

tecomat®

PROGRAMOVATELNÉ AUTOMATY

#program
#unit #reg
byte float
high #table
indx

PŘÍRUČKA PROGRAMÁTORA PLC TECOMAT

PŘÍRUČKA PROGRAMÁTORA PLC TECOMAT

10. vydání - leden 2005

OBSAH

1. ÚVOD	5
2. PLC A UŽIVATELSKÝ PROGRAM.....	7
2.1. Zapínací sekvence.....	8
2.2. Pracovní režimy PLC	8
2.3. Restarty uživatelského programu	10
3. STRUKTURA UŽIVATELSKÉHO PROGRAMU	12
4. STRUKTURA INSTRUKCÍ A OPERANDŮ	16
4.1. Bezprostřední operand	17
4.1.1. Číselné soustavy	17
4.1.2. Typy bezprostředních dat	18
4.2. Adresový operand.....	20
4.2.1. Operand typu bool.....	22
4.2.2. Operand typů byte, usint, sint.....	23
4.2.3. Operand typů word, uint, int	24
4.2.4. Operand typů dword, udint, dint	25
4.2.5. Operand typu real.....	27
4.2.6. Operand typu lreal.....	29
4.3. Cíl přechodu	30
4.4. Parametr instrukce.....	31
5. STRUKTURA ZÁPISNÍKOVÉ PAMĚTI.....	32
5.1. Obrazy vstupů X	33
5.2. Obrazy výstupů Y	33
5.3. Systémové registry S.....	33
5.4. Uživatelské registry R	42
6. PŘÍMÉ PŘÍSTUPY KE VSTUPŮM A VÝSTUPŮM.....	43
6.1. Přímé přístupy ke vstupům a výstupům - model 16 bitů	43
6.1.1. Fyzické adresy v PLC TECOMAT NS950	44
6.1.2. Fyzické adresy v PLC TECOMAT TC400, TC500, TC600	45
6.2. Přímé přístupy ke vstupům a výstupům - model 32 bitů	45
7. OSTATNÍ ADRESOVÉ PROSTORY	46
7.1. Data D	46
7.2. Tabulky T	46
7.3. Přídavná paměť dat DataBox	48
8. ZÁSOBNÍK VÝSLEDKŮ.....	52
8.1. Struktura zásobníku	53
8.2. Interpretace dat na zásobníku - model 16 bitů.....	54
8.2.1. Data typu bool - model 16 bitů.....	54

8.2.2. Data typů byte, usint, sint - model 16 bitů.....	55
8.2.3. Data typů word, uint, int - model 16 bitů	55
8.2.4. Data typů dword, udint, dint - model 16 bitů	55
8.2.5. Data typu real - model 16 bitů	56
8.3. Interpretace dat na zásobníku - model 32 bitů.....	56
8.3.1. Data typu bool - model 32 bitů.....	56
8.3.2. Data typů byte, usint, sint - model 32 bitů.....	57
8.3.3. Data typů word, uint, int - model 32 bitů	57
8.3.4. Data typů dword, udint, dint - model 32 bitů	58
8.3.5. Data typu real - model 32 bitů	58
8.3.6. Data typu lreal - model 32 bitů.....	59
8.4. Přepínání zásobníků.....	59
9. DIREKTIVY PŘEKLADAČE	61
9.1. #program	61
9.2. #unit, #module	62
9.3. #include, #usefile	63
9.4. #def.....	63
9.5. #reg, #rem	64
9.6. #struct.....	66
9.7. #data, #table.....	70
9.8. #if, #elif, #else, #endif	71
9.9. #ifdef, #ifndef, #else, #endif.....	72
9.10. #usi	72
9.11. #label.....	74
9.12. #macro, #endm.....	74
9.13. #mnemo, #mnemoend.....	76
9.14. #useoption	76
10. UŽIVATELSKÉ PROCESY	78
10.1. Všeobecné zásady aktivace	78
10.2. Otočka cyklu	79
10.3. Ošetření restartu - procesy P62, P63	81
10.4. Procesy smyčky.....	82
10.4.1. Základní proces P0.....	82
10.4.2. Čtyřfázově aktivované procesy P1, P2, P3, P4	83
10.4.3. Časově aktivované procesy P5, P6, P7, P8, P9.....	84
10.4.4. Uživatelsky aktivované procesy P10 až P40	85
10.4.5. Závěrečný proces cyklu P64	86
10.5. Přerušující procesy	86
10.5.1. Přerušení od času P41	87
10.5.2. Přerušení od vstupů P42	88
10.5.3. Přerušení od chyby P43	90
10.5.4. Přerušení od hw čítačů nebo od inkrementálního snímače P44.....	91
10.5.5. Přerušení od sériového kanálu CH2 P45	91
10.6. Ošetření ladícího bodu - procesy P50 až P57	91
10.7. Balík podprogramů P60	92
11. SOUBOR INSTRUKCÍ.....	93
12. UŽIVATELSKÉ INSTRUKCE	95
12.1. Použití USI v uživatelském programu	95
12.2. USI pro jednotlivé řady centrálních jednotek	95

12.3. Vytvoření vlastní USI	96
12.4. Používané překladače jazyka C	97
12.5. Příklad vytvoření vlastní instrukce USI	98
12.6. Příklad použití instrukce USI.....	99
12.7. Poznámky na závěr	99
A. PŘÍLOHA	100
A.1. Doby výkonu instrukcí v centrální jednotce CPM-1E TECOMAT NS950	100
A.2. Doby výkonu instrukcí v centrální jednotce CPM-1M TECOMAT NS950.....	102
A.3. Doby výkonu instrukcí v centrální jednotce CPM-2S TECOMAT NS950	105
A.4. Doby výkonu instrukcí v centrálních jednotkách CPM-1D TECOMAT NS950 a TECOMAT TC400, TC500, TC600.....	108
A.5. Doby výkonu instrukcí v centrálních jednotkách CPM-1B, CPM-2B TECOMAT NS950.....	112
A.6. Doby výkonu instrukcí v centrálních jednotkách CP-7001, CP-7002 TECOMAT TC700 a TECOMAT TC650.....	116

1. ÚVOD

Tato příručka má za cíl seznámit uživatele s programovatelnými automaty (dále PLC) TECOMAT a usnadnit jejich programování. V dalším textu jsou rozlišeny centrální jednotky podle řad (viz dále), nikoli podle typů PLC.

Řady centrálních jednotek

Každý typ PLC TECOMAT má k dispozici několik centrálních jednotek (zkratka CPU - Central Processor Unit) odlišených tak zvanou řadou, kterou představuje jedno písmeno. Každá řada centrálních jednotek má danou velikost paměťových prostorů, rozsah instrukčního souboru a operandů, strukturu zásobníku a je určujícím parametrem pro překladač uživatelského programu. Centrální jednotky PLC TECOMAT jsou rozděleny podle svých vlastností do následujících řad:

řada B	- NS950 CPM-1B, CPM-2B
řada C	- TC650, TC700 CP-7001, CP-7002
řada D	- TC400, TC500, TC600, NS950 CPM-1D
řada E	- NS950 CPM-1E
řada M	- NS950 CPM-1M
řada S	- NS950 CPM-1S, CPM-2S

Členění příručky

- ♦ 2. kapitola uvádí obecné zákonitosti zpracování uživatelského programu v PLC
- ♦ 3. kapitola popisuje základní strukturu uživatelského programu ve vývojovém prostředí Mosaic
- ♦ 4. kapitola popisuje strukturu instrukcí a jejich operandů
- ♦ 5. kapitola popisuje strukturu zápisníkové paměti včetně podrobného přehledu systémových služeb obsažených v systémových registrech S
- ♦ 6. kapitola uvádí obecné zákonitosti fyzického adresování jednotek PLC
- ♦ 7. kapitola popisuje ostatní adresové prostory pro data - data D, tabulky T a přídatnou paměť DataBox
- ♦ 8. kapitola popisuje strukturu a chování zásobníku výsledků
- ♦ 9. kapitola obsahuje přehled direktiv použitelných ve vývojovém prostředí Mosaic včetně příkladů jejich použití
- ♦ 10. kapitola se zabývá uživatelskými procesy
- ♦ 11. kapitola obsahuje přehled instrukcí a přípustných operandů
- ♦ 12. kapitola se zabývá uživatelskými instrukcemi

Příklady v této příručce jsou z důvodu úspory místa uváděny pouze v mnemokódu. K zobrazení příkladu v reléovém liniovém schématu použijte prostředí Mosaic.

Návazné příručky

Podrobné informace o jednotlivých instrukcích jsou obsaženy v příručkách Soubor instrukcí PLC TECOMAT model 16 bitů (TXV 001 05.01 - pro CPU řady B, D, E, M, S) a Soubor instrukcí PLC TECOMAT model 32 bitů (TXV 004 01.01 - pro CPU řady C).

Příklady řešení různých dílčích problémů jsou obsaženy v příručce Příklady programování PLC TECOMAT model 16 bitů (TXV 001 07.01 - pro CPU řady B, D, E, M, S) a Příklady programování PLC TECOMAT model 32 bitů (TXV 004 04.01 - pro CPU řady C).

Programování PLC

Programování řídicích algoritmů a testování správnosti napsaných programů pro PLC TECOMAT se provádí na počítačích standardu PC. Pro spojení s PLC se využívá běžný sériový kanál těchto počítačů. Některé typy centrálních jednotek jsou navíc vybaveny rozhraním Ethernet a USB.

Ke každému PLC je dodáván CD-ROM s příklady a vývojovým prostředím Mosaic pro Windows ve verzi Mosaic Lite.

Příklady programů PLC obsahují návody k obsluze různých jednotek PLC a dále jsou zde příklady z příruček TXV 001 07.01 a TXV 004 04.01.

Vývojové prostředí Mosaic

Vývojové prostředí Mosaic je komplexním vývojovým nástrojem pro programování aplikací PLC TECOMAT a regulátorů TECOREG, který umožňuje pohodlnou tvorbu a odladění programu. Jedná se o produkt na platformě Windows 2000 / XP, který využívá řadu moderních technologií. Modulární struktura prostředí Mosaic umožňuje uživateli, aby si z nabízených nástrojů poskládal prostředí podle toho, které části bude potřebovat. Dostupné jsou následující verze:

- | | |
|----------------|---|
| Mosaic Lite | neklíčovaná verze prostředí s možností naprogramovat PLC se dvěma periferními jednotkami |
| Mosaic Compact | umožní bez omezení programovat kompaktní PLC TECOMAT řad TC400, TC500, TC600 a regulátory TECOREG |
| Mosaic Profi | je určena pro všechny systémy firmy Teco bez omezení |

Základní prostředí obsahuje součásti, bez kterých se uživatel při tvorbě programu neobejde nebo které v převážné většině případů využije: textový editor, překladač mnemokódu xPRO, debugger, modul pro komunikaci s PLC, simulátor PLC, konfigurační modul PLC a systém nápovědy. Součástí základního prostředí je simulátor operačních panelů ID-07 / ID-08 a vestavěného panelu TC500.

Rozšíření prostředí se provádí pomocí pluginů - modulů, které jsou spustitelné ve spojení se základním prostředím. Takto lze Mosaic rozšířit o další možnosti programování - strukturovaný text podle normy IEC 61131 (MosaicST plugin), jazyk reléových schémat (MosaicLD plugin - připravuje se) nebo funkční bloky (MosaicFBD plugin - připravuje se) a další podpůrné nástroje pro návrh obrazovek operátorských panelů (PanelMaker), nástroj pro práci s PID regulátory (PIDMaker), grafickou on-line analýzu sledovaných proměnných či off-line analýzu archivovaných dat (GraphMaker).

2. PLC A UŽIVATELSKÝ PROGRAM

Co je to programovatelný automat

Programovatelný automat (dále jen PLC - Programmable Logic Controller) je číslicový řídicí elektronický systém určený pro řízení procesů v průmyslovém prostředí. Používá programovatelnou paměť pro vnitřní ukládání uživatelsky orientovaných instrukcí, jež slouží k realizaci specifických funkcí pro řízení různých druhů strojů nebo procesů prostřednictvím číslicových nebo analogových vstupů a výstupů.

Firma Teco a. s. vyrábí PLC systémy pod ochrannou známkou TECOMAT.

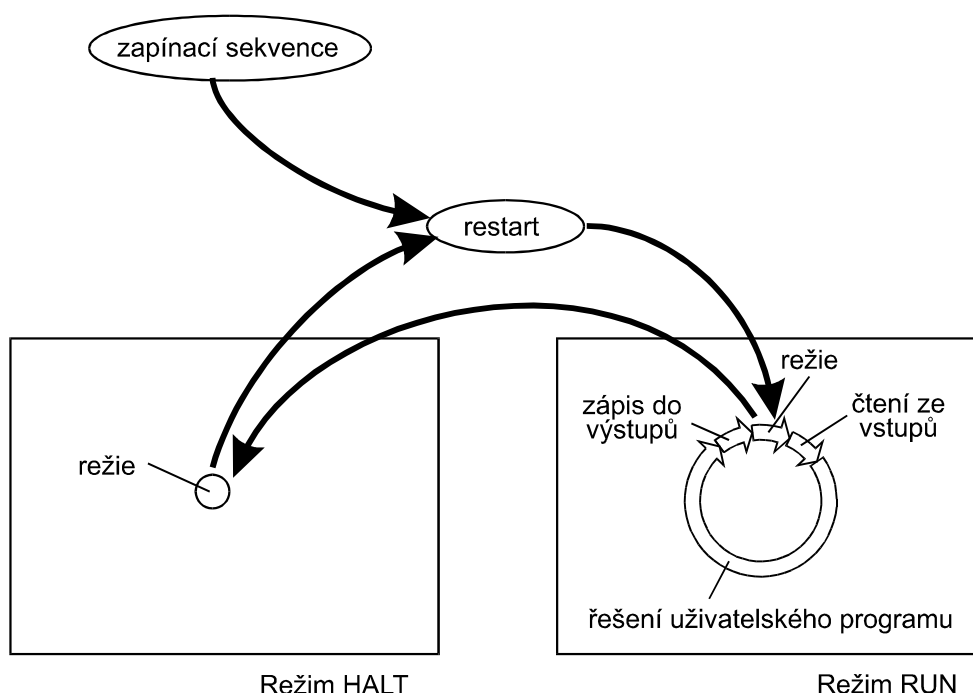
Princip vykonávání uživatelského programu

Řídicí algoritmus programovatelného automatu je zapsán jako posloupnost instrukcí v paměti uživatelského programu. Centrální jednotka postupně čte z této paměti jednotlivé instrukce, provádí příslušné operace s daty v zápisníkové paměti a zásobníku, případně provádí přechody v posloupnosti instrukcí, je-li instrukce ze skupiny organizačních instrukcí. Jsou-li provedeny všechny instrukce požadovaného algoritmu, provádí centrální jednotka aktualizaci výstupních proměnných do výstupních periferních jednotek a aktualizuje stavy ze vstupních periferních jednotek do zápisníkové paměti. Tento děj se stále opakuje a nazýváme jej cyklem programu (obr.2.1).

Cyklické vykonávání uživatelského programu

Jednorázová aktualizace stavů vstupních proměnných během celého cyklu programu odstraňuje možnosti vzniku hazardních stavů při řešení algoritmu řízení (během výpočtu nemůže dojít ke změně vstupních proměnných).

Před zahájením psaní vlastního uživatelského programu pro PLC je třeba si tuto skutečnost uvědomit. V některých případech to usnadňuje řešení problému, v jiných zase komplikuje.



Obr.2.1 Řešení uživatelského programu v PLC

Na obr.2.1 je uvedeno zjednodušené schéma řešení uživatelského programu v PLC s následujícími pojmy:

- ♦ zapínací sekvence je činnost PLC po zapnutí napájení (viz kap.2.1.)
- ♦ restart je činnost PLC bezprostředně před zahájením vykonávání uživatelského programu (viz kap.2.3.)
- ♦ režim RUN a režim HALT představují pracovní režimy PLC (viz kap.2.2.)
- ♦ čtení ze vstupů představuje přepis hodnot ze vstupních jednotek PLC do oblasti X v zápisníkové paměti
- ♦ řešení uživatelského programu probíhá s hodnotami v zápisníkové paměti
- ♦ zápis do výstupů představuje přepis hodnot vypočtených uživatelským programem z oblasti Y do výstupních jednotek PLC
- ♦ režie zahrnuje přípravu centrální jednotky PLC k řešení dalšího cyklu programu

Činnosti zápisu do výstupů, režie a čtení ze vstupů jsou souhrnně nazývány otočka cyklu.

2.1. ZAPÍNACÍ SEKVENCE

Zapínací sekvencí rozumíme činnost PLC bezprostředně po zapnutí napájení. Obsahuje otestování sw i hw PLC a nastavení PLC do definovaného výchozího stavu. U centrálních jednotek vybavených nastavovacími tlačítky nebo v PLC vybavených panelem s klávesnicí (TC500) lze po zapnutí napájení vyvolat nastavovací režim určený pro nastavení parametrů.

Po ukončení zapínací sekvence je proveden restart, PLC přejde do režimu RUN a začne vykonávat uživatelský program. Pokud během zapínací sekvence diagnostika PLC vyhodnotila kritickou chybu, zůstává PLC v režimu HALT a signalizuje chybu.

Pokud byl vyvolán nastavovací režim, po jeho ukončení proběhne zapínací sekvence, ale pak PLC přejde do režimu HALT, uživatelský program není vykonáván, výstupy PLC zůstávají zablokované a PLC očekává příkazy z nadřazeného systému. Uživatelský program lze spustit buď pomocí nadřazeného systému, nebo vypnutím a zapnutím napájení. Tuto vlastnost lze využít v případech, že se do PLC dostane program, který podstatným způsobem narušuje jeho základní funkce.

Podrobnosti o chování jednotlivých typů PLC jsou uvedeny v příslušných příručkách.

2.2. PRACOVNÍ REŽIMY PLC

PLC TECOMAT může pracovat ve dvou základních režimech. Tyto režimy jsou označeny RUN a HALT.

Režim RUN

V režimu RUN PLC načítá hodnoty vstupních signálů ze vstupních jednotek, řeší instrukce uživatelského programu a zapisuje vypočtené hodnoty výstupních signálů do výstupních jednotek. Jedno provedení těchto činností představuje cyklus programu.

Z obr.2.1 vyplývá, že v případě PLC jde o nespojitě vyhodnocování vstupů (obecná vlastnost odlišující digitální systémy od plně analogových), jehož vzorkovací frekvence je daná dobou cyklu, která je určena především velikostí a strukturou uživatelského programu. Doba cyklu se pohybuje podle výkonu centrální jednotky řádově od jednotek přes desítky až po stovky milisekund.

Režim HALT

Režim HALT slouží především k činnostem spojeným s edicí uživatelského programu. V tomto režimu není program vykonáván a není ani prováděn přenos dat mezi centrální jednotkou a periferiemi.

Chování PLC při kritické chybě

Výjimku z uvedených pravidel tvoří situace, kdy v PLC vznikne kritická chyba, která brání v pokračování řízení. V tomto případě je v PLC spuštěn mechanismus ošetření kritické chyby, který provede ošetření chyby z hlediska bezpečnosti řízení a převede PLC **vždy** do režimu HALT.

Změna pracovních režimů

Změnu pracovních režimů PLC lze provádět pomocí nadřazeného systému (počítače), který je připojen na komunikační kanál podporující systémové služby (sériový kanál, Ethernet nebo USB), nebo pomocí služebních vstupů. Typicky je tímto nadřazeným systémem počítač standardu PC, který pracuje ve funkci programovacího zařízení nebo monitorovacího resp. vizualizačního pracoviště pro obsluhu řízeného objektu.

Při změně pracovních režimů PLC jsou některé činnosti prováděny standardně a některé je možno provádět volitelně. Obecně platí, že změna pracovního režimu PLC je činnost vyžadující zvýšenou pozornost obsluhy, neboť v mnoha případech velice výrazně ovlivňuje stav řízeného objektu. Příkladem může být přechod z režimu RUN do režimu HALT, kdy PLC přestane řešit uživatelský program a připojený objekt přestává být řízen. Doporučujeme proto důkladné studium následujícího textu.

V případě, že změna režimu PLC je prováděna pomocí vývojového prostředí Mosaic, jsou volitelné činnosti při změně režimu součástí manažeru projektu ve složce *Prostředí / Ovládání PLC*.

Přechod z HALT do RUN

V přechodu z režimu HALT do RUN se provádí:

- ◆ test neporušenosti uživatelského programu
- ◆ kontrola softwarové konfigurace periferních jednotek uvedené v uživatelském programu
- ◆ spuštění řešení uživatelského programu

Dále je možno volitelně provádět:

- ◆ nulování chyby PLC
- ◆ teplý nebo studený restart
- ◆ blokování výstupů při řešení uživatelského programu

Přechod z RUN do HALT

V přechodu z režimu RUN do HALT se provádí:

- ◆ zastavení řešení uživatelského programu
- ◆ zablokování (odpojení) výstupů PLC

Dále je možno volitelně provádět:

- ◆ nulování chyby PLC
- ◆ nulování výstupů PLC

Vznikne-li během činností prováděných při přechodu mezi režimy kritická chyba, PLC nastaví režim HALT, indikuje chybu a očekává odstranění příčiny chyby.

Upozornění: Zastavení řízení pomocí režimu HALT je určeno pouze pro účely ladění programu PLC. Tato funkce v žádném případě nenahrazuje funkci CENTRAL STOP. Obvody CENTRAL STOP musí být zapojeny tak, aby jejich funkce byla nezávislá na práci PLC!

Podrobnosti o chování a možnostech jednotlivých typů PLC jsou uvedeny v příslušných příručkách.

2.3. RESTARTY UŽIVATELSKÉHO PROGRAMU

Restartem se rozumí taková činnost PLC, jejímž úkolem je připravit PLC na řešení uživatelského programu. Restart se za normálních okolností provádí po úspěšném skončení zapínací sekvence a při každé změně uživatelského programu.

PLC TECOMAT rozlišují dva druhy restartu, teplý a studený. Teplý restart umožňuje zachování hodnot v registrech i během vypnutého napájení (remanentní zóna). Studený restart provádí vždy plnou inicializaci paměti.

Činnosti během restartu

Během restartu se provádí:

- ♦ test neporušenosti uživatelského programu
- ♦ nulování celého zápisníku PLC
- ♦ nulování remanentní zóny (pouze studený restart)
- ♦ nastavení zálohovaných registrů (pouze teplý restart)
- ♦ inicializace systémových registrů S
- ♦ inicializace a kontrola periferního systému PLC

Spuštění uživatelského programu bez restartu

Uživatelský program je také možné spustit bez restartu, v tom případě se provádí pouze test neporušenosti uživatelského programu a kontrola periferního systému PLC (nikoli inicializace).

Uživatelské procesy při restartu

V závislosti na prováděném restartu pracuje také plánovač uživatelských procesů P. Prováděl-li se v přechodu HALT → RUN teplý restart, je jako první po přechodu do RUN řešen uživatelský proces P62 (je-li naprogramován). Při studeném restartu je jako první po přechodu do RUN řešen uživatelský proces P63. Není-li restart při přechodu do RUN prováděn, je jako první po přechodu řešen proces P0.

Změna programu za chodu PLC

Vývojové prostředí Mosaic umožňuje též změnu programu za chodu PLC. Zde je třeba mít na vědomí tu skutečnost, že po dobu nahrávání nového programu je řešení programu pozastaveno bez zablokování výstupů. Tento stav může trvat i několik sekund!

Předvolba typu restartu

Typ restartu pro spuštění uživatelského programu z vývojového prostředí lze v prostředí Mosaic nastavit v manažeru projektu ve složce *Prostředí / Ovládání PLC*. Pro typ restartu po zapnutí napájení PLC (po úspěšné zapínací sekvenci) je určena volba ve složce *Sw / Cpm*. Toto nastavení bude automaticky při překladu přeneseno do uživatelského programu pomocí direktivy *#useoption*.

3. STRUKTURA UŽIVATELSKÉHO PROGRAMU

Programovací jazyk překladače xPRO

Programovací jazyk překladače xPRO obsaženého v prostředí Mosaic, vychází z mne-
monického jazyka PLC popsaného v kap.4. Rozšíření spočívá v možnosti používat sym-
bolická jména proměnných (registru, návěští, atd.) a používání direktiv pro překladač.
Překladač xPRO umožňuje provázání s překladači vyššího jazyka, např. podle IEC 61131.

Pravidla pro zápis programu

Zápis uživatelského programu se řídí několika jednoduchými pravidly:

- ♦ Na jednom řádku zdrojového textu programu může být nejvýše jedna instrukce. To znamená, že řádek může být i prázdný, nebo může obsahovat pouze komentář. Pozor - návěští je také instrukce!
- ♦ Symbolická jména mohou začínat písmeny 'a' až 'z', 'A' až 'Z' nebo '_', a mohou obsahovat znaky 'a' až 'ž', 'A' až 'Ž', '0' až '9' a '_' (podtržítka). Z toho plyne, že symbolické jméno nesmí začínat ani číslicí, ani písmenkem s diakritikou (háčky a čárky).

Pozor! Symbolické jméno nesmí být totožné s jakýmkoli z vyhrazených symbolů překladače (viz tab.3.1).

- ♦ Komentáře jsou uvozeny znakem ';' (středník). Veškerý text za tímto znakem až do konce řádku je překladačem považován za komentář a je ignorován.
- ♦ Malá a velká písmena lze používat libovolně, překladač interně převádí všechna písmena na velká, tedy nerozlišuje malá a velká písmena.

Podle těchto pravidel lze sestavit jednoduchý program:

Příklad 3.1

```
#def StartStop    %X0.0      ;deklarace vstupů, výstupů a konstant
#define Vystup     %Y0.0
#define Hodnota    21
#reg bool  Nulovani      ;deklarace registrů
#reg uint  Casovac, Citac
;
P 0                      ;začátek programu
    LD      StartStop    ;ovládací bit časovače
    LD      Nulovani     ;nulování časovače
    LD      Hodnota      ;předvolba časovače
    RTO     Casovac.3    ;funkce časovače sekund
    WR      Vystup       ;výstup
    JMC     skok         ;podmíněný skok
    INR     Citac        ;počet cyklů, kdy je Vystup = 0
skok:                  ;návěští
E 0                    ;konec programu
```

Program podle příkladu 3.1 představuje jednotlivé základní prvky uživatelského programu v prostředí Mosaic.

Prostředí Mosaic vytvoří automaticky záhlaví programu do řídicího souboru *xxx.mak*, kde *xxx* je jméno projektu v rámci skupiny projektů. V prostředí Mosaic tedy direktivu *#program* na rozdíl od prostředí xPRO do zdrojového souboru nepíšeme.

Následují deklarace vstupů, výstupů, konstant a registrů. Pokud vynecháme tyto deklarace, můžeme pak psát uživatelský program pomocí absolutních operandů (tj. registry X, Y, S, D, R). Tento přístup však zásadně nedoporučujeme, protože je nepřehledný a velmi znesnadňuje případné změny programu. Navíc některé programové konstrukce využívající struktur a symbolických odkazů jsou v absolutním vyjádření prakticky nerealizovatelné. Problémy také mohou nastat při přenosu programového algoritmu v absolutním vyjádření mezi jednotlivými typy centrálních jednotek.

Centrální jednotky s šířkou zásobníku 32 bitů podporující vyšší jazyk vyžadují tzv. procentovou konvenci při psaní absolutních operandů. To znamená, že před znak absolutního operandu musíme napsat znak % (viz příklad 3.1). Dále jsou všechny prefixy (*indx*, *bitpart*, *bitcnt*, *offset*, *sizeof*) 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.

Centrální jednotky s šířkou zásobníku 16 bitů není používání znaku % před absolutním operandem a podtržitek před prefixy povinné, ale doporučujeme je používat z důvodu přenositelnosti kódu.

Konfigurace PLC je popis použitých periferních zařízení pomocí direktiv *#unit*, resp. *#module* (TC650, TC700). Tyto direktivy určují propojení vstupů a výstupů se zápisníkovou pamětí a umožňují tak přenos dat mezi uživatelským programem PLC a okolním prostředím. Prostředí Mosaic generuje seznam direktiv *#unit* / *#module*, příslušných inicializačních tabulek a potřebných deklarací automaticky na základě vyplněných tabulek v konfigurační sekci (manažer projektu, složka *Hw* / *Konfigurace HW*) do samostatného souboru *xxx.hwc*, kde *xxx* je jméno projektu.

Pokud v prostředí Mosaic zaškrtneme volbu *Potlačit direktivu #UNIT* v konfigurační sekci, PLC bude řešit uživatelský program pouze v zápisníkové paměti. Vstupy nebudou do zápisníkové paměti načítány a do výstupů se nebude ze zápisníkové paměti zapisovat. Výstupy zůstanou zablokované. Tento stav může být užitečný při ladění algoritmu na PLC bez připojené technologie.

Následuje vlastní uživatelský program. Vzhledem k tomu, že v integrovaném prostředí Mosaic je obsažen překladač xPRO, je zápis instrukcí uživatelského programu v tomto prostředí totožný jako ve starším vývojovém prostředí xPRO. Každý uživatelský program musí obsahovat proces P0, i kdyby měl být prázdný. Instrukce P 0 a E 0 jsou tedy povinné.

Komentáře samozřejmě nejsou povinné, ale zde platí zásada, že čím lépe a podrobněji komentovaný program, tím snadněji se k němu vrátíme i třeba po roce, kdy už nevíme, o co jde, a potřebujeme v programu něco přidělat nebo pozměnit.

Odsazování instrukcí pomocí tabelátorů, jak je v příkladu naznačeno, také není nutné, k oddělení operandu stačí mezera a všechny instrukce bychom mohli psát hned z kraje řádku, ale snažíme se o co největší přehlednost programu. Odměnou nám bude minimum chyb.

V tabulce 3.1 je seznam vyhrazených symbolů překladače, které nelze použít jako symbolické jméno, protože je překladač používá k označení předem určených objektů. Tomuto seznamu je třeba věnovat zvýšenou pozornost, protože chyby vzniklé použitím vyhrazeného symbolu pro jiný účel, než ke kterému je určen, mohou vést k nepředvídatelnému chování.

3. Struktura uživatelského programu

Tab.3.1 Seznam vyhrazených symbolů překladače, které nelze použít jako symbolické jméno (platí pro velká i malá písmena)

A*	CHGS	DIGITOUT8	FTSS	LDQ	OPTION
ABS	CHPAR	DIGIT_050	G*	LDS	OR
ABSD	CMDF	DIGIT_200	GS*	LDSR	ORC
ABSL	CMF	DIGIT_300	GT	LDU	ORL
ACS	CML	DIGIT_400	GTDF	LDX	P
ACSD	CMP	DIGIT_500	GTF	LDY	PERIOD_500
ADD	CMPS	DIGIT_600	GTS	LEA	PERIOD_600
ADDF	CNT	DIGIT_633	H*	LEAX	PID
ADF	CNV	DIGIT_63X	HIGH	LEAY	PIDA
ADL	COLD	DIGOUT16	HPD	LET	PLC
ADX	COS	DIGOUT8	HPE	LETX	POP
ALIGNED	COSD	DIG_10IN_10OUT	HS*	LINK	POPB
ANALOG_050	COUNT_500	DIG_5IN_6OUT	HYP	LINT	POPL
ANALOG_200	COUNT_600	DINT	HYPD	LMS	POPQ
ANALOG_300	CP7001	DISPASCII	IC_04	LN	POPW
ANALOG_400	CP7002	DISPHEX	IDB	LND	POW
ANALOG_500	CPM1A	DIV	IDFL	LOG	POWD
ANALOG_600	CPM1B	DIVL	IDFW	LOGD	PROGRAM
ANC	CPM2B	DIVS	IF	LONG	PRV
AND	CPM1D	DL0, DL1...*	IFDEF	LOW	PSHB
ANL	CPM1E	DQ0, DQ1...*	IFL	LREAL	PSHL
AN_4IN	CPM1M	DS*	IFNDEF	LT	PSHQ
AN_4IN_4OUT	CPM1S	DST	IFW	LTB	PSHW
AN_8IN	CPM2S	DT	ILDF	LTDF	PUBLIC
AN_8IN_4OUT	CS*	DW0, DW1...*	ILF	LTF	PUT
AS*	CSG	DWORD	IMP	LTS	PUTX
ASB	CSGD	E	INCLUDE	LWORD	R*
ASN	CSGL	EC	INDX**	M*	R0, R1...*
ASND	CTD	ED	INR	M0, M1...*	RCHK
ATN	CTU	EOC	INT	MACRO	RD0, RD1...*
ATND	D*	EQ	INTIN_500	MAX	RDB
B*	D0, D1...*	EQDF	INTIN_600	MAXD	RDT
BAS	DATA	EQF	IRC_500	MAXF	REAL
BCD	DATE	ENDIF	IRC_600	MAXS	REC
BCL	DCR	ENDM	IWDF	MD0, MD1...*	RED
BCMP	DD0, DD1...*	ES*	IWF	MF0, MF1...*	REG
BET	DEF	ETH1, ETH2...	JB	MIN	REI
BETX	DF0, DF1...*	EXP	JC	MIND	RES
BIL	DFF	EXPD	JMC	MINF	RESM
BIN	DFST	EXTB	JMD	MINS	RESX
BIT	DID	EXTW	JMI	ML0, ML1...*	RET
BITCNT**	DIDF	F*	JMP	MNT	RF0, RF1...*
BITPART**	DIF	FDF	JNB	MOD	RFRM
BOOL	DIFCNT100MS	FIL	JNC	MODS	RL0, RL1...*
BOX	DIG2	FIS	JNS	MOV	RND
BP	DIG4	FIT	JNZ	MQ0, MQ1...*	RNDD
BRC	DIG8	FLG	JS	MTN	ROL
BRD	DIGIN16	FLO	JZ	MUD	ROR
BRE	DIGIN8	FLOAT	L	MUDF	RQ0, RQ1...*
BS*	DIGIN8OUT8	FLOD	L0, L1...*	MUF	RTO
BYTE	DIGIT2	FNS	LABEL	MUL	RW0, RW1...*
C*	DIGIT4	FNT	LAC	MULS	S*
CAC	DIGIT8	FS*	LD	MW0, MW1...*	S0, S1...*
CAD	DIGITIN16	FST	LDC	NEG	SCH2
CAI	DIGITIN32	FTB	LDI	NGL	SCMP
CAL	DIGITIN64	FTBN	LDIB	NOP	SCNV
CEI	DIGITIN8	FTM	LDIL	NXT	SCON
CEID	DIGITOUT16	FTMN	LDIQ	OFF	SD0, SD1...*
CH1, CH2...	DIGITOUT32	FTS	LDIW	OFFSET**	SDEL
CHG	DIGITOUT64	FTSF	LDL	ON	SEQ

Tab.3.1 Seznam vyhrazených symbolů překladače, které nelze použít jako symbolické jméno (platí pro velká i malá písmena) (pokračování)

SET	STATM	TER	UNLK	WRX	_ANALOG_
SETX	STDF	TF0, TF1...*	USB1, USB2...	WRY	_CHX
SF0, SF1...*	STE	TIME	USEOPTION	WSTRING	_GT_40_
SFL	STF	TL0, TL1...*	USI	WTB	_GT_40A_
SFND	STK	TOD	USINT	X*	_IC_12_
SFR	STRING	TOF	UWDF	X0, X1...*	_IC_13_
SHL	STRUCT	TON	UWF	XD0, XD1...*	_IM_61_
SHR	SUB	TQ0, TQ1...*	UW0,UW1...	XF0, XF1...*	_INTELIG
SIN	SUDF	TR050	UX_52	XH_04	_INTELLIGENT
SIND	SUF	TR200	VIRTMUX	XL0, XL1...*	_IR_11_
SINS	SUL	TR300	WAC	XOC	_IT_04_
SINT	SUX	TW0, TW1...*	WARM	XOL	_IT_06_
SIZEOF**	SW0, SW1...*	U*	WDB	XOR	_IT_12_
SL0, SL1...*	SWL	U0, U1...*	WMS	XQ0, XQ1...*	_IT_15_
SLEN	SWP	UD0, UD1...*	WORD	XW0, XW1...*	_KEYDISP_200_
SLFT	SYNC	UDFL	WR	X_OFF	_KEYDISP_500_
SMID	SYS	UDFW	WRA	X_ON	_OT_04_
SND	T*	UDINT	WRC	Y*	_OT_04X_
SPEC	T0, T1...*	UF0, UF1...*	WRI	Y_OFF	_OT_05_
SPECIAL	TABLE	UFL	WRIB	Y_ON	_OT_05X
SQ0, SQ1...*	TAN	UFW	WRIL	Y0, Y1...*	_PLCTYPE_
SQR	TAND	UINT	WRIQ	YD0, YD1...*	_SC_11_
SQRD	TC400	UL0, UL1...*	WRIW	YF0, YF1...*	_SPECIALTAB_
SRC	TC500	ULDF	WRS	YL0, YL1...*	_SPECTAB_
SRD	TC600	ULF	WRSR	YQ0, YQ1...*	
SREP	TC700	ULINT	WRT	YW0, YW1...*	
SRGT	TD0, TD1...*	UNIT	WRU	_ANAL_	

* Symboly označené * jsou v centrálních jednotkách s šířkou zásobníku 32 bitů povinně uvozovány znakem % a pak je možné je použít jako symbolické jméno. Např.:

```
%SW12          ;absolutní označení registru SW12
#def SW12 %X0   ;symbolické označení vstupu
```

** Symboly označené ** jsou v centrálních jednotkách s šířkou zásobníku 32 bitů povinně uvozovány znaky __ a pak je možné je použít jako symbolické jméno. Např.:

```
__indx (polozka) ;použití prefixu __indx
#def indx 20     ;symbolické označení konstanty
```

Poznámka: Vzhledem k neustálému vývoji a rozšiřování možností překladače xPRO doporučujeme nepoužívat jedno až čtyřpísmenná symbolická jména (především vycházející z anglických názvů nebo zkratk) jak samostatně tak s číselným indexem. V případě nezbytné nutnosti použít takové jméno doporučujeme napsat před něj znak podtržítka (např. _A).

4. STRUKTURA INSTRUKCÍ A OPERANDŮ

Instrukce

Instrukce je nejmenším prvkem uživatelského programu. Skládá se z mnemokódu a operandu. Z formálního hlediska rozlišujeme instrukce bezoperandové a instrukce s jedním operandem.

Mnemokód

Mnemokódem rozumíme skupinu jednoho až tří písmen, které mají význam zkratky odvozené zpravidla od anglického názvu instrukce (např. AND, OR, XOR, NEG, FLG, RET, ED, EC).

Bezoperandová instrukce

Bezoperandová instrukce zpravidla zpracovává obsah vrcholu, případně i dalších vrstev zásobníku nebo provádí jinou jednoznačně specifikovanou činnost (např. návrat z podprogramu). Bezoperandová instrukce je tvořena pouze mnemokódem a její činnost není třeba více upřesňovat.

Instrukce s jedním operandem

U operandové instrukce následuje za mnemokódem skupina znaků, které dohodnutým způsobem specifikují jeden z operandů, se kterým instrukce provádí pracuje, nebo které upřesňují chování instrukce (např. parametr instrukce, počet opakování základní operace, místo skoku apod.). Druhým operandem pro logické a aritmetické operace je vrchol zásobníku. Operand je od mnemokódu oddělen minimálně jednou mezerou.

Absolutní adresy jsou psány s uvozující 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.

Operandové instrukce mohou mít např. tvar:

AND	%X0
AND	%Y2
OR	%RW4
TON	%RW16.1
JMP	%L15
P	0
NOP	17
LTB	%T5
POP	3
LD	123
LD	%10110110

Druhy operandů

Podle významu můžeme rozlišit čtyři druhy operandů:

bezprostřední operand - operandem instrukce je přímo číselná hodnota (zapsaná ve zvolené číselné soustavě), se kterou se požadovaná instrukce provede
adresový operand - určuje adresu místa, odkud se čte, nebo kam se ukládá výsledek operace

cíl přechodu - operandem je číslo návěští (označené místo v programu), kam je směřován programovaný přechod (skok nebo volání)
 parametr instrukce - prostý číselný nebo písmenný parametr, který označuje danou instrukci (např. číslo procesu, číslo návěští, číslo prázdné instrukce, zásobník) nebo upřesňuje její chování (např. počet posuvů zásobníku).

Z hlediska formátu se jiné instrukce nevyskytují. Vyžaduje-li některá z instrukcí nebo podprogram více parametrů (funkční bloky, tabulkové nebo blokované instrukce), jsou předávány v několika úrovních zásobníku, kam se uloží posloupností instrukcí LD nebo LDC.

Typy operandů

Podle šíře dat rozlišujeme operandy několika typů. Tyto typy jsou označovány dvěma způsoby - jeden způsob je označen jako TECOMAT a jedná se o označení používaná předchozími verzemi překladače, druhý způsob odpovídá IEC 61131. Oba způsoby označování lze běžně používat pro všechny typy PLC. Přehled je uveden v tab.4.1.

Tab.4.1 Typy operandů

TECOMAT	IEC 61131	šířka	číselný rozsah
bit	bool	1 bit	dvouhodnotová informace 0 - 1
byte	byte	8 bitů	0 až 255
	usint	8 bitů	0 až 255
	sint	8 bitů	-128 až +127
word	word	16 bitů	0 až 65 535
	uint	16 bitů	0 až 65 535
	int	16 bitů	-32 768 až +32 767
long	dword	32 bitů	0 až 4 294 967 295
	udint	32 bitů	0 až 4 294 967 295
	dint	32 bitů	-2 147 483 648 až +2 147 483 647
float	real	32 bitů	$\pm 1,175494 \times 10^{-38}$ až $\pm 3,402823 \times 10^{38}$
double	lreal	64 bitů	$\pm 2,2 \times 10^{-308}$ až $\pm 1,8 \times 10^{308}$

Typy float a double, resp. real a lreal, obsahují číselné formáty s plovoucí řádovou čárkou podle IEEE-754.

