LCAMP

Learner Centric Advanced Manufacturing Platform

OSNOVE AVTOMATIZACIJE S PLK

WP6 TOVARNA SODELOVALNEGA UČENJA

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Financirano s strani Evropske unije. Mnenja in mnenja, izražena, pa so izključno avtorjeva in ne odražajo nujno stališč Evropske unije ali Evropske izvršne agencije za izobraževanje in kulturo (EACEA). Niti Evropska unija niti EACEA ne moreta biti odgovorna za njih.

Partnerji LCAMP-a:

TKNIKA – Basque VET Applied Research Centre, CIFP Miguel Altuna LHII, DHBW Heilbronn – Duale Hochschule, Baden-Württemberg, Curt Nicolin High School, AFM – Spanish Association of Machine Tool Industries, EARLALL – European Association of Regional & Local Authorities for Lifelong Learning, FORCAM, CMQE: Association campus des métiers et des qualifications industrie du future, MV: Mecanic Vallée, KIC: Knowledge Innovation Centre, MADE Competence Centre Industria 4.0; AFIL: Associazione Fabbrica Intelligente Lombardia, SIMUMATIK AB; Association HVC Association of Slovene Higher Vocational Colleges; TSCMB:Tehniški šolski center Maribor, KPDoNE: Kocaeli Directorate Of National Education; GEBKİM OIZ and CAMOSUN college.

Povzetek dokumenta

Vrsta dokumenta:	Javni tečaj
Naslov	Osnove avtomatizacije s PLK
Avtor/avtorji	Marjan Bezjak
Pregledovalca	Samo Čretnik
Datelj	November 2025
Status dokumenta	1.0
Raven dokumenta	Zaupno do objave
Opis dokumenta	Ta predmet si prizadeva zajeti konfiguracijo PLK avtomatizacije in programsko strojno opremo, prednosti uporabe PLK v procesni avtomatizaciji, optimalno izbiro PLK in razširitvenih modulov za avtomatizacijo določenega procesa, osnovne funkcije PLK programiranja ter PLK programiranje na primeru praktične naloge.
To objavo navedite kot:	Bezjak M., 2025, PLK
Raven dokumenta	Javno

KAZALO

POVZETEK

V tem tečaju udeležencem nudimo znanje o konfiguraciji avtomatizacije PLK in programski strojni opremi, prednostih uporabe PLK pri avtomatizaciji procesov, optimalni izbiri PLK in razširitvenih modulov za avtomatizacijo določenega procesa, osnovnih funkcijah programiranja PLK, programiranju PLK na primeru praktične naloge (osnovno kombinacijsko in koračno krmiljenje).

1. SPLOŠNO O VAŠEM RAČUNALNIKU

Računalniki imajo velik vpliv na naša življenja v sodobnem času. Segajo v čas tik pred in po prvi in drugi svetovni vojni, prvi uporabni računalniki (npr. Z3, ENIAC) pa so bili ustvarjeni ob koncu druge polovice 20. stoletja in so bili sprva dostopni le vojski in večjim raziskovalnim ustanovam. Ko se je polprevodniška tehnologija v 50. in 60. letih močno razvila, so računalniki postali manj zahtevni, zmogljivejši in cenejši, zaradi česar so bili bolj dostopni širši populaciji.

To je privedlo do razvoja idej, da lahko računalnik upravlja tudi bolj zapletene procese, ne le obdelavo numeričnih podatkov. Največji problem je bil, da so bili še vedno razmeroma dragi in veliki, zato primerni le za omejen nabor uporab.

Z razvojem mikroprocesorjev se je situacija tako močno spremenila, da je računalnik postal "najbolj revolucionaren izdelek v zgodovini človeštva" (Gordon Moore, ustanovitelj INTELA). Danes si življenja brez mikroprocesorjev ni več mogoče predstavljati, saj jih srečujemo na vsakem koraku. V gospodinjstvu se ne znajdemo več brez pralnega ali pomivalnega stroja, brez mikrovalovne pečice in nekaterih drugih aparatov, prav tako pa si je težko predstavljati življenje brez telefona, radia, televizije, kamere… Tudi avtomobili so opremljeni z mikroprocesorskimi sistemi.

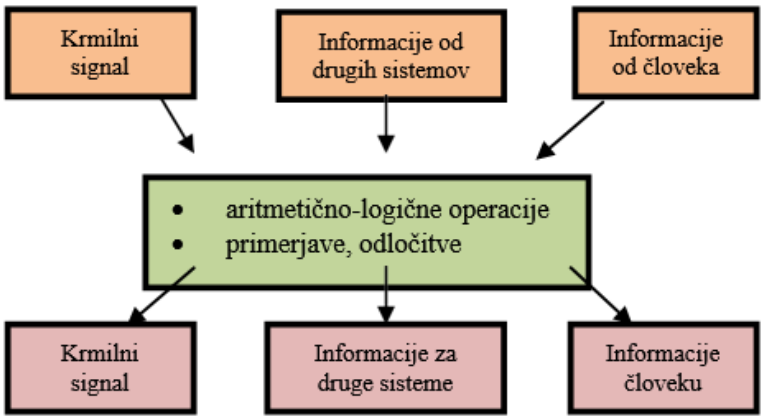
V komercialnih in industrijskih sistemih so sistemi za nadzor proizvodnje, merilne naprave, roboti itd. zgrajeni na osnovi mikroprocesorjev, ki so povezani z lokalnimi omrežji, pri čemer manjši računalniki za nadzor sistemov v končnih proizvodnih celicah nadzorujejo zmogljivejši. Nato upravljajo produkcijo z uporabo baz podatkov, upoštevajo poslovne odločitve itd.

V primerjavi z diskretno logiko mikroračunalniško krmiljenje sestavlja manj komponent, njihov razvoj je hitrejši in enostavnejši, testiranje je lažje, cena je nižja, možnost napake pa manjša. Isto elektroniko lahko uporablja več različnih naprav, saj so standardna mikroračunalniška vezja univerzalno uporabna, in zahvaljujoč tem vezjem se lahko hitro prilagodijo spremembam, saj je potrebna le nova konfiguracija in program. Vzdrževanje je enostavnejše (manj elementov, univerzalni moduli, možnost samotestiranja, standardizirani postopki).

1.1. VLOGA RAČUNALNIKA

Računalnik si lahko predstavljate kot črno skrinjico, ki je inteligenten del med vhodom in izhodom. Da bi računalnik lahko obdeloval podatke, mora biti sposoben:

- preberi napisani program in ga izvede,
- izvajanje aritmetičnih in logičnih operacij na podatkih,
- primerjajte podatke in sprejemajte odločitve,



Slika 1: Računalniško delovanje

1.2. RAČUNALNIŠKO DELJENJE

Informacijski računalniki – v pisarni pomagajo sekretarju pri vodenju podjetja, v razvojnem oddelku pa inženirjem v razvojnem oddelku pri razvoju novih izdelkov.

Krmilni računalniki, ki nadzorujejo stroje in proizvodne linije v proizvodnem obratu, so večinoma mikroračunalniki, saj imajo poleg mikroprocesorja le osnovne komponente (RAM, ROM, VHOD/O vmesnike). PC-je je mogoče uporabljati tudi za krmilne računalnike z ustreznimi programskimi orodji in krmilnimi vmesniki.

CAD/CAM (računalniško podprto načrtovanje/računalniško podprta proizvodnja): Lokalna računalniška omrežja omogočajo povezavo informacijskih in krmilnih računalnikov s CAD/CAM sistemi in kasneje s sistemi za vodstveno spremljanje proizvodnje.

1.3. MIKRORAČUNALNIK KOT PROCESNI KRMILNIK PROGRAMABILNI LOGIČNI KRMILNIK (PLK)

Njena funkcija je nadomestiti človeka pri nadzoru strojev in proizvodnih procesov v industriji. Klasičen nadzor s trajno aktiviranim programom:

Elektromehanski krmilni sistemi: Osnovni gradniki so releji, stikala, ventili. Način delovanja določata funkcionalni načrt in vezavna shema.

Elektronski krmilni sistemi: Sestavljeni so iz logičnih elementov v obliki čipov, ti elementi pa so med seboj povezani po zasnovi. Zasnova je zasnovana na zahtevah tabele regularnosti ali logične funkcije z uporabo metod minimizacije digitalnih vezij. Funkcija krmilne operacije je

določena z medsebojno povezavo elementov. Tudi tukaj pravimo, da program delovanja določa program.

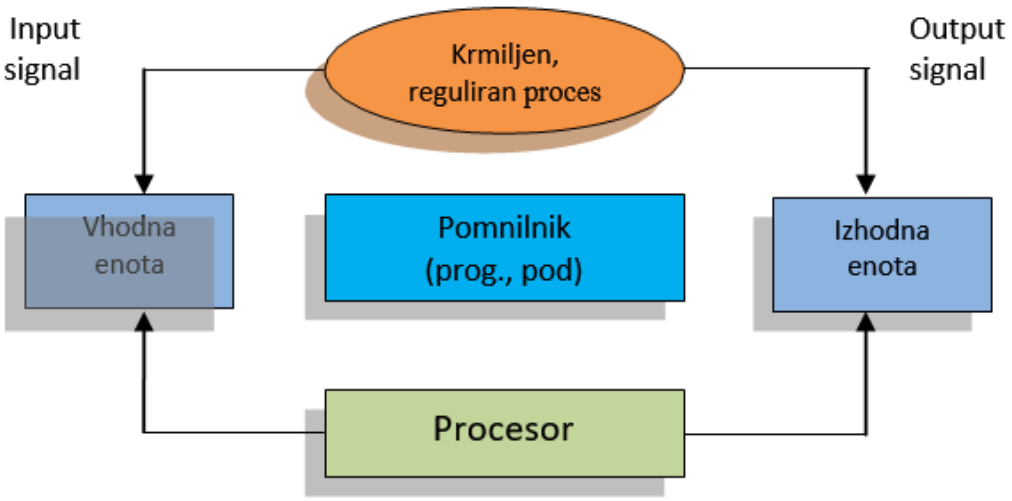
Programabilni logični krmilnik (PLK):

V primeru velikih krmilnih sistemov za nadzor in nadzor industrijskih procesov pa moramo običajno med testnim obdobjem spremeniti operativni program. Včasih se tudi tehnološki proces pogosto spreminja in zahteva nov program. Zato obstaja potreba po večji prilagodljivosti krmiljenja glede na različne potrebe. S takšnimi kontrolami lahko spremenite program, ne da bi morali spreminjati ožičenje.

Krmilni sistemi, ki temeljijo na PLK, sestavljajo:

- naprave ali strojne opreme, ali strojne opreme (krmilnik, pretvorniki, kontaktorji, stikala, ožičenje itd.)
- Programska oprema – Programska oprema (program krmilnih operacij, ki se vnese v programski del pomnilnika). Za spremembo funkcije krmilne operacije je potrebno le zapisati nov program v programski pomnilnik, medtem ko spremembe strojne opreme praviloma niso potrebne.

V nadzorovanem ali reguliranem procesu oddajniki signalov (senzorji) zagotavljajo podatke o stanju in vrednosti posameznih količin (temperatura, tlak, hitrost, pretok, položaj itd.). Ti podatki se v vhodni enoti obdelajo v binarne vrednosti, ki jih lahko obdela procesor. Procesor bere posamezne vhodne podatke v skladu s programom in določa stanja v procesu. Na podlagi stanj v procesu in programu daje ukaze v obliki napetostnih impulzov izvršnim elementom krmiljenja (releji, svetlobni in zvočni detektorji, zasloni, servomotor, koračni motor itd.) preko izhodne enote.



Slika 2: Prikaz blokov in delovanje PLK

S programabilnim logičnim krmilnikom (PLK) lahko izvajate naslednje procesne funkcije:

- zajemanje in urejanje procesnih podatkov,
- izvajanje nadzora in regulacije,

- zaščita procesov v konfliktnih situacijah,
- analizo in oceno rezultatov postopka.

Povzetek:

Danes ima računalnik velik vpliv na naša življenja. Ob koncu 20. stoletja so bili računalniki na voljo le vojski in nekaterim raziskovalnim ustanovam, nato pa je tehnologija v 50. in 60. letih močno napredovala in računalniki so postali manj zahtevni, zmogljivejši in cenejši. Vendar pa je računalnik, ko so se pojavili mikroprocesorji, postal najbolj revolucionaren izdelek v zgodovini človeštva, danes pa se z njimi srečujemo na vsakem koraku. Računalniki so razdeljeni na informacijske, krmilne in CAD/CAM računalnike.

Programabilni logični krmilnik (PLK) nadomešča ljudi pri nadzoru in upravljanju strojev ter proizvodnih procesov v industriji. Sestavljena je iz naprav ali strojne opreme (strojne opreme) in programske opreme (programske opreme). Uporablja se lahko za zajem in urejanje procesnih podatkov, izvajanje virov in predpisov, zaščito procesov v konfliktnih situacijah ter analizo in ocenjevanje rezultatov procesov.

Vprašanja:

1. Kako lahko definirate pojem računalnik?
2. Kaj mora računalnik vedeti za obdelavo podatkov?
3. Od kod lahko računalnik pridobi podatke in kam jih lahko pošlje?
4. Kakšna je razlika v uporabnosti med informacijskimi in krmilnimi računalniki?
5. Kaj pomeni CAD/CAM?
6. Kaj so trajno pritrjeni kontrolni sistemi?
7. Kakšne so prednosti programabilnih logičnih krmilnikov v primerjavi s trajno krmilno ožičenjem?
8. Katere procesne funkcije lahko izvaja programabilni logični krmilnik (PLK)?

2. METODE RAČUNALNIŠKEGA PROGRAMIRANJA

V tem poglavju boste spoznali pomen računalniškega programa in značilnosti programskih jezikov, pa tudi orodja in faze pri ustvarjanju programa. Pojasnjeni so pojmi računalnik, računalniški program, programer in programiranje. Programski jeziki so predstavljeni z primeri in razdeljeni glede na metodo programiranja, čas prevajanja in način vnosa. Sledi predstavitev programov za zajem in obdelavo podatkov Lego Mindstorm, Crocodile Technology in Labview (http://www.ni.com/labview/applications/) ter osnovni pristop k programiranju.

2.1. RAČUNALNIK, RAČUNALNIŠKI PROGRAM IN PROGRAMIRANJE

Računalnik je stroj za obdelavo informacij, ki izvaja operacije ali ukaze v skladu z računalniškim programom.

Računalniški program je navodilo stroju, kaj in kako naj obdeluje informacije.

Programiranje je zapisovanje naloge v obliki, ki je procesorju razumljiva.

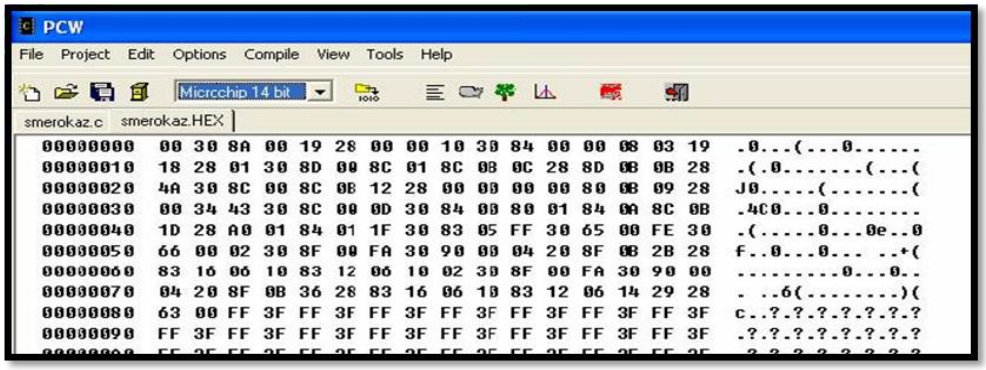
Računalniški programer ali razvijalec programske opreme piše računalniške programe. Pri tem mora poznati probleme in rešitve nalog, ki jih bo računalnik kasneje rešil s pomočjo pisnega programa.

2.2. DELITEV PROGRAMSKIH JEZIKOV Z PRIMERI

a) **Strojna koda**, strojni jezik ali strojno besedilo programa. Strojna koda ali strojni jezik) je besedilo ali koda v izvršljivih datotekah, ustvarjenih iz izvornega besedila s strani prevajalnikov ali prevajalnikov. Strojno besedilo programa sestavljajo zaporedja izvršljivih strojnih ukazov, ki jih na osebnih računalnikih izvaja centralna procesna enota. Vsak procesorski sistem ima svojo arhitekturno zasnovo (platformo). To pomeni, da je strojni jezik med različnimi procesorji zelo različen (vendar enak za tiste z isto zasnovo), zato je treba za vsako platformo uporabiti drugačen prevajalnik. Prevajalnik) za prevod izvorne kode v strojno opremo. Čeprav je najbolj odvisna od zasnove procesorja, je platforma nabor lastnosti vseh delov računalnika. To vključuje tudi operacijski sistem in druge pomembne programe. Programska oprema) ali strojni deli (Eng. Hardware). Centralna procesna enota neposredno razume objektno kodo, vendar za uspešno izvajanje na določenem operacijskem sistemu potrebujemo tudi določeno notacijo izvajanja. Izvedljiv format), ki ga določa vmesnik ABI (ABI). Application Binary Interface) operacijskega sistema. Tako izvedljive datoteke vsebujejo strojno kodo določene izvršljive datoteke, kar je glavni razlog za neprenosljivost računalniških programov med posameznimi operacijskimi sistemi. GNU/Linux in družina BSD uporabljata na primer ELF izvedljivo notacijo. Izvedljiv in povezljiv format), Mac uporablja Mach-o, medtem ko Windows PE (Eng. Portable and

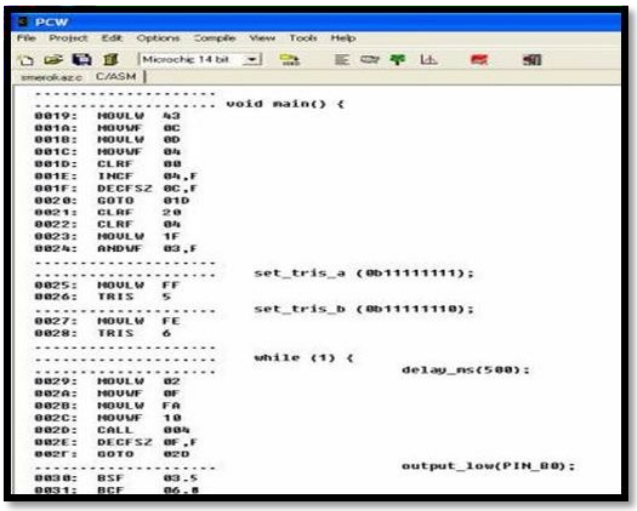
Executable Format). Izvršljiva datoteka s strojno kodo je lahko tudi brez izvršljivega zapisa. Datoteka, ki jo je mogoče zagnati brez operacijskega sistema na centralni procesorski enoti, se imenuje ploska izvršljiva datoteka. Ravna binarna datoteka. Strojna koda se pogosto imenuje prva generacija programskih jezikov.

Primer strojne kode je prikazan na sliki 3.



