 - 7).

Tabulka s časovými údaji *cas* bude tedy obsahovat položky s následující strukturou. V prvním bytu budou minuty, v následujícím bytu budou hodiny na spodních pěti bitech a na horních třech bitech budou dny (hod. + 32 \* den). Na pořadí položek nezáleží.

Tabulky *nastav*, *polarita*, *akce* mají o položku více než *cas*.

Položky tabulky *nastav* rozlišují pozice, které budou jednoznačně nastavené od položek s paměťovou funkcí. Na pozicích se symbody 1, 0 mají položky log.1, na pozicích se symbody D, C mají položky log.0.

Položky tabulky *polarita* rozlišují symboly 0 od 1 a D od C. Na jedničkových pozicích položky bude buď nastavena hodnota log.1, nebo bude pamatovaný údaj negován (C). Na nulových pozicích bude zapsána hodnota log.0, nebo se pamatovaný údaj nezmění (D).

Položky tabulky *akce* obsahují číslo návěští akcí, které odpovídají začátkům zadaných časových okamžiků nebo začátku mezičasu.

*B08307.mos*

```
#reg byte vystup
#reg byte index, indexs
#label akce1,akce2,...,akce15,akce16,nic
#table byte cas = 15,37,45,37, ;po, 5.15, 5.45
 10,38,31,38, ;po, 6.10, 6.31
 ...
 30,241,59,242,14,247 ;ne, 17.30, 18.59, 23.14
#table byte nastav = %10010011,%11110011,%11111111,%11111110,
 ...
 %11011011,%11011110,%11011110,%00000000
#table byte polarita = %10010001,%11011000,%11110101,%01011010,
 ...
 %00010010,%00000001,%00000000,%00000000
#table byte akce = __indx (akce1), __indx (nic), __indx (nic),
 __indx (akce2), ...
 __indx (akce15), __indx (nic), __indx (akce16),
 __indx (nic)

;
P 0
 LD index
 WR indexs ;minulý index
 LD %SW7 ;A0L = min, A0H = hod
 LD %S9 ;den v týdnu
 MUL 32 ;posun o 5 pozic vlevo
 SWP
 OR
 FTB word cas
 WR index
 XOR indexs
 JMC konec ;pokud se nemění, skončit
 LD index ;index
 LTB nastav ;tabulka nastavených výstupů:
 RES vystup ;log.1 = nastavené výstupy,
 ;log.0 = výstupy s pamětovou funkcí

 POP 1
 LTB polarita ;tabulka polarity
 XOR vystup ;log.1 = nastav log.1 nebo C,
 ;log.0 = nastav log.0 nebo D

 WR vystup
 POP 1
 LTB akce ;tabulka návěští
 CAI akce2 ;proved' akci
 ;společné pokračování
konec:
E 0
;
P 60
akce1:
 RET
;
akce2:
 RET
```

```
;
 :
;
akcel15:
 RET
;
akcel16:
 RET
;
nic:
 RET
E 60
```

Zadání úlohy se jeví složité, ale není maximalistické. Můžeme si ještě dále „vymýšlet“. Ve sloupci výstupů můžeme kromě symbolů 0, 1, D, C, doplnit ještě např. I („impulz“), který pro daný výstup předepisuje jedničkový impulz, jehož délku lze zadat (pokud možno zvlášť pro každou pozici - stačí bytová tabulka, jejíž položky pro časovou jednotku 10 s poskytují rozpětí časů od 0 do 42 min. 30 s).

Kromě zadaného týdenního cyklu můžeme zadat a realizovat:

- každodenní cyklus (např. hod., min., 2 s)
- roční cyklus
- sezónní cykly (např. prázdninový cyklus, celozávodní dovolenou, topné období, atd.)
- soubor jednorázových událostí (např. svátky, přechod na zimní nebo letní čas, atd.)

Viz též příklad 8.3.15.

### **Příklad 8.3.8**

Obdobně jako příklady 8.3.6 a 8.3.7 řešily úlohu programování událostí v absolutním čase, může být požadováno obdobně programování v relativním čase - např. od okamžiku spuštění technologického procesu, od dosažení některého stavu sekvenčního řadiče, od zadané kombinace vstupů, apod.

Postup je prakticky stejný jako v příkladech 8.3.6 a 8.3.7 s tím rozdílem, že musíme nějak získat údaj času odvozeného od sledované události. Můžeme použít údaj některého typu časovače. Postup je vhodný zejména v případech, kdy počáteční událost lze odvodit od hladiny některé logické proměnné. Každý typ časovače má své přednosti a nedostatky (**TON** a **TOF** přestává časovat při návratu řídicí proměnné do klidové hladiny, **RTO** přitom také přestává časovat, ale časový údaj nemaže, **IMP** je naopak nepřerušitelný). Časovou jednotku je třeba zvolit tak, aby doba cyklu časovače (od začátku časování do přetečení přes maximum) spolehlivě přesahovala rozpětí programovaných časů.

Bez použití časovače lze problém řešit tak, že v okamžiku počáteční události odložíme počáteční stav průběžně se měnícího časoměrného registru (např. SW14, SW16, SW18 nebo vlastního čítače času, který je poháněn zvolenou časovou jednotkou z S13 nebo z S14 až S19 nebo jinými událostmi (např. počet otáček, dílů, cyklů, apod.). V obou případech jsou údaje času nebo počtu zadávány pouze v počtu zvolených jednotek (např. v desítkách sekund), nikoliv přepočteny na vyšší jednotky - minuty, hodiny, dny, ...

### **Příklad 8.3.9**

Vraťme se k případu rozpoznávání profilů (obecně posloupnosti hran signálů) - příklad 3.4.6 k instrukci **STE**. Postup můžeme obměnit tak, že nejprve (při zacyklení alespoň jedné fotobuňky) budeme registrovat posloupnost úrovní a po osvětlení obou buněk porovnáme nastřádaný údaj s referenčními údaji z tabulky:

```

B08309.mos
#reg byte vstupy
#def fotoc vstupy.1
#def fotod vstupy.2
#reg byte vzorek, radac, radad
#label profil_X,profil_Y,profil_Z,chyba
#table word reference = %0000011100000101, ;profil X
 %0001101100001110, ;profil Y
 %1111011111011101 ;profil Z
#table byte akce = __indx (profil_X), __indx (profil_Y),
 __indx (profil_Z), __indx (chyba)
;
P 0
 LD vzorek ;minulý vzorek c, d
 LD vstupy
 AND 6
 WR vzorek ;současný vzorek c, d
 XOR
 JMC konec ;při shodě nic nedělat - nic se nezměnilo
 LD fotoc
 AND fotod
 JMD porovnani ;při mezeře testuj profil
 LD fotoc ;střádání posloupnosti c
 WRC %S0.3 ;vstupní přenos CI
 LD radac
 LD radac
 ADD
 WR radac ;nasunutí
 LD fotod ;střádání posloupnosti d
 WRC %S0.3
 LD radad
 LD radad
 ADD
 WR radad ;nasunutí
 JMP konec ;hotovo
;
porovnani:
 LD word radac ;porovnání profilu
 FTB reference ;negovaná posloupnost c, d
 LTB akce ;porovnání s referencemi
 CAI
 LD 0 ;číslo akce
 WR word radac ;provedení akce
 WR word radac ;vynulování staré posloupnosti
konec:
E 0
;
P 60
profil_X:
 RET
;
profil_Y:
 RET
;
profil_Z:
 RET
;
chyba:
 RET

```

E 60

;

P 63

LD 6

WR vstupy

;přednastavení fotobuněk c, d

WR vzorek

;(mezera)

E 63

Tabulka *reference* obsahuje vzorové posloupnosti úrovní signálů z fotobuněk c, d pro hledané profily - nižší byte položky je posloupností hladin c, vyšší část položky obsahuje posloupnost d. Zaclonění buňky (neprůhledné části profilu) odpovídá 1, nezakryté buňce odpovídá 0. Pořadí vzorků odpovídá pohybu pásu v naznačeném směru. Posloupnost úrovní c, d nesmí být delší než 8 změn.

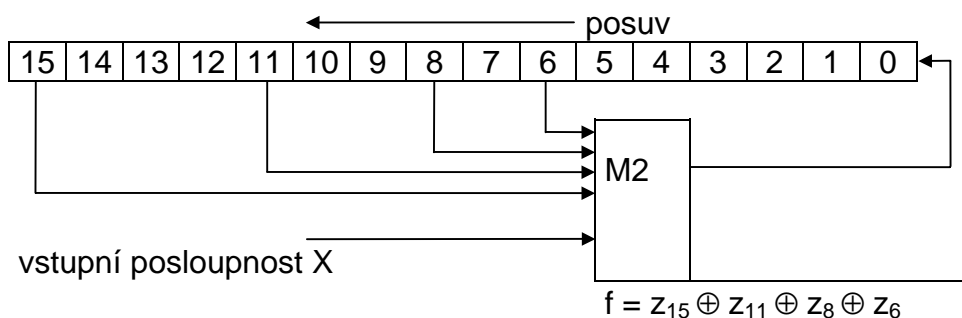
Tabulka *akce* má o položku více než *reference* a obsahuje čísla návěští akcí při rozpoznání profilů. Neznámému profilu odpovídá poslední akce. Po vykonání odpovídající akce jsou již nepotřebné hodnoty posloupnosti c, d vynulovány a připraveny pro nový profil. V případě potřeby může být místo akce při rozpoznání profilu nastavena odpovídající kombinace výstupů, případně obojí. Komparaci provádíme s negovanými hodnotami posloupností c, d. Důvod je čistě formální - abychom při zadávání *reference* mohli popisovat přímo profil (neprůsvitné části = log.1, průsvitné = log.0).

Uvedený program je složitostí srovnatelný s programem s instrukcí **STE**. Je však obecnější:

- vyhodnocuje skutečnost, že profil nebyl rozpoznán
- při požadavku na větší počet profilů narůstá pouze počet položek *reference* (2 byty / profil)
- je možné doplnit do *reference* položky, odpovídající profilům v různých polohách.

Naproti tomu program s instrukcí **STE** narůstá s každým novým profilem zhruba o třetinu původního objemu.

Pro posloupnosti c, d, delší než 8 bitů nebo pro větší počet sledovaných proměnných je však tento postup nevhodný. Z teorie příznakové analýzy a cyklických kódů plyne, že s pomocí zpětnovazebních registrů, např. dle struktury na obrázku, lze transformovat vstupní posloupnost (jedné proměnné nebo střídavě připojovaných proměnných) na výsledný obsah posuvného registru příznaků (počáteční stav by neměl být nulový).



Je velmi malá pravděpodobnost, že dvě různé posloupnosti se transformují na stejný příznak, záleží na délce posloupnosti, délce a struktuře zpětnovazebního registru. Je však jisté, že různým hodnotám příznaků odpovídají různé vstupní posloupnosti. Ty vzácné případy, kdy nejsou rozpoznány různé posloupnosti, je třeba řešit doplňkovým postupem.

Vlastní postup je obdobný, pouze v místech, kde střeďáme posloupnost hodnot c, d, zavoláme podprogram, který realizuje zpětnovazební registr a generuje příznak (viz příklady k instrukci **XOR**). Po rozpoznání je obsah registru nastaven na 65 535. Takto prakticky nejsme omezeni délkou testované posloupnosti, ani počtem vstupů, z nichž ji vytváříme.

Problematickým se může jevit pouze postup zadání tabulky referenčních příznaků. Ruční generování je prakticky neproveditelné. Pro předem známé profily můžeme přízna-



ky generovat na počítači. V praxi může být postup generování tabulky v režimu učení. V tomto režimu (realizovaném zvlášť vytvořeným uživatelským programem) postupně „exponujeme“ různé vzorky profilů a jejich příznaky ukládáme do tabulky.

Postup příznakové analýzy má výrazně širší použití než právě popsané rozpoznávání profilů. Lze jej považovat za jednu z metod zhuštění (komprese) dat. Je výhodný k rozpoznání libovolných posloupností bitů nebo bytů, k rozpoznání zvláštních stavů systému nebo programu, zejména k rozpoznání chybových stavů.

### Příklad 8.3.10

Předpokládejme funkci  $f$  proměnných  $a, b, c, d$  zadanou pravdivostní tabulkou:

| $d$ | $c$ | $b$ | $a$ | $f$ |
|-----|-----|-----|-----|-----|
| 0   | 0   | 0   | 0   | 1   |
| 0   | 0   | 0   | 1   | 1   |
| 0   | 0   | 1   | 0   | 0   |
| 0   | 0   | 1   | 1   | 0   |
| 0   | 1   | 0   | 0   | 1   |
| 0   | 1   | 0   | 1   | 1   |
| 0   | 1   | 1   | 0   | 1   |
| 0   | 1   | 1   | 1   | 1   |

| $d$ | $c$ | $b$ | $a$ | $f$ |
|-----|-----|-----|-----|-----|
| 1   | 0   | 0   | 0   | 0   |
| 1   | 0   | 0   | 1   | 1   |
| 1   | 0   | 1   | 0   | 0   |
| 1   | 0   | 1   | 1   | 0   |
| 1   | 1   | 0   | 0   | 0   |
| 1   | 1   | 0   | 1   | 1   |
| 1   | 1   | 1   | 0   | 0   |
| 1   | 1   | 1   | 1   | 0   |

Její realizace instrukcí **LTB** s bitovou tabulkou byla popsána v příkladech instrukce **LTB**. Uvedme si způsob realizace instrukcí **FTB**.

*B08310.mos*

#reg byte vstupy

#reg bit vf

#table byte maska = %0000,%0001,%0100,%0101,%0110,%0111,%1001,%1101

;

P 0

LD vstupy

AND %1111 ;oddělení proměnné  $a, b, c, d$

FTB maska ;porovnání s maskami

LD %S1 ;příznak nalezení

WR vf

E 0

Položky tabulky *maska* jsou rovné kombinacím vstupních proměnných, pro které je výstup jedničkový - opsané řádky z pravdivostní tabulky. Na pořadí položek nezáleží. Hodnota indexu nás nezajímá, pouze informace o nalezení položky obsažená v S1.0. Pokud se skutečná kombinace vstupů  $a, b, c, d$  shoduje s některou kombinací, pro které je předepsán jedničkový výstup, je jedničková i hodnota funkce.

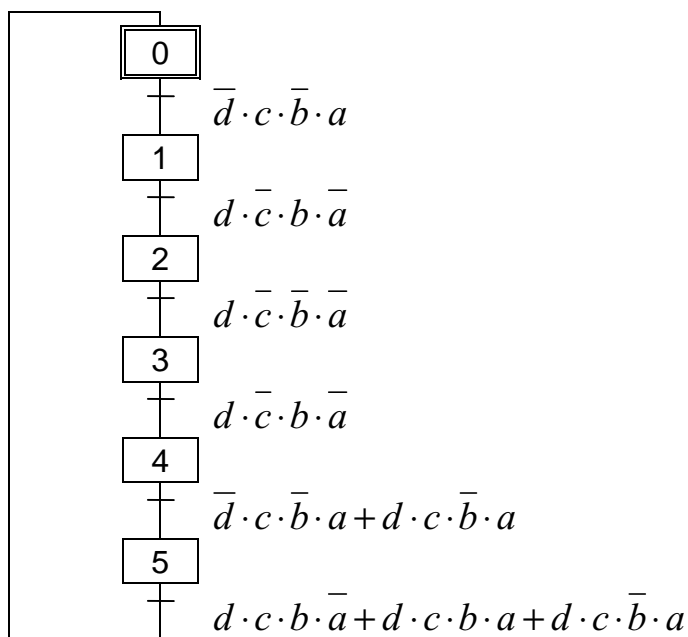
Postupem realizujeme tzv. úplnou disjunktí normální formu, tj. logický výraz ve tvaru součtu součinů, v němž každý součin (term) obsahuje všechny proměnné. Výskytu přímých proměnných odpovídají jedničky v masce, negovaným proměnným odpovídají nuly.

Necháváme stranou, zda pro tento případ je tato realizace optimální (není - optimální je zde postup s **LTB**). Je však mnoho případů, kdy tento postup může být výhodný - zejména v případech funkcí velkého počtu proměnných (8 až 16), jejichž pravdivostní tabulka má malý počet jedničkových řádků.

Viz též příklad 8.3.16.

**Příklad 8.3.11**

Uvažujme krokový řadič dle přechodového diagramu:



Ponecháme stranou možnost použití instrukce **STE**. Podmínky jsou zde důsledně definovány jako součiny čtyř proměnných  $a, b, c, d$  s různými pravdivostními hodnotami. V několika případech se vyskytují stejné podmínky k různým přechodům. Proto nevystačíme s tabulkou masek součinných členů, v nichž by se vyskytovaly pouze vstupní proměnné - do masek zahrneme i čísla stavů, z nichž se přechod uskutečňuje. Na pořadí položek pak nezáleží.

*B08311.mos*

#reg byte vstupy

#reg byte stav

```

#table byte maska = %00000101, ;stav 0, d̄, c̄, b̄, ā
 %00011010, ;stav 1, d̄, c̄, b̄, ā
 %00101000, ;stav 2, d̄, c̄, b̄, ā
 %00111010, ;stav 3, d̄, c̄, b̄, ā
 %01000101, ;stav 4, d̄, c̄, b̄, ā
 %01001101, ;stav 4, d̄, c̄, b̄, ā
 %01011110, ;stav 5, d̄, c̄, b̄, ā
 %01011111, ;stav 5, d̄, c̄, b̄, ā
 %01011101 ;stav 5, d̄, c̄, b̄, ā

```

;

P 0

```

LD vstupy
AND 15 ;oddělení vstupů a, b, c, d
LD stav ;stav
MUL 16 ;horní polovina bytu
OR ;složení stavu a vstupů
FTB maska
LD %S1 ;S1.0 - příznak nalezeno
ADX stav ;aktualizování stavu
CMP 6 ;mod 6
ANC %S0.0 ;příznak rovnosti (log.0 - stav 6)
 ;změna stavu 6 na stav 0
WR stav ;uložení

```

E 0

Od čísla stavu pak můžeme odvodit stavové masky, hodnoty výstupů, čísla akcí, hodnoty časů apod. Obdobný postup lze použít i pro složitější řadiče, kde počet vstupních proměnných a stavových proměnných (počet bitů, nutných k zobrazení čísla stavu) nepřesáhne 16.

Viz též příklad 8.3.16.

