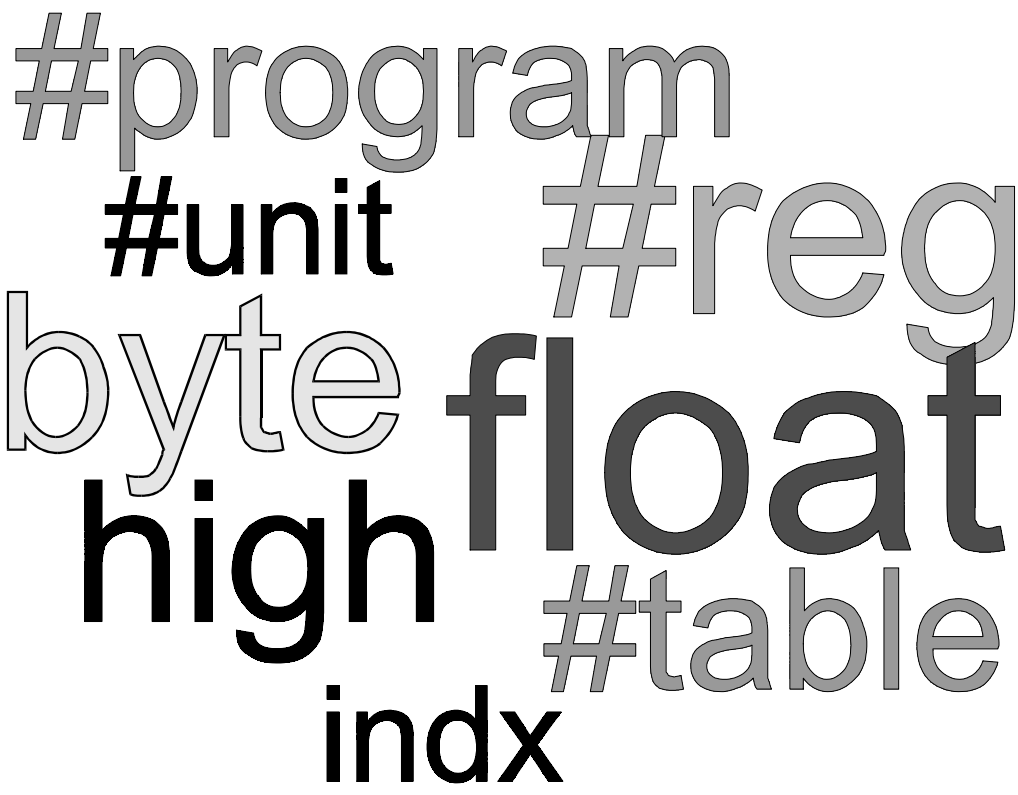


PODRĘCZNIK PROGRAMISTY PLC TECOMAT

PODRĘCZNIK PROGRAMISTY PLC TECOMAT

Wydanie 2. – 6/2000

SPIS TREŚCI

1. WSTĘP

Celem tego podręcznika jest przybliżenie użytkownikowi programowalnych sterowników (PLC) TECOMAT oraz ułatwienie ich programowania.

Zakres podręcznika

Podręcznik jest przeznaczony dla wszystkich PLC TECOMAT. Opisuje jednostki centralne w zależności od typu (patrz dalej), a nie w zależności od rodzaju PLC. Ewentualne różnice pomiędzy poszczególnymi typami PLC (np. fizyczne adresowanie jednostek peryferyjnych) są podane w suplemencie na końcu podręcznika.

Typy jednostek centralnych

Każdy typ PLC TECOMAT ma do dyspozycji kilka centralnych jednostek (skrót CPU - Central Processor Unit), odróżnionych tak zwaną grupą w postaci litery, będącej zazwyczaj częścią nazwy jednostki centralnej. Każda grupa centralnych jednostek ma określony rozmiar obszaru pamięci, zakres zbioru instrukcji i operandów oraz jest parametrem dla kompilatora programu.

Podział podręcznika

- 2. rozdział przedstawia ogólne zasady obsługi programu w PLC
- 3. rozdział opisuje podstawową strukturę programu w środowisku xPRO
- 4. rozdział opisuje strukturę instrukcji i ich operandów
- 5. rozdział opisuje strukturę pamięci wraz ze szczegółowym wykazem funkcji systemowych zawartych w rejestrach systemowych S
- 6. rozdział przedstawia ogólne zasady fizycznego adresowania jednostek PLC
- 7. rozdział opisuje pozostałe obszary adresowe dla danych - dane D, tabelki T oraz dodatkowa pamięć DataBox
- 8. rozdział opisuje strukturę i pracę akumulatora wyników
- 9. rozdział zawiera wykaz dyrektyw stosowanych w środowisku xPRO wraz z przykładami ich zastosowania
- 10. rozdział opisuje procesy programowe
- 11. rozdział zawiera wykaz instrukcji i dopuszczalnych operandów
- 12. rozdział opisuje instrukcje użytkownika

Ze względu na oszczędność miejsca, przykłady w tym podręczniku są podane jedynie w postaci mnemokodu. Do wyświetlenia przykładu w schemacie przekaźnikowym (tzw. drabinkowym), należy użyć środowiska xPRO.

Podręczniki pokrewne

Szczegółowe informacje o poszczególnych instrukcjach są zawarte w podręczniku *Zbiór instrukcji PLC TECOMAT* (TXV 001 05.01).

Przykłady rozwiązania różnych częściowych problemów są zawarte w podręczniku *Przykłady programowania PLC TECOMAT w środowisku xPRO* (TXV 001 07.01).

Programowanie PLC

Programowanie algorytmów sterujących oraz testowanie poprawności programów dla PLC TECOMAT dokonywane jest na komputerach klasy PC. Do połączenia z PLC wykorzystuje się zwykły szeregowy kanał tych komputerów.

Do każdego PLC jest dołączana dyskietka z przykładami i programowym softwarem xPRO w wersji xPRO Lite oraz xPROm.

Przykłady programów PLC zawierają instrukcje obsługi różnych jednostek PLC oraz deklaracje sporządzone dokładnie dla danego zestawu PLC, do którego jest dyskietka dołączona. Te deklaracje można dowolnie użyć w nowych programach. Następnie są tu też przykłady z podręcznika TXV 001 07.01.

Narzędzie programowe xPRO

Specjalny software programowy xPRO jest zintegrowanym środowiskiem do tworzenia aplikacji. Dostępne są następujące wersje:

xPRO - pełna wersja z hw kluczem do profesjonalnej pracy
xPRO Lite - darmowa wersja programu xPRO bez hw klucza
xPROm - darmowa wersja programu xPRO przeznaczona do kontroli pracy technologii bez możliwości ingerencji w program PLC

Program xPRO ma następujące właściwości:

- zintegrowane środowisko Turbo Vision wraz z edytorem
- możliwość pracy z kilkoma plikami jednocześnie, każdy we własnym oknie
- kompilator na kod maszynowy jednostek centralnych wszystkich typów
- nazwy symboliczne etykiet i operandów
- automatyczne przydzielanie zmiennych
- generowanie tabelki symboli i przypisów
- tworzenie makroinstrukcji
- wsteczna kompilacja programu
- wbudowany symulator PLC TECOMAT

- symulacja technologicznego panelu i operatorskich paneli ID-04, ID-05 oraz wbudowanego panelu TC500
- pełne wykorzystanie środków strojenia PLC
- automatyczne generowanie konfiguracji na podstawie podłączonego PLC
- schemat przekaźnikowy (tzw. drabinkowy) z podświetleniem aktywnych gałęzi w trybie strojenia
- zintegrowany system pomocy zawierający cały podręcznik programisty
- sterowanie za pomocą myszy lub klawiatury z możliwością używania jak menu programu (pull down menu), tak i bezpośrednich opcji za pomocą kombinacji klawiszy (hot keys)

2. PLC A PROGRAM UŽYTKOWNIKA

Co to jest sterownik programowalny

Sterownik programowalny (PLC - Programmable Logic Controller) jest cyfrowym elektronicznym systemem sterującym, przeznaczonym do sterowania procesów w środowisku przemysłowym. Używa programowalnej pamięci dla wewnętrznego zapisu instrukcji, służących do realizacji specyficznych funkcji dla sterowania różnych rodzajów maszyn lub procesów poprzez cyfrowe lub analogowe wejścia i wyjścia.

Firma Teco a. s. produkuje systemy PLC pod ochronnym znakiem towarowym TECOMAT.

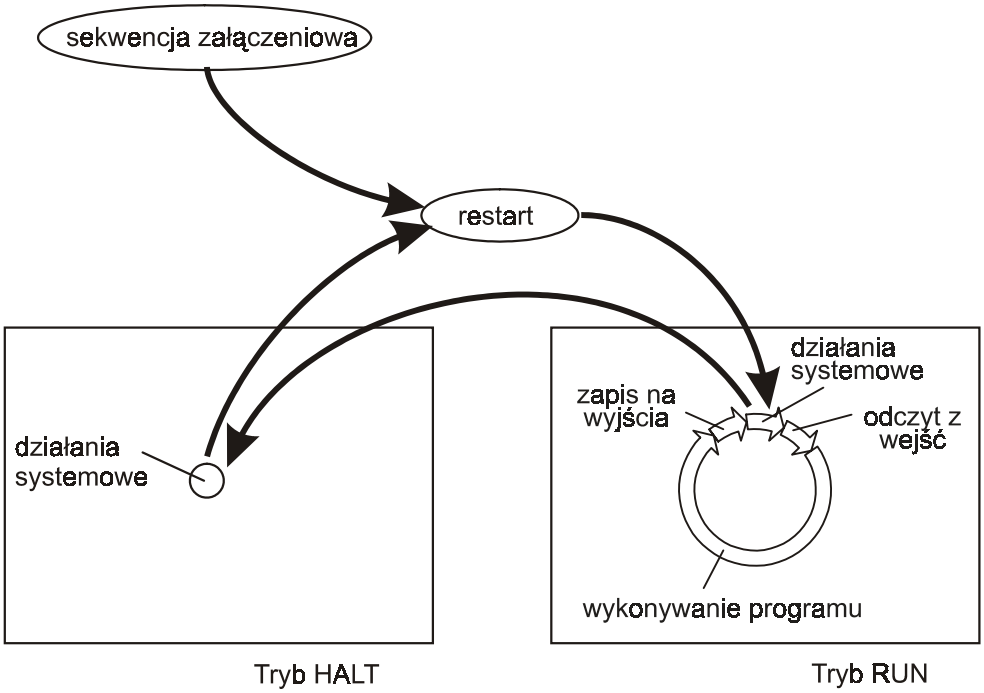
Sposób wykonywania programu

Sterujący algorytm sterownika programowalnego jest zapisany w pamięci programu jako ciąg instrukcji. Centralna jednostka stopniowo odczytuje z tej pamięci poszczególne instrukcje, wykonuje odpowiednie operacje na danych w pamięci i akumulatorze, ewentualnie wykonuje skoki pomiędzy instrukcjami (w przypadku instrukcji z grupy instrukcji organizacyjnych). Jeśli zostały wykonane wszystkie instrukcje algorytmu, centralna jednostka dokonuje aktualizacji wyjściowych zmiennych do wyjściowych jednostek peryferyjnych oraz aktualizuje stany z wejściowych jednostek peryferyjnych do pamięci. Ten proces jest stale powtarzany i nazywany jest cyklem programu (rys.2.1, rys.2.2).

Cykliczne wykonywanie programu

Jednorazowa aktualizacja zmiennych wejściowych w trakcie całego cyklu programu wyklucza możliwość powstawania stanów hazardowych podczas wykonywania algorytmu sterowania (w trakcie wyliczeń nie może dojść do zmiany zmiennych wejściowych).

Przed rozpoczęciem pisania programu dla PLC należy ten fakt wziąć pod uwagę. W niektórych przypadkach ułatwia to rozwiązanie problemu, w innych z kolei komplikuje go.



Rys.2.1 Wykonywanie programu w PLC

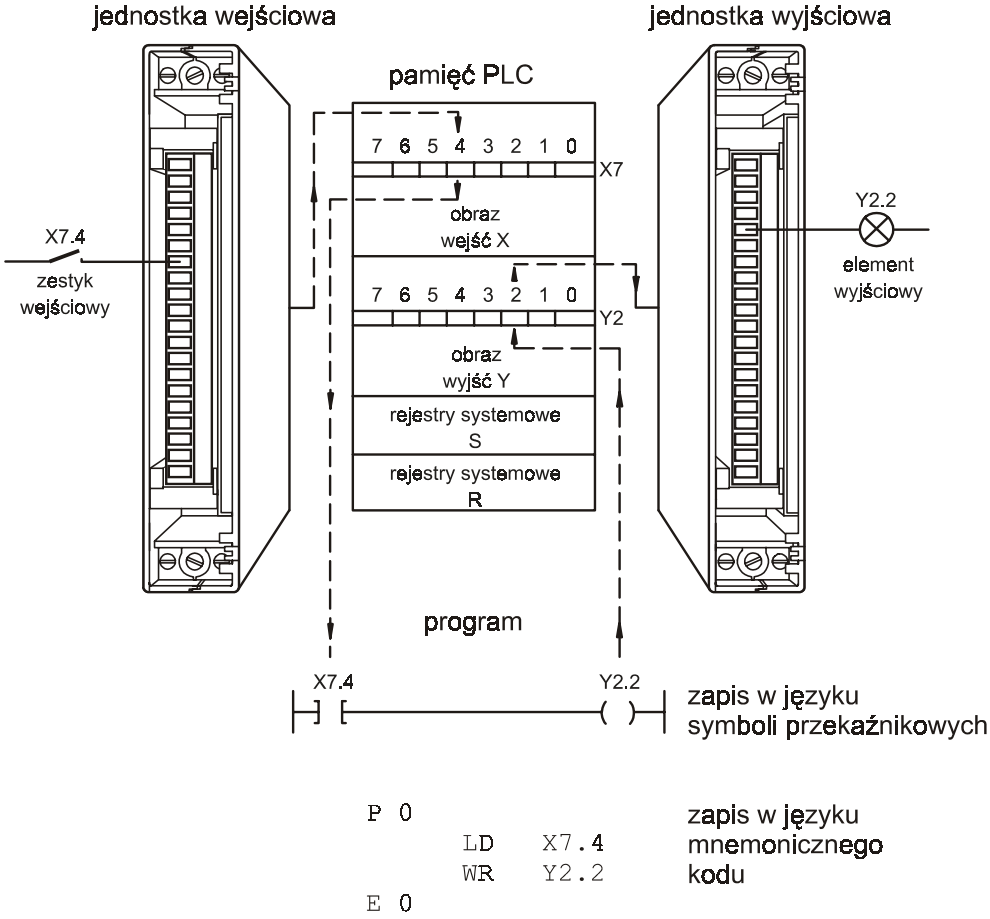
Na rys.2.1 jest podany uproszczony schemat wykonywania programu w PLC z następującymi wyjaśnieniami:

- sekwencja załączeniowa - jest to czynność PLC po załączeniu zasilania (patrz rozdz.2.1.)
- restart - czynność PLC bezpośrednio przed rozpoczęciem wykonywania programu (patrz rozdz.2.3.)
- tryb RUN i tryb HALT są trybami roboczymi PLC (patrz rozdz.2.2.)
- odczyt wejść polega na kopiowaniu wartości z jednostek wejściowych PLC do obszaru X w pamięci
- wykonywanie programu przebiega z wartościami zapisanymi w pamięci

- zapis do wyjść polega na skopiowaniu wartości wyliczonych w programie z obszaru Y do jednostek wyjściowych PLC
- działania wewnętrzne obejmują przygotowanie jednostki centralnej PLC do wykonywania kolejnego cyklu programu

Czynności zapisu do wyjść, działania wewnętrzne oraz odczyt z wejść są wspólnie nazywane obrotem cyklu.

Obsługa wejściowych sygnałów oraz generowanie sygnałów wyjściowych



Rys.2.2 Schemat obsługi sygnału w sterowniku przemysłowym

2.1. SEKWENCJA ZAŁĄCZENIOWA

Załączeniowa sekwencja

Załączeniowa sekwencja jest czynnością PLC wykonywaną bezpośrednio po załączeniu zasilania. Obejmuje testowanie sw i hw PLC oraz nastawienie PLC do zdefiniowanego stanu wyjściowego. W centralnych jednostkach wyposażonych w klawisze nastawieniowe lub w PLC posiadających panel z klawiaturą (TC500), można po załączeniu zasilania wywołać tryb nastawieniowy, przeznaczony do nastawiania parametrów.

Po zakończeniu sekwencji załączeniowej wykonywany jest restart, PLC przechodzi do trybu RUN i zaczyna wykonywać program użytkownika. Jeśli w trakcie sekwencji załączeniowej diagnostyka PLC wykryje błąd krytyczny, PLC pozostaje w trybie HALT i sygnalizuje błąd.

Jeśli został wywołany tryb nastawieniowy, to po jego zakończeniu następuje sekwencja załączeniowa, ale PLC przejdzie wówczas do trybu HALT, program użytkownika nie jest wykonywany, wyjścia PLC pozostają zablokowane a PLC oczekuje na polecenia z nadrzędnego systemu. Program użytkownika można uruchomić za pomocą nadrzędnego systemu lub poprzez wyłączenie i załączenie zasilania. Tę własność można wykorzystać w przypadkach, kiedy do PLC dostanie się program, który w znaczny sposób wpływa na jego podstawowe funkcje.

Szczegóły dotyczące poszczególnych typów PLC są podane w odpowiednich podręcznikach.

2.2. TRYBY ROBOCZE PLC

PLC TECOMAT może pracować we dwóch podstawowych trybach. Tryby te są oznaczone RUN i HALT.

Tryb RUN

W trybie RUN PLC odczytuje wartości sygnałów wejściowych z jednostek wejściowych, wykonuje instrukcje programu i zapisuje wyliczone wartości wyjściowych sygnałów do jednostek wyjściowych. Jedno wykonanie tych czynności przedstawia cykl programu.

Z rys.2.1 wynika, że w przypadku PLC chodzi o nieciągłe odczytywanie wejść (ogólna własność odróżniająca cyfrowe systemy od systemów w pełni analogowych), którego częstotliwość próbkowania jest dana czasem cyklu, zależnym przede wszystkim od rozmiarów i struktury programu. Czas cyklu zależy od mocy obliczeniowej jednostki centralnej i waha się od jednostek do setek milisekund.

Tryb HALT

Tryb HALT służy przede wszystkim do czynności związanych z edycją programu użytkownika. W tym trybie program nie jest wykonywany ani nie są przesyłane dane pomiędzy centralną jednostką a peryferiami.

Działania PLC przy błędzie krytycznym

Wyjątek z podanych reguł tworzy sytuacja, kiedy w PLC powstanie błąd krytyczny, uniemożliwiający kontynuację sterowania. W tym przypadku w PLC uruchomiony jest mechanizm obsługi błędu krytycznego, który obsługuje błędy z punktu widzenia bezpieczeństwa sterowania i **zawsze** wprowadzi PLC do trybu HALT.

Zmiana trybów roboczych

Zmianę trybów roboczych PLC można wykonywać za pomocą nadrzędnego systemu (komputer), który jest podłączony do szeregowego kanału CH1, lub za pomocą wejść służbowych. Typowo tym nadrzędnym systemem jest komputer standardu PC, który pracuje w funkcji urządzenia programowego lub monitorującego, ew. stanowiska wizualizacyjnego dla obsługi sterowanego obiektu.

Przy zmianie roboczych trybów PLC są niektóre czynności wykonywane standardowo, a niektóre można wykonywać opcjonalnie. Ogólnie przyjmuje się, że zmiana trybu roboczego PLC jest czynnością wymagającą zwiększonej uwagi obsługi, ponieważ w wielu przypadkach znacznie wpływa na stan sterowanego obiektu. Przykładem może być przejście z trybu RUN do trybu HALT, kiedy PLC przestanie wykonywać program użytkownika a podłączony obiekt przestaje być sterowany. Dlatego zaleca się dokładnie przeanalizować następujący tekst.

W przypadku, gdy zmiana trybu PLC jest przeprowadzana za pomocą programu xPRO, opcjonalne czynności przy zmianie trybu są częścią menu *Opcje | Automat*.

Przejście z HALT do RUN

Podczas przejścia z trybu HALT do RUN następuje:

- test nienaruszenia programu użytkownika
- kontrola programowej konfiguracji jednostek peryferyjnych w programie użytkownika
- uruchomienie programu użytkownika

Następnie można opcjonalnie wykonywać:

- zerowanie błędów PLC
- ciepły lub zimny restart
- blokowanie wyjść podczas wykonywania programu użytkownika

Przejście z RUN do HALT

Podczas przejścia z trybu RUN do HALT następuje:

- zatrzymanie biegu programu użytkownika
- zablokowanie (odłączenie) wyjść PLC

Następnie można opcjonalnie wykonać:

- zerowanie błędów PLC
- zerowanie wyjść PLC

Jeśli w trakcie czynności wykonywanych podczas przejścia pomiędzy trybami powstanie błąd krytyczny, PLC przejdzie do trybu HALT, wskazuje błąd i oczekuje na usunięcie przyczyny błędu.

Uwaga: Zatrzymanie sterowania za pomocą trybu HALT jest przeznaczone jedynie dla celów strojenia programu PLC. Ta funkcja w żadnym przypadku nie zastępuje funkcji CENTRAL STOP. Układy CENTRAL STOP muszą być podłączone tak, aby ich działanie było niezależne od pracy PLC !

Szczegóły o zachowywaniu się oraz możliwościach poszczególnych typów PLC są podane w odpowiednich podręcznikach.

2.3. RESTARTY PROGRAMU UŻYTKOWNIKA

Restart programu użytkownika

Restartem rozumiana jest czynność PLC, która przygotowuje PLC do wykonywania programu użytkownika. Restart w normalnych warunkach wykonywany jest po pomyśl-

nym ukończeniu sekwencji załączeniowej oraz przy każdej zmianie programu użytkownika.

PLC TECOMAT rozróżniają dwa rodzaje restartu: ciepły i zimny. Ciepły restart umożliwia zachowanie wartości w rejestrach nawet w trakcie wyłączonego zasilania (strefa remanentna). Zimny restart zawsze przeprowadza pełną inicjalizację pamięci.

Czynności wykonywane podczas restartu

Podczas restartu wykonywany jest:

- test nienaruszenia programu użytkownika
- zerowanie całej pamięci PLC
- zerowanie remanentnej strefy (jedynie zimny restart)
- nastawianie podtrzymywanych rejestrów (jedynie ciepły restart)
- inicjalizacja systemowych rejestrów S
- inicjalizacja i kontrola peryferyjnego systemu PLC

Uruchomienie programu bez restartu

Program użytkownika można również uruchomić bez restartu. W tym przypadku następuje jedynie test nienaruszenia programu użytkownika oraz kontrola peryferyjnego systemu PLC (bez inicjalizacji).

Procesy użytkownika podczas restartu

W zależności od rodzaju restartu pracuje również układ planowania procesów użytkownika P. Jeśli podczas przejścia HALT → RUN został wykonany ciepły restart, wówczas po przejściu do trybu RUN jako pierwszy wykonywany jest proces użytkownika P62 (jeśli jest zaprogramowany). Przy zimnym restarcie jako pierwszy po przejściu do trybu RUN jest wykonywany proces użytkownika P63. Jeśli przy przejściu do trybu RUN nie jest żaden restart wykonywany, jako pierwszy wykonywany jest proces P0.

Zmiana programu podczas pracy PLC

Programowy software xPRO umożliwia też zmianę programu podczas pracy PLC. Należy jednak pamiętać o tym, że podczas wgrywania nowego programu jest wstrzymane wykonywanie programu bez zablokowania wyjść. Ten stan może trwać nawet kilka sekund!

Wybór typu restartu

Typ restartu można w środowisku xPRO nastawić w menu *Opcje | Automat* dla uruchomienia programu ze środowiska xPRO. Dla wyboru typu restartu po załączeniu zasilania PLC (po pomyślnej sekwencji załączeniowej) jest przeznaczony wybór w menu *Opcje | Kompilator* lub dyrektywa *#option* umieszczona w programie użytkownika.

3. STRUKTURA PROGRAMU UŻYTKOWNIKA

Programowy język używany w środowisku xPRO

Programowy język xPRO wywodzi się z mnemonicznego języka PLC opisanego w rozdz.4. Rozszerzenie polega na możliwości używania symbolicznych nazw zmiennych (rejestrów, etykiet, itd.) oraz używaniu dyrektyw dla kompilatora.

Reguły dla zapisu programu

Zapis programu użytkownika kieruje się kilkoma prostymi regułami:

- Program zaczyna się nagłówkiem, składającym się z obowiązkowej dyrektywy *#program*, po której następuje nazwa programu (maks. 16 znaków) i ewentualnie odseparowana przecinkiem wersja programu.
- Na jednym wierszu źródłowego tekstu programu może być co najwyżej jedna instrukcja. To oznacza, że wiersz może być też pusty, lub może zawierać jedynie komentarz. Uwaga - etykieta jest również instrukcją!
- Symboliczne nazwy mogą zaczynać się od liter 'a' do 'z', 'A' do 'Z' lub '_', i mogą zawierać znaki 'a' do 'ž', 'A' do 'Ž', '0' do '9' i '_' (podkreślenie). Z tego wypływa, że nazwa symboliczna nie może zaczynać się ani od cyfry, ani od litery z diakrytyką (haczyki i kreski, np. ‘ą’).

Uwaga!	**Nie wolno, żeby symboliczna nazwa była identyczna z którymś ze zastrzeżonych symboli kompilatora (patrz tab.3.1).**

- Komentarze poprzedzone są znakiem ';' (średnik). Wszelki tekst za tym znakiem aż do końca wiersza jest przez kompilator traktowany jako komentarz i jest ignorowany.
- Małe i wielkie litery można używać dowolnie. Kompilator wewnętrznie konwertuje wszystkie litery na wielkie, nie rozróżnia więc małe i wielkie litery.

Zgodnie z tymi regułami można napisać prosty program:

Przykład 3.1

```
#program PRZYKLAD1, V1.0            ;nagłówek programu
;
#def    StartStop     X0.0          ;deklaracja wejść, wyjść
#def    Wyjscie       Y0.0          ;i stałych
#def    Wartosc       21
#reg    bit    StZakaz              ;deklaracja rejestrów
#reg    word   Timer, Licznik
;
#unit 0, 0, DigitIn16, X0, X_on     ;sw konfiguracja
#unit 0, 1, DigitOut16, Y0, Y_on
;
P 0                                 ;początek programu
        LD     StartStop            ;bit sterujący układu czasowego
        LD     Zerowanie            ;zerowanie układu czasowego
        LD     Wartosc              ;próg układu czasowego
        RTO    Timer.3              ;funkcja sekund układu czasowego
        WR     Wyjscie              ;wyjście
        JMC    Skok                 ;warunkowy skok
        INR    Licznik              ;ilość cykli, dopóki Wyjscie = 0
Skok:                               ;etykieta
E 0                                 ;koniec programu
```

Podstawowa struktura programu użytkownika

Nagłówek programu w postaci dyrektywy *#program* z nazwą programu jest obowiązkowy, nie można go więc ominąć.

Jeśli opuści się deklarację wejść, wyjść, stałych i rejestrów, można wówczas pisać program użytkownika za pomocą absolutnych adresów (tj. rejestry X, Y, S, R). Ten sposób jednak stanowczo nie jest zalecany, ponieważ nie jest przejrzysty i bardzo utrudnia ewentualne zmiany programu. Dodatkowo niektóre programowe konstrukcje używające struktury i symboliczne odnośniki są w postaci absolutnej praktycznie nie do zrealizowania.

Jeśli opuści się programową konfigurację (opis użytych jednostek za pomocą dyrektywy *#unit*), PLC będzie wykonywać program użytkownika jedynie w pamięci. Wejścia nie będą do pamięci wgrywane a do wyjść nie będzie się z pamięci zapisywać obliczonych wartości. Wyjścia zostaną zablokowane. Ten stan może być użyteczny podczas strojenia algorytmu w PLC bez podłączonej technologii.

Każdy program użytkownika musi zawierać proces P0, choćby pusty. Instrukcje P 0 i E 0 są więc obowiązkowe.

Komentarze nie są obowiązkowe, ale przyjmuje się zasadę, że im bardziej szczegółowo skomentowany program, tym łatwiej do niego powrócić nawet po roku, kiedy już niewiele się pamięta, a niezbędna jest w programie doróbka lub zmiana.

Wcięcia instrukcji za pomocą tabulatorów, jak zaznaczono w przykładzie, również nie jest konieczne. Do separacji operandu wystarczy spacja a wszystkie instrukcje można pisać od początku wiersza. Wszelkie wcięcia zwiększają jednak czytelność programu, co zminimalizuje ilość błędów.

Zastrzeżone symboly kompilatora xPRO

Tab.3.1 Spis zastrzeżonych symboli kompilatora, które nie można użyć jako symboliczną nazwę (jest ważne dla wielkich i małych liter)

A	CPM1B	ED	L	RTO	WARM
ABS	CPM2B	EOC	L0, L1...	RW0, RW1...	WMS
ACS	CPM1D	EQ	LABEL	S	WORD
ADD	CPM2D	ENDIF	LAC	S0, S1...	WR
ADF	CPM1E	ENDM	LD	SCH2	WRA
ADL	CPM1M	ES	LDC	SEQ	WRC
ADX	CPM1S	EXP	LDL	SET	WRS
ALIGNED	CPM2S	F	LDS	SF0, SF1...	WTB
ANALOG_500	CPM3S	FIL	LET	SFL	X
ANALOG_600	CS	FIS	LMS	SFR	X0, X1...
ANC	CTD	FIT	LN	SIN	XF0, XF1...
AND	CTU	FLG	LOG	SIZEOF	XH_04
ANL	D	FLO	LONG	SQR	XL0, XL1...
AN_4IN	D0, D1...	FLOAT	LOW	SL0, SL1...	XOC
AN_4IN_4OUT	DATA	FNS	LT	SND	XOL
AN_8IN	DCR	FNT	LTB	SPEC	XOR
AN_8IN_4OUT	DEF	FS	MACRO	SPECIAL	XW0, XW1...
AS	DF0, DF1...	FST	MNT	SRC	X_OFF
ASB	DID	FTB	MOV	SRD	X_ON
ASN	DIF	FTM	MTN	STE	Y
ATN	DIFCNT100MS	FTS	MUD	STF	Y_OFF
B	DIG2	G	MUF	STK	Y_ON
BAS	DIG4	GS	MUL	STRUCT	Y0, Y1...
BCD	DIG8	GT	NEG	SUB	YF0, YF1...
BCL	DIGIN16	H	NGL	SUF	YL0, YL1...
BET	DIGIN8	HIGH	NOP	SUL	YW0, YW1...
BIL	DIGIN8OUT8	HPD	NXT	SUX	_ANAL_
BIN	DIGIT2	HPE	OFF	SW0, SW1...	_ANALOG_
BIT	DIGIT4	HS	ON	SWL	_CHX
BITPART	DIGIT8	HYP	OPTION	SWP	_GT_40_
BOX	DIGITIN16	IC_04	OR	SYNC	_GT_40A_
BP	DIGITIN32	IF	ORC	SYS	_IC_12_
BRC	DIGITIN64	IFL	ORL	T	_IC_13_
BRD	DIGITIN8	IFW	P	T0, T1...	_IM_61_
BRE	DIGITOUT16	ILF	PERIOD_500	TABLE	_INTELIG
BS	DIGITOUT32	IMP	PERIOD_600	TAN	_INTELIGENT
BYTE	DIGITOUT64	INCLUDE	PID	TF0, TF1...	_IR_11_
C	DIGITOUT8	INDX	PLC	TL0, TL1...	_IT_04_
CAC	DIGIT_500	INR	POP	TOF	_IT_06_
CAD	DIGIT_600	INTIN_500	POW	TON	_IT_12_
CAI	DIGIT_633	INTIN_600	PROGRAM	TW0, TW1...	_IT_15_
CAL	DIGIT_63X	IRC_500	PRV	U	_KEYDISP_500_
CEI	DIGOUT16	IRC_600	PUBLIC	U0, U1...	_OT_04_
CH1, CH2...	DIGOUT8	IWF	PUT	UF0, UF1...	_OT_04X_
CHG	DIG_10IN_10OUT	JC	R	UFL	_OT_05
CMF	DIG_5IN_6OUT	JMC	R0,R1...	UFW	_OT_05X
CML	DISPASCII	JMD	REC	UL0, UL1...	_PLCTYPE_
CMP	DISPHEX	JMI	RED	ULF	_SC_11_
CNT	DIV	JMP	REG	UNIT	_SPECIALTAB_
CNV	DL0, DL1...	JNC	RES	USI	_SPECTAB_
COLD	DS	JNS	RET	UWF	
COS	DT	JNZ	RF0, RF1...	UW0, UW1...	
COUNT_500	DW0, DW1...	JS	RL0, RL1...	UX_52	
COUNT_600	E	JZ	ROL	VIRTMUX	
CPM1A	EC	KEYDISP_200_	ROR	WAC	

Notatka: Ze względu na nieustający rozwój i rozprzestrzenianie programowego środowiska xPRO polecamy nieużywać jedno- do trzyliterowych symbolicznych nazw tak samodzielnie, jak ani z indeksem numerowym. W przypadku nieodzownej konieczności użyć taką nazwę polecamy napisać przed nią znak podkreślnik (naprz. _A).

4. STRUKTURA INSTRUKCJI I OPERANDÓW

Instrukcja

Instrukcja jest najmniejszym elementem programu użytkownika. Składa się z mnemokodu i operandu. Z formalnego punktu widzenia rozróżnia się instrukcje bezoperandowe oraz instrukcje z jednym operandem.

Mnemokod

Mnemokodem określa się łańcuch jednego do trzech znaków, które mają znaczenie skrótu pochodzącego najczęściej od angielskiej nazwy instrukcji (np. AND, OR, XOR, NEG, FLG, RET, ED, EC).

Bezoperandowa instrukcja

Bezoperandowa instrukcja z reguły obsługuje zawartość szczytu, ewentualnie pozostałych warstw akumulatora lub wykonuje inną jednoznacznie specyfikowaną czynność (np. powrót z podprogramu). Bezoperandową instrukcję tworzy sam mnemokod, a jej działania nie trzeba bliżej specyfikować.

Instrukcja z jednym operandem

W operandowej instrukcji za mnemokodem następuje ciąg znaków, które w określony sposób specyfikują jeden z operandów, z którym instrukcja pracuje, lub które bliżej specyfikują działanie instrukcji (np. parametr instrukcji, ilość powtórzeń podstawowej operacji, miejsce skoku itp.). Drugim operandem dla logicznych i arytmetycznych operacji jest szczyt akumulatora. Operand jest od mnemokodu oddzielony minimalnie jedną spacją. Operandowe instrukcje mogą mieć np. postać:

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

Rodzaje operandów

Ze względu na znaczenie można rozróżnić cztery rodzaje operandów:

bezpośredni operand - operandem instrukcji jest bezpośrednio wartość liczbowa (zapisana w dowolnym układzie liczbowym), z którą instrukcja zostanie wykonana

adresowy operand - określa adres miejsca, skąd następuje odczyt lub gdzie następuje zapis wyniku operacji

cel skoku - operandem jest numer etykiety (oznaczone miejsce w programie), gdzie jest skierowane przejście programowe (skok lub wywołanie podprogramu)

parametr instrukcji - prosty parametr liczbowy lub literowy, który określa daną instrukcję (np. numer procesu, numer etykiety, numer pustej instrukcji, akumulator) lub uściśla jej pracę (np. ilość przesunięć akumulatora).

Z punktu widzenia formatu inne instrukcje nie są stosowane. Jeśli któraś z instrukcji lub podprogram wymaga kilku parametrów (bloki funkcyjne, instrukcje tabelkowe lub blokowe), są one przekazywane w kilku poziomach akumulatora, do których są zapisywane za pomocą instrukcji LD lub LDC.

4.1. BEZPOŚREDNI OPERAND

Stała liczbowa

W tym przypadku jako operand instrukcji zapisana jest liczba, która jest bezpośrednio przetworzona przez instrukcję - dane stanowiące część instrukcji. Bezpośredni operand ma więc znaczenie liczbowe wartości stałej.