Slika 3: Primer programa v strojni kodi

b) **Asembler** ali asembler je nizkonivojski programski jezik druge generacije (prva generacija jezika je strojna koda), ki je napisan z uporabo mnemotehnik. Na splošno so mnemotehnike berljive različice binarnih zaporedij (ničel in enic), ki jih je treba prevesti, da se dobi koda, ki je razumljiva osrednjemu procesorju. Natančneje, mnemotehnike so ukazne kode. Operativne kode, okrajšano Opcodes), ki so definirane v centralni procesni enoti glede na arhitekturo ISA (ISA). Arhitektura nabora ukazov). To kodo je nato običajno treba povezati z določenimi strukturami, da dobimo delujoč izvršljiv program. Določena asemblerska programska oprema, kot je FASM, preprosto zamenja mnemonike in operande oziroma parametre z ustreznimi navodili v strojnem programskem jeziku. Na ta način dobimo ploske binarne izvršilne datoteke, ki vsebujejo (odvisno od izkušenj posameznega programerja) strojno kodo z izjemno algoritmično učinkovitostjo. Koda programa v povzetku je prikazana na sliki 4.



Slika 4: Primer programa v združevanju

Visokonivojski jezik je programski jezik, zasnovan tako, da izpolnjuje zahteve programerja. Ne temelji na notranji strojni kodi določenega računalnika. Uporabljamo visokonivojske jezike

za reševanje problemov in jih pogosto imenujemo problemsko usmerjeni jeziki – na primer, BASIC je bil zasnovan tako, da začetnikom omogoča hitro učenje, COBOL se uporablja za pisanje poslovnih programov, FORTRAN pa za reševanje znanstvenih in matematičnih problemov. Nizkonivojski jeziki, v nasprotju z visokonivojskimi, močno odražajo značilnosti strojne kode določenega računalnika in jih zato imenujemo tudi strojno usmerjeni jeziki. Za razliko od nizkonivojskih jezikov so visokonivojski jeziki razmeroma enostavni za učenje. Njihovi ukazi so podobni človeškemu jeziku, zato programerju ni treba podrobno poznati notranje strukture računalnika. Vsak ukaz visokonivojskega jezika je enakovreden več strojnim ukazom. Programi na visoki ravni so zato bolj kompaktni kot ekvivalentni nizkonivojski programi. Vendar pa mora biti vsak visokonivojski program preveden v strojni jezik, preden se lahko zažene – bodisi s prevajalnikom ali interpreterjem. Visokonivojski jeziki so zasnovani tako, da so prenosljivi. To pomeni, da se lahko program, napisan v visokonivojskem jeziku, zažene na katerem koli računalniku, ki ima prevajalnik ali interpreter za ta jezik. Visokonivojski jezik je C++, prikazan na sliki 5.

```
*********************************************************************

#case              // Da loči male in velike črke
#ZERO_RAM      // Briçe RAM po startu programa
/*********************************/

#include <16F84.h>
#USE DELAY (CLOCK=4000000)
#fuses XT,WDT,PUT,NOPROTECT

void main() {

   set_tris_a (0b11111111);
   set_tris_b (0b11111110);

   while (1) {
         delay_ns(500);
         output_low(PIN_B0);
         delay_ns(500);
         output_high(PIN_B0);
            }   // konec while 1

   }     // konec main
```

Slika 5: Primer programskega jezika v C++

Da bi strojni ukazni jezik približali človeku, uporabimo programski jezik kot vmesno stopnjo. Programski jezik ni enak naravnemu jeziku. Slika 5 prikazuje programsko kodo v okolju C++, kjer program napišete z uporabo nabora ukazov, ki jih zagotavlja programsko orodje, na primer while. To nam predstavi zanko, in lahko uporabite ukaz include za vključitev potrebne knjižnice.

Naravni jezik ima zapletena pravila in obsežna slovnična pravila.

Programski jezik je nedvoumen in formalno opisljiv – bližje strojnemu jeziku.

Programski jezik mora biti sposoben:

- **Opis problema** (opis podatkov, rezultatov in odnosov med njimi). Primer: Izračun izdelka
  - opis števil, rezultat – njihov produkt;
- opis postopka (opis zaporedja korakov, ki nas pripelje do rezultata). Algoritem množenja, napisan v osnovnih korakih.

Programski jezik je zbirka dogovorjenih ukazov, običajno v angleščini, ki izvajajo določeno dejanje. Višji programski jezik ni vezan na vrsto mikroprocesorja. Izbrati morate ustrezen

prevajalnik, ki prevede napisani program iz izvorne kode (npr. C) v ustrezno strojno kodo, ki je razumljiva procesorju (npr. EXE). Vsak ukaz je preveden v več strojnih kod. Programer potrebuje več programerskih orodij. Izvorna koda se vpiše v urejevalnik.

Editor – se s pomočjo ustreznega prevajalnika prevede v modul strojne kode.

Prevajalnik – preveden modul skupaj s predhodno pripravljenimi moduli iz knjižnice se združi v izvršljiv program z uporabo konektorja.

Linker – delovanje programa se postopoma testira s pomočjo razhroščevalnika.

Debugger – običajno so vsa ta orodja združena skupaj v programskem okolju z meniji, npr. DEVCPP, CCSC, Visual Studio NET itd.

2.3. PROGRAMSKA DIVIZIJA

JEZIKI Glede na čas prevajanja:

1. Interpreter (INTERPRETER – ČAS IZVAJANJA).
2. Prevajalnik tvori kodo procesorja, na katerem prevajalnik teče. (npr. TurboC, Delphi za PC).
3. KRIŽNI prevajalnik deluje na razvojnem računalniku (npr. PC) in ustvari programsko kodo za drugo vrsto mikroprocesorja, za različne mikrokrmilnike (npr. PICC, BASCOM in industrijske krmilnike (npr. Mitshubishi-Melsec MEDOC ali Siemens STEP 7 – http://www.crocodile-clips.com/en/Crocodile_Technology/).

Glede na metodo programiranja:

- **Proceduralni** – BASIC, C-jezik, PASCAL (primeren za mikrokrmilnike in večje računalnike).
- **Objektno usmerjeni** – C++, Delphi, JAVA (uporabljajte objektne razrede CLASS).

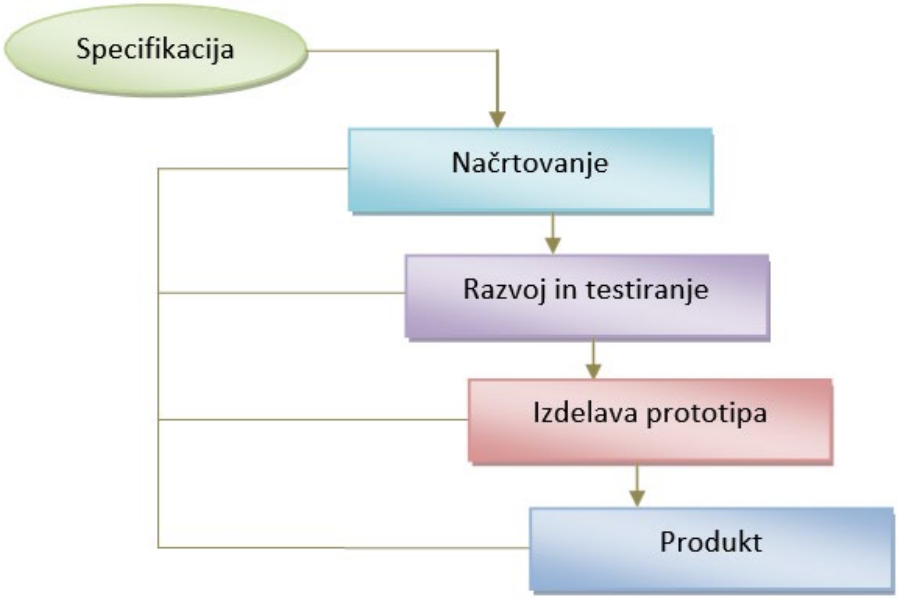
Glede na to, kako se program vpiše:

- **Besedilni (tekstovni)** – to so programski jeziki, ki so bili doslej omenjeni.
- **Grafično –** Visual Basic, Visual C, LabVIEW.

To so jeziki nove generacije; Programiranje poteka v grafičnem vmesniku. Programer zlaga kocke, module, ustrezna programska koda pa se oblikuje v ozadju.

2.4. PRISTOP K PROGRAMIRANJU

Razvoj programa obravnavamo podobno kot razvoj drugih izdelkov ali izdelkov. Proces načrtovanja in razvoja:



Slika 6: Proces načrtovanja in razvoja

2.4.1.FAZE PROGRAMIRANJA

a) Načrtovanje (algoritem, diagram poteka).
b) Kodiranje (pisanje v izbranem programskem jeziku).
c) Prevajanje in testiranje (prevajalnik najde pravopisne napake, razhroščevalnik pa omogoča tudi iskanje logičnih napak).
d) Dokumentacija programa (komentarji in opisi za kasnejše razumevanje napisanega ter navodila za uporabnike programa).

2.4.2.ALGORITEM

Rešitev naloge se izvaja v korakih, v algoritmu.

Definicija: Algoritem je korak za korakom opis postopka, s katerim je mogoče rešiti problem.

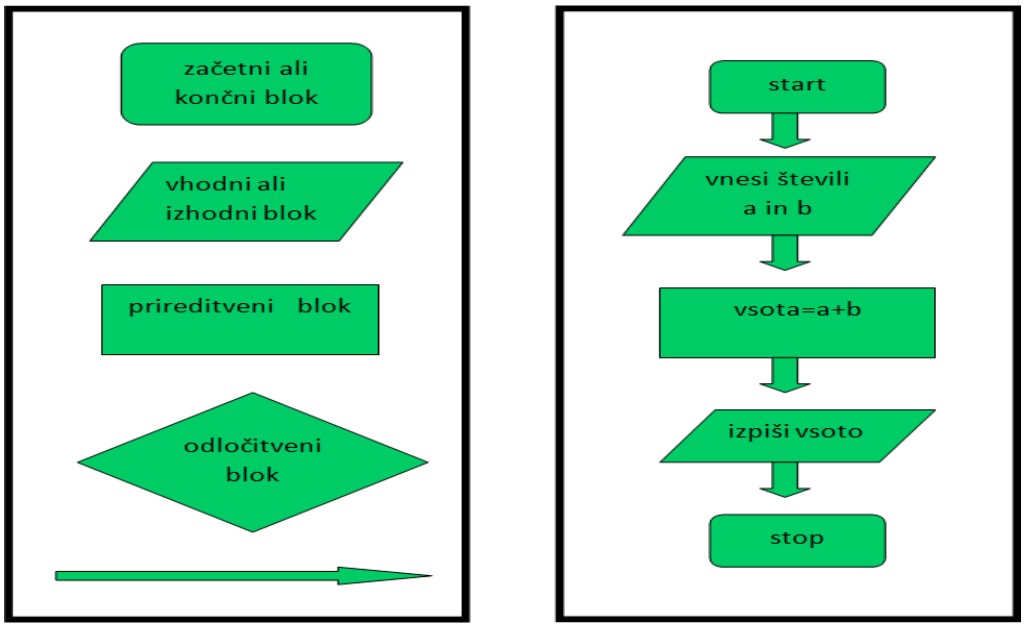
Primer: Ustvarite program, ki sešteje dve številki

- Deklariranje spremenljivk a, b, c;
- Vnesite prvo številko in jo vnesite v spremenljivko a;
- vnesite drugo številko in jo vnesite v spremenljivko B;
- Dodajte spremenljivki a in b;
- Natisni spremenljivko C.

2.4.3.DIAGRAM POTEKA

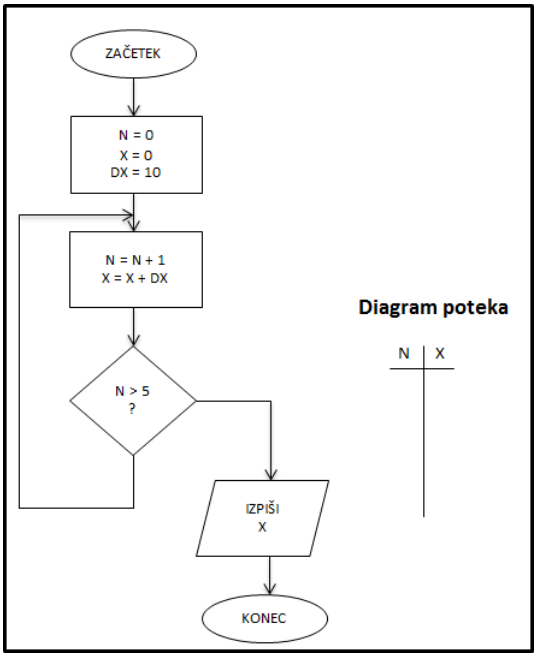
Gre za grafično prikazan algoritem, torej potek dogodkov, opisan na človeku razumljiv način. Uporablja se pri načrtovanju programov in tudi pri opisovanju drugih dejavnosti (npr. zdravnik na domu v obliki knjige, občinska navodila za pridobivanje dovoljenj itd.).

Pametno je pristopiti k problemu programske opreme v obliki diagrama poteka, da postopoma pojasnimo implementacijo in predvidemo, kaj bo računalnik moral storiti.



Slika 7: Bloki diagramov poteka

Nekateri učni programi vam omogočajo, da program napišete neposredno v diagramu poteka. Program, preveden na ta način, je mogoče testirati ali prevesti v besedno obliko, kar močno poenostavi učenje programiranja.



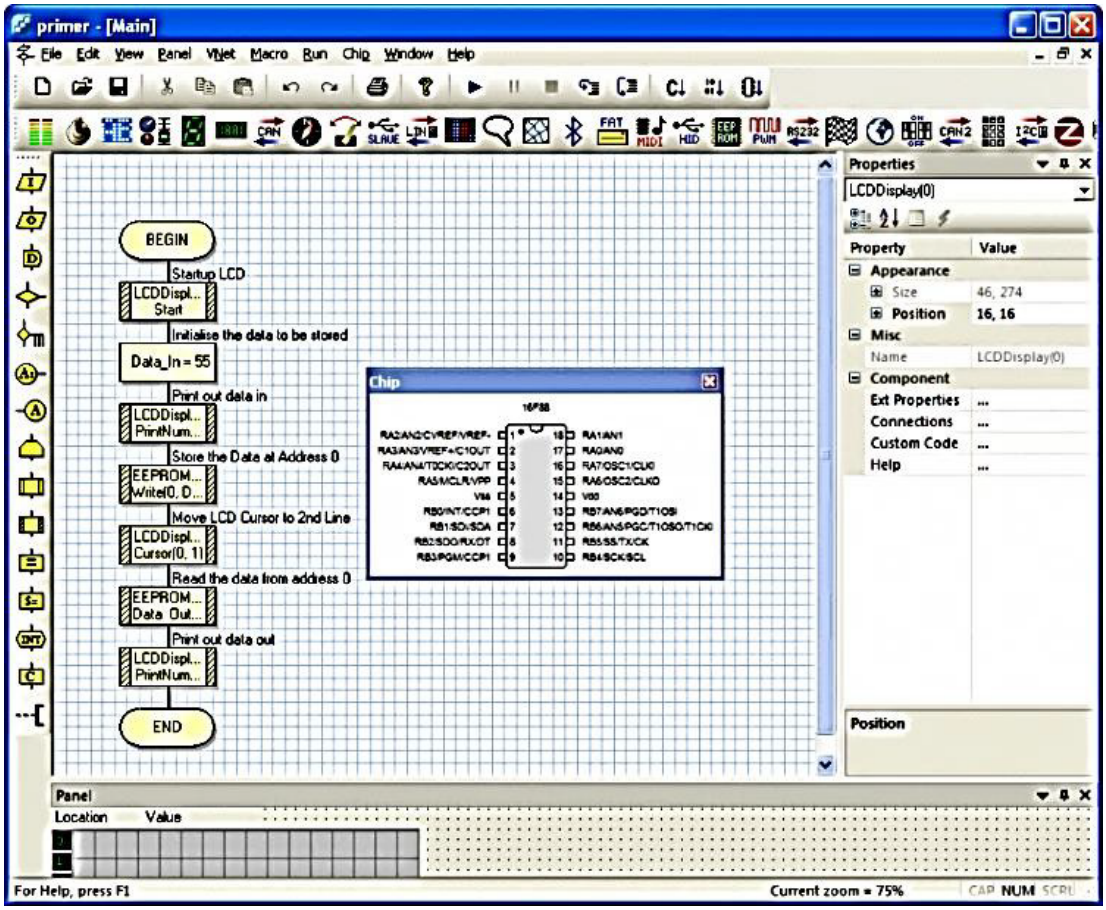
Slika 8: Primer diagrama poteka

2.5. PROGRAMIRANJE LEGO MINDSTORM IN FLOWCODE MEHATRONIČNIH NAPRAV

LEGO MINDSTORM in SIMBOT sta zelo primerna za učenje programiranja. Sestavite avto z vgrajenim mikroračunalnikom iz LEGO kock. Napišete program za ta računalnik, ki nadzoruje motorje in upošteva signale senzorjev.

Slika 9: Lego Mindstorm
Vir: http://mindstorms.lego.com/en-us/Default.aspx

Ena od programskih možnosti je Flowcode.

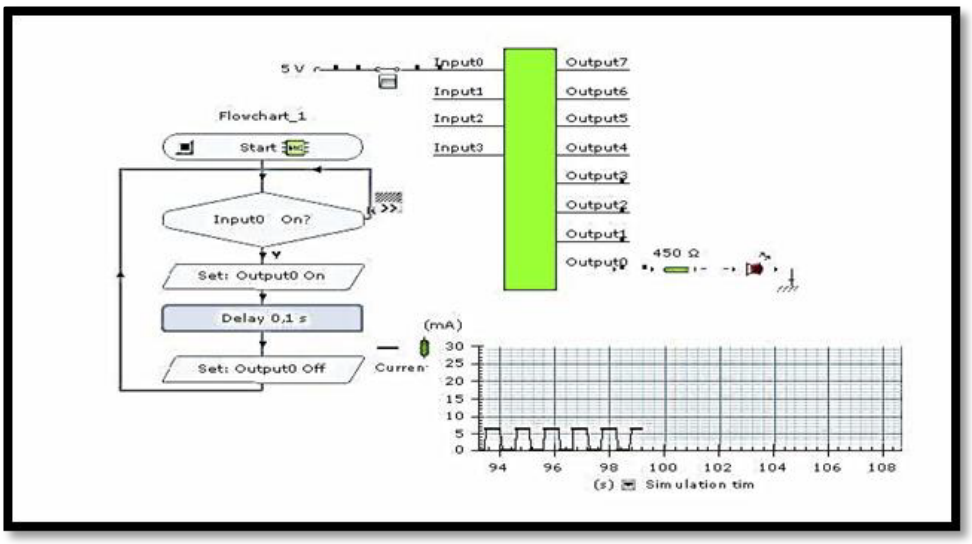


Slika 10: Primer programa v Flowcode
Vir: http://www.imagesco.com/microcontroller/flowcode-compiler.html

2.6. PROGRAM SIMULACIJE TEHNOLOGIJE KROKODILOV

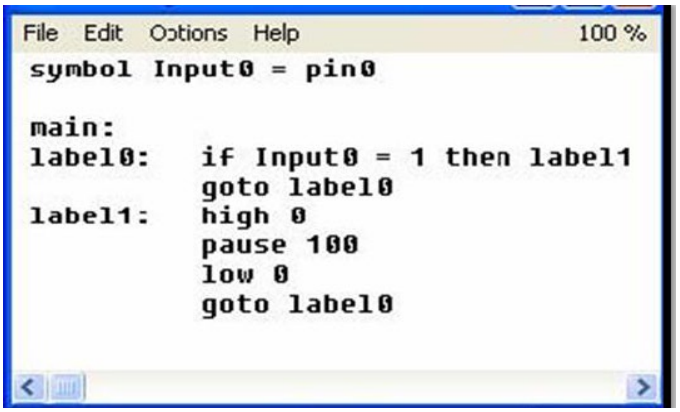
Program simulacije Crocodile Technology je primeren za učenje programiranja mikrokrmilnika PIC. V programu ustvarite električni diagram vezja z elementi iz knjižnice, v naslednjem koraku pa lahko izdelate tudi diagram toka programa. Na diagramu lahko uporabite katerokoli programsko ukaze, pogojne skoke, zanke. Prednost, ki nam omogoča tak način programiranja, je zagotovo večja preglednost in lažja predstavitev dogajanja v programu. Ko se simulacijski program zažene, je vidna tudi aktivnost vsakega vhoda ali izhoda.

