Mikrokontroleri MC9S08

Uvod

Prvi poluprovodnički računari bili napravljeni su sa bipolarnim tranzistorima, i predstavljali su značajni napredak u osnosu na predhodnu generaciju računara koji su bili bazirani na elektronskim cevima. Usavršavanje tehnologije izrade poluprovodničkih komponenata, dovelo je do industrijske proizvodnje integrisanih poluprovodničkih komponenata, skraćeno nazvana integrisana kola. Do 70-tih godina prošlog veka dominira bipolarna tehnologija i nizak stepen integracije komponenata u jednom integrisanom kolu. Tadašnji računari sadrže gomilu integrisanih kola, a kao RAM memoriju koriste feritne mehurice (gvožđe) sa malim kapacitetom i malom brzinom pristupa. Spoljne memorije su bile trake i diskovi, a tehnika virtuelne memorije (na spoljnoj memoriji) je nadoknađivala mali kapacitet RAM-a.

U laboratorijama se tih godina završavaju, i pripremaju za industrijsku proizvodnju, integrisana kola visokog stepena integracije koja u sebi sadrže ključne komponente računara: aritmetičko logičku jedinicu, mikroprogramski sekvencer sa dekoderom naredbi, RAM memoriju i programsku memoriju. To je omogućilo sledeći tehnološki skok u računarskoj tehnologiji, i računari se tada izrađuju na bazi bit-slice strukture. Jedan slece (kriška) je najčešće 4-o bitna (bilo ih je i 1 bitnih) ALU sa akumulatorom i registrima, sa mogućnošću povezivanja više njih, tako da se dobijala 16-to bitna, 32-o bitna, ili neka druga ALU. Mikroprogramski sekvencer je bio integrisan u jednom čipu, ali nije sadržao dekoder naredbi. Mikroprogramski sekvencer je adresirao takozvanu mikroprogramsku memoriju, koja je imala na svakoj lokaciji širinu od više bajtova. Svaki bit mikroprogramske memorije je bio kontrolni signal kojim se upravljalo ALU jedinicom i ostalim resursima racunara. Programski sekvencer je bio obićan registar koji se mogao postaviti na bilo koju vrednost iz ALU i čiji je sadržaj bio dostupan ALU jedinici. Ovaj registar je zapravo programski brojać. Na slićan naćin je napravljen i stek pointer. Za svaku asemblersku naredbu, u mikroprogramskoj memoriji postoji podprogram. Takodje postoji i podprogram za čitanje sledeće asemblerske naredbe iz programske memorije. Mikroprogramski sekvencer radi tako da se posle izvršenja svakog podprograma u mikroprogramskoj memoriji izvršava mikroprogram za čitanje sledeće asemblerske naredbe. Kod pročitane naredbe, mikroprogramski sekvencer pretvara u adresu poziva odgovarajućeg mikroprogramskog podprograma.

Već 70-tih godina postiže se još veći stepen integracije, tako da se pojavljuju prvi mikroprocesori koji integrišu u sebi ALU, sekvencer sa dekoderom naredbi (kontrolna jedinica), ali ne i memoriju. Tako beležimo 1972: 8008, 1974: 8080 i MC6800, 1975: 6502, 1976: Z80 i CDP1802 i konačno 1978: MC6809. Z80 je proistekao iz 8080, unapredivši ga i nadmašivši ga u popularnosti i uspehu na tržištu. Nešto slićno se desilo i sa 6502 koji je sličan sa 6800, ali koji ima jedan akumulator i dva indeksna registra, naspram dva akumulatora i jedanim indeksnim registarom u MC6800. CDP1802 je prvi CMOS mikroprocesor koji je korišćen u vojnim uredjajima, ali i u kosmičkim sondama, jer je CMOS tehnologija otpornija na smetnje i ima veće margine šuma (ispravno prihvata logičke signale uz veći nivo šuma u ulaznom signalu). Najkompleksniji, i sa najboljim pervormansama, MC6809 se javio relativno kasno, jer su u mini raćunarima prevlast poćeli da preuzimaju 16-to bitni mikroprocesori, a pojava mikrokontrolera je sve više potiskivala mikroprocesore u sveri upravljanja procesima i drugim sličnim aplikacijama.

Istorija mikrokontrolera poćinje 1976. god. sa Intelovim 8048. Iz ovog mikrokontrolera će kasnije proizaći 8031, 8051 i niz drugih tipova ove familije. U vreme nastanka ove familije mikrokontrolera, nivo integracije integrisanih kola bio je relativno nizak. Zbog toga su performanse svih tipova iz ove familije daleko ispod performansi tadašnjih mikroprocesora. Morao se naći kompromis kako bi se, na

silicijumu, smestilo sve sto je potrebno jednom mikrokontroleru. Zato ovaj mikrokontroler može direktno da adresira RAM samo do lokacije 127, a programska memorija je morala da bude eksterna. Ako je bilo potrebno, mogao se povezati i eksterni RAM. Set instrukcija je bio prilagodjen ovim skromnim hardverskim performansama. Medjutim, ovaj mikrokontroler je stekao veliku popularnost, i održao se do današnjih dana. Tehnološki napredak u izradi integrisanih kola (veći stepen integracije i veća brzina) nije mogao da se na jednostavan način implementira u ovu familiju. Prepreku je predstavljao skup naredbi koji nije mogao da se menja (zbog kompatibilnosti), a u kodnoj tablici nije bilo mnogo praznih mesta za značajnije proširenje seta instrukcija. Kada je postalo moguće integrisati RAM od 256 bajtova, rešenje je nadjeno u sledećem: direktnim adresiranjem lokacija od 128 do 255 pristupalo se registrima mikrokontrolera, a indirektnim adresiranjem pristupalo se RAM lokacijama. U slučaju većeg RAM-a, pristupanje lokacijama iznad 255 je moguće samo preko posebnog DPR 16-to bitnog registra. Ovaj registar treba da sadrži adresu željene RAM lokacije. Većina instrukcija ovog mikrokontrolera izvršava se za 1 mašinski ciklus (osim instrikcija za uslovni skok). Ali, svaki ciklus se sastojao od 12 faza, što predstavlja nepotrebno usporavanje jer svim instrukcijama nisu potrebne sve faze da bi bile realizovane. Kasnije, drugi proizvodjači ove familije mikrokontrolera će smanjiti broj faza na 4, a pojaviće se i mikrokontroleri ove familije sa jednofaznim mašinskim ciklusom, ali će se odstupiti od toga da se većina instrukcija izvršava za jedan mašinski ciklus, već za onoliko koliko je potrebno, što efikasnije troši procesorsko vreme.

Do danas se pojavio veliki broj familija mikrokontrolera, i veliki broj proizvodjača. Po popularnosti, zastupljenosti na tržištu i korišćenju u najrazličitijim uredjajima, možemo izdvojiti kao najinteresantnije sledeće 8-o bitne mikrokontrolere:

- PIC familiju mikrokontrolera proizvođača Microchip;
- MC908 familija Motorole i
- AVR familija Atmela.

PIC je nastao usavršavanjem jednog čipa koji je imao pomoćnu ulogu u jednom projektu. Vrlo brzo je postao popularan, pre svega dobrim nastupom Microchip-a na tržištu, masovnom i raznovrsnom ponudom, i besplatnim osnovnim alatima za razvoj softvera.

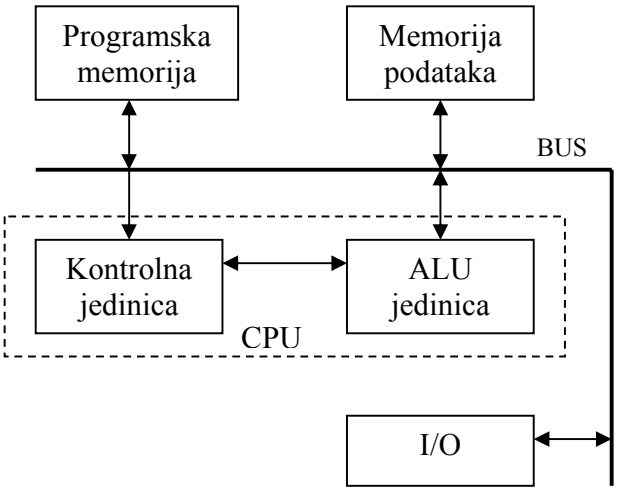
MC908 je nastavak familije MC68HC05 mikrokontrolera. Sadrži sve instrukcije MC68HC05, i u tom smislu su kompatibilni, a ima i značajan broj novih instrukcija i adresnih modova. Mikroračunarske komponente, kao i ovu familiju proizvodi motorolina firma Freescale. Pristup tržištu je tradicionalan, svojstven Motoroli: nudi najbolji kvalitet i najbolju podršku, ali zanemaruje vrlo značajnu grupu hobista. Ova grupa je značajna, jer je samo jedan korak od hobija do profesionalnog bavljenja aplikacijama mikrokontrolera. Familija MC908 sadrži podfamiliju MC9S08 koja je bazirana na unapređenim tehnologijama, tako da su mikrokontroleri iz ove podfamilije kompleksniji i 2.5 puta brži. Oznakom MC9(S)08 podrazumeva se cela familija MC908 sa podfamilijom MC9S08.

Atmel se afirmisao kao proizvođač familije mikrokontrolera baziranih na 8051 familiji. Uneo je niz poboljšanja i produzio život legendarnom 8051. AVR je potpuno nova familija, kojom se pokušava uvesti RISC arhitektura u mikrokontrolersku tehnologiju. Koliko god je ova ideja privlačna, zbog mnogih ograničenja koja postoje kod 8-o bitnih mikrokontrolera, sama realizacija seta instrukcija čini da AVR nije dao ono što bi se očekivalo od jedne RISC arhitekture.

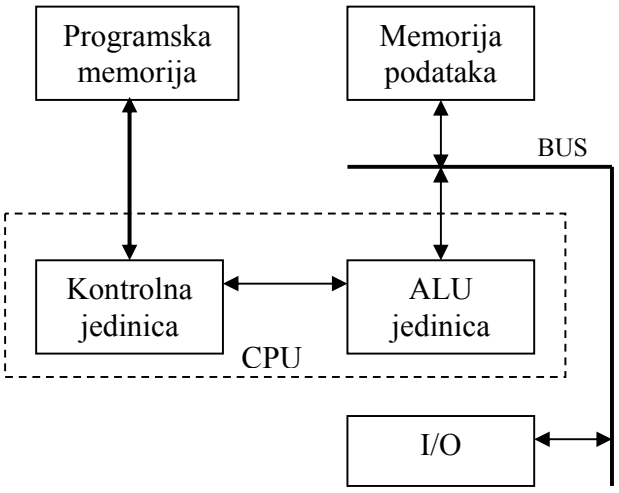
Fon Nojmanova i Harvardska arhitektura mikrokontrolera

Fon Nojmanova (von Neumenn) arhitektura mikrokontrolera ima jednu magistralu (adresna magistrala, magistrala podataka, signal smera podataka i signal validnosti podataka na magistrali) koja povezuje sve interne module mikrokontrolera. Ova arhitektura pruže izvesnu jednostavnost upravljanja resursima

mikrokontrolera, dozvoljava kreiranje efikasnijeg skupa naredbi i dozvoljava jednaku pristupačnost lokacijama sa kodom programa kao i podacima u RAM-u. Osim toga, veze u mikrokontroleru zauzimaju značajnu površinu silicijuma, pa ako njih ima manje, na istoj površini može stati više korisnog hardvera. Mana ove arhitekture je mogućnost zagušenja magistrale, što ograničava brzinu mikrokontrolera.



Harvardska (Harvard) arhitektura sadrži dve magistrale. Jedna povezuje programsku memoriju i kontrolnu jedinicu CPU-a mikrokontrolera, a druga povezuje ALU (aritmetičkologičku) jedinicu i RAM memoriju mikrokontrolera. Ova arhitektura zahteva nešto složeniji mehanizam upravljanja resursima mikrokontrolera, set instrukcija ima različite (i posebne) instrukcije za pristup podacima u RAM-u i za pristup podacima (tabele) u programskoj memoriji. Veze zauzimaju veću površinu silicijuma, na uštrb drugog hardvera mikrokontrolera. Prednost ove arhitekture je u mogućnosti da dok se jedna instrukcija izvršava (tada je zauzeta magistrala prema RAM-u ili nekoj herdverskoj jedinici mikrokontrolera), paralelno se iz programske memorije čita kod i eventualno (ako postoji) parametar sledeće naredbe. Dakle, prevashodni cilj koji se želi postići ovom arhitekturom je povećanje brzine rada mikrokontrolera.



Harvardsku arhitekturu imaju PIC i AVR mikrokontroleri, a Fon Nojmanovu imaju svi motorolini 8-o bitni mikrokontroleri, pa i MC9(S)08.

Razlike između ove dve arhitekture nije presudna kod 8-o bitnih mikrokontrolera. To je zato što se prednosti Harvardske arhitekture ne mogu maksimalno iskoristiti kod 8-o bitnih mikrokontrolera jer je širina magistrale podataka prema programskoj memoriji 12, 14 ili maksimalno 16 bita, a trebala bi biti najmanje 24 bita. Zašto 24 bita? Zato što je za tipičnu naredbu potrebno 8 bita za kod naredbe i 16 bita za adresu memorijske lokacije u 64KB memorijskom prostoru. Medjutim, nisu sve naredbe toliko

zahtevne, postoji znatan broj naredbi koje zahtevaju 2 bajta (relativni skokovi, punjenje registra CPU-a konstantom i t. d.), a takođe nije zanemarljiv ni broj naredbi koji ima samo jedan bajt (operacije nad registarima CPU-a). Statistika kaže da je prosečna dužina naredbe 8-o bitne CPU jedinice 2 bajta. Ako bi, dakle, želeli da izvučemo korist od Hardvarske arhitekture, morali bi da žrtvujemo trećinu programske memorije. Ovako neefikasno korišćenje programske memorije (u daljem tekstu flash-a) je prevelik luksuz za 8-o bitne mikrokontrolere.

Fon Nojmanova arhitektura nema ove probleme, u njoj svaka naredba zauzima flash onoliko koliko je potrebno. Problem zagušenja magistrale se rešava na razne načine, a kod MC9S08 je on ublažen tako što kad god je magistrala slobodna (za naredbe nad registrima CPU-a) čita se kod sledeće naredbe. Tako prosečna instrukcija kod MC9S08 ima dužinu od 2 bajta i izvršava se u proseku za 3 mašinska ciklusa.