4.1.1. Układy liczbowe

Zapis wartości stałej w ogólnym układzie liczbowym

Stałą liczbową można zadać w dowolnym układzie liczbowym w następującej postaci:

#n#cccc

gdzie n jest podstawa układu liczbowego,

cccc jest liczbą w wybranym układzie

Jeśli podstawa układu liczbowego jest większa niż 10, można poszczególne cyfry zapisywać dziesiętnie (w formie liczby dwucyfrowej) i oddzielić je od siebie kropką.

```
LD     #8#360        ;oktalowy (ósemkowy) układ
WR     #60#15.28.35  ;zapis danych czasowych
                     ;w godzinach, minutach i sekundach
                     ;za pomocą układu
                     ;sześćdziesiątkowego
```

Skrócony zapis w najczęściej używanych układach liczbowych

Najczęściej używanymi układami liczbowymi są układy dziesiętne, przede wszystkim dla instrukcji arytmetycznych, binarne (dwójkowe) oraz heksadecymalne (szesnastkowe) przede wszystkim dla instrukcji logicznych. Te układy umożliwiają skrócony zapis.

Jeśli liczba jest zapisana bez podania układu liczbowego, traktowana jest jako dziesiętna. Binarny układ oznacza się znakiem % przed liczbą. Heksadecymalny układ oznacza się znakiem $ przed liczbą, w którym używa się cyfr 0 do 9, oraz liter A do F.

```
LD     152                ;układ dziesiętny
AND    %0111110100110100  ;układ binarny
AND    $7D34              ;układ heksadecymalny
```

Liczby ujemne

Dla układu dziesiętnego dopuszczalny jest też zapis liczby ujemnej. Znak minus powoduje, że zamiast dwójkowego ekwiwalentu zadanej liczby, jako operand instrukcji zostanie zapisane dwójkowe dopełnienie tej liczby w zakresie byte, word lub long (w zależności od typu instrukcji).

Na przykład wartość –1 w formacie byte będzie odpowiadać wartości 255, w formacie word wartości 65 535, w formacie long wartości 4 294 967 295.

Format float już zawiera informację o znaku.

4.1.2. Formaty bezpośrednich danych

Format wartości stałej jest określony przez typ użytej instrukcji

Format stałej wartości liczbowej jest jednoznacznie dany typem instrukcji. Instrukcje oczekują wartości stałych w formatach byte, word, long lub float (patrz opis odpowiedniej instrukcji).

Kompilator automatycznie uzupełnia stałą zerami na wymagany format

Jeśli do instrukcji, która wymaga stałej liczbowej o długości long, zapisze się liczbę o długości byte, kompilator dopełnia ją zerami na długość long. Ten sposób jest więc dopuszczalny.

```
LDL    $1F                ;oczekuje się format long
LDL    $0000001F          ;zgodny zapis
AND    %11                ;oczekuje się format word
AND    %0000000000000011  ;zgodny zapis
```

Stała liczbowa formatu float jest dopuszczalna jedynie dla układu dziesiętnego. Ze względu na to, że format float ma identyczną długość jak format long, ale zupełnie inną interpretację danych, zapis liczbowej wartości stałej formatu float wyróżnia się przede wszystkim tym, że zawiera kopkę dziesiętną nawet wówczas, gdy chodzi o liczbę całkowitą. Obecność kropki dziesiętnej jest więc informacją ważną dla kompilatora xPRO, aby skompilował stałą do poprawnego formatu.

```
LDL    1          ;format long
LDL    1.0        ;format float
```

Symboliczna nazwa jako operand

Jako bezpośredni operand można też użyć symbolicznej nazwy zdefiniowanej za pomocą dyrektywy *#def*. Można również użyć matematycznego wyrażenia.

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

Numer obiektu jako operand

Jeśli potrzebujemy jako operand użyć numer rejestru, tabelki lub etykiety, użyje się prefiksu *indx*.

```
#reg   word   rejestr1,rejestr2          ;RW0,RW2
       LD     indx rejestr1       ;LD 0
       LD     indx rejestr2       ;LD 2
```

Długość obiektu jako operand

Za pomocą prefiksu *sizeof* można pracować z długością obiektu, najczęściej struktury.

```
#struct lista byte pierwszy, word drugi, float trzeci
       LD     sizeof lista        ;LD 7
```

Zakresy bezpośrednich operandów w zależności od formatu

Tab.4.1 Zakresy bezpośrednich operandów

byte	word
–128 do +127	–32768 do +32 767
0 do 255	0 do 65535
%0 do %11111111	%0 do %1111111111111111
$0 do $FF	$0 do $FFFF
#60#0 do #60#8.15	#60#0 do #60#18.12.15

long	float
–2147483648 do +2147483647	±1,175494x10^{-38} do ±3,402823x10^{38}
0 do 4294967295	
%0 do %11111111111111111111111111111111	-
$0 do $FFFFFFFF	-
#60#0 do #60#59.59.59 *	-

* Sześćdziesiątkowy układ stosuje się tylko dla godziny, minuty i sekundy, nie dla dni.

4.2. ADRESOWY OPERAND

Typ obszaru adresowego operandu

Adresowy operand ma znaczenie adresu miejsca, skąd zostaje odczytana informacja lub miejsca, gdzie następuje zapis wyniku operacji. Typ obszaru operandu jest oznaczony pierwszym znakiem w operandzie:

X - obraz wejść w pamięci
Y - obraz wyjść w pamięci
S - systemowe rejestry w pamięci
R - rejestry użytkownika w pamięci
U - fizyczne adresy
D - strefa danych programu użytkownika
T - strefa tabel w programie użytkownika

Ten pierwszy znak jest niekiedy oznaczany jako specyfikator obszaru operandu.

Typ operandu

W zależności od szerokości danych rozróżnia się operandy następujących typów:

bit - dwustanowa informacja 0 - 1, nie - tak, wyłączone - załączone
byte - 8 bitów, wartość liczbowa w zakresie 0 do 255, ew. –128 do +127
word - 16 bitów, 2 bajty, wartość liczbowa w zakresie 0 do 65 535, ew. –32 768 do +32 767
long - 32 bitów, 4 bajty, wartość liczbowa w zakresie 0 do 4 294 967 295, ew. – 147 483 648 do +2 147 483 647
float - 32 bitów, 4 bajty, wartość liczbowa w formacie zmiennoprzecinkowym w zgodzie z IEEE-754 w zakresie ±1,175494x10^{-38} do ±3,402823x10^{38}

Liczba, która się w adresowym operandzie pojawia, jest liczbą (adresem) w danym obszarze. Ta liczba jest w postaci dziesiętnej, z wyjątkiem operandu U, gdzie chodzi o fizyczny heksadecymalny adres o szerokości word.

Symboliczna nazwa jako operand

Jako adresowy operand z reguły używa się symbolicznych nazw przyporządkowanych za pomocą dyrektywy *#def*, *#reg*, *#data* lub *#table*.

```
#def     wejscie    X0
#reg     bit        rejestr          ;R0.0
#data    byte       dane=1,2,3,4     ;D0, D1, D2, D3
#table   word       10,tab=1,2,3,4   ;TW10
```

Zakresy adresowych operandów dla poszczególnych typów CPU

Tab.4.2 Zakresy adresowych operandów jednostek centralnych typu E i M

bit	byte	word
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 *

* W jednostkach centralnych typu E nie jest implementowane

Tab.4.3 Zakresy adresowych operandów jednostek centralnych typu S

bit	byte	word

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.4.4 Zakresy adresowych operandów jednostek centralnych typu B i D

bit	byte	word	long	float
X0.0 - X127.7	X0 - X127	XW0 - XW126	XL0 - XL124	XF0 - XF124
Y0.0 - Y127.7	Y0 - Y127	YW0 - YW126	YL0 - YL124	YF0 - YF124
S0.0 - S63.7	S0 - S63	SW0 - SW62	SL0 - SL60	SF0 - SF60
R0.0 - R8191.7	R0 - R8191	RW0 - RW8190	RL0 - RL8188	RF0 - RF8188
-	U$0000 -U$FFFF	UW$0000 -UW$FFFE	-	-
D0.0 - D255.7	D0 - D255	DW0 - DW254	DL0 - DL252	DF0 - DF252
T0.0 - T255.0	T0 - T255	TW0 - TW255	-	-

Lista tabel T w programie użytkownika

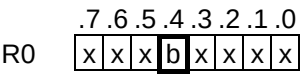
Uwaga: Każda tabelka T oprócz danych zajmuje w pamięci jeszcze dodatkowo 4 bajty dla informacji systemowych. Częścią programu użytkownika jest lista adresów tabel T wytworzona przez kompilator. Ta lista ma n+1 pozycji, gdzie n oznacza najwyższy numer tabelki zadeklarowany w programie. Na liście są adresy wszystkich tabel od T0 do Tn wraz z tymi, które nie są zadeklarowane (mają zerowy adres). Z tego wypływa, że jeśli w programie użyje się jedynie tabelki o wysokim numerze, tabelka adresów zajmie zbytecznie wielką część pamięci przeznaczonej dla programu użytkownika. Dlatego zaleca się numerować tabelki rosnąco od 0 (kompilator xPRO wspiera tę zasadę przy używaniu symbolicznych nazw tabel).

4.2.1. Operand typu bit

Dane formatu bit zajmują w pamięci jeden konkretny bit w bajcie, określony adresem bajtu i numerem bitu.

Zakres formatu

Dane formatu bit osiągają wartości 0 i 1 (ze względu na inną interpretację na akumulatorze oznaczane są jako log.0 i log.1).



Rys.4.1 Zapis danych formatu bit w pamięci
b - wartość logiczna bitu (log.0 lub log.1)

Zapis operandu typu bit

Operand typu bit jest adresowany numerem bajtu w obszarze operandów oraz numerem bitu wewnątrz danego bajtu. W absolutnej postaci adres bajtu jest jednoznacznie określony numerem w adresowym operandzie. Numer bitu ma wartość 0 do 7 i zadawany jest za kropką, np.:

```
LD      X1.2
WR      Y1.7
AND     S53.4
XOR     R123.6
LDC     D25.4
```

Numerowi 0 odpowiada najniższy (dolny) bit bajtu, numerowi 7 odpowiada najwyższy (górny) bit bajtu. Jeśli zawartość bajtu wyświetlana jest jako liczba dwójkowa lub jako ciąg wartości bitowych w jednym wierszu, wówczas dolny bit jest podawany z prawej strony, górny bit z lewej strony.

Operandy U nie umożliwiają bitowego dostępu.

Operandy T określające bitowy dostęp do tabel mają numer bitu zawsze 0. W ten sposób jest odróżniony bitowy dostęp od bajtowego, numer bitu nie ma w tym przypadku żadnego znaczenia.

```
LTB     T4.0    ;odczyt pozycji z bitowej tabelki T4
                ;(numer pozycji jest określony indeksem
                ;zapisanym na szczycie akumulatora)
```

Symboliczny zapis

W symbolicznej postaci operand typu bit jest określony dyrektywami *#def*, *#reg*, *#data* i *#table*.

```
#def    wejście X2.1
#reg    bit     rejestr         ;R0.0
#data   bit     dane=1,0,1,0    ;D0.0, D0.1, D0.2, D0.3
#table  bit     tab=1,0,1,1     ;T0.0
```

Przetypowanie na bit

Jeśli w programie użytkownika zaistnieje potrzeba lokalnej konwersji typu operandu (czyli użycia innego typu operandu, niż był zdefiniowany za pomocą wyżej wymienionych dyrektyw), można go lokalnie skonwertować na typ bit dodając jedynie numer bitu. Bity można numerować w całej szerokości operandu, czyli 0 do 7 dla formatu byte, 0 do 15 dla formatu word oraz 0 do 31 dla format long.

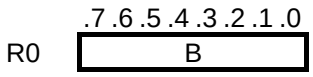
```
#reg    byte   rejestr1         ;R0
#reg    word   rejestr2         ;RW1
#reg    long   rejestr3         ;RL3
#table  byte   tab=1,2,3,4      ;T0
;
        LD     rejestr1.0       ;LD  R0.0
        LD     rejestr2.5       ;LD  R1.5
        LD     rejestr3.31      ;LD  R6.7
        LTB    tab.0            ;LTB T0.0
```

4.2.2. Operand typu byte

Dane formatu byte zajmują w pamięci jeden konkretny bajt określony adresem.

Zakres formatu

Dane formatu byte osiągają wartości 0 do 255 lub przy użyciu najwyższego bitu jako znaku minus –128 do +127.



Rys.4.2 Zapis danych formatu byte w pamięci
B - wartość bajtu

Zapis operandu typu byte

Operand typu byte w postaci absolutnej jest adresowany numerem w operandzie adresowym.

```
        LD      X1
        WR      Y0
        AND     S53
        XOR     R123
        LDC     D25
        LTB     T4
```

Operand U typu byte jest adresem fizycznym adresowanym heksadecymalnie o szerokości word ($ oznacza heksadecymalny układ liczbowy).

```
        LD      U$9101          ;heksadecymalna wartość adresu
```

Symboliczny zapis

W symbolicznym zapisie operand typu byte jest określony dyrektywami *#def*, *#reg*, *#data* oraz *#table*.

```
#def   wejscie X2
#reg   byte    rejestr        ;R0
#data  byte    dane=1,2,3,4   ;D0, D1, D2, D3
#table byte    tab=1,2,3,4    ;T0
```

Przetypowanie na byte

Jeśli w programie zaistnieje potrzeba lokalnego przetypowania operandu (czyli użycia innego typu operandu, niż jest zdefiniowany za pomocą wyżej podanych dyrektyw), można go lokalnie przetypować na typ byte poprzez dopisanie prefiksu *byte*.

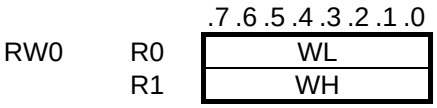
```
#reg   bit    rejestr1             ;R0.0
#reg   word   rejestr2             ;RW1
#reg   long   rejestr3             ;RL3
#table word   tab=1,2,3,4          ;TW0
;
        LD     byte rejestr1       ;LD  R0
        LD     byte rejestr2       ;LD  R1
        LD     byte rejestr3+3     ;LD  R6
        LTB    byte tab            ;LTB T0
```

4.2.3. Operand typu word

Dane formatu word w pamięci zajmują dwa konkretne bajty określone adresem pierwszego z nich. Dane są zapisane tak, że niższy znaczeniowo bajt ma niższy adres niż znaczeniowo wyższy bajt (konwencja Intel).

Zakres formatu

Dane formatu word osiągają wartości 0 do 65 535 lub przy użyciu najwyższego bitu jako znaku minus –32 768 do +32 767.



Rys.4.3 Zapis danych formatu word w pamięci
WH - wartość wyższego bajtu
WL - wartość niższego bajtu

Zapis operandu typu word

Adres operandu typu word w absolutnej postaci jest zapisywany tak, że za specyfikatorem obszaru jest zapisany znak W jako symbol typu operandu, a za nim wartość liczbowa, która adresuje niższy bajt wordu. Jeśli na przykład rejestr R14 posiada wartość $21 a rejestr R15 wartość $43, wówczas za pomocą instrukcji

```
LD      RW14
```

zostanie odczytana wartość $4321. Podobnie zapisywane są pozostałe operandy.

```
LD      XW1
WR      YW0
AND     SW53
XOR     RW123
LDC     DW25
LTB     TW4
```

Podobnie użycie operandu U typu word oznacza pracę z dwoma fizycznymi adresami następującymi po sobie. Jeśli więc mamy na przykład szesnastowejściową jednostkę binarną z adresem 2, której stan sygnałów wejściowych bajtu 0 wynosi binarnie %1101 0001 a bajtu 1 wynosi binarnie %0010 1100, wówczas za pomocą instrukcji

```
LD      UW$9200
```

zostanie odczytana wartość $2CD1.

Kodowanie jednostek czasu w instrukcjach liczników

Instrukcje liczników (TON, TOF, RTO, IMP) pracują jedynie z operandem typu word, który zawiera aktualną wartość układu czasowego. Dodatkowo należy zadać jednostkę czasu, z którą instrukcja ma pracować i w której jest odmierzana wartość przyrostu czasu. Możliwe są cztery jednostki czasu, zadawane w postaci liczby kodowej 0 do 3, podanej w instrukcji po numerze adresu. Obie wartości są odseparowane kropką. Jednostki czasu są zakodowane w następujący sposób:

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

Jeśli kod jednostki czasu nie jest podany, przyjmuje się jako równy zeru, tj. jednostka 10 ms. Poniżej podano przykłady zapisu instrukcji liczników:

```
TON     RW10.0        ;układ czasowy jest w RW10,
                      ;jednostka 10 ms
TON     RW10          ;równoznaczny zapis
TOF     RW24.1        ;układ czasowy jest w RW24,
                      ;jednostka przyrostu czasu 100 ms
RTO     RW32.2        ;układ czasowy jest w RW32,
                      ;jednostka przyrostu czasu 1 s
IMP     RW34.3        ;układ czasowy jest w RW34,
                      ;jednostka przyrostu czasu 10 s
```

Symboliczny zapis

W symbolicznym zapisie operand typu word jest określony dyrektywami *#def*, *#reg*, *#data* oraz *#table*.

```
#def   wejście      XW2
#reg   word   rejestr             ;RW0
#data  word   dane=1,2,3,4        ;DW0, DW2, DW4, DW6
#table word   tab=1,2,3,4         ;TW0
```

Przetypowanie na word

Jeśli w programie potrzebujemy lokalnie przetypować operand (czyli użyć innego typu operandu, niż był zdefiniowany za pomocą wyżej wymienionych dyrektyw), można go lokalnie przetypować na typ word poprzez dodanie prefiksu *word*.

```
#reg   bit    rejestr1           ;R0.0
#reg   byte   rejestr2           ;R1
#reg   long   rejestr3           ;RL2
#table byte   tab=1,2,3,4        ;T0
;
       LD     word rejestr1      ;LD  RW0
```

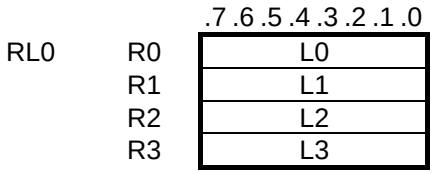
```
LD     word rejestr2       ;LD  RW1
LD     word rejestr3+2     ;LD  RW4
LTB    word tab            ;LTB TW0
```

4.2.4. Operand typu long

Dane formatu long zajmują w pamięci cztery konkretne bajty określone adresem pierwszego z nich. Dane są zapisane tak, że znaczeniowo najniższy bajt ma najniższy adres (konwencja Intel).

Zakres formatu

Dane formatu long osiągają wartości 0 do 4 294 967 295 lub przy użyciu najwyższego bitu jako znaku minus –2 147 483 648 do +2 147 483 647.



Rys.4.4 Zapis danych formatu long w pamięci
L0 - wartość najniższego bajtu
:
L3 - wartość najwyższego bajtu

Zapis operandu typu long

Adres operandu typu long w postaci absolutnej zapisywany jest tak, że za specyfikatorem obszaru jest zapisany znak L jako symbol typu operandu, a za nim wartość liczbowa, która adresuje najniższy bajt longu. Jeśli na przykład rejestr R14 ma wartość $21, rejestr R15 wartość $43, rejestr R16 wartość $65 a rejestr R17 wartość $87, wówczas za pomocą instrukcji

```
LD     RL14
```

zostanie odczytana wartość $87654321. Podobnie są zapisywane pozostałe operandy.

```
LD     XL1
WR     YL0
ADX    SL53
SUX    RL123
LDC    DL25
```

Operandy U oraz T nie umożliwiają dostępu w formacie long.

Symboliczny zapis

W symbolicznym zapisie operand typu long jest określony przez dyrektywy *#def*, *#reg*, *#data* oraz *#table*.

```
#def   wejscie XL4
#reg   long    rejestr          ;RL0
#data  long    dane=1,2,3,4     ;DL0, DL4, DL8, DL12
```

Przetypowanie na long

Jeśli w programie potrzebujemy lokalnie przetypować operand (czyli użyć innego typu operandu, niż był zdefiniowany za pomocą wyżej wymienionych dyrektyw), można go lokalnie przetypować na typ long poprzez dopisanie prefiksu *long*.

```
#reg   bit    rejestr1          ;R0.0
#reg   byte   rejestr2          ;R1
#reg   word   rejestr3          ;RW2
;
       LD     long rejestr1     ;LD  RL0
       LD     long rejestr2     ;LD  RL1
       LD     long rejestr3     ;LD  RL2
```

4.2.5. Operand typu float

Dane formatu float zajmują w pamięci cztery konkretne bajty określone adresem pierwszego z nich. Dane są zapisane tak, że znaczeniowo najniższy bajt ma najniższy adres (konwencja Intel).

Zakres formatu

Dane formatu float zgodnie z IEEE-754 (Institute of Electrical and Electronics Engineers) dla liczb ze zwykłą dokładnością (single precision) osiągają wartości w zakresie ok. $\pm 1{,}175494 \times 10^{-38}$ do $\pm 3{,}402823 \times 10^{38}$ z dokładnością do około 7 cyfr dziesiętnych. Następnie są zdefiniowane trzy wartości określające następujące stany:

$FFFFFFFF - niewłaściwa liczba (NaN - not a number)
$7F800000 - przekroczenie zakresu liczb dodatnich (+INF)

$FF800000 - przekroczenie zakresu liczb ujemnych (–INF)

Struktura formatu float

Dane formatu float są rozdzielone na trzy części. Najwyższy bit oznaczony na rys.4.5 jako **s** określa znak całej liczby. Jeśli s = 0, liczba jest dodatnia, jeśli s = 1, liczba jest ujemna. Następnych 8 bitów oznaczonych jako **e** określają eksponent i niosą informację o rozmiarze liczby. Pozostałe 23 bity oznaczone jako **m** określają mantysę i określają aktualną liczbę (tj. bez lewostronnych zer).

Mantysa ma nieokreślony bit wiodący, co oznacza, że określa liczbę binarną w postaci 1,mmmmmmm. Eksponent z kolei określa ilość binarnych rzędów, o które należy przesunąć przecinek dziesiętny, aby uzyskać wymaganą liczbę. Jeśli przecinek dziesiętny przesuwa się w lewo, będzie eksponent ujemny, jeśli w prawo, będzie eksponent dodatni.

Eksponent nie jest wyrażony w dwójkowym kodzie dopełniającym, ale w kodzie przesuniętego zera, co oznacza, że do rzeczywistej wartości eksponentu dodaje się wartość $7F (127). Zero jest więc zapisane jako $7F, jedynka jako $80, itd. Przy dekodowaniu eksponentu należy odjąć wartość $7F.

Wartość liczby można określić wzorem:

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

Przykład przeliczenia na format float

Liczba 12345 = 1,2345 x 10^4 = $3039
w formacie float: $46 40 E4 00

Wsteczne przeliczenie:

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

Rys.4.5 Zapis danych formatu float w pamięci
s - znak minus (1 bit)
e - eksponent (8 bitów)
m - mantysa (23 bitów)

Zapis operandu typu float

Adres operandu typu float w postaci absolutnej zapisuje się tak, że za specyfikatorem obszaru jest zapisany znak F jako symbol typu operandu a za nim wartość liczbowa, która adresuje najniższy bajt floatu. Sposób jest więc analogiczny do operandu typu long.

```
LD     XF1
WR     YF0
ADF    SF53
SUF    RF123
LD     DF25
```

Operandy U i T nie umożliwiają dostępu w formacie float.

Symboliczny zapis

W symbolicznej postaci operand typu float jest określony dyrektywami *#def*, *#reg*, *#data* oraz *#table*.

```
#def   wejscie XF4
#reg   float   rejestr                ;RF0
#data  float   dane=1.0,2.0,3.0,4.0   ;DF0, DF4, DF8, DF12
```

Przetypowanie na float

Jeśli w programie potrzebujemy lokalnie przetypować operand (czyli użyć innego typu operandu, niż był zdefiniowany za pomocą wyżej wymienionych dyrektyw), można go lokalnie przetypować na typ float poprzez dopisanie prefiksu *float*.

```
#reg   bit    rejestr1           ;R0.0
#reg   byte   rejestr2           ;R1
#reg   word   rejestr3           ;RW2
;
       LD     float rejestr1     ;LD  RF0
       LD     float rejestr2     ;LD  RF1
       LD     float rejestr3     ;LD  RF2
```

Uwaga! Poprzez przetypowanie operandu na float nie nastąpi konwersja zawartości operandu na format float. Do tego należy użyć odpowiednią instrukcję konwersji.

4.3. CEL SKOKU

Etykieta jako operand

W instrukcji skoku i wywołania operandem jest etykieta, na którą nastąpi skok. Instrukcja ma postać np.

```
JMP    L15
```

i przekazuje bieg programu na etykietę L15. W programie etykiety zapisywane są jako instrukcje L z odpowiednim parametrem liczbowym i służą wyłącznie jako cel skoku i wywołania. Przy wykonywaniu programu instrukcja L jest interpretowana jako instrukcja pusta.

Symboliczny zapis

Bardziej efektywne jest używanie symbolicznych etykiet, gdzie zamiast instrukcji L z numerem etykiety zapisuje się symboliczną nazwę, w miejsce której kompilator sam podstawi numer. Program jest wówczas optymalizowany na jak najkrótszą listę adresów etykiet, która stanowi część programu i jest informacją dla PLC przy instrukcjach skoku.

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

Użycie dyrektywy #label

Dyrektywę *#label* używa się tylko wówczas, gdy chcemy którejś z etykiet przyporządkować konkretny numer lub potrzebujemy zapewnić kolejność etykiet dla skoku pośredniego za pomocą instrukcji JMI oraz CAI. Dyrektywa *#label* jest konieczna również w przypadku, gdy użyje się numeru etykiety jako operand (z prefiksem *indx*) lub jako pozycję tabelki wcześniej, niż zostanie etykieta użyta. Dla kompilatora by to wówczas była nieznana nazwa.

Lista etykiet L w programie

Uwaga: Jeśli w programie użyje się etykietę L z najwyższą wartością parametru n, kompilator wytworzy listę adresów etykiet dla wszystkich etykiet od L0 do Ln wraz z tymi, które nie są wykorzystane (mają zerowy adres). Z tego wynika, że jeśli w programie użyje się jedynie etykietę L1023, lista etykiet zajmie zbytecznie całe 2KB pamięci, przy czym adresy wszystkich pozostałych etykiet są zerowe! Dlatego zaleca się numerować etykiety rosnąco od 0 (kompilator xPRO wspiera tę zasadę przy używaniu symbolicznych nazw etykiet).

4.4. PARAMETR INSTRUKCJI

Parametr liczbowy

Niektóre instrukcje wymagają wprowadzenia parametru liczbowego. U niektórych instrukcji parametr jest pasywnie zapisywany i służy jedynie do identyfikacji instrukcji (np. w instrukcji L, NOP, P, E). U innych instrukcji parametr wpływa na sposób wykonania instrukcji, np. u instrukcji ROL, ROR, POP określa ilość kroków (przesunięć).

Znakowy parametr

Instrukcje pracujące z kilkoma akumulatorami (CHG, WAC, LAC) używają znakowego parametru oznaczającego zastosowany akumulator.

Parametr instrukcji zadawany jest więc jako wartość liczbowa albo jako łańcuch jedno lub dwuznakowy. Zakres parametru jest określony przez typ instrukcji.

5. STRUKTURA PAMIĘCI

Funkcja pamięci

Pamięć rejestrów rozumiana jest jako część obszaru pamięciowego PLC, który jest dostępny jak dla odczytu, tak i dla zapisu danych użytkownika. Instrukcje PLC umożliwiają dostęp do dowolnej część pamięci w formatach bit, byte, word lub long. Ta pamięć jest wcześniej rozdzielona na kilka obszarów o różnym znaczeniu. Schematycznie organizacja pamięci podana jest na rys.5.1.

Podział pamięci

Pamięć jest rozdzielona na następujące obszary:

- obrazy sygnałów wejściowych X
- obrazy sygnałów wyjściowych Y
- rejestry systemowe S
- rejestry użytkownika R

Typ jednostek centralnych	E	M	S	D	B
Obrazy sygnałów wejściowych **X**	X0 : X15	X0 : X15	X0 : X127	X0 : X127	X0 : X127
Obrazy sygnałów wyjściowych **Y**	Y0 : Y15	Y0 : Y15	Y0 : Y127	Y0 : Y127	Y0 : Y127
Rejestry systemowe **S**	S0 : S63	S0 : S63	S0 : S63	S0 : S63	S0 : S63
Rejestry użytkownika **R**	R0 : R255	R0 : R255	R0 : R511	R0 : R8191	R0 : R8191

Rys.5.1 Struktura pamięci wraz z zakresem operandów dla poszczególnych typów jednostek centralnych

Dostęp do pamięci

Ogólnie przyjmuje się zasadę, że dostęp systemowego programu do pamięci odbywa się wyłącznie w fazie obrotu cyklu programu. To dotyczy nie tylko odczytu fizycznych wejść (i kopiowania do obszaru X) oraz nastawiania wartości z obszaru Y na fizyczne wyjścia, ale również zmian wartości zmiennych systemowych S. To oznacza, że przez czas cyklu programu użytkownika dane są w pamięci „zamrożone" i są aktualizowane dopiero po najbliższym obrocie cyklu. W ten sposób ograniczona jest możliwość pojawienia się różnych stanów hazardowych w programie użytkownika na skutek asynchronizmu momentów zmian poszczególnych zmiennych. W trakcie cyklu zmieniają się jedynie te zmienne, które wpływają na program użytkownika (bezpośredni zapis do pamięci - WR, WRC, LET, BET, SET, RES), lub wpływ niektórych funkcji (np. nastawianie wskaźników wyników, aktualizacja stanu liczników, układów czasowych, rejestrów przesuwnych, itp.).

Uwaga: Należy pamiętać, że momenty przerwania mogą być asynchroniczne względem cyklu spoczynkowego programu użytkownika, a w wyniku niewłaściwej pracy ze zmiennymi w pamięci użytkownik może wytworzyć wiele hazardowych stanów. Należy więc poświęcić wiele uwagi przyporządkowaniu zmiennych, wytworzeniu reguł dla współpracy pomiędzy procesami cyklu spoczynkowego a procesami przerywającymi. Wyraźne wsparcie pod tym względem posiada narzędzie programowe xPRO.

Podtrzymywanie danych przy zaniku zasilania

Przy zaniku napięcia zasilającego część zawartości pamięci jest podtrzymywana ze źródła zapasowego (tzw. remanentna strefa w rejestrach użytkownika R). Przy ponownym starcie mogą być te podtrzymywane wartości użyte do kontynuacji sterowania - zależy to od sposobu rozbiegu oraz od innych okoliczności (poprawność pamięci, nie zmieniona zawartość programu użytkownika, itp.). Przy wyborze konfiguracji użytkownik może wybrać rozmiar strefy remanentnej.

5.1. OBRAZY WEJŚĆ X

Obszar X

Przed każdym rozpoczęciem cyklu programu jednostka centralna zapewnia aktualizację tego obszaru pamięci z wejściowych jednostek peryferyjnych na podstawie tabelki deklaracyjnej zadanej w programie użytkownika, która opisuje przyporządkowanie obrazów wejść X do fizycznych adresów poszczególnych jednostek (w środowisku xPRO za pomocą dyrektywy *#unit*).

W przypadku niedostatku pamięci w obszarze X, można bez ograniczenia użyć do tego celu obszar rejestrów R.

5.2. OBRAZY WYJŚĆ Y

Obszar Y

Po każdym zakończeniu cyklu programu jednostka centralna zapewnia przesunięcie wyników z tego obszaru pamięci do wyjść jednostek peryferyjnych na podstawie tabelki deklaracyjnej zadanej w programie użytkownika, która opisuje przyporządkowanie obrazów wyjść Y do fizycznych adresów poszczególnych jednostek (w środowisku xPRO za pomocą dyrektywy *#unit*).

W przypadku niedostatku pamięci w obszarze Y, można bez ograniczenia użyć do tego celu obszaru rejestrów R.

5.3. SYSTEMOWE REJESTRY S

Obszar S

Ten obszar pamięci jest przeznaczony do specjalnego wykorzystania przez program systemowy sterownika i nie zaleca się go używać do innych celów. Niektóre bity i bajty są regularnie nastawiane podczas obrotu cyklu za pomocą programu systemowego i powinny być używane jedynie do odczytu. Niektóre bity poprzez swoje nastawienia modyfikują działanie programu systemowego.

Tab.5.1 Wykaz rejestrów systemowych

Rejestry	Zastosowanie	Typ CPU
S0	wskaźniki wyników arytmetycz. operacji	E M S D B
S1	wskaźniki wyników logicznych operacji	E M S D B
S2	wskaźniki stanu systemu	E M S D B
S3	czas poprzedniego cyklu w 10 ms	E M S D B
S4	licznik cykli	E M S D B
S5	licznik dziesiątek milisekund system. czasu	M S D B
S6	licznik sekund systemowego czasu	M S D B
S7	licznik minut systemowego czasu	M S D B
S8	licznik godzin systemowego czasu	M S D B
S9	licznik dni w tygodniu	M S D B
S10	licznik dni w miesiącu	M S D B
S11	licznik miesięcy	M S D B
S12	licznik lat	M S D B
S13	jednostki czasowe	M S D B
S14-S15	licznik w 100ms	M S D B
S16-S17	licznik w 1s	M S D B
S18-S19	licznik w 10s	M S D B
S20	dodatnie zbocza jednostek czasu z S13	M S D B
S21	ujemne zbocza jednostek czasu z S13	M S D B
S22-S23	czas poprzedniego cyklu w 100 µs	S D B
S24-S29	maski sterujące procesów	M S D B
S30-S33	rezerwa	
S34	wewnętrzny kod błędu	E M S D B
S35	wskaźniki stanu hardwaru	E M S D B
S36-S37	rezerwa	
S38	ilość edycji programu użytkownika	E M S D B
S39	ilość zmian zawartości	E M S D B
S40-S41	kod wersji systemowego programu PLC	E M S D B
S42-S43	rezerwa	
S44-S45	typ kompilatora	E M S D B
S46-S47	rezerwa	
S48-S51	kompletny kod błędu	S D B
S52-S63	rezerwa	

Znaczenie rejestrów systemowych:

S0 wskaźniki porównania, przeniesienia i inne arytmetyczne wskaźniki

S0 - wskaźniki wyników operacji arytmetycznych

Wpływają na nie jedynie niektóre instrukcje arytmetyczne, instrukcja porównania, instrukcja liczników i układów czasowych, czyli ADD, BCD, SUB, DIV, DID, DIF, INR, DCR, BCD, EQ, LT, GT, CMP, CML, CMF, ROL, ROR, CTU, CTD, CNT, TON, TOF, RTO i IMP. Pozostałe instrukcje nie zmieniają ich.

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 (=) - zgodność (równość) obu operandów
(ZR) - zerowość wyniku (w zależności od całego słowa wyniku)
- dzielenie zerem dla instrukcji DIV, DID, DIF

S0.1 (<) - pierwszy operand < drugi operand
(CO) - wyjściowe przeniesienie (carry out), w skutek operacji nastąpiło przeniesienie do wyższego rzędu (słowa)

S0.2 (≤) - pierwszy operand ≤ drugi operand
- Suma logiczna bitów S0.0 OR S0.1

S0.3 (CI) - wejściowe przeniesienie (carry in)
- Służą do kaskadowania operacji arytmetycznych ADD, SUB, INR, DCR, EQ, LT, GT. Jeśli chcemy w którejś z tych operacji respektować przeniesienie z niższego rzędu, należy przed tą operacją nastawić CI na wartość przeniesienia. Przy każdym nastawieniu S0 (po wyżej wymienionych instrukcjach) bit CI zostanie wyzerowany, czyli następna arytmetyczna instrukcja nie uwzględnia przeniesienia z dołu. Arytmetyczne operacje stosują kaskadę jedynie wówczas, gdy przed ich użyciem użytkownik zapisze wartość transmisji do bitu CI.

S0.4 do S0.6 (D5) - po instrukcji BCD zawierają najwyższą cyfrę wyniku (maksymalna wartość wynosi 6), po innych instrukcjach bity są zerowane

S0.4 (OV) - przekroczenie maksymalnego zakresu układu czasowego (w tym cyklu lub kiedykolwiek wcześniej przy jednoczesnej aktywacji ukła du czasowego)

S0.5 (OC) - przekroczenie maksymalnego zakresu układu czasowego w bieżącym cyklu

S0.7 - rezerwa

Szczegóły są podane w opisie instrukcji, które wpływają na rejestr S0 (rozdz.8.).

S1 wskaźnik ważności operacji, wskaźniki instrukcji FLG, STE

S1 - wskaźniki wyników operacji logicznych

Tabelkowe instrukcje nastawiają bit S1.0 w znaczeniu:

S1.0 = 1 - pozycja w zakresie tabelki, pozycja znaleziona
S1.0 = 0 - pozycja poza zakresem tabelki, pozycja nie znaleziona

Arytmetyczne instrukcje dla pracy zmiennoprzecinkowej, blokowe operacje oraz operacje ze strukturalnymi tabelami nastawiają bit S1.0 w znaczeniu:

S1.0 = 1 - wejściowe parametry w porządku, wynik jest aktualny
S1.0 = 0 - wejściowe parametry poza zakresem, wynik jest nieaktualny

Instrukcja FLG nastawia rejestr S1 w zależności od zawartości szczytu akumulatora.

