19. INTEGRISANI DIGITALNI PROCESORI SIGNALA

U prethodna dva poglavlja razmotrena su dva načina implementacije sistema za digitalnu obradu signala čije su karakteristike komplementarne. Softverska implementacija odlikuje se velikom fleksibilnošću, ali i malom brzinom rada, tako da nije pogodna za primene u realnom vremenu. S druge strane, hardverska implementacija se odlikuje vrlo velikom brzinom rada, koja se postiže po cenu male fleksibilnosti realizacije, jer se za svaku izmenu u algoritmu zahteva izmena hardvera.

Razvoj tehnologije integrisanih kola omogućio je pojavu vrlo složenih integrisanih kola, koja se sastoje od više stotina hiljada tranzistora na silicijumskoj pločici (čipu). Tipičan primer takvih složenih kola su mikroprocesori i memorije velikog kapaciteta. Ovako velika gustina integracije omogućila je implementaciju više istih ili različitih funkcionalnih blokova na jednom čipu, čime je omogućena primena mnogih tehnika za povećanje brzine rada. Osim toga, povećanje složenosti implementacije omogućilo je i realizaciju programabilnih digitalnih procesora signala, poznatih pod nazivom *integrisani digitalni procesori signala*.

Preteču današnjih integrisanih digitalnih procesora signala predstavljao je procesor Intel 2920, koji je, prema današnjoj terminologiji, ustvari pripadao klasi mikrokontrolera, jer na čipu nije sadržao paralelni množač već samo aritmetičko-logičku jedinicu, A/D i D/A konvertore i memoriju EPROM tipa. Prvi, pravi integrisani digitalni procesor signala, S2811, najavila je firma AMI iz SAD 1978. godine [N-2], ali je, zbog tehnoloških problema, procesor ušao u komercijalnu proizvodnju tek nekoliko godina kasnije kada je već bio zastareo. U međuvremenu, 1981. godine, Bel laboratorije američkog koncerna AT&T publikovale su podatke o integrisanom digitalnom procesoru signala DSP1 [B-19], koji, međutim, nije bio komercijalno raspoloživ već ga je proizvođač interno koristio u proizvodnji telekomunikacione opreme. Nešto kasnije, japanska firma NEC iznela je na tržište integrisani digitalni procesor signala μPD7720 [N-3], koji se u savremenijim varijantama proizvodi i danas. Počev od 1982. god, sa pojavom prvog integrisanog digitalnog procesora signala iz familije TMS320, TMS32010 [M-2], [M-16], koji je proizvela američka firma Texas Instruments, počinje era integrisanih digitalnih procesora signala.

Integrisane digitalne procesore signala danas proizvodi oko petnaest proizvođača, a broj različitih integrisanih kola prelazi šezdeset. Savremeni integrisani digitalni procesori signala sreću se u tri varijante. Prvu i najveću grupu čine *integrisani digitalni procesori signala opšte namene* kojima se mogu realizovati svi algoritmi obrade signala. Drugu, znatno manju grupu, čine *konvolucioni procesori* koji su specijalizovani za izvršavanje operacije konvolucije i digitalno filtriranje, mada se mogu iskoristiti i za neke druge primene. Treću grupu čine *vektorski procesori* koji su optimizovani za rad sa vektorima i matricama.

U ovom poglavlju prvo će biti razmotreni opšti principi na kojima je zasnovana arhitektura integrisanih digitalnih procesora signala opšte namene, a zatim će biti analizirane karakteristike nekih poznatijih integrisanih digitalnih procesora signala. Posebna pažnja biće posvećena familiji integrisanih digitalnih procesora signala TMS320, s obzirom na izuzetno veliku rasprostranjenost procesora iz ove familije. Osim hardverskih i softverskih karakteristika procesora, biće prikazani i alati za razvoj softvera i hardvera za primene u kojima se koriste integrisani digitalni procesori signala. Na kraju će biti prikazani i integrisani digitalni procesori signala iz ostale dve grupe, tj. konvolucioni i vektorski procesori.

19.1 ARHITEKTURA DIGITALNIH PROCESORA SIGNALA

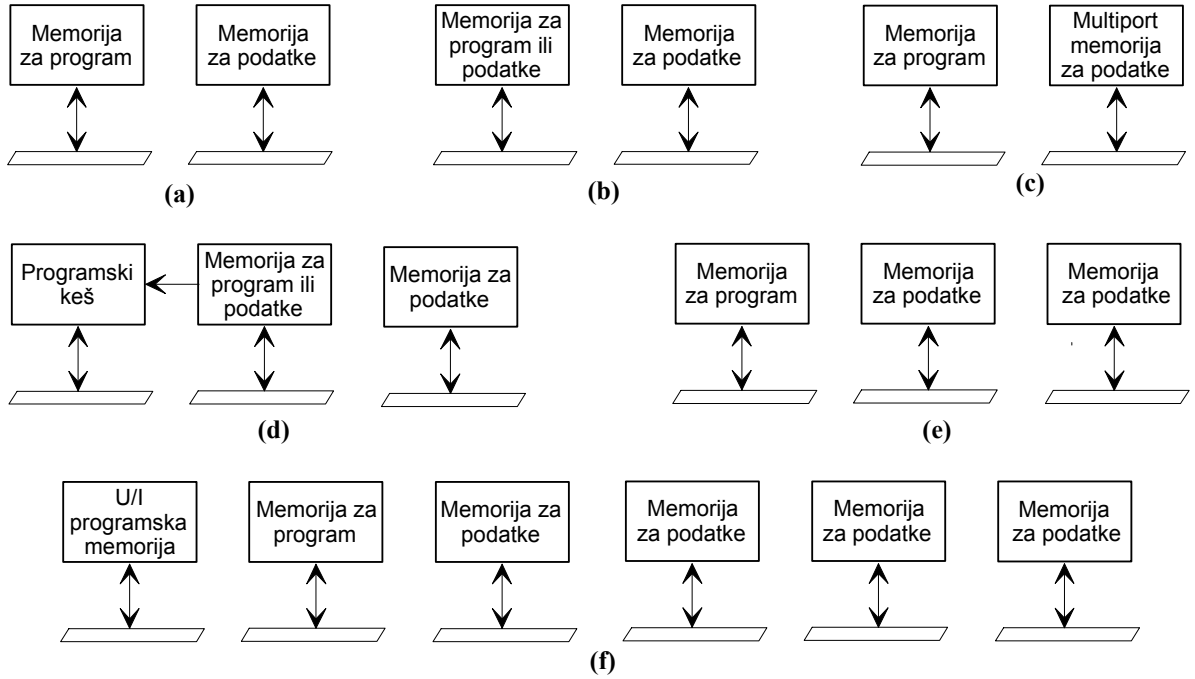
Programabilni integrisani digitalni procesori signala predstavljaju mikroračunare koji su specijalizovani za digitalnu obradu signala u realnom vremenu. S obzirom da algoritmi za digitalnu obradu signala sadrže veliki broj numeričkih izračunavanja, arhitektura digitalnih procesora signala je optimizovana za efikasno izvršavanje aritmetičkih instrukcija. Zbog toga se arhitektura integrisanih digitalnih procesora signala u mnogim aspektima znatno razlikuje od arhitekture konvencionalnih mikroračunara i mikroprocesora. Pre svega, svaki savremeni digitalni procesor signala sadrži integrisani paralelni množač, koji je tako postavljen, da se operacija množenja (a često i sabiranja) može izvršiti u toku samo jednog instrukcijskog ciklusa. Druga bitna razlika odnosi se na arhitekturu memorije, koja znatno odstupa od klasične arhitekture memorije kod mikroračunara. Osim toga, intenzivnim i efikasnim korišćenjem principa protočnosti u izvršavanju instrukcija, postignuto je znatno skraćenje takt intervala i paralelni rad više funkcionalnih jedinica. Navedene razlike, uz još nekoliko manjih, uslovile su drastično povećanje numeričkih performansi integrisanih digitalnih procesora signala u odnosu na klasične mikroračunare iste tehnološke generacije. Optimizacija numeričkih performansi postignuta je, na žalost, po cenu složenijeg postupka programiranja. Naime, programiranje integrisanih digitalnih procesora signala izvodi se najčešće korišćenjem asemblera, jer je kvalitet prevodilaca za više programske jezike još uvek prilično loš.

19.1.1 ARHITEKTURA MEMORIJE

Kod klasičnih mikroračunara memorija je povezana sa centralnim procesorom pomoću magistrale koja se sastoji od izvesnog broja adresnih linija i izvesnog broja linija kojima se vrši transfer podataka od procesora do memorije i obratno. Za ovu arhitekturu, koja se u računarskoj literaturi naziva arhitektura Fon Nojmanovog (Von Neumann) tipa, karakteristično je da su program i podaci smešteni u istoj memoriji i da im se pristupa preko iste magistrale. Da bi se izbeglo zagušenje magistrale zbog čestog pristupa memoriji, centralni procesor obično sadrži skup registara za čuvanje međurezultata, čime se smanjuje protok podataka po magistrali. Međutim, u digitalnoj obradi signala se obično radi sa vrlo velikim količinama podataka, tako da uvođenje registara ne doprinosi mnogo smanjenju protoka podataka po magistrali. Zbog toga se, u integrisanim digitalnim procesorima signala, memorija obično deli na dve ili više memorijskih banki kojima se može simultano pristupiti. Takva arhitektura je poznata pod nazivom *Harvard arhitektura*.

19.1.1.1 Harvard arhitektura memorije

U osnovnoj verziji Harvard arhitekture, memorija je podeljena na dve nezavisne banke do kojih vode dve posebne magistrale. Jedna od memorija služi za smeštaj programa (instrukcija) a druga za smeštaj podataka, kao što je prikazano na slici 19.1a. Na taj način se omogućava da se učitavanje jedne instrukcije izvršava simultano sa učitavanjem podatka (operanda) potrebnog za izvršavanje prethodne instrukcije. Pod uslovom da je ostatak hardvera dovoljno brz, instrukcije sa jednim operandom iz memorije se mogu izvršavati za vreme jednako memorijskom ciklusu. Nasuprot tome, kod klasičnih mikroračunara, učitavanje instrukcije i operanda se vrši na sekvencijalni način iz iste memorije, što po pravilu zahteva znatno više vremena.

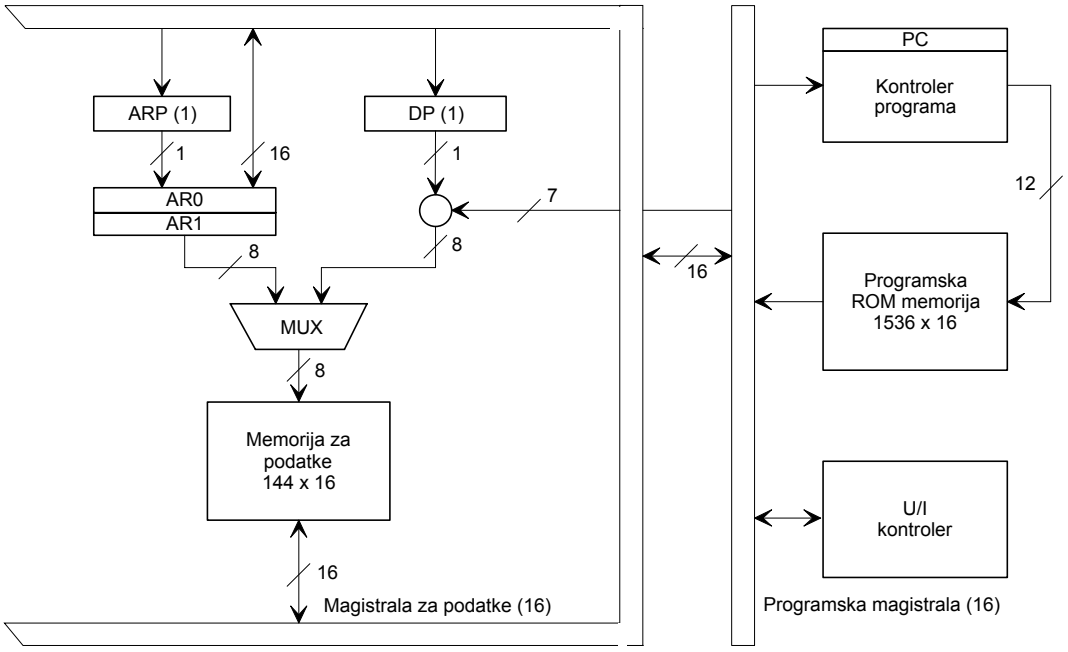


Slika 19.1 Harvard arhitektura memorije (a), i njene modifikacije (b)-(f).

Osnovna Harvard arhitektura korišćena je kod starijih integrisanih procesora signala. Na primer, kod poznatog procesora TMS32010 [T-2], primenjena je Harvard arhitektura memorije sa malom izmenom, koja omogućava transfer podataka iz memorije za podatke u programsku memoriju i obratno, korišćenjem instrukcija `TBLR` i `TBLW`. Uprošćeni blok dijagram koji prikazuje pristup memoriji kod procesora TMS32010 prikazan je na slici 19.2.

Noviji integrisani procesori signala koriste razne modifikacije osnovne Harvard arhitekture u cilju poboljšanja performansi, ili povećanja fleksibilnosti pri programiranju. Na primer, na slici 19.1b je prikazana prva modifikacija Harvard arhitekture, koja se sastoji u tome da se omogući direktno korišćenje podataka koji se nalaze u programskoj memoriji prilikom izvršavanja instrukcije. Naravno, u tom slučaju se iz memorije ne može očitavati instrukcija. Na taj način, omogućeno je da se instrukcije sa dva ili tri operanda izvršavaju u svega dva memorijska ciklusa. Ovaj princip je zastupljen kod mnogih integrisanih digitalnih procesora druge generacije kao što su TMS32020 [M-3] i TMS320C25 [F-5]. Kod ovih procesora vreme izvršavanja instrukcije je jednako memorijskom ciklusu, tako da se instrukcije sa dva operanda iz memorije izvršavaju u dva memorijska ciklusa. Dvostruka uloga programske memorije se može još efikasnije eksploatisati ako se naprave još neke modifikacije o kojima će kasnije biti reči.

Istu modifikaciju sa slike 19.1b koriste i procesori signala DSP32 i DSP32C [K-12], [F-9], ali na drugačiji način. Kod ovih procesora instrukcijski ciklus traje dva memorijska ciklusa. Druga razlika je da obe memorijske banke mogu sadržati i program i podatke. Dakle, u svakom instrukcijskom ciklusu može se dva puta pristupiti do obe memorijske banke i pročitati instrukcija i do tri operanda.



Slika 19.2 Blok dijagram arhitekture memorije digitalnog procesora signala TMS32010.

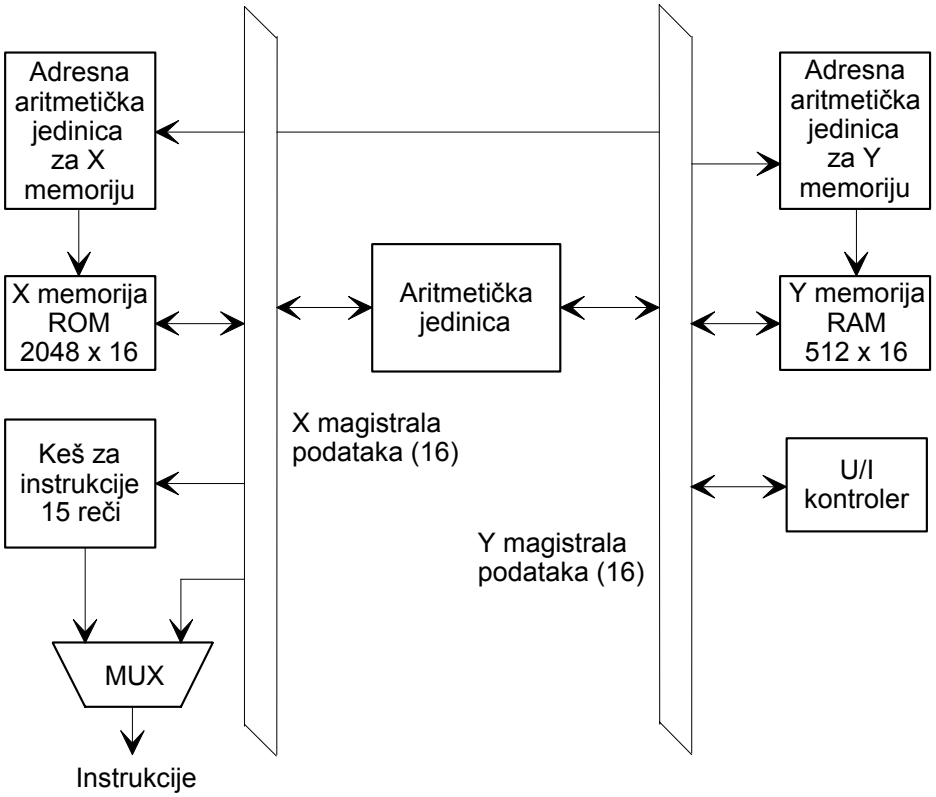
Druga modifikacija osnovne Harvard arhitekture, koja je prikazana na slici 19.1c, koristi multiportnu (višepristupnu) memoriju za podatke. Mada je ovo rešenje jednostavno sa gledišta programiranja, retko se primenjuje u praksi. Razlog za to je visoka cena brzih statičkih memorija sa više pristupa. Jedini procesor koji koristi ovo rešenje je Fujitsu MB86232 čija memorija za podatke, kapaciteta 512 reči, ima tri pristupa [G-3].