PIC i AVR na različite načine rešavaju probleme koje proizvodi neefikasno apliciranje Harvardske arjitekture.

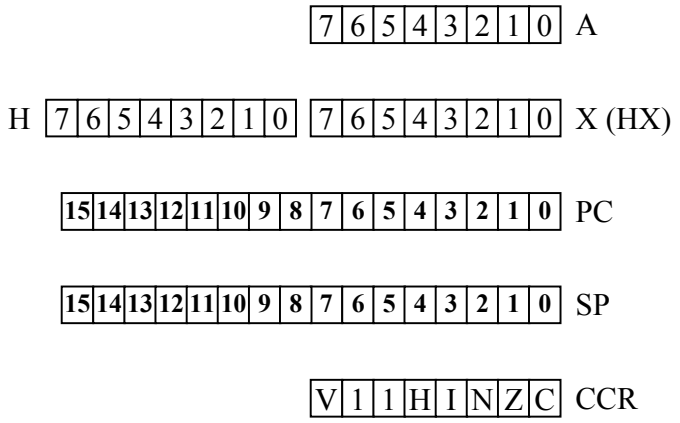
PIC ima najveću širinu magistrale prema flash-u od 16 bita. To znači da nije moguće imati parametar naredbe dužine 16 bita. Zato PIC adresira memoriju u okviru stranice od 256 bajtova. Iz tog razloga postoje bitovi koji adresiraju stranicu memorije. Broj ovih bitova zavisi od veličine RAM-a. Dakle, za proizvoljan pristup memoriji treba upotrebiti dve naredbe, jednom da bi se odredila stranica, a drugom da bi se pristupilo lokaciji u okviru stranice. Za pristup drugoj lokaciji u okviru iste stranice nije potrebno da se stranica ponovo odredjuje. Ovakav pristup zahteva da se aktiviranjem interrupt-a zapamte bitovi koji odredjuju trenutni stranicu sa kojom se radi. Najbolje mesto za ove bitove je statusni registar CPU-a koji ionako mora da se pamti kod svakog interrupt-a. Problem nastaje onda kada je RAM veliki a nema mesta u statusnom registru za još bitova koji određuju radnu stranicu RAM-. Statusni registar ima 3 bita koja su neiskorišćena, pa ako je RAM veći od 2KB onda je potrebano 4 bita za adresiranje 16 stranica (4KB), koja se u tom slučaju smeštaju u poseban registar. Ali tu nije kraj problemima, interrupt kod PIC-a smešta samo adresu povratka na stek, pa je potrebno na početku interrupt procedure zapamtiti negde statusni registar i registar sa bitovima stranice sa kojom se radi, uz podrazumevani radni registar. Na kraju interrupt procedure treba sve ranije zapamćeno obnoviti. Ovo onemogućuje da se u jednom interruptu, na jednostavan način (kao što je moguće kod MC9(S)08), omogući aktiviranje drugog interrupt-a. Problem sa čitanjem tablica u flash-u rešava se korišćenjem posebnog registra koji adresira flash i može da se inkrementira. Indirektno adresiranje je takođe komplikovano i nestandardno tako da nije primenljivo na slučaj adresiranja u odnosu na stek pointer. Zato i nije moguće stavljati podatke na stek, pa je komplikovano stvaranje lokalnih varijabli u višim programskim jezicima (C) što dovodi do manje efikasnog korišćenja RAM-a. Sve ovo pokazuje da se neke naredbe mikrokontrolera Fon Nojmanove arhitekture moraju realizovati sa više naredbi PIC-a. To, naravno, usporava rad PIC-a, iako je cilj korišćenja Hardvarske arhitekture bio suprotan.

AVR ima unapređenu Hardvarsku arhitekturu. On ima prostor za podatke u 2 dela. Pored uobičajenog RAM-a, on ima odvojen dvoportni RAM od 32 bajta. Kod dvoportnog RAM-a može se u isto vreme pristupiti dvema lokacijama, i ALU može da vršu operacije nad njima. ALU jedinica AVR-a ima 32 registra, izdvojeni programski brojač, stek pointer i statusni registar u adresnom prostoru. Naredbe su dvoadresne: u jednoj naredbi (na primer sabiranje) mogu se nalaziti adrese dva registra, nad njima se vrši navedena operacija, a rezultat se smesta u prvi navedeni registar. Širina naredbe je 16 bita, što znači da ni kod AVR-a ne mogu da dođu do izražaja dobre osobine Hardvarske arhitekture. Zapravo, kod malih programa kojima ne treba više od 32 bajta za podatke, nema bržeg 8-o bitnog mikrokontrolera. Za druge slučajeve situacija se značajno menja. Postoje samo dve naredbe za direktan pristup RAM-u. To je naredba kojom se sadržaj iz bilo koje lokacije RAM-a prebacuje u neki od registara, ili obrnuto iz registra u RAM lokaciju. Svaka od ovih naredbi troši po 4 bajta. Prva dva bajta sadrže kod naredbe i adresu registra, a druga dva adresu RAM lokacije. Ne postoje naredbe koje mogu da modifikuju vrednost RAM lokacije, tipa ***inc [Adresa]***. Za neke mikrokontrolere ovo je standardna naredba, ali za njenu realizaciju AVR-u je potrebno 3 naredbe, 10 bajtova i 5 mašinskih ciklusa. Najviša 6 registra čine

tri 16-to bitna indeksna registra, preko kojih je moguć pristup RAM-u sa ofsetom od samo 63. Moguće je prebaciti sadržaj stek pointera u neki indeksni registar i tako iskoristiti stek za lokalne varijable. Dvoadresne naredbe sadrže 10 bitova za adresiranje 2 od 32 registra, tako da za kod dvoadresne naredbe ostaje samo 6 bitova. U stvari ne 6 nego manje jer se moraju ostaviti neke neponovljive kombinacije kodova za ostale naredbe. Naredba punjenje registra konstantom (neposredno adresiranje), mora da sadrži 8 bita vrednosti konstante, 5 bita adrese registra, i za kod ostaje 3 bita. Ovo je premalo, pa je kod AVR-a moguće konstantom puniti samo viših 16 registara. Ovakvih izuzetaka, kao rezultat kompromisa sa malom širinom naredbe (16 bita), ima mnogo. Zbog toga je ofset kod indirektnog adresiranja ograničen na 63. Ne postoji potpun skup naredbi za uslovno granjanje, pa se programer u asembleru mora služiti trikovima. Umesto da svi tipovi AVR-a imaju isti set instrukcija, to nije slučaj. Sve u svemu, pisanje na asembleru je najnapornije za AVR. Za pohvalu je naredba kompariranja sa bitom prenosa (kao da se radi o oduzimanju) jer olakšava upoređivanje višebajtnih varijabli. Doduče, ovakva naredba kompariranja je morala da se uvede, jer ne postoji oduzimanje kod AVR koje ne modifikuje neki registar. Sama arhitektura olakšava implementaciju viših programskih jezika, ali ne postoji način da se AVR učini stedljivijim kada je u pitanju trošenje programske memorije. AVR je ubedljivo nejneefikasniji kada je u pitanju dužina generisanog koda.

Arhitektura CPU-a i naredbe MC9S08 familije

Već je rečeno da MC9(S)08 ima Fon Nojmanovu arhitekturu. CPU (tačnije njen ALU deo) logički ima sledeći izgled:



Značenje pojedinih registara je:

A – 8-o bitni akumulator;
X – 8-o bitni indeksni registar koji postoji radi kompatibilnosti sa 68HC05;
H – 8-o bitni registar koji predstavlja viši bajt indeksnog 16-to bitnog registra;
HX – 16-to bitni indeksni registar sastavljen kao par H i X registra. Ovo je pravi indeksni registar familije MC9(S)08;
PC – 16-to bitni programski brojač;
SP – 16-to bitni stek pointer;
CCR – 8-o bitni statusni registar.

Bitovi statusnog registra se uvek postavljaju kada se vrši neka operacija nad podacima. Oni se postavljaju čak i kada se neki podatak smesti u akumulator ili registar X, kao i obrnuto, kada se sadrzaji ovih registara smestaju u RAM ili neki registar mikrokontrolera. Ovo je stara praksa Motorole, dok kod Intela (i još nekih proizvođača) pomeranje podataka iz memorije u CPU i obrnuto ne menja bitove

statusnog registra. Oba pristupa su dobra, svaki na svoj način. Tako na primer, samim smestanjem podatka u akumulator CPU-a MC9(S)08 postavljaju se bitovi u statusnom registru i odmah se zna da li je podatak različit od nule, kao i njegov znak. U drugom slucaju (Intelovom), ako su potrebne informacije o podatku, potrebna je dodatna naredba komparacije sa nulom. Ali, kada se vrše aritmetičke operacije nad višebajtnim podacima, pomeranje pojedinih bajtova iz akumulatora u RAM briše bit prekoračenja kod MC9(S)08, pa ovaj bit treba testirati pre pomeranja bajta nad kojim je izvršena poslednja operacija. Ovaj problem ne postoji u Intelovoj metodologiji postavljanja statusnih bitova.

Statusni registar sadrži sve bitove potrebne za potpuni skup naredbi uslovnih skokova, što nije slučaj sa AVR-om. Pojedini bitovi imaju sledeće značenje:

C – bit prenosa, ili bit pozajmice. Naredba sabiranja dodaje na akumulator sadržaj adresirane memorijske lokacije i rezultat ostavlja u akumulatoru. Bit C će se postaviti na log. 1 ako rezultat bude veći od 255. Kod oduzimanja, C bit će se postaviti na log. 1 ako je vrednost koja se oduzima od vrednosti u akumulatoru veća. Sabiranje sa prenosom i oduzimanje sa pozajmicom na isti način generiše C bit. Kod naredbe kompariranja, bit C se postavlja na isti način kao kod oduzimanja bez pozajmice.

Ako se podatak (jedan bajt) sifta levo ili desno, bit C dobija vrednost bita koji je ispao prilikom siftanja. Isto se dešava prilikom rotacije, ali sa tom razlikom što predhodna vrednost bita C dolazi na upražnjeno mesto bita siftanog podatka. Naredba rotacije bajta podatka se koristi prilikom siftovanja višebajtnih podataka.

Uslovna naredba relativnog skoka koji zavisi od vrednosti jednog od bitova adresiranog bajta, postavlja C bit na vednost testiranog bita.

Naredba koja od neke vrednosti bajta pravi negativnu vrednost, postavlja C bit na isti način kao da se vrsi oduzimanje te vrednosti od nule. Naredba komplementiranja uvek postavlja C bit na log. 1.

Naredba celobrojnog deljenja setuje bit C ako se deli sa nulom (što izaziva prekoračenje). Naredba celobrojnog mnozenja uvek kliruje bit C.

Statusni bit C se može naredbom ***clc*** postaviti na log. 0, ili se sa ***sec*** može postaviti na log. 1.

Z – bit statusnog registra se postavlja na log. 1 posle aritmetičkih ili logičkih naredbi koja kao rezultat daju vrednost 0, u ostalim slučajevima ostaje klirovan, odnosno log. 0. Pomeranje bajta iz ili u CPU mikrokontrolera, takodje postavlja ovaj statusni bit. Uporedjenje (komparacija) dva bajta setuje Z bit ako su ovi bajtovi jednakih vrednosti.

N – bit statusnog registra sadrži informaciju o znaku bajta, bilo posle neke operacije bilo posle pomeranja iz ili u CPU mikrokontrolera. Ovaj bit, ustvari, sadrži MSB (najstariji) bit bajta.

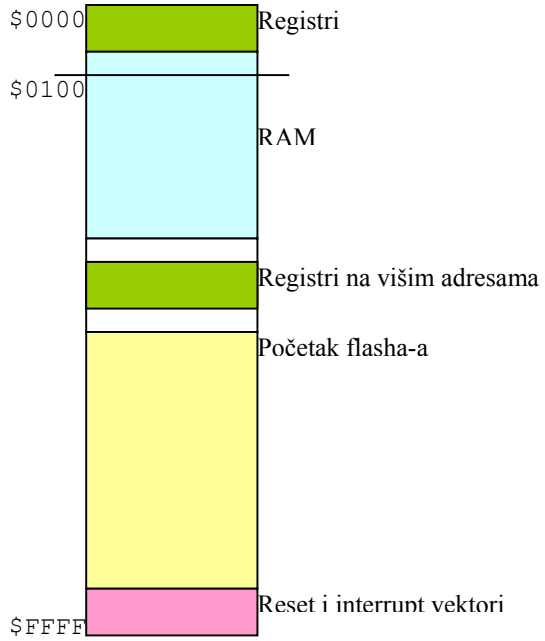
I – bit statusnog registra je maskirajući bit za sve interrupt-e mikrokontrolera. Ako je ovaj bit na log. 1 onda su svi interrupt-i zabranjeni, i obrnuto, ako je I bit na log. 0 svi interupti su omogućeni. Naime, svaki interrupt ima svoju lokalnu masku, a statusni I bit ima ulogu globalne maske za sve interrupt-e. Posle reseta I bit je postavljen na log. 1, što znači da su svi interrupt-i zabranjeni. Takodje, kada se aktivira neki interrupt I bit se automatski postavlja na log. 1, čime onemogućava da se interrupt-i do u beskonačnost aktiviraju. Kao posledica toga interrupt je zabranjen u proceduri koja ga opslužuje. Statusni I bit se može postaviti na log. 0 ili log. 1 naredbama ***cli*** i ***sei*** respektivno.

H – ovo je takodje bit prenosa, kao C statusni bit, ali se odnosi na prenos sa trećeg na četvrti bit, odnosno sa nižeg nibla na viši (nibl je 4 bita). Koristi ga naredba decimalnog podešavanja, ali se statusni bit H može koristiti i u naredbama uslovnog relativnog skoka.

