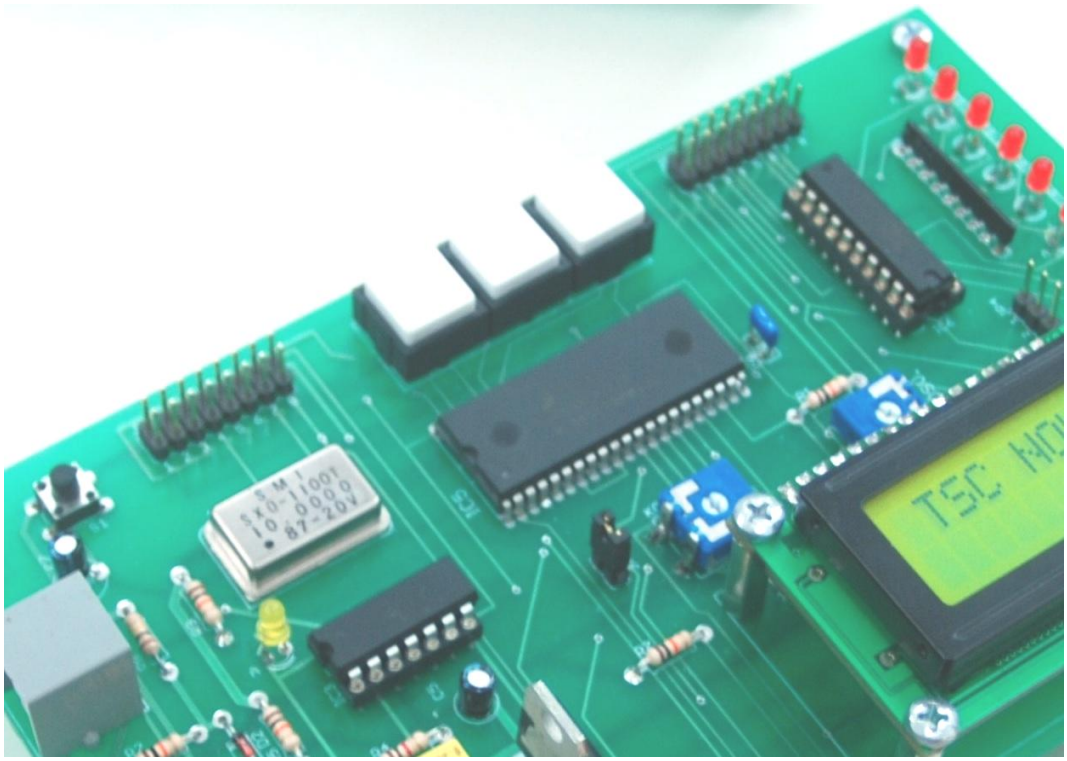
REPUBLIKA SLOVENIJA
MINISTRSTVO ZA IZOBRAŽEVANJE, ZNANOST, KULTURO IN ŠPORT

KONZORCIJ ŠOLSKIH CENTROV

Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski sklad za regionalni razvoj

ELEKTROTEHNIKA

MIKROPROCESORJI



Renato Reščič

www.bodiprofi.si

munus 2

SPLOŠNE INFORMACIJE GRADIVA

Izobraževalni program: Elektrotehnika
Ime modula: Uporaba mikroprocesorskih naprav
Naslov učnih tem ali kompetenc, ki jih obravnava učno gradivo:

Zgradba in delovanje mikroprocesorjev
Uporaba mikroprocesorskih sistemov
Razvojno okolje mikroprocesorjev
Programiranje mikroprocesorjev
Krmiljenje, zajemanje podatkov in regulacije z mikroprocesorskim vezjem

Naslov enote učnega gradiva: Mikroprocesorji

POVZETEK

Spoznali bomo zgradbo in delovanje mikroprocesorjev in mikrokontrolerjev ter njihovo uporabo v sistemih za realizacijo krmilij in regulacij. Na primeru mikrokontrolerja MC908GP32 proizvajalca *Freescale, Inc.* se bomo naučili programiranja v zbirnem jeziku. Videli bomo, kako se mikrokontroler uporablja v realnem času z uporabo časovnikov, kako nanj priključimo in uporabimo senzorje in podobno. S temami, ki so tukaj zajete, se dijaki srečajo delno pri modulu *Upravljanje s programirljivimi napravami* in v višjih letnikih pri modulu *Uporaba mikroprocesorskih naprav*. To gradivo je namenjeno prav podpori slednjemu.

Ključne besede: mikrokrmilnik, mikrokontroler, pomnilnik, programiranje, zbirni jezik, časovnik, analogno-digitalni pretvornik, senzor, LCD prikazovalnik, servomotor, pulzno-širinska modulacija, PWM signal

Avtor: Renato Reščič
Recenzent: Edvard Ipavec
Lektorica: Bojana Modrijančič Reščič
Datum: september 2012

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za izobraževanje, znanost, kulturo in šport.

KAZALO

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za izobraževanje, znanost, kulturo in šport.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za izobraževanje, znanost, kulturo in šport.

KAZALO SLIK

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za izobraževanje, znanost, kulturo in šport.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za izobraževanje, znanost, kulturo in šport.

KAZALO TABEL

KAZALO ENAČB

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za izobraževanje, znanost, kulturo in šport.

PREDSTAVITEV CILJEV

V vsakdanje življenju se na vsakem koraku srečujemo z elektronskimi napravami, ki skrbijo za naše dobro počutje, varnost, zabavo in drugo. Tako praktično vsak dan uporabljamo komunikacijske, avdio- in videonaprave, kuhinjske aparate, stroje za pranje in pomivanje ter še veliko drugih naprav.

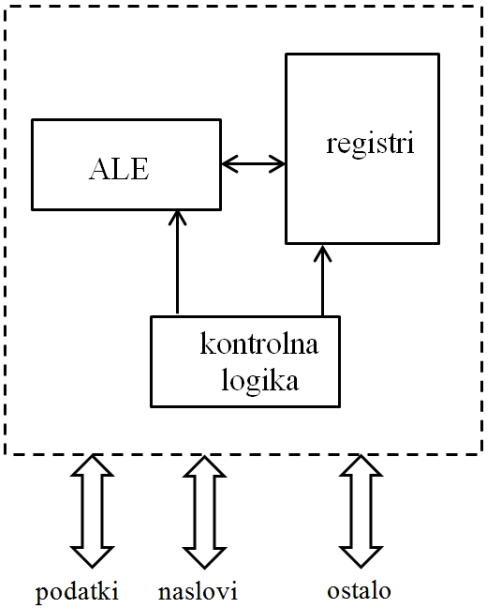
Večino teh naprav krmilijo mikroprocesorji in mikrokontrolerji. Poznavanje zgradbe in delovanja le-teh nam olajša uporabo naprav ter pomaga razumeti odzivanje naprav v raznih situacijah. Če se potrudimo, si manjše pripomočke lahko izdelamo tudi sami. Znanje s tega področja spretno uporabljajo tudi strokovnjaki v razvojnih oddelkih podjetij, ki proizvajajo tovrstne izdelke. Del njihovega znanja lahko pridobimo z doseganjem spodaj zastavljenih ciljev:

- poznavanje zgradbe, delovanja in uporabe mikroprocesorskih vezij
- razvijanjem algoritmičnega razmišljanja
- uporabo razvojnega okolja mikroprocesorja (WINIDE)
- znanje programiranja v zbirnem jeziku
- uporabo znanih programskih rešitev v novih situacijah
- priključevanjem in uporabo senzorjev
- pridobivanjem podatkov o elektronskih komponentah iz različnih virov informacij
- zmožnostjo dela v skupini

To gradivo nam bo pomagalo doseči te cilje.

MIKROPROCESORJI

Mikroprocesor je element, ki lahko opravlja logične in aritmetične operacije. Zgrajen je iz aritmetično-logične enote (ALE), registrov in kontrolne logike. Aritmetično-logične enota ima vhode za ukaze in podatke. Binarna vrednost na vhodih za ukaze določa vrsto operacije, ki so bo izvedla. Rezultat operacije se pojavi na izhodih ALE. Podatki in rezultati se ohranjajo v ustreznih registrih, za pravilno in usklajeno delovanje vseh komponent pa skrbi kontrolna logika. Sam mikroprocesor lahko izvrši ukaz glede na binarno kombinacijo (ukaz), ki ga prejme.



Slika 1: Blokovna shema mikroprocesorja

Vendar samo z mikroprocesorjem ne moremo krmiliti nobene naprave ali opraviti kakršnekoli druge naloge, saj moramo program, po katerem naj bi mikroprocesor delal, nekje shraniti. Poleg tega mu moramo dati vhodne podatke ali pogoje, na katere se bo ustrezno odzival, upravljal z napravami ali prikazoval stanja in vrednosti. Zato ga z drugimi elementi združimo v mikroprocesorski sistem.

Ponovimo

Mikroprocesor je sestavljen iz aritmetično-logične enote in registrov. Kontrolna logika skrbi, da ALE dobi ukaze in lahko izvaja zahtevane operacije. Vendar pa mikroprocesor rabi še druge enote, s katerimi ga povežemo v mikroprocesorski sistem, ki lahko opravlja naloge s področja krmiljenja in regulacij po napisanem programu.

MIKROPROCESORSKI SISTEMI

Minimalni mikroprocesorski sistem

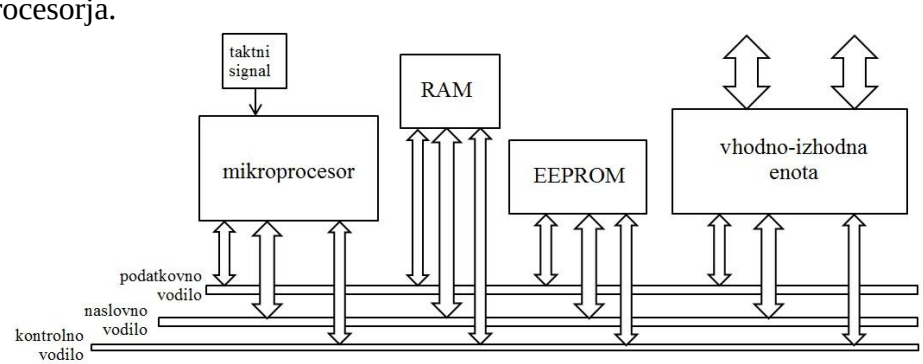
Kako sestavimo mikroprocesorski sistem, ki nam lahko po naših zahtevah krmili ali regulira določen proces? Ker je mikroprocesor element, ki vhodne podatke obdela in postavi izhode v ustrezna stanja, mora za svoje delovanje imeti tudi vhodno-izhodne vmesnike.

Odzivanje mikroprocesorja na vhodne parametre določa program. Le-tega moramo shraniti v pomnilnik, ki podatke ohranja tudi po izklopu napajalne napetosti, saj le tako lahko vedno zagotovimo pravilno delovanje naprave, ki jo mikroprocesor upravlja. Torej potrebujemo tudi pomnilnik, večinoma je to pomnilnik z naključnim dostopom (EEPROM oz. *Flash* EEPROM). Med delovanjem mikroprocesor shranjuje nekatere vmesne podatke. Ker shranjevanje podatkov v *Flash* EEPROM zahteva izvajanje predpisanih korakov in je zato zamudno, take vrednosti raje shranjujemo v bralno-pisalni pomnilnik RAM.

Vsi ti elementi morajo delovati časovno usklajeno, za kar skrbi taktni signal (*Clock*). Ta tudi določa hitrost izvajanja ukazov in s tem odzivnost sistema.

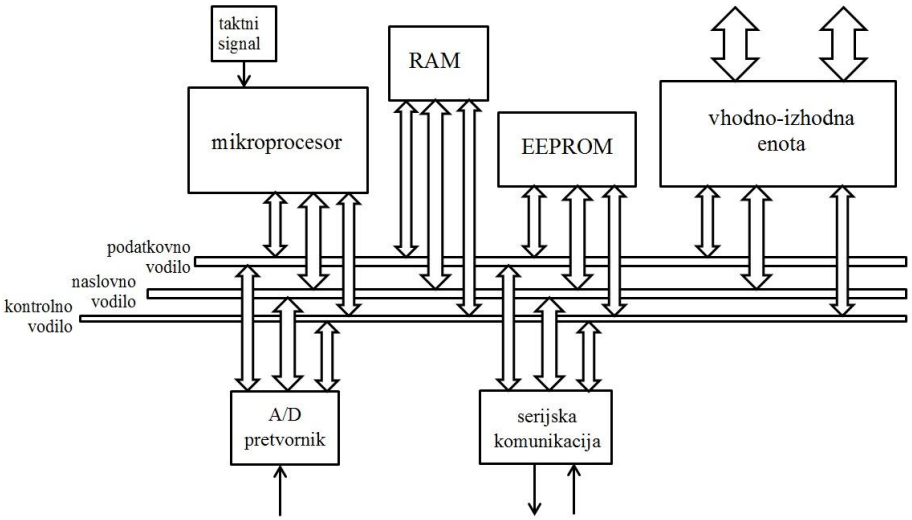
Enote morajo biti med sabo povezane. Poznamo tri vrste povezav, ki jim rečemo vodila:
- podatkovno vodilo
- naslovno vodilo
- kontrolno vodilo

Podatkovno vodilo poskrbi, da se iz pomnilnika prenašajo ukazi v mikroprocesor ter da se podatki lahko prenašajo med mikroprocesorjem, pomnilniki in vhodno-izhodnimi vmesniki. Pomnilniki in vhodno-izhodne enote ohranjajo podatke na naslovih, ki jih moramo pri programiranju upoštevati. Signali naslovnega vodila določajo, kateri ukaz ali podatek bo na voljo na podatkovnem vodilu ali kam se bo nek podatek shranil. Kontrolno vodilo pa sestavljajo povezave, ki prenašajo taktni signal, stanje na Reset ali IRQ (prekinitvenem) priključku mikroprocesorja, povezave za napajalno napetost in drugi. Število in vrsta signalov je odvisna od tipa mikroprocesorja.



Slika 2: Minimalni mikroprocesorski sistem

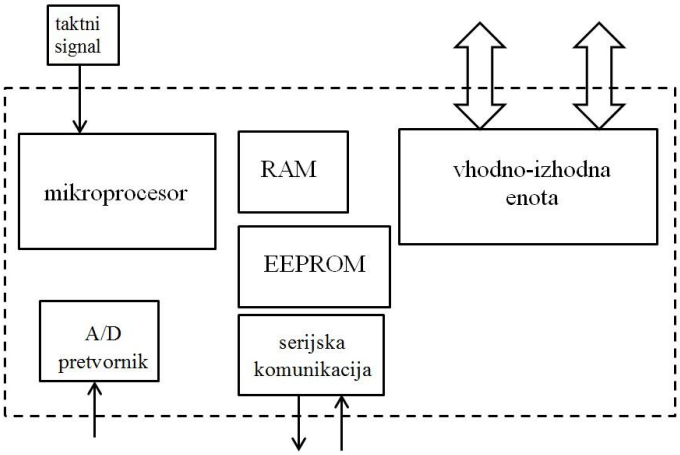
Če z mikroprocesorjem nadziramo fizikalne veličine (temperaturo, svetlobo, silo in podobno), potrebujemo senzor, ki nam to veličino pretvori v električno napetost. Zato mikroprocesorskemu sistemu dodamo še ustrezno povezan analogno-digitalni pretvornik. Za povezavo dveh ali več mikroprocesorskih sistemov med seboj ali povezavo mikroprocesorja z računalnikom potrebujemo še vmesnik za serijsko komunikacijo.



Slika 3: Razširjen mikroprocesorski sistem

Mikrokrmilnik (mikrokontroler)

Kot vidimo na slikah 2 in 3, ima tak sistem poleg mikroprocesorja še veliko drugih elementov in temu ustrezno število povezav med njimi. To povzroči veliko težav in stroškov pri projektiranju in izdelavi tiskanih vezij za tak sistem. Zato so proizvajalci združili posamezne enote takega sistema v eno integrirano vezje, ki mu pravimo mikrokrmilnik ali mikrokontroler (iz angleščine *microcontroller*). Ker ima tako integrirano vezje že vse komponente mikroprocesorskega sistema vgrajene in pravilno povezane med seboj, je potrebno zelo malo dodatnih elektronskih komponent, da lahko sistem uporabimo za želeno nalogo. Tako nam proizvajalci olajšajo projektiranje in izdelavo tiskanega vezja, izdelki so lahko manjših dimenzij. Zaradi vsega tega in majhnega števila komponent so zato izdelki z mikrokrmilnikom cenejši, kar poveča možnosti uporabe mikrokrmilnikov na vseh področjih.



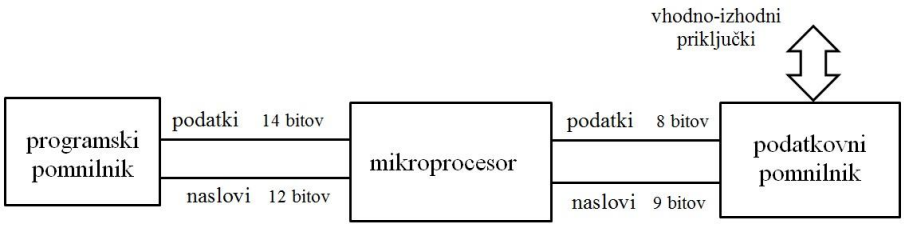
Slika 4: Mikrokontroler

Arhitektura mikroprocesorskega sistema oz. mikrokrmilnika

Proizvajalci so pri izdelavi mikroprocesorjev in kasneje mikrokontrolerjev prišli do dveh zasnov zgradbe in povezav njegovih enot. Tako obstajata Harvardska in Von Neumannova zasnova oziroma arhitektura. Vsaka ima svoje prednosti in tudi slabosti.

Harvardski model

Pri tem modelu mikroprocesorja imamo ločena pomnilnika za podatke in ukaze. Ukazni in podatkovni pomnilnik ponavadi nimata enake bitne širine vodila. Pri 8-bitnem mikroprocesorju so podatki 8-bitni in tako je organiziran tudi podatkovni pomnilnik. Ukazni pomnilnik pa je lahko npr. 12- ali 16-bitni. Tako dolžino imajo tudi ukazi takega mikroprocesorja oziroma mikrokrmilnika. Pomnilnika se razlikujeta tudi v velikosti, zato imata različni naslovni vodili. Oba pomnilnika sta razdeljena na posamezne strani oziroma "banke", kar pomeni, da do neke pomnilniške lokacije ali registra ne moremo dostopati vedno v enem koraku, ampak moramo najprej priti do ustrezne "banke", šele nato imamo dostop do želene lokacije ali registra. Naslovi se na straneh oziroma "bankah" ponavljajo, zato en naslov ne pomeni točno določenega mesta v pomnilniku.

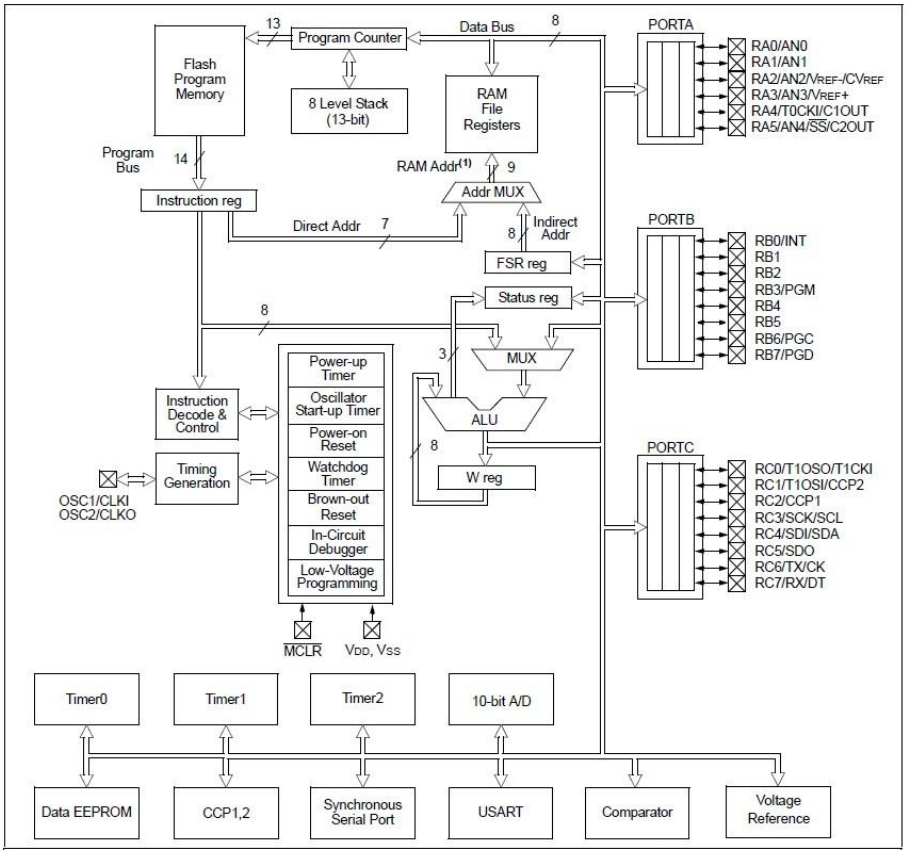


Slika 5: Harvardski model mikroprocesorja

Mikroprocesor lahko pri Harvardski zasnovi hkrati prebere oziroma naloži ukaz in podatek v svoje registre, saj potujeta vsak po svojih povezavah. To omogoča hitrejše izvajanje programov in s tem odziv naprave. Ukazi se izvedejo že v eni ali dveh periodah taktnega signala.
Taki mikroprocesorji in mikrokontrolerji imajo manjši nabor ukazov (*RISC - Reduced Instruction Set Computers*). To nam včasih lahko povzroči probleme pri programiranju, kar posledično poveča število ukazov za neko operacijo.

Tipična predstavnika take zasnove sta mikrokontrolerja PIC proizvajalca *Microchip Technology, Inc.* ter AVR proizvajalca *Atmel Corporation.*

Na naslednji sliki je blokovna shema mikrokontrolerja PIC 16F877, kjer lahko med drugim vidimo ločena programski in podatkovni pomnilnik ter različne širine vodil za ukaze in podatke:

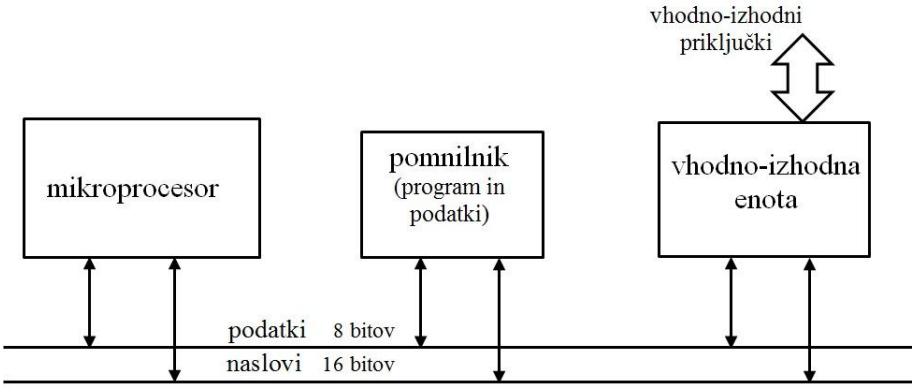


Slika 6: Zgradba mikrokontrolerja PIC 16F877
(Vir: PIC16F87XA Data Sheet, Microchip Technology Inc., 28.7.2003, (pdf datoteka proizvajalca))

Von Neumannov model

Ta zasnova temelji na enotnem pomnilniškem prostoru, ki je namenjen shranjevanju podatkov in programske kode. Pri 8-bitnem mikroprocesorju je tudi celoten pomnilnik 8-bitni. Z določevanjem naslovov na naslovnem vodilu lahko pridemo do podatkov ali ukazov. Vsak naslov določa samo eno mesto v pomnilniku.

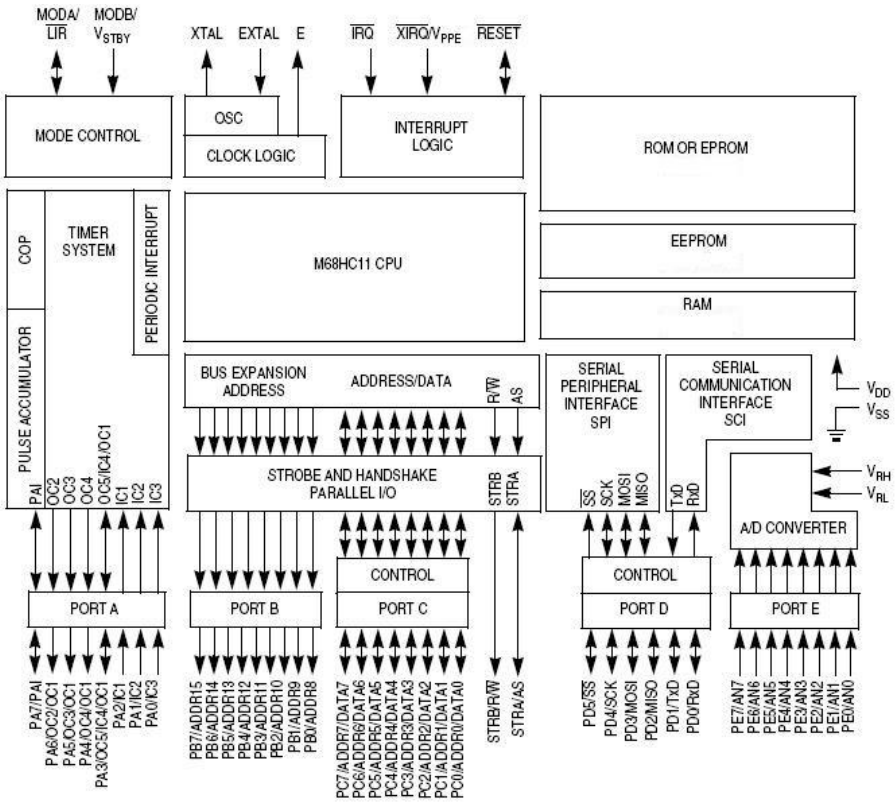
Ker se podatki in ukazi prenašajo po istem (podatkovnem) vodilu, je izvrševanje ukazov počasnejše kot pri Harvardski zasnovi. Mikroprocesor mora namreč najprej dobiti ukaz, zatem pa še podatek za obdelavo. Ukazi se izvajajo različno dolgo – ponavadi od enega do 5 period taktnega signala, pri posebnih ukazih tudi do 9 period.



Slika 7: Von Neumannov model mikroprocesorja

Pri mikroprocesorjih s tako zasnovo imamo širok nabor ukazov (*CISC – Complex Instruction Set Computer*), kar olajša programiranje. Programi so zato krajši.

Freescale Semiconductor, Inc. (bivše Motorolino podjetje) je eden od proizvajalcev mikrokontrolerjev na osnovi tega modela. Na spodnji sliki je blokovna shema enega izmed njihovih 8-bitnih mikrokontrolerjev, v nadaljevanju pa bomo podrobno spoznali njihov mikrokontroler MC908GP32.



Slika 8: Zgradba mikrokontrolerja MC68HC11
(Vir: http://www.freescale.com/files/microcontrollers/doc/data_sheet/M68HC11E.pdf)

Na shemi vidimo, da pomnilnik ni posebej ločen na programski in podatkovni.

Programiranje mikroprocesorja oz. mikrokrmilnika

Program za mikroprocesor ali mikrokontroler je zaporedje operacij, ki jih mora izvajati aritmetično-logična enota, da realizira zahtevano nalogo. Poznamo več načinov programiranja, v vsakem primeru pa mora ALE dobiti ukaze in podatke v binarni obliki. Vsak ukaz ima svojo binarno operacijsko kodo. To je večbitna binarna vrednost, ki jo aritmetično-logična enota dobi na svoje ukazne priključke. Tako ve, kaj mora v naslednjem koraku narediti. Programiramo lahko na naslednje načine:

- v strojnem jeziku
- v zbirnem jeziku
- v višjih programskih jezikih (tekstovno, grafično)

Programiranje v strojnem jeziku pomeni pisanje logičnih enic in ničel, ki predstavljajo ukaze in operande (naslove ali podatke). Zaradi boljše preglednosti lahko v urejevalniku pišemo vse vrednosti v šestnajstiški obliki. Tako napisan program lahko naložimo v pomnilnik mikroprocesorja in ga le-ta začne izvajati. Ker pa je takšno pisanje ukazov človeku manj jasno in pregledno, so nastali razni programski jeziki, ki nam olajšajo programiranje. S pomočjo prevajalnikov nato vsak program prevedemo v binarno obliko – strojno kodo. Programski jezik,

ki je najbližji računalniku, hkrati pa tudi človeku razumljiv, je zbirni jezik ali asembler. Ukazi v tem jeziku so kratice ali okrajšave angleških izrazov za operacije, ki jih ta ukaz opravi. Tem kraticam pravimo mnemonične kode. Z nekaj osnovnega znanja angleščine lahko hitro spoznamo pomen posameznih kod.

Od tekstovnih višjih programskih jezikov se za programiranje manjših (8-bitnih) mikroprocesorjev in mikrokontrolerjev največkrat uporablja programski jezik C oz. C++. Obstajajo tudi grafična orodja za programiranje, kjer narišemo shemo, ki je praktično diagram poteka programa z vsemi potrebnimi parametri, ki določajo odzivanje programa na vhodne podatke.

V nadaljevanju bomo ob spoznavanju mikrokontrolerja spoznali tudi programiranje v zbirnem jeziku.

Ponovimo

V mikroprocesorskem sistemu imamo poleg samega mikroprocesorja še RAM in EEPROM pomnilnike ter vhodno-izhodne vmesnike. V EEPROM shranimo program, po katerem sistem deluje, RAM pa uporabljamo za začasna shranjevanja podatkov in vmesnih rezultatov med izvajanjem programa. Z vhodno-izhodnimi vmesniki omogočimo mikroprocesorju komuniciranje z zunanjim svetom. Z izhodnimi priključki vmesnikov lahko krmilimo naprave, informacije iz okolja in pogoje delovanja pa mikroprocesor sprejema preko vhodnih priključkov vmesnikov. Tak sistem lahko dopolnimo še z drugimi enotami, kot sta na primer analogno-digitalni pretvornik in vmesnik za serijsko komunikacijo med mikroprocesorji.

Proizvajalci nam ponujajo vse te in še druge enote v enem ohišju. Takim elementom pravimo mikrokrmilniki ali mikrokontrolerji. Tako projektantom in ostalim uporabnikom olajšajo izvedbo krmilnega sistema, zmanjšajo pa se tudi dimenzije izdelkov in njihova cena.