Treća modifikacija Harvard arhitekture, koja je prikazana na slici 19.1d, dobija se kada se memoriji za program ili podatke, u varijanti sa slike 19.1b, doda keš za instrukcije. Za izvršavanje instrukcija koje se nalaze u kešu nije potrebno pristupati programskoj memoriji, pa se iz nje mogu očitavati podaci. Kod procesora druge generacije TMS32020 i TMS320C25, koji imaju keš samo za jednu instrukciju, ovaj princip je implementiran na sledeći način. U skupu instrukcija postoji instrukcija `RPTK` koja omogućava ponavljanje izvršenja naredne instrukcije specificirani broj puta. Na primer, deo asemblerskog programa za ova dva procesora može izgledati ovako:

```
RPTK    N
MACD    m1, m2
```

gde instrukcija `RPTK` izaziva učitavanje sledeće instrukcije `MACD` u instrukcijski keš, dekodovanje instrukcije `MACD` i izvršavanje instrukcije `MACD` $N+1$ puta. Posle učitavanja instrukcije `MACD` programska memorija više nije potrebna, pa se u drugom i sledećem izvršavanju instrukcije `MACD` podaci koji se nalaze na adresama `m1` (u programskoj memoriji) i `m2` (u memoriji za podatke) mogu simultano učitavati. Instrukcija `MACD` takođe obezbeđuje i inkrementiranje, ili dekrementiranje, adresa `m1` i `m2`. Dakle, instrukcija `MACD` se prvi put izvršava u dva ciklusa, dok je za svako sledeće izvršavanje potreban samo po jedan ciklus.

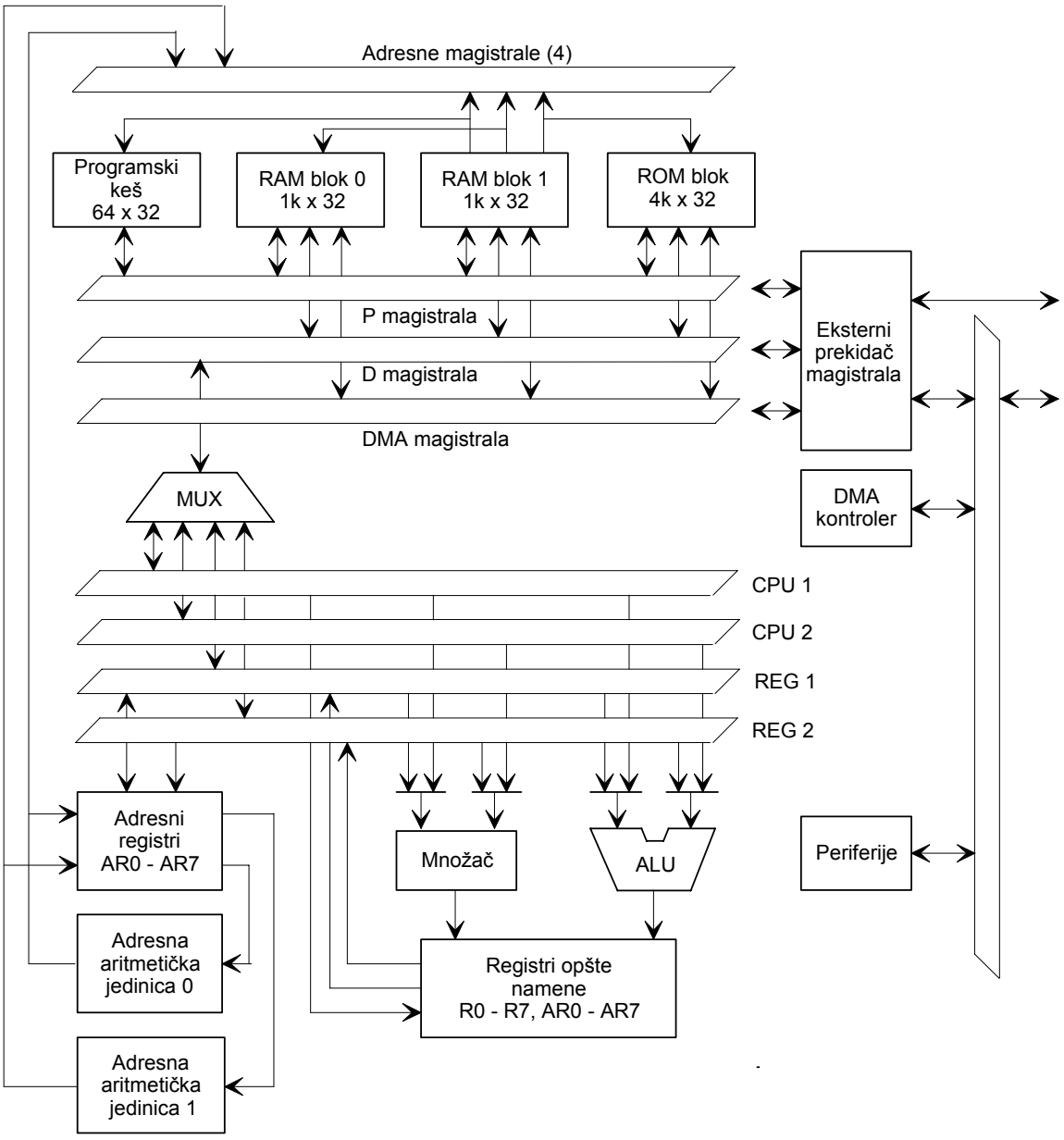
Kod klasičnih mikroračunara, prilikom iterativnog izvršavanja petlje, koristi se softverski brojač prolazaka kroz petlju koji se na početku rada postavi na traženu vrednost, a zatim se na kraju svakog prolaza vrši testiranje i grananje. Na taj način se, skoro neprimetno za programera, povećava vreme za izvršavanje instrukcija u petlji. Vreme koje se troši na izvršavanje petlje nezavisno od njenog sadržaja naziva se *režijsko vreme* (engl. overhead). Instrukcija `RPTK` uz odgovarajući hardver i keš za instrukcije omogućava formiranje petlje bez instrukcija (i utroška vremena) za postavljanje stanja brojača, testiranje i grananje. Takve petlje nazivaju se *petlje sa malim režijskim vremenom* (engl. low overhead loops). Ovaj princip je prvi put primenjen kod procesora Hitachi 61810 [H-1], a posle toga je implementiran kod skoro svih novijih procesora. Da bi se efikasnije podržao rad sa petljama sa malim režijskim vremenom, neki od procesora imaju keš za veći broj instrukcija. Na primer, procesori DSP16 i DSP16A imaju keš za 15 instrukcija, procesor ADSP-2100 [R-27] ima keš za 16 instrukcija, a najveći programski keš za 64 instrukcije ima procesor TMS320C30. Na slici 19.3 je prikazana arhitektura memorije digitalnih procesora signala DSP16 i DSP16A koje proizvodi firma AT&T.



Slika 19.3 Arhitektura memorije digitalnih procesora signala DSP16 i DSP16A.

Sledeća, četvrta modifikacija Harvard arhitekture memorije, koja je prikazana na slici 19.1e, razlikuje se od prethodnih modifikacija po tome što ima tri memorijske banke. Poznati procesor TMS320C30 [S-8], [P-2] koristi ovaj tip arhitekture s tim što su dve memorijske banke RAM tipa a jedna memorijska banka ROM tipa, što je prikazano na slici 19.4. Ako je program smešten u ROM memoriji, onda je za izvršavanje instrukcije sa dva operanda potreban samo jedan memorijski ciklus. Interesantno je takođe da se kod procesora TMS320C30 *svakoj memorijskoj banci može pristupiti dva puta u toku instrukcijskog ciklusa*. Tako se i u slučaju kada se dva operanda nalaze u istoj memoriji, za izvršavanje instrukcije troši samo jedan instrukcijski ciklus. Ukoliko dođe do konflikta, na primer ako treba tri puta pristupiti istoj memoriji u jednom instrukcijskom ciklusu,

konflikt se hardverski detektuje i izvršavanje naredne instrukcije se odloži za jedan ciklus. Procesor TMS320C30 ima programski keš za 64 instrukcije, što je takođe prikazano na slici 19.4. Kod ovog procesora, programski keš nema nikakvog efekta kada je program smešten u ROM memoriji na čipu, međutim, kada se program nalazi u memoriji van čipa, postojanje programskog keša može znatno ubrzati izvršavanje programa.



Slika 19.4 Uprošćeni blok dijagram digitalnog procesora signala TMS320C30.

Zbog postojanja programskog keša, kod procesora TMS320C30 mogu se realizovati dva tipa petlji sa malim režijskim vremenom. Prvi tip petlje se formira pomoću instrukcije `RPTS` u slučaju kada se u petlji nalazi samo jedna instrukcija. Format instrukcije ima oblik:

```
RPTS    N
```

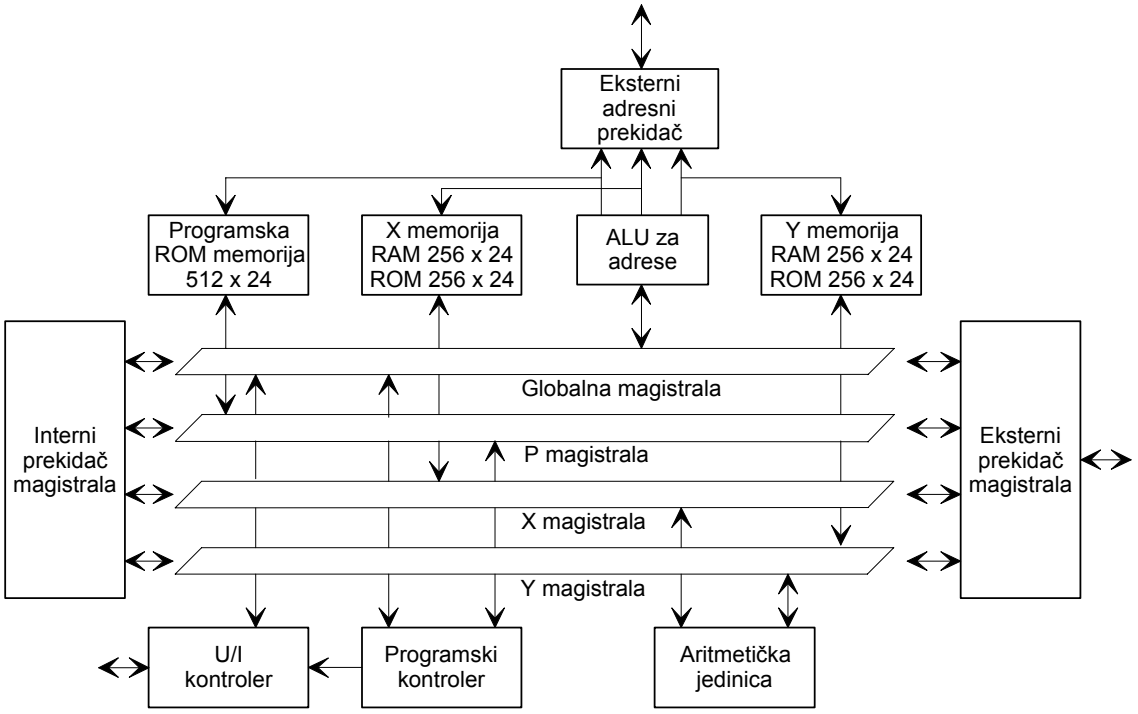
Instrukcija `RPTS` je po funkciji veoma slična sa instrukcijama `RPT` i `RPTK` kod procesora TMS32020/C25 jer se sledeća instrukcija ponavlja *N* puta, pri čemu se samo jednom učitava i dekoduje, a zatim se drži u registru za instrukcije tako da su magistrale slobodne za transfer podataka. Interesantno je da se ovakva petlja ne može prekinuti signalom za prekid.

Drugi tip petlje se formira instrukcijom `RPTB` u slučaju kada se u petlji nalazi više instrukcija. Format takve instrukcije je:

```
        RPTB    labela
        .......
        instrukcije u petlji
        .......
labela  poslednja instrukcija petlje
```

Za ovakav način formiranja petlje koriste se tri od 12 kontrolnih registara. U jednom registru se čuva adresa početka bloka instrukcija koji čini petlju, u drugom adresa kraja bloka, dok se treći registar koristi kao brojač prolazaka kroz petlju. Asembler automatski postavlja sadržaj prva dva registra, dok korisnik inicijalizuje brojač pre ulaska u petlju. Na ovaj način petlja se izvršava istom brzinom kao da je napisana u razvijenom (engl. straight-line) kodu.

Isti tip arhitekture memorije, ali bez keša za instrukcije, primenjen je i kod procesora DSP56001 [K-14] i DSP96002 [S-12]. Na slici 19.5 je prikazana arhitektura memorije digitalnog procesora signala DSP56001. U ovoj arhitekturi, pored programske memorije, postoje i dve memorije za podatke X i Y, pa je moguće u istom memorijskom ciklusu učitati instrukciju i dva operanda. Interesantno je, takođe, da je polovina svake memorije za podatke ROM tipa. Ove dve ROM memorije su unapred programirane i sadrže tablice trigonometrijskih funkcija potrebnih za realizaciju DFT algoritama kao i tablice za kompresiju i ekspanziju digitalizovanih govornih signala koji se sreću kod impulsno kodovane modulacije.



Slika 19.5 Uprošćeni blok dijagram digitalnog procesora signala DSP56001.

Procesor DSP96002, koji je po arhitekturi sličan sa procesorom DSP56001, ima dodatnu mogućnost da se u svakom instrukcijskom ciklusu može *dva puta* pristupiti svim memorijama, ali se drugi pristup memoriji, za razliku od procesora TMS320C30, koristi samo za DMA transfer.

Petlje sa malim režijskim vremenom mogu se konstruisati i bez keša za instrukcije što je primenjeno kod procesora DSP56001 [K-14] i DSP96002 [S-12]. Pošto se instrukcije drže u programskoj memoriji, a ne u kešu, petlja može sadržati proizvoljan broj instrukcija, petlja se može

prekinuti, i može se formirati petlja unutar petlje. Smatra se da je od svih integrisanih procesora signala formiranje petlji kod ova dva procesora rešeno na najefikasniji način. Asemblerska sintaksa za petlju je veoma jednostavna, što se vidi iz sledećeg primera:

```
        do   10, end
        .......
        instrukcije u petlji
        .......
end
```

Na kraju, posmatrajmo još jednu modifikaciju Harvard arhitekture koja je prikazana na slici 19.1f, a koja je primenjena kod procesora Hitachi DSPi [K-6]. Memorija je podeljena na šest memorijskih banki, kojima se može istovremeno pristupiti. Na primer, za izvršavanje jedne instrukcije sa tri operanda (dva očitavanja i jedan upis), koriste se tri memorije za podatke i programska memorija. Istovremeno, neka U/I instrukcija iz U/I programske memorije, može koristiti slobodnu četvrtu memoriju za podatke. Mada su na ovaj način drastično povećane mogućnosti za transfer podataka, programiranje je veoma otežano, jer se zahteva posebno uređenje podataka za puno iskorišćenje mogućnosti pristupa memorijama.

Kod integrisanih digitalnih procesora signala može se definisati *faktor potražnje* koji predstavlja ukupan broj pristupa memorijama u toku jedne instrukcije. Ovaj faktor zavisi od broja memorijskih banki i broja memorijskih ciklusa u jednom instrukcijskom ciklusu. Faktor potražnje kod savremenih procesora kreće se u granicama od dva do šest. Ako je faktor potražnje dva, onda se radi efikasnijeg iskorišćenja memorije mora primeniti programski keš, poseban način adresiranja poznat kao cirkularno adresiranje, ili specijalni ciklus upisa. Procesori koji imaju faktor potražnje tri nemaju programski keš, ali i dalje koriste cirkularno adresiranje za kašnjenje podataka. Ako je faktor potražnje četiri ili više, nije potreban ni programski keš ni cirkularno adresiranje, jer ima dovoljno mogućnosti za pristup memoriji.

19.1.1.2 Interne i eksterne memorije

Kao što je već rečeno, integrisani digitalni procesori signala predstavljaju kompletne integrisane mikroračunare, i, kao takvi, moraju na čipu imati i memoriju. Međutim, pošto se po pravilu radi o brzoj memoriji, za realizaciju RAM memorije se koriste *statičke memorije* koje zahtevaju veliku površinu čipa. Zbog toga je kapacitet *interne RAM memorije* na čipu mali i retko prelazi nekoliko stotina reči. Ukoliko je potreban veći kapacitet RAM memorije, ostatak memorijskog prostora se popunjava *eksternim memorijskim modulima* izvan procesorskog čipa.

Bar jedan deo interne memorije može se realizovati kao ROM memorija. U praksi se ROM memorija najčešće koristi za realizaciju interne programske memorije. Programska memorija se u tom slučaju može proširiti dodavanjem eksternih memorijskih modula. Ukoliko je interna programska memorija ROM tipa, kod nekih procesora ona se može učiniti neaktivnom, tako da kompletna programska memorija bude eksterna, što je posebno pogodno u fazi razvoja. Da bi se olakšao razvoj uređaja sa digitalnim procesorima signala, koriste se i procesori sa internom programskom memorijom EPROM tipa. Takvi procesori su još uvek retki, s obzirom na visoku cenu brzih EPROM memorija.

Uvođenje eksterne memorije ima i negativne posledice. Pre svega, ako treba realizovati više eksternih memorijskih banki, potreban je vrlo veliki broj spoljašnjih priključaka što povećava cenu procesora. Osim toga, pristup spoljašnjoj memoriji je često usporen, jer je ponekad potrebna i dodatna logika za upravljanje memorijom. Zbog toga većina savremenih integrisanih procesora

signala ima dovoljno interne memorije za najvažnije operacije. Pošto se time smanjuje potreba za eksternom memorijom, *interne magistrale se multipleksiraju* i postoji samo jedna eksterna magistrala. Projektant onda mora da odluči koju će memorijsku banku proširiti izvan čipa da bi postigao maksimalnu brzinu rada, odnosno da se pomiri sa činjenicom da formiranje dve eksterne memorijske banke smanjuje brzinu rada. Kod nekih novijih procesora, na primer kod DSP96002, moguće je formirati dve eksterne memorijske banke a da se ipak postigne maksimalna brzina rada.

Performanse sistema koji koristi eksternu memoriju mogu se znatno popraviti ako se koristi keš za instrukcije, koji se sreće kod mnogih procesora signala novije generacije. Kada se učitava instrukcija iz eksterne memorije, ona se istovremeno smešta u keš. Kada se kasnije koristi ista instrukcija, prvo se proverava da li se nalazi u kešu, i ako se nalazi, ne učitava se iz eksterne memorije. Time se značajno ubrzava rad sistema, jer je učitavanje iz programskog keša znatno brže a, osim toga, multipleksirana magistrala je slobodna za transfer podataka. Programski keš je naročito značajan kod izvršavanja petlji, o čemu je ranije već bilo reči.