### Příklad 8.3.12

Předpokládejme logickou funkci, která je definovaná párovými dvojicemi hodnot vstupních a výstupních vektorů, například:

| <i>vstup.7</i> | <i>.6</i> | <i>.5</i> | <i>.4</i> | <i>.3</i> | <i>.2</i> | <i>.1</i> | <i>.0</i> |   | <i>vystup.7</i> | <i>.6</i> | <i>.5</i> | <i>.4</i> | <i>.3</i> | <i>.2</i> | <i>.1</i> | <i>.0</i> |
|----------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|---|-----------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 0              | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0 | 0               | 0         | 0         | 0         | 0         | 0         | 0         | 1         |
| 1              | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 1 | 0               | 0         | 0         | 0         | 0         | 0         | 1         | 0         |
| 1              | 0         | 0         | 0         | 0         | 0         | 0         | 1         | 0 | 0               | 0         | 0         | 0         | 0         | 0         | 1         | 0         |
| 1              | 1         | 0         | 0         | 0         | 0         | 0         | 1         | 1 | 1               | 0         | 0         | 0         | 0         | 1         | 0         | 0         |
| 0              | 1         | 1         | 1         | 0         | 1         | 1         | 0         |   | 1               | 1         | 0         | 1         | 0         | 1         | 1         | 0         |
| 1              | 0         | 1         | 1         | 0         | 0         | 0         | 1         | 1 | 0               | 1         | 1         | 0         | 0         | 1         | 0         | 1         |
| 1              | 1         | 1         | 0         | 0         | 0         | 0         | 1         | 0 | 1               | 0         | 0         | 0         | 0         | 1         | 1         | 0         |
| ostatní        |           |           |           |           |           |           |           |   | 1               | 0         | 1         | 1         | 1         | 0         | 0         | 0         |

Úlohu lze řešit dvěmi tabulkami. Prvá tabulka *vstup\_maska* obsahuje ty kombinace vstupních proměnných, pro které jsou definovány výstupní proměnné. Druhá tabulka *vystup\_maska* obsahuje ve stejném pořadí odpovídající kombinace výstupních proměnných. Jinak na pořadí položek v tabulkách nezáleží - pouze musí být zajištěno, aby odpovídající si položky byly zařazeny pod stejným indexem. Tabulka *vystup\_maska* má položku navíc, která ošetřuje všechny nedefinované vstupní proměnné.

*B08312.mos*

```
#reg byte vstupy, vystupy
```

```
#table byte vstup_maska = %00000000,%10000001,%10000010,%11000011,
 %01110110,%10110011,%111100010
```

```
#table byte vystup_maska = %00000001,%00000010,%00000010,%10000100,
 %11010110,%01100101,%10000110,%10111000
```

```
;
```

```
P 0
```

```
LD vstupy
```

```
FTB vstup_maska
```

```
LTB vystup_maska
```

```
WR vystupy
```

```
E 0
```

Viz též příklad 8.3.15.

### Příklad 8.3.13

Předpokládejme, že máme opět realizovat funkci časového procesoru (viz příklad 8.3.7) zadaného tabulkou časů a výstupů (údaje neodpovídají žádnému reálnému procesu - slouží pouze k ilustraci postupů):

## 8. Tabulkové instrukce

| index | čas (den, hod., min.) | vystup.7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 | akce  |
|-------|-----------------------|----------|----|----|----|----|----|----|----|-------|
| 0     | po, 5.15              | 1        | D  | D  | D  | D  | 1  | D  | D  | akce1 |
| 1     | so, 7.50              | 1        | D  | D  | D  | D  | 0  | D  | D  | akce1 |
| 2     | ne, 8.35              | 1        | D  | D  | D  | D  | 1  | D  | D  | akce1 |
| 3     | so,ne, 5.45           | D        | D  | D  | D  | D  | 1  | D  | D  | -     |
| 4     | X, 5.45               | 1        | D  | D  | D  | D  | 1  | D  | D  | akce1 |
| 5     | X, 17.50              | 0        | D  | D  | D  | D  | 0  | D  | D  | akce2 |
| 6     | X, X.00               | D        | 1  | C  | C  | C  | 0  | 0  | D  | akce3 |
| 7     | X, X.20               | D        | D  | D  | D  | C  | 0  | D  | D  | akce4 |
| 8     | X, X.30               | D        | 0  | C  | D  | D  | 0  | 0  | D  | -     |
| 9     | X, X.40               | D        | D  | D  | D  | C  | 0  | D  | 0  | akce5 |
| 10    | X, lichá.31           | D        | D  | D  | D  | D  | 1  | D  | 1  | -     |
| 11    | X, čtvrtá.šestnáctá   | D        | D  | D  | D  | D  | 0  | 1  | D  | akce6 |
| 12    | X, X.lichá            | D        | D  | D  | D  | D  | 1  | D  | D  | -     |
| 13    | X, X.sudá             | D        | D  | D  | D  | D  | 0  | D  | D  | -     |
| 14    | v mezích              | D        | D  | D  | D  | D  | D  | D  | D  | -     |

V časovém sloupci označují symboly X, že na jejich hodnotě nezáleží. Symboly D značí nezměněnou hodnotu, symboly C změnu hodnoty.

V některých případech může záležet na pořadí položek, při prohledávání se instrukce zastaví u první nalezené položky. Prvních šest položek slouží k ovládání *vystup.7*, který je zapnut v pondělí v 5.15, v sobotu v 7.50, v neděli v 8.35 a v ostatní dny v 5.45. Kdyby položka 4 byla zařazena jako první, pak by se *vystup.7* zapínal od úterý do neděle v 5.45 a položky pro sobotu a neděli by byly zbytečné (v pondělí se zapíná dříve, než v ostatní dny, takže položka 0 se uplatní). Aby se eliminovala položka 4 pro sobotu a neděli, je překryta položkou 3.

Podobné situace mohou nastat i u ostatních položek. Např. položky 12 a 13 se navzájem doplňují tak, že položka 14 je v podstatě zbytečná.

Obecně platí, že nejdříve by měly být uvedeny položky úzce specifikující časové okamžiky a postupně řazeny položky širší.

V tabulce jsou dále uvedeny některé zajímavé situace:

- *vystup.6* se zapíná v každou celou hodinu a po půlhodině se vypíná (položky 6 a 8)
- *vystup.5* také mění stav každou půlhodinu (položky 6 a 8), může však spínat současně s *vystup.6* nebo v protifázi k němu, záleží na počáteční hodnotě
- *vystup.4* mění stav každou celou hodinu (položka 6)
- *vystup.3* mění stav každých 20 minut (v 0, 20, 40 - položky 6, 7, 9)
- *vystup.2* se zapíná každou sudou minutu, každou lichou se vypíná (aby nedošlo k překrytí předchozími položkami, je třeba tento výstup ošetřit i v těchto předcházejících položkách)
- *vystup.1* se zapíná od minuty 16 do 30 nebo od 48 do 00, pokud jsou hodiny 4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23
- *vystup.0* se zapíná každou lichou hodinu od 31 do 40 minut

Postup je prakticky stejný jako v příkladu 8.3.7. Pouze místo instrukce **FTB** používá **FTM**.

*B08313.mos*

#reg byte vystup

#reg byte index, indexs

#label akce1,akce2,akce3,akce4,akce5,akce6,nic

```
#table byte cas = 15, 37,$FF,$FF, ; 0 - po, 5.15
 50,199,$FF,$FF, ; 1 - so, 7.50
 35,232,$FF,$FF, ; 2 - ne, 8.35
 45,197,$FF,$DF, ; 3 - so,ne, 5.45
 45, 5,$FF,$1F, ; 4 - X, 5.45
```

```

50, 17,$FF,$1F, ; 5 - X, 17.50
0, 0,$FF,$00, ; 6 - X, X.00
20, 0,$FF,$00, ; 7 - X, X.20
30, 0,$FF,$00, ; 8 - X, X.30
40, 0,$FF,$00, ; 9 - X, X.40
31, 1,$FF,$01, ;10 - X, lichá.31
16, 4,$1F,$04, ;11 - X, čtvrtá.šestnáctá
1, 0,$01,$00, ;12 - X, X.lichá
0, 0,$01,$00 ;13 - X, X.sudá

#table byte nastav = %10000100,%10000100,%10000100,%00000100,
%10000100,%10000100,%01000110,%00000100,
%01000110,%00000101,%00000101,%00000110,
%00000100,%00000100,%00000000

#table byte polarita = %10000100,%10000000,%10000100,%00000100,
%10000100,%00000000,%01111000,%00001000,
%00100000,%00001000,%00000101,%00000010,
%00000100,%00000000,%00000000

#table byte akce =
__indx (akcel1), __indx (akcel1), __indx (akcel1),
__indx (nic), __indx (akcel1), __indx (akce2),
__indx (akce3), __indx (akce4), __indx (nic),
__indx (akce5), __indx (nic), __indx (akce6),
__indx (nic), __indx (nic), __indx (nic)

;
P 0
LD index
WR indexes ;minulý index
LD %SW7 ;A0L = min, A0H = hod
LD %S9 ;den v týdnu
MUL 32 ;posun o 5 pozic vpravo
SWP
OR ;A0L = min, A0H = den x 32 + hod.
FTM word cas
WR index
XOR indexes
JMC konec ;pokud se nemění, skončit
LD index ;index
LTB nastav ;tabulka nastavených výstupů:
RES vystup ;log.1 = nastavené výstupy,
;log.0 = výstupy s pamětovou funkcí

POP 1
LTB polarita ;tabulka polarity
XOR vystup ;log.1 = nastav log.1 nebo C,
;log.0 = nastav log.0 nebo D

WR vystup
POP 1
LTB akce ;tabulka návěští
CAI ;provedení akce

konec: ;společné pokračování
E 0
;
P 60
akcel1:
RET
;
:
akce6:
RET
;

```

nic:

RET

E 60

Tabulka není jednoduchá, ale při použití **FTB** pro stejnou úlohu by byla výrazně objemnější. Na obsahu hodnotové masky v pozicích vyloučených proměnných nezáleží. Obsah může být libovolný - zde jsme zvolili nulový.

### Příklad 8.3.14

Vraťme se k příkladu 8.3.10. Zadanou funkci  $f$  lze minimalizovat na výraz  $f = a \cdot \bar{b} + c \cdot \bar{d} + a \cdot b \cdot c \cdot \bar{d}$ , který lze přímo realizovat prohledáváním s výběrem bitů (**FTM**).

B08314.mos

#reg byte vstupy

#reg bit vf

```
#table byte maska = %0001, ;hodnotová maska a.b
 %0011, ;výběrová maska
 %0100, ;hodnotová maska c.d̄
 %1100, ;výběrová maska
 %0000, ;hodnotová maska a.b.c.d̄
 %1111 ;výběrová maska
```

;

P 0

LD vstupy

FTM maska ;porovnání s maskami

LD %S1 ;příznak nalezení

WR vf

E 0

Postup se od příkladu 8.3.10 liší pouze zrušením instrukce **AND**, v instrukci **FTM** a v tabulce *maska*.

Každému součinnému členu odpovídá jedna položka. Její hodnotovou (pravdivostní) masku (prvou část položky) vytvoříme tak, že na pozicích proměnných v přímé hodnotě (bez negace) píšeme log.1, na pozicích negovaných proměnných píšeme log.0. Na pozicích proměnných, které se v součinu nevyskytují, může být libovolný obsah - např. log.0.

Výběrovou masku (druhou část položky) tvoříme tak, že na pozicích proměnných, které jsou zastoupeny v součinném členu, píšeme log.1, na pozicích vyloučených proměnných log.0. V našem případě je postup s **FTM** úspornější, rozdíl je však nepřesvědčivý - příklad je uveden pro ilustraci.

### Příklad 8.3.15

Uvažujme zadání kombinační funkce, definované párovými dvojicemi vstup - výstup:

| vstup.7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 | vystup.7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---------|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|
| 1       | 1  | X  | X  | X  | X  | X  | X  | 1        | 0  | 1  | 1  | 1  | 0  | 1  | 1  |
| X       | X  | X  | X  | X  | X  | 1  | 1  | 0        | 0  | 0  | 1  | 1  | 1  | 0  | 0  |
| 0       | 1  | 1  | X  | 1  | 0  | X  | 1  | 1        | 0  | 1  | 0  | 0  | 1  | 1  | 0  |
| 1       | 0  | 1  | 1  | 0  | 1  | 0  | X  | 0        | 0  | 0  | 0  | 1  | 1  | 0  | 0  |
| ostatní |    |    |    |    |    |    |    | 0        | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Hodnoty tabulky byly zvoleny zcela náhodně, pouze pro ilustraci postupu. Symbolu X ve sloupci vstupu odpovídá vyloučená proměnná - její hodnota nerozhoduje o výstupech. Postup je obdobný řešení příkladu 8.3.12.

```

B08315.mos
#reg byte vstup, vystup
#table byte vstup_maska = %11000000,%11000000,%00000011,%00000011,
 %01101001,%11101101,%10110100,%11111110
#table byte vystup_maska = %10111011,%00011100,%10100110,%00001100,
 %00000000

;
P 0
 LD vstup
 FTM vstup_maska
 LTB vystup_maska
 WR vystup
E 0

```

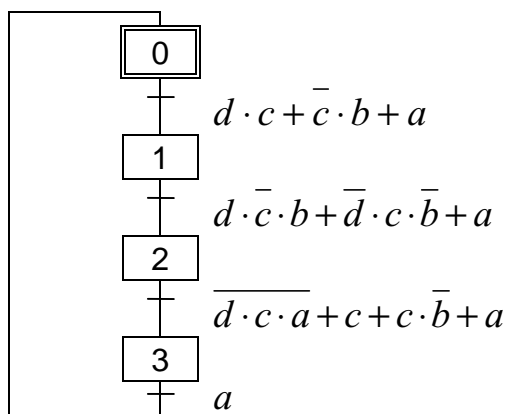
Tabulka *vstup\_maska* má opět dvojité položky. Hodnotové (pravdivostní) masky tvoříme tak, že symboly 0, 1 opisujeme ze sloupce vstupů pravdivostní tabulky, místo X můžeme psát libovolnou hodnotu, např. log.0. Do výběrové masky píšeme log.1 tam, kde vstupy měly hodnotu 0 nebo 1, na pozicích s X píšeme log.0.

Tabulka *vystup\_maska* je o položku delší. Hodnoty položek přímo opisujeme ze sloupce výstupů.

Výhodnost instrukce **FTM** se projeví zejména při zpracování širokého souboru podmínek (8 až 16 bitů), kdy tradiční postupy jsou krajně neekonomické.

### Příklad 8.3.16

Uvažujme sekvenční řadič dle diagramu:



Proměnná *a* se přičítá k podmínkám všech přechodů - její jedničková hodnota tedy působí vždy, nezávisle na stavu.

Postup se od řešení příkladu 8.3.11 liší pouze použitím instrukce **FTM** místo **FTB** a jiným obsahem tabulky.

```

B08316.mos
#reg byte vstupy
#reg byte stav
#table byte maska = %00001100,%11111100, ;stav 0, d.c
 %00000100,%11110110, ;stav 0, c.b
 %00000001,%00000001, ;všechny stavy, a
 %00011010,%11111110, ;stav 1, d.c.b
 %00010100,%11111110, ;stav 1, d.c.b̄
 %00100000,%11111101, ;stav 2, d.c.a
 %00100100,%11110100, ;stav 2, c
 %00100100,%11110110, ;stav 2, c.b
;

```

```

P 0
 LD vstupy
 AND 15 ;oddělení vstupů a, b, c, d
 LD stav ;stav
 MUL 16 ;horní polovina bytu
 OR ;složení stavu a vstupů
 FTM maska
 LD %S1 ;S1.0 - příznak nalezeno
 ADX stav ;aktualizování stavu
 CMP 6 ;modulo 6
 ANC %S0.0 ;příznak rovnosti (log.0 - stav 6)
 ;změna stavu 6 na stav 0
 WR stav ;uložení
E 0

```

**Příklad 8.3.17**

Tabulka *maska* obsahuje masky součinných členů pro realizaci skalární funkce proměnných *vstupy* instrukcí **FTB**. Po dobu uvádění technologie do chodu nejsou snímače na vstupech 4 až 7 funkční. Vstupy 4 a 5 mají trvale jedničkovou hodnotu, vstupy 6 a 7 jsou trvale nulové. Máme po přechodnou dobu realizovat tuto omezenou funkční závislost, aniž by se měnil obsah tabulky *maska*.

```

B08317.mos
#reg byte vstupy
#reg bit vystup
#reg byte funkce[8]
#table byte maska = %00010101,%11010001,%01000101,%11100010,
 %11101100,%11010010,%10100100,%10001010
;
P 0
 LD 0 ;od začátku tabulky
cyklus:
 LTB maska ;původní položka
 JNS konec ;test konce tabulky
 OR %00110000 ;jedničkové vstupy
 AND %00111111 ;nulové vstupy
 WTB funkce ;modifikovaná položka
 POP 1
 INR ;zvýšení indexu tabulky
 JMP cyklus ;další položka
;
konec:
 ;využití modifikované tabulky
 POP 1
 DCR ;srovnání indexu tabulky
 LD vstupy ;vstupní vektor
 FTB funkce ;výstupní funkce
 LD %S1
 WR vystup
E 0

```

**Příklad 8.3.18**

Pro stejný případ upravte obsah bytové tabulky *maska*, která je určena pro instrukci **FTM**. V tomto případě stačí upravit výběrové masky v položkách.



```

B08318.mos
#reg byte vstupý
#reg bit vystup
#reg byte funkce[16]
#table byte maska = %00010101,%11111111,%11010001,%11111100,
 %01000101,%10111111,%11100010,%11110111,
 %11101100,%00001111,%11010010,%11111111,
 %10100100,%11111110,%10001010,%11111111
;
P 0
 LD 0
cyklus:
 LTB word maska ;A0L = hodnota, A0H = výběr
 JNS konec
 AND %0000111111111111
 WTB word funkce ;modifikace výběrové masky
 POP 1
 INR ;zvýšení indexu tabulky
 JMP cyklus ;znovu
konec:
 ;použití modifikované tabulky
 POP 1
 DCR ;srovnání indexu tabulky
 LD vstupý
 FTM funkce
 LD %S1
 WR vystup
E 0

```

### Příklad 8.3.19

Na *vstupy* je připojena klávesnice, kde jeden vstup odpovídá jedné klávese. Požadujeme zjištění ASCII kódu stisknuté klávesy (příslušný vstup je log.1).

Předpokládáme, že v jednom okamžiku bude stisknuta jen jedna klávesa. V případě stisknutí více kláves současně bude detekována jen ta, která je připojena na vstup s nejnižší vahou.

```

B08319a.mos
#reg long vstupý ;32 kláves
#reg byte kod
#table byte prevod = '01234567', ;vstupý
 '89-+. */', ;vstupý+1
 '(', $11, $12, $13, $14, $1B, $0D, ;vstupý+2
 $F1, $F2, $F3, $F4, $F5, $F6, $F7, $F8, ;vstupý+3
 0, ;žádná klávesa
;
P 0
 LD 31 ;bitová mez
 LD 1 ;hledání log.1
 FTB vstupý.0
 LTB prevod
 WR kod ;ASCII kód stisknuté klávesy
E 0

```

V tomto příkladu zpracováváme přímo vstupý a předáváme ASCII kód trvale. Požadavek jednorázového předání kódu snadno realizujeme testem náběžné nebo sestupné hrany.