Zgradba mikroprocesorjev in mikrokontrolerjev sloni na dveh osnovnih modelih, Harvardskem in Von Neumannovem. Za prvega je značilno, da ima pomnilnik, v katerega shranimo programsko kodo (programski pomnilnik), ter pomnilnik, ki je namenjen shranjevanju podatkov in nastavitev (podatkovni pomnilnik). Tudi vodila, ki povezujejo pomnilnike z mikroprocesorjem, so različna. S tem je povezana tudi dolžina ukazov, ki se razlikuje od dolžine podatkov. Vsi ukazi se izvajajo enako hitro, le nabor ukazov je majhen.

Pri Von Neumannovem modelu je pomnilnik enoten, prav tako tudi vodilo, po katerem se prenašajo ukazi in podatki. Ukazi se izvajajo različno dolgo, nabor ukazov pa je velik.

Mikroprocesorje in mikrokontrolerje lahko programiramo v strojnem in zbirnem jeziku ter v višjih programskih jezikih, ki nam omogočajo tekstovni ali grafični način programiranja.

Vprašanja

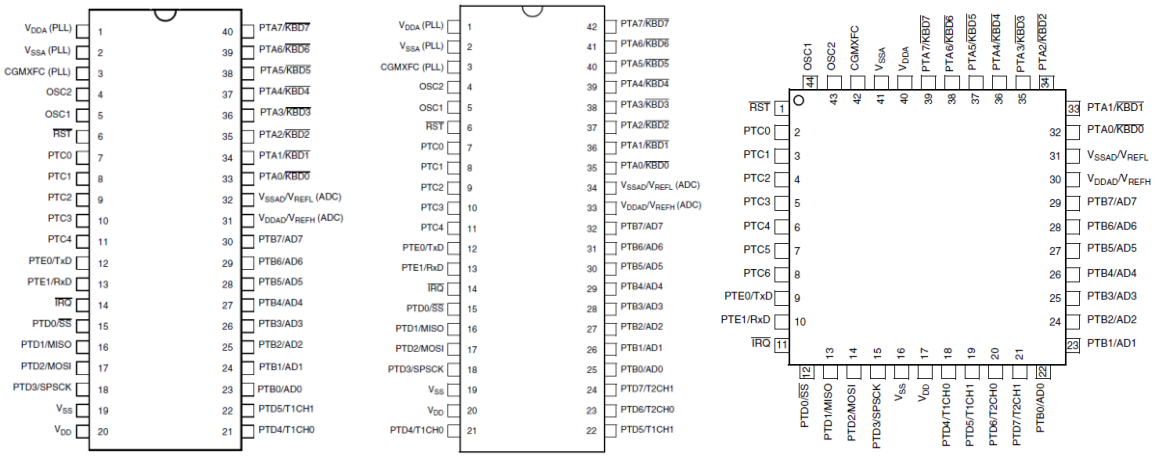
1. Katere komponente mora nujno vsebovati mikroprocesorski sistem, da lahko opravlja svojo funkcijo?
2. Kako pravimo povezavam med komponentami mikroprocesorskega sistema in katere vrste povezav poznamo?
3. Kje se v mikroprocesorskem sistemu nahaja programska koda, po kateri mikroprocesor deluje?
4. Kaj je značilno za Harvardski model mikroprocesorja?
5. Kaj je značilno za Von Neumannov model mikroprocesorja?
6. Kaj je mikrokrmilnik oz. mikrokontroler?

MIKROKRMILNIK MC908GP32

Lastnosti in ohišja

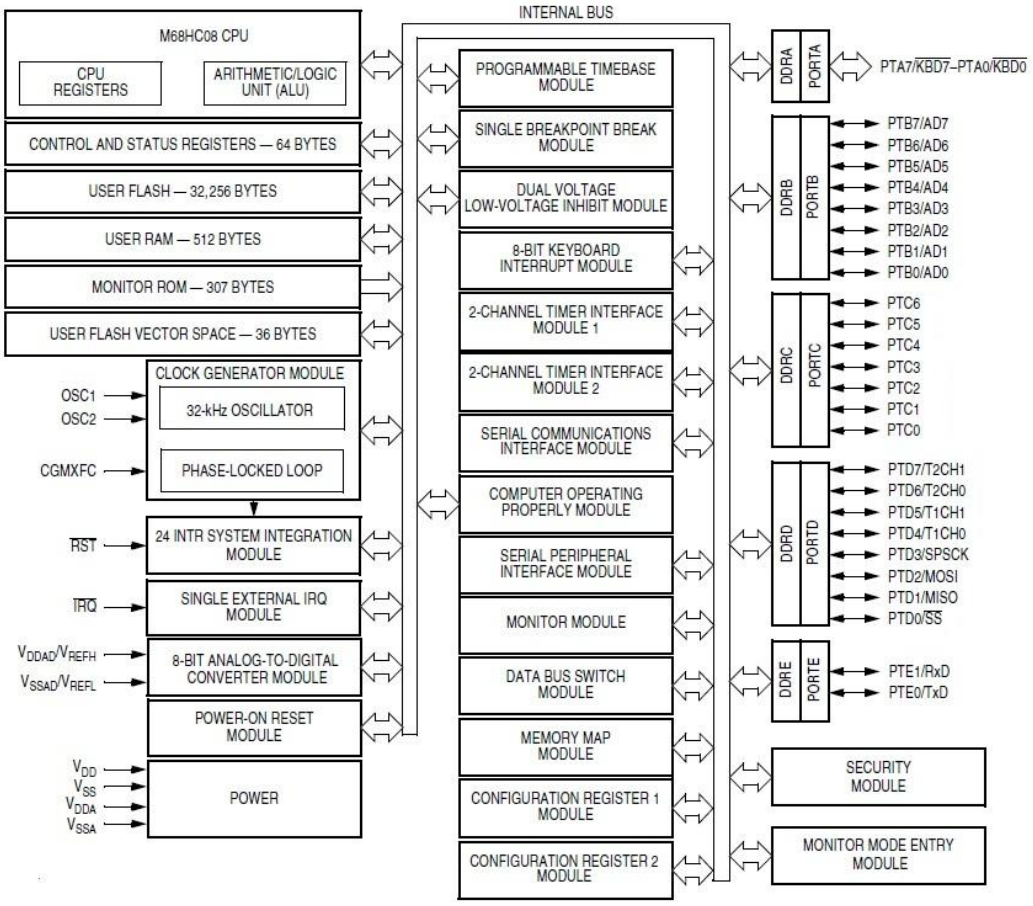
MC908GP32 je eden iz množice 8-bitnih mikrokontrolerjev podjetja *Freescale Semiconductor, Inc.*, s katerim se bomo seznanili in se ga naučili uporabljati. Njegove glavne značilnosti so:
- 8-bitna ALE,
- 16-bitno naslovno vodilo,
- 512 bajtov pomnilnika RAM,
- 32 kB *Flash* EEPROM pomnilnika z možnostjo zaščite pred branjem,
- do 33 I/O priključkov na paralelnih portih (port A – E) – obremenitve posameznih priključkov so do 10 mA, na portu C do 15 mA,
- 8-kanalni A/D pretvornik (8-bitni),
- serijska komunikacija (SCI in SPI),
- 2 večfunkcijska časovnika (timerja) z možnostjo generiranja PWM signala,
- programiranje in nadzor delovanja z računalnikom s pomočjo tovarniškega programa v pomnilniku ROM,
- delovanje z napajalno napetostjo 3 V ali 5 V,
- delovanje v načinu z majhno porabo (*Wait* in *Stop* način),
- drugo.

Dobimo ga lahko v treh različnih ohišjih:
- 40-pinsko DIP ohišje,
- 42-pinsko SDIP ohišje,
- 44-pinsko QFP ohišje.



Slika 9: Ohišja MC908GP32
(vir: MC68HC908GP32 Data Sheet, Rev. 10 1/2008, Freescale Semiconductor, Inc., (pdf datoteka proizvajalca))

Na spodnji sliki je blokovna shema mikrokontrolerja z vsemi priključki, v notranjosti pa so vidni vsi moduli, ki so na voljo uporabniku oziroma programerju:

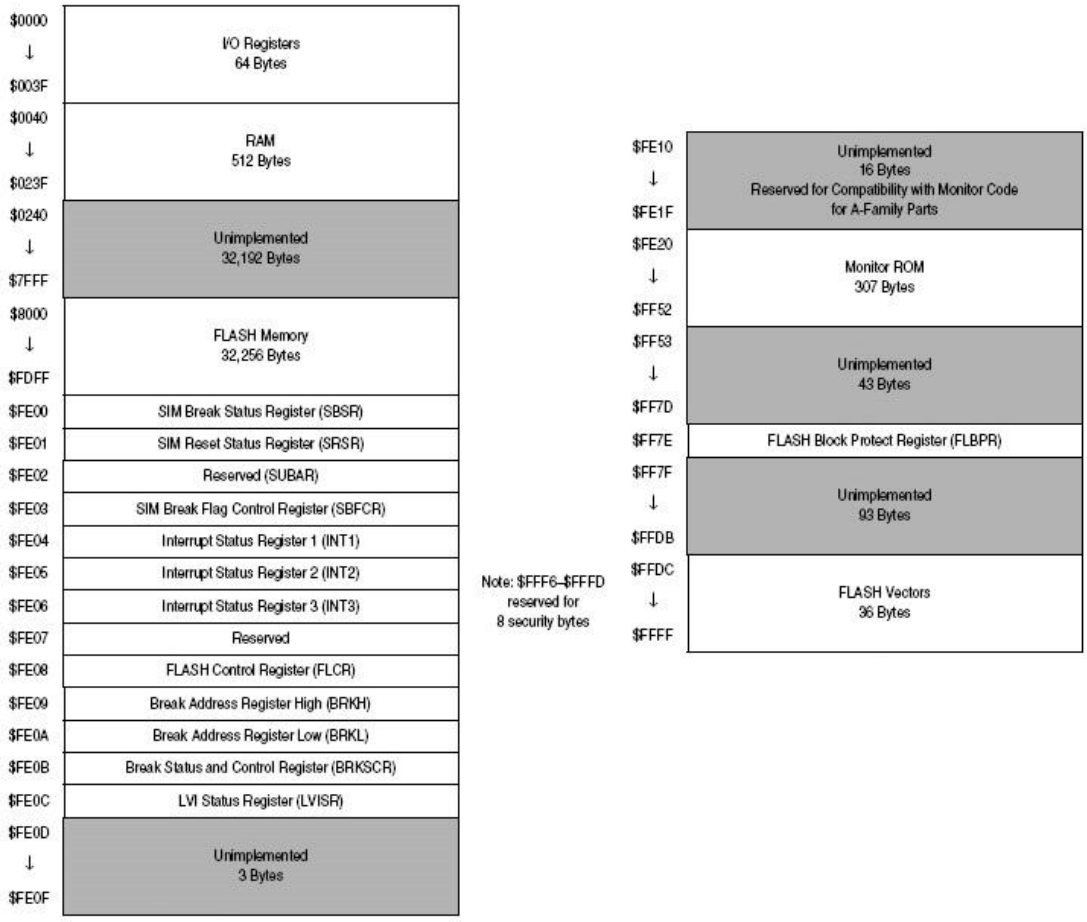


Slika 10: Zgradba MC908GP32 (blokovna shema)
(vir: MC68HC908GP32 Data Sheet, Rev. 10 1/2008, Freescale Semiconductor, Inc., (pdf datoteka proizvajalca))

Na desni strani so prikazani vsi vhodno-izhodni priključki. Vidimo lahko, da imajo priključki večinoma dvojno funkcijo, saj si jih porti delijo z drugimi moduli.

Naslovni prostor

MC908GP32 je zgrajen na osnovi Von Neumannovega modela in ima torej enoten naslovni prostor, v katerem so vsi registri portov in ostalih modulov mikrokontrolerja ter pomnilniki RAM, ROM in *Flash* EEPROM. Vsak od registrov in lokacije v pomnilnikih so dosegljivi na enak način z določitvijo naslova želene lokacije. Razporeditev pomnilnikov in ostalih enot mikrokontrolerja v naslovnem prostoru prikazuje spodnja slika:



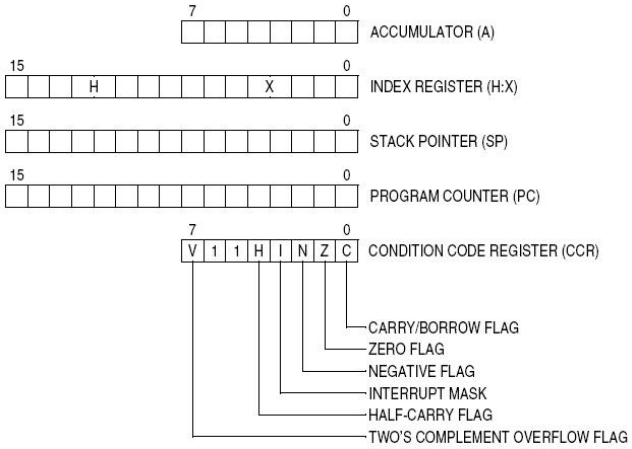
Slika 11: Naslovni prostor mikrokontrolerja
(vir: MC68HC908GP32 Data Sheet, Rev. 10 1/2008, Freescale Semiconductor, Inc., (pdf datoteka proizvajalca))

S sivo barvo so označena območja, ki niso uporabljena.

Registri centralno-procesne enote

Centralno-procesna enota mikrokontrolerja ima poleg aritmetično-logične enote še registre in kontrolno logiko. Z ukazi dostopamo in uporabljamo naslednje registre:

- akumulator (*Accumulator – A*),
- indeksni register (*Index Register – H:X*),
- kazalec sklada (*Stack Pointer – SP*),
- programski števec (*Program Counter – PC*),
- register stanj (*Condition Code Register – CCR*).



Slika 12: Registri centralno-procesne enote
(vir: MC68HC908GP32 Data Sheet, Rev. 10 1/2008, Freescale Semiconductor, Inc., (pdf datoteka proizvajalca))

Akumulator

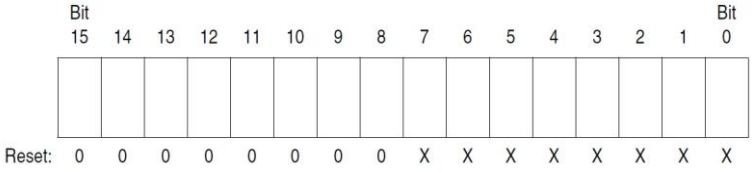
Je 8-bitni register, ki se uporablja za shranjevanje podatkov, izvajanje operacij nad podatki ter za shranjevanje rezultatov po izvedenih operacijah aritmetično-logične enote. Resetiranje mikrokontrolerja ne vpliva na njegovo vsebino.

Slika 13: Akumulator

Indeksni register

Je 16-bitni register in je razdeljen na višji del, označen s H, ter nižji del, označen z X. Uporabljamo ga pri indeksnem načinu naslavljanja. Z njim lahko tudi shranjujemo podatke ter

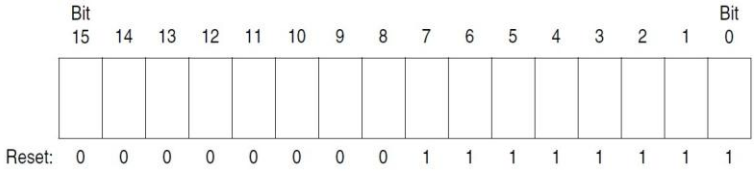
opravljamo preproste matematične operacije. Po resetiranju mikrokontrolerja se vsi biti v H delu postavijo na vrednost 0, na X del pa resetiranje ne vpliva.



Slika 14: Indeksni register

Kazalec sklada

To je 16-bitni register, ki vsebuje začetni naslov spominskih lokacij skladovnega pomnilnika. Uporabljamo ga lahko tudi za drugo vrsto indeksnega naslavljanja. Po resetiranju mikrokontrolerja je vrednost v njem $00FF.



Slika 15: Kazalec sklada

Programski števec

Ta 16-bitni register vsebuje naslov naslednjega ukaza, ki ga mora aritmetično-logična enota izvesti. Po resetiranju mikrokontrolerja se vanj naložita vrednosti z naslovov $FFFE in $FFFF (Reset vektor). Na ta naslova moramo shraniti naslov prvega ukaza programa, ki ga želimo z mikrokontrolerjem izvajati.

Slika 16: Programski števec

Register stanj

Večina bitov v njem opisuje lastnosti rezultata zadnje izvršene operacije. Po vsakem izvršenem ukazu se ti biti postavijo v stanja, ki opisujejo, kakšen je rezultat. Vendar pa ukazi ne vplivajo vedno na vse bite registra stanj. Kako določen ukaz vpliva na posamezen bit, je razvidno v tabelah ukazov, ki se nahajajo v prilogi na koncu gradiva.
Po resetiranju mikrokontrolerja se bit I postavi na 1, ostali biti pa ostanejo nespremenjeni.

	Bit 7	6	5	4	3	2	1	Bit 0
	V	1	1	H	I	N	Z	C
Reset:	X	1	1	X	1	X	X	X

Slika 17: Register stanj

Pomen posameznih bitov registra stanj:
– C (*carry/borrow*): Postavi se na logično vrednost 1, ko nastopi v rezultatu zadnje operacije prekoračitev 8-bitne vrednosti.
– V (*overflow*): Postavi se na logično vrednost 1, ko nastopi v rezultatu zadnje operacije prekoračitev vrednosti +127 ali -128.
– Z (*zero*): Postavi se na logično vrednost 1, ko je rezultat zadnje operacije nič.
– N *(negative)*: Postavi se na logično vrednost 1, ko je rezultat zadnje operacije negativno število. V dvojiškem komplementu je z osmimi biti negativno število predstavljeno s sedmimi biti ter osmim (najvišjim) na logični vrednosti 1. Torej je N bit registra stanj enak najvišjemu bitu rezultata zadnje izvršene operacije.
– I *(interupt mask)*: Če je postavljen na logično vrednost 1, preprečuje izvajanje prekinitev. Postavi se na logično vrednost 1, ko se začne izvajati prekinitveni. Z ukazoma SEI in CLI lahko med izvajanjem programa postavljamo ali brišemo I bit in s tem dovolimo ali preprečimo izvajanje prekinitvenih programov.
– H *(half-carry)*: Postavi se na logično vrednost 1, ko nastopi v rezultatu zadnje operacije prenos iz nižjih štirih na višje štiri bite.

Ponovimo

Spoznali smo osnovne lastnosti mikrokontrolerja MC908GP32, njegovo zgradbo, razporeditev enot v naslovnem prostoru mikrokontrolerja ter registre njegove centralno-procesne enote. Te registre uporabljamo pri programiranju, saj se veliko ukazov nanaša na izvajanje operacij v njih.

Med temi registri je tudi register stanj, ki z vrednostjo posameznih bitov opisuje po vsakem izvedenem ukazu lastnosti rezultata, ki je nastal.

Vprašanja

1. Katere so glavne značilnosti mikrokontrolerja MC908GP32?
2. Na katerem naslovu se začne pomnilnik RAM mikrokontrolerja?
3. Na katerem naslovu se začne *Flash* pomnilnik mikrokontrolerja?
4. Katere registre ima centralno-procesna enota predstavljenega mikrokontrolerja in kaj je zanje značilno?
5. Razloži pomen bitov v registru stanj.
6. Kaj se dogaja s programskim števcem po resetiranju mikrokontrolerja?

PROGRAMIRANJE MC908GP32

V tem poglavju si bomo podrobneje pogledali programiranje v zbirnem jeziku. Čeprav so si ukazi zbirnega jezika različnih proizvajalcev med seboj podobni, ima vsaka družina mikrokontrolerjev zaradi različne zgradbe, predvsem različnih registrov centralno-procesne enote, tudi različne ukaze v zbirnem jeziku. Ukazi se med seboj razlikujejo tudi po dolžini in sestavi. Na to vpliva zasnova mikrokontrolerja.

Načini naslavljanja

Ukazu (mnemonični kodi) ponavadi sledi operand, ki je lahko neka vrednost ali naslov, kjer se vrednost nahaja ali kamor jo je potrebno shraniti. Zaradi tega lahko isti ukaz z operandom različne vrste opravi operacijo s popolnoma drugačnim podatkom, ki ga dobi na čisto drugem naslovu. Z načinom naslavljanja je določeno, kako je ukaz uporabljen. To določa, kaj predstavlja operand, kje se nahaja vrednost, ki jo ukaz uporablja, ter kako dolg bo ukaz.

Naš mikrokontroler pozna šest osnovnih načinov naslavljanja:

- vsebujoče (*inherent* oz. *implied*)
- takojšnje (*immediate*)
- neposredno ali direktno (*direct*)
- razširjeno (*extended*)
- relativno (*relative*)
- indeksno (*indexed*)

Nekateri sestavljeni ukazi lahko uporabljajo dve različni kombinaciji naslavljanj. Vse ukaze s kratko razlago uporabe in delovanja ter njihove operacijske kode in druge podatke dobimo v tabelah v prilogi na koncu tega gradiva.

Vsebujoče

Ukaz je dolg 1 bajt in ga tvori le mnemonična koda oziroma njena operacijska koda, ki nosi v sebi vse potrebne informacije o operandu. Operandi so registri centralno-procesne enote, njihovo ime pa je vsebovano v operacijski kodi.

Primeri ukazov:

`CLRA` – Ukaz postavi vse bite v akumulatorju na log. vrednost 0.

`CLRX` – Ukaz postavi vse bite v X delu indeksnega registra na log. vrednost 0.

`COMA` – Ukaz negira vrednost v akumulatorju.

`ASLA` – Ukaz pomakne vse bite v akumulatorju za eno mesto v levo.

`DECX` – Ukaz zmanjša za 1 vrednost X dela indeksnega registra.

`INCA` – Ukaz poveča za 1 vrednost akumulatorja.

Takojšnje

Pri tej vrsti naslavljanja mnemonični oz. operacijski kodi sledi vrednost (konstanta). Dolžina ukaza je odvisna od uporabljenega registra. Če je to akumulator ali katerakoli lokacija v pomnilniku, je dolžina ukaza 2 bajta (prvi je operacijska koda, drugi pa vrednost). Pri uporabi indeksnega registra ali kazalca sklada je vrednost dolga 2 bajta, zato je dolžina ukaza 3 bajte.

Primeri ukazov:

`LDA #$10` – Ukaz naloži v akumulator šestnajstiško vrednost 10 oziroma desetiško 16. Znak # pred operandom pove, da je število za ukazom konstanta, ki jo ukaz uporabi.

Znak $ uporabimo pred številom, ki ga zapišemo v šestnajstiški obliki.

`ADD #!23` – Ukaz prišteje trenutni vrednosti v akumulatorju desetiško vrednost 23, vsota se ohrani v akumulatorju.

Znak ! uporabimo pred številom, ki ga zapišemo v desetiški obliki.

`CMP #$AE` – Ukaz primerja vrednost v akumulatorju s šestnajstiško vrednostjo AE. Primerjava se opravi z odštevanjem vrednosti operanda od vrednosti akumulatorja. Rezultat se ne ohrani v akumulatorju, le biti v registru stanj z ustreznimi stanji opišejo njegove lastnosti.

`ORA #%10011100` – Ukaz opravi logično operacijo ALI (*OR*) med vrednostjo v akumulatorju in dvojiško vrednostjo 10011100. Vse logične operacije se opravijo med istoležnimi biti vrednosti akumulatorja in operanda. Rezultat se ohrani v akumulatorju.

Znak % uporabimo pred številom, ki ga zapišemo v dvojiški obliki.

Neposredno (direktno)

Ukaz za neposredno (direktno) naslavljanje je dolg 2 bajta. V prvem je mnemonična oz. operacijska koda ukaza, v drugem pa naslov, kjer se bo ukaz izvajal. Za naslov je uporabljen le nižji bajt naslovnega vodila mikrokrokontrolerja. Z enim bajtom lahko naslovimo le prvih 256 spominskih lokacij (od $0000 do $00FF). Prednost takega načina naslavljanja je večja hitrost izvajanja ukazov ter manj porabljenega pomnilnika za posamezen ukaz. Zato ta način

naslavljanja uporabljamo, če je le mogoče. Tako pri uporabi pomnilnika RAM, ki se nahaja na naslovih od $0040 do $023F, uporabljamo raje naslove do $00FF.

Primeri ukazov:

`LDA $10` – Ukaz naloži v akumulator vrednost z naslova $10.

Pri ukazih, kjer je operand nek naslov, ni znaka # pred operandom.

`ADD !23` – Ukaz prišteje trenutni vrednosti v akumulatorju vrednost, ki se nahaja na naslovu 23 oziroma $17, vsota se ohrani v akumulatorju.

`CPX $0F` – Ukaz primerja vrednost v X delu indeksnega registra z vrednostjo, ki se nahaja na naslovu $0F. Rezultat se ne ohrani v X registru, le biti v registru stanj z ustreznimi stanji opišejo njegove lastnosti.

`AND %10011100` – Ukaz opravi logično operacijo IN (*AND*) med vrednostjo v akumulatorju in vrednostjo, ki se nahaja na naslovu %10011100 oziroma $9C. Rezultat se ohrani v akumulatorju.

`STA $50` – Ukaz shrani vrednost akumulatorja na naslov $50.

Razširjeno

Ukaz je dolg 3 bajte. Prvi bajt zaseda mnemonična oz. njena operacijska koda, drugi in tretji pa sta naslov, kjer se bo ukaz izvajal. S tem načinom naslavljanja imamo dostop do vseh spominskih lokacij, ki nam jih omogoča 16-bitno naslovno vodilo našega sistema (od $0000 do $FFFF).

Primeri ukazov:

`LDA $0010` – Ukaz, ki smo ga prej uporabili z neposrednim naslavljanjem, lahko napišemo tudi z razširjenim naslavljanjem. Naredil bo popolnoma isto operacijo, le naslov zapišemo na daljši način.

`STHX $0123` – Ukaz shrani vrednost celotnega indeksnega registra (H in X del) na naslov $0123. Ker je celoten indeksni register 16-bitni oziroma 2-bajtni, se njegova vsebina ne more shraniti samo na podani naslov, ampak se shranjevanje izvede na dveh zaporednih naslovih, podanem in naslednjem. Tako se vrednost višjega bajta indeksnega registra (H del) shrani na naslov $0123, vrednost nižjega bajta (X del) pa na naslov $0124.

`SUB $FFC0` – Ukaz odšteje vrednost, ki se nahaja na naslovu $FFC0 od vrednosti akumulatorja. Rezultat se ohrani v akumulatorju.

Relativno

Ta način naslavljanja uporabljamo pri vejitvenih ukazih (*Branch* ukazi), pri katerih preverjamo določen pogoj (največkrat postavitev bitov v registru stanj). Če je pogoj izpolnjen, se vejitev opravi in program se nadaljuje v vrstici, ki jo določimo z oznako vrstice oz. labelo. Če pogoj ni izpolnjen, se program nadaljuje z naslednjim ukazom po ukazu z relativnim naslavljanjem.
Vejitveni ukazi nam omogočajo preskoke po programu naprej ali nazaj. Tako lahko realiziramo zanke, s katerimi ponavljamo ukaze. S preskokom naprej po programu lahko preskočimo ukaze, ki se zaradi tega ne izvedejo. Dolžina skoka je omejena na manj kot 130 naslovov naprej ali nazaj po programu. Z oznako vrstice (labelo) določimo, kje se bo program nadaljeval v primeru izpolnjenega pogoja za vejitev.

Vejitveni ukaz najprej preveri, če je izpolnjen pogoj za vejitev. Preverjajo se lahko vrednosti posameznih bitov v registru stanj ali kombinacije več bitov tega registra. Na ta način lahko opravimo nek preskok v programu glede na lastnosti rezultata nekega ukaza. Zato je pomembno, da izberemo ustrezen vejitveni ukaz in ga postavimo na mesto, kjer hočemo preveriti nastali rezultat. Ponavadi postavljamo vejitvene ukaze za ukazi primerjanja, povečevanja ali zmanjševanja vrednosti.

Primer ukaza:

```
S1    DECA
      BNE     S1
```

– Ukaz `BNE` preveri stanje Z bita v registru stanj. Če je ta bit na logični vrednosti 0, se izvrši vejitev oziroma preskok in nadaljevanje programa v vrstici, ki ima oznako S1. V tem primeru se program nadaljuje z ukazom `DECA`, ki bi zmanjševal vrednost akumulatorja, dokler le-ta ne bi dosegla 0. Ker bi se v tem primeru Z bit postavil na logično vrednost 1, bi se program nadaljeval z naslednjim ukazom v programu, ki bi sledil ukazu `BNE`.

Indeksno

Uporablja se, ko hočemo z istim ukazom doseči več zaporednih naslovov v pomnilniku. Ukaz je sestavljen iz mnemonične oz. njene operacijske kode in relativnega naslova. Naslov, kjer se ukaz izvede, mikrokontroler izračuna s seštevanjem relativnega naslova in vsebine indeksnega registra. Relativni naslov je lahko samo pozitivno 8- ali 16-bitno število.

Če hočemo z istim ukazom doseči več zaporednih naslovov, moramo izvajanje programa postaviti v zanko, v kateri je poleg ukaza z indeksnim naslavljanjem tudi ukaz, ki spreminja vrednost indeksnega registra. Tako lahko uporabimo ukaz `INCX`, ki vsakokrat za ena poveča vrednost indeksnega registra. Če uporabimo ukaz `DECX`, vsakokrat za ena zmanjšamo vrednost indeksnega registra. Povečevanje ali zmanjševanje te vrednosti nam omogoča dostop do višjih ali nižjih naslovov od začetnega. Indeksno naslavljanje lahko uporabljamo tudi s kazalcem sklada.

Primer ukaza:

`LDHX  #$1234` – S tem ukazom v indeksni register naložimo vrednost $1234.

`LDA  $10,X` – Ta ukaz uporablja indeksno naslavljanje. To vidimo iz zapisa ukaza, ki ima za operandom še črko X, z vejico ločeno od operanda. Ukaz `LDA` naloži v akumulator neko vrednost. V tem primeru vrednost dobi na naslovu, ki je določen z vsoto vrednosti indeksnega registra ($1234) in operanda ukaza ($10 – relativni naslov). Tako dobimo naslov $1244, s katerega ukaz naloži vrednost v akumulator.