V – ovaj statusni bit je bit prekoracenja prilikom korišćenja označenih broja u dvojičnom komplementu. Označeni bajt ima vrednosti u opsegu od –128 do +127. Prekoračenje ovih vrednosti postavlja statusni bit V na log. 1. Na primer sabiranjem broja 100 ($64) i 50 ($32) dobija se $96, što ne setuje statusni bit C, ali setuje bit V. Dakle, ako navedene brojeve smatramo neoznačenim, onda će se za proveru prekoračenja koristiti statusni bit C, a ako ih smatramo osnačenim, anda treba, u istu svrhu koristiti statusni bit V. I stvarno, $96 je 150 što je ispravan rezultat za sabiranje neoznačenih brojeva, dok $96 kao označen broj iznosi –106, što je netačan rezultat zbog prekoračenja.

Memorijaska mapa

Saglasno Fon Nojmanovoj arhitekturi, memorijska mapa mikrokontrolera MC9(S)08 predstavlja kontinualan i jedinstven memorijski prostpr od adrese $0000 do adrese $FFFF. Memorijska mapa se u detaljima razlikuje od tipa do tipa, ali njen opšti izgled je na sledećoj slici.



Zelenom bojom su označeni registri mikrokontrolera. Koliko god da ih ima (zavisno od tipa), nisu svi smešteni na nultoj strani. Nulta strana je memorijski prostor od adrese $0000 do adrese $00FF. Uvek se ostavlja makar 128 slobodnih lokacija na nultoj strani, jer naredbe koje modifikuju sadržaj memorijske lokacije to rade samo nad lokacijama nulte strane. Na nultoj strani su registri koji upravljaju portovima, kao i registri nekih integrisanih modula koji se najčešće koriste. Ostatak registara je izmešten na drugo mesto, najčešće iza RAM-a.

Programska memorija se uvek završava na lokaciji $FFFF, a početna lokacija zavisi od veličine programske memorije. Lokacije na samom kraju programske memorije (najčešće njih 48) su rezervisane za interrupt vektore i na samom vrhu ($FFFE i $FFFF) je reset vektor. Neposredno ispred prostora za interrupt vektore nalaze se dva neizmenljiva registara (definišu se programiranjem flash-a). Jednim se može zaštititi deo programske memorije od slučajnih ili namernih izmena, a drugi služi da odredi opcije zaštite programske memorije od neovlašćenog pristupa. Oba registra se sa resetom kopiraju u odgovarajuće registre MCU-a. Pored ovih registara (ustvari lokacija u programskoj memoriji) posoji i ključ od 8 bajta kojim se žaključava programska i RAM memorija, i tako štiti od neovlašćenog pristupa. Ključ se može zaobići samo brisanjem cele programske memorije.

Asemblerske naredbe sa implicitnim (IMP) adresnim modom

Kada se podatak već nalazi u nekom od ranije pobrojanih CPU registara, on se može modifikovati, izmedju ostalih, i naredbama koje pripadaju implicitnom adresnom modu. Implicitna naredba nema parametar, a svojim kodom odredjuje registar nad kojim se ona izvršava (registar je implicitno sadržan u kodu naredbe). Implicitne naredbe jednako postoje za akumulator (A) i indeksni registar (X). One su prikazane u sledećoj tablici:

A	X	ciklusa	opis
asla	aslx	1	Aritmetičko ili logičko (isto je) pomeranje levo, MS bit ide u C statusni bit, a na mesto LS bita dolazi 0
asra	asrx	1	Aritmetičko pomeranje desno, na upražnjeno MS mesto, ponovo se postavlja MS bit, čime se čuva znak vrednosti, a LS bit ide u C statusni bit.
clra	clrx	1	Registar dobija vrednost $00
coma	comx	1	Komplementiraju se (invertuju se) svi bitovi registra
deca	decx	1	Umanjuje za 1 vrednost registra.
inca	incx	1	Uvećanje za 1 vrednosti registra.
lsla	lslx	1	Isto kao asla i aslx.
lsra	lsrx	1	Logičko pomeranje udesno, LS bit ide u C statusni bit, a na upražnjeno mesto MS upisuje se log. 0.
nega	negx	1	Promena znaka vrednosti u registru primenom dvojičnog komplementiranja.
psha	pshx	2	Stavlja vrednost registra na stek.
pula	pulx	3	Skida vrednost sa vrha steka i postavlja je u registar.
rola	rolx	1	Rotira sadržaj registra ulevo, tako što na upražnjeno LS mesto dolazi vrednost C statusnog bita, a istovremeno MS bit se prenosu u C statusni bit.
rora	rorx	1	Rotira sadržaj registra udesno, tako što na upražnjeno MS mesto dolazi vrednost C statusnog bita, a istovremeno LS bit se prenosu u C statusni bit.
tsta	tstx	1	Testira vrednost registra kao da ga komparira sa $00.

Ostale naredbe implicitnog adresnog moda su:

naredba	ciklusa	opis
bgnd	5	Zaustavlja rad mikrokontrolera, prevodeći ga u aktivni BDM i čeka BDM naredbe.
clc	1	Statusni bit C postavlja na log. 0
cli	1	Globalnu interrupt masku postavlja na log. 0, čime se globalno omogućuju interrupt-i
clrh	1	H registar dobija vrednost $00
daa	1	Decimalno podešavanje vrednosti u akumulatoru. Primenjuje se neposredno posle sabiranja (sa ili bez prenosa) koje je binarno, da rezultat pretvori u dve BCD cifre.
div	6	Celobrojno deljenje 16-to bitne vrednosti u reg. paru HA sa vrednošću u reg. X. Rezultat je u akumulatoru a ostatak u reg. H.
mul	5	Množenje vrednosti akumulatora i indeksnog registra X, rezultat je u reg. paru XA.
nop	1	Naredba bez efekta.
nsa	1	Niži i viši nibl (4 bita) u akumulatoru zamenjuju mesta.
pshh	2	Stavlja vrednost H registra na stek.
pulh	3	Skida vrednost sa vrha steka i postavlja je u registar H.
rsp	1	Resetuje stek pointer, postavljajući njegov niži bajt na vrednost $FF. Nema efekta na viši bajt stek pointera, i postoji samo zbog kompatibilnosti sa 68HC05.
rti	9	Naredba povratka iz interrupt-a uz obnovu svih registara osim H registra.
rts	6	Naredba povratka iz podprograma.
sec	1	Statusni bit C postavlja na log. 1
sei	1	Globalnu interrupt masku postavlja na log. 1, čime se globalno onemogućuju interrupt-i
stop	2	Zaustavlja rad mikrokontrolera i postavlja statusni I bit na log. 0, omogućujući globalno interrupt-e.
swi	11	Softversko izazivanje interrupt-a, jednako kao da ga je izazvao neki hardverski

		događaj.
tap	1	Sadržaj akumulatora prebacuje u statusni registar.
tax	1	Sadržaj akumulatora prebacuje u indeksni registar X.
tpa	1	Sadržaj statusnog registra prebacuje u akumulator.
tsx	2	Vrednost stek pointra, uvecanu za 1 prebacuje u indeksni registar HX.
txa	1	Vrednost indeksnog registra X prebacuje u akumulator.
txs	2	Vrednost indeksnog registra HX, umanjenu za 1 i prebacuje u stek pointer.
wait	2	Zaustavlja rad mikrokontrolera i postavlja statusni I bit na log. 0, omogućujući globalno interrupt-e.

Za vreme izvršenja naredbi implicitnog adresnog moda, transverzala podataka je slobodna (izuzev za naredbe koje salju sadržaj reg. na stek, ili obrnuto), tako da se tada ona koristi za čitanje koda sledeće naredbe. Dakle, svako koriscenje ovih naredbi štedi jedan mašinski ciklus, pa su zato one kraće za taj jedan ciklus u odnosu na iste naredbe kod MC908.

Već iz pregleda ovih naredbi vidi se da indeksni registar X nije zaista indeksni registar. Kod MC9(S)08 to je 16-to bitni registar HX.

Postoje 3 naredbe koje zaustavljaju rad mikrokontrolera. Naredba WAIT izaziva zaustavljanje čitanje sledeće naredbe i omogućuje globalno interrupt-e. Ukida se klok CPU-a, dok svi ostali resursi rade normalno. U WAIT modu potrošnja je smanjena, i može se još smanjiti ako se disejbluju pojedini moduli mikrokontrolera. Iz WAIT moda se izlazi na RESET, IRQ, KBI (Keyboard interrupt) ili RTI (real-time interrupt). KBI je u vezi sa jednim brojem I/O pinova koji su namenjeni vezivanju za tastaturu, tako da kada se aktivira neki taster može se izazvati interrupr. STOP naredba, takođe zaustavlja mikrokontroler, ali u jednom od tri stanja: STOP1, STOP2 ili STOP3. Izbor nivoa se vrši setovanjem bitova u određenim registrima pre naredbe STOP. U STOP1 nivou prekinut je rad oscilatora, a potrošnja je najmanja i iz njega se izlazi na RESET ili IRQ. U STOP2 i STOP3 nivou rade neki moduli mikrokontrolera i iz njih se izlazi kao iz WAIT moda, mada detalji zavise od konkretnog tipa mikrokontrolera. Ako interrupt izvodi mikrokontroler iz WAIT ili STOP stanja, onda se posle završetka interrupt procedure program vraća na naredbu koja neposredno sledi iza naredbe WAIT ili STOP.

BGND naredba služi samo u fazi razvoja softvera. Ona ne treba da ostane u kodu završenog programa. Ova naredba zaustavlja rad mikrokontrolera i aktivira BDC (pozadinski dibag kontroler) koji se nalazi u svim tipovima mikrokotrolera familije MC9S08. BGND naredbom se BDC aktivira u aktivnom BDM (pozadinski dibag mod) koji preuzima kontrolu nad CPU-om mikrokontrolera i preko jednog pina uspostavlja vezu sa spoljnim interfejs kolom koje je vezano na PC. Aktivni BDM dozvoljava da se mogu čitati i postavljati vrednosti registara CPU-a, takodje se mogu čitati i upisivati memorijske lokacije ili registri mikrokontrolera. Vrlo je važno da se u ovom modu mogu izvršavati naredbe programa korak po korak, ili da se nastavi kontinualno izvršavanje programa.. Ovaj mod se koristi i za progamiranje flash-a mikrokontrolera. Uzgred, napomenimo da BDC može biti uveden i u nenamitljivi BDM kada mikrokontroler normalno izvršava program, a istovremeno BDC komunicira sa PC-jem. U tom modu je moguče čitati proizvoljne lokacije memorije u realnom vremenu, a da ni za malo nije usporen rad mikrokontrolera. U ovom (nenametljivom) modu, mogu se zadati adrese tačaka prekida. Kada program dodje do tačke prekida, onda MCU ulazi u aktivan BDM, kada se mogu procitati svi registri CPU-a, ili bilo koje memorijske lokacije. Od prekidne tačke, može se nastaviti izvršenje programa korak po korak, ili pokrenuti izvršavanje programa kontinualno, sve do ponovnog nailaska na prekidnu tačku (ako ona i dalje postoji).

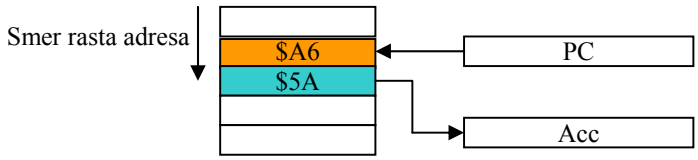
Najveći broj naredbi vrši neku matematičku ili logičku operaciju izmedju vrednosti u akumulatoru i adresirane memorijske lokacije. Takodje, postoje i naredbe kojima se vrednosti CPU registara salju u memorijsku lokaciju, ili obrnuto.

Adresni modovi

Osim opisanog implicitnog adresnog moda, postoje i drugi adresni modovi. Adresni mod odredjuje način na koji se definiše adresa memorijske lokacije kojoj se pristupa. Ti adresni modovi su:

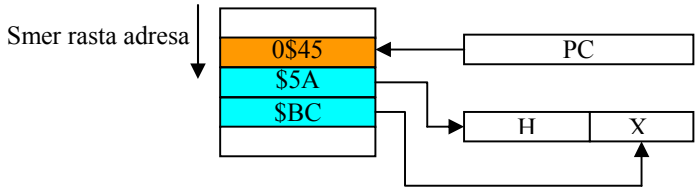
1. **IMM8**: neposredni podatak je 8-o bitni i nalazi se neposredno iza koda naredbe. To znaci da je podatak kojem se pristupa u flash-u (osim ako se program ne izvršava iz RAM-a ili EEPROM-a), i da ga adresira programski brojač.
2. **IMM16**: neposredni podatak je 16-to bitni i nalazi se neposredno iza koda naredbe. To znaci da je podatak kojem se pristupa u flash-u (osim ako se program ne izvršava iz RAM-a ili EEPROM-a), i da ga adresira programski brojač.
3. **DIR**: direktno adresiranje memorije. Iza koda naredbe sledi 8-o bitna adresa memorijske lokacije. Ovim adresnim modom se pristupa nultoj stranici.
4. **EXT**: prošireno adresiranje memorije. Iza koda naredbe sledi 16-to bitna adresa memorijske lokacije. Ovim adresnim modom se pristupa bilo kojoj memorijskoj lokaciji u celom adresnom prostoru.
5. **IX**: indirektno adresiranje bez indeksa. Iza koda naredbe nema nikakvog parametra. Adresa memorijske lokacije kojoj se pristupa je sadrzana u indeksnom registru HX.
6. **IX+**: indirektno adresiranje bez indeksa. Iza koda naredbe nema nikakvog parametra. Adresa memorijske lokacije kojoj se pristupa je sadrzana u indeksnom registru HX. Posle izvršenja naredbe indeksni registar se uvećava za 1.
7. **IX1**: indirektno indeksno adresiranje sa 8-o bitnim inteksom. Indeks je iza koda naredbe, a adresa je zbir vrednosti u indeksnom registru HX i 8-o bitnog indeksa.
8. **IX1+**: indirektno indeksno adresiranje sa 8-o bitnim indeksom. Indeks je iza koda naredbe, a adresa je zbir vrednosti u indeksnom registru HX i 8-o bitnog indeksa. Posle izvršenja naredbe indeksni registar se uvećava za 1.
9. **IX2**: indirektno indeksno adresiranje sa 16-to bitnim indeksom. Indeks je iza koda naredbe, a adresa je zbir vrednosti u indeksnom registru HX i 16-to bitnog ineksa.
10. **REL**: relativno adresiranje. Iza koda naredbe je 8-o bitna relativna adresa. Ovaj način adresiranja koriste naredbe uslovnih i nekih bezslovnih skokova. Relativna adresa je označena (u opsegu je od –128 pa do 127) i po završetku naredbe, ako ima uslova za skok, ona se sabira sa vrednošću programskog brojača. Drugim rečima: relativna adresa se sabira sa adresom na kojoj je sledeća naredba.
11. **SP1**: indirektno indeksno adresiranje sa 8-o bitnim indeksom. Indeks je iza koda naredbe, a adresa je zbir vrednosti stek pointera i 8-o bitnog indeksa.
12. **SP2**: intirektno indeksno adresiranje sa 16-to bitnim indeksom. Indeks je iza koda naredbe, a adresa je zbir vrednosti stek pointera i 16-to bitnog indeksa.