Izbrani program omogoča tudi merjenje vrednosti signala z virtualnimi merilniki iz grafa. Na primer, lahko opazujete, kaj se dogaja na izhodu mikrokrmilnika.



Slika 11: Primer programskega okolja za programiranje mobilnih robotov
Vir: http://www.crocodile-clips.com/en/Crocodile_Technology/

Program je mogoče tudi izvoziti v BASIC ali neposredno vnesti prek programerja v pravi čip.

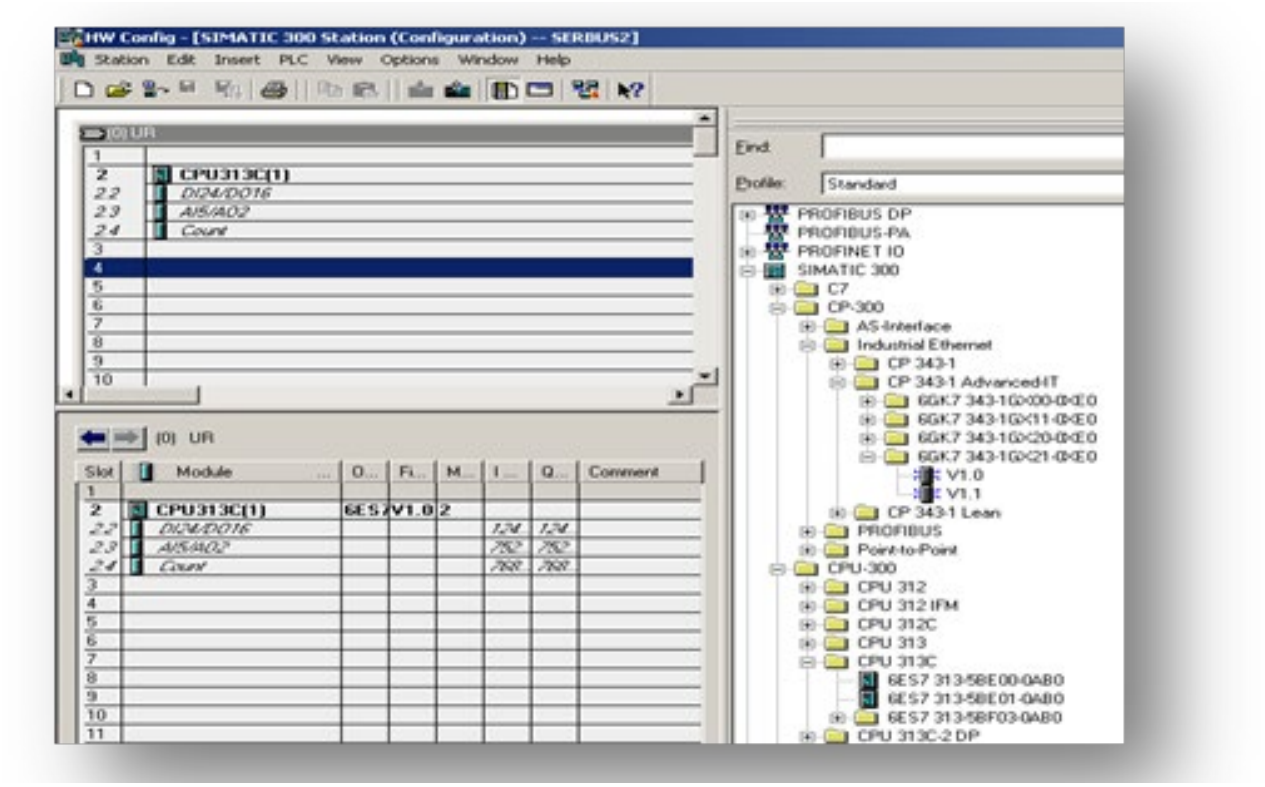


Slika 12: Primer programske kode v Basicu
Vir: http://www.justbasic.com/learnmore.html

2.7. PROGRAMIRANJE MIKROKRMILNIKOV – PLK

Prav tako je programiranje industrijskih krmilnikov grafično in hitro razumljivo. PLK programska orodja podpirajo tri metode programiranja:

- preko električnega krmilnega sistema releja (LADDER),
- z elementi digitalne elektronike – blok diagram (FBD),
- z ustnimi ukazi – ukazi (INSTR),
- in nekaj dodatnega grafičnega načina.



Slika 13: Primer programskih jezikov PLK
Vir: http://support.automation.siemens.com

Takšna programska orodja omogočajo spremljanje in testiranje programa po nalaganju v krmilnik, nekatera pa omogočajo tudi simulacijo delovanja na računalniku (izven povezave).

2.8. PROGRAMIRANJE VIRTUALNE INSTRUMENTACIJE – LABVIEW

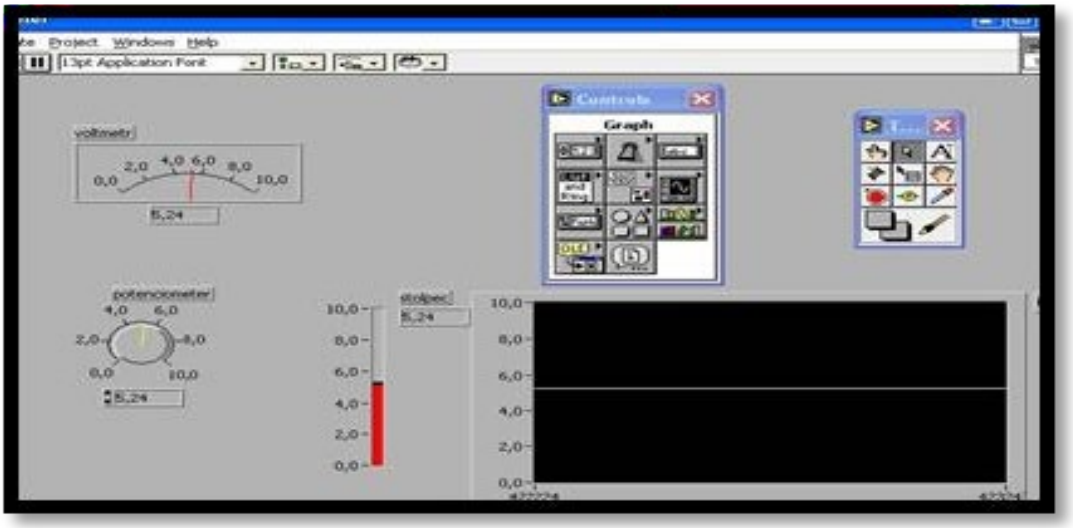
Računalnik se lahko uporablja kot industrijski krmilnik pri avtomatizaciji merilnih procesov. Za ta namen se poleg opreme (merilni vmesniki za računalnike itd.) pridobijo tudi grafična programska okolja. V Sloveniji je oprema podjetja široko uporabljena

National Instruments in njihovo programsko orodje LabVIEW (http://www.ni.com/labview/applications/).

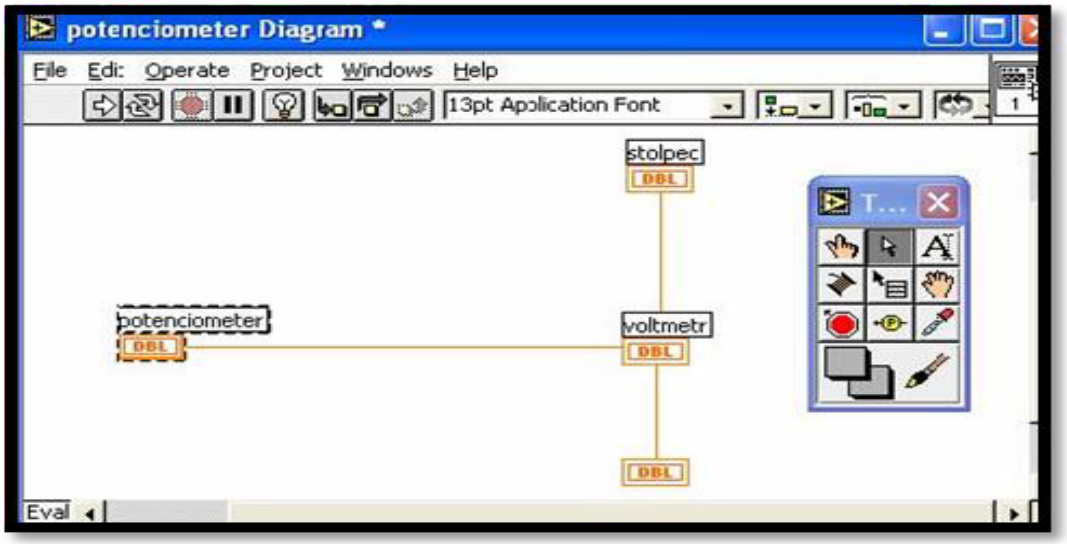
To programsko orodje je zasnovano za avtomatizacijo meritev. Potrebujete merilne module in vmesnike, ki jih vgradite v računalnik (DAQ kartice), ti vmesniki pa prenašajo analogne in digitalne signale med števcem in računalnikom. Klasični merilni instrumenti z računalniško povezavo so prav tako uporabni, saj so lahko zunanje enote.

Ustvarjanje programov je preprosto in pregledno. Dve programski okni sta odprti hkrati: sprednja plošča (Panel), na kateri uporabljate in rišete prikaze in kontrolne gumbe, ter blok shema (Diagram), kjer povezujete elemente s sprednje plošče z žicami v ozadju in omogočate pretok podatkov. Iz bogate knjižnice funkcij izberete ustrezne bloke za obdelavo merilnih podatkov, ki jih povežete s shemo.

Prikazi in kontrolni gumbi za VIM so nameščeni na sprednji plošči (VI-Virtual Instrument).



Slika 14: Demonstracija programiranja v LabVIEW
Vir: http://www.ni.com/labview/



Slika 15:Diagram
Vir: http://www.ni.com/labview/

V ozadju, v programskem oknu (diagram), povežete vire in uporabnike podatkovnega toka ter zgradite zanke programa.

Program lahko prevedeš v izvršljivo (exe) datoteko ali ga zaženeš neposredno v razvojnem okolju. Računalnik prebere izmerjene podatke iz zunanjih merilnih instrumentov ali merilnih vmesnikov in na podlagi njih odloči o ustreznem ukrepu preko izhodov iz računalnika, zapisuje datoteko na disk, poroča na zaslonu ali prek interneta.

Delovna orodja lahko izberete iz palete TOOLS, ki je dostopna v obeh oknih. Pri gradnji sprednje plošče imate paleto CONTROLS na desnem gumbu miške, v blok shemi pa paleto

FUNCTIONS na istem mestu, kjer je knjižnica funkcij. Dobra pomoč v angleščini nam olajša delo in odkrivanje napak. Ustvarjeni program iz obeh oken je shranjen pod imenom

*.VI (virtualni instrument). Lahko si pomagaš učiti se z množico primerov, ki so bili že narejeni.

Program začnemo na  delovni vrstici.

Če želimo, da se ponovi samodejno, potem z . 

Za boljše razumevanje, kako deluje, vključimo prikaz podatkovnega toka z . 

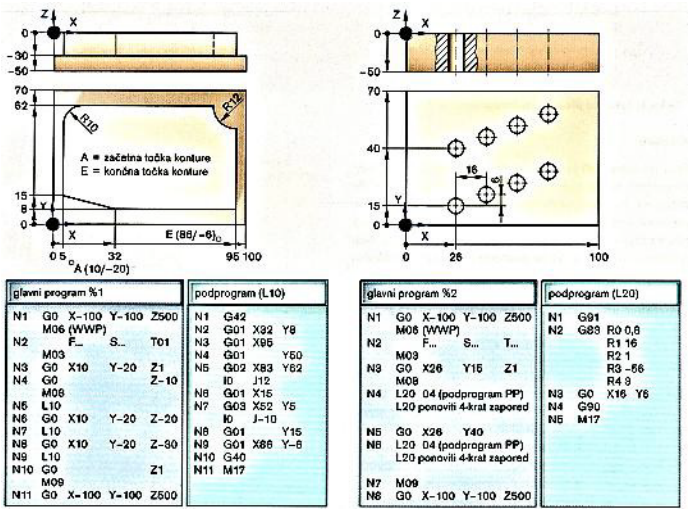
2.9. PROGRAMIRANJE NUMERIČNO VODENIH STROJEV – CNC G-CODE

CNC (računalniško numerično krmilno) stroj je stružnica ali podoben stroj za obdelavo kovin, ki ga nadzoruje računalniški krmilnik. G kode so standard za številčno krmiljene strojne stroje.

CNC programerji pišejo programe iz tehnoloških risb in jih vnašajo v krmilnik stroja. Dobri mehanski CAD programi lahko samodejno ustvarijo G-kodo iz narisanega 3D objekta in ustvarijo objekt na CNC stroju.

Slika 16: CNC stroj

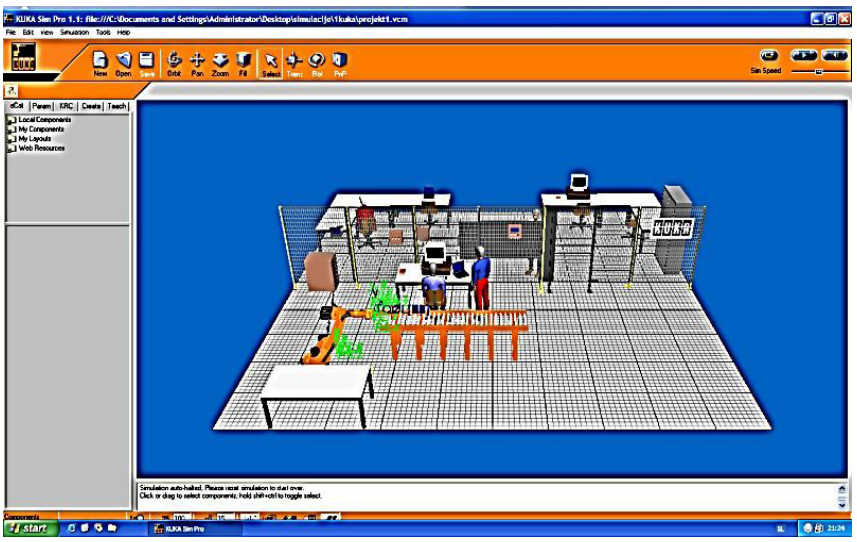
Spodnja slika prikazuje primer kode za programiranje CNC rezkalnega stroja, ki ga upravlja računalnik. Pod vsakim od objektov je prav tako označena koda za konstrukcijo objekta.



Slika 16: Primer CNC kode z vključenimi podprogrami
Vir: J. Bartenschlager, MECHATRONICS 2009

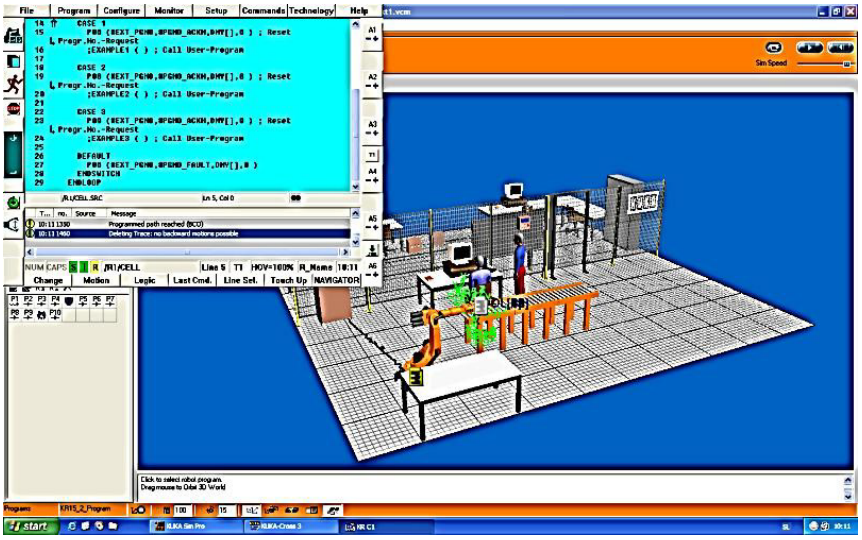
2.10. ROBOTSKI PROGRAMSKI JEZIK: INDUSTRIJSKI KLJUKASTI ROBOTI

Simulacijski program vam omogoča, da se v simulatorju naučite gibov in dejanj. Lahko zgradiš robotsko celico in napišeš robotski program. Nato ga lahko preizkusite in prenesete na robotski krmilnik. Preden zgradite pravo robotsko celico, je dobro narediti simulacijo in v njej predstaviti vse robne delovne pogoje.



Slika 17: simulacijski program KUKA Sim Pro
Vir: http://www.kuka-robotics.com/en/products/software/

Poleg razvojnega programskega okolja lahko simulator povežete tudi z Office Lite, ki vam omogoča simulacijo programske naprave.



Slika 18: Simulacijski program
Vir: http://www.kuka-robotics.com/en/products/software/

Zgornja slika prikazuje povezavo med simulacijskim programom KUKA Sim Pro in simulacijskim podprogramom operacijske plošče Office Lite.

V primeru povezave programske opreme Kuka Sim pro in Office Lite 4.1 lahko na računalniku izvedete enaka dejanja kot na pravem robotu. To je prikazano na zgornji sliki. Lahko izvajate gibe, zanke, nastavljate hitrosti in končni program neposredno prenašate prek omrežja na robotski KRC krmilnik.

Slika 19: Naprava za programiranje snemanja gibanja (Teach Box KR C2)

Povzetek:

Programske jezike lahko v osnovi razdelimo na tri dele: strojno kodo, asembler in višje programske jezike. Strojna koda programa je sestavljena iz zaporedij izvršljivih strojnih ukazov, ki jih na osebnih računalnikih izvaja centralna procesna enota.

Collection (asembler, asembler) je nizkonivojski programski jezik druge generacije (prva generacija je strojna koda), ki je napisan z uporabo mnemotehnik. Na splošno so mnemotehnike berljive različice binarnih zaporedij (ničel in enic), ki jih je treba prevesti, da se dobi koda, ki je razumljiva osrednjemu procesorju.

Visokonivojski jezik je programski jezik, zasnovan tako, da izpolnjuje zahteve programerja. Ne temelji na notranji strojni kodi določenega računalnika. Za reševanje problemov uporabljamo visokonivojske jezike in jih pogosto imenujemo problemsko usmerjeni jeziki.