V následujícím textu budeme používat typy podle IEC 61131. Uvedené informace platí i pro ekvivalentní typy TECOMAT.

4.1. BEZPROSTŘEDNÍ OPERAND

V tomto případě je jako operand instrukce zapsáno číslo, které je přímo zpracováno instrukcí - data nesená bezprostředně v instrukci. Bezprostřední operand má tedy význam číselné konstanty.

4.1.1. Číselné soustavy

Zápis konstanty v obecné číselné soustavě

Číselnou konstantu lze zadat v libovolné číselné soustavě v následujícím tvaru:

#n#cccc

kde n je základ číselné soustavy,
cccc je vlastní číslo ve zvolené soustavě

Pokud je základ číselné soustavy větší než 10, můžeme jednotlivé číslice zapisovat desítkově (ve formě dvojmístného čísla) a navzájem je oddělíme tečkou.

LD	#8#360	;oktalová (osmičková) soustava
WR	#60#15.28.35	;zápis časového údaje v hodinách, ;minutách a sekundách pomocí ;šedesátkové soustavy

Zkrácený zápis v nejčastějších číselných soustavách

Nejčastěji používané číselné soustavy jsou soustava dekadická (desítková) především pro aritmetické instrukce, binární (dvojková) a hexadecimální (šestnáctková) především pro logické instrukce. Tyto soustavy umožňují zkrácený zápis.

Pokud je číslo zapsáno bez udání číselné soustavy, je považováno za dekadické číslo. Binární soustava se označuje znakem % před vlastním číslem. Hexadecimální soustava se označuje znakem \$ před vlastním číslem, ve kterém se používají číslice 0 až 9, A až F.

LD	152	;dekadická soustava
AND	%0111110100110100	;binární soustava
AND	\$7D34	;hexadecimální soustava

Záporná čísla

Pro desítkovou soustavu je přípustný i zápis záporného čísla. Záporné znaménko způsobí, že místo dvojkového ekvivalentu zadaného čísla se jako operand instrukce uloží dvojkový doplněk tohoto čísla šířky 8 bitů (sint), 16 bitů (int) nebo 32 bitů (dint) podle typu instrukce.

Například hodnota -1 bude u typu sint odpovídat hodnotě 255, u typu int hodnotě 65 535, u typu dint hodnotě 4 294 967 295.

Typy real a lreal již informaci o znaménku obsahují.

4.1.2. Typy bezprostředních dat

Typ číselné konstanty je jednoznačně dán typem instrukce. Instrukce očekávají konstanty typu odpovídajícího popisu příslušné instrukce.

Pokud k instrukci, která vyžaduje číselnou konstantu šířky 32 bitů, napíšeme číslo šířky 8 bitů, je toto číslo překladačem doplněno nulami na šířku 32 bitů. Tento postup je tedy přípustný.

LDL	\$1F	;očekává se šířka 32 bitů
LDL	\$0000001F	;totožný zápis
AND	%11	;očekává se šířka 16 bitů
AND	%0000000000000011	;totožný zápis

Číselná konstanta typů real a lreal je přípustná pouze pro dekadickou soustavu. Vzhledem k tomu, že typ real má stejnou délku jako typy dword, udint a dint, ale zcela jinou interpretaci dat, vyznačuje se zápis číselné konstanty typu real především tím, že obsahuje desetinnou tečku i tehdy, jde-li o celé číslo. Přítomnost desetinné tečky je tedy informace důležitá pro překladač xPRO, aby přeložil konstantu do správného formátu.

LDL	1	;typ dint
LDL	1.0	;typ real
LDQ	1.0	;typ lreal

Poznámka: Místo instrukce LDL se v uživatelských programech pro centrální jednotky se zásobníkem šířky 32 bitů (CPU řady C) používá instrukce LD. Nicméně překladač akceptuje i LDL a automaticky ji převede na LD, čímž je zajištěna přenositelnost programů (viz kap.8. Zásobník výsledků).

Tab.4.2 Rozsahy bezprostředních operandů

byte / sint	byte / usint	word / int	word / uint
-128 až +127	0 až 255	-32768 až +32 767	0 až 65535
	%0 až %11111111		%0 až %1111111111111111
-\$80 až \$7F	\$0 až \$FF	-\$8000 až \$FFFF	\$0 až \$FFFF
	#60#0 až #60#8.15		#60#0 až #60#18.12.15

dword / dint	dword / uint
-2147483648 až +2147483647	0 až 4294967295
	%0 až %11111111111111111111111111111111
-\$80000000 až \$FFFFFFFF	\$0 až \$FFFFFFFF
	#60#0 až #60#59.59.59 *

real	lreal
$\pm 1,175494 \times 10^{-38}$ až $\pm 3,402823 \times 10^{38}$	$\pm 2,2 \times 10^{-308}$ až $\pm 1,8 \times 10^{308}$

* Šedesátková soustava platí jen pro hodiny, minuty a sekundy, ne pro dny.

Symbolické jméno jako operand

Jako bezprostřední operand lze použít i symbolické jméno definované direktivou *#def*. Lze použít i matematický výraz.

```
#def cislo 10
    LD    cislo          ;LD 10
    LD    cislo*4        ;LD 40
```

Číslo objektu jako operand - prefixy __indx, __bitpart, __bitcnt

Pokud potřebujeme jako operand použít číslo registru, tabulky nebo návěští, použijeme prefix *__indx*.

```
#reg uint   registr1,registr2    ;RW0, RW2
    LD      __indx (registr1)    ;LD 0
    LD      __indx (registr2)    ;LD 2
```

Pokud potřebujeme jako operand použít číslo bitu v registru, použijeme prefix *__bitpart*.

```
#reg bool   registr1,registr2    ;R10.0, R10.1
    LD      __bitpart (registr1) ;LD 0
    LD      __bitpart (registr2) ;LD 1
```

Pokud potřebujeme jako operand použít pořadové číslo bitu v celé zóně registrů, použijeme prefix *__bitcnt*.

```
#reg bool   registr1,registr2    ;R10.0, R10.1
    LD      __bitcnt (registr1)   ;LD 80
    LD      __bitcnt (registr2)   ;LD 81
```

Adresa objektu jako operand - prefix __offset

Pomocí prefixu *__offset* můžeme pracovat s adresou objektu, což vyžadují některé speciální instrukce jako ukazovátka pro umístění dat.

```
#reg usint   prac[20]            ;R150
    LD      __offset (seznam)     ;LD 790 - pro CPU řady B, D, E, M, S
```

;LD 24726 - pro CPU řady C

Délka objektu jako operand - prefix `__sizeof`

Pomocí prefixu `__sizeof` můžeme pracovat s délkou objektu, nejčastěji struktury.

```
#struct seznam  usint prvni, uint druhy, real treti
    LD      __sizeof (seznam)      ;LD 7
```

4.2. ADRESOVÝ OPERAND

Typ prostoru adresového operandu

Adresový operand má význam adresy místa, odkud se čte zpracovávaná informace nebo místa, kam se ukládá výsledek operace. Typ prostoru operandu je označen prvním znakem v operandu:

- X - obraz vstupů v zápisníku
- Y - obraz výstupů v zápisníku
- S - systémové registry v zápisníku
- R - uživatelské registry v zápisníku
- U - fyzické adresy
- D - zóna datových konstant v uživatelském programu
- T - zóna tabulek v uživatelském programu

Tento první znak je někdy označován jako specifikátor prostoru operandu.

V uživatelských programech pro centrální jednotky se zásobníkem šířky 32 bitů je povinný zápis uvozujícího znaku `%` před specifikátorem. Je tak jednoznačně určeno, že jde o absolutní adresu a ne o symbolické jméno. U centrálních jednotek se zásobníkem šířky 16 bitů tento způsob zápisu není povinný, ale doporučuje se s ohledem na přenositelnost uživatelských programů.

Symbolické jméno jako operand

Jako adresový operand zpravidla používáme symbolické jméno přiřazené pomocí direktiv `#def`, `#reg`, `#rem`, `#data` nebo `#table`.

```
#def vstup %X0
#rem  bool   registr1           ;R0.0 remanentní
#reg  bool   registr2           ;R1.0
#data  usint  zaznam = 1,2,3,4   ;D0, D1, D2, D3
#table uint   10,tab = 1,2,3,4   ;TW10
```

Tab.4.3 Rozsahy adresových operandů centrálních jednotek řady E a M

bool	byte / usint / sint	word / uint / int
X0.0 - X15.7	X0 - X15	XW0 - XW14
Y0.0 - Y15.7	Y0 - Y15	YW0 - YW14
S0.0 - S63.7	S0 - S63	SW0 - SW62
R0.0 - R255.7	R0 - R255	RW0 - RW254
-	U\$0000 - U\$FFFF *	UW\$0000 - UW\$FFFE *
D0.0 - D255.7	D0 - D255	DW0 - DW254
T0.0 - T255.0 *	T0 - T255 *	TW0 - TW255 *

* V centrálních jednotkách řady E není implementováno

Tab.4.4 Rozsahy adresových operandů centrálních jednotkách řady S

bool	byte / usint / sint	word / uint / int
X0.0 - X127.7	X0 - X127	XW0 - XW126
Y0.0 - Y127.7	Y0 - Y127	YW0 - YW126
S0.0 - S63.7	S0 - S63	SW0 - SW62
R0.0 - R511.7	R0 - R511	RW0 - RW510
-	U\$0000 - U\$FFFF	UW\$0000 - UW\$FFFE
D0.0 - D255.7	D0 - D255	DW0 - DW254
T0.0 - T255.0	T0 - T255	TW0 - TW255

Tab.4.5 Rozsahy adresových operandů centrálních jednotkách řady B a D

bool	byte / usint / sint	word / uint / int
X0.0 - X127.7	X0 - X127	XW0 - XW126
Y0.0 - Y127.7	Y0 - Y127	YW0 - YW126
S0.0 - S63.7	S0 - S63	SW0 - SW62
R0.0 - R8191.7	R0 - R8191	RW0 - RW8190
-	U\$0000 - U\$FFFF	UW\$0000 - UW\$FFFE
D0.0 - D255.0	D0 - D255	DW0 - DW254
T0.0 - Tmax	T0 - T255	TW0 - TW255

dword / udint / dint	real
XL0 - XL124	XF0 - XF124
YL0 - YL124	YF0 - YF124
SL0 - SL60	SF0 - SF60
RL0 - RL8188	RF0 - RF8188
-	-
DL0 - DL252	DF0 - DF252
-	-

Tab.4.6 Rozsahy adresových operandů centrálních jednotkách řady C*

bool	byte / usint / sint	word / uint / int
X0.0 - X8191.7	X0 - X8191	XW0 - XW8190
Y0.0 - Y8191.7	Y0 - Y8191	YW0 - YW8190
S0.0 - S6143.7	S0 - S6143	SW0 - SW6142
R0.0 - R40955.7	R0 - R40955	RW0 - RW40954
D0.0 - D2047.7	D0 - D2047	DW0 - DW2046
T0.0 - Tmax.0	T0 - Tmax	TW0 - TWmax

dword / udint / dint	real	lreal
XL0 - XL8188	XF0 - XF8188	XD0 - XD8184
YL0 - YL8188	YF0 - YF8188	YD0 - YD8184
SL0 - SL6140	SF0 - SF6140	SD0 - SD6136
RL0 - RL40952	RF0 - RF40952	RD0 - RD40948
DL0 - DL2044	DF0 - DF2044	DD0 - DD2040
TL0 - TLmax	TF0 - TFmax	-

* V centrálních jednotkách řady C jsou operandy U nahrazeny systémovou instrukcí RFRM, která provádí okamžitou aktualizaci dat určeného periferního modulu

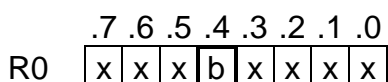
Upozornění: Každá tabulka T zabírá v paměti kromě dat ještě 4 byty navíc pro služební informace. Součástí uživatelského programu je seznam adres tabulek T vytvořená překladačem. Tento seznam má n+1 položek, kde n je nejvyšší číslo tabulky v uživatelském programu deklarované. V seznamu jsou adresy všech tabulek od T0 do Tn včetně těch, které nejsou deklarovány (mají nulovou adresu). Z toho vyplývá, že použijeme-li v programu pouze

tabulky vysokých čísel, zabere nám tabulka adres zbytečně velkou část paměti určené pro uživatelský program. Proto doporučujeme číslovat tabulky vzestupně od 0 (překladač xPRO tuto zásadu podporuje při používání symbolických jmen tabulek).

4.2.1. Operand typu bool

Data typu bool v zápisníku představují jeden konkrétní bit v bytu daný adresou bytu a číslem bitu.

Data nabývají hodnot 0 a 1 (kvůli odlišné interpretaci na zásobníku je označujeme jako log.0 a log.1).



*Obr.4.1 Uložení dat typu bool v zápisníku
b - logická hodnota bitu (log.0 nebo log.1)*

Operand typu bool je adresován adresou bytu v prostoru operandů a číslem bitu uvnitř tohoto bytu. V absolutním vyjádření je adresa bytu jednoznačně určena číslem v adresovém operandu. Číslo bitu má hodnotu 0 až 7 a zadává se za tečkou, např.:

```
LD    %X1.2
WR    %Y1.7
LD    %S53.4
WR    %R123.6
LD    %D25.6
```

Číslo 0 odpovídá nejnižší (dolní) bit bytu, číslu 7 odpovídá nejvyšší (horní) bit bytu. Pokud obsah bytu zobrazujeme jako dvojkové číslo nebo jako posloupnost bitových hodnot v řádku, pak dolní bit je uváděn nejvíce vpravo, horní bit nejvíce vlevo.

Operandy U neumožňují bitový přístup.

Operandy T určující bitový přístup k tabulkám mají číslo bitu vždy 0. Je tak odlišen bitový přístup od bytového, jinak nemá číslo bitu v tomto případě žádný význam.

```
LTB   %T4.0           ;čtení položky z bitové tabulky T4 (číslo položky
                      ;je určeno indexem uloženým na vrcholu zásobníku)
```

Symbolické vyjádření

V symbolickém vyjádření je operand určen direktivami *#def*, *#reg*, *#rem*, *#data* a *#table*.

```
#def vstup %X2.1
#rem  bool  registr1          ;R0.0 remanentní
#reg  bool  registr2          ;R1.0
#reg  bool  registr3          ;R1.1
#data  bool  zaznam = 1,0,1,0 ;D0.0, D0.1, D0.2, D0.3
#table bool  tab = 1,0,1,1    ;T0.0
```

Přetypování na bool

Pokud v uživatelském programu potřebujeme místně přetypovat operand (tedy použít jiný typ operandu, než je výše uvedenými direktivami definován), lze jej místně přetypovat na typ bool pouhým připsáním čísla bitu. Bity lze číslovat v celé šířce operandu, tedy 0 až 7 pro šířku 8 bitů, 0 až 15 pro šířku 16 bitů a 0 až 31 pro šířku 32 bitů.

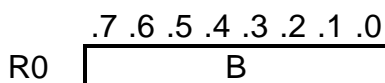
```
#reg  usint  registr1          ;R0
```

```
#reg   uint   registr2           ;RW1
#reg   uint   registr3           ;RL3
#table usint   tab = 1,2,3,4      ;T0
;
      LD      registr1.0          ;LD   %R0.0
      LD      registr2.5          ;LD   %R1.5
      LD      registr3.31         ;LD   %R6.7
      LTB     tab.0               ;LTB  %T0.0
```

4.2.2. Operand typu byte, usint, sint

Data typů byte, usint a sint v zápisníku představují jeden konkrétní byte daný adresou.

Data nabývají hodnot 0 až 255 (byte, usint) nebo při použití nejvyššího bitu jako znaménka –128 až +127 (sint).



Obr.4.2 Uložení dat typů byte, usint a sint v zápisníku
B - hodnota bytu

Operand v absolutním vyjádření je adresován číslem v adresovém operandu.

```
LD      %X1
WR      %Y0
LD      %S53
WR      %R123
LD      %D25
LTB     %T4
```

Operand U je adresován fyzickou hexadecimální adresou šířky 16 bitů (\$ značí hexadecimální číselnou soustavu).

```
LD      %U$9101      ;hexadecimální hodnota adresy
```

Symbolické vyjádření

V symbolickém vyjádření je operand určen direktivami *#def*, *#reg*, *#rem*, *#data* a *#table*.

```
#def vstup %X2
#rem   usint   registr1           ;R0 remanentní
#reg   byte    registr2           ;R1
#reg   usint   registr3           ;R2
#reg   sint    registr4           ;R3
#data  usint   zaznam = 1,2,3,4    ;D0, D1, D2, D3
#table usint   tab = 1,2,3,4      ;T0
```

Přetypování na byte, usint, sint

Pokud v uživatelském programu potřebujeme místně přetypovat operand (tedy použít jiný typ operandu, než je výše uvedenými direktivami definován), lze jej místně přetypovat připsáním prefixu *byte*, *usint*, *sint*.

```
#reg   bool    registr1           ;R0.0
#reg   uint    registr2           ;RW1
#reg   uint    registr3           ;RL3
#table uint    tab = 1,2,3,4      ;TW0
;
      LD      usint registr1       ;LD   %R0
```

```
LD    sint registr2      ;LD  %R1
LD    usint registr3+3   ;LD  %R6
LTB   usint tab          ;LTB %T0
```

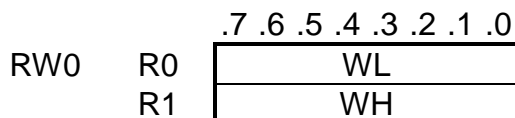
Proměnnou typů word, uint, int můžeme přetypovat na dvě proměnné typů byte, usint, sint také pomocí prefixů `__high` a `__low`.

```
LD    __high (registr2)   ;LD  %R2
LD    __low  (registr2)   ;LD  %R1
```

4.2.3. Operand typu word, uint, int

Data typů word, uint a int v zápisníku představují dva konkrétní byty dané adresou prvního z nich. Data jsou uložena tak, že významově nižší byte má nižší adresu než významově vyšší byte (konvence Intel).

Data nabývají hodnot 0 až 65 535 (word, uint) nebo při použití nejvyššího bitu jako znaménka -32 768 až +32 767 (int).



Obr.4.3 Uložení dat typů word, uint a int v zápisníku
WH - hodnota vyššího bytu
WL - hodnota nižšího bytu

Adresa operandu v absolutním vyjádření se zapisuje tak, že za specifikátorem prostoru je zapsán znak W jako symbol typu operandu a za ním číselný údaj, který adresuje nižší byte. Pokud například registr R14 obsahuje hodnotu \$21 a registr R15 hodnotu \$43, pak instrukcí

```
LD    %RW14
```

přečteme hodnotu \$4321. Podobně se zapisují ostatní operandy.

```
LD    %XW1
WR    %YW0
LD    %SW53
WR    %RW123
LD    %DW25
LTB   %TW4
```

Obdobně použití operandu U znamená práci s dvěma fyzickými adresami po sobě následujícími. Máme-li tedy například šestnáctivstupovou binární jednotku s adresou 2 v PLC NS950, jejíž stav vstupních signálů bytu 0 je binárně %1101 0001 a bytu 1 binárně %0010 1100, pak instrukcí

```
LD    %UW$9200
```

přečteme hodnotu \$2CD1.

Kódování časové jednotky u instrukcí časovačů

Instrukce časovačů (TON, TOF, RTO, IMP) pracují pouze s operandem typu uint, který obsahuje aktuální hodnotu časovače. Navíc je třeba zadat časovou jednotku, se kterou instrukce má pracovat a v níž je odměřována hodnota časovače. Možné jsou čtyři časové jednotky, které se zadávají kódovým číslem 0 až 3 uvedeným v instrukci za číslem adresy.

Oba údaje jsou odděleny tečkou. Časové jednotky jsou zakódovány následujícím způsobem:

- .0 - jednotka 10 ms
- .1 - jednotka 100 ms
- .2 - jednotka 1 s
- .3 - jednotka 10 s

Pokud kód časové jednotky není uveden, chápe se jako nulový, tj. jednotka 10 ms. Uvedme si příklady zápisu instrukcí časovačů:

```
TON  %RW10.0      ;časovač je v RW10, jednotka 10 ms
TON  %RW10         ;rovnocenný zápis
TOF  %RW24.1      ;časovač je v RW24, jednotka 100 ms
RTO  %RW32.2      ;časovač je v RW32, jednotka 1 s
IMP  %RW34.3      ;časovač je v RW34, jednotka 10 s
```

Symbolické vyjádření

V symbolickém vyjádření je operand určen direktivami *#def*, *#reg*, *#rem*, *#data* a *#table*.

```
#def vstup %XW2
#rem  uint  registr1      ;RW0 remanentní
#reg  word  registr2      ;RW2
#reg  uint  registr3      ;RW4
#reg  int   registr4      ;RW6
#data  uint  zaznam = 1,2,3,4 ;DW0, DW2, DW4, DW6
#table uint  tab = 1,2,3,4   ;TW0
```

Přetypování na word, uint, int

Pokud v uživatelském programu potřebujeme místně přetypovat operand (tedy použít jiný typ operandu, než je výše uvedenými direktivami definován), lze jej místně přetypovat připsáním prefixu *word*, *uint*, *int*.

```
#reg  bool  registr1      ;R0.0
#reg  usint registr2      ;R1
#reg  udint registr3      ;RL2
#table urint tab = 1,2,3,4 ;T0
;
      LD    uint registr1      ;LD  %RW0
      LD    int  registr2      ;LD  %RW1
      LD    uint registr3+2    ;LD  %RW4
      LTB   uint tab          ;LTB %TW0
```

4.2.4. Operand typu dword, udint, dint

Data typů *dword*, *udint* a *dint* v zápisníku představují čtyři konkrétní byty dané adresou prvního z nich. Data jsou uložena tak, že významově nejnižší byte má nejnižší adresu (konvence Intel).

Data nabývají hodnot 0 až 4 294 967 295 nebo při použití nejvyššího bitu jako znaménka -2 147 483 648 až +2 147 483 647.

		.7 .6 .5 .4 .3 .2 .1 .0
RL0	R0	L0
	R1	L1
	R2	L2
	R3	L3

Obr.4.4 Uložení dat typů *dword*, *udint* a *dint* v zápisníku

L0 - hodnota nejnižšího bytu

:

L3 - hodnota nejvyššího bytu

Adresa operandu v absolutním vyjádření se zapisuje tak, že za specifikátorem prostoru je zapsán znak L jako symbol typu operandu a za ním číselný údaj, který adresuje nejnižší byte. Pokud například registr R14 obsahuje hodnotu \$21, registr R15 hodnotu \$43, registr R16 hodnotu \$65 a registr R17 hodnotu \$87, pak instrukcí

```
LD    %RL14
```

přečteme hodnotu \$87654321. Podobně se zapisují ostatní operandy.

```
LD    %XL1
WR    %YL0
LD    %SL53
WR    %RL123
LD    %DL25
LTB   %TL4
```

Operand U neumožňuje přístup tohoto typu.

Symbolické vyjádření

V symbolickém vyjádření je operand určen direktivami *#def*, *#reg*, *#rem*, *#data* a *#table*.

```
#def vstup %XL4
#rem  udint  registr1          ;RL0 remanentní
#reg  dword  registr2          ;RL4
#reg  udint  registr3          ;RL8
#reg  dint   registr4          ;RL12
#data udint  zaznam = 1,2,3,4   ;DL0, DL4, DL8, DL12
#table udint  tab = 1,2,3,4     ;TL0
```

Přetypování na *dword*, *udint*, *dint*

Pokud v uživatelském programu potřebujeme místně přetypovat operand (tedy použít jiný typ operandu, než je výše uvedenými direktivami definován), lze jej místně přetypovat připsáním prefixu *dword*, *udint*, *dint*.

```
#reg  bool   registr1          ;R0.0
#reg  usint   registr2          ;R1
#reg  uint    registr3          ;RW2
#table usint  tab = 1,2,3,4     ;T0
;
LD    udint  registr1          ;LD %RL0
LD    dint   registr2          ;LD %RL1
LD    udint  registr3          ;LD %RL2
LTB   udint  tab                ;LTB %TL0
```

4.2.5. Operand typu real

Data typu real v zápisníku představují čtyři konkrétní byty dané adresou prvního z nich. Data jsou uložena tak, že významově nejnížší byte má nejnížší adresu (konvence Intel).

Data podle IEEE-754 (Institute of Electrical and Electronics Engineers) pro čísla s jednoduchou přesností (single precision) nabývají hodnot v rozsahu cca $\pm 1,175494 \times 10^{-38}$ až $\pm 3,402823 \times 10^{38}$ s přesností na přibližně 7 platných dekadických číslic. Dále jsou definovány čtyři hodnoty označující následující stavy:

\$7FFFFFFF	- neplatné číslo (NaN - not a number)
\$FFFFFFFF	- neplatné číslo (NaN - not a number)
\$7F800000	- překročení rozsahu kladných čísel (+INF)
\$FF800000	- překročení rozsahu záporných čísel (-INF)

Struktura typu real

Data jsou rozdělena do třech částí. Nejvyšší bit označený na obr.4.5 jako **s** určuje znaménko celého čísla. Je-li $s = 0$, je číslo kladné, je-li $s = 1$, je číslo záporné. Další 8 bitů označených jako **e** se nazývají exponent a nese informaci o velikosti čísla. Ostatních 23 bitů označených jako **m** se nazývají mantisa a nesou platné číslice (tj. bez nevýznamných levostranných nul).

Mantisa má nevýjádřený vedoucí bit, což znamená, že vyjadřuje binární číslo ve tvaru 1,mmmmmmmm. Exponent potom určuje počet binárních řádů, o které musíme desetinnou čárku pomyslně posunout, abychom získali požadované číslo. Posouváme-li desetinnou čárku doleva, bude exponent záporný, posouváme-li doprava, bude exponent kladný.

Exponent není vyjádřen ve dvojkovém doplňkovém kódu, ale v kódu posunutě nuly, to znamená, že se ke skutečné hodnotě exponentu přičte hodnota \$7F (127). Nula je tedy zapsána jako \$7F, jednička jako \$80, atd. Při dekódování exponentu musíme hodnotu \$7F odečíst.

Hodnotu čísla lze vyjádřit vztahem:

$$val = (-1)^s \times 2^{(e-127)} \times 1, m$$

Příklad přepočtu do formátu real

Číslo $12345 = 1,2345 \times 10^4 = \3039

ve formátu real: \$46 40 E4 00

Zpětný přepočet:

s e m
0 | 1000 1100 | 1000 0001 1100 1000 0000 000

$$val = (-1)^0 \times 2^{(140-127)} \times \left(\frac{1}{2} + \frac{1}{256} + \frac{1}{512} + \frac{1}{1024} + \frac{1}{8192} \right)$$

$$val = 1,5069578 \times 2^{13} = 12344,998$$

		.7	.6	.5	.4	.3	.2	.1	.0
RF0	R0	m	m	m	m	m	m	m	m
	R1	m	m	m	m	m	m	m	m
	R2	e	m	m	m	m	m	m	m
	R3	s	e	e	e	e	e	e	e

Obr.4.5 Uložení dat typu *real* v zápisníku
s - znaménko (1 bit)
e - exponent (8 bitů)
m - mantisa (23 bitů)

Adresa operandu v absolutním vyjádření se zapisuje tak, že za specifikátorem prostoru je zapsán znak F jako symbol typu operandu a za ním číselný údaj, který adresuje nejnižší byte. Princip je tedy analogický operandu typů *dword*, *uint*, *dint*.

```
LD    %XF1
WR    %YF0
LD    %SF53
WR    %RF123
LD    %DF25
LTB   %TF4
```

Operand U neumožňuje přístup tohoto typu.

Symbolické vyjádření

V symbolickém vyjádření je operand určen direktivami *#def*, *#reg*, *#rem*, *#data* a *#table*.

```
#def vstup %XF4
#rem   real   registr1           ;RF0 remanentní
#reg   real   registr2           ;RF4
#data  real   zaznam = 1.0,2.0,3.0,4.0 ;DF0, DF4, DF8, DF12
#table real   tab = 1.0,2.0,3.0,4.0    ;TF0
```

Přetypování na *real*

Pokud v uživatelském programu potřebujeme místně přetypovat operand (tedy použít jiný typ operandu, než je výše uvedenými direktivami definován), lze jej místně přetypovat připsáním prefixu *real*.

```
#reg   bool   registr1           ;R0.0
#reg   uint   registr2           ;R1
#reg   uint   registr3           ;RW2
#table uint   tab = 1,2,3,4      ;T0
;
LD     real   registr1           ;LD %RF0
LD     real   registr2           ;LD %RF1
LD     real   registr3           ;LD %RF2
LTB    real   tab                 ;LTB %TF0
```

Pozor! Pouhým přetypováním operandu na typ *real* nedojde k převodu obsahu operandu na typ *real*. K tomu je třeba použít příslušnou převodní instrukci (to se týká i převodů mezi typy *real* a *lreal*).

4.2.6. Operand typu lreal

Data typu lreal v zápisníku představují osm konkrétních bytů daných adresou prvního z nich. Data jsou uložena tak, že významově nejnižší byte má nejnižší adresu (konvence Intel).

Data podle IEEE-754 (Institute of Electrical and Electronics Engineers) pro čísla s dvojnásobnou přesností (double precision) nabývají hodnot v rozsahu cca $\pm 2,2 \times 10^{-308}$ až $\pm 1,8 \times 10^{308}$ s přesností na přibližně 16 platných dekadických číslic. Dále jsou definovány čtyři hodnoty označující následující stavy:

- \$7FFFFFFF FFFFFFFF - neplatné číslo (NaN - not a number)
- \$FFFFFFFF FFFFFFFF - neplatné číslo (NaN - not a number)
- \$7FF00000 00000000 - překročení rozsahu kladných čísel (+INF)
- \$FFF00000 00000000 - překročení rozsahu záporných čísel (-INF)

Struktura typu lreal

Data jsou rozdělena do třech částí. Nejvyšší bit označený na obr.4.6 jako **s** určuje znaménko celého čísla. Je-li $s = 0$, je číslo kladné, je-li $s = 1$, je číslo záporné. Dalších 11 bitů označených jako **e** se nazývají exponent a nese informaci o velikosti čísla. Ostatních 52 bitů označených jako **m** se nazývají mantisa a nesou platné číslice (tj. bez nevýznamných levostranných nul).

Mantisa má vyjádřený vedoucí bit, což znamená, že vyjadřuje binární číslo ve tvaru 1,mmmmmmmm. Exponent potom určuje počet binárních řádů, o které musíme desetinnou čárku pomyslně posunout, abychom získali požadované číslo. Posouváme-li desetinnou čárku doleva, bude exponent záporný, posouváme-li doprava, bude exponent kladný.

Exponent není vyjádřen ve dvojkovém doplňkovém kódu, ale v kódu posunutě nuly, to znamená, že se ke skutečné hodnotě exponentu přičte hodnota \$3FF (1023). Nula je tedy zapsána jako \$3FF, jednička jako \$400, atd. Při dekódování exponentu musíme hodnotu \$3FF odečíst.

Hodnotu čísla lze vyjádřit vztahem:

$$val = (-1)^s \times 2^{(e-1023)} \times 1, m$$

		.7	.6	.5	.4	.3	.2	.1	.0
RD0	R0	m	m	m	m	m	m	m	m
	R1	m	m	m	m	m	m	m	m
	R2	m	m	m	m	m	m	m	m
	R3	m	m	m	m	m	m	m	m
	R4	m	m	m	m	m	m	m	m
	R5	m	m	m	m	m	m	m	m
	R6	e	e	e	e	m	m	m	m
	R7	s	e	e	e	e	e	e	e

Obr.4.6 Uložení dat formátu lreal v zápisníku

s - znaménko (1 bit)
e - exponent (11 bitů)
m - mantisa (52 bitů)

Adresa operandu v absolutním vyjádření se zapisuje tak, že za specifikátorem prostoru je zapsán znak D jako symbol typu operandu a za ním číselný údaj, který adresuje nejnižší byte.

```
LD    %XD1
WR    %YD0
LD    %SD53
WR    %RD123
LD    %DD25
```

Operandy U a T neumožňují přístup tohoto typu.

Symbolické vyjádření

V symbolickém vyjádření je operand určen direktivami *#def*, *#reg*, *#rem* a *#data*.

```
#def vstup %XD4
#rem   lreal   registr1           ;RD0 remanentní
#reg   lreal   registr2           ;RD8
#data  lreal   zaznam = 1.0,2.0,3.0,4.0 ;DD0, DD8, DD16, DD24
```

Přetypování na lreal

Pokud v uživatelském programu potřebujeme místně přetypovat operand (tedy použít jiný typ operandu, než je výše uvedenými direktivami definován), lze jej místně přetypovat připsáním prefixu *lreal*.

```
#reg udint   registr1           ;RL0
#reg uint    registr2           ;RW4
;
LD   lreal   registr1           ;LD   %RD0
LD   lreal   registr2           ;LD   %RD4
```

Pozor! Pouhým přetypováním operandu na typ *lreal* nedojde k převodu obsahu operandu na typ *lreal*. K tomu je třeba použít příslušnou převodní instrukci (to se týká i převodů mezi typy *real* a *lreal*).

4.3. CÍL PŘECHODU

U instrukcí skoků a volání je operandem návěští, na které se přechod uskuteční. Instrukce má tvar např.

```
JMP    %L15
```

a předává vykonání programu na návěští L15. V programu se návěští zapisuje jako instrukce L s odpovídajícím číselným parametrem a slouží výhradně jako cíl skoků a volání. Při vykonávání se instrukce L interpretuje jako prázdná instrukce.

Mnohem účinnější je používání symbolických návěští, kde místo instrukce L s číslem návěští píšeme symbolické jméno, za které překladač sám dosadí číslo. Uživatelský program je tak optimalizován na co nejkratší seznam adres návěští, který je součástí programu a je informací pro PLC při instrukcích skoku.

```
skok1:                ;L 0
:
JMC   skok2            ;JMC %L1
JMP   skok1            ;JMP %L0
:
skok2:                ;L 1
```

Použití direktivy #label

Direktivu *#label* používáme jen tehdy, chceme-li některému návěští přiřadit konkrétní číslo nebo potřebujeme zaručit pořadí návěští pro nepřímé skoky instrukcemi JMI a CAI. Direktiva *#label* je nutná také v případě, že použijeme číslo návěští jako operand (s prefixem *__indx*) nebo jako položku tabulky dříve, než je návěští použito. Pro překladač by to bylo neznámé jméno.

Upozornění: Použijeme-li v uživatelském programu návěští L s nejvyšší hodnotou parametru *n*, překladač vytvoří seznam 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 L1023, zabere nám seznam návěští zbytečně celé 2 KB paměti, přičemž adresy všech ostatních návěští jsou nulové! Proto doporučujeme číslovat návěští vzestupně od 0 (překladač xPRO tuto zásadu podporuje při používání symbolických jmen návěští).

4.4. PARAMETR INSTRUKCE

Některé instrukce vyžadují zadání číselného parametru. U některých instrukcí se parametr pasivně ukládá a slouží pouze k identifikaci instrukce (např. u instrukce L, NOP, P, E). U některých instrukcí parametr ovlivňuje způsob vykonání instrukce, např. u instrukcí ROL, ROR, POP určuje počet kroků (posunutí).

Instrukce pracující s více zásobníky (CHG, CHGS, WAC, LAC) používají číselný parametr označující použitý zásobník.

Parametr instrukce se tedy zadává jako číselná hodnota. Jeho rozsah je určen typem instrukce.

5. STRUKTURA ZÁPISNÍKOVÉ PAMĚTI

Funkce zápisníku

Zápisníkem nebo též zápisníkovou pamětí rozumíme úsek paměťového prostoru PLC, který je přístupný jak pro čtení, tak i pro zápis uživatelských dat. Instrukce PLC umožňují přístup na libovolnou část zápisníku ve všech podporovaných formátech (podle řady CPU). Tato paměť je předem rozdělena do několika částí s vyhrazeným významem. Schématicky je uspořádání zápisníkové paměti zobrazeno na obr.5.1.

Rozdělení zápisníku

Zápisníková paměť je rozdělena na tyto části:

- ♦ obrazy vstupních signálů X
- ♦ obrazy výstupních signálů Y
- ♦ systémové registry S
- ♦ uživatelské registry R

Řada centrálních jednotek	E	M	S	D	B	C
Obrazy vstupních signálů X	X0 : X15	X0 : X15	X0 : X127	X0 : X127	X0 : X127	X0 : X8191
Obrazy výstupních signálů Y	Y0 : Y15	Y0 : Y15	Y0 : Y127	Y0 : Y127	Y0 : Y127	Y0 : Y8191
Systémové registry S	S0 : S63	S0 : S63	S0 : S63	S0 : S63	S0 : S63	S0 : S6143
Uživatelské registry R	R0 : R255	R0 : R255	R0 : R511	R0 : R8191	R0 : R8191	R0 : R40955

Obr.5.1 Struktura zápisníkové paměti včetně rozsahů operandů pro jednotlivé řady centrálních jednotek

Přístup k zápisníku

Všeobecně je dodržována zásada, že přístup systémového programu k zápisníkové paměti se uskutečňuje výhradně ve fázi otočky cyklu uživatelského programu. To se týká nejenom snímání fyzických vstupů do oblasti X a nastavování hodnot z oblasti Y na fyzické výstupy, ale i změn hodnot systémových proměnných S. To znamená, že po dobu cyklu uživatelského programu jsou údaje zápisníku zmrazeny a aktualizují se až po nejbližší otočce cyklu. Tím je výrazně omezena možnost výskytu různých hazardních stavů v uživatelském programu v důsledku asynchronnosti okamžiků změn jednotlivých proměnných. V průběhu cyklu se mění pouze ty proměnné, které ovlivňuje uživatelský program (přímý zápis do zápisníku - WR, WRC, WRA, PUT, LET, BET, SET, RES), nebo účinky některých funkcí (např. nastavení příznaků výsledků, aktualizace stavu čítačů, časovačů, posuvných registrů, apod.).

Upozornění: Je třeba si uvědomit, že okamžiky uživatelského přerušení bývají asynchronní vůči klidovému cyklu uživatelského programu a nesystematickým hospodařením nad proměnnými zápisníku si uživatel může vytvořit dostatek svých hazardních stavů. Je tedy nutné věnovat potřebnou péči přiřazení proměnných, vytvoření pravidel pro spolupráci mezi procesy klidové-

ho cyklu a mezi přerušujícími procesy. Výraznou podporu v tomto smyslu obsahuje vývojové prostředí Mosaic.

Zálohování dat při výpadku napájení

Při výpadku napájecího napětí je část obsahu zápisníku zálohována z náhradního zdroje (tzv. remanentní zóna v uživatelských registrech R). Při opětovném startu mohou být tyto zálohované hodnoty použity i pro další řízení - záleží na způsobu rozběhu a na dalších okolnostech (neporušenost zápisníku, nezměněný obsah uživatelského programu, apod.). Při volbě konfigurace si může uživatel zvolit velikost remanentní zóny.

5.1. OBRAZY VSTUPŮ X

Před každým začátkem cyklu programu zajišťuje centrální jednotka aktualizaci této oblasti zápisníkové paměti ze vstupních periferních jednotek na základě deklarační tabulky zadané v uživatelském programu, která popisuje přiřazení mezi obrazy vstupů X a fyzickými adresami jednotlivých jednotek.

V případě nedostatku paměti v oblasti X, lze bez omezení použít k témuž účelu oblast uživatelských registrů R.

5.2. OBRAZY VÝSTUPŮ Y

Po každém ukončení cyklu programu zajišťuje centrální jednotka přesun výsledků z této oblasti zápisníkové paměti do výstupů periferních jednotek na základě deklarační tabulky zadané v uživatelském programu, která popisuje přiřazení mezi obrazy výstupů Y a fyzickými adresami jednotlivých jednotek.

V případě nedostatku paměti v oblasti Y, lze bez omezení použít k témuž účelu oblast uživatelských registrů R.

5.3. SYSTÉMOVÉ REGISTRY S

Tato oblast zápisníkové paměti je vyhrazena pro specifické použití systémovým programem automatu a nedoporučuje se ji používat pro jiný účel. Některé bity a byty jsou pravidelně v otočce cyklu nastavovány systémovým programem a jsou vhodné pouze pro čtení. Některé bity naopak modifikují svým nastavením chování systémového programu.

5. Struktura zápisníkové paměti

Tab.5.1 Přehled systémových registrů

Registry	Použití	Řada CPU
S0	příznaky výsledků aritmetických operací	E M S D B C
S1	příznaky výsledků logických operací	E M S D B C
S2	příznaky stavu systému	E M S D B C
S3	doba minulého cyklu v 10 ms	E M S D B C
S4	čítač cyklů	E M S D B C
S5	čítač desítek milisekund systémového času	M S D B C
S6	čítač sekund systémového času	M S D B C
S7	čítač minut systémového času	M S D B C
S8	čítač hodin systémového času	M S D B C
S9	čítač dnů v týdnu	M S D B C
S10	čítač dnů v měsíci	M S D B C
S11	čítač měsíců	M S D B C
S12	čítač roků	M S D B C
S13	časové jednotky	M S D B C
S14 - S15	čítač v 100 ms	M S D B C
S16 - S17	čítač v 1 s	M S D B C
S18 - S19	čítač v 10 s	M S D B C
S20	náběžné hrany časových jednotek z S13	M S D B C
S21	sestupné hrany časových jednotek z S13	M S D B C
S22 - S23	doba minulého cyklu v 100 μs	S D B C
S24 - S29	řídící masky procesů	M S D B C
S30 - S33	rezerva	
S34	interní kód chyby	E M S D B C
S35	příznaky stavu hardwaru	E M S D B C
S36	teplota procesorové desky	C
S37	rezerva	
S38	číslo edice uživatelského programu	E M S D B C
S39	číslo změny obsahu	E M S D B C
S40 - S41	kód verze systémového programu PLC	E M S D B C
S42 - S43	rezerva	
S44 - S45	typ překladače	E M S D B C
S46 - S47	rezerva	
S48 - S51	úplný kód chyby	S D B C
S52 - S55	čítač 1 ms	C
S56 - S57	adresa přerušujícího modulu	C
S58 - S63	rezerva	
S64 - S75	systémové registry vyššího jazyka	C
S76 - S99	rezerva	
S100 - S227	stavová zóna periferního systému	C
S228 - S2047	rezerva	
S2048 - S6143	systémový stack	C

Pozor! Neobsazené systémové registry (v tab.5.1 označené jako rezerva) nesmějí být v žádném případě použity jako uživatelská paměť! Tyto registry mohou být používány systémem pro účely testování a diagnostiky. Zápis do nich může mít nepředvídatelné následky!

