1. Upute za korištenje PLCsim simulatora

PLCsim simulator je simulator zamišljenog PLC-a (koji je stvoren po uzoru na Siemensov s7-200) čija namjena je upoznavanje studenata s osnovama rada programirljivih logičkih kontrolera. PLCsim je, zbog prenosivosti, razvijan za Java okruženje pa je prije pokretanja simulatora na vlastitom računalu potrebno instalirati Java Runtime Environment (ukoliko nije ranije instaliran). JRE se može besplatno skinuti sa Sunovih web-stranica (http://java.sun.com).

PLCsim se nalazi u jednoj JAR (**J**ava **ar**chive) datoteci koja sadrži sve potrebne klase nužne za pokretanje simulatora. PLCsim se može pokrenuti dvostrukim klikom miša na ikonu PLCsim.jar arhive (u Windows okruženju) ili se može pokrenuti iz naredbenog retka sljedećom naredbom:

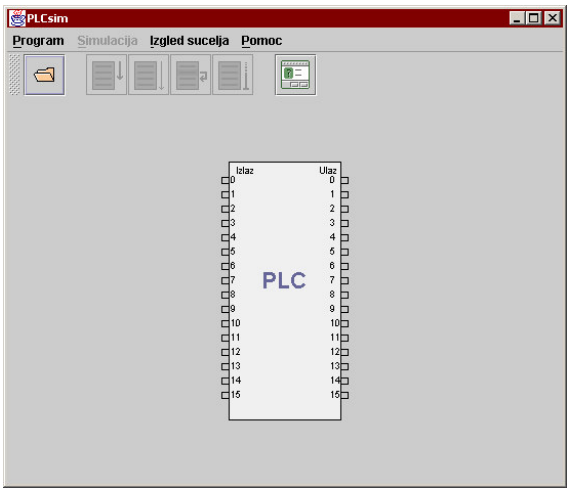
```
java -jar PLCsim.jar [konfiguracijska datoteka]
```

PLCsim koristi grafičko sučelje; ako pokušate pokrenuti PLCsim npr. na Linuxu, a da niste pokrenuli X server ili neko drugo grafičko okruženje (npr. Fresco, http://www.fresco.org), Java interpreter će vam javiti grešku.

Argument koji se može (ali i ne mora) poslati simulatoru je putanja do konfiguracijske datoteke. Konfiguracijska datoteka sadrži opis ulaznih i izlaznih modula koji su spojeni na PLC. Konfiguracijska datoteka se također može učitati i iz grafičkog sučelja PLCsim-a.

Nakon pokretanja PLCsim-a, otvara se glavni prozor programa.

U sredini prozora nalazi se grafički prikaz PLC-a s modulima koji su spojeni na njega (na popratnoj slici nije spojen niti jedan modul). Na vrhu prozora nalaze se izbornici i traka s prečacima. Pomoću izbornika **Program** možemo učitati novi program u simulator, učitati novu konfiguracijsku datoteku ili zatvoriti PLCsim. Izbornik **Simulacija** nam nudi nekoliko opcija za kontrolu simulacije programa na PLC-u koje će biti kasnije podrobno opisane. Pomoću izbornika **Izgled sučelja** možemo mijenjati način na koji će elementi grafičkog sučelja programa biti prikazani, a izbornik **Pomoć** prikazuje osnovne informacije o programu.



1.1. Učitavanje i prevođenje STL programa

Programi pisani u asembleru za PLC kojeg simulira `PLCsim` moraju imati nastavak `.STL`. Opis PLC-a i asemblera dan je u sljedećem poglavlju.

Prevođenje (asembliranje) STL programa pokreće se odabirom izbornika **Program → Učitaj STL program** ili odabirom prvog prečaca na traci s prečacima. Tada se otvara prozor za izbor datoteke u kojem treba odabrati datoteku koja sadržava željeni program. Nakon što smo odabrali datoteku, automatski se pokreće asembler koji ju pokušava prevesti: ukoliko uspješno završi prevođenje, prikazuje poruku o uspjehu, a ako u izvornom kodu nađe greške, tada prikaže prozor s greškama:

Ako je prevođenje uspjelo, može se pokrenuti simulacija.