Pred začetkom programiranja je zagotovo nujno rešiti sam problem, ki ga bomo rešili s programom. To počnemo z algoritmom, kjer problem razdelimo na posamezne korake in je lažje rešiti.

Vprašanja:

1. Kaj pomeni izraz »računalniški program« in kaj pomeni »programer«?
2. Kakšne so značilnosti programskega jezika in kaj omogoča?
3. Katera programska orodja programer potrebuje in za kaj so namenjena?
4. Razdelimo programske jezike in navedite nekaj primerov.
5. Katere faze mora programer prehoditi pri ustvarjanju programa?
6. Kaj pomeni beseda algoritem?
7. Napiši algoritem za kavni aparat in ga razloži.
8. Kateri simboli se uporabljajo v diagramu poteka in kaj pomenijo?
9. Katere metode pisanja programov se uporabljajo v programabilnih logičnih krmilnikih?
10. Kako izgleda program za CNC stroj? Napiši vsaj 3 ukaze.
11. Kako so programirani roboti? Opišite primer.

3. OSNOVE PROGRAMSKIH JEZIKOV

V poglavju PROGRAMSKI JEZIKI se bomo seznanili s programskimi jeziki in njihovimi osnovnimi ukazi. Ukazi so ilustrirani z zgledži.

Današnji računalniki razumejo le bitni jezik, ki se imenuje tudi strojna koda. Torej, če želite, da naš računalnik opravi nalogo ali nalogo, jo morate dati v obliki, ki jo razume, torej v enicah in ničlah. Vendar pa, ker za nas ljudi ni najbolj naravno razmišljati v bitnem jeziku, je bila potrebna višja, bolj razumljiva beseda, ki bi jo nato lahko prevedli v strojno kodo. Programski jezik nam omogoča, da računalniku dajemo ukaze in naloge z vsakdanjimi izrazi, kot sta "A" in "A".

"Beri in piši." Ena izmed razvojnih programskih orodij je jezik C++, ki je vsestranski za uporabo. Najdemo ga lahko pri programiranju mikroprocesorjev, ustvarjanju programov za tiskanje potovalnih nalogov in je vse pogosteje prisotna tudi pri razvoju spletnih aplikacij.

3.1. UKAZI V C++

Program je zaporedje ukazov, programiranje pa je pisanje ukazov za procesor. Ukazi so v vsakem programskem jeziku različni. To poglavje bo opisalo ukaze v programskem jeziku C++.

Poleg C++ in C# poznamo tudi programske jezike, kot so Visual Basic, Delphi, Java, SQL, Python itd.

Kratek pregled zgodovine C++:

V poznih sedemdesetih letih je Bjarne Stroustrup začel eksperimentirati z razširitvijo široko uporabljenega programskega jezika C. Programski jezik C++ je razširitev programskega jezika C, Bjarne Stroustrup pa je nekaj dodatkov črpal iz obstoječih jezikov, kot sta Simula67 in Algol68. Ime C++ se prvič pojavi leta 1983, v drugi polovici osemdesetih pa je C++ doživel radikalne spremembe.

3.2. UKAZI V C++

Glavni program (blok ukazov)

Vsi programi morajo imeti glavni program (glavni), sicer jih ni mogoče izvajati. To je jedro, v katerega boš začel programirati. V vsak program vključimo metodo main(), ki ji sledijo zaviti oklepaji {}. V programu morate določiti začetek in konec programa, pri čemer je primerno, da je blok ukazov enolično označen.

```
int main() (začetek glavnega programa)
{ - začetek bloka
cout << "To je program v c++" ;
return 0;
} - konec bloka
```

Na levi strani lahko vidite program, ki na zaslonu prikazuje stavek – to je program v C++.

KNJIŽNICE

Poznamo veliko knjižnic. Knjižice so datoteke, v katere je zapisana strojna koda funkcij. Pri pisanju programov morate vključiti knjižnice, uporabljene v programu, npr. *(#include<iostream>)* in vključiti vhodni izhodni tok podatkov. Najpogosteje uporabljene knjižnice so navedene spodaj:

```
#include <iostream> Vhodno-izhodni ukazi (knjiţnica uporabljena samo v C++)
#include <stdlib.h> Standardni ukazi
#include <string> Za laţje delo z nizi, knjiţnica je veljavna le v C++
```

KOMENTAR

Komentar je namenjen programerju, da olajša razumevanje programa. V primeru bolj zapletenih programov, ki niso dosledno komentirani, obstajajo velike težave, če želite kaj spremeniti ali popraviti. Značilnost komentarja je, da ga prevajalnik preskoči. Svoje komentarje pišete na kratko in jedrnato; Lahko so enovrstni, označeni z (//) ali večvrstni, označeni z (/* .....*/).

SPREMENLJIVKE

Spremenljivke so kos pomnilnika, ločen po tipu in velikosti. Vrsta spremenljivke nam pove, kaj v resnici pomenijo vsa ta bitja, ki jih zaseda spremenljivka. Naj bo to črka, številka ali kaj drugega.

PRAVILA ZA POIMENOVANJE SPREMENLJIVK:

- Prvi znak naj bo črka, ne številka.
- Preostali znaki so lahko črke, številke ali podčrte "_".
- V imenih ni presledkov.
- Velikost črk je pomembna. Vsakič jih moraš zapisati natanko enako. Na primer
- *Avto* ni isto kot *avto*.
- Posebna značilnost, pomembna za Slovence – ne uporabljajte šepet (č,ţ,š).
- Običajno se imena spremenljivk začnejo z majhno črko, za sestavljene besede pa vsako naslednjo z veliko začetnico. Nekaj primerov: starostOtrok, *hitrostAvta*, *velikostStopala.*
- Imena spremenljivk bi morala biti smiselna. Uporabite besede, ki povedo pomen spremenljivke, npr. *diameterCircle*, *lengthBars*.

Psevdokoda za deklaracijo spremenljivk:

TIP IME_SPREMENLJIVKE;

Psevdokoda za definicijo spremenljivk:

IME_SPREMENLJIVKE = NEKA_VREDNOST;

Spremenljivko je treba najprej razglasiti, šele nato jo je mogoče definirati. Ta dva koraka lahko združimo:

TIP IME_SPREMENLJIVKE = NEKA_VREDNOST;

Najmanjša velikost spremenljivke, ki jo je mogoče nasloviti v programskem jeziku C++, je en (1) bajt, torej osem bitov, kar pomeni 256 (28) različnih stanj. Tak tip se imenuje **char** (znak) v C++, vanj **pa je mogoče vnesti** celo število. Poleg tega poznamo tudi celo število **int**, ki običajno zavzema 4 bajte. Obstajajo tudi manjši in večji celoštevilski tipi, ki zavzamejo 2 ali 8 bajtov, vendar se njihova deklaracija razlikuje od prevajalnika do prevajalnika. Znake lahko uporabite tudi pri definiranju spremenljivk.

Primer:

```
/*
Primeri deklaracij ter definicij celoštevilčnih spremenljivk.
*/
// deklaracija
char a;
int b;
// deklaracija ter definicija, uporaba predznaka
char c = 10;
int d =+10;
char e =-10;
int f =+10;
```

Program na levi strani prikazuje deklaracijo spremenljivk. Začne se z prikazom celovrstičnega komentarja, sledi deklaracija spremenljivk, npr. spremenljivka b je tipa integer.

C++ pozna tudi **modifikatorje tipov** (samo za celoštevilske tipe): *kratek, dolg, brez predznaka, z predznakom.* **Kratka in** dolga **se uporabljata** skupaj s **tipom int** in določata velikost tipa int (odvisno od posameznih prevajalnikov).

Psevdokoda:

MODIFIKATOR TIPA IME_SPREMENLJIVKE;

TIP MODIFIKATORJA IME_SPREMENLJIVKE = NEKA_VREDNOST;

Primer:

```
// uporaba short ter long

shortint majhno = 100; // velikost spremenljivke naj bi bila 2 byta

longint veliko = 10000; // velikost spremenljivke naj bi bila 8 bytov
```

Pri uporabi modifikatorjev lahko tip **int** izpustite, saj prevajalnik ugotovi, da gre za celoštevilčni tip **int.**

Modifikatorja **signed** in **unsigned** pa določata, ali imajo naši celoštevilčni tipi tudi predznak, oz. ali lahko z njimi zapisujete tudi negativna števila. Vsi osnovni celoštevilčni tipi znajo zapisovati tudi negativna števila (privzeti način). Na primer tip **char**. Ker je velik 1 byte, lahko

z njim predstavite 256 različnih moţnosti (stanj). Tako lahko z njim zapišete števila od -128 do +127 (tudi 0). Če na tipu **char** uporabite modifikator **unsigned**, negativna števila odpadejo, tako da lahko vanj zapišete števila od 0 do 255. To velja tudi za vse druge celoštevilčne tipe.
Primer:

```
// uporaba modifikatorjev signed ter unsigned
char a; // -127 do +128, signed je privzeta vrednost
signedchar b; // -127 do +128
unsignedchar c; // 0 do 255
int d; // - 21474836648 do +21474836647
unsignedint e; // 0 do 42944967296
```

Lahko si tudi zapišete črke in pripadajoče simbole. Imaš več možnosti. Nekako so najbolj uporabljeni **ASCII** in **UNICODE** zapisi. Simbolna notacija po ASCII tabeli je velika en bajt, medtem ko UNICODE uporablja 2 bajta za notacijo. Notacija UNICODE je še posebej primerna za razvoj prenosljivih programov za tuje trge (ali jezike). ASCII pa nam omogoča preprost in predvsem manj pomnilniško zahteven zapis. ASCII format uporablja tako imenovano tabelo kod, ki je odvisna od sistemskih nastavitev (jezik in država, zato so težave z šumom). UNICODE pa uporablja drugačno notacijo, in sicer ima vsak simbol svojo edinstveno vrednost. Črke in simboli se v računalniku zapisujejo kot številke, šele ko jih začnemo "gledati" ali tiskati, dobijo svojo pravo obliko, ki jo razumemo. Na primer, velika črka A ima vrednost 65, črka B vrednost 66 in tako naprej.

Primer:

```
char a ='A'; // ascii zapis
wchar_t b ='B'; // unicode zapis
char c = 65; // ascii zapis, tokrat uporabim kar ascii vrednost ('A')
```

Poleg teh osnovnih tipov C++ pozna **tudi logični tip bool, ki je po velikosti in strukturi enak tipu char, ter tako imenovani tip ničelne praznine.**

Imena spremenljivk ne smejo imeti enakih imen kot ključne besede.
Seznam vseh ključnih besed, ki jih ne bi smeli uporabljati za imena spremenljivk:

asm, auto, break, case, catch, char, class, const, continue, default, delete, do, double, else, enum, extern, float, for, friend, goto, if, inline, int, long, new, operator, private, protected, public, register, return, short, signed, sizeof, static, struct, switch, template, this, throw, try, typedef, union, unsigned, virtual, void, volatile, wchar_t, while, itd.

Imena spremenljivk prav tako ne smejo začeti s številko (npr. *100abcd*), lahko pa vsebujejo števke na katerem koli drugem mestu (npr. *abcd123).*

ODLOČITEV (IF STAVEK)

Kot v vsakdanjem življenju, kjer so potrebne stalne odločitve, tudi v programiranju uporabljate

odločitveno izjavo (*if); na primer,* če je spremenljivka a večja od spremenljivke b, zapišete vrednost c. V pogojnem stavku imamo dve vrednosti rešitve, in sicer vrednost resnično *in* vrednost false. Pogoj je izjava (zapis), ki lahko vsebuje oklepaje, vrednosti različnih tipov, spremenljivke, konstante različnih vrst in operatorje. Operatorji so posebni simboli, ki jih lahko uporabite za združevanje spremenljivk ali vrednosti.

Prva skupina operatorjev predstavlja matematične operacije, kot so seštevanje, odštevanje, množenje in deljenje.

Tabela 1: Operator Group

Operaterja	Ime	Primer	Izraziti z besedami
+	Plus	1 + 2	Ena plus dva
-	Minus	7 – 3	Sedem minus tri
*****	Krat	5 * 2	Petkrat dva
/	Skúpen	9 / 3	devet deljeno s tremi

Relacijski operatorji primerjajo dve vrednosti, da ugotovijo, ali sta enaki ali je prva večja ali manjša od druge.

Tabela 2: Skupina relacijskih operatorjev

Operaterja	Pomen	Primer	Izraziti z besedami
==	je enak	a == b	A je enako B.
!=	Ni isto	c != d	C ni isto kot D.
<	manj kot	x = 10	x je manjše od 10
>	večji od	y = 0	y je večje od 10
<=	manjše ali enako	starost <= 25 let	Starost manjša ali enaka 25 let (vključno z 25 leti)
>=	večje ali enako	višina >= 180	višina večja ali enaka 180 (vključno s 180)

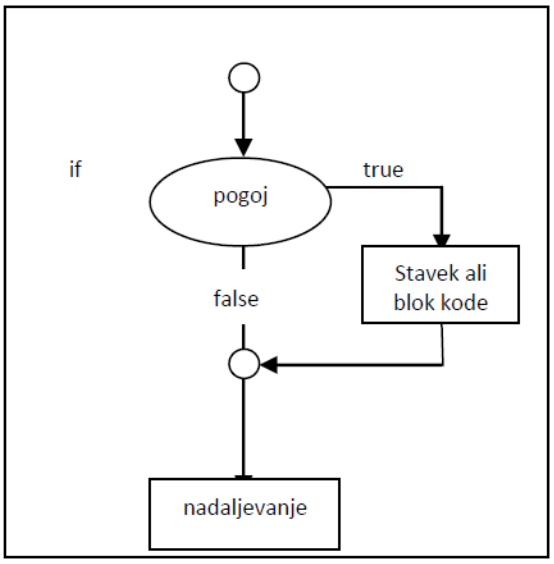
Pogoji so razdeljeni na:

- Preprost
- sestavljen (mora vsebovati vsaj 1 logični operator).

Ukazi, napisani znotraj okvira pogojne klavzule, se izvedejo, če je izpolnjen dani logični pogoj. Uporabi pogojni ukaz IF, ko želiš, da se koda izvede le, če je pogoj izpolnjen.

Splošna oblika stavka z IF je naslednja:

```
if (pogoj) // DA akcije
{
true
} else // NE akcije
{
false
```



Če je pogoj odobren in je vrednost resnična, bo dejanje xecutirano. Če pogoj ni izpolnjen, se program nadaljuje brez ukrepanja programa.

Primer programa z če: (program šteje in izpisuje številke od 1 do 10):

```
#include<iostream>
using namespace std;
int main()
{
int stevec;
stevec=1;
nazaj:
cout<<stevec<<endl;
stevec=stevec+1;
if(stevec>10)
{
system("Pause");
return 0;
}
else
{
goto nazaj;
}
}
```

Program na levi prešteje in izpiše števila od ena do deset. Najprej v program vključite vhodno izhodni tok podatkov (iostream), nato se začne izvajati glavni program. Deklarirate spremenljivko števec in jo postavite na vrednost ena. Vrednost spremenljivke števec se povečuje za ena in to tako dolgo, dokler ni vrednost spremenljivke števec večja od deset.

IZPIS

V C++ poznamo **ukaz cout,** ki se uporablja za tiskanje besedila ali številk na zaslonu. Podobni ukazi kot coutu so *izpisit (*tiska besedilo) *in izpist (*tiska številke). Ti dve ukazi uporabljate v "podprogramih", ki bodo opisani nekoliko kasneje.

Zgradba:

```
cout<<" Neko besedilo ki se izpise na ekran "<<endl
Primer izpisa na ekran:
#include<iostream>
using namespace std;
int main()
{
cout<<"To je izpis na ekran (cout)"<<endl;
system("pause");
return 0;
}
```

BRANJE

Za branje v C++ uporabiš **ukaza** cin **in getline.**

Cin bere do prvega praznega znaka in zato ni primeren za večznakovno pisavo. Ta problem **reši ukaz getline , ki je v knjižnici nizov.**

Imate različne možnosti, in sicer:
getline (cin,ime_prostora); berite, dokler ne pritisnete enter – do konca vrstice; getline (cin, ime_prostora, lik); berite, dokler ne naletite na pravi znak; getline(ime_prostora,100); bereš tolikokrat, kolikor je napisano (v tem primeru 100);
getline(ime_prostora; 500,b); berete do točke B, lahko je največ 500 znakov, ne več, v tem primeru je več kot 500 znakov, lahko preberete do 500 znakov, ali dokler ne pridemo do znaka B).

Primer programa:

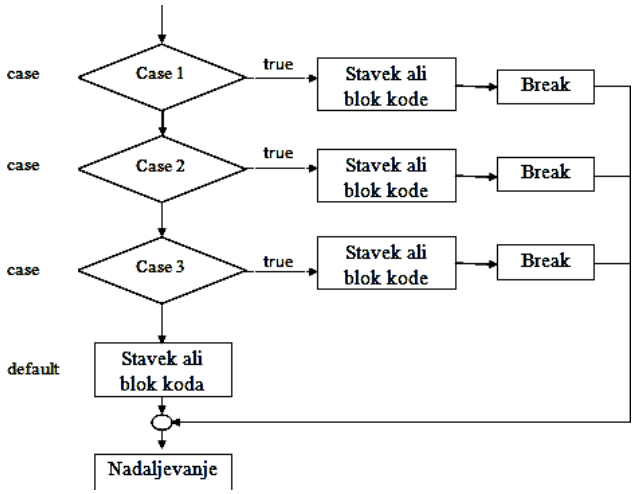
```
#include <iostream>
using namespace std;
int main () {
char name[256], title[256];
cout << "Vase ime je: ";
cin.getline (name,256);
cout << "Vas najljubsi film: ";
cin.getline (title,256);
cout << name << " vas najljubsi film je: " << title;
system("pause");
return 0;
}
```

IZBIRA

Izbira – izbirate med več različicami.

V C++ obstaja ukaz za izbiro, imenovan **switch/case**. Struktura ohišja stikala:

```
switch (izraz)
{
case vrednost A; blok ukazov A
break;
case vrednost XY; blok ukazov XY
break;
default; // ni obvezen
akcija default;
```



Slika 21: Prikaz ukaza za izbiro (stikalo/ohišje

Ukaz:
prekinitev – preskočite celotno strukturo;
Nadaljuj – en korak, trenutna akcija je prekinjena;
Goto – Uporabite ta stavek, kadar nočete, da se stavki v programski kodi izvajajo v zaporedju (izogibajte se ukazu goto, ker lahko vodi do nestrukturirane kode!!).