Významy systémových registrů jsou následující:

S0 - příznaky výsledků aritmetických operací

Ovlivňují jej pouze některé aritmetické instrukce, instrukce porovnání, instrukce časovačů a čítačů. Ostatní instrukce jej nemění.

S0.7	S0.6	S0.5	S0.4	S0.3	S0.2	S0.1	S0.0
0	D5.2	D5.1	D5.0	CI	≤	<(CO)	=(ZR)

- S0.0 (=) - rovnost obou operandů
(ZR) - nulovost výsledku
- dělení nulou při instrukcích dělení
- S0.1 (<) - první operand < druhý operand
(CO) - výstupní přenos (carry out), při operaci došlo k přenosu do vyššího řádu
- S0.2 (≤) - první operand ≤ druhý operand
- Logický součet bitů S0.0 OR S0.1
- S0.3 (CI) - vstupní přenos (carry in)
- Slouží ke kaskádování aritmetických operací ADD, SUB, INR, DCR, EQ, LT, GT (pouze v modelu 16 bitů). Pokud chceme v některé z těchto operací respektovat přenos z nižšího řádu, je třeba před touto operací nastavit CI na hodnotu přenosu. Při každém nastavení S0 (po vyjmenovaných instrukcích) se bit CI nuluje, takže další aritmetická instrukce se provádí bez přenosu zdola. Aritmetické operace se tedy kaskádují pouze tehdy, pokud před nimi uživatel zapíše hodnotu přenosu do bitu CI.
V modelu 32 bitů je tento příznak nefunkční.
- S0.4 až S0.6 (D5) - po instrukci BCD obsahují nejvyšší číslici výsledku (maximální hodnota je 6), po jiných instrukcích nastavujících S0 jsou bity nulovány
- S0.4 (OV) - překročení maximálního rozsahu časovače (v tomto cyklu nebo kdykoliv dříve během současné aktivace časovače)
- S0.5 (OC) - překročení maximálního rozsahu časovače právě v tomto cyklu
- S0.7 - rezerva

Podrobnosti jsou uvedeny v popisu instrukcí, které registr S0 ovlivňují.

S1 - příznaky výsledků logických operací

Tabulkové instrukce nastavují bit S1.0 ve významech:

S1.0 = 1 - položka v rozsahu tabulky, položka nalezena

S1.0 = 0 - položka mimo rozsah tabulky, položka nenalezena

Aritmetické instrukce pro práci v plovoucí řádové čárce, blokové operace a operace se strukturovanými tabulkami nastavují bit S1.0 ve významech:

S1.0 = 1 - vstupní parametry v pořádku, výsledek je platný

S1.0 = 0 - vstupní parametry mimo rozsah, výsledek je neplatný

Instrukce FLG nastavuje registr S1 podle obsahu vrcholu zásobníku.

S1.7	S1.6	S1.5	S1.4	S1.3	S1.2	S1.1	S1.0
ORH	ORL	ANH	ANL	N3	N2	N1	N0

5. Struktura zápisníkové paměti

- S1.0 až S1.3 (N) - dvojkové číslo (N4, N3, N2, N1, N0) nabývající hodnot 00000 až 10000, které mají význam počtu jedničkových bitů vrcholu zásobníku (bit N4 je zveřejněn ve všech bitech vrcholu zásobníku)
- S1.4 (ANL) - logický součin bitů dolního bytu vrcholu zásobníku A0
- S1.5 (ANH) - logický součin bitů horního bytu vrcholu zásobníku A0
- S1.6 (ORL) - logický součet bitů dolního bytu vrcholu zásobníku A0
- S1.7 (ORH) - logický součet bitů horního bytu vrcholu zásobníku A0

Instrukce STE nastavuje bity S1.0 a S1.1 ve významu:

- S1.0 = 1 - změnil se stav řadiče
- S1.0 = 0 - stav řadiče se nemění
- S1.1 = 1 - otáčka (přenos řadiče - stav se mění z 15 na 0)
- S1.1 = 0 - ostatní případy

Podrobnosti jsou uvedeny v popisu instrukcí, které registr S1 ovlivňují.

S2 - příznaky stavu systému

Nastavován systémovým programem podle jeho stavu v otočce cyklu.

S2.7	S2.6	S2.5	S2.4	S2.3	S2.2	S2.1	S2.0
LIM	NRS	ON	RST	HOT	RUN	MS	SP

- S2.0 (SP) - stav služebního vstupu SP - externí spouštění PLC
- S2.1 (MS) - stav služebního vstupu MS - externí blokování výstupů
- S2.2 (RUN) - 1 - start cyklu, vykonává se program (režim RUN)
0 - stop cyklu, nevykonává se program, PLC setrvává v otočce cyklu (režim HALT)
- S2.3 (HOT) - 1 - první průchod cyklem po teplém restartu
- S2.4 (RST) - 1 - první průchod cyklem po studeném restartu
- S2.5 (ON) - 1 - aktivní výstupy
0 - zablokované výstupy
- S2.6 (NRS) - 1 - první průchod cyklem bez restartu
- S2.7 (LIM) - 1 - překročena první mez doby cyklu (varování)

Bity S2.3, S2.4 a S2.6 jsou výhodné pro inicializaci proměnných.

Upozornění: Registr S2 je určen pouze pro indikaci, nikoli pro zápis.

S3 - doba minulého cyklu v 10 ms

Binární údaj s jednotkou 10 ms (rozsah 0 - 2,55 s) udává dobu trvání minulého cyklu uživatelského programu.

S4 - čítač cyklů

Binární údaj, který se při restartu systému nuluje a při každé otočce se zvýší o jedničku. Umožňuje složitější rozfázování řídicího algoritmu do jednotlivých oběhů.

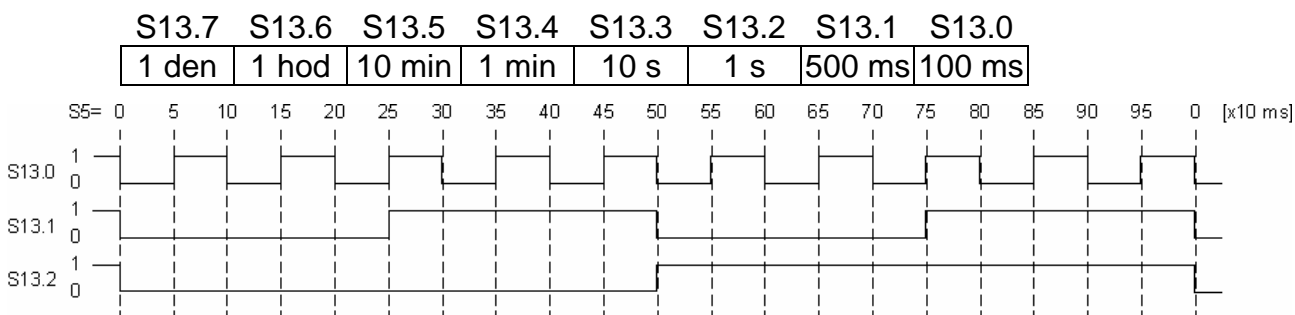
S5 až S12 - systémový čas a datum

Soubor binárních údajů, které mají význam času v časových jednotkách. Umožňuje používat v uživatelském programu hodinové a datumové údaje bez složitých přepočtů. Časový údaj je získáván z obvodu reálného času a obnovuje se v každé otočce cyklu. Při výpadku napájení se čas nezastaví, protože tento obvod je zálohován baterií.

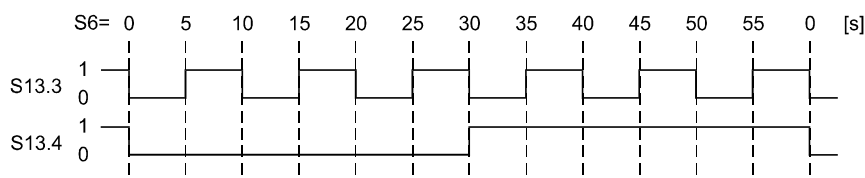
- S5 - čítač desítek milisekund (0 - 990 ms)
- S6 - čítač sekund (0 - 59 s)
- S7 - čítač minut (0 - 59 min)
- S8 - čítač hodin (0 - 23 hod)
- S9 - čítač dnů v týdnu (1 - 7)
- S10 - čítač dnů v měsíci (1 až poslední den v aktivním měsíci, přestupný rok je respektován)
- S11 - čítač měsíců (1 až 12)
- S12 - čítač roků - poslední dvojčíslí letopočtu (0 - 99)

S13 - časové jednotky

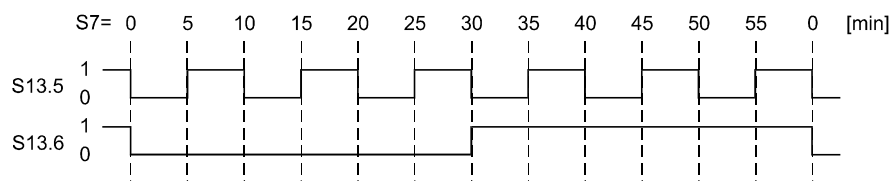
Soubor bitových proměnných, které změní svůj stav jednou za uvedenou jednotku. Průběh těchto časových signálů má střidu přibližně 1 : 1 a je odvozen od stavu S5 až S8 (obr.5.2 až obr.5.5). Mohou být využity jako zdroj časových impulsů pro uživatelské čítače, pro realizaci časových funkcí (blikání, obvod D, T, JK). Časoměrné proměnné jsou použitelné za předpokladu, že doba cyklu uživatelského programu je spolehlivě kratší, než polovina využívané časové jednotky. Jednotlivé bity mají význam:



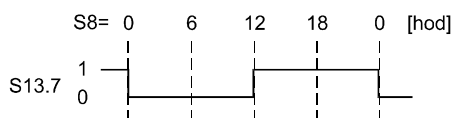
Obr.5.2 Stav bitových proměnných S13 ve vztahu k čítači desítek milisekund S5



Obr.5.3 Stav bitových proměnných S13 ve vztahu k sekundovému čítači S6



Obr.5.4 Stav bitových proměnných S13 ve vztahu k minutovému čítači S7



Obr.5.5 Stav bitových proměnných S13 ve vztahu k hodinovému čítači S8

S14, S15 - čítač po 100 ms

S16, S17 - čítač po 1 s

S18, S19 - čítač po 10 s

Každý word SW14, SW16, SW18 obsahuje dvojkový údaj v rozsahu do 65 535 časových jednotek. Změny těchto údajů probíhají synchronně se změnami S5 až S13. Hlavním využitím je náhrada časovačů, zejména v případech realizace posloupnosti časových intervalů (sekvenční a časové řízení). Jednotlivé bity mohou být využity jako zdroje časových jednotek. Podle potřeby může být využit jen dolní nebo horní byte.

S20 - náběžné hrany časových jednotek z S13

S21 - sestupné hrany časových jednotek z S13

Na pozicích se stejným významem jako u S13 jsou nastavovány změnové signály při změně stejnohléhlého obsahu S13 z log.0 do log.1 (S20) nebo z log.1 do log.0 (S21). Změny se vyhodnocují oproti stavu z minulého cyklu.

S20.7	S20.6	S20.5	S20.4	S20.3	S20.2	S20.1	S20.0
S21.7	S21.6	S21.5	S21.4	S21.3	S21.2	S21.1	S21.0
1 den	1 hod	10 min	1 min	10 s	1 s	500 ms	100 ms

S22, S23 - doba minulého cyklu v 100 μs

Dvojkový údaj s jednotkou 100 μs (rozsah 0 - 6,5535 s) udává dobu trvání minulého cyklu uživatelského programu. Jde o zpřesněný údaj registru S3.

S24 až S29 - řídicí masky procesů

Řídicí masky pro ovládání a indikaci aktivovaných procesů P1 až P48 (kap.10.). Platí přiřazení:

	.7	.6	.5	.4	.3	.2	.1	.0
S24	P8	P7	P6	P5	P4	P3	P2	P1
S25	P16	P15	P14	P13	P12	P11	P10	P9
S26	P24	P23	P22	P21	P20	P19	P18	P17
S27	P32	P31	P30	P29	P28	P27	P26	P25
S28	P40	P39	P38	P37	P36	P35	P34	P33
S29	P48	P47	P46	P45	P44	P43	P42	P41

Jedničky odpovídají aktivním procesům, nuly pasivním procesům.

Bity S24.0 až S25.0 nastavuje plánovač procesů v otočce cyklu a rozhoduje tak o aktivaci procesů P1 až P9. Pokud jsou tyto bity přepsány uživatelem, naplánovaný proces je přesto vykonán.

Bity S25.1 až S28.7 jsou k dispozici uživateli k ovládání procesů P10 až P40. Nastavení bitu na log.1 má za následek zařazení příslušného procesu do následujícího cyklu. Po restartu jsou tyto bity vždy vynulovány.

Bity S29.0 až S29.7 jsou k dispozici uživateli k zakázání nebo povolení provádění přerušovacích procesů P41 až P48. Tyto bity nejsou implementovány v centrálních jednotkách řady M, takže nelze vykonávání přerušovacího procesu zakázat. Po restartu jsou bity odpovídající přerušovacím procesům použitým v uživatelském programu nastaveny na log.1.

S30 až S33 - rezerva

S34 - interní kód chyby

První byte kódu poslední vzniklé chyby.

S34.7 = 1 (kód ≥ 128) - závažná chyba, vykonávání uživatelského programu se zastaví, PLC přejde do režimu HALT a zablokuje výstupy

S34.7 = 0 (kód < 128) - ostatní chyby neovlivňující závažně vlastní řízení, uživatelský program se vykonává dál, tyto chyby lze ošetřit uživatelsky pomocí tohoto systémového registru

- S34 = 0 - bezchybný stav
- 2 - chyba sériové komunikace
- 7 - chyba při kontrole remanentní zóny
- 8 - překročení první meze hlídání doby cyklu
- 9 - chybný systémový čas obvodu RTC
- 16 - dělení nulou
- 17 - počáteční index pro instrukci WMS je mimo tabulku T
- 18 - počáteční index pro instrukci LMS je mimo tabulku T
- 19 - tabulková instrukce nad zápisníkem překročila jeho rozsah
- 20 - zdrojový blok dat byl definován mimo rozsah zápisníku, dat či tabulky
- 21 - cílový blok dat byl definován mimo rozsah zápisníku či tabulky
- 32 - zjištěno porušení uživatelského programu při průběžné kontrole
- 80 - centrální jednotka nepodporuje služby pro uživatelské instrukce
- 81 - centrální jednotka nepodporuje požadovanou službu pro uživatelské instrukce
- 128 - chyby uživatelského programu
- 129 - chyby v periferním systému
- 130 - chyby komunikace s rozšiřujícími rámy
- 131 - chyby sériových kanálů
- 160 až 175 - chyby v periferním systému

S35 - příznaky stavu hardwaru

S35.7	S35.6	S35.5	S35.4	S35.3	S35.2	S35.1	S35.0
STEN	STIN	0	0	0	0	ERO	BAT

- S35.0 (BAT) - stav zálohovací baterie RAM a RTC
 - 1 - napětí zálohovací baterie je nižší než 2,1 V
 - 0 - zálohovací baterie je v pořádku
- S35.1 (ERO) - 1 - chyba binárního výstupu (zkrat výstupu, nezatížený výstup) (obsahují TC500, TC600)
- S35.6 (STIN) - indikace aktuálního času (TC700)
 - 0 - zimní čas
 - 1 - letní čas
- S35.7 (STEN) - automatický přechod mezi letním a zimním časem (TC700)
 - 0 - vypnuto, čas celý rok bez posunů
 - 1 - zapnuto, čas se mění automaticky

S36 - teplota procesorové desky (TC700)

Teplota měřená na procesorové desce centrální jednotky ve °C. Uživatelským programem tak lze od této teploty ovládat např. ventilátory v rozvaděči.

S37 - rezerva

S38 - číslo edice uživatelského programu

V zapínací sekvenci se kopíruje z konfigurační konstanty uživatelského programu. Hodnotu této konstanty je možné zadat prostřednictvím prostředí Mosaic.

S39 - číslo změny obsahu

V zapínací sekvenci se kopíruje z konfigurační konstanty uživatelského programu. Hodnotu této konstanty je možné zadat prostřednictvím prostředí Mosaic. Je určena především jako rozlišení verzí uživatelského programu.

S40, S41 - kód verze systémového programu centrální jednotky

Například pro verzi 4.1 bude S40 = 4 a S41 = 1.

S42, S43 - rezerva

S44, S45 - typ překladače

Značka překladače, kterým byl vytvořen uživatelský program.

S46, S47 - rezerva

S48 až S51 - úplný kód chyby

Úplný kód poslední vzniklé chyby určený pro snadné přečtení poslední vzniklé chyby nadřazeným systémem. Kód chyby je uložen jako typ udint, tj. první byte je uložen v registru S51 (totožný s obsahem registru S34), poslední byte je uložen v registru S48. Přehled kódů chyb je uveden v příručce příslušného typu PLC.

S52 až S55 - čítač po 1 ms

Čítač jednotek milisekund typu udint SL52. Umožňuje přesnější časové řízení. Jednotlivé bity mohou být využity jako zdroje časových jednotek. Podle potřeby může být využit libovolný byte nebo word.

S56, S57 - adresa přerušujícího modulu

Při vstupu do přerušujícího procesu P42 (přerušení od periferního modulu) obsahuje registr S56 pozici v rámu a registr S57 číslo rámu, kde je modul, který vyvolal toto přerušování, osazen. Tyto informace slouží k rozlišení přerušování od více modulů a lze je také využít pro získání parametrů k instrukcím RFRM a STATM. Podrobnosti jsou uvedeny v kap.10.5.

S58 až S63 - rezerva

S64 až S75 - systémové registry vyššího jazyka

Registry SL64, SL68, SL72 jsou využívány systémovými prostředky PLC podporujícími vyšší programovací jazyk. Hodnoty nesmí být v žádném případě měněny přímým přístupem.

S76 až S99 - rezerva

S100 až S227 - stavová zóna periferního systému

Registry S100 až S227 obsahují stavovou zónu periferního systému, která zveřejňuje okamžitý stav každého periferního modulu. To je důležité zejména v případě, kdy je povoleno vyjmutí periferního modulu za chodu a uživatelský program požaduje informaci, jestli jsou data čtená z modulu platná. Jinak může tato zóna sloužit pro podrobnější diagnostiku PLC realizovanou nadřazeným systémem.

Každé pozici v rámu modulárního PLC, resp. v sestavě kompaktního PLC (viz příslušnou příručku popisující konkrétní PLC), odpovídá jeden registr, jehož index lze odvodit podle následujícího vzorce:

$$n = (r * 16) + p + 100$$

kde n je výsledný index registru

r je číslo rámu

p je číslo pozice v rámu

Z toho plyne, že modul osazený v rámu 0 na pozici 0 má přidělen registr S100, modul na pozici 1 registr S101, ..., modul v rámu 1 na pozici 0 registr S116, atd. Všechny registry stavové zóny mají následující strukturu:

Sn.7	Sn.6	Sn.5	Sn.4	Sn.3	Sn.2	Sn.1	Sn.0
POS	OTH	DEC	ERR	0	0	DATA	ECOM

- Sn.0 (ECOM)
 - stav komunikace s modulem
 - 0 - komunikace je v pořádku
 - 1 - modul přestal komunikovat
- Sn.1 (DATA)
 - platnost přenášených dat
 - 0 - data v zápisníku nejsou aktuální, výměna dat neprobíhá
 - 1 - data v zápisníku jsou aktuální, výměna dat probíhá
- Sn.4 (ERR)
 - modul hlásí chybu
 - 0 - modul je bez chyby
 - 1 - modul hlásí závažnou chybu znemožňující výměnu dat
- Sn.5 (DEC)
 - obsluha modulu je deklarována
 - 0 - modul není obsluhován uživatelským programem
 - 1 - modul je obsluhován uživatelským programem
- Sn.6 (OTH)
 - chybný typ modulu
 - 0 - v pozici osazen modul požadovaný deklarací
 - 1 - v pozici osazen modul jiného typu, než je deklarováno
- Sn.7 (POS)
 - pozice obsazena
 - 0 - pozice není obsazena
 - 1 - na pozici byl nalezen modul

Podrobné chování stavové zóny v závislosti na typu PLC je uvedeno v příručce popisující příslušný typ PLC.

S228 až S2047 - rezerva

S2048 až S6143 - systémový stack

Oblast S2048 až S6143 je využita jako systémový stack PLC, využívaný systémovými prostředky PLC podporujícími vyšší programovací jazyk. Hodnoty zde uložené nesmí být v žádném případě měněny přímým přístupem.

5.4. UŽIVATELSKÉ REGISTRY R

Paměťová oblast určená pro proměnné uživatelského programu, pro realizaci čítačů, časovačů, posuvných registrů, krokových řadičů a dynamických tabulek.

V zapínací sekvenci systémového programu po studeném restartu jsou všechny registry R vynulovány. Po teplém restartu je uchována remanentní část registrů R, ostatní jsou vynulovány.

Remanentní zóna

Remanentní zóna je oblast registrů R, jejíž obsah zůstává zachován během vypnutí napájení PLC. Velikost této oblasti volíme pomocí překladače uživatelského programu (v prostředí Mosaic v manažeru projektu složka *Sw / Cpm*, položka *Zálohované registry*), začátek je vždy na registru R0. Maximální velikost remanentní zóny je dána řadou centrální jednotky (tab.5.2).

Pozor! Do remanentní zóny zásadně neumísťujeme obrazy vstupů a výstupů periferních jednotek. V případě inteligentních periferních jednotek může po restartu dojít k narušení zahájení výměny dat a tím k jejich nefunkčnosti.

Je ponecháno zcela na uživateli, jak použije registry R, které využije pro své pracovní proměnné nebo tabulky a které vyhradí pro realizaci funkčních bloků.

Překladač integrovaný v prostředí Mosaic umožňuje přímo deklarovat remanentní proměnnou pomocí direktivy *#rem*, která je ekvivalentní direktivě *#reg*, ale navíc proměnnou umísťuje do remanentní zóny (viz kap.9.5).

Tab.5.2 Maximální velikost remanentní zóny pro jednotlivé řady centrálních jednotek

Řada CPU	Maximální délka remanentní zóny	Zálohovatelné registry
C	16384	R0 - R16383
B	4096	R0 - R4095
D	512	R0 - R511
E, M, S	256	R0 - R255

Použití registrů R pro čítače a časovače

Systém neurčuje, kolik smí být použito čítačů, kolik časovačů apod. Jediným omezením je celkový počet registrů R dělen dvěma (typ uint), resp. čtyřmi (typ uint). Ke každé dvojici registrů R jsou v systému přiřazeny vnitřní příznaky pro funkce časovačů a čítačů. Systém umožňuje používat pro čítače a časovače jak dvojice začínající sudým registrem (RW0, RW2, RW4, ...), tak i lichým (RW1, RW3, RW5, ...). Nelze samozřejmě použít současně registry RW0 a RW1. Jednak dochází ke kolizi v registru R1 a jednak těmto registrům jsou přiřazeny stejné příznaky. Pokud budeme používat symbolické deklarace proměnných pomocí direktiv *#reg*, nemůže k podobné kolizi dojít. Totéž platí pro čítače typu uint.

O tom, zda objekt bude remanentní nebo neremanentní, rozhoduje též uživatel tím, do které oblasti objekt zařadí. Pokud uživatelský program používá časově omezené objekty, které se v čase vylučují, pak mohou být realizovány se stejnými registry R. Registry vyhrazené čítači nebo posuvnému registru mohou být obsluhovány různými instrukcemi čítání a posuvů, aniž by došlo k chybě (neplatí to pro časovače). Registry R přiřazené některému objektu jsou volně přístupné pro ostatní instrukce, takže např. hodnotu čítače nebo časovače lze porovnávat s několika údaji, případně ji měnit.

6. PŘÍMÉ PŘÍSTUPY KE VSTUPŮM A VÝSTUPŮM

6.1. PŘÍMÉ PŘÍSTUPY KE VSTUPŮM A VÝSTUPŮM - MODEL 16 BITŮ

Systémy TECOMAT TC400, TC500, TC600 a NS950 umožňují přímé přístupy do periferních jednotek pomocí operandů U obsahujících tzv. fyzickou adresu vstupu či výstupu.

Fyzická adresa

Pojmem fyzická adresa označujeme skutečnou adresu, kterou obsazují vstupy a výstupy na sběrnici. Pro běžnou obsluhu periférií je direktivou *#unit* přiřazen fyzický adresám obraz v zápisníku (oblast X, Y, R). Automatická aktualizace dat je prováděna vždy v otočce cyklu.

Operand U

Existují však případy, kdy je z časových důvodů nutné načíst z jednotky okamžitý stav, či neprodleně zapsat do jednotky hodnotu. K tomu pak slouží fyzické adresy označované operandem U. Operand U tedy poskytuje alternativu k oblastem X a Y v zápisníku, která umožňuje přímý styk s periferními jednotkami v daném okamžiku bez čekání na otočku cyklu.

Operand U umožňuje přístup šířky 8 a 16 bitů pomocí instrukcí LD a WR. Jeho využití je vhodné pouze pro zabezpečení časově kritických reakcí. Nadbytečné využívání má za následek zpomalení výkonu programu, protože přímý přístup k periferním jednotkám je časově náročnější než operace se zápisníkovou pamětí.

Fyzická adresa se zapisuje bezprostředně za operand U (resp. UW) vždy v hexadecimálním tvaru. Konkrétní fyzické adresy se liší podle typu PLC, což je dáno jeho konstrukcí. Podrobnosti jsou uvedeny v následujících kapitolách.

Omezení použití operandu U

Je třeba si uvědomit, že práce s fyzickými adresami porušuje zásadu neměnnosti vstupních a výstupních údajů během cyklu a zvyšuje riziko hazardů. Použití operandů U by mělo být omezeno výlučně na případy, kdy je třeba zajistit okamžitou odezvu, např. při ošetření havarijních situací, v procesech obsluhy přerušení, apod.

Důležitá upozornění

Zápisem na fyzickou adresu nebo čtením z fyzické adresy periferní jednotky **nedojde** k odpovídající změně hodnoty v obraze této jednotky v zápisníkové paměti!

V případě fyzického čtení dojde k opravě hodnoty v obraze během otočky cyklu a obvykle to není na závadu (je však třeba s tím počítat).

V případě fyzického zápisu je však nutné opravu v zápisníku zabezpečit uživatelským programem, jinak dojde v otočce cyklu k zápisu původní hodnoty ze zápisníku do jednotky.

Pokud je nutné fyzický zápis do jednotky používat, je lepší vypnout obsluhu výstupů jednotky v softwarové konfiguraci při překladači uživatelského programu v překladači (položka direktivy *#unit*) a obsluhovat výstupy výhradně fyzickými zápisy pomocí operandu U.

6.1.1. Fyzické adresy v PLC TECOMAT NS950

Fyzické adresy periferních jednotek jsou přístupné pouze v základním rámu (rám 0 obsluhovaný přímo centrální jednotkou). Periferní jednotky v rozšiřujících rámech jsou přístupné jen přes své obrazy v zápisníkové paměti.

Struktura fyzické adresy

Fyzická adresa periferní jednotky má následující strukturu:

horní byte adresy								dolní byte adresy							
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

- A15 - A12 - typ jednotky (dáno pevně)
0xxx (0-7) - systémové a virtuální jednotky
(speciální případy popsány v příslušných příručkách)
1000 (8) - 8 binárních vstupů nebo výstupů
(jednotky OR-14, OS-28, OS-32, OS-34, XH-04)
1001 (9) - 16 binárních vstupů nebo výstupů
(jednotky IB-36 až IB-47, IB-50, OR-15, OS-29, OS-30, OS-31, OS-33, OS-35)
1010 (A) - 32 binárních vstupů nebo výstupů
(jednotky IB-48, IB-49, OS-26, OS-27, UX-52)
1100 (C) - speciální jednotky bez inicializační tabulky
(jednotka IC-04)
1101 (D) - analogové jednotky bez vlastního procesoru
(jednotky IT-04, IT-12, IT-15, OT-04)
1111 (F) - jednotky s vlastním procesorem
(jednotky IT-06, OT-05, GT-41, IC-12 až IC-15, SC-11, CD-01 až CD-04, UP-01, UP-02)
- A11 - A8 - adresa jednotky v rámu (volitelná propojkami přístupnými na boku pouzdra periferní jednotky)
- A7 - 0 - vstupní adresa
1 - výstupní adresa
jednotky s vlastním procesorem (typ F) nemají vstupní a výstupní adresy odlišeny tímto bitem, jejich vstupní a výstupní zóny jsou přidělovány dynamicky podle požadované velikosti
- A6 - A0 - číslo bytu v rámci jednotky

Příklady použití fyzické adresy

```
LD    %U$8100    ;přímé čtení stavu osmi vstupů jednotky XH-04
                    ;s adresou 1 v rámu
LD    %UW$9200   ;přímé čtení stavu šestnácti vstupů jednotky typů
                    ;IB-36 až IB-47, IB-50 s adresou 2 v rámu
WR    %U$A083     ;přímý zápis hodnoty do osmi výstupů třetího bytu
                    ;jednotky typů OS-26, OS-27 s adresou 0 v rámu
WR    %UW$9380   ;přímý zápis hodnoty do šestnácti výstupů jednotky typů
                    ;OR-15, OS-29, OS-30, OS-31, OS-33, OS-35
                    ;s adresou 3 v rámu
```

Konkrétní adresy obsazované periferními jednotkami, pokud jsou dostupné, jsou uvedeny v příručkách těchto jednotek.

Pozor! Jednotky typů C, D, E, F obvykle vyžadují definovaný způsob obsluhy, který často vylučuje použití fyzické adresy. Proto v **žádném případě** nedoporučujeme používat přístup na fyzické adresy těchto jednotek bez předchozího důkladného studia jejich funkce! Pokud nejsou v popisu těchto jednotek fyzické adresy výslovně uvedeny, jednotky přímé přístupy neumožňují!

6.1.2. Fyzické adresy v PLC TECOMAT TC400, TC500, TC600

Struktura fyzické adresy

Fyzická adresa binárních a analogových vstupů a výstupů má následující strukturu:

horní byte adresy								dolní byte adresy							
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

- A15 - A12 - typ jednotky (dáno pevně)
 - 0xxx (0-7) - rezerva
 - 1000 (8) - deska s max. 8 binárními vstupy nebo výstupy
 - 1001 (9) - deska s max. 16 binárními vstupy nebo výstupy
 - 1010 (A) - deska s max. 24 binárními vstupy nebo výstupy
 - 1101 (D) - deska s analogovými vstupy a výstupy
- A11 - A8 - vždy 0
- A7 - 0 - vstupní adresa
 1 - výstupní adresa
- A6 - A0 - číslo bytu v rámci jednotky

Příklady použití fyzické adresy

```
LD    %UW$9000    ;přímé čtení stavu vstupů
WR    %UW$9080    ;přímý zápis hodnoty do výstupů
```

Konkrétní adresy obsazované periferními jednotkami jsou uvedeny v příručkách Technické vybavení programovatelných automatů TECOMAT TC400 TXV 138 12.01, Technické vybavení programovatelných automatů TECOMAT TC500 TXV 138 07.01 a Technické vybavení programovatelných automatů TECOMAT TC600 TXV 138 08.01.

6.2. PŘÍMÉ PŘÍSTUPY KE VSTUPŮM A VÝSTUPŮM - MODEL 32 BITŮ

Systémy TECOMAT TC650 a TC700 nepoužívají fyzické adresy, ale pro rychlý přístup k periferním modulům mají zvláštní systémovou instrukci RFRM, která provádí okamžitou výměnu dat mezi zápisníkem centrální jednotky a příslušným periferním modulem. S daty se pak operuje pomocí běžných instrukcí pracujících se zápisníkem. Narozdíl od fyzické adresy, zde je automatická vazba na zápisník zajištěna.

Periferní modul, ke kterému realizujeme přímý přístup, **musí mít zapnutou obsluhu** (zde je podstatný rozdíl oproti systémům používajícím operandy U).

I zde platí, že využití přímého přístupu je vhodné pouze pro zabezpečení časově kritických reakcí. Nadbytečné využívání má za následek zpomalení výkonu programu, protože přímý přístup k jednotlivým periferním modulům je časově náročný.

Je třeba si uvědomit, že okamžitá aktualizace dat periferního modulu porušuje zásadu neměnnosti vstupních a výstupních údajů během cyklu a zvyšuje riziko hazardů. Použití by mělo být omezeno výlučně na případy, kdy je třeba zajistit okamžitou odezvu, např. při ošetření havarijních situací, v procesech obsluhy přerušení, apod.

7. OSTATNÍ ADRESOVÉ PROSTORY

ATA

Data D mají význam konstant uživatelského programu. Jsou součástí uživatelského programu a jsou pro něj dostupná pouze pro čtení. Mohou se zadávat a měnit pouze v rámci editace uživatelského programu.

Data D mohou být výhodně použita jako parametry, které modifikují uživatelský program. Pro určitou třídu řídicích algoritmů může být například vytvořen a odladěn jediný uživatelský program, který je před konkrétním použitím přizpůsoben skutečným podmínkám zadáním odpovídajících parametrů v zóně D. Obdobně lze uživatelský program přizpůsobovat měnícím se požadavkům zadavatele, změnám technologie, měnícímu se sortimentu výrobků, apod.

V datech D mohou být uloženy i souvislé posloupnosti hodnot, například tabulky nebo vzory pro nastavení zápisníku a výstupů v klíčových situacích.

7.2. TABULKY T

Tabulky T jsou součástí uživatelského programu. Výhodně mohou být použity jako parametry, které modifikují uživatelský program. Pro určitou třídu řídicích algoritmů může být například vytvořen a odladěn jediný uživatelský program, který je před konkrétním použitím přizpůsoben skutečným podmínkám zadáním odpovídajících parametrů v tabulce T. Obdobně lze uživatelský program přizpůsobovat měnícím se požadavkům zadavatele, změnám technologie, měnícímu se sortimentu výrobků apod.

Tabulky T jsou dostupné pouze zvláštními instrukcemi, které se odvolávají na adresový prostor T (tabulkové instrukce, instrukce blokových přenosů). Mají předepsanou strukturu - je to vždy řada hodnot stejné šířky (1, 8, 16, 32 bitů) s přídavným údajem o délce této řady v bytech. Každé položce (hodnotě z této řady) je přiřazeno pořadové číslo - index. Nejnižší položka má nulový index, index poslední položky nazýváme mezí (počet položek = mez + 1). Tabulky T tedy mohou být zpracovávány buď jako řada hodnot libovolného typu kromě lreal (obr.7.1, obr.7.2, obr.7.3, obr.7.4). Tyto typy jsou bezprostředně určeny typem použité instrukce, nezávisí tedy na tabulce. Fyzicky jsou tabulky T uloženy jako posloupnost hodnot s údajem o její délce v bytech. Při bitovém zpracování tabulky jsou z toho důvodu vždy poslední nevyužité bity doplněny nulami na celý byte. Bitová tabulka vždy začíná na bitu č. 0 (nemůže začínat uprostřed bytu).

Nad tabulkami T lze provádět tyto operace:

- ◆ výběr položky k zadanému indexu (LTB)
- ◆ zápis položky podle zadaného indexu (WTB)
- ◆ nalezení indexu k zadané hodnotě položky (FTB, FTBN)
- ◆ nalezení indexu k vybrané části položky (FTM, FTMN)
- ◆ nalezení indexu třídy - zařazení zadané hodnoty do jedné z tříd (skupin), které jsou určeny tabulkou mezí v uspořádané řadě hodnot (FTS, FTSF, FTSS)
- ◆ blokové přenosy dat mezi tabulkou zápisníkem (SRC, MOV, MTN, MNT)
- ◆ práce se strukturovanými tabulkami (LDS, WRS, FIT, FNT)

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. Výsledný program je názorný a přehledný.

Omezení počtu a velikosti tabulek

Centrální jednotky řady E používají tabulky T výlučně pro inicializaci periferních jednotek.

Centrální jednotky řady M mohou mít maximálně 256 tabulek maximální délky 255 bytů. Z toho vyplývá, že tabulkovými instrukcemi je zpracovatelných maximálně 127 položek velikosti word (254 bytů), 255 položek velikosti byte a 256 položek velikosti bit (32 bytů).

Centrální jednotky řady S mohou mít maximálně 256 tabulek. Délky tabulek nejsou omezené.

U centrálních jednotek řady B, C a D je omezujícím faktorem pouze velikost paměti uživatelského programu.

Příklady tabulek

index =	7	6	5	4	3	2	1	0	
	↓	↓	↓	↓	↓	↓	↓	↓	
	1	1	1	1	1	0	0	1	byte č.0
15 →	1	0	1	0	1	1	1	0	← 8 byte č.1
23 →	1	1	0	0	0	1	1	1	← 16 byte č.2
31 →	0	1	1	1	0	0	0	0	← 24 byte č.3

	0	0	0	0	1	0	0	0	byte č.7
					↑	↑	↑	↑	
bitová mez = 55					51	50	49	48	
	.7	.6	.5	.4	.3	.2	.1	.0	čísla bitu v byte

Obr.7.1 Příklad tabulky typu bool (čtyři poslední bity jsou doplněny 0 na celý byte)

index = 0	156	byte č.0
1	255	byte č.1
2	147	byte č.2
3	64	byte č.3
:
21	0	byte č.21
mez = 22	235	byte č.22

Obr.7.2 Příklad tabulky typu byte, usint, sint

	horní byte položky	dolní byte položky	
index = 0	\$00	\$55	byte č.1 a 0
1	\$12	\$FF	byte č.3 a 2
2	\$01	\$47	byte č.5 a 4
3	\$15	\$40	byte č.7 a 6
:	
14	\$00	\$00	byte č.29 a 28
mez = 15	\$00	\$35	byte č.31 a 30

Obr.7.3 Příklad tabulky typu word, uint, int

	nejvyšší byte položky			nejnižší byte položky	
index = 0	\$15	\$12	\$FF	\$55	byte č.3 až 0
1	\$47	\$11	\$08	\$00	byte č.7 až 4
2	\$00	\$40	\$56	\$00	byte č.11 až 8
3	\$00	AB	\$0C	\$40	byte č.15 až 12
:
8	\$48	\$D8	\$15	\$08	byte č.31 až 28
mez = 9	\$07	\$35	\$20	\$04	byte č.35 až 32

Obr.7.4 Příklad tabulky typu dword, uint, dint a real

7.3. PŘÍDAVNÁ PAMĚŤ DAT DATABOX

DataBox je přídatná paměť určená pro práci s větším množstvím dat, např. archivace údajů o řízeném procesu za delší časové období, apod. V centrálních jednotkách je realizovaná buď ve formě přídatného submodulu nebo je součástí standardního osazení. DataBox je paměť CMOS RAM, která je zálohovaná baterií z centrální jednotky. Data lze do paměti zapisovat resp. číst buď uživatelským programem PLC nebo po sériové lince. V prostředí Mosaic je paměť DataBox přístupná stejným způsobem jako zápisníková paměť, tedy lze ji i ručně editovat.

Sériová komunikace s pamětí DataBox

Pro sériovou komunikaci lze využít libovolný sériový kanál pracující v režimu **PC**. Program umožňující přečíst data z paměti DataBox do souboru, resp. zapsat data ze souboru do paměti DataBox, se jmenuje *complc32.exe* (Windows 2000, XP), resp. *complc.exe* (DOS, Windows 95, 98). Nabízí také možnost otestovat velikost paměti přístupné jako DataBox. Programy *complc32.exe* a *complc.exe* jsou součástí instalace prostředí Mosaic.

Instrukce pro práci s DataBoxem

Uživatel má pro práci s pamětí DataBox k dispozici tři instrukce. Instrukce RDB pro čtení dat z DataBoxu do registrů R, instrukce WDB pro zápis dat z registrů R do DataBoxu a instrukce IDB pro identifikaci velikosti DataBoxu.

Funkce instrukce IDB

Pro zjištění velikosti osazeného DataBoxu je určena instrukce IDB. Tato instrukce nevyžaduje žádné vstupní parametry. Po vykonání zvýší uživatelský zásobník o jednu úroveň a na vrchol zásobníku zapíše zjištěnou velikost DataBoxu v KB, tzn. hodnotu 128, 512, apod. Pokud není DataBox nalezen, vrátí instrukce hodnotu 0.

Při simulaci v prostředí Mosaic instrukce IDB vytvoří soubor databox.\$\$\$ v adresáři projektu (pokud již neexistuje). Velikost souboru, který chceme vytvořit, je předávána ve vrstvě A0 aktivního zásobníku v KB. Nově vytvořený soubor je binární a je naplněn nulami. Instrukce IDB zvýší uživatelský zásobník o jednu úroveň a vrátí požadovanou velikost DataBoxu v následujících případech:

- ♦ soubor databox.\$\$\$ byl úspěšně vytvořen
- ♦ soubor databox.\$\$\$ již existuje a má požadovanou velikost (nebo je dokonce větší)

V následujících případech bude instrukce IDB vracet v A0 hodnotu 0:

- ♦ soubor databox.\$\$\$ se nepodařilo vytvořit (např. je málo místa na disku)
- ♦ soubor databox.\$\$\$ existuje, ale při jeho otvírání vznikla chyba (např. je porušena struktura souboru, nebo řadič disku hlásí vadný sektor, atd.)
- ♦ soubor databox.\$\$\$ existuje, ale má menší velikost, nežli je požadovaná - v tomto případě zůstane obsah souboru databox.\$\$\$ nezměněn. Aby byl pro simulaci vytvořen soubor databox.\$\$\$ s větší velikostí, je nutné původní soubor smazat nebo přesunout do jiného adresáře.