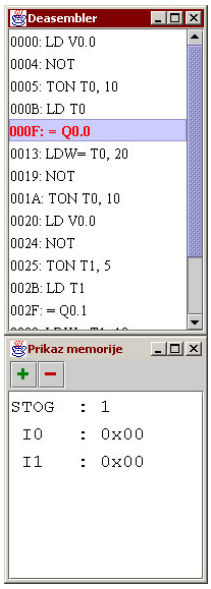
1.2. Simuliranje rada PLC-a

`PLCsim` podržava dva načina simulacije: izvršavanje u realnom vremenu i izvršavanje korak po korak. Izbornikom **Simulacija → Korak po korak** ili pritiskom na treći prečac na traci s prečacima prelazi se iz jednog načina izvođenja u drugi. Simulacija se pokreće odabirom izbornika **Simulacija → Pokreni** ili pritiskom na drugi prečac, a zaustavlja se odabirom izbornika **Simulacija → Zaustavi** ili pritiskom na peti prečac na traci s prečacima.

Nakon što je simulacija pokrenuta ne može se prelaziti iz jednog načina simulacije u drugi: potrebno je prvo zaustaviti simulaciju, promijeniti način simulacije i ponovno pokrenuti simulaciju.

Za vrijeme izvođenja simulacije može se klikom miša utjecati na ulazne module.

Ako se simulacija izvodi korak po korak, s pokretanjem simulacije otvaraju se dva dodatna prozora: jedan prozor sadrži mnemonički prikaz koda, a u drugom prozoru možemo promatrati određene registre odnosno varijable. Za razliku od STL koda kojeg prevodi `PLCsim` asembler, kôd koji će biti prikazan u prozoru s mnemoničkim prikazom imat će adrese zapisane na malo drugačiji način: neće se praviti razlika između adresiranja riječi i adresiranja bajta, odnosno adrese `VB5` i `VW5` bit će obje prikazane kao `V5`. Slično je i s prikazom memorije: možemo promatrati samo bajtove, pa ukoliko nas zanima, npr, riječ koja se nalazi na adresi `VW107`, tada ćemo promatrati bajtove `V107` i `V108`.



Prozor s prikazom memorije u prvom retku prikazuje vrh internog stoga. Radi se o stogu na koji se stavljaju logičke vrijednosti koje pojedine naredbe vraćaju; s tim vrijednostima rade druge naredbe. Interni stog je pojašnjen kod opisa asemblera. Ako se na stogu nalazi mnogo elemenata, neće biti prikazani svi, nego samo nekoliko prvih.

Prozor za prikaz memorije sadrži dva dugmeta označena s **+** i **–** koji služe za dodavanje novih varijabli/registara za nadgledanje, odnosno za brisanje elemenata koji se već nalaze u listi. Element koji prikazuje stanje stoga ne može se obrisati.

Kod izvođenja programa korak po korak, sljedeći se korak izvršava odabirom izbornika **Simulacija → Sljedeći korak** ili odabirom četvrtog prečaca.

2. Asembler za PLC

PLC-ovi su upravljački sklopovi koji se koriste za upravljanje u industrijskim postrojenjima. PLC se u mnogočemu razlikuje od *običnih* procesora: procesori na svojim nožicama imaju linije za adresiranje, linije za podatke, linije koje mogu uzrokovati prekid i sl. PLC ima samo ulaze i izlaze na koje se spajaju razni prekidači i sklopke (ulazi), odnosno motori, žaruljice i sl. (izlazi). Kod procesora u istoj memoriji nalaze se i program i podaci; kad se želi obaviti neka operacija nad podacima, podaci se najprije moraju učitati u registre. PLC nema registara, već može sve operacije obavljati direktno u memoriji. Memorija PLC-a podijeljena je u nekoliko područja, a prevedeni programski kôd se nalazi u zasebnom području kojem se ne može pristupiti. Procesor izvršava program od početka do naredbe `HALT`, a PLC izvršava program u ciklusima.