Primer programa s stikalom:

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int main()
{
int a;
nazaj:
cout<<"1. Ime"<<endl;
cout<<"2. Priimek"<<endl;
cout<<"3. Razred"<<endl;
cout<<"Izhod iz programa"<<endl;
cout<<"Vnesi stevilko za prikaz podatkov: "<<endl;
cin>>a;
switch(a)
{
case 1:
cout<<"Alen\n";
break;
case 2:
cout<<"Korosec\n";
break;
case 3:
cout<<"2.a\n";
break;
case 9:
cout<<"Izhod iz programa\n";
system("pause");
return 0;
default:
cout<<"Stevilo ni veljavno! Vnse drugo"<<endl;
}
system("pause");
system("cls");
goto nazaj;
```

MATRIKA

Polja so skupine enakih objektov ali podatkov. Oštevilčeni so od 0 naprej. Predstavljena bo skupina celih števil (int). Opazite format: {12, 7, 32, 15, 113, 0, 7}.

Poglejmo, kako bi te vrednosti zapisali na računalniku, in dajmo trenutne številke posameznih podatkov v skupini:

Tabela 3: Prikaz vrednosti, zapisanih v polju

Kazalo	0	1	2	3	4	5
Vrednosti	12	7	32	15	113	7

Trenutno število podatkov v takem polju imenujemo indeks. Za pridobitev posameznega elementa polja morate določiti ime polja in za imenom v oklepajih [ ] za imenom napisati indeksno številko. Recimo, da takšnemu polju daš ime *točke Tekmovalci*.

Polje je podatkovna struktura, ki se uporablja, kadar imate več spremenljivk istega tipa in za isti namen.

TIP_IME[n] (deklaracija polja)
n - indeks - število elementov
n-T Indeks je v C++ prepovedan.

Primer programa z poljem: (program izpiše številke od 0□9)

```
#include<iostream>
using namespace std;
int main()
{
int polje[10];
int i;
for(i=0;i<10;i=i+1)
{
polje[i]=i;
}
for(i=0;i<10;i=i+1)
{
cout<<polje[i]<<endl;
}
system("pause");
return 0;
}
```

Program na levi prikazuje uporabo polja.
Deklarirana sta polje celega števila tipa in spremenljivke i, prav tako celega števila tipa.

ZANKA PRAVI:
Dokler je i manjši od deset, povečajte i za eno ali naredite inkrement za i in zapišite vrednosti v polje.
Uporabite ukaz cout za izvlečenje vrednosti iz polja.

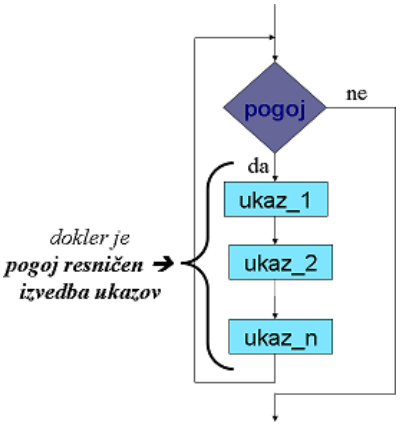
ZANKE

Zanka je osnovni ukaz, ki ga uporabite, ko se kos kode ponavlja (zaporedno) za isti namen.

Teoretični opis zank:
Vsaka zanka ima glavo in telo. V glavi se odločaš, ali boš ukaz ponovil ali ne, in v telesu so ukazi, ki jih je treba ponoviti. V glavi je pogoj in vsaj ena spremenljivka v pogoju, na katero lahko vplivate.

FOR ZANKA

For zanka se ponavlja, dokler pogoj ni resničen. Primerna je, kadar je število ponovitev vnaprej znano (ni izvedena, če pogoj ni izpolnjen).



Slika 20: Primer For zanke

Za obliko zanke:

```
for ( ;pogoj; )
{
Blok ukazov
Vpliv na spremenljivke, ki se nahajajo v pogoju
}
```

Primer programa z for zanko (program napiše ime na zaslonu 10-krat):

```
#include <stdio.h>
#include <iostream>
using namespace std;
int main()
{
int index;
for(index = 0 ; index < 10 ; index = index + 1)
cout<<"Alen"<<endl;
system ("pause");
return 0;
}
```

Program na levi strani prikazuje delovanje for zanke.
For zanke so v bistvu zgoščen zapis za:

- sprožitev merilnika (ali drugega pogoja),
- nastavitev in preverjanje pogoja, ter
- Povečanje števca.

ZANKA WHILE

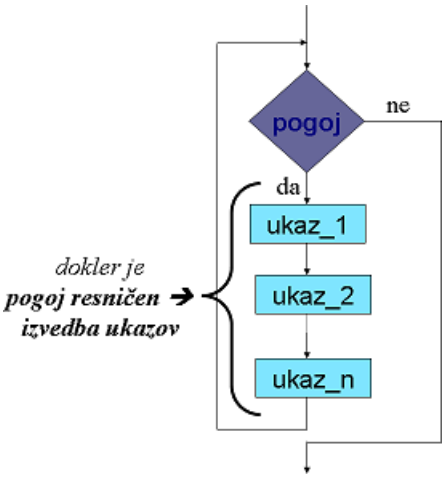
Oblika While zanke:

```
while (pogoj)
{
Blok ukazov
}
```

While zanka nadaljuje z izvajanjem, dokler je nek pogoj resničen. Ko postane pogoj neizpolnjen, se zanka prekine. Torej zanka počne ravno tisto, kar je razvidno ţe iz njenega imena.

Primer programa z zanko While (program napiše ime na zaslonu 10-krat):

```
#include <stdio.h>
#include <iostream>
using namespace std;
int main() /
{
int count;
count = 0;
while (count < 10)
{
cout<<"Alen"<<endl;
count = count + 1;
}
system ("pause");
return 0;
}
```



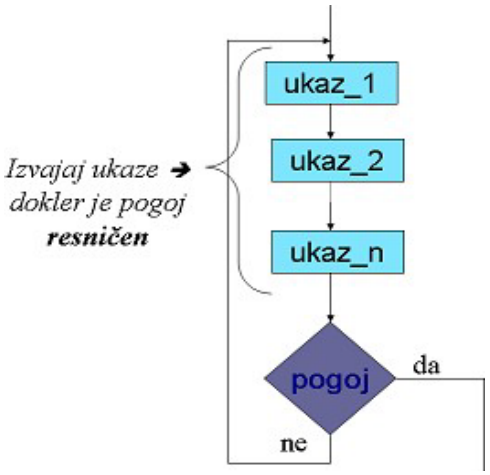
Slika 21: Primer zanke While

DO WHILE ZANKA

Priporočljivo je, da to zanko uporabimo, ko imamo podatke, in na podlagi nje odločimo, ali jo bomo ponovili ali ne. Zanka se ponavlja, dokler je pogoj resničen. Zanka Do While, za razliko od zanke While, se izvede vsaj enkrat, tudi če pogoj ni izpolnjen.

Oblika zanke:

```
do
{
}while();
```



Slika 22: Primer Do While zanke

Primer programa z zanko Do While (program napiše ime na zaslonu 10-krat):

```
#include <stdio.h>
#include <iostream>
using namespace std;
int main() /
{
int i;
i = 0;
do
{
cout<<"Alen"<<endl;
i = i + 1;
} while (i < 10);
system("pause");
return 0;
}
```

V danem programu je uporabljena Do While zanka. Zanka se izvaja tako dolgo, dokler je spremenljivka i manjša od vrednosti 10. Vsakič, ko se zanka izvede, se spremenljivka i poveča za 1.

PODPROGRAMI

Ko se glavni programi izvajajo, pride do klica podprogramov. Podprogrami se večinoma uporabljajo, kadar se koda ponavlja na različnih mestih in če se koda ponavlja zaporedno, vendar vsakič z različnimi podatki. Značilnost podprograma je, da mora delovati samostojno in ne mora biti odvisna od drugih programov v kodi. Podprogrami so razdeljeni na postopke in funkcije (postopek – ime podprograma se ne uporablja za prenos rezultatov, funkcija – ime podprograma se uporablja za prenos rezultata).

Struktura podprograma:

* HEADER (Glava vsebuje vrsto podprograma in ime podprograma).

Parametri – so spremenljivke, ki se uporabljajo samo v podprogramu (v glavi).

* TELO
  - začne se z začetnim znakom bloka { in konča s končnim znakom bloka };
  - V telesu je koda (v kodi je lahko vse, kar poznamo, razen da nam ni dovoljeno začeti nove podrutine).

Razdelitev podprogramov:

- Postopek (ne uporabljamo **imena podprograma** za prenos rezultatov);
- Funkcija (za prenos rezultatov uporabimo ime podprograma).

Za prenos podatkov v podprogram:

a) Globalne spremenljivke.

b) Prenos preko parametrov:
   - po vrednosti (spremembe podatkov niso ohranjene),
   - po referenci (spremembe podatkov so ohranjene). Referenca vpliva na lokacijo klica.

c) Preko datoteke;

d) Ne prenesemo ničesar in samo beremo.

Prenos podatkov iz podprograma:

a) Globalna spremenljivka:
   - Vsaka sprememba mora biti ohranjena tudi zunaj podprograma.

b) Uporaba parametrov:
   - Vsaka sprememba parametra vpliva na spremenljivko, ki je dala podatke temu parametru.

c) Uporabi mapo.

Vsak programski jezik ima poseben ukaz, da lahko uporabimo ime funkcije za prenos rezultata izven podprograma. V C++ je to **ukaz za povratek.**

Povzetek:

Program je zaporedje ukazov, programiranje pa je pisanje ukazov za procesor. Ukazi so v vsakem programskem jeziku različni, to poglavje pa opisuje ukaze v programskem jeziku C++. Poznamo veliko knjižnic za različne aplikacije. Knjižice so datoteke, v katerih je napisana strojna koda funkcij in jih je treba vključiti v program, če želimo te datoteke uporabiti v programu. Spremenljivke so kos pomnilnika, ločen po tipu in velikosti. Vrsta spremenljivke nam pove, kaj v resnici pomenijo vsa ta bitja, ki jih zaseda spremenljivka. Vsaka zanka ima glavo in telo. V glavi odločamo, ali bomo ponovili ali ne. V telesu pa obstajajo ukazi za ponavljanje. V glavi je pogoj in vsaj ena spremenljivka v tem stanju lahko vplivamo.

VPRAŠANJA:

1. Naštejte lastnosti glavnega programa.
2. Kaj so knjižnice?
3. Kaj nam pove vrsta spremenljivke?
4. Kaj se zgodi, če uporabimo nepodpisani modifikator za tip lika?
5. Pojasnite razlike med ASCII in UNICODE modifikatorskimi zapisi.
6. Na zaslon zapišite primer natisnjenega obraza.
7. Napišite program, ki bo zahteval, da vnesete dve številki in izračunate ter natisnete vsoto, produkt, razliko in količnik.
8. Kaj je zanka? Naštej nekaj zank in opiši zanko Do While.
9. Nariši diagram poteka in napiši program, ki bo zahteval vnos 10 številk in jih prikazal po vrstnem redu od največje do najmanjše.
10. Katere vrste spremenljivk so znane v programskem jeziku C++?
11. Komentirajte program spodaj:

```
#include<stdio.h>
int main ()

{
   int a,b,c;