Z uvedeného vyplývá, že v simulaci pracuje instrukce IDB se souborem databox.\$\$\$ v adresáři projektu. Velikost vytvořeného souboru odpovídá požadavku, který se naplní do vrstvy A0 před voláním instrukce. V reálném PLC je velikost DataBoxu daná velikostí osazené paměti, což znamená, že může být větší, než je požadavek. Příklad uvedený na konci této kapitoly řeší tuto situaci neostrou nerovností, a tak je možno použít ho jak pro simulaci tak pro reálný PLC. Testování velikosti DataBoxu se provádí typicky v rámci procesu pro teplý nebo studený restart.

Struktura zóny parametrů pro RDB a WDB

Před voláním instrukcí RDB a WDB je nutno nastavit několik parametrů. Tyto parametry jsou umístěny v registrech R, musí být uloženy těsně za sebou a jejich pořadí je nutné dodržet. Číslo registru, ve kterém je umístěn první parametr, se předává na zásobníku při volání instrukce RDB a WDB (viz dále). Parametry jsou seřazeny v následujícím pořadí:

Název parametru	Typ	Význam
adrDB	udint	adresa v paměti DataBox
indR	uint	index počátečního registru v zápisníku
len	usint	počet přenášených bytů

Parametry lze v programu definovat symbolicky s automatickým přiřazením registrů

```
#reg udint   adrDB
#reg uint    indR
#reg usint   len
```

Z hlediska programování je nejlepší definice pomocí direktivy *#struct*, neboť automaticky zajišťuje požadavek na následnost parametrů:

```
#struct parDB          ;jméno struktury
    udint adrDB,       ;adresa v DataBoxu
    uint  indR,        ;index počátečního registru v zápisníku
    usint len          ;počet přenášených bytů
;
#reg parDB parusi
```

Funkce instrukcí RDB a WDB

Instrukce RDB a WDB vykonáním nemění úroveň uživatelského zásobníku. Na vrcholu zásobníku vrací počet skutečně přenesených dat. Zároveň nastavují obsah systémového registru S1.0 ve významu:

S1.0 = log.1 - vstupní parametry v pořádku, výsledek je platný
 S1.0 = log.0 - vstupní parametry mimo rozsah, výsledek je neplatný

Pokud je S1.0 = log.0, nedojde k přenosu žádných dat a zároveň je nastaven obsah registru S34 ve významu:

S34 = \$14 - zdrojový blok dat byl definován mimo rozsah
 S34 = \$15 - cílový blok dat byl definován mimo rozsah

7. Ostatní adresové prostory

Podle velikosti osazené paměti DataBox je pro použití dostupný adresový prostor uvedený v tab.7.1.

Tab.7.1 Dostupný adresový prostor jednotlivých pamětí DataBox

Velikost paměti DataBox	Dostupný adresový prostor
128 KB (CPU řady D a S)	0 - \$01FFFFB
128 KB (CPU řady B a C)	0 - \$01FFFF
512 KB (CPU řady D a S)	0 - \$07FFFEF
1,5 MB (CPU řady B)	0 - \$17FFFF
3,0 MB (CPU řady C)	0 - \$2FFFFFF

Při pokusu o čtení nebo zápis mimo tento dostupný prostor je nastaven S1.0 = log.0 a současně je nastaven i příslušný chybový kód v S34.

Při simulaci v uživatelském programu instrukce RDB a WDB provádí čtení resp. zápis do souboru databox.\$\$. Pokud nelze tento soubor otevřít nebo vznikne chyba při čtení resp. zápisu do tohoto souboru, nastaví instrukce při simulaci S1.0 = log.0 a zároveň A0 = 0.

Příklad použití

Instrukce RDB, WDB a IDB lze použít v uživatelském programu například takto:

```
#reg uint 100,adrDB      ;proměnné, které řídí činnost RDB
#reg uint indR
#reg uint len
#reg bool DataBoxOK      ;příznak DataBox v pořádku
;
P 63
:
LD 32                    ;požadovaná velikost DataBoxu v aplikaci
IDB                       ;identifikace velikosti DataBoxu
GT
NEG                       ;DataBox aspoň požadované velikosti?
WR DataBoxOK             ;nastavit příznak
:
E 63
;
P 0
:
LD DataBoxOK             ;DataBox v pořádku ?
JMC konecDBX            ;ne
LD $FC00                 ;adresa v DataBoxu (model 32 bitů)
;!! LDL $FC00            ;adresa v DataBoxu (model 16 bitů - long !!!)
WR adrDB
LD 200                   ;do kterého registru se přenesou
WR indR                  ;data z DataBoxu
LD 56                    ;počet přenášených bytů
WR len
LD 100                   ;číslo registru, kde leží parametry
RDB                      ;čtení bloku dat z DataBoxu do zápisníku blok
                           ;o délce 56 bytů se přečte z adresy $FC00
                           ;a uloží se od R200
```

konecDBX:

:

E 0

nebo lépe

```
#struct parDB          ;jméno struktury
    uint   adrDB,      ;adresa v DataBoxu
    uint   indR,       ;index počátečního registru v zápisníku
    usint   len        ;počet přenášených bytů
#reg parDB parusi
#def lenDat 56
#reg usint blokDat[lenDat]
#reg bool   DataBoxOK   ;příznak DataBox v pořádku
;
P 63
:
LD 32                ;požadovaná velikost DataBoxu v aplikaci
IDB                ;identifikace velikosti DataBoxu
GT
NEG                ;DataBox aspoň požadované velikosti?
WR DataBoxOK        ;nastavit příznak
:
E 63
;
P 0
:
LD DataBoxOK        ;DataBox v pořádku?
JMC konecDBX        ;ne
LD $FC00            ;adresa v DataBoxu (model 32 bitů)
;!! LDL $FC00        ;adresa v DataBoxu (model 16 bitů - long !!!)
WR parusi~adrDB
LD __indx (blokDat);do kterého registru se přenesou data
                        ;z DataBoxu

WR parusi~indR
LD lenDat           ;počet přenášených bytů
WR parusi~len
LD __indx (parusi)  ;číslo registru, kde leží parametry
RDB                ;čtení bloku dat z DataBoxu do zápisníku blok
                        ;o délce 56 bytů se přečte z adresy $FC00
                        ;a uloží se do pole blokDat

konecDBX:
:
E 0
```

Uživatelské instrukce

Pokud máme k dispozici starší centrální jednotky, které nemají instrukce RDB, WDB a IDB implementovány, lze použít uživatelské instrukce READDBX, WRITEDBX a SIZEDBX (pojem uživatelské instrukce viz kap.12). Jejich funkce je shodná s odpovídajícími instrukcemi včetně simulace v prostředí Mosaic.

Definice uživatelských instrukcí

Uživatelské instrukce je na začátku uživatelského programu nutné definovat:

```
#usi u_readdbx = readdbx      ;cesta a jméno souboru
#usi u_writedb x = writedb x  ;cesta a jméno souboru
#usi u_sizedbx = sizedbx      ;cesta a jméno souboru
#def RDB usi u_readdbx        ;pojmenování USI
#def WDB usi u_writedb x      ;pojmenování USI
#def IDB usi u_sizedbx        ;pojmenování USI
```

Těchto šest řádků vložíme do uživatelského programu bezprostředně na začátek. Tím jsme do PLC doplnili chybějící funkce. Jinak zůstává uživatelský program beze změn.

8. ZÁSObNÍK VÝSLEDKŮ

Model zásobníku

Při vykonávání uživatelského programu pracuje PLC se zásobníkem, který má 8 úrovní označených A0 až A7 (akumulátor, střadač výsledků). Aktivní úroveň A0 označovaná také jako vrchol zásobníku je využita v naprosté většině instrukcí.

PLC TECOMAT mají dva modely zásobníku, které se od sebe liší šířkou jedné vrstvy. Řady B, D, E, M a S mají jednotlivé vrstvy zásobníku široké 16 bitů (obr.8.1), zatímco řada C má vrstvy zásobníku široké 32 bitů (obr.8.2). Z toho plynou určité rozdíly mezi chováním jednotlivých modelů.

Se zásobníkem pracují logické operace, aritmetické operace, přenosové operace, předávají se v něm logické a číselné parametry složitějších instrukcí a podprogramů.

Zásobník je cyklický (obr.8.3), můžeme si jej představit jako bubnovou paměť podle obr.8.4.

Osmice přepínatelných zásobníků

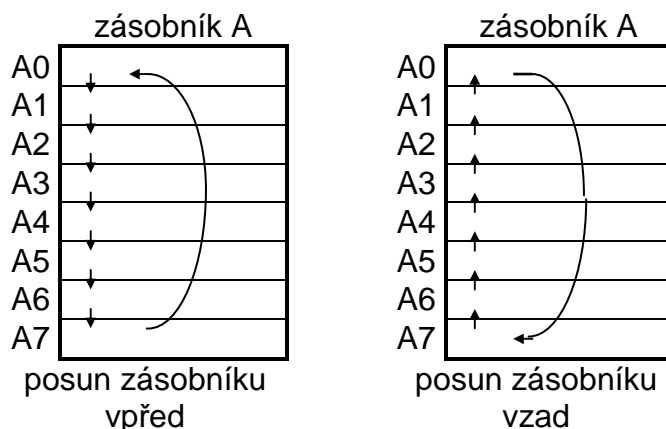
Těchto zásobníků máme k dispozici celkem 8 (kromě centrálních jednotek řady E, kde je jen 1 zásobník) označovaných písmeny A až H. Aktivní je vždy jeden a můžeme mezi nimi přepínat (obr.8.4). Tím jsou otevřeny velké možnosti v oblasti přenosu parametrů mezi funkcemi v uživatelském programu bez nutnosti méně přehledného meziukládání parametrů do zápisníku.

dword	udint	word	horní byte	dolní byte
dint	real	uint	int	
			bit 15 ... 8	bit 7 ... 0
A01		A0		
		A1		
A23		A2		
		A3		
A45		A4		
		A5		
A67		A6		
		A7		

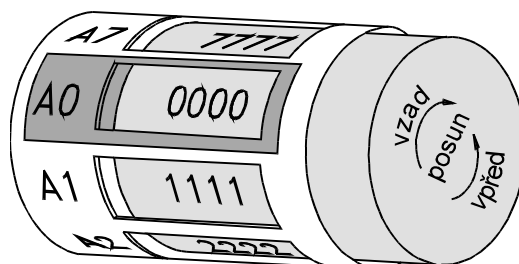
Obr.8.1 Schématické znázornění struktury zásobníku A modelu 16 bitů

dword		udint	horní word				dolní word							
lreal	dint	real	nejvyšší byte				nejnižší byte							
			bit 31	...	24	bit 25	...	16	bit 15	...	8	bit 7	...	0
A01		A0												
		A1												
A23		A2												
		A3												
A45		A4												
		A5												
A67		A6												
		A7												

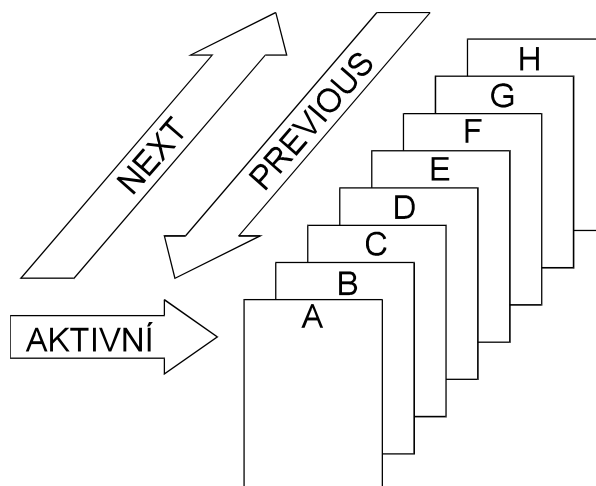
Obr.8.2 Schématické znázornění struktury zásobníku A modelu 32 bitů



Obr.8.3 Funkce cyklického zásobníku A



Obr.8.4 Představa zásobníku A jako bubnové paměti



Obr.8.5 Přepínání zásobníků

8.1. STRUKTURA ZÁSObNÍKU

Vrchol zásobníku

Vrchol zásobníku (A0, resp. A01) je aktivní vrstvou, která má význam střadače (akumulátoru). Další vrstvy (A1 až A7, resp. A23 až A67) obsahují postupně sled předchozích hodnot vrcholu zásobníku.

Posun zásobníku vpřed

Posun zásobníku vpřed způsobují instrukce čtení (LD, LDC, ...) a některé složitější instrukce. Při každém posunu zásobníku vpřed o jednu úroveň jsou hodnoty všech jeho vrstev A0 až A6 přesunuty do vrstev s čísly o jedničku vyššími a vrchol zásobníku A0 je obsazen nově ukládanou hodnotou dle následujícího postupu:

$A0 \leftarrow \text{nová data}$
 $A1 \leftarrow \text{původní obsah } A0$
 $A2 \leftarrow \text{původní obsah } A1$

 $A7 \leftarrow \text{původní obsah } A6$
 Původní obsah $A7$ se nenávratně ztrácí (je přepsán novým obsahem $A0$).

Posun zásobníku vzad

Posun zásobníku vzad provádí instrukce POP a bezoperandové instrukce aritmetických a logických operací. Zpětný posun probíhá dle následujícího postupu:

$A0 \leftarrow \text{původní obsah } A1 \text{ nebo výsledek operace mezi } A0 \text{ a } A1$
 $A1 \leftarrow \text{původní obsah } A2$
 $A2 \leftarrow \text{původní obsah } A3$

 $A7 \leftarrow \text{původní obsah } A0$

8.2. INTERPRETACE DAT NA ZÁSObNÍKU - MODEL 16 BITŮ

Každá vrstva zásobníku má šíři 16 bitů (2 byty). Celou vrstvu označujeme $A0, A1, \dots, A7$ (obr.8.1). Data typů dword, uint, dint a real zabírají dvě vrstvy. Hovoříme pak o tzv. dvojvrstvě označované $A01, A23, A45, A67$.

8.2.1. Data typu bool - model 16 bitů

Pokud načítáme na vrchol zásobníku data typu bool (např. LD %R0.1), pak hodnota bitu (0 nebo 1) je uložena na všech šestnácti bitech vrstvy $A0$ ($A0 = 0$ nebo $A0 = 65535$).

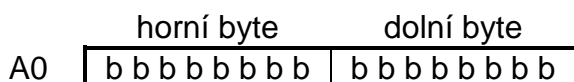
Naopak pokud ukládáme data typu bool (např. WR %R8.5), pak v případě nulové hodnoty $A0$ ukládáme nulovou hodnotu bitu, v případě libovolné nenulové hodnoty $A0$ ukládáme jedničku. Ukládáme tedy podélný logický součet (OR) všech šestnácti bitů vrcholu $A0$.

Tento princip umožňuje snadnou formátovou konverzi, lze tedy kombinovat instrukce s operandy typů bool, byte, usint, sint, word, uint a int prakticky bez omezení.

Pokud tedy mluvíme o obsahu vrcholu zásobníku jako o hodnotě typu bool, budeme dále používat následující označení:

log.0 logická nula, $A0 = 0$
 log.1 logická jednička, $A0 \neq 0$

Vrcholem zásobníku rozumíme celou vrstvu $A0$.



Obr.8.6 *Uložení dat typu bool na vrcholu zásobníku*
b - logická hodnota bitu (log.0 nebo log.1)

8.2.2. Data typů byte, usint, sint - model 16 bitů

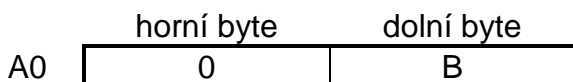
Pokud načítáme na vrchol zásobníku data typů byte, usint, sint (např. LD %R0), pak hodnota tohoto bytu je uložena v dolním bytu vrstvy A0, horní byte vrstvy A0 je vynulován.

Naopak pokud ukládáme data typů byte, usint, sint (např. WR %R8), pak je na cílovou adresu uložen pouze dolní byte vrstvy A0. Horní byte je ignorován.

Tento princip umožňuje snadnou formátovou konverzi u kladných hodnot, lze tedy kombinovat instrukce s operandy typů bool, byte, usint, word a uint prakticky bez omezení. Pro záporné hodnoty (typy sint, int) je nutné ošetřit přenos znaménka.

Data nabývají hodnot 0 až 255 (byte, usint) nebo při použití nejvyššího bitu jako znaménka –128 až +127 (sint).

Vrcholem zásobníku rozumíme celou vrstvu A0.



Obr.8.7 Uložení dat typů byte, usint, sint na vrcholu zásobníku
B - hodnota bytu

8.2.3. Data typů word, uint, int - model 16 bitů

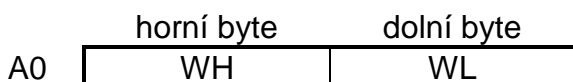
Data typů word, uint, int zabírají 2 byty.

Pokud načítáme na vrchol zásobníku data typů word, uint, int (např. LD %RW0), pak je těmito daty naplněna právě celá vrstva A0. Obsah registru s nižším číslem je uložen do dolního bytu vrstvy A0, obsah registru s vyšším číslem je uložen do horního bytu vrstvy A0.

Naopak pokud ukládáme data typů word, uint, int (např. WR %RW8), pak je na cílovou adresu uložen celý obsah vrstvy A0. Obsah dolního bytu vrstvy A0 je uložen do registru s nižším číslem, obsah horního bytu vrstvy A0 je uložen do registru s vyšším číslem.

Data nabývají hodnot 0 až 65 535 (word, uint) nebo při použití nejvyššího bitu jako znaménka –32 768 až +32 767 (int).

Vrcholem zásobníku rozumíme celou vrstvu A0.



Obr.8.8 Uložení dat typů word, uint a int na vrcholu zásobníku
WH - hodnota vyššího bytu
WL - hodnota nižšího bytu

8.2.4. Data typů dword, udint, dint - model 16 bitů

Data typů dword, udint, dint zabírají 4 byty.

Pokud načítáme na vrchol zásobníku data typů dword, udint, dint (např. LD %RL0), pak jsou těmito daty naplněny celé dvě vrstvy A0 a A1, které dohromady tvoří dvojvrstvu A01. Obsah registru s nejnižším číslem je uložen do dolního bytu vrstvy A0, obsah registru s nejvyšším číslem je uložen do horního bytu vrstvy A1.

Naopak pokud ukládáme data typu dword, udint, dint (např. WR %RL8), pak je na cílovou adresu uložen celý obsah dvojvrstvy A01. Obsah dolního bytu vrstvy A0 je uložen do registru s nejnižším číslem, obsah horního bytu vrstvy A1 je uložen do registru s nejvyšším číslem.

8. Zásobník výsledků

Data nabývají hodnot 0 až 4 294 967 295 (dword, uint) nebo při použití nejvyššího bitu jako znaménka –2 147 483 648 až +2 147 483 647 (dint).

Vrcholem zásobníku rozumíme celou dvojvrstvu A01.

		horní byte	dolní byte
A01	A0	L1	L0
	A1	L3	L2

Obr.8.9 Uložení dat typů dword, uint a dint na vrcholu zásobníku
L0 - hodnota nejnižšího bytu
:
L3 - hodnota nejvyššího bytu

8.2.5. Data typu real - model 16 bitů

Data typu real mají šířku 4 byty stejně jako dword. Platí tedy pro ně stejná výše uvedená pravidla.

Data typu real podle IEEE-754 nabývají hodnot v rozsahu cca $\pm 1,175494 \times 10^{-38}$ až $\pm 3,402823 \times 10^{38}$. Dále existují čtyři hodnoty označující následující stavy:

\$7FFFFFFF	- neplatné číslo (NaN - not a number)
\$FFFFFFFF	- neplatné číslo (NaN - not a number)
\$7F800000	- překročení rozsahu kladných čísel (+INF)
\$FF800000	- překročení rozsahu záporných čísel (–INF)

Vrcholem zásobníku rozumíme celou dvojvrstvu A01.

		horní byte	dolní byte
A01	A0	mmmmmmmm	mmmmmmmm
	A1	s e e e e e e e	e mmmmmmm

Obr.8.10 Uložení dat typu real na vrcholu zásobníku
s - znaménko (1 bit)
e - exponent (8 bitů)
m - mantisa (23 bitů)

8.3. INTERPRETACE DAT NA ZÁSObNÍKU - MODEL 32 BITŮ

Každá vrstva zásobníku má šíři 32 bitů (4 byty). Celou vrstvu označujeme A0, A1, ... A7 (obr.8.2). Data typu lreal zabírají dvě vrstvy. Hovoříme pak o tzv. dvojvrstvě označované A01, A23, A45, A67.

8.3.1. Data typu bool - model 32 bitů

Pokud načítáme na vrchol zásobníku data typu bool (např. LD %R0.1), pak hodnota bitu (0 nebo 1) je uložena na všech 32 bitech vrstvy A0 (A0 = 0 nebo A0 = 4 294 967 295).

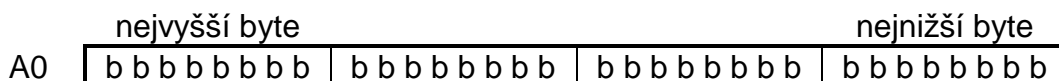
Naopak pokud ukládáme data typu bool (např. WR %R8.5), pak v případě nulové hodnoty A0 ukládáme nulovou hodnotu bitu, v případě libovolné nenulové hodnoty A0 ukládáme jedničku. Ukládáme tedy podélný logický součet (OR) všech 32 bitů vrcholu A0.

Tento princip umožňuje snadnou formátovou konverzi, lze tedy kombinovat instrukce s operandy typu bool, byte, usint, sint, word, uint, int, dword, udint a dint prakticky bez omezení.

Pokud mluvíme o obsahu vrcholu zásobníku jako o hodnotě typu bool, budeme dále používat následující označení:

log.0 logická nula, $A0 = 0$
 log.1 logická jednička, $A0 \neq 0$

Vrcholem zásobníku rozumíme celou vrstvu A0.



Obr.8.11 Uložení dat typu bool na vrcholu zásobníku
b - logická hodnota bitu (log.0 nebo log.1)

8.3.2. Data typů byte, usint, sint - model 32 bitů

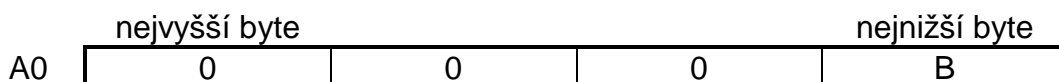
Pokud načítáme na vrchol zásobníku data typů byte, usint, sint (např. LD %R0), pak hodnota tohoto bytu je uložena v nejnižším bytu vrstvy A0, ostatní tři byty vrstvy A0 jsou vynulovány.

Naopak pokud ukládáme data typů byte, usint, sint (např. WR %R8), pak je na cílovou adresu uložen pouze nejnižší byte vrstvy A0. Ostatní byty jsou ignorovány.

Tento princip umožňuje snadnou formátovou konverzi u kladných hodnot, lze tedy kombinovat instrukce s operandy typu bool, byte, usint, word, uint, dword a udint prakticky bez omezení. Pro záporné hodnoty (typy sint, int, dint) je nutné ošetřit přenos znaménka.

Data nabývají hodnot 0 až 255 (byte, usint) nebo při použití nejvyššího bitu jako znaménka -128 až +127 (sint).

Vrcholem zásobníku rozumíme celou vrstvu A0.



Obr.8.12 Uložení dat typů byte, usint a sint na vrcholu zásobníku
B - hodnota bytu

8.3.3. Data typů word, uint, int - model 32 bitů

Data typů word, uint, int zabírají 2 byty.

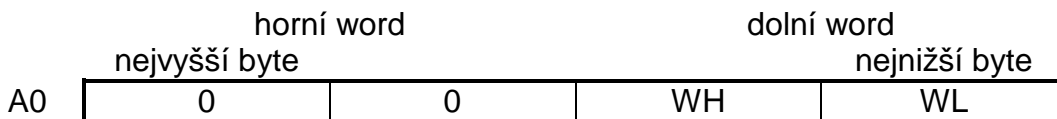
Pokud načítáme na vrchol zásobníku data typů word, uint, int (např. LD %RW0), pak hodnota tohoto wordu je uložena v dolním wordu vrstvy A0, horní word vrstvy A0 je vynulován. Obsah registru s nižším číslem je uložen do nejnižšího bytu vrstvy A0, obsah registru s vyšším číslem je uložen do druhého nejnižšího bytu vrstvy A0.

Naopak pokud ukládáme data typů word, uint, int (např. WR %R8), pak je na cílovou adresu uložen pouze dolní word vrstvy A0. Horní word je ignorován. Obsah nejnižšího bytu vrstvy A0 je uložen do registru s nižším číslem, obsah druhého nejnižšího bytu vrstvy A0 je uložen do registru s vyšším číslem.

Tento princip umožňuje snadnou formátovou konverzi u kladných hodnot, lze tedy kombinovat instrukce s operandy typu bool, byte, usint, word, uint, dword a udint prakticky bez omezení. Pro záporné hodnoty (typy sint, int, dint) je nutné ošetřit přenos znaménka.

Data nabývají hodnot 0 až 65 535 (word, uint) nebo při použití nejvyššího bitu jako znaménka –32 768 až +32 767 (int).

Vrcholem zásobníku rozumíme celou vrstvu A0.



Obr.8.13 Uložení dat typů word, uint a int na vrcholu zásobníku

WH - hodnota vyššího bytu

WL - hodnota nižšího bytu

8.3.4. Data typů dword, uint, dint - model 32 bitů

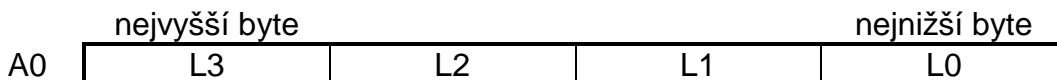
Data typů dword, uint, dint zabírají 4 byty.

Pokud načítáme na vrchol zásobníku data typů dword, uint, dint (např. LD %RL0), pak je těmito daty naplněna právě celá vrstva A0. Obsah registru s nejnižším číslem je uložen do nejnižšího bytu vrstvy A0, obsah registru s nejvyšším číslem je uložen do nejvyššího bytu vrstvy A0.

Naopak pokud ukládáme data typů dword, uint, dint (např. WR %RL8), pak je na cílovou adresu uložen celý obsah vrstvy A0. Obsah nejnižšího bytu vrstvy A0 je uložen do registru s nejnižším číslem, obsah nejvyššího bytu vrstvy A0 je uložen do registru s nejvyšším číslem.

Data nabývají hodnot 0 až 4 294 967 295 (dword, uint) nebo při použití nejvyššího bitu jako znaménka –2 147 483 648 až +2 147 483 647 (dint).

Vrcholem zásobníku rozumíme celou vrstvu A0.



Obr.8.14 Uložení dat typů dword, uint a dint na vrcholu zásobníku

L0 - hodnota nejnižšího bytu

:

L3 - hodnota nejvyššího bytu

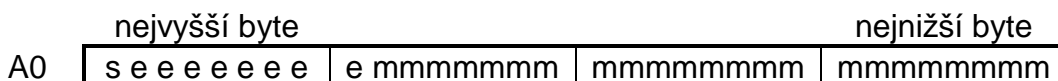
8.3.5. Data typu real - model 32 bitů

Data typu real mají šířku 4 byty stejně jako dword. Platí tedy pro ně stejná výše uvedená pravidla.

Data typu real podle IEEE-754 nabývají hodnot v rozsahu cca $\pm 1,175494 \times 10^{-38}$ až $\pm 3,402823 \times 10^{38}$. Dále existují čtyři hodnoty označující následující stavy:

- \$7FFFFFFF - neplatné číslo (NaN - not a number)
- \$FFFFFFFF - neplatné číslo (NaN - not a number)
- \$7F800000 - překročení rozsahu kladných čísel (+INF)
- \$FF800000 - překročení rozsahu záporných čísel (–INF)

Vrcholem zásobníku rozumíme celou vrstvu A0.



Obr.8.15 Uložení dat typu real na vrcholu zásobníku

s - znaménko (1 bit)
e - exponent (8 bitů)
m - mantisa (23 bitů)

8.3.6. Data typu lreal - model 32 bitů

Data typu lreal mají šířku 8 btů.

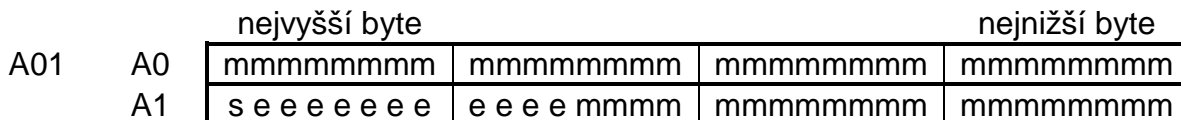
Pokud načítáme na vrchol zásobníku data typu lreal (např. LD %RD0), pak jsou těmito daty naplněny celé dvě vrstvy A0 a A1, které dohromady tvoří dvojvrstvu A01. Obsah registru s nejnižším číslem je uložen do dolního bytu vrstvy A0, obsah registru s nejvyšším číslem je uložen do horního bytu vrstvy A1.

Naopak pokud ukládáme data typu lreal (např. WR %RD8), pak je na cílovou adresu uložen celý obsah dvojvrstvy A01. Obsah dolního bytu vrstvy A0 je uložen do registru s nejnižším číslem, obsah horního bytu vrstvy A1 je uložen do registru s nejvyšším číslem.

Data typu lreal podle IEEE-754 nabývají hodnot v rozsahu cca $\pm 2,2 \times 10^{-308}$ až $\pm 1,8 \times 10^{308}$. Dále existují čtyři hodnoty označující následující stavy:

\$7FFFFFFF FFFFFFFF	- neplatné číslo (NaN - not a number)
\$FFFFFFFF FFFFFFFF	- neplatné číslo (NaN - not a number)
\$7FF00000 00000000	- překročení rozsahu kladných čísel (+INF)
\$FFF00000 00000000	- překročení rozsahu záporných čísel (-INF)

Vrcholem zásobníku rozumíme celou dvojvrstvu A01.



Obr.8.16 Uložení dat typu lreal na vrcholu zásobníku

s - znaménko (1 bit)
e - exponent (11 bitů)
m - mantisa (52 bitů)

8.4. PŘEPÍNÁNÍ ZÁSObNÍKŮ

K dispozici máme 8 zásobníků značených A, B, C, D, E, F, G, H, z nichž každý má takovou strukturu, jak byla popsána pro zásobník A v kap.8.1. Aktivní je vždy jeden zásobník, ostatní jsou uchovány v tom stavu, v jakém se nacházely při jejich opuštění uživatelským programem, a nemá na ně vliv žádný stav uživatelského programu. Například při otevření nového uživatelského procesu je nulován pouze aktivní zásobník (kap.10.).

Po zapnutí nebo restartu PLC jsou všechny zásobníky vynulovány a aktivní je vždy zásobník A. Po otočce cyklu je vždy vynulován zásobník A a je nastaven jako aktivní.

Kdykoliv můžeme zásobníky přepnout, tzn. že dosud aktivní zásobník bude uživatelským programem opuštěn a nadále bude program pracovat s nově zvoleným zásobníkem. Přitom se zálohuje i stav příznakových registrů S0 a S1.

Instrukce pro přepínání zásobníků

K přepínání zásobníků slouží čtyři instrukce:

- NXT - aktivace následujícího zásobníku v řadě (obr.8.5)
např. byl-li aktivní zásobník C, bude nyní aktivován zásobník D
- PRV - aktivace předcházejícího zásobníku v řadě (obr.8.5)
např. byl-li aktivní zásobník C, bude nyní aktivován zásobník B
- CHG - aktivace libovolného zásobníku
např. CHG 7 aktivuje zásobník F, stav příznakových registrů S0 a S1 se nemění
- CHGS- aktivace libovolného zásobníku se zálohováním S0 a S1
např. CHGS 7 uloží stav systémových registrů S0 a S1 k dosud aktivnímu zásobníku a aktivuje zásobník F, v systémových registrech S0 a S1 obnoví stav odpovídající jejich stavu v okamžiku, kdy tehdy aktivní zásobník F byl opuštěn pomocí instrukce CHGS n, kde n je 0 až 7 (odpovídá A až H), nebo pomocí instrukcí NXT a PRV

9. DIREKTIVY PŘEKLADAČE

Direktivy jsou příkazy pro překladač umožňující optimálně řídit překlad. Všechny direktivy začínají znakem #. V překladači xPRO, který obsahuje prostředí Mosaic, jsou k dispozici následující direktivy:

```
#program  
#unit, #module  
#include, #usefile  
#def  
#reg, #rem  
#data, #table  
#struct  
#if, #elif, #else, #endif  
#ifdef, #ifndef  
#macro, #endm  
#label  
#usi  
#mnemo, #mnemoend  
#useoption
```

9.1. #program

Direktiva *#program* označuje začátek programu.

Syntaxe je následující:

```
#program jméno [, [V][xyz]]
```

<i>jméno</i>	- vlastní jméno programu, maximálně 16 znaků (delší jméno je zkráceno na tuto délku)
<i>V</i>	- nepovinný prefix verze programu
<i>xyz</i>	- tři nepovinné znaky charakterizující verzi programu

V prostředí Mosaic uživatel nezapisuje tuto direktivu ručně, ale má k dispozici v manažeru projektu ve složce *Sw / Program* celý formulář, kde vyplní jméno programu, jeho verzi a dále zde může zapsat i další údaje o aplikaci a historii změn v uživatelském programu. Tyto informace jsou součástí projektu a hodí se zejména v situaci pozdějšího návratu k tomuto uživatelskému programu. Pokud uživatel tento formulář nevyplní, přednastavené jméno odpovídá jménu projektu v rámci skupiny projektů a verze je 1.0. Na základě těchto informací pak překladač sám vygeneruje hlavičku programu, která se nachází v řídicím souboru *xxx.mak*, kde *xxx* je jméno projektu v rámci skupiny projektů.

Informace o jménu a verzi uživatelského programu, který právě PLC vykonává, lze z něj zpětně kdykoliv vyčíst, vybereme-li volbu *PLC / Informace o PLC*. Z toho důvodu doporučujeme využívat především verzi programu tak, že při každé změně v programu se změní i jeho verze. Častým problémem v běžné praxi je totiž zjistit, jaká varianta programu je v dané chvíli do PLC nahraná. Důsledné číslování verzí tedy předchází potížím při pozdějších úpravách programu.

9.2. #unit, #module

Pomocí direktiv *#unit* nebo *#module* je vytvářen seznam periferních zařízení potřebný pro správný chod uživatelského programu. Na základě zjištěné konfigurace je překladačem vytvořena konfigurace PLC (viz příručky k PLC).

Syntaxe je následující:

```
#unit    rám, adresa, typ, obraz_X, obraz_Y, aktivace[, tab]
#module TModuleI verze, rám, adresa, lokalizace, délka_X, délka_Y,
__offset (obraz_X), __offset (obraz_Y), __indx (tab)
```

<i>verze</i>	- číslo varianty popisu (vždy 1)
<i>rám</i>	- číslo rámu, ve kterém je modul instalován
<i>adresa</i>	- adresa modulu v rámu (nastavitelná propojkami nebo daná pozicí)
<i>lokalizace</i>	- číselný kód udávající lokalizaci popisované části modulu
<i>typ</i>	- číselný kód charakterizující typ modulu Tento číselný kód se nepoužívá přímo, ale prostřednictvím symbolických jmen definovaných standardně v souboru <i>xpro.sys</i> .
<i>délka_X</i>	- délka vstupních dat definovaného modulu
<i>délka_Y</i>	- délka výstupních dat definovaného modulu
<i>obraz_X</i>	- operand (nemusí být nutně v oblasti X) udávající, do kterých registrů jsou přenášena data ze vstupů definovaného modulu
<i>obraz_Y</i>	- operand (nemusí být nutně v oblasti Y) udávající, ze kterých registrů jsou přenášena data na výstupy definovaného modulu
<i>aktivace</i>	- kód (opět nahrazovaný symbolickým jménem definovaným v souboru <i>xpro.sys</i>) udávající režim modulu Může nabývat hodnot: <i>X_on</i> = vstupy povoleny (jsou kopírovány do <i>obraz_X</i>) <i>Y_on</i> = výstupy povoleny (jsou kopírovány z <i>obraz_Y</i>) <i>On</i> = vstupy i výstupy jsou povoleny <i>Off</i> = vstupy i výstupy jsou zakázány
<i>tab</i>	- nepovinný parametr udávající jméno tabulky (viz <i>#table</i>) s údaji potřebnými k inicializaci jednotky Tuto inicializaci používají některé speciální druhy periferních modulů, systémové sériové kanály, apod.

V prostředí Mosaic uživatel nedefinuje periferní zařízení ručním zápisem direktivy *#unit*, ale pomocí vyplnění tabulek v konfigurační sekci (v manažeru projektu, položka *Hw / Konfigurace HW*). Překladač pak podle těchto podkladů vytvoří samostatný soubor *xxx.hwc*, kde *xxx* je jméno projektu, obsahující jak direktivy *#unit*, tak příslušné inicializační tabulky. Tento soubor je automaticky připojen k uživatelskému programu pomocí direktivy *#usefile* (viz dále) v řídicím souboru *xxx.mak*. Pokud v konfigurační sekci zaškrtneme volbu *Potlačit direktivu #UNIT*, direktivy *#unit* budou v souboru uvedeny pouze jako komentáře (budou začínat znakem středník).

PLC TECOMAT TC650 a TC700 místo direktivy *#unit* používají direktivu *#module*.

V prostředí Mosaic lze z připojeného PLC načíst informace, podle kterých dojde k automatickému naplnění konfigurace. Tyto údaje pak může uživatel upravit podle svých potřeb a na jejich základě bude při překladu vytvořen konfigurační soubor *xxx.hwc*.

9.3. #include, #usefile

Při psaní dlouhých programů je z hlediska přehlednosti výhodné uspořádat celý program do několika menších, logicky kompaktních souborů. Toto tzv. řetězení souborů umožňuje direktiva *#include*.

Syntaxe je následující:

#include *jméno_souboru*

jméno_souboru - označuje soubor, který je třeba připojit k právě překládanému programu. Pokud neleží v aktuálním pracovním adresáři, je nutné zadat i cestu k němu.

Když překladač nalezne ve zdrojovém textu programu direktivu *#include*, přeruší překládání v současném souboru a pokračuje v překládání souboru uvedeného za direktivou *#include*. Po přeložení celého tohoto souboru pokračuje v překládání textu z původního souboru. Překladač zvládá bez problémů i situace, kdy soubor vložený direktivou *#include*, obsahuje další direktivy *#include*.

V prostředí Mosaic je výhodnější použití řetězení souborů pomocí projektu. Každý soubor obsažený v projektu je zařazen do řídicího souboru *xxx.mak* v daném pořadí pomocí direktivy *#usefile*. Tato direktiva je automaticky generována překladačem a není určena pro použití uživatelem. Přidáme-li do projektu další soubor, bude automaticky při překládání připojen v řídicím souboru. Pořadí překládání souborů je dáno jejich pořadím v seznamu a lze jej měnit pomocí přesouvacích šipek.

Syntaxe je následující:

#usefile *jméno_souboru*

jméno_souboru - označuje soubor, který je zařazen do překládaného projektu

V prostředí Mosaic je řetězení souborů usnadněno tím, že překladač podporuje tzv. skládání procesů. To znamená, že pokud v jednom souboru píšeme instrukce v rámci procesu P0 (tedy začínáme instrukcí P 0 a končíme instrukcí E 0) a v jiném souboru píšeme instrukce opět v rámci procesu P0 (tedy opět začínáme instrukcí P 0 a končíme instrukcí E 0), překladač pak výsledně vytvoří jeden proces P0, ve kterém sloučí obě části (vynechá přebytečné instrukce E 0 a P 0) v pořadí, v jakém jsou soubory řazeny v projektu.

Z těchto důvodů ale doporučujeme **nepoužívat** instrukce EC a ED, které způsobí předčasné ukončení procesu, tedy nevykonání těch částí programu, které jsou připojeny překladačem (může jít i o důležité ovladače generované prostředím Mosaic). Skok na konec procesu v rámci jednoho souboru řešíme pomocí skoku na návěští.

9.4. #def

Pomocí direktivy *#def* se definují symbolická jména, která lze potom v programu používat např. místo absolutních operandů.

Syntaxe je následující:

#def *symbolické_jméno* *dosazené_jméno*

symbolické_jméno - posloupnost složená z libovolného počtu povolených znaků (viz direktiva *#program*)

dosazené_jméno - jiný libovolný řetězec složený z povolených znaků (absolutní operand, konstanta atd.)

Direktiva *#def* se může objevit na kterémkoliv místě programu po jeho záhlaví, vždy však před prvním použitím definovaného symbolického jména. Platí tedy obecná zásada, že nejprve je nutno definovat a poté lze definovaný objekt použít.

Příklad 9.1

```
#def Povolovací_Bit    %R5.0           ;definice bitu
#define KONSTANTA      80              ;zpoždění
#define PRODLEVA       KONSTANTA+20
#define sec            2                ;typ časovače 1 s
#define Casovac_T1     %RW0.sec        ;časovač 1 s
#define vystupni_word  %YW0            ;výstup
```

9.5. #reg, #rem

Direktiva *#reg* slouží k tzv. automatické definici proměnných z oblasti R. Pomocí této direktivy přidělíme symbolické jméno proměnné a zároveň požádáme překladač, aby pro tuto proměnnou rezervoval místo v registrech R zápisníkové paměti PLC. Počet registrů, který se pro proměnnou vymezí, se řídí požadovaným datovým typem.