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

S1.0 do S1.3 (N)- dwójkowa liczba (N4, N3, N2, N1, N0) osiągająca wartości 00000 do 10000, które mają znaczenie ilości jedynkowych bitów szczytu akumulatora (bit N4 jest powtórzony we wszystkich bitach szczytu akumulatora)

S1.4 (ANL) - iloczyn logiczny bitów dolnej części akumulatora A0L
S1.5 (ANH) - iloczyn logiczny bitów górnej części akumulatora A0H
S1.6 (ORL) - logiczna suma bitów dolnej części akumulatora A0L
S1.7 (ORH) - logiczna suma bitów górnej części akumulatora A0H

Instrukcja STE nastawia bity S1.0 i S1.1 w znaczeniu:

S1.0 = 1 - zmienił się stan stepperu
S1.0 = 0 - stan stepperu bez zmian
S1.1 = 1 - obrót (przeniesienie stepperu - stan się zmienia z 15 na 0)
S1.1 = 0 - pozostałe przypadki

Szczegóły są podane w opisie instrukcji, które wpływają na rejestr S1 (rozdz.8.).

S2 wskaźniki stanu systemu

S2 - wskaźniki stanu systemu

Nastawiany przez program systemowy w zależności od jego stanu podczas obrotu cyklu.

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

S2.0 (SP) - stan wejścia służbowego SP - zewnętrzne uruchomienie PLC
S2.1 (MS) - stan wejścia służbowego MS - zewnętrzne blokowanie wyjść
S2.2 (RUN) - 1 - start cyklu, program jest wykonywany (tryb RUN)
0 - stop cyklu, program nie jest wykonywany, PLC znajduje się w obrocie cyklu (tryb HALT)
S2.3 (HOT) - 1 - pierwsze przejście przez cykl po ciepłym restarcie
S2.4 (RST) - 1 - pierwsze przejście przez cykl po zimnym restarcie
S2.5 (ON) - 1 - wyjścia aktywne
0 - wyjścia zablokowane
S2.6 (NRS) - 1 - pierwsze przejście przez cykl bez restartu
S2.7 (MEZ) - 1 - przekroczony pierwszy próg czasu cyklu (ostrzeżenie)

Bity S2.3, S2.4 i S2.6 są wygodne dla inicjalizacji zmiennych.

Uwaga: Rejestr S2 jest przeznaczony jedynie do odczytu.

S3 czas poprzedniego cyklu

S3 - czas poprzedniego cyklu w 10 ms

Dwójkowa wartość z jednostką 10 ms (zakres 0 - 2,55 s) podaje czas trwania poprzedniego cyklu programu użytkownika.

S4 licznik cykli

S4 - licznik cykli

Dwójkowa wartość, która podczas restartu systemu jest zerowana a przy każdym obrocie cyklu zwiększy się o jeden. Umożliwia bardziej skomplikowane fazowanie algorytmu sterującego na poszczególne pętle.

S5 - S12 systemowy czas i data

S5 do S12 - systemowy czas i data

(jednostki centralne typu E nie posiadają tych informacji)

Wartości binarne, które określają czas w jednostkach czasu. Umożliwia użycie w programie danych o czasie i dacie bez skomplikowanych obliczeń. Czas jest odczytywany z układu realnego czasu, i jest odświeżany w każdym obrocie cyklu. Przy zaniku zasilania czas jest nadal odmierzany dzięki baterii podtrzymującej.

S5 - licznik dziesiątek milisekund (0 - 990 ms)
S6 - licznik sekund (0 - 59 s)
S7 - licznik minut (0 - 59 min)
S8 - licznik godzin (0 - 23 hod)
S9 - licznik dni w tygodniu (1 - 7)
S10 - licznik dni w miesiącu (od 1 do ostatniego dnia w aktywnym miesiącu, rok przestępny jest respektowany)
S11 - licznik miesięcy (1 do 12)
S12 - licznik lat - ostatnie dwie cyfry roku (0 - 99)

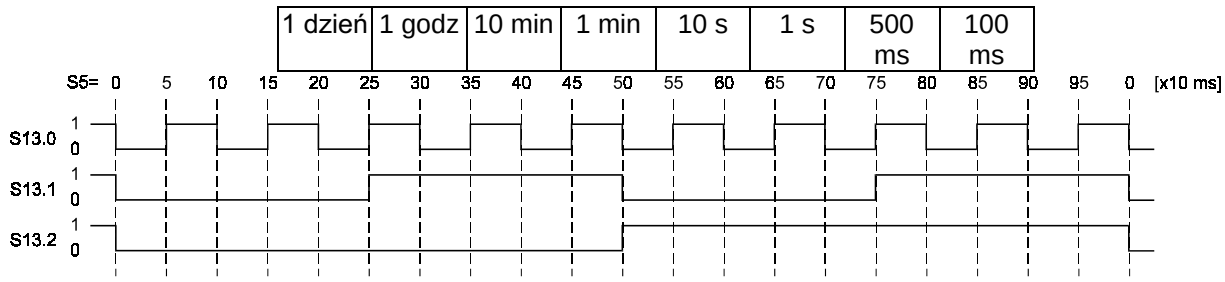
S13 czasowe jednostki

S13 - czasowe jednostki

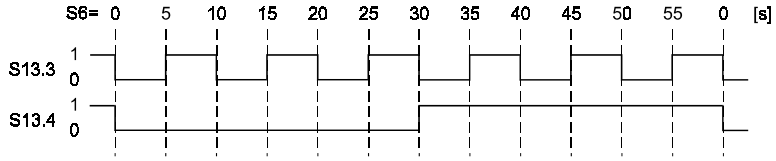
(nie posiadają ich jednostki centralne typu E)

Grupa zmiennych bitowych, które zmieniają swój stan co określony czas. Przebieg tych czasowych sygnałów ma współczynnik wypełnienia w przybliżeniu równy 0,5 i są zależne od stanu S5 do S8 (rys.5.2 do rys.5.5). Mogą być wykorzystane jako źródło czasowych impulsów dla liczników, do realizacji funkcji czasowych (miganie, układ D, T, JK). Zmienne czasowe mogą być użyte pod warunkiem, że czas cyklu programu użytkownika jest krótszy, niż połowa użytej jednostki czasu. Poszczególne bity mają znaczenie:

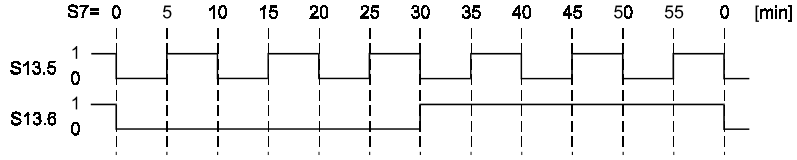
S13.7	S13.6	S13.5	S13.4	S13.3	S13.2	S13.1	S13.0



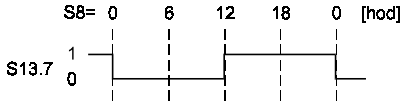
Rys.5.2 Stan zmiennych bitowych S13 w odniesieniu do licznika dziesiątek milisekund S5



Rys.5.3 Stan zmiennych bitowych S13 w odniesieniu do licznika sekund S6



Rys.5.4 Stan zmiennych bitowych S13 w odniesieniu do licznika minut S7



Rys.5.5 Stan zmiennych bitowych S13 w odniesieniu do licznika godzin S8

SW14, SW16, SW18 liczniki jednostek czasu

S14,S15 - licznik 100 ms
S16,S17 - licznik 1 s
S18,S19 - licznik 10 s

(nie posiadają ich jednostki centralne typu E)

Każdy word SW14, SW16, SW18 zawiera dwójkową wartość w zakresie do 65 535 jednostek czasu. Zmiany tych wartości przebiegają synchronicznie ze zmianami S5 do S13. Głównym zastosowaniem jest zastąpienie liczników, zwłaszcza w przypadkach realizacji ciągów przedziałów czasowych (sekwencyjne i czasowe sterowanie). Poszczególne bity mogą być wykorzystane jako źródła jednostek czasu. W zależności od wymagania może być wykorzystany tylko dolny lub górny bajt.

S20, S21 zbocza jednostek czasu

S20 - dodatnie zbocza jednostek czasu z S13
S21 - ujemne zbocza jednostek czasu z S13

(nie posiadają ich jednostki centralne typu E)

Na pozycjach o znaczeniu zgodnym z S13 są nastawiane sygnały w momencie zmiany danego bitu w rejestrze S13 z log.0 na log.1 (S20) lub z log.1 na log.0 (S21). Zmiany są wykrywane na podstawie porównania ze stanem z poprzedniego cyklu.

SW22 czas poprzedniego cyklu

S22, S23 - czas poprzedniego cyklu w 100 µs

(nie posiadają ich jednostki centralne typu E i M)

Dwójkowa wartość z jednostką 100 µs (zakres 0 - 6,5535 s) podaje czas trwania poprzedniego cyklu programu użytkownika. Chodzi o dokładniejszą wartość rejestru S3.

S24 - S29 maski sterujące procesów

S24 do S29 - sterujące maski procesów

(nie posiadają ich jednostki centralne typu E)

Maski sterujące do sterowania i wskazywania aktywnych procesów P1 do P48 (rozdz.10.). Znaczenie bitów:

.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

Jedynki odpowiadają aktywnym procesom, zera pasywnym procesom.
Bity S24.0 do S25.0 nastawia układ planowania procesów podczas obrotu cyklu i decyduje w ten sposób o aktywowaniu procesów P1 do P9. Jeśli te bity zostaną zmienione przez użytkownika, zaplanowany proces jest mimo to wykonany.
Bity S25.1 do S28.7 są dostępne dla użytkownika do sterowania procesami P10 do P40. Nastawianie bitu na log.1 ma w skutku zaplanowanie odpowiedniego procesu do uruchomienia podczas następnego cyklu. Po restarcie bity te są zawsze wyzerowane.
Bity S29.0 do S29.7 umożliwiają użytkownikowi zakazanie lub zezwolenie na wykonywanie procesów przerywających P41 do P48. Te bity nie są implementowane w jednostkach centralnych CPM-1M TECOMAT NS950, czyli nie można tam zakazać wykonywania przerywającego procesu. Po restarcie są bity odpowiadające przerywającym procesom użytym w programie nastawione na log.1.

S30 do S33 - rezerwa

S34 kod błędu

S34 - wewnętrzny kod błędu

Pierwszy bajt kodu ostatniego błędu.

S34.7 = 1 (kod ≥ 128) - błąd krytyczny, program użytkownika zostanie zatrzymany, PLC przejdzie do trybu HALT i zablokuje wyjścia

S34.7 = 0 (kod < 128) - pozostałe błędy nie mające poważnego wpływu na sterowanie, program użytkownika jest nadal wykonywany. Błędy te można obsłużyć wykorzystując ten rejestr systemowy

S34 = 0 - stan bez błędu
7 - błąd podczas kontroli strefy remanentnej
8 - przekroczenie pierwszego progu kontroli czasu cyklu
9 - błędny systemowy czas układu RTC
16 - dzielenie przez zero
17 - początkowy indeks dla instrukcji WMS jest poza tabelką T
18 - początkowy indeks dla instrukcji LMS jest poza tabelką T
19 - instrukcja tabelkowa w obszarze rejestrów przekroczyła jego zakres
20 - blok danych źródłowych był zdefiniowany poza zakresem pamięci rejestrów, danych lub tabelki
21 - docelowy blok danych był zdefiniowany poza zakresem pamięci rejestrów lub tabelki
128 - błąd programu użytkownika
129 - błąd w systemie peryferyjnym
130 - błąd komunikacji z modułami rozszerzeniowymi

S35 wskaźniki stanu hardwaru

S35 - wskaźniki stanu hardwaru

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

S35.0 (BAT) - stan baterii podtrzymującej RAM i RTC
1 - napięcie baterii podtrzymującej jest niższe niż 2,1 V
0 - bateria podtrzymująca jest w porządku

S35.1 (ERO) - 1 - błąd wyjścia binarnego (zwarcie, przerwanie układu) (posiadają TC500, TC600)

S36, S37 - rezerwa

S38 numer edycji

S38 - numer edycji programu użytkownika

W sekwencji załączeniowej kopiowany jest z konfiguracyjnej wartości stałej programu użytkownika. Wartość tej stałej można zadać z poziomu kompilatora xPRO.

S39 liczba zmian zawartości

S39 - liczba zmian zawartości

W sekwencji załączeniowej kopiowana jest z konfiguracyjnej wartości stałej programu użytkownika. Wartość tej stałej można zadać z poziomu kompilatora xPRO. Przeznaczona jest przede wszystkim do rozróżnienia wersji programu użytkownika.

SW40 wersja sw CPU

S40, S41 - kod wersji programu systemowego jednostki centralnej

Na przykład dla wersji 4.1 będzie S40 = 4 a S41 = 1.

S42, S43 - rezerwa

SW44 typ kompilatora

S44, S45 - typ kompilatora

Znak kompilatora, za pomocą którego był wytworzony program użytkownika.

S46, S47 - rezerwa

SL48 całkowity kod błędu

S48 do S51 - całkowity kod błędu

(nie posiadają go jednostki centralne typu E i M)

Całkowity kod ostatniego błędu, przeznaczony do łatwego odczytu ostatnio powstałego błędu przez system nadrzędny. Kod błędu jest zapisany w formacie long, tj. pierwszy bajt jest zapisany w rejestrze S51 (zgodny z zawartością rejestru S34), ostatni bajt jest zapisany w rejestrze S48. Wykaz kodów błędów jest podany w podręczniku odpowiedniego typu PLC.

S52 do S63 - rezerwa

5.4. REJESTRY UŻYTKOWNIKA R

Obszar R

Pamięciowy obszar przeznaczony dla zmiennych programu użytkownika, do realizacji liczników, układów czasowych, rejestrów przesuwnych, sterowników krokowych oraz dynamicznych tabel.

W sekwencji załączeniowej programu systemowego, po zimnym restarcie są wszystkie rejestry R wyzerowane. Po ciepłym restarcie jest zachowana remanentna część rejestrów R, pozostałe są wyzerowane.

Remanentna strefa

Remanentna strefa jest obszarem rejestrów R, których zawartość pozostaje zachowana podczas wyłączonego zasilania PLC. Rozmiar tego obszaru definiuje się za pomocą kompilatora programu, początek jest zawsze na rejestrze R0.

Uwaga! W strefie remanentnej nie powinno się umieszczać obrazów wejść i wyjść jednostek peryferyjnych. W przypadku inteligentnych jednostek peryferyjnych, po restarcie może dojść do naruszenia rozpoczęcia wymiany danych.

Tab.5.2 Maks. rozmiar strefy remanen. dla poszczególnych typów jednostek centralnych

Typ CPU	Maks. długość strefy remanentnej	Podtrzymywane rejestry
B	4096	R0 - R4095
D	512	R0 - R511
S	256	R0 - R255
M	256	R0 - R255
E	256	R0 - R255

Od użytkownika zależy, jak użyje rejestry R, które z nich wykorzysta dla swych zmiennych roboczych lub tabelek, a które przeznaczy do realizacji bloków funkcyjnych.

Użycie rejestrów R dla liczników i układów czasowych

System nie określa, ile może być użytych liczników, ile układów czasowych itp. Jedynym ograniczeniem jest całkowita ilość rejestrów. O tym, czy obiekt będzie remanentny czy też nie, również decyduje użytkownik poprzez wybór obszaru, w którym obiekt umieści. Jeśli program użytkownika używa czasowo ograniczonych obiektów, które się w czasie wykluczają, wówczas mogą być realizowane na zgodnych rejestrach R. Rejestry przeznaczone dla liczników lub rejestrów przesuwnych mogą być obsługiwane poprzez różne instrukcje liczników i przesunięć, nie powodując błędu (inaczej jest w przypadku układów czasowych). Rejestry R przyporządkowane któremuś z obiektów są dostępne dla pozostałych instrukcji, czyli np. wartości liczników lub układy czasowe można porównywać z kilkoma wartościami, ewentualnie je zmieniać.

6. FIZYCZNE ADRESY JEDNOSTEK PERYFERYJNYCH

Fizyczny adres

Adres fizyczny jednostki peryferyjnej oznacza rzeczywisty adres, pod którym jednostka jest umieszczona na szynie. Za pomocą dyrektywy *#unit* (dla zwykłej obsługi jednostek peryferyjnych) do fizycznych adresów jednostki jest przyporządkowany obraz w pamięci (obszar X, Y, R). Aktualizacja danych jest dokonywana zawsze w czasie obrotu cyklu.

Operand U

Istnieją jednak przypadki, kiedy ze względu na czas konieczne jest odczytanie momentalnego stanu z jednostki, lub bezzwłoczne zapisanie wartości do jednostki. Do tego służą wówczas fizyczne adresy oznaczane operandem U. Operand U zapewnia więc alternatywę dla obszarów X i Y w pamięci, która umożliwia bezpośredni styk z jednostkami peryferyjnymi w danym momencie bez czekania na obrót cyklu.

Operand U umożliwia dostęp w formacie byte i word za pomocą instrukcji LD i WR. Stosuje się go jedynie dla zabezpieczenia czasowo krytycznych reakcji. Zbyteczne wykorzystywanie powoduje osłabienie mocy programu, ponieważ bezpośredni dostęp do jednostek peryferyjnych wymaga więcej czasu niż operacje w pamięci.

Konkretne fizyczne adresy różnią się w zależności od typu PLC, co jest dane jego konstrukcją. Szczegóły są podane w suplemencie na końcu tego podręcznika.

Ograniczenie użycia operandu U

Należy pamiętać, że praca z fizycznymi adresami narusza zasadę niezmienności wejściowych i wyjściowych wartości w trakcie cyklu i zwiększa ryzyko hazardów. Użycie operandów U powinno być ograniczone wyłącznie dla przypadków, kiedy należy zapewnić natychmiastową odpowiedź, np. podczas obsługi sytuacji awaryjnych, w procesach obsługi przerwania itp. Należy poświęcić uwagę ważnym ostrzeżeniom podanym w odpowiednim dodatku poświęconym tej problematyce.

Należy sobie uświadomić, że praca z adresami fizycznymi pogwałca zasadę niezmienności danych wejściowych i wyjściowych w trakcie cyklu i zwiększa ryzyko hazardu. Użycie operandu U należy ograniczyć wyłącznie do przypadków, gdy istnieje potrzeba zapewnienia natychmiastowej reakcji, np. podczas opanowywania sytuacji awaryjnych, w procesach obsługi przerwań itp.

Ważne ostrzeżenia:

Adres fizyczny nie ma automatycznego powiązania z pamięcią operacyjną

Podczas zapisu do adresu fizycznego lub odczytu z adresu fizycznego jednostki peryferyjnej **nie dojdzie** do odpowiedniej zmiany wartości w obrazie tej jednostki w pamięci operacyjnej!

W przypadku odczytu fizycznego dojdzie podczas obrotu cyklu do poprawienia wartości w obrazie i zazwyczaj nie jest to żadną przeszkodą (należy się jednak z tym liczyć).

W przypadku zapisu fizycznego jest jednak konieczne zabezpieczenie poprawy wartości w pamięci operacyjnej za pomocą programu użytkownika, w innym razie dojdzie podczas obrotu cyklu do zapisu wartości pierwotnej z pamięci operacyjnej do jednostki.

Jeżeli użycie zapisu fizycznego do jednostki jest niezbędne, lepiej jest wyłączyć podczas kompilacji programu użytkownika w kompilatorze (pozycja dyrektywy *#unit*) obsługę wyjść jednostki w konfiguracji programowej i obsługiwać wyjścia wyłącznie za pomocą zapisów fizycznych używając operandu U.

6.1. ADRESY FIZYCZNE JEDNOSTEK PERYFERYJNYCH W PLC TECOMAT NS950

Adresy fizyczne wyłącznie w module podstawowym

Adresy fizyczne jednostek peryferyjnych są dostępne tylko w module podstawowym (moduł 0 obsługiwany jest bezpośrednio przez jednostkę centralną). Jednostki peryferyjne w modułach rozszerzających są dostępne tylko za pośrednictwem swych obrazów w pamięci operacyjnej.

Struktura adresu fizycznego

Adres fizyczny jednostki peryferyjnej ma następującą strukturę:

górny byte adresu								dolny byte adresu							
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

A15 - A12 - typ jednostki (wyznaczone na stałe)
0xxx (0-7)- jednostki systemowe oraz wirtualne
(specjalne przypadki opisane w odpowiednich podręcznikach)
1000 (8) - 8 wejść lub wyjść binarnych
(jednostki OR-14, OS-28, OS-32, OS-34, XH-04)
1001 (9) - 16 wejść lub wyjść binarnych
(jednostki IB-36 do IB-47, IB-50, OR-15, OS-29, OS-30, OS-31, OS-33, OS-35)
1010 (A) - 32 wejść lub wyjść binarnych
(jednostki IB-48, IB-49, OS-26, OS-27, UX-52)
1011 (B) - rezerwa
1100 (C) - jednostki specjalne bez tablicy inicjalizacyjnej
(jednostka IC-04)
1101 (D) - jednostki analogowe bez własnego procesora
(jednostki IT-04, IT-12, IT-15, OT-04)
1110 (E) - rezerwa
1111 (F) - jednostki z własnym procesorem
(jednostki IT-06, OT-05, GT-40, IC-12 do IC-15, SC-11, CD-01 do CD-04, UP-01, UP-02)

A11 - A8 - adres jednostki w module (do wyboru za pomocą łączy dostępnych z boku skrzynki jednostki peryferyjnej)

A7 - 0 - adres wejściowy
1 - adres wyjściowy
jednostki z własnym procesorem (typ F) nie mają adresów wejściowych i wyjściowych odróżnionych tym bitem, ich obszary wejściowe i wyjściowe są przyporządkowywane dynamicznie w zależności od żądanej wielkości.

A6 - A0 - numer bytu w ramach jednostki

Przykłady użycia adresu fizycznego

```
LD    U$8100    ;bezpośredni odczyt stanu ośmiu wejść jednostki XH-04
                ;z adresem 1 w module
LD    UW$9200   ;bezpośredni odczyt stanu szesnastu wejść jednostki
                ;typów IB-36 do IB-47, IB-50 z adresem 2 w module
WR    U$A083    ;bezpośredni zapis wartości do ośmiu wyjść bytu trzeciego
                ;jednostki typów OS-26, OS-27 z adresem 0 w module
WR    UW$9380   ;bezpośredni zapis wartości do szesnastu wyjść jednostki
                ;typów OR-15, OS-29, OS-30, OS-31, OS-33, OS-35
                ;z adresem 3 w module
```

Konkretne adresy zajmowane przez jednostki peryferyjne przedstawione są w podręcznikach danych jednostek.

Bardziej skomplikowane urządzenia peryferyjne nie muszą umożliwiać obsługę za pomocą adresów fizycznych

Uwaga! Jednostki typów C, D, E, F zazwyczaj wymagają zdefiniowanego sposobu obsługi. Dlatego w **żadnym wypadku** nie wolno używać dostępu do adresu fizycznego tych jednostek bez uprzedniego dokładnego zapoznania się z ich funkcjami!

6.2. ADRESY FIZYCZNE JEDNOSTEK PERYFERYJNYCH W PLC TECOMAT TC400, TC500, TC600

Struktura adresu fizycznego

Adres fizyczny wejść i wyjść cyfrowych i analogowych ma następującą strukturę:

górny byte adresu								dolny byte adresu							
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

A15 - A12 - typ jednostki (wyznaczone na stałe)
0xxx (0-7) - rezerwa
1000 (8) - płyta z maks. 8 wejściami lub wyjściami cyfrowymi
1001 (9) - płyta z maks. 16 wejściami lub wyjściami cyfrowymi
1010 (A) - rezerwa
1011 (B) - rezerwa
1100 (C) - rezerwa
1101 (D) - płyta z wejściami i wyjściami analogowymi
1110 (E) - rezerwa
1111 (F) - rezerwa

A11 - A8 - zawsze 0

A7 - 0 - adres wejściowy

1 - adres wyjściowy

A6 - A0 - numer bytu w ramach jednostki

Przykłady zastosowań adresu fizycznego

```
LD      UW$9000         ;bezpośredni odczyt stanu wejść
WR      UW$9080         ;bezpośredni zapis wartości do wyjść
```

Konkretne adresy zajmowane przez jednostki peryferyjne przedstawione są w podręcznikach: Wyposażenie techniczne sterowników programowalnych TECOMAT TC400 TXV 138 12.01 Wyposażenie techniczne sterowników programowalnych TECOMAT TC500 TXV 138 07.01 oraz Wyposażenie techniczne sterowników programowalnych TECOMAT TC600 TXV 138 08.05.

7. POZOSTAŁE OBSZARY ADRESOWE

7.1. DANE TYPU D

Dane typu D mają znaczenie stałych parametrycznych

Dane typu D mają znaczenie stałych w programie użytkownika. Stanowią część programu użytkownika i są dla niego dostępne jedynie do odczytu. Mogą być zadawane i zmieniane jedynie w ramach edycji programu użytkownika.

Mogą być z wygodą użyte jako parametry, które modyfikują program użytkownika. Dla niektórych rodzajów algorytmów sterujących może być na przykład wytworzony i dostrojony jeden program użytkownika, który jest przed konkretnym zastosowaniem przystosowany do rzeczywistych warunków poprzez zadane odpowiednich parametrów w strefie D. Podobnie można program użytkownika dostosowywać do zmieniających się wymagań zleceniodawcy, zmian technologii, zmieniającego się asortymentu produktów itp.

Użycie danych typu D jest również wygodne w przypadku, kiedy różne miejsca programu używają identycznej stałej liczbowej lub logicznej. Jeśli za każdym razem wartość ta będzie zadana jako bezpośredni operand, wówczas w przypadku potrzeby zmienienia jej należy odszukać i poprawić wszystkie miejsca w programie, gdzie się pojawia. Wygodniej jest zadać tę wartość raz w obszarze danych typu D. Przy jakiekolwiek zmianie wystarczy jedynie zmienić tę jedną wartość bez potrzeby ingerencji w algorytm.

W danych typu D mogą być również zapisane ciągi wartości, na przykład tabelki lub wzory dla nastawiania pamięci i wyjść w kluczowych sytuacjach.

7.2. TABELKI TYPU T

Tabelki typu T są dostępne za pomocą specjalnych instrukcji

Tabelki T są tak samo jak dane D częścią programu użytkownika i użycie ich może być podobne. W odróżnieniu od danych D, które są dostępne przeważnie za pomocą instrukcji do odczytu, tabelki T są dostępne jedynie za pomocą specjalnych instrukcji, które odwołują się do adresowego obszaru T (tabelkowe instrukcje, instrukcje przesunięć blokowych). Dane D mogą być niestrukturalne lub mogą mieć dowolnie skomplikowaną strukturę, można bezpośrednio odczytać ich dowolne miejsce. Tabelki T z kolei mają określoną strukturę - jest to zawsze ciąg wartości identycznego formatu (bit, byte lub word) z dodatkową informacją o długości tego ciągu. Każdej pozycji (wartości z tego ciągu) jest przyporządkowana liczba porządkowa - indeks. Najniższa pozycja ma zerowy indeks, indeks ostatniej pozycji nazywany jest granicą (ilość pozycji = granica + 1). Tabelki T mogą więc być przetworzone w formacie bit, byte lub word (rys.7.1, rys.7.2, rys.7.3). Te formaty są bezpośrednio określone typem użytej instrukcji, nie zależy więc od tabelki. Fizycznie tabelki T są zapisane jako ciąg wartości z informacją o jej długości w bajtach. Dlatego przy bitowym dostępie do tabelki są zawsze ostatnie niewykorzystane bity dopełnione zerami na cały bajt.

Format tabel T jest zmienny i jest bezpośrednio określony typem użytej instrukcji

Na tabelach typu T można wykonywać następujące operacje:

- wybór pozycji odpowiadającej zadanemu indeksowi (LTB)
- zapis na pozycji określonej indeksem (WTB)
- odnalezienie indeksu dla zadanej wartości pozycji (FTB)
- odnalezienie indeksu dla wybranej części pozycji (FTM)
- odnalezienie indeksu grupy - przyporządkowanie zadanej wartości do jednej z grup (przedziału), które są określone za pomocą tabelki progów uporządkowanych rosnąco (FTS)
- blokowe przesunięcia danych pomiędzy tabelką a pamięcią rejestrów (SRC, MOV, MTN, MNT)
- sekwencyjny odczyt z tabelki (LMS)
- sekwencyjny zapis do tabelki (WMS)
- praca ze strukturalnymi tabelami (LDS, WRS, FIT, FNT)

Aparat tabelkowych instrukcji umożliwia w prosty sposób realizowanie nawet bardzo skomplikowanych funkcji, przy czym rozwiązanie bywa bardziej oszczędne i szybsze w przeciwieństwie do tradycyjnego (niekiedy znacznie), zawsze jest jednak bardziej elastyczne i łatwiejsze do adaptacji. Ostateczny program jest czytelny i przejrzysty.

Ograniczenie ilości i rozmiaru tabel

Centralne jednostki typu E używają tabelki T wyłącznie do inicjalizacji jednostek peryferyjnych.

Centralne jednostki typu M mogą mieć maksymalnie 256 tabel o maksymalnej długości 255 bajtów. Z tego wypływa, że za pomocą tabelkowych instrukcji obsługiwanych jest maksymalnie 127 pozycji typu word (254 bajtów), 255 pozycji typu byte i 256 pozycji typu bit (32 bajtów).

Centralne jednostki typu S mogą mieć maksymalnie 256 tabel. Długości tabel nie są ograniczone.

U centralnych jednostek typu B i D czynnikiem ograniczającym jest jedynie rozmiar pamięci programu użytkownika.

Przykłady tabel

indeks =	7 ↓	6 ↓	5 ↓	4 ↓	3 ↓	2 ↓	1 ↓	0 ↓		
	1	1	1	1	1	0	0	1		byte 0
15 →	1	0	1	0	1	1	1	0	← 8	byte 1
23 →	1	1	0	0	0	1	1	1	← 16	byte 2
31 →	0	1	1	1	0	0	0	0	← 24	byte 3
	...	...	...	...	...	...	...	...		. . .
	0	0	0	0	1	0	0	0		byte 7
					↑	↑	↑	↑		
bitová granica = 55					51	50	49	48		
	.7	.6	.5	.4	.3	.2	.1	.0	nr bitu v bajcie	

Rys.7.1 Przykład tabelki w formacie bit (cztery ostatnie bity są dopełnione 0 na cały bajt)

indeks = 0	156	byte 0
1	255	byte 1
2	147	byte 2
3	64	byte 3
:	...	. . .
21	0	byte 21
granica = 22	235	byte 22

Rys.7.2 Przykład tabelki w formacie byte

	górny bajt pozycji	dolny bajt pozycji	
indeks = 0	$00	$55	byte nr 1 i 0
1	$12	$FF	byte nr 3 i 2
2	$01	$47	byte nr 5 i 4
3	$15	$40	byte nr 7 i 6
:		...	. . .
14	$00	$00	byte nr 29 i 28
granica = 15	$00	$35	byte nr 31 i 30

Rys.7.3 Przykład tabelki w formacie word

7.3. DODATKOWA PAMIĘĆ DANYCH DATABOX

Pamięć DataBox

DataBox jest to dodatkowa pamięć przeznaczona do pracy z większą ilością danych, np. archiwizacja danych o sterowanym procesie w dłuższym przedziale czasu, itp. W centralnych jednostkach jest realizowana w formie dodatkowego piggybacku (TC500, TC600, NS950 CPM-2S i CPM-1D), lub stanowi standardowy element (NS950 CPM-1B).

Piggyback DataBox jest pamięcią CMOS RAM rozmiaru 128, 256 lub 512 KB, która jest podtrzymywana baterią z jednostki centralnej. Dane można do pamięci zapisywać ew. odczytywać z poziomu programu PLC lub poprzez linię szeregową.

Szeregowa komunikacja z pamięcią DataBox

Dla szeregowej komunikacji można wykorzystać dowolny szeregowy kanał pracujący w trybie **PC**. Takim kanałem jest zawsze kanał CH1 przeznaczony do programowania jednostki centralnej i ewentualnie któryś z pozostałych szeregowych kanałów (CH2 do CH12), nastawiony na tryb **PC**. Program umożliwiający odczyt danych z pamięci DataBox do pliku, ew. zapis danych z pliku do pamięci DataBox, nazywa się complc.exe. Umożliwia również przetestowanie rozmiaru pamięci dostępnej jako DataBox. Program complc.exe jest częścią dyskietki dystrybucyjnej xPRO, musi być uruchomiony pod systemem operacyjnym MS-DOS. Pracę z pamięcią DataBox wspiera od wersji 1.6.

Instrukcje użytkownika

Do pracy z pamięcią DataBox istnieją trzy instrukcje użytkownika (termin instrukcja użytkownika - patrz rozdz.9). Instrukcja READDBX do odczytu danych z DataBoxu do rejestrów R, instrukcja WRITEDBX do zapisu danych z rejestrów R do DataBoxu, oraz instrukcja SIZEDBX do identyfikacji rozmiaru DataBoxu.

Definicja instrukcji użytkownika

Instrukcję użytkownika należy najpierw zdefiniować:

```
#usi u_readdbx = c:\xpro\usi\readdbx    ;ścieżka i nazwa pliku
#usi u_writedbx = c:\xpro\usi\writedbx  ;ścieżka i nazwa pliku
#usi u_sizedbx = c:\xpro\usi\sizedbx    ;ścieżka i nazwa pliku
#def READDBX usi u_readdbx              ;nazwanie USI
#def WRITEDBX usi u_writedbx            ;nazwanie USI
#def SIZEDBX usi u_sizedbx              ;nazwanie USI
```

Opis działania instrukcji SIZEDBX

Do sprawdzenia rozmiaru zainstalowanego DataBoxu jest przeznaczona instrukcja użytkownika SIZEDBX. Ta instrukcja nie wymaga żadnych parametrów wejściowych. Po wykonaniu zwiększy akumulator o jeden poziom a na szczycie akumulatora zapisze odczytany rozmiar DataBoxu w KB, tzn. wartość 128, 256 lub 512. Jeśli DataBox nie został znaleziony, instrukcja zwraca wartość 0.

Podczas symulacji w programie xPRO używa się USI instrukcję sizedbx.udl, która wytworzy plik databox.$$$ w aktualnym folderze (jeśli jeszcze nie istnieje). Rozmiar pliku, który chcemy wytworzyć, podaje się w warstwie A0 aktywnego akumulatora w KB. Nowo wytworzony plik jest binarny i jest wypełniony zerami. USI instrukcja SIZEDBX zwiększa akumulator o jeden poziom i zwraca wymagany rozmiar DataBoxu w następujących przypadkach:

- plik databox.$$$ był pomyślnie wytworzony
- plik databox.$$$ już istnieje i ma wymagany rozmiar (lub jest większy)

W następujących przypadkach USI SIZEDBX będzie zwracać w A0 wartość 0:

- pliku databox.$$$ nie udało się wytworzyć (np. jest zbyt mało miejsca na dysku)
- plik databox.$$$ istnieje, ale przy jego otwieraniu powstał błąd (np. jest zniszczona struktura pliku lub sterownik dysku wykrył błędny sektor, itd.)
- plik databox.$$$ istnieje, ale ma mniejszy rozmiar, niż jest wymagany - w tym przypadku zostanie zawartość pliku databox.$$$ niezmieniona. Aby dla symulacji był wytworzony plik databox.$$$ o większym rozmiarze, należy pierwotny plik zmazać lub przesunąć do innego folderu.

Z powyższego wypływa, że w symulacji USI SIZEDBX pracuje z plikiem databox.$$$ w aktualnym folderze. Rozmiar wytworzonego pliku odpowiada wymaganiu, które należy zadać w warstwie A0 przed wywołaniem USI. W realnym PLC rozmiar DataBoxu jest dany rozmiarem zainstalowanej pamięci, co oznacza, że może być większy niż jest wymagany. Przykład podany na końcu tego rozdziału rozwiązuje tę sytuację poprzez nieostrą nierówność, dzięki czemu można go użyć jak dla symulacji, tak i dla realnego PLC. Testowanie rozmiaru DataBoxu typowo następuje w ramach procesu dla ciepłego lub zimnego restartu.