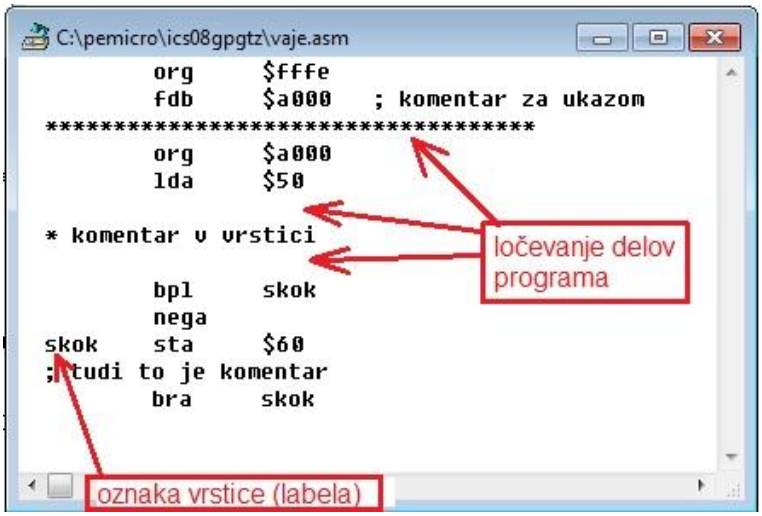
Pisanje in prevajanje programov

Tudi pri pisanju programov v zbirnem jeziku se moramo držati nekaj preprostih pravil. Ker programe prevajamo s programskim orodjem (prevajalnikom), moramo upoštevati zahteve, ki jih ima uporabljeni prevajalnik. V dokumentaciji vsakega prevajalnika so vsa pravila, ukazi in postopki vedno razloženi. Večina teh je splošnih in niso odvisni od posameznega prevajalnika, seveda pa so lahko manjša odstopanja, ki jih pri uporabi nekega prevajalnika moramo spoznati. Mi si bomo pogledali le nekaj osnovnih pravil in ukazov, ki nam bodo omogočili in olajšali delo.

Program vedno pišemo v preprostem urejevalniku besedil, ki poleg napisanega ne dodaja nikakršnih kod. Tako tudi teksta (programa) ne posebej oblikujemo in urejamo. V razvojnem okolju je navadno že vključen tak urejevalnik, lahko pa uporabimo tudi urejevalnik *Beležnica*, ki je del operacijskega sistema *Windows*.

Program v zbirnem jeziku pišemo v štirih stolpcih. V prvem je oznaka vrstice (labela), če le-ta obstaja, v drugem je ukaz in v tretjem je operand (razen pri vsebujočem naslavljanju, ko operanda ni). Četrti stolpec lahko uporabimo za komentar, ker ga prevajalnik ne upošteva. Stolpci so med seboj ločeni vsaj z enim presledkom, ponavadi uporabimo kar zamik s tabulatorjem (tipka Tab). Pišemo lahko z velikimi ali malimi črkami.

Komentarje si ob ukazih in nasploh v programu pišemo, da vedno lahko vemo, kaj posamezen del programa dela. To nam olajša popravljanje, spreminjanje in dodajanje programa tudi, če je od njegovega ustvarjanja minilo že več časa. Komentarje lahko pišemo tudi v vrsticah, ki nimajo ukazov. Če je cela vrstica komentar, mora biti prvi znak v vrstici zvezdica ali podpičje. Če komentar pišemo za ukazi (4. stolpec), pred besedilo postavimo podpičje. Posamezne dele programa med seboj ločimo s praznimi vrsticami ali s črtami, ki jih naredimo z znaki, vendar mora v tem primeru biti prvi znak obvezno zvezdica ali podpičje. Ločevanje delov programa nam izboljša preglednost, s tem pa tudi iskanje in odpravljanje napak, dopolnjevanje programov in drugo. Na spodnji sliki je primer programa s prikazanimi možnostmi pisanja komentarjev:



Slika 18: Pisanje programov s komentarji (primer)

Pomemben del pri pisanju programov predstavljajo tudi psevdoukazi. To so ukazi, ki so namenjeni prevajalniku. Ti ukazi se ne prevajajo, vendar jih prevajalnik upošteva. Na spodnjih primerih bomo spoznali nekaj teh ukazov in njihovo uporabo:

`ORG $9000` – Ta ukaz prevajalniku pove, da ukaze in podatke, ki mu sledijo, postavi na naslove od $9000 naprej. S tem ukazom določimo, kje v pomnilniku bomo imeli program, kam bomo dali npr. tabelo podatkov, z njim vpisujemo reset vektor in ostale vektorje na predpisana mesta in podobno.

`DB !1,$F2,%10011100` – S tem ukazom na zaporedne lokacije od trenutno določene shranimo konstantne enobajtne vrednosti, ki sledijo ukazu. Te vrednosti med seboj ločimo z vejicami, pišemo pa jih v ustreznem številskem formatu. Namesto ukaza `DB` lahko uporabimo tudi ukaz `FCB`. Če hočemo pretvoriti znake v ASCII kode, damo te znake med navednice:

`DB "BESEDILO"` – Znaki se bodo shranili s svojo ASCII kodo na zaporedne naslove.

`FDB $8000,!6,$4A` – Podobno kot prejšnji ukaz tudi ta tvori konstantne vrednosti, ki se shranijo na zaporedne naslove. Vendar se vsako število pretvori v 2-bajtno vrednost in se seveda tudi shrani na dva zaporedna naslova.

`PTA EQU $0000` – Ukaz `EQU` priredi oznaki PTA vrednost $0000. Tako lahko pri programiranju namesto števil uporabljamo oznake. Le-te izberemo tako, da npr. iz imena vemo, kateri register uporabljamo v ukazu. V primeru je oznaka PTA ime registra, ki se nahaja na naslovu $0000. Tako lahko namesto pisanja npr. ukaza

```
LDA   $0000
```

napišemo ukaz

```
LDA   PTA.
```

Prevajalnik bo med prevajanjem upošteval napisano prireditev vrednosti in za operacijsko kodo zapisal pravilni naslov.

`$INCLUDE "program1.asm"` – S tem ukazom na mestu uporabe vključimo del programa iz druge datoteke (v tem primeru datoteke z program1.asm) v program, ki ga trenutno pišemo. To nam omogoča, da daljše programe razdelimo na več manjših delov, kar nam poveča preglednost programov, hkrati pa možnost, da posamezne dele programa uporabljamo tudi pri pisanju drugih programov. Posamezne datoteke, ki so del enega programa, imamo ponavadi v isti mapi. V nasprotnem primeru moramo pred imenom datoteke, ki jo vključujemo v program, podati še pot do nje.

Če upoštevamo pravila pisanja programov in ne naredimo nobene druge napake med tipkanjem, nam prevajalnik uspešno prevede program in shrani v datoteko z enakim imenom, kot ga ima izvorna datoteka, le da ima ta končnico .hex. V tej datoteki je strojna koda programa, ki jo lahko uporabimo v simulaciji delovanja programa ali jo naložimo v *Flash* EEPROM pomnilnik mikrokontrolerja in preizkušamo njegovo delovanje v realnih razmerah.

Če prevajalnik naleti na odstopanje od pravil ali napačno zapisan ukaz, programa ne prevede. Označi nam vrstico, v kateri je po njegovem napaka:

Slika 19: Napaka v programu (oznaka prevajalnika)

V zgornjem primeru je označena vrstica s komentarjem, ker manjka prvi znak v vrstici, ki določa, da je to komentar (* ali ;). Ko napako v programu odpravimo, ponovno zaženemo prevajanje, ki se tudi uspešno zaključi.

Spoznali smo zbirni jezik ter način pisanja programov v tem jeziku. Kako pa programe lahko preizkušamo? Kako jih prenesemo v pomnilnik mikrokontrolerja? Kako lahko vidimo dogajanja v registrih mikrokontrolerja? Na ta in druga vprašanja bomo dobili odgovor v naslednjem poglavju.

Ponovimo

V tem poglavju smo spoznali programiranje v zbirnem jeziku. Ukazi v tem jeziku so preprosti. Mnemonične kode so okrajšave ali kratice angleških izrazov za operacije, ki jih ukazi izvedejo. Ker mora vsak ukaz poleg vrste operacije vsebovati še informacijo o tem, kateri podatek naj mikrokontroler uporabi ali kje naj podatek dobi, ločimo različne načine uporabe ukazov – načine naslavljanja. Poznamo vsebujoče, takojšnje, neposredno, razširjeno, relativno in indeksno naslavljanje. Vsako od njih ima svoj način zapisa. Tudi števila lahko zapisujemo na različne načine oz. v različnih številskih sestavih. Najbolj uporabni so desetiški, dvojiški in šestnajstiški.

Vsa pravila za pisanje programov v zbirnem jeziku so pomembna zato, da nam prevajalnik pravilno prevede program v strojno kodo (binarno obliko). Zato uporabljamo tudi psevdoukaze, ki se ne prevedejo. Namenjeni so prevajalniku, z njimi pa si olajšamo programiranje, postavljanje programov in podatkov na ustrezna mesta v pomnilnik in podobno.

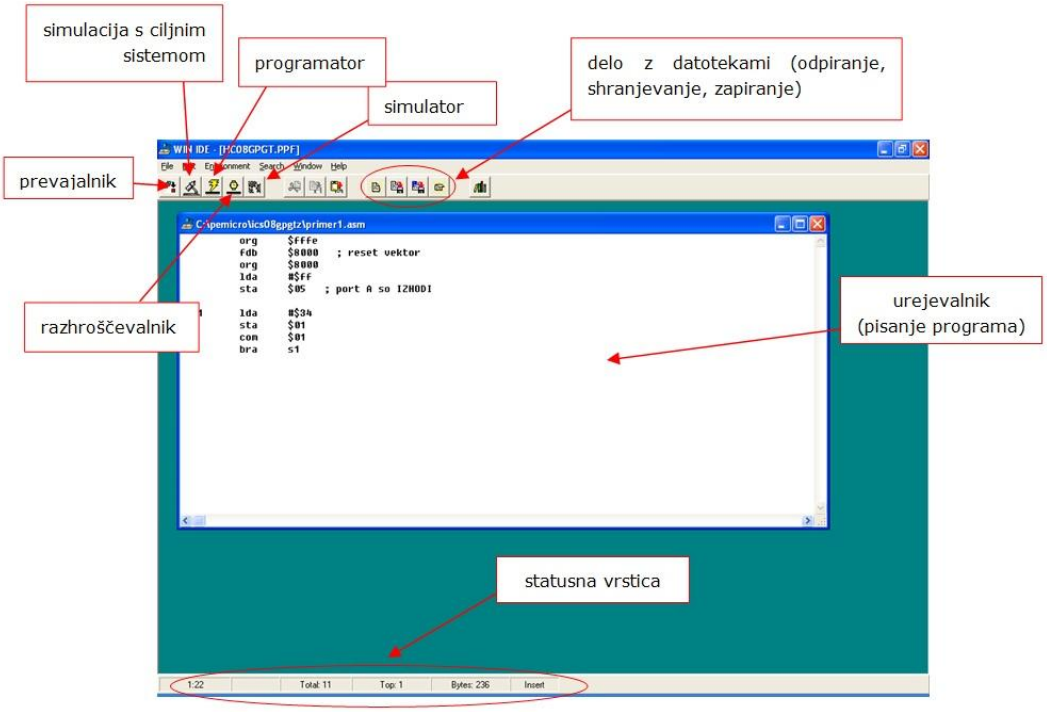
Vprašanja

1. Kaj je značilno za zbirni jezik?
2. Katere osnovne načine naslavljanja poznamo pri MC908GP32?
3. Podaj nekaj primerov ukazov, razloži njihov pomen in uporabljeno naslavljanje.
4. Kako lahko pišemo komentarje v programih, pisanih v zbirnem jeziku?
5. Kaj so to psevdoukazi?
6. Kako poskrbimo, da se bo programska koda v *Flash* pomnilniku shranila od naslova $A000 dalje?
7. Kaj nam omogočajo ukazi z relativnim naslavljanjem?
8. Kako uporabimo ukaz z indeksnim naslavljanjem? Podaj primer.

RAZVOJNO OKOLJE WINIDE

Razvojno okolje nam omogoča pisanje programov, prevajanje, simuliranje, preverjanje ter vnašanje programov v pomnilnik mikrokontrolerja. Na tržišču obstaja več proizvajalcev, ki nam ponujajo brezplačna ali plačljiva razvojna okolja za mikrokontrolerje. Za MC908GP32 dobimo na spletni strani podjetja *P&E Microcomputer Systems, Inc.* iz Združenih držav Amerike (http://www.pemicro.com/support/download_processor.cfm?family=1) brezplačno razvojno okolje *WinIDE Development Environment*.

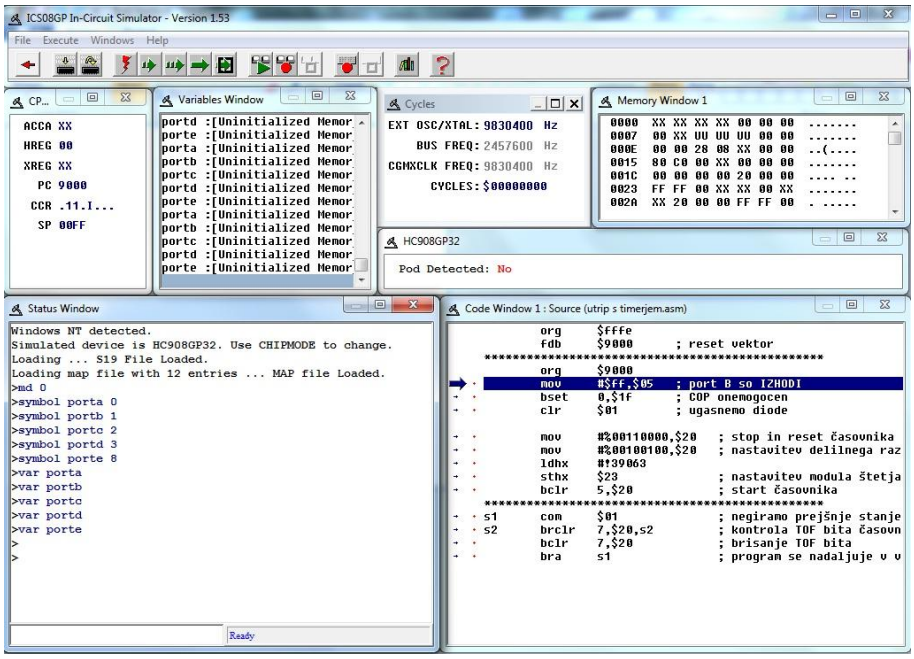
Ko zaženemo programski paket *WinIDE*, se nam odpre delovno okolje, kot ga vidimo na spodnji sliki:



Slika 20: Razvojno okolje WINIDE

Na voljo imamo preprost urejevalnik besedil, ki nam omogoča pisanje in shranjevanje programov v zbirnem jeziku; s preprostim pritiskom na ukazno ikono za prevajanje program tudi prevedemo. Datoteke s programom v zbirnem jeziku imajo končnico *.asm, preveden program pa se samodejno shrani v datoteko z enakim imenom s končnico *.hex.

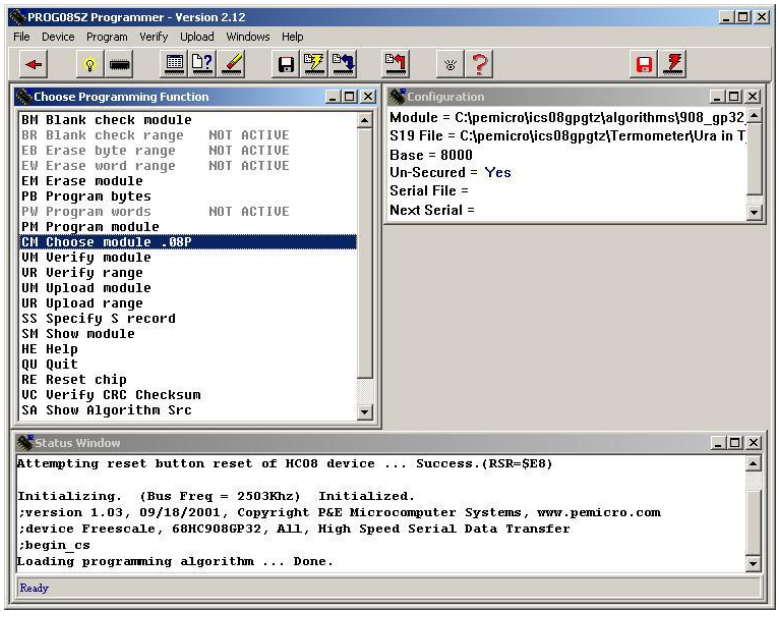
Zatem lahko program preizkušamo na simulatorju (*Simulation*), kjer ne potrebujemo vezja z mikrokontrolerjem. Simulacijo izvajamo tudi s ciljnim sistemom, torej z vezjem, ki deluje z MC908GP32 (*In-Circuit Simulator*). Če hočemo program vnesti v *Flash* pomnilnik mikrokontrolerja, poženemo programsko orodje *Programmer*. Pri iskanju in odpravljanju napak nam je v pomoč razhroščevalnik (*In-Circuit Debugger*). Na sliki vidimo odprto programsko orodje za simulacijo, ki je enako za izvajanje simulacij samo na računalniku in za simuliranje s pomočjo ciljnega sistema z mikrokontrolerjem:



Slika 21: Programsko okolje simulatorjev *Simulator* in *In-Circuit Simulator*

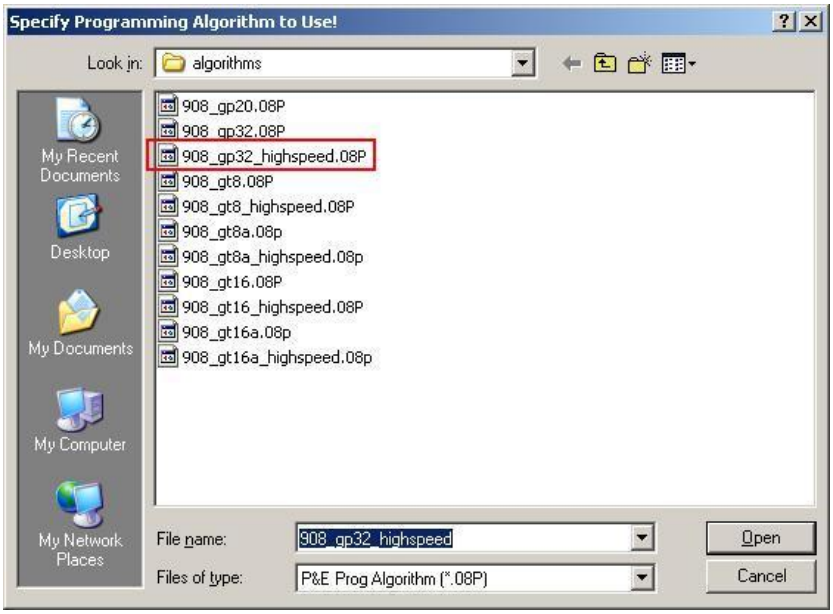
Omogoča nam izvajanje programa po korakih (ukazih) ali v celoti oziroma do prekinitve izvajanja. Med izvajanjem lahko spremljamo stanja registrov mikrokontrolerja ter vsebino pomnilnikov. V program lahko postavimo tudi zaustavitveno točko (*breakpoint*), ki zaustavi izvajanje programa.

Programsko orodje za vnos programa v *Flash* pomnilnik mikrokontrolerja je na spodnji sliki:



Slika 22: Programsko okolje orodja *Programmer*

Z njim lahko tudi zbrišemo ali preberemo vsebino *Flash* pomnilnika in shranimo prebrani program v datoteko. Preden se nam odpre to orodje, moramo določiti algoritem brisanja in vpisovanja podatkov v *Flash* pomnilnik mikrokontrolerja:



Slika 23: Izbira algoritma *Flash* pomnilnika

Za MC908GP32 je ta shranjen v datoteki, ki je na sliki v rdečem pravokotniku.

Uporaba programskega paketa je preprosta, saj so vsi glavni ukazi za delo s posameznimi deli razvojnega okolja jasni in se nahajajo v vrstici z ikonami. Postopki simuliranja in nalaganja programa v pomnilnik mikrokontrolerja pa se izvedejo v nekaj korakih. Poskrbeti moramo le, da lahko računalnik in mikrokontroler začneta komunicirati. Zato moramo mikrokontroler postaviti v poseben način delovanja, imenovan *Monitor Mode*. To dosežemo s postavitvijo zahtevanih električnih pogojev (logičnih stanj in ustrezne napetosti na enem priključku) na določene priključke mikrokontrolerja. V nadaljevanju si bomo pogledali, kaj je potrebno storiti.

Komunikacija med računalnikom in mikrokrmilnikom

Ob priklopu mikrokontrolerja na napajalno napetost ali po resetiranju le-ta najprej preveri stanja na priključkih, ki so pomembni za prehod v način komunikacije z osebnim računalnikom. Če so pogoji za to izpolnjeni, se začne izvajati program iz pomnilnika ROM, ki začne komunicirati z razvojnim okoljem na računalniku. Komunikacija poteka preko serijskega porta (RS232). Če ga na računalniku nimamo, lahko uporabimo USB port z ustreznim pretvornikom.

Mikrokontroler komunicira z računalnikom z enožilno povezavo na priključku PTA0. Pogoji, ki jih moramo zagotoviti, da pride do začetka komuniciranja, so naslednji:

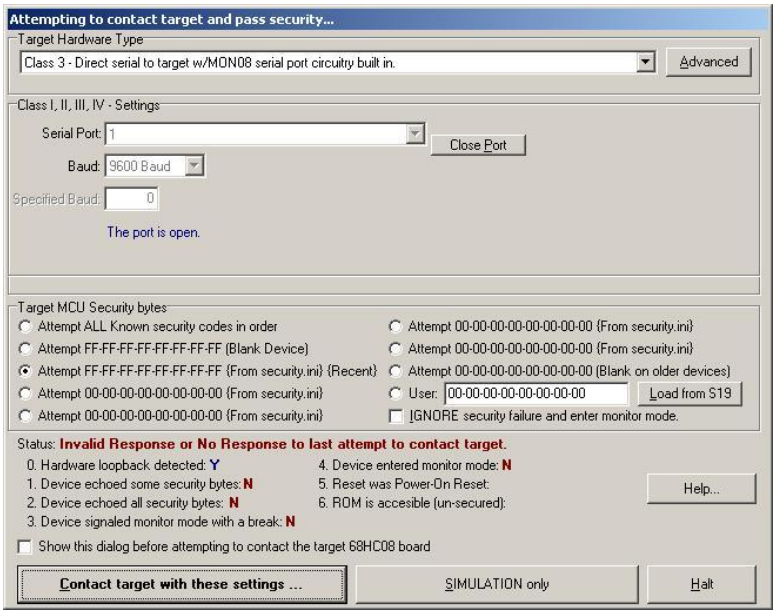
- na priključku PTA7 mora biti logična vrednost 0,
- na priključku PTC0 mora biti logična vrednost 1,

- na priključku PTC1 mora biti logična vrednost 0,
- na priključku PTC3 mora biti logična vrednost1,
- na priključku IRQ mora biti napetost med 7,5 V in 9 V.

Zato se nam ob uporabi programskih orodij *In-Circuit Simulator* in *Programmer* na začetku odpre okno, ki od nas zahteva resetiranje mikrokontrolerja:

Slika 24: Navodilo med vzpostavljanjem komunikacije (resetiranje)

Če kateri od zahtevanih pogojev manjka, se nam odpre okno, kjer lahko preverimo in spremenimo nastavitve serijske komunikacije:



Slika 25: Nastavitve serijske komunikacije

Ko mikrokontrolerja ne povezujemo z razvojnim okoljem, zahtevanih pogojev po resetiranju ali vklopu napajalne napetosti ni na njegovih priključkih, zato začne takoj izvajati program, ki je shranjen v *Flash* pomnilniku.

Ponovimo

Razvojno okolje je programsko orodje, ki nam omogoča programiranje in preizkušanje programov. Sestavlja ga več posameznih orodij, s katerimi pišemo in prevajamo programe, simuliramo njihovo delovanje ter jih vnašamo v *Flash* pomnilnik mikrokontrolerja. Če hočemo vzpostaviti komunikacijo med mikrokontrolerjem in računalnikom z razvojnim okoljem, moramo mikrokontroler postaviti v poseben način delovanja. To dosežemo s postavitvijo ustreznih stanj na nekatere njegove priključke. Če so ta stanja postavljena ob priključitvi mikrokontrolerja na napajalno napetost ali po resetiranju, bo komunikacija med računalnikom in mikrokontrolerjem vzpostavljena, kar nam omogoča uporabo vseh orodij in možnosti razvojnega okolja.

Vaje

1. Napiši program v zbirnem jeziku, ki bo na naslov $50 v pomnilniku RAM shranil vrednost 33. Preizkusi ga s pomočjo simulatorja.
2. Napiši program v zbirnem jeziku, ki bo z naslova $50 naložil v akumulator. To vrednost naj poveča za 5 in shrani na naslov $51. Program preizkusi s simulatorjem.
3. Napiši program v zbirnem jeziku, ki bo izračunal enačbo y = 2x + 3. Spremenljivka x se nahaja na naslovu $40, rezultat pa naj shrani na naslov $50.
4. Napiši program v zbirnem jeziku, ki bo izračunal enačbo y = |x|. Spremenljivka x se nahaja na naslovu $60, rezultat pa naj shrani na naslov $65.
5. Napiši program v zbirnem jeziku, ki bo na naslove od $40 do $4F vpisal vrednost 6. Za shranjevanje uporabi indeksno naslavljanje.

PORTI MIKROKRMILNIKA

MC908GP32 ima 5 portov, preko katerih lahko sprejema ali oddaja binarne vrednosti. Poimenovani so s črkami A, B, C, D in E. Njihova zgradba je podobna, vendar se nekateri med seboj razlikujejo v številu priključkov, ki jih upravljajo, ter v možnosti uporabe vgrajenih pull-up uporov. Vsi priključki so lahko vhodi ali izhodi, funkcijo posameznega priključka pa določimo v smernem registru porta. Tudi uporabo pull-up uporov omogočimo z ustreznim registrom na portu, ki nam to omogoča. Poglejmo si, kako so porti zgrajeni in kako jih pripravimo za uporabo. Nekateri priključki imajo dvojno funkcijo, ker si jih porti delijo z drugimi moduli mikrokontrolerja.

Port A

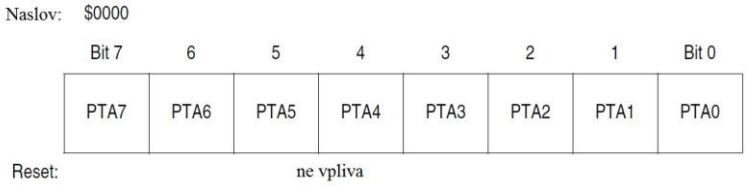
Ta 8-bitni port ima tri registre:

- podatkovni register (*Data Register* – PTA)
- smerni register (*Data Direction Register* – DDRA)
- register za vklop pull-up uporov (*Pullup Enable Register* – PTAPUE)

Zunanje priključke si deli z modulom za tipkovnico (*keyboard interrupt module – KBI*).

Podatkovni register (PTA)

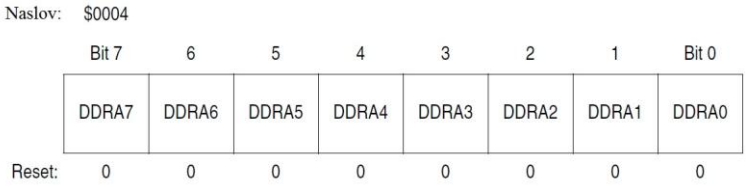
Nahaja se na naslovu $0000. Povezan je z zunanjimi priključki mikrokontrolerja. Če so ti priključki vhodi, vhodne vrednosti preberemo v tem registru. Če so ti priključki izhodi, bodo podatki, ki jih pošljemo v ta register, vidni na zunanjih priključkih mikrokontrolerja. Resetiranje ne vpliva na njegovo vsebino.



Slika 26: Podatkovni register porta A

Smerni register (DDRA)

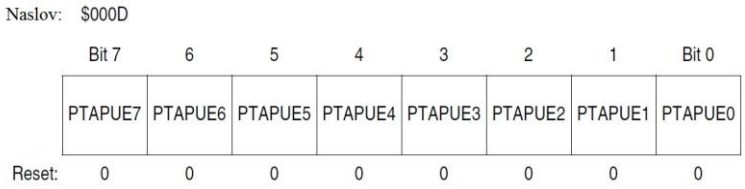
Nahaja se na naslovu $0004. Z njim določamo vrsto priključkov na portu. Z logično 1 v tem registru pripravimo izhode, z logično 0 pa vhode. Določamo lahko vsak priključek posebej. Resetiranje postavi ta register na vrednost 0, torej imamo na začetku določene vse priključke kot vhode.



Slika 27: Smerni register porta A

Register za vklop pull-up uporov (PTAPUE)

Ta register ima naslov $000D. Omogoča nam vklop pull-up uporov, ko so priključki na portu definirani kot vhodi. Z logično 1 upore vklopimo, z logično 0 pa jih izklopimo. Omogočimo lahko vsak upor posebej. Resetiranje postavi ta register na vrednost 0. Če so priključki na portu določeni kot izhodi, pull-up uporov ne moremo uporabiti.



Slika 28: Register za vklop pull-up uporov porta A

Port B

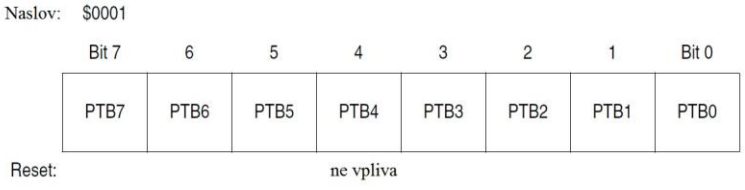
Ta 8-bitni port ima dva registra:

- podatkovni register (*Data Register* – PTB)
- smerni register (*Data Direction Register* – DDRB)

Zunanje priključke si deli z modulom analogno-digitalnega pretvornika (*analog-to-digital converter module – ADC*).

Podatkovni register (PTB)

Nahaja se na naslovu $0001. Povezan je z zunanjimi priključki mikrokontrolerja. Če so ti priključki vhodi, vhodne vrednosti preberemo v tem registru. Če so ti priključki izhodi, bodo podatki, ki jih pošljemo v ta register, vidni na zunanjih priključkih mikrokontrolerja. Resetiranje ne vpliva na njegovo vsebino.



Slika 29: Podatkovni register porta B

Smerni register (DDRB)

Nahaja se na naslovu $0005. Z njim določamo vrsto priključkov na portu. Z logično 1 v tem registru pripravimo izhode, z logično 0 pa vhode. Določamo lahko vsak priključek posebej. Resetiranje postavi ta register na vrednost 0, torej imamo na začetku določene vse priključke kot vhode.