Objašnjenje IMM8: ldaa $5A



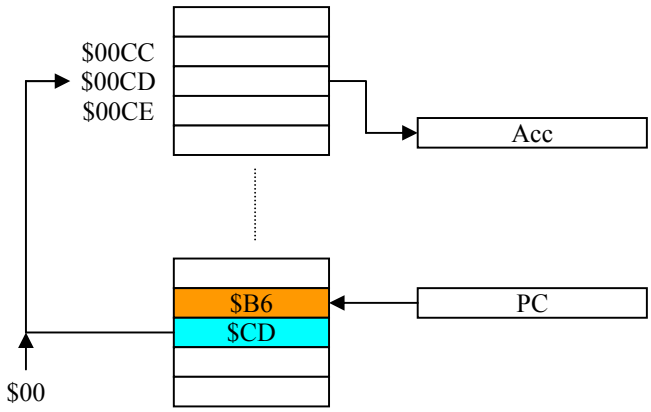
Kod za ldaa u ovom adresnom modu je $A6.

Objašnjenje IMM16: ldhx $5ABC



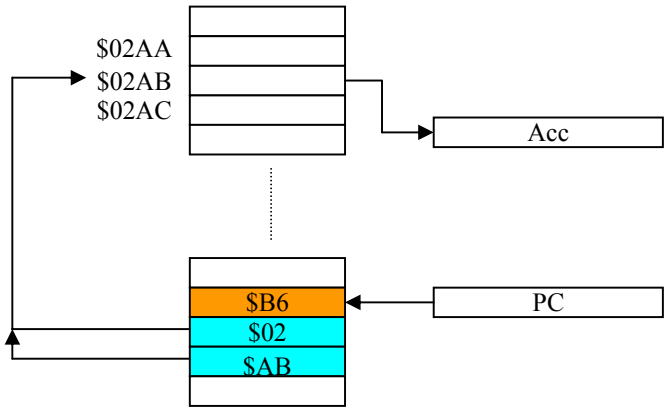
Kod za ldhx u ovom adresnom modu je $45.

Objašnjenje DIR: ldaa [$CD]



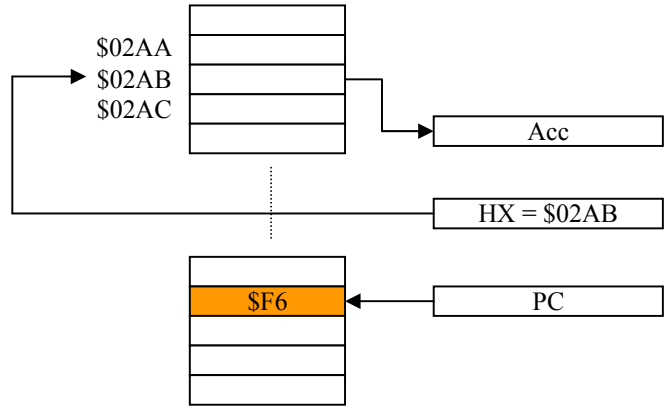
Kod za ldaa u ovom adresnom modu je $B6

Objašnjenje EXT: ldaa [$02AB]



Kod za ldaa u ovom adresnom modu je $B6

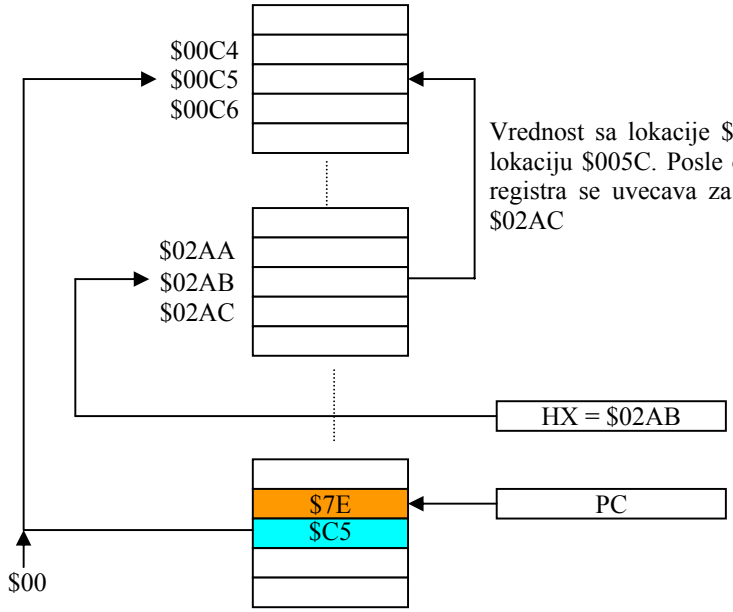
Objašnjenje IX: ldaa x[]



U ovom slučaju HX registar se koristi za indirektno adresiranje. U njemu je adresa lokacije kojoj se pristupa $02AB. Prilikom ovog adresiranja nema indeksa.

Kod za ldaa u ovom adresnom modu je $F6

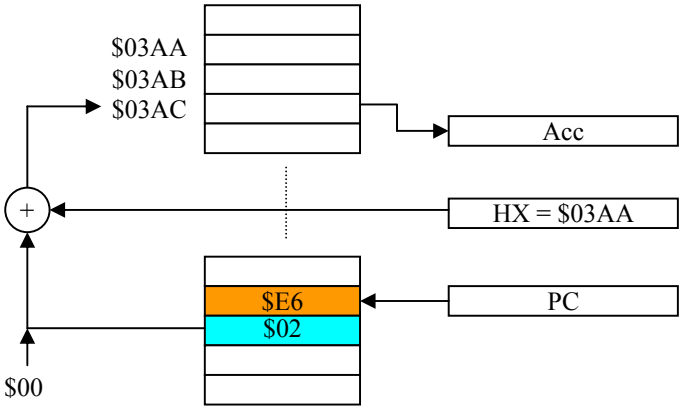
Objašnjenje IX+: mov x+[$C5]



Vrednost sa lokacije $02AB se prenosi u lokaciju $005C. Posle ovoga vrednost HX registra se uvecava za 1, tako da postaje $02AC

Kod za mov u ovom adresnom modu je $7E

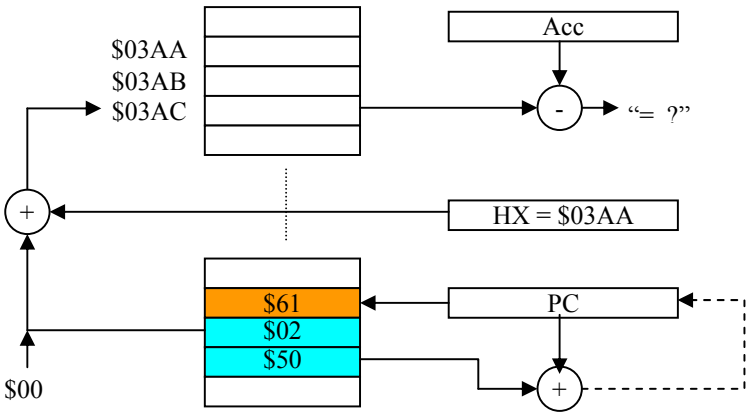
Objašnjenje IX1: ldaa x[$02]



Ovo je slučaj indirektnog indeksnog adresiranja sa 8-o bitnim indeksom. Bazna adresa je u indeksnom registru HX, a indeks je parametar odmah iza koda naredbe . Na primer: bazna adresa može biti adresa nekog niza bajtova, a naredbom u primeru, vrši se pristup trećem elementu niza. Taj element ima indeks 2.

Kod za ldaa u ovom adresnom modu je $E6.

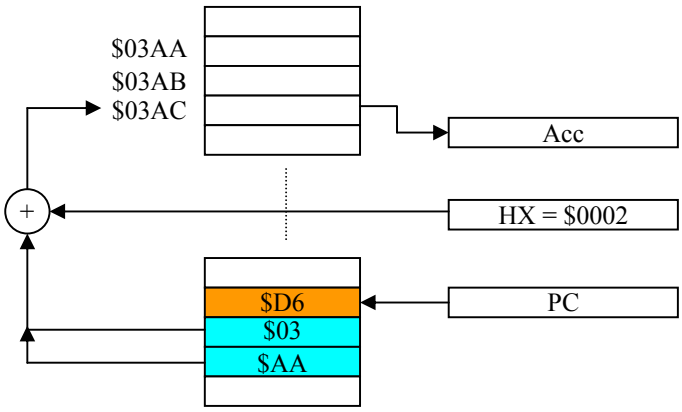
Objašnjenje IX1+: cbeq x+[$02],$50



Ako je vrednost u akumulatoru jednaka vrednosti u lokaciji na adresi $03AC, onda se ostvaruje relativni skok za $0050 lokacija flash-a napred, inace se izvršava naredba koja sledi iza **cbeq**. Ako je skok unazad, onda je relativna adresa negativna. Za jednako dalek skok unazad, naredba bi bila **cbeq** x+[$02],$B0, a programski brojač bi se uvećao za $FFB0. Posle izvršenja ove naredbe, indeksni registar HX se uvećava za 1 i njegova vrednost postaje $03AB.

Kod za cbeq u ovom adresnom modu je $61.

Objašnjenje IX2: ldaa x[$03AA]



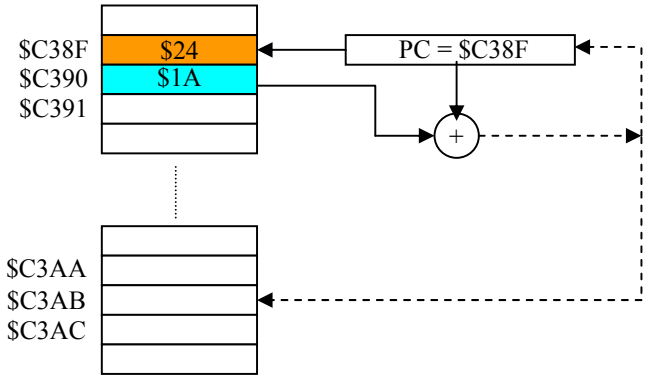
I ovde se radi o indirektnom indeksnom adresiranju. Razlika je u tome što je bazna adresa može biti odmah iza koda naredbe, a indeks da se nalazi u registru HX. Ovaj adresni mod se koristi kod kopiranja nizova:

```
Niz1,Niz2 : array[0..99] of byte

ldhx SizeOf(Niz1);
repeat
  ldaa x[Niz1-1];
  staa x[Niz2-1];
until decr(IndX) =0;
```

Kod za ldaa u ovom adresnom modu je $D6.

Objašnjenje REL: bcc $1A

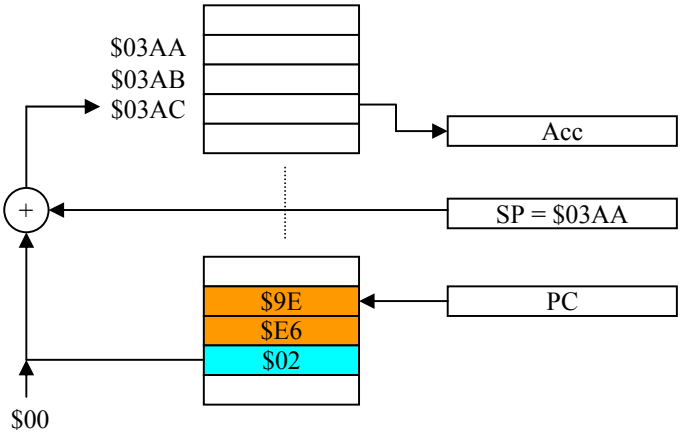


Čitajući naredbu **bcc $1A** programski brojač će se uvećati na vrednost $C391. Izvršavanje ove naredbe može da ide u dva smera:

1. Ako je setovan bit prenosa, nema skoka i izvršava se naredba sa lokacije $C391.
2. Ako nije setovan bit prenosa, vrednost programskog brojača se uvećava za $1A i postaje $C1AB, što je lokacija naredbe od koje se nadalje izvršava program

Kod za bcc je $24.

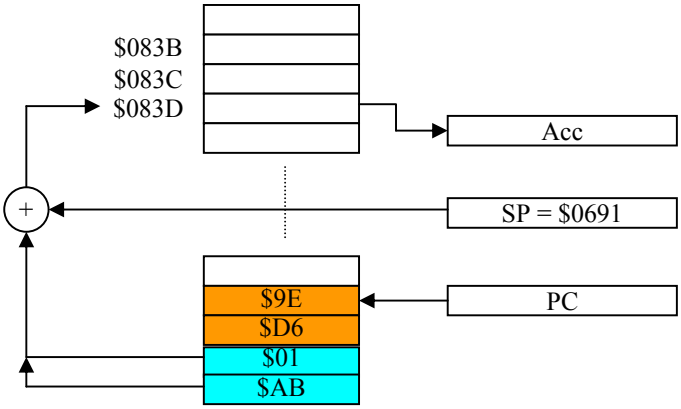
Objašnjenje SP1: ldaa s[$02]



Ovo je slučaj indirektnog indeksnog adresiranja sa 8-o bitnim indeksom. Bazna adresa je u stek pointeru SP, a indeks je parametar odmah iza koda naredbe . Ovaj adresni mod se koristi kada se pristupa dinamičkim varijablama na steku.

Kod za ldaa u ovom adresnom modu je $9EE6. Kod ove naredbe je 16-to bitni, ima prefiks $9E.

Objašnjenje SP2: ldaa s[$03AA]



Ovom naredbom se pristupa podacima na steku, onda kada ih ima više od 255, i kada je ofset nekog podatka u odnosuna stek pointer veći od 255. Ovom naredbom je učinjeno da nema ograničenja u veličini podataka koji će biti smešteni na steku.