Struktura strefy parametrów dla READBX i WRITEDBX

Przed wywołaniem instrukcji READDBX i WRITEDBX należy nastawić kilka parametrów. Te parametry są umieszczone w rejestrach R, muszą być zapisane ściśle za sobą, z zachowaniem kolejności. Numer rejestru, w którym jest umieszczony pierwszy parametr, przekazywany jest w akumulatorze przy wywołaniu instrukcji READDBX i WRITEDBX (patrz dalej). Parametry są uporządkowane w następującej kolejności:

Nazwa parametru	Typ	Znaczenie
adrDB	long	adres w pamięci DataBox
indR	word	indeks początkowego rejestru w pamięci
len	byte	ilość przenoszonych bajtów

Parametry można w programie definiować absolutnie

```
#def adrDB RL100
```

```
#def indR  RW104
#def len   R106
```

lub lepiej symbolicznie z automatycznym przydzieleniem rejestrów

```
#reg long adrDB
#reg word indR
#reg byte len
```

Z punktu widzenia programowania najlepsza jest definicja za pomocą dyrektywy *#struct*, ponieważ automatycznie zapewnia ciągłość deklarowanych parametrów:

```
#struct parDB              ;nazwa struktury
            long adrDB,    ;adres w DataBox
            word indR,     ;indeks rejestru w pamięci
            byte len       ;ilość przenoszonych bajtów
;
#reg parDB parusi
```

Opis działania instrukcji READDBX i WRITEDBX

Instrukcje READDBX i WRITEDBX nie zmieniają poziomu akumulatora. Na szczycie akumulatora zwracają ilość rzeczywiście przesuniętych danych. Jednocześnie nastawiają zawartość systemowego rejestru S1.0 w znaczeniu:

S1.0 = 1 - wejściowe parametry w porządku, wynik jest aktualny
S1.0 = 0 - wejściowe parametry poza zakresem, wynik jest nieaktualny

Jeśli jest S1.0 = 0, nie dojdzie do przeniesienia żadnych danych i jednocześnie nastawiony jest rejestr S34 w znaczeniu:

S34 = 14 - źródłowy blok danych był zdefiniowany poza zakresem
S34 = 15 - docelowy blok danych był zdefiniowany poza zakresem

W zależności od rozmiaru zainstalowanej pamięci DataBox dostępny jest adresowy obszar, który można określić wzorem

dostępny obszar = rozmiarDB – (rozmiarDB / 32 KB)

Tab.7.1 Dostępny adresowy obszar poszczególnych pamięci DataBox

Rozmiar pamięci DataBox	Dostępny adresowy obszar
128 KB	0 - $1FFFB
256 KB	0 - $3FFF7
512 KB	0 - $7FFEF

Przy próbie odczytu lub zapisu poza dostępnym obszarem nastawiony jest S1.0 = 0 i jednocześnie nastawiony jest odpowiedni kod błędu w S34.

Podczas symulacji w programie xPRO użyta jest USI instrukcja readdbx.udl oraz writedbx.udl, która wykonuje odczyt ew. zapis do pliku databox.$$$. Jeśli nie można tego pliku otworzyć lub powstanie błąd podczas odczytu ew. zapisu do tego pliku, przy symulacji USI nastawi S1.0 = 0 i jednocześnie A0 = 0.

Przykład użycia

Instrukcje READDBX, WRITEDBX i SIZEDBX można użyć w programie użytkownika na przykład tak:

```
#program DataBox
;
#usi u_readdbx = c:\xpro\usi\readdbx    ;ścieżka i nazwa pliku
#usi u_writedbx = c:\xpro\usi\writedbx  ;ścieżka i nazwa pliku
#usi u_sizedbx = c:\xpro\usi\sizedbx    ;ścieżka i nazwa pliku
#def READDBX usi u_readdbx              ;nazwanie USI
#def WRITEDBX usi u_writedbx            ;nazwanie USI
#def SIZEDBX usi u_sizedbx              ;nazwanie USI
;
#reg long 100,adrDB        ;zmienne, które sterują czynnością READDBX
#reg word indR
#reg byte len
#reg bit  DataBoxOK        ;wskaźnik DataBox w porządku
;
P 63
      :
      LD     32            ;wymagany rozmiar DataBoxu
                           ;dla aplikacji
      SIZEDBX              ;identyfikacja rozmiaru DataBoxu
      GT
      NEG                  ;DataBox wymaganej lub większej wielkości?
      WR     DataBoxOK     ;nastawić wskaźnik
      :
E 63
;
P 0
      :
```

```
        LD      DataBoxOK     ;DataBox w porządku ?
        JMC     koniecDBX     ;nie
        LDL     $FC00         ;adres w DataBox (long !!!)
        WR      adrDB
        LD      200           ;do którego rejestru zostaną
        WR      indR          ;przesunięte dane z DataBoxu
        LD      56            ;ilość przenoszonych bajtów
        WR      len
        LD      100           ;numer rejestru, gdzie leżą
                              ;parametry
        RDB                   ;odczyt bloku danych z DataBoxu do pamięci
                              ;blok o długości 56 bajtów zostanie
                              ;odczytany z adresu $FC00 i zapisany od R200
koniecDBX:
        :
E 0
```

Instrukcje użytkownika

W przypadku, gdy mamy do dyspozycji starsze typy jednostek centralnych, które nie mają wdrożonych instrukcji RDB, WDB i IDB, to użyć można instrukcji użytkownika READDBX, WRITEDBX i SIZEDBX (pojęcie instrukcja użytkownika patrz rozdz. 12). Ich funkcja jest równoznaczna z odpowiednimi instrukcjami, włącznie z symulacją w środowisku xPRO.

Definicje instrukcji użytkownika

Instrukcje użytkownika należy zdefiniować na początku programu użytkownika::

```
#usi u_readdbx = c:\xpro\usi\readdbx    ;ścieżka i nazwa pliku
#usi u_writedbx = c:\xpro\usi\writedbx  ;ścieżka i nazwa pliku
#usi u_sizedbx = c:\xpro\usi\sizedbx    ;ścieżka i nazwa pliku
#def RDB usi u_readdbx                  ;nazwanie USI
#def WDB usi u_writedbx                 ;nazwanie USI
#def IDB usi u_sizedbx                  ;nazwanie USI
```

Tych sześć linijek wstawimy do programu użytkownika bezpośrednio za pierwszą linijką zawierającą dyrektywę *#program*. W ten sposób uzupełniliśmy w PLC brakujące funkcje. Reszta programu użytkownika pozostaje bez zmian.

8. AKUMULATOR WYNIKÓW

Struktura akumulatora

Podczas wykonywania programu użytkownika PLC pracuje z akumulatorem, który ma 8 poziomów o szerokości word oznaczonych A0 do A7 (akumulator wyników). Jego struktura jest przedstawiona na rys.8.1. Aktywny poziom A0, oznaczany również jako szczyt akumulatora, używany jest w zdecydowanej większości instrukcji.

Z akumulatorem wykonywane są operacje w formatach bit, byte, word, long i float, logiczne operacje, arytmetyczne operacje, operacje przesunięć danych. Przekazywane są w nim logiczne i liczbowe parametry bardziej skomplikowanych instrukcji i podprogramów.

Cykliczny akumulator

Akumulator jest cykliczny (rys.8.2), można go sobie wyobrazić jako bębnową pamięć podaną na rys.8.3.

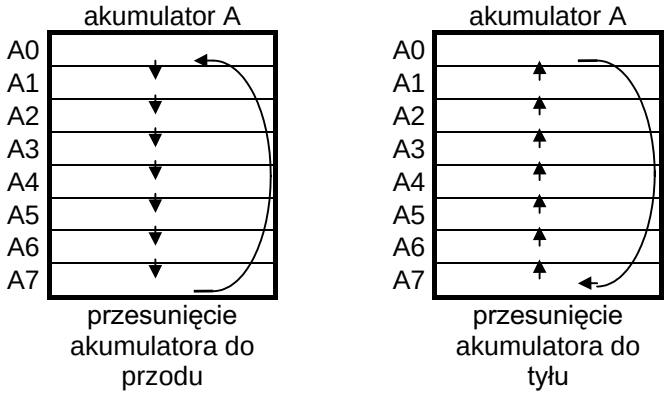
Osiem przełączanych akumulatorów

Akumulatorów jest 8 (oprócz jednostek centralnych typu E, posiadających tylko 1 akumulator) oznaczonych literami A do H. Aktywny jest zawsze jeden i można się pomiędzy nimi przełączać (rys.8.4). W ten sposób otwierają się wielkie możliwości w obszarze przekazywania parametrów pomiędzy funkcjami w programie użytkownika bez konieczności mniej przejrzystego zapamiętywania parametrów w pamięci.

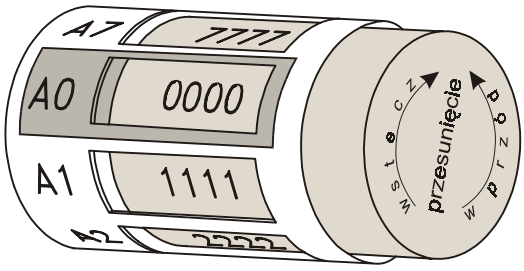
akumulator A

long float	word	górny bajt bit 15 . . . 8	dolny bajt 7 . . . 0
A01	A0	A0H	A0L
	A1	A1H	A1L
A23	A2	A2H	A2L
	A3	A3H	A3L
A45	A4	A4H	A4L
	A5	A5H	A5L
A67	A6	A6H	A6L
	A7	A7H	A7L

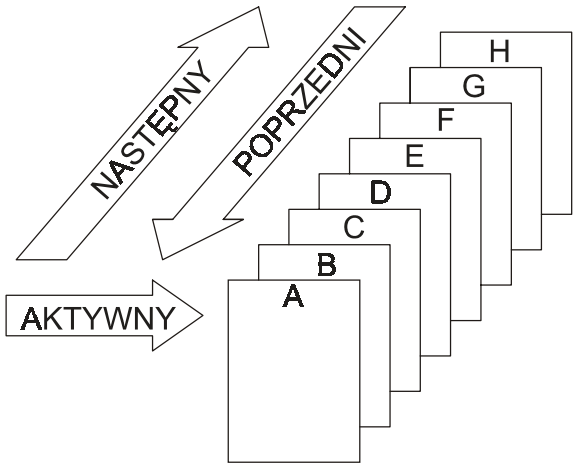
Rys.8.1 Schemat struktury akumulatora A



Rys.8.2 Funkcja cyklicznego akumulatora A



Rys.8.3 Szkic akumulatora A jako pamięci bębnowej



Rys.8.4 Przełączanie akumulatorów

8.1. STRUKTURA AKUMULATORA

Szczyt akumulatora

Szczyt akumulatora (A0, ew. A01) jest aktywną warstwą, która ma za zadanie gromadzić wyniki (akumulator). Następne warstwy (A1 do A7, ew. A23 do A67) zawierają ciąg poprzednich wartości szczytu akumulatora.

Przesunięcie akumulatora do przodu

Przesunięcie akumulatora do przodu powodują instrukcje LD, LDC oraz niektóre bardziej skomplikowane instrukcje. Przy każdym przesunięciu akumulatora do przodu o jeden poziom, wartości wszystkich jego warstw A0 do A6 są przesunięte do warstw z numerami o jeden wyższymi, a na szczycie akumulatora A0 zapisana jest nowa wartość w sposób następujący:

A0 ← nowe dane
A1 ← pierwotna zawartość A0
A2 ← pierwotna zawartość A1
.....
A7 ← pierwotna zawartość A6

Pierwotna zawartość A7 zostanie bezpowrotnie stracona (zastąpiona jest nową wartością A0).

Przesunięcie akumulatora do tyłu

Przesunięcie akumulatora do tyłu wykonuje instrukcja POP oraz bezoperandowe instrukcje operacji arytmetycznych i logicznych. Przesunięcie wstecz przebiega następująco:

A0 ← pierwotna zawartość A1 lub wynik operacji pomiędzy A0 a A1
A1 ← pierwotna zawartość A2
A2 ← pierwotna zawartość A3
.....
A7 ¬ pierwotna zawartość A0

8.2. INTERPRETACJA FORMATÓW DANYCH W AKUMULATORZE

Interpretacja formatów danych

Każda warstwa akumulatora ma szerokość 16 bitów (1 word lub 2 bajty). Przy operacji z akumulatorem dobrze jest oznaczyć symbolicznie poszczególne części (rys.5.1). Dolny bajt oznacza się literą L (low) - A0L, A1L, ... A7L. Górny bajt oznacza się literą H (high) - A0H, A1H, ... A7H. Całą warstwę o szerokości word oznacza się wówczas A0, A1, ... A7. Dane w formatach long i float zajmują dwie warstwy. Chodzi wówczas o tzw. podwójną warstwę oznaczaną A01, A23, A45, A67.

8.2.1. Dane w formacie bit

Format bit jest transformowany na format word

Jeśli na szczyt akumulatora wgrywane są dane formatu bit (np. LD R0.1), wówczas wartość bitu (0 lub 1) jest zapisana na wszystkich szesnastu bitach warstwy A0 (A0 = 0 lub A0 = 65 535).

Odwrotnie jest, gdy zapisuje się dane w formacie bit (np. WR R8.5). Wówczas, w przypadku zerowej wartości A0 zapisywana jest zerowa wartość bitu, a w przypadku

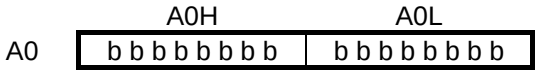
dowolnej niezerowej wartości A0 zapisuje się jedynkę. Zapisywana jest więc logiczna suma (OR) wszystkich szesnastu bitów szczytu A0.

Ten sposób umożliwia łatwą formatową konwersję, można więc kombinować instrukcje z operandami typu bit, byte i word praktycznie bez ograniczeń.

Jeśli więc mówimy o zawartości szczytu akumulatora jako o wartości formatu bit, należy używać następującego oznaczenia:

log.0 logiczne zero, A0 = 0
log.1 logiczna jedynka, A0 ≠ 0

Poprzez szczyt akumulatora rozumiana jest cała warstwa A0.



Rys.8.5 Zapis danych formatu bit na szczycie akumulatora
b - logiczna wartość bitu (log.0 lub log.1)

8.2.2. Dane w formacie byte

Format byte jest uzupełniony zerem na format word

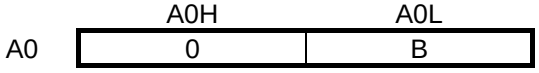
Jeśli na szczyt akumulatora wgrywane są dane formatu byte (np. LD R0), wówczas wartość tego bajtu jest zapisana w dolnym bajcie warstwy A0 (bajt A0L), górny bajt warstwy A0 jest wyzerowany (A0H = 0).

Natomiast jeśli zapisuje się dane formatu byte (np. WR R8), wówczas na adres docelowy zapisany jest jedynie dolny bajt warstwy A0 (bajt A0L). Górny bajt A0H jest ignorowany.

Ten sposób umożliwia łatwą formatową konwersję, można więc kombinować instrukcje z operandami bit, byte i word praktycznie bez ograniczeń.

Dane w formacie byte osiągają wartości 0 do 255.

Poprzez szczyt akumulatora rozumiana jest cała warstwa A0.



Rys.8.6 Zapis danych formatu byte na szczycie akumulatora
B - wartość bajtu

8.2.3. Dane w formacie word

Format word odpowiada podstawowemu formatowi akumulatora

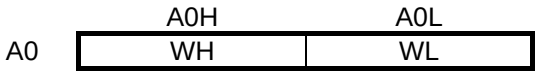
Dane w formacie word mają szerokość 2 bajtów.

Jeśli na szczyt akumulatora wgrywane są dane formatu word (np. LD RW0), wówczas cała aktualna warstwa A0 jest napełniona tymi danymi. Zawartość rejestru o niższym numerze jest zapisana w dolnym bajcie warstwy A0L, zawartość rejestru o wyższym numerze jest zapisana w górnym bajcie warstwy A0H.

Natomiast jeśli zapisuje się dane w formacie word (np. WR RW8), wówczas na adresie docelowym zapisana jest cała zawartość warstwy A0. Zawartość dolnego bajtu warstwy A0L jest zapisana w rejestrze o niższym numerze, zawartość górnego bajtu warstwy A0H jest zapisana do rejestru o wyższym numerze.

Dane w formacie word osiągają wartości 0 do 65 535.

Poprzez szczyt akumulatora rozumie się całą warstwę A0.



Rys.8.7 Zapis danych formatu word na szczycie akumulatora
WH - wartość wyższego bajtu
WL - wartość niższego bajtu

8.2.4. Dane w formacie long

Format long wykorzystuje tzw. podwójne warstwy

Dane w formacie long mają szerokość 4 bajtów.

Jeśli na szczyt akumulatora wgrywane są dane formatu long (np. LD RL0), wówczas zapełniają one całe dwie warstwy A0 i A1, które łącznie tworzą podwójną warstwę A01. Zawartość rejestru o najniższym numerze jest zapisana do najniższego bajtu podwójnej warstwy A0L, zawartość rejestru o najwyższym numerze jest zapisana do najwyższego bajtu podwójnej warstwy A1H.

Natomiast jeśli zapisuje się dane w formacie long (np. WR RL8), wówczas na adres docelowy zapisana jest cała zawartość podwójnej warstwy A01. Zawartość najniższego bajtu podwójnej warstwy A0L jest zapisana do rejestru o najniższym numerze, zawartość najwyższego bajtu podwójnej warstwy A1H jest zapisana do rejestru o największym numerze.

Dane w formacie long osiągają wartości od 0 do 4 294 967 295.

Poprzez szczyt akumulatora rozumiana jest cała podwójna warstwa A01.

		A0H	A0L
A01	A0	L1	L0
	A1	L3	L2
		A1H	A1L

Rys.8.8 Zapis danych formatu long na szczycie akumulatora
L0 - wartość najniższego bajtu
:
L3 - wartość najwyższego bajtu

8.2.5. Dane w formacie float

Format float odpowiada formatowi long

Dane formatu float mają szerokość 4 bajtów tak samo jak format long. Przyjmuje się więc dla nich te same (podane wyżej) reguły.

Dane formatu float zgodnie z IEEE-754 osiągają wartości w zakresie ok. $\pm 1{,}175494 \times 10^{-38}$ do $\pm 3{,}402823 \times 10^{38}$. Następnie istnieją trzy wartości oznaczające następujące stany:

$FFFFFFFF - niewłaściwa liczba (NaN - not a number)
$7F800000 - przekroczenie zakresu dodatnich liczb (+INF)
$FF800000 - przekroczenie zakresu liczb ujemnych (–INF)

Poprzez szczyt akumulatora rozumiana jest cała podwójna warstwa A01.

		A0H	A0L
A01	A0	mmmmmmmm	mmmmmmmm
	A1	s e e e e e e e	e mmmmmmm
		A1H	A1L

Rys.8.9 Zapis danych formatu float na szczycie akumulatora
s - znak (1 bit)
e - eksponent (8 bitów)
m - mantysa (23 bitów)

8.3. PRZEŁĄCZANIE AKUMULATORÓW

Przełączanie akumulatorów

Istnieje 8 akumulatorów oznaczonych A, B, C, D, E, F, G, H, z których każdy ma taką strukturę, jaka była opisana dla akumulatora A w rozdz.8.1. Aktywny jest zawsze jeden akumulator, pozostałe są zachowane w takim stanie, w jakim się znajdowały przed ich opuszczeniem w programie użytkownika, i żaden stan programu użytkownika nie ma na nie wpływu. Na przykład przy otwarciu nowego procesu zerowany jest jedynie aktywny akumulator (rozdz.10.).

Początkowy stan programu użytkownika jest zawsze taki sam

Po załączeniu lub restarcie PLC są wszystkie akumulatory wyzerowane i aktywny jest zawsze akumulator A. Po obrocie cyklu zawsze jest wyzerowany akumulator A i nastawiony jest jako aktywny.

W dowolnym momencie można akumulatory przełączyć, tzn. że program opuści dotychczas aktywny akumulator i będzie nadal pracować z nowo wybranym akumulatorem. Jednocześnie pamiętany jest stan rejestrów wskaźnikowych S0 i S1.

Instrukcje do przełączania akumulatorów

Do przełączania akumulatorów służą trzy instrukcje:

NXT - (next) aktywacja następnego akumulatora w kolejności (rys.8.4)
np. jeśli był aktywny akumulator C, będzie następnie aktywowany akumulator D

PRV - (previous) aktywacja poprzedniego akumulatora w kolejności (rys.8.4)
np. jeśli był aktywny akumulator C, będzie następnie aktywowany akumulator B

CHG - aktywacja dowolnego akumulatora
np. CHG F aktywuje akumulator F, stan wskaźnikowych rejestrów S0 i S1 się nie zmienia
np. CHG FS zapisze stan systemowych rejestrów S0 i S1 dotyczących dotychczas aktywnego akumulatora i aktywuje akumulator F, w systemowych rejestrach S0 i S1 odświeży stan odpowiadający chwili, w której aktywny akumulator F był opuszczony za pomocą instrukcji CHG mS, gdzie m jest akumulatorem A do H, lub za pomocą instrukcji NXT lub PRV

9. DYREKTYWY KOMPILATORA

Wykaz dyrektyw

Dyrektywy są poleceniami dla kompilatora, umożliwiającymi optymalne sterowanie procesem kompilacji. Wszystkie dyrektywy zaczynają się od znaku #. W środowisku xPRO istnieją następujące dyrektywy:

#program
#unit
#include
#def
#reg
#table
#data
#struct
#if, #elif, #else, #endif
#macro, #endm
#label
#usi
#mnemo, #mnemoend
#option

9.1. #program

Dyrektywa dla początku programu

Dyrektywa *#program* oznacza początek programu. Musi być obowiązkowo podana w pierwszym wierszu programu (tj. mogą ją poprzedzać jedynie puste wiersze lub wiersze rozpoczynające się od znaku ';').

Składnia jest następująca:

```
#program    nazwa [,[V][xyz]]
```

nazwa - nazwa własna programu, maksymalnie 16 znaków (dłuższa nazwa jest obcięta na tę długość)
V - nieobowiązkowy prefiks wersji programu
xyz - trzy nieobowiązkowe znaki charakteryzujące wersję programu

Z powyższego nagłówku programu kompilator wytworzy początkową część programu. To oznacza, że nazwa programu i jego wersja stanie się częścią skompilowanego kodu maszynowego. Te informacje można w dowolnym momencie odczytać z PLC, wybierając z menu *Automat | Informacje o PLC.* Z tego powodu zaleca się przede wszystkim wykorzystywać wersję programu tak, aby przy każdej zmianie w programie zmieniła się też jego wersja. Częstym problemem w praktyce jest sprawdzenie, jaki wariant programu jest w danej chwili wgrany do PLC. Dokładne numerowanie wersji zapobiega więc problemom przy późniejszych zmianach programu.

9.2. #unit

Dyrektywa dla softwarowej konfiguracji

Za pomocą dyrektywy *#unit* wytworzona jest lista jednostek w poszczególnych modułach, potrzebna do poprawnej pracy programu użytkownika. Na podstawie wykrytej konfiguracji jest przez kompilator wytworzona softwarowa konfiguracja (patrz podręczniki dla PLC).

Składnia jest następująca:

```
#unit   moduł,jednostka,typ_jedn,obraz_X,obraz_Y,aktywacja[,tab]
```

moduł - adres modułu, w którym jednostka jest zainstalowana
jednostka - adres jednostki w module (nastawiany złączkami)
typ_jedn - liczbowy kod charakteryzujący jednostkę
Ten kod liczbowy nie jest używany bezpośrednio, ale za pomocą symbolicznych nazw definiowanych standardowo w pliku xpro.sys
obraz_X - operand (nie musi być koniecznie w obszarze X) określający, do których rejestrów są przesyłane dane z wejść zdefiniowanej jednostki
obraz_Y - operand (nie musi być koniecznie w obszarze Y) określający, z których rejestrów dane są przesyłane do wyjść zdefiniowanej jednostki
aktywacja - kod (również zastąpiony symboliczną nazwą zdefiniowaną w pliku xpro.sys) określający tryb jednostki

Może osiągać wartości:
X_on = wejścia aktywne (są kopiowane do *obraz_X*)
Y_on = wyjścia aktywne (są kopiowane z *obraz_Y*)
On = wejścia i wyjścia są aktywne
Off = wejścia i wyjścia są nieaktywne

tab - nieobowiązkowy parametr, określający nazwę tabelki (patrz dyrektywa *#table*) z wartościami potrzebnymi do inicjalizacji jednostki
Tę inicjalizację używają jedynie niektóre specjalne rodzaje jednostek peryferyjnych, systemowe kanały szeregowe, itp.

Przykład 9.1

Definiujmy jednostkę wejściową z 16-bitowymi wejściami, umieszczoną w module o numerze 0, z adresem jednostki 1, dane z jednostki są kopiowane do rejestrów X2 i X3, wejścia są aktywne.

```
#unit 0, 1, DigitIn16, X2, X_on
```

Tekstowy plik zawierający opis wszystkich jednostek zawartych w PLC systemu można wygenerować automatycznie w środowisku xPRO za pomocą polecenia *Automat* | *Generuj plik modułów*. Zostanie wytworzony plik xxx.mdl, gdzie xxx jest nazwą zgodną z nazwą otwartego pliku źródłowego. Ten wygenerowany plik można wstawić do programu za pomocą dyrektywy *#include* (patrz dalej). Wytworzony plik jest tekstowy. Oznacza to, że można go w razie potrzeby zmienić za pomocą edytora w xPRO. Opisane postępowanie jest w praktyce najczęściej używanym sposobem tworzenia softwarowej konfiguracji PLC. Informacje zawarte w dyrektywach *#unit* po kompilacji stają się częścią kodu maszynowego i służą jednostce centralnej PLC do sterowania transmisją danych pomiędzy pamięcią a jednostkami peryferyjnymi.

9.3. #include

Dyrektywa do wstawiania plików

Podczas pisania długich programów, z punktu widzenia przejrzystości wygodnie jest rozdzielić cały program na kilka mniejszych, logicznie kompaktowych plików. Takie łączenie plików umożliwia dyrektywa *#include***.**

Składnia jest następująca:

```
#include nazwa_pliku
```

nazwa_pliku - oznacza plik, który należy podłączyć (wstawić) do aktualnie kompilowanego programu. Jeśli nie znajduje się w aktualnym folderze roboczym, należy zadać też ścieżkę dostępu.

Kiedy kompilator znajdzie w źródłowym tekście programu dyrektywę *#include*, przerwie kompilowanie w aktualnym pliku i kontynuuje kompilację pliku zdefiniowanego przez dyrektywę *#include*. Po skompilowaniu całego tego pliku kontynuuje kompilację tekstu z poprzedniego pliku, który opuścił. Kompilator radzi sobie bez problemów w sytuacjach, gdy plik zdefiniowany przez dyrektywę #*include* zawiera kolejne dyrektywy *#include*.

W przypadku, gdy używa się łączenia (wstawiania) plików, wygodnie jest w środowisku xPRO w pozycji *Opcje* | *Kompilator* | *Główny plik* zadać nazwę pliku, który ma być zawsze kompilowany jako pierwszy. Z reguły jest to plik zawierający nagłówek programu. Nowsze wersje środowiska xPRO mają tę opcję zastąpioną możliwością wstawienia kompilowanych plików do projektu. Każde zadanie może mieć własny projekt, a xPRO wspiera wówczas szybkie przejście pomiędzy poszczególnymi projektami, co jest cenne zwłaszcza przy programowaniu kilku systemów PLC w sieci.

Przykład 9.2

```
#program Demo_include, v1.0
;
#include KONFIG.MDL          ;sw konfiguracja
#include DEFINICJA.950       ;wstawianie plików
;
P 0
        LD     X0.0
        WR     Y0.0
E 0
```

9.4. #def

Dyrektywa do definiowania nazw symbolicznych

Za pomocą dyrektywy *#def* definiuje się nazwy symboliczne, które można następnie używać w programie np. zamiast operandów absolutnych.

Składnia jest następująca:

```
#def  symboliczna_nazwa podstawiona_nazwa
```

symboliczna_nazwa - ciąg złożony z dowolnej ilości dozwolonych znaków (patrz dyrektywa *#program*)

podstawiona_nazwa - inny dowolny łańcuch złożony z dozwolonych znaków (absolutny operand, stała itd.)

Dyrektywa *#def* może się pojawić w dowolnym miejscu programu po jego nagłówku, zawsze jednak przed pierwszym użyciem zdefiniowanej nazwy symbolicznej. Przyjmuje się więc ogólną zasadę, że najpierw należy obiekt zdefiniować, a następnie można go dopiero użyć.

Przykład 9.3

```
#def    Zezwalajacy_Bit     R5.0          ;definicja bitu
#def    STALA         80                  ;opóźnienie
#def    ZWLOKA        STALA+20
#def    sec           2                   ;typ układu czasowego 1 s
#def    Ukl_czas_T1   RW0.sec             ;układ czasowy 1 s
#def    wyjsciowy_word      YW0           ;wyjście
```

9.5. #reg

Dyrektywa do definiowania zmiennych

Dyrektywa *#reg* służy do tzw. automatycznego definiowania zmiennych w obszarze R. Za pomocą tej dyrektywy przydziela się symboliczne nazwy zmiennych i jednocześnie wymaga się od kompilatora, aby dla tej zmiennej zarezerwował miejsce w rejestrach R w pamięci PLC. Ilość rejestrów, które zostaną przeznaczone dla zmiennej, określa wymagany typ danych.

Składnia jest następująca:

```
#reg typ [indeks,]nazwa_zmiennej[,następna_zmienna...]
```

typ - określa typ zmiennej przyporządkowanej symbolicznie do nazwy nazwa_zmiennej
Dozwolone typy przyporządkowanej zmiennej są podane w tab.9.1.

nazwa_zmiennej, następna_zmienna - symboliczne nazwy definiowanych rejestrów

indeks - nieobowiązkowy parametr, podaje numer rejestru przyporządkowany nazwie symbolicznej nazwa_zmiennej w obszarze R

Tab.9.1 Dozwolone typy przyporządkowanej zmiennej

Typ	Zakres	Przykład
bit	R0.0 - Rmaks.7*	R10.2, R2.15
byte	R0 - Rmax*	R1, R2001
word	R0 - Rmax–1*	RW2, RW2002
long	R0 - Rmax–4*	RL4, RL2004
float	R0 - Rmax–4*	RF4, RF2004
struktura**		

* Rmaks zależy od typu użytej jednostki centralnej.

** Struktura jest nazwą (tag) struktury zdefiniowanej za pomocą dyrektywy *#struct*.

Kompilator automatycznie przydziela indeksy w kolejności rosnącej. Zadając nieobowiązkowy parametr index nastawia się wewnętrzny licznik na jego wartość, która jest następnie jako indeks przydzielona pierwszej na liście symbolicznej nazwie (*nazwa_zmiennej*). Dla następnej nazwy zmiennej (*następna_zmienna*) na liście przydzielanie jest kontynuowane od nastawionego indeksu. Najlepiej jest ręcznie indeksowane operandy umieścić na początku definicji rejestrów.

Za pomocą jednej dyrektywy *#reg* można wytworzyć dowolną ilość operandów, nazwy są w tym przypadku odseparowane przecinakami. Jeśli nazwy nie zmieszczą się w jednym wierszu, można kontynuować deklaracje na następnym wierszu.

Używanie dyrektywy *#reg* uwalnia programistę od żmudnej pracy z przydzielaniem indeksów dla zmiennych, podczas której można łatwo popełnić błąd.

Przyporządkowane wartości można sprawdzić po przeprowadzonej kompilacji programu za pomocą polecenia *Wyświetl* | *Symbole* - <Alt-F4> lub z wytworzonego pliku wypisu programu, którego częścią jest tabelka nazw symbolicznych. Jeśli załączy się opcję *Opcje* | *Kompilator* | *Generuj mapę,* to po skompilowaniu programu rezerwacja rejestrów będzie dostępna w pliku xxx.map, gdzie xxx jest zgodne z nazwą pliku zawierającego dyrektywę *#program*.

Przykład 9.4

```
#reg bit  Rbit00, Rbit01, Rbit02   ;Rbit00 = R0.0,
                                   ;Rbit01 = R0.1,
                                   ;Rbit02 = R0.2
#reg byte Rbyte0, Rbyte1           ;Rbyte0 = R1,
                                   ;Rbyte1 = R2
#reg word 10, Rword0, Rword1       ;Rword0 = RW10,
                                   ;Rword1 = RW12
```

Dyrektywa #reg umożliwia również deklarowanie jednowymiarowych pól. Za nazwą zmiennej w tym przypadku podaje się w kwadratowych nawiasach ilość elementów pola.

Przykład 9.5

Zdefiniujmy zmienną *pole*, która ma 20 elementów typu *word*, pierwszy element ma indeks 0 a ostatni element jest z indeksem 19. Jak widać, indeksy pól tak samo jako wszystko pozostałe w PLC TECOMAT, zaczynają się od 0.

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

Jeśli w programie chcemy odczytać wartość elementu pola np. z indeksem 15, dodać jedynkę i zapisać do elementu z indeksem 16, można użyć następujący zapis.

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

9.6. #struct

Dyrektywa dla deklarowania struktur danych

Dyrektywa *#struct* używana jest do deklarowania struktur danych, którymi są w zasadzie nowe typy danych wywodzące się od podstawowych typów danych (bit, byte, word, long, float) lub od wcześniej deklarowanych struktur. Za pomocą deklaracji struktury nie powstaje więc żadne wymaganie na rezerwację pamięci PLC, jedynie wprowadza się nowy typ danych, który można używać do automatycznego przydzielania (rezerwacji) zmiennych w dyrektywie *#reg*, przy definicji tabel, itd.

Deklaracja struktury składa się z nazwy (tag) struktury oraz typów i nazw poszczególnych elementów struktury.

Składnia jest następująca:

```
#struct struct_tag  [zaokrągl] typ0[[powt0] elem0[,...]...]
                    [zaokrągl] typn[[powtn] elemn]
```

struct_tag - nazwa (tag) struktury
zaokrągl - nieobowiązkowe zaokrąglanie indeksu elementu struktury na liczbę parzystą
typ0 – typn - typy elementów struktury (byte, word...) lub tag innej struktury
elem0 – elemn - nazwy elementów struktury
powt0 - powtn - nieobowiązkowe powtarzanie elementów struktury, zamknięte w nawiasach kwadratowych, są to liczby całkowite > 0

Dla definicji struktury przyjmuje się następujące reguły:

- jeśli definicja nie zmieści się w jednym wierszu, można ją kontynuować na następnym wierszu
- definicje poszczególnych elementów są odseparowane znakiem przecinek (',')
- każdy element struktury musi być jakiegoś typu i musi mieć przydzieloną nazwę, nazwa członka struktury **musi być niepowtarzalna** w całym programie
- elementem struktury może być też jednowymiarowe pole danego typu
- struktury w strefie T i D są zawsze zainicjalizowane, elementy struktury są zainicjalizowane w kolejności, w jakiej były zdefiniowane, brakujące wartości inicjalizacyjne są wypełnione zerami

Przykład 9.6

Prostą strukturę można zdefiniować np. następująco:

```
;deklaracja prostej struktury
#struct typCzas           ;nazwa struktury
  byte Godzina,           ;pierwszy element struktury
  byte Minuta,            ;drugi element struktury
  byte Sekunda            ;ostatni element struktury
```

Podana definicja wytworzyła nowy typ danych nazwany *typCzas.* Ten typ ma długość 3 bajtów i składa się z pozycji G*odzina, Minuta i Sekunda.* Wszystkie pozycje są typu

byte. Teraz można już za pomocą dyrektywy *#reg* zdefiniować zmienne, które będą typu *typCzas*.

```
#reg typCzas CzasUruchomienia
```

Za pomocą dyrektywy *#reg* została wytworzona zmienna *CzasUruchomienia,* która ma typ struktury *typCzas i* ma pozycje *Godzina, Minuta i Sekunda*. Pozycje są typu byte. Dostęp do zmiennej z poziomu programu przedstawia następująca instrukcja.

```
;dostęp do pozycji zmiennej CzasUruchomienia
      LD    CzasUruchomienia~Godzina
      LD    CzasUruchomienia~Minuta
      LD    CzasUruchomienia~Sekunda
```

Dla odnośników do elementów struktury w programie przyjmuje się:

- element struktury jest oddzielony znakiem '~' (tylda)
- wielowymiarowe elementy można indeksować od 0 do (*defSize*-1), gdzie *defSize* jest rozmiarem elementu
- jeśli rozwinięcie do skończonego elementu struktury nie powiedzie się, wówczas za operand uważany jest najbliższy element struktury

W programie można oczywiście wytworzyć dowolną ilość zmiennych typu zadeklarowanego za pomocą dyrektywy *#struct typCzas*.

```
;deklaracja kilku zmiennych typu typCzas
#reg typCzas Wymeldowanie, Zameldowanie, CzasObiadu
```

Następujący wiersz pokazuje deklarację pola zmiennych, których typ był w poprzednim tekście zdefiniowany za pomocą dyrektywy *#struct*.

```
;deklaracja pola zmiennych typu typCzas
#reg typCzas poleCzasu[10]       ;pole zmiennych
```

Do poszczególnych elementów pola można odnosić się z programu np. tak:

```
      LD    poleCzasu[0]~Godzina
      LD    poleCzasu[0]~Minuta
      LD    poleCzasu[0]~Sekunda
```

W dyrektywach *#struct* można podczas definiowania typu poszczególnych pozycji używać również typy zdefiniowane przez poprzednie dyrektywy *#struct*. Wspomniane, struktury można nawzajem kombinować i wytwarzać w ten sposób i skomplikowane typy.