Naslov: $0005

	Bit 7	6	5	4	3	2	1	Bit 0
	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
Reset:	0	0	0	0	0	0	0	0

Slika 30: Smerni register porta B

Port C

Ta port je 7-bitni pri mikrokontrolerju v 44-pinskem ohišju, v 40- in 42-pinskem ohišju pa imamo na razpolago le 5 priključkov (PTC4 do PTC0). Ima tri registre:

- podatkovni register (*Data Register* – PTC)
- smerni register (*Data Direction Register* – DDRC)
- register za vklop pull-up uporov (*Pullup Enable Register* – PTCPUE)

Podatkovni register (PTC)

Nahaja se na naslovu $0002. Povezan je z zunanjimi priključki mikrokontrolerja. Če so ti priključki vhodi, vhodne vrednosti preberemo v tem registru. Če so ti priključki izhodi, bodo podatki, ki jih pošljemo v ta register, vidni na zunanjih priključkih mikrokontrolerja. Resetiranje ne vpliva na njegovo vsebino.

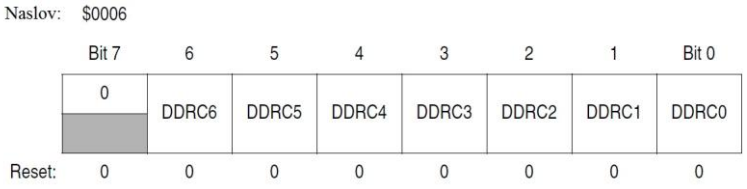
Naslov: $0002

	Bit 7	6	5	4	3	2	1	Bit 0
		PTC6	PTC5	PTC4	PTC3	PTC2	PTC1	PTC0
Reset:	ne vpliva							

Slika 31: Podatkovni register porta C

Smerni register (DDRC)

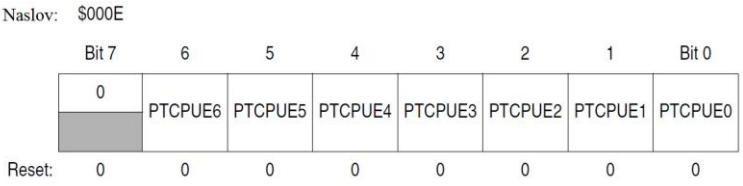
Nahaja se na naslovu $0006. Z njim določamo vrsto priključkov na portu. Z logično 1 v tem registru pripravimo izhode, z logično 0 pa vhode. Določamo lahko vsak priključek posebej. Resetiranje postavi ta register na vrednost 0, torej imamo na začetku določene vse priključke kot vhode.



Slika 32: Smerni register porta C

Register za vklop pull-up uporov (PTCPUE)

Ta register ima naslov $000E. Omogoča nam vklop pull-up uporov, ko so priključki na portu definirani kot vhodi. Z logično 1 upore vklopimo, z logično 0 pa jih izklopimo. Omogočimo lahko vsak upor posebej. Resetiranje postavi ta register na vrednost 0. Če so priključki na portu določeni kot izhodi, pull-up uporov ne moremo uporabiti.



Slika 33: Register za vklop pull-up uporov porta C

Port D

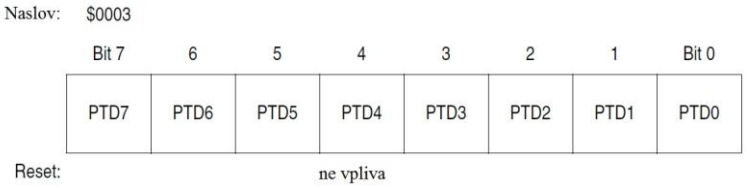
Ta port je 8-bitni pri mikrokontrolerju v 44- in 42-pinskem ohišju, v 40- pinskem pa imamo na razpolago le 6 priključkov (PTD5 do PTD0). Ima tri registre:

- podatkovni register (*Data Register* – PTD)
- smerni register (*Data Direction Register* – DDRD)
- register za vklop pull-up uporov (*Pullup Enable Register* – PTDPUE)

Zunanje priključke si deli z modulom časovnika (*timer interface module – TIM1, TIM2*) in modulom serijske komunikacije SPI (*serial peripheral interface module – SPI*).

Podatkovni register (PTD)

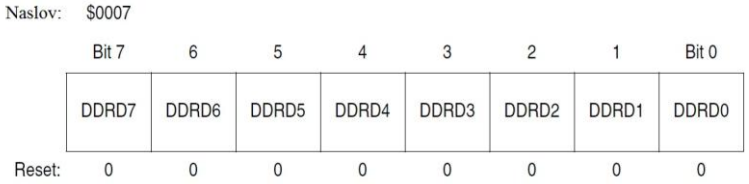
Nahaja se na naslovu $0003. Povezan je z zunanjimi priključki mikrokontrolerja. Če so ti priključki vhodi, vhodne vrednosti preberemo v tem registru. Če so ti priključki izhodi, bodo podatki, ki jih pošljemo v ta register, vidni na zunanjih priključkih mikrokontrolerja. Resetiranje ne vpliva na njegovo vsebino.



Slika 34: Podatkovni register porta D

Smerni register (DDRD)

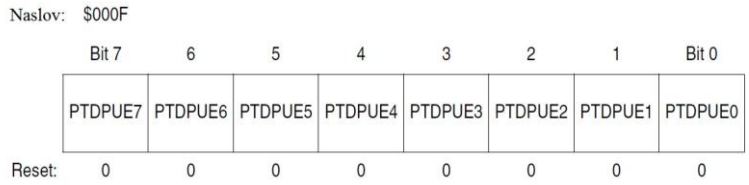
Nahaja se na naslovu $0007. Z njim določamo vrsto priključkov na portu. Z logično 1 v tem registru pripravimo izhode, z logično 0 pa vhode. Določamo lahko vsak priključek posebej. Resetiranje postavi ta register na vrednost 0, torej imamo na začetku določene vse priključke kot vhode.



Slika 35: Smerni register porta D

Register za vklop pull-up uporov (PTDPUE)

Ta register ima naslov $000F. Omogoča nam vklop pull-up uporov, ko so priključki na portu definirani kot vhodi. Z logično 1 upore vklopimo, z logično 0 pa jih izklopimo. Omogočimo lahko vsak upor posebej. Resetiranje postavi ta register na vrednost 0. Če so priključki na portu določeni kot izhodi, pull-up uporov ne moremo uporabiti.



Slika 36: Register za vklop pull-up uporov porta D

Port E

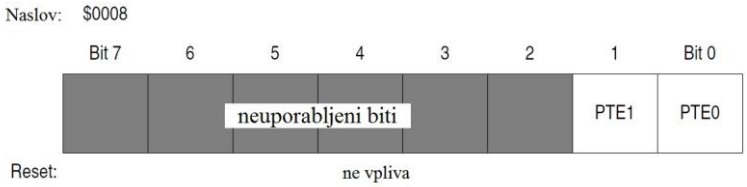
To je 2-bitni port in ima dva registra:

- podatkovni register (*Data Register* – PTE)
- smerni register (*Data Direction Register* – DDRE)

Zunanje priključke si deli z modulom serijske komunikacije SCI (*serial communications interface module – SCI*).

Podatkovni register (PTE)

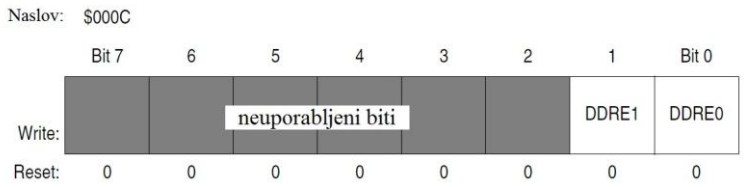
Nahaja se na naslovu $0008. Povezan je z zunanjimi priključki mikrokontrolerja. Če sta ta dva priključka vhoda, vhodni vrednosti preberemo v tem registru. Če sta priključka izhoda, bo podatek, ki ga pošljemo v ta register, viden na zunanjih priključkih mikrokontrolerja. Resetiranje ne vpliva na njegovo vsebino.



Slika 37: Podatkovni register porta E

Smerni register (DDRE)

Nahaja se na naslovu $000C. Z njim določamo vrsto priključkov na portu. Z logično 1 v tem registru pripravimo izhode, z logično 0 pa vhode. Določamo lahko vsak priključek posebej. Resetiranje postavi ta register na vrednost 0, torej imamo na začetku določena oba priključka kot vhoda.

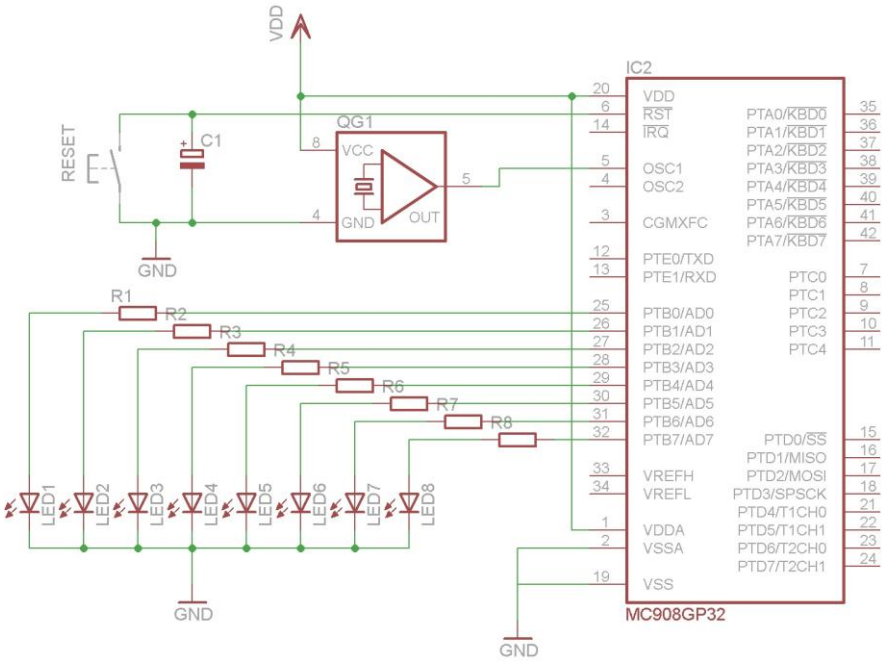


Slika 38: Smerni register porta D

Primeri uporabe portov

Primer 1

Ker porti služijo pridobivanju informacij iz okolja ter upravljanju naprav in prikazovanju stanj ter vrednosti, si bomo pogledali, kako to lahko počnemo. Za začetek bomo priključili svetleče diode (LED) na port B. Ker je največji dovoljeni tok na posameznem izhodu porta B 10 mA, moramo z ustrezno vrednostjo uporov ta tok omejiti. S programom bomo prižgali diode na nižjih štirih priključkih porta (PTB3 do PTB0), diode na priključkih od PTB7 do PTB4 pa naj ostanejo ugasnjene.



Slika 39: El. shema vezja z mikrokontrolerjem za Primer 1

V programu moramo poskrbeti, da bo mikrokontroler po priklopu na napajanje oziroma po resetiranju našel začetek programa. To naredimo tako, da na naslova $FFFE in $FFFF vpišemo naslov prvega ukaza v našem programu (Reset vektor). Zatem moramo določiti vrsto priključkov na portu B. Če hočemo prižigati diode, moramo določiti vse priključke porta B kot izhode. To naredimo v smernem registru porta B. Ostane nem samo še pošiljanje ustrezne vrednosti v podatkovni register na portu B, da prižgemo želene svetleče diode. Z logično enico diodo prižgemo, logična vrednost 0 pa diodo ugasne:

C:\pemicro\ics08gpgtz\LED1.asm

```
        ORG     $FFFE
        FDB     $8000       ; vpis Reset vektorja
*************************
        ORG     $8000       ; določimo naslov začetka programa
        LDA     #$FF        ; v akumulator damo same log. 1
        STA     $05         ; vsebino akumulatorja shranimo v smerni register
        LDA     #$0F        ; v akumulator damo log. 1 na nižje 4 bite, log. 0 pa na višje štiri bite
        STA     $01         ; vsebino akumulatorja shranimo v podatkovni register
S1      BRA     S1
```

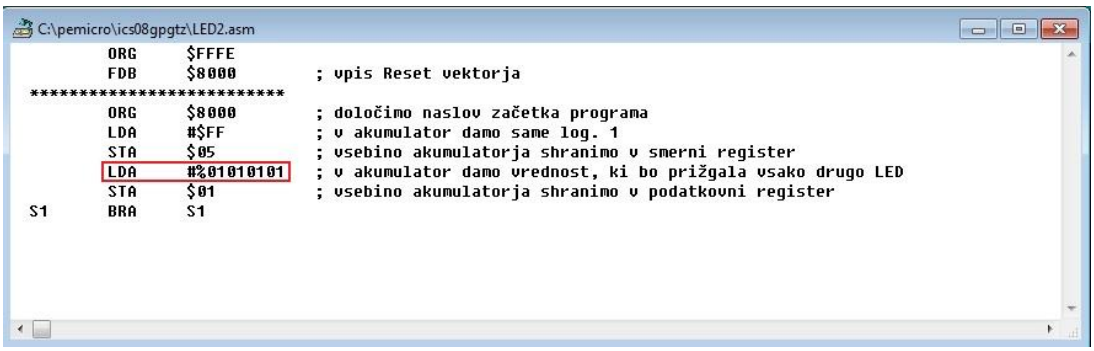
Slika 40: Program v zbirnem jeziku za Primer 1

Program smo v tem primeru umestili na naslove od $8000, kar je na začetku *Flash* pomnilnika. Nato smo s pomočjo akumulatorja shranili v smerni register porta B vrednost, ki nam pripravi izhodne priključke na portu. Ko pošljemo v podatkovni register vrednost $0F, le-ta povzroči zahtevani vklop svetlečih diod. Program je zaključen z neskončno zanko, v kateri se ponavlja ukaz `BRA`, ki povzroča stalno preskakovanje v vrstico s tem ukazom. Delovanje programa lahko preverimo s simulatorjem (*In-Circuit Simulator*). Na ta način lahko preverimo odzivanje mikrokontrolerja po vsakem izvedenem ukazu. Tako preverjanje je sicer počasnejše, saj

mikrokontroler zaradi stalne komunikacije z osebnim računalnikom izvaja ukaze več časa. Prednost takega načina pa je ta, da lahko vedno spremljamo stanja vseh registrov mikrokontrolerja in njegovega pomnilnika, tudi morebitne napake v programih lažje odkrijemo.

Program lahko s programskim orodjem *Programmer* vnesemo tudi v *Flash* pomnilnik in po resetiranju preizkusimo njegovo delovanje v realnem času. Vendar na tak način nimamo vpogleda v mikrokontroler (v njegove registre in pomnilnik), iskanje morebitnih napak v programih je zato težje.

Kaj moramo v programu spremeniti, da bi prižgali vsako drugo LED in sicer na PTA0, PTA2, PTA4 in PTA6?

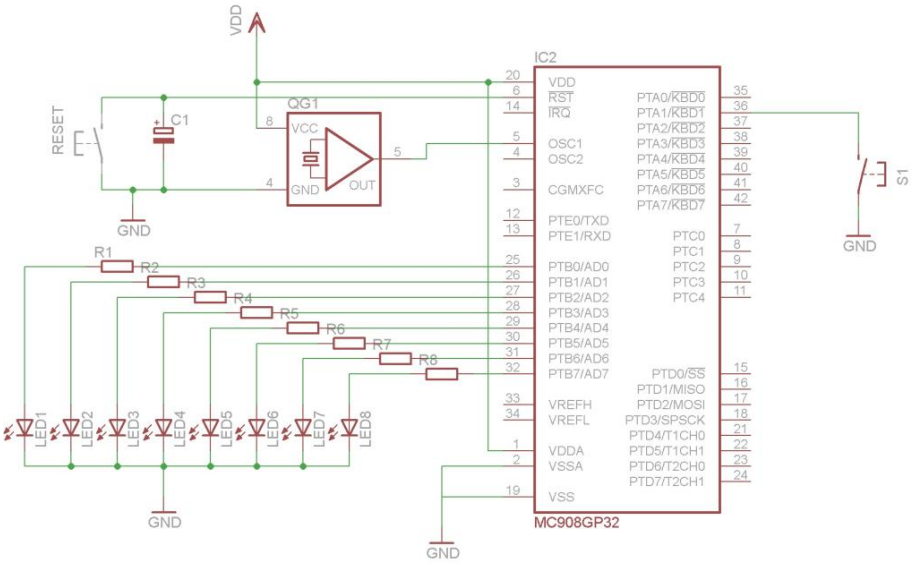


Slika 41: Sprememba v programu za Primer 1

Kot lahko vidimo na zgornji sliki, spremenimo samo vrednost, ki jo pošljemo v podatkovni register (obkroženi ukaz `LDA`). S tem spremenimo vrednost, ki se bo shranila v podatkovni register. Ta vrednost se pojavi na priključkih porta in določa, katere diode bodo svetile.

Primer 2

Na portu A imamo priključeno tipko na PTA1, drugi priključki so neuporabljeni, na portu B pa imamo svetleče diode (slika 42). Na začetku naj bodo diode ugasnjene. Ko pritisnemo na tipko, naj zasveti dioda na PTB3, ki ostane prižgana tudi, ko tipko spustimo.



Slika 42: El. shema vezja z mikrokontrolerjem za Primer 2

Ker bomo sedaj uporabljali poleg porta B z diodami še port A, na katerem imamo tipko, moramo pripraviti tudi priključke na tem portu. Ker je stanje v smernem registru po resetiranju (ali priklopu) mikrokontrolerja tako, da so vsi priključki določeni kot vhodi, nam ni potrebno v tem registru nič spremeniti. Kot vidimo na shemi, je tipka na PTA1 povezana na maso. Ko tipko pritisnemo, se zato na tem priključku pojavi logična vrednost 0. Kaj pa če tipka ni pritisnjena? Ker imajo vhodi v mikrokontroler veliko upornost, bi v tem primeru lahko s prebiranjem vrednosti dobili naključno vrednost, ki je odvisna predvsem od statičnega električnega naboja na priključku. Zato bomo uporabili pull-up upor, ki bo v takem primeru na vhodu zagotavljal logično vrednost 1.

Ko pripravimo priključke, ki jih potrebujemo, moramo najprej zagotoviti, da diode ne bodo svetile. Nato moramo preverjati stanje na vhodu PTA1, dokler tipke ne pritisnemo. Glede na vezavo tipke bo to stanje logična 1. Ko tipko pritisnemo, se na vhodu pojavi logična 1. Takrat moramo prižgati LED na PTB3.

C:\pemicro\ics08gpgtz\LED3.asm

```
        ORG     $FFFE
        FDB     $8000           ; vpis Reset vektorja
*******************************
        ORG     $8000           ; določimo naslov začetka programa
        MOV     #$FF,$05        ; določimo izhode na portu B
        MOV     #%00000010,$0D  ; vklopimo pull-up upor na vhodu PTA1
*******************************
        CLR     $01             ; vse bite v PTB postavimo na log. 0
K1      BRSET   1,$00,K1        ; preverjamo stanje tipke na PTA1
        MOV     #%00001000,$01  ; prižgemo LED na PTB3
S1      BRA     S1
```

Slika 43: Program v zbirnem jeziku za Primer 2

V tem programu je uporabljen ukaz `MOV`, ki je nadomestil ukaza `LDA` in `STA`. To je eden izmed sestavljenih ukazov, ki nam je omogočil, da smo želene vrednosti shranili v ustrezna registre. Pri tem ukazu imamo dva operanda, ločena z vejico. Prvi določa, katero vrednost bomo prenesli oz.

shranili, v drugem pa določimo, na kateri naslov bomo vrednost shranili. V našem primeru je bila vrednost vedno podana že v samem ukazu, lahko pa podamo tudi naslov, na katerem ukaz dobi vrednost za shranjevanje oziroma prenašanje.

Po pripravi portov najprej z ukazom `CLR` poskrbimo, da so vsi biti v `PTB` na logični 0, kar pomeni, da diode ne bodo svetile. Zatem z ukazom `BRSET` kontroliramo stanje na PTA1. Ta ukaz je eden izmed vejitvenih ukazov. Z njim lahko preverjamo stanje določenega bita na nekem naslovu. Če je ta bit na logični 1, se vejitev opravi in program se nadaljuje v vrstici, ki ima podano oznako. Če ta pogoj ni izpolnjen, se program nadaljuje z naslednjim ukazom. V našem primeru preverjamo stanje bita 1 na naslovu $00, torej priključka PTA1, na katerem je tipka. Dokler tipke ne pritisnemo, je na vhodu logična 1. Vejitveni ukaz zato povzroči nadaljevanje programa v vrstici z oznako K1. To pa je ponovno ukaz, ki preverja stanje tipke. To se ponavlja, dokler tipke ne pritisnemo. Takrat se program nadaljuje z ukazom, ki prižge LED na izhodu PTB3.

Kako pa bi naredili program, ki bi povzročil npr. utripanje vseh svetlečih diod na portu B? Videli smo že, kako lahko prižgemo ali ugasnemo diode. Če bi napisali program, ki bi imel ukaze za prižiganje in ugašanje diod, bi med izvajanjem takega programa videli prižgane vse diode. Zakaj?

Izvajanje ukazov v mikrokontrolerju je namreč tako hitro, da bi se prižiganje in ugašanje dogajalo v nekaj μs. Tako hitrim spremembam pa naše oko ne more slediti. Zato moramo po vsaki spremembi stanja diod to stanje pustiti na portu več časa. V programu moramo narediti zakasnitve. Za to pa rabimo časovnik, kar bomo spoznali v naslednjem poglavju.

Ponovimo

Priključki mikrokontrolerja so razdeljeni v skupine, ki jih imenujemo porti. Vsak priključek je lahko binarni vhod ali izhod. Nekateri imajo dvojno funkcijo, saj jih lahko uporabimo tudi z drugimi enotami mikrokontrolerja (moduli), kot so A/D pretvornik, časovnik, modul serijske komunikacije in drugi. Ta mikrokontroler ima pet portov z oznakami A, B, C, D in E.

Nastavitve in uporabo portov izvajamo s pomočjo registrov. Vsak port ima podatkovni in smerni register. Prvi omogoča prenos podatkov med programom in zunanjimi priključki, drugi pa je namenjen določevanju vhodov in izhodov na portu. Nekateri porti imajo vgrajene še pull-up upore, ki jih z ustrezno nastavitvijo lahko uporabimo in tako zmanjšamo število zunanjih komponent sistema. S podanimi primeri smo spoznali, ka ko lahko nastavimo in uporabimo priključke mikrokontrolerja.

Vaje

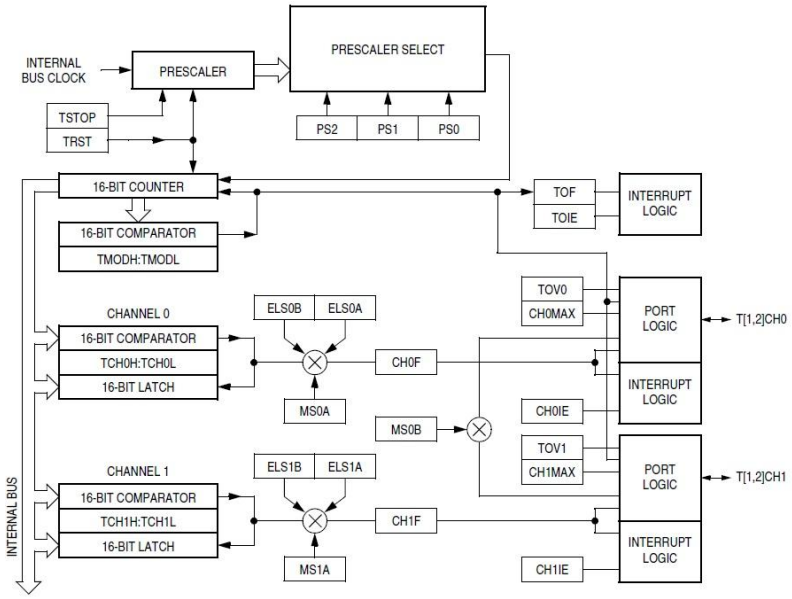
Delovanje programov preveri na mikrokontrolerju s simulatorjem (*In-Circuit Simulator*).

1. Na portu B imamo priključene svetleče diode. Napiši program, ki bo prižgal diode na priključkih PTB0 in PTB4.
2. Na portu B imamo priključene svetleče diode. Napiši program, ki bo povzročil utripanje diod na priključkih od PTB0 do PTB3.
3. Na portu A imamo priključeno tipko na PTA1, na portu B svetleče diode. Napiši program, ki bo najprej prižgal diodo na priključku PTB7. Nato bo po pritisku na tipko prižgal vse LED na portu B.
4. Na portu A imamo priključeno tipko na PTA1, na portu B svetleče diode. Napiši program, ki bo najprej prižgal diodo na priključku PTB0. Nato bo po vsakem pritisku na tipko prižgal naslednjo diodo vse do tiste na PTB7. Ob naslednjem pritisku tipke naj se zopet prižge dioda na PTB0. Pri tem mora program vsakokrat po pritisku tipke počakati tudi na spust tipke.

ČASOVNIK

Zgradba in delovanje

MC908GP32 ima dva neodvisna enaka časovnika. Omogočata nam realizacijo zakasnitev, ki jih uporabimo takrat, ko je delovanje mikrokontrolerja prehitro in želimo upočasniti njegovo odzivanje. Osnova časovnika je 16-bitni števec, ki šteje impulze do nastavljene vrednosti. Ko to doseže, nam to javi, nato postavi vse bite na logično 0 (se resetira) ter nadaljuje znova s štetjem impulzov. To se ponavlja, dokler časovnika ne zaustavimo. To storimo tako, da blokiramo prihod impulzov do števca.



Slika 44: Blokovna shema časovnika
(vir: MC68HC908GP32 Data Sheet, Rev. 10 1/2008, Freescale Semiconductor, Inc., (pdf datoteka proizvajalca))

Izvor impulzov za števec je taktni signal mikrokontrolerja. Ta signal gre na delilnik frekvence, kjer se lahko njegova frekvenca zmanjša. Delilno razmerje programsko določamo z biti PS2, PS1 in PS0. Nato ta signal pride do vhoda števca. Stanje števca se stalno primerja z nastavljeno vrednostjo. Ko jo števec doseže, primerjalnik to javi s postavitvijo bita TOF na logično 1, hkrati pa števec resetira. Z nastavitvami lahko določimo, da ob tem časovnik povzroči tudi zahtevo po prekinitvi.

Poleg te osnovne funkcije zakasnitve ima časovnik še druge možnosti uporabe. Pogledali si bomo, kako z njim lahko izdelamo pulzno-širinsko moduliran signal, ki ga dobimo na njegovih zunanjih priključkih.

Vse nastavitve in trenutna stanja v časovniku opravljamo in vidimo v njegovih registrih.

Registri časovnika

Mikrokontroler ima dva enaka časovnika. Oba imata po 5 registrov. Delovanje in nastavitve so za oba časovnika enake, le naslovi registrov so različni. Tako bomo videli v oznakah registrov številki 1 in 2, ki se nanašata na časovnik 1 (*Timer 1*) in časovnik 2 (*Timer 2*).

Statusni in kontrolni register

Ta register nam omogoča nastavitve delilnega razmerja taktnega signala, zaustavitev ali zagon časovnika, brisanje nastavitev, omogočanje zahteve po prekinitvi ter kontrolo konca zakasnitve oz. konec štetja 16-bitnega števca. Naslova registrov obeh časovnikov in razporeditev bitov v registru so podani na spodnji sliki:

Naslov: T1SC:$0020 in T2SC:$002B

	Bit 7	6	5	4	3	2	1	Bit 0
Branje:	TOF	TOIE	TSTOP	0	0	PS2	PS1	PS0
Pisanje:	0			TRST				
Reset:	0	0	1	0	0	0	0	0

Slika 45: Statusni in kontrolni register časovnika

Pomen bitov:

TOF – Ta bit nam s postavitvijo na logično 1 javi, da je 16-bitni števec dosegel nastavljeno vrednost štetja. Če je njegova vrednost 0, števec še ni preštel impulzov do nastavljene vrednosti. Bit moramo z ukazom postaviti na logično 0, da lahko zaznamo nov konec nastavljene zakasnitve.

TOIE – Če ta bit postavimo na logično 1, omogočimo časovniku zahtevo po prekinitvi.

TSTOP – Če je ta bit na logični 1, je delovanje časovnika zaustavljeno. S postavitvijo bita na logično 0 poženemo časovnik oz. dovolimo taktnim impulzom dostop do 16-bitnega števca časovnika.

TRST – Ko postavimo ta bit na logično 1, pobrišemo nastavitev delilnega razmerja za taktni signal ter vrednost 16-bitnega števca časovnika.

PS2, PS1 in PS0 – Omogočajo nam nastavitev delilnega razmerja in s tem frekvence taktnega signala, ki spreminja stanje 16-bitnega števca časovnika. V spodnji tabeli so podana delilna razmerja:

PS2	PS1	PS0	Frekvenca signala v časovniku
0	0	0	Taktni signal mikrokontrolerja : 1
0	0	1	Taktni signal mikrokontrolerja : 2
0	1	0	Taktni signal mikrokontrolerja : 4
0	1	1	Taktni signal mikrokontrolerja : 8
1	0	0	Taktni signal mikrokontrolerja : 16
1	0	1	Taktni signal mikrokontrolerja : 32
1	1	0	Taktni signal mikrokontrolerja : 64
1	1	1	neuporabljeno

Tabela 1: Nastavitve delilnega razmerja taktnega signala

Register števca časovnika

V tem registru lahko dobimo trenutno stanje 16-bitnega števca časovnika. Zato je tudi ta register 16-bitni. Nahaja se na naslednjih naslovih:

$0021 in $0022 – višji in nižji bajt registra števca časovnika 1 (T1CNTH in T1CNTL)
$002C in $002D – višji in nižji bajt registra števca časovnika 2 (T2CNTH in T2CNTL)

Register modula štetja

Vrednost, ki jo vpišemo v ta 16-bitni register, določa modul štetja 16-bitnega števca časovnika. Digitalni komparator primerja stanje števca z vrednostjo tega registra. Ko števec doseže to vrednost, komparator postavi TOF bit na logično 1, resetira 16-bitni števec in povzroči zahtevo po prekinitvi, če je le-ta omogočena. Nahaja se na naslednjih naslovih:

$0023 in $0024 – višji in nižji bajt registra modula štetja časovnika 1 (T1MODH in T1MODL)
$002E in $002F – višji in nižji bajt registra modula štetja časovnika 2 (T2MODH in T2MODL)

Statusni in kontrolni register kanala

Vsak časovnik ima na razpolago dva zunanja priključka mikrokontrolerja, imenovana kanal 0 in kanal 1. Nahajata se na portu D in sta lahko vhodna ali izhodna. Ko uporabljamo ta priključka s časovnikom, se nastavitve v registrih porta D ne upoštevajo. Uporabo teh priključkov določamo s statusnim in kontrolnim registrom posameznega kanala ter z registrom kanala. Nastavitve in uporaba teh registrov so za srednješolski nivo zahtevne in obsežne, zato si bomo pogledali samo uporabo kanala za generiranje pulzno-širinsko moduliranega signala (*PWM*).