Kod za ldaa u ovom adresnom modu je $9ED6. Kod ove naredbe je 16-to bitni, ima prefiks $9E.

Naredbe kojima se vrednost memorijske lokacije prebacuje u neki od CPU registara su:

naredba	opis
lda	U akumulator smešta vrednost adriserane memorijske lokacije
ldx	U registar X smešta vrednost adresirane memorijske lokacije
ldhx	U indeksni registar HX se smestaju 2 bajta, adresiran je viši bajt
jmp	U programski brojač se smešta adresa skoka
jsr	U programski brojač se smešta adresa podprograma

Kao što je uobičajeno kod Motorole, pa je tako i kod MC9(S)08, kada su u pitanju višebajtni podaci, ako se ide u smeru rasta adresa, prvi bajt je bajt najveće težine.

	IMP	IMM8	IMM16	DIR	EXT	IX	IX+	IX1	IX1+	IX2	REL	SP1	SP2

adca		A9		B9	C9	F9		E9		D9		9EE9	9ED9
adda		AB		BB	CB	FB		EB		DB		9EEB	9EDB
addsp		A7											
addhx		AF											
anda		A4		B4	C4	F4		E4		D4		9EE4	9ED4
asl				38		78		68				9E68	
asla													
aslx	48												
asr	58			37		77		67				9E67	
asra	47												
asrx	57												
bcc											24		
bclr 0				11									
bclr 1				13									
bclr 2				15									
bclr 3				17									
bclr 4				19									
bclr 5				1B									
bclr 6				1D									
bclr 7				1F									
bcs											25		
beq											27		
bge											90		
bgnd	82										92		
bgt											28		
bhcc											29		
bhcs											22		
bhi											24		
bhs											2F		
bih											2E		
bil													
bita		A5		B5	C5	F5		E5		D5		9EE5	9ED5
ble											93		
blo											25		
bls											23		
blt											91		
bmc											2C		
bmi											2B		
bms											2D		
bne											26		
bpl											2A		
bra											20		
brclr 0				01							*		
brclr 1				03							*		
brclr 2				05							*		
brclr 3				07							*		
brclr 4				09							*		
brclr 5				0B							*		
brclr 6				0D							*		
brclr 7				0F							*		
brn											21		
brset 0				00							*		
brset 1				02							*		
brset 2				04							*		
brset 3				06							*		
brset 4				08							*		
brset 5				0A							*		
brset 6				0C							*		
brset 7				0E							*		
bset 0				10									
bset 1				12									
bset 2				14									
bset 3				16									

bset 4				18									
bset 5				1A									
bset 6				1C									
bset 7				1E									
bsr											AD		
cbeqa		41		31			71		61		*	9E61	
cbeqx		51									*		
clc	98												
cli	9A												
clr				3F		7F		6F				9E6F	
clra	4F												
clrh	8C												
clrx	5F												
cmpa		A1		B1	C1	F1		E1		D1		9EE1	9ED1
cmphx			65	75	3E							9EF3	
cmpx		A3		B3	C3	F3		E3		D3		9EE3	9ED3
com				33		73		63				9E63	
coma	43												
comx	53												
daa	72												
dbnz				3B		7B		6B			*	9E6B	
dbnza	4B										*		
dbnzx	5B										*		
dec				3A		7A		6A				9E6A	
deca	4A												
decx	5A												
div	52												
eora		A8		B8	C8	F8		E8		D8		9EE8	9ED8
inc				3C		7C		6C				9E6C	
inca	4C												
incx	5C												
jmp		BC	CC			FC		EC		DC			
jsr		BD	CD			FD		ED		DD			
ldaa		A6		B6	C6	F6		E6		D6		9EE6	9ED6
ldhx			45	55	32	9EAE		9ECE		9EBE		9EFE	
ldx		AE		BE	CE	FE		EE		DE		9EEE	9EDE
lsl				38		78		68				9E68	
lsla	48												
lslx	58												
lsr				34		74		64				9E64	
lsra	44												
lsrx	54												
mov	**												
mul	42												
neg				30		70		60				9E60	
nega	40												
negx	50												
nop	9D												
nsa	62												
oraa		AA		BA	CA	FA		EA		DA		9EEA	9EDA
psha	87												
pshh	8B												
pshx	89												
pula	86												
pulh	8A												
pulx	88												
rol				39		79		69				9E69	
rola	49												
rolx	59												
ror				36		76		66				9E66	
rora	46												
rorx	56												
rsp	9C												

rti	**80**												
rts	**81**												
sbca		**A2**		**B2**	**C2**	**F2**		**E2**		**D2**		**9EE2**	**9ED2**
sec	**99**												
sei	**9B**												
staa				**B7**	**C7**	**F7**		**E7**		**D7**		**9EE7**	**9ED7**
sthx				**35**	**96**							**9EFF**	
stop	**8E**												
stx				**BF**	**CF**	**FF**		**EF**		**DF**		**9EEF**	**9EDF**
suba		**A0**		**B0**	**C0**	**F0**		**E0**		**D0**		**9EE0**	**9ED0**
swi	**83**												
tap	**84**												
tax	**97**												
tpa	**85**												
tst				**3D**		**7D**		**6D**				**9E6D**	
tsta	**4D**												
tstx	**5D**												
tshx	**95**												
txa	**9F**												
thxs	**94**												
wait	**8F**												

*) Ove naredbe sadrže i relativni skok
**) Naredba mov je dvoadresna i nju opisuje sledeća tabela:

mov	**DIR**	**IX+**
IMM8	6E	
DIR	4E	5E
IX+	7E	

Naredbe koje modifikuju sadržaj memorijskih lokacija, imaju samo jedan 8-o bitni parametar koji je ili 8-o bitna adresa nulte strane (DIR), ili indeks prilikom indirektnog indeksnog adresiranja (IX1 i SP1). U slučaju indirektnog adresiranja (IX) naredna nema parametar. Setovanje ili klirovanje bita (**bset**, **bclr**) implicitno (u kodu) sadrži broj bita (0..7), a kao parametar je adresa lokacije na nultoj strani. Ove dve naredbe se izvršavaju za 5 ciklusa (dva ciklusa za čitanje koda naredbe i adrese, čitanje podatka, modifikacija i na kraju ponovni upis podatka). Vremenski one traju 250ns. Naspram ovih naredbi, postoje i naredbe koje na isti način testiraju bit neke lokacije na nultoj strani, vrednost bita upisuju u bit prenosa, i mogu da izvrše relativni skok zavisno od vrednosti testiranog bita. Te naredbe su **brset** i **brclr**. I ove naredbe imaju trajanje 5 ciklusa.

Drugu grupu naredbi koje modifikuju vrednost memoriske lokacije čine naredbe za inkrementiranje i dekrementiranje, komplementiranje, negaciju, klirovanje svih bitova, siftanje i rotaciju vrednosti. Sve se naredne izvršavaju za isti broj ciklusa, ali zavisno od načina adresiranja: DIR 5 ciklusa, IX 4 ciklusa, IX1 5 ciklusa i SP1 6 ciklusa.

Inkrementiranje vrednosti lokacije van nulte strane može se izvesti na sledeći način:

```
ldhx [$1234]; //3 ciklusa, 3 bajta
inc  x[];     //4 ciklusa, 1 bajt
```

Dakle, ova operacija je izvršena za 7 ciklusa i potrošena su 4 bajta. U odnosu na istu narebu primenjenu na vrednost sa nulte strane, potrošeno je 2 ciklusa vise i 2 bajta više. Ustvari, kada bi ova naredba postojala za bilo koju adresu (kao kod HC11), onda bi ona potrošila 6 ciklusa i 3 bajta koda. U tablici kodova nema mesta za ove naredbe i sa DIR i EXT adresiranjem, pa su se projektanti MC9(S)08 odlučili samo za DIR adresiranje koje je brže i troši 1 bajt koda manje.

Primećuje se da je zbog popunjene tablice kodovima naredbi, korišćen prefiks $9E za adresni mod SP1 i SP2. Zato ove naredbe traju jedan ciklus duže i troše 1 bajt koda više. Zato je:

```
ldaa s[$AB];    //4 ciklusa i 3 bajta
```

i

```
tshx;           //2 ciklusa i 1 bajt
ldaa x[$AB-1];  //3 ciklusa i 2 bajta
```

skoro isto jer je potrošen isti broj bajtova, a trajanje je duže za 1 ciklus. Naredba **tshx** prebacuje sadržaj SP uvećan za 1 u indeksni registar HX, pa se zato u poslednjoj naredbi piše $AB-1. Međutim, ako se koristi kopiranje višebajtnih vrednosti, na primer 4-ro bajtnih, onda će se korišćenjem SP1 adresnog moda potrošiti 32 ciklusa i 24 bajta koda, a korišćenjem IX1 adresnog moda trajanje ce biti 26 ciklusa i potrošiće se 17 bajta koda. Dobitak je 6 ciklusa i 7 bajta koda. Alternativa za ovaj slučaj je korišćenje HX indeksnog registra i naredbi **ldhx** i **sthx** u SP1 adresnom modu. Tada će biti trajanje 20 ciklusa i potrošiće se 12 bajta koda. Opšte uzevši, ako se ne koristi IX, IX1 ili IX2 adresni mod, najefikasnije je kopiranje višebajtnih vrednosti preko 16-to bitnog indeksnog registra HX.

Karakteristične programske strukture

Primeri će biti napisani na MegaMakro asembleru (MMA) sa prikazom prevedenog koda radi eventualnog lakšeg razumevanja.

<u>if..then:</u>

```
ldaa [bB];
if  =0 then
begin
  ldhx bA;
  incr x[];
end;
```

Ako je vrednost varijable **bB** jednaka nuli, onda se vrednost varijable **bA** inkrementira za 1

i prevod:

```
$e00f : $c60101     [4]  ldaa [$0101]
$e012 : $2604       [3]  bne  $e018

$e014 : $450100     [3]  ldhx $0100
$e017 : $7c         [4]  inc  x[]
$e018 : $81         [6]  rts
```

Prva kolona je adresa flash-a, zatim sledi kod naredbe, pa u uglastim zagradama trajanje izvršenja naredbe u ciklusima, i na kraju sama naredba. Primer je pisan u podprogramu, pa je zato zadnja naredba **rts**.

<u>if..then..else:</u>

```
ldhx bA;
ldaa [bB];
if  =0 then incr x[]
       else decr x[];
```

```
$e00f : $450100     [3]  ldhx  $0100
$e012 : $c60101     [4]  ldaa [$0101]
$e015 : $2603       [3]  bne   $e01a

$e017 : $7c         [4]  inc  x[]
$e018 : $2001       [3]  bra   $e01b
-------------------------------------
$e01a : $7a         [4]  dec  x[]
$e01b : $81         [6]  rts
```

<u>repeat..untl:</u>

```
ldhx SizeOf(NizA);
repeat
  ldaa x[NizA-1];
```

```
$e00f : $450033     [3]  ldhx     $0033
$e012 : $d60103     [4]  ldaa   x[$0103]
$e015 : $d70136     [4]  staa   x[$0136]
$e018 : $5bf8       [4]  dbnzx    $e012
```

```
  staa x[NizB-1];
until decr(IndX) =0;
```

Ovo je primer kopiranja NizA u NizB. Oba niza su iste dužine (51 bajt) manje od 256 bajtova. Kada je ovakav slučaj, onda se naredbom **`ldhx SizeOf(NizA)`** H registar postavlja na nulu, a X registar sadrži vrednost dužine niza, pa je zato moguće koristiti naredbu za kraj petlje **dbnzx**. U slučaju da su nizovi duži od 255 bajtova, program za kopiranje bi bio:

```
ldhx SizeOf(NizA);
repeat
  ldaa x[NizA-1];
  staa x[NizB-1];
  addhx -1;
  cmphx 0;
until =0;
```

```
$e00f : $45015f     [3]  ldhx     $015f
$e012 : $d60103     [4]  ldaa   x[$0103]
$e015 : $d70262     [4]  staa   x[$0262]
$e018 : $afff       [2]  addhx    $ff
$e01a : $650000     [3]  cmphx    $0000
$e01d : $26f3       [3]  bne      $e012
```

Predhodni primeri za **repeat..until** neće biti ispravni ako je dužina niza jednaka 0. Naravno, ovo je teoretski slučaj, jer se nizovi nikada neće deklarisati tako da nemaju ni jedan element. Ovo se ipak može desiti kod kopiranja stringova (niza znakova) ako se radi o praznom stringu. Za to, i za mnoge druge slučajeve (kada se nezna da li je pre ulaska u **repat..until** petlju ispunjen uslov za izlazak iz petlje) koristi se **while..do** petlja. Ona je modifikovana **repat..until** petlja, u smislu da se prvo ispituje uslov za ne izlazak iz petlje, pa ako je on ispinjen, počinje da se izvršava petlja, sve dok uslov postoji. U MMA ne postoji **while..do** petlja, već modifikacija **repeat..until** petlje u **repeat by..prior..until** petlju. Predhodni primer bi, prema do sada napisanom bio:

```
ldhx SizeOf(NizA);
repeat by
  ldaa x[NizA-1];
  staa x[NizB-1];
  addhx -1;
prior
  cmphx 0;
until =0;
```

```
$e00f : $45015f     [3]  ldhx     $015f
$e012 : $2008       [3]  bra      $e01c
-------------------------------------
$e014 : $d60103     [4]  ldaa   x[$0103]
$e017 : $d70262     [4]  staa   x[$0262]
$e01a : $afff       [2]  addhx    $ff
$e01c : $650000     [3]  cmphx    $0000
$e01f : $26f3       [3]  bne      $e014
```

Ovo je trivijalan primer jer se **`SizeOf(NizA)`** unapred zna, ali ipak dobro ilustruje ideju predhodnog ispitivanja pre ulaska u petlju.