Direktiva *#rem* slouží k tzv. automatické definici proměnných z remanentní oblasti R. Pomocí této direktivy přidělíme symbolické jméno proměnné a zároveň požádáme překladač, aby pro tuto proměnnou rezervoval místo v remanentních registrech R zápisníkové paměti PLC. Počet registrů, který se pro proměnnou vymezí, se řídí požadovaným datovým typem. Pokud se proměnná do remanentní zóny již nevejde, překladač na tuto skutečnost upozorní vyhlášením chyby a nabídne automatickou změnu velikosti remanentní zóny. Pokud velikost remanentní zóny již nebyla nastavena na maximum, při opakovaném překladu je vše v pořádku.

Syntaxe je následující:

```
#reg [public] [aligned] typ [index,]
                                jméno_prom[[délka_pole]][,další_prom...]
#rem [public] [aligned] typ [index,]
                                jméno_prom[[délka_pole]][,další_prom...]
```

- public* - nepovinný příkaz ke generování do seznamu veřejných proměnných pro vizualizace pomocí nastavení v manažeru projektu složka *Sw / Překladač položka Generovat soubor / PUBLIC*
- aligned* - nepovinný příkaz pro přiřazení proměnných na sudý registr, resp. bit 0
- typ* - určuje datový typ proměnné přiřazované symbolickému jménu *jméno_prom*
Povolené typy přiřazované proměnné jsou uvedeny v tab.9.1.

Tab.9.1 Povolené typy přiřazované proměnné

Typ	Rozsah	Příklad
bool	R0.0 - Rmax.7*	R10.2, R2.15
byte, usint, sint	R0 - Rmax*	R1, R2001
word, uint, int	R0 - Rmax-1*	RW2, RW2002
dword, udint, dint	R0 - Rmax-4*	RL4, RL2004
real	R0 - Rmax-4*	RF8, RF2008
lreal	R0 - Rmax-8*	RD12, RD2012
struktura**		

* Rmax je dáno řadou použité centrální jednotky.

** Struktura je jméno (tag) struktury definované pomocí direktivy *#struct*.

jméno_prom, *další_prom* - symbolická jména definovaných registrů
délka_pole - nepovinný parametr uzavřený do hranatých závorek [], udává velikost pole ve formátu odpovídajícímu parametru *typ*
index - nepovinný parametr, udává číslo registru přiřazené symbolickému jménu *jméno_prom* v oblasti R

Překladač automaticky přiděluje indexy v kontinuálně stoupajícím pořadí. Uvedením nepovinného parametru *index* se nastaví vnitřní počítadlo na jeho hodnotu, která je potom jako index přidělena prvnímu symbolickému jménu seznamu (*jméno_prom*). Pro další jména proměnných (*další_prom*) v seznamu přidělování pokračuje od takto nastaveného indexu. Nejvhodnější je tyto ručně indexované operandy umístit na začátek definic registrů.

Na jedno použití *#reg* a *#rem* lze vytvořit libovolný počet operandů, jména jsou v tomto případě oddělena čárkami. Pokud se jména nevejdou na jeden řádek, lze pokračovat na dalším řádku.

Používání direktiv *#reg* a *#rem* zbavuje programátora namáhavé práce s přidělováním indexů proměnným, při kterém lze snadno udělat chybu.

O skutečně přiřazených hodnotách se lze přesvědčit po provedeném překladu programu jednak prostřednictvím příkazu *Zobrazit / Symboly* a nebo z vytvořeného souboru výpisu programu, jehož součástí je i tabulka symbolických jmen.

Pokud v prostředí Mosaic v manažeru projektu ve složce *Sw / Překladač* zaškrtneme položku *Generovat soubor / Mapa registrů*, najdeme po překladu programu přiřazení registrů v souboru *xxx.map*, kde *xxx* je jméno projektu.

Příklad 9.2

```
#reg bool    Rbit00, Rbit01, Rbit02      ;Rbit00    ... %R0.0
                                           ;Rbit01    ... %R0.1
                                           ;Rbit02    ... %R0.2
#reg usint   Rbyte0, Rbyte1             ;Rbyte0    ... %R1
                                           ;Rbyte1    ... %R2
#reg uint    10, Rword0, Rword1         ;Rword0    ... %RW10
                                           ;Rword1    ... %RW12
#reg udint   Rlong0, Rlong1             ;Rlong0    ... %RL14
                                           ;Rlong1    ... %RL18
#reg real    Rfloat0, Rfloat1           ;Rfloat0    ... %RF22
                                           ;Rfloat1    ... %RF26
#reg lreal   Rdouble0, Rdouble1         ;Rdouble0   ... %RD30
                                           ;Rdouble1   ... %RD38
```

Direktivy *#reg* a *#rem* umožňují deklarovat také jednorozměrná pole. Za jméno proměnné se v tomto případě uvede v hranatých závorkách počet prvků pole.

Příklad 9.3

Definujme proměnnou *pole*, která má 20 prvků typu *word*, první prvek má index 0 a poslední prvek je s indexem 19. Jak je vidět, indexy polí, stejně jako vše ostatní v PLC TECOMAT, začínají od 0.

```
#reg word pole[20]                      ;pole[0]    ... %RW14
                                           ;pole[1]    ... %RW16
                                           ;
                                           ;pole[19]   ... %RW52
```

Pokud chceme v programu načíst hodnotu prvku pole např. s indexem 15, přičíst jedničku a zapsat do prvku s indexem 16, můžeme použít následující zápis.

```
LD    pole[15]
INR
WR    pole[16]
```

Pokud chceme v programu použít bitové pole, ke kterému budeme přistupovat tabulkovými instrukcemi, toto pole **musí začínat na bitu 0!!** K tomu použijeme příkaz *aligned*. Pokud počet bitových položek pole není násobkem 8, doporučujeme nepoužívat bity za tímto polem, které zbývají do celého bytu, pro další bitové proměnné.

```
#reg bool  prom1, prom2          ;%R0.0, %R0.1
#reg aligned bool  pole[12]      ;%R2.0 - %R3.3
#reg aligned bool  prom3, prom4  ;%R4.0, %R5.0
#reg bool  prom5, prom6          ;%R5.1, %R5.2
:
LTB  pole
:
```

9.6. #struct

Direktiva *#struct* se používá k deklaraci datových struktur, což jsou v podstatě nové datové typy odvozené od základních datových typů nebo od dříve deklarovaných struktur. Deklarací struktury tedy nevzniká žádný nárok na obsazení paměti PLC, pouze se zavádí nový datový typ, který lze používat pro automatické přidělování proměnných v direktivě *#reg*, dále při definici tabulek, atd.

Deklarace struktury se skládá ze jména (tag) struktury a typů a jmen jednotlivých členů struktury.

Syntaxe je následující:

```
#struct jméno_str [aligned] typ0[[opak0] člen0[,...]....]
                [aligned] typn[[opakn] členn]
```

<i>jméno_str</i>	- jméno (tag) struktury
<i>aligned</i>	- nepovinné zaokrouhlování indexu člena struktury na sudý index
<i>typ0 - typn</i>	- typy členů struktury (byte, word...) nebo tag jiné struktury
<i>člen0 - členn</i>	- jména členů struktury
<i>opak0 - opakn</i>	- nepovinná opakování členů struktury povinně uzavřená v hranatých závorkách, jsou to celá čísla > 0

Pro definici struktury platí tato pravidla:

- ♦ nevejde-li se definice na jeden řádek, lze pokračovat na novém
- ♦ definice jednotlivých členů jsou odděleny znakem čárka (',')
- ♦ každý člen struktury musí být nějakého typu a musí mít přiděleno jméno, které musí být **jedinečné** v rámci jedné struktury
- ♦ členem struktury může být i jednorozměrné pole daného typu
- ♦ struktury v tabulkách T a datech D jsou inicializovány, členy struktury jsou inicializovány z výčtu hodnot v pořadí, v jakém jsou definovány, chybějící inicializační hodnoty jsou nahrazeny nulami

Příklad 9.4

Jednoduchou strukturu můžeme nadefinovat např. následovně:

```
#struct typCas          ;jméno struktury
  usint Hodina,         ;první člen struktury
  usint Minuta,         ;druhý člen struktury
  usint Sekunda         ;poslední člen struktury
```

Uvedená definice založila vlastně nový datový typ nazvaný *typCas*. Tento typ má délku 3 byty a skládá se z položek *Hodina*, *Minuta* a *Sekunda*. Všechny položky jsou typu *usint*. Nyní můžeme pomocí direktivy *#reg* definovat proměnné, které budou typu *typCas*.

```
#reg typCas CasSpusteni
```

Direktivou *#reg* jsme založili proměnnou *CasSpusteni*, která je typu struktura *typCas* a má položky *Hodina*, *Minuta* a *Sekunda*. Položky jsou typu *byte*. Přístup k proměnné z programu ukazují následující instrukce.

```
;přístup k položkám proměnné CasSpusteni
LD    CasSpusteni~Hodina
LD    CasSpusteni~Minuta
LD    CasSpusteni~Sekunda
```

Pro odkazy na členy struktur v programu platí:

- ♦ člen struktury je oddělen znakem '~' (tilda)
- ♦ vícerozměrné členy lze indexovat indexy 0 až *defSize*–1, kde *defSize* je rozměr členu
- ♦ pokud se neprovede rozvoj až do konečného prvku struktury, je za operand považován nejbližší člen struktury

V programu můžeme samozřejmě založit libovolné množství proměnných, které budou nového typu deklarovaného direktivou *#struct typCas*.

```
;deklarace několika proměnných typu typCas
#reg typCas    Odchod, Prichod, CasObeda
```

Následující řádek ukazuje deklaraci pole proměnných, jejichž typ byl předchozím textu definován direktivou *#struct*.

```
;deklarace pole proměnných typu typCas
#reg typCas    poleCasu[10]           ;pole proměnných
```

K jednotlivým prvkům pole můžeme z programu přistoupit např. následovně:

```
LD    poleCasu[0]~Hodina
LD    poleCasu[0]~Minuta
LD    poleCasu[0]~Sekunda
```

V direktivách *#struct* lze při definici typu jednotlivých položek používat rovněž typy definované předchozími direktivami *#struct*. Jinak řečeno, struktury lze vzájemně vnořovat a vytvářet tak i složité odvozené typy.

Příklad 9.5

```
;definice jednoduché struktury
#struct typCas                ;jméno struktury
    usint Hodina,             ;první člen struktury
    usint Minuta,             ;druhý člen struktury
    usint Sekunda             ;poslední člen struktury
;
;definice další jednoduché struktury
#struct typDatum              ;jméno struktury
    usint Den,                ;první člen struktury
    usint Mesic,              ;druhý člen struktury
    uint  Rok                  ;poslední člen struktury
;
;definice struktury, jejímiž členy jsou jiné struktury (vnoření struktur)
#struct casZnacka              ;jméno struktury
    typCas  Cas,              ;první člen struktury
```

```

    typDatum Datum,          ;druhý člen struktury
    uint      PocetKusu      ;poslední člen struktury
;
#reg casZnacka Davka         ;proměnná typu casZnacka
#reg casZnacka Zaznam[10]    ;pole proměnných typu casZnacka
;
P 0
    LD      Davka~Cas~Hodina
    LD      Davka~Datum~Rok
    LD      Davka~PocetKusu
    :
    LD      Zaznam[1]~Cas~Hodina
    LD      Zaznam[9]~Datum~Rok
    LD      Zaznam[0]~PocetKusu
    :
E 0

```

Příklad 9.6

V některých případech je potřebné, aby položkou struktury bylo pole. Definice takové struktury může vypadat např. následovně:

```

#struct typCas              ;jméno struktury
    usint Hodina,           ;první člen struktury
    usint Minuta,           ;druhý člen struktury
    usint Sekunda           ;poslední člen struktury
;
#struct typDatum            ;jméno struktury
    usint Den,              ;první člen struktury
    usint Mesic,            ;druhý člen struktury
    uint  Rok                ;poslední člen struktury
;
#struct slozitejsiZaznam
    typCas[2] Casy,
    typDatum[2] Datумы,
    uint      Kusy
;
#reg slozitejsiZaznam Obrobek
;
P 0
    LD      Obrobek~Casy[0]~Hodina
    LD      Obrobek~Datумы[1]~Rok
    LD      Obrobek~Kusy
    :
E 0

```

Asi nebude překvapením, že i z takto definovaných struktur lze vytvářet pole proměnných.

```

#reg slozitejsiZaznam Obrobky[3]
;
P 0
    LD      Obrobky[0]~Casy[0]~Hodina
    LD      Obrobky[0]~Datумы[1]~Rok
    LD      Obrobky[0]~Kusy
    :
E 0

```

Na závěr těchto ukázek uveďme ještě konkrétní obsazení registrů v zápisníkové paměti PLC pro tento příklad. Níže uvedený výpis obsazení registrů lze po překladu programu získat v souboru s příponou *.map*.

```
Mapa obsazení oblasti registrů 'R'
;* Zdrojový soubor 'C:\XPRO\PRAC\STRUCT.950
;
R      0      :  47      =SLOZITEJSIZAZNAM OBROBKY[3]  ;
      R0      >> OBROBKY[0]~CASY[0]~HODINA
      R1      >> OBROBKY[0]~CASY[0]~MINUTA
      R2      >> OBROBKY[0]~CASY[0]~SEKUNDA
      R3      >> OBROBKY[0]~CASY[1]~HODINA
      R4      >> OBROBKY[0]~CASY[1]~MINUTA
      R5      >> OBROBKY[0]~CASY[1]~SEKUNDA
      R6      >> OBROBKY[0]~DATUMY[0]~DEN
      R7      >> OBROBKY[0]~DATUMY[0]~MESIC
      RW8     >> OBROBKY[0]~DATUMY[0]~ROK
      R10     >> OBROBKY[0]~DATUMY[1]~DEN
      R11     >> OBROBKY[0]~DATUMY[1]~MESIC
      RW12    >> OBROBKY[0]~DATUMY[1]~ROK
      RW14    > OBROBKY[0]~KUSY
      R16     >> OBROBKY[1]~CASY[0]~HODINA
      R17     >> OBROBKY[1]~CASY[0]~MINUTA
      R18     >> OBROBKY[1]~CASY[0]~SEKUNDA
      R19     >> OBROBKY[1]~CASY[1]~HODINA
      R20     >> OBROBKY[1]~CASY[1]~MINUTA
      R21     >> OBROBKY[1]~CASY[1]~SEKUNDA
      R22     >> OBROBKY[1]~DATUMY[0]~DEN
      R23     >> OBROBKY[1]~DATUMY[0]~MESIC
      RW24    >> OBROBKY[1]~DATUMY[0]~ROK
      R26     >> OBROBKY[1]~DATUMY[1]~DEN
      R27     >> OBROBKY[1]~DATUMY[1]~MESIC
      RW28    >> OBROBKY[1]~DATUMY[1]~ROK
      RW30    > OBROBKY[1]~KUSY
      R32     >> OBROBKY[2]~CASY[0]~HODINA
      R33     >> OBROBKY[2]~CASY[0]~MINUTA
      R34     >> OBROBKY[2]~CASY[0]~SEKUNDA
      R35     >> OBROBKY[2]~CASY[1]~HODINA
      R36     >> OBROBKY[2]~CASY[1]~MINUTA
      R37     >> OBROBKY[2]~CASY[1]~SEKUNDA
      R38     >> OBROBKY[2]~DATUMY[0]~DEN
      R39     >> OBROBKY[2]~DATUMY[0]~MESIC
      RW40    >> OBROBKY[2]~DATUMY[0]~ROK
      R42     >> OBROBKY[2]~DATUMY[1]~DEN
      R43     >> OBROBKY[2]~DATUMY[1]~MESIC
      RW44    >> OBROBKY[2]~DATUMY[1]~ROK
      RW46    > OBROBKY[2]~KUSY
```

Mapa kompletní.

Direktivy *#struct* nám tedy umožňují deklarovat nové datové typy. Takto definované typy lze použít nejen při definici proměnných, ale i při definici tabulek T. Blíže viz popis direktivy *#table*.

9.7. #data, #table

Práce s tabulkami je silnou stránkou PLC TECOMAT. Proto i nástrojům na jejich vytváření v programu byla věnována značná pozornost. Naprosto shodným způsobem jako tabulky T se definuje i datová oblast D.

Obecný tvar definic tabulky a dat jsou:

```
#data typ [index,]jméno0 = [[hod0[opak0]],  
    [jméno1 = ]hod1[[opak1]], ...,  
    [jménon-1 = ]hodn-1[[opakn-1]],  
    [jménon = ]hodn[[opakn]]  
#table typ [index,]jménoTab = [[hod0[opak0]],] hod1[[opak1]],  
    ..., hodn-1[[opakn-1]], hodn[[opakn]]
```

<i>typ</i>	- typ prvků tabulky, může být stejný jako v případě <i>#reg</i> (tab.9.1)
<i>index</i>	- nepovinné číslo, které nastavuje index vytvářené tabulky Po vnucení indexu tabulky uvedením čísla <i>index</i> jsou dalším definovaným tabulkám přiřazovány indexy v rostoucím pořadí počínaje indexem <i>index+1</i> . Pokud je <i>index</i> vynechán, je následujícím tabulkám automaticky přiřazován vzrůstající index (stejně tak jako v případě <i>#reg</i>).
<i>jménoTab</i>	- symbolické jméno tabulky T
<i>jméno0 - jménon</i>	- symbolická jména dat D
<i>hod0 - hodn</i>	- prvky tabulky Mohou být zadávány s použitím libovolné povolené číselné soustavy. Pokud hodnota prvku přeteče maximální hodnotu danou typem tabulky (např. pro typ byte je to 255), je číslo oříznuto; v případě definice bitové tabulky je její prvek nenulový, pokud je nenulová jeho hodnota <i>hodx</i> .
<i>opak0 - opakn</i>	- nepovinné opakování Jsou to čísla uzavřená párem hranatých závorek (znaky '[' a ']') a umožňují v tabulce zopakovat před nimi předcházející hodnotu <i>hodx opakx-krát</i> , je-li hodnota <i>hodx</i> textový řetězec, je opakován pouze jeho poslední znak.

Příklad 9.7

```
#def Zapnuto 1  
#def Vypnuto 0  
#def Auto 1  
#def Rucne 0  
#struct typZaznam  
    bool ZapVyp,  
    bool AutMan,  
    usint cisloStroje,  
    uint casPredvolba,  
    udint citacCyklu  
;  
#table typZaznam Zaznam1 = Zapnuto, Rucne, 20, $3009, 1236789  
#table typZaznam[2] Zaznam2 = Zapnuto, Rucne, 20, $3009, 1236789,  
    Vypnuto, Auto, 21, 19029, 1236789  
#table bool BitTable = 0,1,1,0,1,1,1,0,1[8],0,0  
#table usint 10,ByteTable = $12,34,%01010110,60#56  
#data uint WordData = 1,2,3,  
    NextData = 4,$4567[12],8,9,10,0[11],3  
;
```

P 0
:
E 0

9.8. #if, #elif, #else, #endif

Tyto direktivy slouží k podmíněnému překladu.
Jejich syntaxe je následující:

```
#if podminka_if          ;úvodní podmínka a začátek prvního bloku
                        ;podmíněného překladu

    :      ;část programu překládána, je-li aritmetický nebo logický
    :      ;výraz podminka_if nenulový

#elif podminka_elif_1 ;začátek dalšího bloku podmíněného překladu
    :      ;část programu překládána, je-li výraz podminka_if nulový
    :      ;a výraz podminka_elif_1 nenulový

#elif podminka_elif_n ;začátek předposledního bloku podmíněného
                        ;překladu

    :      ;část programu překládána, je-li výraz podminka_if nulový
    :      ;a současně výrazy podminka_elif_1 až podminka_elif_n-1
    :      ;nulové a výraz podminka_elif_n nenulový

#else                    ;začátek posledního bloku podmíněného překladu

    :      ;část programu překládána, není-li splněna ani jedna
    :      ;z předchozích podmínek

#endif                  ;konec podmíněného překladu

podminka_if, podminka_elif_1, ..., podminka_elif_n
    - relační nebo matematické výrazy
```

Je překládána vždy jen jedna z větví a to ta, která první splní svoji podmínku.
Direktivy `#if ... #endif` lze vzájemně vnořovat. Platí přitom následující pravidla:

- ♦ každé `#if` musí mít svoje `#endif`
- ♦ `#elif` nebo `#else` se vztahuje k nejbližšímu předchozímu `#if`
- ♦ `#elif` a `#else` jsou nepovinné

Příklad 9.8

Velmi častým případem použití podmíněných překladů je situace, kdy při ladění programu v simulaci programátor simuluje programem odezvy reálného stroje. Tyto části programu jsou prakticky vždy překládány podmíněně jen v případě ladění v simulaci.

```
#def LADENI 1          ;1 ... ladění v simulaci
                        ;0 ... reálný stroj

;

#if LADENI == 1
#include simstroj.mos    ;při ladění překládat simstroj.mos
#endif
```

Příklad 9.9

Podmíněné překlady lze s výhodou použít i v případech, kdy uživatelský program potřebujeme přeložit pro centrální jednotky různých řad. K tomu slouží vnitřní proměnná `_PLCTYPE_`, jejíž hodnota určuje řadu PLC, pro kterou se program překládá. V prostředí Mosaic je nastavení překladače dáno volbou typu PLC v manažeru projektu ve složce *Hw / Výběr řady PLC*.

```
#if _PLCTYPE_ == CPM1D
    INR    promenna
#else
    LD     promenna
    INR
    WR     promenna
#endif
```

9.9. #ifdef, #ifndef, #else, #endif

Tyto direktivy slouží k podmíněnému překladu.

Podmínkou pro překlad je existence symbolického jména v předcházejícím programu. Jejich syntaxe je následující:

```
#ifdef symbolické_jméno    ;podmínka pro překlad
    :                      ;část programu překládaná, je-li v předcházející části
                          ;programu definováno uvedené symbolické jméno
#else
    :                      ;přepínač bloku podmíněného překladu
    :                      ;část programu překládaná, není-li v předcházející části
                          ;programu definováno uvedené symbolické jméno
#endif
                          ;konec podmíněného překladu
```

Direktiva `#ifndef` umožní překlad, pokud ještě neexistuje symbolické jméno uvedené v direktivě. Její použití je stejné jako použití direktivy `#ifdef`.

```
#ifndef POČET
    #def POČET 12
    #reg byte pole[POČET]
#endif
```

9.10. #usi

Překladač xPRO umožňuje v uživatelském programu používat uživatelské instrukce USI napsané pro PLC TECOMAT. Instrukce USI je vlastně funkce napsaná v jazyce C, jejíž přeložený kód je překladačem přilinkován ke strojovému kódu uživatelského programu. Tak lze rozšiřovat instrukční soubor centrální jednotky PLC o funkce, které nejsou součástí standardního instrukčního souboru. K definici instrukcí USI v programu slouží direktiva `#usi`.

Jeho syntaxe je následující:

```
#usi [index,]jméno_instrukce = jméno_souboru
```

index - nepovinné číslo, které nastavuje index vytvářené instrukce
Po vnucení indexu instrukce uvedením čísla *index* jsou dalším definovaným instrukcím přiřazovány indexy v rostoucím pořadí počít-

	naje indexem <i>index+1</i> . Pokud je <i>index</i> vynechán, je následujícím instrukcím automaticky přiřazován vzrůstající index.
<i>jméno_instrukce</i>	- symbolické jméno indexu uživatelské instrukce, v případě, že je uveden index, je nepovinné, jinak povinné
<i>jméno_souboru</i>	- povinné jméno binárního diskového souboru, který obsahuje výkonnou část uživatelské instrukce

S vývojovým prostředím Mosaic je dodávána řada instrukcí USI, které jsou po instalaci prostředí typicky umístěny v adresáři *Mosaic1\USI*. Každá instrukce USI je přeložena několikrát pro různé typy centrálních jednotek PLC. To znamená, že ke každé instrukci USI zpravidla existuje několik souborů v adresáři USI. Typ centrální jednotky, pro kterou je soubor s kódem instrukce USI určen, je rozlišen příponou souboru. Pokud neuvedeme příponu souboru v direktivě *#usi*, překladač automaticky vybere soubor se správnou příponou podle typu centrální jednotky, pro kterou se právě překládá. Pro simulace jsou určeny soubory s příponou *.dll*. Tyto soubory použije prostředí automaticky v režimu simulovaného PLC nezávisle na tom, pro jakou řadu centrálních jednotek byl proveden překlad. Soubory *.dll* jsou použity pouze pro simulaci funkce USI, do strojového kódu přeloženého programu je uložen soubor s kódem, který odpovídá zvolenému typu centrální jednotky. Takže při přechodu z režimu simulace PLC do ladění s reálným PLC není potřeba v souvislosti s použitými instrukcemi USI nic ošetřovat.

Vytváření uživatelských instrukcí je popsáno v kap.12.

Přípona souboru s kódem USI	Určeno pro centrální jednotku
.uia	řady A
.uib	řady B
.uic	řady C
.uid	řady D
.uim	řady M
.uis	řady S
.dll	pro simulace PLC v prostředí Mosaic

Příklad 9.10

```
#usi operacniPanel = ter_id04 ;definice
;
P 0
:
  USI operacniPanel ;použití v programu
:
E 0

nebo

#usi operacniPanel = ter_id04 ;definice
#def TERM USI operacniPanel
;
P 0
:
  TERM ;použití v programu
:
E 0
```

9.11. #label

Pomocí direktivy *#label* se nastavuje počáteční index pro automatické přidělování návěští a provádí rezervace symbolických návěští pro použití např. v indexovaných nebo relativních skocích atd. Tato direktiva je tedy určena pouze pro ty případy, kdy programátor úlohy potřebuje rezervovat konkrétní čísla návěští. Pokud potřebujeme do programu PLC umístit návěští např. jako cíl podmíněného skoku, není potřeba toto návěští deklarovat v direktivě *#label*. Stačí pouze napsat symbolické jméno návěští na samostatný řádek programu a ukončit ho dvojtečkou (viz kap.4.3.).

Direktiva *#label* má následující tvar:

```
#label      [index,][jméno_L0[[opak0]][,jméno_L1[[opak1]],
            ...jméno_Ln[[opakn]]]
```

index - nepovinné kladné číslo určující první index pro automatické přidělování návěští

jméno_L0 - jméno_Ln - libovolná symbolická jména návěští, kterým jsou přiřazena absolutní návěští, seznam jmen návěští může pokračovat na více řádcích

opak0 - opakn - nepovinné opakovací povinně uzavřené v hranatých závorkách určující, kolik indexů návěští vynechat za definovaným návěštím

Pozor! Používají-li se v programu absolutní návěští, je třeba zaručit, aby nebyla přidělena jinému (symbolickému) návěští. V programech psaných v prostředí Mosaic doporučujeme absolutní návěští nepoužívat. Předejte se tak kolizím při dvojí deklaraci téhož návěští, apod.

Příklad 9.11

```
;prostor od L 0 do L 9 je neobsazený
#label      10, PrvniNavesti [3],          ;PrvniNavesti = L 10
            Nav, Navesti[10],              ;Nav = L 13, Navesti = L 14
            DalsiNavesti                    ;DalsiNavesti = L 24
#label      100, NavestiSto                 ;NavestiSto = L 100
```

9.12. #macro, #endm

Překladač xPRO disponuje i tak mocným prostředkem, jakým je makroinstrukce. Zkušený programátor dokáže pomocí makroinstrukcí značně zpřehlednit jinak dlouhý a těžko čitelný program.

Makroinstrukce (nebo také zkráceně makra) se používají všude tam, kde se v programu vyskytují shodné části, které však používají rozdílné operandy (například ovládání několika motorů). Pokud tu část ovladače, jež je všude stejná (používá stejný sled instrukcí), napíšete jako makroinstrukci, potom se každá obsluha motoru v programu zapíše snadno jako makroinstrukce. Rozdílné budou pouze parametry předávané této makroinstrukci.

Makroinstrukce lze vzájemně vnořovat, to znamená používat v těle makroinstrukce jinou makroinstrukci.

Příklad 9.12

Definujme krátkou makroinstrukci, která logicky sečte dva bity a výsledek zapíše do jiného bitu:

```
#reg bool vstup, vystup, va, vb, vc
;
;definice makroinstrukce
#macro prvni_makro (prvni, druhy, treti)
    LD    prvni
    OR    druhy        ;logický součet
    WR    treti        ;výsledek
#endm
;
P 0
:
LDC  vstup
prvni_makro (va, vb, vc)
WRC  vystup
:
E 0
```

Jak je vidět, při definici makroinstrukce je důležité dodržet tyto zásady:

- ♦ Definice makroinstrukce začíná direktivou *#macro* a končí direktivou *#endm*.
- ♦ Za jménem makroinstrukce následuje seznam tzv. formálních parametrů makra, který je uzavřen v kulatých závorkách. I když makroinstrukce nepoužívá žádné parametry, je nutno použít závorek.
- ♦ Pokud se seznam formálních parametrů nevejde na jeden řádek, lze pokračovat na dalším řádku. Např.:

```
#macro dlouha_makroinstrukce (prvni_parametr,
                                druhy_parametr)
```

- ♦ Tělo makroinstrukce se skládá z posloupností instrukcí. Uvnitř makroinstrukce mohou být použita klíčová slova *#def*, *#reg*, *#label*, *#table* a *#data*. Takto definovaná symbolická jména jsou lokální, tj. jsou známa pouze v těle makroinstrukce. Pokud je lokální symbolické jméno stejné jako symbolické jméno definované v hlavním programu, platí lokální.

Použití makroinstrukce v textu programu je následující:

```
prvni_makro (Blokada_M1, Spoustec_M1, Vystup_M1)
```

Při použití makroinstrukce se tedy napíše jméno makroinstrukce a do závorek předávané, tzv. skutečné, parametry. Ty při rozvoji makroinstrukce nahradí její formální parametry. Počet parametrů v definici makroinstrukce (tj. počet formálních parametrů) musí být shodný s počtem předávaných (skutečných parametrů).

Jak se zpracovávají makroinstrukce?

Pokud při překladu překladač narazí na jméno makroinstrukce, rozvine ji. To znamená, že jméno makroinstrukce nahradí posloupností instrukcí tvořících tělo makroinstrukce. Formální parametry jsou přitom nahrazeny skutečnými.

V případě již dříve definovaného makra *prvni_makro* bude program

```
LDC  vstup
prvni_makro (va, vb, vc)
WRC  vystup
```

po rozvinutí makroinstrukce *prvni_makro* překladačem vypadat takto:

```
LDC  vstup
LD   va        ;tady začíná rozvoj makra
OR   vb
```

```
WR      vc      ;poslední instrukce rozvoje makra
WRC     vystup
```

9.13. #mnemo, #mnemoend

Direktiva *#mnemo* informuje editor reléových schémat, aby veškeré instrukce následující tuto direktivu byly vypsány ve své textové podobě, nikoliv ve formě reléového schématu. Návrat k reléovému zobrazování se provede po direktivě *#mnemoend*.

Jejich syntaxe je:

```
#mnemo
:      ;úsek programu, který se vypíše v instrukcích
#mnemoend
```

Při programování reléových schémat je někdy nutno vytvořit takovou sekvenci instrukcí, jejichž interpretace v reléových symbolech není logická. Tuto část programu je možno uzavřít párem *#mnemo* / *#mnemoend* a bude vždy zobrazována ve své textové podobě.

Poznámka: Direktivy by měly být párové, to znamená, že direktiva *#mnemo* má mít svoji vypínací *#mnemoend*.

9.14. #useoption

V prostředí Mosaic jsou při překladu na základě nastavení překladače vygenerovány direktivy *#useoption* do řídicího souboru *xxx.mak*, kde *xxx* je jméno projektu. Tím je do projektu zaneseno nastavení překladače a není třeba jej v programu ručně měnit. Případná změna v nastavení překladače se projeví při následujícím překladu. Díky vygenerování direktiv *#useoption* do řídicího souboru *xxx.mak* máme navíc vizuální kontrolu parametrů překladače.

Syntaxe direktivy *#useoption* je následující:

```
#useoption mod = n ;komentář
```

kde *mod* je jeden z následujících parametrů:

CPM - řada centrální jednotky PLC
0 ... řada A
1 ... řada S
2 ... řada M
3 ... řada E
4 ... řada D
5 ... řada B
6 ... řada C

Tato hodnota je uložena do vnitřní proměnné *_PLCTYPE_*, kterou lze použít kdekoli v uživatelském programu, např. pro podmíněný překlad (viz příklad 9.11).

BlockOut - externí blokování výstupů PLC
0 ... vypnuto
1 ... aktivní v log.0
2 ... aktivní v log.1

EnableRun - externí povolení RUN
0 ... vypnuto
1 ... aktivní v log.0
2 ... aktivní v log.1

<i>AlarmTime</i>	- délka cyklu pro varování [ms]
<i>MaxCycleTime</i>	- max. délka cyklu [ms]
<i>RemZone</i>	- délka remanentní zóny 0 ... žádný registr není zálohován ≠0 ... počet zálohovaných registrů R počínajíc od R0
<i>PlcStart</i>	- typ startu PLC po zapnutí napájení 0 ... teplý (obsah zálohovaných registrů je zachován) 1 ... studený (všechny registry R po zapnutí vynulovány)
<i>ProtTable</i>	- chráněné tabulky T - má význam pouze v případě, že uživatelský program je zálohován v paměti EEPROM 0 ... vypnuto (po zapnutí napájení PLC bude z paměti EEPROM natažen celý uživatelský program včetně tabulek T) 1 ... zapnuto (po zapnutí napájení PLC bude z paměti EEPROM natažen celý uživatelský program kromě tabulek T)

Příklad 9.13

; Varování: Tento soubor je spravován vývojovým prostředím Mosaic.
; Nedoporučuje se upravovat ho ručně!

```
#program Plc1 , V1.0
;*****
;<ActionName/>
;<Programmer/>
;<FirmName/>
;<Copyright/>
;*****
;<History>
;</History>
;*****
#useoption CPM = 5           ; Typ CPM: B
#useoption RemZone = 2       ; délka remanentní zóny
#useoption AlarmTime = 150   ; první výstraha [ms]
#useoption MaxCycleTime = 250 ; maximální cyklus [ms]
#useoption PLCstart = 1      ; studený start
#useoption BlockOut = 0      ; externí blokování výstupů vypnuto
#useoption EnableRun = 0     ; externí blokování vykonávání programu
                             ; PLC vypnuto
#useoption ProtTable = 0     ; tabulky nejsou při restartu PLC chráněny
;*****
#usefile "Plc1.hwc"
#usefile "Plc1.mos"
#usefile "..\akce.sym"
#usefile "Plc1.sym"
```

Uvedený příklad ukazuje řídicí soubor *Plc1.mak* v prostředí Mosaic pro centrální jednotku řady B a studený restart po zapnutí napájení PLC se zálohováním obsahu registrů R0 a R1. Tyto registry mají po zapnutí napájení stejný obsah jako před vypnutím, ostatní registry jsou vynulovány. Jméno skupiny projektů je *akce* a jméno projektu je *Plc1*.

10. UŽIVATELSKÉ PROCESY

10.1. VŠEOBECNÉ ZÁSADY AKTIVACE

Uživatelský program se skládá z uživatelských procesů. Teoreticky jich smí být až 65 (P0 až P64), prakticky jich bývá výrazně méně. Na rozdíl od tradičních operačních systémů reálného času pro počítače zde uživatel nemá takovou volnost při ovládání procesů. Procesy jsou aktivovány podle předem definovaných pravidel. V rámci těchto pravidel můžeme dodatečně ovlivnit aktivaci většiny procesů v průběhu uživatelského programu.

Tab.10.1 Přehled procesů uživatelského programu a jejich určení

Procesy	Určení
P0	základní proces
P1 až P4	čtyřfázově aktivované procesy
P5 až P9	časově aktivované procesy
P10 až P40	uživatelsky aktivované procesy
P41 až P48	přerušovací procesy
P49	systémový proces - nepoužívat!
P50 až P57	ošetření ladicího bodu
P58, P59	systémové procesy - nepoužívat!
P60	balík podprogramů
P61	systémový proces - nepoužívat!
P62	teplý restart
P63	studený restart
P64	závěrečný proces cyklu

Centrální jednotky řady E mají možnost naprogramovat pouze proces P0.

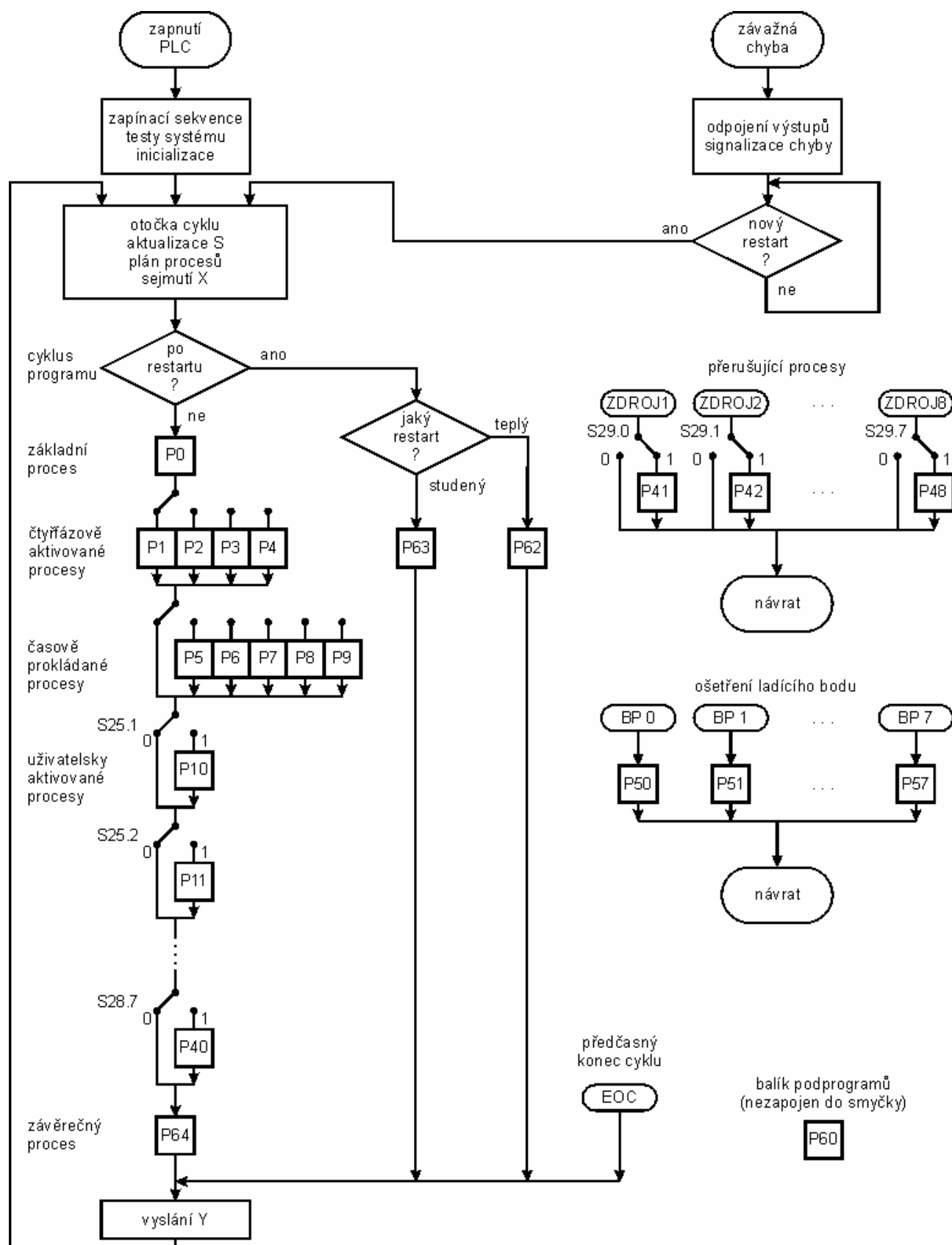
Schéma aktivace procesů je na obr.10.1. Slabě jsou orámovány akce systému, silně jsou orámovány uživatelské procesy.

Vyloučení procesů

Uživatel není nucen využívat všechny procesy. Pokud mu vyhovuje klasické jednosmyčkové řízení, může zadat pouze proces P0. Procesy lze vyloučit trojím způsobem:

- ♦ Proces není naprogramován, tj. nejsou použity závorkové instrukce P a E příslušného procesu. Jedině proces P0 nelze takto vyloučit, může však být prázdný.
- ♦ Proces je prázdný, tj. mezi závorkovými instrukcemi P a E příslušného procesu není další instrukce. Jeho aktivace se projeví jen jako prázdná operace.
- ♦ Aktivační maska procesu je vynulovaná. Aktivační masky procesů P10 až P48 jsou obsaženy v systémových registrech S25 až S29 (kap.5.3.). Proces s vynulovanou aktivační maskou bude potlačen v následujícím cyklu, resp. ihned, jedná-li se o přerušující proces P41 až P48. Systémem spravované procesy P0 až P9 nelze takto potlačit.

Upozornění: Při vstupu do řešení kteréhokoliv z procesů P0 až P40, P62, P63, P64 je aktivní uživatelský zásobník vynulován (při vstupu do řešení procesu P0 je vždy aktivní zásobník A).