Požadujeme-li sejmutí více kláves současně, upravíme algoritmus tak, že současně stisknuté klávesy zapíšeme do pole *kod* a zakončíme kódem \$00. Příznaky současně

## 8. Tabulkové instrukce

stisknutých kláves střádáme do proměnné *klavesnice* a zpracujeme je až v okamžiku, kdy je uvolněna poslední klávesa.

```
B08319b.mos
#reg long vstupy ;32 kláves
#reg byte kod[8], index, poradi, pomoc
#reg long klavesnice
#table byte prevod = '01234567', ;vstupy
 '89-+. */', ;vstupy+1
 '()', $11,$12,$13,$14,$1B,$0D, ;vstupy+2
 $F1,$F2,$F3,$F4,$F5,$F6,$F7,$F8, ;vstupy+3
 0 ;žádná klávesa

;
P 0
 LD vstupy
 SET word klavesnice ;střádání současně stisknutých kláves
 POP 1
 SET word klavesnice+2
 OR word vstupy ;dokud je nějaká klávesa stisknutá,
 JMD konec ;nedělat nic dalšího
 LD klavesnice
 OR
 JMC konec ;byla stisknuta klávesa?
 LD 0
 WR poradi
hledani:
 LD 31 ;bitová mez
 LD 1 ;hledání log.1
 FTB klavesnice.0
 WR index
 LTB prevod
 WR pomoc ;ASCII kód stisknuté klávesy
 LD 8 ;mez
 LD poradi
 LD pomoc
 WTB kod ;zápis do pole kódů
 JMC hotovo ;$00 ukončuje hledání
 INR poradi
 LD 31 ;bitová mez
 LD index ;index nalezené klávesy
 LD 0 ;smazání příznaku klávesy
 WTB klavesnice.0
 JMP hledani ;hledání další klávesy
;
hotovo:
 LD 0
 WR klavesnice ;smazání příznaků
konec:
E 0
```

### 8.4. Třídění hodnot podle tabulek (FTS)

Tabulkové instrukce **FTS** umožňují třídění hodnot podle obsahu tabulek T a tabulek v zápisníku ve formátech byte a word.

### Příklad 8.4.1

Sledovaný proces začíná s příchodem náběžné hrany signálu *vstup*. Je třeba realizovat časovou posloupnost, jejíž průběh je nezávislý na dalším průběhu signálu *vstup*.

| čas                      | akce          |
|--------------------------|---------------|
| 0 - 0,25 s               | <i>akce0</i>  |
| do 0,75 s                | <i>akce1</i>  |
| do 1,35 s                | <i>akce2</i>  |
| do 3,45 s                | <i>akce3</i>  |
| do 7,75 s                | <i>akce4</i>  |
| do 11,15 s               | <i>akce5</i>  |
| :                        | :             |
| do 154,50 s              | <i>akce15</i> |
| 154,50 - 2305 s          | <i>akce16</i> |
| pak čekání na nový start | <i>akce0</i>  |

Bylo by možné postupovat stejně jako u časových procesorů v příkladech instrukce **FTB**. Zde však jsou mezní okamžiky zadány v desítkách ms a při delším cyklu uživatelského programu je riziko, že budou vynechány. Spolehlivější realizaci poskytuje tedy třídění.

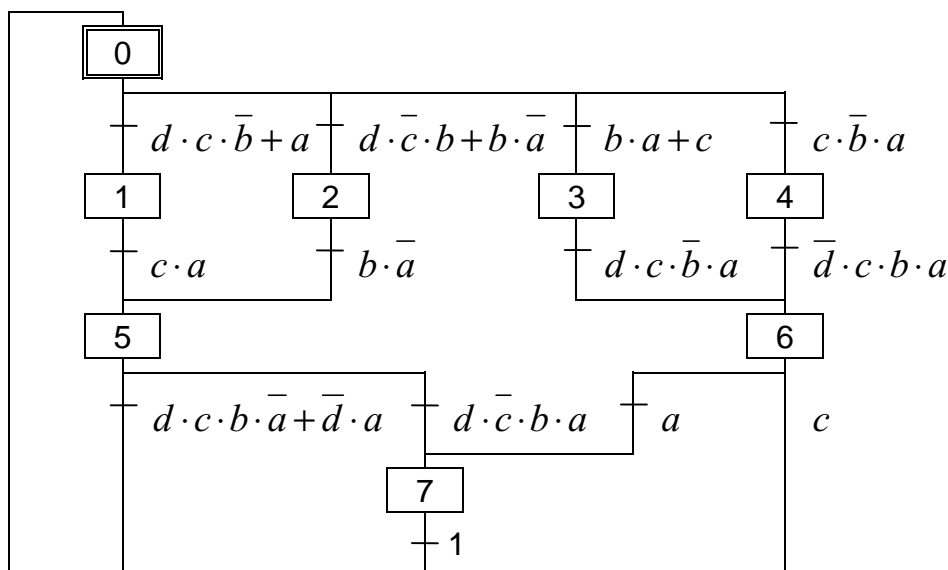
Tabulka *interval* obsahuje mezní hodnoty časových intervalů v jednotkách 10 ms. Akce jsou číslovány pořadově, není tedy nutné je transformovat tabulkou.

*B08401.mos*

```
#reg bit vstup
#reg word casovac
#label 0,akce0,akce1,akce2,akce3,akce4,akce5,...,akce15,akce16
#table word interval = 25,75,135,345,775,1115,...,15450
;
P 0
 LD vstup ;řídící vstup
 LD 23050 ;konec procesu = předvolba
 IMP casovac ;časovač
 LD casovac
 FTS interval ;index intervalu
 CAI ;provedení akce
E 0
;
P 60
akce0:
 RET
;
:
;
akce16:
 RET
E 60
```

**Příklad 8.4.2**

Realizujte sekvenční řadič dle přechodového diagramu:



Nejedná se již o krokový řadič, ale o sekvenční automat bez výstupů (ty jsme pro jednoduchost výkladu neuvedli). Struktura je smyšlená pouze pro potřebu ilustrace postupů. Na rozdíl od krokového řadiče zde již nestačí pouhé stanovení podmínky přechodu, je nutné určit i cíl přechodu. Nabízí se postup:

*B08402a.mos*

#reg byte vstupy

#reg byte stav

```

#table byte prechod = %00001100,%11111110, ;stav 0, dcb
 %00000001,%11110001, ;stav 0, a_
 %00001010,%11111110, ;stav 0, dcb
 %00000010,%11110011, ;stav 0, ba
 %00000011,%11110011, ;stav 0, ba
 %00000100,%11110100, ;stav 0, c_
 %00000101,%11110111, ;stav 0, cba
 %00010101,%11110101, ;stav 1, ca_
 %00100010,%11110011, ;stav 2, ba_
 %00111101,%11111111, ;stav 3, dcba
 %01000111,%11111111, ;stav 4, dcba_
 %01011110,%11111111, ;stav 5, dcba
 %01010001,%11111001, ;stav 5, da_
 %01011011,%11111111, ;stav 5, dcba
 %01100001,%11110001, ;stav 6, a
 %01100100,%11110100, ;stav 6, c
 %01110000,%11110000 ;stav 7, 1

```

```

#table byte cil = 1,1,2,2,3,3,4,5,5,6,6,0,0,7,7,0,0

```

;

P 0

```

LD stav
MUL 16 ;posunutí stavu o 4 vlevo
LD vstupy
AND 15 ;oddělení d, c, b, a
OR ;složení
FTM prechod ;podmínky přechodu
JNS konec ;nesplněná podmínka - nic nedělat
LTB cil ;nový stav
WR stav

```

konec:

E 0

Tabulka *prechod* definuje podmínky přechodu - stejný formát a způsob zobrazení jako v příkladu krokového řadiče v předchozí kapitole. Jedničkovou hodnotu podmínky (podmínka permanentně splněná) realizujeme nulovými hodnotami výběrové masky - „když se na nic neptám, nemohu dostat nepříznivou odpověď“.

Tabulka *cil* obsahuje čísla stavů, do kterých přechod směřuje.

Mohli bychom si dále vymýšlet rozšířená zadání, např. při splnění podmínky můžeme další tabulkou definovat:

- číslo akce, ošetřující přechod (Mealyho automat)
- hodnoty výstupů spojených s přechodem (Mealy) - např. s využitím symbolů 0, 1, D, C, případně I
- hodnoty časů maximálního, minimálního setrvání ve stavu nebo času zpoždění přechodu
- vynulovat staré a nastavit nové hodnoty stavových masek

Nezávisle na splnění podmínky můžeme v závěrečné fázi (po návěští *konec*) definovat:

- číslo klidové akce odpovídající stavu (Moore)
- hodnoty výstupů (Moore)

Rozsah druhé tabulky *cil* můžeme redukovat (někdy výrazně). Stačí, pokud podmínky přechodu v *prechod* uspořádáme tak, aby přechody směřující do stejného stavu byly vždy spolu a aby po třídění instrukcí **FTM** bylo číslo třídy shodné s číslem cílového stavu.

Program se změní pouze v tom, že instrukci **LTB cil** nahradíme instrukcí **FTS meze**, kde tabulka *meze* obsahuje hranice stejného cíle v tabulce *prechod*.

*B08402b.mos*

#reg byte vstupy

#reg byte stav

```

;
; přechod do 0
#table byte prechod = %01011110,%11111111, ;stav 5, d_cba
 %01010001,%11111001, ;stav 5, d_a
 %01100100,%11110100, ;stav 6, c
 %01110000,%11110000, ;stav 7, 1
; přechod do 1
 %00001100,%11111110, ;stav 0, dcb
 %00000001,%11110001, ;stav 0, a
; přechod do 2
 %00001010,%11111110, ;stav 0, dcb
 %00000010,%11110011, ;stav 0, ba
; přechod do 3
 %00000011,%11110011, ;stav 0, ba
 %00000100,%11110100, ;stav 0, c
; přechod do 4
 %00000101,%11110111, ;stav 0, cba
; přechod do 5
 %00010101,%11110101, ;stav 1, ca
 %00100010,%11110011, ;stav 2, ba
; přechod do 6
 %00111101,%11111111, ;stav 3, d_cba
 %01000111,%11111111, ;stav 4, d_cba
; přechod do 7
 %01011011,%11111111, ;stav 5, d_cba
 %01100001,%11110001 ;stav 6, a
#table byte meze = 3,5,7,9,10,12,14,16

```

## 8. Tabulkové instrukce

```

;
P 0
 LD stav
 MUL 16 ;posunutí stavu o 4 vlevo
 LD vstupy
 AND 15 ;oddělení d, c, b, a
 OR ;složení
 FTM prechod ;podmínky přechodu
 JNS konec ;nesplněná podmínka - nic nedělat
 FTS meze ;nový stav
 WR stav
konec:
E 0

```

Zdánlivě je rozdíl pouze formální. Významně odlišný však může být rozsah tabulek - u složitějších automatů může počet přechodů významně převyšovat počet stavů.

### Příklad 8.4.3

Máme realizovat kombinační logickou funkci zadanou párovými relacemi vstup - výstup. Stejná kombinace výstupů však může být přiřazena většímu počtu vstupních kombinací:

|    | vstup.7 | .6 | .5 | .4      | .3 | .2 | .1 | .0 |  | vystup.7 | .6 | .5 | .4 | .3 | .2 | .1 | .0              |
|----|---------|----|----|---------|----|----|----|----|--|----------|----|----|----|----|----|----|-----------------|
| 0  | 1       | 1  | 0  | 0       | 1  | 1  | 1  | 0  |  | 1        | 0  | 0  | 0  | 0  | 1  | 1  | 0               |
| 1  | 0       | 1  | 1  | 0       | 0  | 0  | 1  | 1  |  |          |    |    |    |    |    |    |                 |
| 2  | 0       | 0  | X  | X       | 1  | 1  | X  | X  |  |          |    |    |    |    |    |    |                 |
| 3  | X       | X  | X  | 1       | 1  | X  | X  | 1  |  |          |    |    |    |    |    |    |                 |
| 4  | 0       | 0  | 0  | X       | 1  | 0  | 0  | X  |  | 0        | 1  | 1  | 1  | 0  | 0  | 0  | 1               |
| 5  | 1       | 0  | 0  | 1       | 0  | X  | X  | 0  |  |          |    |    |    |    |    |    |                 |
| 6  | 1       | 1  | 1  | 1       | 0  | 1  | X  | X  |  | 1        | 1  | 0  | 0  | 0  | 1  | 0  | 0               |
| :  |         |    |    |         |    |    |    |    |  |          |    |    |    |    |    |    |                 |
| 23 | 0       | 0  | 0  | 0       | 1  | 0  | 0  | 0  |  | 1        | 1  | 0  | 0  | 0  | 1  | 0  | 0               |
| 24 |         |    |    | ostatní |    |    |    |    |  |          |    |    |    |    |    |    | neměnit výstupy |

Jiná úloha, ale postup je téměř stejný, jako v předchozím příkladu.

```

B08403a.mos
#reg byte vstupy, vystupy
#table byte prechod = %11001110,%11111111,%01100011,%11111111,
 %00001100,%11001100,%00011001,%00011001,
 %00001000,%11101110,%10010000,%11111001,
 %11110100,%11111100,..
 %00001000,%11111111
#table byte hodnota = %10000110,%10000110,%10000110,%10000110,
 %01110001,%01110001,%11000100,..
 %11000100

;
P 0
 LD vstupy
 FTM prechod ;podmínky změny výstupů
 JNS konec ;nesplněná podmínka - nic nedělat
 LTB hodnota ;nový stav
 WR vystupy
konec:
E 0

```

Pokud se rozhodneme pro postup s tříděním (s instrukcí **FTS**), pak bude program vypadat takto:

*B08403b.mos*

```
#reg byte vstupy, vystupy
;
;kombinace výstupů 0
#table byte prechod = %11001110,%11111111,%01100011,%11111111,
 %00001100,%11001100,%00011001,%00011001,
 ;kombinace výstupů 1
 %00001000,%11101110,%10010000,%11111001,
 ;kombinace výstupů 2
 %11110100,%11111100,...
 %00001000,%11111111
#table byte meze = 3,5,23
#table byte hodnota = %10000110,%01110001,%11000100
;
P 0
 LD vstupy
 FTM prechod ;podmínky změny výstupů
 JNS konec ;nesplněná podmínka - nic nedělat
 FTS meze
 LTB hodnota ;nový stav
 WR vystupy
konec:
E 0
```

Při jiném zadání opět můžeme pracovat s několika tabulkami, např. doprovodnými akcemi, ošetření výstupních symbolů 1, 0, D, L, případně I, atd.

Při zachování zadání a postupu z předchozího příkladu není třeba zvlášť ošetřovat vstupní položku č. 24, protože v situaci „ostatní“ není třídění aktivováno, tedy ani překlad tabulkami. Pokud by situaci „ostatní“ odpovídal požadavek na konkrétní nastavení výstupů nebo na jinou akci, stačí doplnit o položku navíc do tabulky *hodnota* a vynechat instrukci **JNS konec**.

## 9. BLOKOVÉ OPERACE

### 9.1. Přesuny bloků dat (SRC, MOV)

Přesuny bloků dat v zápisníku, v tabulkách T i navzájem umožňují instrukce **SRC** a **MOV**.

Pozor! Zápis do tabulek T při zapnuté uživatelské paměti EEPROM je možný, pokud při překladu uživatelského programu překladačem xPRO označíme volbu *Chráněné tabulky T*, jinak budou po vypnutí a opětovném zapnutí PLC všechny změny ztraceny (z EEPROM se zkopírují hodnoty deklarované ve zdrojovém textu uživatelského programu).

#### Příklad 9.1.1

Požadujeme archivování událostí do zásobníku v zápisníkové paměti. Záznam o události má velikost 8 bytů.

```
B09101.mos
#def delka 8
#reg byte udalost[delka]
#reg byte zasobnik[delka*128]
#reg byte index
;
P 0
 LD 0 ;INDEX_SRC
 SRC udalost ;zdroj
 LD index ;číslo události
 MUL delka ;ukazovátka do zásobníku
 LD delka ;délka události
 MOV zasobnik ;cíl
 LD index
 INR ;zvýšení indexu
 AND 127 ;zacyklení zásobníku
 WR index
E 0
```

#### Příklad 9.1.2

Vyjděme z předchozího příkladu a doplňme jej o postupné vyčítání událostí do zóny pro výstupní zařízení.

```
B09102.mos
#def delka 8
#reg byte udalost[delka], vystup[delka]
#reg byte zasobnik[delka*128]
#reg byte indexz, indexc
#reg bit zapis, cteni, plno, prazdno
;
P 0
 LD zapis
 JMC skok1
 RES zapis ;nová událost
 RES prazdno
 LD 0 ;INDEX_SRC
 SRC udalost ;zdroj
```



```

LD indexz ;číslo události
MUL delka ;ukazovátka do zásobníku
LD delka ;délka události
MOV zasobnik ;cíl
LD indexz
INR ;zvýšení indexu
AND 127 ;zacyklení zásobníku
WR indexz
LD indexc
EQ ;test zaplnění
SET plno ;indexz = indexc -> plno = log.1
skok1:
LD prazdno
JMD skok2 ;je co číst?
LD cteni
JMC skok2
RES cteni ;přečtení nejstarší události
LD indexc ;číslo události
MUL delka ;ukazovátka do zásobníku
SRC zasobnik ;zdroj
LD 0 ;INDEX_MOV
LD delka ;délka události
MOV vystup ;cíl
LD indexc
INR ;zvýšení indexu
AND 127 ;zacyklení zásobníku
WR indexc
LD indexz
EQ ;test vyprázdnění
SET prazdno ;indexz = indexc -> prazdno = log.1
skok2:
E 0
;
P 63
LD 1
WR prazdno ;v zásobníku nic není
E 63

```

Na zásobník událostí ukazují dvě ukazovátka, *indexz* pro zápis a *indexc* pro čtení. *Indexz* ukazuje na pozici, na kterou se bude zapisovat, *indexc* ukazuje na pozici, ze které se bude číst. Zápis události do zásobníku je vázán na příznak *zapis*, čtení nejstarší události ze zásobníku do pole *vystup* je vázán na příznak *cteni*. Zásobník je uzavřen do kruhu a má kapacitu 128 událostí.

Pokud po zápisu události a zvýšení ukazovátka *indexz* jsou obě ukazovátka shodná, znamená to zaplnění zásobníku (další nová událost by přepsala nejstarší ještě nevyčtenou událost) - nastaven příznak *plno*. Pokud po přečtení události a zvýšení ukazovátka *indexc* jsou obě ukazovátka shodná, znamená to vyprázdnění zásobníku - nastaven příznak *prazdno*. Tento stav je nastaven jako výchozí po startu PLC.

Viz též příklady 10.1.1, 10.1.2.

## 9.2. Kopírování tabulky do zápisníku a naopak (MTN, MNT)

Instrukce **MTN** kopíruje obsah tabulky do zápisníku, instrukce **MNT** naopak naplní celou tabulku obsahem zápisníku.

Pozor! Zápis do tabulek T při zapnuté uživatelské paměti EEPROM je možný, pokud při překladu uživatelského programu překladačem xPRO označíme volbu *Chráněné tabulky T*, jinak budou po vypnutí a opětovném zapnutí PLC všechny změny ztraceny (z EEPROM se zkopírují hodnoty deklarované ve zdrojovém textu uživatelského programu).

### Příklad 9.2.1

Chceme zobrazovat na displeji různé texty v závislosti na hodnotě registru *cislo\_textu*.

Jednotlivé texty nadeklarujeme pomocí tabulek a jejich zobrazení budeme realizovat překopírováním textu z příslušné tabulky do zóny displeje pomocí instrukce **MTN**. Čísla textu sesouhlasíme s čísly tabulek.

```
B09201.mos
#reg byte displej[32]
#reg byte cislo_textu
;
#table byte 0,text0 = ' : první řádek :: druhý řádek :
#table byte 1,text1 = ' toto je text číslo 0 '
#table byte 2,text2 = ' toto je text číslo 1 '
#table byte 3,text3 = ' toto je text číslo 2 '
#table byte 4,text4 = ' toto je text číslo 3 '
#table byte 4,text4 = ' toto je text číslo 4 '
;
P 0
 LD cislo_textu ;zdroj
 LD __indx (displej) ;cíl
 MTN
E 0
```

### Příklad 9.2.2

Potřebujeme řešit složitý algoritmus pro čtyři shodné objekty.