Ciklus PLC-a započinje čitanjem ulaza; vrijednosti s ulaznih nožica čitaju se i prepisuju u ulazni (**I**) registar[1]. Zatim se program počinje izvoditi sve dok ne dođe do naredbe `END`. S tom naredbom završava ciklus PLC-a. Na kraju ciklusa PLC čita vrijednosti u izlaznom (**Q**) registru i postavlja vrijednosti izlaznih linija na pročitane vrijednosti. Nakon završetka ciklusa započinje se s izvršavanjem sljedećeg ciklusa.

PLC koji se simulira na vježbama ima 6 vidljivih memorijskih područja. To su ulazni (**I**) i izlazni (**Q**) registar, registar zastavica (**SM**) i područje varijabli (**V**). Osim ta 4 područja postoji još područje brojača (counters; **C**) i vremenskih brojača (timers; **T**).

Ulazni i izlazni registar veličine su 2 bajta, registar zastavica velik je 3 bajta, a područje varijabli veliko je 2048 bajtova. Brojači i vremenski brojači posebno su objašnjeni kasnije; njih ima po 256 od svake vrste.

PLC može adresirati podatke veličine bita, bajta i riječi (2 bajta). Kod izravnog adresiranja navodi se ime registra, način adresiranja i adresa bajta. Sintaksa adresiranja je sljedeća:

```
<adresiranje bita>   → <ime registra> <broj bajta> . <broj bita>
<adresiranje bajta>  → <ime registra> B <broj bajta>
<adresiranje riječi> → <ime registra> W <broj bajta>
```

Slijede primjeri za svaku od navedenih vrsta adresiranja:

`I0.0` – adresira se najmanje značajan bit prvog bajta u registru I
`SMB2` – adresira se posljednji bajt u registru zastavica
`VW17` – adresira se riječ koja počinje bajtom na adresi 17 u registru varijabli.

Riječi se sastoje od 2 bajta; bajt na nižoj adresi je značajniji od bajta na višoj adresi.

[1] U daljnjem tekstu mješat će se izrazi registar i memorijsko područje iako se radi o jedno te istoj stvari. Memorija je podijeljena u nekoliko memorijskih područja, odnosno registara. Dakle registar I je isto što i memorijsko područje I.

Neke naredbe PLC-a mogu uzimati konstante kao argumente. Konstante mogu biti zapisane u dekadskom zapisu ili u heksadekadskom pomoću prefiksa `16#` (npr. `255` i `16#00FF` predstavljaju zapis istog broja).

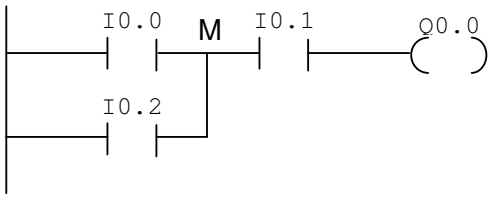
Osim direktnog adresiranja moguće je koristiti i indirektno adresiranje. Referenca na određeni bajt u memoriji tretira se kao obična konstanta duljine 2 bajta, pa se svim naredbama koje mogu kao argument uzeti konstantu može predati i referenca na bajt. Referenca se piše kao da se direktno adresira bajt, s time da se doda prefiks `&` (npr. `&VB0`). Budući da referenca predstavlja adresu određenog podatka, nema razlike između reference na bajt i reference na riječ. Dakle, nema razlike između `&VB15` i `&VW15`. Nakon što smo referencu spremili u memoriju, možemo pristupati podatku na kojeg imamo referencu tako da ispred adrese riječi u kojoj je zapisana referenca stavimo prefiks `*`. Sljedeći kôd prikazuje indirektno adresiranje:

```
MOVW &IB0, VW23
MOVB *VW23, QB1        =        MOVB IB0, QB1
```