   a=5;
   b=6;
   c=7;
printf(˝Spremenljivke: %d, %d, %d, a, b, c =;
```

4. PROGRAMABILNI LOGIČNI KRMILNIK (PLK)

V zadnjem desetletju se je področje krmilne tehnologije precej spremenilo. V preteklosti smo uporabljali le žične kontrole, danes pa se vse bolj uporabljajo tako imenovani programski krmilniki (angleška okrajšava za programabilni logični krmilnik je PLK – Programmable Logic Controller).

Največja prednost, ki jo imajo krmilniki v primerjavi z vsemi starejšimi krmilnimi sistemi, je, da omogočajo programiranje delovanja. Programiranje se lahko uporablja za določanje delovanja krmilnika, kar omogoča veliko prilagodljivost sistema, saj so možnosti programiranja skoraj neomejene. Klasični krmilniki so vsebovali fiksno logiko – bila je zelo zapletena in predvsem draga za spreminjanje ali popravljanje.

Razdelek Programabilni logični krmilnik (PLK) prikazuje, kako je PLK zgrajen in kako deluje. Predstavljen je tudi nadzor naprave (v odprti in zaprti zanki). Predstavljeni so nam zaporedni nadzorniki. Siemensov krmilnik in strojna oprema sta predstavljena bolj podrobno. Nekaj besed je namenjenih tudi programski opremi in programiranju.

PLK je digitalno delujoča elektronska naprava, ki na podlagi ukazov, shranjenih v programabilnem pomnilniku, izvaja logične, zaporedne, časovne in aritmetične operacije ter tako usmerja različne naprave in procese prek digitalnih in analognih vhodov in izhodov.

4.1. ZGRADBA

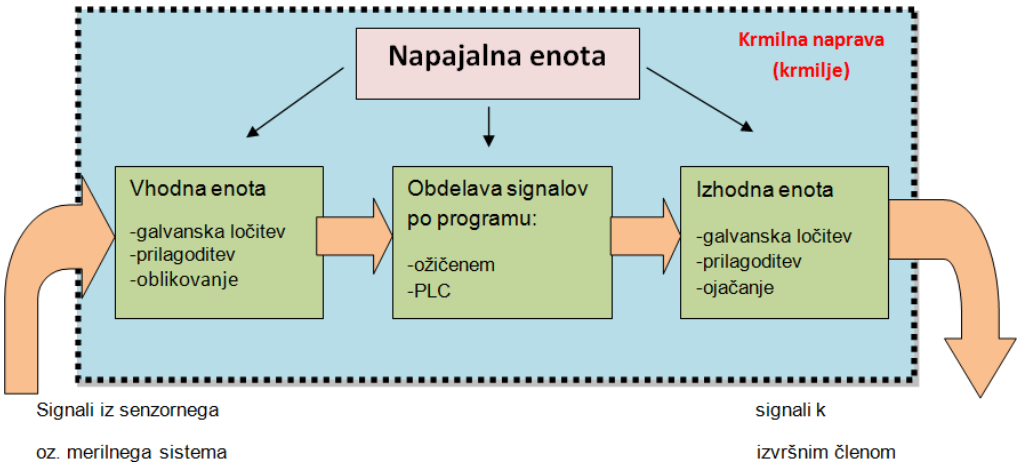
Vhodna enota je enota za zajem procesnih spremenljivk in spremenljivk operaterja, ki transformira in prilagaja njihove signale ter jih posreduje procesni enoti. Galvanska ločitev signalov je tukaj pomembna.

ENOTA ZA OBDELAVO SIGNALOV

- **a)** V stalno ožičenih sistemih je to pogosto vezje z elektronsko logiko, časom, štetjem, računalniškim ... elemente, ki odpravljajo funkcijo krmiljenja;
- **b)** pri prosto programabilnih krmilnikih je to osrednja procesna enota (mikroprocesor), skupaj s pomnilniki in perifernimi napravami; Krmilni program je shranjen v pomnilniku programa, kar odpravi krmilno funkcijo.

Izhodna enota je enota za transformacijo, prilagajanje in prenos krmilnih signalov izvršnim članom procesa. Tudi tukaj je pomembna galvanska ločitev signalov.

Napajalna enota je enota, ki napaja napajalno napetost krmilni opremi. Pri PPK se uporabljajo standardne različice z enosmerno napetostjo 24 V.

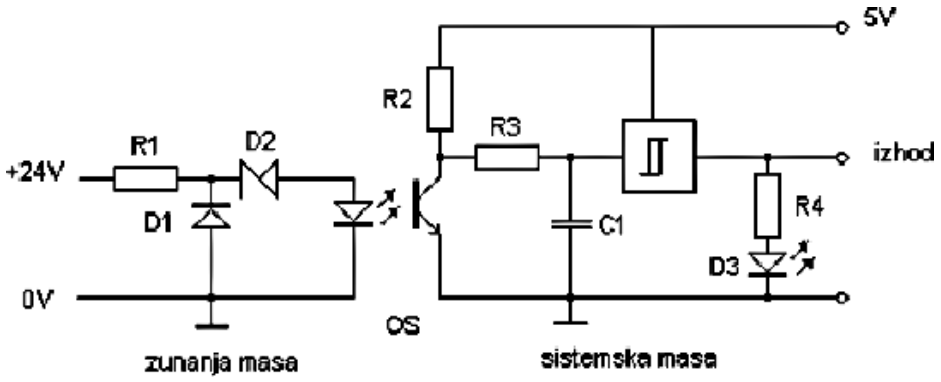


Slika 23: Struktura krmilnega sistema Vhodni moduli

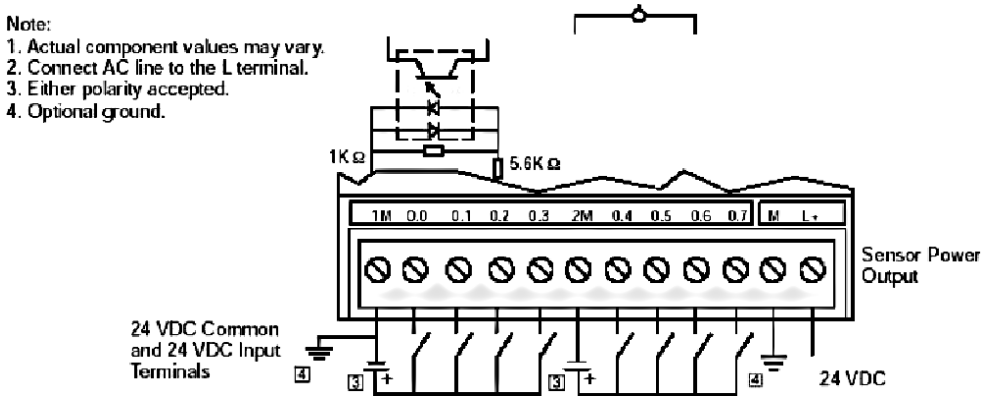
Da se motnje ne prenašajo, se uporablja oktopter ali oktokopter, ki motnje odstrani.

Poleg zgoraj omenjenih nalog (nastavitev nivoja, galvansko ločevanje, zaščita pred kratkim stikom) obstajajo še dodatne možnosti:

- možnost nastavitve časa vhodnega filtra (odprava hitrih stikal na vhodu zaradi motenj),
- različno število vhodov (razširljivo),
- Označitev logične enote.



Slika 24: Električni diagram digitalnega modula z DC vhodom
Vir: http://support.automation.siemens.com

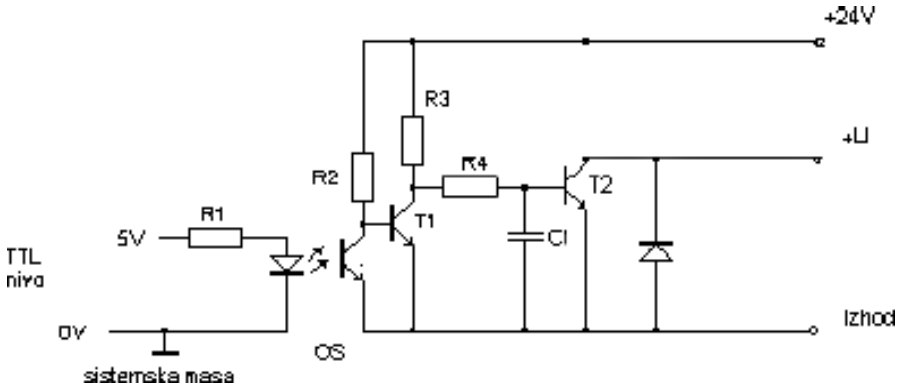


Slika 25: Povezava vhodov z vhodnim digitalnim modulom na S7-300
Vir: http://support.automation.siemens.com

IZHODNI MODULI

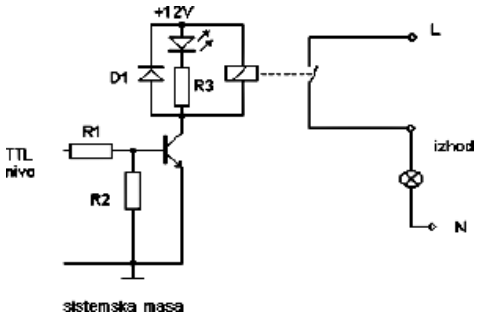
- Zasnovani so za prenos in tvorbo signala, ki ga oddaja centralna krmilna enota. Signali iz centralne procesne enote so običajno na ravni TTL (5 V napajalnik).
- Višje ravni signala je mogoče doseči s tranzistorskim ojačevalnikom, pomožnimi releji ali trakovi.
- Tokovne obremenitve so v razponu od nekaj mA do nekaj A.
- Galvansko ločevanje dela sistema od zunanjega procesa.
- Označevanje enote ločevanja.

Porabnik je lahko priključen le na enosmerno napetost (obremenitve z nižjim tokom).

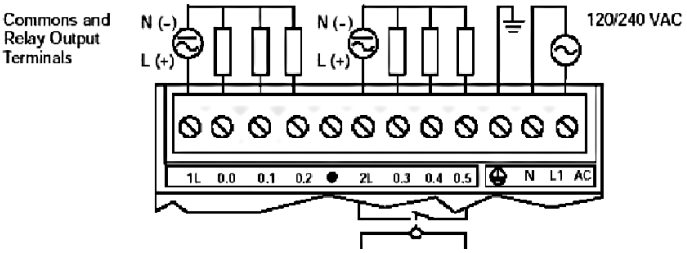


Slika 26: Digitalni izhodni modul z izhodom tranzistorja
Vir: http://support.automation.siemens.com

Odjemalnik je lahko priključen na DC ali AC napetost.



Slika 27: Digitalni izhodni modul z relejnim izhodom
Vir: http://support.automation.siemens.com

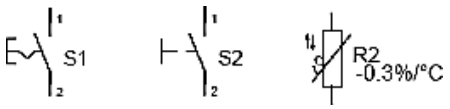


Slika 28: Povezava izhodnih enot z izhodnim modulom digitalnega releja
Vir: http://support.automation.siemens.com

ODDAJNIKI SIGNALA

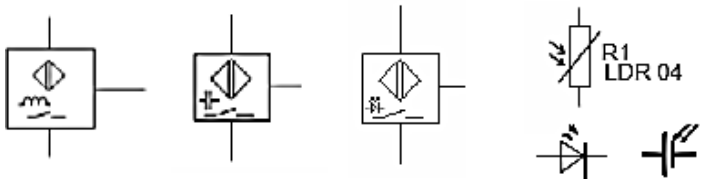
Naloga oddajnikov signalov (senzorjev, detektorjev, merilnih elementov) je merjenje ali zaznavanje količine v krmilnem sistemu, pretvorba signala v ustrezno obliko (ojačanje, kompenzacija temperature, zamik itd.) in prenos informacij na krmilno napravo.
Obstajajo različne vrste oddajnikov:

- mehanska pretvarjajo silo, ki deluje na kodirnik, v električni signal (končna stikala, gumbi, lahko pa tudi silo pretvorijo v tok ali napetost),
- **Upori** uporabljajo lastnost za spreminjanje upornosti elementa. Uporabljajo se mostna vezja (uporne plošče, termistori, linearni in rotacijski potenciometri...),



Slika 29: Mehanski in uporni kodirnik položaja

- **induktivno –** izmerjena fizikalna količina vpliva na spremembo induktivnosti, izhod je lahko binaren z delovnim ali mirovalnim kontaktom,
- **kapacitivna deluje** na spremembo kapacitivnosti, lahko meri premike, nivoje, uporablja se kot limitna stikala za zoom,
- **optoelektroni** delujejo na spremembo svetlobe, ki jo pretvorijo v električno napetost (npr. fotocelico), ali pa na spremembo ohmske upornosti (npr. fotoupor, fotodioda, sonde uporabljajo UV ali IR svetlobo),
- **CMOS-senzorji** (senzor in elektronika v enem čipu – učinek senzorja v Si, nižja cena, višja kakovost, manj motenj).

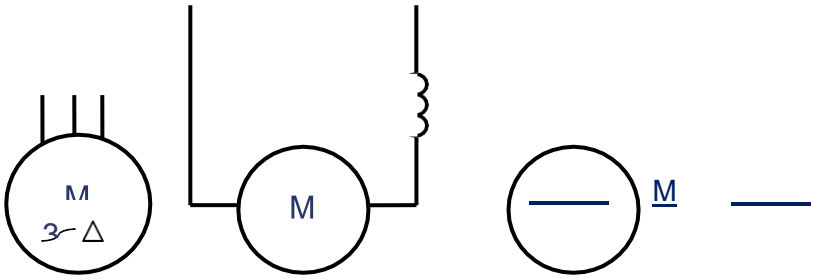


Slika 30: Induktivni, kapacitivni, optoelektronski in CMOS kodirniki signalov

IZVRŠNI ČLANI – AKTUATORJI

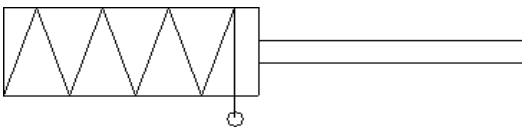
Naloga izvršnih članov je izvajanje ukazov, poročanje in signaliziranje o stanju postopka.

- Električni motorji



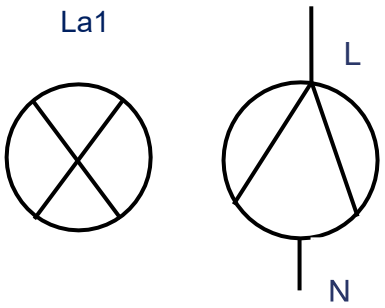
Slika 31: Simboli za nekatere električne motorje

- Pnevmatski valji



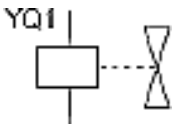
Slika 32: Pnevmatski valj

- Luči, grelniki, ventilatorji



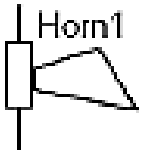
Slika 33: Svetilke, grelniki, ventilatorji

- Ventili



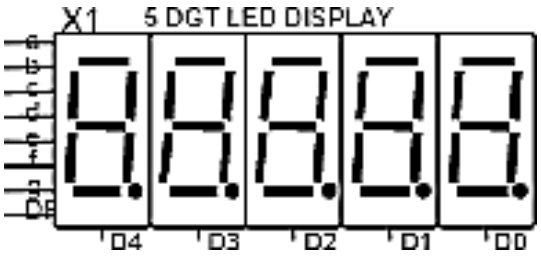
Slika 34:Ventili

- Obvestila



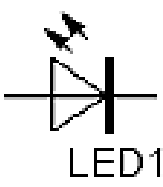
Slika 35:Poročanje

- Prikaže



Slika 36:Prikaz

- Semaforjev



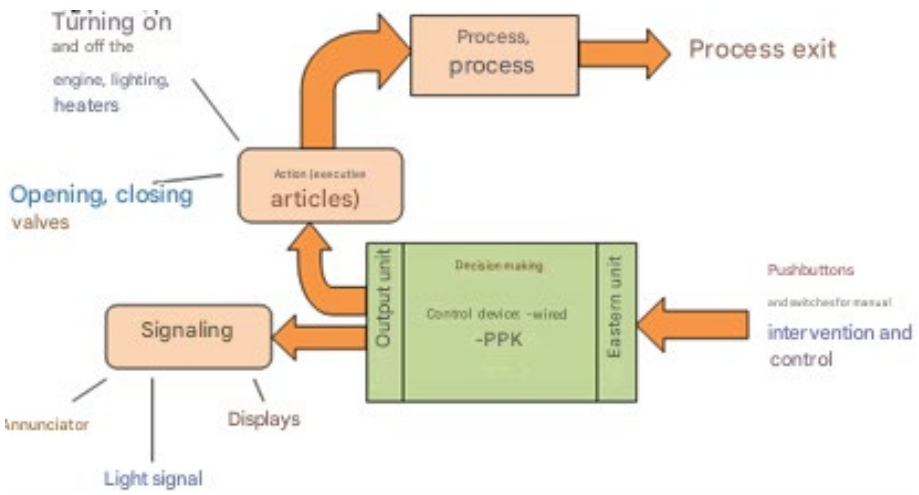
Slika 37:Semaforjev

KRMILJENJE NAPRAVE

a) **<u>Krmiljenje odprte zanke</u>** (kombinacija vhodnih spremenljivk s pomočjo krmilne naprave aktivira krmilne signale in jih prenese izvršnim članom preko izhodne enote – iz procesa ni povratne zanke).

Delovanje ali upravljanje procesa poteka v dveh fazah:

- Odločanje (obdelava podatkov, signalov).
- Akcija (prenos in izvedba ukazov, običajno spremljana z ustrezno signalizacijo).

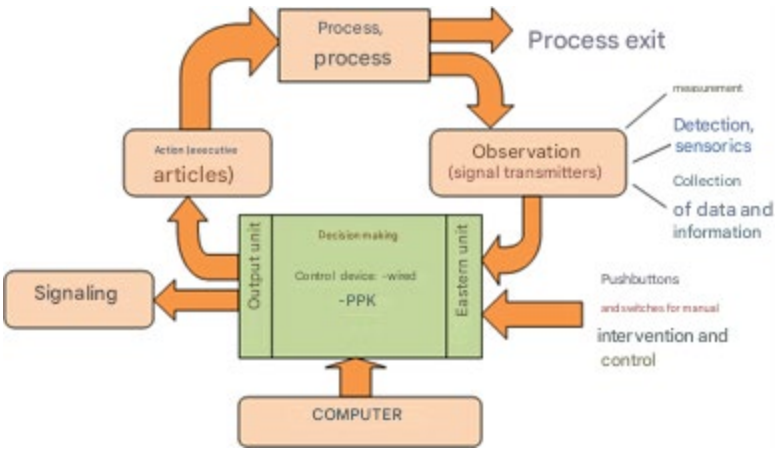


Slika 38: Krmiljenje odprte zanke

b) **<u>Zaprta zanka krmiljenja</u>** (najpogostejša metoda, zlasti pri krmiljenju industrijskih procesov; signali iz procesa pridejo na vhod krmilne naprave).

Delovanje ali upravljanje procesa vključuje dodatno fazo:

- **Opazovanje** (zbiranje informacij in podatkov)

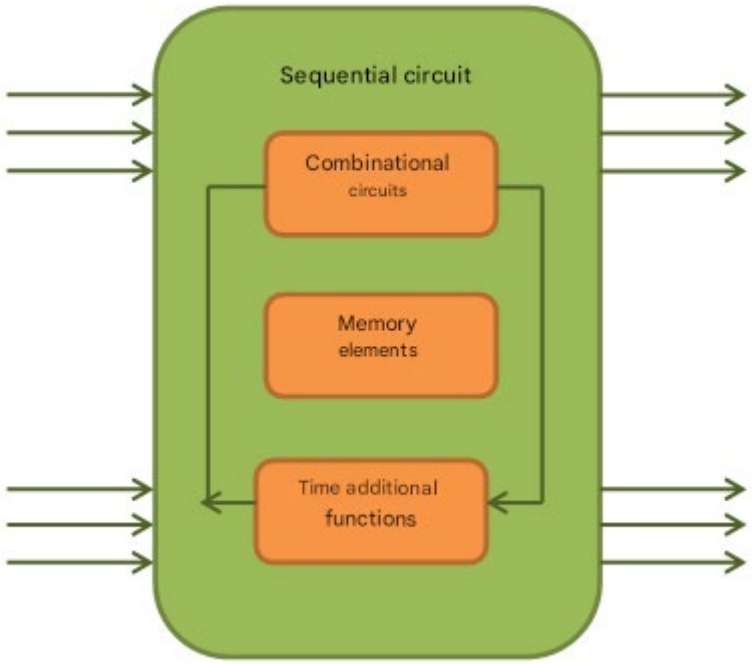


Slika 39: Zaprta zanka krmiljenja

4.2. ZAPOREDNO KRMILJENJE

4.2.1.ZNAČILNOSTI ZAPOREDNIH KONTROL

- Izhod ni odvisen le od trenutnega stanja vhodov, temveč tudi od prejšnjih stanj sistema.
- Poleg osnovnih logičnih funkcij (kombinacijsko vezje) zaporedni krmilniki vključujejo tudi pomnilniške elemente, čas, število in druge dodatne funkcije.
- Enako kombinacijo vhodnih stanj je mogoče preslikati na različne kombinacije izhodov.
- Lahko jih primerjate z asinhronimi zaporednimi vezji ter jih opišete in oblikujete z uporabo diagrama stanj, tabele stanj ali verig korakov.



Slika 40: Zaporedno vezje

4.2.2.PREDNOSTI ZAPOREDNIH KRMILNIKOV

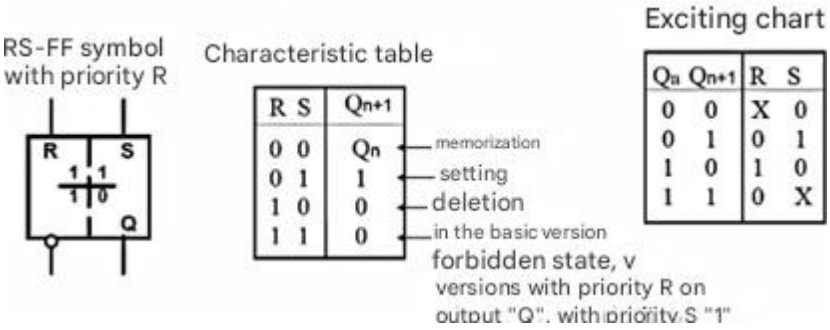
a) Gre za preprostejši – manjši in enostavnejši program za krmilnik.
b) Manj občutljivosti na šumne signale in motnje.
c) Sistem stabilnih stanj; prehod v novo stanje se izvede le pod določenimi pogoji; Sistem se ne odziva na druge spremembe.
d) Lažje zaznavanje napak.

Zaporedni nadzor je razdeljen na:

- **Prosti nadzor** – na vhodu se lahko pojavi katerakoli kombinacija v poljubnem zaporedju.
- **Nadzor korak za korakom** – kombinacije vnosov se vedno pojavijo v določenem zaporedju.

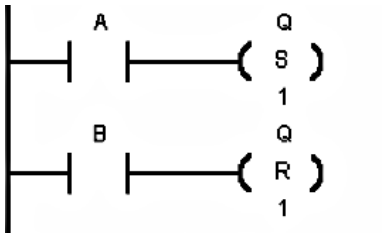
4.2.3.GRADNIKI ZAPOREDNIH KRMILNIKOV

Funkcije pomnilnika (pomnilnika):

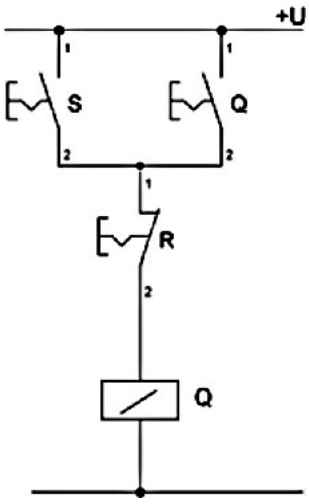


Slika 41: RS funkcije
Vir: Simatic Step7 Help

Pomnilnik (pomnilnik) deluje v skaliranem diagramu:



Slika 42: RS-funkcija (diagram merila)



Slika 43: RS- funkcija
Vir: Simatic Step7 Help

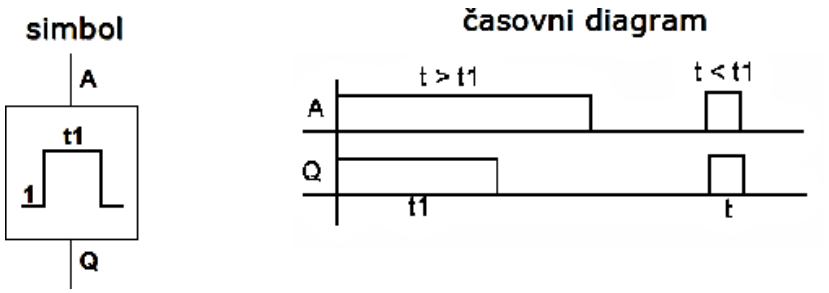
V digitalni tehnologiji obstajajo še tri vrste FF: JK, D- in T-FF (števci in registri). Kot samostojni FF-ji redko delujejo v kontrolah. Če jih potrebujemo, jih lahko enostavno implementiramo (programiramo) na osnovi RS-FF.

ČASOVNE FUNKCIJE

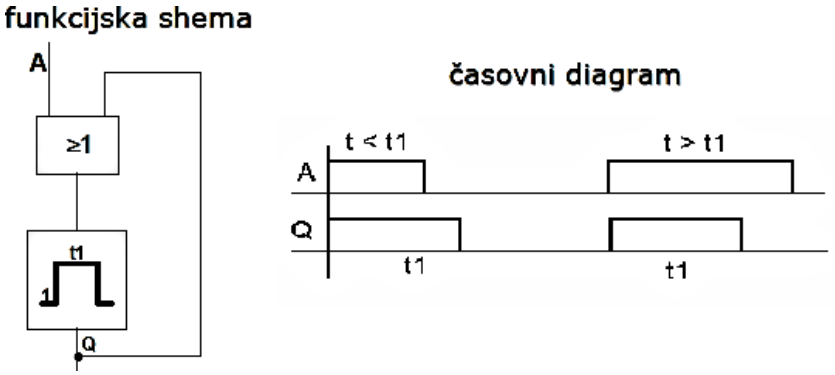
Časovne funkcije se uporabljajo za določanje trajanja logičnih signalov – lahko jih skrajšate, podaljšate ali premaknete v času.

Obstajajo tri osnovne vrste časovnih funkcij:

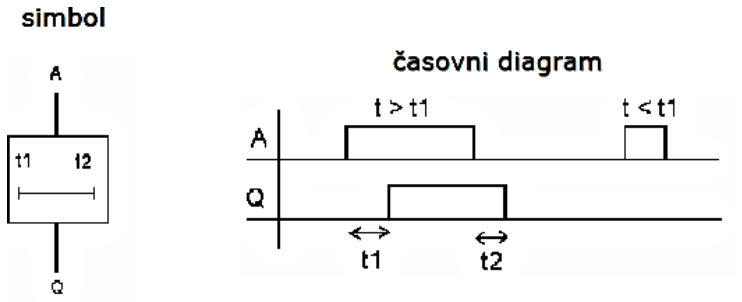
- skrajšanje dolgih pulzov,
- podaljševanje kratkih pulzov,
- časovni premiki impulzov.



Slika 44: Skrajšanje pulza
Vir: Simatic Step7 Help



Slika 45: Raztezanje pulza
Vir: Simatic Step7 Help

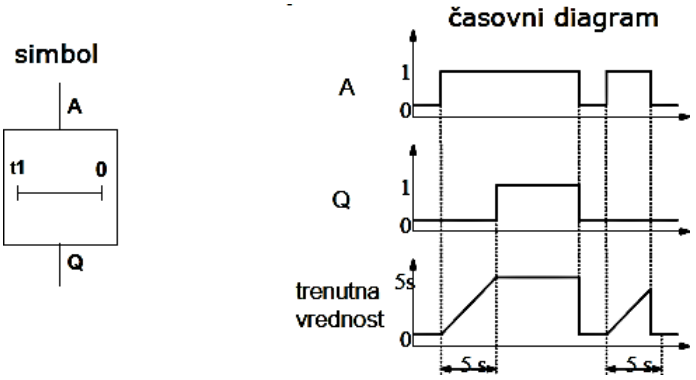


Slika 46: Premikajoči se pulzi
Vir: Simatic Step7 Help

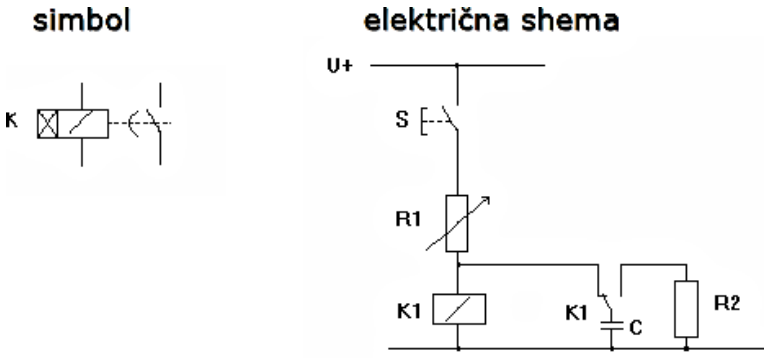
Posebne oddaje Time:

V krmilni tehnologiji se pogosto uporabljata še dva načina posebnih časovnih funkcij. To so zamiki, ki jih realizirajo časovne funkcije in logična vrata:

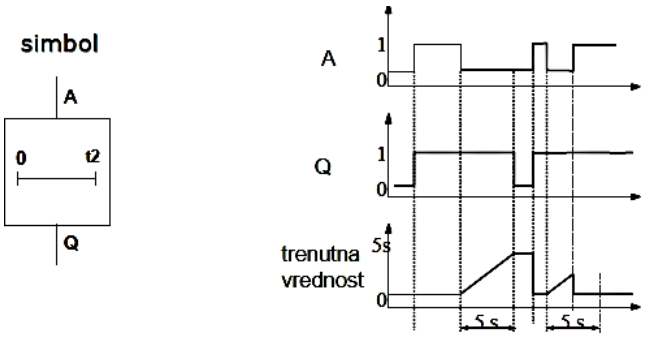
- Zakasnitev vklopa.