Na spodnjih slikah lahko vidimo naslove registrov obeh časovnikov ter razporeditve bitov v njih. Statusni in kontrolni register kanala 0:

Naslov: T1SC0:$0025 in T2SC0:$0030

	Bit 7	6	5	4	3	2	1	Bit 0
Branje:	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	TOV0	CH0MAX
Pisanje:	0							
Reset:	0	0	0	0	0	0	0	0

Slika 46: Statusni in kontrolni register kanala 0

Statusni in kontrolni register kanala 1:

Naslov: T1SC1:$0028 in T2SC1:$0033

	Bit 7	6	5	4	3	2	1	Bit 0
Branje:	CH1F	CH1IE	0	MS1A	ELS1B	ELS1A	TOV1	CH1MAX
Pisanje:	0							
Reset:	0	0	0	0	0	0	0	0

Slika 47: Statusni in kontrolni register kanala 0

Register kanala

Ko uporabljamo zunanji priključek časovnika (kanal) kot izhodni priključek za PWM signal, ima ta register podobno funkcijo kot register modula štetja. Vrednost, ki jo vpišemo vanj, bo določala, kdaj bo v periodi signala nastala sprememba logičnega stanja (napetosti) na izhodu. S tem določimo razmerje med impulzom in pavzo (*duty cycle*) v PWM signalu. Vrednost tega registra mora biti manjša od vrednosti v registru modula štetja.

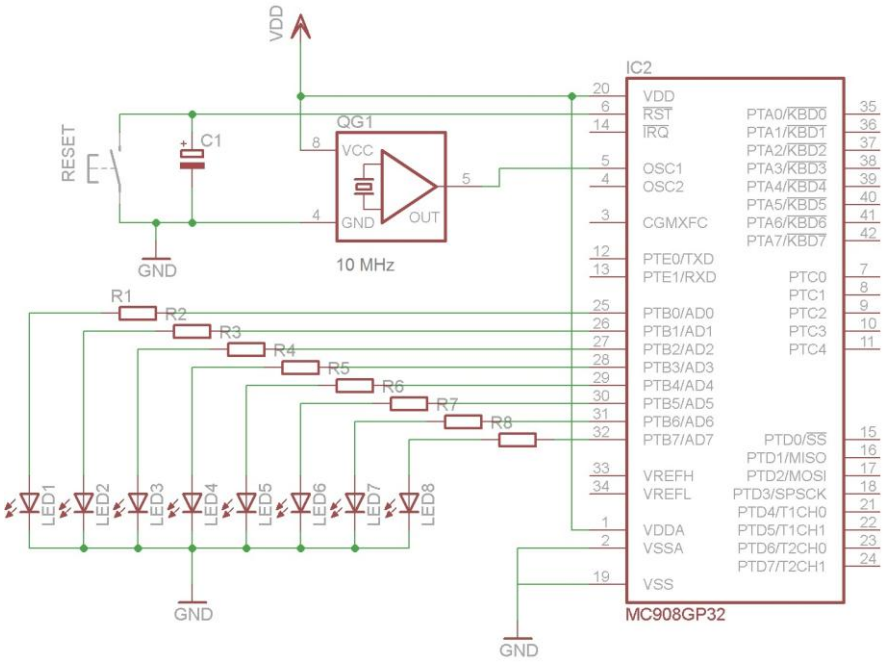
Registri kanala, ki so 16-bitni, se nahajajo na naslednjih naslovih:

$0026 in $0027 – višji in nižji bajt registra kanala 0 časovnika 1 (T1CH0H in T1CH0L)
$0031 in $0032 – višji in nižji bajt registra kanala 0 časovnika 2 (T2CH0H in T2CH0L)
$0029 in $002A – višji in nižji bajt registra kanala 1 časovnika 1 (T1CH1H in T1CH1L)
$0034 in $0035 – višji in nižji bajt registra kanala 1 časovnika 2 (T2CH1H in T2CH1L)

Uporaba časovnika

Izvajanje zakasnitev

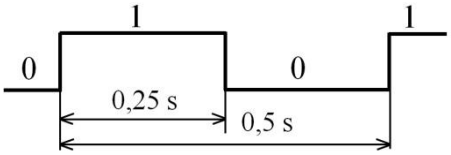
Vrnimo se na problem, zastavljen na koncu prejšnjega poglavja. Na portu B imamo priključene svetleče diode. Želimo, da utripajo, in sicer s frekvenco 2 Hz.



Slika 48: El. shema vezja z mikrokontrolerjem za uporabo časovnika

Na mikrokontroler imamo priključen oscilator, ki daje signal s frekvenco 10 MHz. Na vhodu mikrokontrolerja je delilnik frekvence, ki frekvenco tega signala deli s 4. Tako ima mikrokontroler taktni signal s frekvenco 2,5 MHz.

Port B pripravimo tako, kot smo to storili že v primeru prižiganja LED v prejšnjem poglavju. Podrobneje si bomo pogledali nastavitev časovnika. Če želimo, da diode utripajo s frekvenco 2 Hz, jih moramo krmiliti s signalom, ki ima čas trajanja periode 0,5 s. Torej moramo spreminjati logično stanje diod vsakih 0,25 s. To bo tudi čas zakasnitve, ki ga bomo nastavili v časovniku.



Slika 49: Časovni diagram signala na izhodu

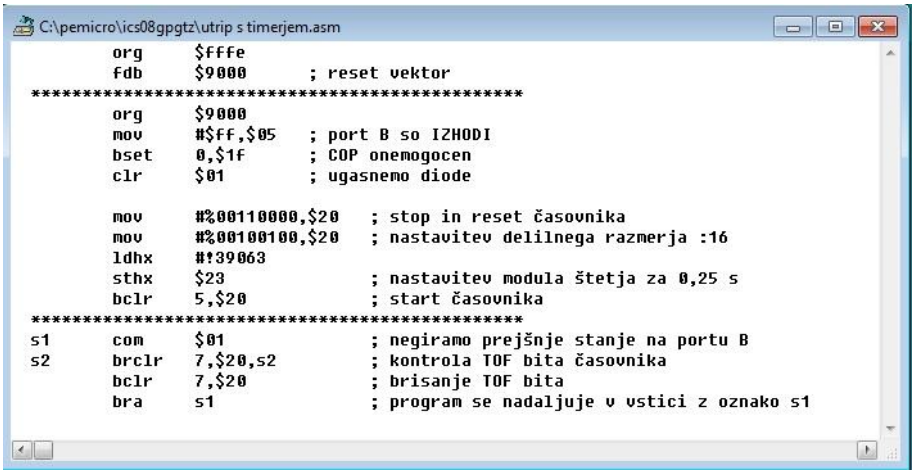
Izberemo si časovnik 1. Najprej časovnik zaustavimo in resetiramo. To storimo s postavitvijo bitov TSTOP in TRST v njegovem kontrolnem in statusnem registru na logično 1. Nato določimo delilno razmerje, s katerim se bo delil taktni signal (2,5 MHz). Izberemo delilno razmerje :16. Tako je frekvenca impulzov, ki prihajajo na vhod 16-bitnega števca, 156,25 kHz. Čas trajanja periode tega signala je zato 6,4 µs. Modul štetja števca dobimo tako, da izračunamo, koliko impulzov mora števec prešteti, da preteče čas 0,25 s. To dobimo s spodnjo enačbo:

$$n = \frac{0{,}25\ s}{6{,}4\ \mu s} = 39062{,}5 \approx 39063 \quad ;\ n = \check{s}tevilo\ impulzov$$

Enačba 1: Izračun modula štetja časovnika

Ker dobimo število impulzov z decimalnim mestom, vrednost zaokrožimo na celo število. To vrednost vpišemo v register modula štetja časovnika. Zaradi zaokroževanja bo sicer čas zakasnitve odstopal od želenega, vendar bo to odstopanje za 3,2 μs, kar je zanemarljivo in za nas neopazno.

Do tega dela programa mora biti časovnik zaustavljen. Sedaj ga poženemo, in sicer tako, da TSTOP bit postavimo na logično 0. Zatem moramo samo spreminjati vrednost na portu B, kar naredimo z negacijo stanja v podatkovnem registru (PTB). Po spremembi stanja uporabimo zakasnitev. To storimo tako, da preverjamo stanje TOF bita v statusnem in kontrolnem registru časovnika (T1SC), dokler se ne postavi na logično 1 (ukaz BRCLR). Ko preteče nastavljeni čas, se ta bit postavi na logično 1 in program se nadaljuje z ukazom, ki briše TOF bit. S tem ga pripravimo za naslednje javljanje konca zakasnitve.



Slika 50: Program za utripanje svetlečih diod

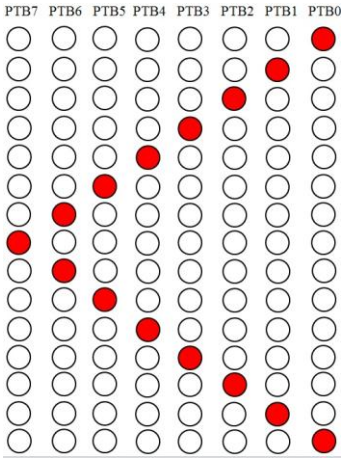
V programu zasledimo tudi ukaz, ob katerem je komentar "`COP onemogočen`". S tem ukazom (`bset 0,$1f`) zaustavimo delovanje varnostnega modula COP (*Computer Operating Properly*). Ta modul povzroči resetiranje mikrokontrolerja, če se program zaradi morebitnih napak ne bi več odzival. Ker za naše primere ni potrebno, da ta modul deluje, ga onemogočimo v konfiguracijskem registru 1 (CONFIG1), ki se nahaja na naslovu $001F. V njem je bit COPD (bit 0 tega registra), ki zaustavi delovanje modula, če je na logični vrednosti 1.

Program vnesemo v *Flash* pomnilnik mikrokontrolerja (*Programmer*) in preverimo delovanje. Ko uporabljamo časovnik, delovanja programa praviloma ne preizkušamo s simulatorjem *(In-Circuit Simulator*), ker bi zaradi počasnega izvajanja posameznih ukazov čakali npr. konec zakasnitve izredno veliko časa. Lahko si pomagamo z razhroščevalnikom (*In-Circuit Debugger*), kjer med drugim lahko nastavimo, na kateri točki programa se bo izvajanje le-tega zaustavilo (*Breakpoint*). Takrat lahko pogledamo stanje registrov in pomnilnika mikrokontrolerja. Mesta zaustavitve programa lahko premeščamo in tako na različnih delih programa preverimo pravilnost delovanja.

Tekoča luč

Isto povezavo svetlečih diod (slika 48) bomo uporabili še za en primer. Sestavili bomo program, ki bo v zaporedju prižigal po eno svetlečo diodo, najprej od najnižjega bita (priključka PTB0) proti najvišjemu (priključku PTB7), nato pa zopet proti najnižjemu. Prižiganje diod naj se spreminja vsako desetinko sekunde.

Zaporedje prižiganja svetlečih diod prikazuje slika 51:



Slika 51: Zaporedje prižiganja LED

Tudi pri tem programu moramo pripraviti na portu B izhode. Poleg tega bomo zaustavili delovanje COP modula, za realizacijo zakasnitve pa bomo uporabili časovnik 1 (*Timer 1*), ki ga bomo nastavili tako, da bomo z njim dosegli zakasnitev 0,1 sekunde.

C:\pemicro\ics08gpgtz\TEKOČA LUČ.asm

```
        org     $fffe
        fdb     $a000               ; reset vektor
        org     $a000
        mov     #$ff,$05            ; port B so IZHODI
        bset    0,$1f               ; COP onemogočen

        mov     #%00110000,$20      ; stop in reset
        mov     #%00100011,$20      ; stop in CLK (fclk/8)  TIMER 1
        ldhx    #!31250             ; modul štetja
        sthx    $23                 ; 0,1s
        bclr    5,$20               ; start timerja
************************************************
        lda     #%00000001
        sta     $01          ; vklop diode na PTB0
s1      brclr   7,$20,s1     ; zakasnitev 0,1s
        bclr    7,$20        ; brisanje TOF bita
        lsla                 ; pomik vsebine akumulatorja za eno mesto v levo
        sta     $01          ; novo vrednost shranimo v PTB
        cmp     #%10000000   ; preverjamo, če je že prižgana dioda na PTB7
        bne     s1           ; če ni, izvajamo isti pomik po zakasnitvi

**** program začne izvajati pomikanje v desno  ****

s2      brclr   7,$20,s2     ; zakasnitev 0,1s
        bclr    7,$20        ; brisanje TOF bita
        lsra                 ; pomik vsebine akumulatorja za eno mesto v desno
        sta     $01          ; novo vrednost shranimo v PTB
        cmp     #%00000001   ; preverjamo, če je že prižgana dioda na PTB0
        bne     s2           ; če ni, izvajamo isti pomik po zakasnitvi

        bra     s1           ; izvajanje programa se po zakasnitvi
                             ; nadaljuje s pomikanjem v levo
```

Slika 52: Program za tekočo luč

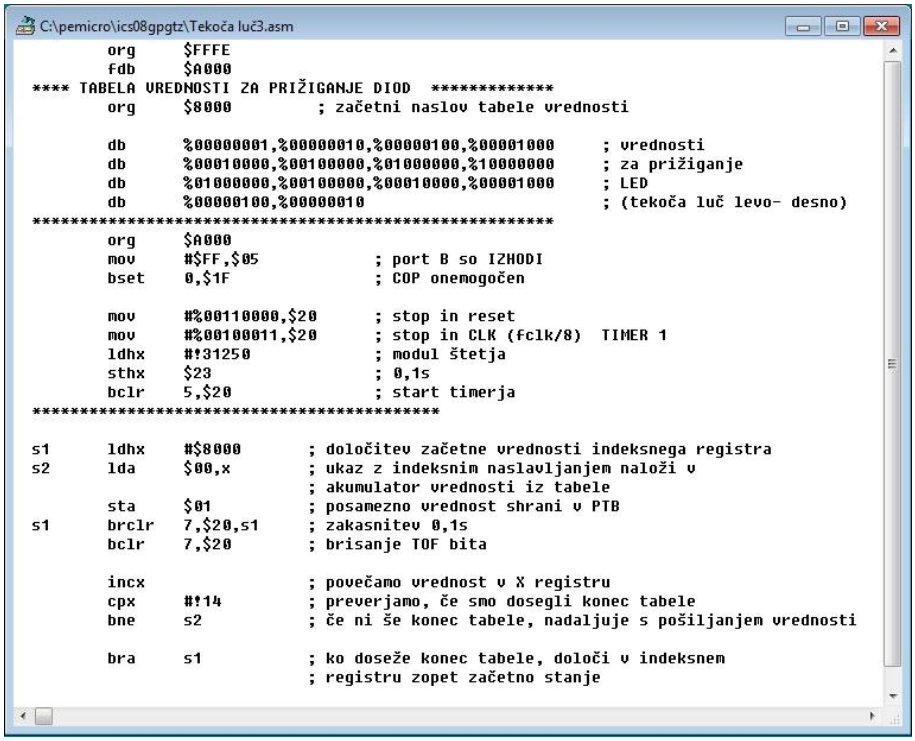
V programu izvajamo najprej prižiganje diod na priključkih od PTB0 do PTB7. To nam omogoča ukaz `lsla`, s katerim vsakokrat pomaknemo vsebino akumulatorja za eno mesto v levo. Tako se logična 1, ki smo jo v začetku imeli na najnižjem bitu, pomika proti najvišjemu bitu akumulatorja. Ker vsebino tega registra shranjujemo v podatkovni register porta B, nam ta logična 1 tudi prižiga svetleče diode v istem zaporedju.

Z ukazom za primerjanje `cmp #%10000000` preverjamo, če je logična 1 v akumulatorju že dosegla najvišji bit. Ta ukaz namreč odšteva od trenutne vrednosti akumulatorja podano konstantno vrednost. Ko bo logična 1 v akumulatorju s pomikanjem zasedla najvišji bit, bo rezultat primerjave oziroma odštevanja postal 0. Takrat ne bo izpolnjen pogoj za vejitev z ukazom `bne`, zato se bo program nadaljeval v drugi polovici, kjer se z ukazom `lsra` izvaja pomik vsebine akumulatorja v desno. V tem delu preverjamo, kdaj bo logična 1 v akumulatorju zasedla mesto najnižjega bita.

S časovnikom poskrbimo, da vsako stanje na portu B ostane nespremenjeno za nastavljeni čas zakasnitve. Če zakasnitve ne bi imeli, naše oko ne bi sledilo hitrim spremembam na portu. Zdelo bi se nam, kot da so vse svetleče diode stalno prižgane. Kot vidimo, je zakasnitev v programu uporabljena dvakrat. Tudi ukaza, ki omogočata izvedbo zakasnitve, sta obakrat napisana. V nekaterih programih bomo isto kombinacijo ukazov potrebovali tudi večkrat. Večkratnemu pisanju istih ukazov se lahko izognemo z uporabo podprogramov, kjer ponavljajočo se kombinacijo ukazov za izvajanje določene naloge enkrat zapišemo, potem pa z enim ukazom kjerkoli v programu zahtevamo izvajanje takega podprograma. To bomo podrobneje spoznali v naslednjem poglavju.

Enako prižiganje svetlečih diod lahko dosežemo na več načinov. Spoznali bomo še en način, katerega princip bomo lahko uporabili pri mnogih drugih programih. Uporabili bomo tabelo vrednosti, s katero bomo diode prižigali. Na sliki 51 vidimo zaporedje prižiganja diod na portu B. Vsako stanje, ko je ena dioda prižgana, ustreza 8-bitni vrednosti, ki je takrat v podatkovnem registru porta (PTB). Te vrednosti bomo vnesli v *Flash* pomnilnik na zaporedne naslove, od koder jih bomo pošiljali v podatkovni register porta B. Vmes bomo med posameznimi pošiljanji uporabili še zakasnitev, ki bo enaka kot v prejšnjem primeru.

Da bomo lahko dostopali do teh vrednosti v tabeli, bomo pri ukazu, ki bo "poiskal" posamezno vrednost, uporabili indeksno naslavljanje.



Slika 53: Program za tekočo luč z uporabo tabele in indeksnim naslavljanjem

Priprava porta in nastavitve časovnika so pri tem programu enake kot pri prejšnjem. Programu smo dodali tabelo vrednosti, ki jih med izvajanjem programa pošiljamo na izhodne priključke. Te vrednosti nam določajo način prižiganja svetlečih diod. Ta tabela je postavljena v *Flash* pomnilnik od naslova $8000 dalje. Ker je v njej 14 vrednosti, se tabela konča z zadnjo vrednostjo na naslovu $800D.

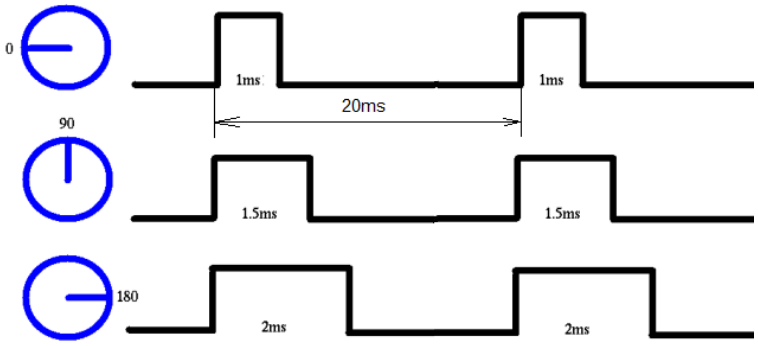
Program, ki vrednosti iz tabele prenaša v podatkovni register porta B (PTB), uporablja ukaz `lda` z indeksnim naslavljanjem (`lda $00,x`). To pomeni, da je naslov, ki ga ukaz uporabi, določen z vsoto trenutne vrednosti indeksnega registra in relativnega naslova. Ta relativni naslov je vrednost, ki jo zapišemo v ukazu. V našem primeru je 0 ($00). Torej bo naslov, s katerega se bo vrednost naložila v akumulator, kar enaka vrednosti indeksnega registra. Kot vidimo v programu, je začetna vrednost registra $8000, kar predstavlja tudi začetni naslov tabele vrednosti za pošiljanje na port B. Po vsaki poslani vrednosti sledi zakasnitev v trajanju 0,1 s, zatem pa z

ukazom `incx` za 1 povečamo vrednost v indeksnem registru. To naredimo zato, da ob naslednji uporabi ukaza `lda $00,x` v akumulator dobimo vrednost z naslednjega naslova ($8001, zatem $8002 in tako naprej). Ker je dolžina tabele omejena na 14 vrednosti in prav toliko naslovov, moramo paziti, kdaj pošljemo zadnjo vrednost iz tabele. To nadziramo z ukazom `cpx #!14`, ki primerja vrednost X dela (spodnjega bajta) indeksnega registra z vrednostjo 14 oziroma $0E. Če te vrednosti še ni v registru, program nadaljuje s povečevanjem naslovov pošiljanjem pripadajočih vrednosti iz tabele. Ko pa vrednost v indeksnem registru doseže vrednost $800E, leta predstavlja naslov izven naše tabele. Zato takrat izvajanje programa nadaljujemo z ukazom, ki v indeksni register spet naloži začetno vrednost.

Če želimo spremeniti vzorec prižiganja svetlečih diod, spremenimo vrednosti v tabeli. Paziti moramo le, da ustrezno spremenimo tudi primerjalno vrednost za končanje zanke, če se nam spremeni obseg (dolžina) tabele.

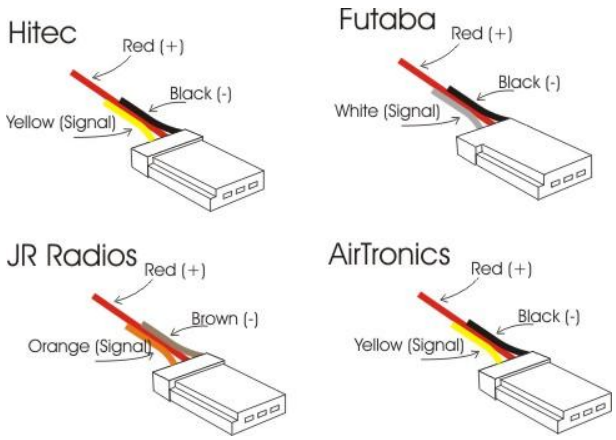
Realizacija PWM signala

Z mikrokontrolerjem bi radi krmilili servomotor. Te motorje krmilimo s signali, ki imajo periodo okrog 20 ms, čas impulza (logične 1) pa od 1 ms do 2 ms:



Slika 54: PWM signal za krmiljenje servomotorjev

Za začetek bi ga postavili v sredinski položaj. Uporabili bomo kanal 0 časovnika 1 (priključek PTD4/T1CH0), na katerega povežemo krmilni priključek servomotorja. Motor ima poleg krmilnega še napajalna priključka (+5 V, GND). Na spodnji sliki vidimo konektorje servomotorjev različnih proizvajalcev.



Slika 55: Konektorji servomotorjev
(vir: http://www.imagesco.com/servo/connection-types.html)

Pregledali bomo korake, ki so potrebni pri pripravi in uporabi zunanjih priključkov časovnika za PWM signal:

- zaustavi in resetiraj časovnik
- nastavi delilno razmerje
- nastavi modul štetja za želeni čas trajanja periode signala (T1MODH in T1MODL)
- nastavi vrednost za želeni čas trajanja impulza signala (*duty cycle*) v registru kanala (T1CH0H in T1CH0L)
- vnesi potrebne nastavitve v statusni in kontrolni register kanala (T1SC0):
  - vpiši 0:1 v MS0B:MS0A (unbuffered output compare or PWM signals)
  - vpiši 1 v TOV0 (*toggle-on-overflow bit*) – spremeni stanje ob koncu periode
  - vpiši 1:0 v ELS0B:ELS0A (*clear output on compare*) – log. 0 po impulzu
- poženi timer

Program, ki upošteva te korake, je na spodnji sliki:

C:\pemicro\ics08gpgtz\PWM\Servo motor.asm

```
        org     $fffe
        fdb     $a000   ; reset vektor

****** KRMILJENJE SERVO MOTORJA (postavitev v srednji položaj) *****
****** uporabljen TIMER 1, kanal 0   ********************************

        org     $a000
        bset    0,$1f   ;COP onemogocen
        clr     $01

        mov     #$30,$20        ;stop in reset
        mov     #$25,$20        ;stop in CLK   (:32)

        ldhx    #!1562          ;modul štetja (za T=20ms)
        sthx    $23

        ldhx    #!150           ;PWM modul štetja (za čas impulza t=2ms)
        sthx    $26
        mov     #%00011110,$25   ; unbuff. PWM, clear on compare, toggle on overflow

        bclr    5,$20           ;start TIMER 1

s1      bra     s1
```

Slika 56: Program za krmiljenje servomotorja

V rdečem pravokotniku je del programa, kjer pripravimo in zaženemo časovnik, da dobimo želeni signal za krmiljenje servomotorja. Če bi hoteli motorju spremeniti položaj, bi mu spremenili vrednost v registru kanala (T1CH0H in T1CH0L), ki določa čas trajanja impulza. Da bi dosegli stalno spreminjanje položaja motorja, bi morali tudi stalno spreminjati vrednost registra kanala.

Ponovimo

Mikrokontroler MC908GP32 ima dva enaka časovnika. Omogočata nam, da upočasnimo delovanje mikrokontrolerja oziroma izvajanja ukazov, ko le-to poteka prehitro. To dosežemo tako, da 16-bitnemu števcu časovnika ukažemo štetje impulzov do nastavljene vrednosti. To vrednost in frekvenco impulzov lahko programsko določamo. Ko števec doseže nastavljeno vrednost, nam to pokaže s postavitvijo bita v svojem registru (TOF bit v statusnem in kontrolnem registru), kar programsko tudi preverjamo. Poleg tega ima časovnik še nekaj funkcij, od katerih smo spoznali generiranje signala s pulzno-širinsko modulacijo (PWM) ter krmiljenje servomotorjev s takim signalom.

Vaje

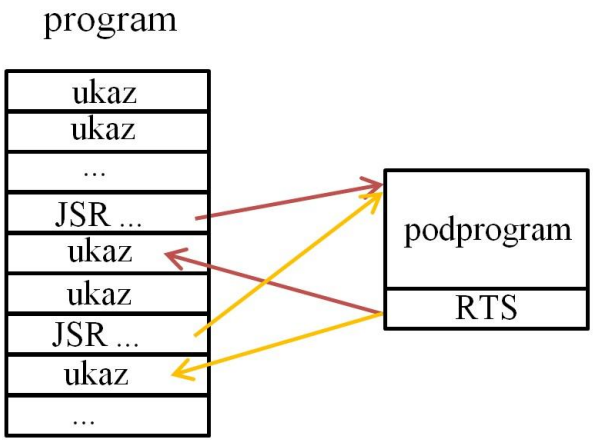
Vsak program naloži v *Flash* pomnilnik mikrokontrolerja in preveri njegovo delovanje.

1. Na portu B imamo priključene svetleče diode. Napiši program, ki bo povzročil utripanje diode na priključku PTB4. Utripa naj s frekvenco 4 Hz, za zakasnitev pa uporabi časovnik 1 (*Timer 1*).
2. Na port B mikrokontrolerja priključi 8-bitni D/A pretvornik. Napiši program, da bo na izhodu D/A pretvornika nastal izmenični pravokotni signal frekvence 100 Hz in amplitude 1 V. Natančnost parametrov signala preveri z osciloskopom.
3. S svetlečimi diodami rdeče, rumene in zelene barve sestavi maketo semaforja (samo za avtomobile) in jo priključi na port B. Napiši program, ki bo prižigal diode po podanem zaporedju in s podanimi časi.
4. Na port D priključi servomotor tako, da bo na krmilni priključek dobil PWM signal iz časovnika 2. Napiši program, ki bo stalno spreminjal položaj osi motorja od ene skrajne lege do druge in nazaj.

PODPROGRAMI, RESETIRANJE IN PREKINITVE

Podprogrami

V prejšnjem poglavju smo v primeru programa tekoče luči v obeh verzijah programa uporabili zakasnitev s časovnikom na dveh mestih v programu. Na taka ponavljanja niza ukazov lahko med programiranjem večkrat naletimo. Če take dele programa uporabimo kot podprogram, jih bomo zapisali samo enkrat, izvajanje teh ukazov pa kjerkoli v programu in tolikokrat, kot jih potrebujemo, zahtevamo z ukazom za izvajanje podprograma. Taka ukaza sta dva, `BSR` in `JSR`. Razlika med njima je samo ta, da ima ukaz `BSR` omejen doseg na 127 naslovov naprej in 128 naslovov nazaj od naslova, kjer se ta ukaz nahaja. Ta ukaz je zato uporaben le v krajših programih. Podprograme namreč ponavadi zapišemo na koncu glavnega programa. Izhod iz podprograma povzroči ukaz `RTS`.