2.1. Programiranje na temelju ljestvičastih dijagrama

Prije pojave PLC-ova, upravljanje u industrijskim postrojenjima rješavalo se relejima koji su se spajali u mreže. Sheme takvih sklopova podsjećaju na ljestve, pa otuda i ime. Budući da su PLC-ovi nastali kao zamjena za relejne sklopove, ostala je potreba da se PLC-ove može programirati na isti način na koji su se prethodno slagale mreže releja, odnosno da se već postojeće sheme mogu jednostavno *prepisati* u asembler za PLC. Da bi PLC mogao simulirati rad mreža releja on ima interni stog na koji se stavljaju binarne vrijednosti. Interni stog predstavlja ožičene veze između raznih blokova u shemi.

Pogledajmo npr. sljedeću mrežu i pripadni programski kôd:




```
LD I0.0
O  I0.2
A  I0.1
=  Q0.0
END
```

U točki M bit će logička jedinica ako je bilo koji od ulaza `I0.0` i `I0.2` uključen. Ako je u točki M logička jedinica, i ako je još i ulaz `I0.1` uključen, tada će logička jedinica biti postavljena i na izlaz `Q0.0`. Ova mreža može se opisati sljedećim logičkim izrazom:

```
Q0.0 = (I0.0 | I0.2) & I0.1
```

Program u asembleru opisuje ovaj logički izraz. Naredba `LD` učitava vrijednost ulaza i postavlja tu vrijednost na vrh internog stoga. Naredba `O` učitava vrijednost drugog izlaza, skida sa stoga staru vrijednost, izvršava logičko **ILI** između te dvije vrijednosti i rezultat stavlja na stog. Analogno tome, naredba `A` izvršava logičko **I** između vrijednosti na stogu i vrijednosti pročitane s ulaza. Na kraju se vrijednost na vrhu stoga skida sa stoga i postavlja na izlaz.

2.2. Brojači i vremenski brojači (*timers & counters*)

PLC sadrži 256 brojača koji broje promjene stanja i 256 vremenskih brojača koji se mogu upotrijebiti za brojanje vremena. Obje vrste brojača sastoje se od dva elementa: varijable brojanja koja je veličine riječi (2 bajta) i bita brojača. Varijabla brojanja sadrži trenutno stanje brojača (dakle koliko je puta nastupio događaj kojeg brojač prebrojava). Bit brojača postavlja se u logičko jedan nakon što je varijabla brojanja prešla određenu vrijednost. Naredbe koje kao argument uzimaju adresu bita mogu kao argument uzeti i neki brojač; tada će se pristupati bitu brojača (npr.: `LD T22`). Naredbe koje kao argument uzimaju adresu riječi u memoriji mogu kao argument uzeti i neki brojač; tada će se pristupati varijabli brojanja (npr.: `MOVW C100, VW0`).

Brojači: counter broji svaki prijelaz iz niskog u visoko stanje na svojim ulazima. Postoje tri vrste countera: Up counter, Up/down counter i down counter. Up i down counteri broje svaku promjenu ulaznog bita iz 0 u 1. Up counter broji od nule a kad izbrojena vrijednost prijeđe unaprijed zadanu konstantu, counter bit se postavlja u 1. Kad izbrojena vrijednost dosegne 32767, brojanje se zaustavlja. Down counter također broji sve prijelaze iz 0 u 1, s time da on kreće od unaprijed zadane konstante i smanjuje izbrojenu vrijednost dok ne dođe do 0. Kad dođe do 0, postavlja counter bit na 1 i prekida daljnje brojanje. Oba ta countera imaju i reset ulaz koji ih resetira u početno stanje kad je na logičkom 1. Taj reset ulaz nalazi se na vrhu stoga, a prva vrijednost ispod vrha stoga tumači se kao ulaz koji se promatra. Up / down counter kreće od 0, povećava vrijednost kad na *up* liniji naiđe na rastući brid, a smanjuje vrijednost kad na *down* liniji naiđe na rastući brid. Ako je *reset* ulaz u 1, on se resetira. Ako prijeđe predefiniranu vrijednost, tada se postavlja counter bit u 1. Ako vrijednost izađe iz dvobajtnog opsega (-32768 do 32767), brojanje se zaustavlja. Vrh stoga tumači se kao *reset* ulaz, prva vrijednost ispod vrha stoga je *count down* ulaz, a sljedeća vrijednost niže na stogu je *count up* ulaz.