- Zamuda pri izklopu.



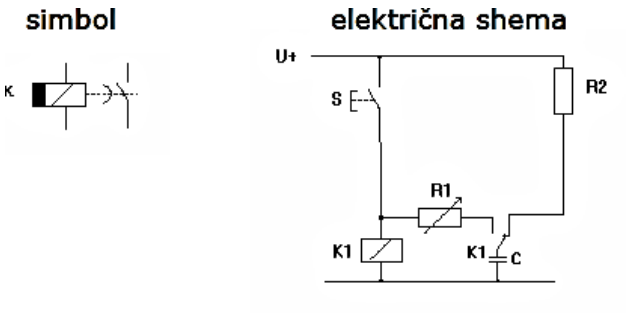
Slika 47: Razlaga zakasnitve ob vklopu
Vir: Simatic Step7 Help



Slika 48: Zakasnitev vklopa (rele)
Vir: Simatic Step7 Help



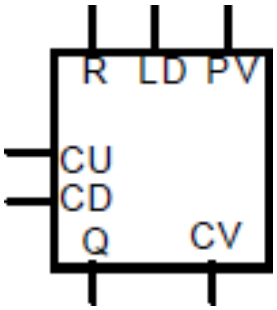
Slika 49: Razlaga zakasnitve izklopa
Vir: Simatic Step7 Help



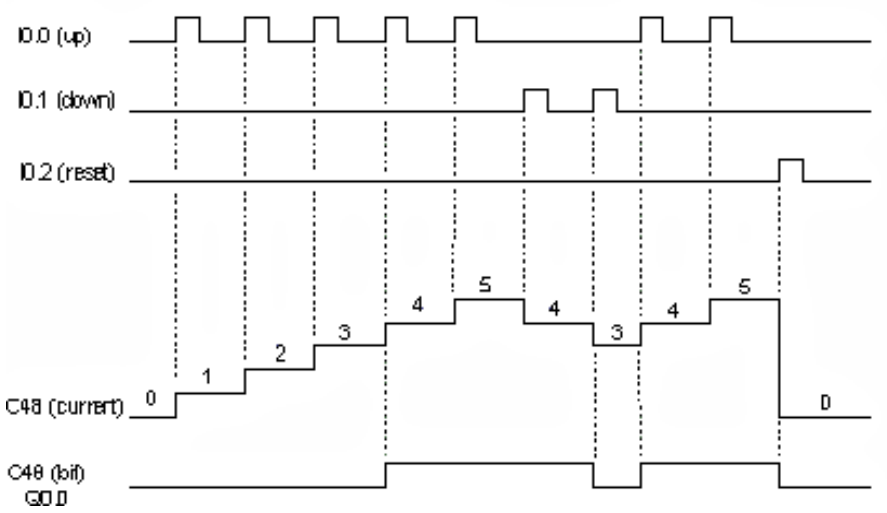
Slika 50: Implementacija zakasnitve izklopa (rele)
Vir: Simatic Step7 Help

Števci (štetje dogodkov)

CU: Count Up (dinamično) CD: Count Down (dinamično) R: Reset
PV: nastavljena vrednost
LD: obremenitev PV vrednost (dinamična) Q: izhod (bool)
CV: Vrednost števca (celoštevilska vrednost)



Slika 51:Counter
Vir: Simatic Step7 Help



Slika 52: Časovni diagram delovanja merilnika UP/DOWN
Vir: Simatic Step7 Help

4.3. LASTNOSTI SIEMENSOVEGA KRMILNIKA

Prosto programabilni krmilniki so bili bistven del sistemske avtomatizacije in nadzora že od samega začetka. Njihova uporaba je postala tako razširjena, da je skoraj popolnoma nadomestila uporabo tako imenovanih žično povezanih logičnih in analognih krmilnikov. Razlog za njihovo uvedbo je predvsem sposobnost hitrega programiranja brez spreminjanja ožičenja.

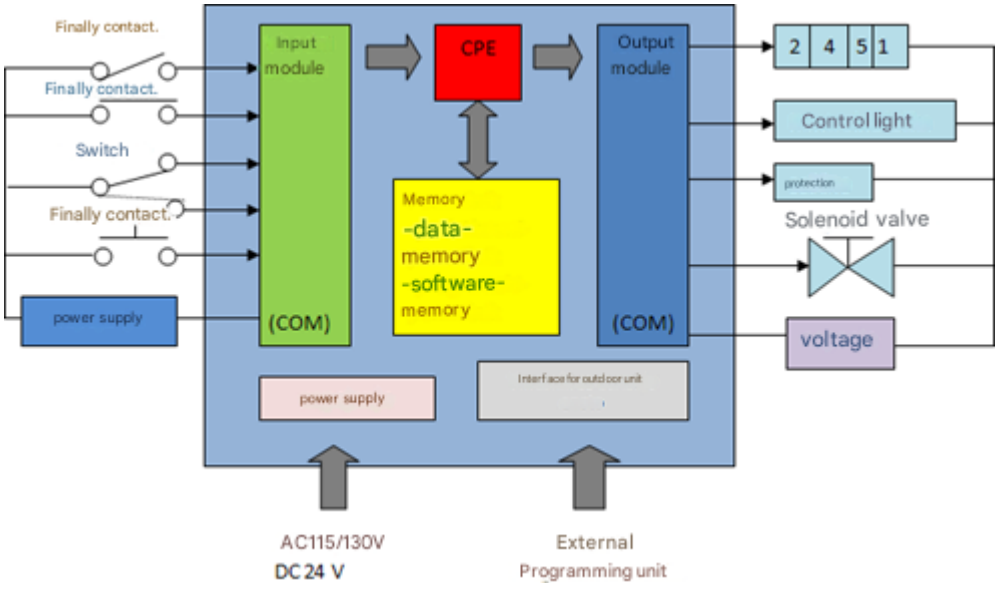
Zaradi množične proizvodnje in uporabe prosto programabilnih krmilnikov v industriji so ti postali:

- Je razmeroma poceni,
- zelo zanesljivo v uporabi,
- Enostavno za uporabo (programiranje, namestitev in servisiranje).

Uporaba prosto programabilnih krmilnikov je danes razširjena. Glavna področja uporabe so:

- zajemanje in obdelava analognih in digitalnih podatkov,
- regulacija različnih sistemov,
- izvajanje računalniških operacij,
- kot vmesnik za izvajanje oddaljenih ukazov,
- Za komunikacijo.

Prosto programabilni krmilniki so računalniki brez zaslona in tipkovnice, vendar imajo vse druge komponente, kot so procesor, pomnilnik, vhodno/izhodni vmesniki in komunikacijski vmesniki za povezovanje z drugimi sistemi, ki delujejo na osnovi uporabniškega programa. Prilagojen je za delovanje v industrijskem okolju. Z razvojem krmilnikov so bila razvita različna programska orodja, s pomočjo katerih pišemo uporabniške programe za različne vrste krmilnikov. S pomočjo uporabniških programov krmilniku dodelimo naloge, ki jih mora opraviti. Programe najpogosteje pišemo na računalnikih. Pri tem je treba paziti na uporabo ustreznega komunikacijskega vmesnika za delovanje krmilnika.



Slika 53: Struktura blokovnega krmilnika

4.4. STROJNA OPREMA SIMATIC SIEMENS

SIMATIC krmilni sistemi so razdeljeni v tri podskupine, in sicer:

- SIMATIC S7 programabilni krmilniki,
- SIMATIC M7 sistemi in
- Popoln krmilni sistem SIMATIC C7.

Programabilni krmilniki SIMATIC S7 so prav tako razdeljeni glede na njihovo velikost ali zmogljivost:

- nižji razred krmilnikov, serija S7-200;
- krmilniki srednjega razreda, serija S7-300;
- višji razred krmilnikov, serija S7-400.

Sistemi SIMATIC M7 omogočajo integracijo matematičnih operacij in obdelavo podatkovnih baz v krmilni program STEP7.

Sistemi SIMATIC C7 vključujejo družino S7-300 programabilni krmilnik in nadzorno ploščo (OP) v eni napravi.

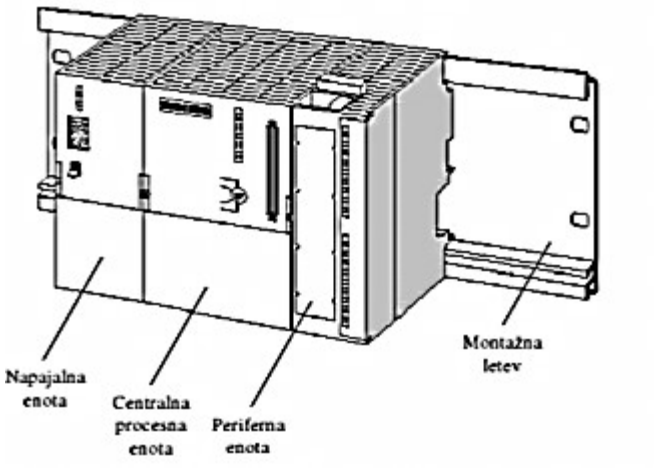
Komunikacija med operaterjem in napravo poteka prek naprav za upravljanje in opazovanje. SIMATIC ponuja celoten nabor teh naprav, ki večinoma podpirajo SIMATIC sisteme ter nekatere druge sisteme drugih proizvajalcev.

Vse te sisteme je mogoče programirati s posebnimi napravami, programiranimi s PG, ki so zelo drage, zato je bolje uporabljati osebne računalnike ali prenosnike, na katerih je naložen programski paket STEP 7.

Da sistemi delujejo pravilno, morajo tudi med seboj komunicirati. Poznamo različne komunikacijske sisteme, ki se med seboj razlikujejo po hitrosti, prenosni zmogljivosti in seveda ceni.

Sistemi, ki jih v industriji najpogosteje najdemo, so:

- PROFIBUS,
- PROFINET,
- SAFETYBUS in
- Industrijski thernet



Slika 54: Prikaz posameznih modulov Siemensovega krmilnega sistema Vir: http://support.automation.siemens.com

4.5. KOMPONENTE PROSTO PROGRAMABILNEGA SISTEMA NAPAJALNA ENOTA (PS 307 5A)

Napajalna enota nam omogoča spreminjanje napetosti iz 230V na 24V. To nam omogoča transformator, ki je v njem. Do enote se pripeljejo fazni vodnik, nevtralni vodnik in ozemljitveni vodnik, skozi katere teče napetost 230V. Iz enote dobimo napetost 24V, na katero je mogoče priključiti vse ostale komponente.

Slika 55: Napajalna enota

CENTRALNA PROCESNA ENOTA (CPU 313C)

V našem primeru smo uporabili procesor 313c, ki skrbi za obdelavo, obdelavo in nadgradnjo podatkov. Ta komponenta je najpomembnejša in najzahtevnejša v celotnem modularnem krmilniku. Seveda so na centralni procesorski enoti tudi vhodi in izhodi (DI 16/DO 16* DC 12V), kjer so nameščeni signali senzorjev, motorja, releja in robota.

Slika 56: Centralna procesna enota

KOMPONENTE POSTAVITVE OMREŽJA (CP 343-1 ADVANCED, SKALANCA X208)

Za vzpostavitev omrežja najprej potrebujete tako imenovani omrežni modul, ki vsebuje protokol Simatic Net in deluje na naprednem modulu CP 343-1. Ta modul skrbi za nastavitev omrežja, vendar se pojavi težava, in sicer, da ima ta modul le en RJ 45 vmesnik, zato potrebujete razdelilnik, tako imenovani Scalance X208.

Slika 57: Komunikacijski procesor X208 CP 343-1 Advanced

Slika 58: Skalanca stikala

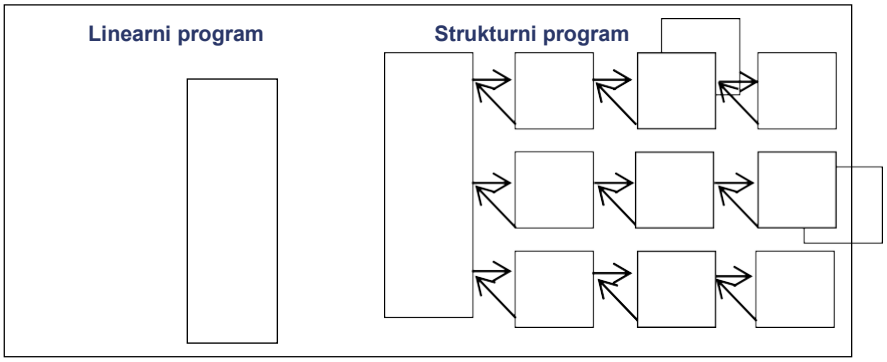
4.6. PROGRAMSKA OPREMA

Za programsko podporo uporabnikom izdelkov skupine Simatic je Siemens razvil programski paket STEP 7. Programska oprema je združljiva z operacijskimi sistemi Microsoft Windows in ustreza EN 61131-3 [7].

Razdelimo ga v štiri skupine programskih izdelkov:

- Osnovna programerska orodja (*Standardna orodja*),
- *Inženirska orodja*,
- programska orodja za delo v realnem času (*Runtime Software*) in
- programska orodja za vmesnik človek-stroj (*vmesnik človek-stroj*).

STRUKTURA PROGRAMA STEP 7:



Slika 59: Struktura programa STEP 7

Celoten uporabniški program se zaključi v enem bloku (OB1). Procesor izvaja program linearno po stavku, kot je vtipkano. Takšna organizacija uporabniškega programa je prisotna v avtomatiziranem sistemu S7-200. Poleg tega se v njem lahko uporablja tudi tehnika podprogramov.

STRUKTURNI PROGRAM

Celoten uporabniški program je razdeljen na bloke. Blok je samodejni del programa, ki se od ostalih razlikuje po svoji funkciji, nalogi in prioriteti.

RAZLIKUJEMO SEDEM VRST RAZLIČNIH BLOKOV:

Organizacijski bloki (OB) uravnavajo ciklično, časovno in alarmno obdelavo programov.
Lastnosti:

- uporabnikov dostop do operacijskega sistema,
- prednost po stopnjah, in
- Dodatne informacije o zagonu lokalnega sklada.

Funkcijski bloki (FB) vsebujejo tehnološke funkcije in imajo rezerviran del pomnilnika za shranjevanje lokalnih spremenljivk do naslednjega klica bloka.
Lastnosti:

- uporablja statične podatke,
- dodeljeni podatkovni blok, in
- Primerno za programiranje kompleksnih, pogosto uporabljenih funkcij.

Funkcije (FC) prav tako vsebujejo tehnološke funkcije, vendar nimajo pomnilniške funkcije za svoje lokalne spremenljivke.
Lastnosti:

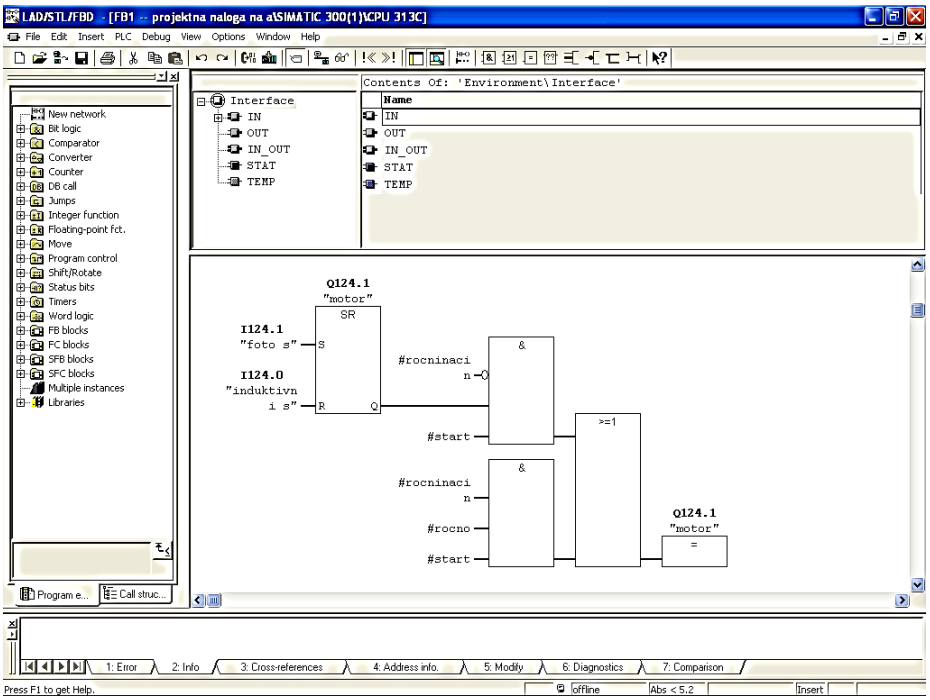
- vrne njegov rezultat (brez statičnih podatkov) programu, in
- Večinoma brez pomnilnika (nimajo dodeljenega podatkovnega bloka).

Podatkovni bloki (DB) se uporabljajo za shranjevanje različnih uporabniških podatkov, ki so na voljo vsem delom aplikacije.
Lastnosti:

- Strukturirana shramba lokalnih podatkov, in
- Strukturno shranjevanje globalnih podatkov (podatki so na voljo celotni aplikaciji).

Sistemski bloki (SFB, SFC, SDB) so bloki, ki vsebujejo sistemske funkcije. Ti bloki so sestavni del operacijskega sistema in ne uporabljajo uporabniškega pomnilnika. Bloke lahko uporabnik uporablja v svojem aplikacijskem programu. SDB-ji vsebujejo podatke za konfiguracijo posameznih modulov in komunikacije.
Lastnosti:

- Sistemski funkcijski bloki (SFB) so integrirani v operacijski sistem procesorja,
- sistemske funkcije (SFC) so integrirane v operacijski sistem procesorja in jih lahko uporablja le uporabnik, vendar jih ni mogoče spreminjati; podobno kot FC blok je SFC blok brez statičnih spremenljivk in
- Sistemski podatkovni bloki (SDB) vsebujejo podatke za konfiguracijo sistema.



Slika 60: Program v FB1

4.7. KORAK 7 – PROGRAMSKA OPREMA IN KONFIGURACIJSKA OPREMA ZA SIMATIC

STEP 7 je programska oprema in konfiguracijska programska oprema za SIMATIC S7. Sestavljen je iz več posameznih aplikacij, od katerih vsaka opravlja določeno funkcijo. Tako imamo funkcije, ki nas spremljajo od začetka ustvarjanja projekta do njegovega zaključka.

Funkcije lahko razvrstimo v naslednje skupine:

- Funkcije konfiguracije strojne opreme
- Funkcije za konfiguracijo omrežij,
- Programske funkcije,
- Funkcije testiranja in servisiranja
- Funkcije dokumentacije in arhiviranja.

Glavni grafični vmesnik v STEP 7 je SIMATIC Manager. Zbere vse potrebne podatke iz različnih aplikacij za načrtovanje celotnega projekta. V samem projektu so podatki razdeljeni po funkcijah in predstavljeni kot objekti. Ko želimo delati na posameznem projektu, se aktivira tudi ustrezno orodje za delo s tem objektom.

Obstajata dva načina programiranja:

- neposreden vnos programa v centralno enoto (spletni programi) in
- Programiranje pomnilniškega modula v programirano napravo brez povezav z avtomatizirano napravo.

Pomnilniški modul se nato vstavlja v osrednjo enoto (off-line programiranje). Programiranje uporabniškega programa poteka zaradi strukture krmilnika na ločeni programirani napravi. Prenos programa iz programirane naprave na krmilnik je mogoče izvesti s prenosom programa v pomnilnik EPROM in kasnejšim vstavljanjem v krmilnik, ali pa se program programira neposredno v pomnilnik krmilnika. To zahteva večtočkovno povezavo (MPI) med programirano napravo in krmilnikom.