Klasické řešení tohoto případu vede na čtyřnásobné řešení stejného algoritmu pro čtveřici různých objektů.

```
B09202a.mos
#reg byte data00, data01, data02, ...
#reg byte data10, data11, data12, ...
#reg byte data20, data21, data22, ...
#reg byte data30, data31, data32, ...
;
P 0
 LD data00 ;objekt 0
 :
 WR data00
;
 LD data10 ;objekt 1
 :
 WR data10
;
 LD data20 ;objekt 2
 :
 WR data20
;
 LD data30 ;objekt 3
 :
 WR data30
E 0
```

Problém stejného algoritmu nad různými objekty můžeme řešit pomocí tabulek. Data musíme přeskupit tak, že čtveřice odpovídajících proměnných budou vždy pohromadě.

*B09202b.mos*

```
#reg byte data00, data10, data20, data30
#reg byte data01, data11, data21, data31
#reg byte data02, data12, data22, data32
...
#reg byte objekt, hodnota
;
P 0
 LD 0
 WR objekt
algoritmus:
 LD 3 ;LIMIT
 LD objekt ;INDEX
 LTB data00
 :
 LD 3 ;LIMIT
 LD objekt ;INDEX
 LD hodnota ;zapisovaná hodnota
 WTB data00
 INR objekt ;další objekt
 LD objekt
 CMP 4
 JC algoritmus
E 0 ;hotovo
```

U složitějších algoritmů je však jejich realizace tabulkovými instrukcemi poněkud těžkopádná, ne-li nemožná. Problém lze řešit poměrně elegantně pomocí pracovní datové zóny, do které vždy data před vlastním řešením překopírujeme a pak je zase uložíme zpět. Pokud máme nouzi o paměť v zápisníku, lze s výhodou použít tabulky.

*B09202c.mos*

```
#reg byte data0, data1, data2, ... ;pracovní zóna
;
#table byte zona0 = 0,1,25,... ;objekt 0 s počátečními hodnotami
#table byte zona1 = 0,1,25,... ;objekt 1 s počátečními hodnotami
#table byte zona2 = 0,1,25,... ;objekt 2 s počátečními hodnotami
#table byte zona3 = 0,1,25,... ;objekt 3 s počátečními hodnotami
;
P 0
 LD __indx (zona0) ;objekt 0
 LD __indx (data0)
 MTN
 CAL algoritmus
 LD __indx (zona0)
 LD __indx (data0)
 MNT
;
 LD __indx (zona1) ;objekt 1
 LD __indx (data0)
 MTN
 CAL algoritmus
 LD __indx (zona1)
 LD __indx (data0)
 MNT
;
 LD __indx (zona2) ;objekt 2
```

```

LD __indx (data0)
MTN
CAL algoritmus
LD __indx (zona2)
LD __indx (data0)
MNT
;
LD __indx (zona3) ;objekt 3
LD __indx (data0)
MTN
CAL algoritmus
LD __indx (zona3)
LD __indx (data0)
MNT
E 0
;
P 60
algoritmus:
LD data0 ;objekt n
:
WR data0
RET
E 60

```

Z časových důvodů může být výhodnější řešit v jednom cyklu jen jeden objekt a zrychlit tak odezvu PLC na ostatní signály.

*B09202d.mos*

```

#reg byte data0,data1,data2,... ;pracovní zóna
#reg byte objekt
;
#table byte 0,zona0 = 0,1,25,... ;objekt 0 s počátečními hodnotami
#table byte 1,zona1 = 0,1,25,... ;objekt 1 s počátečními hodnotami
#table byte 2,zona2 = 0,1,25,... ;objekt 2 s počátečními hodnotami
#table byte 3,zona3 = 0,1,25,... ;objekt 3 s počátečními hodnotami
;
P 0
LD objekt ;číslo objektu
LD __indx (data0)
MTN
LD data0 ;algoritmus
:
WR data0
LD objekt
LD __indx (data0)
MNT
LD objekt
INR ;další objekt v příštím cyklu
CMP 4
AND %S0.1 ;objekt < 4 -> S0.1 = log.1
WR objekt ;S0.1 = log.0 -> objekt = 0
E 0

```

Je zřejmé, že každá z uvedených variant má své výhody a nevýhody. Je proto na uživateli, aby se pro některou rozhodl na základě svých požadavků na rychlost programu, zaplnění paměti dat a programu, složitost uživatelského programu, apod.

### 9.3. Plnění zápisníku konstantou (FIL)

Plnění části zápisníku konstantou umožňuje instrukce **FIL**.

#### **Příklad 9.3.1**

Na náběžnou hranu signálu *reset* reagujeme vynulováním pole *obraz*.

```
B09301.mos
#def delka 20
#reg byte obraz[delka]
#reg bit reset, stav
;
P 0
 LD reset
 LET stav
 JMC nic
 LD delka
 LD 0
 FIL obraz
nic:
E 0
```

#### **Příklad 9.3.2**

Na náběžnou hranu signálu *reset* reagujeme vyplněním pole *obraz* ASCII kódem mezery.

```
B09302.mos
#def delka 20
#reg byte obraz[delka]
#reg bit reset, stav
;
P 0
 LD reset
 LET stav
 JMC nic
 LD delka
 LD $2020 ;šířka word!
 FIL obraz
nic:
E 0
```

Kdybychom místo instrukce **LD \$2020** použili instrukci **LD \$20**, pole *obraz* by obsahovalo hodnoty \$20 00 20 00 20 00 20 00 ...

## 10. OPERACE SE STRUKTUROVANÝMI TABULKAMI

### 10.1. Čtení a zápis do strukturované tabulky (LDS, WRS, FIS, FIT)

Operace se strukturovanými tabulkami umožňují práci s tabulkami organizovanými po větších blocích (jednotlivé položky jsou větší než word). Instrukce umožňují čtení a zápis položky z a do tabulky (**LDS**, **WRS**), nebo vyplnění položky konstantou (**FIS**, **FIT**).

Pozor! Zápis do tabulek T při zapnuté uživatelské paměti EEPROM je možný, pokud při překladu uživatelského programu překladačem xPRO označíme volbu *Chráněné tabulky T*, jinak budou po vypnutí a opětovném zapnutí PLC všechny změny ztraceny (z EEPROM se zkopírují hodnoty deklarované ve zdrojovém textu uživatelského programu).

#### Příklad 10.1.1

Požadujeme archivování událostí do tabulky a postupné vyčítání událostí do zóny pro výstupní zařízení. Událost má velikost 8 bytů.

Vycházíme ze zadání příkladu 9.1.2.

```

B10101.mos
#def delka 8
#reg byte udalost[delka], vystup[delka]
#reg byte indexz, indexc
#reg bit zapis, cteni, plno, prazdno
#table byte zasobnik = 0[delka*128] ;vynulování zásobníku
;
P 0
 LD zapis
 JMC skok1
 RES zapis ;nová událost
 RES prazdno
 LD indexz ;INDEX - číslo události
 LD delka ;SIZE - délka události
 LD __indx (zasobnik) ;TAB
 LD __indx (udalost) ;REG
 WRS ;zápis události do zásobníku
 LD indexz
 INR ;zvýšení indexu
 AND 127 ;zacyklení zásobníku
 WR indexz
 LD indexc
 EQ ;test zaplnění
 SET plno ;indexz = indexc -> plno = log.1
skok1:
 LD prazdno
 JMD skok2 ;je co číst?
 LD cteni
 JMC skok2
 RES cteni ;přečtení nejstarší události
 LD indexc ;INDEX - číslo události
 LD delka ;SIZE - délka události
 LD __indx (zasobnik) ;TAB
 LD __indx (vystup) ;REG
 LDS ;čtení události ze zásobníku

```

```

LD indexc
INR ;zvýšení indexu
AND 127 ;zacyklení zásobníku
WR indexc
LD indexz
EQ ;test vyprázdnění
SET prazdno ;indexz = indexc -> prazdno = log.1
skok2:
E 0
;
P 63
LD 1
WR prazdno ;v zásobníku nic není
E 63

```

Na zásobník událostí ukazují dvě ukazovátka, *indexz* pro zápis a *indexc* pro čtení. *Indexz* ukazuje na pozici, na kterou se bude zapisovat, *indexc* ukazuje na pozici, ze které se bude číst. Zápis události do zásobníku je vázán na příznak *zapis*, čtení nejstarší události ze zásobníku do pole *vystup* je vázán na příznak *cteni*. Zásobník je uzavřen do kruhu a má kapacitu 128 událostí.

Pokud po zápisu události a zvýšení ukazovátko *indexz* jsou obě ukazovátko shodná, znamená to zaplnění zásobníku (další nová událost by přepsala nejstarší ještě nevyčtenou událost) - nastaven příznak *plno*. Pokud po přečtení události a zvýšení ukazovátko *indexc* jsou obě ukazovátko shodná, znamená to vyprázdnění zásobníku - nastaven příznak *prazdno*. Tento stav je nastaven jako výchozí po startu PLC.

### Příklad 10.1.2

Předchozí příklad doplníme o smazání události v zásobníku bezprostředně po jejím vyčtení na výstup.

```

B10102.mos
#define delka 8
#reg byte udalost[delka], vystup[delka]
#reg byte indexz, indexc
#reg bit zapis, cteni, plno, prazdno
#table byte zasobnik = 0[delka*128] ;vynulování zásobníku
;
P 0
LD zapis
JMC skok1
RES zapis ;nová událost
RES prazdno
LD indexz ;INDEX - číslo události
LD delka ;SIZE - délka události
LD __indx (zasobnik) ;TAB
LD __indx (udalost) ;REG
WRS ;zápis události do zásobníku
LD indexz
INR ;zvýšení indexu
AND 127 ;zacyklení zásobníku
WR indexz
LD indexc
EQ ;test zaplnění
SET plno ;indexz = indexc -> plno = log.1
skok1:
LD prazdno
JMD skok2 ;je co číst?

```

```

LD cteni
JMC skok2
RES cteni ;přečtení nejstarší události
LD indexc ;INDEX - číslo události
LD delka ;SIZE - délka události
LD __indx (zasobnik) ;TAB
LD __indx (vystup) ;REG
LDS ;čtení události ze zásobníku
POP 1 ;zachování vrstev A1, A2, A3
LD 0 ;VAL
FIT ;smazání přečtené události
LD indexc
INR ;zvýšení indexu
AND 127 ;zacyklení zásobníku
WR indexc
LD indexz
EQ ;test vyprázdnění
SET prazdno ;indexz = indexc -> prazdno = log.1
skok2:
E 0
;
P 63
 LD 1
 WR prazdno ;v zásobníku nic není
E 63

```

### Příklad 10.1.3

Na náběžnou hranu signálu *hotovo* reagujeme vyplněním právě aktivní části pole *obraz* ASCII kódem mezery.

Instrukce **FIS** narozdíl od instrukce **FIL** umožňuje dynamicky měnit všechny parametry. Pokud během programu potřebujeme plnit konstantou různé části zápisníku, lze toto pomocí jedné instrukce **FIS** s měnící se hodnotou REG (index prvního mazaného registru) nebo INDEX (číslo položky), pokud je zápisník pravidelně organizován.

```

B10103.mos
#define delka 20
#reg byte obraz[4*delka], index
#reg bit hotovo, stav
;
P 0
 LD hotovo
 LET stav
 JMC nic
 LD index
 LD delka
 LD __indx (obraz)
 LD $2020 ;šířka word!
 FIS
 LD index
 INR
 AND 3
 WR index ;zacyklení indexu
nic:
E 0

```

Kdybychom místo instrukce **LD \$2020** použili instrukci **LD \$20**, příslušná část pole *obraz* by obsahovala hodnoty \$20 00 20 00 20 00 20 00 ...



## 10.2. Hledání ve strukturovaných tabulkách (FNS, FNT)

Instrukce **FNS** a **FNT** umožňují hledání položky ve strukturované tabulce.

### Příklad 10.2.1

Realizujeme algoritmus, který bude dávat hlášení v předem daných časech. Tento algoritmus lze využít například pro různá hlášení pro údržbu oznamující, že je třeba provést kontrolu určité části zařízení, protože od poslední uběhla už příslušná doba.

*B10201.mos*

```
#struct zaznam byte den, ;definice struktury zaznam
 byte mesic,
 byte rok,
 byte[16] text
#define delka __sizeof (zaznam) ;velikost struktury zaznam v bytech
#table zaznam [3] hlaseni = 10, 5, 97, ' proved kontrolu',
 15, 9, 99, 'proved generálku',
 1, 3, 09, ' do šrotu ! '
#reg zaznam datum ;proměnná datum s položkami den, mesic, rok, text
#reg byte displej[16], index
;
P 0
 LD 2 ;testujeme rok
 LD delka ;velikost položky
 LD __indx (hlaseni) ;prohledávaná tabulka
 LD %S12 ;současný rok
 FNT
 JNS nic
 LD 1 ;testujeme měsíc
 LD delka ;velikost položky
 LD __indx (hlaseni) ;prohledávaná tabulka
 LD %S11 ;současný měsíc
 FNT
 JNS nic
 LD 0 ;testujeme den
 LD delka ;velikost položky
 LD __indx (hlaseni) ;prohledávaná tabulka
 LD %S10 ;dnešek
 FNT
 JNS nic
 WR index ;index nalezené položky
 LD delka ;velikost položky
 LD __indx (hlaseni) ;tabulka
 LD __indx (datum) ;zápisník
 LDS datum ;načtení aktuální položky
 LD 0
 SRC datum~text[0]
 LD 0
 LD 16
 MOV displej ;zobrazení textu
 LD index
 JMD zmena ;hlášení 0
 LD datum~mesic
 DIV 12 ;12 -> 0
 SWP
 INR
 WR datum~mesic ;za měsíc opakovat
```

## 10. Operace se strukturovanými tabulkami

---

```
 AND $FF00
 JMC zapis ;nový rok?
 INR datum~rok
 JMP zapis
;
zmena:
 CMP 1
 JNZ nic ;hlášení 1
 LD datum~rok
 ADD 5
 WR datum~rok ;další za 5 let
zapis:
 LD index
 LD delka ;velikost položky
 LD __indx (hlaseni) ;tabulka
 LD __indx (datum) ;zápisník
 WRS ;zápis opravené položky
nic:
E 0
```

Pro zjednodušení jsme neuvažovali korekci datumu na pracovní dny (zde je nepřetržitý provoz). Do algoritmu lze doplnit potvrzování provedené údržby a další akce v závislosti na vyvolaném hlášení.

## 11. ARITMETICKÉ INSTRUKCE V PLOVOUCÍ ŘÁDOVÉ ČÁRCE

### 11.1. Příklady instrukcí ADF, SUF, MUF, DIF

Aritmetické instrukce pracující s vrcholem zásobníku a vstupní proměnnou ve formátu float umožňují sčítání, odčítání, násobení, dělení (**ADF**, **SUF**, **MUF**, **DIF**).

Tytéž instrukce existují též jako bezoperandové, pracující na dvojrvcích 01 a 23 aktivního zásobníku.

Mějme stále na paměti, že formát float má naprosto odlišné kódování od ostatních formátů a že pro převody čísel musíme použít speciální převodní instrukce.

#### **Příklad 11.1.1**

Proveďme operaci  $a = b - c$ .

```
B11101.mos
#reg float va, vb, vc
;
P 0
 LD vb
 SUF vc
 WR va
E 0
```

#### **Příklad 11.1.2**

Proveďme operaci  $a = b + c$ .

```
B11102.mos
#reg float va, vb, vc
;
P 0
 LD vb
 ADF vc
 WR va
E 0
```

#### **Příklad 11.1.3**

Obsah proměnné *cislo* zvětšeme o konstantu 13 541.

```
B11103.mos
#reg float cislo
;
P 0
 LD cislo
 ADF 13541.0
 WR cislo
E 0
```

### Příklad 11.1.4

Do  $a$  uložme součin  $b$  a  $c$ .

```
B11104.mos
#reg float va, vb, vc
;
P 0
 LD vb
 MUF vc
 WR va
E 0
```

### Příklad 11.1.5

Provedte operaci  $a = \frac{b - c \cdot 2,95}{d + 4,71}$ .

```
B11105.mos
#reg float va, vb, vc, vd
;
P 0
 LD vb
 LD vc
 MUF 2.95
 SUF
 LD vd
 ADF 4.71
 DIF
 WR va
E 0
```

## 11.2. Porovnání a limitní funkce (CMF)

Instrukce **CMF** nastavuje v S0 příznaky porovnání dvou hodnot, zásobník zůstává nezměněn.

Příznaky mají následující význam (u operandových instrukcí je  $a$  dvojvrstva A01,  $b$  operand, u bezoperandových instrukcí je  $a$  dvojvrstva A23,  $b$  dvojvrstva A01):

|                         |                         |
|-------------------------|-------------------------|
| S0.0 = log.1 $a = b$    | S0.0 = log.0 $a \neq b$ |
| S0.1 = log.1 $a < b$    | S0.1 = log.0 $a \geq b$ |
| S0.2 = log.1 $a \leq b$ | S0.2 = log.0 $a > b$    |

### Příklad 11.2.1

Nastavte bit výsledek, pokud  $\text{cislo1} \neq \text{cislo2}$ .

```
B11201.mos
#reg float cislo1, cislo2
#reg bit vysledek
;
P 0
 LD cislo1
 CMF cislo2
 LDC %S0.0
 WR vysledek
```

E 0

### Příklad 11.2.2

Nastavme bit *vysledek*, pokud *cislo* = 23 115.

```
B11202.mos
#reg float cislo
#reg bit vysledek
;
P 0
 LD cislo
 CMF 23115.0
 LD %S0.0
 WR vysledek
E 0
```

### Příklad 11.2.3

Nastavme bit *vysledek.0*, pokud *cislo* = 23 115, bit *vysledek.1*, pokud *cislo* < 23 115 a bit *vysledek.2*, pokud *cislo* ≤ 23 115.

V našem případě mají bity *vysledek.0*, *.1*, *.2* stejný význam jako bity v S0. Pokud tedy budou tyto bity součástí jednoho bytu *vysledek*, stačí obsah S0 přesunout najednou.

```
B11203.mos
#reg float cislo
#reg byte vysledek
;
P 0
 LD cislo
 CMF 23115.0
 LD %S0
 WR vysledek
E 0
```

### Příklad 11.2.4

Nastavme bit *vysledek*, pokud *cislo* ≥ 15 734.

```
B11204.mos
#reg float cislo
#reg bit vysledek
;
P 0
 LD cislo
 CMF 15734.0
 LDC %S0.1
 WR vysledek
E 0
```

### Příklad 11.2.5

Nastavme bit *vysledek* na log.1, pokud *cislo* = 0.