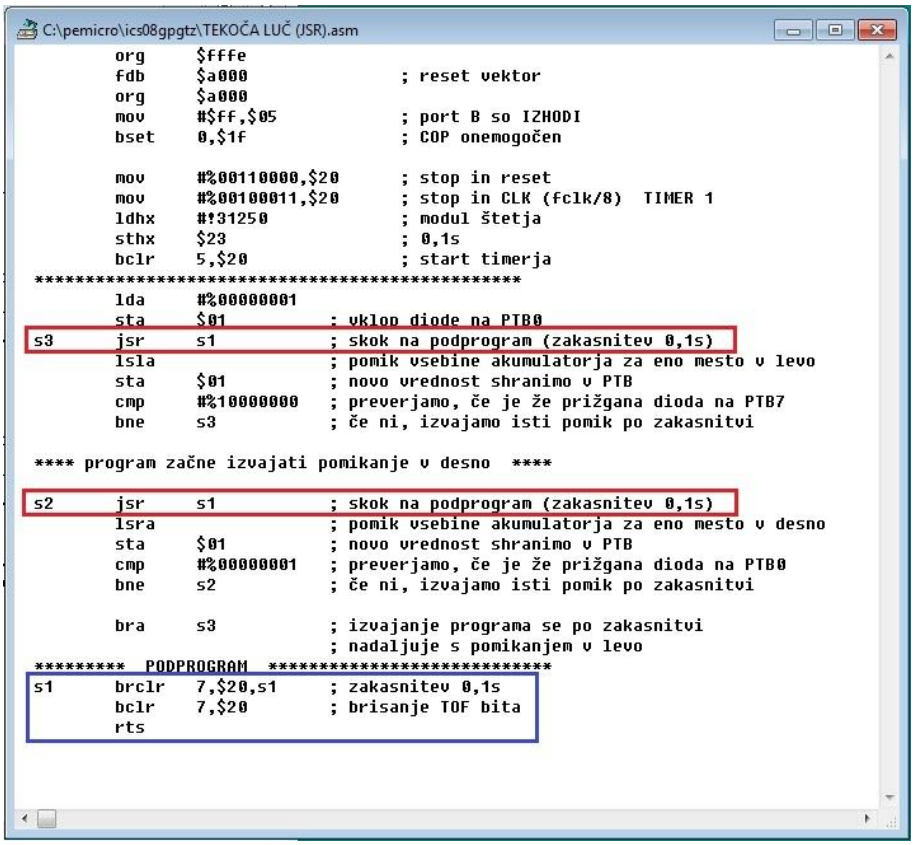


Slika 57: Izvajanje podprograma

Podprogrami pa niso samo deli programa, ki se večkrat ponavljajo. Lahko razdelimo tudi posamezne dele programa na podprograme. V takih primerih pazimo, da so to taki deli, ki opravljajo neko nalogo in predstavljajo zaključeno enoto programa. Tako razdelimo obsežnejši program na več krajših enot, kar nam olajša programiranje in iskanje napak v programu, poveča preglednost napisanega programa, posamezne enote pa lahko uporabimo tudi v drugih programih.

Preden se začne izvajanje podprograma, se v skladovni pomnilnik shrani naslov naslednjega ukaza v glavnem programu (vsebina programskega števca). Ta ukaz se bo izvedel ob vrnitvi iz podprograma v glavni program. Vsebine vseh ostalih registrov pa se ne shranijo in se lahko med izvajanjem podprograma izgubijo. Če so te vrednosti pomembne za izvajanje glavnega programa, jih moramo shraniti v sklad pred začetkom izvajanja podprograma (ukaz `PSH`). Ob vrnitvi v glavni program te vrednosti vrnemo nazaj (z ukazom `PUL`).

Pisanje in uporabo podprograma si poglejmo na primeru programa za tekočo luč iz prejšnjega poglavja. Spremenili bomo 1. verzijo programa ter namesto posameznih kontrol TOF bita v statusnem in kontrolnem registru časovnika dali ukaz za skok v podprogram:



Slika 58: Program za tekočo luč z uporabo podprograma

Podprogram je na sliki v modrem okvirju, ukaza za skok na izvajanje podprograma pa sta v rdečem okvirju.

Ker se pri izvajanju podprogramov uporablja skladovni pomnilnik (sklad), ga moramo v programu tudi definirati. Za sklad rezerviramo del bralno-pisalnega pomnilnika (*RAM*-a). Teh naslovov ne smemo uporabljati za shranjevanje drugih podatkov. Začetni naslov skladovnega pomnilnika določa vrednost v kazalcu sklada (*Stack Pointer*). Ta register ima po resetiranju vrednost $00FF, kar lahko že uporabimo za začetni naslov sklada. Če bi začetni naslov sklada radi spremenili, bi morali vanj naložiti drugo vrednost. Skladovni pomnilnik se polni od začetnega naslova proti nižjim, kar moramo upoštevati pri uporabi *RAM*-a za shranjevanje ostalih podatkov med izvajanjem programa.

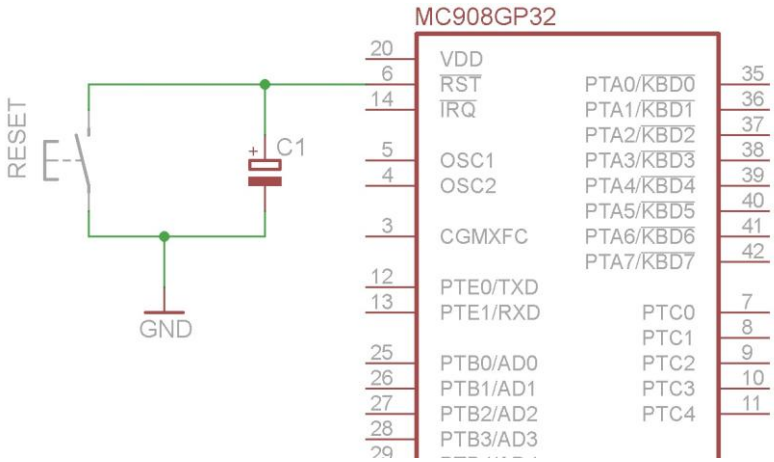
Resetiranje (ponastavitev)

Resetiranje ali ponastavitev prekine izvajanje kateregakoli ukaza in povzroči ponovno izvajanje programa od začetka. Pri tem se v programski števec vpiše dvobajtna vrednost z naslovov $FFFE

in $FFFF, kamor moramo vpisati naslov prvega ukaza programa, po katerem naj bi mikrokontroler delal. Poleg tega resetiranje povzroči postavitve nekaterih bitov v raznih nastavitvenih registrih ter nekatere registre na določeno začetno vrednost. Te začetne vrednosti so tudi podane pri opisih posameznih registrov v tem gradivu. MC908GP32 ima več izvorov resetiranja:

- zunanje resetiranje s priključkom RST
- resetiranje ob priklopu na napajalno napetost (*Power-On Reset*)
- resetiranje , ki ga povzroči modul *Computer Operating Properly – COP*
- resetiranje zaradi prenizke napajalne napetosti (*Low-Voltage Inhibit Reset*)
- neveljaven (neobstoječ) ukaz v programu
- neveljaven (neobstoječ) naslov v programu

Zunanje resetiranje dosežemo, ko se na priključku RST pojavi logično stanje 0. Ponavadi to realiziramo s tipko:



Slika 59: Priklop tipke za resetiranje na mikrokontroler

Vsi ostali izvori resetiranja so notranji, kar pomeni, da jih povzročajo nadzorni moduli mikrokontrolerja.

Prekinitve

Prekinitve začasno zaustavijo izvajanje programa, ki se trenutno izvaja, ko to zahteva določen modul mikrokontrolerja ali zunanja naprava, ki je povezana z mikrokontrolerjem. To povzroči izvajanje ustreznega prekinitvenega programa. Tako se lahko izvede na primer branje ali pošiljanje podatkov, zaznavanje pogojev delovanja in podobno. Ko prekinitveni program konča potrebno opravilo, se nadaljuje prekinjeni program tam, kjer je bil prekinjen.

Začetek programa, ki ustreza posamezni zahtevi po prekinitvi, je podan vedno na določenih naslovih. Tako lahko mikrokontroler vsakokrat najde glavni program in posamezne prekinitvene programe. Naslovom, ki podajajo začetke posameznih prekinitvenih programov, pravimo prekinitveni vektorji ("kažejo" na začetek prekinitvenega programa). Za vsak izvor prekinitvene zahteve ima mikrokontroler svoj prekinitveni vektor:

IZVOR PREKINITVE	*Zastavica*	*Maska (maskirni bit)*	*Naslov prekinitvenega vektorja*
Reset	---	---	$FFFE–$FFFF
SWI ukaz	---	---	$FFFC–$FFFD
IRQ pin	IRQF	IMASK	$FFFA–$FFFB
CGM (PLL)	PLLF	PLLIE	$FFF8–$FFF9
TIM 1 modul	CH0F	CH0IE	$FFF6–$FFF7
	CH1F	CH1IE	$FFF4–$FFF5
	TOF	TOIE	$FFF2–$FFF3
TIM 2 modul	CH0F	CH0IE	$FFF0–$FFF1
	CH1F	CH1IE	$FFEE–$FFEF
	TOF	TOIE	$FFEC–$FFED
SPI modul	SPRF	SPRIE	$FFEA–$FFEB
	OVRF	ERRIE	
	MODF	ERRIE	
	SPTE	SPTIE	$FFE8–$FFE9
SCI modul	OR	ORIE	$FFE6–$FFE7
	NF	NEIE	
	FE	FEIE	
	PE	PEIE	
	SCRF	SCRIE	$FFE4–$FFE5
	IDLE	ILIE	
	SCTE	SCTIE	$FFE2–$FFE3
	TC	TCIE	
KBI modul	KEYF	IMASKK	$FFE0–$FFE1
A/D pretvornik	COCO	AIEN	$FFDE–$FFDF
TBM modul	TBIF	TBIE	$FFDC–$FFDD

Tabela 2: Seznam prekinitev in prekinitvenih vektorjev

V tabeli imamo poleg prekinitvenih vektorjev podano še zastavico in maskirni bit za posamezno prekinitveno zahtevo. Zastavica je tisti bit, ki nam zaradi nekega dogodka v modulu s svojo spremenjeno vrednostjo pokaže, da je do tega dogodka prišlo. Z ustrezno vrednostjo maskirnega bita lahko omogočimo, da to povzroči tudi zahtevo po prekinitvi, ki zatem sproži izvajanje pripadajočega prekinitvenega programa. Če maskirnega bita ni, se prekinitvena zahteva vedno upošteva in izvede prekinitveni program. Zaradi tega delimo prekinitve na maskirne in nemaskirne. Poleg maskirnih bitov v posameznih modulih mikrokontrolerja je v registru stanj (*CCR*) še maskirni bit I (*Interrupt Mask*), s katerim preprečimo izvajanje vseh maskirnih prekinitev, če je na logični vrednosti 1.

Izvajanje prekinitev

Če so ob nastopu dogodka za prekinitev odstranjene maske za zahtevo po prekinitvi iz nekega izvora, mikrokontroler najprej dokonča izvajanje ukaza, ki se trenutno izvaja. Nato shrani vrednosti registrov centralno-procesne enote v skladovni pomnilnik. Zatem postavi I bit v registru stanj na logično 1 in s tem prepreči nadaljnje zahteve po prekinitvi. V programski števec (*PC*) naloži vrednost prekinitvenega vektorja, kar povzroči skok na začetek prekinitvenega programa in njegovo izvajanje. Konec prekinitvenega programa določimo z ukazom *RTI* (*Return*

from Interrupt). Ta ukaz povzroči nalaganje shranjene vsebine registrov iz skladovnega pomnilnika. Zadnja se naloži vrednost programskega števca. Ker se je ta vrednost pred začetkom izvajanja prekinitvenega programa spremenila tako, da je programski števec vseboval naslov naslednjega ukaza v programu, se izvajanje programa nadaljuje s tem ukazom v programu.

Pred začetkom izvajanja prekinitvenega programa mikrokontroler shranjuje vrednosti registrov CPE v tem vrstnem redu:

1. nižji bajt programskega števca
2. višji bajt programskega števca
3. nižji bajt indeksnega registra (*X*)
4. akumulator
5. register stanj

Ob izhodu iz prekinitvenega programa se vračanje teh vrednosti nazaj v registre izvaja v obratnem vrstnem redu, od registra stanj do programskega števca.

Primer uporabe prekinitve

Na preprostem primeru si bomo ogledali potrebne nastavitve za uporabo prekinitve z modula KBI. Ta modul si deli zunanje priključke s portom A. Z njim lahko povzročimo zahtevo po prekinitvi, ko se na vhodnem priključku pojavi logična 0. Da bi se to zgodilo, moramo omogočiti maskirne prekinitve s postavitvijo I bita v registru stanj na logično 0, v registru *INTKBIER* (*Keyboard Interrupt Enable Register*) pa s postavitvijo ustreznega bita na logično 1 omogočiti zahtevo po prekinitvi z željenega priključka na portu A:

Naslov: $001B

	Bit 7	6	5	4	3	2	1	Bit 0
	KBIE7	KBIE6	KBIE5	KBIE4	KBIE3	KBIE2	KBIE1	KBIE0
Reset:	0	0	0	0	0	0	0	0

Slika 60: Register INTKBIER

S to nastavitvijo se na uporabljenih vhodih vklopijo tudi pull-up upori. Poleg tega registra ima KBI modul še statusni in kontrolni register (*Keyboard Status and Control Register – INTKBSCR*):

Naslov: $001A

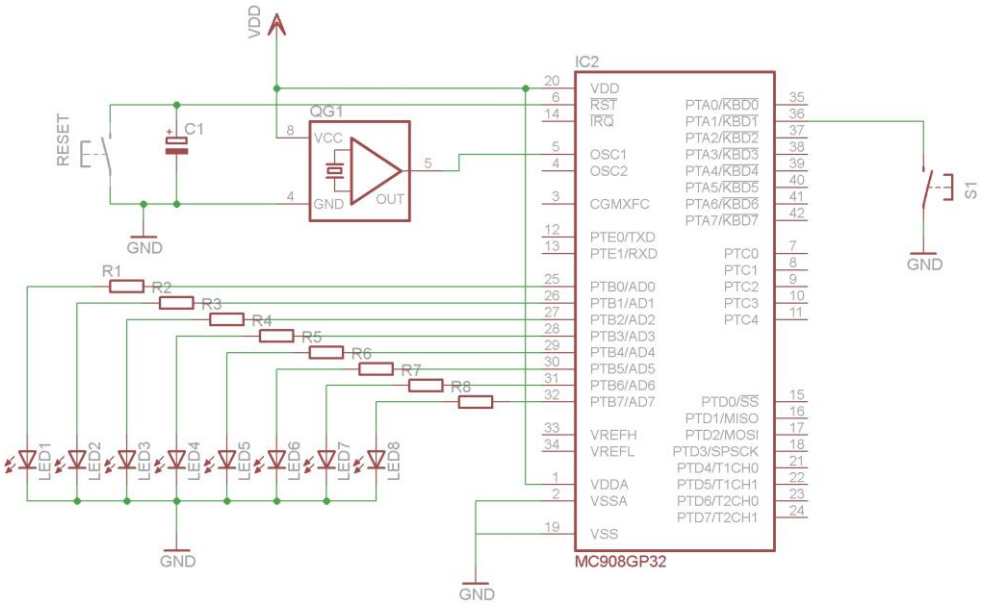
	Bit 7	6	5	4	3	2	1	Bit 0
Branje:	0	0	0	0	KEYF	0	IMASKK	MODEK
Pisanje:						ACKK		
Reset:	0	0	0	0	0	0	0	0

Slika 61: Register INTKBSCR

V njem je bit IMASKK, ki maskira (onemogoči) zahtevo po prekinitvi iz KBI modula, če je na logični 1. Ker se ta bit po resetiranju postavi na logično 0, ga ni potrebno spreminjati.

V programu moramo še poskrbeti za vpis naslova začetka prekinitvenega programa na naslov $FFE0 in $FFE1 (KBI prekinitveni vektor) ter napisati prekinitveni program.

Za primer bomo uporabili vezavo mikrokontrolerja s svetlečimi diodami in tipko, s katero bomo povzročili izvajanje prekinitvenega programa:



Slika 62: El. shema vezja z mikrokontrolerjem za primer uporabe prekinitve

V glavnem programu se bodo prižigale svetleče diode v zaporedju od LED1 do LED8 in nazaj (tekoča luč), s pritiskom na tipko S1 pa bomo povzročili, da bodo vse tipke utripale. Po treh periodah utripanja se bo prekinitveni program končal in nadaljevalo se bo izvajanje tekoče luči. Program za realizacijo tekoče luči s podprogramom imamo že izdelan. Sedaj ga bomo samo še dopolnili s prekinitvenim programom in potrebnimi nastavitvami:

```
**    PTA1 povzroči prekinitev (KBI zahteva)  ***

     org     $fffe
     fdb     $9000     ; reset vektor
     org     $ffe0
     fdb     $A000     ; KBIE vektor
*********************************************
     org     $9000
     mov     #$ff,$05  ; port B so IZHODI
     bset    1,$1B     ; omogočena KBI prekinitev na tipki PTA1
     bset    0,$1F     ; COP onemogočen
*********************************************
     mov     #%00110000,$20      ; stop in reset
     mov     #%00100011,$20      ; stop in CLK (fclk/8)  TIMER 1
     ldhx    #!31250             ; modul štetja
```

```
      sthx    $23                ; 0,1s
      bclr    5,$20              ;start časovnika 1

      cli                ; omogočimo maskirne prekinitve

      lda     #%00000001
      sta     $01          ; vklop diode na PTB0
s3    jsr     s1           ; skok na podprogram (zakasnitev 0,1s)
      lsla                 ; pomik vsebine akumulatorja za eno mesto v levo
      sta     $01          ; novo vrednost shranimo v PTB
      cmp     #%10000000   ; preverjamo, če je že prižgana dioda na PTB7
      bne     s3           ; če ni, izvajamo isti pomik po zakasnitvi

**** program začne izvajati pomikanje v desno  ****

s2    jsr     s1          ; skok na podprogram (zakasnitev 0,1s)
      lsra                ; pomik vsebine akumulatorja za eno mesto v desno
      sta     $01          ; novo vrednost shranimo v PTB
      cmp     #%00000001   ; preverjamo, če je že prižgana dioda na PTB0
      bne     s2           ; če ni, izvajamo isti pomik po zakasnitvi

      bra     s3           ; izvajanje programa se po zakasnitvi
                           ; nadaljuje s pomikanjem v levo

*********  PODPROGRAM  ***************************
s1    brclr   7,$20,s1     ; zakasnitev 0,1s
      bclr    7,$20        ; brisanje TOF bita
      rts                  ; konec podprograma

*****     IRQ program    ****
      org     $A000      ; začetni naslov prekinitvenega programa
      lda     #!6        ; števec utripanja LED (3x ugasnjene, 3x prižgane)
      clr     $01        ; ugasnemo vse LED
s4    jsr     s1         ; skok na podprogram (zakasnitev 0,1s)
      com     $01        ; negacija stanja v PTB (sprememba stanja LED)
      deca               ; zmanjša za 1 števec utripanja
      bne     s4         ; če števec utripanja ni 0, nadaljuje z utripi
      rti                ; konec prekinitvenega programa
```

Začetek prekinitvenega programa je postavljen na naslov $A000, kar je tudi zapisano v KBI prekinitvenem vektorju. Podprogram za zakasnitev smo uporabili tudi v prekinitvenem programu.

Ponovimo

Podprograme lahko uporabljamo, da razdelimo daljši program na več manjših enot, ali da niz ukazov, ki jih v programu večkrat uporabimo, zapišemo enkrat, njihovo izvajanje pa z ukazom za skok na ta podprogram izvajamo, kjer je v programu to potrebno.

Uporaba prekinitev nam olajša programiranje, saj nam ni potrebno stalno preverjati, ali je do dogodka, ki ga pričakujemo, že prišlo ali ne. Posamezni moduli mikrokontrolerja nam namreč to javijo z zahtevo po prekinitvi izvajanja programa. Takrat se začne izvajati prekinitveni program. Ko se zaključi, se nadaljuje izvajanje ukazov v glavnem programu tam, kjer se je prekinilo. Da lahko prekinitve uporabljamo, moramo opraviti ustrezne nastavitve bitov v registrih modulov, ki bodo postavljali zahteve po prekinitvah.

Tudi resetiranju (ponastavitvi) mikrokontrolerja lahko rečemo prekinitev, saj se takrat brezpogojno preneha izvajanje programa. Le-to povzroči ponovno izvajanje programa od začetka, poleg tega postavi v začetno stanje tudi nekatere registre in bite v registrih.

Vprašanja in vaje

1. Opiši dogajanje v mikrokontrolerju po resetiranju.
2. Kaj je skladovni pomnilnik in čemu služi?
3. Kako v programu uporabimo podprogram? Kako se konča izvajanje podprograma?
4. Kaj moramo narediti, da omogočimo zahtevo po prekinitvi iz KBI modula ob pritisku na tipko, priključeno na PTA3?
5. Kaj moramo narediti, da omogočimo zahtevo po prekinitvi iz časovnika 1 ob preteku nastavljenega časa?
6. Program za krmiljenje semaforja iz 3. vaje prejšnjega poglavja spremeni tako, da bo zakasnitev realizirana s podprogramom.

ANALOGNO-DIGITALNI PRETVORNIK

Analogno-digitalni pretvornik potrebujemo, ko hočemo z mikrokontrolerjem nadzirati fizikalne veličine, kot so temperatura, vlaga, svetloba, razdalja, sila in druge. Zato rabimo senzorje, ki nam pretvorijo vrednost fizikalne veličine v električno napetost ali tok. Le-ta se s spreminjanjem veličine zvezno spreminja. Takih signalov ne moremo pripeljati na binarne vhode mikrokontrolerja. Zato uporabimo pretvornik, ki različne nivoje napetosti pretvori v večbitno vrednost. To vrednost uporabimo v mikrokontrolerju za prikaz vrednosti ali izvajanje regulacije in krmiljenja naprav glede na kontrolirano veličino.

Priključki PTB7/AD7–PTB0/AD0 so lahko vhodi 8-kanalnega 8-bitnega A/D pretvornika. Ko jih določimo za uporabo z A/D pretvornikom, na te priključke ne vplivajo nastavitve smernega registra porta B (vhod/ izhod). Vhodna analogna napetost se pretvori v 8-bitno vrednost. Če je vhodna napetost enaka napetosti na priključku VSSAD/VREFL, je pretvorjena vrednost enaka $00. Če je vhodna napetost enaka napetosti na priključku VDDAD/VREFH, je pretvorjena vrednost enaka $FF. Vhodna napetost ne sme biti višja od napetosti na priključku VDDAD/VREFH ali nižja od napetosti na priključku VSSAD/VREFL.

V nadaljevanju bomo spoznali, kako pripravimo A/D pretvornik, da bo lahko opravil pretvorbo vhodne analogne napetosti.

Registri pretvornika

Modul A/D pretvornika ima naslednje registre:

- statusni in kontrolni register A/D pretvornika (*ADC Status and Control Register* – ADSCR)
- podatkovni register A/D pretvornika (*ADC Data Register* – ADR)
- register taktnega signala A/D pretvornika (*ADC Clock Register* – ADCLK)

Statusni in kontrolni register

Nahaja se na naslovu $003C. Omogoča nam izbiro vhoda za A/D pretvorbo, uporabo prekinitve ter izbiro stalnega pretvarjanja ali ene pretvorbe.

Naslov: $003C

	Bit 7	6	5	4	3	2	1	Bit 0
	COCO	AIEN	ADCO	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0
Reset:	0	0	0	1	1	1	1	1

Slika 63: Statusni in kontrolni register A/D pretvornika

Bit COCO (*Conversions Complete*) nam v načinu posameznih pretvorb s postavitvijo v logično stanje 1 pove, da se je zahtevana pretvorba izvršila.

Z bitom AIEN (*ADC Interrupt Enable Bit*) lahko omogočimo zahtevo po prekinitvi, ki jo povzroči A/D pretvornik, ko se konča pretvorba. To storimo s postavitvijo tega bita na logično 1. Zahtevo po prekinitvi umaknemo oziroma izničimo, ko z ukazom preberemo vrednost podatkovnega registra pretvornika ali s spreminjanjem nastavitev v njegovem statusnem in kontrolnem registru.

Bit ADCO (ADC Continuous Conversion Bit) določa:
- log. 1 stalna pretvorba
- log. 0 ena pretvorba

Z biti ADCH4 do ADCH0 (*ADC Channel Select Bits*) izbiramo vhodni priključek za A/D pretvorbo:

ADCH4	ADCH3	ADCH2	ADCH1	ADCH0	IZBRANI VHOD
0	0	0	0	0	PTB0/AD0
0	0	0	0	1	PTB1/AD1
0	0	0	1	0	PTB2/AD2
0	0	0	1	1	PTB3/AD3
0	0	1	0	0	PTB4/AD4
0	0	1	0	1	PTB5/AD5
0	0	1	1	0	PTB6/AD6
0	0	1	1	1	PTB7/AD7
0 ↓ 1	1 ↓ 1	0 ↓ 1	0 ↓ 0	0 ↓ 0	Rezervirano
1	1	1	0	1	V_{REFH}
1	1	1	1	0	V_{REFL}
1	1	1	1	1	ADC izklopljen

Tabela 3: Izbira vhodnega priključka A/D pretvornika

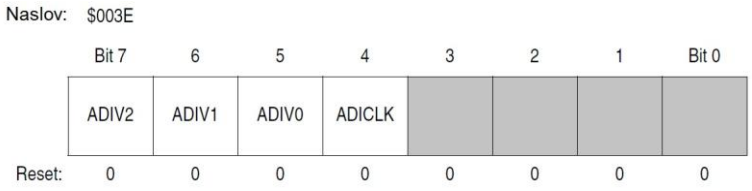
S preklopom na pretvorbo referenčnih napetosti (priključka VREFH in VREFL) lahko preverjamo delovanje A/D pretvornika. Zadnja kombinacija v tabeli izklopi A/D pretvornik. Na ta način lahko zmanjšamo porabo mikrokontrolerja, ko pretvornika ne potrebujemo.

Podatkovni register

Nahaja se na naslovu $003D. Po vsaki končani pretvorbi se v njem nahaja 8-bitna pretvorjena vrednost, ki se ohrani do konca naslednje pretvorbe.

Register taktnega signala

Nahaja se na naslovu $003E. Z njim izberemo izvor taktnega signala (*ADC Clock*) ter delilno razmerje za signal iz izbranega izvora.



Slika 64: Register taktnega signala A/D pretvornika

Z bitom ADICLK (*ADC Input Clock Select Bit*) določimo izvor taktnega signala za A/D pretvornik. Logična vrednost 1 določa, da je to notranji taktni signal mikrokontrolerja.

Biti ADIV2 – ADIV20 (*ADC Clock Prescaler Bits*) določajo delilno razmerje izbranega taktnega signala. Proizvajalec priporoča izbor takega delilnega razmerja, da je frekvenca taktnega signala A/D pretvornika okrog 1 MHz.

ADIV2	ADIV1	ADIV0	DELILNO RAZMERJE
0	0	0	frekvenca izvora :1
0	0	1	frekvenca izvora :2
0	1	0	frekvenca izvora :4
0	1	1	frekvenca izvora :8
1	X	X	frekvenca izvora :16

Tabela 4: Izbira delilnega razmerja za taktni signal pretvornika

Uporaba pretvornika

Preden začnemo uporabljati A/D pretvornik, moramo njegovo delovanje nastaviti v registru taktnega signala (ADCLK) ter statusnem in kontrolnem registru (ADSCR). V registru taktnega signala izberemo izvor taktnega signala ter njegovo delilno razmerje. Nato v statusnem in kontrolnem registru določimo način delovanja pretvornika.

Lahko ga uporabljamo samo za posamezno pretvorbo priključene analogne napetosti. V tem primeru mu vsakokrat, ko želimo pretvorbo, pošljemo ukaz, ki vsebuje informacijo o izbranem vhodnem priključku za pretvorbo, ter vrednost ADCO bita na logični 0. Tak način delovanja izberemo predvsem takrat, ko uporabljamo več kanalov pretvornika, torej pri priključitvi več senzorjev na vhode mikrokontrolerja. Če uporabimo samo en analogni vhod, lahko nastavimo stalno pretvorbo na izbranem vhodu.

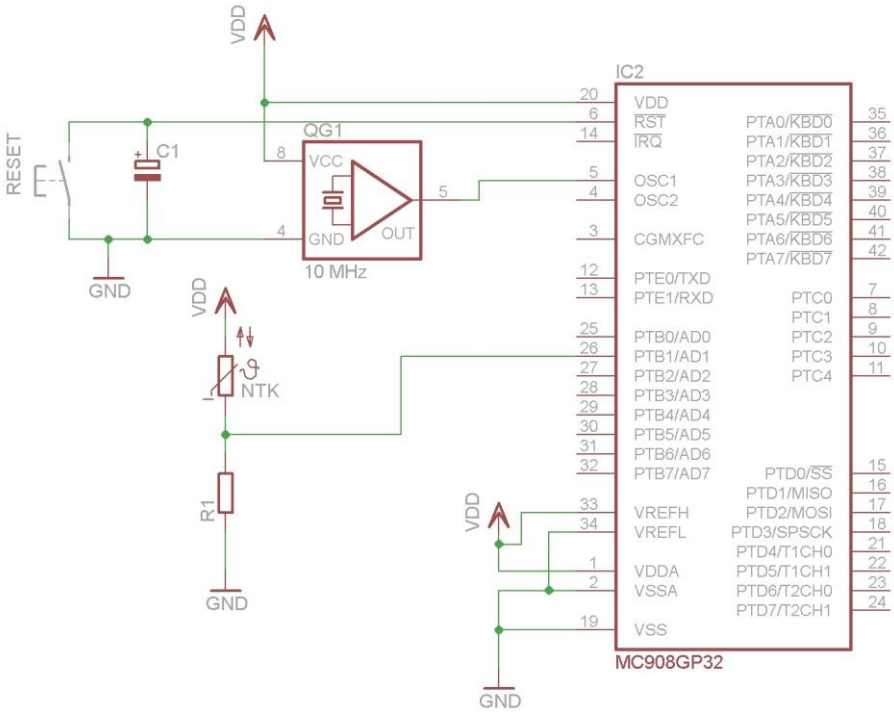
Pri obeh načinih delovanja oziroma uporabe lahko omogočimo tudi zahtevo po prekinitvi, ki nastane ob koncu pretvorbe.

Uporaba senzorjev

Najpogostejša uporaba A/D pretvornika je povezana s senzorji. Z njimi namreč nadzorujemo vrednosti fizikalnih veličin, kot so temperatura, vlaga, svetloba, sila, razdalja, hitrost in druge. Na analogni vhod mikrokontrolerja lahko priključimo pasivne ali aktivne senzorje. Pasivnim se upornost spreminja s spreminjanjem fizikalne veličine, aktivni pa na izhodu ustvarjajo napetost, ki je odvisna od opazovane fizikalne veličine. Najprej si bomo ogledali priključitev in nastavitve A/D pretvornika z enim pasivnim senzorjem.

Za merjenje npr. temperature poznamo NTK in PTK upore, upornost fotoupora je odvisna od jakosti svetlobe itd. Take senzorje lahko preprosto uporabimo v kombinaciji s stalnim (nespremenljivim) uporom, s katerim izdelamo delilnik napetosti. Vendar imajo taki senzorji ponavadi nelinearno karakteristiko, kar nam lahko oteži uporabo oziroma programiranje.