case..of

Ova programska struktura izvršava jednu sekvencu naredbi zavisno od više mogućih skupova vrednosti registra CPU-a (Acc, X ili HX). Iza **case** se piše ime registra kao: AccA ili IndX ili IndHX. Komparacije sa vrednostima slučajeva u **case** strukturi se vrše kao sa neoznačenim vrednostima. To podrazumeva da je vrednost u naznačenom registru neoznacena. Ako je vrednost u naznačenom registru označena, onda se piše, na primer, **case signed** AccA **of**. Slučaj **else** nije obavezan, a ako se on koristi onda se kod pridružen tom slučaju izvršava onda kada vrednost u naznačenom registru ne pripada ni jednom slučaju. Kompajler ne vrši proveru da li se slučajevi ponavljaju, već će izvšiti kod prvog slučaja koji se bude pojavio. Kompajler ne pravi tablicu skokova, već redom proverava svaki slušaj. Za mali broj slučajeva, ovo je efikasnije od metoda sa pravljanjem tablice skokova, ali za veći broj slučajeva to neće biti.

```
ldaa [bb];
case AccA of
  1,9,21..30 : inca;
  31..50,70..90 : deca;
  else clra;
end;
staa [bb];
```

```
$f06a : $c60113     [4]  ldaa    [$0113]
$f06d : $a101       [2]  cmpa     $01
$f06f : $270c       [3]  beq      $f07d

$f071 : $a109       [2]  cmpa     $09
$f073 : $2708       [3]  beq      $f07d

$f075 : $a115       [2]  cmpa     $15
$f077 : $2508       [3]  bcs      $f081

$f079 : $a11e       [2]  cmpa     $1e
$f07b : $2204       [3]  bhi      $f081

$f07d : $4c         [1]  inca
$f07e : $ccf096     [4]  jmp      $f096
-------------------------------------
$f081 : $a11f       [2]  cmpa     $1f
$f083 : $2504       [3]  bcs      $f089

$f085 : $a132       [2]  cmpa     $32
$f087 : $2308       [3]  bls      $f091

$f089 : $a146       [2]  cmpa     $46
$f08b : $2508       [3]  bcs      $f095

$f08d : $a15a       [2]  cmpa     $5a
$f08f : $2204       [3]  bhi      $f095

$f091 : $4a         [1]  deca
$f092 : $ccf096     [4]  jmp      $f096
-------------------------------------
$f095 : $4f         [1]  0/238 clra
$f096 : $c70113     [4]  0/240 staa    [$0113]
```

Interni moduli mikrokontrolera MC9S08

Haredver ugradjen u mikrokontrolere ove familije je raznovrstan i varira od tipa do tipa. Sledeći modulu se nalaze u svakom tipu:

1. CPU
2. Flash;
3. RAM
4. Ulazno izlazni modul sa portovima;
5. Oscilator;
6. Generator taktnih signala;
7. BDC (pozadinski dibag kontroler);
8. Tajmerska jedinica;
9. SCI (serijski komunikacioni interfejs);
10. SPI (serijski periferijski interfejs);
11. I^2C komunikacija
12. ADC (analogno-digitalni konvertor);
13. WDT (watch-dog tajmer);
14. Keyboard interrupt modul;

Pojedini tipovi sadrže još i sledeče module:

1. USB komunikaciju;
2. CAN komunikaciju (Controller Area Network);
3. J11850 komunikacija; <naci ovo>
4. RTC (Real Time Counter);

5. MTIM (Modulo Timer);
6. EEPROM;
7. Drajver LCD-a;
8. Analogni komparator;
9. CTM (Carrier Modulator Timer).

Ovi moduli će biti opisani na osnovu tehničkih podataka sledećih tipova mikrokontrolera: MC9S08AW, MC9S08DZ, MC9S08GB, MC9S08JM, MC9S08LC, MC9S08QE, MC9S08QD, MC9S08RG i MC9S08QG.

Flash programska memorija

Izvršni kod se upisuje u flash memoriju koja čuva svoj sadržaj i po gubitku napajanja mikrokontrolera. Flash memorija može više puta da se briše i ponovo u nju upisuje. Broj ciklusa brisanja i upisa, a da su podaci još uvek dobro upisani, je tipično 100000, a minimalno 10000 na –40^{o}C. Jednom upisani podaci ostaju nepromenjeni tipično 100 god., a minimalno 15 god. na –40^{o}C.

Flash je podeljen na stranice i redove. Jedna stranica sadrži 512 bajtova, a red 64 bajta. Moguće je odjednom obrisati celu flash memoriju (obrisana lokacija ima vrednost $FF), ili samo jednu stranicu. Brisanje cele flash memorije troši oko 120ms, a stranice 24ms.

Programiranje po bajtu troši 54μs, a ako se koristi takozvani burst mod onda se bajt programira za 24μs. Praktično ovo znači da se za programiranje 32KB flash-a troši: 120ms za brisanje i 32768 x 24μs = 786.4ms, što je ukupno 906ms (zaokruženo).

Prenos izvršnog koda počinje slanjem niza naredbi (u aktivnom BDM) za brisanje flash-a. Potom se u RAM upise program za burst programiranje. Poslednja naredba ovog programa treba da je **bgnd** kojom se mikrokontroler vraca u aktivan BDM. Posle ovoga (dok je mikrokontroler još u aktivnom BDM) upisuju se 64 bajta (veličina reda flash-a) koda na odredjeno mesto u RAM-u. Nakon ovoga startuje se izvršenje programa iz RAM-a, čime se napusta aktivni BDM. Po završenom programiranju ovih 64 bajta, mikrokontroler se vraća u aktivan BDM i vrsi se upis u RAM sledećih 64 bajta. Ovo se ponavlja sve dok se ne prenese ceo izvršni kod. Ovakav način programiranja flash-a je jednostavan i razumljiv, ali nije efikasan. Naime, u svakom trenutku ili se prenose bajtovi koda u RAM, ili se programira flash.

Drugi način je da se paralelno sa programiranjem jednih 64 bajta upisuju sledeća 64 bajta u RAM. Tada treba imati malo modifikovan program za programiranje flash-a i mesta u RAM-u za dve grupe od po 64 bajta. Takodje treba odvojiti jedan bajt koji će sadržati bitove za sinhronizaciju celog procesa. Dok se programira flash, prenos koda se vrši u nenametljivom BDM. Brzina prenosa u BDM-u, pri preddefinisanoj frekvenciji BDC-a od 8MHz, je teoretski 500K bita u sekundi, ili 62.5KB u sekundi. Sa druge strane, u sekundi se programira 41,7KB. Ovi podaci su teoretske, maksimalne vrednosti koje se u praksi ne mogu ostvariti, ali im se može približiti dobro koncipiranim programom za loadovanje programa u mikrokontroler.

Ovi mikrokontroleri imaju napajanje od 1.8V do 5.5V (jedna grupa se napaja sa 1.8V do 3,6V, a druga sa 2.7V do 5.5V), ali za programiranje flash-a je potrebno napajanje veće od 2.05V.
Mehanizam mikrokontrolera vezan za flash memoriju sadrži i mogućnost zaštite od neovlašćenog pristupa flash-u i RAM-u. U osnovi, ovaj mehanizam se sastoji u upisu 8 bajta na odredjenim flash lokacijama. Ovo je ključ za pristup flash-u i RAM-u. Brisanjem flash-a briše se i ovaj ključ, pa je ovo jedan od načina da se on zaobiđe, ali ne i da se pročita program koji je bio u flasu-u. Ovaj postupak se koristi u toku razvoja programa. Kada se program završi i upiše u flash, onda setovanjem odredjenih bitova u kontrolnom registru, koji upravlja mehanizmom zaštite flash-a i RAM-a, može se učiniti

nemogućim uvođenje mikrokontrolera u BDM, a time i pristup flash-u, RAM-u i ostalim resursima mikrokontrolera. Na ovaj način zaključan flash nije moguće otključati.

Portovi

Preko portova mikrokonteiler komunicira sa spoljašnjim hardverom. Portovi su po pravilu 8-o bitni, ali neki mogu imati i manji broj bitova. Svaki bit ima svoj pin na kućištu mikrokontrolera. Svaki port ima svoj registar podataka na odgovarajućoj adresi. U ovaj registar se mogu upisivati podaci, ali se mogu i citati. Takođe, svaki port ima i registar koji idređuje smer svakog bita porta. Ako je neki bit registra smera log. 0, onda je njemu odgovarajući bit, i odgovarajući pin ulaz za signale, a ako je on log. 1 onda se radi o izlaznom pinu. Čitanjem porta koji ima i ulazne i izlazne pinove, na mestu izlaznih će se pročitati vrednosti bitova koji su pre toga bili upisani, a na mestu ulaznih biće pročitane trenutne vrednosti signala priključenih na ulazne pinove. Svaki ulazni pin ima histerezis.

Uz svaki port postoji i registar koji uključuje (sa log. 1) pull-up otpornik samo ako je odgovarajući pin ulazni. Ovo setovanje nema efekta na izlazne pinove. Takođe, svaki port ima registar u kome je moguće setovati (log. 1) da odgovarajući izlazni pin ima usporenu ivicu prilikom promene signala na njemu. Ovim se smanjuje nivo smetnji koje može da zrači mikrokontroler. Ovo setovanje nema efekta na ulazne pinove. Izlaznim pinovima je namenjen još jeden registar. U ovom registru se mogu setovanjem odgovarajućih bitova na log. 1 učiniti da odgovarajući izlazni pinovi mogu da rade sa većim strujama.

Broj ovih ulazno-izlaznih pinova zavisi od veličine kućišta, i kreće se od 6 do 56.

Kada se omogući rad nekog internog modula (serijske komunikacije, tajmera i t. d.) onda on zauzima pinove porta na koje je predviđeno da bude povezan. Tada ovi pinovi pripadaju internom modulu i ne mogu se koristiti kao ulazno-izlazni pinovi.

Tajmer

Tajmerska jedinica služi za vremensku obradu signala koji se uvode u mikrokontroler, ili za vremenski kontrolisane signale koje generise mikrokontroler. Motorola koristi isti princip na kojima bazira tajmersku jedinicu u svim svojim 8-o bitnim i 16-to bitnim mikrokontrolerima, a takodje i u 32-o bitnim uz izvesnu modifikaciju. Princip je da jedna tajmerska jedinica ima do 8 16-to bitnih tajmeskih kanala. Svaki kanal može da se konfiguriše kao:

- input capture;
- output compare;
- edge aligned PWM ili
- center aligned CPWM

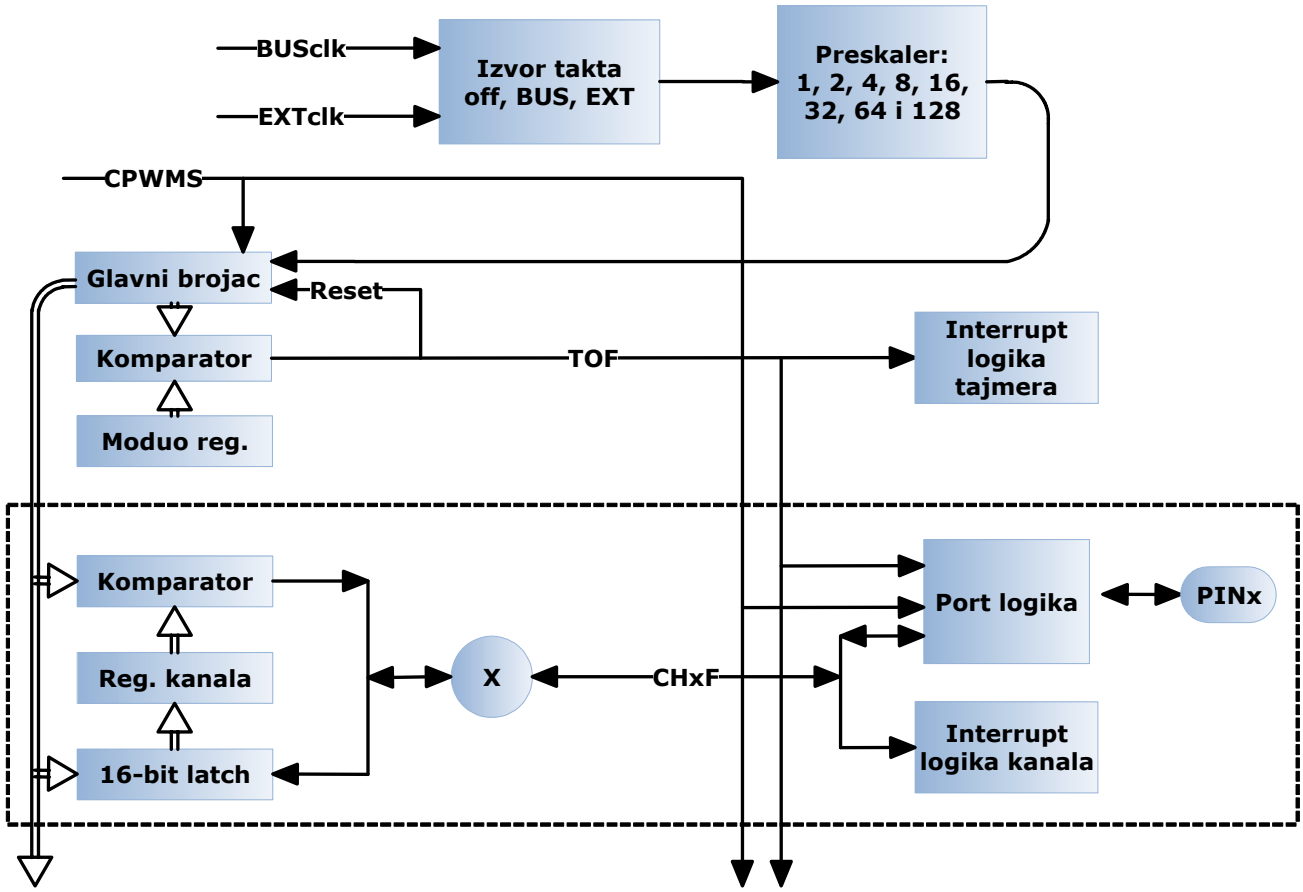
Kada je kanal konfigurisan kao input capture, može se podesiti da se ova funkcija realizuje na: opadajuću ivicu ulaznog signala, ili na usponsku ivicu ulaznog signala, ili na bilo koju ivicu.

Ako je kanal konfigurisan kao output compare, onda se on može programirati da tajmerski događaj setuje izlazni pin, ili restuje izlazni pin, ili menja logičko stanje na izlaznom pinu. U ovoj konfiguraciji tajmerskog kanala, on se može programirati i da nema uticaja na pin mikrokontrolera, već da se taj pin može koristiti kai I/O pin. U tom slučaju tajmerski kanal se koristi kao izvor periodičnog interrupt-a.