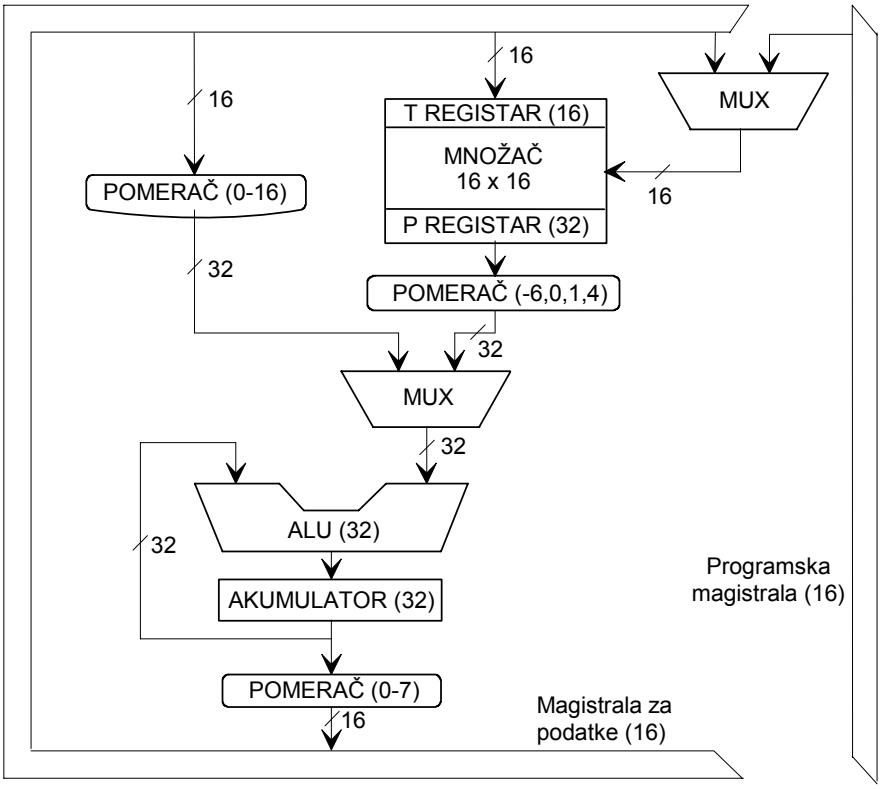
19.1.2 ARHITEKTURA ARITMETIČKO-LOGIČKE JEDINICE

Aritmetička jedinica integrisanih digitalnih procesora signala mora da bude prilagođena za *efikasno izračunavanje sume proizvoda* koja se pojavljuje kod svih klasičnih algoritama digitalne obrade signala: DFT, FIR i IIR filtriranja i konvolucije. Zbog toga svaka aritmetička jedinica mora imati brzi *množač* i *sabirač*. Radi izvođenja logičkih operacija, sabirač se često zamenjuje univerzalnom aritmetičko-logičkom jedinicom (ALU). Osim ova dva funkcionalna bloka, potrebni su još i *registri* za prihvatanje operanada i međurezultata, kao i *pomerači* kojima se se vrši skaliranje signala za vreme transfera iz bloka u blok. Množač se u aritmetičkim jedinicama digitalnih procesora signala najčešće realizuje kao paralelni kombinacioni množač, dok se sabirač realizuje korišćenjem ALU i jednog registra koji formiraju akumulacioni sabirač.

Kao tipičan primer arhitekture aritmetičko-logičke jedinice, na slici 19.6 je prikazana arhitektura aritmetičko-logičke jedinice jednog od najkorišćenijih digitalnih procesora signala, TMS320C25 [F-5], kod koga se za predstavljanje podataka koristi 16-bitni format sa fiksnom tačkom. Binarna tačka je postavljena odmah iza bita za znak. Pošto efekti kvantovanja kod 16-bitnih reči mogu biti značajni, aritmetička jedinica je tako koncipirana da obezbedi veliku brzinu rada uz smanjenje uticaja kvantovanja. Postupak izračunavanja sume proizvoda počinje resetovanjem akumulatora i dovođenjem jednog činioca preko magistrale podataka u prihvatni T registar. Drugi činilac se može dovesti u narednom ciklusu na drugi ulaz množača preko bilo koje od magistrala. Naravno, ako se programska magistrala ne koristi za čitanje instrukcije, oba činioca se mogu istovremeno pročitati, čime množenje postaje jednociklusna instrukcija. Rezultat množenja se upisuje u 32-bitni P registar i u narednom ciklusu se može sabrati sa akumulatorom. ALU u kojoj se vrši sabiranje i akumulatorski registar su napravljeni u proširenoj tačnosti od 32 bita. Skraćenje na potrebnu dužinu od 16 bita se vrši tek posle izračunavanja kompletne sume proizvoda. Time je u velikoj meri smanjen uticaj kvantovanja proizvoda.

Drugi važan problem koji se pojavljuje kod predstavljanja brojeva sa fiksnom tačkom je moguće prekoračenje opsega prilikom sabiranja. Ovaj problem se može rešiti na tri načina. Prvi način je da se prilikom formiranja sume proizvoda svaki sabirak podeli sa istim faktorom (najčešće stepenom broja dva), tako da ne dođe do prekoračenja opsega. Kod procesora TMS320C25 u tu svrhu je dodat programabilni pomerač (kombinaciona mreža) između P registra i ALU kojim se vrši

traženo skaliranje proizvoda bez utroška dodatnog vremena. Ovaj pomerač može izvršiti pomeranje za 6 bita udesno, propustiti podatak bez pomeranja, ili izvršiti pomeranje za 1 ili 4 bita ulevo. Drugi način je da se koristi sabiranje sa ograničenjem opsega, čime se samo delimično rešava problem prekoračenja. Treći način je da se ALU i akumulator prošire i sa leve strane odgovarajućim brojem bita (u praksi dva do sedam), čime se omogućava sabiranje većeg broja sabiraka.

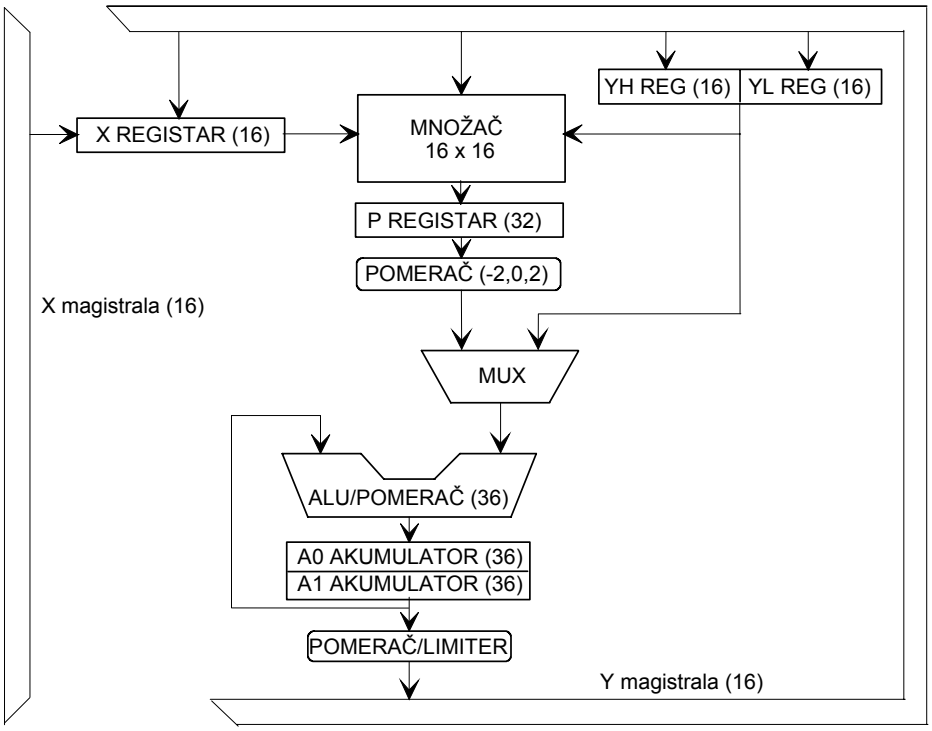


Slika 19.6 Uprošćena struktura aritmetičke jedinice integrisanog digitalnog procesora signala TMS320C25.

U kolu sa slike 19.6 postoje još dva pomerača. Prvi pomerač, preko koga se podatak sa magistrale podataka direktno može uvesti u ALU, vrši programabilno pomeranje za 0 do 16 bita ulevo i ima 16-bitni ulaz i 32-bitni izlaz. Drugi pomerač pomera za 0 do 7 bita ulevo prilikom smeštanja rezultata iz akumulatora u memoriju za podatke.

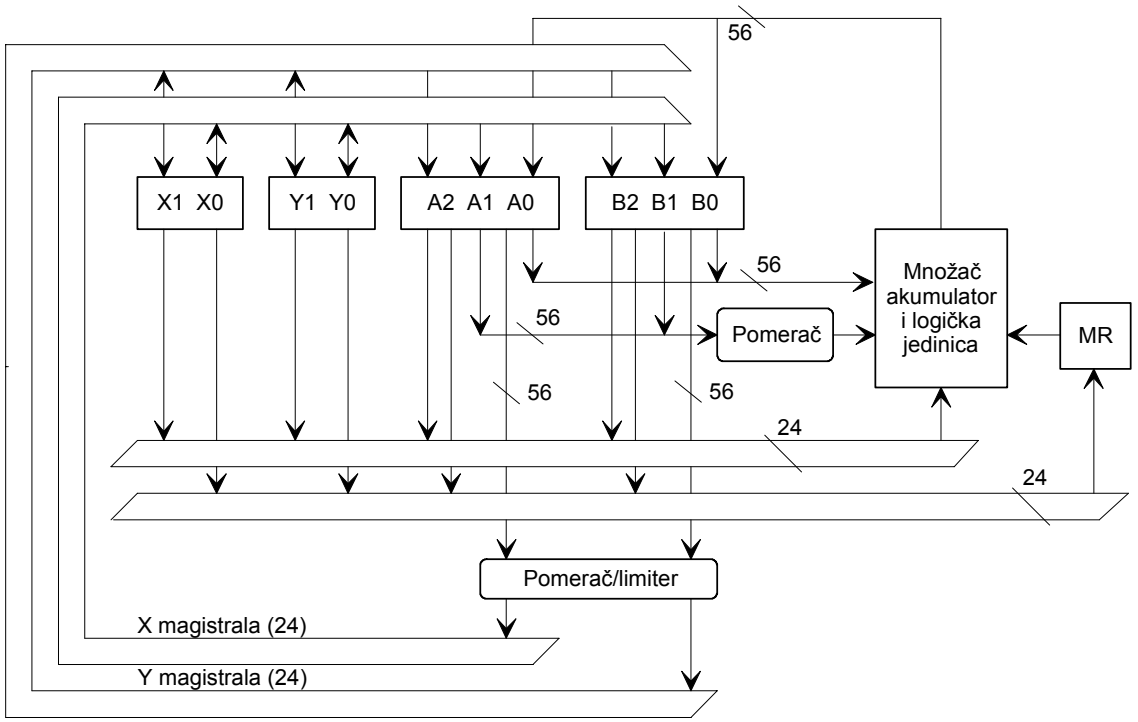
Arhitekture aritmetičko-logičkih jedinica većine novijih procesora imaju vrlo sličnu strukturu. Na primer, na slici 19.7 je prikazana arhitektura aritmetičke jedinice digitalnog procesora signala DSP16 (DSP16A). Kod ovog procesora implementirana su sva tri načina za eliminisanje prekoračenja opsega prilikom sabiranja. Pošto je izvršeno proširenje ALU i akumulatora za četiri bita sa leve strane, moguće je sabrati $2^4 - 1 = 15$ 32-bitnih proizvoda sa akumulatorom, a da se sigurno ne pojavi prekoračenje opsega. Korišćenje dva akumulatora proširene tačnosti olakšava rad sa kompleksnim brojevima, jer se i realni i imaginarni deo međurezultata mogu zapamtiti u proširenom obliku. Još jedna razlika aritmetičke jedinice procesora DSP16 u odnosu na aritmetičku jedinicu procesora TMS320C25 je u tome što DSP16 ima prihvatne registre za oba činioca kod množača, od kojih činilac koji dolazi preko Y magistrale može biti smešten u YH ili YL registar.

Kao treći primer aritmetičko-logičke jedinice, posmatrajmo aritmetičko-logičku jedinicu digitalnog procesora signala DSP56001, koji proizvodi Motorola, a koja je prikazana na slici 19.8.



Slika 19.7 Uprošćena struktura aritmetičke jedinice integrisanog digitalnog procesora signala DSP16.

Kod ovog procesora su takođe preduzeta sva tri načina za sprečavanje prekoračenja opsega prilikom sabiranja. Pošto je kod procesora DSP56001 korišćena dužina reči od 24 bita, ALU i akumulatorski registri A i B su dužine 56 bita, od kojih su 48 korišćeni za predstavljanje razlomačkog dela, a 8 za celobrojni deo i znak. Time je omogućeno sabiranje 255 sabiraka bez prekoračenja opsega.



Slika 19.8 Uprošćena struktura aritmetičke jedinice integrisanog digitalnog procesora signala DSP56001.

U odnosu na prethodno opisane realizacije aritmetičko-logičke jedinice, novina je još postojanje prihvatnih registara X i Y dvostruke dužine od 48 bita čiji se delovi mogu i nezavisno puniti

preko X i Y magistrala. Za razliku od prethodnih realizacija aritmetičko-logičke jedinice, u ovoj realizaciji ne postoji registar za prihvatanje proizvoda jer su operacije množenja i akumuliranja proizvoda sjedinjene u jedinstvenu hardversku celinu. Takav integrisani blok za izvođenje aritmetičkih operacija može da u jednoj instrukcijskom ciklusu (97.5 ns pri taktu od 20.5 MHz) izvrši množenje i sabiranje ili neku logičku operaciju. Tom prilikom, jedan od 56-bitnih ulaza služi kao ulaz u akumulator, dok se oba 24-bitna činioca dovode preko drugog 56-bitnog ulaza. Druga razlika je da se sabirak ne može nikako dovesti u računsku jedinicu osim preko množača. Tako množač preuzima ulogu ulaznog pomerača koji vrši pomeranje ulevo ili udesno za 1-23 bita množenjem podatka sa konstantom ili promenljivom. U kolu postoje dva pomerača. Jedan pomerač se nalazi na putu od akumulatora do aritmetičkog bloka i može da vrši pomeraj 56-bitnog podatka iz akumulatorskog registra A ili B za jedan bit ulevo ili udesno. Drugi pomerač u kolu se nalazi na putu podataka od akumulatora do magistrala. Taj pomerač takođe pomera podatak koji se šalje na magistralu za jedan bit ulevo ili udesno, a u njega je ugrađen ograničavač, koji sprečava prekoračenje opsega prilikom prebacivanja sadržaja akumulatora u memoriju.

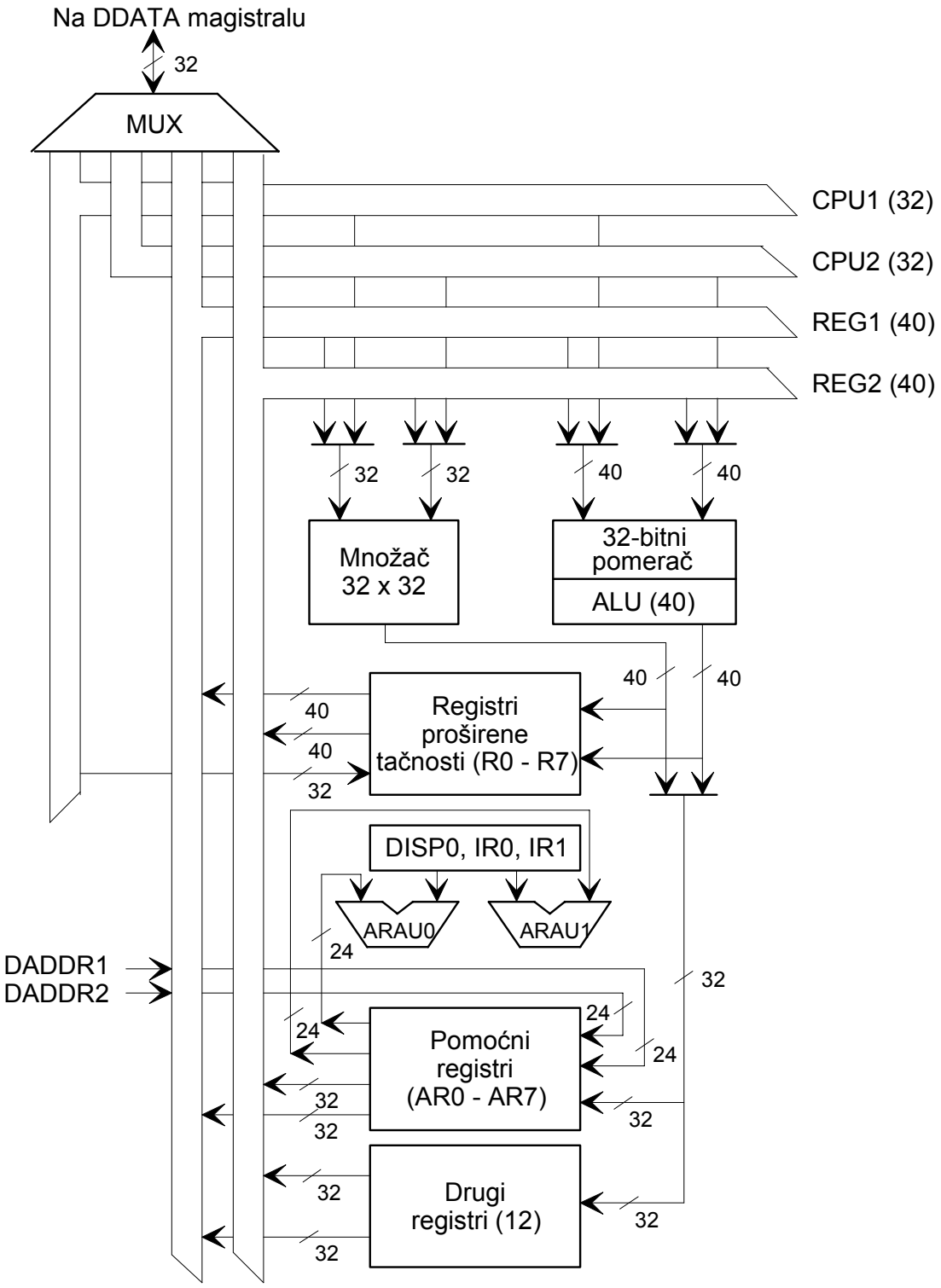
Najnoviji digitalni procesori signala, naročito oni koji koji operišu sa podacima u formatu sa pokretnom tačkom, imaju još složeniju konstrukciju aritmetičko-logičke jedinice. Na primer, na slici 19.9 je prikazana aritmetička jedinica digitalnog procesora signala TMS320C30 [P-2]. Najvažnija razlika u odnosu na prethodne realizacije odnosi se na uvođenje četiri lokalne magistrale i osam registara opšte namene u proširenoj tačnosti od 40 bita, R0-R7, koji služe kao akumulatori.