Programski jezik STEP 7 nam omogoča programiranje s tremi programskimi metodami, ki so značilne za pomnilniško programabilne kontrole:

- Funkcijski blok diagram *(FBD*)
- kontaktni način – LAD (*Ladder Logic*),
- nabor ukazov – STL (*Statement List*).

FUNKCIONALNI NAČRT – FBD:

FBD je programska metoda, pri kateri se ukazi vnašajo z grafično metodo. Načrt funkcije uporablja logične bloke, znane iz Booleove algebre. Uporabite ga lahko za kompleksne funkcije, ki jih lahko neposredno predstavite logični bloki. Njegova velika prednost je neposreden vnos

programa, saj je programiranje z logičnimi bloki enostavno in hitro za razumevanje. Vendar ta metoda programiranja ni primerna za programiranje računalniških operacij in bolj zapleteno obdelavo digitalnih vrednosti.

NAČRT STIKOV – LAD:

LAD je tudi grafična programska metoda, kjer so ukazi predstavljeni s simboli, izpeljanimi iz kontaktne tehnike. Glavna elementa kontaktnega načrta sta delovno in mirno stikalo, zato se ga je enostavno naučiti. Slabost te programske metode je neprosojnost, zato ni primerna za zahtevne naloge.

NABOR UKAZOV – STL:

STL je programska metoda, kjer je mogoče ukaze vnesti v pisni obliki. Ta programski jezik je zelo podoben asemblerju in je sestavljen iz serije mnemotehničnih ukazov – to so izvršilni ukazi. Vsak ukaz ustreza koraku procesorja skozi program. Prednost nabora ukazov je v hitri izvedbi programa, saj sta tako čas obdelave kot lokacija pomnilnika optimizirana. V samem urejevalniku je delo še lažje, saj lahko uporabite simbolni opis, poleg tega pa so na voljo tudi funkcije za iskanje in preverjanje pravilnega vnosa v realnem času. Urejevalnik omogoča shranjevanje ali shranjevanje ponovno uporabljenih programskih delov v standardni programski knjižici, kar omogoča kasnejšo ponovno uporabo. Vsak ukaz v STL je sestavljen iz operacije in operanda. Operand je nadalje razdeljen na oznako operanda in parameter. V samem ukazu je mogoče vnesti tudi oznako, ki se uporablja za skoke, v urejevalniku pa lahko vnesemo komentarje za lažje razumevanje programa.

KRMILNIKI ZA SIMATIC S7:

- Standardni PID krmilnik omogoča uporabo neprekinjenih regulatorjev, regulatorjev za pulzne in stopenjske odzive v uporabniškem programu,
- Modularni PID krmilnik se uporablja, če standardni PID ni zadosten za rešitev avtomatizacijske naloge, in
- Zamegljeni krmilniki se uporabljajo za ustvarjanje logičnih sistemov. Ti sistemi se uporabljajo, kadar je proces težko ali nemogoče matematično opisati, ko se procesi obnašajo nepredvidljivo, pri obravnavi nelinearnih dogodkov, vendar so izkušnje s procesi na voljo.

PROGRAMSKA ORODJA ZA POVEZOVANJE ČLOVEKA IN STROJA:

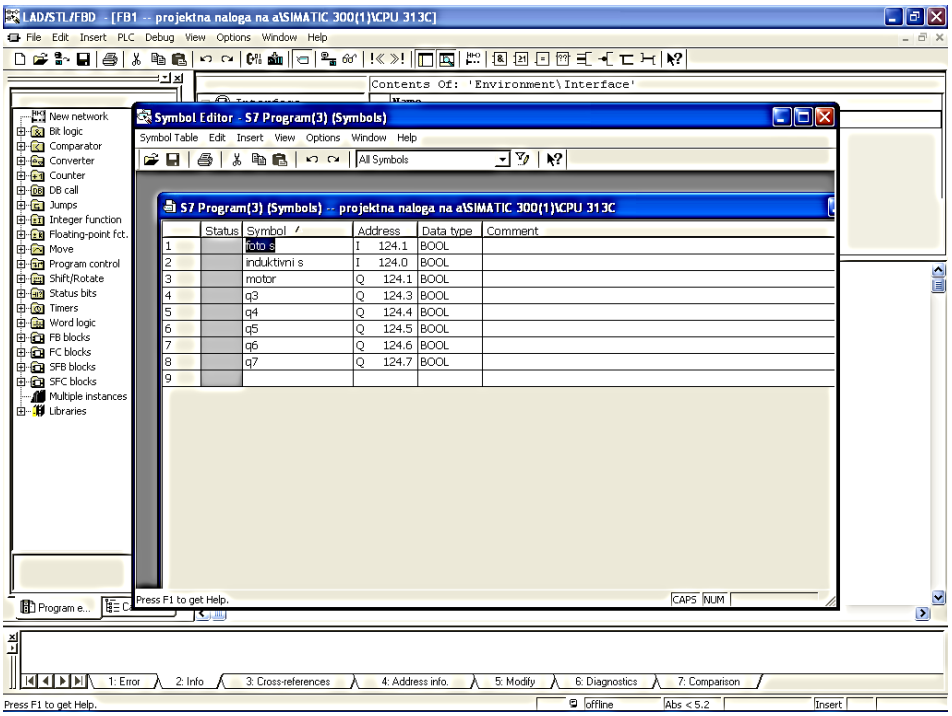
HMI je posebna programska oprema za nadzor in nadzor operaterjev:

- SIMATIC WinCC SCADA je odprta vizualizacija procesov,
- SIMATIC ProoTool in SIMATIC ProoTool/Lite sta sodobna orodja za konfiguracijo SIMATIC operaterskih panelov in kompaktnih enot SIMATIC C7 (različica Lite je namenjena le besedilnim panelom) in
- Pro Agent omogoča hitro, načrtovano diagnostiko procesov v načrtih in strojih v iskanju informacij o lokaciji in vzrokih napak.

4.8. PROGRAMIRANJE KRMILNIKOV

Najprej morate napisati tabelo simbolov za vhodne/izhodne signale, ki jo kasneje dopolnite z označevalci, števci in časovniki. Poleg tega je tabela simbolov zapisana po standardu, vse lokacije v krmilniku so vnaprej določene. Tabela simbolov se uporablja za lažje programiranje, saj bi bil tak program v primeru zapisovanja signalov le po lokacijah v krmilniku zelo nejasen in bi ga verjetno bilo težko le dokončati.

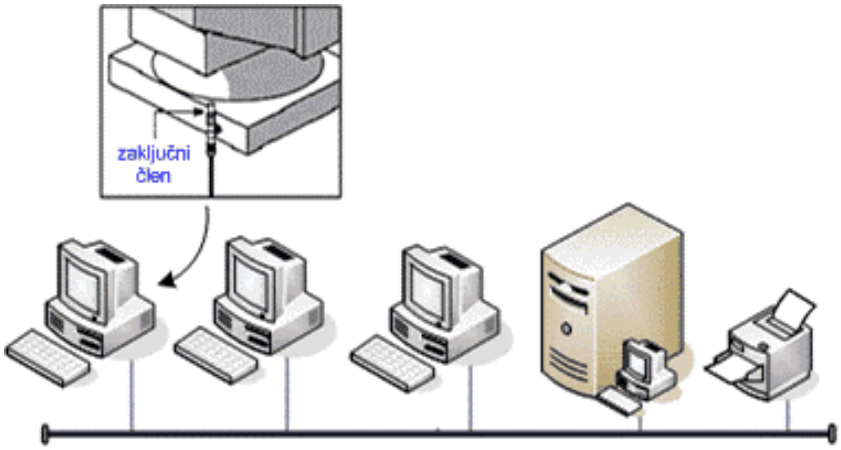
Spodnja slika prikazuje del tabele simbolov za vhodne signale. Ena vrstica tabele predstavlja en signal, in sicer ime (*simbol*), naslov ali lokacijo v krmilniku (*naslov),* podatkovni tip (*podatkovni tip*) in komentar (komentar) signala. Zastavice, ki jih vidimo na začetku vrste, pravijo, da se ti signali uporabljajo v varnostnih blokih.



Slika 61: Tabela simbolov

4.9. POVEZOVANJE KRMILNIKA Z ETHERNET OMREŽJEM

Ethernet je najbolj priljubljen protokol za lokalna omrežja. Ta industrijski standard so sprejeli številni proizvajalci omrežne strojne opreme. Danes je veliko težav, ki bi lahko nastale zaradi nezdružljivosti strojne opreme različnih proizvajalcev, praktično neobstoječih pri Ethernetu. Danes Ethernet omrežja delujejo s hitrostmi 10, 100 in 1.000 Mbps (1 Gbps), kar omogoča aplikacijam iz domačih in malih poslovnih omrežij dostop do visokozmogljivih omrežnih hrbtenic.



Slika 62: Omrežna povezava
Vir: http://www.s-sers.mb.edus.si/gradiva/w3/omrezja/01_omrezja/01_omrezja.html

Standardi fizične plasti Ethernet standarda opisujejo vrste kablov, iz katerih lahko zgradite omrežje, določajo topologijo, največjo dolžino kabelskega segmenta in število obnovitev, ki jih je mogoče uporabiti. Pomembno je upoštevati standarde, saj je mehanizem CSMA/CD občutljiv na preklapljanje in dušenje.

Miza 3:Standardi

Oznaka	Kabel	Topologija	Hitrost	Segment
10Base5	RG8 koaksialni	Vodnik	10 Mbps	500 metrov
10Base2	RG58 koaksialni kabel	Vodnik	10 Mbps	185 metrov
10BaseT	Kategorija 3 UTP	Zvezda	10 Mbps	100 metrov
FOIRL	Večspolno optično vlakno 62,5/125	Zvezda	10 Mbps	1.000 metrov
10BaseFL	Večspolno optično vlakno 62,5/125	Zvezda	10 Mbps	2.000 metrov
10BaseFB	Večspolno optično vlakno 62,5/125	Zvezda	10 Mbps	2.000 metrov

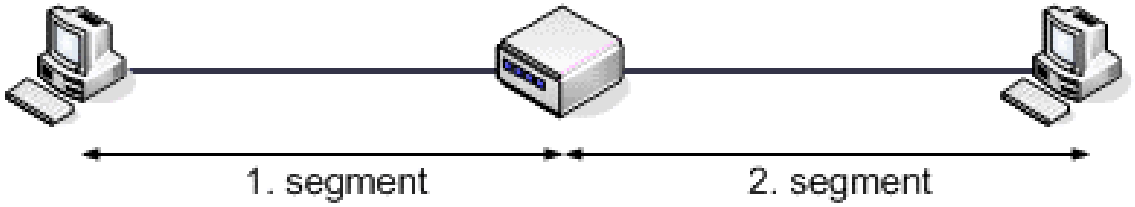
10BaseFP	Večspolno optično vlakno 62,5/125	Zvezda	10 Mbps	500 metrov
100BaseTX	Kategorija 5 UTP	Zvezda	100 Mbps	100 metrov
100BaseT4	Kategorija 3 UTP	Zvezda	100 Mbps	100 metrov

4.10. UTP ETHERNET OMREŽJE

Vse druge fizične Ethernet topologije uporabljajo zvezdasto topologijo, pri kateri vsak segment povezuje računalnik z vozliščem. Najbolj priljubljen kabel v današnjem Ethernetu je nezaščiteni par (UTP). Omogoča povezave pri hitrostih 10 Mbps, 100 Mbps in 1000 Mbps. 10BaseT uporablja le dva para, enega za sprejem in enega za oddajanje.

Hitri Ethernet (IEEE 802.3u) opisuje dve specifikaciji, ki prav tako dovoljujeta največjo dolžino segmenta 100 metrov. 100BaseTX zahteva kable kategorije 5, ki imajo boljši prenos signala. 100BaseT4 pa vam omogoča nadgradnjo omrežja s kabli kategorije 3. 100BaseT4 uporablja vse štiri pare.

Večina standardov za gigabitni Ethernet predpisuje optične kable, obstaja pa tudi možnost uporabe UTP v skladu s standardom IEEE 802.3ab. Standard 1000BaseT opisuje možnost nadgradnje obstoječega omrežja s kabli kategorije 5 (še bolje, če so kabli pogosto označeni kot 5E po novih, strožjih kriterijih). 1000BaseT doseže višje hitrosti z uporabo vseh štirih parov in PAM–5 pulzno-amplitudno modulacijo.

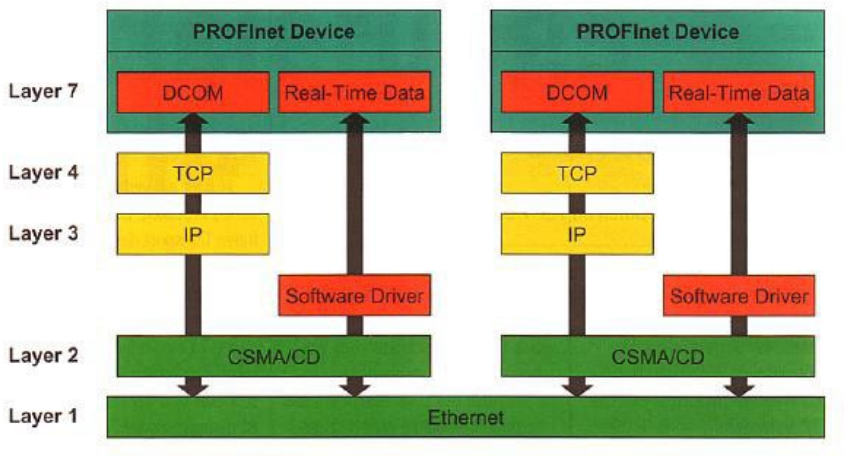


Slika 63: UTP Ethernet
Vir: http://www.s-ers.mb.edus.si/gradiva/w3/omrezja/01_omrezja/01_omrezja.html

4.11. NETWORK PROFINET

Vzporedno z industrijskim Ethernetom že 15 let obstaja najbolj razširjen in najbolj razširjen realnočasovni protokol Profibus, ki ga podpira Siemens. Omrežje Profibus je zasnovano za hitro in zanesljivo komunikacijo v avtomatizacijskih procesih. Največja hitrost prenosa podatkov je 12 Mbit/s, kar je bilo ob uvedbi revolucionarno, danes pa ne pokriva več zahtev. Težave se lahko pojavijo tudi s številom postaj, povezanih v omrežje, ki lahko doseže do 127, vsak repetitor, ki poveča doseg omrežja, pa predstavlja tudi eno od teh naprav. Omrežje Profibus tako zaradi teh pomanjkljivosti dosega mejo svoje zmogljivosti

Siemens dela na novem protokolu s pomočjo nekaterih partnerjev. Visoke hitrosti prenosa Ethernet odpirajo nove možnosti za komunikacijo v realnem času. Ethernet v svoji klasični različici ne zagotavlja deterministčnega vedenja, za nekatere sisteme pa je determinizem prenosnega časa ključen. V industriji so procesi razdeljeni glede na reakcijski čas. Za večino naprav je dovoljen odzivni čas med 10 in 100 ms, kar je doseženo z uporabo klasičnega industrijskega Ethernet-a z ustreznimi gonilniki. Profinet V2 ponuja optimiziran programski komunikacijski kanal. Sinhronizacija ure pa postavlja še večje zahteve glede obnašanja v realnem času.



Slika 64: Profinet

Povzetek:

Krmilnik je programabilna naprava, ki vsebuje svoj mikroprocesor, RAM in visokohitrostne vhodno-izhodne enote. Omogoča regulacijo in nadzor celotne proizvodnje ali le ene naprave, lahko sprejema signale preko senzorjev ali neposredno iz krmilnega objekta. Ti signali se nato obdelajo v krmilniku. Procesirani signali potujejo nazaj do krmilnega objekta. Komunikacija med PC-jem in programabilnim logičnim krmilnikom običajno poteka preko serijskega priključka ali TCP/IP protokola. Programabilni krmilnik je v bistvu CPU (*centralna procesna enota*), ki vsebuje program in je povezan z vhodnimi/izhodnimi napravami. Program nadzoruje PC (*programabilni krmilnik*) v tem smislu: ko se vhodni signal na vhodni napravi spremeni, se ustvari ustrezni odziv (glede na program, zapisan v pomnilniku krmilnika). Vhodne naprave so lahko fotoelektrični senzorji, stikala, omejevalniki in končna stikala. Izhodni signali so lahko 24V napetost, tokovni signal, signal, ki vklopi rele.

VPRAŠANJA:

1. Struktura krmilnega sistema. Kaj je napajalnik?

2. Vrste oddajnikov signalov.
3. Značilnosti in prednosti zaporednih kontrol – na kaj jih razdelimo?
4. Za kaj uporabljamo časovne funkcije in njihove vrste?
5. Naštej glavna področja uporabe programabilnih krmilnikov.
6. Razdelitev SIMATIC krmilnih sistemov na podskupine in glede na velikost.
7. Opišite in narišite osrednjo procesno enoto.
8. Kaj veste o Siemensovi programski opremi?
9. Opišite kontaktni načrt (LAD).
10. Kakšna je razlika med kompaktnim in modularnim PLK-jem? Naštejte prednosti in slabosti vsake vrste.

5. KAZALO SLIK

6. KAZALO TABEL

7. PRILOGE

LCAMP

Learner Centric Advanced Manufacturing Platform

Co-funded by
the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.