Przykład 9.7

```
#program Demo_struct, v1.0
;definicja prostej struktury
#struct typCzas           ;nazwa struktury
  byte Godzina,           ;pierwszy element struktury
  byte Minuta,            ;drugi element struktury
  byte Sekunda            ;ostatni element struktury
;
;definicja następnej prostej struktury
#struct typData           ;nazwa struktury
  byte Dzien,             ;pierwszy element struktury
  byte Miesiac,           ;drugi element struktury
  word Rok                ;ostatni element struktury
;
;definicja struktury, której elementy stanowią inną strukturę
;(wgłębianie struktur)
#struct czasZnak          ;nazwa struktury
  typCzas   Czas,         ;pierwszy element struktury
  typData   Data,         ;drugi element struktury
  word      IloscSztuk    ;ostatni element struktury
;
#reg casZnak Dawka        ;zmienna typu czasZnak
#reg czasZnak Nota[10]    ;pole zmiennych typu czasZnak
;
P 0
      LD    Dawka~Czas~Godzina
      LD    Dawka~Data~Rok
      LD    Dawka~IloscSztuk
      :
      LD    Nota[1]~Czas~Godzina
      LD    Nota[9]~Data~Rok
      LD    Nota[0]~IloscSztuk
      :
E 0
```

Przykład 9.8

W niektórych przypadkach wymagane jest, aby pozycją struktury było pole. Definicja takiej struktury może wyglądać np. następująco :

```
#program Demo_struct, v1.1
;
#struct typCzas            ;nazwa struktury
  byte Godzina,            ;pierwszy element struktury
  byte Minuta,             ;drugi element struktury
  byte Sekunda             ;ostatni element struktury
;
#struct typData            ;nazwa struktury
  byte Dzien,              ;pierwszy element struktury
  byte Miesiac,            ;drugi element struktury
  word Rok                 ;ostatni element struktury
;
#struct SkomplikowanaNota
  typCas[2]   Czasy,
  typDatum[2] Daty,
  word        Sztuki
;
#reg SkomplikowanaNota Detal
;
P 0
       LD     Detal~Czasy[0]~Godzina
       LD     Detal~Daty[1]~Rok
       LD     Detal~Sztuki
       :
E 0
```

Chyba nie będzie zaskoczeniem, że nawet z tak zdefiniowanych struktur można wytwarzać pole zmiennych.

```
#reg SkomplikowanaNota Detale[3]
;
P 0
       LD     Detale[0]~Czasy[0]~Godzina
       LD     Detale[0]~Daty[1]~Rok
       LD     Detale[0]~Sztuki
       :
E 0
```

Na zakończenie tych demonstracji podamy jeszcze konkretne rozmieszczenie rejestrów w pamięci PLC dla tego przykładu. Podany niżej wypis rozmieszczenia rejestrów można uzyskać po skompilowaniu programu w pliku z rozszerzeniem .map.

```
Mapa rozmieszczenia obszaru rejestrów 'R'
;* Źródłowy plik˙C:\XPRO\PROGRAMY\STRUCT.950
;
R    0   :  47   =SKOMPLIKOWANANOTA DETALE [3]  ;
       R0     >> DETALE[0]~CZASY[0]~GODZINA
       R1     >> DETALE[0]~CZASY[0]~MINUTA
       R2     >> DETALE[0]~CZASY[0]~SEKUNDA
       R3     >> DETALE[0]~CZASY[1]~GODZINA
       R4     >> DETALE[0]~CZASY[1]~MINUTA
       R5     >> DETALE[0]~CZASY[1]~SEKUNDA
       R6     >> DETALE[0]~DATY[0]~DZIEN
       R7     >> DETALE[0]~DATY[0]~MIESIAC
       RW8    >> DETALE[0]~DATY[0]~ROK
       R10    >> DETALE[0]~DATY[1]~DZIEN
       R11    >> DETALE[0]~DATY[1]~MIESIAC
       RW12   >> DETALE[0]~DATY[1]~ROK
       RW14   > DETALE[0]~SZTUKI
       R16    >> DETALE[1]~CZASY[0]~GODZINA
       R17    >> DETALE[1]~CZASY[0]~MINUTA
       R18    >> DETALE[1]~CZASY[0]~SEKUNDA
       R19    >> DETALE[1]~CZASY[1]~GODZINA
       R20    >> DETALE[1]~CZASY[1]~MINUTA
       R21    >> DETALE[1]~CZASY[1]~SEKUNDA
       R22    >> DETALE[1]~DATY[0]~DZIEN
       R23    >> DETALE[1]~DATY[0]~MIESIAC
       RW24   >> DETALE[1]~DATY[0]~ROK
       R26    >> DETALE[1]~DATY[1]~DZIEN
       R27    >> DETALE[1]~DATY[1]~MIESIAC
       RW28   >> DETALE[1]~DATY[1]~ROK
       RW30   > DETALE[1]~SZTUKI
       R32    >> DETALE[2]~CZASY[0]~GODZINA
       R33    >> DETALE[2]~CZASY[0]~MINUTA
       R34    >> DETALE[2]~CZASY[0]~SEKUNDA
```

```
      R35    >> DETALE[2]~CZASY[1]~GODZINA
      R36    >> DETALE[2]~CZASY[1]~MINUTA
      R37    >> DETALE[2]~CZASY[1]~SEKUNDA
      R38    >> DETALE[2]~DATY[0]~DZIEN
      R39    >> DETALE[2]~DATY[0]~MIESIAC
      RW40   >> DETALE[2]~DATY[0]~ROK
      R42    >> DETALE[2]~DATY[1]~DZIEN
      R43    >> DETALE[2]~DATY[1]~MIESIAC
      RW44   >> DETALE[2]~DATY[1]~ROK
      RW46   > DETALE[2]~SZTUKI
Mapa kompletna.
-----------
```

Dyrektywy *#struct* umożliwiają więc deklarowanie nowych typów. Zdefiniowane w ten sposób typy można użyć nie tylko przy definiowaniu zmiennych, ale również przy definiowaniu tabel T lub danych D. Bliższe informacje podane są w opisie dyrektywy *#table* i *#data.*

9.7. #table, #data

Dyrektywy do deklarowania tabel i danych

Praca z tabelami jest silną stroną PLC TECOMAT. Dlatego też narzędziom do ich tworzenia w programie było poświęcone sporo uwagi. W taki sam sposób jak tabelki T definiuje się obszar danych D.

Ogólne postacie definicji tabelki i danych są:

```
#table typ   [index,]nazwaTab=[[wart0[powt0]],]
             wart1[[powt1]],
             ...,wartn-1[[powtn-1]],wartn[[powtn]]

#data  typ   [index,]nazwa0=[[wart0[powt0]],]
             [nazwa1=]wart1[[powt1]],...,
             [nazwan-1=]wart-1[[powtn-1]],
             [nazwan=]wart[[powtn]]
```

typ - typ elementów tabelki, może być taki sam jak w przypadku *#reg* (tab.9.1)

index - nieobowiązkowa liczba, która nastawia indeks wytwarzanej tabelki lub pierwszego rejestru D
Po wymuszeniu indeksu tabelki poprzez podanie liczby index, następne zdefiniowane tabelki mają przyporządkowane indeksy w rosnącej kolejności, zaczynając od indeksu *index+1*. Jeśli indeks jest ominięty, wówczas następne tabelki mają automatycznie przydzielony indeks (rosnąco, tak samo jak w przypadku *#reg*). Dla danych typu D przyjmuje się to samo.

nazwaTab - symboliczna nazwa tabelki T

nazwa0 - nazwan - symboliczna nazwa danych D

wart0 - wartn - elementy tabelki
Mogą być zadawane z wykorzystaniem dowolnego dozwolonego układu liczbowego. Jeśli wartość elementu przekroczy maksymalną wartość określoną przez typ tabelki (np. dla typu byte jest to 255), wówczas liczba jest obcięta; w przypadku definicji tabelki bitowej jej element jest niezerowy wówczas, gdy jego wartość wart*x* jest niezerowa.

powt0 - powtn nieobowiązkowa ilość powtórzeń
Są to liczby zamknięte w kwadratowych nawiasach (znaki '[' i ']'), umożliwiające powtórzenie w tabelce wartości *wartx* podanej przed nawiasem *powtx*-razy. Jeśli wartość *wartx* jest łańcuchem tekstowym, wówczas jest powtarzany jedynie jego ostatni znak.

Przykład 9.9

```
#program Demo_table_data, v1.0
;
#table bit    BitTable = 0,1,1,0,1,1,1,0,1[8],0,0
#table byte   10,ByteTable = $12,34,%01010110,60#56
#data  word   WordData = 1,2,3,
              NextData = 4,$4567[12],8,9,10,0[11],3
;
#def Zalaczone 1
#def Wylaczone 0
#def Auto    1
#def Recznie   0
#struct typNota
```

```
  bit ZalWyl,
  bit AutRecznie,
  byte numerMaszyny,
  word czasProg,
  long licznikCykli
;
#table typNota Nota1 = Zalaczone, Recznie, 20, $3009, 1236789
#table typNota[2] Nota2 =
                    Zalaczone, Recznie, 20, $3009, 1236789
                    Wylaczone, Auto,  21, 19029, 1236789
;
P 0
      :
E 0
```

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

Dyrektywy dla warunkowej kompilacji

Te dyrektywy służą do warunkowej kompilacji części programu.
Składnia jest następująca:

```
#if warunek_if          ;początkowy warunek i początek pierwszego
                        ;bloku warunkowej kompilacji
      :   ;część programu, kompilowana gdy arytmetyczne
      :   ;lub logiczne wyrażenie warunek_if jest niezerowe
#elif warunek_elif_1        ;początek kolejnego bloku
                            ;uwarunkowej kompilacji
      :     ;część programu, kompilowana gdy wyrażenie
      :     ;warunek_if jest zerowe a wyrażenie
      :     ;warunek_elif_1 jest niezerowe
#elif warunek_elif_n        ;początek przedostatniego bloku
                            ;warunkowej kompilacji
      :     ;część programu, kompilowana gdy wyrażenie
      :     ;warunek_if jest zerowe i jednocześnie wyrażenia
      :     ;warunek_elif_1 do warunek_elif_n-1 są zerowe
      :     ;a wyrażenie warunek_elif_n jest niezerowe
#else                       ;początek ostatniego bloku
                            ;warunkowej kompilacji
      :     ;część programu, kompilowana gdy nie jest
      :     ;spełniony ani jeden z poprzednich warunków
#endif                      ;koniec warunkowej kompilacji
```

warunek_if, *warunek_elif_1*, ..., *warunek_elif_n*
- wyrażenia relacyjne lub matematyczne

Kompilowana jest tylko ta gałąź, która pierwsza spełni swój warunek.

Dyrektywy *#if* ... *#endif* można nawzajem wgłębiać (zanurzać). Przyjmuje się dla nich następujące reguły:

- każdy *#if* musi mieć swój *#endif*
- *#elif* lub *#else* odnosi się do najbliższego poprzedniego *#if*
- *#elif* i *#else* są nieobowiązkowe

Przykład 9.10

Bardzo częstym przypadkiem użycia warunkowych kompilacji jest sytuacja, kiedy podczas strojenia programu w symulacji xPRO programista symuluje w programie odpowiedzi realnego urządzenia. Te części programu są praktycznie zawsze kompilowane warunkowo w przypadku strojenia podczas symulacji.

```
#program Demo_if, v1.0
;
#def SYMULACJA 1            ;1 ... strojenie w symulacji
                            ;0 ... realna maszyna
;
#if SYMULACJA == 1
#include symMaszyna.950     ;podczas strojenia kompiluj
                            ;symMaszyna.950
#endif
```

9.9. #usi

Dyrektywa dla deklarowania instrukcji użytkownika

Programowe środowisko xPRO umożliwia w programie używanie instrukcji użytkownika USI napisanych dla PLC TECOMAT. Instrukcja USI jest w zasadzie funkcją napisaną w języku C, której skompilowany kod jest przez kompilator xPRO dołączony do kodu maszynowego programu użytkownika. W ten sposób można rozszerzać zbiór instrukcji jednostki centralnej PLC o funkcje, które nie są częścią standardowego zbioru instrukcji. Do definiowania USI instrukcji w programie służy dyrektywa *#usi***.**

Jej składnia jest następująca:

```
#usi   [index,]nazwa_instrukcji = nazwa_pliku
```

index - nieobowiązkowa liczba, która nastawia indeks wytwarzanej instrukcji
Po wymuszeniu indeksu instrukcji poprzez podanie liczby index, kolejnym zdefiniowanym instrukcjom są przyporządkowane indeksy w rosnącej kolejności, zaczynając od indeksu *index+1*. Jeśli *index* jest pominięty, wówczas kolejnym instrukcjom jest automatycznie przydzielony rosnący indeks.

nazwa_instrukcji - symboliczna nazwa indeksu instrukcji użytkownika. W przypadku, gdy jest podany indeks, jest ona nieobowiązkowa, w przeciwnym razie jest obowiązkowa

nazwa_pliku - obowiązkowa nazwa binarnego pliku na dysku, który zawiera kod instrukcji użytkownika

Wraz z programowym środowiskiem xPRO dostarczany jest szereg USI instrukcji, które po zainstalowaniu programu są typowo umieszczone w folderze XPRO\USI. Każda USI instrukcja jest skompilowana kilka razy dla różnych typów jednostek centralnych PLC. Oznacza to, że dla każdej USI instrukcji z reguły istnieje kilka plików w folderze XPRO\USI. Typ centralnej jednostki, dla której jest plik z kodem USI instrukcji przeznaczony, jest odróżniony za pomocą rozszerzenia pliku. Jeśli rozszerzenie pliku w dyrektywie *#usi* nie zostanie podane, kompilator xPRO automatycznie wybierze plik z poprawnym rozszerzeniem w zależności od typu centralnej jednostki, z którą aktualnie pracuje. Dla symulacji są przeznaczone pliki z rozszerzeniem *.udl. Te pliki xPRO użyje automatycznie w trybie symulowanego PLC niezależnie od tego, dla jakiego typu jednostek centralnych była wykonana kompilacja. Pliki *.udl są użyte jedynie dla symulacji funkcji USI, do kodu maszynowego skompilowanego programu jest dołączony plik z kodem, który odpowiada wybranemu typowi jednostki centralnej. Oznacza to, że podczas przejścia z trybu symulacji PLC do strojenia z użyciem realnego PLC nie należy nic więcej robić w związku z użytymi instrukcjami USI.

Rozszerzenie pliku z kodem USI	Jednostka centralna
.uim	typ M.
.uis	typ S
.uid	typ D
.uia	typ A
.uib	typ B
.udl	dla symulacji PLC w środowisku xPRO

Przykład 9.11

```
#program Demo_usi, v1.0
;
#usi   panelOperacyjny = C:\XPRO\USI\ter_id04  ;definicja
;
P 0
       :
       USI  panelOperacyjny       ;użycie w programie
       :
E 0
```

lub

```
#program Demo_usi, v1.1
;
#usi   panelOperacyjny = C:\XPRO\USI\ter_id04  ;definicja
#def   TER    USI panelOperacyjny
;
P 0
       :
       TER                        ;użycie w programie
       :
E 0
```

9.10. #label

Dyrektywa dla sterowania numeracją etykiet

Za pomocą dyrektywy *#label* nastawia się początkowy indeks dla automatycznego przydzielania etykiet i rezerwuje się symboliczne etykiety do użycia np. w indeksowanych lub relatywnych skokach, itd. Ta dyrektywa jest więc przeznaczona jedynie dla tych przypadków, kiedy programista potrzebuje zarezerwować konkretne numery etykiet. Jeśli istnieje potrzeba umieszczenia w programie PLC etykiety np. jako celu skoku warunkowego, nie należy tej etykiety deklarować poprzez dyrektywę *#label*. Wystarczy jedynie napisać symboliczną nazwę etykiety na osobnym wierszu programu i zakończyć ją dwukropkiem (patrz rozdz.4.3).

Dyrektywa *#label* ma następującą postać:

```
#label [index,][nazwa_L0[[powt0]][,nazwa_L1[[powt1]],
                ...nazwa_Ln[[powtn]]]
```

index - nieobowiązkowa liczba dodatnia określająca pierwszy indeks dla automatycznego przydzielania etykiet.

nazwa_L0 - nazwa_Ln - dowolne symboliczne nazwy etykiet, do których są przyporządkowana absolutne etykiety, lista nazw etykiet może być kontynuowana na kilku rządkach

powt0 - powtn - nieobowiązkowa ilość powtórzeń zamknięta w kwadratowych nawiasach, określająca ile indeksów etykiet opuścić za zdefiniowaną etykietą.

Uwaga! Jeśli w programie używa się absolutne etykiety, należy zapewnić, aby żadna z nich nie była przydzielona innej (symbolicznej) etykiecie. W programach pisanych w środowisku xPRO nie zaleca się używać absolutnych etykiet. Zapobiega się w ten sposób kolizjom w przypadku podwójnej deklaracji tej samej etykiety, itp.

Przykład 9.12

```
#program Demo_label, v1.0
;obszar od L 0 do L 9 jest niewykorzystany
#label 10, PierwszaEtykieta [3],  ;PierwszaEtykieta = L 10
       Etyk, Etykieta[10],        ;Etyk = L 13, Etykieta = L 14
       NastepnaEtykieta           ;NastepnaEtykieta = L 24
#label 100, EtykietaSto           ;EtykietaSto = L 100
;
P 0
       :
E 0
```

9.11. #macro, #endm

Dyrektywy do definiowania makroinstrukcji

Kompilator xPRO dysponuje też silnym środkiem, jakim jest makroinstrukcja. Doświadczony programista potrafi za pomocą makroinstrukcji sprawić, że długi i trudno czytelny program stanie się bardziej przejrzysty.

Makroinstrukcji (w skrócie makro) używa się wszędzie tam, gdzie w programie pojawiają się takie same części, lecz używające inne operandy (na przykład sterowanie kilku silników). Jeśli ta część sterowania, która jest wszędzie taka sama (używa tego samego ciągu instrukcji), zostanie napisana jako makroinstrukcja, wówczas każda obsługa silnika w programie zostanie w prosty sposób zapisana w postaci makroinstrukcji. Różnić się będą jedynie parametry przekazywane tej makroinstrukcji.

Makroinstrukcje można wzajemnie zagłębiać, to znaczy używać w ciele makroinstrukcji inną makroinstrukcję.

Przykład 9.13

Zdefiniujmy krótką makroinstrukcję, która zsumuje logicznie dwa bity, a wynik zapisze do innego bitu:

```
#program Demo_macro, v1.0
#reg bit wejscie,wyjscie,a,b,c
;
;definicja makroinstrukcji
#macro  pierwsze_makro (pierwszy, drugi, trzeci)
        LD     pierwszy
        OR     drugi        ;logiczna suma
        WR     trzeci       ;wynik
#endm                       ;koniec definicji makroinstrukcji
;
P 0
        :
```

```
        LDC    wejście
        pierwsze_makro (a, b, c)
        WRC    wyjscie
        :
E 0
-----------
```

Jak widać, przy definicji makroinstrukcji ważne jest zachowanie następujących zasad:

- Definicja makroinstrukcji zaczyna się od dyrektywy *#macro* i kończy dyrektywą *#endm*.
- Za nazwą makroinstrukcji następuje lista tzw. formalnych parametrów makra, która jest zamknięta w okrągłych nawiasach. Nawet kiedy makroinstrukcja nie używa żadnych parametrów, należy użyć nawiasów.
- Jeśli lista formalnych parametrów nie zmieści się w jednym wierszu, można ją kontynuować na kolejnym wierszu. Np.:

```
#macro  dluga_makroinstrukcja (pierwszy_parametr,
        drugi_parametr)
```

- Ciało makroinstrukcji składa się z ciągu instrukcji. Wewnątrz makroinstrukcji mogą być użyte kluczowe słowa *#def*, *#reg*, *#label*, *#table* oraz *#data*. W ten sposób zdefiniowane nazwy symboliczne są lokalne, tzn. są znane jedynie w ciele makroinstrukcji. Jeśli lokalna nazwa symboliczna jest taka sama, jak symboliczna nazwa zdefiniowana w głównym programie, bierze się pod uwagę tę lokalną.

Użycie makroinstrukcji w tekście programu jest następujące:

```
pierwsze_makro (Blokada_M1, Uruchom_M1, Wyjscie_M1)
```

Przy użyciu makroinstrukcji zapisuje się więc nazwę makroinstrukcji, a w nawiasach przekazywane są tzw. rzeczywiste parametry. Przy rozwinięciu makroinstrukcji są one zastąpione jej formalnymi parametrami. Ilość parametrów w definicji makroinstrukcji (tj. ilość formalnych parametrów) musi być zgodna z ilością przekazywanych (rzeczywistych parametrów).

Jak jest wykonywana makroinstrukcja?

Jeśli podczas kompilacji kompilator natrafi na nazwę makroinstrukcji, rozwinie ją. Oznacza to, że nazwa makroinstrukcji zostanie zastąpiona przez ciąg instrukcji tworzących ciało makroinstrukcji. Formalne parametry są jednocześnie zastąpione rzeczywistymi.

W przypadku wcześniej zdefiniowanego makra *pierwsze_makro,* będzie program

```
LDC    wejście
pierwsze_makro (a, b, c)
WRC    wyjscie
```

po rozwinięciu przez kompilator makroinstrukcji *pierwsze_makro* wyglądać następująco:

```
LDC    wejście
LD     a            ;tutaj zaczyna się rozwinięcie makra
OR     b
WR     c            ;ostatnia instrukcja rozwinięcia makra
WRC    wyjscie
```

9.12. #mnemo, #mnemoend

Dyrektywy określające wyświetlanie w edytorze schematu przekaźnikowego

Dyrektywa *#mnemo* informuje edytor schematu przekaźnikowego, aby wszelkie instrukcje następujące po tej dyrektywie były wypisane w tekstowej postaci, a nie w formie schematu przekaźnikowego. Powrót do przekaźnikowego wyświetlania następuje po dyrektywie *#mnemoend*.

Ich składnia jest następująca:

```
#mnemo
      :     ;część programu, który będzie zapisany
            ;jako instrukcje
#mnemoend
```

Podczas programowania schematów przekaźnikowych niekiedy konieczne jest wytworzenie takiej sekwencji instrukcji, których interpretacja w przekaźnikowych symbolach nie jest logiczna. Tę część programu można oznaczyć za pomocą pary *#mnemo* / *#mnemoend,* dzięki czemu będzie ona zawsze wyświetlana we swej tekstowej postaci.

Uwaga: Dyrektywy powinny być zapisane parami, co oznacza, że dyrektywa *#mnemo* musi mieć swój zakończeniowy *#mnemoend*.

9.13. #option

Dyrektywa dla nastawiania parametrów kompilatora

Dyrektywa *#option* służy do „rutynowego“ nastawiania parametrów kompilatora. Parametry kompilatora są w ten sposób zachowane wraz z tekstem źródłowym programu. W przypadku, gdy powróci się do programu po pewnym czasie, kompilator jest zawsze nastawiony tak samo, nawet wtedy, gdy dla różnych programów nastawienia mogą się znacznie różnić. Parametry nastawione przez dyrektywy *#option* po kompilacji programu nastawiają opcje w oknie *Opcje | Kompilator*.

Aby dyrektywa *#option* pracowała poprawnie, należy ją używać zaraz po dyrektywie *#program* przed wszystkimi pozostałymi dyrektywami oraz instrukcjami programu.

Użycie dyrektywy *#option* jest następujące:

```
#option   mod = n                ;komentarz
```

gdzie *mod* jest jednym z następujących parametrów:

CPM - typ jednostki centralnej PLC
0 ... typ A
1 ... typ S
2 ... typ M
3 ... typ E
4 ... typ D
5 ... typ B

BlockOut - zewnętrzne blokowanie wyjść PLC
0 ... wyłączone
1 ... aktywne w log.0
2 ... aktywne w log.1

EnableRun - zewnętrzne umożliwienie RUN
0 ... wyłączone
1 ... aktywne w log.0
2 ... aktywne w log.1

AlarmTime - długość cyklu dla ostrzeżenia [ms]

Alarm - długość cyklu dla ostrzeżenia [10 ms]!!, parametr przywrócony z powodu kompatybilności z poprzednimi wersjami, zaleca się jednak używać nastawianie za pomocą AlarmTime

MaxCycleTime - maks. długość cyklu [ms]

MaxCycle - maks. długość cyklu [10 ms]!!, parametr przywrócony z powodu kompatybilności s poprzednimi wersjami, zaleca jednak się używać nastawianie za pomocą MaxCycleTime

RemZone - długość strefy remanentnej
0 ... żaden rejestr nie jest podtrzymywany
≠0 ... ilość podtrzymywanych rejestrów R zaczynając od R0

PlcStart - typ startu PLC po załączeniu zasilania
0 ... ciepły (zawartość podtrzymywanych rejestrów jest zachowana)
1 ... zimny (wszystkie rejestry R są po załączeniu wyzerowane)

ProtTable - chronione tabelki T - ma znaczenie jedynie w przypadku, gdy program użytkownika jest podtrzymywany w pamięci EEPROM
0 ... wyłączone (po załączeniu zasilania PLC będzie z pamięci EEPROM wgrany cały program wraz z tabelami typu T)
1 ... załączone (po załączeniu zasilania PLC będzie z pamięci EEPROM wgrany cały program oprócz tabel typuT)

W pliku xpro.sys można znaleźć symboliczne nazwy dla często używanych wartości w dyrektywie *#option*.

Przykład 9.14

```
#program Demo_option, v1.0
;
#option CPM = CPM1D
#option RemZone = 12
#option PlcStart = 0
;
P 0
      :
E 0
```

Podany przykład pokazuje nastawianie kompilatora dla jednostki centralnej typu D oraz ciepłego restartu po załączeniu zasilania PLC z podtrzymywaniem zawartości rejestrów R0 do R11. Te rejestry mają po załączeniu zasilania taką samą zawartość, jak przed wyłączeniem. Pozostałe rejestry są wyzerowane. Symbolicznej nazwie CPM1D jest w pliku xpro.sys za pomocą dyrektywy *#def* przyporządkowana stała 4.

- - - - - - - - - - -

10. PROCESY UŻYTKOWNIKA

10.1. OGÓLNE ZASADY AKTYWACJI

Procesy użytkownika

Program użytkownika składa się z procesów użytkownika. Teoretycznie może ich być do 65 (P0 do P64), praktycznie bywa ich znacznie mniej. W odróżnieniu od tradycyjnych systemów operacyjnych realnego czasu dla komputerów użytkownik nie ma takiej swobody przy sterowaniu procesami. Procesy są aktywowane w zależności od wcześniej zdefiniowanych reguł. W ramach tych reguł można dodatkowo wpłynąć na aktywowanie większości procesów z poziomu programu użytkownika.

Tab.10.1 Wykaz procesów programu użytkownika i ich przeznaczenie

Procesy	Przeznaczenie
P0	podstawowy proces
P1 do P4	czterofazowo aktywowane procesy
P5 do P9	czasowo aktywowane procesy
P10 do P40	procesy aktywowane przez użytkownika
P41 do P48	przerywające procesy
P50 do P57	obsługa punktu strojenia programu
P60	zbiór podprogramów
P62	ciepły restart
P63	zimny restart
P64	końcowy proces cyklu

Centralne jednostki typu E mają możliwość zaprogramowania jedynie procesu P0.

Schemat aktywowania procesów podany jest na rys.10.1. Cienko obramowane są działania systemu, podwójnie są obramowane procesy użytkownika.

Wykluczenie procesów

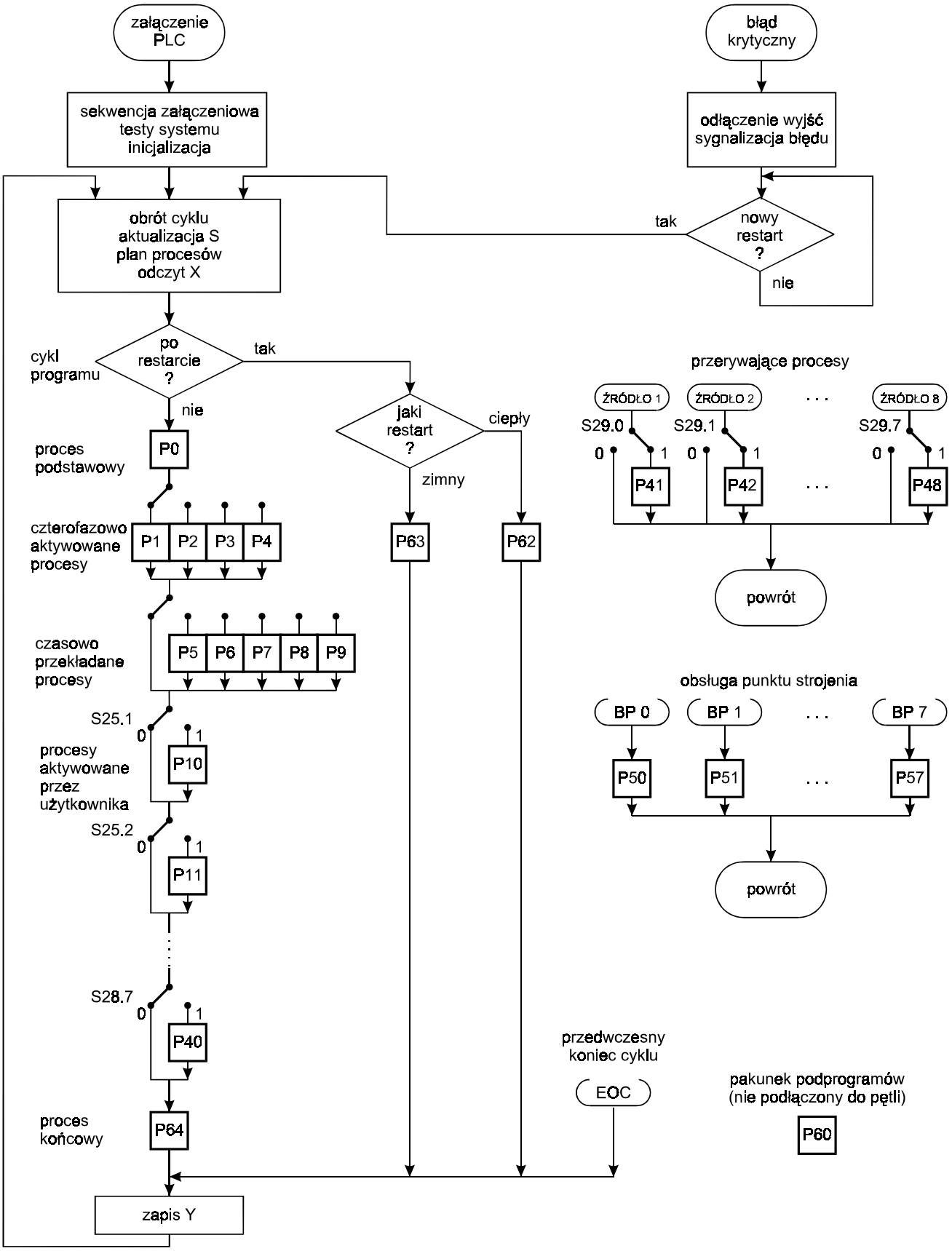
Użytkownik nie musi wykorzystywać wszystkie procesy. Jeśli nadaje mu się klasyczne jednopętlowe sterowanie, można zdefiniować jedynie proces P0. Procesy można wykluczyć na trzy sposoby:

- Proces nie jest zaprogramowany, tzn. nie są użyte instrukcje P i E odpowiedniego procesu. Jedynie procesu P0 nie można w ten sposób wykluczyć, może on jednak być pusty.
- Proces jest pusty, tj. pomiędzy instrukcjami P a E odpowiedniego procesu nie ma żadnej instrukcji. Jego aktywacja przejawi się jedynie jako operacja pusta.
- Aktywacyjna maska procesu jest wyzerowana. Aktywacyjne maski procesów P10 do P48 są zawarte w systemowych rejestrach S25 do S29 (rozdz.5.3.). Proces z wyzerowaną maską aktywacyjną będzie pominięty w następnym cyklu, (lub natychmiast w przypadku przerywającego procesu P41 do P48). Procesy P0 do P9 obsługiwane przez system nie można w ten sposób wykluczyć.

Uwaga: Przy przejściu do wykonywania któregokolwiek z procesów P0 do P40, P62, P63, P64 aktywny akumulator jest wyzerowany (przy przejściu do wykonywania procesu P0 jest zawsze aktywny akumulator A).

Skok pomiędzy procesami

Nie jest zabronione (pomimo, iż nie zaleca się) przekazywanie sterowania z jednego procesu do innego poprzez instrukcję skoku lub wywołania podprogramu. Najbliższa instrukcja E lub ED, EC ze spełnionym warunkiem, zakończy jednak aktualnie aktywowany proces - użytkownik może przechodzić pomiędzy programami dowolnych procesów, systemowy program nie rejestruje tych zmian jako zmiany aktywacji procesów, a detekcję końca procesu traktuje jako powrót z procesu aktywowanego przez systemowy program, a nie jako powrót z wywołania podprogramu. Jeśli na przykład z procesu P0 odskoczy się na etykietę umieszczoną w ostatnim procesie P64, wówczas E64 nie spowoduje obrotu cyklu, ale zamknięcie P0 (tak samo jak E0) i jego powrót do systemowego programu, który wywoła następujący proces użytkownika. Akumulator będzie w tym przypadku wyzerowany dopiero po zamknięciu procesu P0, przejście pomiędzy procesami za pomocą instrukcji skoku lub wywołania nie powoduje więc jakiejkolwiek zmiany wartości w akumulatorze.



Rys.10.1 Schemat aktywowania procesów

Przedwczesny koniec cyklu

Jedynie instrukcja EOC jest zdolna wpłynąć na ciąg procesów zaplanowanych dla aktualnej pętli i bezpośrednio (oraz bezwarunkowo) wykonać obrót cyklu. Może to być wykorzystane np. przy obsłudze przerwania lub dla zapewnienia szybkiej odpowiedzi na sytuacje krytyczne. Podczas planowania procesów dla następnego cyklu po tak wymuszonym obrocie cyklu nie bierze się pod uwagę faktu, że niektóre z pierwotnie zaplanowanych procesów nie były aktywowane - zaczyna się planować od początku.

Przerywające procesy

Którykolwiek z procesów pętli może być przerwany poprzez któryś z przerywających procesów P41 do P48. Instrukcja rozpoczęta w momencie przerwania zostanie dokończona i dopiero po niej jest wykonana pierwsza instrukcja przerywającego procesu. Po jego zakończeniowej instrukcji E (lub ED, EC) kontynuowany jest proces, który był przerwany. Przerywający proces nie zmienia stan żadnego poziomu aktywnego akumulatora.

Czas cyklu przy użyciu przerywających procesów

Czas cyklu wydłuża się o sumę czasów procesów przerywających, które były aktywowane w trakcie jednego cyklu. Dlatego programy obsługi przerwania powinny zawierać jedynie niezbędne instrukcje dla zabezpieczenia szybkiej odpowiedzi na sytuację, która przerwanie wywołała. W przeciwnym razie może dojść do wyraźnego przedłużenia czasu cyklu. Ekstremalnym przypadkiem by było zatrzymanie wykonywania pozostałych procesów, a system by obsługiwał jedynie przerwania. Ten stan jest ograniczony systemowym warunkiem, który ogranicza czas trwania przerywającego procesu do 5 ms. Z tego powodu nie jest możliwe zagłębianie przerwania (ponownego przerwania procesu przerywającego).

Mechanizm przerwania jest bardzo skutecznym aparatem, który umożliwia znaczne skrócenie odpowiedzi systemu na sytuacje krytyczne, musi jednak być stosowany z rozwagą.

10.2. OBRÓT CYKLU

Obrót cyklu

Po sekwencji załączeniowej (patrz rozdz.2.1), po zakończeniu ostatniego z procesów zaplanowanych dla aktualnego cyklu lub po instrukcji EOC, dokonywany jest tzw. obrót cyklu. Jego czas jest zależny od konfiguracji systemu, długości strefy remanentnej oraz zakresu innych funkcji systemowych (patrz suplement).