U procesorskoj jedinici postoje dve 32-bitne, CPU1 i CPU2, kao i dve 40-bitne, REG1 i REG2, lokalne magistrale. Sve četiri magistrale su preko multipleksera povezane na DDATA magistralu preko koje se može pristupiti do bilo koje od dve interne memorije za podatke. Uvođenje lokalnih magistrala oslobađa memorijske magistrale posla oko transfera podataka iz registara do množača ili ALU bloka, tako da se one mogu koristiti za pristup memorijama.

Množač u kolu sa slike 19.9 u jednom ciklusu (60 ns pri taktu od 33.33 MHz) množi dva 32-bitna broja i daje 40-bitni rezultat. Činioci mogu biti celi brojevi ili brojevi sa pokretnom tačkom. Rezultat množenja se uvek smešta u neki od osam registara proširene tačnosti (R0-R7), u neki od osam 32-bitnih pomoćnih registara AR0-AR7, ili u neki od 12 preostalih 32-bitnih kontrolnih registara.

Aritmetičko-logička jedinica ima dva 40-bitna ulaza na koje se mogu dovesti podaci preko REG magistrala iz 40-bitnih registara proširene tačnosti R0-R7, iz pomoćnih registara AR0-AR7 ili nekog od 12 ostalih registara. Podaci za ALU se takođe mogu dovesti i preko CPU magistrala iz neke od memorija za podatke. ALU vrši aritmetičke operacije sa celim brojevima ili brojevima sa pokretnom tačkom kao i logičke operacije. Rezultat iz ALU uvek se smešta u neki od raspoloživih registara. Prilikom izvođenja aritmetičkih operacija sa pokretnom tačkom u množaču ili ALU, vrši se automatska normalizacija rezultata. Ako dođe do prekoračenja opsega eksponenta, postavlja se odgovarajuća maksimalna ili minimalna vrednost i setuje odgovarajući bit koji ukazuje da je došlo do prekoračenja.

Na slici 19.9 se mogu uočiti i dve aritmetičke jedinice ARAU0 i ARAU1 čija je osnovna namena simultano izračunavanje dve adrese. Ovim je omogućena izuzetno velika fleksibilnost indirektnog adresiranja, jer se istovremeno mogu generisati adrese dva podatka koji su potrebni za izvršenje instrukcije. Ovoj fleksibilnosti doprinosi i postojanje dva indeks registra, o čemu će biti više reči kada se govori o načinima adresiranja.



Slika 19.9 Uprošćena struktura aritmetičke jedinice integrisanog digitalnog procesora signala TMS320C30.

19.1.3 PROGRAMIRANJE DIGITALNIH PROCESORA SIGNALA

Implementacija sistema za digitalnu obradu signala pomoću integrisanog digitalnog procesora signala predstavlja odličan kompromis između softverske i hardverske implementacije, tj. ima fleksibilnost softverske implementacije i veliku brzinu rada koja potiče od specijalizovane arhitekture hardvera. Da bi se maksimalno iskoristile prednosti hardverske arhitekture, potrebno je posebnu pažnju pokloniti efikasnosti programiranja. Zbog toga se programi za većinu integrisanih

digitalnih procesora signala pišu u asembleru jer se na taj način najbolje koriste raspoloživi resursi procesora. Viši programski jezici za digitalne procesore signala se znatno ređe koriste jer efikasnost prevodioca za više programske jezike još uvek nije dovoljno dobra.

19.1.3.1 Skup instrukcija

Skup instrukcija za programiranje digitalnog procesora signala mora da podrži numerički intenzivne operacije koje se pojavljuju u digitalnoj obradi signala, ali i operacije opšte namene koje su više karakteristične za nenumeričke operacije klasičnih mikroprocesora opšte namene. Zbog toga se skup instrukcija asemblera za digitalni procesor signala razlikuje od skupa asemblerskih instrukcija klasičnih mikroprocesora. Pre svega, instrukcije za digitalni procesor signala omogućavaju veći paralelizam rada funkcionalnih blokova. Drugo, velika većina instrukcija se izvršava u jednom jedinom ciklusu. Instrukcije koje traju dva ciklusa su znatno ređe, a instrukcije koje traju tri ili više ciklusa se izuzetno retko sreću i to uglavnom za ulazno-izlazne operacije kod starijih procesora.

Po svojoj funkciji, instrukcije se mogu podeliti u nekoliko grupa, od kojih su najvažnije:

1. Aritmetičke instrukcije,
2. Logičke instrukcije,
3. Instrukcije za čitanje i upis podataka,
4. Kombinovane instrukcije za aritmetičke (logičke) operacije i upis podataka,
5. Instrukcije za grananje,
6. Kontrolne instrukcije,
7. Instrukcije za ulazno-izlazne operacije.

Zbog toga što se teži da se svaka instrukcija izvrši u što manjem broju ciklusa, najveći broj instrukcija smešten je u jednoj memorijskoj reči. Samo instrukcije grananja, ulazno-izlazne instrukcije i mali broj instrukcija sa visokim stepenom paralelizma zahteva dve memorijske reči. Na primer, kod procesora TMS320C25 [T-3] skup instrukcija sadrži 133 instrukcije, od kojih su svega 27 smeštene u dve memorijske reči. Što se tiče vremena izvršenja, ono zavisi i od toga da li se koristi samo jedna ili dve eksterne memorijske banke. Ako se koristi samo jedna eksterna memorija, tj. ako se može raditi punom brzinom, 97 instrukcija se izvršava se u jednom instrukcijskom ciklusu. Od 36 višeciklusnih instrukcija, 21 instrukcija se odnosi na grananja, pozive i povratak iz potprograma, tj. to su sve one instrukcije koje zahtevaju izmenu sadržaja programskog brojača. Sedam višeciklusnih instrukcija su dvorečne instrukcije koje sadrže operand u drugoj reči instrukcije. Poslednjih osam instrukcija su ulazno-izlazne instrukcije, instrukcije za transfer podataka između memorija i instrukcije za množenje sa akumulacijom. Ovih osam instrukcija se mogu pretvoriti u jednociklusne, ako se koristi instrukcija za ponovljeno izvršavanje naredne instrukcije `RPT` ili `RPTK`.

19.1.3.2 Načini adresiranja

Uvođenje više memorijskih banki kojima se može simultano pristupiti sigurno povećava brzinu transfera podataka, ali ne rešava jedan važan problem. Naime, jedna instrukcija može zahtevati najviše tri obraćanja memoriji. Čak i ako adresni prostor nije preterano veliki, broj bita koji je potreban za specificiranje sve tri adrese postaje veliki, a dužina instrukcije još veća. Zbog

toga se povećava potreban kapacitet programske memorije, potrebna je šira programska magistrala ili je potrebno više memorijskih ciklusa za učitavanje instrukcije.

Jedno od rešenja ovog problema je *indirektno adresiranje pomoću registara* koje se univerzalno primenjuje kod svih integrisanih digitalnih procesora signala. Kod indirektnog adresiranja, adresa operanda se nalazi u jednom od *adresnih* ili *pomoćnih registara*. Savremeni procesori signala imaju 2 do 24 adresna registra, od kojih je aktivan onaj na koji pokazuje *pokazivač registra* (engl. auxiliary register pointer - ARP). Pošto je broj registara mali, za postavljanje pokazivača na željeni položaj potreban je mali broj bita (1-5). Pomoćni registri i ARP mogu se puniti iz memorije podataka ili neposredno iz same instrukcije. Pored toga, većina procesora omogućava inkrementiranje ili dekrementiranje sadržaja adresnih registara simultano sa izvršenjem neke instrukcije. Neki procesori poseduju i posebnu aritmetičku jedinicu za izračunavanje adrese, koja omogućava sabiranje ili oduzimanje sadržaja dva adresna registra u istom ciklusu sa izvršenjem tekuće instrukcije.

Indirektno adresiranje je veoma pogodno za realizaciju operacija filtriranja i DFT. Na primer, za realizaciju FIR filtra potrebno je upisati u jedan adresni registar adresu poslednjeg koeficijenta filtra i zatim izvršiti sledeću grupu operacija onoliko puta koliko ima koeficijenata (ćelija filtra):

1. Učitavanje dva operanda koristeći indirektno adresiranje,
2. Množenje operanada i akumuliranje proizvoda,
3. Dekrementiranje sadržaja adresnih registara,
4. Pomeranje operanda koji predstavlja signal radi formiranja linije za kašnjenje.

Kao primer posmatrajmo program za realizaciju FIR filtra na digitalnom procesoru signala TMS32010 koji je prikazan na slici 19.10.

```
LARK   AR0,adresa poslednjeg koeficijenta
LARK   AR1,adresa poslednjeg podatka
LARP   0         ; postavljanje ARP
LT     *-,AR1    ; punjenje T registra
MPY    *-,AR0    ; mnozenje sa signalom
LTD    *-,AR1    ; LT plus sabiranje proizvoda i pomeranje podataka
MPY    *-,AR0    ; mnozenje sa signalom
.............
LTD    *-,AR1    ; punjenje T registra i pomeranje podataka
MPY    *-,AR0    ; mnozenje sa signalom
APAC             ; sabiranje P registra sa akumulatorom
ADD    ONE,14    ; sabiranje LSB i pomeranje ulevo za 14 bita
SACH   RESULT,1  ; smestanje gornjih 16 bita akumulatora
```

Slika 19.10 Deo asemblerskog programa za implementaciju FIR filtra na procesoru TMS32010.

Procesor signala TMS32010 ima samo dva adresna registra AR0 i AR1 tako da ARP ima samo jedan bit. U prve dve instrukcije `LARK` pune se adresni registri adresama poslednjeg koeficijenta i poslednjeg podatka, dok treća instrukcija `LARP` postavlja ARP na nulu ukazujući tako na AR0 kao na aktivni adresni registar. Sledeća instrukcija `LT` puni prihvatni T registar množača koeficijentom, čija je adresa u tekućem adresnom registru AR0 (što pokazuje simbol "*"), dekrementira tekući adresni registar za jedan (što pokazuje simbol "-") i postavlja ARP na AR1. Naredna instrukcija `MPY` množi sadržaj T registra podatkom na koji ukazuje sadržaj tekućeg adresnog registra (AR1), smešta proizvod u P registar, dekremenitira tekući adresni registar AR1 za jedan i postavlja ARP an AR0. Instrukcija `LTD` se razlikuje od instrukcije `LT` po tome što osim

punjenja T registra vrši i sabiranje sadržaja P registra sa akumulatorom, kao i pomeranje podatka kopiranjem na sledeću višu lokaciju, čime se ostvaruje linija za kašnjenje. Dakle, za implementaciju FIR filtra na procesoru TMS32010, potrebne su *dve instrukcije za svaku ćeliju filtra*.

Instrukcijom `APAC` sabira se poslednji izračunati proizvod sa akumulatorom. Pretposlednja instrukcija `ADD` vrši sabiranje LSB sa akumulatorom i pomeranje za 14 bita ulevo, čime se ustvari vrši zaokruživanje i transfer rezultata u gornju polovinu akumulatora. Poslednja instrukcija `SACH` vrši smeštanje rezultata iz gornje polovine akumulatora u memoriju za podatke. Naravno, kompletan program je nešto duži, jer treba obezbediti i prihvatanje podataka iz A/D konvertora u memoriju za podatke, kao i eventualno izbacivanje podataka na D/A konvertor.

Vrlo slično indirektnom adresiranju je *indeksno adresiranje*. Osnovna razlika između ova dva načina adesiranja je što se kod indeksnog adresiranja inkrement (koji se nalazi u indeks registru) sabira sa sadržajem adresnog registra *pre* učitavanja operanda, a ne *posle* učitavanja. Najvažnija primena indeksnog adresiranja je za adresiranje elemenata vektora i matrica.

Kod digitalnih procesora signala se takođe često primenjuje i *direktno adresiranje*. Kod direktnog adresiranja izvestan broj bita (najčešće 7-16) iz instrukcije određuje adresu operanda. Kako se na taj način ne može obuhvatiti ceo adresni prostor, primenjuje se *adresiranje po stranicama* gde se *pokazivačem stranice* (engl. data page pointer - DP) pokazuje koja je stranica memorije aktivna. Bitovi koji se nalaze u DP registru predstavljaju bitove veće težine u adresi i spajaju se sa bitovima iz instrukcije prilikom formiranja adrese operanda. Druga varijanta direktnog adresiranja je *relativno adresiranje* kod koga se u instrukciji zadaje razlika između apsolutne adrese i trenutnog sadržaja programskog brojača. Ovaj način adresiranja je pogodan za realizaciju kontrolnih instrukcija.

Još jedan uobičajeni način adresiranja je *neposredno adresiranje*. Kod neposrednog adresiranja izvestan broj bita iz instrukcije se koristi kao operand, tako da nije potreban pristup memoriji za podatke. Primer instrukcija koje koriste neposredno adresiranje su instrukcije za punjenje akumulatora, adresnih registara, pokazivača registara, pokazivača stranice kao i instrukcije za sabiranje i množenje sa konstantom. *Registarsko adresiranje* je jedna od varijanti neposrednog adresiranja gde se operand nalazi u nekom od registara opšte namene. S obzirom da vrlo mali broj digitalnih procesora signala ima registre opšte namene, u kojima se mogu čuvati podaci, ovaj način adresiranje se vrlo retko sreće u praksi.

Neki procesori, naročito oni koji koriste 32-bitne podatke, imaju tzv. *kratke* i *duge* instrukcije za direktno i neposredno adresiranje. Kod kratkih instrukcija, adresa (ili operand) je smeštena u istoj reči sa instrukcijom, pa se takva instrukcija izvršava u jednom ciklusu. Duge instrukcije su dvorečne jer se adresa ili operand nalaze u sledećoj memorijskoj reči, pa zahtevaju dva instrukcijska ciklusa za izvršenje.

Pored opisanih načina formiranja adrese operanda, koji su preuzeti od mikroprocesora opšte namene, kod novijih digitalnih procesora signala sreću se još neki načini adresiranja koji su prilagođeni strukturama podataka koje se sreću u digitalnoj obradi signala. Jedna od varijanti indirektnog adresiranja je *ciklično adresiranje* ili *adresiranje po modulu*. Takav način adresiranja je pogodan za formiranje linije za kašnjenje koja se sreće kod realizacije FIR filtara. Kroz takvu liniju za kašnjenje stalno treba pomerati podatke, kako bi se napravilo mesto za prihvatanje narednog ulaznog odbirka. Iako kod većine procesora postoje instrukcije koje takav pomeraj obavljaju paralelno sa drugim operacijama, nepogodnost takvog pristupa je što se angažuje memorija za

podatke. Paralelizam rada funkcionalnih blokova može se bolje iskoristiti cikličnim adresiranjem. Kod cikličnog adresiranja se ulazni odbirak smešta u memoriju za podatke na mesto najstarijeg podatka, koji više nije potreban. Da bi se to ostvarilo, ako je početna adresa bloka podataka L, a poslednja adresa $L+M-1$, potrebno je obezbediti da se prilikom inkrementiranja adresnog registra sa adrese $L+M-1$ skoči na adresu L. Ako se adresni registar dekrementira potrebno je obezbediti skok sa L na $L+M-1$. To se može uraditi, ako se obezbedi formiranje adrese u aritmetici po modulu M. Većina procesora koji poseduju ovakav način adresiranja (na primer, DSP16 ili DSP56001), omogućava softversko zadavanje početne adrese i krajnje adrese (ili dužine) bloka podataka. Svođenje adrese na traženi opseg izvršava se hardverski. Da bi se uprostila konstrukcija aritmetičke jedinice za izračunavanje adrese, obično se zahteva da početna adresa mora biti stepen broja 2.

Kao primer primene cikličnog adresiranja, posmatrajmo procesor DSP56001, koji ima 8 adresnih registara za ciklično adresiranje. Svakom od registara Rx ($x=0-7$), pridružen je par registara Mx i Nx. U registru Mx se nalazi dužina bloka podataka (modul), dok se u registru Nx nalazi inkrement kojim se inkrementira registar Rx. Autoinkrementiranje i aritmetika po modulu se izvršavaju u dve adresne aritmetičke jedinice, tako da se simultano mogu izračunati dve adrese za obe memorijske banke. Deo programa za realizaciju FIR filtra prikazan je na slici 19.11.

```
start move    adresa pocetka bloka podataka,r0
      move    adresa prvog koeficijenta,r4
      move    red-FIR-filtra - 1,m0
      move    m0,m4
fir   movep             x:input,(r0)
      clr     a         x:(r0)-,x0  y(r4)+,y0
      rep     m0
      mac     x0,y0,a   x:(r0)-,x0  y(r4)+,y0
      macr    x0,y0,a   (r0)+
      movep             a,x:output
      jmp     fir
```

Slika 19.11 Program za implementaciju FIR filtra na procesoru DSP56001.

Odbirci signala smešteni su u X memoriji, a koeficijenti u Y memoriji, pa se za pristup obema memorijama koristi ciklično adresiranje. Zbog toga su u trećoj i četvrtoj instrukciji u registre M0 i M4 smeštene dužine blokova podataka koeficijenata i signala. Prva instrukcija u petlji sa labelom `fir` učitava ulazni odbirak, dok druga instrukcija `clr` inicijalizuje akumulator i učitava prvi par operanada. Sledeća instrukcija `rep` označava da se naredna instrukcija `mac` (množenje i akumuliranje proizvoda) izvršava `(red-FIR-filtra - 1)` puta. Instrukcija `macr` izvršava množenje i akumuliranje proizvoda sa zaokružavanjem, dok poslednja instrukcija `movep` izbacuje rezultat na izlaz.