Vremenski brojači: razlučivost timera je 100 ms što znači da ćemo, ako želimo postaviti timer na jednu sekundu, zadati da broji do 1000ms / 100 ms = 10. Postoje tri vrste timera koji se postavljaju sljedećim naredbama: `TON`, `TONR` i `TOF`. Timeri `TON` i `TONR` čitaju vrijednost bita sa vrha stoga. Ako je taj bit postavljen u logičko 1, tada ti timeri počinju brojati. Kad trenutna vrijednost prijeđe predefiniranu vrijednost, nastavlja se brojati, a timer bit se postavlja u 1. Kad vrijednost brojanja dođe do 32767, brojanje se zaustavlja. Ako u nekom trenutku vrh stoga prijeđe u logičko 0, tada oba ova timera prestaju brojati. `TON` timer automatski resetira timer bit u 0 i izbrojenu vrijednost u 0, a `TONR` zadržava trenutno stanje i nastavlja s brojanjem nakon što vrh stoga opet prijeđe u 1. `TOF` timer služi za odgađanje prebacivanja izlaza na 0 nakon što vrh stoga prijeđe u 0. Ako je vrh stoga 0, tada brojanje počinje, a timer bit se postavlja u 1. Ako je vrh stoga 1, brojanje se zaustavlja, izbrojena vrijednost se resetira u 0, i timer bit se postavlja u 1. Nakon što brojanje prijeđe predefiniranu vrijednost, timer bit se postavlja u 0 i brojanje se prekida.

2.3. Popis naredbi

b – adresiranje bita
B – adresiranje bajta
W – adresiranje riječi
N – zapis konstante

I – ulazni registar
Q – izlazni registar
SM – registar zastavica
V – područje varijabli
C – adresiranje brojača
T – adresiranje vremenskog brojača
c – konstanta
& – referenca na adresu
* – indirektno adresiranje

Zastavice
Z – zero
N – negative
O – overflow
D – division by zero
B – BCD conversion error

Naredbe za rad nad bitovima

Mnemonik	Argumenti	Zastavice	Opis
`LD b`	b: I, Q, SM, V, T, C	-	PUSH (b)
`A b`	b: I, Q, SM, V, T, C	-	PUSH (b & POP( ))
`O b`	b: I, Q, SM, V, T, C	-	PUSH (b | POP( ))
`LDN b`	b: I, Q, SM, V, T, C	-	PUSH (!b)
`AN b`	b: I, Q, SM, V, T, C	-	PUSH ((!b) & POP( ))
`ON b`	b: I, Q, SM, V, T, C	-	PUSH ((!b) | POP ( ))
`LDI b`	b: I	-	kao LD, s time da prije izvršavanja naredbe obnovi vrijednosti s ulaza a ne čeka početak ciklusa da to obavi
`AI b`	b: I	-	analogno prethodnoj
`OI b`	b: I	-	analogno prethodnoj
`LDNI b`	b: I	-	analogno prethodnoj
`ANI b`	b: I	-	analogno prethodnoj
`ONI b`	b: I	-	analogno prethodnoj
`= b`	b: I, Q, SM, V	-	b = POP ( )
`=I b`	b: Q	-	b = POP ( ); odmah se obnavljaju stanja izlaznih linija prema stanju Q registra, ne čeka se kraj ciklusa
`NOT`	-	-	PUSH (! POP ( ))
`NOP`	-	-	tko ne zna što radi NOP nek se ispiše s faksa
`EU`	-	-	ako je vrijednost na vrhu stoga prešla iz 0 u 1, tada se na stog stavlja 1; inače se stavlja 0
`ED`	-	-	ako je vrijednost na vrhu stoga prešla iz 1 u 0, tada se na stog stavlja 1; inače se stavlja 0
`S b,N`	b: I, Q, SM, V N: konstanta	-	ako je vrijednost na vrhu stoga 1, tada postavlja N bitova počevši od bita *b* u 1; inače ne radi ništa
`R b,N`	b: I, Q, SM, V N: konstanta	-	ako je vrijednost na vrhu stoga 1, tada postavlja N bitova počevši od bita *b* u 0; inače ne radi ništa
`SI b,N`	b: Q N: konstanta	-	analogno naredbi *S*, s time da se odmah obnavljaju stanja izlaznih linija, ne čeka se kraj ciklusa
`RI b,N`	b: Q N: konstanta	-	analogno naredbi *R*, s time da se odmah obnavljaju stanja izlaznih linija, ne čeka se kraj ciklusa