I ve formátu float není nutná žádná komparace, protože hodnota 0 je opravdu nulová.

```
B11205.mos
#reg float cislo
#reg bit vysledek
;
```

```
P 0
 LD cislo
 WRC vysledek
E 0
```

### Příklad 11.2.6

Nastavme bit *vysledek* na log.1, pokud je splněna dvojitá nerovnost  $19,5 \leq cislo \leq 34$  (byte *cislo* je uvnitř uzavřeného intervalu). Splněny musí být současně obě dílčí nerovnosti (tedy AND obou dílčích výsledků).

Klasické řešení vypadá takto:

```
B11206a.mos
#reg float cislo
#reg bit vysledek
;
P 0
 LD cislo
 CMF 34.0
 LD %S0.2
 WR vysledek
 POP 1 ;vrácení hodnoty cislo na vrchol zásobníku
 CMF 19.5
 LDC %S0.1
 AND vysledek
 WR vysledek
E 0
```

Příklad má obdobné zadání jako příklad 4.4.9, kde jsme si pomohli posunem rozsahu k nule, jenže tam jsme předpokládali pouze kladná čísla. Pokud uvažujeme i záporná čísla, což v případě formátu float musíme, odečteme od mezí hodnotu získanou součtem původní dolní meze a poloviny rozdílu obou mezí ( $19,5 + 7,25 = 26,75$ ). Celý testovaný interval se posune tak, že je symetrický vůči nule:  $-7,25 \leq cislo \leq 7,25$ , což je rovnocenné jediné nerovnosti  $|cislo| \leq 7,25$ .

```
B11206b.mos
#reg float cislo
#reg bit vysledek
;
P 0
 LD cislo
 SUF 26.75
 ABS
 CMF 7.25
 LD %S0.2
 WR vysledek
E 0
```

### Příklad 11.2.7

Požadujeme realizovat omezovač maximální hodnoty obsahu proměnné *cislo*. Pokud je obsah  $cislo \leq 21,5$ , zůstává zachován, při překročení této meze je touto mezí nahrazen, tj.  $cislo = 21,5$ .

```
B11207.mos
#reg float cislo
;
P 0
 LD cislo
```

```
CMF 21.5
JC nic
LDL 21.5
WR cislo
```

```
nic:
E 0
```

### 11.3. Matematické funkce (CEI, FLO, ABS, LOG, EXP, LN, POW, SQR, SIN, ASN, COS, ACS, TAN, ATN, HYP)

Instrukce matematických funkcí pracující se zásobníkem umožňují zaokrouhlování (**CEI**, **FLO**), absolutní hodnotu (**ABS**), logaritmické a exponenciální funkce (**LOG**, **EXP**, **LN**), mocninu a odmocninu (**POW**, **SQR**), goniometrické funkce (**SIN**, **ASN**, **COS**, **ACS**, **TAN**, **ATN**) a Euklidovskou vzdálenost (**HYP**).

Instrukce matematických funkcí provádějí danou operaci s proměnnou formátu float uloženou na vrcholu zásobníku (dvojvrstva A01). Jejich možnosti a použití jsou zřejmé už z popisu těchto instrukcí.

#### Příklad 11.3.1

Zaokrouhleme proměnnou *cislo* na celé číslo podle matematických zvyklostí (tj. x,0 až x,4 dolů a x,5 až x,9 nahoru).

Musíme si pomoci instrukcí **FLO**, která odstraňuje desetinná místa. Zaokrouhluje tedy vždy dolů. Abychom dosáhli matematického zaokrouhlování, hodnotu *cislo* zvětšíme o 0,5. Toto však platí pouze pro kladná čísla. Pro čísla záporná musíme naopak 0,5 odečíst. Pokud víme, že budeme pracovat jen s kladnými čísly, můžeme zápornou větev vynechat.

```
B11301.mos
#reg float cislo
;
P 0
 LD cislo.31 ;nejvyšší bit udává znaménko
 JMD minus
 LD cislo ;kladné číslo
 ADF 0.5
 JMP skok
minus:
 LD cislo ;záporné číslo
 SUF 0.5
skok:
 FLO
 WR cislo
E 0
```

#### Příklad 11.3.2

$$\text{Vypočítejme výraz } d = \frac{\sqrt[3]{a^2 + b^3}}{\sin c}.$$

```
B11302.mos
#reg float va, vb, vc, vd
;
P 0
 LD va ; 2
 MUF va ;a
```

```

LD vb
LDL 3.0 ; 3
POW ;b
ADF ;součet
LDL 0.3333333
POW ;třetí odmocnina
LD vc
SIN ;sinus
DIF ;zlomek
WR vd
E 0

```

### 11.4. Převody čísel mezi formáty (UWF, IWF, ULF, ILF, UFW, IFW, UFL, IFL)

Převody mezi formátem float a celočíselnými formáty šířky word a long se znaménkem i bez znaménka provádějí instrukce:

|            |                       |
|------------|-----------------------|
| <b>UWF</b> | unsigned word → float |
| <b>IWF</b> | signed word → float   |
| <b>ULF</b> | unsigned long → float |
| <b>ILF</b> | signed long → float   |
| <b>UFW</b> | float → unsigned word |
| <b>IFW</b> | float → signed word   |
| <b>UFL</b> | float → unsigned long |
| <b>IFL</b> | float → signed long   |

#### Příklad 11.4.1

Na vstup přicházejí pulzy odměřující jednosměrně posuv o 2,5  $\mu\text{m}$ . Hodnotu čítače převedme pro další výpočty na milimetry ujeté dráhy.

Použijeme proměnnou *citac* typu word, její obsah v případě potřeby převedeme na typ float a potom přepočítáme na mm.

```

B11401.mos
#reg bit vstup, reset
#reg word citac
#reg float draha
;
P 0
 LD vstup
 LD reset
 CTU citac
 UWF ;převod na float
 MUF 0.0025 ;přepočet na mm
 WR draha
E 0

```

Mohli bychom realizovat čítač typu float. Je však třeba si uvědomit, že formát float poskytuje sice velký rozsah čísel, ale s přesností na 5 platných číslic. To znamená, že pokud budeme proměnnou typu float zvyšovat o 1, po překročení hodnoty 100 000 se hodnota přestane zvyšovat, protože oproti hodnotě  $1 \times 10^5$  bude hodnota 1 za hranicí přesnosti. Pokud nám tedy nestačí čítač šířky word, musíme použít kaskádování čítačů.



**Příklad 11.4.2**

Na *vstup1* přicházejí pulzy odměřující posuv o 2,5 µm v kladném směru, na *vstup2* přicházejí pulzy odměřující posuv o 2,5 µm v záporném směru. Polohu přepočteme na milimetry.

Použijeme proměnnou *citac* typu word, její obsah převedeme na typ float a potom přepočítáme na mm. Tentokrát využíváme nejvyšší bit hodnoty jako znaménko.

```
B11402.mos
#reg bit vstup1, vstup2, reset
#reg word citac
#reg float poloha
;
P 0
 LD vstup1 ;nahoru
 LD vstup2 ;dolů
 LD reset
 CNT citac
 IWF ;převod na float včetně znaménka
 MUF 0.0025 ;přepočet na mm
 WR poloha
E 0
```

Pokud nám nestačí čítač šířky word, musíme použít kaskádování čítačů. O realizaci čítače typu float platí totéž, co v předchozím příkladu.

**Příklad 11.4.3**

Před převodem čísla z formátu float na formát long provedme jeho zaokrouhlení.

Pokud chceme zaokrouhlit číslo vždy nahoru, před převodem použijeme instrukci **CEI**. Pokud chceme zaokrouhlit číslo vždy dolů, stačí provést převod a desetinná místa budou odříznuta. Pokud ale chceme zaokrouhlit matematicky, tedy od x,0 až x,4 dolů a od x,5 do x,9 nahoru, vyřešíme problém přičtením 0,5 (viz příklad 11.3.1).

```
B11403a.mos
#reg float cislo
#reg long vysledek
;
P 0
 LD cislo
 ADF 0.5
 UFL ;převod na long bez znaménka
 WR vysledek
E 0
```

Přičtením 0,5 jsme posunuli číslo tak, že pak už stačí jen oříznout desetinná místa a výsledek je zaokrouhlen. Toto ale platí jen pro kladná čísla. U záporných čísel naopak musíme 0,5 odečíst. Pokud tedy budeme pracovat i se zápornými čísly, musíme algoritmus rozšířit.

```
B11403b.mos
#reg float cislo
#reg long vysledek
;
P 0
 LD cislo.31 ;nejvyšší bit udává znaménko
 JMD zaporne
 LD cislo ;kladné číslo
 ADF 0.5
```

## 11. Aritmetické instrukce v plovoucí řádové čárce

---

```
 JMP hotovo
;
zaporne:
 LD cislo ;záporné číslo
 SUF 0.5
hotovo:
 IFL ;převod na long se znaménkem
 WR vysledek
E 0
```

## 12. INSTRUKCE REGULÁTORU PID

Používání instrukcí **PID** a **CNV** a programování regulace obecně je natolik samostatný a rozsáhlý problém, že není v silách této příručky se mu věnovat.

Regulací se zabývá nástroj PIDMaker, který je součástí vývojového prostředí Mosaic. Tento nástroj pomocí nastavení parametrů regulace sám vytvoří algoritmus regulátoru PID, který se stane součástí uživatelského programu. PIDMaker navíc umožňuje sledovat chování regulátoru jak v simulované soustavě, tak i v reálném prostředí.

## 13. OPERACE SE ZNAKY ASCII

### 13.1. Operace se znaky ASCII (BAS, ASB, STF, FST)

Sada kódů ASCII obsahuje číselnou reprezentaci písmen a dalších znaků, kterou používají znakové zobrazovače. Instrukce **BAS** a **ASB** provádí převod binárního čísla šířky word na 4 znaky ASCII a naopak. Instrukce **STF**, **FST** provádí převod čísla ve formátu float na textový řetězec a naopak.

#### Příklad 13.1.1

Dvojkový obsah proměnné *cislo* šířky word převedme do BCD kódu a předejme do proměnné *obraz*, která je zobrazována na operačním panelu. Překročení rozsahu 9999 signalizujeme na bitu *pretececi*.

```
B13101.mos
#reg long obraz
#reg word cislo
#reg bit pretececi
;
P 0
 LD cislo
 BCD ;převod na BCD
 BAS ;převod na ASCII
 WR obraz
 LD %S0
 AND %1110000 ;5. číslice převodu na BCD
 WR pretececi
E 0
```

#### Příklad 13.1.2

Zobrazujeme hodnotu proměnné *cislo* v plném rozsahu šířky long. Požadujeme, aby byly zobrazeny pouze platné číslice (aby např. číslo 25 nebylo zobrazeno jako 0000000025).

```
B13102.mos
#reg long cislo
#reg byte obraz[10], index
;
P 0
 LD cislo
 BCL ;převod na BCD
 BAS
 WR long obraz+6 ;7. až 10. číslice
 POP 2
 BAS
 WR long obraz+2 ;3. až 6. číslice
 POP 2
 BAS
 WR word obraz ;1. a 2. číslice
 LD 9 ;mez tabulky
 LD 0
 WR index ;index pole
cyklus:
 LTB obraz
```

```

CMP $30 ;nebo CMP `0'
JNZ konec ;nula na začátku?
LD 9 ;mez tabulky
LD index ;index pole
LD $20 ;nebo LD ` '
WTB obraz ;přepis nuly mezerou
INR index ;další číslice
LD 9 ;mez tabulky
LD index
CMP 9
JNZ cyklus ;zbyla poslední číslice?

```

konec:

E 0

### Příklad 13.1.3

Převědme hodnotu zadanou na klávesnici v ASCII formátu do pole *obraz* na číselnou hodnotu šířky long.

*B13103.mos*

#reg long cislo

#reg byte obraz[10], pomoc[10], indexo, indexp, zaloha

;

P 0

```

LD 10
LD $3030
FIL pomoc ;vynulování pomocného pole
LD 9 ;mez tabulky
LD 9
WR indexo ;index pole
WR indexp

```

cyklus:

```

LD 9
LD indexo
LTB obraz
CMP $20 ;nebo CMP ` '
JNZ skok ;mezera na začátku?
WR zaloha
LD 9 ;mez tabulky
LD indexp ;index pole
LD zaloha
WTB pomoc ;posun čísla vpravo
DCR indexp

```

skok:

```

DCR indexo ;další číslice
LD indexo
CMP 255
JNZ cyklus ;zbyla číslice?
LD word pomoc ;1. a 2. číslice
LD $3030 ;doplnění nulami v ASCII na long
ASB
LD long pomoc+2 ;3. až 6. číslice
ASB
LD long pomoc+6 ;7. až 10. číslice
ASB
BIL
;převod do binárního kódu
WR cislo

```

E 0

### **Příklad 13.1.4**

Pole *obraz* je zobrazováno na operačním panelu. Je požadováno zpracovat proměnnou *cislo* formátu float tak, aby zobrazení jejího rozsahu plně využívalo velikost pole.

```
B13104.mos
#def delka 12
#reg byte obraz[delka]
#reg float cislo
;
P 0
 LD cislo
 LD __indx (obraz)
 LD delka
 FST ;převod na ASCII
E 0
```

### **Příklad 13.1.5**

Převeďme hodnotu zadanou na klávesnici v ASCII formátu do pole *obraz* na číselnou hodnotu formátu float.

```
B13105.mos
#def delka 12
#reg byte obraz[delka]
#reg float cislo
;
P 0
 LD __indx (obraz)
 LD delka
 STF ;převod z ASCII
 WR cislo
E 0
```

## 14. SYSTÉMOVÉ INSTRUKCE

### 14.1. Ovládání odezvy komunikací (HPE, HPD)

Vizualizační programy používají pro výměnu dat s PLC datové služby režimu PC (viz příručka Sériová komunikace programovatelných automatů TECOMAT a regulátorů TECOREG TXV 001 06.01). Standardně tyto komunikace mají tzv. nízkou prioritu (low priority), což znamená, že po přijetí služby PLC počká na dokončení smyčky uživatelského programu a v otočce provede požadovanou akci (zápis nebo čtení dat). Pak zahájí vysílání odpovědi.

Výhodou tohoto přístupu je zaručená časová konzistence dat, tj. že všechna přečtená data pocházejí ze stejného časového okamžiku. Dále je zde zaručen princip neměnnosti obsahu zápisníku během smyčky uživatelského programu jiným způsobem než činností uživatelského programu samého.

Nevýhodou je zpoždění odpovědi PLC až o dobu cyklu, což při komunikaci s více systémy může znamenat znatelný pokles propustnosti linky. Tuto nevýhodu lze potlačit pomocí instrukcí **HPE** a **HPD**.

#### Příklad 14.1.1

Máme síť pěti PLC připojených k jednomu počítači PC s vizualizací. Datové parametry sítě jsou uvedeny v tab.14.1.

Tab.14.1 Parametry sítě podřízených PLC z příkladu 14.1.1

| PLC  | adresa | rychlost | prodleva odpovědi | počet čtených dat | počet zapisovaných dat | doba cyklu |
|------|--------|----------|-------------------|-------------------|------------------------|------------|
| PLC0 | 0      | 19,2 kBd | 0 (1 byte)        | 80                | 20                     | 100 ms     |
| PLC1 | 1      |          | 0 (1 byte)        | 50                | 30                     | 60 ms      |
| PLC2 | 2      |          | 0 (1 byte)        | 100               | 40                     | 200 ms     |
| PLC3 | 3      |          | 0 (1 byte)        | 40                | 10                     | 50 ms      |
| PLC4 | 4      |          | 0 (1 byte)        | 60                | 50                     | 250 ms     |

Požadujeme zkonfigurovat síť tak, aby se všechna data vyměnila minimálně dvakrát za sekundu.

V tab.14.2 je uvedena časová analýza sítě podle zadání.

Tab.14.2 Výchozí časová analýza sítě z příkladu 14.1.1

| PLC       | rychlost     | prodleva odpovědi | přenášená data + rámce a klidové stavy linky | celková doba komunikace |
|-----------|--------------|-------------------|----------------------------------------------|-------------------------|
| PLC0      | 0,57 ms/byte | 0,57 - 100 ms     | 100 + 22 bytů                                | 69,54 ms                |
| PLC1      |              | 0,57 - 60 ms      | 80 + 22 bytů                                 | 58,14 ms                |
| PLC2      |              | 0,57 - 200 ms     | 140 + 22 bytů                                | 92,34 ms                |
| PLC3      |              | 0,57 - 50 ms      | 50 + 22 bytů                                 | 41,04 ms                |
| PLC4      |              | 0,57 - 250 ms     | 110 + 22 bytů                                | 75,24 ms                |
| dohromady |              | 2,85 - 660 ms     | 480 + 110 bytů                               | 336,30 ms               |

Výsledný komunikační cyklus sítě je součet trvání přenosu dat a prodlevy odpovědi všech PLC a jak je uvedeno v tabulce, výsledná hodnota se pohybuje v rozmezí 339 až 996 ms. Požadavek výměny dat ve vizualizaci dvakrát za sekundu tedy není splněn.

Zde nepomůže žádné zvýšení komunikační rychlosti, protože maximální celková prodleva odpovědi PLC, jak vyplývá z údajů v tabulce 14.2, je 660 ms. To je čas, který nezávisí na komunikační rychlosti, ale na době cyklu každého PLC.

Data pro vizualizaci z výše uvedeného příkladu by se při použití instrukcí **HPD**, **HPE** obnovovala téměř třikrát za sekundu.

Pro výměnu dat vyhradíme zvláštní komunikační zóny v zápisníku, které bude uživatelský program zpracovávat a aktualizovat jen na jednom místě. Pokud je třeba do komunikačních zón přistupovat víckrát, je lepší vytvořit tzv. stínové komunikační zóny, se kterými bude pracovat uživatelský program a které se jednou za cyklus sesouhlasí se skutečnými komunikačními zónami.

*B14101.mos*

```
#def delkadovnitř 10
#def delkaven 10
#reg byte stindovnitř[delkadovnitř],stinven[delkaven]
#reg byte dovnitř[delkadovnitř],ven[delkaven]
;
P 0
:
: ;uživatelský program pracuje výlučně
: ;se zónami stindovnitř a stinven
:
HPD ;nízká priorita
LD 0
SRC dovnitř
LD 0
LD delkadovnitř
MOV stindovnitř ;kopie přijímací zóny
LD 0
SRC stinven
LD 0
LD delkaven
MOV ven ;kopie vysílací zóny
HPE ;vysoká priorita
:
: ;uživatelský program pracuje výlučně
: ;se zónami stindovnitř a stinven
:
E 0
;
P 63
:
HPE ;zapnutí vysoké priority
:
E 63
```

### 14.2. Čtení a zápis do obvodu reálného času (RDT, WRT)

Instrukce **RDT** slouží k vytváření přesných časových značek k určité události (obvykle zpracovávané v přerušovacím procesu). Zatímco v registrech S5 až S12 je čas aktualizován vždy v otočce cyklu a během zpracovávání uživatelského programu je tedy neměnný, instrukce **RDT** čte aktuální čas přímo z obvodu reálného času centrální jednotky a provádí synchronizační korekci.