Podczas obrotu cyklu stan rejestrów Y wysyłany jest do wyjść, aktualizowane są rejestry X według stanu wejść, aktualizowany jest systemowy czas, planowane są procesy dla następnego cyklu, generowane są dodatnie i ujemne zbocza, podejmowana jest decyzja o trybie systemu (tryb RUN - HALT, blokowanie wyjść, typ ewentualnego restartu), nastawiany jest aktualny stan systemowych rejestrów S oraz jest aktywowany i zerowany akumulator A.

10.3. OBSŁUGA RESTARTU - PROCESY P62, P63

Procesy P62, P63

P62 - ciepły restart
P63 - zimny restart

W pierwszym cyklu po restarcie może być aktywowany jeden z procesów P62, P63. Te procesy służą do inicjalizacji zmiennych. Jeśli wykonywany jest jeden z procesów P62, P63, wówczas w danym cyklu nie jest już wykonywany żaden inny proces, nawet P0. Jeśli jest zaprogramowany jedynie jeden z P62, P63, wówczas aktywowany jest zaprogramowany proces bez odróżnienia typu restartu. Jeśli żaden z nich nie jest zaprogramowany, aktywowany jest proces P0. Niezależnie od aktywowania P62, P63 nastawiane są bity S2.3 i S2.4 we swym niezmienionym znaczeniu. Centralne jednostki typów S, D i B mają w trakcie wykonywania procesów P62, P63 wstrzymane aktywowanie procesów przerywających w celu wykluczenia możliwości hazardu, polegającego na pracy z niezinicjalizowanymi strukturami danych.

Uwaga: Do odróżnienia pierwszego cyklu po ponownym starcie (SP = 1 lub zmiana trybu HALT → RUN bez restartu) przeznaczony jest bit S2.6, który jest nastawiony na log.1 wówczas, gdy system był aktualnie uruchomiony bez wykonania restartu.

Jeśli jest nastawiony ciepły restart, zostanie wykonany proces P62. W następnym cyklu zacznie być wykonywany proces P0 i kolejne zaprogramowane procesy. Jeśli nie jest zaprogramowany proces P62, zamiast niego zostanie wykonany proces P63. Jeśli nie jest zaprogramowany żaden z procesów P62, P63, zacznie się wykonywanie procesu P0 i kolejne zaprogramowane procesy.

Jeśli jest nastawiony zimny restart, zostanie wykonany proces P63 a w następnym cyklu zacznie być wykonywany proces P0 i kolejne zaprogramowane procesy. Jeśli nie jest zaprogramowany proces P63, zamiast niego zostanie wykonany proces P62. Jeśli nie jest zaprogramowany ani jeden z procesów P62, P63, zacznie się wykonywanie procesu P0 i kolejne zaprogramowane procesy.

Jeśli jest nastawiony tryb bez restartu, nie zostanie wykonany ani jeden z procesów P62, P63, ale zacznie się wykonywanie procesu P0 i kolejne zaprogramowane procesy.

Przykład 10.1

```
#program restart
#reg byte rejestr
;
P 0
      :
E 0
;
P 62                ;wspólny dla ciepłego i zimnego restartu
      LD    $10
      WR    rejestr       ;początkowa wartość
E 62
```

10.4. PROCESY PĘTLI

Procesy pętli

Z rys.10.1 wynika, że jedynie procesy P0 i P64 są aktywowane w każdym cyklu, procesy P1 do P9 są aktywowane w wybranych cyklach, procesy P10 do P40 aktywuje użytkownik poprzez maski sterujące. Razem procesy te przejawiają się jako różne pętle programu użytkownika, z których każda ma inny czas cyklu. Dlatego ten sposób aktywacji można oznaczyć jako wielopętlowe sterowanie.

10.4.1. Podstawowy proces P0

Proces P0

P0 - początkowy proces każdego cyklu

Podstawowy proces P0 jest obowiązkową częścią struktury programu użytkownika. Nawet gdyby nie było potrzeby użycia procesu P0, należy go zaprogramować (obowiązkowe instrukcje P 0, E 0).

Podstawowy proces P0 jest aktywowany w każdym cyklu jako pierwszy, z wyjątkiem restartu, kiedy to aktywowany jest jedynie jeden z procesów P62 lub P63.

Powinny się w nim znaleźć wszystkie początkowe operacje i układ planowania procesów użytkownika. Ponieważ jest on aktywowany zawsze, powinny w nim być zawarte jedynie te zadania, które wymagają krótkiego czasu odpowiedzi. Nie powinien być zbytecznie zapełniony zadaniami o niższym priorytecie.

Przykład 10.2

```
#program podstawowy
#reg byte wejscie,wyjscie
;
P 0
LD    wejscie
      :
WR    wyjscie
E 0
```

10.4.2. Czterofazowo aktywowane procesy P1, P2, P3, P4

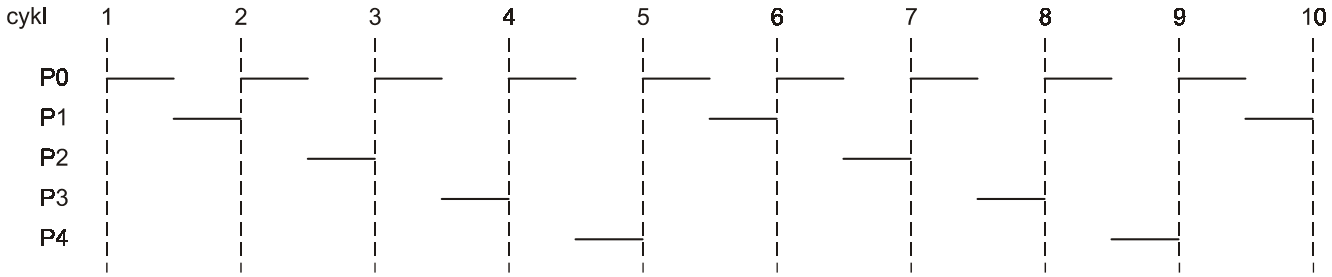
Procesy P1 - P4

P1 - proces uruchomiony w każdym pierwszym z czterech cykli
P2 - proces uruchomiony w każdym drugim z czterech cykli
P3 - proces uruchomiony w każdym trzecim z czterech cykli
P4 - proces uruchomiony w każdym czwartym z czterech cykli

Procesy są cyklicznie zmieniane w kolejności P1, P2, P3, P4, P1, ... (rys.10.2). Ich podstawową właściwością jest to, że w każdym cyklu jest aktywny tylko jeden z tych czterech procesów. To umożliwia zastosowanie tych procesów do programowania sytuacji kolizyjnych, które nie mogą być wykonywane w tym samym cyklu, lub rozłożenie jednego długiego algorytmu na cztery części w celu skrócenia czasu cyklu programu użytkownika i przyspieszenia odpowiedzi PLC. W ten sposób można za pomocą pro-

stych środków dokonać konsekwentnej synchronizacji algorytmu i uniemożliwić stany hazardowe, błędne przejścia i niepożądane stany przejściowe.

Procesy P1 do P4 są aktywowane zawsze po ukończeniu podstawowego procesu P0. Aktywacja procesów P1 do P4 opiera się na stanie licznika cykli w S4. Z tego wynika, że jeśli np. użyje się tylko procesy P1, P2 i P3, wówczas w cyklu, który by odpowiadał procesowi P4, nie będzie aktywowany żaden z procesów tej grupy. W pierwszym cyklu po restarcie systemu jest zawsze aktywowany proces P1.



Rys.10.2 Czasowe przebiegi aktywacji procesów P0 do P4

Przykład 10.3

```
#program czterofazowy
#reg byte wejscie,wyjscie
;
P 0
      LD    wejscie
      :                    ;wykonywane w każdym cyklu
      WR    wyjscie
E 0
;
P 1
:           ;wykonywane w 1., 5., 9., ... cyklu
E 1
;
P 2
:           ;wykonywane w 2., 6., 10., ... cyklu
E 2
;
P 3
:           ;wykonywane w 3., 7., 11., ... cyklu
E 3
;
P 4
:           ;wykonywane w 4., 8., 12., ... cyklu
E 4
```

10.4.3. Czasowo aktywowane procesy P5, P6, P7, P8, P9

Procesy P5 - P9

P5 - proces uruchomiony co 400 ms
P6 - proces uruchomiony co 3,2 s (przesunięcie o 200 ms względem P5)
P7 - proces uruchomiony co 25,6 s (przesunięcie o 400 ms względem P6)
P8 - proces uruchomiony co 204,8 s, tj. 3,4 min. (przesunięcie o 800 ms względem P7)
P9 - proces uruchomiony co 1638,4 s, tj. 27,2 min. (przesunięcie o 1,6 s względem P8)

Procesy P5, P6, P7, P8, P9 są aktywowane zawsze po upłynięciu określonego czasu. Dokładność tego odstępu czasowego jest określona przez czas cyklu. Aktywacje poszczególnych procesów tej grupy są dodatkowo przesunięte względem siebie tak, aby w jednym cyklu był aktywowany maksymalnie jeden z tych procesów (rys.10.3).

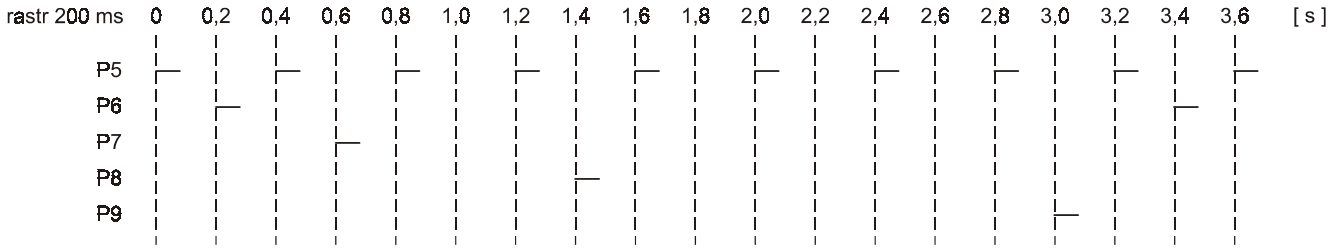
Procesy P5 do P9 są aktywowane po czterofazowo aktywowanych procesach P1 do P4. Aktywacje procesów są sterowane na podstawie liczników jednostek czasowych SW14. Częstość aktywacji procesu o wyższym numerze jest zawsze 8 razy mniejsza niż częstość aktywacji procesu o numerze o jeden niższym (ma 8 razy dłuższy interwał). Warunkiem dla poprawnego aktywowania procesów P5 do P9 jest czas cyklu krótszy niż 200 ms. Przy przekroczeniu tego czasu może dojść do aktywowania kilku procesów tej grupy w jednym cyklu oraz do pominięcia aktywacji procesu P5.

Użycie procesów P5 do P9 jest wygodne zwłaszcza w przypadkach typu:

„kilka razy na sekundę wykonaj...“,

„po kilku sekundach wykonaj...“,
„kilka razy na minutę ...“,
„co kilka minut...“,
„co mniej więcej pół godziny...“.

Wówczas nie należy pracować z układami czasowymi ani rejestrami mierzącymi czas, wystarczy jedynie wstawić zadanie do odpowiedniego procesu.



Rys.10.3 Możliwe momenty aktywacji procesów P5 do P9

Przykład 10.4

```
#program przekladanie
#reg byte wejscie,wyjscie
;
P 0
      LD    wejscie
      :                   ;wykonywane w każdym cyklu
      WR    wyjscie
E 0
;
P 5
:                 ;wykonywane co 400 ms
E 5
;
P 6
:                 ;wykonywane co 3,2 s
E 6
;
P 7
:                 ;wykonywane co 25,6 s
E 7
;
P 8
:                 ;wykonywane co 3,4 min.
E 8
;
P 9
:                 ;wykonywane co 27,2 min.
E 9
```

10.4.4. Procesy aktywowane przez użytkownika P10 do P40

Procesy P10 - P40

P10 do P40- procesy aktywowane przez użytkownika

Procesy P10 do P40 są aktywowane przez użytkownika poprzez nastawienie bitów sterujących w rejestrach systemowych S25 do S28. Po nastawieniu odpowiedniego bitu na log.1 jest w następnym cyklu aktywowany dany proces użytkownika. Przyporządkowanie procesu do jego maski jest następujące: P10 - S25.1, P11 - S25.2, ..., P40 - S28.7. Wartości na pozycjach S25.1 do S28.7 nastawiane są jedynie przez użytkownika, systemowy program ich nie nastawia. Procesy P10 do P40 są więc aktywne dopiero od momentu, kiedy użytkownik je uaktywni, do czasu, kiedy je ponownie deaktywuje. Procesy są aktywowane w kolejności według rys.10.1 (rosnąco według kolejności numeracji), której nie można zmienić w żadnym kierunku. Po restarcie PLC są wszystkie bity wyzerowane a procesy P10 do P40 nie są aktywowane.

	.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

Maski S24.0 do S25.0 są przyporządkowane do procesów P1 do P9 i obsługiwane przez systemowy program podczas obrotu cyklu (w trakcie cyklu nie są zmieniane). Wartości maski wskazują procesy zaplanowane do aktywowania dla bieżącego cyklu. Zmieniając maski z poziomu programu nie można wpłynąć na aktywowanie procesów P1 do P9.

Zastosowanie procesów P10 do P40 jest szerokie. Za ich pomocą można przejrzyście realizować warunkowe wykonywanie różnych czynności, gdy poszczególne czynności są programowane w odpowiednich procesach, w podstawowym procesie P0 natomiast są sprawdzane warunki, na podstawie których są uruchomione poszczególne procesy. Od użytkownika zależy, jakie znaczenie przyporządkuje procesom P10 do P40 oraz w zależności od jakich reguł będzie je aktywować i na jaki czas (od jednorazowych aktywacji aż po długoczasowy tryb pracy).

Procesy P10 do P40 są aktywowane po procesach przekładanych w czasie P5 do P9 rosnąco, zgodnie z numeracją. Zmiana odpowiedniego bitu sterującego nastąpi dopiero w następnym cyklu.

Przykład 10.5

```
#program uzytkownik
#reg bit wejscie0, wejscie1, wejscie2, wejscie3,test1,test2,test3
;
P 0
      LD    wejscie0
      WR    S25.1        ;P10 aktywny przy wejscie0 = log.1
      LD    wejscie1
      LET   test1        ;P11 aktywowany jednorazowo przy
      SET   S25.2        ;dodatnim zboczu wejscie1
      LD    wejscie2
      LET   test2        ;P12 aktywowany przy dodatnim
      SET   S25.3        ;zboczu wejscie2
      LD    wejscie3
      LET   test3        ;P12 deaktywowany przy dodatnim
      RES   S25.3        ;zboczu wejscie3
E 0
;
P 10
      :
E 10
;

P 11
      :
      LD    0
      WR    S25.2  ;P11 jest jednorazowy, sam się deaktywuje
E 11
;
P 12
      :
E 12
```

10.4.5. Końcowy proces cyklu P64

Proces P64

P64 - proces umieszczony zawsze na końcu cyklu

Proces P64 jest wykonywany zawsze jako ostatni proces w cyklu. Stosowany jest do programowania algorytmów, które należy wykonać dopiero po wykonaniu procesów P1 do P40. Jego zaprogramowanie nie jest obowiązkowe.

Przykład 10.6

```
#program zakonczenie
;
P 0
:           ;czynności wykonywane na początku każdego cyklu
E 0
;
P 5
:           ;wykonywane w środku cyklu co 400 ms
E 5
;
P 64
:           ;wykonywane na końcu każdego cyklu
E 64
```

10.5. PROCESY PRZERYWAJĄCE

Działanie przerywających procesów

Którykolwiek z procesów pętli może być przerwany na dowolnej instrukcji. Systemowy program zapewnia dokończenie rozpoczętej instrukcji, zapamiętuje stan aktywnego akumulatora, systemowych rejestrów S0, S1 i przekazuje sterowanie na początek procesu przerywającego. Końcowa instrukcja tego procesu (E lub ED, EC przy spełnionym warunku) zwraca niezmieniony stan akumulatora oraz powraca do miejsca w programie, w którym nastąpiło przerwanie. Przerywający proces nie przeprowadza fazy początku ani końca cyklu. Z tego wynika, że nie można przenosić parametrów z przerywanego procesu do procesu przerywającego w aktywnym akumulatorze.

Ograniczenie czasu trwania procesów przerywających

Przerywający proces powinien być jak najkrótszy i nie może przekroczyć 5 ms. W przeciwnym razie nastąpi zatrzymanie PLC z powodu błędu krytycznego 80 31 PC PC. Przerywające procesy ogólnie służą do obsługi sytuacji krytycznych oraz szybkich procesów (stanów). Nie powinny pracować z wejściami X (mogą być przestarzałe), ale z bezpośrednimi wejściami U oraz powinny bezpośrednio nastawiać wartości jednostek wyjściowych (patrz rozdz.6. i odpowiednie dodatki).

Nie można zagłębiać przerwania

Przerywający proces nie może być ponownie przerwany (nie jest możliwe zagłębianie przerwania). Jeśli w trakcie przerywającego procesu nastąpiło następne żądanie przerwania, zostanie ono zapamiętane i odpowiedni proces przerywający będzie wywołany. Jeśli takich wymagań nadejdzie kilka, są one sumowane logicznie, nie co do ilości, i respektowany jest następujący priorytet przy porządkowaniu przerywających procesów:

Priorytety przerwania

1. priorytet - przerwanie podczas błędu (P43)
2. priorytet - przerwanie od wejść (P42)
3. priorytet - przerwanie od czasu (P41)
4. i pozostałe priorytety - inne źródła przerwania (patrz suplement)

Zakazanie aktywacji przerywającego procesu

Centralne jednostki typów S, D i B umożliwiają przejściowe zakazanie procesu przerywającego. Po restarcie są bity w rejestrze S29 odpowiadające zaprogramowanym procesom nastawione na log.1. Poprzez wyzerowanie któregokolwiek bitu osiągnie się to, że odpowiedni proces przerywający natychmiast przestanie być wywoływany (nie czeka się na obrót cyklu). Po ponownym nastawieniu tego bitu odpowiedni proces zacznie od tego momentu reagować na przerwanie. Jeśli wymaganie na przerwanie nastąpiło w momencie, gdy odpowiedni proces był odłączony, wówczas wymaganie to jest stracone. Nie można więc wykorzystać tej możliwości do chwilowego wstrzymania procesu przerywającego.

Po restarcie, w trakcie wykonywania procesów P62 i P63, przerywające procesy są chwilowo zakazane w celu uniknięcia hazardu na skutek nezinicjalizowanych wartości. System je odblokuje tuż przed pierwszym wejściem do procesu P0. Jeśli więc nie chcemy, aby doszło do przerwania, należy zakazać przerywający proces poprzez wyzerowanie odpowiedniego bitu S29 już w procesie P62, ew. P63.

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

10.5.1. Przerwanie od czasu P41

Proces P41

Proces P41 jest aktywowany regularnie co 10 ms. Stosowany jest do obsługi zdarzeń wymagających czasu reakcji krótszego niż czas cyklu. Jeśli pracuje się z wejściami i wyjściami, należy pracować bezpośrednio z fizycznymi adresami, a nie z ich obrazami w pamięci, które w trakcie cyklu nie zmieniają się.

Proces P41 ma 3. najwyższy priorytet procesów przerywających, który zostanie uwzględniony przy jednoczesnych wymogach na kilka procesów przerywających.

Przykład 10.7

```
#program czas
#def   wyjscie      Y0
#reg bit  stan
#reg long licznik
;
P 0
       :
E 0
;
P 41
      LD     U$9000        ;fizyczny adres wejścia X0
      AND    %1000         ;wejście X0.3
```

```
      LET   stan          ;test dodatniego zbocza
EC
INR   licznik             ;dodanie impulsu
E 41
```

10.5.2. Przerwania od wejść P42

Proces P42

Proces P42 jest aktywowany przy zmianie wejścia przerywającego. Konkretne rozwiązanie zależy od typu PLC.

Proces P42 stosuje się do obsługi zdarzeń wymagających krótszego czasu reakcji, niż czas cyklu. Należy pracować bezpośrednio z fizycznymi adresami wejść i wyjść, a nie z ich obrazami w pamięci, które w trakcie cyklu nie zmieniają się.

Proces P42 ma 2. najwyższy priorytet procesów przerywających, który zostanie uwzględniony w momencie wymagania na kilka przerywających procesów.

Przykład 10.8

```
#program wejscie
#def   wyjscie      Y0
#reg bit  stan,pomoc
;
P 0
      :
E 0
;
P 42
      LD    U$9000        ;fizyczny adres wejścia X0
      AND   1             ;wejście X0.0
      LET   stan          ;test dodatniego zbocza
      SET   pomoc         ;wartość wyjścia
      LD    pomoc
      AND   %100          ;wyjście Y0.2
      LD    wyjscie
      AND   %11111011     ;zmazanie starej wartości Y0.2
      OR                  ;dodanie nowej wartości Y0.2
      WR    wyjscie       ;kopia do pamięci
      WR    U$9100        ;zapis na adres fizyczny
E 42
```

Aktywacja P42 w NS950

W PLC NS950 proces P42 jest aktywowany przy jakiejkolwiek zmianie wejść 0.0 tych jednostek wejściowych, które za pomocą złączki mają uaktywnione przerwania (jednostki IB-36 do IB-47).

Jeśli kilka jednostek wejściowych ma aktywne przerwanie i wymagane jest odróżnienie tych przerwań, to należy tego dokonać programowo. W programie użytkownika w każdym cyklu wykonuje się kopię obrazów X tych jednostek, od których można oczekiwać przerwania. Następnie w procesie przerywającym P42 poprzez porównanie bezpośrednich wejść U z kopiami odpowiednich obrazów X uzyska się informację, na której jednostce nastąpiła zmiana wartości na najniższym bicie, wywołująca przerwanie.

Kopiowanie obrazów X jest konieczne dlatego, że jeśli któraś jednostka wywoła przerwanie w trakcie obrotu cyklu, może z wielkim prawdopodobieństwem dojść do sytuacji, w której do obrazów X w pamięci zostanie zapisany nowy stan wejść tej jednostki, ale proces przerywający będzie wywołany dopiero po rozpoczęciu nowego cyklu. Temu czasowemu hazardowi można zapobiec tak, że bezpośrednie wejścia porównuje się z obrazami X o jeden cykl starszymi. W tym przypadku jednak minimalny czas pomiędzy przerwaniami od tej samej jednostki musi być dłuższy niż czas najdłuższego cyklu.

Aktywacja P42 w TC500 i TC600

W PLC TC500 i TC600 proces P42 jest aktywowany przy jakiejkolwiek zmianie poziomu logicznego wejść DI0 do DI3. Aktywowanie procesu P42 umożliwiają warianty TC503 do TC507, TC513 do TC517, TC603 do TC607.

Szczegóły dotyczące obsługi procesu przerywającego P42 są podane w podręcznikach Wyposażenie techniczne sterowników programowalnych TECOMAT TC500 TXV 138 07.01 oraz Wyposażenie techniczne sterowników programowalnych TECOMAT TC600 TXV 138 08.01.

10.5.3. Przerwania od błędu P43

Proces P43

Proces P43 jest aktywowany przy pojawieniu się błędu, który nie spowoduje zatrzymania pracy PLC (kod błędu w rejestrze S34).

Proces P43 stosuje się do zbiorowej obsługi stanów awaryjnych.

Proces P43 ma najwyšzy priorytet z procesów przerywających, który zostanie uwzględniony przy wymaganiu na kilka procesów przerywających.

Przykład 10.9

```
#program blad
#reg bit  blad
#reg byte a,b,c
;
P 0
      LD     a
      DIV    b ;przy dzieleniu zerem zostanie wywołany proces P43
      WR     c
E 0
;
P 43
      LD     1
      WR     blad   ;nastawianie wskaźnika błędu
E 43
```

Opisy procesów przerywających dla poszczególnych typów PLC są podane w suplemencie na końcu podręcznika.

10.5.4. Przerwanie od hw liczników lub od czujnika inkrementalnego P44

Proces P44

W PLC TC500 i TC600 proces P44 jest aktywowany przy osiągnięciu progu lub przy przepełnieniu zakresu hw licznika. Aktywowanie procesu P44 umożliwiają warianty TC503 do TC507, TC513 do TC517, TC603 do TC607.

Szczegóły dotyczące obsługi procesu przerywającego P44 są podane w podręcznikach Wyposażenie techniczne sterowników programowalnych TECOMAT TC500 TXV 138 07.01 oraz Wyposażenie techniczne sterowników programowalnych TECOMAT TC600 TXV 138 08.01.

Proces P44 ma 4. priorytet kolejności przerywających procesów, który ma znaczenie przy jednoczesnym wywołaniu kilku procesów przerywających.

10.5.5. Przerwanie od portu szeregowego CH2 P45

Proces P45

Jeśli port szeregowy CH2 nastawiony jest na tryb uni, odbiór informacji wywoła proces P45. Bezpośrednio przed uruchomieniem tego procesu do strefy odbiorczej zapisane są odebrane dane. Dochodzi tutaj do zmiany danych w trakcie cyklu programu użytkownika, czyli należy obsłużyć te dane w procesach pętli programowej.

Proces P45 służy przede wszystkim do natychmiastowej obsługi mniejszej ilości danych. Przy większej ilości danych lub jeśli nie jest wymagana ich natychmiastowa obsługa, pozostawia się obsługę portu szeregowego w procesach pętli programowej. Odebrane dane będą wówczas przekazane do rejestrów zawsze w trakcie obrotu cyklu.

Proces P45 ma 5. priorytet kolejności przerywających procesów, który ma znaczenie przy jednoczesnym wywołaniu kilku procesów przerywających.

10.6. OBSŁUGA PUNKTU STROJENIA (BREAK POINT) - PROCESY P50 DO P57

Procesy P50 - P57

Po wykonaniu instrukcji BP 0 do BP 7 sterowanie jest przekazane do procesu P50 - P57 (ostatnia cyfra w numerze procesu jest zgodna z operandem instrukcji BP). W tym procesie można wówczas przetwarzać informacje z danego miejsca programu. Przy wejściu do procesu P5x zachowany jest stan akumulatora. Po zakończeniu procesu, w akumulatorze i rejestrach S0 i S1 przywrócone są pierwotne wartości. Ta funkcja ułatwia strojenie programu użytkownika bez nieczytelnych ingerencji w program.

Przykład 10.10

```
#program punkt_strojenia
#reg word strojenie[128],pomoc
;
P 0
      :
BP    0            ;wstawiona instrukcja strojenia
      :
BP    0            ;wstawiona instrukcja strojenia
      :
```

```
E 0
;
P 50
      WR    pomoc  ;zapamiętanie wartości szczytu akumulatora
      LD    127          ;granica tabeli
      LD    index
      LD    pomoc
      WTB   strojenie
E 50
```

10.7. PAKUNEK PODPROGRAMÓW P60

Proces P60

Ten proces nie jest aktywowany z programu systemowego i służy jedynie do gromadzenia podprogramów wywoływanych z różnych procesów.

Jeśli podprogramy są umieszczone wewnątrz aktywnego procesu, wówczas muszą być przeskakiwane, ponieważ nie mogą leżeć ani na początku, ani na końcu procesu. Wymaga to dodatkowo instrukcji JMP i etykiet L, powodując obniżenie przejrzystości programu.

Przykład 10.11

```
#program pakunek
;
P 0
      :
CAL   podprogram   ;wywołanie podprogramu
      :
E 0
;
P 10
      :
CAL   podprogram   ;wywołanie podprogramu
      :
E 10
;
P 60
podprogram:
      :
RET
E 60
```

11. ZBIÓR INSTRUKCJI

Jednostki centralne PLC TECOMAT danej grupy posiadają różne zbiory instrukcji w zależności od mocy jednostki centralnej.

Redukowany zbiór instrukcji

Centralne jednostki serii E zawierają redukowany zbiór instrukcji, którego częścią są:

- bitowe operacje logiczne
- podstawowe operacje liczników i układów czasowych
- podstawowe instrukcje organizacyjne i skoki w programie
- porównania w zakresie word
- jednopętlowe sterowanie

Standardowy zbiór instrukcji

Centralne jednostki serii M i S posiadają standardowy zbiór instrukcji, zawierający w porównaniu do redukowanego dodatkowo:

- logiczne operacje w zakresie byte i word
- rozszerzone operacje liczników, układów czasowych, rejestrów przesuwnych
- arytmetyczne instrukcje, konwersje i porównania w zakresie word
- rozszerzone instrukcje organizacyjne, skoki w programach
- tabelkowe instrukcje na tabelach w pamięci, które umożliwiają optymalną realizację nawet bardzo skomplikowanych kombinacyjnych i sekwencyjnych bloków funkcyjnych, dekoderów, sterowników czasowych i sekwencyjnych, generatorów sekwencyjnych, oraz ułatwiają realizację funkcji diagnostycznych, rozpoznanie stanów krytycznych, sekwencyjny zapis zdarzeń, protokoły o procesie, diagnostyczne komunikaty typu „black box“ (czarna skrzynka)
- tabelkowe instrukcje pracujące z obszarem zmiennych umożliwiają operowanie ze zmiennymi indeksowanymi, realizację linii opóźniającej, długich rejestrów przesuwnych, konwersje na kod „1 z n“, wybór zmiennych, sterowniki krokowe, zapis zdarzeń oraz różne struktury akumulatorowe
- instrukcje sterownika sekwencyjnego
- instrukcje realizujące zbiór operacji logicznych, wraz z wyliczeniem ilości jedynkowych bitów w operandzie typu word - można w ten sposób łatwo realizować majorytet oraz ogólne funkcje progowe, funkcje parzystości (MOD 2) oraz dowolne funkcje symetryczne
- system posiada 8 akumulatorów oraz instrukcje do ich przełączania - stosowane w celu przekazywania kilku parametrów pomiędzy funkcjami, które nie następują bezpośrednio po sobie, zapis aktualnego stanu akumulatora, itp.
- wygodna jest automatyczna konwersja długości operandów oraz wyników częściowych przy kombinacji instrukcji bitowych, bajtowych i wordowych lub instrukcji logicznych z arytmetycznymi
- wygodnym narzędziem jest zbiór zmiennych systemowych, w których realizowany jest systemowy czas, systemowe jednostki czasowe oraz ich zbocza, zmienne komunikacyjne, zmienne wskaźnikowe i rozkazowe, komunikaty systemowe
- do skrócenia czasu odpowiedzi oraz do łatwiejszego programowania służą procesy przerywające
- instrukcje użytkownika USI realizują w sposób optymalny (na poziomie instrukcji mikroprocesora) skomplikowane zadania (specjalne komunikacje, regulacja, czasowo krytyczne zadania)

Rozszerzony zbiór instrukcji

Centralne jednostki serii D i B zawierają rozszerzony zbiór instrukcji, który w odróżnieniu od standardowego posiada dodatkowo:

- logiczne operacje w zakresie long
- arytmetyczne instrukcje, konwersje i porównania w zakresie long
- skoki warunkowe w zależności od wskaźników porównania
- arytmetyczne instrukcje w formacie zmiennoprzecinkowym (floating point)
- rozszerzone tabelkowe instrukcje na tabelach o wielkim zakresie
- tabelkowe instrukcje ze strukturalnym dostępem
- instrukcję regulatora PID

Kompletny opis zbioru instrukcji podany jest w podręczniku *Zbiór instrukcji PLC TECOMAT*, nr kat. TXV 001 05.01.