Naredbe za usporedbu

Mnemonik	Argumenti	Zast.	Opis
LDB= B1,B2	B: I, Q, SM, V, *	-	(B1 == B2)? PUSH(true) : PUSH(false);
AB= B1,B2	B: I, Q, SM, V, *	-	(B1 == B2)? PUSH(true && POP()) : PUSH(false && POP());
OB= B1,B2	B: I, Q, SM, V, *	-	(B1 == B2)? PUSH(true || POP()) : PUSH(false || POP());
LDB<> B1,B2	B: I, Q, SM, V, *	-	(B1 != B2)? PUSH(true) : PUSH(false);
AB<> B1,B2	B: I, Q, SM, V, *	-	(B1 != B2)? PUSH(true && POP()) : PUSH(false && POP());
OB<> B1,B2	B: I, Q, SM, V, *	-	(B1 != B2)? PUSH(true || POP()) : PUSH(false || POP());
LDB< B1,B2	B: I, Q, SM, V, *	-	(B1 < B2)? PUSH(true) : PUSH(false);
AB< B1,B2	B: I, Q, SM, V, *	-	(B1 < B2)? PUSH(true && POP()) : PUSH(false && POP());
OB< B1,B2	B: I, Q, SM, V, *	-	(B1 < B2)? PUSH(true || POP()) : PUSH(false || POP());
LDB> B1,B2	B: I, Q, SM, V, *	-	(B1 > B2)? PUSH(true) : PUSH(false);
AB> B1,B2	B: I, Q, SM, V, *	-	(B1 > B2)? PUSH(true && POP()) : PUSH(false && POP());
OB> B1,B2	B: I, Q, SM, V, *	-	(B1 > B2)? PUSH(true || POP()) : PUSH(false || POP());
LDB<= B1,B2	B: I, Q, SM, V, *	-	(B1 <= B2)? PUSH(true) : PUSH(false);
AB<= B1,B2	B: I, Q, SM, V, *	-	(B1 <= B2)? PUSH(true && POP()) : PUSH(false && POP());
OB<= B1,B2	B: I, Q, SM, V, *	-	(B1 <= B2)? PUSH(true || POP()) : PUSH(false || POP());
LDB>= B1,B2	B: I, Q, SM, V, *	-	(B1 >= B2)? PUSH(true) : PUSH(false);
AB>= B1,B2	B: I, Q, SM, V, *	-	(B1 >= B2)? PUSH(true && POP()) : PUSH(false && POP());
OB>= B1,B2	B: I, Q, SM, V, *	-	(B1 >= B2)? PUSH(true || POP()) : PUSH(false || POP());