Obr.10.1 Schéma aktivace procesů

Skok mezi procesy

Není vyloučeno (i když to přímo nedoporučujeme) převedení řízení z jednoho procesu do jiného instrukcí skoku nebo volání. Nejbližší instrukce E nebo ED, EC se splněním podmínkou, však zakončí řádně aktivovaný proces - uživatel může přecházet mezi programy libovolných procesů, systémový program tyto změny neregistruje jako změny aktivace procesů a vyhodnocení konce procesu chápe jako návrat z procesu aktivovaného systémovým programem (nikoliv uživatelskými přechody). Jestliže například z procesu P0 skočíme na návěští umístěné v posledním procesu P64, pak E64 nezpůsobí otočku cyklu, ale uzavření P0 (stejně jak E0) a jeho návrat do systémového programu, který vyvolá následující uživatelský proces. Zásobník bude v tomto případě vynulován až po uzavření procesu P0, přechod mezi procesy instrukcí skoku nebo volání tedy nemá za následek jakoukoli změnu hodnot v zásobníku.

Předčasný konec cyklu

Pouze instrukce EOC je schopna porušit posloupnost procesů naplánovaných pro současnou smyčku a přímo (a nepodmíněně) provést otočku cyklu. To může být účelné např. při obsluze přerušení nebo pro zajištění rychlé odezvy na kritické situace. Při plánování procesů pro další oběh v takto vynucené otočce cyklu se nerespektuje, že některé z původně naplánovaných procesů nebyly aktivovány - začíná se plánovat znovu.

Přerušující procesy

Kterýkoliv z procesů smyčky může být přerušen některým z přerušujících procesů P41 až P48. Instrukce rozpracovaná v okamžiku přerušení se dokončí a teprve po ní je provedena první instrukce přerušujícího procesu. Po jeho instrukci E (nebo ED, EC) pokračuje přerušený proces. Přerušující proces nemění stav žádné úrovně aktivního zásobníku.

Doba cyklu při použití přerušujících procesů

Doba cyklu se prodlužuje o součet dob přerušujících procesů, které byly aktivovány během průchodu smyčkou. Obslužné programy by proto měly obsahovat pouze nezbytně nutné instrukce pro zabezpečení rychlé odezvy na situaci, která přerušení vyvolala. V opačném případě může dojít k výraznému prodloužení doby cyklu. Extrémním případem by bylo zastavení provádění ostatních procesů a systém by obsluhoval pouze přerušení. Tento stav je omezen systémovou podmínkou, která omezuje dobu trvání přerušujícího procesu na 5 ms. Z téhož důvodu není umožněno vnořování přerušení (přerušení přerušujícího procesu).

Mechanismus přerušení je velmi účinný aparát, který umožňuje výrazně zkrátit odezvu systému na kritické situace, musí však být používán uvážene.

10.2. OTOČKA CYKLU

Po zapínací sekvenci (viz kap.2.1), po ukončení posledního z procesů naplánovaných pro aktivní cyklus nebo po instrukci EOC se provádí tzv. otočka cyklu. Její doba je závislá na konfiguraci systému, délce remanentní zóny a rozsahu jiných systémových služeb (viz příloha).

V otočce cyklu se vysílá stav registrů Y do výstupů, aktualizují se registry X podle stavu vstupů, aktualizuje se systémový čas, plánují se procesy pro aktivaci příštího cyklu, generují se náběžné a sestupné hrany, rozhoduje se o režimu systému (režim RUN - HALT, blokování výstupů, typ případného restartu), nastavuje se platný stav systémových registrů S a aktivuje a nuluje se zásobník A.

10.3. OŠETŘENÍ RESTARTU - PROCESY P62, P63

P62 - teplý restart

P63 - studený restart

V prvním cyklu po restartu může být aktivován jeden z procesů P62, P63. Tyto procesy slouží k inicializaci proměnných. Pokud je prováděn jeden z procesů P62, P63, není v tomto cyklu již prováděn žádný jiný proces, tedy ani P0. Centrální jednotky řad S, D, B a C mají během vykonávání procesů P62, P63 pozastavenou aktivaci přerušujících procesů z důvodu odstranění možnosti hazardu spočívajícího v práci s nenainicializovanými datovými strukturami.

Poznámka: K rozlišení prvního cyklu po opětovném odstartování (SP = 1 nebo změna režimu HALT → RUN bez restartu) je určen bit S2.6, který je nastaven na log.1 tehdy, jestliže byl systém právě tímto cyklem odstartován bez provedení restartu.

Teplý restart - model 16 bitů

Pokud je nastaven teplý restart, provede se proces P62 a v následujícím cyklu se začne provádět proces P0 a další naprogramované procesy. Není-li naprogramován proces P62, provede se místo něj proces P63. Pokud není naprogramován ani jeden z procesů P62, P63, začne se rovnou provádět proces P0 a další naprogramované procesy.

Teplý restart - model 32 bitů

Pokud je nastaven teplý restart, provede se proces P62 a v následujícím cyklu se začne provádět proces P0 a další naprogramované procesy. Není-li naprogramován proces P62, začne se rovnou provádět proces P0 a další naprogramované procesy.

Centrální jednotky s šířkou zásobníku 32 bitů **nemají** automatické zdvojení procesu ošetření restartu. Pokud chceme spouštět tentýž algoritmus při teplém i studeném restartu, napíšeme jej jako podprogram do procesu P60 a zavoláme jej z obou procesů P62 a P63.

Studený restart - model 16 bitů

Pokud je nastaven studený restart, provede se proces P63 a v následujícím cyklu se začne provádět proces P0 a další naprogramované procesy. Není-li naprogramován proces P63, provede se místo něj proces P62. Pokud není naprogramován ani jeden z procesů P62, P63, začne se rovnou provádět proces P0 a další naprogramované procesy.

Studený restart - model 32 bitů

Pokud je nastaven studený restart, provede se proces P63 a v následujícím cyklu se začne provádět proces P0 a další naprogramované procesy. Není-li naprogramován proces P63, začne se rovnou provádět proces P0 a další naprogramované procesy.

Centrální jednotky s šířkou zásobníku 32 bitů **nemají** automatické zdvojení procesu ošetření restartu. Pokud chceme spouštět tentýž algoritmus při teplém i studeném restartu, napíšeme jej jako podprogram do procesu P60 a zavoláme jej z obou procesů P62 a P63.

Bez restartu

Pokud je nastaven režim bez restartu, neprovede se ani jeden z procesů P62, P63, ale začne se rovnou provádět proces P0 a další naprogramované procesy.

Příklad 10.1

```
#reg byte registr
;
P 0
:
E 0
;
P 62          ;teplý restart
    LD    $10
    WR    registr    ;počáteční hodnota
E 62
;
P 63          ;studený restart
    LD    $20
    WR    registr    ;počáteční hodnota
E 63
```

10.4. PROCESY SMYČKY

Z obr.10.1 je patrné, že pouze procesy P0 a P64 se aktivují v každém cyklu, procesy P1 až P9 se aktivují ve vybraných cyklech, procesy P10 až P40 aktivuje uživatel prostřednictvím řídicích masek. Výsledně se tyto procesy jeví jako různé smyčky uživatelského programu, z nichž každá má jinou dobu cyklu. Proto tento způsob aktivace můžeme označit jako vícesmyčkové řízení.

10.4.1. Základní proces P0**P0 - úvodní proces každého cyklu**

Základní proces P0 je povinnou součástí základní struktury uživatelského programu. I kdybychom proces P0 nechtěli použít, musíme jej naprogramovat (povinné instrukce P 0, E 0).

Základní proces P0 je aktivován v každém cyklu jako první s výjimkou restartu, kdy je aktivován pouze jeden z procesů P62 nebo P63.

Je účelné sem zařadit všechny úvodní operace a uživatelský plánovač procesů. Protože je aktivován vždy, měly by v něm být zařazeny pouze ty úlohy, u nichž je požadována krátká doba odezvy. Neměl by být zbytečně plněn úlohami s menší naléhavostí.

Příklad 10.2

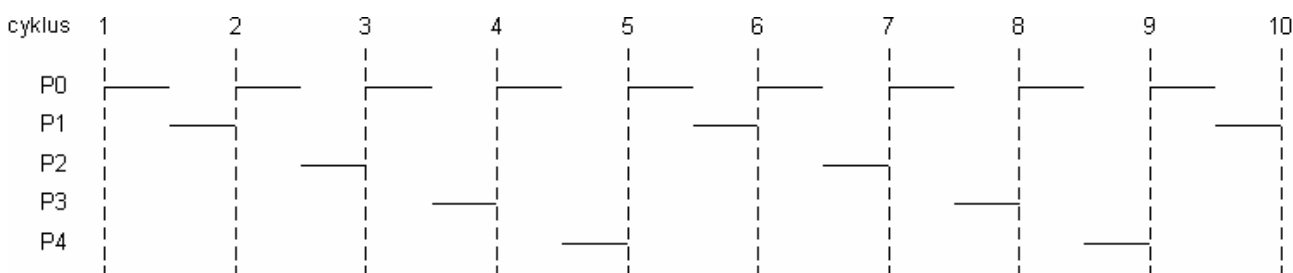
```
#reg byte vstup,vystup
;
P 0
    LD    vstup
    :
    WR    vystup
E 0
```

10.4.2. Čtyřfázově aktivované procesy P1, P2, P3, P4

- P1** - proces zařazený v každém prvním cyklu ze čtyř
- P2** - proces zařazený v každém druhém cyklu ze čtyř
- P3** - proces zařazený v každém třetím cyklu ze čtyř
- P4** - proces zařazený v každém čtvrtém cyklu ze čtyř

Procesy se v aktivaci cyklicky střídají v pořadí P1, P2, P3, P4, P1, ... (obr.10.2). Jejich základní vlastností je, že v každém cyklu je aktivní právě jen jeden z těchto čtyř procesů. To umožňuje tyto procesy použít pro programování kolizních akcí, které se nesmějí provádět ve stejném cyklu, nebo rozložení jednoho dlouhého algoritmu na čtyři části a docílit tak zkrácení doby cyklu uživatelského programu a zrychlení odezvy PLC. Je tak možné jednoduchými prostředky provést důslednou synchronizaci algoritmu a zabránit hazardním souběhům, chybným přechodům a nežádoucím přechodovým dějům.

Procesy P1 až P4 jsou aktivovány vždy po skončení základního procesu P0. Aktivace procesů P1 až P4 je odvozena od stavu čítače cyklů v S4. Z toho vyplývá, že pokud použijeme např. jen procesy P1, P2 a P3, v cyklu, který by příslušel procesu P4, se nebude aktivovat žádný z procesů této skupiny. V prvním cyklu po restartu systému je vždy aktivován proces P1.



Obr.10.2 Časové průběhy aktivací procesů P0 až P4

Příklad 10.3

```
#reg byte vstup,vystup
;
P 0
    LD    vstup
    :
    WR    vystup          ;prováděno v každém cyklu
E 0
;
P 1
    :
    :          ;prováděno v 1., 5., 9., ... cyklu
E 1
;
P 2
    :
    :          ;prováděno v 2., 6., 10., ... cyklu
E 2
;
P 3
    :
    :          ;prováděno v 3., 7., 11., ... cyklu
E 3
;
P 4
    :
    :          ;prováděno v 4., 8., 12., ... cyklu
E 4
```

10.4.3. Časově aktivované procesy P5, P6, P7, P8, P9

- P5** - proces zařazený každých 400 ms
P6 - proces zařazený každých 3,2 s (posun o 200 ms vůči P5)
P7 - proces zařazený každých 25,6 s (posun o 400 ms vůči P6)
P8 - proces zařazený každých 204,8 s, tj. 3,4 min. (posun o 800 ms vůči P7)
P9 - proces zařazený každých 1638,4 s, tj. 27,2 min. (posun o 1,6 s vůči P8)

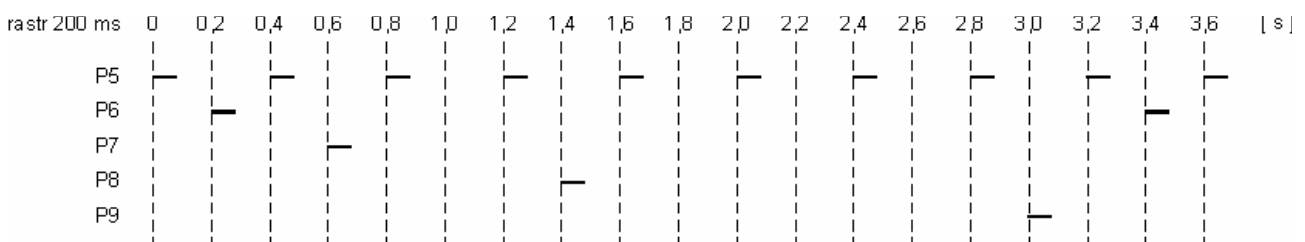
Procesy P5, P6, P7, P8, P9 jsou aktivovány vždy po uplynutí určitého časového intervalu. Přesnost tohoto intervalu je dána dobou cyklu. Aktivace jednotlivých procesů této skupiny jsou navíc vůči sobě posunuty tak, aby se v jednom cyklu aktivoval maximálně jeden z těchto procesů (obr.10.3).

Procesy P5 až P9 jsou aktivovány po čtyřfázově aktivovaných procesech P1 až P4. Aktivace procesů jsou odvozeny od čítače časových jednotek SW14. Četnost aktivace procesu vyššího čísla je vždy 8 krát menší než četnost aktivace procesu s číslem o jednu menšího (má 8 krát delší interval). Podmínkou pro správnou aktivaci procesů P5 až P9 je doba cyklu kratší než 200 ms. Při překročení této doby může dojít k aktivaci více procesů této skupiny v jednom cyklu a k vynechání aktivace procesu P5.

Použití procesů P5 až P9 je výhodné zejména v případech typu:

- „několikrát za sekundu proved...“,
- „po několika sekundách proved...“,
- „několikrát za minutu ...“,
- „po několika minutách...“,
- „zhruba po půlhodině...“.

Pak není potřeba pracovat s časovači ani časoměrnými registry a stačí úlohu pouze zařadit do vhodného procesu.



Obr.10.3 Možné okamžiky aktivací procesů P5 až P9

Příklad 10.4

```

#reg byte vstup,vystup
;
P 0
    LD    vstup
    :                    ;prováděno v každém cyklu
    WR    vystup
E 0
;
P 5
    :                    ;prováděno každých 400 ms
E 5
;
P 6
    :                    ;prováděno každých 3,2 s
E 6
;
P 7

```

```

:           ;prováděno každých 25,6 s
E 7
;
P 8
:           ;prováděno každých 3,4 min.
E 8
;
P 9
:           ;prováděno každých 27,2 min.
E 9

```

10.4.4. Uživatelsky aktivované procesy P10 až P40

P10 až P40 - procesy zařazené uživatelem

Procesy P10 až P40 jsou aktivovány uživatelem nastavením řídicích bitů v systémových registrech S25 až S28. Nastavením příslušného bitu na log.1 se v následujícím cyklu aktivuje daný uživatelský proces. Přirazení procesů k maskám je pořadové: P10 - S25.1, P11 - S25.2, ..., P40 - S28.7. Hodnoty na pozicích S25.1 až S28.7 ovlivňuje pouze uživatel, systémový program na nich nehospodaří. Procesy P10 až P40 jsou tedy aktivní od doby, kdy je uživatel zařadil, do doby, kdy je opět vyřadí. Procesy se aktivují v pořadí dle obr.10.1 (vzestupně podle jejich čísel) a toto pořadí nelze změnit v dopředném ani zpětném směru. Po restartu PLC jsou všechny bity vynulovány a procesy P10 až P40 nejsou aktivovány.

	.7	.6	.5	.4	.3	.2	.1	.0
S24	P8	P7	P6	P5	P4	P3	P2	P1
S25	P16	P15	P14	P13	P12	P11	P10	P9
S26	P24	P23	P22	P21	P20	P19	P18	P17
S27	P32	P31	P30	P29	P28	P27	P26	P25
S28	P40	P39	P38	P37	P36	P35	P34	P33

Masky S24.0 až S25.0 jsou přiřazeny procesům P1 až P9 a hospodaří na nich systémový program při otočce cyklu (během cyklu je nemění). Hodnoty masek jsou určeny k indikaci procesů naplánovaných k aktivaci pro tento cyklus. Přepisem masek uživatelským programem nelze ovlivnit aktivaci procesů P1 až P9.

Použití procesů P10 až P40 je široké. Jejich pomocí lze přehledně realizovat podmíněné provádění různých činností, kdy jednotlivé činnosti jsou programovány v příslušných procesech, v základním procesu P0 jsou vyhodnocovány podmínky a na jejich základě jsou spouštěny jednotlivé procesy. Je na uživateli, jaký význam procesům P10 až P40 přiřadí a podle jakých pravidel je bude aktivovat a na jakou dobu (od jednorázových aktivací až po dlouhodobou volbu režimu).

Procesy P10 až P40 jsou aktivovány po časově prokládaných procesech P5 až P9 vzestupně podle čísel. Změna příslušného řídicího bitu se projeví vždy až v následujícím cyklu.

Příklad 10.5

```

#reg bool vstup0, vstup1, vstup2, vstup3, test1, test2, test3
;
P 0
    LD    vstup0
    WR    %S25.1      ;P10 aktivní při vstup0 = log.1
    LD    vstup1

```

```

    LET test1      ;P11 aktivován jednorázově při náběžné
    SET %S25.2      ;hraně vstup1
    LD vstup2
    LET test2      ;P12 aktivován při náběžné hraně
    SET %S25.3      ;vstup2
    LD vstup3
    LET test3      ;P12 deaktivován při náběžné hraně
    RES %S25.3      ;vstup3
E 0
;
P 10
:
E 10
;
P 11
:
    LD 0
    WR %S25.2      ;P11 je jednorázový, sám se zruší
E 11
;
P 12
:
E 12

```

10.4.5. Závěrečný proces cyklu P64

P64 - proces zařazený vždy na konec cyklu

Proces P64 je prováděn vždy jako poslední uživatelský proces v cyklu. Je vhodný pro naprogramování algoritmů, které je třeba provést až po provedení procesů P1 až P40. Jeho naprogramování není povinné.

Příklad 10.6

```

P 0
:      ;prováděno na začátku každého cyklu
E 0
;
P 5
:      ;prováděno uprostřed cyklu každých 400 ms
E 5
;
P 64
:      ;prováděno na konci každého cyklu
E 64

```

10.5. PŘERUŠUJÍCÍ PROCESY

Chování přerušujících procesů

Kterýkoliv z procesů smyčky smí být přerušen na libovolné instrukci. Systémový program zajistí dokončení rozpracované instrukce, odloží stav aktivního zásobníku, systémových registrů S0, S1 a předá řízení na začátek přerušujícího procesu. Jeho závěrečná instrukce (E nebo ED, EC se splněnou podmínkou) vrací nepoškozený stav zásobníku a vrací řízení přerušnému procesu za místo přerušení. Přerušující proces neprovádí fázi

začátku ani konce cyklu. Z toho vyplývá, že nelze přenášet parametry z přerušovaného procesu do procesu přerušujícího v aktivním zásobníku.

Omezení doby trvání přerušujících procesů

Přerušující proces by měl být co nejkratší, nesmí překročit dobu 5 ms, jinak dojde k zastavení PLC z důvodu závažné chyby 80 31 pcpc. Přerušující procesy obecně slouží k ošetření kritických situací a rychlých dějů. Zde je třeba si uvědomit rozdíl mezi obrazem vstupů a výstupů v zápisníku, který se aktualizuje pouze v otočce cyklu, a přímým přístupem na vstupy a výstupy, který umožňuje přečíst aktuální stav a realizovat okamžitou reakci. Nadbytečné používání přímých přístupů však vede k prodlužování doby cyklu a také ke zvýšenému riziku časových hazardů (viz kap.6).

Nelze vnořovat přerušení

Přerušující proces již nemůže být přerušen (není možné vnořovat přerušení). Pokud přišel během přerušujícího procesu další požadavek na přerušení, neztrácí se a příslušný přerušující proces se následně vyvolá. Přejde-li těchto požadavků více, sčítají se logicky, ne co do množství, a respektuje se následující priorita v zařazování přerušujících procesů:

Priority přerušení

1. priorita - přerušení při chybě (P43)
2. priorita - přerušení od vstupů (P42)
3. priorita - přerušení od času (P41)
4. a další priorita - další zdroje přerušení vzestupně podle čísel (P44 až P48)

Zakázání aktivace přerušujícího procesu

Centrální jednotky řad S, D, B a C umožňují přechodné zakázání přerušujícího procesu. Po restartu jsou bity v registru S29 příslušné naprogramovaným procesům nastaveny na log.1. Vynulováním kteréhokoliv bitu způsobíme, že příslušný přerušující proces se okamžitě přestane vyvolávat (nečeká se na otočku cyklu). Opětovným nastavením tohoto bitu docílíme, že příslušný proces se ihned od tohoto okamžiku začne po vzniku požadavku vyvolávat. Pokud došlo k požadavku na přerušení v době, kdy byl příslušný proces odpojen, tento požadavek je ztracen, nelze tedy tuto možnost využít k pozdržení přerušujícího procesu.

Po restartu během vykonávání procesů P62 a P63 jsou přerušující procesy dočasně zakázány, aby nedošlo k hazardu z důvodu nenainicializovaných hodnot. Systém je uvolní až před prvním vstupem do procesu P0. Pokud tedy nechceme, aby k přerušení došlo, zakážeme přerušovací proces vynulováním příslušného bitu S29 už v procesu P62, resp. P63.

	.7	.6	.5	.4	.3	.2	.1	.0
S29	P48	P47	P46	P45	P44	P43	P42	P41

10.5.1. Přerušení od času P41

Proces P41 je aktivován pravidelně každých 10 ms. Je vhodný pro ošetření dějů vyžadujících kratší reakční dobu, než je doba cyklu. Pokud pracujeme se vstupy a výstupy, musíme použít přímé přístupy, nikoli obrazy v zápisníku, které se během cyklu nemění (viz kap.6.).

Proces P41 má 3. nejvyšší prioritu řazení přerušujících procesů, která se uplatní při setkání požadavků na více přerušujících procesů.

Příklad 10.7*model 16 bitů*

```

#reg bool    stav
#reg udint   citac
;
P 0
:
E 0
;
P 41
    LD    %U$9000    ;fyzická adresa vstupu X0
    AND   %1000      ;vstup X0.3
    LET   stav        ;test náběžné hrany
    EC
    INR   citac        ;přičtení pulzu
E 41

```

model 32 bitů

```

#reg bool    stav
#reg udint   citac
;
P 0
:
E 0
;
P 41
    LD    1          ;PAR - číst vstupy
    LD    1          ;RM - číslo rámu
    LD    4          ;POS - pozice modulu v rámu
    RFRM                    ;načtení aktuálních dat do struktury r1_p4_DI
    LD    r1_p4_DI~DI3 ;vstup 3
    LET   stav        ;test náběžné hrany
    EC
    INR   citac        ;přičtení pulzu
E 41

```

10.5.2. Přerušení od vstupů P42

Proces P42 je aktivován při změně přerušovacího vstupu. Konkrétní řešení je závislé na typu PLC.

Proces P42 je vhodný pro ošetření dějů vyžadujících kratší reakční dobu, než je doba cyklu. Pokud pracujeme se vstupy a výstupy, musíme použít přímé přístupy, nikoli obrazy v zápisníku, které se během cyklu nemění (viz kap.6.).

Proces P42 má 2. nejvyšší prioritu řazení přerušujících procesů, která se uplatní při setkání požadavků na více přerušujících procesů.

Aktivace P42 v NS950

V PLC NS950 je aktivován proces P42 při jakékoliv změně vstupů 0.0 těchto vstupních jednotek, které mají propojkou aktivováno přerušení (jednotky IB-36 až IB-47).

Pokud má aktivní přerušení více vstupních jednotek než jedna a je požadováno rozlišení těchto přerušení, toto rozlišení je třeba provést uživatelsky následujícím způsobem. V uživatelském programu provedeme v každém cyklu kopii obrazů X těchto jednotek, od kte-

rych lze očekávat přerušení. V přerušujícím procesu P42 pak porovnáním přímých vstupů U s kopiemi odpovídajících obrazů X zjistíme, na které jednotce došlo ke změně hodnoty na nejnižším bitu a tím k vyvolání přerušení.

Kopírování obrazů X je nutné proto, že pokud některá jednotka vyvolá přerušení během otočky cyklu, může s velkou pravděpodobností dojít k tomu, že se do obrazů X v zápisníku zapíše již nový stav vstupů této jednotky, ale přerušující proces se vyvolá až po zahájení nového cyklu. Tento časový hazard odstraníme právě tím, že přímé vstupy porovnáváme s obrazy X o jeden cyklus staršími. Tomu však musí odpovídat i minimální doba mezi přerušeními od téže jednotky, která musí být o něco delší než doba nejdelšího cyklu.

Příklad 10.8a

model 16 bitů


```
#reg byte   vystup
#reg bool   stav,pomoc
;
P 0
:
E 0
;
P 42
    LD    %U$9000      ;fyzická adresa vstupu X0
    AND   1             ;vstup X0.0
    LET   stav          ;test náběžné hrany
    SET   pomoc         ;hodnota výstupu
    LD    pomoc
    AND   %100          ;výstup bit 2
    LD    vystup
    AND   %11111011     ;smazání staré hodnoty výstupu bitu 2
    OR    %1             ;přičtení nové hodnoty výstupu bitu 2
    WR    vystup        ;kopie do zápisníku
    WR    %U$9100       ;zápis na fyzickou adresu
E 42
```

Aktivace P42 v TC500 a TC600

V PLC TC500 a TC600 je aktivován proces P42 při jakékoliv změně logické úrovně vstupů DI0 až DI3. Aktivaci procesu P42 umožňují varianty TC503 až TC507, TC513 až TC517, TC603 až TC607.

Podrobnosti k obsluze přerušovacího procesu P42 jsou uvedeny v příručkách Technické vybavení programovatelných automatů TECOMAT TC500 TXV 138 07.01 a Technické vybavení programovatelných automatů TECOMAT TC600 TXV 138 08.01.

Aktivace P42 v TC700

V PLC TC700 může vyvolat proces P42 ten modul, který má v panelu *Nastavení modulu*, přístupném z manažeru projektu ve složce *Hw | Konfigurace HW* přes ikonu , zaškrtnutou volbu *Modul může vyvolat přerušení*. Pokud volba není zaškrtnutá, tento modul přerušení nevyvolá. Pokud v panelu tato volba není, znamená to, že tento typ modulu vyvolat přerušení nemůže v žádném případě.

Pokud má povoleno přerušení více modulů než jeden, k rozlišení těchto přerušení slouží systémové registry S56 a S57. Po průchodu instrukcí P 42 obsahuje registr S56 pozici modulu, který přerušení vyvolal, v rámu a S57 obsahuje číslo tohoto rámu (nastavené přepínačem na rámu). Tím je jednoznačně určen přerušující modul. Na uživateli je, aby na začátku procesu provedl pomocí těchto registrů rozhodnutí, jak přerušení ošetřit.

Pokud ve stejný okamžik vyvolají přerušení dva periferní moduly, je proces P42 spuštěn dvakrát po sobě, pro každý modul zvlášť. Znamená to tedy, že při každém vyvolání přerušení obsluhujeme vždy pouze jeden modul.

Pro urychlení obsluhy jsou vždy před průchodem instrukcí P 42 aktualizovány ty vstupy nebo celé objekty modulu, které mají pomoci podrobného nastavení povoleno vyvolávat přerušení, zatímco ostatní vstupy jsou zmrazeny, aby nedošlo ke změně jejich hodnoty během cyklu.

Pokud tedy například máme na vstupním periferním modulu nastaveno vyvolávání přerušení od náběžné hrany vstupu DI0 a sestupné hrany vstupu DI6, budou v okamžiku vyvolání přerušení od tohoto modulu aktualizovány pouze hodnoty těchto dvou vstupů a přerušovací příznaky, zatímco hodnoty ostatních vstupů nikoliv. Díky této funkci lze s ostatními vstupy pracovat v přerušovaných procesech bez rizika časového hazardu.

Podrobnosti k vyvolání přerušení vztahující se na konkrétní modul jsou uvedeny příslušné příručce.

Příklad 10.8b

model 32 bitů

```
P 0
      :
E 0
;
P 42
      LD    r1_p4_INT~INT0    ;příznak přerušení od náběžné hrany vstupu 0
      WR    r1_p5_DO~DO2      ;výstup 2
      LD    2                  ;PAR - zapisovat výstupy
      LD    1                  ;RM - číslo rámu
      LD    5                  ;POS - pozice modulu v rámu
      RFRM                      ;zápis aktuálních dat ze struktury r1_p5_DO
E 42
```

10.5.3. Přerušení od chyby P43

Proces P43 je aktivován při výskytu chyby, která nezpůsobí zastavení chodu PLC (zapišuje do registru S34).

Proces P43 je vhodný pro hromadné ošetření chybových stavů.

Proces P43 má nejvyšší prioritu řazení přerušujících procesů, která se uplatní při setkání požadavků na více přerušujících procesů.

Příklad 10.9

```
#reg bool   chyba
#reg uint   va, vb, vc
;
P 0
      LD    va
      LD    vb
      DID                      ;při dělení nulou se vyvolá proces P43
      WR    vc
E 0
;
P 43
      LD    1
      WR    chyba              ;nastavení chybového příznaku
E 43
```

10.5.4. Přerušení od hw čítačů nebo od inkrementálního snímače P44

V PLC TC500 a TC600 je aktivován proces P44 při dosažení předvolby nebo při přetečení rozsahu hw čítače. Aktivaci procesu P44 umožňují varianty TC503 až TC507, TC513 až TC517, TC603 až TC607.

Podrobnosti k obsluze přerušovacího procesu P44 jsou uvedeny v příručkách Technické vybavení programovatelných automatů TECOMAT TC500 TXV 138 07.01 a Technické vybavení programovatelných automatů TECOMAT TC600 TXV 138 08.01.

Proces P44 má 4. prioritu řazení přerušujících procesů, která se uplatní při setkání požadavků na více přerušujících procesů.

10.5.5. Přerušení od sériového kanálu CH2 P45

Pokud je sériový kanál CH2 v PLC TC400, TC500, TC600 a NS950 CPM-2S, CPM-1D nastaven do režimu UNI, příjem zprávy vyvolá proces P45. Bezprostředně před spuštěním tohoto procesu jsou do přijímací zóny zapsána přijatá data. Dochází zde ke změně dat během cyklu uživatelského programu, takže je třeba ošetřit případné manipulace s těmito daty v procesech smyčky.

Proces P45 slouží především pro okamžité zpracování menšího množství dat. Při větším počtu dat nebo pokud není požadavek na okamžité zpracování, necháme obsluhu sériového kanálu v procesech smyčky. Přijatá data budou pak předána vždy v otočce cyklu.

Proces P45 má 5. prioritu řazení přerušujících procesů, která se uplatní při setkání požadavků na více přerušujících procesů.

10.6. OŠETŘENÍ LADÍČÍHO BODU - PROCESY P50 AŽ P57

Po vykonání instrukce BP 0 až BP 7 je řízení předáno procesu P50 až P57 (poslední číslice v čísle procesu je shodná s operandem instrukce BP). V tomto procesu lze pak ošetřit získání informací z tohoto místa programu. Při vstupu do procesu P5x je zachován zásobník. Po skončení procesu jsou zásobník a registry S0 a S1 obnoveny v původních hodnotách. Tato funkce usnadňuje ladění uživatelského programu bez nepřehledných zásahů do těla programu.

Příklad 10.10

```
#reg uint  ladeni[128], pomoc, index
;
P 0
    :
    BP    0          ;vložená ladící instrukce
    :
    BP    0          ;vložená ladící instrukce
    :
E 0
;
P 50
    WR    pomoc      ;odložení hodnoty vrcholu zásobníku
    LD    127        ;mez
    LD    index
```

```
LD    pomoc
WTB    ladeni
E 50
```

10.7. BALÍK PODPROGRAMŮ P60

Tento proces není aktivován ze systémového programu a slouží pouze k uskladnění souboru podprogramů volaných z různých procesů.

Jsou-li podprogramy umístěny uvnitř aktivního procesu, pak musí být přeskakovány, protože nemohou ležet ani na úplném začátku, ani na úplném konci procesu. To vyžaduje navíc instrukci JMP a návěští L a dochází ke snížení přehlednosti programu.

Příklad 10.11

```
P 0
:
CAL    podprogram    ;volání podprogramu
:
E 0
;
P 10
:
CAL    podprogram    ;volání podprogramu
:
E 10
;
P 60
podprogram:
:
RET
E 60
```

11. SOUBOR INSTRUKCÍ

Centrální jednotky PLC TECOMAT dané řady obsahují soubor instrukcí různého rozsahu podle výkonu centrální jednotky.

Instrukční soubor řady E

Centrální jednotky řady E mají zásobník šířky 16 bitů. Obsahují soubor instrukcí, jehož součástí jsou:

- ◆ bitové logické operace
- ◆ základní operace čítačů a časovačů
- ◆ základní organizační instrukce a přechody v programu
- ◆ porovnání šířky 16 bitů
- ◆ jednosmyčkové řízení

Instrukční soubor řady M a S

Centrální jednotky řady M a S mají zásobník šířky 16 bitů. Obsahují soubor instrukcí, jehož součástí jsou oproti řadě E navíc:

- ◆ logické operace šířky 8 a 16 bitů
- ◆ rozšířené operace čítačů, časovačů, posuvných registrů
- ◆ aritmetické instrukce, převody a porovnání šířky 16 bitů
- ◆ rozšířené organizační instrukce, přechody v programech
- ◆ tabulkové instrukce nad tabulkami T a nad zápisníkovou pamětí
- ◆ instrukce sekvenčního řadiče
- ◆ instrukce realizující soubor logických operací, včetně spočtení jedničkových bitů v operandu šířky 16 bitů
- ◆ systém obsahuje 8 uživatelských zásobníků a instrukce pro jejich přepínání - vhodné pro předávání více parametrů mezi funkcemi, které nenásledují bezprostředně po sobě, uložení okamžitého stavu zásobníku, apod.
- ◆ automatická konverze délky operandů a mezivýsledků při kombinaci instrukcí s různými typy operandů nebo logických instrukcí s aritmetickými
- ◆ soubor systémových proměnných, ve kterých je realizován systémový čas, systémové časové jednotky a jejich hrany, komunikační proměnné, příznakové a povelové proměnné, systémová hlášení
- ◆ ke zkrácení doby odezvy i k snazšímu programování přispívá tzv. multiprogramování (vícesmyčkové řízení) včetně přerušovacích procesů
- ◆ uživatelské instrukce USI realizují optimálním způsobem (na úrovni instrukcí mikroprocesoru) složité speciální úlohy

Instrukční soubor řady D a B

Centrální jednotky řady D a B mají zásobník šířky 16 bitů. Obsahují soubor instrukcí, jehož součástí jsou oproti řadám M a S navíc:

- ◆ logické operace šířky 32 bitů
- ◆ aritmetické instrukce, převody a porovnání šířky 32 bitů
- ◆ podmíněné skoky podle příznaků porovnání
- ◆ aritmetické instrukce ve formátu s pohyblivou řádovou čárkou (single precision floating point - typ real)

- ◆ rozšířené tabulkové instrukce s tabulkami velkého rozsahu
- ◆ tabulkové instrukce se strukturovaným přístupem
- ◆ instrukce PID regulátoru
- ◆ systémové instrukce pro optimalizaci práce centrální jednotky a podporu speciálních služeb

Instrukční soubor řady C

Centrální jednotky řady C mají zásobník šířky 32 bitů. Obsahují soubor instrukcí, jehož součástí jsou oproti řadám D a B navíc:

- ◆ instrukce čtení a zápisu s nepřímým adresováním
- ◆ aritmetické instrukce, převody a porovnání se znaménkem (záporná čísla ve dvojkovém doplňku)
- ◆ aritmetické instrukce ve formátu s pohyblivou řádovou čárkou s dvojnásobnou přesností (double precision floating point - typ lreal)
- ◆ tabulkové instrukce s tabulkami obsahujícími položky šířky 32 bitů
- ◆ čítače, posuvné registry a krokový řadič šířky 32 bitů
- ◆ limitní funkce, posun hodnoty
- ◆ instrukce obsluhy operátorského panelu

Úplný popis instrukčního souboru je uveden v příručkách Soubor instrukcí PLC TECOMAT - model 16 bitů (řady B, D, E, M, S), obj. č. TXV 001 05.01, a Soubor instrukcí PLC TECOMAT - model 32 bitů (řada C), obj. č. TXV 004 01.01.

12. UŽIVATELSKÉ INSTRUKCE

Uživatelská instrukce USI je uživatelem nadefinovaná instrukce PLC, která je určena k takovým operacím, jejichž realizace ostatními instrukcemi PLC by byla obtížná až nemožná. To znamená, že instrukční soubor PLC TECOMAT není uzavřen a může být podle potřeby doplňován novými instrukcemi bez nároků na změnu systémového softwaru centrálních jednotek.

Uživatelské instrukce podporují všechny centrální jednotky kromě řady E.

12.1. POUŽITÍ USI V UŽIVATELSKÉM PROGRAMU

Deklarace #usi

Instrukce USI vyžadují před použitím v uživatelském programu definici, která obsahuje informace o tom, v kterém adresáři a v kterém souboru je uložen binární kód instrukce USI. K tomu slouží direktiva *#usi*. Její syntaxe je následující:

```
#usi [index,]jméno_instrukce = jméno_souboru
```

index - nepovinné číslo, které nastavuje index vytvářené instrukce.
Po vnucení indexu instrukce uvedením tohoto čísla jsou dalším definovaným instrukcím přiřazovány indexy v rostoucím pořadí počínaje číslem *index+1*. Pokud je položka *index* vynechána, je následujícím instrukcím automaticky přiřazován vzrůstající index.

jméno_instrukce - nepovinné symbolické jméno uživatelské instrukce

jméno_souboru - jméno binárního diskového souboru, který obsahuje výkonnou část uživatelské instrukce (může obsahovat celou cestu)

Další použití je stejné jako u standardních instrukcí PLC.

Příklad 12.1

```
#usi UserInstr = instfile          ;definice USI
;
P 0
    :
    USI UserInstr          ;použití USI v programu
    :
E 0
```

12.2. USI PRO JEDNOTLIVÉ ŘADY CENTRÁLNÍCH JEDNOTEK

Strojové kódy instrukcí USI nejsou mezi jednotlivými řadami centrálních jednotek přenosné z důvodu různých použitých procesorů a odlišného mapování systémové paměti.

Součástí každého strojového kódu instrukcí USI je označení řady centrální jednotky, pro kterou je kód určen. Pokud omylem spustíme instrukci USI na jiné centrální jednotce, než pro kterou je instrukce určena, jednotka vyhlásí chybu neregulární uživatelská instrukce (\$80 17 pcpc) a program je zastaven.

Součástí instalace prostředí Mosaic je řada již vytvořených instrukcí USI, které je možné využívat ve vlastních programech. Popis funkce těchto instrukcí je též součástí instalace.

Soubory obsahující strojové kódy instrukcí USI mají přípony .ui-, kde poslední písmeno přípony udává řadu centrální jednotky, pro kterou je kód určen (viz tab.12.1). Soubory s příponou *.dll jsou strojové kódy instrukcí USI pro simulátor PLC v prostředí Mosaic.

Tab.12.1 Seznam přípon souborů s kódem instrukcí USI

Řada centrálních jednotek	Přípona souboru
A	.uia
B	.uib
C	.uic
D	.uid
E	nepodporuje
M	.uim
S	.uis
simulátor PLC v prostředí Mosaic	.dll

Práce s instrukcemi USI je podporována překladačem, který navíc podporuje automatické přiřazení přípony binárního souboru v deklaracích *#usi*. V praxi to znamená, že pokud není přípona ve jméně souboru v deklaraci *#usi* uvedena, přiřadí se automaticky podle toho, pro jakou centrální jednotku je kód generován.

12.3. VYTVOŘENÍ VLASTNÍ USI

Deklarace USI se provádí v jazyce C. Při programování se doporučuje vycházet z publikace The C Programming Language autorů Brian W. Kernighan a Dennis M. Ritchie vydané nakladatelstvím Prentice-Hall Inc. Pro zápis funkce je k dispozici hlavička, která popisuje a zpřístupňuje zásobník a zápisník PLC. Vstupní parametry stejně jako výstupy funkce USI mohou být umístěny kdekoliv v těchto strukturách.

Funkce USI musí být uživatelem naprogramována následovně :

```
#include "usi.h"    /* soubor příslušný použité řadě CPU */

void NameUSI( p1, p2)
struct notePLC *p1;
struct accPLC *p2;
{
    :
    tělo funkce USI;
    :
    return;
};
```

Z uvedené deklarace uživatelské funkce USI je zřejmé, že funkci USI jsou systémovým programem PLC předávány ukazatele na struktury *notePLC* (zápisník PLC) a *accPLC* (zásobník PLC). Deklarace těchto struktur obsahuje soubor *usi.h*, jehož výpis je uveden v kap.12.5.

V případě, že program pro instrukci USI je složen z více funkcí, je nutné nejprve deklarovat hlavní funkci, jak ukazuje následující příklad.

```
#include "usi.h"
int pomocna_fce1(); /* prototyp pomocné funkce pro USI */
int pomocna_fce2(); /* prototyp další pomocné funkce */

void funkceUSI(p1,p2)
struct notePLC *p1;
struct accPLC *p2;
```



```
{
  p2->a[0]=pomocna_fce1() + pomocna_fce2();
  return();
}
int pomocna_fce1()
{
  /* pomocný výpočet pro hlavní funkci */
}
int pomocna_fce2()
{
  /* další pomocná funkce */
}
```

12.4. POUŽÍVANÉ PŘEKLADAČE JAZYKA C

Pro překlad USI instrukcí lze použít následující překladače:

Pro CPU řady A, B	Microware OS-9 / 68000 C Compiler Microware Systems Corporation
Pro CPU řady C	GNU 68K C Compiler Free Software Foundation, Inc.
Pro CPU řady D, M, S	Keil C-Compiler-51 Keil Elektronik GmbH

Tyto překladače používají datové typy uvedené v tab.12.2. Z tabulky je vidět, že překladače pro různé typy procesorů používají v případě typu **int** různou velikost vytvářeného objektu a tím zároveň odlišný rozsah zobrazení čísel. Pokud chceme psát funkce použitelné pro všechny řady centrálních jednotek PLC, je proto lepší používat ve funkcích typy deklarované v souboru *usi.h* pomocí direktivy *#typedef*.