Časová zóna v zápisníku má následující strukturu:

| index registru | časový údaj | rozsah                                      |
|----------------|-------------|---------------------------------------------|
| REG            | rok         | 0 - 99                                      |
| REG+1          | měsíc       | 1 - 12                                      |
| REG+2          | den         | 1 - 28 / 29 / 30 / 31 (podle měsíce a roku) |
| REG+3          | hodina      | 0 - 23                                      |
| REG+4          | minuta      | 0 - 59                                      |
| REG+5          | sekunda     | 0 - 59                                      |
| REG+6          | den v týdnu | 1 - 7                                       |
| REG+7, +8      | milisekunda | 0 - 999 (v pořadí dolní byte, horní byte)   |

Položka *milisekunda* je umístěna až za *den v týdnu* z důvodu kompatibility se strukturou používanou instrukcí **WRT**. Časový údaj čtený instrukcí **RDT** je synchronizován, to znamená, že okamžikem zápisu času do RTC buď instrukcí **WRT** nebo komunikační službou po sériové lince je vynulován údaj milisekund. Naproti tomu v registrech S5 až S12 je čas průběžný, to znamená, že milisekundy se nenulují. Tento čas je oproti časové značce čtené instrukcí **RDT** posunut o 0 až 999 ms. Proto není vhodné používat souběžně oba časové údaje.

Instrukce **WRT** slouží k přenastavení obvodu reálného času centrální jednotky. To má význam zejména pro změnu času z letního na zimní a naopak, případně pro synchronizaci času s vnějším signálem.

Časová zóna v zápisníku má následující strukturu:

| index registru | časový údaj | rozsah                                      |
|----------------|-------------|---------------------------------------------|
| REG            | rok         | 0 - 99                                      |
| REG+1          | měsíc       | 1 - 12                                      |
| REG+2          | den         | 1 - 28 / 29 / 30 / 31 (podle měsíce a roku) |
| REG+3          | hodina      | 0 - 23                                      |
| REG+4          | minuta      | 0 - 59                                      |
| REG+5          | sekunda     | 0 - 59                                      |
| REG+6          | den v týdnu | 1 - 7                                       |

Všechny časové údaje jsou ukládány v binárním kódu.

### Příklad 14.2.1

Požadujeme přesnou časovou značku při výskytu události. Pro větší přesnost budeme kritické vstupy vzorkovat v přerušení 10 ms (P41). Záznamy budeme ukládat do kruhového zásobníku s kapacitou 256 záznamů.

*B14201.mos*

```
#def periferie U$8000 ;fyzická adresa periferie
 ;závisí na typu PLC (zde NS950)
#def pocet 256 ;velikost zásobníku záznamů
#struct cas
 byte rok, byte mesic, byte den, byte hod, byte min,
 byte sek, byte denvt, word milisek
#struct zaznam
 word vyskyt, word hodnota, cas znacka
#reg word odloz, ukaz
#reg zaznam historie[pocet] ;zásobník záznamů
#reg zaznam prac ;pracovní záznam
;
```

```

P 0
:
E 0
;
P 41
 LD periferie ;přímé čtení z periferie
 BET odloz ;vyhodnocení změny proti původní hodnotě
 EC ;pokud není změna, skončit
 WR prac~vyskyt ;na místech změny jsou jedničky
 LD odloz
 WR prac~hodnota ;aktuální stav periferie
 LD __indx (prac~znacka)
 RDT ;zápis přesné časové značky
 LD 0
 SRC prac
 LD ukaz ;ukazovátka do zásobníku záznamů
 LD __sizeof (zaznam)
 MOV historie[0] ;přesun záznamu do zásobníku
 LD ukaz
 ADX __sizeof (zaznam) ;posun ukazovátka na další záznam
 WR ukaz
 EQ pocet * __sizeof (zaznam) ;zacyklení ukazovátka
 NEG
 AND ukaz ;při dosažení meze se nuluje ukazovátka
 ;(AND 0)
 WR ukaz
E 41

```

### Příklad 14.2.2

Požadujeme automatickou změnu času při přechodu na letní čas a zpět.

Program pracuje na principu porovnávání aktuálního času s časem přechodu uloženém v tabulce. V tabulce *letnicas* jsou datумы a hodiny přechodu na letní čas a v tabulce *zimnicas* jsou datумы a hodiny přechodu na zimní čas. Po zapnutí se v procesu P63 zjišťuje podle aktuálního času, jestli je právě letní nebo zimní čas a jestli došlo ke změně času po dobu vypnutého napájení. Bitová proměnná *letni* musí být remanentní.

*B14202.mos*

```

#rem bit letni ;příznak letního času - remanentní zóna
;
#usi ucmparray = cmparray
#define CMPARRAY usi ucmparray
#struct cas byte rok, byte mesic, byte den, byte hod, byte min,
 byte sek, byte denvt
#table cas[2] letnicas =
 01, 03, 25, 02, 00, 00, 07, ;25. 3. 2001, 2:00:00, neděle
 02, 03, 24, 02, 00, 00, 07, ;24. 3. 2002, 2:00:00, neděle
#table cas[2] zimnicas =
 01, 10, 21, 03, 00, 00, 07, ;21. 10. 2001, 3:00:00, neděle
 02, 10, 20, 03, 00, 00, 07, ;20. 10. 2002, 3:00:00, neděle
;
#reg cas prechod ;pomocná proměnná s časem přechodu
#reg cas pracovni ;pomocná proměnná s aktuálním časem
#reg word milisek ;musí být za proměnnou pracovni kvůli
 ;vyrovnání rozdílu velikosti položek

#reg word tabulka
#reg byte delka, index
;

```

```

P 0
 LD letni
 JMD test1
 LD __indx (letnicas) ;ted' je zimní čas, bude se testovat
 ;přechod na letní
 JMP test2
;
test1:
 LD __indx (zimnicas) ;ted' je letní čas, bude se testovat
 ;přechod na zimní
test2:
 WR tabulka ;číslo tabulky s porovnávanými časy
 CAL zmena ;kontrola změny času
E 0
;
P 60
;
zmena:
 LD 0 ;index údaje - rok
 LD __sizeof (cas) ;velikost položky
 LD tabulka ;tabulka přechodů na zimní / letní čas
 LD %S12 ;rok
 FNT ;hledání přechodu příslušného roku
 LD __sizeof (cas) ;velikost položky
 LD tabulka ;tabulka přechodů na zimní / letní čas
 LD __indx (prechod) ;pomocná proměnná
 LDS ;načtení nejbližšího přechodu
 LD __indx (pracovni)
 RDT ;načtení aktuálního času
 LD __indx (prechod) ;položka 1
 LD __indx (pracovni) ;položka 2
 LD __sizeof (cas) ;délka položek
 CMPARRAY ;USI pro porovnání dlouhých položek
 REC ;výsledek = 0 -> položky nejsou shodné
 ; - žádná změna

 LD letni
 JMD sniz
 INR pracovni~hod ;přechod zimní -> letní čas
 LD __indx (pracovni)
 WRT ;oprava aktuálního času
 LD 1
 WR letni ;nastavit příznak letního času
 RET
;
sniz:
 DCR pracovni~hod ;přechod letní -> zimní čas
 LD __indx (pracovni)
 WRT ;oprava aktuálního času
 LD 0
 WR letni ;smazat příznak letního času
 LD 1
 WR opak ;nastavení příznaku opakované hodiny
 RET
;
E 60
;

```

P 63

```

;naastavení příznaku letního času porovnáním aktuálního času
;s přechodovými časy
LD 0 ;index údaje - rok
LD __sizeof (cas) ;velikost položky
LD __indx (letnicas) ;tabulka přechodů na letní čas
LD %S12 ;rok
FNT ;hledání přechodu příslušného roku
LD __sizeof (cas) ;velikost položky
LD __indx (letnicas) ;tabulka přechodů na letní čas
LD __indx (prechod) ;pomocná proměnná
LDS ;načtení letošního přechodu
LD __indx (pracovni)
RDT ;načtení aktuálního času
LD __sizeof (cas)
WR delka
LD 0
WR index
cyklus1:
LD __sizeof (cas)
LD index
LTB prechod ;byte z okamžiku přechodu
POP 2 ;odložení na A6
LTB pracovni ;odpovídající byte aktuálního času
POP -1 ;posun načtených hodnot na A0, A1
CMP ;porovnání
JC zima ;skok - čas před změnou
JNZ jaro ;skok - čas po změně
INR index
DCR delka ;čas stejný - porovnávat dál
JNZ cyklus1
jaro:
LD 0 ;index údaje - rok
LD __sizeof (cas) ;velikost položky
LD __indx (zimnicas) ;tabulka přechodů na zimní čas
LD %S12 ;rok
FNT ;hledání přechodu příslušného roku
LD __sizeof (cas) ;velikost položky
LD __indx (zimnicas) ;tabulka přechodů na zimní čas
LD __indx (prechod) ;pomocná proměnná
LDS ;načtení letošního přechodu
LD __sizeof (cas)
WR delka
LD 0
WR index
cyklus2:
LD __sizeof (cas)
LD index
LTB prechod ;byte z okamžiku přechodu
POP 2 ;odložení na A6
LTB pracovni ;odpovídající byte aktuálního času
POP -1 ;posun načtených hodnot na A0, A1
CMP ;porovnání
JC leto ;skok - čas před změnou
JNZ zima ;skok - čas po změně
INR index
DCR delka ;čas stejný - porovnávat dál
JNZ cyklus2

```

```

leto:
 LD letni
 JMD hotovo ;došlo ke změně času během vypnutí?
 INR pracovni~hod ;přechod zimní -> letní čas
 LD __indx (pracovni)
 WRT
 LD 1 ;oprava aktuálního času
 ;letní čas
 WR letni
 JMP hotovo

;
zima:
 LD letni
 JMC hotovo ;došlo ke změně času během vypnutí?
 DCR pracovni~hod ;přechod letní -> zimní čas
 LD __indx (pracovni)
 WRT
 LD 0 ;oprava aktuálního času
 ;zimní čas
 WR letni

hotovo:
E 63

```

### 14.3. Práce s pamětí DataBox (IDB, RDB, WDB)

Přídavná paměť DataBox je určena především pro archivaci většího množství dat. Je přístupná z uživatelského programu pomocí speciálních instrukcí **IDB**, **RDB**, **WDB** a zvenčí po sériové lince.

#### Příklad 14.3.1

Požadujeme archivování událostí do DataBoxu. Záznam o události má velikost 8 bytů.

```

B14301.mos
#struct parDB ;jméno struktury
 long adrDB, ;adresa v DataBoxu
 word indR, ;index počátečního registru v zápisníku
 byte len ;počet přenášených bytů
#def velikost 8 ;délka události
#def startDBX 0 ;první adresa archivu v DataBoxu
#def konecDBX $1FFFB ;poslední adresa archivu v DataBoxu
#def maxDBX konecDBX-velikost+1
 ;max. adresa poslední položky archivu v DataBoxu

#reg parDB parametry
#reg byte udalost[delka]
#reg bit DataBoxOK ;příznak DataBox v pořádku
;
P 0
 LD DataBoxOK ;DataBox v pořádku ?
 JMC zaver ;ne
 LD __indx (parametry) ;číslo registru, kde leží parametry
 WDB
 LD parametry~adrDB
 ADL velikost ;posun adresy
 CML maxDBX ;test konce archivu
 JC dale
 LDL startDBX ;zacyklení archivu
dale:
 WR parametry~adrDB ;adresa pro budoucí zápis

```

```

zaver:
E 0
;
P 63
 LD 128 ;požadovaná velikost DataBoxu v KB
 IDB ;identifikace velikosti DataBoxu
 GT
 NEG ;DataBox aspoň požadované velikosti?
 WR DataBoxOK ;nastavit příznak
 LDL startDBX
 WR parametry~adrDB ;nastavení adresy do DataBoxu
 ;na začátek archivu
 LD __indx (udalost) ;ze kterého registru se přenáší
 ;záznamy do DataBoxu

 WR parametry~indR
 LD velikost ;počet přenášených bytů záznamu
 WR parametry~len
E 63

```

### Příklad 14.3.2

Řízená technologie vyžaduje za chodu měnit parametry chování uživatelského programu. Toho můžeme dosáhnout tak, že v DataBoxu vytvoříme archiv receptur (programů, parametrů, apod.), ze kterého si vybereme požadovanou recepturu. Do tohoto archivu můžeme nahrávat nové receptury i nezávisle na chodu uživatelského programu. Buď máme vytvořený mechanismus zápisu receptury v uživatelském programu (podle předchozího příkladu), nebo do DataBoxu zapisujeme přímo po sériové lince z nadřazeného systému.

```

B14302.mos
#struct parDB ;jméno struktury
 long adrDB, ;adresa v DataBoxu
 word indR, ;index počátečního registru v zápisníku
 byte len ;počet přenášených bytů
#define delka 24 ;délka receptury
#define startDBX 0 ;první adresa archivu receptur v DataBoxu
#reg parDB parametry
#reg byte kod ;číslo receptury
#reg byte receptura[delka] ;obsah receptury
 ;(místo pole může být struktura)
#reg bit DataBoxOK ;příznak DataBox v pořádku
;
P 0
 LD DataBoxOK ;DataBox v pořádku ?
 JMC konec ;ne
 LD kod ;číslo požadované receptury
 MUD delka ;výpočet adresy receptury
 ADL startDBX
 WR parametry~adrDB ;nastavení adresy receptury
 LD __indx (parametry) ;číslo registru, kde leží parametry
 RDB ;přečtení receptury z DataBoxu

konec:
E 0
;
P 63
 LD 128 ;požadovaná velikost DataBoxu v KB
 IDB ;identifikace velikosti DataBoxu
 GT

```

|     |                    |                                      |
|-----|--------------------|--------------------------------------|
| NEG |                    | ;DataBox aspoň požadované velikosti? |
| WR  | DataBoxOK          | ;nastavit příznak                    |
| LDL | startDBX           |                                      |
| WR  | parametry~adrDB    | ;nastavení adresy do DataBoxu        |
|     |                    | ;na začátek archivu                  |
| LD  | __indx (receptura) | ;do kterého registru se přenáší      |
|     |                    | ;záznamy z DataBoxu                  |
| WR  | parametry~indR     |                                      |
| LD  | delka              | ;počet přenášených bytů záznamu      |
| WR  | parametry~len      |                                      |

E 63

# REJSTŘÍK PROBLÉMOVÝCH OKRUHŮ V PŘÍKLADECH

| Problém                                      | Příklad                                                                                                                                                                      |
|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| alternace bitu                               | 1.2.1                                                                                                                                                                        |
| archivace dat                                | 8.2.6, 8.2.7, 9.1.1, 9.1.2, 10.1.1, 10.1.2, 14.3.1., 14.3.2                                                                                                                  |
| aritmetické operace v pevné řádové čárce     | 4.5.1                                                                                                                                                                        |
| aritmetické operace v pohyblivé řádové čárce | 11.1.1 - 11.1.5, 11.2.1 - 11.2.7                                                                                                                                             |
| aritmetické operace, celá čísla se znaménkem | 4.4.1                                                                                                                                                                        |
| aritmetické operace, kladná celá čísla       | 4.1.1 - 4.1.9, 4.2.1 - 4.2.10, 4.3.1 - 4.3.12, 5.2.3                                                                                                                         |
| bezpečnostní dvoutlačítkové ovládání         | 3.3.3                                                                                                                                                                        |
| bezpečnostní vstupy                          | 2.4.3, 2.4.4                                                                                                                                                                 |
| blokové přenosy                              | 9.1.1, 9.1.2, 9.2.1, 9.2.2                                                                                                                                                   |
| cyklus                                       | 4.1.9                                                                                                                                                                        |
| časovače                                     | 3.3.1 - 3.3.10, 8.4.1                                                                                                                                                        |
| časová značka                                | 14.2.1                                                                                                                                                                       |
| časový procesor                              | 8.3.6 - 8.3.8, 8.3.13, 8.4.1, 10.2.1                                                                                                                                         |
| čítače                                       | 2.1.12, 2.4.1, 2.4.2, 3.1.1 - 3.1.5, 4.1.7, 4.1.8, 4.2.5, 4.2.6, 4.6.4, 8.1.3 - 8.1.5, 11.4.1, 11.4.2                                                                        |
| čtení dat                                    | 1.1.1                                                                                                                                                                        |
| dekódování klávesnice                        | 3.4.7, 8.3.19                                                                                                                                                                |
| filtrace krátkých impulzů                    | 3.3.1, 3.3.4, 3.3.10                                                                                                                                                         |
| generátor náhodných čísel                    | 2.5.7                                                                                                                                                                        |
| kaskádování časovačů                         | 3.3.8                                                                                                                                                                        |
| kaskádování čítačů                           | 3.1.4                                                                                                                                                                        |
| kód 1 z n                                    | 3.3.1 - 3.3.3, 8.2.4                                                                                                                                                         |
| kód 2 z 5                                    | 1.1.4, 8.1.2, 8.3.1                                                                                                                                                          |
| logické operace                              | 2.1.1 - 2.1.12, 2.2.1 - 2.2.9, 2.3.1 - 2.3.6, 2.4.5 - 2.4.8, 2.5.8, 2.5.9, 2.6.1 - 2.6.3, 4.2.9, 8.1.1, 8.2.1 - 8.2.3, 8.3.10, 8.3.12, 8.3.14, 8.3.15, 8.3.17, 8.3.18, 8.4.3 |
| matematické funkce                           | 11.3.2                                                                                                                                                                       |
| makroinstrukce                               | 5.2.2                                                                                                                                                                        |
| měření délky impulzu                         | 3.3.6, 3.3.7                                                                                                                                                                 |
| náběžná a sestupná hrana signálu             | 2.3.3 - 2.3.6, 3.1.1 - 3.1.5, 4.1.7, 4.1.8, 9.3.1, 9.3.2, 10.1.3, 11.4.1, 11.4.2                                                                                             |
| násobení a dělení                            | 4.2.1 - 4.2.10, 4.3.1, 4.5.1, 11.1.4, 11.1.5, 11.3.2                                                                                                                         |
| nepřímé čtení a zápis do prvků pole          | 5.2.2                                                                                                                                                                        |
| normalizace impulzů                          | 3.3.9                                                                                                                                                                        |
| omezení maximální délky impulzu              | 3.3.2                                                                                                                                                                        |
| plnění pole konstantou                       | 9.3.1, 9.3.2                                                                                                                                                                 |
| podmíněné skoky                              | 6.1.1 - 6.1.5, 7.1.1, 7.1.2, 8.3.2 - 8.3.5, 9.3.1, 9.3.2, 10.1.3                                                                                                             |
| podmíněný a vícenásobný konec procesu        | 7.1.1, 7.1.2                                                                                                                                                                 |
| podprogramy                                  | 6.2.1 - 6.2.3                                                                                                                                                                |
| podprogramy s nepřímým voláním               | 8.1.3 - 8.1.5, 8.3.4, 8.3.5                                                                                                                                                  |
| porovnání operandů                           | 2.1.3, 2.1.4, 2.2.4, 4.2.7, 4.3.1 - 4.3.12, 6.1.1 - 6.1.3, 6.2.1, 6.2.3, 7.1.1, 7.1.2, 8.3.2 - 8.3.5, 11.2.1 - 11.2.7                                                        |



| Problém                                         | Příklad                                                                        |
|-------------------------------------------------|--------------------------------------------------------------------------------|
| posuvné registry                                | 2.5.1 - 2.5.7, 3.2.1, 8.3.9                                                    |
| přepočet časových jednotek                      | 4.2.4, 4.2.8, 4.2.10                                                           |
| převod do ASCII kódu                            | 4.6.2, 4.6.3, 13.1.1, 13.1.2, 13.1.4                                           |
| převod do BCD kódu                              | 4.6.2, 4.6.3, 13.1.1, 13.1.2                                                   |
| převod do formátu float                         | 11.4.1, 11.4.2, 13.1.5                                                         |
| převod z ASCII kódu                             | 13.1.3, 13.1.5                                                                 |
| převod z BCD kódu                               | 4.6.1, 13.1.3                                                                  |
| převod z formátu float                          | 11.4.3, 13.1.4                                                                 |
| rotace                                          | 2.5.2 - 2.5.6                                                                  |
| sčítání a odčítání                              | 4.1.1 - 4.1.9, 4.5.1, 11.1.1 - 11.1.3, 11.1.5, 11.3.2                          |
| sčítání čistého času                            | 3.3.5, 3.3.8                                                                   |
| sekvenční automat                               | 8.4.2                                                                          |
| sekvenční řadič                                 | 3.4.1 - 3.4.6, 6.1.5, 6.2.2, 7.2.1, 8.1.6, 8.2.4, 8.2.5, 8.3.9, 8.3.11, 8.3.16 |
| shodný algoritmus pro více objektů              | 8.2.1 - 8.2.3, 9.2.2                                                           |
| snímač otáčení                                  | 8.1.3, 8.1.4                                                                   |
| struktury                                       | 10.1.1 - 10.1.3, 10.2.1                                                        |
| synchronní sériový přenos                       | 3.2.1                                                                          |
| tabulky                                         | 8.1.1 - 8.1.6, 8.2.1 - 8.2.7, 8.3.1 - 8.3.19, 8.4.1 - 8.4.3                    |
| třídění položek                                 | 8.4.1                                                                          |
| vícefázový časový průběh                        | 2.5.3 - 2.5.6, 8.1.5                                                           |
| výběr zobrazovaných textů                       | 9.2.1, 10.2.1                                                                  |
| výměna dat s nadřazeným systémem                | 14.1.1                                                                         |
| zámkový kontakt                                 | 2.3.1, 2.3.2                                                                   |
| zaokrouhlování                                  | 4.2.10, 11.3.1, 11.4.3                                                         |
| zápis dat                                       | 1.1.2                                                                          |
| zápis dat v časovém rastru                      | 1.1.3                                                                          |
| změna času                                      | 14.2.2                                                                         |
| zrcadlení pořadí bytů (převod Intel - Motorola) | 2.5.11, 2.5.12                                                                 |