LDW= W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 == W2)? PUSH(true) : PUSH(false);
AW= W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 == W2)? PUSH(true && POP()) : PUSH(false && POP());
OW= W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 == W2)? PUSH(true || POP()) : PUSH(false || POP());
LDW<> W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 != W2)? PUSH(true) : PUSH(false);
AW<> W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 != W2)? PUSH(true && POP()) : PUSH(false && POP());
OW<> W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 != W2)? PUSH(true || POP()) : PUSH(false || POP());
LDW< W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 < W2)? PUSH(true) : PUSH(false);
AW< W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 < W2)? PUSH(true && POP()) : PUSH(false && POP());
OW< W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 < W2)? PUSH(true || POP()) : PUSH(false || POP());
LDW> W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 > W2)? PUSH(true) : PUSH(false);
AW> W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 > W2)? PUSH(true && POP()) : PUSH(false && POP());
OW> W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 > W2)? PUSH(true || POP()) : PUSH(false || POP());
LDW<= W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 <= W2)? PUSH(true) : PUSH(false);
AW<= W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 <= W2)? PUSH(true && POP()) : PUSH(false && POP());
OW<= W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 <= W2)? PUSH(true || POP()) : PUSH(false || POP());
LDW>= W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 >= W2)? PUSH(true) : PUSH(false);
AW>= W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 >= W2)? PUSH(true && POP()) : PUSH(false && POP());
OW>= W1,W2	W: I, Q, SM, V, C, T, c, *	-	(W1 >= W2)? PUSH(true || POP()) : PUSH(false || POP());

Naredbe za rad s brojačima

Mnemonik	Argumenti	Zast.	Opis
TON T,PT	PT: c	-	Rad s brojačima detaljnije je opisan u prethodnom tekstu
TONR T,PT	PT: c	-	Rad s brojačima detaljnije je opisan u prethodnom tekstu
TOF T,PT	PT: c	-	Rad s brojačima detaljnije je opisan u prethodnom tekstu
CTU C,PV	PV: c	-	Rad s brojačima detaljnije je opisan u prethodnom tekstu
CTUD C,PV	PV: c	-	Rad s brojačima detaljnije je opisan u prethodnom tekstu
CTD C,PV	PV: c	-	Rad s brojačima detaljnije je opisan u prethodnom tekstu

Cjelobrojne naredbe

Naredbe iz sljedeće grupe provjeravaju vrh stoga: ako je vrh stoga u logičkoj jedinici, naredba će se izvršiti, a ako je u logičkoj nuli, naredba se neće izvršiti. Vrijednost s vrha stoga se NE skida.

Mnemonik	Argumenti	Zast.	Opis
+I Win,Wout	Win: I, Q, V, T, C, *, c Wout: I, Q, V, *	Z, N, O	Wout = Wout + Win
-I Win,Wout	Win: I, Q, V, T, C, *, c Wout: I, Q, V, *	Z, N, O	Wout = Wout – Win
*I Win,Wout	Win: I, Q, V, T, C, *, c Wout: I, Q, V, *	Z, N, O	Wout = Wout * Win
/IN Win,Wout	Win: I, Q, V, T, C, *, c Wout: I, Q, V, *	Z, N, O, D	Wout = Wout / Win
INCB Bout	Bout: I, Q, V, *	O, Z	Bout ++
DECB Bout	Bout: I, Q, V, *	O, Z	Bout --
INCW Wout	Wout: I, Q, V, *	O, N, Z	Wout ++
DECW Wout	Wout: I, Q, V, *	O, N, Z	Wout --

Naredbe za prijenos vrijednosti

Naredbe iz sljedeće grupe provjeravaju vrh stoga: ako je vrh stoga u logičkoj jedinici, naredba će se izvršiti, a ako je u logičkoj nuli, naredba se neće izvršiti. Vrijednost s vrha stoga se NE skida.

Mnemonik	Argumenti	Zast.	Opis
MOVB Bin,Bout	Bin: I, Q, V, T, C, * Bout: I, Q, V, *	Z	Bout = Bin
MOVW Win,Wout	Win: I, Q, V, T, C, *, &, c Wout: I, Q, V, *	Z	Wout = Win

Logičke naredbe

Naredbe iz sljedeće grupe provjeravaju vrh stoga: ako je vrh stoga u logičkoj jedinici, naredba će se izvršiti, a ako je u logičkoj nuli, naredba se neće izvršiti. Vrijednost s vrha stoga se NE skida.