Tab.12.2 Datové typy používané překladači jazyka C

Datový typ	Počet bytů		Interní reprezentace
	Microware GNU	Keil	
char	1	1	two's complement binary
unsigned char	1	1	unsigned binary
short	2	2	two's complement binary
unsigned short	2	2	unsigned binary
int	4	2	two's complement binary
unsigned int	4	2	unsigned binary
long	4	4	two's complement binary
float	4	4	binary floating point
pointer to ...	4	3	address

Uživatelské funkce lze psát také v prostředích pro vývoj aplikací do PC v jazyce C (např. C++ Builder), kde lze s výhodou využít ladicích možností, a teprve závěrečný překlad se provede s příslušnými překladači.

Pozor! Pro uživatelské instrukce napsané pro centrální jednotky řad M, S a D platí následující omezení: nelze používat inicializaci lokálních proměnných při deklaraci proměnné, např.:

```
char pole[4] = {{1,2,3,4}};
```

Proměnné musí být inicializovány přiřazením při výpočtu funkce, např.:

```
pole[0]=1; pole[1]=2; pole[2]=3; pole[3]=4;
```

Tato nepříjemnost souvisí s relokací funkce USI při jejím začleňování do uživatelského programu. Nerespektování tohoto postupu může mít nepředvídatelné následky !!!

12.5. PŘÍKLAD VYTVOŘENÍ VLASTNÍ INSTRUKCE USI

```
/* soubor usi.h pro CPU řady D */
typedef unsigned long   long_word;
typedef unsigned short  word;
typedef signed short    signed_word;
typedef unsigned char   byte;
typedef signed char     signed_byte;

/* deklarace struktur pro přístup k zásobníku PLC */
struct accPLC {          /* struktura zásobníku PLC */
    word a[8];           /* jednotlivé vrstvy zásobníku */
};

/* konstanty pro definice velikosti zón zápisníku PLC */
#define MAXX  128 /* počet byte X v zápisníku */
#define MAXY  128 /* počet byte Y v zápisníku */
#define MAXS   64 /* počet byte S v zápisníku */
#define REZS   64 /* rezerva v zóně S */
#define MAXD  256 /* počet byte D v zápisníku */
/* kopírováno z kódu programu */
#define MAXR 8192 /* počet byte R v zápisníku */
struct notePLC { /* struktura zápisníkové paměti PLC */
    u_char x[MAXX], /* obraz vstupů X */
           y[MAXY], /* obraz výstupů Y */
           s[MAXS], /* systémové registry S */
           rs[REZS], /* rezerva pro systémovou zónu */
           d[MAXD], /* kopie dat D z uživatelského programu */
           r[MAXR]; /* uživatelské registry R */
};
/* konec souboru usi.h pro CPU řady D */
```

Uživatelská instrukce MUL16 násobí hodnotu v A0 zásobníku PLC hodnotou z vrstvy A1, výsledek ukládá do dvojvrstvy A01.

Zdrojový text instrukce MUL16:

```
#include "usi.h"

void Mull6(p1, p2) /* binární násobení A0 * A1 = A01 */
{
    struct notePLC *p1;
    struct accPLC *p2;
    {
        union {
            long_word l;
            word w[2];
        } result;

        result.l=((long_word)p2->a[0])*((long_word)p2->a[1]);
        p2->a[0]=result.w[1]; /* A0 nižší řády výsledku */
        p2->a[1]=result.w[0]; /* A1 vyšší řády výsledku */
        return;
    }
}
```

Program funkce MUL16 je potřeba přeložit kompilátorem jazyka C do strojového kódu příslušného procesoru.

12.6. PŘÍKLAD POUŽITÍ INSTRUKCE USI

Zařazení uživatelské instrukce do uživatelského programu PLC provedeme v programu xPRO direktivou *#usi*, která přiřazuje symbolickému jménu instrukce soubor s binárním kódem instrukce.

Příklad 12.2

```
#usi MUL16 = mul16      ;definice USI s automatickým přiřazením přípony
                        ;souboru s kódem instrukce podle řady CPU

#reg uint   va, vb
#reg udint  vc
;
P 0
    LD      va          ;načti první činitel
    LD      vb          ;načti druhý činitel
    USI     MUL16        ;A01 = a.b
    WR      vc          ;ulož výsledek
E 0
```

12.7. POZNÁMKY NA ZÁVĚR

Při psaní uživatelských instrukcí je nutné respektovat několik málo pravidel, které vyplývají ze způsobu činnosti PLC. Zejména je potřebné uvědomit si následující omezení:

Čas výpočtu USI - jakákoliv uživatelem nadefinovaná funkce USI bude stejně jako ostatní instrukce PLC v okamžiku zavolání spotřebovávat čas procesoru PLC. O tento čas se prodlouží doba vykonávání jednoho cyklu PLC. U funkcí, které potřebují pro získání výsledku velký počet iterací, hrozí nebezpečí neúměrného prodloužení doby cyklu PLC a výrazného zpomalení jeho reakce na jednotkovou změnu na vstupu, což může vést až k zastavení činnosti PLC z důvodu překročení maximální doby cyklu. Takové funkce lze programovat tak, že při každém volání USI je proveden pouze definovaný počet iterací, aby spotřebovaný čas procesoru byl únosný. Takto naprogramovaná USI pak produkuje výsledek jednou za několik cyklů PLC, což je v řadě případů přijatelné řešení.

Nároky na paměť - strojový kód instrukce USI je součástí uživatelského programu stejně jako struktury pro jeho zpřístupnění procesoru PLC. Z tohoto pohledu je dobré vyvarovat se při programování takových algoritmů, které vedou k rozsáhlým strojovým kódům. To se týká především používání knihoven jazyka C. Zkušený programátor a dobře optimalizující kvalitní kompilátor jsou v této otázce zajisté přínosem.

Typ procesoru PLC - použitý procesor přináší určitá omezení týkající se např. velikosti operační paměti, prostoru pro stack, využití hw prostředků procesoru atd. Z tohoto pohledu je vhodné konzultovat USI s vývojovými pracovníky TECO a.s.

A. PŘÍLOHA

A.1. DOBY VÝKONU INSTRUKCÍ V CENTRÁLNÍ JEDNOTCE CPM-1E TECOMAT NS950

Doby výkonu instrukcí nezahrnují vlivy systémových procesů, které přerušují provádění uživatelského programu. Těmito procesy jsou sériové komunikace a obsluhy některých periferních jednotek. Jejich vlivem dochází k prodloužení doby cyklu.

Přehled použitých symbolů operandů:

Z - zápisník X, Y, S, R

- konstanta

A - bez operandu (pracuje pouze na zásobníku)

n - číselný parametr

Instrukce pro čtení a zápis dat

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])				Význam instrukce
	bool	Z byte usint sint	word uint int	# word uint int	
LD	63 (3)	53 (3)	61 (3)	50 (3)	Čtení přímých dat
LDC	64 (3)	54 (3)	63 (3)	52 (3)	Čtení negovaných dat
WR	67 (3)	50 (3)	57 (3)	-	Zápis přímých dat
WRC	67 (3)	51 (3)	59 (3)	-	Zápis negovaných dat
PUT	77 (3)	60 (3)	66 (3)	-	Podmíněný zápis dat - podmínka splněna
	51	46	46	-	- podmínka nesplněna

Logické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	Z bool	A word uint int	
AND	66 (3)	50 (1)	AND s přímým operandem
ANC	67 (3)	-	AND s negovaným operandem
OR	66 (3)	50 (1)	OR s přímým operandem
ORC	67 (3)	-	OR s negovaným operandem
XOR	66 (3)	50 (1)	XOR s přímým operandem
XOC	67 (3)	-	XOR s negovaným operandem
SET	66 (3)	-	Podmíněné nastavení
RES	67 (3)	-	Podmíněné nulování
LET	79 (3)	-	Impulz od náběžné hrany

Čítače, časovače

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
CTU	186 (3)	Dopředný čítač
CTD	186 (3)	Zpětný čítač
TON	220 (3)	Časovač (zpožděný přitah)

Aritmetické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
	A word uint int	
EQ	92 (1)	Porovnání (rovnost)

Operace se zásobníky

Mnemo-kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
POP n	42 + 7n (3)	Posun (rotace) zásobníku zpět o n úrovní

Instrukce skoků a volání

Mnemo-kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	průchod	skok	
JMP	-	114 (3)	Nepodmíněný skok
JMD	42	120 (3)	Skok podmíněný nenulovostí výsledku
L	36 (3)	-	Návěští n (cíl skoků a volání)

Organizační instrukce

Mnemo-kód	Doba výkonu [μs] (Délka kódu [B])					Význam instrukce
	průchod	skok	P41 - P49	P50 - P57	P62 - P64	
P	151 (3)	-	76	73	143	Začátek procesu
E	-	54 (3)	229	77	54	Nepodmíněný konec procesu
NOP	36 (3)	-	-	-	-	Prázdná operace

A.2. DOBY VÝKONU INSTRUKCÍ V CENTRÁLNÍ JEDNOTCE CPM-1M TECOMAT NS950

Doby výkonu instrukcí nezahrnují vlivy systémových procesů, které přerušují provádění uživatelského programu. Těmito procesy jsou sériové komunikace a obsluhy některých periferních jednotek. Jejich vlivem dochází k prodloužení doby cyklu.

Přehled použitých symbolů operandů:

Z - zápisník X, Y, S, D, R

T - tabulky

- konstanta

A - bez operandu (pracuje pouze na zásobníku)

U - fyzická adresa periferní jednotky

n - číselný parametr

Instrukce pro čtení a zápis dat

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])						Význam instrukce
	bool	Z byte usint sint	word uint int	# word uint int	U byte usint sint	word uint int	
LD	63 (3)	53 (3)	61 (3)	50 (3)	81 (3)	109 (3)	Čtení přímých dat
LDC	64 (3)	54 (3)	63 (3)	52 (3)	-	-	Čtení negovaných dat
WR	67 (3)	50 (3)	57 (3)	-	80 (3)	113 (3)	Zápis přímých dat
WRC	67 (3)	51 (3)	59 (3)	-	-	-	Zápis negovaných dat
PUT	77 (3)	60 (3)	66 (3)	-	-	-	Podmíněný zápis dat
	51	46	46	-	-	-	- podmínka splněna - podmínka nesplněna

Logické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])				Význam instrukce
	bool	Z byte usint sint	# word uint int	A word uint int	
AND	66 (3)	53 (3)	49 (3)	50 (1)	AND s přímým operandem
ANC	67 (3)	54 (3)	-	-	AND s negovaným operandem
OR	66 (3)	49 (3)	49 (3)	50 (1)	OR s přímým operandem
ORC	67 (3)	50 (3)	-	-	OR s negovaným operandem
XOR	66 (3)	49 (3)	49 (3)	50 (1)	XOR s přímým operandem
XOC	67 (3)	50 (3)	-	-	XOR s negovaným operandem
NEG	-	-	-	42 (1)	Negace vrcholu zásobníku
SET	66 (3)	50 (3)	-	-	Podmíněné nastavení
RES	67 (3)	53 (3)	-	-	Podmíněné nulování
LET	79 (3)	55 (3)	-	-	Impulz od náběžné hrany
FLG	-	-	-	192 (1)	Logické funkce A0
STK	-	-	-	152 (1)	Sklopení úrovně zásobníku do A0
ROL n	-	-	-	81+10n (3)	Rotace čísla vlevo
SWP	-	-	-	39 (1)	Záměna horního a dolního bytu v A0

Čítače, posuvné registry, časovače, krokový řadič

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
CTU	186 (3)	Dopředný čítač
CTD	186 (3)	Zpětný čítač
CNT	219 (3)	Obousměrný čítač
SFL	190 (3)	Posuvný registr vlevo
SFR	190 (3)	Posuvný registr vpravo
TON	220 (3)	Časovač (zpožděný přítah)
TOF	221 (3)	Časovač (zpožděný odpad)
RTO	235 (3)	Integrovaný časovač, měřič času
IMP	222 (3)	Časovač - generátor impulzu zadané délky
STE	129 (3)	Krokový řadič (stepper) - změna stavu
	106	- stav nezměněn

Aritmetické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])						Význam instrukce
	Z		#		A		
	byte usint	word uint	byte usint	word uint	byte usint	word uint	
ADD	-	99 (3)	-	88 (3)	-	88 (1)	Sčítání s přenosem
SUB	-	101 (3)	-	90 (3)	-	90 (1)	Odčítání s přenosem
MUL	59 (3)	-	50 (3)	-	51 (1)	-	Násobení
DIV	74 (3)	-	65 (3)	-	66 (1)	-	Dělení
INR	-	-	-	-	-	79 (1)	Inkrementace (+ 1)
DCR	-	-	-	-	-	79 (1)	Dekrementace (– 1)
EQ	-	103 (3)	-	92 (3)	-	92 (1)	Porovnání (rovnost)
LT	-	103 (3)	-	92 (3)	-	92 (1)	Porovnání (menší než)
GT	-	104 (3)	-	93 (3)	-	93 (1)	Porovnání (větší než)
BIN	-	-	-	-	-	80 (1)	Převod čísla do binárního formátu
BCD	-	-	-	-	-	226 (1)	Převod čísla do BCD

Operace se zásobníky

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
POP n	42 + 7n (3)	Posun (rotace) zásobníku zpět o n úrovní
NXT	375 (3)	Aktivace následujícího zásobníku v řadě
PRV	375 (3)	Aktivace předcházejícího zásobníku v řadě
CHG	356 (3)	Aktivace zvoleného zásobníku bez zálohování S0 a S1
CHGS	373 (3)	Aktivace zvoleného zásobníku se zálohováním S0 a S1

Instrukce skoků a volání

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	průchod	skok	
JMP	-	114 (3)	Nepodmíněný skok
JMD	42	120 (3)	Skok podmíněný nenulovostí výsledku
JMC	42	120 (3)	Skok podmíněný nulovostí výsledku
JMI	-	115 (1)	Skok na nepřímý cíl
CAL	-	138 (3)	Nepodmíněné volání podprogramu
CAD	42	144 (3)	Volání podprogramu podmíněné nenulovostí výsledku
CAC	42	144 (3)	Volání podprogramu podmíněné nulovostí výsledku
CAI	-	137 (1)	Volání podprogramu nepřímého cíle
RET	-	48 (1)	Nepodmíněný návrat z podprogramu
RED	40	54 (1)	Návrat z podprogramu podmíněný nenulovostí výsledku
REC	40	54 (1)	Návrat z podprogramu podmíněný nulovostí výsledku
L	36 (3)	-	Návěští n (cíl skoků a volání)

Příloha - Doby výkonu instrukcí

Organizační instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])					Význam instrukce
	průchod	skok	P41 - P49	P50 - P57	P62 - P64	
P	151 (3)	-	76	73	143	Začátek procesu
E	-	54 (3)	229	77	54	Nepodmíněný konec procesu
ED	42 (1)	65	240	88	65	Konec procesu při nenulovém výsledku
EC	42 (1)	65	240	88	65	Konec procesu při nulovém výsledku
EOC	-	35 (1)	-	-	-	Konec cyklu
NOP	36 (3)	-	-	-	-	Prázdná operace
BP	-	189 (3)	-	-	-	Ladící bod

Tabulkové instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])						Význam instrukce
	Z			T			
	bool	byte usint sint	word uint int	bool	byte usint sint	word uint int	
LTB	136 (3)	118 (3)	131 (3)	258 (3)	225 (3)	250 (3)	Čtení položky z tabulky
WTB	151 (3)	126 (3)	140 (3)	-	-	-	Zápis položky do tabulky
LMS	-	-	-	-	-	248 (3)	Čtení položky sekvenčně
WMS	-	-	-	-	-	362 (3)	Zápis položky sekvenčně
						+11	- časová přírážka na 1 položku tabulky
FTB	-	68 (3)	75 (3)	-	182 (3)	203 (3)	Hledání položky v tabulce
		+49	+55		+39	+46	- čas. přírážka na 1 prohledávanou pol.
FTM	-	66 (3)	66 (3)	-	192 (3)	192 (3)	Hledání části položky v tabulce
		+61	+90		+51	+80	- čas. přírážka na 1 prohledávanou pol.
FTS	-	71 (3)	64 (3)	-	184 (3)	191 (3)	Zařazení položky podle tabulky
		+51	+74		+41	+64	- čas. přírážka na 1 prohledávanou pol.

Blokové operace

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	Z	T	
SRC	99 (3)	195 (3)	Specifikace zdroje dat pro přesun
MOV	188 (3)	392 (3)	Přesun bloku dat
	+17	+28	- časová přírážka na 1 položku bloku
FIL	69 (3)	-	Naplnění bloku konstantou
	+48		- časová přírážka na 1 položku bloku

A.3. DOBY VÝKONU INSTRUKCÍ V CENTRÁLNÍ JEDNOTCE CPM-2S TECOMAT NS950

Doby výkonu instrukcí nezahrnují vlivy systémových procesů, které přerušují provádění uživatelského programu. Těmito procesy jsou sériové komunikace a obsluhy některých periferních jednotek. Jejich vlivem dochází k prodloužení doby cyklu.

Přehled použitých symbolů operandů:

Z - zápisník X, Y, S, D, R

T - tabulky

- konstanta

A - bez operandu (pracuje pouze na zásobníku)

U - fyzická adresa periferní jednotky

n - číselný parametr

Instrukce pro čtení a zápis dat

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])						Význam instrukce
		Z		#	U		
	bool	byte usint sint	word uint int	word uint int	byte usint sint	word uint int	
LD	12,8 (3)	12,8 (3)	13,9 (3)	11,2 (3)	18,3 (3)	25,5 (3)	Čtení přímých dat
LDC	13,0 (3)	13,0 (3)	14,3 (3)	11,6 (3)	-	-	Čtení negovaných dat
WR	13,4 (3)	11,6 (3)	13,0 (3)	-	17,5 (3)	25,7 (3)	Zápis přímých dat
WRC	13,4 (3)	11,8 (3)	13,4 (3)	-	-	-	Zápis negovaných dat
PUT	15,0 (3)	13,2 (3)	14,6 (3)	-	-	-	Podmíněný zápis dat
	8,9	10,5	10,5	-	-	-	- podmínka splněna - podmínka nesplněna

Logické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])				Význam instrukce
		Z	#	A	
	bool	byte usint sint	word uint int	word uint int	
AND	12,1 (3)	12,5 (3)	11,0 (3)	10,7 (1)	AND s přímým operandem
ANC	12,3 (3)	12,7 (3)	-	-	AND s negovaným operandem
OR	12,5 (3)	11,8 (3)	11,0 (3)	10,7 (1)	OR s přímým operandem
ORC	12,5 (3)	11,9 (3)	-	-	OR s negovaným operandem
XOR	13,0 (3)	11,8 (3)	11,0 (3)	10,7 (1)	XOR s přímým operandem
XOC	13,2 (3)	11,9 (3)	-	-	XOR s negovaným operandem
NEG	-	-	-	9,2 (1)	Negace vrcholu zásobníku
SET	11,9 (3)	11,9 (3)	-	-	Podmíněné nastavení
RES	13,6 (3)	12,5 (3)	-	-	Podmíněné nulování
LET	15,7 (3)	12,8 (3)	-	-	Impulz od náběžné hrany
FLG	-	-	-	44,1 (1)	Logické funkce A0
STK	-	-	-	36,5 (1)	Sklopení úrovně zásobníku do A0
ROL n	-	-	-	17,5+2n (3)	Rotace čísla vlevo
SWP	-	-	-	8,7 (1)	Záměna horního a dolního bytu v A0

Příloha - Doby výkonu instrukcí

Čítače, posuvné registry, časovače, krokový řadič

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
CTU	40,3 (3)	Dopředný čítač
CTD	40,3 (3)	Zpětný čítač
CNT	48,3 (3)	Obousměrný čítač
SFL	41,4 (3)	Posuvný registr vlevo
SFR	41,4 (3)	Posuvný registr vpravo
TON	47,0 (3)	Časovač (zpožděný přítah)
TOF	47,2 (3)	Časovač (zpožděný odpad)
RTO	50,3 (3)	Integrovaný časovač, měřič času
IMP	46,7 (3)	Časovač - generátor impulzu zadané délky
STE	29,7 (3)	Krokový řadič (stepper) - změna stavu
	23,7	- stav nezměněn

Aritmetické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])						Význam instrukce
	Z		#		A		
	byte usint	word uint	byte usint	word uint	byte usint	word uint	
ADD	-	19,5 (3)	-	18,1 (3)	-	16,8 (1)	Sčítání s přenosem
SUB	-	20,1 (3)	-	18,6 (3)	-	17,4 (1)	Odčítání s přenosem
MUL	13,6 (3)	-	11,4 (3)	-	11,0 (1)	-	Násobení
DIV	15,9 (3)	-	14,5 (3)	-	13,4 (1)	-	Dělení
INR	-	-	-	-	-	14,6 (1)	Inkrementace (+ 1)
DCR	-	-	-	-	-	14,6 (1)	Dekrementace (– 1)
EQ	-	21,0 (3)	-	19,5 (3)	-	18,3 (1)	Porovnání (rovnost)
LT	-	21,0 (3)	-	19,5 (3)	-	18,3 (1)	Porovnání (menší než)
GT	-	20,8 (3)	-	19,4 (3)	-	18,1 (1)	Porovnání (větší než)
BIN	-	-	-	-	-	17,5 (1)	Převod čísla do binárního formátu
BCD	-	-	-	-	-	47,6 (1)	Převod čísla do BCD

Operace se zásobníky

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
POP n	10,7 + 1,4n (3)	Posun (rotace) zásobníku zpět o n úrovní
NXT	92,4 (3)	Aktivace následujícího zásobníku v řadě
PRV	92,4 (3)	Aktivace předcházejícího zásobníku v řadě
CHG	87,9 (3)	Aktivace zvoleného zásobníku bez zálohování S0 a S1
CHGS	92,1 (3)	Aktivace zvoleného zásobníku se zálohováním S0 a S1

Instrukce skoků a volání

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	průchod	skok	
JMP	-	22,2 (3)	Nepodmíněný skok
JMD	10,1	23,5 (3)	Skok podmíněný nenulovostí výsledku
JMC	10,1	23,5 (3)	Skok podmíněný nulovostí výsledku
JMI	-	21,7 (1)	Skok na nepřímý cíl
CAL	-	25,3 (3)	Nepodmíněné volání podprogramu
CAD	10,1	26,6 (3)	Volání podprogramu podmíněné nenulovostí výsledku
CAC	10,1	26,6 (3)	Volání podprogramu podmíněné nulovostí výsledku
CAI	-	24,2 (1)	Volání podprogramu nepřímého cíle
RET	-	10,3 (1)	Nepodmíněný návrat z podprogramu
RED	9,0	11,6 (1)	Návrat z podprogramu podmíněný nenulovostí výsledku
REC	9,0	11,6 (1)	Návrat z podprogramu podmíněný nulovostí výsledku
L	8,9 (3)	-	Návěští n (cíl skoků a volání)

Organizační instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])					Význam instrukce
	průchod	skok	P41 - P49	P50 - P57	P62 - P64	
P	30,6 (3)	-	14,8	14,1	28,2	Začátek procesu
E	-	11,2 (3)	53,2	51,4	11,2	Nepodmíněný konec procesu
ED	9,0 (1)	12,8	54,8	53,0	12,8	Konec procesu při nenulovém výsledku
EC	9,0 (1)	12,8	54,8	53,0	12,8	Konec procesu při nulovém výsledku
EOC	-	8,1 (1)	-	-	-	Konec cyklu
NOP	8,9 (3)	-	-	-	-	Prázdná operace
BP	-	72,0 (3)	-	-	-	Ladicí bod

Tabulkové instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])						Význam instrukce
	Z			T			
	bool	byte usint sint	word uint int	bool	byte usint sint	word uint int	
LTB	30,6 (3)	25,1 (3)	27,5 (3)	46,8 (3)	37,6 (3)	40,5 (3)	Čtení položky z tabulky
WTB	33,1 (3)	26,0 (3)	28,8 (3)	-	-	-	Zápis položky do tabulky
LMS	-	-	-	-	-	47,9 (3)	Čtení položky sekvenčně
WMS	-	-	-	-	-	61,7 (3)	Zápis položky sekvenčně
FTB	-	19,7 (3) +3,1	22,1 (3) +3,6	-	32,2 (3) +3,1	34,7 (3) +3,6	- čas. přírážka na 1 položku tabulky Hledání položky v tabulce
FTM	-	20,3 (3) +4,3	21,9 (3) +7,4	-	32,9 (3) +4,3	35,1 (3) +7,4	- čas. přírážka na 1 prohledávanou pol. Hledání části položky v tabulce
FTS	-	19,7 (3) +3,6	21,0 (3) +6,3	-	32,7 (3) +3,6	33,6 (3) +6,3	Zařazení položky podle tabulky - čas. přírážka na 1 prohledávanou pol.

Blokové operace

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	Z	T	
SRC	21,3 (3)	35,4 (3)	Specifikace zdroje dat pro přesun
MOV	49,0 (3) +3,1	74,3 (3) +5,4	Přesun bloku dat - časová přírážka na 1 položku bloku
FIL	20,4 (3) +2,2	-	Naplnění bloku konstantou - časová přírážka na 1 položku bloku

A.4. DOBY VÝKONU INSTRUKCÍ V CENTRÁLNÍCH JEDNOTKÁCH CPM-1D TECOMAT NS950 A TECOMAT TC400, TC500, TC600

Doby výkonu instrukcí nezahrnují vlivy systémových procesů, které přerušují provádění uživatelského programu. Těmito procesy jsou sériové komunikace a obsluhy některých periferních jednotek. Jejich vlivem dochází k prodloužení doby cyklu.

Přehled použitých symbolů operandů:

Z - zápisník X, Y, S, D, R

T - tabulky

- konstanta

A - bez operandu (pracuje pouze na zásobníku)

U - fyzická adresa periferní jednotky

n - číselný parametr

Instrukce pro čtení a zápis dat

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])								Význam instrukce
	bool	Z byte usint sint	word uint int	dword udint dint real	# word uint int	dword udint dint real	U byte usint sint	word uint int	
LD	12,8 (3)	12,8 (3)	13,9 (3)	18,6 (4)	11,2 (3)	-	18,3 (3)	25,5 (3)	Čtení přímých dat
LDL	-	-	-	-	-	16,1 (6)	-	-	Čtení přímých dat
LDC	13,0 (3)	13,0 (3)	14,3 (3)	19,4 (4)	11,6 (3)	-	-	-	Čtení negovaných dat
WR	13,4 (3)	11,6 (3)	13,0 (3)	17,9 (4)	-	-	17,5 (3)	25,7 (3)	Zápis přímých dat
WRC	13,4 (3)	11,8 (3)	13,4 (3)	18,6 (4)	-	-	-	-	Zápis negovaných dat
WRA	-	13,6 (4)	15,0 (4)	19,0 (4)	-	-	-	-	Zápis přímých dat s alternací
PUT	15,0 (3)	13,2 (3)	14,6 (3)	19,5 (4)	-	-	-	-	Podmíněný zápis dat
	8,9	10,5	10,5	11,4	-	-	-	-	- podmínka splněna - podmínka nesplněna

Logické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])							Význam instrukce
	bool	Z byte usint sint	word uint int	word uint int	# dword udint dint	A word uint int	dword udint dint	
AND	12,1 (3)	12,5 (3)	15,2 (4)	11,0 (3)	-	10,7 (1)	-	AND s přímým operandem
ANL	-	-	-	-	17,0 (6)	-	15,4 (2)	AND s přímým operandem
ANC	12,3 (3)	12,7 (3)	15,6 (4)	-	-	-	-	AND s negovaným operandem
OR	12,5 (3)	11,8 (3)	15,2 (4)	11,0 (3)	-	10,7 (1)	-	OR s přímým operandem
ORL	-	-	-	-	17,0 (6)	-	15,4 (2)	OR s přímým operandem
ORC	12,5 (3)	11,9 (3)	15,6 (4)	-	-	-	-	OR s negovaným operandem
XOR	13,0 (3)	11,8 (3)	15,2 (4)	11,0 (3)	-	10,7 (1)	-	XOR s přímým operandem
XOL	-	-	-	-	17,0 (6)	-	15,4 (2)	XOR s přímým operandem
XOC	13,2 (3)	11,9 (3)	15,6 (4)	-	-	-	-	XOR s negovaným operandem
NEG	-	-	-	-	-	9,2 (1)	-	Negace vrcholu zásobníku
NGL	-	-	-	-	-	-	13,0 (2)	Negace vrcholu zásobníku
SET	11,9 (3)	11,9 (3)	14,6 (4)	-	-	-	-	Podmíněné nastavení
RES	13,6 (3)	12,5 (3)	15,7 (4)	-	-	-	-	Podmíněné nulování
LET	15,7 (3)	12,8 (3)	16,5 (4)	-	-	-	-	Impulz od náběžné hrany
BET	15,4 (3)	13,6 (4)	16,1 (4)	-	-	-	-	Impulz od libovolné hrany
FLG	-	-	-	-	-	44,1 (1)	-	Logické funkce A0
STK	-	-	-	-	-	36,5 (1)	-	Sklopení úrovně zásobníku do A0
ROL n	-	-	-	-	-	17,5+2n (3)	-	Rotace čísla vlevo
ROR n	-	-	-	-	-	18,1+2n (3)	-	Rotace čísla vpravo
SWP	-	-	-	-	-	8,7 (1)	-	Záměna horního a dolního bytu v A0
SWL	-	-	-	-	-	-	12,7 (2)	Záměna vrstev A0 a A1

Čítače, posuvné registry, časovače, krokový řadič

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
CTU	40,3 (3)	Dopředný čítač
CTD	40,3 (3)	Zpětný čítač
CNT	48,3 (3)	Obousměrný čítač
SFL	41,4 (3)	Posuvný registr vlevo
SFR	41,4 (3)	Posuvný registr vpravo
TON	47,0 (3)	Časovač (zpožděný přitah)
TOF	47,2 (3)	Časovač (zpožděný odpad)
RTO	50,3 (3)	Integrovaný časovač, měřič času
IMP	46,7 (3)	Časovač - generátor impulzu zadané délky
STE	29,7 (3)	Krokový řadič (stepper) - změna stavu
	23,7	- stav nezměněn

Aritmetické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])									Význam instrukce
	byte usint	Z word uint	dword uint	byte usint	# word uint	dword uint	byte usint	A word uint	dword uint	
ADD	-	19,5 (3)	-	-	18,1 (3)	-	-	16,8 (1)	-	Sčítání s přenosem
ADX	13,6 (4)	14,3 (4)	18,6 (4)	-	-	-	-	-	-	Sčítání
ADL	-	-	-	-	-	17,0 (6)	-	-	16,3 (2)	Sčítání
SUB	-	20,1 (3)	-	-	18,6 (3)	-	-	17,4 (1)	-	Odčítání s přenosem
SUX	14,1 (4)	14,8 (4)	19,5 (4)	-	-	-	-	-	-	Odčítání
SUL	-	-	-	-	-	17,4 (6)	-	-	16,5 (2)	Odčítání
MUL	13,6 (3)	-	-	11,4 (3)	-	-	11,0 (1)	-	-	Násobení
MUD	-	54 (4)	-	-	52 (4)	-	-	51 (2)	-	Násobení
DIV	15,9 (3)	-	-	14,5 (3)	-	-	13,4 (1)	-	-	Dělení
DID	-	314 (4)	-	-	313 (4)	-	-	311 (2)	-	Dělení
INR	12,8 (4)	13,6 (4)	13,6 (4)	-	-	-	-	14,6 (1)	-	Inkrementace (+ 1)
DCR	14,5 (4)	15,4 (4)	15,4 (4)	-	-	-	-	14,6 (1)	-	Dekrementace (- 1)
EQ	-	21,0 (3)	-	-	19,5 (3)	-	-	18,3 (1)	-	Porovnání (rovnost)
LT	-	21,0 (3)	-	-	19,5 (3)	-	-	18,3 (1)	-	Porovnání (menší než)
GT	-	20,8 (3)	-	-	19,4 (3)	-	-	18,1 (1)	-	Porovnání (větší než)
CMP	17,5 (4)	18,8 (4)	24,1 (4)	-	17,5 (4)	-	-	17,5 (2)	-	Porovnání
CML	-	-	-	-	-	23,3 (6)	-	-	21,2 (2)	Porovnání
BIN	-	-	-	-	-	-	-	17,5 (1)	-	Převod čísla do bin. form.
BIL	-	-	-	-	-	-	-	-	173 (2)	Převod čísla do bin. form.
BCD	-	-	-	-	-	-	-	47,6 (1)	-	Převod čísla do BCD
BCL	-	-	-	-	-	-	-	-	314 (2)	Převod čísla do BCD

Operace se zásobníky

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
POP n	10,7 + 1,4n (3)	Posun (rotace) zásobníku zpět o n úrovní
NXT	92,4 (3)	Aktivace následujícího zásobníku v řadě
PRV	92,4 (3)	Aktivace předcházejícího zásobníku v řadě
CHG	87,9 (3)	Aktivace zvoleného zásobníku bez zálohování S0 a S1
CHGS	92,1 (3)	Aktivace zvoleného zásobníku se zálohováním S0 a S1
LAC	23,9 (4)	Načtení hodnoty z vrcholu zvoleného zásobníku
WAC	22,6 (4)	Zápis hodnoty na vrchol zvoleného zásobníku

Příloha - Doby výkonu instrukcí

Instrukce skoků a volání

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	průchod	skok	
JMP	-	22,2 (3)	Nepodmíněný skok
JMD	10,1	23,5 (3)	Skok podmíněný nenulovostí výsledku
JMC	10,1	23,5 (3)	Skok podmíněný nulovostí výsledku
JMI	-	21,7 (1)	Skok na nepřímý cíl
JZ	11,4	24,8 (4)	Skok podmíněný nenulovostí příznaku rovnosti ZR
JNZ	11,4	24,8 (4)	Skok podmíněný nulovostí příznaku rovnosti ZR
JC	11,4	24,8 (4)	Skok podmíněný nenulovostí příznaku přenosu CO
JNC	11,4	24,8 (4)	Skok podmíněný nulovostí příznaku přenosu CO
JS	11,4	24,8 (4)	Skok podmíněný nenulovostí příznaku S1.0
JNS	11,4	24,8 (4)	Skok podmíněný nulovostí příznaku S1.0
CAL	-	25,3 (3)	Nepodmíněné volání podprogramu
CAD	10,1	26,6 (3)	Volání podprogramu podmíněné nenulovostí výsledku
CAC	10,1	26,6 (3)	Volání podprogramu podmíněné nulovostí výsledku
CAI	-	24,2 (1)	Volání podprogramu nepřímého cíle
RET	-	10,3 (1)	Nepodmíněný návrat z podprogramu
RED	9,0	11,6 (1)	Návrat z podprogramu podmíněný nenulovostí výsledku
REC	9,0	11,6 (1)	Návrat z podprogramu podmíněný nulovostí výsledku
L	8,9 (3)	-	Návěští n (cíl skoků a volání)

Organizační instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])					Význam instrukce
	průchod	skok	P41 - P49	P50 - P57	P62 - P64	
P	30,6 (3)	49,0 *	14,8	14,1	28,2	Začátek procesu (*skok na náv. dané SEQ)
E	-	11,2 (3)	53,2	51,4	11,2	Nepodmíněný konec procesu
ED	9,0 (1)	12,8	54,8	53,0	12,8	Konec procesu při nenulovém výsledku
EC	9,0 (1)	12,8	54,8	53,0	12,8	Konec procesu při nulovém výsledku
EOC	-	8,1 (1)	-	-	-	Konec cyklu
NOP	8,9 (3)	-	-	-	-	Prázdná operace
BP	-	72,0 (3)	-	-	-	Ladící bod
SEQ	14,5 (3)	18,8	-	-	-	Podmíněné přerušení procesu

Tabulkové instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])						Význam instrukce
	Z			T			
	bool	byte usint sint	word uint int	bool	byte usint sint	word uint int	
LTB	30,6 (3)	25,1 (3)	27,5 (3)	46,8 (3)	37,6 (3)	40,5 (3)	Čtení položky z tabulky
WTB	33,1 (3)	26,0 (3)	28,8 (3)	52,3 (4)	45,0 (4)	50,1 (4)	Zápis položky do tabulky
LMS	-	-	-	-	-	47,9 (3)	Čtení položky sekvenčně
WMS	-	-	-	-	-	61,7 (3)	Zápis položky sekvenčně
FTB	23,0 (4)	19,7 (3)	22,1 (3)	38,7 (4)	32,2 (3)	34,7 (3)	Hledání položky v tabulce
	+4,1	+3,1	+3,6	+4,1	+3,1	+3,6	- čas. přiřázka na 1 prohledávanou pol.
FTM	-	20,3 (3)	21,9 (3)	-	32,9 (3)	35,1 (3)	Hledání části položky v tabulce
		+4,3	+7,4		+4,3	+7,4	- čas. přiřázka na 1 prohledávanou pol.
FTS	-	19,7 (3)	21,0 (3)	-	32,7 (3)	33,6 (3)	Zařazení položky podle tabulky
		+3,6	+6,3		+3,6	+6,3	- čas. přiřázka na 1 prohledávanou pol.

Blokové operace

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])			Význam instrukce
	Z	T	A	
SRC	21,3 (3)	35,4 (3)	-	Specifikace zdroje dat pro přesun
MOV	49,0 (3) +3,1	74,3 (3) +5,4	-	Přesun bloku dat - časová přiřázka na 1 položku bloku
MTN	-	-	35,6 (2) +3,1	Přesun tabulky do zápisníku - časová přiřázka na 1 položku tabulky
MNT	-	-	40,5 (2) +4,0	Naplnění tabulky ze zápisníku - časová přiřázka na 1 položku tabulky
FIL	20,4 (3) +2,2	-	-	Naplnění bloku konstantou - časová přiřázka na 1 položku bloku

Operace se strukturovanými tabulkami

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
LDS	47,6 (2) +3,1	Čtení položky ze strukturované tabulky T - časová přírážka na 1 byte položky
WRS	57,1 (2) +5,2	Zápis položky do strukturované tabulky T - časová přírážka na 1 byte položky
FIS	28,0 (2) +2,2	Plnění položky strukturované tabulky v zápisníku - časová přírážka na 1 byte položky
FIT	51,4 (2) +4,3	Plnění položky strukturované tabulky T - časová přírážka na 1 byte položky
FNS	23,9 (2) +4,3	Hledání položky strukturované tabulky v zápisníku - časová přírážka na 1 položku
FNT	36,2 (2) +6,9	Hledání položky strukturované tabulky T - časová přírážka na 1 položku

Aritmetické instrukce v plovoucí řádové čárce (časové údaje jsou orientační, závisí na vstupní hodnotě)

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])			Význam instrukce
	Z	#	A	
ADF	83 (4)	81 (6)	79 (2)	Sčítání
SUF	83 (4)	81 (6)	79 (2)	Odčítání
MUF	125 (4)	123 (6)	121 (2)	Násobení
DIF	332 (4)	329 (6)	328 (2)	Dělení
CMF	54 (4)	52 (6)	50 (2)	Porovnání
CEI	-	-	550 (2)	Zaokrouhlení nahoru
FLO	-	-	520 (2)	Zaokrouhlení dolů
ABS	-	-	11,6 (2)	Absolutní hodnota
LOG	-	-	1580 (2)	Dekadický logaritmus
LN	-	-	1580 (2)	Přirozený logaritmus
EXP	-	-	4200 (2)	Exponenciální funkce
POW	-	-	4200 (2)	Obecná mocnina
SQR	-	-	710 (2)	Druhá odmocnina
HYP	-	-	980 (2)	Euklidovská vzdálenost
SIN	-	-	1320 (2)	Sinus
ASN	-	-	2590 (2)	Arc sinus
COS	-	-	1650 (2)	Cosinus
ACS	-	-	2770 (2)	Arc cosinus
TAN	-	-	2410 (2)	Tangens
ATN	-	-	1650 (2)	Arc tangens
UWF	-	-	40 ÷ 170 (2)	Převod uint na real
IWF	-	-	40 ÷ 170 (2)	Převod int na real
ULF	-	-	40 ÷ 170 (2)	Převod uint na real
ILF	-	-	40 ÷ 170 (2)	Převod dint na real
UFW	-	-	110 ÷ 200 (2)	Převod real na uint
IFW	-	-	110 ÷ 200 (2)	Převod real na int
UFL	-	-	110 ÷ 200 (2)	Převod real na uint
IFL	-	-	110 ÷ 200 (2)	Převod real na dint

Instrukce regulátoru PID (časové údaje jsou orientační, závisí na zvolených funkcích a vstupních hodnotách)

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
	A	
CNV	900 (2)	Konverze a zpracování dat z analogových jednotek
PID	2000 ÷ 10000 (2)	PID regulátor

Operace se znaky ASCII

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	word uint int	real	
BAS	21,7 (2)	-	Převod čísla na ASCII
ASB	22,4 (2)	-	Převod čísla z ASCII
STF		2000 (2)	Převod ASCII řetězce na real
FST		1000 (2)	Převod real na ASCII řetězec

A.5. DOBY VÝKONU INSTRUKCÍ V CENTRÁLNÍCH JEDNOTKÁCH CPM-1B, CPM-2B TECOMAT NS950

Doby výkonu instrukcí nezahrnují vlivy systémových procesů, které přerušují provádění uživatelského programu. Těmito procesy jsou sériové komunikace a obsluhy některých periferních jednotek. Jejich vlivem dochází k prodloužení doby cyklu.