Drugi način cikličnog adresiranja sreće se kod digitalnog procesora signala TMS320C30. Dužina bloka podataka određena je sadržajem kontrolnog registra BK. Adresa operanda određuje se sadržajem ma kog adresnog registra ARx ($x=0-7$), koristeći notaciju `*ARx++(IRy)%`, što znači da se sadržaj adresnog registra ARx modifikuje posle učitavanja operanda sabiranjem indeks registra IRy sa adresnim registrom, tako da rezultujuća adresa leži unutar zadatih granica (što je označeno sa %). Umesto indeks registra može se koristiti bilo koja konstanta iz opsega 0-255, a znak ++ za sabiranje može biti zamenjen znakom -- za oduzimanje. Naravno, početni sadržaj adresnog registra mora ležati u zadatom opsegu. Kao primer, posmatrajmo program za realizaciju FIR filtra na procesoru TMS320C30, koji je prikazan na slici 19.12.

```
   LDI     red filtra + 1,BK
   LDI     adresa poslednjeg koeficijenta,AR0
   LDI     kraj linije za kasnjenje,AR1
   LDF     0,R0
   LDF     0,R2
   RPTS    red filtra
   MPYF3   *AR0++(1)%,*AR1++(1)%,R0
|| ADDF3   R0,R2,R2
   ADDF    R0,R2
```

Slika 19.12 Deo programa za implementaciju FIR filtra na procesoru TMS320C30.

Prvih pet instrukcija pune potrebne registre BK, AR0, AR1, R0 i R2 potrebnim celobrojnim podacima i podacima sa pokretnom tačkom, koristeći instrukcije sa neposrednim adresiranjem. Šesta instrukcija označava da se naredna instrukcija izvršava `(red filtra + 1)` puta. Sedma instrukcija obezbeđuje paralelno izvršenje množenja, sabiranja i modifikaciju adresnih registara AR0 i AR1. Treba primetiti da se množenje i sabiranje izvršavaju simultano, tj. da se proizvod operanada smešta u R0 i akumulira u R2 prilikom sledećeg izvršenja iste instrukcije. Prilikom prvog sabiranja sabiraju se dve nule, a poslednje sabiranje ostvaruje se poslednjom instrukcijom `ADDF`. Kompletan program sadrži i ulazno-izlazne instrukcije kao i spoljnu petlju koji u ovom primeru nisu prikazani.

Kod najnovijih digitalnih procesora signala sreće se još jedan način adresiranja, koji ne postoji u mikroprocesorskoj tehnici. To je *bit-inverzno adresiranje*. Kao što je poznato, kod brzih algoritama za izračunavanje DFT potrebno je izvršiti preuređivanje ulazne ili izlazne sekvence podataka. Takvo preuređivanje zahteva određeno vreme, a često nije ni moguće kada se radi u realnom vremenu. Da bi se izbeglo premeštanje podataka, podaci se slažu u prirodnom redosledu, ali se njima pristupa po bit-inverznom redosledu indeksa (adresa) što se ostvaruje nešto složenijom konstrukcijom adresne aritmetičke jedinice. Naime, u aritmetičkoj jedinici za izračunavanje adresa potrebno je obezbediti da se eventualni prenos pri sabiranju cifara iz najvišeg razreda upiše na LSB poziciju pri čemu se preostali bitovi pomere za jedno mesto ulevo. Zatim se, ako treba izračunati DFT sekvence od 2^p elemenata, pri adresiranju elemenata sekvence vrši postinkrementiranje sa korakom od 2^{p-1}, pri čemu se sa prenosom postupa na opisani način. Lako se pokazuje da su tada adrese elemenata generisane u bit-inverznom redosledu.

19.1.4 PROTOČNOST INSTRUKCIJA

U prethodnom poglavlju ukazano je na značaj protočne hardverske strukture u realizaciji složenih procesa. Isti princip podele posla na manje funkcionalne celine može se primeniti i na izvršavanje instrukcija iz programa za digitalni procesor signala. Naime, u objašnjavanju prednosti Harvard arhitekture, već je rečeno da je njena prednost nad klasičnom mikroprocesorskom arhitekturom u tome što se instrukcija i operand mogu simultano učitavati. Međutim, pri tome se *operand koji se učitava ne odnosi na učitanu instrukciju*, jer ona tek treba da se dekoduje, već na *prethodnu* instrukciju koja je već dekodovana. Dakle, u najprostijem slučaju, faze učitavanja instrukcije, učitavanja operanda i izvršavanja instrukcije se mogu simultano izvršavati ako se ne odnose na istu instrukciju jer angažuju različite funkcionalne blokove digitalnog procesora signala. Takav pristup izvršavanja instrukcija naziva se *protočni princip* (engl. pipelining) i sreće se sa manjim varijacijama kod svih digitalnih procesora signala. Kada se instrukcije izvršavaju koristeći princip protočnosti, na izlazu se dobija rezultat posle svakog instrukcijskog ciklusa, bez obzira na

to što stvarno izvršavanje instrukcije traje više ciklusa. Protočnost, dakle, može znatno ubrzati izračunavanje, ali ima vrlo ozbiljne posledice na proces programiranja koji postaje složeniji.

Važan pojam u vezi sa protočnim izvršavanjem instrukcija je *dubina protoka*. Pod dubinom protoka podrazumeva se broj različitih instrukcija koje se simultano procesiraju u različitim fazama. Kod savremenih digitalnih procesora signala dubina protoka se nalazi u granicama od dva do pet.

Kod digitalnih procesora signala sreću se tri različita metoda realizacije principa protočnosti, koji se uglavnom razlikuju po načinu na koji rešavaju konflikte koji mogu nastati usled simultanog izvršavanja delova više instrukcija. To su *blokiranje*, *vremenski stacionarno kodovanje* i *kodovanje sa stacionarnim podacima*. Granice između ovih kategorija nisu stroge i u mnogim digitalnim procesorima signala sreće se mešavina ovih principa.

19.1.4.1 Blokiranje

Jedan pristup paralelnom izvršavanju više operacija je da *programer ne treba da vodi računa o internom vremenskom odnosu operacija i paralelizmu u radu funkcionalnih blokova*. Takav pristup podrazumeva da programer piše asemblerski program na takav način kao da se svaka akcija koju inicira neka instrukcija završava pre nego što se započne izvršenje sledeće instrukcije. Dakle, procesor može da bude protočan ali to ne sme da se primećuje.

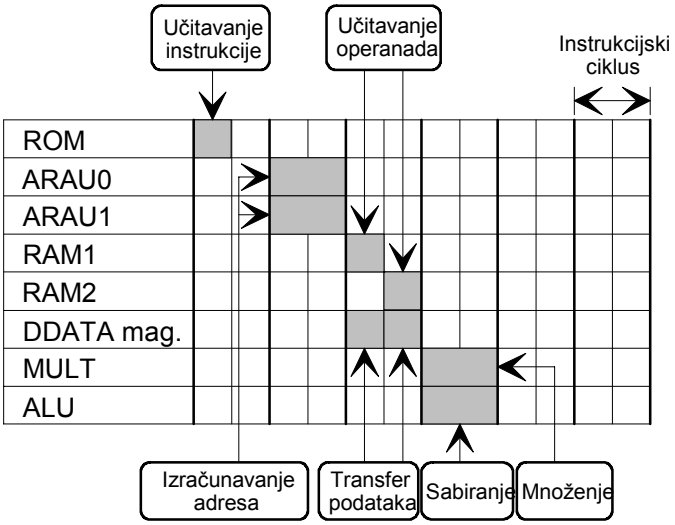
Najprostiji model protočnosti programabilnog procesora deli svaku instrukciju na nekoliko faza. Na primer, faze jedne instrukcije mogu biti: učitavanje instrukcije, dekodovanje instrukcije, učitavanje operanda (ili više operanada) i izvršenje instrukcije. Naravno, zavisno od arhitekture raspoloživog hardvera, moguća je i neka druga podela. Ako postoji mogućnost da se rezultat svake faze zapamti i sačuva do sledeće faze, moguće je tako organizovati izvršavanje instrukcija da se preklapaju različite faze susednih instrukcija u programu. Osnovna ideja protočnog izvršavanja instrukcija je da se izvrši preklapanje izvršenja instrukcija na opisani način, a da se još uvek ima utisak da se svaka instrukcija završava pre nego što se započne procesiranje sledeće instrukcije.

Princip blokiranja najviše koriste digitalni procesori signala firme Texas Instruments koji pripadaju familiji TMS320. Interni vremenski dijagrami za izvršenje instrukcija kod procesora iz ove familije su vrlo složeni, ali se pažljivim proučavanjem vremenskih dijagrama mogu izvući izvesni zaključci [L-5], koji će biti analizirani radi objašnjenja principa blokiranja.

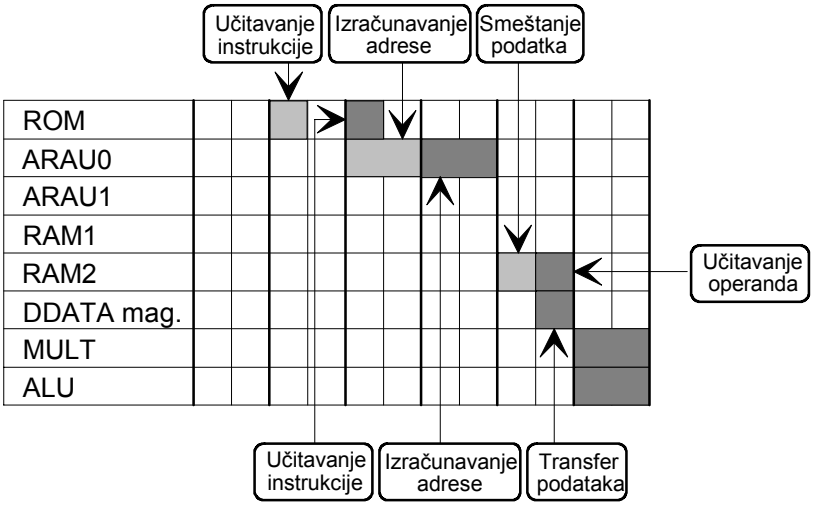
Posmatrajmo prvo kako se simultano izvršavaju operacije množenja i sabiranja kod procesora TMS320C30 kada se koristi samo interna memorija. Na vremenskom dijagramu koji je prikazan na slici 19.13, u prvoj koloni su prikazani hardverski resursi koji učestvuju u izvršavanju instrukcije `MPYF||ADDF` (prikazani na slikama 19.5 i 19.9), a u narednim kolonama su šrafiranim poljima prikazani vremenski intervali kada su pojedini hardverski resursi angažovani. Svaki instrukcijski ciklus (60 ns pri taktu od 33.33 MHz) je podeljen na dva dela. Tokom jednog instrukcijskog ciklusa se može pristupiti svakoj internoj memoriji dva puta, dok pristup eksternoj memoriji može trajati ceo instrukcijski ciklus pa i duže. Da bi se simultano pristupilo do dve memorije, procesor poseduje dve posebne aritmetičke jedinice za izračunavanje adresa ARAU0 i ARAU1. Ove dve jedinice posebno dolaze do izražaja prilikom indeksnog adresiranja kada treba sabrati indeks i adresu pre adresiranja operanada. Operandi se mogu nalaziti u različitim memorijama, što je prikazano na slici 19.13, ali i u istoj memoriji. Za transfer operanada koristi se magistrala podataka DDATA, a za adresiranje, adresne magistrale DADDR1 i DADDR2. Kao što

se vidi sa slike, stvarno vreme koje je potrebno za simultano množenje i sabiranje iznosi četiri kompletna instrukcijska ciklusa. Međutim, takođe se vidi da se već u narednom ciklusu može započeti procesiranje iste (ili slične) instrukcije, a da ne dođe do konflikta oko korišćenja hardverskih resursa. Zbog toga programer ne mora biti svestan da postoji protočnost instrukcija.

Posle izvršenja instrukcije za množenje i sabiranje može uslediti i instrukcija za smeštaj rezultata množenja ili sabiranja (ili oba rezultata), `STF`, u memoriju bez ikakvog kašnjenja. Sa vremenskog dijagrama za operacije množenja i sabiranja se vidi da su pomenuti rezultati najranije raspoloživi u toku petog instrukcijskog ciklusa na slici 19.13. Vremenski dijagram izvršenja instrukcije za smeštaj podataka u memoriju prikazan je na slici 19.14.



Slika 19.13 Vremenski dijagram izvršenja instrukcije množenja i sabiranja, MPYF||ADDF, kod procesora TMS320C30.

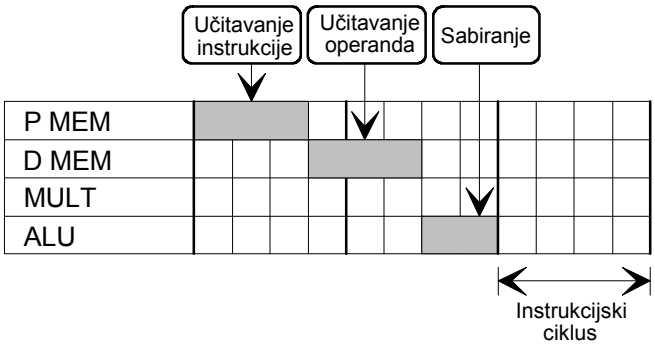


Slika 19.14 Vremenski dijagram izvršenja instrukcije za smeštaj rezultata u memoriju za podatke, STF, kod procesora TMS320C30.

Vremenski dijagram instrukcije `STF` počinje jedan ciklus iza poslednje instrukcije množenja i traje takođe četiri instrukcijska ciklusa, ali se instrukcija `STF` može savršeno preklopiti sa prethodnom, ili narednom instrukcijom, čije je izvršenje na vremenskom dijagramu prikazano ukrštenom šrafurom. Da bi se sakrilo da se instrukcije izvršavaju protočno, sledeća instrukcija mora biti sposobna da koristi tek upisani podatak, što zaista i može.

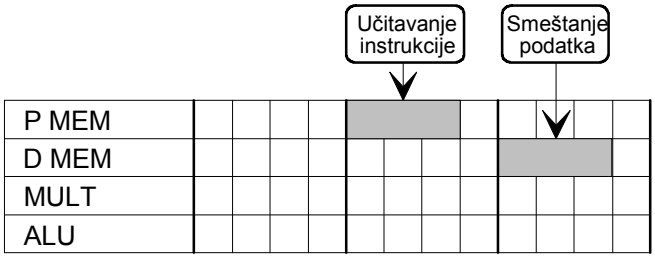
Međutim, u realizaciji principa protočnog izvršenja instrukcija mogu se pojaviti i neki problemi. Pretpostavimo da posle instrukcije STF dolazi instrukcija aritmetičkog tipa sa dva operanda iz memorije RAM2. Ako STF instrukcija smešta podatak u RAM1, neće se pojaviti nikakav problem. Međutim, ako instrukcija STF smešta podatak u RAM2, kao na slici, u petom ciklusu od početka posmatranja pojavljuju se tri zahteva za pristup memoriji RAM2. U tom slučaju kontrolni hardver detektuje konflikt i odlaže izvršenje aritmetičke instrukcije za jedan ciklus. Ovo odlaganje se naziva *blokiranje*, po čemu je ovaj metod realizacije principa protočnosti dobio naziv. Programer ne mora biti svestan da postoji ovo kašnjenje i može ga otkriti samo uz pomoć simulatora.

Kod starijih procesora iz familije TMS320, kao što su, na primer, procesori prve i druge generacije TMS32010/20/C25, aritmetičke instrukcije se takođe izvršavaju primenom protočnog principa, ali je dubina protoka tri. Na primer, vremenski dijagram izvršenja instrukcije ADD, koja vrši sabiranje jednog operanda sa akumulatorom, prikazan je na slici 19.15, sa koga se vidi da se u prvoj fazi učitava (i dekoduje) instrukcija, da se u drugoj fazi vrši učitavanje operanda, a da je poslednja faza izvršna. Svaki instrukcijski ciklus (100 do 200 ns, zavisno od procesora) podeljen je interno na četiri dela. Faze instrukcije u kojima se pristupa memoriji, tj. faze učitavanja instrukcije ili operanda traju tri podciklusa, dok izvršna faza traje dva podciklusa. Ukupno vremensko trajanje instrukcije ADD traje dva instrukcijska ciklusa, mada je sa dijagrama sasvim jasno da se niz ADD (ili sličnih) instrukcija može izvršavati brzinom od jedne instrukcije po instrukcijskom ciklusu bez ikakvog konflikta oko korišćenja resursa. Takođe je interesantno da jedna ADD instrukcija može bez konflikta koristiti rezultat prethodne instrukcije koji je smešten u akumulatoru.



Slika 19.15 Vremenski dijagram izvršenja instrukcije sabiranja sa akumulatorom, ADD, kod procesora TMS320C25.

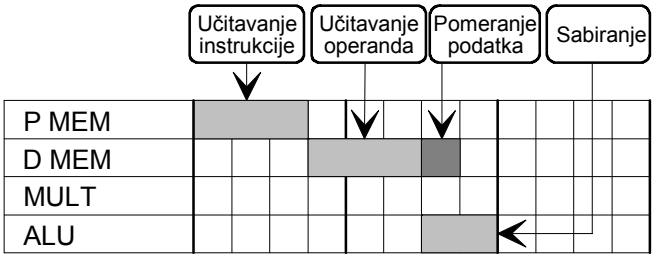
Ukoliko posle sabiranja sa akumulatorom sledi instrukcija za smeštaj sadržaja akumulatora u memoriju SACH, vremenski dijagram izgleda kao na slici 19.16. Za izvršenje ove instrukcije je neophodno da stanje akumulatora bude korektno na kraju drugog ciklusa posle početka izvršavanja instrukcije ADD, što je upravo slučaj. Dakle, instrukcija SACH može da sledi za instrukcijom ADD, ali važi i obrnuto. U oba slučaja instrukcije se izvršavaju u protoku bez konflikta.



Slika 19.16 Vremenski dijagram izvršenja instrukcije za smeštaj sadržaja akumulatora u memoriju za podatke, SACH, kod procesora TMS320C25.