Če želimo na primer prikazovati temperaturo v prostoru, se napetost na analognem vhodu mikrokontrolerja ne bo linearno spreminjala s spreminjanjem temperature v prostoru. To pomeni, da vrednost temperature ne moremo preprosto izračunati iz pretvorjene vrednosti vhodne napetosti. V tem primeru si pomagamo s tabelo pretvorbe (*Look-up table*), kjer za vsako binarno vrednost pretvorbe poiščemo ustrezno temperaturo in jo izpišemo. Priključitev takega senzorja vidimo na spodnji shemi:



Slika 65: El. shema vezja z mikrokontrolerjem in NTK uporom

Pripravo A/D pretvornika s stalnim pretvarjanjem na priključku PTB1/AD1, kjer je senzor priključen, bi izvedli tako:

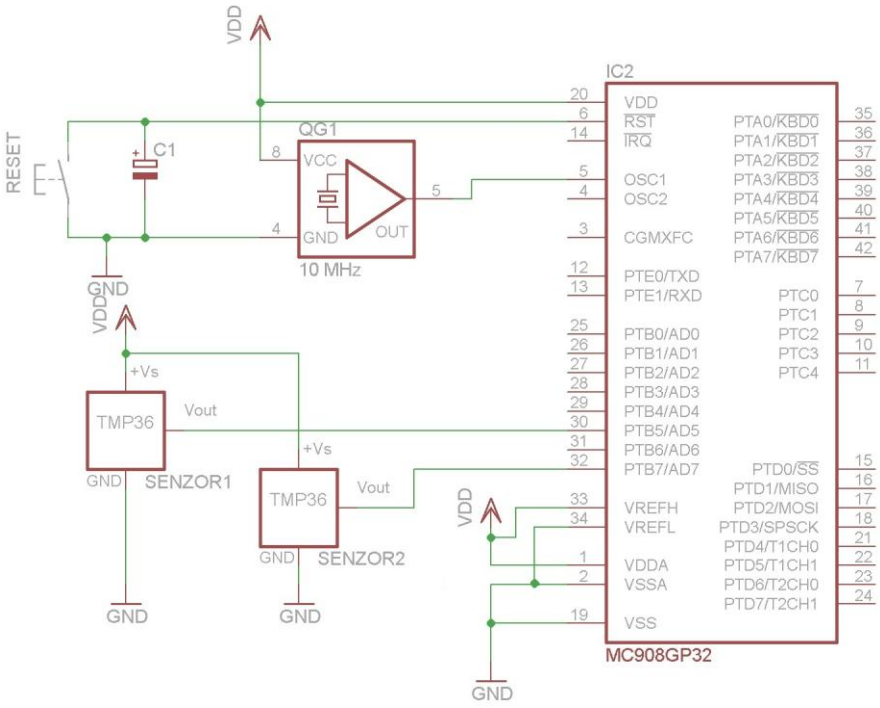
```
mov     #%00110000,$3E          ; frekvenca CLK za ADC (fclk :2)
mov     #%00100001,$3C          ; stalna pretvorba na PTB1/AD1
```

Med izvajanjem programa bi informacijo o trenutni temperaturi prebrali z enim ukazom v podatkovnem registru vsakokrat, ko bi jo potrebovali:

```
lda     $3D       ; v akumulator naložimo zadnjo pretvorjeno vrednost
```

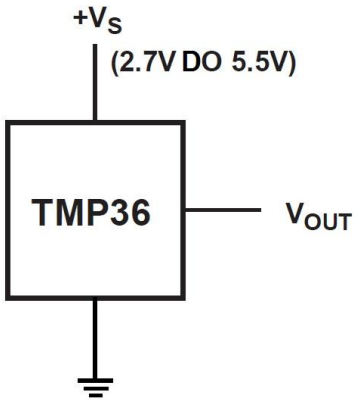
To vrednost nato lahko uporabimo za prikaz temperature na izbranem prikazovalniku.

Poglejmo si še primer, ko uporabimo več senzorjev. Za razliko od prejšnjega primera bomo sedaj uporabili aktivni senzor za merjenje temperature. Spodnja shema prikazuje priključitev dveh senzorjev TMP36 proizvajalca *Analog Devices, Inc.* iz ZDA. Priključena sta na vhoda PTB5/AD5 (SENZOR1) in PTB7/AD7 (SENZOR2).

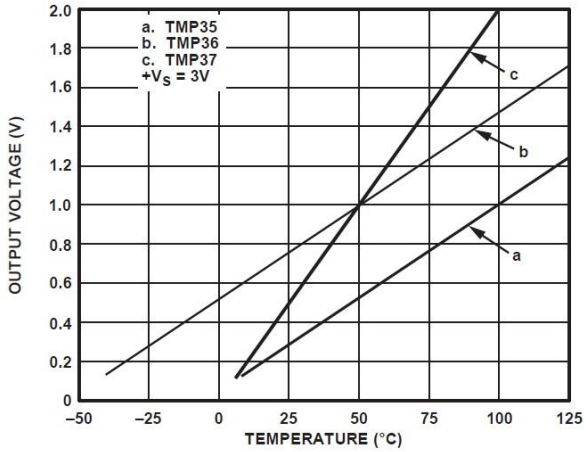


Slika 66: El. shema vezja z mikrokontrolerjem in senzorjem temperature TMP 36

Ti senzorji na izhodu dajejo enosmerno napetost, ki se linearno spreminja s spreminjanjem temperature. Izhodna napetost se spreminja 10 mV/°C, uporabni pa so v temperaturnem območju od –40 °C do +125 °C:



Slika 67: Priključki senzorja TMP 36



Slika 68: Graf $U_{izh}(T)$ za senzor TMP 36
(vir: http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf)

A/D pretvorniku bi najprej določili izvor taktnega signala in frekvenco oziroma delilno razmerje zanjo:

```
    mov     #%00110000,$3E        ; frekvenca CLK za ADC (fclk :2)
```

Nato bi med izvajanjem programa informacijo o izmerjeni trenutni temperaturi posameznega senzorja dobili tako, da bi najprej zahtevali začetek pretvorbe na enem vhodu, po končani pretvorbi pa bi v podatkovnem registru A/D pretvornika dobili izmerjeno vrednost. Isto bi ponovili tudi za drugi analogni vhod in senzor:

```
    mov    #%00000101,$3C        ; ena pretvorba na PTB5/AD5
s1  brclr  7,$3C,s1    ; čakamo konec pretvorbe (bit COCO se postavi na 1)
    lda    $3D         ; v akumulator naložimo pretvorjeno vrednost
       ...
    mov     #%00000111,$3C       ; ena pretvorba na PTB7/AD7
s2  brclr   7,$3C,s2  ; čakamo konec pretvorbe (bit COCO se postavi na 1)
    lda     $3D         ; v akumulator naložimo pretvorjeno vrednost
```

Vsako pretvorjeno vrednost, ki jo preberemo v podatkovnem registru A/D pretvornika, zatem lahko prikažemo na izbranem (in priključenem) prikazovalniku. V nadaljevanju si bomo pogledali zahtevnejši projekt, in sicer izdelavo termometra z uporabo NTK upora, za prikaz temperature pa bomo uporabili prikazovalnik s tekočimi kristali – LCD prikazovalnik.

Termometer z LCD prikazovalnikom

Uporabili bomo NTK upor z nazivno vrednostjo 10 kΩ. Tako vrednost ima upor pri 25 °C. Spodnja tabela podaja v zadnjem stolpcu vrednosti upora pri različnih temperaturah:

RESISTANCE VALUES AT INTERMEDIATE TEMPERATURES

T_{oper} (°C)	R_T/R_{25}	ΔR DUE TO B-TOLERANCE (%)	TC (%/K)	R_{25} (kΩ)					
				6.222	6.272	6.332	6.472	6.682	6.103
−40	33.21	2.66	6.57	73.06	89.67	109.6	156.1	225.8	332.1
−35	23.99	2.41	6.36	52.78	64.77	79.17	112.8	163.1	240.0
−30	17.52	2.17	6.15	38.55	47.31	57.82	82.35	119.1	175.2
−25	12.93	1.94	5.95	28.44	34.91	42.67	60.77	87.92	129.3
−20	9.636	1.71	5.76	21.20	26.02	31.80	45.30	65.53	96.36
−15	7.250	1.50	5.58	15.95	19.58	23.93	34.08	49.30	72.50
−10	5.505	1.29	5.40	12.11	14.86	18.16	25.87	37.43	55.05
0	3.255	0.89	5.08	7.162	8.790	10.74	15.30	22.14	32.56
5	2.534	0.70	4.92	5.575	6.842	8.362	11.91	17.23	25.34
10	1.987	0.52	4.78	4.372	5.366	6.558	9.340	13.51	19.87
15	1.570	0.34	4.64	3.454	4.239	5.181	7.378	10.67	15.70
20	1.249	0.17	4.50	2.747	3.372	4.121	5.869	8.492	12.49
25	1.000	0.00	4.37	2.200	2.700	3.300	4.700	6.800	10.00
30	0.8059	0.16	4.25	1.773	2.176	2.660	3.788	5.480	8.059
35	0.6535	0.32	4.13	1.438	1.764	2.156	3.072	4.444	6.535
40	0.5330	0.47	4.02	1.173	1.439	1.759	2.505	3.624	5.330
45	0.4372	0.62	3.91	0.9618	1.180	1.443	2.055	2.972	4.372
50	0.3605	0.77	3.80	0.7932	0.973	1.190	1.694	2.451	3.606
55	0.2989	0.91	3.70	0.6575	0.807	0.9863	1.405	2.032	2.989
60	0.2490	1.05	3.60	0.5478	0.672	0.8217	1.170	1.693	2.490
65	0.2084	1.18	3.51	0.4586	0.562	0.6879	0.9797	1.417	2.084
70	0.1753	1.31	3.42	0.3857	0.473	0.5785	0.8239	1.192	1.753
75	0.1481	1.44	3.33	0.3258	0.399	0.4887	0.6960	1.007	1.481
80	0.1256	1.57	3.25	0.2764	0.339	0.4146	0.5905	0.8544	1.256
85	0.1070	1.69	3.16	0.2355	0.289	0.3532	0.5031	0.7278	1.070
90	0.09154	1.81	3.09	0.2014	0.247	0.3021	0.4303	0.6225	0.9154
95	0.07860	1.93	3.01	0.1729	0.212	0.2594	0.3694	0.5345	0.7860
100	0.06773	2.04	2.94	0.1490	0.182	0.2235	0.3183	0.4607	0.6773
105	0.05858	2.15	2.87	0.1289	0.158	0.1933	0.2753	0.3983	0.5858
110	0.05083	2.26	2.80	0.1118	0.137	0.1677	0.2389	0.3457	0.5083
115	0.04426	2.37	2.73	0.0974	0.1195	0.1461	0.2080	0.3010	0.4426
120	0.03866	2.47	2.67	0.0851	0.1044	0.1276	0.1817	0.2629	0.3866
125	0.03387	2.57	2.61	0.0745	0.0915	0.1118	0.1592	0.2303	0.3387
130	0.02977	2.67	2.55	0.0655	0.0804	0.0982	0.1399	0.2024	0.2977
135	0.02624	2.77	2.49	0.0577	0.0709	0.0866	0.1233	0.1784	0.2624
140	0.02319	2.86	2.43	0.0510	0.0626	0.0765	0.1090	0.1577	0.2319
145	0.02055	2.96	2.38	0.0452	0.0555	0.0678	0.0966	0.1398	0.2055
150	0.01826	3.05	2.33	0.0402	0.0493	0.0603	0.0858	0.1242	0.1826

Tabela 5: Upornost NTK upora pri različnih temperaturah
(vir: Vishay BCcomponents: NTC Thermistors, Accuracy Line, pdf datoteka proizvajalca *Vishay Intertechnology, Inc.*, Document Number: 91000, Revision: 08-Apr-05)

Za prikaz temperature bomo uporabili znakovni LCD prikazovalnik z dvema vrsticama, v vsaki po 16 znakov. Take prikazovalnike dobimo kot module, ki jih upravlja za tak namen izdelan mikrokontroler (npr. HD44780U). Le-ta sprejema ukaze in ASCII kode znakov, ki jih hočemo izpisati na prikazovalniku.

Slika 69: Priključki modula LCD prikazovalnika

Ti moduli so lahko eno-, dvo- ali štirivrstični (od 1x8 znakov do 4x40 znakov) s standardiziranimi priključki in načinom komunikacije z mikrokontrolerji. Imajo 8 podatkovnih priključkov (D7 – D0), tri kontrolne priključke (E, RS in R/W), vhod za nastavitev kontrasta ter napajalna priključka. Večina jih ima še priključka za osvetlitev ozadja prikazovalnika, ki je izvedeno s svetlečimi diodami. Podatkovni priključki omogočajo prenos ukazov in kod znakov med mikrokontrolerjem in LCD modulom. Priključek R/W določa, kdaj modulu pošiljamo podatke in kdaj jih z njega prebiramo. RS priključek omogoča izbiro med registroma, ki sprejemata ukaze in kode znakov, ob ustreznem prehodu logičnega stanja na priključku E pa modul prebere poslani podatek s podatkovnih priključkov.

LCD modul ima naslednje registre:

- podatkovni register (*DR – Data Register*),
- ukazni register (*IR – Instruction Register*),
- DDRAM (*Display Data RAM*),
- CGROM (*Character Generator ROM*),
- CGRAM (*Character Generator RAM*),
- naslovni števec (*AC – Address Counter*).

Podatkovni register je 8-bitni register, ki sprejme kode znakov ter jih posreduje v DDRAM ali CGRAM. Ukazni register sprejema ukaze, ki določajo način delovanja LCD modula, prikaz znakov, pomikanje napisa in podobno. Spodnja tabela prikazuje ukaze za delo z LCD modulom:

UKAZ	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	OPIS UKAZA	ČAS IZVAJANJA UKAZA (µs)
Brisanje zaslona	0	0	0	0	0	0	0	0	0	1	zbriše napis na zaslonu, postavi naslov za DDRAM v AC-ju na 0	1,52 ms
Vračanje v izhodišče	0	0	0	0	0	0	0	0	1	X	postavi naslov za DDRAM na 0, izniči morebitni pomik napisa	1,52 ms
Način vpisovanja	0	0	0	0	0		0	1	I/D	S	določi smer pomika kurzorja in pomik napisa	37 µs
Vklop/ izklop zaslona	0	0	0	0	0	0	1	D	C	B	povzroči vklop ali izklop zaslona ter utripanje znaka na mestu kurzorja	37 µs
Pomik kurzorja	0	0	0	0	0	1	S/C	R/L	X	X	povzroči pomik kurzorja in napisa na zaslonu	37 µs
Funkcijske nastavitve	0	0	0	0	1	DL	N	F	X	X	določi 4- ali 8-bitni način komuniciranja, število vrstic in višino znakov na zaslonu	37 µs
Naslov CGRAM-a	0	0	0	1	CGRAM NASLOV						določi naslov v CGRAM-u za vpis znaka	37 µs
Naslov DDRAM-a	0	0	1	DDRAM NASLOV							določi naslov v DDRAM-u za vpis znaka	37 µs
Vpis znakov	1	0	PODATEK (KODA ZNAKA)								vpiše kodo znaka v DDRAM ali CGRAM	37 µs

Tabela 6: Ukazi za delo z LCD prikazovalnikom

Pomen bitov v ukazih je razviden iz spodnje tabele:

OZNAKA BITA	POMEN	
I/D	0 = pomik kurzorja nazaj	1 = pomik kurzorja naprej
S	0 = ni pomika napisa	1 = pomik napisa
D	0 = izklop zaslona (ni napisa)	1 = vklop zaslona (napisa)
C	0 = brez kurzorja	1 = prikaz kurzorja
B	0 = ni utripanja na mestu kurzorja	1 = utripanje na mestu kurzorja
S/C	0 = pomik kurzorja	1 = pomik napisa
R/L	0 = pomik levo	1 = pomik desno
DL	0 = 4-bitna komunikacija	1 = 8-bitna komunikacija
N	0 = 1 vrstica	1 = 2 vrstici
F	0 = velikost znaka 5x7 pik	1 = velikost znaka 5x10 pik

Tabela 7: Pomen bitov v ukazih za LCD prikazovalnik

Kode, ki so vpisane v DDRAM-u, se s pomočjo registrov CGROM in CGRAM izpišejo na zaslonu. Ta register ima na voljo 40 lokacij za vsako vrstico zaslona.

	1	2	3	4	5		39	40
DDRAM naslovi (šestnajstiško)	00	01	02	03	04	· · · · · · · · · · · · · · · · · ·	26	27
	40	41	42	43	44	· · · · · · · · · · · · · · · · · ·	66	67

Slika 70: Naslovi pomnilnika DDRAM LCD prikazovalnika

Če ima zaslon LCD modula za prikaz znakov npr. 16 mest v dveh vrsticah, se na ta mesta izpišejo znaki, katerih kode se nahajajo na naslovih od $00 do $0F za 1. vrstico ter na naslovih od $40 do $4F za 2. vrstico. Prikaz znakov z ostalih naslovov dosežemo s pošiljanjem ukazov za pomik napisa. Tak ukaz ne spreminja kod na naslovih v DDRAM-u, ampak samo spremeni začetni naslov za izpisovanje šestnajstih znakov v obeh vrsticah. Tako bomo po ukazu za pomik napisa v levo na zaslonu videli znake, ki so shranjeni na naslovih od $01 do $10 za 1. vrstico ter na naslovih od $41 do $50 za 2. vrstico.

Register CGROM omogoča prikaz znakov na zaslonu tako, da ASCII kode pretvori v podatke, ki prikažejo ustrezen znak. Z registrom CGRAM lahko naredimo do 8 lastnih znakov. Le-ti imajo kode od $00 do $07. Tabela z ASCII kodami znakov in pripadajočimi znaki je v prilogi na koncu gradiva.

Naslovni register skrbi za zaporedno vpisovanje znakov oziroma njihovih kod na naslove v DDRAM-u in CGRAM-u. Pri tem se lahko naslovi povečujejo ali zmanjšujejo, kar določimo z ustreznim ukazom.

LCD module lahko uporabljamo v 4-bitnem ali 8-bitnem načinu komunikacije z mikrokontrolerji. Za naš primer bomo uporabili 4-bitno povezavo. Pri takem načinu ne povezujemo podatkovnih priključkov od D3 do D0. Vsak ukaz ali kodo znaka moramo poslati v dveh delih, najprej višje štiri bite, nato še nižje štiri. Pri pošiljanju podatkov postavimo priključek E na LCD modulu na logično 1. Ko pošljemo posamezni del podatka, postavimo priključek E na logično 0. Takrat mikrokontroler LCD modula prebere stanje na podatkovnih priključkih. Glede na stanje na RS priključku bo ta podatek shranil v podatkovni ali ukazni register.

Preden začnemo LCD prikazovalnik uporabljati, ga moramo pripraviti. Mikrokontrolerju, ki je na modulu prikazovalnika, moramo prenesti podatke o načinu komunikacije z našim mikrokontrolerjem, številu vrstic, ki jih ima zaslon, višini znakov, uporabi kurzorja, smeri vpisovanja znakov in drugo. Primer priprave LCD modula pri 4-bitnem načinu povezave in komunikacije je podan spodaj:

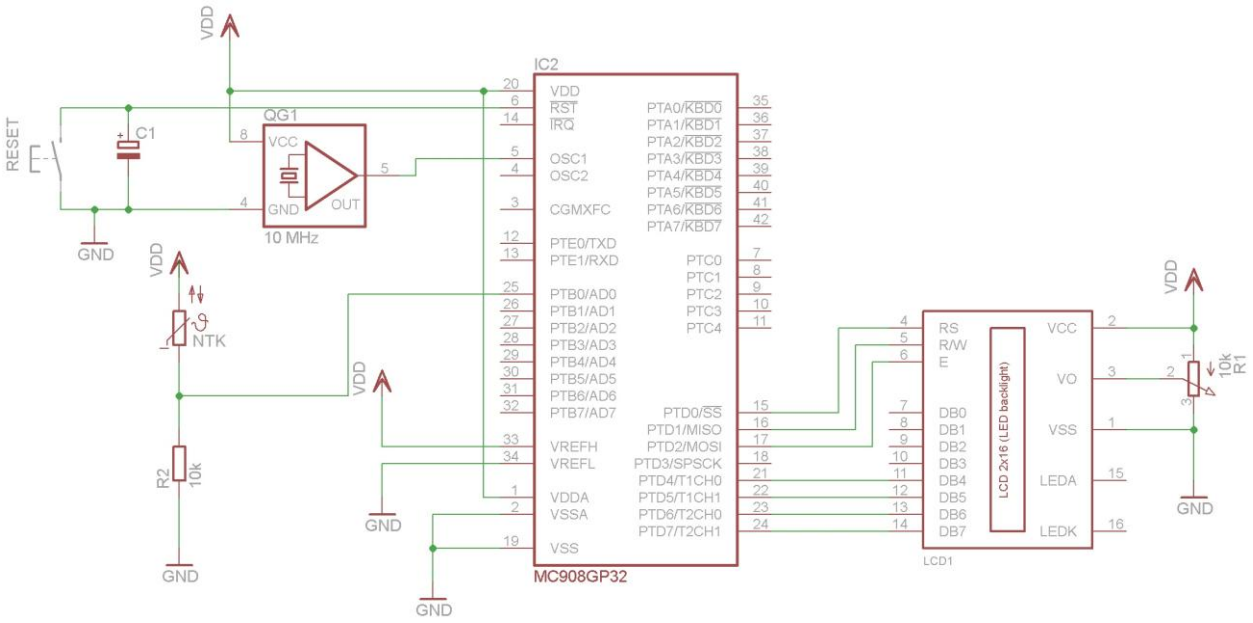
```
*** PRIPRAVA LCD (D7 – D4->PTD7 – PTD4, E->PTD2, R/W->PTD1, RS->PTD0) ***
****************************************
*******   programski RESET modula   *****

        lda     #!100
skok11  jsr     zak             ; 15 ms zakasnitev
        deca
        bne skok11
```

```
        mov     #%00110100,$03
        bclr     2,$03

        lda     #!22
skok12  jsr     zak             ; 4,1 ms zakasnitev
        deca
        bne skok12

        mov     #%00110100,$03
        bclr     2,$03

        lda     #!5
skok22  jsr     zak             ; 100 us zakasnitev
        deca
        bne skok22

        mov     #%00110100,$03
        bclr     2,$03
        jsr     zak             ; zakasnitev 37 us
*************************************
        clr$03
        mov      #%00100100,$03   ;VPIS NAČINA DELOVANJA (4-bitni podatki)
        bclr     2,$03
        jsr      zak
************************
        mov      #%00100100,$03
        bclr     2,$03
        mov      #%10000100,$03    ; določanje 4-bitnega prenosa podatkov,
        bclr     2,$03             ; števila vrstic in velikosti znakov
        jsr      zak               ; zakasnitev 37 us
*************************
        mov      #%00000100,$03
        bclr     2,$03
        mov      #%00010100,$03    ; brisanje zaslona, naslov v AC postavi
        bclr     2,$03             ; na $00
        lda      #$50
reset   jsr      zak               ; zakasnitev 1,52 ms
        deca
        bne      reset
*****************     ON/OFF    ************
        mov      #%00000100,$03
        bclr     2,$03
        mov      #%11000100,$03    ; določimo vklop/izklop prikazovalnika,
        bclr     2,$03             ; kurzorja in utripanje le-tega
        jsr     zak                ; zakasnitev 37 us
***************
        mov      #%00000100,$03
        bclr     2,$03
        mov      #%01100100,$03     ; določimo smer vpisovanja in pomik
        bclr     2,$03              ; napisa
        jsr     zak                 ; zakasnitev 37 us
```

Na začetku te priprave modula je dodano programsko resetiranje njegovega mikrokontrolerja, ki ga je potrebno izvesti, da bosta naš mikrokontroler in tisti na LCD modulu vedno pravilno komunicirala med seboj. Če namreč naš mikrokontroler resetiramo, bo začel s pošiljanjem

ukazov LCD modulu tako kot vedno s prvim ukazom, ki sploh pove modulu, da komunikacija poteka v 4-bitnem načinu. Ta ukaz je za razliko od ostalih 4-bitni. Če mikrokontroler na LCD modulu ni resetiran, že deluje v 4-bitnem načinu in pričakuje vsak ukaz ali podatek v dveh 4-bitnih delih. Zato sprejema napačne ukaze in podatke. Z vsakokratnim programskim resetiranjem LCD modula se temu problemu izognemo.

Vezalno shemo termometra vidimo na sliki 71.



Slika 71: El. shema povezave mikrokontrolerja, senzorja temperature in LCD prikazovalnika

Kot vidimo, je temperaturni senzor priključen na PTB0/AD0, LCD modul pa na port D, in sicer s 4-bitno podatkovno povezavo. Pripravili bomo program, ki bo na zaslonu prikazoval trenutno temperaturo, merjeno z NTK uporom. Za pretvorbo vrednosti A/D pretvornika v temperaturo bomo sestavili tabelo pretvorbe. Najprej moramo izvedeti, kako se odziva NTK upor na temperaturo v delilniku napetosti z uporom R2. Zato bomo namesto senzorja v delilnik najprej povezali dekadni upor, s katerim bomo simulirali upornost NTK upora pri različnih temperaturah. Na dekadnem uporu bomo nastavljali vrednosti, ki so podane v tabeli 5. Na zaslonu bomo prikazali vrednost, ki jo bo za posamezno nastavljeno temperaturo (in s tem vhodno napetost na PTB0/AD0) pretvoril A/D pretvornik. Vrednost bomo izpisali v desetiški obliki. Za to potrebujemo tabelo, v kateri so ASCII kode števil od 0 do 255. Ker je vrednost pretvorbe lahko večmestna, bomo kode za posamezen izpis dobili na več zaporednih naslovih. Vrednost, ki jo bomo prebrali v podatkovnem registru A/D pretvornika, bomo uporabili za dostop do ustreznih znakov (števil) v tabeli pretvorbe. Zato bomo morali predelati ta podatek v tako obliko, da nam bo omogočal prebiranje kod na 4 zaporednih naslovih. Tabelo pretvorbe bomo postavili na začetek *Flash* pomnilnika (od naslova $8000 dalje). Če dobimo iz A/D pretvornika vrednost 0 (%00000000), moramo s predelanim podatkom dobiti vrednost $8000 (%1000000000000000). To vrednost bomo uporabili za ukaz, ki nam bo s pomočjo indeksnega naslavljanja poiskal kodo znaka na naslovu $8000. S spreminjanjem najnižjih dveh bitov naslova pridemo do vseh znakov, ki predstavljajo desetiško vrednost pretvorbe A/D pretvornika. Tako

bomo za poljubno 8-bitno vrednost, ki jo bomo prebrali v podatkovnem registru A/D pretvornika, dobili ustrezne znake (kode števil) za izpis na zaslonu. Postopek take "predelave" podatka vidimo na sliki:

								AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	podatek iz A/D pretvornika
+	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	dodamo vrednost, ki nam omogoči dostop do tabele pretvorbe
	0	0	1	0	0	0	0	0	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	dobljeno vrednost dvakrat pomaknemo
	1	0	0	0	0	0	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	0	0	vrednost po dveh pomikih v levo
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	naslovni biti
	1	0	0	0	0	0	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	0	0	naslov 1. znaka za izpis
	1	0	0	0	0	0	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	0	1	naslov 2. znaka za izpis
	1	0	0	0	0	0	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	1	0	naslov 3. znaka za izpis
	1	0	0	0	0	0	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	1	1	naslov 4. znaka za izpis

Slika 72: Pridobivanje naslova znakov v tabeli za izpis na LCD prikazovalnik

Izpis bomo izvedli kar na začetku 1. vrstice LCD prikazovalnika. Po izpisu znakov moramo LCD modulu poslati ukaz, ki bo postavil naslovni register spet na začetek zaslona, da bo naslednji izpis spet na istih mestih zaslona. Ker bi prehitro izvajanje izpisov na zaslon povzročilo hitro menjavanje števil na zaslonu in s tem nečitljiv napis, bomo med posameznimi izpisi naredili zakasnitev, ki bo trajala eno sekundo. A/D pretvorniku bomo določili stalno pretvorbo na uporabljenem priključku, odčitavanje pretvorjene vrednosti v njegovem podatkovnem registru pa bomo izvršili pred vsakim izpisom, torej vsako sekundo.