11.1. WYKAZ INSTRUKCJI WRAZ Z DOPUSZCZALNYMI OPERANDAMI

Wykaz użytych symboli operandów:

Z - strefa pamięci X, Y, S, R
D - dane
- stała
U - fizyczny adres peryferyjny jednostki
T - tabelki
A - bez operandu (pracuje jedynie z akumulatorem)
Ln - etykieta numer n
n - parametr liczbowy
m - parametr literowy

Instrukcje do odczytu i zapisu danych

Mnemokod	Typ operandu					Znaczenie instrukcji
	bit	byte	word	long	float	
LD	ZD	ZD U	ZD#U	ZD	ZD	Odczyt danych bezpośrednich
LDL				#	#	Odczyt danych bezpośrednich
LDC	ZD	ZD	ZD#	ZD		Odczyt negowanych danych
WR	Z	Z U	Z U	Z	Z	Zapis danych bezpośrednich
WRC	Z	Z	Z	Z		Zapis negowanych danych
WRA		Z	Z	Z		Zapis danych bezpośrednich z alternacją
PUT	Z	Z	Z	Z	Z	Warunkowy zapis danych

Logiczne instrukcje

Mnemokod	Typ operandu				Znaczenie instrukcji
	bit	byte	word	long	
AND	ZD	ZD	ZD#A		AND z bezpośrednim operandem
ANL				#A	AND z bezpośrednim operandem
ANC	ZD	ZD	ZD		AND z negowanym operandem
OR	ZD	ZD	ZD#A		OR z bezpośrednim operandem
ORL				#A	OR z bezpośrednim operandem
ORC	ZD	ZD	ZD		OR z negowanym operandem
XOR	ZD	ZD	ZD#A		XOR z bezpośrednim operandem
XOL				#A	XOR z bezpośrednim operandem
XOC	ZD	ZD	ZD		XOR z negowanym operandem
SET	Z	Z	Z		Warunkowe nastawianie
RES	Z	Z	Z		Warunkowe zerowanie
LET	Z	Z	Z		Impuls od dodatniego zbocza
BET	Z	Z	Z		Impuls od dowolnego zbocza
NEG			A		Negacja szczytu akumulatora
NGL				A	Negacja szczytu akumulatora

Liczniki, rejestry przesuwne, układy czasowe, sterownik krokowy

Mnemokod	Operand	Znaczenie instrukcji
CTU	RW	Licznik wprzód
CTD	RW	Licznik wstecz
CNT	RW	Dwukierunkowy licznik
SFL	RW	Rejestr przesuwny w lewo
SFR	RW	Rejestr przesuwny w prawo
TON	RW *	Układ czasowy (opóźnione załączenie)
TOF	RW *	Układ czasowy (opóźnione odłączenie)
RTO	RW *	Całkujący układ czasowy, miernik czasu
IMP	RW *	Układ czasowy - generator impulsu o zadanej długości
STE	RW	Sterownik krokowy (stepper)

* Każdy z układów czasowych może być programowany z jednostką inkrementu:
.0 - 10ms; .1 - 100ms; .2 - 1s; .3 - 10s

Instrukcje aritmetyczne

Mnemokod	Typ operandu			Znaczenie instrukcji
	byte	word	long	
ADD		ZD#A		Dodawanie z przeniesieniem
ADX	ZD	ZD	ZD	Dodawanie
ADL			#A	Dodawanie
SUB		ZD#A		Odejmowanie z przeniesieniem
SUX	ZD	ZD	ZD	Odejmowanie
SUL			#A	Odejmowanie
MUL	ZD#A			Mnożenie (byte x byte = word)
MUD		ZD#A		Mnożenie (word x word = long)
DIV	ZD#A			Dzielenie (byte / byte = byte)
DID		ZD#A		Dzielenie (long / word = long)
INR	Z	Z A	Z	Inkrementacja (+ 1)
DCR	Z	Z A	Z	Dekrementacja (– 1)
ROL n		A		Rotacja liczby w lewo n-razy
ROR n		A		Rotacja liczby w prawo n-razy
EQ		ZD#A		Porównanie (równość)
LT		ZD#A		Porównanie (mniejsze niż)
GT		ZD#A		Porównanie (większe niż)
CMP	ZD	ZD#A	ZD	Porównanie
CML			#A	Porównanie
BIN		A		Konwersja liczby z BCD formatu na binarny
BIL			A	Konwersja liczby z BCD formatu na binarny
BCD		A		Konwersja liczby z binarnego formatu na BCD
BCL			A	Konwersja liczby z binarnego formatu na BCD
BAS		A		Konwersja liczby z binarnego formatu na ASCII
ASB		A		Konwersja liczby z ASCII na format binarny

Operacje na akumulatorach

Mnemokod	Operand	Znaczenie instrukcji
STK	A	Złożenie logicznych wartości 8 poziomów akumulatora do A0
SWP	A	Zamiana górnego i dolnego bajtu w A0
SWL	A	Zamiana warstw A0 i A1
FLG	A	Logiczny AND wszystkich bitów oraz poprzeczne funkcje bajtów A0 w S1
POP	n	Przesunięcie (rotacja) akumulatora wstecz o n poziomów
NXT		Aktywacja następnego akumulatora w szeregu
PRV		Aktywacja poprzedniego akumulatora w szeregu
CHG	m	Aktywacja wybranego akumulatora (m oznacza A do H, ew. AS do HS)
LAC	m	Odczyt wartości ze szczytu wybranego akumulatora (m oznacza A do H)
WAC	m	Zapis wartości na szczyt wybranego akumulatora (m oznacza A do H)

Instrukcje skoku i odwołania

Mnemokod	Operand	Znaczenie instrukcji
JMP	Ln	Bezwarunkowy skok
JMD	Ln	Skok uwarunkowany niezerową wartością wyniku
JMC	Ln	Skok uwarunkowany zerową wartością wyniku
JMI	A	Skok na cel pośredni
JC	Ln	Skok uwarunkowany niezerową wartością wskaźnika przeniesienia CO
JNC	Ln	Skok uwarunkowany zerową wartością wskaźnika przeniesienia CO
JS	Ln	Skok uwarunkowany niezerową wartością wskaźnika S1.0
JNS	Ln	Skok uwarunkowany zerową wartością wskaźnika S1.0
JZ	Ln	Skok uwarunkowany niezerową wartością wskaźnika równości ZR
JNZ	Ln	Skok uwarunkowany zerową wartością wskaźnika równości ZR
CAL	Ln	Bezwarunkowe wywołanie podprogramu
CAD	Ln	Wywołanie podprogramu uwarunkowane niezerową wartością wyniku
CAC	Ln	Wywołanie podprogramu uwarunkowane zerową wartością wyniku
CAI	A	Wywołanie podprogramu z adresem pośrednim
RET		Bezwarunkowy powrót z podprogramu
RED		Powrót z podprogramu uwarunkowany niezerową wartością wyniku
REC		Powrót z podprogramu uwarunkowany zerową wartością wyniku
L	n	Etykieta n (cel skoków i wywołania podprogramów)

Organizacyjne instrukcje

Mnemokod	Operand	Znaczenie instrukcji
P	n	Początek procesu
E	n	Bezwarunkowy koniec procesu
ED		Koniec procesu przy niezerowym wyniku
EC		Koniec procesu przy zerowym wyniku
EOC		Koniec cyklu
NOP	n	Pusta operacja
BP	n	Punkt strojenia
SEQ	Ln	Warunkowe przerwanie procesu

Tabelkowe instrukcje

Mnemokod	Typ operandu			Znaczenie instrukcji
	bit	byte	word	
LTB	ZDT	ZDT	ZDT	Odczyt pozycji z tabelki
WTB	Z T	Z T	Z T	Zapis pozycji do tabelki
LMS			T	Sekwencyjny odczyt pozycji
WMS			T	Sekwencyjny zapis pozycji
FTB	ZDT	ZDT	ZDT	Szukanie pozycji w tabelce
FTM		ZDT	ZDT	Szukanie części pozycji w tabelce
FTS		ZDT	ZDT	Przyporządkowanie pozycji w zależności od tabelki

Arytmetyczne instrukcje zmiennoprzecinkowe

Mnemokod	Typ operandu float	Znaczenie instrukcji
ADF	ZD#A	Dodawanie
SUF	ZD#A	Odejmowanie
MUF	ZD#A	Mnożenie
DIF	ZD#A	Dzielenie
CMF	ZD#A	Porównanie
CEI	A	Zaokrąglenie w górę
FLO	A	Zaokrąglenie w dół
ABS	A	Wartość bezwzględna (moduł)
LOG	A	Logarytm dziesiętny
LN	A	Logarytm naturalny
EXP	A	Funkcja wykładnicza
POW	A	Ogólna potęga
SQR	A	Pierwiastek
SIN	A	Sinus
ASN	A	Arc sinus
COS	A	Cosinus
ACS	A	Arc cosinus
TAN	A	Tangens
ATN	A	Arc tangens
HYP	A	Euklidowska odległość
UWF	A	Konwersja word bez znaku na float
IWF	A	Konwersja word ze znakiem na float
ULF	A	Konwersja long bez znaku na float
ILF	A	Konwersja long ze znakiem na float
UFW	A	Konwersja float na word bez znaku
IFW	A	Konwersja float na word ze znakiem
UFL	A	Konwersja float na long bez znaku
IFL	A	Konwersja float na long ze znakiem
STF	A	Konwersja łańcucha ASCII znaków na float
FST	A	Konwersja float na łańcuch ASCII znaków

Blokowe operacje

Mnemokod	Operand	Znaczenie instrukcji
SRC	ZDT	Specyfikacja źródła danych dla przesunięcia
MOV	Z T	Przesunięcie bloku danych
MTN	A	Przesunięcie tabelki do pamięci
MNT	A	Napełnienie tabelki z pamięci
FIL	Z	Napełnienie bloku wartością stałą

Operacje ze strukturalnymi tabelami

Mnemokod	Operand	Znaczenie instrukcji
LDS	A	Odczyt pozycji ze strukturalnej tabelki T
WRS	A	Zapis pozycji do strukturalnej tabelki T
FIS	A	Napełnianie pozycji strukturalnej tabelki w pamięci
FIT	A	Napełnianie pozycji strukturalnej tabelki T
FNS	A	Szukanie pozycji strukturalnej tabelki w pamięci
FNT	A	Szukanie pozycji strukturalnej tabelki T

Instrukcja regulatora PID

Mnemokod	Operand	Znaczenie instrukcji
CNV	A	Konwersja i obsługa danych z jednostek analogowych
PID	A	PID regulator

Instrukcja użytkownika

Mnemokod	Znaczenie instrukcji
USI	Instrukcja użytkownika

12. INSTRUKCJA UŻYTKOWNIKA

Instrukcja użytkownika USI

Instrukcja użytkownika USI jest instrukcją zdefiniowaną przez użytkownika PLC, przeznaczoną do takich operacji, których realizacja za pomocą pozostałych instrukcji PLC byłaby trudna lub niemożliwa. Oznacza to, że zbiór instrukcji PLC TECOMAT nie jest zamknięty i może być uzupełniany nowymi instrukcjami bez konieczności zmiany systemowego softwaru jednostek centralnych.

Instrukcja użytkownika jest stosowana we wszystkich jednostkach centralnych oprócz serii E.

12.1. SPOSÓB UŻYCIA USI W PROGRAMIE UŻYTKOWNIKA

Deklaracja #usi

Instrukcje USI przed zastosowaniem w programie wymagają definicji, która zawiera informacje o tym, w którym folderze i w którym pliku jest zapisany kod binarny instrukcji USI. Do tego celu w programie xPRO służy dyrektywa *#usi*. Jej składnia jest następująca:

```
#usi [index,] [nazwa_instrukcji] = nazwa_pliku
```

- index - nieobowiązkowy numer, który nastawia indeks wytwarzanej instrukcji. Po wymuszeniu indeksu instrukcji poprzez podanie tego numeru, następuje dla kolejnych zdefiniowanych instrukcji przyporządkowanie indeksów w rosnącej kolejności, zaczynając od numeru index+1. Jeśli pozycja index jest pominięta, wówczas kolejnym instrukcjom jest automatycznie przydzielony rosnący indeks.
- nazwa_instrukcji - nieobowiązkowa symboliczna nazwa instrukcji USI
- nazwa_pliku - nazwa binarnego pliku na dysku, który zawiera kod instrukcji użytkownika (może zawierać całą ścieżkę dostępu)

Używanie instrukcji jest zgodne ze standardowymi instrukcjami PLC.

Przykład 12.1

```
#program Test_USI, v1.0
;
#usi UserInstr = instfile.uid    ;definicja USI
;
P 0
        :
       USI UserInstr       ;użycie USI w programie
        :
E 0
```

12.2. USI DLA POSZCZEGÓLNYCH SERII CENTRALNYCH JEDNOSTEK

Maszynowe kody instrukcji USI różnią się w zależności od typu CPU

Maszynowe kody instrukcji USI nie są przenośne pomiędzy poszczególnymi typami jednostek centralnych z powodu odmiennych procesorów oraz odmiennego mapowania pamięci systemowej.

Częścią każdego kodu maszynowego instrukcji USI jest oznaczenie serii centralnej jednostki, dla której jest kod przeznaczony. Jeśli przez pomyłkę zostanie uruchomiona instrukcja USI na innej jednostce centralnej, niż jest przeznaczona, jednostka ogłosi błąd instrukcji użytkownika ($80 17 PC PC) i program jest zatrzymany.

Na dyskietce dostarczanej z każdym PLC znajdują się instrukcje USI wraz z opisem, które można wykorzystywać we własnych programach.

Rozszerzenia plików z kodem źródłowym instrukcji USI

Pliki zawierające kody maszynowe instrukcji USI mają rozszerzenia .ui-, gdzie ostatnia litera rozszerzenia podaje typ centralnej jednostki, dla której jest kod przeznaczony (patrz tab.12.1). Pliki z rozszerzeniem .udl są maszynowymi kodami instrukcji USI dla symulatora PLC, który jest zintegrowany w programie xPRO v2.1.

Praca z instrukcjami USI jest wspierana przez kompilator xPRO, który dodatkowo umożliwia automatyczne przyporządkowanie rozszerzenia pliku binarnego w deklaracjach *#usi*. W praktyce oznacza to, że jeśli w deklaracji *#usi* w nazwie pliku nie jest podane rozszerzenie, zostanie ono automatycznie dobrane w zależności od tego, dla jakiej centralnej jednostki jest kod generowany.

Tab.12.1 Lista rozszerzeń plików z kodem instrukcji USI

Seria centralnych jednostek	Rozszerzenie pliku
B	.uib
D	.uid
S	.uis
M	.uim
E	nie umożliwia
symulator PLC w xPRO v2.1	.udl

12.3. WYTWORZENIE WŁASNEJ USI

Instrukcje USI są programowane w języku C

Deklaracji USI dokonuje się w języku C. Podczas programowania zaleca się stosować publikację *The C Programming Language* autorów Brian W. Kernighan i Dennis M. Ritchie, wydanej przez Prentice-Hall Inc. Dla zapisu funkcji istnieje nagłówek, który opisuje i udostępnia akumulator oraz pamięć PLC. Wejściowe parametry, tak samo jak wyjścia funkcji USI, mogą być umieszczone w dowolnym miejscu tych struktur.

Funkcja USI musi być przez użytkownika zaprogramowana następująco :

```
#include "usi.h"   /* plik odpowiadający użytej serii CPU */

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

Z podanej deklaracji USI wynika, że w funkcji USI są za pomocą systemowego programu PLC przekazywane wskaźniki na struktury *notePLC* (pamięć PLC) oraz *accPLC* (akumulator PLC). Deklaracja tych struktur zawiera plik usi.h, którego wypis jest podany w rozdz.12.5.

W przypadku, gdy program dla instrukcji USI składa się z kilku funkcji, konieczne jest wcześniejsze zdeklarowanie głównej funkcji, jak pokazuje następujący przykład.

```
#include "usi.h"
int pom_fcja1(); /* prototyp pomocniczej funkcji dla USI */
int pom_fcja2(); /* prototyp następnej pomocniczej funkcji */

void funkcjeUSI(p1,p2)
struct notePLC *p1;
struct accPLC *p2;
{
  p2->a[0]=pom_fcja1() + pom_fcja2();
  return();
}
int pom_fcja1()
{
  /* pomocnicze działania dla głównej funkcji */
}
int pom_fcja2()
{
  /* następna pomocnicza funkcja */
}
```

12.4. UŻYWANE KOMPILATORA JĘZYKA C

Kompilatory języka C i ich różnice

Dla kompilacji USI instrukcji można użyć następujących kompilatorów:

Dla CPU serii B	Microware OS-9 / 68000 C Compiler Microware Systems Corporation
Dla CPU serii D, S a M	Keil C-Compiler-51 Keil Elektronik GmbH

Powyższe kompilatory używają typów danych podanych w tab.12.2.

Tab.12.2 Typy danych używane przez kompilator Microware OS-9 / 68000 C Compiler oraz Keil C-Compiler-51

Typ danych	Ilość bajtów		Wewnętrzna reprezentacja
	Microwa-re	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
double	8	-	binary floating point
pointer to ...	4	3	address

Z tabelki widać, że kompilatory dla różnych typów procesorów używają w przypadku typu **int** odmiennej wielkości wytwarzanego obiektu, i jednocześnie innego zakresu wyświetlania liczb. Jeśli chcemy pisać funkcje dla wszystkich serii jednostek centralnych PLC, lepiej jest używać w funkcjach typy deklarowane w pliku "usi.h" za pomocą dyrektywy *#typedef*.

Funkcje użytkownika można też pisać w środowisku firmy Borland, z wygodą wykorzystując możliwości strojenia programu, a dopiero końcową kompilację wykonać z odpowiednimi kompilatorami.

Kod instrukcji USI dla niektórych CPU nie może zawierać tabelki z danymi

Uwaga! Dla instrukcji użytkownika napisanych dla centralnej jednostki typu M, S i D przyjmuje się następujące ograniczenie: nie można używać inicjalizacji zmiennych lokalnych przy deklaracji zmiennej, np.:

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

Zmienne muszą być zainicjalizowane dopiero podczas wyliczeń funkcji, np.:

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

Ta niewygoda związana jest z relokacją funkcji USI przy jej podłączeniu do programu użytkownika. Nie respektowanie tego postępowania może mieć nieprzewidziane skutki !!!

12.5. PRZYKŁAD WYTWORZENIA WŁASNEJ INSTRUKCJI USI

Plik usi.h dla CPU serii D

```
/* plik usi.h dla CPU serii D */
typedef unsigned long  long_word;
typedef unsigned short word;
typedef signed short   signed_word;
typedef unsigned char  byte;
typedef signed char    signed_byte;

/* deklaracja struktur dla dostępu do akumulatora PLC */
struct accPLC {       /* struktura akumulatora PLC NS950 */
  word a[8];          /* poszczególne warstwy akumulatora */
};

/* wartości stałe dla definicji rozmiaru stref pamięci PLC */
#define MAXX  128 /* ilość bajtów X w pamięci */
#define MAXY  128 /* ilość bajtów Y w pamięci */
#define MAXS   64 /* ilość bajtów S w pamięci */
#define REZS   64 /* rezerwa w strefie S */
#define MAXD  256 /* ilość bajtów D w pamięci */
                  /* skopiowane z kodu programu */
#define MAXR 8192 /* ilość bajtów R w pamięci */
struct notePLC {  /* struktura pamięci PLC NS950 */
  u_char x[MAXX], /* obraz wejść X */
         y[MAXY], /* obraz wyjść Y */
         s[MAXS], /* systemowe rejestry S */
        rs[REZS], /* rezerwa dla strefy systemowej */
         d[MAXD], /* kopie danych D z programu użytkownika */
         r[MAXR]; /* rejestry użytkownika R */
  };
/* koniec pliku usi.h dla CPU serii D */
```

Przykład instrukcji użytkownika

Instrukcja użytkownika MUL16 mnoży wartość A0 akumulatora PLC przez wartość z warstwy A1, wynik zapisuje do podwójnej warstwy A01.

Źródłowy tekst instrukcji MUL16:

```
#include "usi.h"

void Mul16(p1, p2)   /* binarne mnożenie 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 niższy rząd wyniku */
  p2->a[1]=result.w[0];         /* A1 wyższy rząd wyniku */
  return;
}
```

Program funkcji MUL16 należy skompilować kompilatorem języka C na kod maszynowy odpowiedniego procesora.

12.6 PRZYKŁAD UŻYCIA INSTRUKCJI USI

Przykład użycia gotowej instrukcji użytkownika

Przyporządkowanie instrukcji użytkownika do programu PLC wykonuje się w programie xPRO za pomocą dyrektywy *#usi*, która symbolicznej nazwie instrukcji przyporządkowuje plik z binarnym kodem instrukcji.

Przykład 12.2