Přehled použitých symbolů operandů:

Z - zápisník X, Y, S, D, R

T - tabulky

- konstanta

A - bez operandu (pracuje pouze na zásobníku)

U - fyzická adresa periferní jednotky

n - číselný parametr

Instrukce pro čtení a zápis dat

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])								Význam instrukce
	bool	byte usint sint	word uint int	dword udint dint real	word uint int	dword udint dint real	byte usint sint	word uint int	
LD	4,0 (4)	3,2 (4)	4,2 (4)	5,0 (4)	2,8 (4)	-	200 (4)	210 (4)	Čtení přímých dat
LDL	-	-	-	-	-	3,6 (6)	-	-	Čtení přímých dat
LDC	4,0 (4)	3,4 (4)	4,4 (4)	5,2 (4)	2,9 (4)	-	-	-	Čtení negovaných dat
WR	3,1 (4)	2,1 (4)	3,7 (4)	4,2 (4)	-	-	130 (4)	130 (4)	Zápis přímých dat
WRC	3,1 (4)	2,3 (4)	3,9 (4)	4,4 (4)	-	-	-	-	Zápis negovaných dat
WRA	-	4,1 (4)	4,4 (4)	6,0 (4)	-	-	-	-	Zápis přímých dat s alternací
PUT	4,2 (4)	3,2 (4)	4,8 (4)	5,3 (4)	-	-	-	-	Podmíněný zápis dat
	2,8	2,8	2,8	2,8	-	-	-	-	- podmínka splněna - podmínka nesplněna

Logické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])							Význam instrukce
	Z			#		A		
	bool	byte usint sint	word uint int	word uint int	dword udint dint	word uint int	dword udint dint	
AND	3,2 (4)	3,1 (4)	3,5 (4)	2,6 (4)	-	3,2 (2)	-	AND s přímým operandem
ANL	-	-	-	-	3,7 (6)	-	4,9 (2)	AND s přímým operandem
ANC	3,6 (4)	3,3 (4)	3,5 (4)	-	-	-	-	AND s negovaným operandem
OR	3,2 (4)	3,1 (4)	3,5 (4)	2,6 (4)	-	3,2 (2)	-	OR s přímým operandem
ORL	-	-	-	-	3,7 (6)	-	4,9 (2)	OR s přímým operandem
ORC	3,6 (4)	3,3 (4)	3,5 (4)	-	-	-	-	OR s negovaným operandem
XOR	3,2 (4)	3,1 (4)	3,5 (4)	2,6 (4)	-	3,2 (2)	-	XOR s přímým operandem
XOL	-	-	-	-	3,7 (6)	-	4,9 (2)	XOR s přímým operandem
XOC	3,6 (4)	3,3 (4)	3,5 (4)	-	-	-	-	XOR s negovaným operandem
NEG	-	-	-	-	-	2,6 (2)	-	Negace vrcholu zásobníku
NGL	-	-	-	-	-	-	3,6 (2)	Negace vrcholu zásobníku
SET	3,3 (4)	2,9 (4)	3,3 (4)	-	-	-	-	Podmíněné nastavení
RES	3,3 (4)	2,9 (4)	3,3 (4)	-	-	-	-	Podmíněné nulování
LET	5,0 (4)	3,5 (4)	4,3 (4)	-	-	-	-	Impulz od náběžné hrany
BET	4,8 (4)	3,9 (4)	4,3 (4)	-	-	-	-	Impulz od libovolné hrany
FLG	-	-	-	-	-	16,1 (2)	-	Logické funkce A0
STK	-	-	-	-	-	16,4 (2)	-	Sklopení úrovní zásobníku do A0
ROL n	-	-	-	-	-	4,2 (4)	-	Rotace čísla vlevo
ROR n	-	-	-	-	-	4,2 (4)	-	Rotace čísla vpravo
SWP	-	-	-	-	-	3,4 (2)	-	Záměna horního a dolního bytu v A0
SWL	-	-	-	-	-	-	4,3 (2)	Záměna vrstev A0 a A1

Čítače, posuvné registry, časovače, krokový řadič

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
CTU	12,7 (4)	Dopředný čítač
CTD	13,1 (4)	Zpětný čítač
CNT	16,4 (4)	Obousměrný čítač
SFL	12,6 (4)	Posuvný registr vlevo
SFR	12,6 (4)	Posuvný registr vpravo
TON	15,6 (4)	Časovač (zpožděný přitah)
TOF	17,1 (4)	Časovač (zpožděný odpad)
RTO	17,6 (4)	Integrovaný časovač, měřič času
IMP	16,8 (4)	Časovač - generátor impulzu zadané délky
STE	4,8 (4)	Krokový řadič (stepper)

Aritmetické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])									Význam instrukce
	byte usint	Z word uint	dword uint	byte usint	# word uint	dword uint	byte usint	A word uint	dword uint	
ADD	-	6,2 (4)	-	-	5,8 (4)	-	-	6,5 (2)	-	Sčítání s přenosem
ADX	3,1 (4)	3,4 (4)	6,4 (4)	-	-	-	-	-	-	Sčítání
ADL	-	-	-	-	-	5,5 (6)	-	-	6,7 (2)	Sčítání
SUB	-	6,0 (4)	-	-	5,7 (3)	-	-	6,6 (2)	-	Odčítání s přenosem
SUX	3,1 (4)	3,2 (4)	5,9 (4)	-	-	-	-	-	-	Odčítání
SUL	-	-	-	-	-	5,5 (6)	-	-	6,7 (2)	Odčítání
MUL	4,6 (4)	-	-	4,4 (4)	-	-	5,2 (2)	-	-	Násobení
MUD	-	5,7 (4)	-	-	5,5 (4)	-	-	5,5 (2)	-	Násobení
DIV	6,2 (4)	-	-	6,0 (4)	-	-	6,8 (2)	-	-	Dělení
DID	-	32,1 (4)	-	-	31,9 (4)	-	-	31,9 (2)	-	Dělení
INR	3,0 (4)	3,8 (4)	4,5 (4)	-	-	-	-	5,8 (2)	-	Inkrementace (+ 1)
DCR	4,2 (4)	4,7 (4)	5,3 (4)	-	-	-	-	5,8 (2)	-	Dekrementace (- 1)
EQ	-	6,5 (4)	-	-	6,2 (4)	-	-	6,9 (2)	-	Porovnání (rovnost)
LT	-	6,5 (4)	-	-	6,2 (4)	-	-	6,9 (2)	-	Porovnání (menší než)
GT	-	6,5 (4)	-	-	6,2 (4)	-	-	6,9 (2)	-	Porovnání (větší než)
CMP	5,1 (4)	5,1 (4)	6,5 (4)	-	4,7 (4)	-	-	4,8 (2)	-	Porovnání
CML	-	-	-	-	-	5,9 (6)	-	-	7,6 (2)	Porovnání
BIN	-	-	-	-	-	-	-	6,7 (2)	-	Převod čísla do bin. form.
BIL	-	-	-	-	-	-	-	-	17,8 (2)	Převod čísla do bin. form.
BCD	-	-	-	-	-	-	-	21,3 (2)	-	Převod čísla do BCD
BCL	-	-	-	-	-	-	-	-	42,9 (2)	Převod čísla do BCD

Operace se zásobníky

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
POP n	3,1 (4)	Posun (rotace) zásobníku zpět o n úrovní
NXT	11,0 (4)	Aktivace následujícího zásobníku v řadě
PRV	10,9 (4)	Aktivace předcházejícího zásobníku v řadě
CHG	10,1 (4)	Aktivace zvoleného zásobníku bez zálohování S0 a S1
CHGS	10,5 (4)	Aktivace zvoleného zásobníku se zálohováním S0 a S1
LAC	6,4 (4)	Načtení hodnoty z vrcholu zvoleného zásobníku
WAC	5,6 (4)	Zápis hodnoty na vrchol zvoleného zásobníku

Příloha - Doby výkonu instrukcí

Instrukce skoků a volání

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	průchod	skok	
JMP	-	4,7 (4)	Nepodmíněný skok
JMD	2,0	5,2 (4)	Skok podmíněný nenulovostí výsledku
JMC	2,0	5,2 (4)	Skok podmíněný nulovostí výsledku
JMI	-	5,0 (2)	Skok na nepřímý cíl
JZ	2,0	5,2 (4)	Skok podmíněný nenulovostí příznaku rovnosti ZR
JNZ	2,0	5,2 (4)	Skok podmíněný nulovostí příznaku rovnosti ZR
JC	2,0	5,2 (4)	Skok podmíněný nenulovostí příznaku přenosu CO
JNC	2,0	5,2 (4)	Skok podmíněný nulovostí příznaku přenosu CO
JS	2,0	5,2 (4)	Skok podmíněný nenulovostí příznaku S1.0
JNS	2,0	5,2 (4)	Skok podmíněný nulovostí příznaku S1.0
CAL	-	6,4 (4)	Nepodmíněné volání podprogramu
CAD	2,0	7,9 (4)	Volání podprogramu podmíněné nenulovostí výsledku
CAC	2,0	7,9 (4)	Volání podprogramu podmíněné nulovostí výsledku
CAI	-	7,7 (2)	Volání podprogramu nepřímého cíle
RET	-	3,0 (2)	Nepodmíněný návrat z podprogramu
RED	1,8	3,5 (2)	Návrat z podprogramu podmíněný nenulovostí výsledku
REC	1,8	3,5 (2)	Návrat z podprogramu podmíněný nulovostí výsledku
L	1,5 (4)	-	Návěští n (cíle skoků a volání)

Organizační instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])					Význam instrukce
	průchod	skok	P41 - P49	P50 - P57	P62 - P64	
P	1,5 (4)	5,5*	4,5	4,5	1,5	Začátek procesu (*skok na náv. dané SEQ)
E	-	2,2 (4)	16,9	16,9	2,2	Nepodmíněný konec procesu
ED	1,8 (2)	3,7	18,5	18,5	3,7	Konec procesu při nenulovém výsledku
EC	1,8 (2)	3,7	18,5	18,5	3,7	Konec procesu při nulovém výsledku
EOC	-	3,4 (2)	-	-	-	Konec cyklu
NOP	1,5 (4)	-	-	-	-	Prázdná operace
BP	-	19,5 (4)	-	-	-	Ladící bod
SEQ	4,5 (4)	6,0	-	-	-	Podmíněné přerušení procesu

Tabulkové instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])						Význam instrukce
	Z			T			
	bool	byte usint sint	word uint int	bool	byte usint sint	word uint int	
LTB	11,2 (4)	9,5 (4)	10,6 (4)	10,7 (4)	8,9 (4)	10,3 (4)	Čtení položky z tabulky
WTB	10,5 (4)	8,7 (4)	9,9 (4)	14,5 (4)	12,5 (4)	15,3 (4)	Zápis položky do tabulky
FTB	10,9 (4)	7,8 (4)	9,1 (4)	9,4 (4)	8,6 (4)	10,2 (4)	Hledání položky v tabulce
FTM	+1,0	+1,0	+1,1	+1,0	+1,0	+1,1	- čas. přiřázka na 1 prohledávanou pol.
	-	8,3 (4)	9,0 (4)	-	9,4 (4)	10,0 (4)	Hledání části položky v tabulce
FTS	-	+1,5	+3,5	-	+1,5	+3,5	- čas. přiřázka na 1 prohledávanou pol.
	-	7,8 (4)	8,3 (4)	-	8,6 (4)	9,4 (4)	Zařazení položky podle tabulky
		+1,0	+2,1		+1,0	+2,1	- čas. přiřázka na 1 prohledávanou pol.

Blokové operace

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])			Význam instrukce
	Z	T	A	
SRC	5,1 (4)	6,0 (4)	-	Specifikace zdroje dat pro přesun
MOV	7,8 (4)	10,4 (4)	-	Přesun bloku dat
MTN	+1,0	+1,0	-	- časová přiřázka na 1 položku bloku
	-	-	7,5 (2)	Přesun tabulky do zápisníku
	-	-	+1,0	- časová přiřázka na 1 položku tabulky
MNT	-	-	7,5 (2)	Naplnění tabulky ze zápisníku
	-	-	+1,5	- časová přiřázka na 1 položku tabulky
FIL	7,5 (4)	-	-	Naplnění bloku konstantou
	+1,0	-	-	- časová přiřázka na 1 položku bloku

Operace se strukturovanými tabulkami

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
LDS	12,5 (2) +1,0	Čtení položky ze strukturované tabulky T - časová přirážka na 1 byte položky
WRS	14,0 (2) +1,5	Zápis položky do strukturované tabulky T - časová přirážka na 1 byte položky
FIS	8,5 (2) +1,0	Plnění položky strukturované tabulky v zápisníku - časová přirážka na 1 byte položky
FIT	8,9 (2) +1,2	Plnění položky strukturované tabulky T - časová přirážka na 1 byte položky
FNS	8,5 (2) +1,2	Hledání položky strukturované tabulky v zápisníku - časová přirážka na 1 položku
FNT	8,9 (2) +1,4	Hledání položky strukturované tabulky T - časová přirážka na 1 položku

Aritmetické instrukce v plovoucí řádové čárce (časové údaje jsou orientační, závisí na vstupní hodnotě)

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])			Význam instrukce
	Z	#	A	
ADF	93 (4)	97 (6)	97 (2)	Sčítání
SUF	94 (4)	98 (6)	98 (2)	Odčítání
MUF	79 (4)	83 (6)	83 (2)	Násobení
DIF	80 (4)	84 (6)	84 (2)	Dělení
CMF	57 (4)	61 (6)	61 (2)	Porovnání
CEI	-	-	83 (2)	Zaokrouhlení nahoru
FLO	-	-	78 (2)	Zaokrouhlení dolů
ABS	-	-	5,1 (2)	Absolutní hodnota
LOG	-	-	220 (2)	Dekadický logaritmus
LN	-	-	220 (2)	Přirozený logaritmus
EXP	-	-	4500 (2)	Exponenciální funkce
POW	-	-	8000 (2)	Obecná mocnina
SQR	-	-	1050 (2)	Druhá odmocnina
HYP	-	-	1200 (2)	Euklidovská vzdálenost
SIN	-	-	3300 (2)	Sinus
ASN	-	-	4200 (2)	Arc sinus
COS	-	-	3000 (2)	Cosinus
ACS	-	-	4200 (2)	Arc cosinus
TAN	-	-	5800 (2)	Tangens
ATN	-	-	2800 (2)	Arc tangens
UWF	-	-	40 ÷ 100 (2)	Převod uint na real
IWF	-	-	40 ÷ 100 (2)	Převod int na real
ULF	-	-	40 ÷ 100 (2)	Převod uint na real
ILF	-	-	40 ÷ 100 (2)	Převod dint na real
UFW	-	-	50 ÷ 130 (2)	Převod real na uint
IFW	-	-	50 ÷ 130 (2)	Převod real na int
UFL	-	-	50 ÷ 130 (2)	Převod real na uint
IFL	-	-	50 ÷ 130 (2)	Převod real na dint

Instrukce regulátoru PID (časové údaje jsou orientační, závisí na zvolených funkcích a vstupních hodnotách)

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
CNV	500 (2)	Konverze a zpracování dat z analogových vstupů
PID	1000 ÷ 2000 (2)	PID regulátor

Operace se znaky ASCII

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	word	real	
BAS	8,9 (2)	-	Převod čísla na ASCII
ASB	3,6 (2)	-	Převod čísla z ASCII
STF		650 (2)	Převod ASCII řetězce na real
FST		340 (2)	Převod real na ASCII řetězec

A.6. DOBY VÝKONU INSTRUKCÍ V CENTRÁLNÍCH JEDNOTKÁCH CP-7001, CP-7002 TECOMAT TC700 A TECOMAT TC650

Doby výkonu instrukcí jsou orientační a nezahrnují vlivy systémových procesů, které přerušují provádění uživatelského programu. Těmito procesy jsou sériové komunikace a činnost vyrovnávacích pamětí cash. Jejich vlivem dochází k prodloužení i zkracování doby cyklu.

Přehled použitých symbolů operandů:

Z - zápisník X, Y, S, D, R

T - tabulky

- konstanta

A - bez operandu (pracuje pouze na zásobníku)

n - číselný parametr

Instrukce pro čtení a zápis dat

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])							Význam instrukce
	bool	byte usint sint	Z word uint int	dword udint dint real	Ireal	dword udint dint real	# Ireal	
LD	0,9 (4)	0,8 (4)	1,0 (4)	1,3 (4)	2,1 (4)	0,9 (6)	-	Čtení přímých dat
LDQ	-	-	-	-	-	-	1,3 (10)	Čtení přímých dat
LDC	0,9 (4)	0,9 (4)	1,0 (4)	1,3 (4)	-	-	-	Čtení negovaných dat
WR	0,8 (4)	0,6 (4)	0,7 (4)	1,1 (4)	1,9 (4)	-	-	Zápis přímých dat
WRC	0,9 (4)	0,6 (4)	0,7 (4)	1,1 (4)	-	-	-	Zápis negovaných dat
WRA	-	0,9 (4)	1,1 (4)	1,8 (4)	-	-	-	Zápis přímých dat s alternací
PUT	1,0 (4)	0,8 (4)	0,9 (4)	1,3 (4)	-	-	-	Podmíněný zápis dat
	0,7	0,7	0,7	0,7	-	-	-	- podmínka splněna
	0,8 (6)	0,7 (6)	0,7 (6)	0,7 (6)	0,7 (6)	-	-	- podmínka nesplněna
LEA								Načtení adresy

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])					Význam instrukce
	bool	byte usint sint	A word uint int	dword udint dint real	Ireal	
LDIB	1,0 (2)	-	-	-	-	Čtení dat šířky bit z adresy v A0
LDI	-	0,9 (2)	-	-	-	Čtení dat šířky 8 bitů z adresy v A0
LDIW	-	-	1,0 (2)	-	-	Čtení dat šířky 16 bitů z adresy v A0
LDIL	-	-	-	1,3 (2)	-	Čtení dat šířky 32 bitů z adresy v A0
LDIQ	-	-	-	-	2,2 (2)	Čtení dat šířky 64 bitů z adresy v A0
WRIB	1,1 (2)	-	-	-	-	Zápis dat šířky bit do adresy v A0
WRI	-	0,8 (2)	-	-	-	Zápis dat šířky 8 bitů do adresy v A0
WRIW	-	-	0,9 (2)	-	-	Zápis dat šířky 16 bitů do adresy v A0
WRIL	-	-	-	1,3 (2)	-	Zápis dat šířky 32 bitů do adresy v A0
WRIQ	-	-	-	-	2,2 (2)	Zápis dat šířky 64 bitů do adresy v A0

Logické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])							Význam instrukce
	Z				#	A		
	bool	byte usint sint	word uint int	dword udint dint	dword udint dint	word uint int	dword udint dint	
AND	0,9 (4)	0,8 (4)	0,9 (4)	1,3 (4)	0,9 (6)	-	0,8 (2)	AND s přímým operandem
ANC	0,9 (4)	0,9 (4)	1,0 (4)	1,4 (4)	-	-	0,9 (2)	AND s negovaným operandem
OR	0,9 (4)	0,8 (4)	0,9 (4)	1,3 (4)	0,9 (6)	-	0,8 (2)	OR s přímým operandem
ORC	0,9 (4)	0,9 (4)	1,0 (4)	1,4 (4)	-	-	0,9 (2)	OR s negovaným operandem
XOR	0,9 (4)	0,8 (4)	0,9 (4)	1,3 (4)	0,9 (6)	-	0,8 (2)	XOR s přímým operandem
XOC	0,9 (4)	0,9 (4)	1,0 (4)	1,4 (4)	-	-	0,9 (2)	XOR s negovaným operandem
NEG	-	-	-	-	-	-	0,7 (2)	Negace vrcholu zásobníku
SET	0,6 (4)	0,6 (4)	0,6 (4)	0,6 (4)	-	-	-	Podmíněné nastavení
RES	0,6 (4)	0,6 (4)	0,6 (4)	0,6 (4)	-	-	-	Podmíněné nulování
LET	1,2 (4)	1,0 (4)	1,1 (4)	2,0 (4)	-	-	-	Impulz od náběžné hrany
BET	1,2 (4)	1,0 (4)	1,1 (4)	2,0 (4)	-	-	-	Impulz od libovolné hrany
FLG	-	-	-	-	-	2,9 (2)	-	Logické funkce A0
STK	-	-	-	-	-	-	3,6 (2)	Sklopení úrovní zásobníku do A0
ROL n	-	-	-	-	-	1,1 (4)	-	Rotace hodnoty vlevo n-krát
ROL	-	-	-	-	-	-	1,4 (2)	Rotace hodnoty vlevo n-krát
ROR n	-	-	-	-	-	1,1 (4)	-	Rotace hodnoty vpravo n-krát
ROR	-	-	-	-	-	-	1,4 (2)	Rotace hodnoty vpravo n-krát
SHL	-	-	-	-	-	-	1,2 (2)	Posun hodnoty vlevo n-krát
SHR	-	-	-	-	-	-	1,2 (2)	Posun hodnoty vpravo n-krát
SWP	-	-	-	-	-	0,9 (2)	-	Záměna prvního a druhého bytu A0
SWL	-	-	-	-	-	-	0,7 (2)	Záměna dolního a horního wordu A0

Čítače, posuvné registry, časovače, krokový řadič

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		#	Význam instrukce
	word uint	dword udint		
CTU	3,2 (4)	3,4 (4)		Dopředný čítač
CTD	3,3 (4)	3,5 (4)		Zpětný čítač
CNT	4,1 (4)	4,3 (4)		Obousměrný čítač
SFL	3,2 (4)	3,4 (4)		Posuvný registr vlevo
SFR	3,2 (4)	3,4 (4)		Posuvný registr vpravo
TON	3,9 (4)	-		Časovač (zpožděný přítah)
TOF	4,3 (4)	-		Časovač (zpožděný odpad)
RTO	4,4 (4)	-		Integrovaný časovač, měřič času
IMP	4,2 (4)	-		Časovač - generátor impulzu zadané délky
STE	1,2 (4)	1,4 (4)		Krokový řadič (stepper)

Příloha - Doby výkonu instrukcí

Aritmetické instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])							Význam instrukce
	Z			#		A		
	byte usint	word uint	dword udint	byte usint	dword udint	byte usint	dword udint	
ADD	1,0 (4)	1,1 (4)	1,5 (4)	-	0,9 (6)	-	0,9 (2)	Sčítání
SUB	1,0 (4)	1,1 (4)	1,5 (4)	-	0,9 (6)	-	0,9 (2)	Odčítání
MUL	1,2 (4)	1,3 (4)	1,7 (4)	-	1,0 (6)	-	1,0 (2)	Násobení
MULS	1,2 (4)	1,3 (4)	1,7 (4)	-	1,0 (6)	-	1,0 (2)	Násobení se znaménkem
DIV	1,8 (4)	-	-	1,4 (4)	-	1,6 (2)	-	Dělení (/ =)
DID	2,6 (4)	2,7 (4)	3,1 (4)	-	2,4 (6)	-	2,1 (2)	Dělení se zbytkem
DIVL	2,0 (4)	2,1 (4)	2,5 (4)	-	1,8 (6)	-	1,9 (2)	Dělení
DIVS	2,0 (4)	2,1 (4)	2,5 (4)	-	1,8 (6)	-	1,9 (2)	Dělení se znaménkem
MOD	-	-	-	-	-	-	1,8 (2)	Zbytek dělení
MODS	-	-	-	-	-	-	1,8 (2)	Zbytek dělení se znaménkem
INR	1,0 (4)	1,2 (4)	2,0 (4)	-	-	-	0,8 (2)	Inkrementace (+ 1)
DCR	1,0 (4)	1,2 (4)	2,0 (4)	-	-	-	0,8 (2)	Dekrementace (– 1)
EQ	1,6 (4)	1,7 (4)	2,1 (4)	-	1,6 (6)	-	1,5 (2)	Porovnání (rovnost)
LT	1,6 (4)	1,7 (4)	2,1 (4)	-	1,6 (6)	-	1,5 (2)	Porovnání (menší než)
LTS	1,6 (4)	1,7 (4)	2,1 (4)	-	1,6 (6)	-	1,5 (2)	Porovnání se znaménkem (menší než)
GT	1,6 (4)	1,7 (4)	2,1 (4)	-	1,6 (6)	-	1,5 (2)	Porovnání (větší než)
GTS	1,6 (4)	1,7 (4)	2,1 (4)	-	1,6 (6)	-	1,5 (2)	Porovnání se znaménkem (větší než)
CMP	1,4 (4)	1,5 (4)	1,9 (4)	-	1,4 (6)	-	1,4 (2)	Porovnání
CMPS	1,4 (4)	1,5 (4)	1,9 (4)	-	1,4 (6)	-	1,4 (2)	Porovnání se znaménkem
MAX	-	-	-	-	-	-	1,0 (2)	Maximum
MAXS	-	-	-	-	-	-	1,0 (2)	Maximum se znaménkem
MIN	-	-	-	-	-	-	1,0 (2)	Minimum
MINS	-	-	-	-	-	-	1,0 (2)	Minimum se znaménkem
ABSL	-	-	-	-	-	-	0,8 (2)	Absolutní hodnota
CSGL	-	-	-	-	-	-	0,8 (2)	Změna znaménka
EXTB	-	-	-	-	-	-	0,9 (2)	Roztažení znaménka z 8 bitů na 32 bitů
EXTW	-	-	-	-	-	-	0,9 (2)	Roztažení znaménka z 16 bitů na 32 bitů
BIN	-	-	-	-	-	-	2,1 (2)	Převod čísla do bin. form. (8 cifer BCD)
BIL	-	-	-	-	-	-	3,1 (2)	Převod čísla do bin. form. (10 cifer BCD)
BCD	-	-	-	-	-	-	12,6 (2)	Převod čísla do BCD (8 cifer BCD)
BCL	-	-	-	-	-	-	14,7 (2)	Převod čísla do BCD (10 cifer BCD)

Operace se zásobníky

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])	Význam instrukce
POP n	0,8 (4)	Posun (rotace) zásobníku zpět o n úrovní
NXT	2,8 (4)	Aktivace následujícího zásobníku v řadě
PRV	2,8 (4)	Aktivace předcházejícího zásobníku v řadě
CHG	2,7 (4)	Aktivace zvoleného zásobníku se zálohováním S0 a S1
CHGS	2,6 (4)	Aktivace zvoleného zásobníku bez zálohování S0 a S1
LAC	1,6 (4)	Načtení hodnoty z vrcholu zvoleného zásobníku
WAC	1,4 (4)	Zápis hodnoty na vrchol zvoleného zásobníku
PSHB	1,4 (2)	Uložení 8 bitů vrcholu zásobníku do stacku podle SP
PSHW	1,4 (2)	Uložení 16 bitů vrcholu zásobníku do stacku podle SP
PSHL	1,4 (2)	Uložení 32 bitů vrcholu zásobníku do stacku podle SP
PSHQ	1,4 (2)	Uložení 64 bitů vrcholu zásobníku do stacku podle SP
POPB	1,6 (2)	Naplnění 8 bitů vrcholu zásobníku ze stacku podle SP
POPW	1,6 (2)	Naplnění 16 bitů vrcholu zásobníku ze stacku podle SP
POPL	1,6 (2)	Naplnění 32 bitů vrcholu zásobníku ze stacku podle SP
POPQ	1,6 (2)	Naplnění 64 bitů vrcholu zásobníku ze stacku podle SP

Instrukce skoků a volání

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
	průchod	skok	
JMP Ln	-	1,2 (4)	Nepodmíněný skok
JMD Ln	0,5	1,3 (4)	Skok podmíněný nenulovostí výsledku
JMC Ln	0,5	1,3 (4)	Skok podmíněný nulovostí výsledku
JMI Ln	-	1,3 (4)	Skok na nepřímý cíl
JMI	-	1,3 (2)	Skok na nepřímý cíl
JZ Ln	0,5	1,3 (4)	Skok podmíněný nenulovostí příznaku rovnosti ZR
JNZ Ln	0,5	1,3 (4)	Skok podmíněný nulovostí příznaku rovnosti ZR
JC Ln	0,5	1,3 (4)	Skok podmíněný nenulovostí příznaku přenosu CO
JNC Ln	0,5	1,3 (4)	Skok podmíněný nulovostí příznaku přenosu CO
JB Ln	0,5	1,3 (4)	Skok podmíněný nenulovostí příznaku rovnosti S0.2
JNB Ln	0,5	1,3 (4)	Skok podmíněný nulovostí příznaku rovnosti S0.2
JS Ln	0,5	1,3 (4)	Skok podmíněný nenulovostí příznaku S1.0
JNS Ln	0,5	1,3 (4)	Skok podmíněný nulovostí příznaku S1.0
CAL Ln	-	1,7 (4)	Nepodmíněné volání podprogramu
CAD Ln	0,5	2,0 (4)	Volání podprogramu podmíněné nenulovostí výsledku
CAC Ln	0,5	2,0 (4)	Volání podprogramu podmíněné nulovostí výsledku
CAI Ln	-	2,0 (4)	Volání podprogramu nepřímého cíle
CAI	-	2,0 (2)	Volání podprogramu nepřímého cíle
RET	-	1,8 (2)	Nepodmíněný návrat z podprogramu
RED	0,5	1,9 (2)	Návrat z podprogramu podmíněný nenulovostí výsledku
REC	0,5	1,9 (2)	Návrat z podprogramu podmíněný nulovostí výsledku
L n	0,4 (4)	-	Návěští n (cíl skoků a volání)

Organizační instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])					Význam instrukce
	průchod	skok	P41 - P49	P50 - P57	P62 - P64	
P	0,4 (4)	1,4*	1,2	1,2	0,4	Začátek procesu (*skok na návěští dané SEQ)
E	-	0,6 (4)	4,3	4,3	0,6	Nepodmíněný konec procesu
ED	0,5 (2)	0,9	4,7	4,7	0,9	Konec procesu při nenulovém výsledku
EC	0,5 (2)	0,9	4,7	4,7	0,9	Konec procesu při nulovém výsledku
NOP	0,4 (4)	-	-	-	-	Prázdná operace
BP	-	4,9 (4)	-	-	-	Ladící bod
SEQ	1,2 (4)	1,5	-	-	-	Podmíněné přerušení procesu

Tabulkové instrukce

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])					Význam instrukce
	bool	byte usint sint	word uint int	dword udint dint	real	
LTB	3,8 (4)	2,4 (4)	2,7 (4)	3,0 (4)	3,0 (4)	Čtení položky z tabulky
WTB	2,7 (4)	2,2 (4)	2,5 (4)	2,8 (4)	2,8 (4)	Zápis položky do tabulky
FTB	2,8 (4)	2,0 (4)	2,3 (4)	2,6 (4)	2,6 (4)	Hledání položky v tabulce
	+0,2	+0,2	+0,3	+0,4	+0,4	- časová přírážka na 1 prohledávanou položku
FTBN	2,8 (4)	2,0 (4)	2,3 (4)	2,6 (4)	2,6 (4)	Hledání další položky v tabulce
	+0,2	+0,2	+0,3	+0,4	+0,4	- časová přírážka na 1 prohledávanou položku
FTM	-	2,1 (4)	2,3 (4)	2,6 (4)	2,6 (4)	Hledání části položky v tabulce
		+0,4	+0,8	+1,2	+1,2	- časová přírážka na 1 prohledávanou položku
FTMN	-	2,1 (4)	2,3 (4)	2,6 (4)	2,6 (4)	Hledání další části položky v tabulce
		+0,4	+0,8	+1,2	+1,2	- časová přírážka na 1 prohledávanou položku
FTS	-	2,0 (4)	2,1 (4)	2,2 (4)	-	Zařazení položky podle tabulky
		+0,2	+0,5	+0,8	-	- časová přírážka na 1 prohledávanou položku
FTSF	-	-	-	-	2,2 (4)	Zařazení položky podle tabulky
					+1,0	- časová přírážka na 1 prohledávanou položku
FTSS	-	2,0 (4)	2,1 (4)	2,2 (4)	-	Zařazení položky podle tabulky
		+0,2	+0,5	+0,8	-	- časová přírážka na 1 prohledávanou položku

Příloha - Doby výkonu instrukcí

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])					Význam instrukce
	bool	byte usint sint	word uint int	dword udint dint	real	
LTB	3,8 (4)	2,4 (4)	2,7 (4)	3,0 (4)	3,0 (4)	Čtení položky z tabulky
WTB	3,7 (4)	3,2 (4)	3,6 (4)	4,0 (4)	4,0 (4)	Zápis položky do tabulky
FTB	2,6 (4)	2,3 (4)	2,7 (4)	3,1 (4)	3,1 (4)	Hledání položky v tabulce
	+0,2	+0,2	+0,3	+0,4	+0,4	- časová přírážka na 1 prohledávanou položku
FTBN	2,6 (4)	2,3 (4)	2,7 (4)	3,1 (4)	3,1 (4)	Hledání další položky v tabulce
	+0,2	+0,2	+0,3	+0,4	+0,4	- časová přírážka na 1 prohledávanou položku
FTM	-	2,4 (4)	2,8 (4)	3,2 (4)	3,2 (4)	Hledání části položky v tabulce
		+0,4	+0,8	+1,2	+1,2	- časová přírážka na 1 prohledávanou položku
FTMN	-	2,4 (4)	2,8 (4)	3,2 (4)	3,2 (4)	Hledání další části položky v tabulce
		+0,4	+0,8	+1,2	+1,2	- časová přírážka na 1 prohledávanou položku
FTS	-	2,3 (4)	2,5 (4)	2,7 (4)	-	Zařazení položky podle tabulky
		+0,2	+0,5	+0,8		- časová přírážka na 1 prohledávanou položku
FTSF	-	-	-	-	2,7 (4)	Zařazení položky podle tabulky
					+1,0	- časová přírážka na 1 prohledávanou položku
FTSS	-	2,3 (4)	2,5 (4)	2,7 (4)	-	Zařazení položky podle tabulky
		+0,2	+0,5	+0,8		- časová přírážka na 1 prohledávanou položku

Blokové operace

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])			Význam instrukce
	Z	T	A	
SRC	1,3 (4)	1,5 (4)	-	Specifikace zdroje dat pro přesun
MOV	1,9 (4)	2,6 (4)	-	Přesun bloku dat
	+0,2	+0,2		- časová přírážka na 1 položku bloku
MTN	-	-	1,9 (2)	Přesun tabulky do zápisníku
			+0,2	- časová přírážka na 1 položku tabulky
MNT	-	-	1,9 (2)	Naplnění tabulky ze zápisníku
			+0,4	- časová přírážka na 1 položku tabulky
FIL	1,9 (4)	-	-	Naplnění bloku konstantou
	+0,2			- časová přírážka na 1 položku bloku
BCMP	1,9 (2)	-	-	Porovnání bloků
	+0,4			- časová přírážka na 1 položku bloků

Operace se strukturovanými tabulkami

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])		Význam instrukce
LDSR	3,1 (2)		Čtení položky ze strukturované tabulky v zápisníku
	+0,2		- časová přírážka na 1 byte položky
LDS	3,2 (2)		Čtení položky ze strukturované tabulky T
	+0,2		- časová přírážka na 1 byte položky
WRSR	3,4 (2)		Zápis položky do strukturované tabulky v zápisníku
	+0,3		- časová přírážka na 1 byte položky
WRS	3,5 (2)		Zápis položky do strukturované tabulky T
	+0,3		- časová přírážka na 1 byte položky
FIS	2,2 (2)		Plnění položky strukturované tabulky v zápisníku
	+0,2		- časová přírážka na 1 byte položky
FIT	2,3 (2)		Plnění položky strukturované tabulky T
	+0,3		- časová přírážka na 1 byte položky
FNS	2,2 (2)		Hledání položky strukturované tabulky v zápisníku
	+0,3		- časová přírážka na 1 položku
FNT	2,3 (2)		Hledání položky strukturované tabulky T
	+0,3		- časová přírážka na 1 položku

Aritmetické instrukce v plovoucí řádové čárce (časové údaje jsou orientační, závisí na vstupní hodnotě)

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])				Význam instrukce
	Z real	# real	A real	Ireal	
ADF	23 (4)	24 (6)	24 (2)	-	Sčítání
ADDF	-	-	-	24 (2)	Sčítání
SUF	23 (4)	24 (6)	24 (2)	-	Odčítání
SUDF	-	-	-	24 (2)	Odčítání
MUF	20 (4)	21 (6)	21 (2)	-	Násobení
MUDF	-	-	-	21 (2)	Násobení
DIF	20 (4)	21 (6)	21 (2)	-	Dělení
DIDF	-	-	-	21 (2)	Dělení
EQF	18 (4)	19 (6)	19 (2)	-	Porovnání (rovnost)
EQDF	-	-	-	19 (2)	Porovnání (rovnost)
LTF	18 (4)	19 (6)	19 (2)	-	Porovnání (menší než)
LTDF	-	-	-	19 (2)	Porovnání (menší než)
GTF	18 (4)	19 (6)	19 (2)	-	Porovnání (větší než)
GTDF	-	-	-	19 (2)	Porovnání (větší než)
CMF	15 (4)	16 (6)	16 (2)	-	Porovnání
CMDF	-	-	-	16 (2)	Porovnání
MAXF	-	-	24 (2)	-	Maximum
MAXD	-	-	-	24 (2)	Maximum
MINF	-	-	24 (2)	-	Minimum
MIND	-	-	-	24 (2)	Minimum
CEI	-	-	21 (2)	-	Zaokrouhlení nahoru
CEID	-	-	-	21 (2)	Zaokrouhlení nahoru
FLO	-	-	19 (2)	-	Zaokrouhlení dolů
FLOD	-	-	-	19 (2)	Zaokrouhlení dolů
RND	-	-	23 (2)	-	Aritmetické zaokrouhlení
RNDD	-	-	-	23 (2)	Aritmetické zaokrouhlení
ABS	-	-	0,5 (2)	-	Absolutní hodnota
ABSD	-	-	-	0,5 (2)	Absolutní hodnota
CSG	-	-	0,5 (2)	-	Změna znaménka
CSGD	-	-	-	0,5 (2)	Změna znaménka
LOG	-	-	55 (2)	-	Dekadický logaritmus
LOGD	-	-	-	55 (2)	Dekadický logaritmus
LN	-	-	55 (2)	-	Přirozený logaritmus
LND	-	-	-	55 (2)	Přirozený logaritmus
EXP	-	-	1100 (2)	-	Exponenciální funkce
EXPD	-	-	-	1100 (2)	Exponenciální funkce
POW	-	-	2000 (2)	-	Obecná mocnina
POWD	-	-	-	2000 (2)	Obecná mocnina
SQR	-	-	260 (2)	-	Druhá odmocnina
SQRD	-	-	-	260 (2)	Druhá odmocnina
HYP	-	-	300 (2)	-	Euklidovská vzdálenost
HYPD	-	-	-	300 (2)	Euklidovská vzdálenost
SIN	-	-	800 (2)	-	Sinus
SIND	-	-	-	800 (2)	Sinus
ASN	-	-	1000 (2)	-	Arc sinus
ASND	-	-	-	1000 (2)	Arc sinus
COS	-	-	800 (2)	-	Cosinus
COSD	-	-	-	800 (2)	Cosinus
ACS	-	-	1000 (2)	-	Arc cosinus
ACSD	-	-	-	1000 (2)	Arc cosinus
TAN	-	-	1500 (2)	-	Tangens
TAND	-	-	-	1500 (2)	Tangens
ATN	-	-	700 (2)	-	Arc tangens
ATND	-	-	-	700 (2)	Arc tangens
UWF	-	-	10 ÷ 25 (2)	-	Převod uint na real
IWF	-	-	10 ÷ 25 (2)	-	Převod int na real
ULF	-	-	10 ÷ 25 (2)	-	Převod udint na real
ILF	-	-	10 ÷ 25 (2)	-	Převod dint na real

Příloha - Doby výkonu instrukcí

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])				Význam instrukce
	Z	#	A		
	real	real	real	lreal	
ULDF	-	-	-	10 ÷ 25 (2)	Převod uint na lreal
ILDF	-	-	-	10 ÷ 25 (2)	Převod dint na lreal
FDF	-	-	-	10 ÷ 25 (2)	Převod real na lreal
UFW	-	-	10 ÷ 30 (2)	-	Převod real na uint
IFW	-	-	10 ÷ 30 (2)	-	Převod real na int
UFL	-	-	10 ÷ 30 (2)	-	Převod real na uint
IFL	-	-	10 ÷ 30 (2)	-	Převod real na dint
UDFL	-	-	-	10 ÷ 30 (2)	Převod lreal na uint
IDFL	-	-	-	10 ÷ 30 (2)	Převod lreal na dint
DFF	-	-	-	10 ÷ 30 (2)	Převod lreal na real

Instrukce regulátoru PID (časové údaje jsou orientační, závisí na zvolených funkcích a vstupních hodnotách)

Mnemo kód	Doba výkonu [μs] (Délka kódu [B]) A	Význam instrukce
CNV	120 (2)	Konverze a zpracování dat z analogových jednotek
PID	250 ÷ 500 (2)	PID regulátor

Operace se znaky ASCII

Mnemo kód	Doba výkonu [μs] (Délka kódu [B])				Význam instrukce
	A				
	word uint	dword uint	real	lreal	
TER	-	200 (2)	-	-	Terminálová instrukce
BAS	2,3 (2)	-	-	-	Převod čísla z binárního formátu na ASCII
ASB	0,9 (2)	-	-	-	Převod čísla z ASCII do binárního formátu
STF	-	-	170 (2)	-	Převod ASCII řetězce na float
STDF	-	-	-	170 (2)	Převod ASCII řetězce na double
FST	-	-	90 (2)	-	Převod float na ASCII řetězec
DFST	-	-	-	90 (2)	Převod double na ASCII řetězec



teco

Objednávky a informace:

Teco a. s. Havlíčkova 260, 280 58 Kolín 4, tel. 321 737 611, fax 321 737 633

TXV 001 09.01

Výrobce si vyhrazuje právo na změny dokumentace. Poslední aktuální vydání je k dispozici na internetu
www.tecomat.cz