REJSTŘÍK DIREKTIV PŘEKLADAČE

| Direktiva | Příklad                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| #def      | 1.2.1, 3.4.5, 4.4.1, 5.2.2, 6.1.5, 7.2.1, 8.1.3, 8.3.9, 9.1.1, 9.1.2, 9.3.1, 9.3.2, 10.1.1 - 10.1.3, 10.2.1, 13.1.4, 13.1.5, 14.1.1, 14.2.1, 14.2.2, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| #endm     | 5.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| #label    | 6.1.5, 6.2.2, 8.1.3, 8.3.3 - 8.3.7, 8.3.9, 8.3.13, 8.4.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| #macro    | 5.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| #reg      | 1.1.1, 1.1.3, 1.1.4, 1.2.1, 2.1.1 - 2.1.4, 2.1.7 - 2.1.12, 2.2.1 - 2.2.9, 2.3.1 - 2.3.6, 2.4.1 - 2.4.8, 2.5.1 - 2.5.3, 2.5.5, 2.5.7 - 2.5.12, 2.6.1 - 2.6.3, 3.1.1 - 3.1.5, 3.2.1, 3.3.1 - 3.3.10, 3.4.1, 3.4.3 - 3.4.7, 4.1.1 - 4.1.9, 4.2.1 - 4.2.10, 4.3.1 - 4.3.12, 4.4.1, 4.5.1, 4.6.1 - 4.6.4, 5.1.1, 5.2.2, 5.2.3, 6.1.1 - 6.1.5, 6.2.1 - 6.2.3, 7.1.1, 7.1.2, 8.1.1 - 8.1.3, 8.1.6, 8.2.1 - 8.2.4, 8.2.6, 8.2.7, 8.3.1 - 8.3.7, 8.3.9 - 8.3.19, 8.4.1, 8.4.2, 9.1.1, 9.1.2, 9.2.1, 9.2.2, 9.3.1, 9.3.2, 10.1.1 - 10.1.3, 10.2.1, 11.1.1 - 11.1.5, 11.2.1 - 11.2.7, 11.3.1, 11.3.2, 11.4.1 - 11.4.3, 13.1.1 - 13.1.5, 14.1.1, 14.2.1, 14.2.2, 14.3.1, 14.3.2 |
| #rem      | 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| #struct   | 10.2.1, 14.2.1, 14.2.2, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| #table    | 1.1.4, 6.1.4, 8.1.1 - 8.1.3, 8.1.6, 8.2.2, 8.2.3, 8.3.1 - 8.3.7, 8.3.9 - 8.3.19, 8.4.1 - 8.4.3, 9.2.1, 9.2.2, 10.1.1, 10.1.2, 10.2.1, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| #usi      | 4.4.1, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| bit       | 1.1.1, 2.1.1, 2.1.4 - 2.1.6, 2.1.12, 2.2.1, 2.2.8, 2.2.9, 2.3.1, 2.3.3 - 2.3.6, 2.4.3 - 2.4.8, 2.5.5, 2.5.7, 2.5.8, 2.6.1 - 2.6.3, 3.1.1 - 3.1.5, 3.2.1, 3.3.1 - 3.3.10, 3.4.3, 3.4.5, 3.4.6, 4.2.5 - 4.2.7, 4.3.1 - 4.3.10, 4.6.2, 6.2.3, 8.1.1, 8.1.3, 8.1.6, 8.2.1 - 8.2.3, 8.3.6, 8.3.10, 8.3.14, 8.3.17, 8.4.1, 9.1.2, 9.3.1, 9.3.2, 10.1.1 - 10.1.3, 11.2.1, 11.2.2, 11.2.4 - 11.2.6, 11.4.1, 11.4.2, 13.1.1, 14.3.1, 14.3.2                                                                                                                                                                                                                                  |
| byte      | 1.1.1, 1.1.3, 1.1.4, 1.2.1, 2.1.2 - 2.1.11, 2.2.2, 2.2.5, 2.2.6, 2.2.8, 2.3.2, 2.3.6, 2.4.1 - 2.4.4, 2.5.2, 2.5.3, 2.5.5, 2.5.8 - 2.5.10, 2.6.1 - 2.6.3, 3.2.1, 3.4.1, 3.4.3 - 3.4.7, 4.1.1 - 4.1.3, 4.1.5, 4.1.6, 4.1.9, 4.2.1 - 4.2.3, 4.2.5 - 4.2.9, 4.3.3, 4.3.7 - 4.3.9, 4.3.11, 4.3.12, 4.6.1 - 4.6.3, 6.1.1 - 6.1.5, 6.2.1 - 6.2.3, 7.1.1, 7.1.2, 8.1.1 - 8.1.3, 8.1.6, 8.2.1 - 8.2.4, 8.2.6, 8.2.7, 8.3.1 - 8.3.7, 8.3.9 - 8.3.19, 8.4.2, 8.4.3, 9.1.1, 9.1.2, 9.2.1, 9.2.2, 9.3.1, 9.3.2, 10.1.1 - 10.1.3, 10.2.1, 11.2.3, 13.1.2 - 13.1.5, 14.1.1, 14.2.1, 14.2.2, 14.3.1, 14.3.2                                                                         |
| float     | 1.1.1, 11.1.1 - 11.1.5, 11.2.1 - 11.2.7, 11.3.1, 11.3.2, 11.4.1 - 11.4.3, 13.1.4, 13.1.5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| long      | 1.1.1, 2.1.12, 2.2.4, 2.2.7, 2.4.1, 2.5.1, 2.5.12, 3.1.4, 3.3.8, 3.4.1, 4.1.1 - 4.1.4, 4.1.8, 4.2.1, 4.3.3, 4.4.1, 4.5.1, 5.2.3, 8.2.4, 8.3.19, 11.4.3, 13.1.1 - 13.1.3, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| word      | 1.1.1, 2.1.11, 2.2.3, 2.2.9, 2.3.3 - 2.3.6, 2.4.2, 2.4.5 - 2.4.8, 2.5.2, 2.5.7, 2.5.10, 2.5.11, 3.1.1 - 3.1.5, 3.2.1, 3.3.1 - 3.3.10, 3.4.1, 3.4.3 - 3.4.7, 4.1.1 - 4.1.4, 4.1.7, 4.1.8, 4.2.1 - 4.2.4, 4.2.6, 4.2.8 - 4.2.10, 4.3.1 - 4.3.6, 4.3.10, 4.3.11, 4.4.1, 4.5.1, 4.6.2 - 4.6.4, 5.1.1, 5.2.2, 6.1.4, 8.1.1, 8.1.3, 8.3.6, 8.3.9, 8.3.13, 8.3.18, 8.3.19, 8.4.1, 11.4.1, 11.4.2, 13.1.1 - 13.1.3, 14.2.1, 14.2.2, 14.3.1, 14.3.2                                                                                                                                                                                                                          |
| __indx    | 8.3.4, 8.3.7, 8.3.9, 8.3.13, 9.2.2, 10.1.1 - 10.1.3, 10.2.1, 13.1.4, 13.1.5, 14.2.1, 14.2.2, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| __sizeof  | 10.2.1, 14.2.1, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

## REJSTŘÍK SYSTÉMOVÝCH REGISTRŮ

| Registr | Příklad                                                                                                                                                                              |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| S0      | viz instrukce, které jej nastavují<br>2.5.7, 3.2.1, 3.3.8, 4.1.7, 4.1.8, 4.3.3, 4.3.4, 4.3.9, 4.3.12, 4.4.1, 8.1.6, 8.3.9,<br>8.3.11, 8.3.16, 9.2.2, 11.2.1 - 11.2.4, 11.2.6, 13.1.1 |
| S1      | viz instrukce, které jej nastavují<br>1.1.3, 2.4.1 - 2.4.7, 2.5.3, 2.5.5, 2.5.7, 3.4.7, 8.1.6, 8.3.1, 8.3.2, 8.3.6, 8.3.10,<br>8.3.11, 8.3.14, 8.3.16, 8.3.17, 8.3.18                |
| S6      | 4.2.8                                                                                                                                                                                |
| S7      | 4.2.4, 4.2.8, 8.3.6, 8.3.7, 8.3.13                                                                                                                                                   |
| S8      | 4.2.4, 8.3.6, 8.3.7, 8.3.13                                                                                                                                                          |
| S9      | 4.2.4, 8.3.7, 8.3.13                                                                                                                                                                 |
| S10     | 10.2.1                                                                                                                                                                               |
| S11     | 10.2.1                                                                                                                                                                               |
| S12     | 10.2.1, 14.2.2                                                                                                                                                                       |
| SW14    | 8.3.8                                                                                                                                                                                |
| SW16    | 4.3.10, 8.3.8                                                                                                                                                                        |
| SW18    | 4.2.10, 8.3.8                                                                                                                                                                        |
| S20     | 1.1.3, 2.5.3, 2.5.5, 4.1.7, 4.1.8, 4.2.5, 4.2.6                                                                                                                                      |
| S25     | 7.2.1                                                                                                                                                                                |

REJSTŘÍK INSTRUKCÍ PLC

| Instrukce | Příklad                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ABS       | 11.2.6                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| ADD       | 2.4.2, 4.1.4, 4.1.5, 4.1.7, 4.2.6, 4.2.10, 4.4.1, 6.1.5, 8.1.6, 8.3.9, 10.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| ADF       | 11.1.2, 11.1.3, 11.1.5, 11.3.1, 11.3.2, 11.4.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| ADL       | 4.1.3, 4.1.4, 4.4.1, 4.5.1, 5.2.3, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| ADX       | 2.4.1, 2.5.1, 4.1.3, 4.1.7, 4.1.8, 4.2.4, 4.2.5, 4.2.8, 4.2.10, 4.5.1, 8.3.11, 8.3.16, 14.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| ANC       | 2.1.6, 3.4.1, 8.3.11, 8.3.16                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| AND       | 2.1.2, 2.1.7, 2.1.10, 2.2.1 - 2.2.3, 2.4.1 - 2.4.5, 2.5.2, 2.5.3, 2.5.5, 2.5.7, 2.5.8, 2.6.1, 2.6.2, 3.3.2, 3.3.3, 3.3.8 - 3.3.10, 3.4.1, 3.4.4 - 3.4.6, 4.1.7, 4.2.2, 4.2.5 - 4.2.7, 4.2.10, 4.3.9, 4.3.11, 4.3.12, 4.6.2, 8.1.1 - 8.1.3, 8.1.6, 8.2.1, 8.3.6, 8.3.9 - 8.3.11, 8.3.16 - 8.3.18, 8.4.2, 9.1.1, 9.1.2, 9.2.2, 10.1.1 - 10.1.3, 10.2.1, 11.2.6, 13.1.1, 14.2.1                                                                                                                                                                                                                                                                                  |
| ANL       | 2.1.12, 2.2.7                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| ASB       | 13.1.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| BAS       | 4.6.2, 4.6.3, 13.1.1, 13.1.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| BCD       | 4.6.2 - 4.6.4, 13.1.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| BCL       | 13.1.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| BET       | 2.3.5, 14.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| BIL       | 13.1.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| BIN       | 4.6.1, 4.6.4                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| CAD       | 3.2.1, 6.2.1, 6.2.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| CAI       | 6.2.2, 8.1.3, 8.3.3, 8.3.5, 8.3.7, 8.3.9, 8.3.13, 8.4.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| CAL       | 5.2.1, 6.2.1, 9.2.2, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| CHGS      | 5.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| CMF       | 11.2.1 - 11.2.4, 11.2.6, 11.2.7                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| CML       | 4.3.3, 14.3.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| CMP       | 3.2.1, 4.1.9, 4.3.3, 6.1.1 - 6.1.3, 6.1.5, 6.2.2, 7.1.2, 8.3.2, 8.3.3, 8.3.11, 8.3.16, 9.2.2, 10.2.1, 13.1.2, 13.1.3, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| CNT       | 3.1.5, 8.1.3, 11.4.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| CTD       | 3.1.5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| CTU       | 3.1.1 - 3.1.5, 3.3.8, 11.4.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| DCR       | 3.2.1, 4.1.9, 4.2.4, 4.3.5, 8.1.3, 8.3.17, 8.3.18, 13.1.3, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| DID       | 4.2.10, 4.5.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| DIF       | 11.1.5, 11.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| DIV       | 3.4.1, 4.2.5 - 4.2.10, 10.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| E         | 1.1.1, 1.1.3, 1.1.4, 1.2.1, 2.1.1 - 2.1.12, 2.2.1 - 2.2.9, 2.3.1 - 2.3.6, 2.4.1 - 2.4.8, 2.5.1 - 2.5.3, 2.5.5, 2.5.7 - 2.5.12, 2.6.1 - 2.6.3, 3.1.1 - 3.1.5, 3.2.1, 3.3.1 - 3.3.10, 3.4.1, 3.4.3 - 3.4.7, 4.1.1 - 4.1.9, 4.2.1 - 4.2.10, 4.3.1 - 4.3.12, 4.4.1, 4.5.1, 4.6.1 - 4.6.4, 5.1.1, 5.2.1 - 5.2.3, 6.1.1 - 6.1.5, 6.2.1 - 6.2.3, 7.1.1, 7.1.2, 7.2.1, 8.1.1 - 8.1.3, 8.1.6, 8.2.1 - 8.2.4, 8.2.6, 8.2.7, 8.3.1 - 8.3.7, 8.3.9 - 8.3.19, 8.4.1 - 8.4.3, 9.1.1, 9.1.2, 9.2.1, 9.2.2, 9.3.1, 9.3.2, 10.1.1 - 10.1.3, 10.2.1, 11.1.1 - 11.1.5, 11.2.1 - 11.2.7, 11.3.1, 11.3.2, 11.4.1 - 11.4.3, 13.1.1 - 13.1.5, 14.1.1, 14.2.1, 14.2.2, 14.3.1, 14.3.2 |
| EC        | 7.1.1, 7.1.2, 14.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| ED        | 7.1.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| EQ        | 3.1.3, 4.3.1 - 4.3.3, 4.3.5, 4.3.8, 6.1.1, 6.2.1, 7.1.1, 9.1.2, 10.1.1, 10.1.2, 14.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