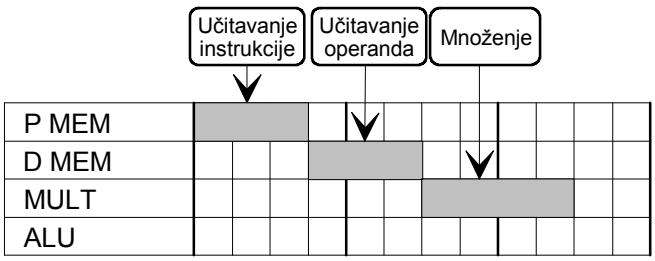
Iz prethodna četiri primera mogu se izvući važni zaključci koji su opšteg karaktera. Prvo, trajanje izvršenja instrukcije ne mora da bude ograničeno na jedan instrukcijski ciklus, a da pri tome *izgleda* da se instrukcija izvršava u jednom instrukcijskom ciklusu. Drugo, vremenski dijagrami izvršavanja instrukcija, koji su retko publikovani u podacima proizvođača, mogu se rekonstruisati na osnovu zahteva i ograničenja koji se postavljaju za određene sekvence instrukcija kao i za karakteristike eksternih memorija.

Kao primer izložene teorije, posmatrajmo program za implementaciju FIR filtarske funkcije na procesoru TMS32010 koji je ranije prikazan na slici 19.10. Kao što se vidi, suštinski deo programa čini ponovljeno izvršavanje samo dve instrukcije `LTD` (`LT`) i `MPY`. Vremenski dijagram `LTD` instrukcije, koja vrši punjenje T registra, kopiranje sadržaja T registra na lokaciju sa višom adresom i sabiranje P registra sa akumulatorom, prikazan je na slici 19.17. Kao što se vidi, stvarno trajanje instrukcije je dva instrukcijska ciklusa, ali je moguće protočno izvršavanje, čime se efektivno trajanje instrukcije svodi na jedan instrukcijski ciklus.



Slika 19.17 Vremenski dijagram izvršenja instrukcije LTD kod procesora TMS32010/20/C25.

Druga važna instrukcija kod implementacije FIR filtra je `MPY`, koja vrši množenje sadržaja T registra sa specificiranim operandom i smeštanje rezultata množenja u P registar. Vremenski dijagram izvršavanja instrukcije `MPY` prikazan je na slici 19.18. Ukupno trajanje instrukcije `MPY` je 2.5 instrukcijskih ciklusa.



Slika 19.18 Vremenski dijagram izvršenja instrukcije MPY kod procesora TMS32010/20/C25.

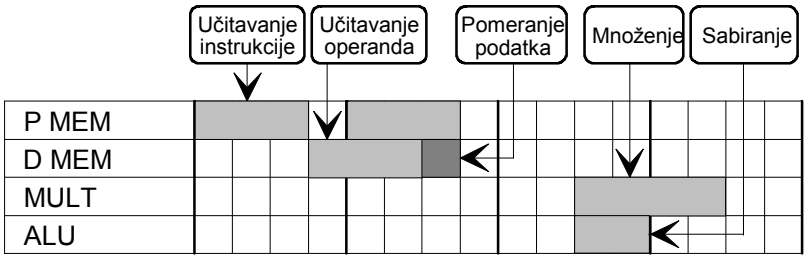
Analizom dijagrama sa slika 19.17 i 19.18 se vidi da je zahvaljujući kasnom početku faze sabiranja na slici 19.17, rezultat množenja iz prethodne instrukcije `MPY` raspoloživ na vreme. Dakle, protočno alternativno izvršavanje instrukcija `LTD` i `MPY` je moguće bez konflikta oko korišćenja potrebnih resursa. Za par instrukcija `LTD` i `MPY` u protočnom izvršavanju potrebna su samo dva instrukcijska ciklusa, tako da se pri realizaciji FIR filtra koriste samo dva instrukcijska ciklusa po ćeliji.

Mnogo kompaktniji (i brži) način implementacije FIR filtra moguć je kod novijih procesora iz familije TMS320, TMS32020/C25, koji poseduju programski keš za jednu instrukciju, kao i instrukciju `RPTK` za višestruko izvršenje sledeće instrukcije. Koristeći instrukcije `RPTK` i `MACD` (koja zamenjuje instrukcije `LTD` i `MPY`), program za implementaciju FIR filtra sa slike se modifikuje i dobija oblik prikazan na slici 19.19.

```
LARK AR0,adresa poslednjeg koeficijenta
LARK AR1,adresa poslednjeg podatka
LARP 0          ; postavljanje ARP
LT   *-,AR1     ; punjenje T registra
RPTK red-FIR-filtra
MACD m1,m2      ; mnozenje sa signalom
APAC            ; sabiranje P registra sa akumulatorom
ADD  ONE,14     ; sabiranje LSB i pomeranje ulevo za 14 bita
SACH RESULT,1   ; smestanje gornjih 16 bita akumulatora
```

Slika 19.19 Deo asemblerskog programa za implementaciju FIR filtra na procesoru TMS32020/C25.

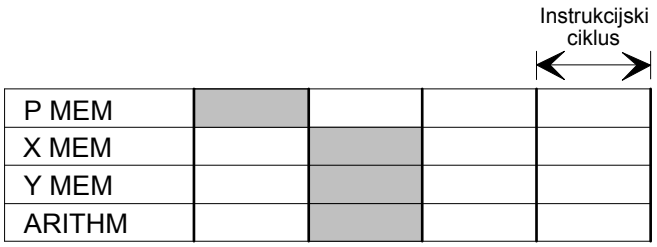
Vremenski dijagram instrukcije `MACD` prikazan je na slici 19.20. Uočava se da je ukupno trajanje izvršenja instrukcije 3.5 instrukcijskih ciklusa. Faze množenja i sabiranja bi, u principu mogle da započnu tri podciklusa ranije, ali je, verovatno zbog jednostavnije realizacije kontrolnog hardvera, u procesoru implementirano prikazano rešenje. U protočnom izvršavanju, instrukcija `MACD` se ponaša kao dvociklusna. Međutim, kada se instrukcija `MACD` izvršava iz keša za instrukcije, faza učitavanja instrukcije nije potrebna i trajanje instrukcije se skraćuje za jedan ciklus, pa se u protočnom izvršavanju ponaša kao jednociklusna. Dakle, kod procesora TMS320/C25 potreban je samo *jedan instrukcijski ciklus po ćeliji FIR filtra*. Slična je situacija i kod novijih digitalnih procesora signala drugih proizvođača.



Slika 19.20 Vremenski dijagram izvršenja instrukcije MACD kod procesora TMS32020/C25.

19.1.4.2 Vremenski stacionarno kodovanje

Mada je veoma pogodna sa programerske tačke gledišta, realizacija protočnosti pomoću blokiranja nije optimalna. Bolje performanse se mogu dobiti ako se programeru dozvoli eksplicitna kontrola svih faza protoka. Takav način realizacije protočnosti naziva se *vremenski stacionarno kodovanje* (engl. time stationary coding). Kod vremenski stacionarnog kodovanja u instrukciji se *eksplicitno specificiraju sve operacije koje treba da se simultano izvrše tokom jednog instrukcijskog ciklusa*. Na primer, na slici 19.21 je prikazan vremenski dijagram izvršavanja jedne aritmetičke operacije na procesoru DSP56001. Jedna takva instrukcija treba eksplicitno da specificira najmanje tri paralelne operacije, tj. dva učitavanja iz memorija i najmanje jednu aritmetičku operaciju. Dakle, *svaka instrukcija specificira simultane a ne sekvencijalne operacije* učitavanja operanda i izvršenja instrukcije. U suštini, programerski model vremenski stacionarnog kodovanja više predstavlja paralelizam nego protočnost.



Slika 19.21 Vremenski dijagram izvršavanja jedne aritmetičke operacije na procesoru DSP56001.

Tipičan primer aritmetičke operacije predstavlja instrukcija za množenje i sabiranje sa akumulatorom koja kod procesora DSP56001 i DSP96002 ima oblik:

```
MAC X0,Y0,A   X:(R0)+,X0   Y:(R4)-,Y0
```

U prikazanoj instrukciji razlikuju se tri polja. U prvom polju se specificira aritmetička instrukcija ili instrukcije, dok sledeća dva polja specificiraju koje operande treba učitati *za sledeću instrukciju*. Operandi za specificiranu aritmetičku operaciju predstavljaju sadržaje registara X0 i Y0, koji su učitani iz memorije tokom izvršenja *prethodne* instrukcije. Rezultat množenja se sabira sa akumulatorskim registrom A. Kao što je već rečeno, procesor DSP56001 ima integrisanu aritmetičku jedinicu kod koje se množenje i sabiranje obavljaju simultano u istom hardveru, a ne sukcesivno. Dakle, instrukcija `MAC` množi sadržaj X0 i Y0 i simultano sabira rezultat sa A, tako da je u sledećem instrukcijskom ciklusu raspoloživ nov sadržaj akumulatora. Ovo je ostvareno po cenu nešto komplikovanije realizacije množača, koji pored množenja može i da sabere rezultat sa ulaznim podatkom. Nažalost, kod aritmetike sa pokretnom tačkom, simultana realizacija množenja i sabiranja je znatno teža, tako da se kod procesora DSP96002, koji je vrlo sličan po arhitekturi, ne primenjuje. Kod ovog procesora množenje i sabiranje se specificiraju posebnim instrukcijama, kao što je na primer:

```
FMPY D4,D5,D0   FADD D0,D1   X:(R0)+,D4   Y:(R4)+,D5
```

Ova instrukcija specificira množenje sadržaja registara D4 i D5 i smeštanje rezultata u registar D0. U isto vreme, *prethodni sadržaj* registra D0 (ne rezultat operacije `FMPY`) se sabira sa registrom D1. Pored toga specificirano je i učitavanje novih operanada u registre D4 i D5 za sledeću operaciju. Dakle, programer eksplicitno oblikuje protok specificirajući aktivnosti u svakoj fazi protoka.

Interesantno je uporediti ovu instrukciju sa ekvivalentnom instrukcijom za paralelno množenje i sabiranje `MPYF3||ADDF3` kod procesora TMS320C30, koja je korišćena u programu sa slike 19.12. Kod procesora TMS320C30 oba operanda su kompletno specificirana u instrukciji *koja ih koristi*. Nasuprot tome, kod procesora DSP96002 operandi se specificiraju u instrukciji *koja prethodi* aritmetičkoj instrukciji koja ih koristi.

Vremenski stacionarno kodovanje ima niz prednosti nad blokiranjem. Pre svega, program je jasniji. Drugo, kod procesora koji koriste blokiranje je vrlo teško odrediti koliko ciklusa će trajati koja instrukcija jer to zavisi i od susednih instrukcija. Prema tome, tačna procena trajanja izvršenja nekog programa je nemoguća ako se ne koristi simulator. Treće, pošto programer ima eksplicitnu kontrolu nad protokom, na efikasan način se može rešiti problem prekida (interapta). Zbog toga procesori DSP56001 i DSP96002 imaju veoma brze prekide koji troše samo dva instrukcijska ciklusa.

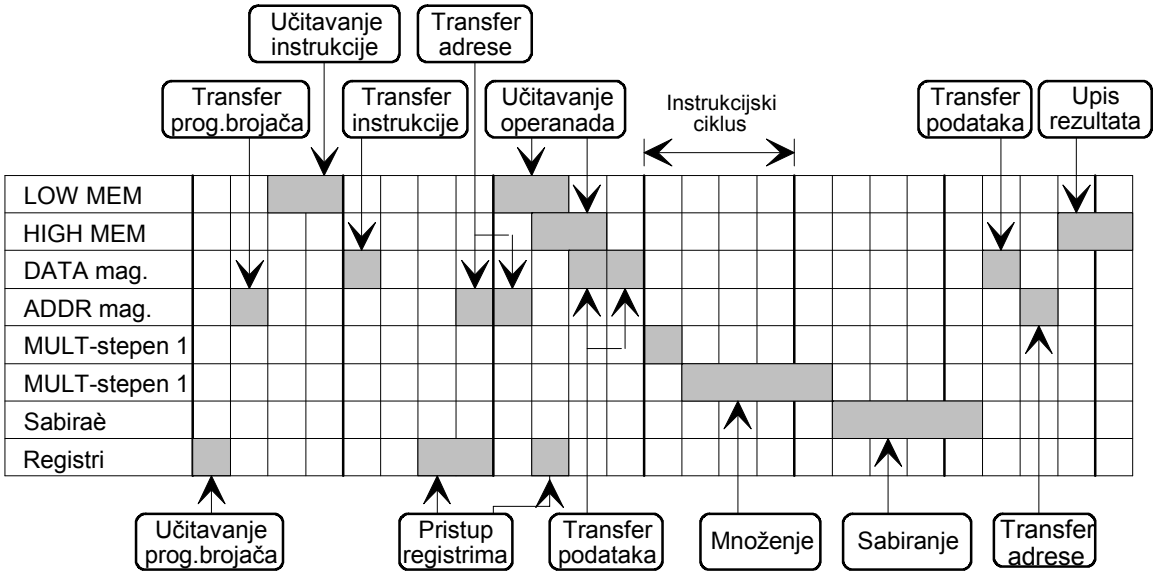
19.1.4.3 Kodovanje sa stacionarnim podacima

Kodovanje sa stacionarnim podacima je mnogo bliže načinu na koji čovek shvata algoritam. Kod pisanja programa po ovom principu, u jednoj instrukciji se specificiraju sve operacije koje treba uraditi sa operandima iz memorije. Dakle, *instrukcija specificira šta se događa sa podacima*, a ne šta se dešava u određenom vremenskom trenutku. Kao posledica ovakvog načina programiranja, rezultat jedne instrukcije ne mora biti odmah raspoloživ za naredne instrukcije.

Ovaj princip implementacije protočnog izvršavanja instrukcija najviše je zastupljen kod procesora DSP32 i DSP32C. Primer tipične aritmetičke instrukcije kod ova dva procesora je:

```
r5++=a1=a0+'r7''r10++r17
```

gde adresni registri `r7` i `r10` specificiraju dva operanda za množač. Registar `r17` specificira post-inkrementiranje registra `r10`. Izračunati proizvod se sabira sa `a0`, a rezultat smešta u `a1` i memorijsku lokaciju specificiranu sa `r5`. Mada je ovakva instrukcija prilično laka za razumevanje, jasno je da se sve ove operacije ne mogu obaviti u toku jednog instrukcijskog ciklusa. Vremenski dijagram koji opisuje izvršavanje ove instrukcije traje 6 instrukcijskih ciklusa i prikazan je na slici 19.22.



Slika 19.22 Vremenski dijagram izvršenja instrukcije za množenje, sabiranje i smeštaj rezultata kod procesora DSP32 i DSP32C.

Međutim, prilikom protočnog izvršavanja instrukcija, u svakom instrukcijskom ciklusu se započinje izvršavanje jedne ovakve instrukcije. Dakle, i sa ovakvom implementacijom principa protočnosti, za implementaciju FIR filtra je potreban jedan instrukcijski ciklus po ćeliji. Pri tome je iskorišćenje hardverskih resursa blizu 100% ako je broj ćelija dovoljno veliki.

Pretpostavimo sada da u sledećoj instrukciji treba pročitati podatak iz memorije i koristiti ga kao operand. Učitavanje takve instrukcije moglo bi da se najranije obavi u četvrtom ciklusu posle instrukcije za množenje i sabiranje. Drugim rečima, tri naredne instrukcije posle instrukcije množenja i sabiranja ne smeju čitati njen rezultat iz memorije, jer on neće biti upisan u memoriju u vreme kada one čitaju svoje operande. Dakle, opisani princip implementacije principa protočnosti ima i sledeća ograničenja:

1. Kada se akumulator a*n* koristi kao operand množača, vrednost akumulatora mora biti poznata tri instrukcije ranije,

2. Kada se rezultat upisuje u memoriju, nova vrednost memorijske lokacije ne sme se koristiti naredna četiri instrukcijska ciklusa.

I pored ovih ograničenja pogrešno je zaključiti da je kodovanje sa stacionarnim podacima manje efikasno od vremenski stacionarnog kodovanja. Naime, kod vremenski stacionarnog kodovanja za specificiranje množenja, sabiranja i smeštanja podataka potrebno je nekoliko instrukcija. Ako je hardverska organizacija oba procesora slična, ukupno potrebno vreme za izvršenje ovih instrukcija je isto kod kodovanja sa stacionarnim podacima kao i kod vremenski

stacionarnog kodovanja. Kod kodovanja sa stacionarnim podacima, paralelizam se ostvaruje istovremenim izvršavanjem susednih instrukcija.

U poređenju sa vremenski stacionarnim kodovanjem, jedini nedostatak kodovanja sa stacionarnim podacima je u tome što je znatno teže ostvariti brzu reakciju na zahtev za prekid. Zbog toga je odziv na signal za prekid kod procesora DSP32C čak tri instrukcijska ciklusa, a ostvaren je tako što je napravljen sekundarni skup protočnih registara u kome se pamti stanje protoka kada se javi zahtev za prekid. Na ovaj način se može upamtiti oko 400 bita iz prekinutog protoka. Naravno, na ovaj način se ne može ostvariti višestruki prekid, jer bi to zahtevalo još jedan skup protočnih registara.

19.1.4.4 Grananje

Iz dosadašnjeg izlaganja o implementaciji principa protočnog izvršavanja instrukcija može se zaključiti da se najveći problemi mogu očekivati prilikom izvršavanja instrukcija za grananje. Izvršavanje instrukcija za grananje otežano je zbog postojanja nekoliko problema. Prvo, vremenski interval između učitavanja dve instrukcije je najčešće prekratak da bi se dekodovanjem otkrilo da je učitana instrukcija za grananje pre nego što se učita sledeća instrukcija. Drugo, ako je adresni prostor veliki, adresa ne može biti smeštena u instrukciju tako da se adresa nalazi u sledećoj reči iz programske memorije. U ovom slučaju postoje rešenja koja koriste stranično ili relativno adresiranje. Treće, u slučaju uslovnog grananja, učitavanje sledeće instrukcije ne sme da se izvrši pre nego što se izvrši testiranje stanja ALU jedinice.