Mnemonik	Argumenti	Zast.	Opis
`ANDB Bin, Bout`	Bin: I, Q, V, T, C, * Bout: I, Q, V, *	Z	Bout = Bout & Bin
`ORB Bin, Bout`	Bin: I, Q, V, T, C, * Bout: I, Q, V, *	Z	Bout = Bout | Bin
`XORB Bin, Bout`	Bin: I, Q, V, T, C, * Bout: I, Q, V, *	Z	Bout = Bout ^ Bin
`ANDW Win, Wout`	Win: I, Q, V, T, C, c, * Wout: I, Q, V, *	Z	Wout = Wout & Win
`ORW Win, Wout`	Win: I, Q, V, T, C, c, * Wout: I, Q, V, *	Z	Wout = Wout | Win
`XORW Win, Wout`	Win: I, Q, V, T, C, c, * Wout: I, Q, V, *	Z	Wout = Wout ^ Win
`INVB Bout`	Bout: I, Q, V, *	Z	Bout = ~Bout
`INVW Wout`	Wout: I, Q, V, *	Z	Wout = ~ Wout

Naredbe za posmak

Naredbe iz sljedeće grupe provjeravaju vrh stoga: ako je vrh stoga u logičkoj jedinici, naredba će se izvršiti, a ako je u logičkoj nuli, naredba se neće izvršiti. Vrijednost s vrha stoga se NE skida.

Mnemonik	Argumenti	Zast.	Opis
`SRB B,N`	B: I, Q, V, * N: c	Z	Shift right byte
`SLB B,N`	B: I, Q, V, * N: c	Z	Shift left byte
`RRB B,N`	B: I, Q, V, * N: c	Z	Rotate right byte
`RLB B,N`	B: I, Q, V * N: c	Z	Rotate left byte
`SRW W,N`	W: I, Q, V, * N: c	Z	Shift right word
`SLW W,N`	W: I, Q, V, * N: c	Z	Shift left word
`RRW W,N`	W: I, Q, V, * N: c	Z	Rotate right word
`RLW W,N`	W: I, Q, V, * N: c	Z	Rotate left word

Naredbe za pretvorbu

Naredbe iz sljedeće grupe provjeravaju vrh stoga: ako je vrh stoga u logičkoj jedinici, naredba će se izvršiti, a ako je u logičkoj nuli, naredba se neće izvršiti. Vrijednost s vrha stoga se NE skida.

Mnemonik	Argumenti	Zast.	Opis
`BCDI W`	W: I, Q, V, *	B	Pretvara cijeli broj u BCD zapis
`IBCD W`	W: I, Q, V, *	B	Pretvara BCD zapis u cijeli broj

Naredbe za upravljanje tokom programa

Mnemonik	Argumenti	Zast.	Opis
`JMP N`	N: c	-	Skače na mjesto u programu označeno s LBL N. N je konstantni broj u intervalu od 0 do 255. Prije izvršavanja ove naredbe provjerava se vrijednost na vrhu stoga (vrijednost ostaje na stogu). Ako je vrijednost na vrhu stoga logičko jedan, tada će se naredba skoka izvršiti.
`LBL N`	N: c	-	Postavlja se labela rednog broja N na prvu sljedeću naredbu.
`END`	-	-	Završava ciklus PLC-a

2.4. Popis zastavica

SM1.0	*Zero bit*, postavlja se u 1 ako je rezultat aritmetičke operacije jednak nuli. Niti jedna ga naredba ne postavlja nazad u nulu.
SM1.1	*Overflow bit*, vrijedi sve kao i za prethodnu zastavicu.
SM1.2	*Negative bit*, vrijedi sve kao i za prethodnu zastavicu.
SM1.3	*Divide by zero error*, isto kao i za prethodnu zastavicu.
SM1.4	Uvijek je postavljena na nulu.
SM1.5	Uvijek je postavljena na nulu.
SM1.6	Ova se zastavica postavlja ako je došlo do greške kod BCD pretvorbe (naredbe `BCDI` i `IBCD`).
SM1.7	Uvijek je postavljena na nulu.