```
#program     Mnożenie
#usi MUL16 = mul16         ;definicja USI z automatycznym
                           ;przydzieleniem rozszerzenia pliku z kodem
                           ;instrukcji w zależności od serii CPU
#reg word a, b
#reg long c
;
P 0
       LD     a            ;wgranie pierwszego czynnika
       LD     b            ;wgranie drugiego czynnika
       USI    MUL16        ;A01 = a.b
       WR     c            ;zapisz wynik
        :
        :
E 0
```

12.7. UWAGI KOŃCOWE

Ważne reguły dla tworzenia instrukcji użytkownika

Podczas tworzenia instrukcji użytkownika należy respektować kilka reguł, które wynikają ze sposobu działania PLC. Należy szczególnie uświadomić sobie następujące ograniczenie:

Czas wyliczeń USI - jakakolwiek zdefiniowana przez użytkownika funkcja USI w momencie wywołania będzie tak samo jak pozostałe instrukcje PLC wymagać określonego czasu procesora PLC. O ten czas wydłuży się wykonywanie jednego cyklu PLC. W przypadku funkcji, które dla uzyskania wyniku wymagają dużej ilości iteracji, istnieje niebezpieczeństwo znacznego wydłużenia czasu cyklu PLC. Doprowadzi to do wyraźnego spowolnienia reakcji PLC na jednostkową zmianę na wejściu, które może doprowadzić do zatrzymania działania PLC na skutek przekroczenia maksymalnego czasu cyklu. Można jednak programować również i takie funkcje, ale w sposób, w którym przy każdym wywołaniu USI wykonana jest jedynie zdefiniowana ilość iteracji tak, aby zużytkowany czas procesora był do przyjęcia. W taki sposób zaprogramowana USI instrukcja produkuje wówczas wynik raz na kilka cykli PLC, co w wielu przypadkach jest dopuszczalne.

Wymagania na pamięć - kod maszynowy instrukcji USI jest częścią programu użytkownika, tak samo jak struktury dla jego udostępnienia dla procesora PLC. Z tego względu dobrze jest unikać programowania takich algorytmów, które prowadzą do rozległych kodów maszynowych. Dotyczy to przede wszystkim używania bibliotek języka C. Doświadczony programista i dobrze optymalizujący kompilator są w tym przypadku mile widziane.

Typ procesora PLC - zastosowany procesor wyznacza pewne ograniczenia np. odnośnie rozmiaru pamięci operacyjnej, obszaru dla stosu (stack), wykorzystania hw środków procesora itd. Dlatego dobrze jest konsultować instrukcje USI z pracownikami firmy TECO a.s.

A.1. CZASY WYKONYWANIA INSTRUKCJI DLA JEDNOSTKI CENTRALNEJ CPM-1E TECOMAT NS950

Czasy wykonywania instrukcji nie obejmują wpływu procesów systemowych, które przerywają wykonywanie programu. Takimi procesami są komunikacja szeregowa oraz obsługa niektórych jednostek peryferyjnych. Powodują one przedłużenie czasu cyklu.

Wykaz użytych symboli operandów:

Z - pamięć X, Y, S, R
D - dane
- wartość stała
A - bez operandu (pracuje jedynie z akumulatorem)
n - parametr liczbowy

Instrukcje do odczytu i zapisu danych

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D bit	byte	word	# word	Znaczenie instrukcji
LD	63 (3)	53 (3)	61 (3)	50 (3)	Odczyt danych bezpośrednich
LDC	64 (3)	54 (3)	63 (3)	52 (3)	Odczyt negowanych danych
WR	67 (3)	50 (3)	57 (3)	-	Zapis danych bezpośrednich
WRC	67 (3)	51 (3)	59 (3)	-	Zapis negowanych danych
PUT	77 (3)	60 (3)	66 (3)	-	Warunkowy zapis danych - warunek spełniony
	51	46	46	-	- warunek nie spełniony

Logiczne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D bit	A word	Znaczenie instrukcji
AND	66 (3)	50 (1)	AND z bezpośrednim operandem
ANC	67 (3)	-	AND z negowanym operandem
OR	66 (3)	50 (1)	OR z bezpośrednim operandem
ORC	67 (3)	-	OR z negowanym operandem
XOR	66 (3)	50 (1)	XOR z bezpośrednim operandem
XOC	67 (3)	-	XOR z negowanym operandem
SET	66 (3)	-	Warunkowe nastawianie
RES	67 (3)	-	Warunkowe zerowanie
LET	79 (3)	-	Impuls od dodatniego zbocza

Liczniki, układy czasowe

Mnemokod	Czas [µs] (Długość kodu [B])	Znaczenie instrukcji
CTU	186 (3)	Licznik wprzód
CTD	186 (3)	Licznik wstecz
TON	220 (3)	Układ czasowy (opóźnione załączenie)

Arytmetyczne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) A word	Znaczenie instrukcji
EQ	92 (1)	Porównanie (równość)

Operacje na akumulatorach

Mnemokod	Czas [µs] (Długość kodu [B])	Znaczenie instrukcji
POP n	42 + 7n (3)	Przesunięcie (rotacja) akumulatora wstecz o n poziomów

Instrukcje skoków i wywołania

Mnemo-kod	Czas [µs] (Długość kodu [B]) przejście	skok	Znaczenie instrukcji
JMP	-	114 (3)	Bezwarunkowy skok
JMD	42	120 (3)	Skok uwarunkowany niezerową wartością wyniku
L	36 (3)	-	Etykieta n (cel skoków i wywołań podprogramów)

Organizacyjne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) przejście	skok	P41 - P49	P50 - P57	P62 - P64	Znaczenie instrukcji
P	151 (3)	-	76	73	143	Początek procesu
E	-	54 (3)	229	77	54	Bezwarunkowy koniec procesu
NOP	36 (3)	-	-	-	-	Pusta operacja

A.2. CZASY WYKONYWANIA INSTRUKCJI DLA CENTRALNEJ JEDNOSTKI CPM-1M TECOMAT NS950

Czasy wykonywania instrukcji nie obejmują wpływu systemowych procesów, które przerywają wykonywanie programu. Takimi procesami są komunikacja szeregowa oraz obsługa niektórych jednostek peryferyjnych. Powodują one wydłużenie czasu cyklu.

Wykaz użytych symboli operandów:

Z - pamięć X, Y, S, R
D - dane
- wartość stała
U - fizyczny adres jednostki peryferyjnej
T - tabelki
A - bez operandu (pracuje jedynie na akumulatorze)
n - liczbowy parametr

Instrukcje do odczytu i zapisu danych

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D bit	Z D byte	Z D word	# word	U byte	U word	Znaczenie instrukcji
LD	63 (3)	53 (3)	61 (3)	50 (3)	81 (3)	109 (3)	Odczyt danych bezpośrednich
LDC	64 (3)	54 (3)	63 (3)	52 (3)	-	-	Odczyt negowanych danych
WR	67 (3)	50 (3)	57 (3)	-	80 (3)	113 (3)	Zapis danych bezpośrednich
WRC	67 (3)	51 (3)	59 (3)	-	-	-	Zapis negowanych danych
PUT	77 (3)	60 (3)	66 (3)	-	-	-	Warunkowy zapis danych - warunek spełniony
	51	46	46	-	-	-	- warunek nie spełniony

Logiczne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D bit	Z D byte	# word	A word	Znaczenie instrukcji
AND	66 (3)	53 (3)	49 (3)	50 (1)	AND z bezpośrednim operandem
ANC	67 (3)	54 (3)	-	-	AND z negowanym operandem
OR	66 (3)	49 (3)	49 (3)	50 (1)	OR z bezpośrednim operandem
ORC	67 (3)	50 (3)	-	-	OR z negowanym operandem
XOR	66 (3)	49 (3)	49 (3)	50 (1)	XOR z bezpośrednim operandem
XOC	67 (3)	50 (3)	-	-	XOR z negowanym operandem
SET	66 (3)	50 (3)	-	-	Warunkowe nastawianie
RES	67 (3)	53 (3)	-	-	Warunkowe zerowanie
LET	79 (3)	55 (3)	-	-	Impuls od dodatniego zbocza
NEG	-	-	-	42 (1)	Negacja szczytu akumulatora

Liczniki, rejestry przesuwne, układy czasowe, sterownik krokowy

Mnemo-kod	Czas [µs] (Długość kodu [B])	Znaczenie instrukcji
CTU	186 (3)	Licznik w przód
CTD	186 (3)	Licznik wstecz
CNT	219 (3)	Dwukierunkowy licznik
SFL	190 (3)	Rejestr przesuwny w lewo
SFR	190 (3)	Rejestr przesuwny w prawo
TON	220 (3)	Układ czasowy (opóźnione załączenie)
TOF	221 (3)	Układ czasowy (opóźnione wyłączenie)
RTO	235 (3)	Całkujący układ czasowy, miernik czasu
IMP	222 (3)	Układ czasowy - generator impulsu zadanej długości
STE	129 (3)	Sterownik krokowy (stepper) - zmiana stanu
	106	- stan niezmienny

Arytmetyczne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D byte	Z D word	# byte	# word	A byte	A word	Znaczenie instrukcji
ADD	-	99 (3)	-	88 (3)	-	88 (1)	Dodawanie z przeniesieniem
SUB	-	101 (3)	-	90 (3)	-	90 (1)	Odejmowanie z przeniesieniem
MUL	59 (3)	-	50 (3)	-	51 (1)	-	Mnożenie
DIV	74 (3)	-	65 (3)	-	66 (1)	-	Dzielenie
INR	-	-	-	-	-	79 (1)	Inkrementacja (+ 1)
DCR	-	-	-	-	-	79 (1)	Dekrementacja (– 1)
ROL n	-	-	-	-	-	81+10n (1)	Rotacja liczby w lewo
EQ	-	103 (3)	-	92 (3)	-	92 (1)	Porównanie (równość)
LT	-	103 (3)	-	92 (3)	-	92 (1)	Porównanie (mniejsze niż)
GT	-	104 (3)	-	93 (3)	-	93 (1)	Porównanie (większe niż)
BIN	-	-	-	-	-	80 (1)	Konwersja liczby na bin.
BCD	-	-	-	-	-	226 (1)	Konwersja liczby na BCD

Operacje z akumulatorami

Mnemo-kod	Czas [µs] (Długość kodu [B])	Znaczenie instrukcji
STK	152 (1)	Złożenie logicznych wartości 8 poziomów akumulatora do A0
SWP	39 (1)	Zamiana górnego i dolnego bajtu w A0
FLG	192 (1)	Logiczne I (AND) wszystkich bitów oraz funkcja poprzeczna bajtów A0 w S1
POP n	42 + 7n (3)	Przesunięcie (rotacja) akumulatora wstecz o n poziomów
NXT	375 (3)	Aktywacja następnego akumulatora
PRV	375 (3)	Aktywacja poprzedniego akumulatora
CHG	373 (3)	Aktywacja wybranego akumulatora z zapamiętaniem S0 i S1
	356	bez zapamiętania S0 i S1

Instrukcje skoków i wywołań podprogramów

Mnemo-kod	Czas [µs] (Długość kodu [B]) przejście	skok	Znaczenie instrukcji
JMP	-	114 (3)	Bezwarunkowy skok
JMD	42	120 (3)	Skok uwarunkowany niezerową wartością wyniku
JMC	42	120 (3)	Skok uwarunkowany zerową wartością wyniku
JMI	-	115 (1)	Skok na pośredni cel
CAL	-	138 (3)	Bezwarunkowe wywołanie podprogramu
CAD	42	144 (3)	Wywołanie podprogramu uwarunkowane niezerową wartością wyniku
CAC	42	144 (3)	Wywołanie podprogramu uwarunkowane zerową wartością wyniku
CAI	-	137 (1)	Wywołanie podprogramu z adresem pośrednim
RET	-	48 (1)	Bezwarunkowy powrót z podprogramu
RED	40	54 (1)	Powrót z podprogramu uwarunkowany niezerową wartością wyniku
REC	40	54 (1)	Powrót z podprogramu uwarunkowany zerową wartością wyniku
L	36 (3)	-	Etykieta n (cel skoków i wywołań)

Organizacyjne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) przejście	skok	P41 - P49	P50 - P57	P62 - P64	Znaczenie instrukcji
P	151 (3)	-	76	73	143	Początek procesu
E	-	54 (3)	229	77	54	Bezwarunkowy koniec procesu
ED	42 (1)	65	240	88	65	Koniec procesu przy niezerowym wyniku
EC	42 (1)	65	240	88	65	Koniec procesu przy zerowym wyniku
EOC	-	35 (1)	-	-	-	Koniec cyklu
NOP	36 (3)	-	-	-	-	Pusta operacja
BP	-	189 (3)	-	-	-	Punkt strojenia

Tabelkowe instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D bit	Z D byte	Z D word	T bit	T byte	T word	Znaczenie instrukcji
LTB	136 (3)	118 (3)	131 (3)	258 (3)	225 (3)	250 (3)	Odczyt pozycji z tabelki
WTB	151 (3)	126 (3)	140 (3)	-	-	-	Zapis pozycji do tabelki
LMS	-	-	-	-	-	248 (3)	Sekwencyjny odczyt pozycji
WMS	-	-	-	-	-	362 (3)	Sekwencyjny zapis pozycji
						+11	- czasowy narzut na 1 pozycję tabelki
FTB	-	68 (3)	75 (3)	-	182 (3)	203 (3)	Szukanie pozycji w tabelce
		+49	+55		+39	+46	- czasowy narzut na 1 przeszukiwaną pozycję
FTM	-	66 (3)	66 (3)	-	192 (3)	192 (3)	Szukanie części pozycji w tabelce
		+61	+90		+51	+80	- czasowy narzut na 1 przeszukiwaną pozycję
FTS	-	71 (3)	64 (3)	-	184 (3)	191 (3)	Przyporządkowanie pozycji według tabelki
		+51	+74		+41	+64	- czasowy narzut na 1 przeszukiwaną pozycję

Blokowe operacje

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D	T	A	Znaczenie instrukcji
SRC	99 (3)	195 (3)	-	Specyfikacja źródła danych dla przesunięcia
MOV	188 (3)	392 (3)	-	Przesunięcie bloku danych
	+17	+28		- czasowy narzut na 1 pozycję bloku
FIL	69 (3)	-	-	Napełnienie bloku wartością stałą
	+48			- czasowy narzut na 1 pozycję bloku

A.3. CZASY WYKONYWANIA INSTRUKCJI DLA JEDNOSTKI CENTRALNEJ CPM-2S TECOMAT NS950

Czasy wykonywania instrukcji nie obejmują wpływu systemowych procesów, które przerywają wykonywanie programu użytkownika. Takimi procesami są komunikacja szeregowa oraz obsługa niektórych jednostek peryferyjnych. Powodują one przedłużenie czasu cyklu.

Wykaz użytych symboli operandów:

Z - pamięć X, Y, S, R
D - dane
- wartość stała
U - fizyczny adres jednostki peryferyjnej
T - tabelki
A - bez operandu (pracuje jedynie na akumulatorze)
n - parametr liczbowy

Instrukcje do odczytu i zapisu danych

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D bit	Z D byte	Z D word	# word	U byte	U word	Znaczenie instrukcji
LD	12,8 (3)	12,8 (3)	13,9 (3)	11,2 (3)	18,3 (3)	25,5 (3)	Odczyt danych bezpośrednich
LDC	13,0 (3)	13,0 (3)	14,3 (3)	11,6 (3)	-	-	Odczyt negowanych danych
WR	13,4 (3)	11,6 (3)	13,0 (3)	-	17,5 (3)	25,7 (3)	Zapis danych bezpośrednich
WRC	13,4 (3)	11,8 (3)	13,4 (3)	-	-	-	Zapis negowanych danych
PUT	15,0 (3)	13,2 (3)	14,6 (3)	-	-	-	Warunkowy zapis danych - warunek spełniony
	8,9	10,5	10,5	-	-	-	- warunek nie spełniony

Logiczne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D bit	Z D byte	# word	A word	Znaczenie instrukcji
AND	12,1 (3)	12,5 (3)	11,0 (3)	10,7 (1)	AND z bezpośrednim operandem
ANC	12,3 (3)	12,7 (3)	-	-	AND z negowanym operandem
OR	12,5 (3)	11,8 (3)	11,0 (3)	10,7 (1)	OR z bezpośrednim operandem
ORC	12,5 (3)	11,9 (3)	-	-	OR z negowanym operandem
XOR	13,0 (3)	11,8 (3)	11,0 (3)	10,7 (1)	XOR z bezpośrednim operandem
XOC	13,2 (3)	11,9 (3)	-	-	XOR z negowanym operandem
SET	11,9 (3)	11,9 (3)	-	-	Warunkowe nastawianie
RES	13,6 (3)	12,5 (3)	-	-	Warunkowe zerowanie
LET	15,7 (3)	12,8 (3)	-	-	Impuls od dodatniego zbocza
NEG	-	-	-	9,2 (1)	Negacja szczytu akumulatora

Liczniki, rejestry przesuwne, układy czasowe, sterownik krokowy

Mnemo-kod	Czas [µs] (Długość kodu [B])	Znaczenie instrukcji
CTU	40,3 (3)	Licznik w przód
CTD	40,3 (3)	Licznik wstecz
CNT	48,3 (3)	Dwukierunkowy licznik
SFL	41,4 (3)	Rejestr przesuwny w lewo
SFR	41,4 (3)	Rejestr przesuwny w prawo
TON	47,0 (3)	Układ czasowy (opóźnione załączenie)
TOF	47,2 (3)	Układ czasowy (opóźnione wyłączenie)
RTO	50,3 (3)	Całkujący układ czasowy, miernik czasu
IMP	46,7 (3)	Układ czasowy - generator impulsu zadanej długości
STE	29,7 (3)	Krokowy sterownik (stepper) - zmiana stanu
	23,7	- stan niezmienny

Arytmetyczne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D byte	Z D word	# byte	# word	A byte	A word	Znaczenie instrukcji
ADD	-	19,5 (3)	-	18,1 (3)	-	16,8 (1)	Dodawanie z przeniesieniem
SUB	-	20,1 (3)	-	18,6 (3)	-	17,4 (1)	Odejmowanie z przeniesieniem
MUL	13,6 (3)	-	11,4 (3)	-	11,0 (1)	-	Mnożenie
DIV	15,9 (3)	-	14,5 (3)	-	13,4 (1)	-	Dzielenie
INR	-	-	-	-	-	14,6 (1)	Inkrementacja (+ 1)
DCR	-	-	-	-	-	14,6 (1)	Dekrementacja (– 1)
ROL n	-	-	-	-	-	17,5+2n (1)	Rotacja liczby w lewo
EQ	-	21,0 (3)	-	19,5 (3)	-	18,3 (1)	Porównanie (równość)
LT	-	21,0 (3)	-	19,5 (3)	-	18,3 (1)	Porównanie (mniejsze niż)
GT	-	20,8 (3)	-	19,4 (3)	-	18,1 (1)	Porównanie (większe niż)
BIN	-	-	-	-	-	17,5 (1)	Konwersja liczby na bin.
BCD	-	-	-	-	-	47,6 (1)	Konwersja liczby na BCD

Operacje z akumulatorami

Mnemo-kod	Czas [µs] (Długość kodu [B])	Znaczenie instrukcji
STK	36,5 (1)	Złożenie logicznych wartości 8 poziomów akumulatora do A0
SWP	8,7 (1)	Zamiana górnego i dolnego bajtu w A0
FLG	44,1 (1)	Logiczne I (AND) wszystkich bitów oraz poprzeczne funkcje bajtów A0 w S1
POP n	10,7 + 1,4n (3)	Przesunięcie (rotacja) akumulatora wstecz o n poziomów
NXT	92,4 (3)	Aktywacja następnego akumulatora
PRV	92,4 (3)	Aktywacja poprzedniego akumulatora
CHG	92,1 (3)	Aktywacja wybranego akumulatora z zapamiętaniem S0 i S1
	87,9	bez zapamiętania S0 i S1

Instrukcje skoków i wywołań

Mnemo-kod	Czas [µs] (Długość kodu [B]) przejście	skok	Znaczenie instrukcji
JMP	-	22,2 (3)	Bezwarunkowy skok
JMD	10,1	23,5 (3)	Skok uwarunkowany niezerową wartością wyniku
JMC	10,1	23,5 (3)	Skok uwarunkowany zerową wartością wyniku
JMI	-	21,7 (1)	Skok na pośredni cel
CAL	-	25,3 (3)	Bezwarunkowe wywołanie podprogramu
CAD	10,1	26,6 (3)	Wywołanie podprogramu uwarunkowane niezerową wartością wyniku
CAC	10,1	26,6 (3)	Wywołanie podprogramu uwarunkowane zerową wartością wyniku
CAI	-	24,2 (1)	Pośrednie wywołanie podprogramu
RET	-	10,3 (1)	Bezwarunkowy powrót z podprogramu
RED	9,0	11,6 (1)	Powrót z podprogramu uwarunkowany niezerową wartością wyniku
REC	9,0	11,6 (1)	Powrót z podprogramu uwarunkowany zerową wartością wyniku
L	8,9 (3)	-	Etykieta n (cel skoków i wywołań)

Organizacyjne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) przejście	skok	P41 - P49	P50 - P57	P62 - P64	Znaczenie instrukcji
P	30,6 (3)	49,0*	14,8	14,1	28,2	Początek procesu (*skok na etykietę daną SEQ)
E	-	11,2 (3)	53,2	51,4	11,2	Bezwarunkowy koniec procesu
ED	9,0 (1)	12,8	54,8	53,0	12,8	Koniec procesu przy niezerowym wyniku
EC	9,0 (1)	12,8	54,8	53,0	12,8	Koniec procesu przy zerowym wyniku
EOC	-	8,1 (1)	-	-	-	Koniec cyklu
NOP	8,9 (3)	-	-	-	-	Pusta operacja
BP	-	72,0 (3)	-	-	-	Punkt strojenia

Tabelkowe instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D bit	byte	word	T bit	byte	word	Znaczenie instrukcji
LTB	30,6 (3)	25,1 (3)	27,5 (3)	46,8 (3)	37,6 (3)	40,5 (3)	Odczyt pozycji z tabelki
WTB	33,1 (3)	26,0 (3)	28,8 (3)	-	-	-	Zapis pozycji do tabelki
LMS	-	-	-	-	-	47,9 (3)	Sekwencyjny odczyt pozycji
WMS	-	-	-	-	-	61,7 (3)	Sekwencyjny zapis pozycji
FTB	-	19,7 (3)	22,1 (3)	-	32,2 (3)	34,7 (3)	Szukanie pozycji w tabelce
		+3,1	+3,6		+3,1	+3,6	- czasowy narzut na 1 przeszukiwaną pozycję
FTM	-	20,3 (3)	21,9 (3)	-	32,9 (3)	35,1 (3)	Szukanie części pozycji w tabelce
		+4,3	+7,4		+4,3	+7,4	- czasowy narzut na 1 przeszukiwaną pozycję
FTS	-	19,7 (3)	21,0 (3)	-	32,7 (3)	33,6 (3)	Przyporządkowanie pozycji według tabelki
		+3,6	+6,3		+3,6	+6,3	- czasowy narzut na 1 przeszukiwaną pozycję

Blokowe operacje

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D	T	A	Znaczenie instrukcji
SRC	21,3 (3)	35,4 (3)	-	Specyfikacja źródła danych dla przesunięcia
MOV	49,0 (3)	74,3 (3)	-	Przesunięcie bloku danych
	+3,1	+5,4		- czasowy narzut na 1 pozycję bloku
FIL	20,4 (3)	-	-	Napełnienie bloku wartością stałą
	+2,2			- czasowy narzut na 1 pozycję bloku

A.4. CZASY WYKONYWANIA INSTRUKCJI DLA JEDNOSTKI CENTRALNEJ CPM-1D TECOMAT NS950 I DLA PLC TECOMAT TC400, TC500, TC600

Czasy wykonywania instrukcji nie obejmują wpływu systemowych procesów, które przerywają wykonywanie programu użytkownika. Takimi procesami są komunikacja szeregowa oraz obsługa niektórych jednostek peryferyjnych. Powodują one wydłużenie czasu cyklu.

Wykaz użytych symboli operandów:

Z - pamięć X, Y, S, R
D - dane
- wartość stała
U - fizyczny adres jednostki peryferyjnej
T - tabelki
A - bez operandu (pracuje jedynie na akumulatorze)
n - parametr liczbowy

Instrukcje do odczytu i zapisu danych

Mnemo-kod	Czas [µs] (Długość kodu [B])								Znaczenie instrukcji
	Z D				#		U		
	bit	byte	word	long float	word	long float	byte	word	
LD	12,8 (3)	12,8 (3)	13,9 (3)	18,6 (4)	11,2 (3)	-	18,3 (3)	25,5 (3)	Odczyt danych bezpośrednich
LDL	-	-	-	-	-	16,1 (6)	-	-	Odczyt danych bezpośrednich
LDC	13,0 (3)	13,0 (3)	14,3 (3)	19,4 (4)	11,6 (3)	-	-	-	Odczyt negowanych danych
WR	13,4 (3)	11,6 (3)	13,0 (3)	17,9 (4)	-	-	17,5 (3)	25,7 (3)	Zapis danych bezpośrednich
WRC	13,4 (3)	11,8 (3)	13,4 (3)	18,6 (4)	-	-	-	-	Zapis negowanych danych
WRA	-	13,6 (4)	15,0 (4)	19,0 (4)	-	-	-	-	Zapis danych bezp. z alternacją
PUT	15,0 (3)	13,2 (3)	14,6 (3)	19,5 (4)	-	-	-	-	Warunkowy zapis danych
	8,9	10,5	10,5	11,4	-	-	-	-	– warunek spełniony
									– warunek nie spełniony

Logiczne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B])							Znaczenie instrukcji
	Z D			#		A		
	bit	byte	word	word	long	word	long	
AND	12,1 (3)	12,5 (3)	15,2 (4)	11,0 (3)	-	10,7 (1)	-	AND z bezpośrednim operandem
ANL	-	-	-	-	17,0 (6)	-	15,4 (2)	AND z bezpośrednim operandem
ANC	12,3 (3)	12,7 (3)	15,6 (4)	-	-	-	-	AND z negowanym operandem
OR	12,5 (3)	11,8 (3)	15,2 (4)	11,0 (3)	-	10,7 (1)	-	OR z bezpośrednim operandem
ORL	-	-	-	-	17,0 (6)	-	15,4 (2)	OR z bezpośrednim operandem
ORC	12,5 (3)	11,9 (3)	15,6 (4)	-	-	-	-	OR z negowanym operandem
XOR	13,0 (3)	11,8 (3)	15,2 (4)	11,0 (3)	-	10,7 (1)	-	XOR z bezpośrednim operandem
XOL	-	-	-	-	17,0 (6)	-	15,4 (2)	XOR z bezpośrednim operandem
XOC	13,2 (3)	11,9 (3)	15,6 (4)	-	-	-	-	XOR z negowanym operandem
SET	11,9 (3)	11,9 (3)	14,6 (4)	-	-	-	-	Warunkowe nastawianie
RES	13,6 (3)	12,5 (3)	15,7 (4)	-	-	-	-	Warunkowe zerowanie
LET	15,7 (3)	12,8 (3)	16,5 (4)	-	-	-	-	Impuls od dodatniego zbocza
BET	15,4 (3)	13,6 (4)	16,1 (4)	-	-	-	-	Impuls od dowolnego zbocza
NEG	-	-	-	-	-	9,2 (1)	-	Negacja szczytu akumulatora
NGL	-	-	-	-	-	-	13,0 (2)	Negacja szczytu akumulatora

Liczniki, rejestry przesuwne, układy czasowe, sterownik krokowy

Mnemo-kod	Czas [µs] (Długość kodu [B])	Znaczenie instrukcji
CTU	40,3 (3)	Licznik w przód
CTD	40,3 (3)	Licznik wstecz
CNT	48,3 (3)	Dwukierunkowy licznik
SFL	41,4 (3)	Rejestr przesuwny w lewo
SFR	41,4 (3)	Rejestr przesuwny w prawo
TON	47,0 (3)	Układ czasowy (opóźnione załączenie)
TOF	47,2 (3)	Układ czasowy (opóźnione wyłączenie)
RTO	50,3 (3)	Całkujący układ czasowy, miernik czasu
IMP	46,7 (3)	Układ czasowy - generator impulsu zadanej długości
STE	29,7 (3)	Sterownik krokowy (stepper) - zmiana stanu
	23,7	- stan niezmienny

Instrukcje skoków i wywołań

Mnemo-kod	Czas [µs] (Długość kodu [B])		Znaczenie instrukcji
	przejście	skok	
JMP	-	22,2 (3)	Bezwarunkowy skok
JMD	10,1	23,5 (3)	Skok uwarunkowany niezerową wartością wyniku
JMC	10,1	23,5 (3)	Skok uwarunkowany zerową wartością wyniku
JMI	-	21,7 (1)	Skok na pośredni cel
JC	11,4	24,8 (4)	Skok uwarunkowany niezerową wartością wskaźnika przeniesienia CO
JNC	11,4	24,8 (4)	Skok uwarunkowany zerową wartością wskaźnika przeniesienia CO
JS	11,4	24,8 (4)	Skok uwarunkowany niezerową wartością wskaźnika S1.0
JNS	11,4	24,8 (4)	Skok uwarunkowany zerową wartością wskaźnika S1.0
JZ	11,4	24,8 (4)	Skok uwarunkowany niezerową wartością wskaźnika równości ZR
JNZ	11,4	24,8 (4)	Skok uwarunkowany zerową wartością wskaźnika równości ZR
CAL	-	25,3 (3)	Bezwarunkowe wywołanie podprogramu
CAD	10,1	26,6 (3)	Wywołanie podprogramu uwarunkowane niezerową wartością wyniku
CAC	10,1	26,6 (3)	Wywołanie podprogramu uwarunkowane zerową wartością wyniku
CAI	-	24,2 (1)	Pośrednie wywołanie podprogramu
RET	-	10,3 (1)	Bezwarunkowy powrót z podprogramu
RED	9,0	11,6 (1)	Powrót z podprogramu uwarunkowany niezerową wartością wyniku
REC	9,0	11,6 (1)	Powrót z podprogramu uwarunkowany zerową wartością wyniku
L	8,9 (3)	-	Etykieta n (cel skoków i wywołań)

Arytmetyczne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B])									Znaczenie instrukcji
	Z D			#			A			
	byte	word	long	byte	word	long	byte	word	long	
ADD	-	19,5 (3)	-	-	18,1 (3)	-	-	16,8 (1)	-	Dodawanie z przeniesieniem
ADX	13,6 (4)	14,3 (4)	18,6 (4)	-	-	-	-	-	-	Dodawanie
ADL	-	-	-	-	-	17,0 (6)	-	-	16,3 (2)	Dodawanie
SUB	-	20,1 (3)	-	-	18,6 (3)	-	-	17,4 (1)	-	Odejmowanie z przeniesieniem
SUX	14,1 (4)	14,8 (4)	19,5 (4)	-	-	-	-	-	-	Odejmowanie
SUL	-	-	-	-	-	17,4 (6)	-	-	16,5 (2)	Odejmowanie
MUL	13,6 (3)	-	-	11,4 (3)	-	-	11,0 (1)	-	-	Mnożenie
MUD	-	54 (4)	-	-	52 (4)	-	-	51 (2)	-	Mnożenie
DIV	15,9 (3)	-	-	14,5 (3)	-	-	13,4 (1)	-	-	Dzielenie
DID	-	314 (4)	-	-	313 (4)	-	-	311 (2)	-	Dzielenie
INR	12,8 (4)	13,6 (4)	13,6 (4)	-	-	-	-	14,6 (1)	-	Inkrementacja (+ 1)
DCR	14,5 (4)	15,4 (4)	15,4 (4)	-	-	-	-	14,6 (1)	-	Dekrementacja (– 1)
ROL n	-	-	-	-	-	-	-	17,5+2n (1)	-	Rotacja liczby w lewo
ROR n	-	-	-	-	-	-	-	18,1+2n (1)	-	Rotacja liczby w prawo
EQ	-	21,0 (3)	-	-	19,5 (3)	-	-	18,3 (1)	-	Porównanie (równość)
LT	-	21,0 (3)	-	-	19,5 (3)	-	-	18,3 (1)	-	Porównanie (mniejsze)
GT	-	20,8 (3)	-	-	19,4 (3)	-	-	18,1 (1)	-	Porównanie (większe)
CMP	17,5 (4)	18,8 (4)	24,1 (4)	-	17,5 (4)	-	-	17,5 (2)	-	Porównanie
CML	-	-	-	-	-	23,3 (6)	-	-	21,2 (2)	Porównanie
BIN	-	-	-	-	-	-	-	17,5 (1)	-	Konwersja liczby na bin.
BIL	-	-	-	-	-	-	-	-	173 (2)	Konwersja liczby na bin.
BCD	-	-	-	-	-	-	-	47,6 (1)	-	Konw. liczby na BCD
BCL	-	-	-	-	-	-	-	-	314 (2)	Konw. liczby na BCD
BAS	-	-	-	-	-	-	-	21,7 (2)	-	Konw. liczby na ASCII
ASB	-	-	-	-	-	-	-	22,4 (2)	-	Konw. liczby z ASCII

Operacje z akumulatorami

Mnemo-kod	Czas [µs] (Długość kodu [B])	Znaczenie instrukcji
STK	36,5 (1)	Złożenie logicznych wartości 8 poziomów akumulatora do A0
SWP	8,7 (1)	Zamiana górnego i dolnego bajtu w A0
SWL	12,7 (2)	Zamiana warstw A0 i A1
FLG	44,1 (1)	Logiczne AND wszystkich bitów oraz funkcje poprzeczne bajtów A0 w S1
POP n	10,3 + 1,4n (3)	Przesunięcie (rotacja) akumulatora wstecz o n poziomów
NXT	92,4 (3)	Aktywacja następnego akumulatora
PRV	92,4 (3)	Aktywacja poprzedniego akumulatora
CHG	92,1 (3)	Aktywacja wybranego akumulatora z zapamiętaniem S0 i S1
	87,9	bez zapamiętania S0 i S1
LAC	23,9 (4)	Odczyt wartości ze szczytu wybranego akumulatora
WAC	22,6 (4)	Zapis wartości na szczyt wybranego akumulatora

Organizacyjne instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B])					Znaczenie instrukcji
	przejście	skok	P41 - P49	P50 - P57	P62 - P64	
P	30,6 (3)	49,0*	14,8	14,1	28,2	Początek procesu (*skok na etykietę daną SEQ)
E	-	11,2 (3)	53,2	51,4	11,2	Bezwarunkowy koniec procesu
ED	9,0 (1)	12,8	54,8	53,0	12,8	Koniec procesu przy niezerowym wyniku
EC	9,0 (1)	12,8	54,8	53,0	12,8	Koniec procesu przy zerowym wyniku
EOC	-	8,1 (1)	-	-	-	Koniec cyklu
NOP	8,9 (3)	-	-	-	-	Pusta operacja
BP	-	72,0 (3)	-	-	-	Punkt strojenia
SEQ	14,5 (3)	18,8	-	-	-	Warunkowe przerwanie procesu

Tabelkowe instrukcje

Mnemo-kod	Czas [µs] (Długość kodu [B])						Znaczenie instrukcji
	Z D			T			
	bit	byte	word	bit	byte	word	
LTB	30,6 (3)	25,1 (3)	27,5 (3)	46,8 (3)	37,6 (3)	40,5 (3)	Odczyt pozycji z tabelki
WTB	33,1 (3)	26,0 (3)	28,8 (3)	52,3 (4)	45,0 (4)	50,1 (4)	Zapis pozycji do tabelki
LMS	-	-	-	-	-	47,9 (3)	Sekwencyjny odczyt pozycji
WMS	-	-	-	-	-	61,7 (3)	Sekwencyjny zapis pozycji
FTB	23,0 (4)	19,7 (3)	22,1 (3)	38,7 (4)	32,2 (3)	34,7 (3)	Szukanie pozycji w tabelce
	+4,1	+3,1	+3,6	+4,1	+3,1	+3,6	- czasowy narzut na 1 przeszukiwaną pozycję
FTM	-	20,3 (3)	21,9 (3)	-	32,9 (3)	35,1 (3)	Szukanie części pozycji w tabelce
		+4,3	+7,4		+4,3	+7,4	- czasowy narzut na 1 przeszukiwaną pozycję
FTS	-	19,7 (3)	21,0 (3)	-	32,7 (3)	33,6 (3)	Przyporządkowanie pozycji według tabelki
		+3,6	+6,3		+3,6	+6,3	- czasowy narzut na 1 przeszukiwaną pozycję

Blokowe operacje

Mnemo-kod	Czas [µs] (Długość kodu [B])			Znaczenie instrukcji
	Z D	T	A	
SRC	21,3 (3)	35,4 (3)	-	Specyfikacja źródła danych dla przesunięcia
MOV	49,0 (3)	74,3 (3)	-	Przesunięcie bloku danych
	+3,1	+5,4		- czasowy narzut na 1 pozycję bloku
MTN	-	-	35,6 (2)	Przesunięcie tabelki do pamięci
			+3,1	- czasowy narzut na 1 pozycję tabelki
MNT	-	-	40,5 (2)	Napełnienie tabelki z pamięci
			+4,0	- czasowy narzut na 1 pozycję tabelki
FIL	20,4 (3)	-	-	Napełnienie bloku wartością stałą
	+2,2			- czasowy narzut na 1 pozycję bloku

Operacje ze strukturalnymi tabelami

Mnemo-kod	Czas [µs] (Długość kodu [B])	Znaczenie instrukcji
LDS	47,6 (2)	Odczyt pozycji ze strukturalnej tabelki T
	+3,1	- czasowy narzut na 1 bajt pozycji
WRS	57,1 (2)	Zapis pozycji do strukturalnej tabelki T
	+5,2	- czasowy narzut na 1 bajt pozycji
FIS	28,0 (2)	Napełnianie pozycji strukturalnej tabelki w pamięci
	+2,2	- czasowy narzut na 1 bajt pozycji
FIT	51,4 (2)	Napełnianie pozycji strukturalnej tabelki T
	+4,3	- czasowy narzut na 1 bajt pozycji
FNS	23,9 (2)	Szukanie pozycji strukturalnej tabelki w pamięci
	+4,3	- czasowy narzut na 1 pozycję
FNT	36,2 (2)	Szukanie pozycji strukturalnej tabelki T
	+6,9	- czasowy narzut na 1 pozycję

Arytmetyczne instrukcje zmiennoprzecinkowe (dane są orientacyjne, zależą od wartości wejściowych)

Mnemo-kod	Czas [µs] (Długość kodu [B]) Z D	#	A	Znaczenie instrukcji
ADF	83 (4)	81 (6)	79 (2)	Dodawanie
SUF	83 (4)	81 (6)	79 (2)	Odejmowanie
MUF	125 (4)	123 (6)	121 (2)	Mnożenie
DIF	332 (4)	329 (6)	328 (2)	Dzielenie
CMF	54 (4)	52 (6)	50 (2)	Porównanie
CEI	-	-	550 (2)	Zaokrąglenie w górę
FLO	-	-	520 (2)	Zaokrąglenie w dół
ABS	-	-	11,6 (2)	Wartość bezwzględna (moduł)
LOG	-	-	1580 (2)	Logarytm dziesiętny
EXP	-	-	2100 (2)	Funkcja wykładnicza
LN	-	-	1580 (2)	Logarytm naturalny
POW	-	-	2110 (2)	Ogólna potęga
SQR	-	-	710 (2)	Pierwiastek drugiego stopnia
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
HYP	-	-	980 (2)	Euklidowska odległość
UWF	-	-	40 ÷ 170 (2)	Konwersja word bez znaku na float
IWF	-	-	40 ÷ 170 (2)	Konwersja word ze znakiem na float
ULF	-	-	40 ÷ 170 (2)	Konwersja long bez znaku na float
ILF	-	-	40 ÷ 170 (2)	Konwersja long ze znakiem na float
UFW	-	-	110 ÷ 200 (2)	Konwersja float na word bez znaku
IFW	-	-	110 ÷ 200 (2)	Konwersja float na word ze znakiem
UFL	-	-	110 ÷ 200 (2)	Konwersja float na long bez znaku
IFL	-	-	110 ÷ 200 (2)	Konwersja float na long ze znakiem
STF	-	-	2000 (2)	Konwersja łańcucha znaków ASCII na float
FST	-	-	1000 (2)	Konwersja float na ASCII łańcuch znaków

Instrukcja regulatora PID (czasowe dane są orientacyjne, zależą od wybranych funkcji oraz wartości wejściowych)

Mnemo-kod	Czas [µs] (Długość kodu [B])	Znaczenie instrukcji
CNV	900 (2)	Konwersja i obsługa danych z analogowych jednostek
PID	2000 ÷ 10000 (2)	PID regulator

A.5. CZASY WYKONYWANIA INSTRUKCJI DLA JEDNOSTKI CENTRALNEJ CPM-1B, CPM-2B TECOMAT NS950

W czasach wykonywania instrukcji nie uwzględniono wpływy procesów systemowych przerywających wykonywanie programu użytkownika. Do tych procesów zaliczamy komunikacje szeregowe oraz obsługi niektórych jednostek peryferyjnych. Wpływają one na przedłużenie czasu trwania cyklu.

Lista użytych symboli operandów:

Z - pamięć operacyjna X, Y, S, R
D - dane
- stała
U - adres fizyczny jednostki peryferyjnej
T - tablice
A - bez operandu (pracuje tylko z akumulatorem)
n - parametr liczbowy

Instrukcje odczytu i zapisu danych

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B]) Z D				#		U		Znaczenie instrukcji
	bit	byte	word	long float	word	long float	byte	word	
LD	4,0 (4)	3,2 (4)	4,2 (4)	5,0 (4)	2,8 (4)	-	200 (4)	210 (4)	Odczyt danych bezpośrednich
LDL	-	-	-	-	-	3,6 (6)	-	-	Odczyt danych bezpośrednich
LDC	4,0 (4)	3,4 (4)	4,4 (4)	5,2 (4)	2,9 (4)	-	-	-	Odczyt negowanych danych
WR	3,1 (4)	2,1 (4)	3,7 (4)	4,2 (4)	-	-	130 (4)	130 (4)	Zapis danych bezpośrednich
WRC	3,1 (4)	2,3 (4)	3,9 (4)	4,4 (4)	-	-	-	-	Zapis negowanych danych
WRA	-	4,1 (4)	4,4 (4)	6,0 (4)	-	-	-	-	Zapis danych bezpośrednich z wyborem
PUT	4,2 (4)	3,2 (4)	4,8 (4)	5,3 (4)	-	-	-	-	War. zapis danych - warunek spełniony
	2,8	2,8	2,8	2,8	-	-	-	-	- warunek nie spełniony

Instrukcje logiczne

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B]) Z D			#		A		Znaczenie instrukcji
	bit	byte	word	word	long	word	long	
AND	3,2 (4)	3,1 (4)	3,5 (4)	2,6 (4)	-	3,2 (2)	-	AND z operandem bezpośrednim
ANL	-	-	-	-	3,7 (6)	-	4,9 (2)	AND z operandem bezpośrednim
ANC	3,6 (4)	3,3 (4)	3,5 (4)	-	-	-	-	AND z operandem pośrednim
OR	3,2 (4)	3,1 (4)	3,5 (4)	2,6 (4)	-	3,2 (2)	-	OR z operandem bezpośrednim
ORL	-	-	-	-	3,7 (6)	-	4,9 (2)	OR z operandem bezpośrednim
ORC	3,6 (4)	3,3 (4)	3,5 (4)	-	-	-	-	OR z operandem pośrednim
XOR	3,2 (4)	3,1 (4)	3,5 (4)	2,6 (4)	-	3,2 (2)	-	XOR z operandem bezpośrednim
XOL	-	-	-	-	3,7 (6)	-	4,9 (2)	XOR z operandem bezpośrednim
XOC	3,6 (4)	3,3 (4)	3,5 (4)	-	-	-	-	XOR z operandem pośrednim
SET	3,3 (4)	2,9 (4)	3,3 (4)	-	-	-	-	Warunkowe ustawienie
RES	3,3 (4)	2,9 (4)	3,3 (4)	-	-	-	-	Warunkowe zerowanie
LET	5,0 (4)	3,5 (4)	4,3 (4)	-	-	-	-	Impuls od zbocza narastającego
BET	4,8 (4)	3,9 (4)	4,3 (4)	-	-	-	-	Impuls od zbocza dowolnego
NEG	-	-	-	-	-	2,6 (2)	-	Negacja wierzchołka akumulatora
NGL	-	-	-	-	-	-	3,6 (2)	Negacja wierzchołka akumulatora

Liczniki, rejestry przesuwne, timery, sterownik krokowy

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B])	Znaczenie instrukcji
CTU	12,7 (4)	Licznik zliczający w górę
CTD	13,1 (4)	Licznik zliczający w dół
CNT	16,4 (4)	Licznik dwukierunkowy
SFL	12,6 (4)	Rejestr przesuwny w lewo
SFR	12,6 (4)	Rejestr przesuwny w prawo
TON	15,6 (4)	Timer (opóźnione włączenie)
TOF	17,1 (4)	Timer (opóźnione wyłączenie)
RTO	17,6 (4)	Timer integrujący, miernik czasu
IMP	16,8 (4)	Timer - generator impulsów zadanej długości
STE	4,8 (4)	Sterownik kroków (stepper)

Operacje na akumulatorach

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B])	Znaczenie instrukcji
STK	16,4 (2)	Założenie wartości logicznych 8 poziomów akumulatora do A0
SWP	3,4 (2)	Zamiana górnego i dolnego bajta poziomu A0
SWL	4,3 (2)	Zamiana poziomów A0 i A1
FLG	16,1 (2)	AND logiczne wszystkich bitów oraz funkcje wzdłużne bajtów A0 w S1
POP	3,1 (4)	Rotacja akumulatora do tyłu o n poziomów
NXT	11,0 (4)	Aktywacja następnego akumulatora
PRV	10,9 (4)	Aktywacja poprzedniego akumulatora
CHG	10,5 (4)	Aktywacja dowolnego akumulatora z Archiwizacją S0 i S1
	10,1	bez Archiwizacji S0 i S1
LAC	6,4 (4)	Odczyt wartości z wierzchołka wybranego akumulatora
WAC	5,6 (4)	Zapis wartości do wierzchołka wybranego akumulatora

Instrukcje arytmetyczne

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B])									Znaczenie instrukcji
	Z D			#			A			
	byte	word	long	byte	word	long	byte	word	long	
ADD	-	6,2 (4)	-	-	5,8 (4)	-	-	6,5 (2)	-	Sumowanie z przeniesieniem
ADX	3,1 (4)	3,4 (4)	6,4 (4)	-	-	-	-	-	-	Sumowanie
ADL	-	-	-	-	-	5,5 (6)	-	-	6,7 (2)	Sumowanie
SUB	-	6,0 (4)	-	-	5,7 (3)	-	-	6,6 (2)	-	Różnica z przeniesieniem
SUX	3,1 (4)	3,2 (4)	5,9 (4)	-	-	-	-	-	-	Różnica
SUL	-	-	-	-	-	5,5 (6)	-	-	6,7 (2)	Różnica
MUL	4,6 (4)	-	-	4,4 (4)	-	-	5,2 (2)	-	-	Mnożenie
MUD	-	5,7 (4)	-	-	5,5 (4)	-	-	5,5 (2)	-	Mnożenie
DIV	6,2 (4)	-	-	6,0 (4)	-	-	6,8 (2)	-	-	Dzielenie
DID	-	32,1 (4)	-	-	31,9 (4)	-	-	31,9 (2)	-	Dzielenie
INR	3,0 (4)	3,8 (4)	4,5 (4)	-	-	-	-	5,8 (2)	-	Inkrementacja (+ 1)
DCR	4,2 (4)	4,7 (4)	5,3 (4)	-	-	-	-	5,8 (2)	-	Dekrementacja (– 1)
ROL	-	-	-	-	-	-	-	4,2 (2)	-	Rotacja liczby w lewo
ROR	-	-	-	-	-	-	-	4,2 (2)	-	Rotacja liczby w prawo
EQ	-	6,5 (4)	-	-	6,2 (4)	-	-	6,9 (2)	-	Porównanie (równość)
LT	-	6,5 (4)	-	-	6,2 (4)	-	-	6,9 (2)	-	Porównanie (mniejsze niż)
GT	-	6,5 (4)	-	-	6,2 (4)	-	-	6,9 (2)	-	Porównanie (większe od)
CMP	5,1 (4)	5,1 (4)	6,5 (4)	-	4,7 (4)	-	-	4,8 (2)	-	Porównanie
CML	-	-	-	-	-	5,9 (6)	-	-	7,6 (2)	Porównanie
BIN	-	-	-	-	-	-	-	6,7 (2)	-	Zmiana układu na bin.
BIL	-	-	-	-	-	-	-	-	17,8 (2)	Zmiana układu na bin.
BCD	-	-	-	-	-	-	-	21,3 (2)	-	Konwersja liczby do BCD
BCL	-	-	-	-	-	-	-	-	42,9 (2)	Konwersja liczby do BCD
BAS	-	-	-	-	-	-	-	8,9 (2)	-	Konwersja liczby na ASCII
ASB	-	-	-	-	-	-	-	3,6 (2)	-	Konwersja liczby z ASCII

Instrukcje skoku i odwołania

Mnemo-kod	Czas wykonywania [µs] (Dług. kodu [B])		Znaczenie instrukcji
	przejście	skok	
JMP	-	4,7 (4)	Skok bezwarunkowy
JMD	2,0	5,2 (4)	Skok uwarunkowany wynikiem niezerowym
JMC	2,0	5,2 (4)	Skok uwarunkowany wynikiem zerowym
JMI	-	5,0 (2)	Skok niejednoznaczny
JC	2,0	5,2 (4)	Skok uwarunkowany niezerowym ustawieniem znacznika przeniesienia CO
JNC	2,0	5,2 (4)	Skok uwarunkowany zerowym ustawieniem znacznika przeniesienia CO
JS	2,0	5,2 (4)	Skok uwarunkowany niezerowym ustawieniem znacznika S1.0
JNS	2,0	5,2 (4)	Skok uwarunkowany zerowym ustawieniem znacznika S1.0
JZ	2,0	5,2 (4)	Skok uwarunkowany niezerowym ustawieniem znacznika równości ZR
JNZ	2,0	5,2 (4)	Skok uwarunkowany zerowym ustawieniem znacznika równości ZR
CAL	-	6,4 (4)	Odwołanie bezwarunkowe programu
CAD	2,0	7,9 (4)	Odwołanie podprogramu uwarunkowane wynikiem niezerowym
CAC	2,0	7,9 (4)	Odwołanie podprogramu uwarunkowane wynikiem zerowym
CAI	-	7,7 (2)	Odwołanie niejednoznaczne podprogramu
RET	-	3,0 (2)	Bezwarunkowy powrót z podprogramu
RED	1,8	3,5 (2)	Powrót z podprogramu uwarunkowany wynikiem niezerowym
REC	1,8	3,5 (2)	Powrót z podprogramu uwarunkowany wynikiem zerowym
L	1,5 (4)	-	Etykieta n (cel skoku i odwołania)

Instrukcje organizacyjne

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B])					Znaczenie instrukcji
	przejście	skok	P41 - P49	P50 - P57	P62 - P64	
P	1,5 (4)	5,5*	4,5	4,5	1,5	Początek procesu (*skok do etykiety SEQ)
E	-	2,2 (4)	16,9	16,9	2,2	Bezwarunkowy koniec procesu
ED	1,8 (2)	3,7	18,5	18,5	3,7	Koniec procesu przy niezerowym wyniku
EC	1,8 (2)	3,7	18,5	18,5	3,7	Koniec procesu przy zerowym wyniku
EOC	-	3,4 (2)	-	-	-	Koniec cyklu
NOP	1,5 (4)	-	-	-	-	Operacja pusta
BP	-	19,5 (4)	-	-	-	Punkt strojenia
SEQ	4,5 (4)	6,0	-	-	-	Warunkowe przerwanie procesu

Instrukcje na tablicach

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B])						Znaczenie instrukcji
	Z D			T			
	bit	byte	word	bit	byte	word	
LTB	11,2 (4)	9,5 (4)	10,6 (4)	10,7 (4)	8,9 (4)	10,3 (4)	Odczyt danych z tablicy
WTB	10,5 (4)	8,7 (4)	9,9 (4)	14,5 (4)	12,5 (4)	15,3 (4)	Zapis danych do tablicy
FTB	10,9 (4)	7,8 (4)	9,1 (4)	9,4 (4)	8,6 (4)	10,2 (4)	Odszukanie danej w tablicy
	+1,0	+1,0	+1,1	+1,0	+1,0	+1,1	- narzut czasu na 1 przeszukiwaną daną
FTM	-	8,3 (4)	9,0 (4)	-	9,4 (4)	10,0 (4)	Odszukanie części danej w tablicy
		+1,5	+3,5		+1,5	+3,5	- narzut czasu na 1 przeszukiwaną daną
FTS	-	7,8 (4)	8,3 (4)	-	8,6 (4)	9,4 (4)	Zatrzymanie danej według tablicy
		+1,0	+2,1		+1,0	+2,1	- narzut czasu na 1 przeszukiwaną daną

Operacje blokowe

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B]) Z D	T	A	Znaczenie instrukcji
SRC	5,1 (4)	6,0 (4)	-	Specyfikacja źródła danych
MOV	7,8 (4) +1,0	10,4 (4) +1,0	-	Przesunięcie bloku danych - narzut czasu na 1 blok
MTN	-	-	7,5 (2) +1,0	Przepisanie tablicy do pamięci operacyjnej - narzut czasu na 1 pozycję tablicy
MNT	-	-	7,5 (2) +1,5	Napełnienie tablicy z pamięci operacyjnej - narzut czasu na 1 pozycję tablicy
FIL	7,5 (4) +1,0	-	-	Zapełnienie bloku stałą - narzut czasu na 1 blok

Operacje na tablicach strukturowanych

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B])	Znaczenie instrukcji
LDS	12,5 (2) +1,0	Odczyt danej z tablicy strukturowanej T - narzut czasu na 1 bajt danej
WRS	14,0 (2) +1,5	Zapis danej do tablicy strukturowanej T - narzut czasu na 1 bajt danej
FIS	8,5 (2) +1,0	Dopełnienie danej tablicy strukturowanej w pamięci operacyjnej - narzut czasu na 1 bajt danej
FIT	8,9 (2) +1,2	Dopełnienie danej tablicy strukturowanej T - narzut czasu na 1 bajt danej
FNS	8,5 (2) +1,2	Odszukanie danej tablicy strukturowanej w pamięci operacyjnej - narzut czasu na 1 daną
FNT	8,9 (2) +1,4	Odszukanie danej tablicy strukturowanej - narzut czasu na 1 daną

Instrukcje arytmetyczne na liczbach zmiennoprzecinkowych (dane czasowe są orientacyjne, zależą od wartości wejściowej)

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B]) Z D	#	A	Znaczenie instrukcji
ADF	93 (4)	97 (6)	97 (2)	Sumowanie
SUF	94 (4)	98 (6)	98 (2)	Różnica
MUF	79 (4)	83 (6)	83 (2)	Mnożenie
DIF	80 (4)	84 (6)	84 (2)	Dzielenie
CMF	57 (4)	61 (6)	61 (2)	Porównanie
CEI	-	-	83 (2)	Zaokrąglenie w górę
FLO	-	-	78 (2)	Zaokrąglenie w dół
ABS	-	-	5,1 (2)	Wartość bezwzględna
LOG	-	-	220 (2)	Logarytm dziesiętny
EXP	-	-	4500 (2)	Funkcja wykładnicza
LN	-	-	220 (2)	Logarytm naturalny
POW	-	-	8000 (2)	Potęgowanie
SQR	-	-	1050 (2)	Pierwiastek kwadratowy
SIN	-	-	3300 (2)	Sinus
ASN	-	-	4200 (2)	Arcus sinus
COS	-	-	3000 (2)	Cosinus
ACS	-	-	4200 (2)	Arcus cosinus
TAN	-	-	5800 (2)	Tangens
ATN	-	-	2800 (2)	Arcus tangens
HYP	-	-	1200 (2)	Odległość Euklidesa
UWF	-	-	40 ÷ 100 (2)	Konwersja word bez znaku na float
IWF	-	-	40 ÷ 100 (2)	Konwersja word ze znakiem na float
ULF	-	-	40 ÷ 100 (2)	Konwersja long bez znaku na float
ILF	-	-	40 ÷ 100 (2)	Konwersja long ze znakiem na float
UFW	-	-	50 ÷ 130 (2)	Konwersja float na word bez znaku
IFW	-	-	50 ÷ 130 (2)	Konwersja float na word ze znakiem
UFL	-	-	50 ÷ 130 (2)	Konwersja float na long bez znaku
IFL	-	-	50 ÷ 130 (2)	Konwersja float na long ze znakiem
STF	-	-	650 (2)	Konwersja kodu ASCII na float
FST	-	-	340 (2)	Konwersja float na kod ASCII

Instrukcje regulatora PID (dane czasowe są orientacyjne, zależą od wybranych funkcji oraz wartościach wejściowych)

Mnemo-kod	Czas wykonywania [µs] (Długość kodu [B])	Znaczenie instrukcji
CNV	500 (2)	Konwersja i opracowanie danych z jednostek analogowych
PID	1000 ÷ 2000 (2)	regulator PID

teco

Zamówienia i informacje:
Teco a.s., Havlíčkova 260, 280 58 Kolín 4, tel./fax: +420 321 72 57 48
P.H.U. LOGICON, Kolbego 7, 59-220 Legnica, tel./faks: 076/866 32 71

TXV 001 09.05