Kod digitalnih procesora signala različitih proizvođača sreću se različita rešenja za izvršenje grananja. Na primer, kod prve i druge generacije procesora signala iz familije TMS320, tj. kod procesora TMS32010/20/C25, radi prikrivanja protočnosti, instrukcije grananja zahtevaju više instrukcijskih ciklusa za izvršenje. Tačan broj ciklusa zavisi od konfiguracije sistema. U slučaju bezuslovnog grananja, potreban je dodatni instrukcijski ciklus da bi se iz programske memorije učitala adresa na koju se skače. Ako je u pitanju uslovno grananje, testiranje kontrolnih bitova koji označavaju stanje ALU može se izvršiti istovremeno sa učitavanjem adrese skoka.

Kod procesora DSP16 za izvršenje bezuslovnih instrukcija grananja potrebna su dva instrukcijska ciklusa, a za uslovna grananja tri ciklusa.

Interesantno rešenje je primenjeno kod procesora DSP32 i DSP32C. Kada se izvršava neka od instrukcija za skokove, (`if`, `call`, `return`, `goto`), instrukcije koje slede za njom se izvršavaju pre nego što se izvrši grananje. Ovaj princip se naziva *grananje sa kašnjenjem*. Ako se uslovno grananje vrši na osnovu ispitivanja rezultata neke aritmetičke operacije, uslov za testiranje određuje poslednja aritmetička operacija koja se nalazi najmanje *četiri instrukcije pre testa*.

Kod procesora TMS320C30 za grananje se mogu koristiti standardne višeciklusne instrukcije grananja, ali i instrukcije za grananje sa kašnjenjem. Višeciklusne instrukcije grananja prekidaju protok i za njihovo izvršenje su potrebna četiri instrukcijska ciklusa. Standardno bezuslovno grananje je ilustrovano sledećim primerom:

```
        BR   labela
        INS1          ; Prva instrukcija koja se preskace
        INS2          ; Druga instrukcija koja se preskace
        INS3          ; Treca instrukcija koja se preskace
        ..........
labela  Instrukcija koja se ucitava tri ciklusa posle BR
        ..........
```

Drugi tip instrukcija grananja, tj instrukcije za grananje sa kašnjenjem, ne prekidaju protok i garantuju da će biti učitane i izvršene sledeće tri instrukcije pre nego što se grananjem promeni sadržaj programskog brojača. Za njihovo izvršenje je tada potreban samo jedan instrukcijski ciklus. Ukoliko je nemoguće preurediti program tako da se posle instrukcije grananja umetnu tri instrukcije, može se umetnuti i jedna do tri `NOP` instrukcije. Standardno bezuslovno grananje sa kašnjenjem je ilustrovano sledećim primerom:

```
        BRD     labela
        INS1            ; Prva instrukcija koja se izvrsava
        INS2            ; Druga instrukcija koja se izvrsava
        INS3            ; Treca instrukcija koja se izvrsava
        INS4            ; Instrukcija koja se ne izvrsava
        ..........
labela  Instrukcija koja se ucitava posle ucitavanja INS3
        ..........
```

Kao što se vidi, nijedno od rešenja za uslovno i bezuslovno grananje nije dovoljno efikasno i brzo. Zbog toga se instrukcije za grananje retko koriste u programima za digitalnu obradu signala. Na sreću, takve instrukcije su retko potrebne u programiranju klasičnih algoritama digitalne obrade signala. Jedini izuzetak predstavljaju petlje, kada je pogodnije koristiti instrukcije koje su specijalno namenjene za ponavljanje instrukcija ili instrukcije za realizaciju petlji sa malim režijskim vremenom.

Kod nekih procesora sreću se i *uslovne instrukcije*. To su instrukcije za izvršavanje neke operacije (upisa, čitanja, itd.) pod određenim uslovima. Takve instrukcije su vremenski efikasnije od kombinacije instrukcije za uslovno grananje i izvršne instrukcije.

19.1.5 ALATI ZA RAZVOJ SOFTVERA I HARDVERA

Savremeni digitalni procesori signala namenjeni su za realizaciju složenih algoritama. Sa porastom složenosti algoritma, koji treba realizovati na određenom procesoru signala, rastu i teškoće oko testiranja algoritma u realnim uslovima rada. Zbog toga proizvođači integrisanih digitalnih procesora signala svojim korisnicima nude obimnu podršku za razvoj softvera i hardvera.

Za svaki integrisani procesor signala proizvođač razvija *asembler* koji prevodi izvorni asemblerski program u objektni kod u mašinskom jeziku. Izvorni program sadrži instrukcije, asemblerske direktive i makro direktive. Asemblerske direktive usmeravaju proces prevođenja izvornog programa, dok makro direktive omogućavaju koncizniju reprezentaciju grupa instrukcija koje se često ponavljaju.

Kod većine digitalnih procesora signala, datoteke sa objektnim kodom kombinuju se u jedinstveni izvršni modul korišćenjem *linkera*. Prilikom formiranja izvršnog modula linker obavlja operacije relokacije kao i razrešavanje eksternih referenci. Pojedini delovi objektnog koda se mogu izvući i iz raspoloživih biblioteka. Direktive linkera usmeravaju proces povezivanja objektnih datoteka, postavljanja sekcija programa ili simbola na zadate adrese, definisanje globalnih simbola, itd.

Važan deo podrške za razvoj softvera predstavlja *softverski simulator*. Simulator predstavlja program čiji je ulaz izvršni kod koji se dobija procesom asembliranja i linkovanja. Program simulira svaku instrukciju pri čemu se vodi računa o sadržaju memorije kao i svih raspoloživih registara. Simulacioni program se može izvršavati korak po korak ili punom brzinom. Postoji mogućnost definisanja prekidnih tačaka. Bolji simulatori imaju sposobnost disasembliranja, prikaz

izvornog programa, kao i prikaz stanja u protoku radi otkrivanja konfliktnih situacija. Ulaz i izlaz podataka simuliraju se pomoću datoteka. Najveći problem predstavlja simulacija zahteva za prekid, koju može vršiti samo mali broj simulatora.

Programske biblioteke mogu biti veoma velika pomoć u razvoju složenih aplikacija. Programske biblioteke su skup asemblerskih potprograma, ili makroa, kojima se realizuju neke česte i tipične operacije: FFT leptiri, ćelije drugog reda za kaskadnu ili paralelnu realizaciju, FIR filtar, ulaz i izlaz podataka, itd. Često proizvođači procesora podstiču korisnike da im dostavljaju svoja rešenja, koja posle distribuiraju ostalim korisnicima uz minimalnu nadoknadu.

Viši programski jezici su još uvek retkost kod digitalnih procesora signala i razvijeni su uglavnom za novije procesore. Najviše se koristi prevodilac za programski jezik C, ali postoje i prevodioci za programske jezike PASKAL, FORTRAN i ADA. Na žalost, efikasnost izvršavanja programa napisanih u višim programskim jezicima nije uvek zadovoljavajuća, mada se u toj oblasti u poslednje vreme primećuje veliki napredak. Razlog za malu efikasnost takvih programa najviše leži u tome što digitalni procesori signala nemaju registre opšte namene, na kojima je u velikoj meri zasnovana optimizacija objektnog koda, kod prevodilaca za više programske jezike. U tom pogledu izuzetak predstavlja procesor TMS320C30 koji je, između ostalog, projektovan sa ciljem da ima i podršku za C prevodilac.

Izlaz iz C prevodioca najčešće nije objektni kod, već asemblerski program. Ova činjenica omogućava programeru da načini naknadne intervencije u vremenski kritičnim delovima programa. Druga mogućnost, koja se sreće kod većine prevodilaca za C, je direktiva `asm`, koja označava da iza nje sledi asemblerski kod. Ova direktiva omogućava mešanje asemblerskog i C koda, čime se znatno mogu popraviti performanse rezultujućeg objektnog koda.

Evaluacioni modul predstavlja realizaciju tipičnog sistema za procesiranje signala na jednoj nezavisnoj štampanoj ploči. Modul se može povezati paralelnim ili serijskim interfejsom sa terminalom, računarom opšte namene i štampačem. Rad takvog modula kontrolisan je jednostavnim monitorskim programom, a njime se može ispitati rad programa u sličnim uslovima kao kod aplikacije koja se razvija.

Nešto jednostavniji razvoj softvera postiže se pomoću *sistema za razvoj softvera*. Takav sistem je veoma sličan evaluacionom modulu, ali se od njega razlikuje po tome što je štampana ploča predviđena za priključivanje na sistemsku magistralu nekog računara opšte namene. U praksi se kao glavni računar najčešće koristi neki od IBM PC kompatibilnih računara. Ovakve sisteme proizvode proizvođači procesora, ali i veliki broj nezavisnih proizvođača. Prednost ovakvih sistema je u boljoj i bržoj komunikaciji sa glavnim računarom, tako da je razvojni ciklus koji se sastoji od pisanja asemblerskog programa, prevođenja, linkovanja i testiranja znatno ubrzan. Ovakav sistem omogućava i izvršavanje programa u skoro realnim uslovima. Ponekad su sistemi za razvoj opremljeni i A/D i D/A konvertorima tako da mogu da posluže i za krajnju realizaciju sistema, koja se onda svodi samo na pisanje softvera.

Najbolju podršku razvoju hardvera i softvera daje *emulator*. Emulator omogućava emulaciju sistema sa stvarnim hardverom, postavljanje prekidnih tačaka i izvršavanje programa iz memorije realizovanog sistema. Jedino se pomoću emulatora može postići potpuna integracija razvijenog hardvera i softvera, a da se pri tome imaju i dijagnostičke mogućnosti. Prekidne tačke se mogu postaviti na osnovu internih uslova ili na osnovu spoljnih događaja. Slično kao kod simulatora, moguć je kompletan pregled i izmena svih registara i memorijskih lokacija.

19.2 KONVOLUCIONI PROCESORI

Integrisani digitalni procesori signala opšte namene, koji su opisani u prethodnom odeljku, prilagođeni su po arhitekturi velikom broju algoritama koji se sreću u digitalnoj obradi signala. U najvećem broju slučajeva takav procesor ima samo *jednu* aritmetičku jedinicu (paralelni množač i aritmetičko-logičku jedinicu) i brzu memoriju za smeštanje signala i koeficijenata. Ovakva arhitektura zahteva da se u svakom instrukcijskom ciklusu obave dva čitanja iz memorije i jedan upis u memoriju. Čak i u slučajevima kada se svim memorijama može simultano pristupiti, *brzina rada procesora signala sa jednom aritmetičkom jedinicom je ograničena trajanjem memorijskog ciklusa*. Dakle, ovaj problem ne može se u potpunosti rešiti ni višestrukim magistralama niti beskonačno brzom aritmetičkom jedinicom.

Za ilustraciju ograničene brzine rada, posmatrajmo implementaciju FIR filtra 31. reda (sa 32 koeficijenta) pomoću procesora sa standardnom arhitekturom. Kod većine procesora signala, za realizaciju FIR filtra je potreban samo jedan instrukcijski ciklus po ćeliji. Dakle, implementacija tražene filtarske funkcije zahteva 32 instrukcijska ciklusa po odbirku, ne računajući ulazno-izlazne instrukcije. Ako jedan instrukcijski ciklus traje 50 ns, što je slučaj kod danas najbržih procesora, za implementaciju filtra je potrebno 1.6 μs po odbirku što odgovara učestanosti odabiranja od oko 600 KHz. Ako učestanost odabiranja treba da bude veća, kao što je to slučaj kod video signala, mora se u cilju povećanja brzine rada promeniti arhitektura procesora.

Povećanje brzine rada može se ostvariti uvođenjem više paralelnih množača sa akumulatorom. Ako se obezbedi da svaka ćelija FIR filtra ima svoje hardverske resurse, i ako se obezbedi simultani rad svih funkcionalnih blokova, dobiće se višestruko povećanje brzine rada, potrebno za obradu video signala. Takva struktura, koja je optimizovana za izvršenje FIR filtriranja, odnosno konvolucije, naziva se *konvolucioni procesor* ili *procesor za FIR filtriranje*.

Realizacija više paralelnih množača na jednom silicijumskom čipu skopčana je sa više problema. Pošto paralelni množač zauzima veliku površinu silicijuma, u realizaciji konvolucionih procesora koristi se realizacija množača koja kombinuje rezultate množenja sa manjem brojem bita. Na primer, množač 16×16 bita se realizuje korišćenjem četiri rezultata množenja 16×4 bita. Time se, naravno, usporava rad, ali se to nadoknađuje većim brojem množača. Osim toga, dobija se na fleksibilnosti jer se može praviti kompromis između tačnosti predstavljanja signala ili koeficijenata i brzine rada.

Osim razlike u broju množača, arhitektura konvolucionog procesora razlikuje se od arhitekture procesora signala opšte namene i po koncepciji samog procesora. Naime, *konvolucioni procesori se uvek koriste kao sekundarni procesori* nekog glavnog procesora za obradu signala, tj. nisu namenjeni za samostalni rad.

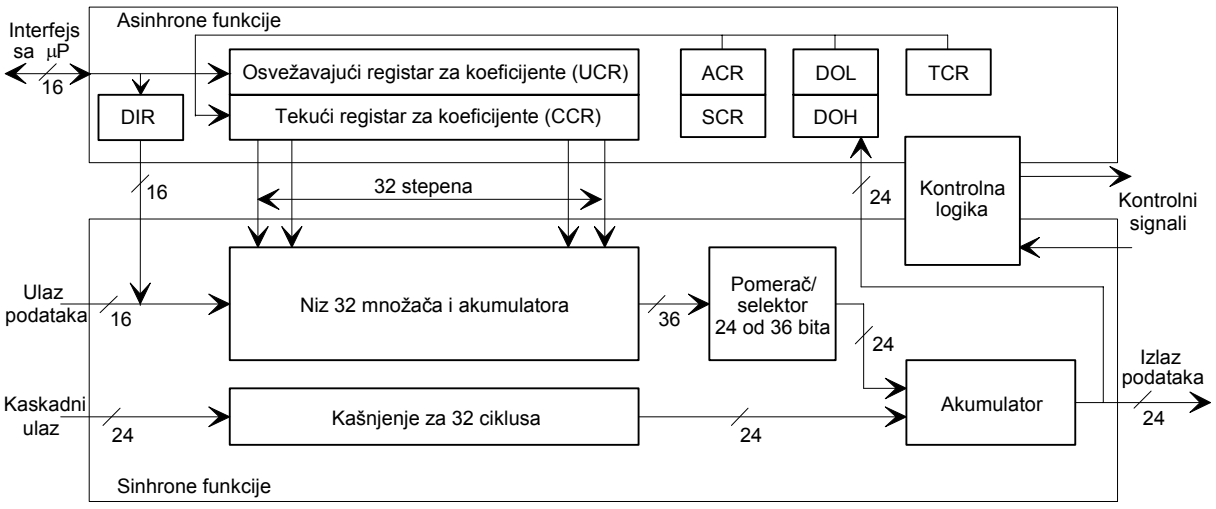
Direktna struktura za realizaciju FIR filtra, koja je prikazana na slici 7.1, nije pogodna za praktičnu realizaciju zbog problema oko realizacije niza sabirača. Naime, takav lanac sabirača unosi nedopustivo dugo kašnjenje. Za praktičnu realizaciju je mnogo pogodnija transponovana realizacija FIR filtra prikazana na slici 7.3. Kod ove realizacije se rezultat svakog množenja sabira sa prethodno zapamćenom parcijalnom sumom i rezultat upisuje u registar ili memoriju. Time se razbija lanac sabirača koji je unosio veliko kašnjenje. Ako se obezbedi da svaka ćelija FIR filtra ima svoj množač, sabirač i registar za smeštaj rezultata, onda je vreme potrebno za obradu jednog odbirka jednako vremenu potrebnom za *jedno* množenje, sabiranje i smeštaj međurezultata. Kako se ove operacije mogu obaviti u najviše nekoliko ciklusa, povećanje brzine rada u odnosu na

standardne procesore signala je višestruko. Ovakvu arhitekturu koriste praktično svi konvolucioni procesori jer je brza, modularna i omogućava kaskadno povezivanje.

Mada opisana struktura izgleda pogodna samo za FIR filtriranje, transverzalna struktura se može prilagoditi i za druge algoritme digitalne obrade signala, matrično množenje, pa čak i za brzo izračunavanje DFT. Zbog toga su konvolucioni procesori našli široku primenu u telekomunikacijama, radarskoj tehnici i obradi slike u realnom vremenu. Proizvodi ih više proizvođača. Tipične dužine reči su 8 do 16 bita za signale, i 4 do 16 bita za koeficijente. Broj ćelija FIR filtra koji je realizovan na jednom integrisanom kolu kreće se od 4 do 32, ali većina proizvođača obezbeđuje i kaskadno povezivanje procesora.

Među brojnim konvolucionim procesorima raznih proizvođača, najviše se po svojim performansama ističe integrisani procesor signala A100 britansko-američke firme Inmos. Visoki stepen integracije omogućio je implementaciju čak 32 ćelije FIR filtra na jednom integrisanom kolu. Arhitektura procesora A100 prikazana je na slici 19.23.

Ulazni podaci u procesor A100 unose se sinhrono sa taktom u 16-bitnom paralelnom formatu u komplementu dvojke. Ulazni podaci se takođe mogu unositi i na asinhroni način, koristeći spegu sa glavnim procesorom i prihvatni registar DIR. Koeficijenti FIR filtra mogu biti predstavljeni sa 4, 8, 12 i 16 bita, takođe u formatu komplementa dvojke. Ako se za predstavljanje koficijenata koristi B_c bita, onda je maksimalna učestanost odabiranja, tj. učestanost kojom pritiču ulazni podaci, data sa $2F_{cl}/B_c$, gde je F_{cl} učestanost takta. Kako je maksimalna učestanost takta 30 MHz, maksimalna učestanost odabiranja iznosi 15 MHz ako su koeficijenti predstavljeni sa 4 bita, odnosno 3.75 MHz za 16-bitne koeficijente. Drugačije gledano, to iznosi između 2.1 ns i 8.3 ns po ćeliji FIR filtra, što je desetak puta brže od najboljih procesora signala standardne arhitekture. Broj operacija u sekundi kreće se između 120 MOPS ($120 \cdot 10^6$ op/s) i 480 MOPS, zavisno od broja bita za predstavljanje koeficijenata. Ovi impresivni rezultati postignuti su na račun vrlo velike specijalizacije procesora.