Tajmerski kanal koji je konfigurisan edge aligned PWM generiše signal konstantne periode u kome se vremenski pomera ivica signala menjajući odnos trajanja log. 1 i log. 0 (duty cucle). Ovako konfigurisan

kanal može biti programiran da generise i invertovan signal. Ako je kanal konfigurisan kao CPWM, onda se upravlja sirinom impulsa dok perioda ostaje konstantna.

Tajmerska jedinica, pored tajmerskih kanala ima preskaler, 16-to bitni glavni brojač i 16-to bitni registar koji određuje moduo brojanja glavnog brojača. Taktni signal tajmerske jedinice može biti izabran, tako da bude interni BUS takt, ili eksterni sa odgovarajućeg pina.



Predhodna slika prikazuje principijelnu šemu jednog tajmerskog kanala (uokviren crtkastim pravougaonukom) i hardver tajmerske jedinice koji je zajednički za sve kanale. Svi registri su 16-to bitni, a prikazana magistrala je 16-to bitna magistrala tajmerske jedinice. Ova magistrala povezuje glavni brojac i komparatore i 16-to bitme latch-eve svih kanala. Na 8-o bitnu magistralu mikrokontrolera su vezani glavni brojač, moduo registar i registri svih kanala.

U bloku: izbor takta, bira se bus frekvencija (može da bude 24MHz maksimalno) ili frekvencija na spoljasnjem pinu kao taktna frekfencija tajmera. Ako se ne izabere ni jedna (off) tajmer je iskljucen. Izabrani takt ide u preskaler koji njegovu frekvenciju deli sa 1 do 128, kao što je navedeno na slici, kako bi se dobila taktna frekvencija tajmera. Izbor deljenja preskalera se određuje sa tri bita u jednom od kontrolnih registara tajmera.

Glavni brojač stalno broji taktove tajmera od 0 do 65535. Ovo važi u slučaju ako je vrednost u moduo registru nula. Ako se u moduo registar upiše neka vrednost, onda glavni brojač broji od 0 do vrednosti upisane u moduo registru. Komparator obezbeđuje resetovanje glavnog brojača kada njegova vrednost postane jednaka vrednosti koja je upisana u moduo registru. Uvek kada vrednost glavnog brojača postane nula, generiše se se signal TOF koji, ako mu se dozvoli može da izazove interrupt. Signal TOF se vodi u sve tajmerske kanale.

Signal CPWMS je bit u kontrolnom registru tajmerske jedinice koji se setuje za CPWM.

Svaki kanal poseduje komparator, registar podataka i latch. Sa X je označena prekidačka mreža koja usmerava signale zavisno od setovane konfiguracije tajmerskog kanala.

Svi mikrokontroleri ove familije imaju po dve tajmerske jedinice (izuzev nekih u kućištu sa malim brojem pinova).

Input capture funkcija se ostvaruje tako što signal sa pina, CHxF, okida latch, tako da on uzima vrednost glavnog brojača u tom trenutku. Istovremeno signal CHxF može da izazove interrupt. Konfiguracija kanala definiše u bloku pin logika koja će ivica generisati CHxF, usponska, opadajuća ili obe. U interrupt proceduri treba obrisati CHxF (u odredjenom statusnom registru kanala) i pročitati registar kanala, jer će u njega biti prebačena vrednost iz latch-a. Kako se čita 16-to bitna vrednost preko 8-o bitnog bus-a, mogli bi nastati problemi, ali su oni izbegnuti dodatnim hardverom koji dozvoljava da se u proizvoljnom redosledu čitaju bajtovi iz registra podataka kanala. Ovaj mehanizam je obezbeđen za sva čitanja i upise u 16-to bittne registre mikrokontrolera.

Na primer: želimo da merimo periodu nekog spoljnjeg signala. Povežimo ga na pin koji odgovara, na primer, kanalu 0 tajmera 1. U inicijalizaciji, ili na nekom drugom mestu, pre nego sto dozvolimo globalni interrupt konfigurišimo kanal 0 da radi kao input capture. Inače, dozvoljeno je bio gde i bilo kad u programu menjati konfiguraciju tajmerskog kanala, ali se mora voditi računa da se to ispravno uradi kako ne bi proizvelo neželjene posledice. Neka su dve globalne varijable:

```
T  : word;
dT : word;  //perioda
```

Izaberimo da je izvor takta tajmera bus frekvencija, a prema očekivanoj periodi spoljnjeg signala izaberemo odgovarajući preskaler. Izaberimo jednu od ivica signala koja ce okidati kanal (sve jedno je da li je to opadajuća ili rastuća ivica). U registru TPM1C0SC setujmo bit CH0IE (dozvolimo interrupt od kanala 0) i dozvolimo generalni interrupt naredbom ***cli***. Interuupt od ivice ulaznog signala će pokrenuti izvršenje sledeće procedure:

```
procedure MerenjePeriode; interrupt $FFF4
var dynamic
  pT : word;
begin
  pshh;
  ldaa [TPM1C0SC];  //Prvo treba obrisati bit CH0F koji se postavio kada
  anda not |CH0F|;  //je izazvan interrupt. On se briše čitanjem TPM1C0SC
  staa [TPM1C0SC];  //i upisom log. 0 na mestu bita CH0F

  ldhx [TPM1C0V];
  sthx s[pT];       //Privremeno čuvanje pročitanog vremena

  txa;              //Izračunavanje razlike vremena između dve ivice
  suba [T.1];
  staa [dT.1];
  pshh;
  pula;
  sbca [T.0];
  staa [dT.0];

  ldhx s[pT];       //Vreme sadašnje ivice se pamti kao predhodna ivica
  sthx [T];
  pulh;
end;
```

Statusno-kontrolni registar TPM1C0SC (tajmer 1 kanal 0) se deklariše kao varijabla tipa skupa od 8 bita na absolutnoj adresi, na sledeći način:

```
TPM1C0SC : set of (No0,No1,ELS0A,ELS0B,MS0A,MS0B,CH0IE,CH0F) absolute $0025;
```

	7	6	5	4	3	2	1	0
	CHnF	CHnIE	MSnB	MSnA	ELSnB	ELSnA	0	0
RESET	0	0	0	0	0	0	0	0

Gde je:

- ELS0A, ELS0B određuje ivicu koja okida tajmerski kanal u input capture modu, ili određuje u output compare modu da li se setuje ili resetuje, ili samo menja log. nivo na pinu koji je uvezi sa kanalom 0. Kada su oba ova bita na log. 0, onda pin nije u vezi sa tajmerskim kanalom (tada je on I/O pin), i u output compare modu može se koristiti samo interrupt mehanizam tajmerskog kanala.
- MS0A, MS0B određuje mod rada tajmerskog kanala. Svaka od 4 kombinacija ovih bitova određuje jedan mod rada, naveden na početku ovog poglavlja.
- CH0IE je bit koji dozvoljava interrupt od ovog tajmerskog kanala.
- CH0F je bit koji se setuje kada se u input capture modu pojavi aktuelna ivica spoljasnjeg signala, ili kada se u output compare modu izjednači sadržaj glavnog brojača i registra podataka kanala TPM1C0V.

Naredba: `anda not |CH0F|;` u prevodu kompajlera je: `anda $7F;` Deklaracijom varijable TPM1C0SC bit CH0F je konstanta kojoj je dodeljenja vrednost 7. Vertikalne crte se navode da bi kompajler to izračunao kao 2^7 = $80. Ako ima više konstanti između vertikalnih crta, kompajler izračunava sumu stepena na 2. Na primer: $|2,5,6| = 2^2+2^5+2^6$ = $C4.

Da bi program u interrupt proceduri radio ispravno, moduo registar mora imati vrednost 0. Naime, glavni brojač briji po modulu 2^{16}, pa i oduzimanje vremena pojavljivanja dve uzastopne ivice ulaznog signala se vrši po modulu 2^{16}, jer se vremena beleže kao 16-to bitne vrednosti. Tako, ako je izmereno vreme 10000, a predhodno 50000, onda razlika 10000-50000 = -40000, što je $63C0 odnosno 25536. Ova vrednost je tačna razlika vremena između dve uzastopne ivice ulaznog signala. Pravilo je da ako se pri oduzimanju pojavi negativna vrednost, onda razlici treba dodati moduo brojanja. Ako je moduo brojanja 2^{16}, a oduzimaju se 16-to bitne varijable, onda nije potrebno nikakvo dodavanje za slučaj negativne razlike.

Međutim, ako se podesi da moduo brojanja bude, na primer 40000, onda zamislimo da je izmereno vreme 5000, a predhodno 30000. U ovom slučaju razlika će biti negativna, pa prema pravilu, razlika je 5000-30000+40000 = 15000. Dakle, ako je moduo brojanja glavnog brojača različita od 2^{16}, što se postiže upisom u moduo registar broj različit od nule (u predhodnom primeru bi se upisalo 40000), onda program u interrupt proceduri treba modifikovati u smisku ovoga što je napisano u ovom pasusu.

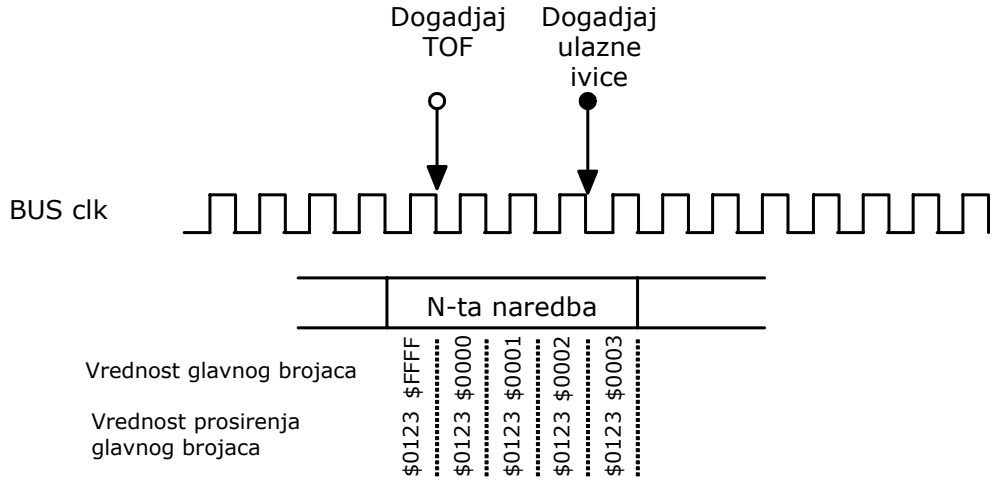
Ako se za frekvenciju tajmerske jedinice izabere bus frekvencija, koja može da bude 20MHz (samo kod jednog tipa ona je 24MHz) i preskaler deli sa 1, onda najduži interval koji može da se izmeri iznosi 65535 * 50ns = 3276.75µs. Druga krajnost je da je preskaler deli sa 128, pa će najduži interval koji može da se izmeri biti: 65535*128*50ns = 419.424ms. Ako se želi merenje dužih intervala, onda se može postupiti na jedan od tri načina:

1. Smanjiti radnu frekvenciju mikrokontrolera, a time i BUS frekvenciju. Ovaj metod često neće biti poželjan, jer ne doprinosi značajnom produženju maksimalne periode koja može da se izmeri, a znatno smanjuje brzinu mikrokontrolera.
2. Kao taktni signal tajmerske jedinice može se uzeti neka spoljna frekvencija koja odgovara željenoj maksimalnoj periodi koja treba da se izmeri. Ovaj metod zahteva dodatni spoljni

hardver, sto je nedostatak. Ova mogućnost (korišćenje spoljnjeg takta tajmera) je pre svega namenjena slučajevima kada se traži posebno visoka tačnost merenja, koja prevazilazi tačnost kristala koji taktuje mikrokontroler. Tada se koriste posebni, vrlo tačni izvori periodičnih signala za taktovanje tajmerske jedinice.

3. Softversko proširenje glavnog brojača i registra podataka kanala. I brojaču i registru se mogu, na primer dodati po dva bajta koja bi bila deklarisana kao varijable tipa word, i oni bi bili pridruženi glavnom brojaču i registru podataka kanala kao viši bajtovi. Tada glavni brojač postaje 32-o bitni, a takođe i registar podataka kanala. Mehanizam funkcioniše na sledeći način: glavni brojač se inkrementira na svaki takt, a korišćenjem interrupt-a od TOF inkrementira se softverski za 1 njemu pridružena dva viša bajta. Ovakav 32-o bitni glavni brojač broji od 0 do 2^{32}-1. Kada se desi interrupt od ulaznog signala, trenutna vrednost glavnog brojača (16 bita) se automatski prebacuju u registar podataka kanala. U toj interrupt proceduri treba softverski prebaciti i proširenje glavnog brojača u proširenje registra podataka kanala. Na ovaj način izmereno vreme ima vrednost u opsegu od 0 do 2^{32}-1 i predstavlja 32-o bitnu vrednost. Za korišćenje ovog metoda moduo brojanja mora biti 2^{16}.

Metod pod 3 krije jednu zamku koja može da izazove veliku grešku ako se ne vodi računa o njoj. Sledeća slika će pomoći da se objasni uzrok problema.



Prikazan je BUS clk signal, uz pretpostavku da je tajmerska jedinica taktovana ovim signalom sa preskalerom koji deli sa 1. Takodje je prikazana N-ta naredba koja ima trajanje od 5 cikljusa. Prikazani su događaji od TOF i od ivice ulaznog signala koji se dešavaju za vreme trajanja jedne iste naredbe. Na slici je prikazan redosled ovih događaja koji dovodi do greške. Takođe su prikazane i vrednosti glavnog brojača i proširenja glavnog brojača (za primer je uzeta vrednost $0123) iz ciklusa u ciklus. Događaj od ivice ulaznog signala (sinhroniše se na takt tajmera) proizvodi prenošenje vrednosti glavnog brojača i registar podataka tajmerskog kanala. U toku N-te naredbe se postavljaju dva zahteva za interrupt: od događaja TOF i od ivice ulaznog signala. Mehanizam interrupt-a mikrokontrolera daje veći prioritet interrupt-u od ivice ulaznog signala u odnosu na interrupt od TOF. To znači da će se po završetku N-te naredbe prvo pokrenuti interrupt procedura izazvana ivicom ulaznog signala i zabeležiti vreme događaja pomenute ivice, zajedno sa proširenjem, sa vrednošću $01230002. Ovo je očigledno pogrešna vrednost. Sobzirom da se predhodno desio događaj TOF, ispravna vrednost bi bila: $01240002. Medjutim, posle obrade interrupt-a od ivice ulaznog signala, obradiće se interrupt od TOF, kada će proširenje glavnog brojača dobiti tačnu vrednost.