Program, ki nam izpisuje vrednost analogno-digitalne pretvorbe v desetiški obliki, izgleda tako:

```
        org     $FFFE
        fdb     $A000   ; reset vektor
*************************************************************
        org     $8000   ; tabela za izpis pretvorbe A/D pretvornika
        DB      "   0   1   2   3   4   5   6   7   8   9"
        DB      "  10  11  12  13  14  15  16  17  18  19"
        DB      "  20  21  22  23  24  25  26  27  28  29"
        DB      "  30  31  32  33  34  35  36  37  38  39"
        DB      "  40  41  42  43  44  45  46  47  48  49"
        DB      "  50  51  52  53  54  55  56  57  58  59"
        DB      "  60  61  62  63  64  65  66  67  68  69"
        DB      "  70  71  72  73  74  75  76  77  78  79"
        DB      "  80  81  82  83  84  85  86  87  88  89"
        DB      "  90  91  92  93  94  95  96  97  98  99"
        DB      " 100 101 102 103 104 105 106 107 108 109"
        DB      " 110 111 112 113 114 115 116 117 118 119"
        DB      " 120 121 122 123 124 125 126 127 128 129"
        DB      " 130 131 132 133 134 135 136 137 138 139"
        DB      " 140 141 142 143 144 145 146 147 148 149"
        DB      " 150 151 152 153 154 155 156 157 158 159"
        DB      " 160 161 162 163 164 165 166 167 168 169"
```

```
        DB      " 170 171 172 173 174 175 176 177 178 179"
        DB      " 180 181 182 183 184 185 186 187 188 189"
        DB      " 190 191 192 193 194 195 196 197 198 199"
        DB      " 200 201 202 203 204 205 206 207 208 209"
        DB      " 210 211 212 213 214 215 216 217 218 219"
        DB      " 220 221 222 223 224 225 226 227 228 229"
        DB      " 230 231 232 233 234 235 236 237 238 239"
        DB      " 240 241 242 243 244 245 246 247 248 249"
        DB      " 250 251 252 253 254 255"

        org     $a000
*********************************************
        MOV     #$FF,$07        ; port D so IZHODI
        BSET    0,$1F           ; COP onemogočen

$INCLUDE  "timer1_2.asm"        ; priprava obeh časovnikov
$INCLUDE  "Priprava LCD2.asm"   ; priprava LCD modula na izpis 2x16 znakov

        MOV     #%00110000,$3E        ; frekvenca CLK za ADC (fbus:2)
        MOV     #%00100000,$3C        ; stalna pretvorba na PTB0
*********************************************

KONEC   MOV     #$20,$50
        MOV     $3D,$51         ; prebere pretvorbo z A/D-ja
        CLC
        ROL     $51             ;
        ROL     $50             ;
        ROL     $51             ; pripravi začetni naslov
        ROL     $50             ; za znake iz tabele
        MOV     #!4,$60

        LDHX    $50
SKOK1   LDA     !0,X
        AND     #%11110000
        ADD     #%00000101
        STA     $03
        BCLR     2,$03
        LDA     !0,X            ;vpiše PRETVORBO NA LCD
        ASLA
        ASLA
        ASLA
        ASLA
        ADD#%00000101
        STA$03
        BCLR     2,$03
        JSR     zak
        INCX
        DEC     $60
        BNE     SKOK1           ; izpiše 4 znake iz tabele
        CLR     $03

*****************     CURSOR HOME  *******
        MOV #%00000100,$03
        BCLR     2,$03
        MOV #%00100100,$03       ; postavi kurzor na začetek zaslona
        BCLR     2,$03           ; za naslednji izpis
```

```
        JSR     zak
********** ZAKASNITEV  NASLEDNJEGA IZPISA PRETVORBE ***************
        BCLR    5,$2B       ;start
s8      BRCLR   7,$2B,s8    ;kontrola štetja
        BCLR    7,$2B       ;brisanje TOF
        BSET    5,$2B       ;stop
        BRA     KONEC
*****************************************
********** ZAKASNITEV (podprogram) *********

zak     BCLR    5,$20       ;start
s9      BRCLR   7,$20,s9    ;kontrola štetja
        BSET    5,$20       ;stop
        BCLR    7,$20       ;brisanje TOF
        RTS
```

S tem programom dobimo vrednosti A/D pretvorbe za temperature, ki so podane v tabeli 4. Dobljene vrednosti poiščemo v tabeli pretvorbe in jih nadomestimo s temperaturami, ki ustrezajo nastavitvam dekadnega upora. Ker so te temperature podane na 5 °C natančno, moramo vmesne temperature preračunati in nadomestiti še ostale desetiške vrednosti v tabeli. Po vnesenih temperaturah (med –24 °C in +55 °C) je tabela pretvorbe takšna:

```
        org     $8000                                   ;******
        DB      " Err Err Err Err Err Err Err Err Err Err"
        DB      " -24 -23 -22 -21 -20 -19 -19 -18 -17 -17"
        DB      " -16 -15 -15 -14 -13 -13 -12 -11 -11 -10"
        DB      "-9.5-0.0-8.5-8.5-8.0-8.0-7.5-7.0-7.0-6.5"
        DB      "-6.5-6.0-5.5-5.0-5.0-4.5-4.0-3.5-3.5-3.0"
        DB      "-2.5-2.5-2.0-1.5-1.0-1.0-0.5 0.0 0.0 0.5"
        DB      " 1.0 1.0 1.5 2.0 2.0 2.5 3.0 3.0 3.5 3.5"
        DB      " 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.0 7.5 8.0"
        DB      " 9.0 8.5 9.0 9.0 9.510.010.010.511.011.5"
        DB      "12.012.512.513.013.013.514.014.514.515.0"
        DB      "15.516.016.516.517.017.517.518.018.518.5"
        DB      "19.019.519.520.020.020.521.021.021.522.0"
        DB      "22.022.523.023.524.024.525.025.526.026.5"
        DB      "27.027.528.028.029.029.029.529.530.030.5"
        DB      "31.031.031.531.532.032.032.533.033.033.5"
        DB      "34.034.034.534.535.035.536.036.036.537.0"
        DB      "37.037.538.038.538.539.039.039.540.040.5"
        DB      "40.541.041.542.042.543.043.043.544.044.0"
        DB      "44.544.545.045.546.046.547.047.047.547.5"
        DB      "48.048.048.548.549.049.549.550.050.050.5"
        DB      "50.551.051.051.551.552.052.052.552.553.0"
        DB      "53.053.553.554.054.054.555.055.055.555.5"
        DB      " --  --  --  --  --  --  --  --  --  -- "
        DB      " --  --  --  --  --  --  --  --  --  -- "
        DB      " Err Err Err Err Err Err Err Err Err Err"
        DB      " Err Err Err Err Err Err "
```

Program nato samo še dopolnimo z izpisom enote za temperaturo in imamo izdelan termometer. Njegova natančnost je predvsem odvisna od natančnosti nastavitve upornosti med merjenjem z dekadnim uporom.

Ponovimo

Analogno-digitalni pretvornik nam omogoča uporabo senzorjev, s katerimi lahko nadziramo fizikalne veličine, kot so temperatura, svetloba, sila in druge. Senzorje moramo priključiti tako, da nam sprememba fizikalne veličine povzroči spremembo električne napetosti na vhodu A/D pretvornika. Nivo napetosti se pretvori v 8-bitno vrednost, ki jo lahko preberemo v podatkovnem registru pretvornika. Z ustreznimi nastavitvami v registru taktnega signala in predvsem v statusnem in kontrolnem registru A/D pretvornika lahko na zahtevo (z ukazom) dobimo eno pretvorbo ali pa stalno pretvarjanje vhodne napetosti. Določamo tudi, katerega od 8 vhodov bomo uporabili za pretvorbo.

Na primerih smo spoznali tudi priključitve pasivnega in aktivnega senzorja ter priključitev in uporabo LCD prikazovalnika.

Vaje

1. Sestavi mikrokontrolersko vezje s senzorjem razdalje in LCD prikazovalnikom. Program za merjenje in prikaz temperature iz tega poglavja predelaj tako, da bo prikazoval razdaljo med senzorjem in oviro. Razdaljo podaj v centimetrih na 1 cm natančno.
2. Z uporabo senzorja temperature (NTK upora) in mikrokontrolerja realiziraj regulator temperature v prostoru oz. v maketi. Mikrokontroler naj grelec vklopi, ko je merjena temperatura nižja od 23 °C, izklopi pa, ko preseže 25 °C. Temperaturo naj meri vsakih 5 sekund.
3. Program iz prejšnje vaje dopolni s prikazom trenutne temperature prostora na LCD prikazovalniku.

PRILOGA

Ukazi mikrokontrolerja

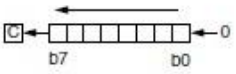
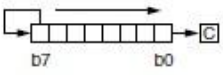
Source Form	Operation	Description	Effect on CCR V	H	I	N	Z	C	Address Mode	Opcode	Operand	Cycles
ADC #*opr* ADC *opr* ADC *opr* ADC *opr*,X ADC *opr*,X ADC ,X ADC *opr*,SP ADC *opr*,SP	Add with Carry	A ← (A) + (M) + (C)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A9 B9 C9 D9 E9 F9 9EE9 9ED9	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
ADD #*opr* ADD *opr* ADD *opr* ADD *opr*,X ADD *opr*,X ADD ,X ADD *opr*,SP ADD *opr*,SP	Add without Carry	A ← (A) + (M)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	AB BB CB DB EB FB 9EEB 9EDB	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
AIS #*opr*	Add Immediate Value (Signed) to SP	SP ← (SP) + (16 « M)	–	–	–	–	–	–	IMM	A7	ii	2
AIX #*opr*	Add Immediate Value (Signed) to H:X	H:X ← (H:X) + (16 « M)	–	–	–	–	–	–	IMM	AF	ii	2
AND #*opr* AND *opr* AND *opr* AND *opr*,X AND *opr*,X AND ,X AND *opr*,SP AND *opr*,SP	Logical AND	A ← (A) & (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A4 B4 C4 D4 E4 F4 9EE4 9ED4	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
ASL *opr* ASLA ASLX ASL *opr*,X ASL ,X ASL *opr*,SP	Arithmetic Shift Left (Same as LSL)		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
ASR *opr* ASRA ASRX ASR *opr*,X ASR *opr*,X ASR *opr*,SP	Arithmetic Shift Right		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	37 47 57 67 77 9E67	dd ff ff	4 1 1 4 3 5
BCC *rel*	Branch if Carry Bit Clear	PC ← (PC) + 2 + rel ? (C) = 0	–	–	–	–	–	–	REL	24	rr	3
BCLR *n*, *opr*	Clear Bit n in M	Mn ← 0	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	11 13 15 17 19 1B 1D 1F	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BCS *rel*	Branch if Carry Bit Set (Same as BLO)	PC ← (PC) + 2 + *rel* ? (C) = 1	–	–	–	–	–	–	REL	25	rr	3
BEQ *rel*	Branch if Equal	PC ← (PC) + 2 + *rel* ? (Z) = 1	–	–	–	–	–	–	REL	27	rr	3
BGE *opr*	Branch if Greater Than or Equal To (Signed Operands)	PC ← (PC) + 2 + *rel* ? (N ⊕ V) = 0	–	–	–	–	–	–	REL	90	rr	3
BGT *opr*	Branch if Greater Than (Signed Operands)	PC ← (PC) + 2 + *rel* ? (Z) | (N ⊕ V) = 0	–	–	–	–	–	–	REL	92	rr	3
BHCC *rel*	Branch if Half Carry Bit Clear	PC ← (PC) + 2 + *rel* ? (H) = 0	–	–	–	–	–	–	REL	28	rr	3
BHCS *rel*	Branch if Half Carry Bit Set	PC ← (PC) + 2 + *rel* ? (H) = 1	–	–	–	–	–	–	REL	29	rr	3
BHI *rel*	Branch if Higher	PC ← (PC) + 2 + *rel* ? (C) | (Z) = 0	–	–	–	–	–	–	REL	22	rr	3

Tabela 8: Ukazi za MC908GP32 (1/6)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
BHS *rel*	Branch if Higher or Same (Same as BCC)	PC ← (PC) + 2 + *rel* ? (C) = 0	–	–	–	–	–	–	REL	24	rr	3
BIH *rel*	Branch if $\overline{IRQ}$ Pin High	PC ← (PC) + 2 + *rel* ? $\overline{IRQ}$ = 1	–	–	–	–	–	–	REL	2F	rr	3
BIL *rel*	Branch if $\overline{IRQ}$ Pin Low	PC ← (PC) + 2 + *rel* ? $\overline{IRQ}$ = 0	–	–	–	–	–	–	REL	2E	rr	3
BIT #*opr* BIT *opr* BIT *opr* BIT *opr*,X BIT *opr*,X BIT ,X BIT *opr*,SP BIT *opr*,SP	Bit Test	(A) & (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A5 B5 C5 D5 E5 F5 9EE5 9ED5	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
BLE *opr*	Branch if Less Than or Equal To (Signed Operands)	PC ← (PC) + 2 + *rel* ? (Z) ⊕ (N ⊕ V) = 1	–	–	–	–	–	–	REL	93	rr	3
BLO *rel*	Branch if Lower (Same as BCS)	PC ← (PC) + 2 + *rel* ? (C) = 1	–	–	–	–	–	–	REL	25	rr	3
BLS *rel*	Branch if Lower or Same	PC ← (PC) + 2 + *rel* ? (C) ⊕ (Z) = 1	–	–	–	–	–	–	REL	23	rr	3
BLT *opr*	Branch if Less Than (Signed Operands)	PC ← (PC) + 2 + *rel* ? (N ⊕ V) =1	–	–	–	–	–	–	REL	91	rr	3
BMC *rel*	Branch if Interrupt Mask Clear	PC ← (PC) + 2 + *rel* ? (I) = 0	–	–	–	–	–	–	REL	2C	rr	3
BMI *rel*	Branch if Minus	PC ← (PC) + 2 + *rel* ? (N) = 1	–	–	–	–	–	–	REL	2B	rr	3
BMS *rel*	Branch if Interrupt Mask Set	PC ← (PC) + 2 + *rel* ? (I) = 1	–	–	–	–	–	–	REL	2D	rr	3
BNE *rel*	Branch if Not Equal	PC ← (PC) + 2 + *rel* ? (Z) = 0	–	–	–	–	–	–	REL	26	rr	3
BPL *rel*	Branch if Plus	PC ← (PC) + 2 + *rel* ? (N) = 0	–	–	–	–	–	–	REL	2A	rr	3
BRA *rel*	Branch Always	PC ← (PC) + 2 + *rel*	–	–	–	–	–	–	REL	20	rr	3
BRCLR *n,opr,rel*	Branch if Bit *n* in M Clear	PC ← (PC) + 3 + *rel* ? (Mn) = 0	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BRN *rel*	Branch Never	PC ← (PC) + 2	–	–	–	–	–	–	REL	21	rr	3
BRSET *n,opr,rel*	Branch if Bit *n* in M Set	PC ← (PC) + 3 + *rel* ? (Mn) = 1	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BSET *n,opr*	Set Bit *n* in M	Mn ← 1	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 12 14 16 18 1A 1C 1E	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BSR *rel*	Branch to Subroutine	PC ← (PC) + 2; push (PCL) SP ← (SP) – 1; push (PCH) SP ← (SP) – 1 PC ← (PC) + *rel*	–	–	–	–	–	–	REL	AD	rr	4
CBEQ *opr,rel* CBEQA #*opr,rel* CBEQX #*opr,rel* CBEQ *opr*,X+,*rel* CBEQ X+,*rel* CBEQ *opr*,SP,*rel*	Compare and Branch if Equal	PC ← (PC) + 3 + rel ? (A) – (M) = $00 PC ← (PC) + 3 + rel ? (A) – (M) = $00 PC ← (PC) + 3 + rel ? (X) – (M) = $00 PC ← (PC) + 3 + rel ? (A) – (M) = $00 PC ← (PC) + 2 + rel ? (A) – (M) = $00 PC ← (PC) + 4 + rel ? (A) – (M) = $00	–	–	–	–	–	–	DIR IMM IMM IX1+ IX+ SP1	31 41 51 61 71 9E61	dd rr ii rr ii rr ff rr rr ff rr	5 4 4 5 4 6
CLC	Clear Carry Bit	C ← 0	–	–	–	–	–	0	INH	98		1
CLI	Clear Interrupt Mask	I ← 0	–	–	0	–	–	–	INH	9A		2

Tabela 9: Ukazi za MC908GP32 (2/6)

Source Form	Operation	Description	Effect on CCR V	H	I	N	Z	C	Address Mode	Opcode	Operand	Cycles
CLR *opr* CLRA CLRX CLRH CLR *opr*,X CLR ,X CLR *opr*,SP	Clear	M ← $00 A ← $00 X ← $00 H ← $00 M ← $00 M ← $00 M ← $00	0	–	–	0	1	–	DIR INH INH INH IX1 IX SP1	3F 4F 5F 8C 6F 7F 9E6F	dd ff ff	3 1 1 1 3 2 4
CMP #*opr* CMP *opr* CMP *opr* CMP *opr*,X CMP *opr*,X CMP ,X CMP *opr*,SP CMP *opr*,SP	Compare A with M	(A) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A1 B1 C1 D1 E1 F1 9EE1 9ED1	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
COM *opr* COMA COMX COM *opr*,X COM ,X COM *opr*,SP	Complement (One's Complement)	M ← (M̄) = $FF – (M) A ← (Ā) = $FF – (M) X ← (X̄) = $FF – (M) M ← (M̄) = $FF – (M) M ← (M̄) = $FF – (M) M ← (M̄) = $FF – (M)	0	–	–	↕	↕	1	DIR INH INH IX1 IX SP1	33 43 53 63 73 9E63	dd ff ff	4 1 1 4 3 5
CPHX #*opr* CPHX *opr*	Compare H:X with M	(H:X) – (M:M + 1)	↕	–	–	↕	↕	↕	IMM DIR	65 75	ii ii+1 dd	3 4
CPX #*opr* CPX *opr* CPX *opr* CPX ,X CPX *opr*,X CPX *opr*,X CPX *opr*,SP CPX *opr*,SP	Compare X with M	(X) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A3 B3 C3 D3 E3 F3 9EE3 9ED3	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
DAA	Decimal Adjust A	$(A)_{10}$	U	–	–	↕	↕	↕	INH	72		2
DBNZ *opr*,*rel* DBNZA *rel* DBNZX *rel* DBNZ *opr*,X,*rel* DBNZ X,*rel* DBNZ *opr*,SP,*rel*	Decrement and Branch if Not Zero	A ← (A) – 1 or M ← (M) – 1 or X ← (X) – 1 PC ← (PC) + 3 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 3 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 4 + *rel* ? (result) ≠ 0	–	–	–	–	–	–	DIR INH INH IX1 IX SP1	3B 4B 5B 6B 7B 9E6B	dd rr rr rr ff rr rr ff rr	5 3 3 5 4 6
DEC *opr* DECA DECX DEC *opr*,X DEC ,X DEC *opr*,SP	Decrement	M ← (M) – 1 A ← (A) – 1 X ← (X) – 1 M ← (M) – 1 M ← (M) – 1 M ← (M) – 1	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3A 4A 5A 6A 7A 9E6A	dd ff ff	4 1 1 4 3 5
DIV	Divide	A ← (H:A)/(X) H ← Remainder	–	–	–	–	↕	↕	INH	52		7
EOR #*opr* EOR *opr* EOR *opr* EOR *opr*,X EOR *opr*,X EOR ,X EOR *opr*,SP EOR *opr*,SP	Exclusive OR M with A	A ← (A ⊕ M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A8 B8 C8 D8 E8 F8 9EE8 9ED8	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
INC *opr* INCA INCX INC *opr*,X INC ,X INC *opr*,SP	Increment	M ← (M) + 1 A ← (A) + 1 X ← (X) + 1 M ← (M) + 1 M ← (M) + 1 M ← (M) + 1	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3C 4C 5C 6C 7C 9E6C	dd ff ff	4 1 1 4 3 5

Tabela 10: Ukazi za MC908GP32 (3/6)

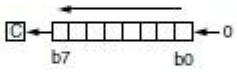
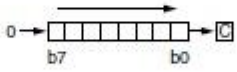
Source Form	Operation	Description	Effect on CCR V	H	I	N	Z	C	Address Mode	Opcode	Operand	Cycles
JMP *opr* JMP *opr* JMP *opr*,X JMP *opr*,X JMP ,X	Jump	PC ← Jump Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BC CC DC EC FC	dd hh ll ee ff ff	2 3 4 3 2
JSR *opr* JSR *opr* JSR *opr*,X JSR *opr*,X JSR ,X	Jump to Subroutine	PC ← (PC) + *n* (*n* = 1, 2, or 3) Push (PCL); SP ← (SP) – 1 Push (PCH); SP ← (SP) – 1 PC ← Unconditional Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff	4 5 6 5 4
LDA #*opr* LDA *opr* LDA *opr* LDA *opr*,X LDA *opr*,X LDA ,X LDA *opr*,SP LDA *opr*,SP	Load A from M	A ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A6 B6 C6 D6 E6 F6 9EE6 9ED6	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
LDHX #*opr* LDHX *opr*	Load H:X from M	H:X ← (M:M + 1)	0	–	–	↕	↕	–	IMM DIR	45 55	ii jj dd	3 4
LDX #*opr* LDX *opr* LDX *opr* LDX *opr*,X LDX *opr*,X LDX ,X LDX *opr*,SP LDX *opr*,SP	Load X from M	X ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	AE BE CE DE EE FE 9EEE 9EDE	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
LSL *opr* LSLA LSLX LSL *opr*,X LSL ,X LSL *opr*,SP	Logical Shift Left (Same as ASL)		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
LSR *opr* LSRA LSR*X* LSR *opr*,X LSR ,X LSR *opr*,SP	Logical Shift Right		↕	–	–	0	↕	↕	DIR INH INH IX1 IX SP1	34 44 54 64 74 9E64	dd ff ff	4 1 1 4 3 5
MOV *opr*,*opr* MOV *opr*,X+ MOV #*opr*,*opr* MOV X+,*opr*	Move	(M)$_{Destination}$ ← (M)$_{Source}$ H:X ← (H:X) + 1 (IX+D, DIX+)	0	–	–	↕	↕	–	DD DIX+ IMD IX+D	4E 5E 6E 7E	dd dd dd ii dd dd	5 4 4 4
MUL	Unsigned multiply	X:A ← (X) × (A)	–	0	–	–	–	0	INH	42		5
NEG *opr* NEGA NEGX NEG *opr*,X NEG ,X NEG *opr*,SP	Negate (Two's Complement)	M ← –(M) = $00 – (M) A ← –(A) = $00 – (A) X ← –(X) = $00 – (X) M ← –(M) = $00 – (M) M ← –(M) = $00 – (M)	↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	30 40 50 60 70 9E60	dd ff ff	4 1 1 4 3 5
NOP	No Operation	None	–	–	–	–	–	–	INH	9D		1
NSA	Nibble Swap A	A ← (A[3:0]:A[7:4])	–	–	–	–	–	–	INH	62		3
ORA #*opr* ORA *opr* ORA *opr* ORA *opr*,X ORA *opr*,X ORA ,X ORA *opr*,SP ORA *opr*,SP	Inclusive OR A and M	A ← (A) | (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	AA BA CA DA EA FA 9EEA 9EDA	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
PSHA	Push A onto Stack	Push (A); SP ← (SP) – 1	–	–	–	–	–	–	INH	87		2
PSHH	Push H onto Stack	Push (H); SP ← (SP) – 1	–	–	–	–	–	–	INH	8B		2
PSHX	Push X onto Stack	Push (X); SP ← (SP) – 1	–	–	–	–	–	–	INH	89		2

Tabela 11: Ukazi za MC908GP32 (4/6)

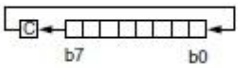
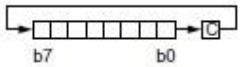
Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
PULA	Pull A from Stack	SP ← (SP + 1); Pull (A)	–	–	–	–	–	–	INH	86		2
PULH	Pull H from Stack	SP ← (SP + 1); Pull (H)	–	–	–	–	–	–	INH	8A		2
PULX	Pull X from Stack	SP ← (SP + 1); Pull (X)	–	–	–	–	–	–	INH	88		2
ROL *opr* ROLA ROLX ROL *opr*,X ROL ,X ROL *opr*,SP	Rotate Left through Carry		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	39 49 59 69 79 9E69	dd ff ff	4 1 1 4 3 5
ROR *opr* RORA RORX ROR *opr*,X ROR ,X ROR *opr*,SP	Rotate Right through Carry		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	36 46 56 66 76 9E66	dd ff ff	4 1 1 4 3 5
RSP	Reset Stack Pointer	SP ← $FF	–	–	–	–	–	–	INH	9C		1
RTI	Return from Interrupt	SP ← (SP) + 1; Pull (CCR) SP ← (SP) + 1; Pull (A) SP ← (SP) + 1; Pull (X) SP ← (SP) + 1; Pull (PCH) SP ← (SP) + 1; Pull (PCL)	↕	↕	↕	↕	↕	↕	INH	80		7
RTS	Return from Subroutine	SP ← SP + 1; Pull (PCH) SP ← SP + 1; Pull (PCL)	–	–	–	–	–	–	INH	81		4
SBC #*opr* SBC *opr* SBC *opr* SBC *opr*,X SBC *opr*,X SBC ,X SBC *opr*,SP SBC *opr*,SP	Subtract with Carry	A ← (A) – (M) – (C)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A2 B2 C2 D2 E2 F2 9EE2 9ED2	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
SEC	Set Carry Bit	C ← 1	–	–	–	–	–	1	INH	99		1
SEI	Set Interrupt Mask	I ← 1	–	–	1	–	–	–	INH	9B		2
STA *opr* STA *opr* STA *opr*,X STA *opr*,X STA ,X STA *opr*,SP STA *opr*,SP	Store A in M	M ← (A)	0	–	–	↕	↕	–	DIR EXT IX2 IX1 IX SP1 SP2	B7 C7 D7 E7 F7 9EE7 9ED7	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5
STHX *opr*	Store H:X in M	(M:M + 1) ← (H:X)	0	–	–	↕	↕	–	DIR	35	dd	4
STOP	Enable Interrupts, Stop Processing, Refer to MCU Documentation	I ← 0; Stop Processing	–	–	0	–	–	–	INH	8E		1
STX *opr* STX *opr* STX *opr*,X STX *opr*,X STX ,X STX *opr*,SP STX *opr*,SP	Store X in M	M ← (X)	0	–	–	↕	↕	–	DIR EXT IX2 IX1 IX SP1 SP2	BF CF DF EF FF 9EEF 9EDF	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5
SUB #*opr* SUB *opr* SUB *opr* SUB *opr*,X SUB *opr*,X SUB ,X SUB *opr*,SP SUB *opr*,SP	Subtract	A ← (A) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A0 B0 C0 D0 E0 F0 9EE0 9ED0	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Tabela 12: Ukazi za MC908GP32 (5/6)

Source Form	Operation	Description	Effect on CCR V	H	I	N	Z	C	Address Mode	Opcode	Operand	Cycles
SWI	Software Interrupt	PC ← (PC) + 1; Push (PCL) SP ← (SP) – 1; Push (PCH) SP ← (SP) – 1; Push (X) SP ← (SP) – 1; Push (A) SP ← (SP) – 1; Push (CCR) SP ← (SP) – 1; I ← 1 PCH ← Interrupt Vector High Byte PCL ← Interrupt Vector Low Byte	–	–	1	–	–	–	INH	83		9
TAP	Transfer A to CCR	CCR ← (A)	↕	↕	↕	↕	↕	↕	INH	84		2
TAX	Transfer A to X	X ← (A)	–	–	–	–	–	–	INH	97		1
TPA	Transfer CCR to A	A ← (CCR)	–	–	–	–	–	–	INH	85		1
TST *opr* TSTA TSTX TST *opr*,X TST ,X TST *opr*,SP	Test for Negative or Zero	(A) – $00 or (X) – $00 or (M) – $00	0	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3D 4D 5D 6D 7D 9E6D	dd ff ff	3 1 1 3 2 4
TSX	Transfer SP to H:X	H:X ← (SP) + 1	–	–	–	–	–	–	INH	95		2
TXA	Transfer X to A	A ← (X)	–	–	–	–	–	–	INH	9F		1
TXS	Transfer H:X to SP	(SP) ← (H:X) – 1	–	–	–	–	–	–	INH	94		2
WAIT	Enable Interrupts; Wait for Interrupt	I bit ← 0; Inhibit CPU clocking until interrupted	–	–	0	–	–	–	INH	8F		1

A	Accumulator	*n*	Any bit
C	Carry/borrow bit	*opr*	Operand (one or two bytes)
CCR	Condition code register	PC	Program counter
dd	Direct address of operand	PCH	Program counter high byte
dd rr	Direct address of operand and relative offset of branch instruction	PCL	Program counter low byte
DD	Direct to direct addressing mode	REL	Relative addressing mode
DIR	Direct addressing mode	*rel*	Relative program counter offset byte
DIX+	Direct to indexed with post increment addressing mode	rr	Relative program counter offset byte
ee ff	High and low bytes of offset in indexed, 16-bit offset addressing	SP1	Stack pointer, 8-bit offset addressing mode
EXT	Extended addressing mode	SP2	Stack pointer 16-bit offset addressing mode
ff	Offset byte in indexed, 8-bit offset addressing	SP	Stack pointer
H	Half-carry bit	U	Undefined
H	Index register high byte	V	Overflow bit
hh ll	High and low bytes of operand address in extended addressing	X	Index register low byte
I	Interrupt mask	Z	Zero bit
ii	Immediate operand byte	&	Logical AND
IMD	Immediate source to direct destination addressing mode	|	Logical OR
IMM	Immediate addressing mode	⊕	Logical EXCLUSIVE OR
INH	Inherent addressing mode	( )	Contents of
IX	Indexed, no offset addressing mode	–( )	Negation (two's complement)
IX+	Indexed, no offset, post increment addressing mode	#	Immediate value
IX+D	Indexed with post increment to direct addressing mode	«	Sign extend
IX1	Indexed, 8-bit offset addressing mode	←	Loaded with
IX1+	Indexed, 8-bit offset, post increment addressing mode	?	If
IX2	Indexed, 16-bit offset addressing mode	:	Concatenated with
M	Memory location	↕	Set or cleared
N	Negative bit	—	Not affected

Tabela 13: Ukazi za MC908GP32 (6/6)

ASCII tabela

Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	q	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	-¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	ˣ	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		−	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	°	ö	

Tabela 14: ASCII kode in znaki

MEDPREDMETNO POVEZOVANJE

Povezava s predmetom *Angleščina*:

- uporaba podatkov in razlag o zgradbi in delovanju mikrokontrolerja proizvajalca (pdf datoteka proizvajalca): MC68HC908GP32 Data Sheet, Rev. 10 1/2008, 1/2008 Freescale Semiconductor, Inc.
- druge uporabljene pdf datoteke
- iskanje podatkov o elektronskih elementih na spletu

Povezava s predmetom *Fizika*:

- merjenje temperature, svetlobe, vlage, pritiska in drugih fizikalnih veličin (poznavanje veličin ter njihovih nivojev v pogojih uporabe, zagotavljanje dobre toplotne prevodnosti med senzorjem temperature in opazovanim medijem – zrak, kovina, tekočina)

Povezava z modulom *Upravljanje s programirljivimi napravami*:

- poznavanje osnov digitalne tehnike, logičnih in sekvenčnih funkcij
- programiranje, algoritmično razmišljanje, razvojno okolje, delovanje prosto programirljivih naprav (PLK)
- uporaba senzorjev

Povezava z modulom *Zajemanje in obdelava procesnih veličin*:

- poznavanje principov pretvorbe fizikalnih veličin v električne
- principi A/D in D/A pretvorbe

LITERATURA IN VIRI

Splet: pridobljeno 20.7.2012 iz:
http://en.wikipedia.org/wiki/Harvard_architecture

Splet: pridobljeno 20.7.2012 iz:
http://en.wikipedia.org/wiki/Von_Neumann_architecture

Splet: pridobljeno 15.7.2012 iz:
http://www.freescale.com/files/microcontrollers/doc/data_sheet/M68HC11E.pdf

MC68HC908GP32 Data Sheet, Rev. 10 1/2008, Freescale Semiconductor, Inc., (pdf datoteka proizvajalca)

PIC16F87XA Data Sheet, Microchip Technology Inc., 28.7.2003, (pdf datoteka proizvajalca)

Vishay BCcomponents: NTC Thermistors, Accuracy Line; Document Number: 91000, Revision: 08-Apr-05, Vishay Intertechnology, Inc. (pdf datoteka proizvajalca)

HD44780U (Dot Matrix Liquid Crystal Display Controller/Driver), Hitachi, Ltd. Semiconductor & Integrated Circuits, ADE-207-272(Z) '99.9, Rev. 0.0, (pdf datoteka proizvajalca)

LCD Module DEM 16216 SYH-LY Product specification, Display Elektronik, GmbH, 8. 4. 2003, Version 1, (pdf datoteka proizvajalca)