Slika 19.23 Arhitektura konvolucionog procesora A100.

Za smeštaj koeficijenata postoje dva registra: tekući registar (CCR) i osvežavajući registar (UCR). Kapacitet svakog registra je 32 16-bitne reči. Koeficijenti koji se nalaze u tekućem registru množe odgovarajuće signale u svakom ciklusu, dok se istovremeno preko asinhronog interfejsa može izvršiti upis novih koeficijenata u osvežavajući registar. Svaki koeficijent se može promeniti za manje od 100 ns. Zamena uloga tekućeg i osvežavajućeg registra može se izvršiti bez prekida

toka ulaznih podataka. Uvođenje registra za koeficijente takođe povećava brzinu rada sistema, jer se istovremeno čitaju 32 koeficijenta, za šta bi kod standardnog digitalnog procesora signala bila potrebna 32 memorijska ciklusa.

Procesor A100 predviđen je za kaskadno povezivanje radi realizacije FIR filtara sa većim brojem koeficijenata od 32. Kaskadno povezivanje se izvodi tako što se na kaskadni ulaz dovodi izlaz prethodnog stepena u 24-bitnom formatu, zakasni u pomeračkom registru koji ima 32 stepena i sabere sa izlazom iz niza množača-akumulatora. Interesantno je da su i kaskadni ulaz i izlaz multipleksirani da bi se smanjio broj spoljnih priključaka, tj. postoji samo po 12 ulaznih i izlaznih priključaka.

Da bi se ublažili efekti konačne dužine reči kod realizacije FIR filtara sa velikim brojem koeficijenata, niz množača-akumulatora radi sa povećanom tačnošću od 36 bita. Time je skoro sasvim otklonjen uticaj grešaka usled kvantovanja proizvoda. Osim toga, pošto je proširenje formata podataka izvedeno i sa leve strane, omogućeno je i sabiranje velikog broja sabiraka bez prekoračenja opsega. Ova činjenica je veoma značajna kod realizacije FIR filtara sa velikim brojem koeficijenata, koji može iznositi nekoliko desetina ili stotina. Na primer, u slučaju realizacije FIR filtra sa 1024 koeficijenta sa 16-bitnim formatom signala i koeficijenata, za predstavljanje rezultata može biti potrebno čak 42 bita, što za većinu standardnih procesora signala predstavlja ozbiljan problem. Međutim, kod procesora A100, rezultat se posle svaka 32 stepena svodi na 24-bitni format. Pri tome se koristi pomerač, kojim se ustvari iz 36 bitne reči selektuje jedno od četiri 24-bitnih polja (počevši od sedmog, jedanaestog, petnaestog ili dvadesetog bita).

Kao što se vidi, procesor A100 nije programabilan u pravom smislu reči, jer stalno izvodi isti skup operacija (množenje i akumuliranje). Programiranjem se mogu menjati samo vrednosti koeficijenata množača. Zbog toga je procesor signala A100 predviđen je da radi kao sekundarni procesor u nekom sistemu za obradu signala. Glavni procesor u sistemu može biti bilo kakav mikroprocesor opšte namene ili standardni digitalni procesor signala. Sa programerske tačke gledišta, procesor A100 predstavlja blok od 128 16-bitnih reči, koji je smešten negde u memorijskom prostoru glavnog procesora. Prve 64 reči predstavljaju registre za koeficijente UCR i CCR, dok se u preostalih 64 reči nalaze aktivni kontrolni registar (ACR), statički kontrolni registar (TCR), kontrolni registar za testiranje (TCR), ulazni registar DIR i dva izlazna registra DOL i DOH. Preostalih 58 registara nema specificiranu namenu. Registri se adresiraju preko sedmobitne adresne magistrale, a podaci se dovode ili uzimaju iz procesora A100 preko standardnog nemultipleksiranog 16-bitnog interfejsa.

Primene opisanog konvolucionog procesora vrlo su brojne. Osim FIR filtriranja i konvolucije, za koje je procesor očigledno namenjen, pomoću procesora A100 mogu se realizovati i korelacija, matrično množenje, adaptivni filtri, sinteza talasnih oblika pa čak i Diskretna Furijeova transformacija.

Primena konvolucionog procesora za izračunavanje DFT je vrlo interesantna. Za izračunavanje DFT preko konvolucije mogu se iskoristiti FFT algoritmi sa prostim faktorima, opisani u odeljku 5.9.3, kao i CTA algoritam, opisan u odeljku 5.11. U slučaju kada je broj odbiraka veći od 32, koriste se algoritmi za dekompoziciju, kojima se DFT dugačke sekvence svodi na više kratkih cirkularnih konvolucija. Na taj način se, na primer, može realizovati DFT sekvence od oko 1000 odbiraka za manje od 10 ms. Ako se koriste dva procesora A100 i brzi procesor

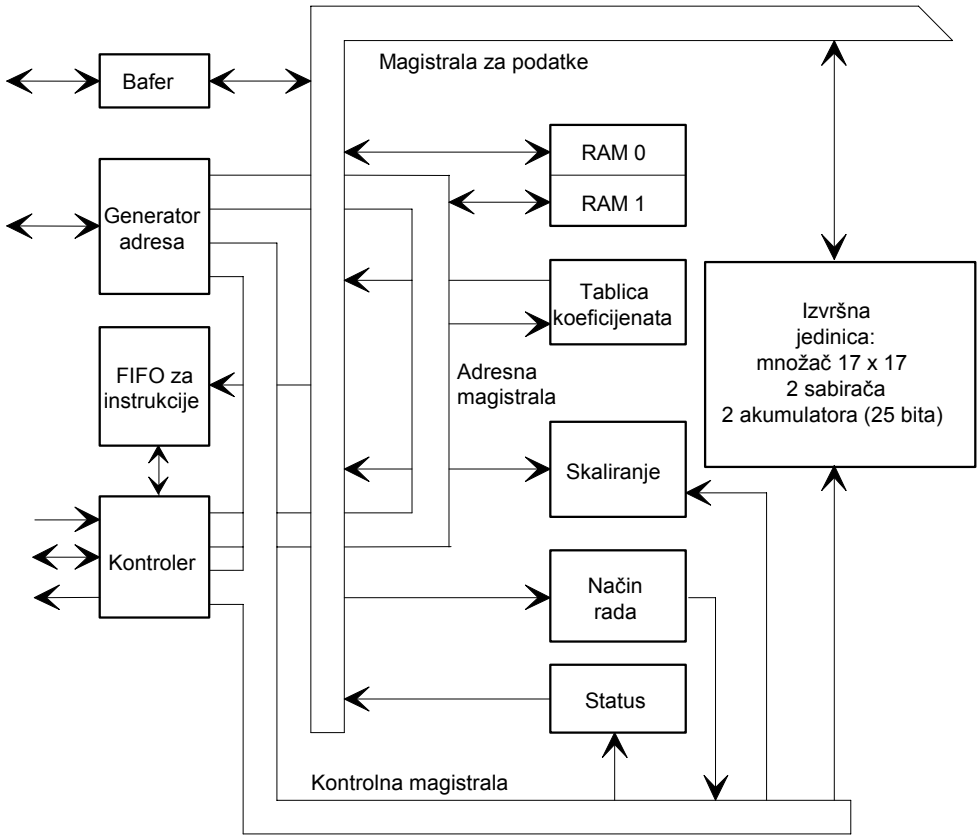
signala kao kontroler, DFT iste sekvence se može odrediti za 1-2 ms, što nije moguće ostvariti ni sa jednim standardnim integrisanim procesorom signala.

19.3 VEKTORSKI PROCESORI

Vektorski procesor je integrisani digitalni procesor signala koji je optimizovan za rad sa podacima koji predstavljaju vektore ili matrice. Po svojim karakteristikama leži negde između standardnih procesora signala i konvolucionih procesora signala. Naime, vektorski procesor poseduje samo jednu aritmetičku jedinicu, ali je predviđen da radi kao sekundarni procesor za obradu signala sa vrlo malim skupom mogućih operacija.

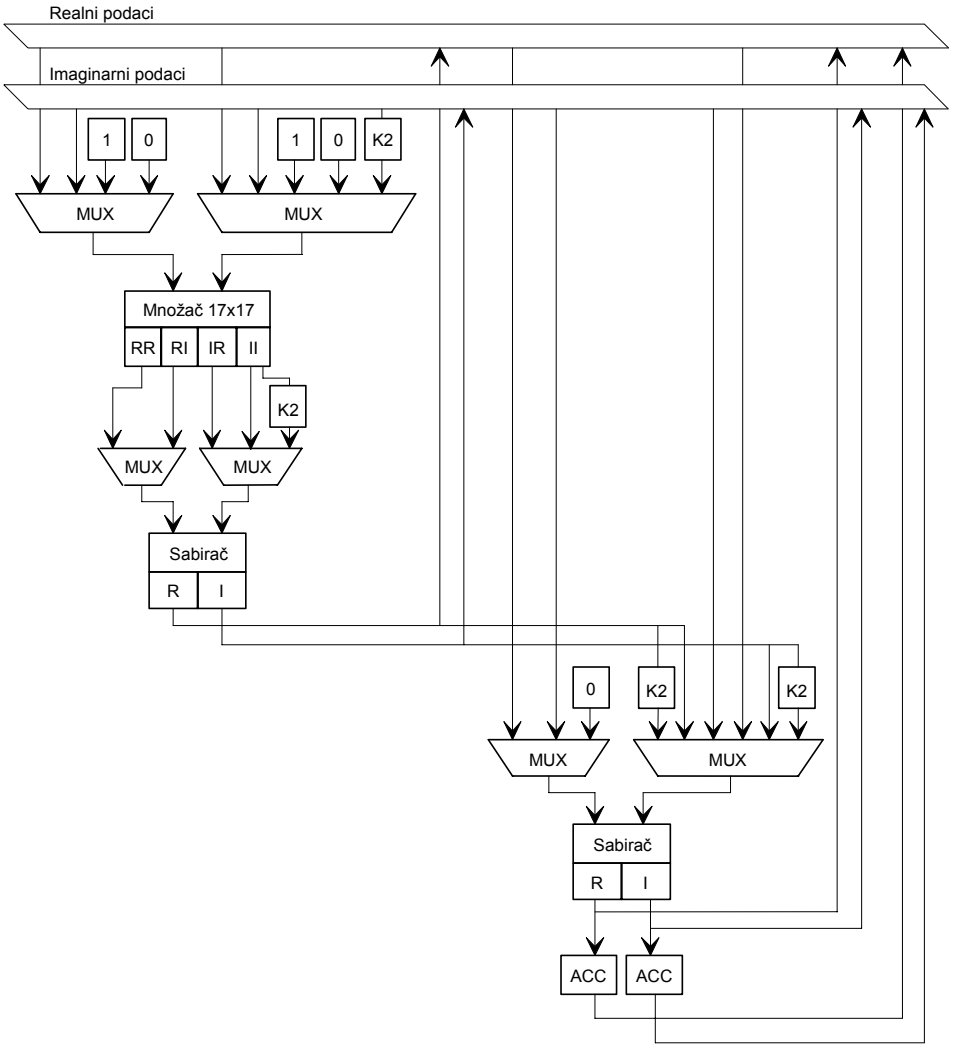
Mada su vektorski procesori vrlo česti sklopovi kod velikih superračunara, u integrisanoj formi vektorske procesore proizvodi samo firma Zoran Corporation iz SAD [V-3]. Tipičan predstavnik ove familije vektorskih procesora je procesor ZR34161 [Z-2] koji će biti opisan u daljem izlaganju. Uprošćeni prikaz arhitekture vektorskog procesora ZR34161 prikazan je na slici 19.24.

Na blok dijagramu arhitekture vektorskog procesora mogu se uočiti tri osnovna modula. To su interfejsni blok prema magistrali, izvršna jedinica i memorijsko-registarski blok. Interfejsni blok izvršava sve funkcije vezane za magistralu preko koje su spregnuti glavni procesor i vektorski procesor. Najvažnije funkcije ovog bloka se odnose na učitavanje i dekodovanje instrukcija, ulaz i izlaz podataka, komunikaciju između internih i eksternih memorija i DMA. Da bi se obavile ove funkcije interfejsni blok sadrži bidirekcioni bafer za podatke, FIFO bafer za instrukcije (kapaciteta 4 instrukcije), generator adresa i kontrolerski blok. Komunikacija između pojedinih funkcionalnih blokova unutar vektorskog procesora obavlja se preko tri magistrale: magistrale za podatke, adresne magistrale i kontrolne magistrale.



Slika 19.24 Arhitektura vektorskog procesora ZR34161.

Arhitektura izvršne jedinice prikazana je na slici 19.25. Izvršna jedinica se sastoji od paralelnog množača 17×17 bita, dva sabirača kompleksnih brojeva i dva akumulaciona registra (za realni i imaginarni deo rezultata). Oduzimanje se obavlja pomoću kola za formiranje komplementa dvojke (blok K2). Kao što se vidi, arhitektura aritmetičke jedinice je veoma slična arhitekturi aritmetičke jedinice standardnih digitalnih procesora signala, ali je dodatno optimizovana za implementaciju leptir operacije kod FFT algoritama. U aritmetičkoj jedinici podaci su predstavljeni sa 17 bita u formatu komplementa dvojke sa fiksnom tačkom, ili blokovskom predstavom sa pokretnom tačkom. Akumulatorski registri su prošireni na 25 bita da bi se omogućilo sabiranje 256 17-bitnih brojeva bez prekoračenja opsega.



Slika 19.25 Arhitektura izvršne jedinice procesora signala ZR 34161.

Instrukcije koje upravljaju radom aritmetičke jedinice su veoma visokog nivoa i njima se realizuju sledeće operacije:

1. Realni ili kompleksni skalarni proizvod vektora (`MLTR` ili `MLTC`),
2. Realno ili kompleksno sabiranje (`ADDR` ili `ADDC`),
3. Realna ili imaginarna akumulacija vektora (`ACCR` ili `ACCI`),
4. Modulacija ili demodulacija (`MODLT` ili `DEMO`),
5. Apsolutna vrednost, (`ABS`)
6. Kvadriranje amplitude i akumuliranje, (`MGSQ`)

7. Određivanje kompleksno konjugovane vrednosti, (`CMCN`)

8. Brza Furijeova transformacija (`FFT`).

Memorijsko registarski blok se sastoji od brze RAM memorije ukupnog kapaciteta 128 reči od 38 bita, RAM memorije za skaliranje kapaciteta 64 reči od 4 bita, tabele sinusnih i kosinusnih koeficijenata koja je smeštena u ROM memoriji kapaciteta 256 reči od 17 bita, registara i logike za skaliranje i registara za status i način rada.

RAM memorija za podatke prilagođena je za smeštanje 128 kompleksnih podataka, jer se realni i imaginarni delovi predstavljaju sa po 19 bita. Dva dodatna bita su uvedena radi smanjenja mogućnosti za prekoračenje opsega i nisu pristupačna van vektorskog procesora. Poslednji LSB bit takođe nije pristupačan spolja, a koristi se radi smanjenja grešaka kvantovanja unutar procesora. RAM memorija se može podeliti na dve nezavisne sekcije od po 64 kompleksnih reči. Prednost ovakve konfiguracije je što se jednoj sekciji može pristupiti spolja preko interfejsnog bloka, dok drugu sekciju istovremeno koristi aritmetička jedinica. Ovakva konfiguracija je veoma pogodna za rad u realnom vremenu kada se mora čekati na prikupljanje podataka, kao što je to slučaj kod izračunavanja DFT.

Skup instrukcija vektorskog procesora je veoma mali i sastoji se od svega 23 instrukcije. U skupu postoji 5 instrukcija za premeštanje podataka iz interne memorije u eksternu memoriju i obrnuto, 4 instrukcije za aritmetičke operacije između dva vektora od kojih je jedan u internoj a drugi u eksternoj memoriji, 11 instrukcija za aritmetičke operacije nad jednim vektorom u internoj memoriji i tri kontrolne instrukcije. Instrukcije su veoma visokog nivoa, odnosno jednom instrukcijom se izvršava veoma veliki broj paralelnih ili sekvencijalnih operacija. Na primer, za izvršavanje FFT algoritma potrebna je samo jedna instrukcija. Za izračunavanje kvadrata spektra neke sekvence potrebne su svega četiri instrukcije:

```
LD    NMPT:128   RS:0     MDF:2    ZR:1     MBA:0;
FFT   NMPT:128   RS:0     FPS:64   LPS:1;
MGSQ  NMPT:128   RS:0     ADF:2;
ST    NMPT:128   RS:0     RV:1     MDF:2    MBA:256;
```

od kojih prva vrši prebacivanje vektora od 128 odbiraka iz eksterne u internu memoriju, druga izračunava FFT, treća nalazi kvadrat amplitudskog spektra a četvrta smešta rezultate u eksternu memoriju.

Sve instrukcije su višeciklusne, pri čemu broj ciklusa linearno zavisi od dužine vektora nad kojim se izvodi operacija. Ova osobina predstavlja važnu razliku između vektorskih procesora i standardnih i konvolucionih procesora signala. Na primer, instrukcija FFT primenjena na sekvencu od 128 kompleksnih odbiraka traje čak 1850 ciklusa od 100 ns, odnosno 185 μs.

Vektorski procesori su veoma pogodni za multiprocesorske konfiguracije zbog svoje arhitekture i skupa instrukcija. Naime, zbog toga što znatan broj instrukcija operiše sa vektorima iz interne memorije, kada se podaci i odgovarajuće instrukcije prebace u interne memorije i registre vektorskog procesora vektorski procesor više nema potrebe za pristup eksternoj memoriji. To znači da za to vreme eksternoj memoriji može pristupiti glavni procesor ili neki drugi vektorski procesor. Zbog toga što priključenjem više vektorskih procesora na magistralu ne dolazi do zagušenja magistrale, performanse multiprocesorske konfiguracije skoro linearno rastu sa povećanjem broja vektorskih procesora. Na primer, prema podacima proizvođača, jedan vektorski procesor ZR34161 može izračunati DFT kompleksne sekvence od 1024 odbirka za 3.3 ms, dva vektorska procesora za 1.9 ms, a četiri vektorska procesora za 1.2 ms.