Greška je posledica toga da se prvo dešava događaj nižeg prioriteta, a potom događaj višeg prioriteta u toku izvršenja jedne naredbe. U obrnutom redosledu greška se ne javlja.

Jedan od nacina da se reši ovaj problem je:

- Posle brisanja bita CH0F treba dovoliti globalno interrup-e sa ***cli*** čime će biti dozvoljen interrupt od TOF. Taj interrupt će pokrenuti interrupt proceduru koja će uvećati za 1 proširenje glavnog brojača. Treba znati da se sa aktiviranjem svake interrupt procedure automatski onemogućuje globalni interrupt (kao naredbom ***sei***), pa ako je potrebno da se u interrupt proceduri omogući aktiviranje drugih interrupt-a, onda se to cini naredbom ***cli***.
- Ako postoji zahtev za interrupt-om od TOF, on će odmah posle ***cli*** pokrenuti proceduru koja će uvećati za 1 proširenje glavnog brojača. Tako će sledće naredbe kojima se čita vreme dešavanja ivice ulaznog signala biti tačno.

Najkraći intervali koji mogu da se izmere zavise od dužine interrupt procedure kojom se obrađuje događaj od spoljne ivice. Primer interrupt procedure `MerenjePeriode` pokazuje da se ona izvršava za 46 ciklusa, čemu treba dodati sikluse za ulazak i izlazak iz nje, što je 46+20 = 66 cilusa. Pri taktu mikrokontrolera od 20MHz ovo vreme iznosi 3.3µs. Dakle, nije moguće meriti događaje od spoljnih ivica koji se događaju sa manjom periodom od 3.3µs. Drugim metodama, koje spadaju u domen dosetki, ovo vreme se može skratiti, ali se onda kao po pravilu radi o specifičnim slučajevima.

Output compare funkcija može da se ostvari: sa povezivanjem kanala na svoj I/O pin, ili bez povezivanja tajmerskog kanala na I/O pin. Ako se želi generisanje signala, onda se port logika konfigurise tako da se flip-flop u njoj resetuje, setuje ili menja stanje pri svakoj pojavi signala CHxF, pri čemu je tajmerski kanal povezan sa pripadajućim I/O pinom. Signal CHxF se postavlja na log. 1 uvek kad se izjednače po vrednosti sadržaj glavnog brojača i sadržaj registra podataka tajmerskog kanala. Kada se jednom CHxF postavi na log. 1, onda on ostaje stalno na toj vrednosti (kao i kod Input capture funkcije). Zato je potrbno omogućiti interrupt od CHxF signala, i u interrupt proceduri obrisati CHxF bit, odnosno postaviti ga na log. 0. Ako se želi generisanje periodičnog interrupt-a, bez generisanja signala na I/O pinu, onda je postupak isti, samo se tajmerski kanal ne povezuje na I/O pin. U ovom slučaju nije važno kako je setuje port logika.

Primer interrupt procedure kojom se generiše periodičan signak na I/O pinu koji pripada kanalu 0 tajmera 1 je:

```
procedure GenerisanjePeriode; interrupt $FFF4
const
  Perioda = 10000;
begin
  pshh;
  ldaa [TPM1C0SC];  //Prvo treba obrisati bit CH0F koji se postavio kada
  anda not |CH0F|;  //je izazvan interrupt. On se briše čitanjem TPM1C0SC
  staa [TPM1C0SC];  //i upisom log. 0 na mestu bita CH0F

  ldhx [TPM1C0V];               //Uvećanje registra podataka kanala
  txa;                          //za vrednost trajanja priode
  adda Perioda and 255;
  tax;
  pshh;
  pula;µ
  adca (Perioda shr 8) and 255;
  psha;
  pulh;
  sthx [TPM1C0V];
  pulh;
end;
```

Ova interrupt procedura menja vrednost na pripadajućem I/O pinu svakih 1000 ciklusa tajmerske jedinice. Ako je takt tajmera 50ns, i ako je port logika konfigurisana tako da menja stanje I/O pina na svaki CH0F signal, onda će se dobiti periodični signal trajanja 2000*50ns = 100μs (po 1000 za svaku poluperiodu).

Analize koje su date za Input capture funkciju, u smisku najkraće i najduže periode, kao i mogućnostima proširenja najduže periode slične su i za Output compare funkciju.

Generisanje PWM signala ima za cilj dobijanje povorke impulsa stalne periode, ali sa promenljjivim odnosom impuls/pauza. Kada se tajmerski kanal setuje da generiše PWM signal, onda se u port logici na TOF setuje na log. 1 pripadajući I/O pin, a na CHxF on se kliruje na log. 0. Perioda se podešava upisom željene vrednosti u moduo registar, a odnos impuls/pauza se podešava vrednošću koja se upisuje u registru podataka kanala koji se koristi za generisanje PWM signala. Zapaža se da jedna tajmerska jedinica može da generiše onoliko PWM signala koliko ima kanala, ali svi ti PWM signali moraju imati istu periodu. Ovako generisani PWM signal je ivično podesiv signal. Odnos impuls/pauza je podesiv od 0% do 100%. Kada je vrednost u registru podataka 0, onda jr odnos impuls/pauza 0%, a kada je pomenuta vrednost jednaka ili veća od vrednosti moduo registra, onda je odnos impuls/pauza 100%.

Drugi način generisanja PWM signala je centralno podešen signal. U ovom modu generisanja PWM signala, glavni brojač broji unazad od maksimalne vrenosti (zadate vrednošću moduo registra) do nule, zatim počinje da broji unapred do maksimalne vrednosti, ponavljajući neprestano ovakav način brojanja. Pri ovome se dešava da u jednoj periodi (brojanje glavnog brojača unazad pa unapred) dva puta dođe do izjednačavanja vrednosti glavnog brojača i vrednosti registra podataka kanala, odnosno dva puta se generiše signal CHxF. Kada glavni brojač broji unazad, i generiše se CHxF, onda se setuje na log. 1 I/O pripadajući pin, a kada se pri brojanju napred desi CHxF, onda se kliruje na log. 0 isti I/O pin. U ovom modu generisanja PWM signala vrednosti u registrima se tretiraaaju kao 16-to bitne označene vrednosti. Tako moduo registar može imati vrednost u opsegu od $0001 do $7FFE. Dvostruka vrednost moduo registra određuje periodu PWM signala. Dvostruka vrednost registra podataka kanala kojim se generiše PWM određuje trajanje impulsa u PWM signalu. Ako registar kanala ima vrednost 0 ili negativnu (15-ti bit je log. 1) onda je odnos impuls/pauza 0%. Kada je vrednost registra podataka kanala pozitivna i veća od vrednosti moduo registra, onda je odnos impuls/pauza 100%. Za generisanje ovakvog PWM signala koristi se samo CHxF signal, što daje manji šum u generisanom signalu u odnosu na PWM signal generisan na bazi podešavanja ivice.

Svaka tajmerska jedinica poseduje statusni i kontrolni registar TPMxSC, gde je x broj tajmerske jedinice (1,2..). Bitovi ovog registra su:

	7	6	5	4	3	2	1	0
	TOF*	TOIE	CPWMS	CLKSB	CLKSA	PS2:PS0		
RESET	0	0	0	0	0	0	0	0

*) Može samo da se čita

- PS0, PS1, PS2 određuju vrednost preskalera;
- CLKSA, CLKSB odredđuju izvor taktovanja tajmerske jedinice;
- CPWMS kad je setovan na 1 onda se u PWM modu generiše centralno podešen PWM signal;
- TOIE setovanjem na log. 1 omogućuje interrupt od prekoračenja glavnog brojača (kada od maksimalne vrednosti postane 0)
- TOF je bit koji se setuje kada dođe do prekoračenja glavnog brojača. Ako je dozvoljen interrupt, sa TOIE = 1, onda ovaj bit pokreće interrupt mehanizam.

Sa resetom svi bitovi registra TPMxSC se postavljaju na 0. Svi bitovi mogu da se čitaju ili upisuju, osim bita TOF koji može samo da se čita.

CLKSB:CLKSA	Izvor takta tajmera
00	Takt nije selektovan, tajmer je disejblovan
01	BUS klok
10	Fiksni sistemski klok
11	Ekstrni klok

Fiksni sistemski klok daje oscilator mikrokontrolera, a BUS klok se dobija iz PLL kola ili FLL kola. Eksterni klok mora da ima četri puta manju frekvenciju od BUS frekvencije.

PS2:PS1:PS0	Preskaler deli sa:
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Svaki kanal, svake tajmerske jedinice poseduje statusni i kontrolni registar TPMxCnSC. Broj tajmerske jedinice je x, a broj kanala n. Bitovi ovog registra, poredjani od najnižeg su sledeći:

- Dva najniža bita se ne koriste, u njih se ne može ništa upisati, a čitanjem se dobijaju logičke nule na njihovom mestu;
- ELSnA, ELSnB programiraju port logiku, određujući aktivnu ivicu;
- MsnA, MSnB selektuje mod rada tajmerskog kanala;
- CHnIE bit koji dozvoljava interrupt tajmerskog kanala ako je setovan na log. 1;
- CHnF bit koji se setuje na svaki događaj tajmerskog kanala, i koji pokreće interrupt ako je on dozvoljen.

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mod	Konfiguracija
x	xx	00	Tajmerski kanal ne koristi spoljni pin, i on se može koristiti kao I/O pin	
0	00	01	Input capture	Capture usponske ivice
		10		Capture opadajuće ivice
		11		Capture bilo koje ivice
	01	00	Output compare	Izlaz nije povezan na pin, moguć je samo interrupt
		01		Menja log. nivo pina na događaj
		10		Kliruje pin na događaj(1)
		11		Setuje pin na događaj
	1x	10	Ivično podesiv PWM	Neinvertovan PWM signal
		x1		Invertovan PWM signal
1	xx	10	Centralno podesiv PWM	Neinvertovan PWM signal
		x1		Invertovan PWM signal

Napomena 1) pod događajem se smatra trenutak jednakosti vrednosti glavnog brojača i vrednosri u registru podataka tajmerskog kanala.

Analogno digitalni konvertor

ADC (analogno digitalni konvertor) je interni modul mikrokontrolera koji analogni signal, prisutan na odgovarajućem pinu, pretvara u odgovarajuću brojnu vrednost. ADC mikrokontrolera familije MC9S08 je C-2C sa sukcesivnim aproksimacijama. Parametri ADC-a su: kvantizacija vrednosti analognog signala po amplitudi, i kvantizacija po vremenu.

Kvantizacija po amplitudi je rezolucija ADC-a, odnosno njegova sposobnost da razlikuje određen broj vrednosti analognog signala. Ako se kaže da je neki ADC 8-o bitni, to znaci da on razlikuje 256 (2^8) nivoa analognog signala, i svakom nivou odgovara jedan broj iz skupa vrednosti od 0 do 255. Krajnjim vrednostima (0 i 255) moraju da odgovaraju dva analogna nivoa VrefL (za 0) i VrefH (za 255). Ako analogni signal uvek ima vrednost izmedju ova dva nivoa, onda će ADC uspešno konvertovati njigovu vrednost u odgovarajući broj. Mikrokontroleri familije MC9S08 imaju izdvojene pinove za VrefL i VrefH. U praksi se najčešće VrefL vezuje na signalnu masu, a VrefH na napajanje mikrokontrolera. Takođe, veća kućišta MC9S08 imaju pinove (2 pina) preko kojih se može posebnim naponom napajati ADC. Ova mogućnost dozvoljava dovođenje ADC-u posebno dobro filtriran napon (što za napajanje mikrokontrolera nije neophodno) kako bi se iskoristila visoka rezolucija ADC-a od 10 bita (1024 nivoa kvantizacije) a posebno kod MC9S08QE koji ima 12-to bitni ADC (4096 nivoa kvantizacije).

Kvantizacija po vremenu određuje najmanje vreme između dve uzastopne konverzije. Zapravo to je brzina konverzije. Iako je ADC 10-to bitni ili 12-to bitni, moguc je mod 8-o bitne konverzije, 10-to bitne konverzije i kod MC9S08QE 12-to bitne konverzije. Pri 8-o bitnoj konverziji brzina konverzije je 2.125μs, a kod 10-to bitne i 12-to bitne 2.5 μs.

Izvor takta za ADC se bira između: BUS frekvencije, BUS/2 frekvencije, frekvencije direktno iz oscilatora a pre FLL kola, i frekvencije koja se generise u samom ADC modulu i može da se koristi u wait modu ili stop3 modu. Izabrana frekvencija se deli preskalerom sa: 1, 2, 4 ili 8. Uobičajeno je da se pre konverzije analogni signal sempluje u trajanju od 3.5 ADC kloka. To je u redu kada izvor analognog signala ima malu izlaznu impedansu (kao operacioni pojačivač). Međutim, kada je izlazna impedansa izvora analognog signala velika, treba programirati ADC da ima produžen impuls za semplovanje koji tada iznosi 23.5 ADC kloka. Ovo je potrebno da bi se napunili interni kondenzatori, i kako bi konverzija bila tačna. Cena produženog impulsa za semplovanje je za toliko duže vreme konverzije.

ADC može izvući mikrokontroler iz wait ili stop3 moda ako se kao izvor njegovog takta koristi njegov sopstveni oscilator. To znači da kada se mikrokontroler zaustavi radi uštede napajanja (ako je ono baterijsko), može se ponovo pokrenuti kada ulazni analogni signal dostigne neki nivo. Da bi ova funkcija mogla da radi, u ADC-u postoji jedan 16-to bitni registar i komparator. Komparator upoređuje vrednost ovog registra sa konvertovanom vrednošću analognog signala, i kada se one izjednače mikrokontroler se ponovo pokreće da radi punom brzinom.

Statusno kontrolni registar ADC1SC1

	7	6	5	4	3	2	1	0
	COCO*	AIEN	ADCO	ADCH				
RESET	0	0	0	1	1	1	1	1

*) Može samo da se čita