**Příklady programování PLC TECOMAT - model 16 bitů**

| <b>Instrukce</b> | <b>Příklad</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FIL              | 9.3.1, 9.3.2, 13.1.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| FIS              | 10.1.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| FIT              | 10.1.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| FLG              | 2.4.1 - 2.4.8, 2.5.7                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| FLO              | 11.3.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| FNT              | 10.2.1, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| FST              | 13.1.4                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| FTB              | 1.1.4, 3.4.7, 6.1.4, 8.3.1 - 8.3.7, 8.3.9 - 8.3.12, 8.3.17, 8.3.19                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| FTM              | 8.3.13 - 8.3.16, 8.3.18, 8.4.2, 8.4.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| FTS              | 8.4.1 - 8.4.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| GT               | 3.3.8, 4.2.7, 4.3.4, 4.3.7, 4.3.9 - 4.3.11, 6.1.2, 6.2.3, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| HPD              | 14.1.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| HPE              | 14.1.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| IDB              | 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| IFL              | 11.4.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| IMP              | 3.3.9, 3.3.10, 8.4.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| INR              | 2.1.12, 3.2.1, 4.1.6, 4.1.9, 6.1.1 - 6.1.4, 6.2.1, 6.2.3, 7.1.1, 7.1.2, 8.1.3, 8.2.3, 8.2.6, 8.2.7, 8.3.17 - 8.3.19, 9.1.1, 9.1.2, 9.2.2, 10.1.1 - 10.1.3, 10.2.1, 13.1.2, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| IWF              | 11.4.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| JC               | 3.2.1, 6.1.2, 6.1.3, 6.2.2, 7.1.2, 9.2.2, 11.2.7, 14.2.2, 14.3.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| JMC              | 2.1.12, 3.2.1, 3.3.6, 3.3.7, 3.3.10, 4.4.1, 6.1.1, 6.1.2, 6.2.1, 8.3.7, 8.3.9, 8.3.13, 8.3.19, 9.1.2, 9.3.1, 9.3.2, 10.1.1 - 10.1.3, 10.2.1, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| JMD              | 3.2.1, 3.4.7, 8.3.9, 8.3.19, 9.1.2, 10.1.1, 10.1.2, 10.2.1, 11.3.1, 11.4.3, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| JMI              | 6.1.5, 8.3.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| JMP              | 6.1.2, 6.1.3, 6.1.5, 8.3.3, 8.3.9, 8.3.17 - 8.3.19, 10.2.1, 11.3.1, 11.4.3, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| JNC              | 3.2.1, 6.1.5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| JNS              | 8.3.17, 8.3.18, 8.4.2, 8.4.3, 10.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| JNZ              | 4.1.9, 6.1.1, 10.2.1, 13.1.2, 13.1.3, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| JS               | 3.4.7, 6.1.4, 8.2.6, 8.2.7                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| JZ               | 6.1.3, 8.3.2, 8.3.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| L (návěští)      | 2.1.12, 3.2.1, 3.3.6, 3.3.7, 3.3.10, 3.4.7, 4.1.9, 4.4.1, 5.2.1, 6.1.1 - 6.1.5, 6.2.1 - 6.2.3, 7.1.2, 7.2.1, 8.1.3, 8.2.6, 8.2.7, 8.3.2 - 8.3.5, 8.3.7, 8.3.9, 8.3.13, 8.3.17 - 8.3.19, 8.4.1 - 8.4.3, 9.1.2, 9.2.2, 9.3.1, 9.3.2, 10.1.1 - 10.1.3, 10.2.1, 11.2.7, 11.3.1, 11.4.3, 13.1.2, 13.1.3, 14.2.2, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                         |
| LAC              | 5.2.2, 5.2.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| LD               | 1.1.1, 1.1.3, 1.1.4, 1.2.1, 2.1.1 - 2.1.12, 2.2.1 - 2.2.8, 2.3.1 - 2.3.3, 2.3.5, 2.3.6, 2.4.1 - 2.4.8, 2.5.1 - 2.5.3, 2.5.5, 2.5.7 - 2.5.12, 2.6.1 - 2.6.3, 3.1.1 - 3.1.5, 3.2.1, 3.3.1 - 3.3.10, 3.4.1, 3.4.3 - 3.4.7, 4.1.1 - 4.1.5, 4.1.7 - 4.1.9, 4.2.1 - 4.2.10, 4.3.1 - 4.3.12, 4.4.1, 4.5.1, 4.6.1 - 4.6.4, 5.2.2, 5.2.3, 6.1.1 - 6.1.5, 6.2.1 - 6.2.3, 7.1.1, 7.1.2, 7.2.1, 8.1.1 - 8.1.3, 8.1.6, 8.2.1 - 8.2.4, 8.2.6, 8.2.7, 8.3.1 - 8.3.7, 8.3.9 - 8.3.19, 8.4.1 - 8.4.3, 9.1.1, 9.1.2, 9.2.1, 9.2.2, 9.3.1, 9.3.2, 10.1.1 - 10.1.3, 10.2.1, 11.1.1 - 11.1.5, 11.2.1 - 11.2.7, 11.3.1, 11.3.2, 11.4.1 - 11.4.3, 13.1.1 - 13.1.5, 14.1.1, 14.2.1, 14.2.2, 14.3.1, 14.3.2 |
| LDC              | 1.1.1, 2.2.5, 2.2.9, 2.3.4, 2.4.2, 3.1.1, 3.2.1, 3.3.3, 3.3.6, 3.3.7, 3.3.10, 3.4.7, 4.3.12, 4.4.1, 8.1.6, 8.3.1, 8.3.2, 11.2.1, 11.2.4, 11.2.6                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| LDL              | 1.1.1, 11.2.7, 11.3.2, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| LDS              | 10.1.1, 10.1.2, 10.2.1, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| LET              | 2.3.3, 2.3.4, 2.3.6, 2.4.1, 2.4.2, 3.2.1, 3.3.6, 3.3.10, 3.4.3, 6.2.3, 9.3.1, 9.3.2, 10.1.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| LT               | 4.3.3, 4.3.4, 4.3.9, 4.3.10, 4.3.12                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

# Rejstříky

| Instrukce | Příklad                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LTB       | 3.2.1, 5.2.2, 8.1.1 - 8.1.3, 8.1.6, 8.2.1 - 8.2.3, 8.2.6, 8.2.7, 8.3.4, 8.3.5, 8.3.7, 8.3.9, 8.3.12, 8.3.13, 8.3.15, 8.3.17 - 8.3.19, 8.4.2, 8.4.3, 9.2.2, 13.1.2, 13.13, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| MNT       | 9.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| MOV       | 9.1.1, 9.1.2, 10.2.1, 14.1.1, 14.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| MTN       | 9.2.1, 9.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| MUD       | 4.2.1, 4.4.1, 4.5.1, 5.2.3, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| MUF       | 11.1.4, 11.1.5, 11.3.2, 11.4.1, 11.4.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| MUL       | 2.5.2, 2.5.3, 2.5.5, 2.5.8, 2.5.9, 4.2.1 - 4.2.4, 4.2.8, 8.1.3, 8.2.3, 8.3.7, 8.3.11, 8.3.13, 8.3.16, 8.4.2, 9.1.1, 9.1.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| NEG       | 2.2.5, 2.2.6, 2.2.8, 2.6.3, 3.3.2, 3.3.3, 3.3.9, 14.2.1, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| NGL       | 2.2.7                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| NXT       | 5.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| OR        | 2.1.1, 2.1.8, 2.1.10, 2.1.11, 2.2.1 - 2.2.3, 2.2.5, 2.4.6, 2.4.7, 2.5.3, 2.5.9, 2.6.2, 2.6.3, 3.1.5, 3.3.10, 3.4.5, 4.3.3, 4.3.4, 4.3.9, 4.3.12, 8.1.3, 8.3.7, 8.3.11, 8.3.13, 8.3.16, 8.3.17, 8.3.19, 8.4.2                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| ORC       | 2.1.5, 2.2.5, 3.3.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| ORL       | 2.2.7                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| P 0       | 1.1.1, 1.1.3, 1.1.4, 1.2.1, 2.1.1 - 2.1.12, 2.2.1 - 2.2.9, 2.3.1 - 2.3.6, 2.4.1 - 2.4.8, 2.5.1 - 2.5.3, 2.5.5, 2.5.7 - 2.5.12, 2.6.1 - 2.6.3, 3.1.1 - 3.1.5, 3.2.1, 3.3.1 - 3.3.10, 3.4.1, 3.4.3 - 3.4.7, 4.1.1 - 4.1.9, 4.2.1 - 4.2.10, 4.3.1 - 4.3.12, 4.4.1, 4.5.1, 4.6.1 - 4.6.4, 5.1.1, 5.2.1 - 5.2.3, 6.1.1 - 6.1.5, 6.2.1 - 6.2.3, 7.1.1, 7.1.2, 7.2.1, 8.1.1 - 8.1.3, 8.1.6, 8.2.1 - 8.2.4, 8.2.6, 8.2.7, 8.3.1 - 8.3.7, 8.3.9 - 8.3.19, 8.4.1 - 8.4.3, 9.1.1, 9.1.2, 9.2.1, 9.2.2, 9.3.1, 9.3.2, 10.1.1 - 10.1.3, 10.2.1, 11.1.1 - 11.1.5, 11.2.1 - 11.2.7, 11.3.1, 11.3.2, 11.4.1 - 11.4.3, 13.1.1 - 13.1.5, 14.1.1, 14.2.1, 14.2.2, 14.3.1, 14.3.2 |
| P 10      | 7.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| P 41      | 14.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| P 60      | 3.2.1, 5.2.1, 6.2.1 - 6.2.3, 8.1.3, 8.3.4, 8.3.5, 8.3.7, 8.3.9, 8.3.13, 8.4.1, 9.2.2, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| P 63      | 2.4.2, 2.5.3, 2.5.5, 2.5.7, 3.2.1, 3.4.6, 7.2.1, 8.3.9, 9.1.2, 10.1.1, 10.1.2, 14.1.1, 14.2.2, 14.3.1, 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| POP       | 2.4.2, 3.1.4, 3.2.1, 3.3.6 - 3.3.8, 4.2.10, 5.1.1, 5.2.2, 8.1.1, 8.1.6, 8.2.3, 8.2.6, 8.2.7, 8.3.7, 8.3.13, 8.3.17 - 8.3.19, 10.1.2, 11.2.6, 13.1.2, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| POW       | 11.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| PRV       | 5.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| PUT       | 1.1.3, 1.1.4, 2.5.3, 2.5.5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| RDB       | 14.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| RDT       | 14.2.1, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| REC       | 6.2.3, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| RES       | 2.3.1 - 2.3.6, 2.5.3, 3.2.1, 3.3.3, 3.3.8, 3.3.9, 3.4.1, 3.4.4 - 3.4.6, 7.2.1, 8.1.1 - 8.1.3, 8.1.6, 8.3.2, 8.3.6, 8.3.7, 8.3.13, 9.1.2, 10.1.1, 10.1.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| RET       | 3.2.1, 5.2.1, 6.2.1 - 6.2.3, 8.1.3, 8.3.4, 8.3.5, 8.3.7, 8.3.9, 8.3.13, 8.4.1, 9.2.2, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| ROL       | 2.5.2, 2.5.7, 3.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| ROR       | 3.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| RTO       | 3.3.5 - 3.3.8                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| SEQ       | 7.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| SET       | 2.3.1, 2.3.2, 2.5.2, 2.5.3, 2.5.5, 2.5.9, 2.6.1, 3.2.1, 3.3.3, 3.3.8, 3.4.4 - 3.4.6, 7.2.1, 8.1.1 - 8.1.3, 8.3.1, 8.3.6, 8.3.19, 9.1.2, 10.1.1, 10.1.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| SFL       | 3.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

| Instrukce | Příklad                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SFR       | 3.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| SIN       | 11.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| SRC       | 9.1.1, 9.1.2, 10.2.1, 14.1.1, 14.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| STE       | 3.4.1, 3.4.3 - 3.4.7                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| STF       | 13.1.5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| STK       | 2.4.3, 2.6.1 - 2.6.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| SUB       | 2.4.2, 4.1.8, 4.3.1, 4.3.8, 4.3.9                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| SUF       | 11.1.1, 11.1.5, 11.2.6, 11.3.1, 11.4.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| SUL       | 4.1.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| SUX       | 4.1.1, 4.1.2, 4.3.10                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| SWL       | 2.5.12, 4.6.3, 5.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| SWP       | 2.5.2, 2.5.3, 2.5.5, 2.5.8, 2.5.9 - 2.5.12, 3.2.1, 3.4.1, 4.2.5, 4.2.8, 4.6.2, 4.6.3, 8.3.7, 8.3.13, 10.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| TOF       | 3.3.4                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| TON       | 3.3.1 - 3.3.3, 3.4.5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| UFL       | 11.4.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| UWF       | 11.4.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| USI       | 4.4.1, 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| WAC       | 5.2.2, 5.2.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| WDB       | 14.3.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| WR        | 1.1.2, 1.1.3, 2.1.1 - 2.1.5, 2.1.7 - 2.1.12, 2.2.1 - 2.2.8, 2.3.6, 2.4.1 - 2.4.8, 2.5.1 - 2.5.3, 2.5.5, 2.5.7 - 2.5.12, 2.6.2, 3.1.3, 3.1.5, 3.2.1, 3.3.1, 3.3.2, 3.3.4 - 3.3.10, 3.4.1, 3.4.3 - 3.4.7, 4.1.1 - 4.1.5, 4.1.7 - 4.1.9, 4.2.1 - 4.2.10, 4.3.1 - 4.3.5, 4.3.7 - 4.3.12, 4.4.1, 4.5.1, 4.6.1 - 4.6.4, 5.1.1, 5.2.3, 6.1.2 - 6.1.4, 7.1.2, 7.2.1, 8.1.1, 8.1.3, 8.1.6, 8.2.2 - 8.2.4, 8.2.6, 8.2.7, 8.3.1, 8.3.2, 8.3.5, 8.3.7, 8.3.9 - 8.3.19, 8.4.2, 8.4.3, 9.1.1, 9.1.2, 9.2.2, 10.1.1 - 10.1.3, 10.2.1, 11.1.1 - 11.1.5, 11.2.1 - 11.2.4, 11.2.6, 11.2.7, 11.3.1, 11.3.2, 11.4.1 - 11.4.3, 13.1.1 - 13.1.3, 13.1.5, 14.2.1, 14.2.2, 14.3.1, 14.3.2 |
| WRA       | 1.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| WRC       | 1.1.2, 2.1.6, 2.2.6, 2.2.8, 2.2.9, 2.5.2, 2.6.3, 3.1.5, 3.2.1, 4.2.5, 4.2.6, 4.3.1, 4.3.4 - 4.3.6, 4.3.8, 4.3.10, 8.1.6, 8.3.9, 11.2.5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| WRS       | 10.1.1, 10.1.2, 10.2.1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| WRT       | 14.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| WTB       | 3.2.1, 5.2.2, 8.2.1 - 8.2.4, 8.2.6, 8.2.7, 8.3.17 - 8.3.19, 9.2.2, 13.1.2, 13.1.3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| XOL       | 2.2.4                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| XOR       | 2.1.3, 2.1.4, 2.1.9 - 2.1.11, 2.2.5, 2.2.8, 2.3.6, 2.4.4, 2.4.7, 2.4.8, 2.5.7, 3.3.10, 3.4.6, 4.4.1, 4.3.1, 4.3.8, 8.1.6, 8.2.1, 8.3.7, 8.3.9, 8.3.13                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

SEZNAM PŘÍKLADŮ

| Příklad | str. | Příklad | str. | Příklad | str. | Příklad | str. | Příklad | str. |
|---------|------|---------|------|---------|------|---------|------|---------|------|
| 1.1.1   | 5    | 2.5.2   | 21   | 4.1.5   | 57   | 6.1.5   | 83   | 8.4.3   | 134  |
| 1.1.2   | 6    | 2.5.3   | 22   | 4.1.6   | 57   | 6.2.1   | 85   | 9.1.1   | 136  |
| 1.1.3   | 6    | 2.5.4   | 23   | 4.1.7   | 57   | 6.2.2   | 86   | 9.1.2   | 136  |
| 1.1.4   | 6    | 2.5.5   | 23   | 4.1.8   | 58   | 6.2.3   | 87   | 9.2.1   | 138  |
| 1.2.1   | 7    | 2.5.6   | 24   | 4.1.9   | 58   | 7.1.1   | 88   | 9.2.2   | 138  |
| 2.1.1   | 8    | 2.5.7   | 24   | 4.2.1   | 59   | 7.1.2   | 88   | 9.3.1   | 141  |
| 2.1.2   | 8    | 2.5.8   | 26   | 4.2.2   | 60   | 7.2.1   | 89   | 9.3.2   | 141  |
| 2.1.3   | 8    | 2.5.9   | 26   | 4.2.3   | 60   | 8.1.1   | 91   | 10.1.1  | 142  |
| 2.1.4   | 8    | 2.5.10  | 27   | 4.2.4   | 61   | 8.1.2   | 93   | 10.1.2  | 143  |
| 2.1.5   | 9    | 2.5.11  | 27   | 4.2.5   | 61   | 8.1.3   | 94   | 10.1.3  | 144  |
| 2.1.6   | 9    | 2.5.12  | 27   | 4.2.6   | 62   | 8.1.4   | 98   | 10.2.1  | 145  |
| 2.1.7   | 9    | 2.6.1   | 28   | 4.2.7   | 63   | 8.1.5   | 98   | 11.1.1  | 147  |
| 2.1.8   | 9    | 2.6.2   | 28   | 4.2.8   | 63   | 8.1.6   | 100  | 11.1.2  | 147  |
| 2.1.9   | 10   | 2.6.3   | 29   | 4.2.9   | 64   | 8.2.1   | 106  | 11.1.3  | 147  |
| 2.1.10  | 10   | 3.1.1   | 31   | 4.2.10  | 64   | 8.2.2   | 106  | 11.1.4  | 148  |
| 2.1.11  | 10   | 3.1.2   | 31   | 4.3.1   | 65   | 8.2.3   | 107  | 11.1.5  | 148  |
| 2.1.12  | 11   | 3.1.3   | 31   | 4.3.2   | 65   | 8.2.4   | 108  | 11.2.1  | 148  |
| 2.2.1   | 11   | 3.1.4   | 32   | 4.3.3   | 65   | 8.2.5   | 108  | 11.2.2  | 149  |
| 2.2.2   | 11   | 3.1.5   | 32   | 4.3.4   | 66   | 8.2.6   | 108  | 11.2.3  | 149  |
| 2.2.3   | 12   | 3.2.1   | 33   | 4.3.5   | 67   | 8.2.7   | 109  | 11.2.4  | 149  |
| 2.2.4   | 12   | 3.3.1   | 36   | 4.3.6   | 68   | 8.3.1   | 110  | 11.2.5  | 149  |
| 2.2.5   | 12   | 3.3.2   | 36   | 4.3.7   | 68   | 8.3.2   | 110  | 11.2.6  | 150  |
| 2.2.6   | 13   | 3.3.3   | 36   | 4.3.8   | 68   | 8.3.3   | 111  | 11.2.7  | 150  |
| 2.2.7   | 13   | 3.3.4   | 37   | 4.3.9   | 69   | 8.3.4   | 112  | 11.3.1  | 151  |
| 2.2.8   | 13   | 3.3.5   | 37   | 4.3.10  | 70   | 8.3.5   | 113  | 11.3.2  | 151  |
| 2.2.9   | 14   | 3.3.6   | 38   | 4.3.11  | 70   | 8.3.6   | 115  | 11.4.1  | 152  |
| 2.3.1   | 14   | 3.3.7   | 38   | 4.3.12  | 71   | 8.3.7   | 116  | 11.4.2  | 153  |
| 2.3.2   | 15   | 3.3.8   | 39   | 4.4.1   | 73   | 8.3.8   | 118  | 11.4.3  | 153  |
| 2.3.3   | 15   | 3.3.9   | 39   | 4.5.1   | 75   | 8.3.9   | 118  | 13.1.1  | 156  |
| 2.3.4   | 15   | 3.3.10  | 40   | 4.6.1   | 76   | 8.3.10  | 121  | 13.1.2  | 156  |
| 2.3.5   | 16   | 3.4.1   | 42   | 4.6.2   | 76   | 8.3.11  | 122  | 13.1.3  | 157  |
| 2.3.6   | 16   | 3.4.2   | 43   | 4.6.3   | 77   | 8.3.12  | 123  | 13.1.4  | 158  |
| 2.4.1   | 16   | 3.4.3   | 43   | 4.6.4   | 77   | 8.3.13  | 123  | 13.1.5  | 158  |
| 2.4.2   | 17   | 3.4.4   | 43   | 5.1.1   | 78   | 8.3.14  | 126  | 14.1.1  | 159  |
| 2.4.3   | 18   | 3.4.5   | 46   | 5.2.1   | 78   | 8.3.15  | 126  | 14.2.1  | 161  |
| 2.4.4   | 18   | 3.4.6   | 50   | 5.2.2   | 79   | 8.3.16  | 127  | 14.2.2  | 162  |
| 2.4.5   | 19   | 3.4.7   | 51   | 5.2.3   | 79   | 8.3.17  | 128  | 14.3.1  | 165  |
| 2.4.6   | 19   | 4.1.1   | 55   | 6.1.1   | 81   | 8.3.18  | 128  | 14.3.2  | 166  |
| 2.4.7   | 20   | 4.1.2   | 55   | 6.1.2   | 82   | 8.3.19  | 129  |         |      |
| 2.4.8   | 20   | 4.1.3   | 56   | 6.1.3   | 82   | 8.4.1   | 131  |         |      |
| 2.5.1   | 20   | 4.1.4   | 56   | 6.1.4   | 83   | 8.4.2   | 132  |         |      |





teco

Objednávky a informace:

Teco a. s. Havlíčkova 260, 280 58 Kolín 4, tel./fax: 321 725 748

TXV 001 07.01